用户指南

Amazon EKS



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon EKS: 用户指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Amazon Web Services 文档中描述的 Amazon Web Services 服务或功能可能因区域而异。要查看适用于中国区域的差异，请参阅 中国的 Amazon Web Services 服务入门 (PDF)。

Table of Contents

..... xxvi
..... xxvii
什么是 Amazon EKS？ 1
Amazon EKS：简化 Kubernetes 管理 1
使用 Kubernetes 进行构建和扩展：Amazon EKS 功能 2
Amazon EKS 的功能 3
相关服务 4
Amazon EKS 定价 5
常见使用案例 5
架构 7
控制面板 7
计算 7
EKS 功能 8
Kubernetes 概念 9
为何选择 Kubernetes？ 10
集群 14
工作负载 17
后续步骤 21
部署选项 22
了解 Amazon EKS 部署选项 22
云中的 Amazon EKS 22
数据中心或边缘环境中的 Amazon EKS 23
适用于气隙环境的 Amazon EKS Anywhere 24
Amazon EKS 工具 24
设置 26
后续步骤 26
设置 Amazon CLI 26
创建访问密钥 26
配置 Amazon CLI 27
获取安全令牌 27
验证用户身份 28
后续步骤 28
设置 kubectl 和 eksctl 28
安装或更新 kubectl 29

第 1 步：查看 kubectl 是否已安装 29
第 2 步：安装或更新 kubectl 29
安装 eksctl 43
后续步骤 44
快速入门 45
在本教程中： 45
先决条件 45
配置集群 45
创建 IngressClass 46
部署 2048 游戏示例应用程序 47
使用 Amazon EKS 自动模式持久保存数据 50
清理 52
学习 Amazon EKS 53
概览 53
Amazon EKS 研讨会 53
Amazon EKS 集群设置动手实验教程 54
Amazon EFS 示例 55
Amazon 教程 56
开发人员讲习会 56
Terraform 讲习会 56
Amazon Amazon EKS 培训 56
开始使用 58
创建集群（EKS 自动模式） 58
创建集群（eksctl） 59
先决条件 59
第 1 步：创建 Amazon EKS 集群和节点 59
第 2 步：查看 Kubernetes 资源 61
第 3 步：删除集群和节点 62
后续步骤 63
创建集群（控制台和 CLI） 63
先决条件 63
第 1 步：创建 Amazon EKS 集群 64
第 2 步：将计算机配置为与您的集群通信 67
第 3 步：创建节点 68
第 4 步：查看资源 69
第 5 步：删除资源 70

后续步骤 71
EKS 自动模式 72
特征 72
自动化组件 73
配置 74
责任共担模式 74
创建集群 76
eksctl CLI 76
Amazon CLI 78
管理控制台 85
启用现有集群 87
迁移参考 88
迁移 EBS 卷 88
迁移负载均衡器 89
在集群上启用 89
从 Karpenter 迁移 93
从 MNG 迁移 96
从 Fargate 迁移 97
运行工作负载 102
部署 inflate 工作负载 102
部署负载均衡器 105
部署有状态工作负载 110
部署加速型工作负载 115
Configure 122
创建节点类 125
创建节点池 132
静态容量节点池 138
创建 IngressClass 144
创建服务 151
创建 StorageClass 157
禁用 EKS 自动模式 164
更新 Kubernetes 版本 166
检查内置节点池 167
控制部署 169
运行关键附加组件 170
使用网络策略 172

为子网添加标签 181
生成 CIS 报告 183
加密根卷 185
更新 AMI 控制 189
控制容量预留 192
使用 Local Zones 196
工作方式 200
托管式实例 201
身份和访问权限 204
Networking 210
故障排除 213
节点监控代理 214
使用 Amazon EC2 CLI 获取 EC2 托管式实例的控制台输出 214
使用调试容器和 kubectl CLI 获取节点日志 214
在 Amazon 控制台中查看与 EKS 自动模式关联的资源 215
查看您 Amazon 账户中的 IAM 错误 216
对容器组（pod）无法调度到自动模式节点进行问题排查 216
对节点无法加入集群进行问题排查 216
在容器组（pod）之间共享卷 219
在控制面板日志中查看 Karpenter 事件 219
对自动模式涉及的控制器进行问题排查 221
相关资源 185
发行说明 221
2025 年 11 月 19 日 221
2025 年 11 月 19 日 222
2025 年 10 月 23 日 222
2025 年 10 月 1 日 222
2025 年 9 月 30 日 222
2025 年 9 月 29 日 222
2025 年 9 月 10 日 222
2025 年 8 月 24 日 222
2025 年 8 月 15 日 222
2025 年 8 月 6 日 223
2025 年 6 月 30 日 223
2025 年 6 月 20 日 223
2025 年 6 月 13 日 223

2025 年 4 月 30 日 223
2025 年 4 月 18 日 223
2025 年 4 月 11 日 223
2025 年 4 月 4 日 224
2025 年 3 月 31 日 224
2025 年 3 月 21 日 224
2025 年 3 月 14 日 224
配置群集 225
创建自动集群 225
先决条件 45
创建集群 – Amazon 控制台 226
创建集群 – Amazon CLI 230
后续步骤 85
创建集群 235
先决条件 45
第 1 步：创建集群 IAM 角色 236
第 2 步：创建集群 237
第 3 步：更新 kubeconfig 245
第 4 步：集群设置 246
后续步骤 85
创建预置式控制面板 247
概述 83
使用案例 130
控制面板扩展层级 249
注意事项 89
预置式控制面板入门 252
集群见解 259
集群见解类型 259
注意事项 260
使用案例 260
开始使用 261
查看集群见解 261
更新 Kubernetes 版本 267
Amazon EKS 自动模式的注意事项 268
摘要 268
第 1 步：准备升级 268

第 2 步：检查升级注意事项 269
第 3 步：更新集群控制面板 270
第 4 步：更新集群组件 272
降级 Amazon EKS 集群的 Kubernetes 版本 273
删除集群 273
注意事项 89
删除集群（eksctl） 274
删除集群（Amazon 控制台） 275
删除集群（Amazon CLI） 276
防止意外删除 EKS 集群 277
集群终端节点访问 278
IPv6 集群端点格式 279
IPv4 集群端点格式 279
集群私有端点 280
修改集群终端节点访问 280
访问私有 API 服务器 283
配置端点访问权限 284
启用 Windows 支持 287
注意事项 89
先决条件 45
启用 Windows 支持 289
部署 Windows 容器组（pod） 291
在 Windows 节点上支持更高的容器组（pod）密度 291
禁用 Windows 支持 292
私有集群 292
集群架构要求 292
节点要求 293
容器组（pod）要求 294
自动扩缩 296
EKS 自动模式 296
其他解决方案 297
了解可用区转移 297
了解容器组（pod）之间的东-西网络流量 298
EKS 可用区转移要求 302
常见问题 309
其他资源 311

启用可用区转移 312
注意事项 312
什么是 Amazon 应用程序恢复控制器？ 312
什么是可用区转移？ 313
什么是可用区自动转移？ 313
EKS 在自动转移期间会做什么？ 313
向 Amazon 应用程序恢复控制器（ARC）（Amazon 控制台）注册 EKS 集群 313
后续步骤 85
管理访问权限 315
常见任务 315
背景 181
EKS 自动模式注意事项 274
Kubernetes API 访问权限 316
将 IAM 身份与 Kubernetes 权限相关联 317
设置集群身份验证模式 317
访问条目 319
aws-auth ConfigMap 369
关联 OIDC 提供者 378
取消关联 OIDC 提供者 383
访问集群资源 383
所需的权限 384
使用 kubectl 访问集群 389
自动创建 kubeconfig 文件 390
Amazon 工作负载访问权限 391
服务账户令牌 391
集群附加组件 393
向 Amazon Elastic Kubernetes Service 集群上的工作负载授予 Amazon Identity and Access Management 权限 393
包含 IRSA 的凭证 396
容器组身份 418
管理计算 448
比较计算选项 448
托管节点组 452
托管节点组概念 453
托管节点组容量类型 455
创建 458

更新 467
更新行为详细信息 471
启动模板 474
删除 488
自行管理的节点 490
Amazon Linux 491
Bottlerocket 499
Windows 502
Ubuntu Linux 510
更新方法 512
Amazon Fargate 524
Amazon Fargate 注意事项 524
Fargate 比较表 526
开始使用 528
定义配置文件 533
删除配置文件 538
容器组（pod）配置详细信息 539
操作系统补丁事件 541
收集指标 544
日志记录 546
Amazon EC2 实例类型 556
Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量 558
EKS 自动模式注意事项 274
预先构建的优化型 AMI 559
AL2 AMI 弃用 560
Amazon Linux 565
Bottlerocket 573
Ubuntu Linux 578
Windows 578
节点运行状况 634
节点监控代理 634
节点自动修复 635
节点运行状况问题 636
查看节点运行状况 644
获取节点日志 646
混合节点 649

特征 649
限制 650
注意事项 650
其他资源 651
先决条件 651
运行混合节点 709
配置 728
工作方式 807
混合节点 nodeadm 846
问题排查 861
应用程序数据存储 880
Amazon EBS 880
注意事项 880
先决条件 881
第 1 步：创建 IAM 角色 881
第 2 步：获取 Amazon EBS CSI 驱动程序 889
第 3 步：部署示例应用程序 890
Amazon EFS 890
注意事项 890
先决条件 890
第 1 步：创建 IAM 角色 891
第 2 步：获取 Amazon EFS CSI 驱动程序 897
第 3 步：创建 Amazon EFS 文件系统 897
第 4 步：部署示例应用程序 898
Amazon FSx for Lustre 898
部署驱动程序 898
优化（EFA） 905
优化（非 EFA） 921
Amazon FSx for NetApp ONTAP 929
Amazon FSx for OpenZFS 929
Amazon File Cache 930
适用于 Amazon S3 的 Mountpoint 930
注意事项 930
部署驱动程序 931
移除驱动程序 940
CSI 快照控制器 942

配置联网 943
通过管理控制台将现有 VPC 子网添加到 Amazon EKS 集群 943
VPC 和子网要求 943
VPC 要求和注意事项 944
子网要求和注意事项 945
共享子网要求和注意事项 950
创建 VPC 951
先决条件 45
公有子网和私有子网 952
仅公有子网 954
仅私有子网 955
安全组要求 956
默认集群安全组 956
限制集群流量 958
共享安全组 959
管理联网附加组件 959
内置附加组件 959
可选 Amazon 联网附加组件 960
Amazon VPC CNI 961
备用 CNI 插件 1071
Multus 1072
Amazon 负载均衡器控制器 1073
CoreDNS 1090
kube-proxy 1106
工作负载 1111
示例部署（Linux） 1111
先决条件 45
创建命名空间 1112
创建 Kubernetes 部署 1112
创建服务 1113
审核已创建的资源 1114
在容器组（pod）上运行 Shell 1117
后续步骤 1118
示例部署（Windows） 1118
先决条件 45
创建命名空间 1112

创建 Kubernetes 部署 1112
创建服务 1113
审核已创建的资源 1121
在容器组（pod）上运行 Shell 1117
后续步骤 1125
Vertical Pod Autoscaler 1125
部署 Vertical Pod Autoscaler 1125
测试 Vertical Pod Autoscaler 安装 1127
Horizontal Pod Autoscaler 1130
运行 Horizontal Pod Autoscaler 测试应用程序 1131
网络负载均衡 1133
先决条件 45
注意事项 89
创建网络负载均衡器 1136
（可选）部署示例应用程序 1139
应用程序负载均衡 1142
先决条件 45
在入口组中重复使用 ALB 1145
（可选）部署示例应用程序 1146
限制服务外部 IP 1148
将镜像复制到存储库 1151
查看 Amazon 映像注册表 1153
Amazon EKS 附加组件 1157
注意事项 1157
附加组件的自定义命名空间 1158
Amazon EKS 自动模式的注意事项 1159
支持 1160
Amazon 附加组件 1161
社区附加组件 1179
Marketplace 附加组件 1184
创建附加组件 1200
更新附加组件 1211
验证兼容性 1218
移除附加组件 1219
IAM 角色 1223
可以自定义的字段 1229

验证容器镜像 1232
EKS 功能 1233
可用的功能 1233
Amazon Controllers for Kubernetes (ACK) 1233
Argo CD 1234
kro (Kube Resource Orchestrator) 1234
EKS 功能的优势 1234
定价 1235
EKS 功能的工作原理 1235
常见使用案例 1236
使用功能 1237
EKS 功能资源 1237
了解功能状态 1238
创建功能 1239
列出功能 1239
描述功能 1240
更新功能的配置 1242
删除功能 1242
后续步骤 85
Kubernetes 资源 1244
Argo CD 资源 1244
kro 资源 1245
ACK 资源 1246
资源限制 1247
后续步骤 85
注意事项 1248
功能 IAM 角色和 Kubernetes RBAC 1248
多集群架构模式 1248
将 EKS 功能与自主管理型解决方案进行比较 1250
Amazon Controllers for Kubernetes 1252
ACK 的工作原理 1252
ACK 的优势 1253
支持的 Amazon 服务 1253
与其他 EKS 托管功能集成 1254
开始使用 ACK 1254
创建 ACK 功能 1255

ACK 概念 1264
配置权限 1271
注意事项 1276
问题排查 1283
与自主管理型的比较 1287
Argo CD 1288
Argo CD 的工作原理 1288
Argo CD 的优势 1289
与 Amazon Identity Center 集成 1290
与其他 EKS 托管功能集成 1254
开始使用 Argo CD 1290
创建 Argo CD 功能 1290
Argo CD 概念 1300
配置权限 1307
使用 Argo CD 1314
Argo CD 注意事项 1351
问题排查 1359
与自主管理型的比较 1363
kro 1367
kro 工作原理 1367
kro 优势 1368
与其他 EKS 托管功能集成 1254
kro 入门 1369
创建 kro 功能 1369
kro 概念 1379
配置权限 1386
注意事项 1390
问题排查 1396
与自主管理型的比较 1402
问题排查功能 1403
一般问题排查方法 1403
检查功能运行状况 1404
常见功能状态 1405
验证 Kubernetes 资源状态 1405
查看 IAM 权限和集群访问权限 1406
功能特定问题排查 1407

所有功能的常见问题 1408
后续步骤 85
集群管理 1410
成本监控 1410
按容器组（pod）查看成本 1411
安装 Kubecost 1412
访问 Kubecost 控制面板 1412
了解有关 Kubecost 的更多信息 1413
Metrics Server 1421
注意事项 89
将 Amazon EKS 附加组件作为社区附加组件部署 1422
使用清单部署 1423
使用 Helm 部署应用程序 1423
标记您的资源 1425
有关标签的基本知识 1425
标记您的资源 1426
标签限制 1426
标记资源以便于计费 1427
通过控制台使用标签 1428
通过 CLI、API 或 eksctl 使用标签 1428
服务配额 1430
在 Amazon Web Services 管理控制台中查看 EKS 服务配额 1430
使用 Amazon CLI 查看 EKS 服务配额 1431
Amazon EKS 服务配额 1431
Amazon Fargate 服务配额 1432
安全性 1433
最佳实践 1434
分析漏洞 1434
适用于 Amazon EKS 的互联网安全中心（CIS）基准 1434
Amazon EKS 平台版本 1435
操作系统漏洞列表 1435
使用 Amazon Inspector 进行节点检测 1435
使用 Amazon GuardDuty 进行集群和节点检测 1435
验证合规性 1435
EKS 注意事项 1436
基础结构安全性 1436

恢复能力 1440
防止跨服务混淆代理 1441
Kubernetes 的注意事项 1443
证书签名 1443
默认的角色和用户 1445
启用密钥加密 1450
Amazon Secrets Manager 1454
所有 Kubernetes API 数据默认启用信封加密 1455
EKS 自动模式注意事项 1461
API 安全性和身份验证 1461
网络安全 1461
EC2 托管式实例安全性 1462
自动化的资源管理 1463
安全最佳实践 1463
EKS 功能的注意事项 1464
功能 IAM 角色 1464
EKS 访问条目 1465
其他 Kubernetes 权限 1466
功能所需的 IAM 权限 1468
安全最佳实践 1468
后续步骤 85
IAM 参考 1471
受众 1471
使用身份进行身份验证 1472
使用策略管理访问 1474
Amazon EKS 和 IAM 1476
基于身份的策略 1480
服务相关角色 1486
容器执行 IAM 角色 1500
Connector IAM 角色 1504
Amazon 托管策略 1508
故障排除 1531
集群 IAM 角色 1534
节点 IAM 角色 1537
自动模式集群 IAM 角色 1542
自动模式节点 IAM 角色 1545

功能 IAM 角色 1548
监控集群 1554
Amazon EKS 上的监控和日志记录 1554
Amazon EKS 监控和日志记录工具 1555
可观测性仪表板 1558
摘要 1558
集群运行状况 1559
控制面板监控 1559
集群见解 1561
节点运行状况问题 1561
EKS 功能 1561
容器网络可观测性 1561
使用案例 130
功能 72
开始使用 261
先决条件和重要提示 1563
使用 Amazon CLI、EKS API 和 NFM API 1564
使用基础设施即代码（IaC） 1566
如何工作？ 175
注意事项和限制 1576
Prometheus 指标 1576
第 1 步：开启 Prometheus 指标 1577
第 2 步：使用 Prometheus 指标 1579
第 3 步：管理 Prometheus 抓取程序 1579
使用 Helm 部署 1579
控制面板 1582
Amazon CloudWatch 1589
Amazon CloudWatch 中的基本指标 1589
Amazon CloudWatch Observability Operator 1596
控制面板日志 1597
启用或禁用控制面板日志 1598
查看集群控制面板日志 1600
Amazon CloudTrail 1601
参考信息 1602
日志文件条目 1603
自动扩缩组指标 1605

ADOT Operator 1606
Amazon EKS 控制面板 1606
什么是 Amazon EKS 控制面板？ 1606
控制面板如何使用 Amazon Organizations？ 1607
使用 Amazon 控制台启用 EKS 控制面板 1608
查看 EKS 控制面板 1608
配置控制面板 1609
使用 Amazon 控制台禁用 EKS 控制面板 1610
EKS 控制面板问题排查 1610
配置组织 1611
使用其他服务 1617
Amazon Backup 1617
Amazon CloudFormation 1618
Amazon EKS 和 Amazon CloudFormation 模板 1618
了解有关 Amazon CloudFormation 的更多信息 1618
Amazon CodeConnections 1619
在 Argo CD 中使用 Amazon CodeConnections 1619
搭配 Argo CD 使用 CodeConnections 的注意事项 1620
Amazon Detective 1620
将 Amazon Detective 与 Amazon EKS 结合使用 1621
Amazon GuardDuty 1621
Amazon Local Zones 1622
Amazon Resilience Hub 1623
Amazon Secrets Manager 1623
在 Argo CD 中使用 Amazon Secrets Manager 1623
Amazon Security Lake 1625
将 Security Lake 与 Amazon EKS 结合使用的益处 1625
为 Amazon EKS 启用 Security Lake 1626
在 Security Lake 中分析 EKS 日志 1626
Amazon VPC Lattice 1626
Amazon EKS Connector 1627
Amazon EKS Connector 注意事项 1627
Amazon EKS Connector 所需的 IAM 角色 1628
连接集群 1628
注意事项 1628
先决条件 1628

步骤 1：注册集群 1629
步骤 2：安装 eks-connector 代理 1631
后续步骤 1633
授予对集群的访问权限 1633
先决条件 1633
注销集群 1635
注销 Kubernetes 集群 1635
清除 Kubernetes 集群中的资源 1636
EKS Connector 问题排查 1637
基本问题排查 1637
Helm 问题：403 Forbidden 1639
控制台错误：集群卡在待处理状态 1639
控制台错误：用户“system:serviceaccount:eks-connector:eks-connector”无法在集群范围内模拟 API 组中的资源用户 1639
控制台错误：[…] 已禁止：用户 […] 无法在集群范围内列出 API 组中的列表资源 […] 1640
控制台错误：Amazon EKS 无法与您的 Kubernetes 集群 API 服务器进行通信。集群必须处于 ACTIVE（活动）状态才能成功连接。过几分钟再试。 1640
Amazon EKS Connector 容器组（pod）处于崩溃循环 1641
无法启动 eks-connector：InvalidActivation 1641
集群节点缺少出站连接 1642
Amazon EKS Connector 容器组（pod）处于 ImagePullBackOff 状态 1642
常见问题 1643
安全性注意事项 1644
Amazon 责任 1644
客户责任 1644
Amazon EKS on Amazon Outposts 1646
何时使用每个部署选项 1646
部署选项比较 1647
运行本地集群 1649
支持的 Amazon 区域 1649
部署本地集群 1650
EKS 平台版本 1659
创建 VPC 和子网 1672
做好断网准备 1675
容量注意事项 1679
集群问题排查 1681

Nodes 1690
eksctl 1691
Amazon Web Services 管理控制台 1692
EKS 上的人工智能/机器学习 1699
为何选择 EKS 部署人工智能/机器学习？ 1699
关键使用案例 1699
案例研究 1700
开始在 EKS 上部署机器学习 1701
实时推理 1701
创建集群 1701
使用 vLLM 进行 LLM 推理 1735
集群配置 1751
使用 EKS Linux GPU AMI 1752
安装适用于 GPU 的设备插件 1756
设置 Windows GPU AMI 1762
使用 EFA 设置训练集群 1767
使用 Inferentia 设置推理集群 1777
容量管理 1783
为托管节点组预留 GPU 1783
为自行管理的节点预留 GPU 1786
将 P6e-GB200 与 EKS 搭配使用 1789
配方 1798
防止在特定节点上调度容器组（pod） 1799
资源 1800
研讨会 1801
最佳实践 1801
参考架构 1802
教程 1803
工具 1805
模型上下文协议（MCP） 1805
概述 83
集成示例 1806
开始使用 261
开始使用 1807
工具 1823
Amazon Q 1833

工作原理 434
使用 Amazon Q 进行问题排查 1834
注意事项 89
了解更多 1835
版本控制 1836
Kubernetes 版本 1836
标准支持上的可用版本 1836
延期支持的可用版本 1837
Amazon EKS Kubernetes 发布日历 1837
使用 Amazon CLI 获取版本信息 1838
Amazon EKS 版本常见问题 1839
Amazon EKS 扩展支持常见问题 1841
标准支持版本 1843
扩展支持版本 1847
查看支持期 1849
查看升级策略 1850
启用扩展支持 1852
禁用扩展支持 1853
平台版本 1853
Kubernetes 版1.34 1855
Kubernetes 版1.33 1855
Kubernetes 版1.32 1858
Kubernetes 版1.31 1861
Kubernetes 版1.30 1865
Kubernetes 版1.29 1870
获取当前平台版本 1875
更改平台版本 1876
故障排除 1877
容量不足 1877
节点未能加入集群 1877
未经授权或访问被拒绝 (kubectl) 1878
hostname doesn't match 1879
getsockopt: no route to host 1879
Instances failed to join the Kubernetes cluster 1880
托管节点组错误代码 1880
Not authorized for images 1884

节点处于 NotReady 状态 1884
EKS 日志收集器 1885
容器运行时网络未准备就绪 1886
TLS 握手超时 1887
InvalidClientTokenId 1887
在升级控制面板前，节点组必须匹配 Kubernetes 版本 1888
启动多个节点时，出现 Too Many Requests 错误 1888
Kubernetes API 服务器请求的 HTTP 401 未经授权的错误响应 1889
Amazon EKS 平台版本比当前平台版本落后两个版本以上 1889
集群运行状况常见问题解答和错误代码以及解析路径 1892
与 Amazon EKS 相关的项目 1897
支持部署到 EKS 的软件 1897
管理工具 1898
eksctl 1898
Amazon Controllers for Kubernetes 1898
kro (Kube Resource Orchestrator) 1898
Argo CD 1898
Flux CD 1899
CDK for Kubernetes 1899
Networking 1899
适用于 Kubernetes 的 Amazon VPC CNI 插件 1899
Amazon适用于 Kubernetes 的负载均衡器控制器 1900
ExternalDNS 1900
Machine Learning 1900
Kubeflow 1900
Auto Scaling 1900
Cluster Autoscaler 1900
Karpenter 1901
Escalator 1901
监控 1901
Prometheus 1901
持续集成/持续部署 1901
Jenkins X 1902
新增功能和路线图 1903
文档历史记录 1904
改进 1947

编辑单个页面 1947
先决条件 45
过程 91
拉取请求概述 1949
使用 GitHub 编辑文件 1950
先决条件 45
过程 91
查看样式反馈 1951
安装 Vale 1952
安装 VS Code Vale 扩展 1952
同步 Vale 1953
在 VS Code 中查看样式反馈 1953
创建页面 1953
创建页面 1953
将页面添加到导航 1954
插入链接 1954
链接到《EKS 用户指南》中的页面或章节 1954
链接到 Amazon 文档中的其他指南 1955
链接到外部网页 1955
使用 Amazon Q 创建 1955
使用 VS Code 安装 Amazon Q 1956
登录 Amazon Q 1956
使用 Amazon Q 创建内容 1957
提示 1957
查看 PR 预览 1957
查看预览 1958
预览限制 1959
AsciiDoc 语法 1959
新页面 1959
标题 1959
文本格式设置 1960
Lists 1960
链接 1960
代码示例 1961
映像 1961
表 1961

标注 1962
包含其他文件 1962
属性（类似于实体） 1962
过程 1963

帮助改进此页面

要帮助改进本用户指南，请选择位于每个页面右侧窗格中的在 GitHub 上编辑此页面链接。

什么是 Amazon EKS？

Amazon EKS：简化 Kubernetes 管理

Amazon Elastic Kubernetes Service（EKS）提供完全托管式的 Kubernetes 服务，让 Kubernetes 集群的运行不再复杂。使用 EKS，您可以：

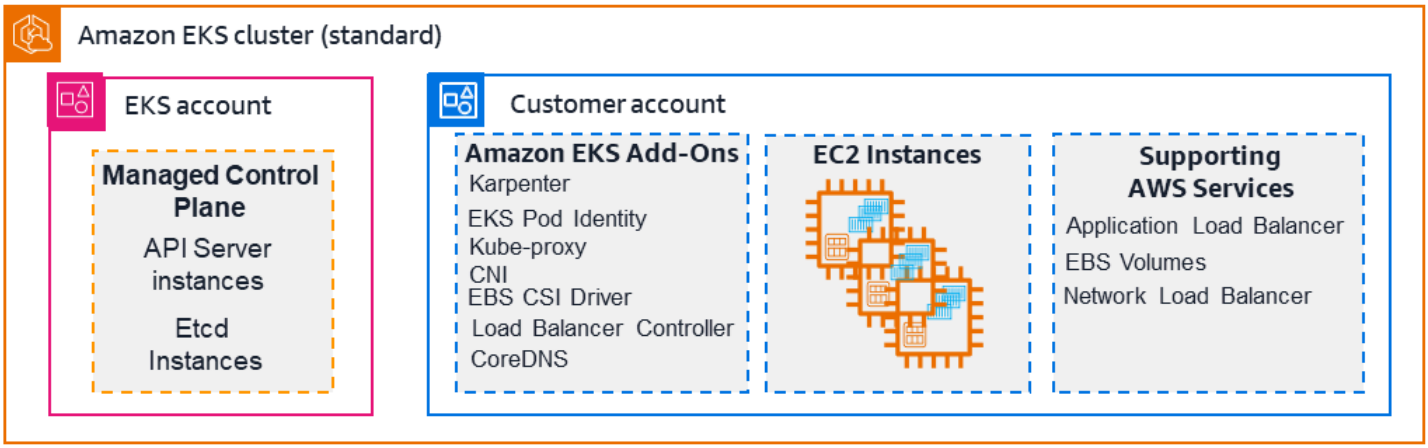
- 以更少的运营开销更快地部署应用程序
- 无缝扩展以满足不断变化的工作负载需求
- 通过 Amazon 集成和自动更新提高安全性
- 可选择标准版 EKS 或完全自动化的 EKS 自动模式

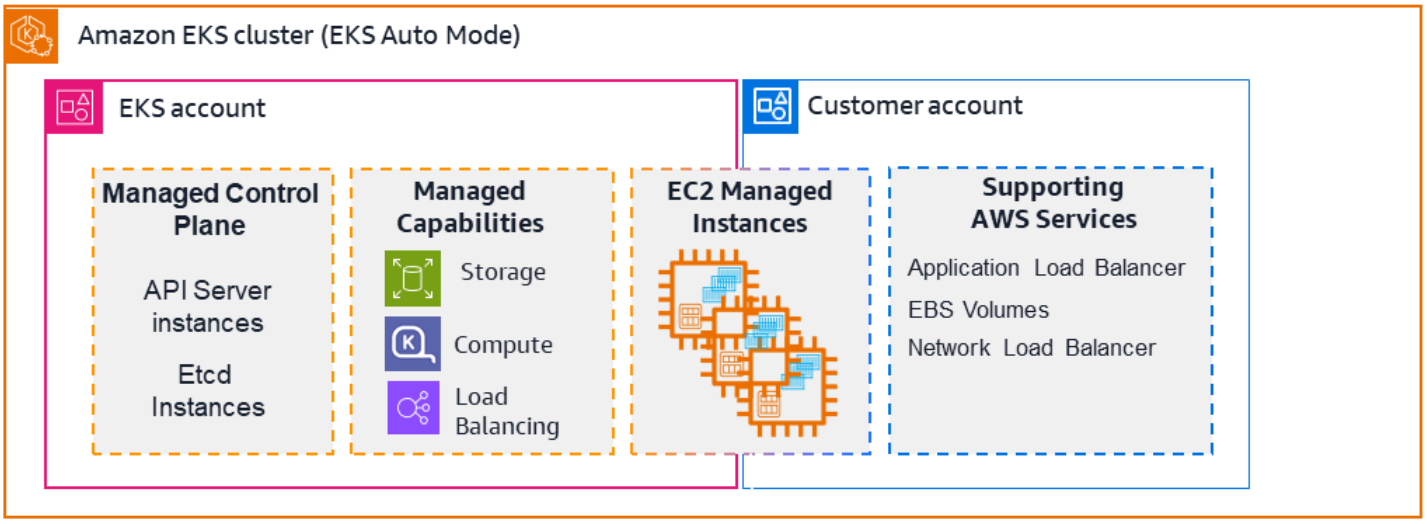
Amazon Elastic Kubernetes Service（Amazon EKS）是运行 Kubernetes 集群的优选平台，无论是在亚马逊云科技（Amazon）云端还是在您自己的数据中心（EKS Anywhere 和 Amazon EKS 混合节点功能）中。

Amazon EKS 简化了 Kubernetes 集群的构建、保护和维护过程。在提供充分的资源来满足高峰需求方面，比自行维护数据中心更经济高效。使用 Amazon EKS 的方法主要有两种，具体如下：

- EKS 标准模式：当您使用 EKS 创建集群时，Amazon 负责管理 Kubernetes 控制面板。为确保集群正常运行所需的节点管理、工作负载调度、Amazon 云集成以及控制面板信息的存储和扩展等组件，都由服务自动为您处理。
- EKS 自动模式：使用 EKS 自动模式功能时，EKS 的控制范围将扩展到管理节点（Kubernetes 数据面板）。通过自动预置基础设施、选择最佳计算实例、动态扩展资源、持续优化成本、修补操作系统以及与 Amazon 安全服务集成，简化了 Kubernetes 的管理。

下图演示了 Amazon EKS 会如何根据您选择的集群创建方法，将 Kubernetes 集群与 Amazon 云集成：





Amazon EKS 有助于您消除摩擦，加快投产速度，提高性能、可用性和韧性，以及增强系统安全性。有关更多信息，请参阅 Amazon Elastic Kubernetes Service。

使用 Kubernetes 进行构建和扩展：Amazon EKS 功能

Amazon EKS 不仅可以帮助您构建和管理集群，还可以帮助您使用 Kubernetes 构建和扩展应用程序系统。Amazon EKS 功能是完全托管的集群服务，可通过免提 Kubernetes 原生工具扩展集群的功能，包括：

- Argo CD：Argo CD 可为您的工作负载、Amazon 资源和云基础设施提供基于 GitOps 的声明性持续部署。
- Amazon Controllers for Kubernetes（ACK）：ACK 支持 Amazon 资源的 Kubernetes 原生创建和生命周期管理，并统一了工作负载编排和基础设施即代码工作流程。
- kro（Kube Resource Orchestrator）：kro 可扩展 Kubernetes 原生功能，以简化自定义资源的创建、编排与组合，并为您提供创建自己的自定义云构建块所需的工具。

EKS 功能是一系列云资源，可最大限度地减少在集群中安装、维护和扩展这些基础平台组件的运营负担，让您可以专注于构建软件而不是集群平台操作。

要了解更多信息，请参阅EKS 功能。

Amazon EKS 的功能

Amazon EKS 提供以下高级功能：

管理界面

EKS 支持多个可用于预置、管理和维护集群的接口，包括 Amazon Web Services 管理控制台、Amazon EKS API/SDK、CDK、Amazon CLI、eksctl CLI、Amazon CloudFormation 和 Terraform 等。有关更多信息，请参阅开始使用和配置群集。

访问控制工具

EKS 依赖 Kubernetes 和 Amazon Identity and Access Management（Amazon IAM）功能，来进行用户和工作负载的访问权限管理。有关更多信息，请参阅the section called “Kubernetes API 访问权限”和the section called “Amazon 工作负载访问权限 ”。

计算资源

对于计算资源，EKS 支持将各种 Amazon EC2 实例类型和 Amazon 创新（例如 Nitro 和 Graviton）与 Amazon EKS 结合使用，从而优化工作负载的计算。有关更多信息，请参阅 管理计算。

仓储服务

EKS 自动模式使用 EBS 卷自动创建存储类。借助容器存储接口（CSI）驱动程序，您还可以使用 Amazon S3、Amazon EFS、Amazon FSX 和 Amazon File Cache 来满足应用程序存储需求。有关更多信息，请参阅 应用程序数据存储。

安全性。

采用责任共担模式，因为这与 Amazon EKS 中的安全性有关。有关更多信息，请参阅安全最佳实践、基础设施安全性和 Kubernetes 安全性。

监控工具

可使用可观测性控制面板来监控 Amazon EKS 集群。监控工具包括 Prometheus、CloudWatch、Cloudtrail 和 ADOT Operator。有关控制面板、指标服务器和其他工具的更多信息，请参阅 EKS 集群成本和 Kubernetes 指标服务器。

集群功能

EKS 根据开源创新，为持续部署、云资源管理和资源组合提供托管式集群功能。EKS 在您的集群中安装 Kubernetes API，但控制器和其他组件在 EKS 中运行并完全托管，从而提供自动修补、扩展和监控。有关更多信息，请参阅 EKS 功能。

Kubernetes 兼容性与支持

Amazon EKS 已通过 Kubernetes 符合性认证，因此无需重构即可部署与 Kubernetes 兼容的应用程序，并且可以使用 Kubernetes 社区工具和插件。EKS 为 Kubernetes 提供标准支持和扩展支持。有关更多信息，请参阅 the section called “Kubernetes 版本”。

相关服务

可与 Amazon EKS 结合使用的服务

可以将其他 Amazon 服务与使用 Amazon EKS 部署的集群一起使用：

（Amazon EC2）

使用 Amazon EC2 获得可扩展的按需型计算容量。

Amazon EBS

使用 Amazon EBS 挂载可扩展的高性能块存储资源。

Amazon ECR ()

使用 Amazon ECR 安全地存储容器映像。

Amazon CloudWatch

使用 Amazon CloudWatch 实时监控 Amazon 资源和应用程序。

Amazon Prometheus

使用 Amazon Managed Service for Prometheus 跟踪容器化应用程序的指标。

Elastic Load Balancing

使用弹性负载均衡在多个目标之间分配传入的流量。

Amazon GuardDuty

使用 Amazon GuardDuty 检测 EKS 集群面临的威胁。

Amazon 韧性监测中心

使用 Amazon 韧性监测中心评估 EKS 集群的韧性。

Amazon EKS 定价

Amazon EKS 的定价包括基于 Kubernetes 集群版本支持的每集群定价、Amazon EKS 自动模式定价以及 Amazon EKS 混合节点功能的每 vCPU 定价。

使用 Amazon EKS 时，您需要为在 Kubernetes Worker 节点上运行应用程序所使用的 Amazon 资源单独付费。例如，假设您将 Kubernetes Worker 节点作为具有 Amazon EBS 卷和公有 IPv4 地址的 Amazon EC2 实例运行，则需要为 Amazon EC2 实例容量、Amazon EBS 卷容量以及 Amazon VPC 的 IPv4 地址付费。

请访问您用于 Kubernetes 应用程序的 Amazon 服务的相应定价页面，了解详细定价信息。

- 对于 Amazon EKS 集群、Amazon EKS 自动模式、Amazon EKS 功能和 Amazon EKS 混合节点功能定价，请参阅 Amazon EKS 定价。
- 对于 Amazon EC2 定价，请参阅 Amazon EC2 按需型定价和 Amazon EC2 竞价型定价。
- 对于 Amazon Fargate 定价，请参阅 Amazon Fargate 定价。
- 您可以将自己的节省计划用于在 Amazon EKS 集群中使用的计算。有关更多信息，请参阅 Savings Plans 定价。

Amazon EKS 中的常见使用案例

Amazon EKS 在 Amazon 上提供强大的 Kubernetes 托管服务，旨在优化容器化应用程序。以下是 Amazon EKS 的一些常见使用案例，可帮助您利用其优势来满足您的特定需求。

部署高可用性应用程序

借助 Elastic Load Balancing，可以确保您的应用程序在多个可用区高度可用。

构建微服务架构

将 Kubernetes 服务发现功能与 Amazon Cloud Map 或 Amazon VPC Lattice 结合使用，构建弹性系统。

自动化软件发布流程

管理持续集成和持续部署（CI/CD）管道，简化应用程序的自动化构建、测试和部署流程。有关声明式持续部署，请参阅使用 Argo CD 进行持续部署。

运行无服务器应用程序

将 Amazon Fargate 与 Amazon EKS 结合使用，运行无服务器应用程序。这意味着您可以只专注于应用程序开发，由 Amazon EKS 和 Fargate 来负责底层基础设施。

执行机器学习工作负载

Amazon EKS 与流行的机器学习框架兼容，比如 TensorFlow、MXNet 和 PyTorch。借助 GPU 支持，您甚至可以有效处理复杂的机器学习任务。

在本地和云端一致部署

为了简化在本地环境中运行 Kubernetes 的过程，您可以使用相同的 Amazon EKS 集群、功能和工具，在 Amazon Outposts 上运行自主管理型节点，也可以在自己的基础设施上使用 Amazon EKS 混合节点功能。对于独立的气隙环境，您可以使用 Amazon EKS Anywhere 在自己的基础设施上自动管理 Kubernetes 集群的生命周期。

运行经济高效的批处理和大数据工作负载

只需很小的成本即可利用竞价型实例运行批处理和大数据工作负载，比如 Apache Hadoop 和 Spark。这让您能以折扣价利用未使用的 Amazon EC2 容量。

管理来自 Kubernetes 的 Amazon 资源

使用 Amazon Controllers for Kubernetes（ACK），使用原生 Kubernetes API 直接从您的 Kubernetes 集群创建和管理 Amazon 资源。

构建平台工程抽象

使用 kro（Kube Resource Orchestrator）创建自定义 Kubernetes API，将多个资源组合成更高级别的抽象。

保护应用程序并确保合规性

实施强大的安全实践并保持对 Amazon EKS 的合规，它与 Amazon Identity and Access Management（IAM）、Amazon Virtual Private Cloud（Amazon VPC）和 Amazon Key Management Service（Amazon KMS）等 Amazon 安全服务集成。这确保了数据隐私和保护符合行业标准。

Amazon EKS 架构

Amazon EKS 与 Kubernetes 的通用集群架构保持一致。有关更多信息，请参阅 Kubernetes 文档中的 Kubernetes 组件。以下部分介绍了 Amazon EKS 的一些额外架构细节。

控制面板

Amazon EKS 确保每个集群都有自身独有的 Kubernetes 控制面板。这种设计使每个集群的基础设施保持独立，集群或 Amazon 账户之间没有重叠。该设置包括：

分布式组件

控制面板将至少两个 API 服务器实例和三个 etcd 实例放置在 Amazon 区域内的三个 Amazon 可用区中。

最佳性能

Amazon EKS 会主动监控和调整控制面板实例，以保持最高性能。

恢复能力

如果控制面板实例发生故障，Amazon EKS 会迅速将其替换，必要时会使用不同的可用区。

稳定的正常运行时间

通过在多个可用区运行集群，实现了可靠的 API 服务器端点可用性服务水平协议（SLA）。

Amazon EKS 使用 Amazon Virtual Private Cloud（Amazon VPC）来限制单个集群中控制面板组件之间的流量。除非得到 Kubernetes 基于角色的访问控制（RBAC）策略授权，否则，集群组件无法查看或接收来自其他集群或其他 Amazon 账户的通信。

计算

除了控制面板，Amazon EKS 集群还有一组称为节点的工作计算机。选择适当的 Amazon EKS 集群节点类型对于满足您的特定要求和优化资源利用率至关重要。Amazon EKS 提供以下主节点类型：

EKS 自动模式

EKS 自动模式将 Amazon 管理扩展到控制面板之外，将数据面板也包括在内，从而实现集群基础设施管理的自动化。自动模式将 Kubernetes 的核心功能集成为内置组件，包括计算自动扩缩、联网、负载均衡、DNS、存储和 GPU 支持等。EKS 自动模式使用具有增强安全功能的不可变 AMI，根据工作负载需求动态管理节点。在尊重容器组中断预算的同时自动更新和升级，并且包含原本需

要附加组件管理的托管式组件。此选项非常适合希望利用 Amazon 的专业知识满足日常运营需要，尽可能减少运营开销，并专注于应用程序开发而不是基础设施管理的用户。

AmazonFargate

Fargate 是一个用于容器的无服务器计算引擎，无需管理底层实例。借助 Fargate，您可以指定应用程序的资源需求，然后 Amazon 自动预置、扩展和维护基础设施。对于优先考虑易用性并希望专注于应用程序开发和部署而不是管理基础设施的用户来说，此选项非常适合。

Karpenter

Karpenter 是一款灵活的高性能 Kubernetes 集群自动扩缩器，可帮助提高应用程序可用性和集群效率。Karpenter 可启动适当规模的计算资源来响应不断变化的应用程序负载。此选项可以预置即时计算资源，以满足您的工作负载要求。

托管节点组

托管节点组是自动化和自定义的混合体，用于管理 Amazon EKS 集群中的 Amazon EC2 实例集合。Amazon 负责修补、更新和扩展节点之类的任务，从而简化操作方面的工作。同时，还支持自定义 `kubelet` 参数，为高级 CPU 和内存管理策略提供了可能性。此外，它们还通过服务账户的 Amazon Identity and Access Management（IAM）角色增强安全性，同时限制每个集群对单独权限的需求。

自行管理的节点

自行管理节点可完全控制 Amazon EKS 集群中的 Amazon EC2 实例。您负责管理、扩展和维护节点，从而完全控制底层基础设施。对于需要对其节点进行精细控制和自定义，并准备投入时间管理和维护其基础设施的用户来说，此选项非常适合。

Amazon EKS 混合节点功能

借助 Amazon EKS 混合节点功能，您可以将本地和边缘基础设施作为 Amazon EKS 集群中的节点。Amazon EKS 混合节点功能实现了跨环境的统一 Kubernetes 管理，将本地和边缘应用程序的 Kubernetes 控制面板管理工作转移到 Amazon。

EKS 功能

Amazon EKS 提供完全托管的集群功能，会在集群中安装并管理 Kubernetes API（包含 Kubernetes 自定义资源定义），同时在 Amazon 自有基础设施中运行控制器及其他组件，与您的集群相互隔离。EKS 会对这些功能进行自动化补丁更新、扩缩与监控，对其生命周期进行完全托管，从而减轻您在运行集群内工作负载编排、Amazon 资源管理等相关服务时的运维负担。

EKS 提供以下功能类型：

Amazon Controllers for Kubernetes（ACK）

Amazon Controllers for Kubernetes（ACK）支持您通过 Kubernetes API 管理 Amazon 资源，您可将 S3 存储桶、RDS 数据库、IAM 角色及其他 Amazon 资源定义为 Kubernetes 自定义资源。您能够使用统一的工具与流程，将 Amazon 资源和 Kubernetes 工作负载纳入同一管理体系，该工具支持包括 S3、RDS、DynamoDB 及 Lambda 在内的 50 余种 Amazon 服务。

Argo CD

Argo CD 基于 GitOps 理念为应用工作负载、Amazon 资源及集群配置实现持续部署功能，并将 Git 存储库作为基准依据。Argo CD 会自动将集群与 Git 存储库进行同步，同时检测配置偏差，通过持续协调来确保已部署的应用和资源，与版本控制系统中定义的预期状态保持一致。您可借助 Argo CD 管理指定集群上的应用，也能通过单个 Argo CD 资源跨多个集群部署并管理应用；当 Git 存储库中提交变更时，该工具会自动触发部署流程。

kro（Kube Resource Orchestrator）

kro（Kube Resource Orchestrator）支持您创建自定义 Kubernetes API，可将多种资源组合为更高级别的抽象资源，方便平台团队针对常见的资源组合模式，定义可复用的标准化模式。借助这一能力，平台团队能够为开发者提供带有合规管控规则的自助服务功能，让开发者通过简洁且面向特定场景的 API 即可配置复杂基础设施，同时保障相关配置始终符合企业规范与最佳实践。

Kubernetes 概念

Amazon Elastic Kubernetes Service（Amazon EKS）是一项基于开源 Kubernetes 项目的 Amazon 托管服务。虽然需要了解 Amazon EKS 服务如何与 Amazon Cloud 集成（尤其是首次创建 Amazon EKS 集群时），但只要启动并运行起来，您就可以像使用任何其他 Kubernetes 集群那样使用自己的 Amazon EKS 集群。因此，要开始管理 Kubernetes 集群和部署工作负载，您至少需要对 Kubernetes 概念有基本的了解。

本页将 Kubernetes 概念分为三个部分：“the section called “为何选择 Kubernetes？””“the section called “集群””和“the section called “ 工作负载””。第一部分介绍了运行 Kubernetes 服务的价值，尤其是作为托管服务（如 Amazon EKS）运行的价值。“工作负载”部分介绍如何构建、存储、运行和管理 Kubernetes 应用程序。“集群”部分列出构成 Kubernetes 集群的不同组件，以及您在创建和维护 Kubernetes 集群方面的责任。

主题

- 为何选择 Kubernetes？
- 集群

- 工作负载
- 后续步骤

在浏览本内容时，如果您想深入了解我们在此处介绍的任何主题，则链接将引导您进一步了解 Amazon EKS 和 Kubernetes 文档中的 Kubernetes 概念。有关 Amazon EKS 如何实施 Kubernetes 控制面板和计算功能的详细信息，请参阅 the section called “架构”。

为何选择 Kubernetes？

Kubernetes 旨在提高运行任务关键型、生产质量容器化应用程序时的可用性和可扩展性。Kubernetes 不仅仅是在一台计算机上运行 Kubernetes（尽管可以这样做），还通过允许您在可以扩缩来满足需求的计算机组上运行应用程序来实现这些目标。Kubernetes 内含的功能有助于执行以下操作：

- 在多台机器上部署应用程序（使用部署在容器组（pod）中的容器）
- 监控容器运行状况并重启失败的容器
- 根据负载纵向扩展和缩减容器
- 使用新版本更新容器
- 在容器之间分配资源
- 平衡计算机间的流量

通过让 Kubernetes 自动执行这些类型的复杂任务，应用程序开发人员可以专注于构建和改进其应用程序工作负载，而不必担心基础设施。开发人员通常会创建格式为 YAML 文件的配置文件，这些文件用于描述应用程序的所需状态。这可能包括要运行的容器、资源限制、容器组（pod）副本数量、CPU/内存分配、关联性规则等等。

Kubernetes 属性

为实现其目标，Kubernetes 具有以下属性：

- 容器化 – Kubernetes 是一种容器编排工具。要使用 Kubernetes，您必须先将应用程序容器化。根据应用程序的类型，它可以是一组微服务、批处理作业或其他形式。然后，您的应用程序可以利用包含庞大工具生态系统的 Kubernetes 工作流，在该工作流中，容器可以作为映像存储在容器注册表中，部署到 Kubernetes 集群，然后在可用节点上运行。在将各个容器部署到 Kubernetes 集群之前，您可以使用 Docker 或其他容器运行时在本地计算机上构建和测试这些容器。
- 可扩缩 – 如果对应用程序的需求超过这些应用程序正在运行的实例容量，则 Kubernetes 可以纵向扩展。根据需要，Kubernetes 可以判断应用程序是否需要更多 CPU 或内存，并通过自动扩展可用容

量或使用更多的现有容量来做出响应。如果有足够的计算能力来运行更多的应用程序实例（水平容器组（pod）自动缩放），则可以在容器组（pod）级别进行扩展；如果需要调出更多节点来处理增加的容量（集群 Autoscaler 或 Karpenter），则可以在节点级别进行扩展。由于不再需要容量，这些服务可以删除不必要的容器组（pod）并关闭不需要的节点。

- 可用 – 如果某个应用程序或节点运行状况不佳或不可用，Kubernetes 可以将正在运行的工作负载移至其他可用节点。您只需删除正在运行的工作负载实例或正在运行您的工作负载的节点即可强制解决问题。这里的底线是，如果工作负载不能再在当前位置运行，则可以将其调到其他位置。
- 声明式 – Kubernetes 使用主动协调来持续检查您为集群声明的状态是否与实际状态相符。例如，通过将 Kubernetes 对象应用于集群（通常通过 YAML 格式的配置文件），您可以要求启动要在集群上运行的工作负载。您可以稍后更改配置，执行使用更高版本的容器或分配更多内存这样的操作。Kubernetes 将尽其所能建立所需的状态。这可能包括启动或关闭节点、停止和重启工作负载，或拉取更新的容器。
- 可组合 – 由于应用程序通常由多个组件组成，因此您会希望能够将某个组件组合（通常由多个容器表示）统一管理。虽然 Docker Compose 提供使用 Docker 直接执行此操作的方法，但 Kompose 命令可以帮助您使用 Kubernetes 执行该操作。有关如何执行此操作的示例，请参阅将 Docker Compose 文件翻译成 Kubernetes 资源。
- 可扩展 – 与专有软件不同的是，Kubernetes 开源项目采用开放设计，能让您以所需方式扩展 Kubernetes 来满足需求。API 和配置文件可以直接修改。我们鼓励第三方编写自己的控制器，以同时扩展基础设施和最终用户 Kubernetes 功能。Webhook 使您能够设置集群规则，以强制执行策略并适应不断变化的条件。有关如何扩展 Kubernetes 集群的更多想法，请参阅扩展 Kubernetes。
- 可移植 – 许多组织已将其在 Kubernetes 上的操作标准化，从而能够以相同的方式管理所有应用程序需求。开发人员可以使用相同的管道来构建并存储容器化应用程序。然后，可以将这些应用程序部署到在本地、云、餐厅的销售点终端上，或分散在公司远程站点的 IOT 设备上运行的 Kubernetes 集群中。开源性质使人们有可能开发这些特殊的 Kubernetes 发行版，以及管理这些发行版所需的工具。

管理 Kubernetes

Kubernetes 源代码免费提供，因此您可以使用自己的设备自行安装和管理 Kubernetes。但是，自行管理 Kubernetes 需要深厚的运营专业知识，并且需要时间和精力来维护。出于这些原因，大多数部署生产工作负载的人员都会选择云提供商（例如 Amazon EKS）或本地提供商（例如 Amazon EKS Anywhere），这些提供商拥有经过测试的 Kubernetes 发行版和 Kubernetes 专家支持。这使您可以卸下维护集群所需的大量千篇一律的工作，包括：

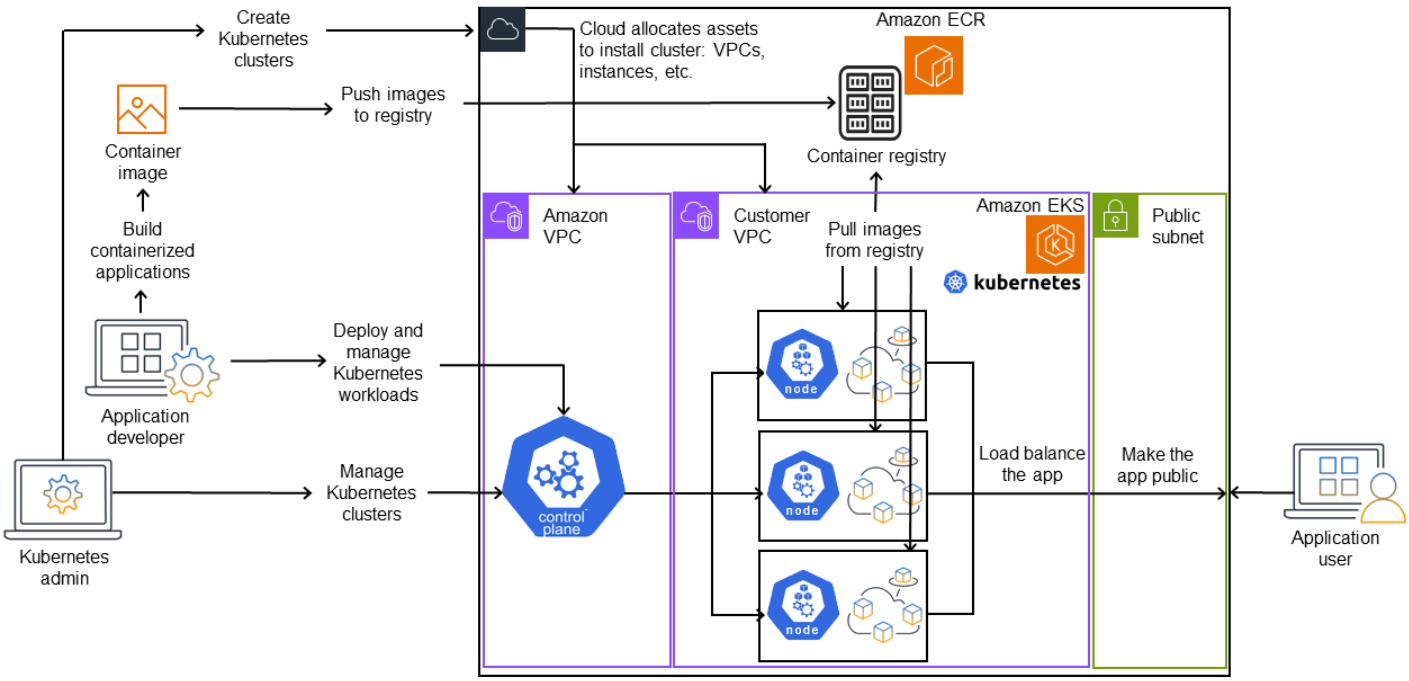
- 硬件 – 如果您不具备可根据要求运行 Kubernetes 的硬件，则 Amazon Amazon EKS 等云提供商可以为您节省前期成本。借助 Amazon EKS，这意味着您可以使用 Amazon 提供的最佳云资源，包括计算机实例（Amazon Elastic Compute Cloud）、您自己的私有环境（Amazon VPC）、中央

身份和权限管理（IAM）和存储（Amazon EBS）。Amazon 管理计算机、网络、数据中心和运行 Kubernetes 所需的所有其他物理组件。同样，您不必计划数据中心在需求最高的时期处理最大容量。对于 Amazon EKS Anywhere 或其他本地 Kubernetes 集群，您负责管理 Kubernetes 部署中使用的基础设施，但您仍然可以依靠 Amazon 让 Kubernetes 保持最新状态。

- 控制面板管理 – Amazon EKS 负责管理 Amazon 托管的 Kubernetes 控制面板的安全性和可用性，控制面板负责调度容器、管理应用程序的可用性以及其他关键任务，让您能够专注于应用程序工作负载。如果您的集群中断，则 Amazon 应该有办法将集群恢复到运行状态。对于 Amazon EKS Anywhere，您将自己管理控制面板。
- 经过测试的升级 – 升级集群时，Amazon EKS 或 Amazon EKS Anywhere 会提供其经测试的 Kubernetes 发行版。
- 附加组件 – 有数百个项目为扩展和使用 Kubernetes 而构建，您可以将其添加到集群的基础设施中，也可以用它们来辅助工作负载的运行。Amazon 并未自己构建和管理这些附加组件，而是提供可用于集群的 the section called “Amazon EKS 附加组件”。Amazon EKS Anywhere 提供精选软件包，其中包括许多热门开源项目的版本。因此，您不必自己构建软件，也不必管理关键的安全补丁、漏洞修复或升级。同样，如果默认设置满足您的需求，则通常只需要对这些插件进行很少的配置。有关使用插件扩展集群的详细信息，请参阅the section called “扩展集群”。

Kubernetes 的实际应用

下图显示您作为 Kubernetes 管理员或应用程序开发人员在创建和使用 Kubernetes 集群时要执行的关键活动。在此过程中，以 Amazon 云作为底层云提供商为例，说明了 Kubernetes 组件是如何相互交互的。



Kubernetes 管理员使用特定于将在其上构建集群的提供商类型的工具来创建 Kubernetes 集群。此示例使用 Amazon 云作为提供商，提供名为 Amazon EKS 的托管 Kubernetes 服务。此托管式服务会自动分配创建集群所需的资源，包括为集群创建两个新的虚拟私有云（Amazon VPC）、设置联网以及将 Kubernetes 权限直接映射到用于云资产管理的新 VPC。此托管式服务还会确认存在可用于运行控制面板服务的空间，并分配零个或多个 Amazon EC2 实例以作为运行工作负载的 Kubernetes 节点。Amazon 会管理用于控制面板的一个 Amazon VPC 本身，而另一个 Amazon VPC 则包含运行工作负载的客户节点。

Kubernetes 管理员今后的许多任务都是使用 `kubectl` 之类的 Kubernetes 工具完成的。该工具直接向集群的控制面板发出服务请求。因此，对集群进行查询和更改的方式与您在任何 Kubernetes 集群上执行查询和更改的方式非常相似。

想要将工作负载部署到此集群的应用程序开发人员可以执行多项任务。开发人员需要将应用程序构建到一个或多个容器映像中，然后将这些映像推送到 Kubernetes 集群可以访问的容器注册表。Amazon 为此提供 Amazon Elastic Container Registry（Amazon ECR）。

要运行应用程序，开发人员可以创建 YAML 格式的配置文件，告诉集群如何运行应用程序，包括要从注册表中提取哪些容器以及如何将这些容器封装在容器组（pod）中。控制面板（调度程序）将容器调度到一个或多个节点，每个节点上的容器运行时实际上会拉取并运行所需的容器。开发人员还可以设置

应用程序负载均衡器，以均衡在每个节点上运行的可用容器的流量，并公开应用程序，使其可以在公共网络上对外开放。完成所有操作后，想要使用应用程序的人可以连接到应用程序端点进行访问。

以下部分将从 Kubernetes 集群和工作负载的角度详细介绍每项功能。

集群

如果作业是启动和管理 Kubernetes 集群，则应了解如何创建、增强、管理和删除 Kubernetes 集群。您还应该了解构成集群的组件是什么，以及需要做些什么来维护这些组件。

用于管理集群的工具可以处理 Kubernetes 服务与底层硬件提供商之间的重叠问题。因此，这些任务的自动化往往由 Kubernetes 提供商（例如 Amazon EKS 或 Amazon EKS Anywhere）使用特定于提供商的工具来完成。例如，要启动 Amazon EKS 集群，您可以使用 `eksctl create cluster`，而对于 Amazon EKS Anywhere，您可以使用 `eksctl anywhere create cluster`。请注意，虽然这些命令创建 Kubernetes 集群，但它们是特定于提供商的，而不是 Kubernetes 项目本身的一部分。

集群创建和管理工具

Kubernetes 项目提供手动创建 Kubernetes 集群的工具。因此，如果要在单台计算机上安装 Kubernetes，或者在计算机上运行控制面板并手动添加节点，则可以使用 Kubernetes 安装工具下列出的 kind、minikube 或 kubeadm 之类的 CLI 工具。要简化和自动执行集群创建和管理的整个生命周期，使用现有 Kubernetes 提供商（例如 Amazon EKS 或 Amazon EKS Anywhere）支持的工具要容易得多。

在 Amazon Cloud 中，您可以使用 CLI 工具（例如 eksctl）或更多声明性工具（例如 Terraform）创建 Amazon EKS 集群（请参阅 Terraform 的 Amazon EKS 蓝图）。您还可以从 Amazon Web Services 管理控制台中创建集群。要了解使用 Amazon EKS 能够获得的能力，请参阅 Amazon EKS 功能。Amazon EKS 代您承担的 Kubernetes 职责包括：

- 托管式控制面板 – Amazon 为您管理控制面板并使其跨 Amazon 可用区可用，从而确保 Amazon EKS 集群可用且可扩缩。
- 节点管理 – 您可以使用托管式节点组（请参阅the section called “托管节点组”）或 Karpenter，让 Amazon EKS 根据需要自动创建节点，而无需手动添加节点。托管节点组已与 Kubernetes 集群自动扩缩集成。使用节点管理工具，您可以利用竞价型实例、节点整合及可用性等优势来节省成本，并使用计划功能来设置工作负载的部署方式和节点的选择方式。
- 集群联网 – `eksctl` 使用 CloudFormation 模板在 Kubernetes 集群中的控制面板和数据面板（节点）组件之间设置联网。它还设置可以通过其进行内部和外部通信的端点。有关详细信息，请参阅揭秘 Amazon EKS 工作线程节点的集群联网。Amazon EKS 中容器组（pod）之间的通信使用

Amazon EKS 容器组身份完成（请参阅the section called “容器组身份”），这提供了一种让容器组（pod）利用 Amazon 云方法来管理凭证和权限的方式。

- 附加组件 – 使用 Amazon EKS，您无需构建和添加支持 Kubernetes 集群的常用软件组件。例如，当您从 Amazon Web Services 管理控制台创建 Amazon EKS 集群时，系统会自动添加 Amazon EKS kube-proxy（the section called “kube-proxy”）、适用于 Kubernetes 的 Amazon VPC CNI 插件（the section called “Amazon VPC CNI”）和 CoreDNS（the section called “CoreDNS”）附加组件。有关这些附加组件的更多信息，包括可用附件组件列表，请参阅 the section called “Amazon EKS 附加组件”。

为了在您自己的本地计算机和网络上运行集群，Amazon 提供了 Amazon EKS Anywhere。您可以选择使用自己的设备在 VMWare vSphere、裸机（Tinkerbell 提供程序）、Snow、CloudStack 或 Nutanix 平台上运行 Amazon EKS Anywhere，而不是将 Amazon Cloud 作为提供商。

Amazon EKS Anywhere 基于 Amazon EKS 使用的相同 Amazon EKS Distro 软件。但是，Amazon EKS Anywhere 依赖 Kubernetes 集群 API（CAPI）接口的不同实现来管理 Amazon EKS Anywhere 集群中计算机的整个生命周期（例如 vSphere 的 CAPV 和 CloudStack 的 CAPC）。由于整个集群都在您的设备上运行，因此您还要承担管理控制面板和备份其数据的额外责任（请参阅本文档后面的 `etcd`）。

集群组件

Kubernetes 集群组件分为两个主要区域：控制面板和 Worker 节点。控制面板组件管理集群并提供对其 API 的访问权限。Worker 节点（有时简称为“节点”）提供运行实际工作负载的地方。节点组件由在每个节点上运行的服务组成，这些服务用于与控制面板通信并运行容器。您的集群的 Worker 节点组称为数据面板。

控制面板

控制面板由一组管理集群的服务组成。这些服务可能全部在单台计算机上运行，也可能分布在多台计算机上。在内部，它们被称为控制面板实例（CPI）。CPI 的运行方式取决于集群的大小和对高可用性的要求。随着集群中的需求增加，控制面板服务可以扩展以提供该服务的更多实例，并在实例之间实现请求的负载均衡。

Kubernetes 控制面板组件执行的任务包括：

- 与集群组件（API 服务器）通信 – API 服务器（kube-apiserver）会公开 Kubernetes API，因此可以从集群内部和外部都发出对集群的请求。换句话说，添加或更改集群对象（容器组（pod）、服务、节点等）的请求可能来自外部命令，例如来自 `kubectl` 的运行容器组（pod）的请求。同样，可以从 API 服务器向集群内的组件发出请求，例如向 `kubelet` 服务查询容器组（pod）的状态。

- 存储有关集群的数据（**etcd** 键值存储）– etcd 服务提供了跟踪集群当前状态的关键功能。如果 etcd 服务变得不可访问，则您将无法更新或查询集群的状态，尽管工作负载会继续运行一段时间。因此，关键集群通常会同时运行多个负载均衡的 etcd 服务实例，并定期备份 etcd 键值存储以防数据丢失或损坏。请记住，在 Amazon EKS 中，这一切都是默认自动为您处理的。Amazon EKS Anywhere 提供 etcd 备份和恢复的说明。请参阅 etcd Data Model，以了解 etcd 管理数据的方式。
- 将容器组（pod）调度到节点（调度程序）– 有关启动或停止 Kubernetes 中容器组（pod）的请求会被定向到 Kubernetes 调度器（kube-scheduler）。由于一个集群可能有多个能够运行容器组（pod）的节点，因此应由调度程序来选择容器组（pod）应在哪个节点（或哪些节点，如果是副本）上运行。如果没有足够的可用容量在现有节点上运行所请求的容器组（pod），除非您进行了其他规定，否则请求将失败。此类预置可能包括启用托管式节点组（the section called “托管节点组”）或 Karpenter 之类的服务，这些服务可以自动启动新节点来处理工作负载。
- 将组件保持在所需状态（控制器管理器）– Kubernetes 控制器管理器作为进程守护程序（kube-controller-manager）运行，可监视集群的状态，以及更改集群来重新建立预期状态。特别是，有几个控制器可以监视不同的 Kubernetes 对象，包括 statefulset-controller、endpoint-controller、cronjob-controller、node-controller。
- 管理云资源（云控制器管理器）– Kubernetes 与执行底层数据中心资源请求的云提供商之间的交互由云控制器管理器（cloud-controller-manager）处理。由云控制器管理器管理的控制器可以包括路由控制器（用于设置云网络路由）、服务控制器（用于使用云负载均衡服务）和节点生命周期控制器（用于在节点的整个生命周期让节点与 Kubernetes 保持同步）。

Worker 节点（数据面板）

对于单节点 Kubernetes 集群，工作负载与控制面板在同一台计算机上运行。但是，更标准的配置是拥有一个或多个专门用于运行 Kubernetes 工作负载的独立计算机系统（节点）。

首次创建 Kubernetes 集群时，您可以使用某些集群创建工具，配置一定数量的节点添加到集群中（通过识别现有的计算机系统或让提供商创建新的计算机系统）。在向这些系统添加任何工作负载之前，向每个节点添加服务以实现以下功能：

- 管理每个节点 (**kubelet**) – 此 API 服务器会与每个节点上运行的 kubelet 服务进行通信，以确保节点已正确注册且调度程序请求的容器组正在运行。kubelet 可以读取容器组（pod）清单，并在本地系统上设置容器组（pod）所需的存储卷或其他功能。它还可以检查本地运行的容器的运行状况。
- 在节点上运行容器（容器运行时）– 每个节点上的容器运行时管理分配给节点的每个容器组所请求的容器。这意味着它可以从相应的注册表中拉取容器映像、运行容器、停止容器，并响应有关容器的查询。默认的容器运行时是 containerd。截至 Kubernetes 1.24，可以用作容器运行时的 Docker (dockershim) 的特殊集成已从 Kubernetes 中删除。虽然您仍然可以使用 Docker 在本地系统上测

试和运行容器，但要将 Docker 与 Kubernetes 结合使用，您现在必须在每个节点上安装 Docker 引擎才能将其与 Kubernetes 结合使用。

- 管理容器之间的联网 (**kube-proxy**) – 为了能够支持容器组（pod）之间的通信，Kubernetes 使用一种功能来设置可跟踪与这些容器组（pod）关联的 IP 地址和端口的容器组（pod）网络，此功能称为服务。kube-proxy 服务在每个节点上运行，以允许容器组（pod）之间进行通信。

扩展集群

您可以添加一些服务到 Kubernetes 中来支持集群，但不能在控制面板中运行这些服务。这些服务通常直接在 kube-system 命名空间或其自己的命名空间中的节点上运行（就像第三方服务提供商经常做的那样）。一个常见的例子是 CoreDNS 服务，它为集群提供 DNS 服务。有关如何查看集群上的 kube-system 中正在运行哪些集群服务的信息，请参阅发现内置服务。

您可以考虑向集群添加不同类型的附加组件。为了保持集群正常运行，您可以添加使您能够执行日志记录、审计等操作的可观测性功能（请参阅监控集群），以及指标。使用这些信息，您就可以对发生的问题进行排查，通常通过相同的可观测性接口排查。这些类型的服务的示例包括 Amazon GuardDuty、CloudWatch（请参阅 the section called “Amazon CloudWatch”）、适用于 OpenTelemetry 的 Amazon Distro、适用于 Kubernetes 的 Amazon VPC CNI 插件（请参阅the section called “Amazon VPC CNI”）和 Grafana Kubernetes 监控。对于存储（请参阅应用程序数据存储），相关 Amazon EKS 附加组件包括 Amazon Elastic Block Store CSI 驱动程序（请参阅 the section called “Amazon EBS”）、Amazon Elastic File System CSI 驱动程序（请参阅 the section called “Amazon EFS”）和多种第三方存储附加组件（例如 适用于 NetApp ONTAP 的 Amazon FSx CSI 驱动程序 the section called “Amazon FSx for NetApp ONTAP”）。

有关可用的 Amazon EKS 附加组件的更完整列表，请参阅 the section called “Amazon EKS 附加组件”。

工作负载

Kubernetes 将工作负载定义为“在 Kubernetes 上运行的应用程序”。该应用程序可以由一组在容器组（pod）中作为容器运行的微服务组成，也可以作为批处理作业或其他类型的应用程序运行。Kubernetes 的作业是确保您为要设置或部署的对象发出的请求得到执行。作为部署应用程序的人，您应该了解容器是如何构建的、容器组（pod）是如何定义的，以及您可以使用哪些方法来部署它们。

容器

您在 Kubernetes 中部署和管理的应用程序工作负载中最基本的元素是容器组（pod）。容器组（pod）代表一种保存应用程序组件以及定义描述容器组（pod）属性的规范的方法。与此形成鲜明对

比的是 RPM 或 Deb 软件包，后者将适用于 Linux 系统的软件打包在一起，但其本身并不作为一个实体运行。

由于容器组（pod）是最小的可部署单元，因此它通常只能容纳一个容器。但是，如果容器紧密耦合，则一个容器组（pod）中可以有多个容器。例如，Web 服务器容器可能被打包在带有 sidecar 类型的容器的容器组（pod）中，该容器类型可以提供日志记录、监控或其他与 Web 服务器容器紧密相关的服务。在这种情况下，位于同一容器组（pod）中可确保对于容器组（pod）的每个正在运行的实例，两个容器始终在同一个节点上运行。同样，容器组（pod）中的所有容器共享同一个环境，容器组（pod）中的容器就像在同一个隔离主机中一样运行。这样做的效果是，容器共享一个提供对容器组（pod）的访问权限的 IP 地址，并且容器可以像在自己的本地主机上运行一样相互通信。

容器组（pod）规范（PodSpec）定义容器组（pod）的所需状态。您可以通过使用工作负载资源管理容器组（pod）模板来部署单个或多个容器组（pod）。工作负载资源包括部署（用于管理多个容器组（pod）副本）、StatefulSets（用于部署需要具有唯一性的容器组（pod），例如数据库容器组（pod））和 DaemonSets（其中容器组（pod）需要在每个节点上持续运行）。稍后会详细介绍。

虽然容器组（pod）是您部署的最小单元，但容器是您构建和管理的最小单元。

构建容器

容器组（pod）实际上只是围绕一个或多个容器的结构，每个容器本身都包含文件系统、可执行文件、配置文件、库和其他用于实际运行应用程序的组件。由于 Docker Inc. 的公司率先推广了容器，因此也有人将容器称为 Docker 容器。但是，开放容器计划此后为该行业定义了容器运行时、映像和分发方法。再加上容器是由许多现有的 Linux 功能创建的这一事实，另一些人通常将容器称为 OCI 容器、Linux 容器或仅仅是容器。

当您构建容器时，通常会从 Dockerfile（字面意思）着手开始。在那个 Dockerfile 中，您可以识别：

- 基础映像 – 基础容器映像是一种容器，通常由操作系统（例如 Red Hat Enterprise Linux 或 Ubuntu）的最小文件系统版本或经过增强以提供运行特定类型应用程序（例如 nodejs 或 python 应用程序）所需软件的最小系统构建。
- 应用程序软件 – 您可以将应用程序软件添加到容器中，方法与将其添加到 Linux 系统中的方法大致相同。例如，在您的 Dockerfile 中，您可以运行 `npm` 和 `yarn` 安装 Java 应用程序，或运行 `yum` 和 `dnf` 安装 RPM 软件包。换句话说，在 Dockerfile 中使用 RUN 命令，您可以运行基础映像文件系统中提供的任何命令，以在生成的容器映像中安装软件或配置软件。
- 指令 – Dockerfile 参考描述了在配置 Dockerfile 时可以将其添加到 Dockerfile 中的指令。这些指令包括用于构建容器本身（本地系统的 `ADD` 或 `COPY` 文件）中的内容、识别容器运行时要执行的命令（`CMD` 或 `ENTRYPOINT`），以及将容器连接到其运行的系统（通过识别要运行的身份 `USER`、要挂载的本地 `VOLUME` 或要 `EXPOSE` 的端口）的指令。

虽然传统上使用 `docker` 命令和服务来构建容器（`docker build`，但其他可用于构建容器映像的工具包括 podman 和 nerdctl。要了解如何构建容器，请参阅 Building Better Container Images或 Overview of Docker Build。

存储容器

构建容器镜像后，可以将其存储在工作站上的容器分发注册表或公共容器注册表中。在工作站上运行私有容器注册表可使您在本地存储容器映像，使它们随时可供您使用。

要以更公开的方式存储容器映像，可以将其推送到公共容器注册表。公共容器注册表为存储和分发容器映像提供了一个中心位置。公共容器注册表的示例包括 Amazon Elastic Container Registry、Red Hat Quay 注册表和 Docker Hub 注册表。

在 Amazon Elastic Kubernetes Service（Amazon EKS）上运行容器化工作负载时，建议您拉取存储在 Amazon Elastic Container Registry 中的 Docker 官方映像的副本。自 2021 年以来，Amazon ECR 一直在存储这些映像。您可以在 Amazon ECR 公开映像浏览馆中搜索常用容器映像，在 Amazon ECR Docker 映像浏览馆中搜索 Docker Hub 映像。

正在运行的容器

由于容器是以标准格式构建的，因此容器可以在任何可运行容器运行时（例如 Docker）且其内容与本地计算机的架构（例如 `x86_64` 或 `arm`）相匹配的计算机上运行。要测试容器或者只是在本地桌面上运行它，您可以使用 `docker run` 或 `podman run` 命令在本地主机上启动容器。但是对于 Kubernetes，每个 Worker 节点都部署了容器运行时，它可以依据 Kubernetes 请求节点运行容器。

将容器分配到某个节点上运行后，该节点将查看节点上是否已存在所请求的容器映像版本。如果没有，则 Kubernetes 告诉容器运行时从相应的容器注册表中拉取该容器，然后在本地运行该容器。请记住，容器映像是指在笔记本电脑、容器注册表和 Kubernetes 节点之间移动的软件包。容器是指该映像的运行实例。

容器组（pod）

容器准备就绪后，使用容器组（pod）的操作包括配置、部署和使容器组（pod）可访问。

配置容器组（pod）

当您定义容器组（pod）时，会为其分配一组属性。这些属性必须至少包含容器组（pod）名称和要运行的容器映像。但是，您还需要使用容器组（pod）定义配置许多其他内容（有关可以进入容器组（pod）的内容的详细信息，请参阅 PodSpec 页面）。这些指令包括：

- 存储 – 停止并删除正在运行的容器后，该容器中的数据存储将消失，除非您设置了更持久的存储。Kubernetes 支持许多不同的存储类型，并在卷的保护下对其进行抽象。存储类型包括

CephFS、NFS、iSCSI 等。您甚至可以在本地计算机上使用本地块设备。使用集群中提供的其中一种存储类型，您可以将存储卷挂载到容器文件系统中的选定挂载点。永久卷是指删除容器组（pod）后继续存在的卷，而临时卷在容器组（pod）被删除时会被删除。如果集群管理员为集群创建了不同的存储类，则可以选择所用存储的属性，例如：卷在使用后是删除还是回收，卷是否会在需要更多空间时扩展，甚至卷是否满足某些性能要求。

- 密钥 – 通过在容器组规范中向容器提供密钥，您可以提供这些容器访问文件系统、数据库或其它受保护资产所需的权限。密钥、密码和令牌是可以作为密钥存储的项目之一。通过使用密钥，您不必将这些信息存储在容器映像中，而只需要将密钥提供给正在运行的容器即可。与密钥相似的是 ConfigMaps。`ConfigMap` 往往会保存不太重要的信息，例如用于配置服务的键值对。
- 容器资源 – 用于进一步配置容器的对象可以采用资源配置的形式。对于每个容器，您可以请求其可以使用的内存和 CPU 量，以及对容器可以使用的资源总量设置限制。有关示例，请参阅容器组（pod）和容器的资源管理。
- 中断 – 容器组可以被非自愿中断（节点关闭），也可以自愿中断（需要升级）。通过配置容器组（pod）中断预算，您可以在一定程度上控制发生中断时应用程序的可用性。有关示例，请参阅为您的应用程序指定中断预算。
- 命名空间 – Kubernetes 提供不同的方法来相互隔离 Kubernetes 组件和工作负载。在同一个命名空间中运行特定应用程序的所有容器组（pod）是共同保护和管理这些容器组（pod）的常用方法。您可以创建自己的命名空间以供使用，也可以选择不指定命名空间（这会导致 Kubernetes 使用 `default` 命名空间）。Kubernetes 控制面板组件通常在 kube-system 命名空间中运行。

刚才所说的配置通常汇集在要应用于 Kubernetes 集群的 YAML 文件中。对于个人 Kubernetes 集群，您可以将这些 YAML 文件存储在本地系统上。但是，对于更关键的集群和工作负载，GitOps 是自动存储和更新工作负载和 Kubernetes 基础设施资源的常用方式。

用于将容器组（pod）信息收集在一起并进行部署的对象由以下部署方法之一定义。

部署 Pod

您为部署容器组（pod）而选择的方法取决于您计划使用这些容器组（pod）运行的应用程序的类型。以下是您的一些选择：

- 无状态应用程序 – 无状态应用程序不会保存客户端的会话数据，因此另一个会话无需重新引用前一个会话发生的情况。这样一来，能更容易地在容器组（pod）运行不正常时将其替换为新的容器组（pod）或者在不保存状态的情况下移动它们。如果您运行的是无状态应用程序（例如 Web 服务器），则可以使用部署来部署容器组（pod）和 ReplicaSets。ReplicaSet 定义您想同时运行容器组（pod）的多少个实例。尽管您可以直接运行 ReplicaSet，但通常是在部署中直接运行副本，以定义一个容器组（pod）一次应该运行多少个副本。

- 有状态应用程序 – 有状态应用程序是看重容器组（pod）的身份和容器组（pod）的启动顺序的应用程序。这些应用程序需要稳定的永久存储，并且需要以一致的方式进行部署和横向缩减。要在 Kubernetes 中部署有状态应用程序，可以使用 StatefulSets。通常以 StatefulSet 形式运行的应用程序的一个示例是数据库。在 StatefulSet 中，您可以定义副本、容器组（pod）及其容器、要挂载的存储卷以及容器中存储数据的位置。有关将数据库部署为 ReplicaSet 的示例，请参阅运行已复制的有状态应用程序。
- 每节点应用程序 – 有时您需要在 Kubernetes 集群中的每个节点上运行应用程序。例如，您的数据中心可能要求每台计算机都运行监控应用程序或特定的远程访问服务。对于 Kubernetes，您可以使用 DaemonSet 确保所选应用程序在集群中的每个节点上运行。
- 应用程序运行至完成 – 您希望运行一些应用程序来完成特定的任务。这可能包括运行月度状态报告或清理旧数据的应用程序。Job 对象可用于设置应用程序以启动和运行，然后在任务完成后退出。CronJob 对象允许您使用由 Linux crontab 格式定义的结构将应用程序设置为在特定的小时、分钟、一月中的某一天、月份或一周中的某一天运行。

使应用程序可以从网络访问

由于应用程序通常部署为一组会移动到不同地方的微服务，因此 Kubernetes 需要一种方法让这些微服务能够找到彼此。此外，为了让其他人访问 Kubernetes 集群之外的应用程序，Kubernetes 需要一种在外部地址和端口上公开该应用程序的方法。这些与联网相关的功能分别通过 Service 和 Ingress 对象完成：

- 服务：容器组（pod）可以移动到不同的节点和地址，因此需要与第一个容器组（pod）通信的另一个容器组（pod）可能会很难找到它的位置。要解决此问题，Kubernetes 可以让您将应用程序表示为服务。使用服务，您可以识别具有特定名称的一个容器组（pod）或一组容器组（pod），然后指明哪个端口从容器组（pod）中公开该应用程序的服务，以及其他应用程序可以使用哪些端口来联系该服务。集群中的另一个容器组（pod）只需按名称请求服务，然后 Kubernetes 将该请求定向到运行该服务的容器组（pod）实例的正确端口。
- 入口 – 入口让 Kubernetes 服务代表的应用程序可供集群外的客户端使用。Ingress 的基本功能包括负载均衡器（由 Ingress 管理）、Ingress 控制器以及将请求从控制器路由到服务的规则。Kubernetes 中有多个 Ingress 控制器可供选择。

后续步骤

了解基本 Kubernetes 概念及其与 Amazon EKS 的关系将有助于您浏览 Amazon EKS 文档和 Kubernetes 文档，以找到您管理 Amazon EKS 集群和向这些集群部署工作负载所需的信息。要开始使用 Amazon EKS，请从以下选项中进行选择：

- the section called “创建集群（eksctl）”
- the section called “创建集群”
- the section called “示例部署（Linux）”
- 集群管理

跨云端和本地环境部署 Amazon EKS 集群

了解 Amazon EKS 部署选项

Amazon Elastic Kubernetes Service（Amazon EKS）是一项完全托管式的 Kubernetes 服务，可让您在云端和本地环境中无缝运行 Kubernetes。

对于云端，Amazon EKS 会自动管理 Kubernetes 控制面板和节点的 Kubernetes 集群基础设施。这对于调度容器、管理应用程序可用性、动态扩展资源、优化计算、存储集群数据以及执行其他关键功能至关重要。使用 Amazon EKS 时，您不仅可以利用 Amazon 基础设施的强大性能、可扩展性、可靠性和可用性，同时还可与 Amazon 联网、安全、存储和可观测性服务原生集成。

为了简化在本地环境中运行 Kubernetes 的过程，您可以使用相同的 Amazon EKS 集群、功能和工具the section called “Nodes”，或在自己的基础设施上使用 Amazon EKS 混合节点功能，也可以使用 Amazon EKS Anywhere 建立受物理隔离的独立环境。

云中的 Amazon EKS

您可以将 Amazon EKS 与 Amazon 区域、Amazon Local Zones 和 Amazon Wavelength Zones 中的计算配合使用。在云端使用 Amazon EKS 时，Kubernetes 控制面板的安全性、可扩展性和可用性完全由 Amazon 在 Amazon 区域中管理。在 Amazon 区域中运行具有计算功能的应用程序时，您可以全面利用 Amazon 和 Amazon EKS 的各种功能，包括 Amazon EKS 自动模式。使用自动模式时，您可以一键式完全自动管理 Amazon 上有关计算、存储和联网方面的 Kubernetes 集群基础设施。在 Amazon Local Zones and Amazon Wavelength Zones 中运行具有计算功能的应用程序时，您可以使用 Amazon EKS 自主管理型节点连接 Amazon EC2 实例以进行集群计算，还可以使用 Amazon Local Zones and Amazon Wavelength Zones 中可用的其他 Amazon 服务。有关更多信息，请参阅 Amazon Local Zones 功能和 Amazon Wavelength Zones 功能。

	Amazon 区域中的 Amazon EKS	Local/Wavelength Zones 中的 Amazon EKS
Kuberenetes 控制面板管理	Amazon 托管式	Amazon 托管式

	Amazon 区域中的 Amazon EKS	Local/Wavelength Zones 中的 Amazon EKS
Kubernetes 控制面板位置	Amazon 区域	Amazon 区域
Kubernetes 数据面板	• Amazon EKS 自动模式 • Amazon EKS 托管式节点组 • Amazon EC2 自主管理型节点 • Amazon Fargate	• Amazon EKS 托管式节点组（仅限 Local Zones） • Amazon EC2 自主管理型节点
Kubernetes 数据层面位置	Amazon 区域	Amazon Local 或 Wavelength Zones

数据中心或边缘环境中的 Amazon EKS

如果需要在自己的数据中心或边缘环境中运行应用程序，则可以使用 Amazon EKS on Amazon Outposts 或 Amazon EKS 混合节点功能。您可以使用自主管理型节点，将 Amazon Outposts 上的 Amazon EC2 实例作为集群计算，也可以使用 Amazon EKS 混合节点功能，将自己的本地或边缘基础设施作为集群计算。AmazonOutposts 是您在自己的数据中心或托管设施中运行的 Amazon 托管式基础设施，Amazon EKS 混合节点功能则在您在本地或边缘环境中管理的物理计算机或虚拟机上运行。Amazon Outposts 上的 Amazon EKS 和 Amazon EKS 混合节点功能需要在您的本地环境与某个 Amazon 区域之间建立可靠的连接，并且您可以使用与在云端运行应用程序相同的 Amazon EKS 集群、功能和工具。在 Amazon Outposts 上运行时，您还可以在 Amazon Outposts 上部署整个 Kubernetes 集群，并将 Amazon EKS 本地集群部署到 Amazon Outposts 上。

	Amazon EKS 混合节点功能	Amazon EKS on Amazon Outposts
Kuberenetes 控制面板管理	Amazon 托管式	Amazon 托管式
Kubernetes 控制面板位置	Amazon 区域	Amazon 区域或 Amazon Outposts
Kubernetes 数据面板	客户自主管理型物理计算机或虚拟机	Amazon EC2 自主管理型节点

	Amazon EKS 混合节点功能	Amazon EKS on Amazon Outposts
Kubernetes 数据层面位置	客户数据中心或边缘环境	客户数据中心或边缘环境

适用于气隙环境的 Amazon EKS Anywhere

Amazon EKS Anywhere 通过自动化无差别的繁琐工作，例如本地和边缘环境中的基础设施设置和 Kubernetes 集群生命周期操作，从而简化了 Kubernetes 集群管理。与 Amazon EKS 不同，Amazon EKS Anywhere 是一种客户自主管理型产品，由客户负责集群生命周期操作和 Amazon EKS Anywhere 集群的维护。Amazon EKS Anywhere 以 Kubernetes 子项目集群 API（CAPI）为基础，支持多种基础设施，包括 VMware vSphere、裸机、Nutanix、Apache CloudStack 和 Amazon Snow。Amazon EKS Anywhere 可以在气隙环境中运行，并提供与区域性 Amazon 服务的可选集成，从而满足可观测性和身份管理的需要。要获取有关 Amazon EKS Anywhere 的支持并访问由 Amazon 销售的 Kubernetes 附加组件，您可以购买 Amazon EKS Anywhere 企业订阅。

	Amazon EKS Anywhere
Kuberenetes 控制面板管理	客户托管
Kubernetes 控制面板位置	客户数据中心或边缘环境
Kubernetes 数据面板	客户自主管理型物理计算机或虚拟机
Kubernetes 数据层面位置	客户数据中心或边缘环境

Amazon EKS 工具

您可以使用 Amazon EKS 连接器注册任何符合要求的 Kubernetes 集群并将其连接至 Amazon，然后在 Amazon EKS 控制台中查看。连接集群后，您可以在 Amazon EKS 控制台中查看集群的状态、配置和工作负载。您可以使用此功能在 Amazon EKS 控制台中查看已连接的集群，但是 Amazon EKS 连接器不支持通过 Amazon EKS 控制台对已连接的集群进行管理或变异操作。

Amazon EKS Distro 是支持所有 Amazon EKS 产品的底层 Kubernetes 组件的 Amazon 发行版。其中包括正常 Kubernetes 集群所需的核心组件，例如 Kubernetes 控制面板组件（etcd、kube-apiserver、kube-scheduler、kube-controller-manager）和联网组件（CoreDNS、kube-proxy、CNI

插件）。Amazon EKS Distro 可用于通过您选择的工具自主管理 Kubernetes 集群。Amazon EKS Distro 部署部署 Amazon Support 计划的范围。

进行设置以使用 Amazon EKS

要准备通过命令行来管理 Amazon EKS 集群，您需要安装多种工具。使用以下工具设置凭证、创建和修改集群，以及在集群运行后使用集群：

- 设置 Amazon CLI – 获取 Amazon CLI 以设置和管理使用 Amazon EKS 集群所需的服务。您将尤其需要通过 Amazon CLI 来配置凭证，而其它 Amazon 服务也会需要它。
- 设置 kubectl 和 eksctl – `eksctl` CLI 与 Amazon 交互来创建、修改和删除 Amazon EKS 集群。集群启动后，可使用开源 `kubectl` 命令管理 Amazon EKS 集群中的 Kubernetes 对象。
- 设置开发环境（可选）– 考虑添加以下工具：
  - 本地部署工具 – 如果您不熟悉 Kubernetes，可以考虑安装像 minikube 或 kind 这样的本地部署工具。这些工具让您能够在本地计算机上设置 Amazon EKS 集群，从而测试应用程序。
  - 软件包管理器：Helm 是 Kubernetes 的一款常用软件包管理器，该工具简化了复杂软件包的安装和管理。使用 Helm，可以更轻松地在 Amazon EKS 集群上安装和管理 Amazon 负载均衡器控制器等软件包。

后续步骤

- 设置 Amazon CLI
- 设置 kubectl 和 eksctl
- 快速入门：部署 Web 应用程序并存储数据

设置 Amazon CLI

Amazon CLI 是与 Amazon 服务（包括 Amazon EKS）结合使用的命令行工具。该工具还可用于对从本地计算机访问 Amazon EKS 集群和其他 Amazon 资源的 IAM 用户或角色进行身份验证。要通过命令行在 Amazon 中预置资源，您需要获取 Amazon 访问密钥 ID 和私有密钥，以便在命令行中使用。然后，您需要在 Amazon CLI 中配置这些凭证。如果尚未安装 Amazon CLI，请参阅《Amazon 命令行界面用户指南》中的安装或更新 Amazon CLI 的最新版本。

创建访问密钥

1. 登录到 Amazon Web Services 管理控制台。
2. 对于单用户或多用户账户：

- 但用户账户 – 在右上角，选择您的 Amazon 用户名以打开导航菜单。例如，选择 **webadmin**。
- 多用户账户 – 从服务列表中选择 IAM。在 IAM 控制面板中，选择用户，然后选择用户的名称。

3. 选择安全凭证。
4. 在访问密钥下，选择创建访问密钥。
5. 选择命令行界面（CLI），然后选择下一步。
6. 选择创建访问密钥。
7. 选择下载 .csv 文件。

配置 Amazon CLI

安装 Amazon CLI 后，请按照以下步骤进行配置。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的配置 Amazon CLI。

1. 在终端窗口中，输入以下命令：

```
aws configure
```

   您还可以选择配置一个命名配置文件，例如 `--profile cluster-admin`。如果您在 Amazon CLI 中配置了命名配置文件，则必须始终在后续命令中传递此标志。

2. 输入 Amazon 凭证。例如：

```
Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: region-code
Default output format [None]: json
```

获取安全令牌

如果需要，运行以下命令以获取 Amazon CLI 的新安全令牌。有关更多信息，请参阅《Amazon CLI 命令参考》中的 get-session-token。

默认情况下，令牌的有效期为 15 分钟。要更改默认会话超时，请传递 `--duration-seconds` 标志。例如：

```
aws sts get-session-token --duration-seconds 3600
```

此命令将返回 Amazon CLI 会话的临时安全凭证。您应看到以下响应输出：

```
{
    "Credentials": {
        "AccessKeyId": "ASIA5FTRU3LOEXAMPLE",
        "SecretAccessKey": "JnKgvwfqUD9mNsPoi9IbxAYEXAMPLE",
        "SessionToken": "VERYLONGSESSIONTOKENSTRING",
        "Expiration": "2023-02-17T03:14:24+00:00"
    }
}
```

验证用户身份

如果需要，可运行以下命令来验证用于终端会话的 IAM 用户身份（例如 *ClusterAdmin*）的 Amazon 凭证。

```
aws sts get-caller-identity
```

此命令将返回为 Amazon CLI 配置的 IAM 实体的 Amazon 资源名称（ARN）。您应看到以下示例响应输出：

```
{
    "UserId": "AKIAIOSFODNN7EXAMPLE",
    "Account": "01234567890",
    "Arn": "arn:aws-cn:iam::01234567890:user/ClusterAdmin"
}
```

后续步骤

- 设置 kubectl 和 eksctl
- 快速入门：部署 Web 应用程序并存储数据

设置 kubectl 和 eksctl

安装 Amazon CLI 后，还需要安装另外两个工具来创建和管理 Kubernetes 集群：

- kubectl：kubectl 命令行工具是用于管理 Kubernetes 集群内资源的主要工具。本页介绍如何下载和设置与 Kubernetes 集群版本相匹配的 kubectl 二进制文件。请参阅安装或更新 kubectl。

- eksctl：eksctl 命令行工具用于在 Amazon 云端或本地（使用 EKS Anywhere）创建 EKS 集群，以及修改和删除这些集群。请参阅安装 eksctl。

安装或更新 kubectl

本主题将帮助您下载并安装或更新设备上的 kubectl 二进制文件。该二进制文件与上游社区版本相同。该二进制文件不为 Amazon EKS 或 Amazon 所特有。使用以下步骤获取所需的 kubectl 特定版本，尽管许多生成器只是运行 brew install kubectl 来安装它。

> **Note**
>
> 您必须使用与您的 Amazon EKS 集群控制面板不同的一个次要版本内的 kubectl 版本。例如，1.32 kubectl 客户端使用 Kubernetes 1.31、1.32 和 1.33 集群。

第 1 步：查看 kubectl 是否已安装

确定您是否已将 kubectl 安装在设备上。

```
kubectl version --client
```

如果您已将 kubectl 安装在设备路径中，示例输出包括类似于如下的信息。如果要使用更高版本更新当前已安装的版本，请完成下一步，确保将新版本安装在当前版本所在的位置。

```
Client Version: v1.31.X-eks-1234567
```

如果您没有收到任何输出，则可能尚未安装 kubectl，或者未安装在设备路径中的位置。

第 2 步：安装或更新 kubectl

在以下操作系统之一上安装或更新 kubectl：

- the section called “macOS”
- the section called “Linux（amd64）”
- the section called “Linux（arm64）”
- the section called “Windows”

> **Note**
>
> 如果从本节中使用的 Amazon 区域下载到您所在 Amazon 区域的速度很慢，可考虑设置 CloudFront 以预加载内容。有关更多信息，请参阅开始使用基本 CloudFront 分配。

macOS

请按照以下步骤，在 macOS 上安装 `kubectl`。这些步骤包括：

- 选择并下载所需 Kubernetes 版本的二进制文件。
- （可选）检查二进制文件的校验和。
- 将执行权限添加到二进制文件的权限中。
- 将二进制文件复制到路径中的文件夹。
- （可选）将二进制文件的目录添加到路径中。

过程：

1. 从 Amazon S3 为集群的 Kubernetes 版本下载二进制文件。
   - Kubernetes `1.34`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.2/2025-11-13/bin/darwin/amd64/kubectl
     ```

   - Kubernetes `1.33`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.5/2025-11-13/bin/darwin/amd64/kubectl
     ```

   - Kubernetes `1.32`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.9/2025-11-13/bin/darwin/amd64/kubectl
     ```

   - Kubernetes `1.31`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.13/2025-11-13/bin/darwin/amd64/kubectl
     ```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2025-11-13/bin/darwin/amd64/kubectl
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-11-13/bin/darwin/amd64/kubectl
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-11-13/bin/darwin/amd64/kubectl
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2025-01-10/bin/darwin/amd64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/darwin/amd64/kubectl
```

2. （可选）使用二进制文件的 SHA-256 校验和验证下载的二进制文件。

   a. 下载集群 Kubernetes 版本的 SHA-256 校验和。

   - Kubernetes 1.34

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.2/2025-11-13/bin/darwin/amd64/kubectl.sha256
   ```

   - Kubernetes 1.33

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.5/2025-11-13/bin/darwin/amd64/kubectl.sha256
   ```

   - Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.9/2025-11-13/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.13/2025-11-13/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2025-11-13/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-11-13/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-11-13/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2025-01-10/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
darwin/amd64/kubectl.sha256
```

b. 检查下载的二进制文件的 SHA-256 校验和。

```
openssl sha1 -sha256 kubectl
```

c. 确保输出中生成的校验和与下载的 kubectl.sha256 文件中的校验和相匹配。

3. 将执行权限应用于二进制文件。

```
chmod +x ./kubectl
```

4. 将二进制文件复制到您的 PATH 中的文件夹。如果您已经安装了某个版本的 kubectl，建议您创建一个 $HOME/bin/kubectl 并确保 $HOME/bin 先出现在您的 $PATH 中。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (可选) 将 $HOME/bin 路径添加到 Shell 初始化文件，以便在打开 Shell 时配置此路径。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bash_profile
```

Linux（amd64）

请按照以下步骤，在 Linux（amd64）上安装 kubectl。这些步骤包括：

- 选择并下载所需 Kubernetes 版本的二进制文件。
- （可选）检查二进制文件的校验和。
- 将执行权限添加到二进制文件的权限中。
- 将二进制文件复制到路径中的文件夹。
- （可选）将二进制文件的目录添加到路径中。

过程：

1. 从 Amazon S3 为集群的 Kubernetes 版本下载 kubectl 二进制文件。
   - Kubernetes 1.34

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.2/2025-11-13/bin/linux/amd64/kubectl
     ```

   - Kubernetes 1.33

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.5/2025-11-13/bin/linux/amd64/kubectl
     ```

   - Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.9/2025-11-13/bin/linux/amd64/kubectl
```

- Kubernetes `1.31`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.13/2025-11-13/bin/linux/amd64/kubectl
```

- Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2025-11-13/bin/linux/amd64/kubectl
```

- Kubernetes `1.29`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-11-13/bin/linux/amd64/kubectl
```

- Kubernetes `1.28`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-11-13/bin/linux/amd64/kubectl
```

- Kubernetes `1.27`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/amd64/kubectl
```

- Kubernetes `1.26`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/linux/amd64/kubectl
```

2. （可选）使用二进制文件的 SHA-256 校验和验证下载的二进制文件。

   a. 使用适用于设备硬件平台的命令，从 Amazon S3 为集群的 Kubernetes 版本下载 SHA-256 校验和。

      - Kubernetes `1.34`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.2/2025-11-13/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.33`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.5/2025-11-13/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.32`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.9/2025-11-13/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.31`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.13/2025-11-13/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2025-11-13/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.29`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-11-13/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.28`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-11-13/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.27`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.26`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

b. 使用下面的命令之一检查下载的二进制文件的 SHA-256 校验和。

```
sha256sum -c kubectl.sha256
```

或者

```
openssl sha1 -sha256 kubectl
```

c. 对于第一步，您应该看到 `kubectl: OK`，对于第二步，您可以检查输出中生成的校验和与下载的 `kubectl.sha256` 文件中的校验和是否匹配。

3. 将执行权限应用于二进制文件。

```
chmod +x ./kubectl
```

4. 将二进制文件复制到您的 PATH 中的文件夹。如果您已经安装了某个版本的 `kubectl`，建议您创建一个 `$HOME/bin/kubectl` 并确保 `$HOME/bin` 先出现在您的 `$PATH` 中。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (可选) 将 `$HOME/bin` 路径添加到 Shell 初始化文件，以便在打开 Shell 时配置此路径。

> **Note**
>
> 这一步假设您使用 Bash Shell；如果使用其他 Shell，请将命令更改为使用您的特定 Shell 的初始化文件。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Linux（arm64）

请按照以下步骤，在 Linux（arm64）上安装 `kubectl`。这些步骤包括：

- 选择并下载所需 Kubernetes 版本的二进制文件。

- （可选）检查二进制文件的校验和。
- 将执行权限添加到二进制文件的权限中。
- 将二进制文件复制到路径中的文件夹。
- （可选）将二进制文件的目录添加到路径中。

过程：

1. 从 Amazon S3 为集群的 Kubernetes 版本下载 `kubectl` 二进制文件。
   - Kubernetes `1.34`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.2/2025-11-13/bin/linux/arm64/kubectl
     ```

   - Kubernetes `1.33`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.5/2025-11-13/bin/linux/arm64/kubectl
     ```

   - Kubernetes `1.32`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.9/2025-11-13/bin/linux/arm64/kubectl
     ```

   - Kubernetes `1.31`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.13/2025-11-13/bin/linux/arm64/kubectl
     ```

   - Kubernetes `1.30`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2025-11-13/bin/linux/arm64/kubectl
     ```

   - Kubernetes `1.29`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-11-13/bin/linux/arm64/kubectl
     ```

   - Kubernetes `1.28`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-11-13/bin/linux/arm64/kubectl
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/arm64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/linux/arm64/kubectl
```

2. （可选）使用二进制文件的 SHA-256 校验和验证下载的二进制文件。

   a. 使用适用于设备硬件平台的命令，从 Amazon S3 为集群的 Kubernetes 版本下载 SHA-256 校验和。

   - Kubernetes 1.34

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.2/2025-11-13/bin/linux/arm64/kubectl.sha256
   ```

   - Kubernetes 1.33

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.5/2025-11-13/bin/linux/arm64/kubectl.sha256
   ```

   - Kubernetes 1.32

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.9/2025-11-13/bin/linux/arm64/kubectl.sha256
   ```

   - Kubernetes 1.31

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.13/2025-11-13/bin/linux/arm64/kubectl.sha256
   ```

   - Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2025-11-13/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-11-13/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-11-13/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

b. 使用下面的命令之一检查下载的二进制文件的 SHA-256 校验和。

```
sha256sum -c kubectl.sha256
```

或者

```
openssl sha1 -sha256 kubectl
```

c. 对于第一步，您应该看到 kubectl: OK，对于第二步，您可以检查输出中生成的校验和与下载的 kubectl.sha256 文件中的校验和是否匹配。

3. 将执行权限应用于二进制文件。

```
chmod +x ./kubectl
```

4. 将二进制文件复制到您的 PATH 中的文件夹。如果您已经安装了某个版本的 kubectl，建议您创建一个 $HOME/bin/kubectl 并确保 $HOME/bin 先出现在您的 $PATH 中。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (可选) 将 $HOME/bin 路径添加到 Shell 初始化文件，以便在打开 Shell 时配置此路径。

> **Note**
>
> 这一步假设您使用 Bash Shell；如果使用其他 Shell，请将命令更改为使用您的特定 Shell 的初始化文件。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Windows

请按照以下步骤，在 Windows 上安装 kubectl。这些步骤包括：

- 选择并下载所需 Kubernetes 版本的二进制文件。
- （可选）检查二进制文件的校验和。
- 将二进制文件复制到路径中的文件夹。
- （可选）将二进制文件的目录添加到路径中。

过程：

1. 打开 PowerShell 终端。
2. 从 Amazon S3 为集群的 Kubernetes 版本下载 kubectl 二进制文件。
   - Kubernetes 1.34

     ```
     curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.2/2025-11-13/bin/windows/amd64/kubectl.exe
     ```

   - Kubernetes 1.33

     ```
     curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.5/2025-11-13/bin/windows/amd64/kubectl.exe
     ```

   - Kubernetes 1.32

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.9/2025-11-13/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.31

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.13/2025-11-13/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.30

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2025-11-13/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.29

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-11-13/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.28

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-11-13/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.27

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.26

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/windows/amd64/kubectl.exe
```

3. （可选）使用二进制文件的 SHA-256 校验和验证下载的二进制文件。

   a. 为 Windows 集群的 Kubernetes 版本下载 SHA-256 校验和。

   - Kubernetes 1.34

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.2/2025-11-13/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.33

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.5/2025-11-13/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.32

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.9/2025-11-13/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.31

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.13/2025-11-13/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.30

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2025-11-13/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.29

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-11-13/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.28

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-11-13/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.27

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.26

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/windows/amd64/kubectl.exe.sha256
```

b. 检查下载的二进制文件的 SHA-256 校验和。

```
Get-FileHash kubectl.exe
```

c. 确保输出中生成的校验和与下载的 kubectl.sha256 文件中的校验和相匹配。PowerShell 的输出应为等效的大写字符串。

4. 将二进制文件复制到您的 PATH 中的文件夹。如果 PATH 中有现有目录可用于命令行实用程序，请将二进制文件复制到该目录。否则，请完成以下步骤。

   a. 为命令行二进制文件创建一个新目录，例如 C:\bin。

   b. 将 kubectl.exe 二进制文件复制到新目录。

   c. 编辑用户或系统 PATH 环境变量，将新目录添加到 PATH 中。

   d. 关闭 PowerShell 终端并打开一个新的来选取新的 PATH 变量。

5. 安装 kubectl 后，您可以验证其版本。

```
kubectl version --client
```

6. 首次安装 kubectl 时，它尚未配置为与任何服务器通信。我们将根据需要在其他过程中介绍此配置。如果您需要更新配置以与特定集群通信，可以运行以下命令。将 *region-code* 替换为您的集群所在的 Amazon 区域。将 *my-cluster* 替换为您的集群的名称。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

7. 考虑配置自动完成，这样您就可以在键入前几个字母后使用 Tab 键完成 kubectl 子命令。有关详细信息，请参阅 Kubernetes 文档中的 Kubectl autocomplete。

安装 eksctl

eksctl CLI 可用于处理 EKS 集群。它可自动化处理各种单独的任务。有关安装 eksctl 的说明，请参阅 eksctl 文档中的 Installation。对于 Linux，请使用 UNIX 指令。

使用 eksctl 时，所使用的 IAM 安全主体必须具有使用 Amazon EKS IAM 角色、服务相关角色、Amazon CloudFormation、VPC 和相关资源的权限。有关更多信息，请参阅《IAM 用户指南》中的操作和编辑服务相关角色。您必须以同一用户身份完成本指南中的所有步骤。要查看当前用户，请运行以下命令：

```
aws sts get-caller-identity
```

后续步骤

- 快速入门：部署 Web 应用程序并存储数据

快速入门：部署 Web 应用程序并存储数据

本快速入门教程将指导您完成使用 eksctl 在 Amazon EKS 自动模式集群上部署 2048 游戏示例应用程序，并将其数据持久保存的步骤。

Amazon EKS 自动模式可自动执行块存储、联网、负载均衡和计算自动扩缩等例行任务，从而简化集群管理。在设置过程中，该模式负责创建包含 EC2 托管式实例、应用程序负载均衡器和 EBS 卷的节点。

总之，您将部署一个示例工作负载，其中包含与 Amazon 服务无缝集成所需的自定义注释。

在本教程中：

使用之后的 `eksctl` 集群模板，您将构建一个启用了 EKS 自动模式的集群，以实现自动节点预置。

- VPC 配置：使用下面的 eksctl 集群模板时，eksctl 会自动为该集群创建一个 IPv4 虚拟私有云（VPC）。默认情况下，除创建公有和私有端点外，eksctl 还会配置一个满足所有联网要求的 VPC。
- 实例管理：EKS 自动模式会根据 Kubernetes 应用程序的需求，动态添加或移除 EKS 集群中的节点。
- 数据持久性：使用 EKS 自动模式的块存储功能来确保应用程序数据的持久性，即使在涉及容器组（pod）重启或故障等场景时也不例外。
- 外部应用程序访问：使用 EKS 自动模式的负载均衡功能来动态预调配应用程序负载均衡器（ALB）。

先决条件

开始之前，请确保您已执行以下任务：

- 为 Amazon EKS 设置您的环境
- 安装 eksctl 的最新版本

配置集群

在本节中，您将使用 EKS 自动模式创建一个集群，以进行动态节点预置。

创建一个 `cluster-config.yaml` 文件，并将以下内容粘贴到该文件中。将 `region-code` 替换为有效的区域（例如 `us-east-1`）。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: web-quickstart
  region: <region-code>

autoModeConfig:
  enabled: true
```

现在，我们可以随时创建集群。

使用 `cluster-config.yaml` 创建 EKS 集群：

```
eksctl create cluster -f cluster-config.yaml
```

> **Important**
>
> 如果您不使用 eksctl 创建集群，则需要手动标记 VPC 子网。

创建 IngressClass

为 EKS 自动模式创建一个 Kubernetes 。`IngressClass`IngressClass 定义 EKS 自动模式将如何处理 Ingress 资源。此步骤会配置 EKS 自动模式的负载均衡功能。为应用程序创建 Ingress 资源时，EKS 自动模式使用此 IngressClass 自动预置和管理负载均衡器，从而将您的 Kubernetes 应用程序与 Amazon 负载均衡服务集成。

将以下 yaml 文件另存为 `ingressclass.yaml`：

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
```

```
  controller: eks.amazonaws.com/alb
```

将 IngressClass 应用到集群：

```
kubectl apply -f ingressclass.yaml
```

部署 2048 游戏示例应用程序

在这一部分中，我们将演示在集群中部署流行的“2048 游戏”这款示例应用程序的步骤。提供的清单包括自定义的应用程序负载均衡器（ALB）注释。这些注释与 EKS 集成并指示后者将传入的 HTTP 流量作为“面向互联网”的流量处理，并使用目标类型“ip”将其路由到 game-2048 命名空间中的相应服务。

> **Note**
>
> 示例中的 docker-2048 映像为 x86_64 容器映像，不会在其他架构上运行。

1. 创建一个名为 game-2048 的 Kubernetes 命名空间，带 --save-config 标志。

   ```
   kubectl create namespace game-2048 --save-config
   ```

   您应看到以下响应输出：

   ```
   namespace/game-2048 created
   ```

2. 部署 2048 游戏示例应用程序。

   ```
   kubectl apply -n game-2048 -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.8.0/docs/examples/2048/2048_full.yaml
   ```

   此清单为 game-2048 命名空间设置了 Kubernetes 部署、服务和入口，从而创建了在集群中部署和公开 game-2048 应用程序所需的资源。这包括创建一个名为 service-2048 的服务，用于在端口 80 上公开该部署；以及一个名为 ingress-2048 的入口资源，用于定义传入的 HTTP 流量的路由规则，以及面向互联网的应用程序负载均衡器（ALB）的注释。您应看到以下响应输出：

   ```
   namespace/game-2048 configured
   deployment.apps/deployment-2048 created
   service/service-2048 created
   ```

```
ingress.networking.k8s.io/ingress-2048 created
```

3. 运行以下命令以获取 game-2048 命名空间的入口资源。

```
kubectl get ingress -n game-2048
```

您应看到以下响应输出：

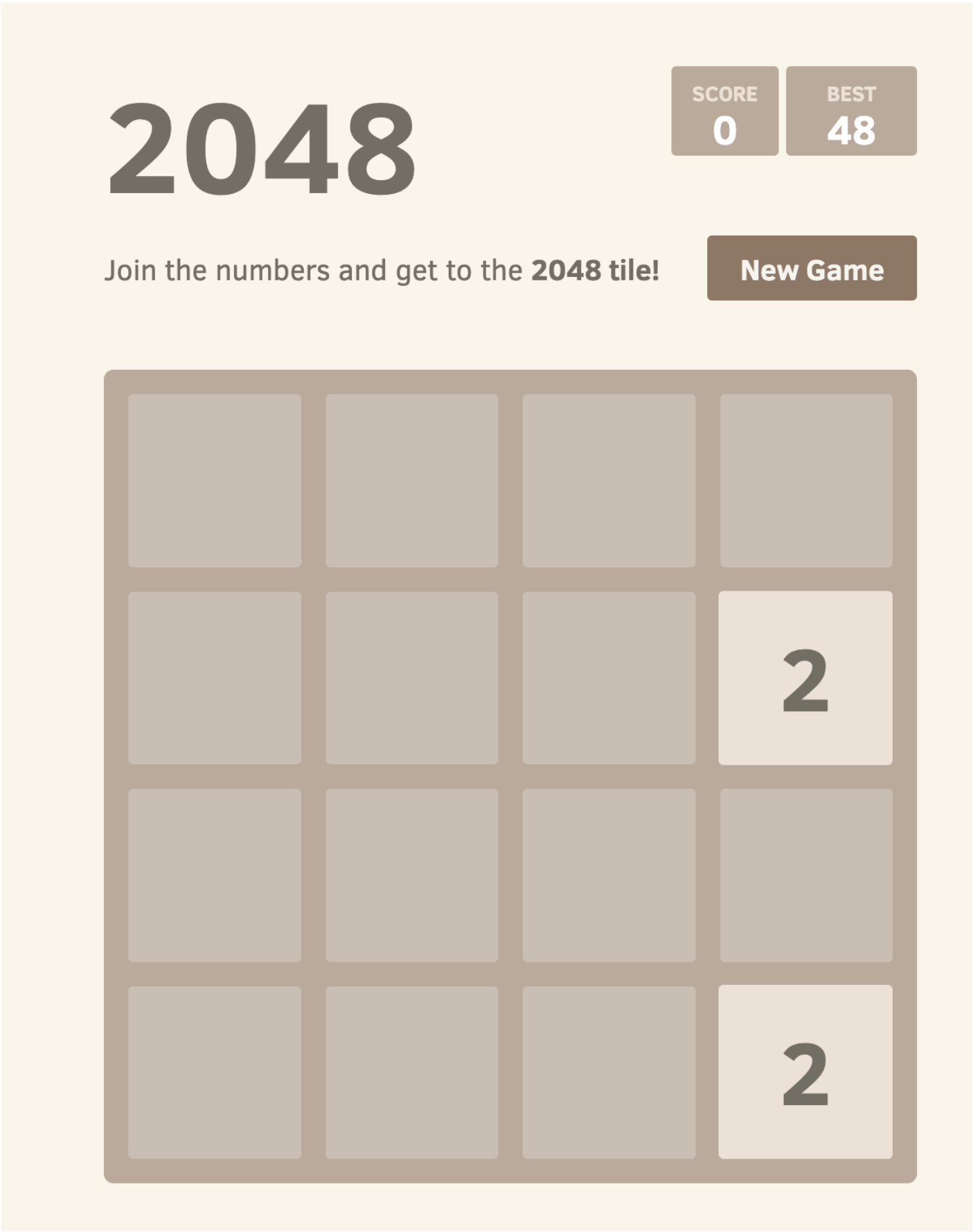
```
NAME           CLASS   HOSTS   ADDRESS
                     PORTS   AGE
ingress-2048   alb     *       k8s-game2048-ingress2-eb379a0f83-378466616.region-
code.elb.amazonaws.com   80      31s
```

在开始以下步骤之前，您需要等待几分钟，以便应用程序负载均衡器（ALB）完成预置。

4. 打开 Web 浏览器，并输入上一步中的 ADDRESS 以访问该 Web 应用程序。例如：

```
k8s-game2048-ingress2-eb379a0f83-378466616.region-code.elb.amazonaws.com
```

您应该会在浏览器中看到 2048 游戏。开始玩吧！


2048
SCORE
0
BEST
48
Join the numbers and get to the 2048 tile!
New Game
2
2

使用 Amazon EKS 自动模式持久保存数据

现在 2048 游戏已在 Amazon EKS 集群上启动并运行，可以使用 Amazon EKS 自动模式的块存储功能来确保安全地持久保存游戏数据。

1. 创建一个名为 `storage-class.yaml` 的文件：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. 应用 `StorageClass`：

```
kubectl apply -f storage-class.yaml
```

3. 创建持久卷注册声明（PVC）以请求存储游戏数据。创建一个名为 `ebs-pvc.yaml` 的文件，并将以下内容添加到该文件：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: game-data-pvc
  namespace: game-2048
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: auto-ebs-sc
```

4. 将该 PVC 应用到集群：

```
kubectl apply -f ebs-pvc.yaml
```

您应看到以下响应输出：

```
persistentvolumeclaim/game-data-pvc created
```

5. 现在，您需要更新 2,048 游戏部署才能使用此 PVC 存储数据。以下部署配置为使用 PVC 来存储游戏数据。创建一个名为 ebs-deployment.yaml 的文件，并将以下内容添加到该文件：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  replicas: 3  # Adjust the number of replicas as needed
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - name: app-2048
          image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          volumeMounts:
            - name: game-data
              mountPath: /var/lib/2048
      volumes:
        - name: game-data
          persistentVolumeClaim:
            claimName: game-data-pvc
```

6. 应用更新后的部署：

```
kubectl apply -f ebs-deployment.yaml
```

您应看到以下响应输出：

```
deployment.apps/deployment-2048 configured
```

完成这些步骤后，集群上的 2048 游戏现已设置为使用 Amazon EKS 自动模式的块存储功能持久保存数据。这样可以确保即使在容器组或节点出现故障，游戏进度和数据也同样安全。

如果您喜欢本教程，请给我们反馈，以便我们可以为您提供更多与此类似的应用场景专用快速入门教程。

清理

为避免将来产生费用，您需要手动删除关联的 CloudFormation 堆栈，以删除在本指南中创建的所有资源，包括 VPC 网络。

删除 CloudFormation 堆栈：

```
eksctl delete cluster -f ./cluster-config.yaml
```

通过示例学习 Amazon EKS

概览

本 Amazon EKS 用户指南包含通过 命令行或 Amazon Web Services 管理控制台创建第一个 EKS 集群的通用过程，以及有关所有主要 Amazon EKS 组件的可靠参考。不过，作为 Amazon EKS 集群管理员或开发人员，除本指南之外，您还可以通过其他网站中存在的学习路径来更深入了解 Amazon EKS。借助这些网站，您可以学习：

- 设置特定类型的集群。具体的集群类型可能取决于您的工作负载类型或安全要求。例如，您可能需要优化集群以运行批处理、机器学习或计算密集型工作负载。
- 强化您的集群。您可以向集群添加各种高级功能，以提供可观测性、灵活存储、自动扩缩或专用集群网络等功能。
- 自动更新。借助 GitOps 等功能，您可以进行相应的设置，以根据 Git 存储库中相关组件发生的变化自动预置集群基础设施和工作负载。
- 使用高级集群设置工具。您不仅可以利用 `eksctl` 快速创建集群，此外还可以借助多种其他的工具，更轻松地配置和升级更复杂的集群。这包括 Terraform 和 CloudFormation 等工具。

要启动自己的 Amazon EKS 学习路径，我建议您访问本页中描述的一些网站。如果您在此过程中遇到问题，还可以使用一些资源来帮助解决问题。例如，您可以通过 Re:post 知识中心搜索有关 Amazon EKS 相关支持问题的支持数据库。此外，Amazon EKS 最佳实践指南还提供了有关设置生产级集群的最佳方法提示。

Amazon EKS 研讨会

Amazon EKS 讲习会是一个非常实用的学习平台，从 Kubernetes 和容器的基础知识开始，为集群管理员讲解 Amazon EKS 的重要功能。您可以通过以下方式参加 Amazon EKS 讲习会：

- Amazon EKS 基础知识：观看概述页面上的视频，了解 Amazon EKS 如何在 Amazon 云上实现相关 Kubernetes 功能。如果需要了解更多有关 Kubernetes 的基础知识，可以观看什么是 Kubernetes 视频。
- Amazon EKS 设置：如果您有 Amazon 账户，则可通过设置部分来了解如何设置 CloudShell 环境以便创建集群。在这一部分中，您可以选择 eksctl（一个简单的集群创建命令行）和 Terraform（一种更类似于基础设施即代码的集群创建方法）来创建 Amazon EKS 集群。

- Amazon EKS 入门：从示例应用部分试用一个简单的网上商店。您可以将此示例用于其他练习。在这一部分中，您还可以了解如何打包容器映像，以及如何使用 Kubernetes 容器组、部署、服务、StatefulSets 和命名空间来管理微服务。然后使用 Kustomize 将更改部署到 Kubernetes 清单中。
- Amazon EKS 基本功能：此讲习会使用 Amazon Load Balancer Controller 等 Amazon 功能，演示了如何对外公开您的应用程序。对于存储，此讲习会演示了如何在 Amazon 中使用 Amazon EBS 进行块存储，使用 Amazon EFS 进行文件系统存储，以及使用 Amazon FSx for NetApp ONTAP 来管理 ONTAP 文件系统。对于节点管理，此讲习会提供了有关设置托管式节点组的帮助。
- Amazon EKS 高级功能：此 Amazon EKS 讲习会提供演示了更多高级功能，包括有关以下功能设置的实验：
  - 自动扩缩：这包括节点自动扩缩（使用集群自动扩缩器或 Karpenter）和工作负载自动扩缩（使用横向容器组自动扩缩器和集群比例自动扩缩器）。
  - 可观测性：通过一系列可观察性实验，来了解日志记录、OpenSearch、Amazon EKS 上的 Container Insights 以及使用 Kubecost 实现成本可见性等功能。
  - 安全性：通过这组安全性实验，您可以了解密钥管理、Amazon GuardDuty、容器组安全标准和 Kyverno 策略管理等功能。
  - 联网：通过联网实验来学习 Amazon EKS 的联网功能，包括 Amazon VPC CNI（支持网络插件）和 Amazon VPC Lattice（用于跨 VPC 和用户账户配置集群）。
  - 自动化：通过自动化实验，您可以分步详细了解管理集群和项目的 GitOps 方法，例如适用于 Kubernetes 的 Amazon 控制器和用于管理 Amazon EKS 控制面板的 Crossplane。

Amazon EKS 集群设置动手实验教程

Amazon 社区网站提供了一系列 Amazon EKS 集群设置教程，可为您提供有关创建特殊用途 Amazon EKS 集群以及以各种方式强化这些集群的帮助。这些教程分为三种类型：

集群构建类

此类教程可为您提供有关构建特殊用途集群的帮助。特殊用途包括下列功能：

- 运行基于 IPv6 的全球可扩展应用程序
- 运行异步批处理任务
- 运行大流量微服务
- 在 Fargate 上使用 Karpenter 进行自动扩缩
- 运行金融工作负载

- 运行 Windows 托管式节点组

集群强化类

如果您已经拥有集群，则可以对现有的集群进行扩展和强化以便运行专门的工作负载，以及通过其他方式强化集群。此类教程包含了下列集群强化方式：

- 使用 EFS CSI 提供存储解决方案
- 使用 EBS CSI 提供动态数据库存储
- 使用 Amazon Load Balancer Controller 公开 IPv4 集群上的应用程序
- 使用 Amazon Load Balancer Controller 公开 IPv6 集群上的应用程序

Amazon 服务优化类

此类教程有助您更好地将集群与 Amazon 服务集成。此类教程可在下列方面为您提供帮助：

- 使用 ExternalDNS 管理微服务的 DNS 记录
- 使用 CloudWatch 监控应用程序
- 使用 SQS 和 EFS 存储管理异步任务
- 使用来自工作负载中的 Amazon Secrets Manager 密钥
- 使用 Fargate、NGINX 和 ACM PCA 设置 mTLS

Amazon EFS 示例

Amazon EKS 示例存储库存储了可用于 Amazon EKS 的清单。借助这些清单，您可以在 Amazon EKS 中试用不同类型的应用程序或创建特定类型的 Amazon EKS 集群。这些示例包含下列用途的清单：

- 创建 Amazon Amazon EKS Fargate 集群
- 使用现有 IAM 角色创建集群
- 将 Ubuntu 托管式节点组添加到集群
- 使用卷快照备份和还原容器组存储
- 恢复作为 PVC 挂载到多个账户的 EBS 卷
- 为使用经典负载均衡器的 NGINX 入口控制器启用代理协议
- 配置将 Fargate 上的日志记录到 Amazon OpenSearch
- 使用 Web 联合身份提供者运行 Python SDK

- 在 NFS CSI 控制器上部署示例应用程序
- 将卷快照用于 StatefulSets
- 跨不同可用区的节点部署容器组

请记住，这些示例仅供学习和测试之用，并非旨在用于生产环境。

Amazon 教程

Amazon 教程网站发布了一些 Amazon EKS 教程，此外还提供了搜索工具，可用于查找在 Amazon 网站（例如 Amazon 社区网站）上发布的其他教程。直接在该网站上发布的 Amazon EKS 教程包括：

- 在 Amazon EKS 上部署容器化 Web 应用程序
- 以更低的成本运行 Kubernetes（Amazon EKS 和竞价型实例）
- 如何优化 Kubernetes 上的 Jenkins 作业成本

开发人员讲习会

如果您是软件开发人员，并且想要创建或重构应用程序以在 Amazon EKS 上运行，那么 Amazon EKS 开发人员讲习会是一个不错的敲门砖。该讲习会不仅提供了有关如何构建容器化应用程序的帮助，还提供了有关如何将容器部署到容器注册表（ECR），然后再部署到 Amazon EKS 集群的帮助。

您可以首先学习 Amazon EKS Python 讲习会，了解重构 python 应用程序的过程，然后设置开发环境以做好部署应用程序的准备。仔细阅读有关容器、Kubernetes 和 Amazon EKS 的章节，了解如何做好准备以在这些环境中运行容器化应用程序。

Terraform 讲习会

`eksctl` 是一种十分简便的集群创建工具，而对于更复杂的基础设施即代码类 Amazon EKS 部署，更常用的 Amazon EKS 集群创建和管理工具是 Terraform。Terraform Amazon EKS 讲习会讲解了如何使用 Terraform 构建 Amazon VPC、创建 Amazon EKS 集群以及为集群添加可选的增强功能。尤其是其中有一章节介绍了如何创建私有 Amazon EKS 集群

Amazon Amazon EKS 培训

Amazon 提供了学习 Amazon EKS 的正规培训。这一为期三天的培训课程名为 Running Containers on Amazon Elastic Kubernetes Service，将会讲授下列方面的知识：

- Kubernetes 和 Amazon EKS 基础知识
- 如何构建 Amazon EKS 集群
- 使用 Amazon IAM 和 Kubernetes RBAC 授权来保护 Amazon EKS
- GitOps 自动化工具
- 监控工具
- 优化成本、效率和弹性的技巧

开始使用 Amazon EKS

在阅读入门指南之前，请确保您已设置好使用 Amazon EKS。有关更多信息，请参阅 设置。

有两个入门指南可用于在 Amazon EKS 中使用节点创建新的 Kubernetes 集群：

- 开始使用 Amazon EKS – eksctl – 此入门指南可帮助您使用 `eksctl` 来安装开始使用 Amazon EKS 所需的所有资源。eksctl 是用于在 Amazon EKS 上创建和管理 Kubernetes 集群的简单命令行实用程序。在教程的最后，将为您提供一个运行中的 Amazon EKS 集群，您可向其部署应用程序。这是开始使用 Amazon EKS 的最快、最简单的方式。
- 开始使用 Amazon EKS – Amazon Web Services 管理控制台和 Amazon CLI – 此入门指南可帮助您使用 Amazon Web Services 管理控制台和 Amazon CLI 创建开始使用 Amazon EKS 所需的所有资源。在教程的最后，将为您提供一个运行中的 Amazon EKS 集群，您可向其部署应用程序。在本指南中，您将手动创建 Amazon EKS 集群所需的每个资源。这些程序可让您了解每个资源的创建方式以及资源之间的相互交互方式。

我们还提供以下参考资料：

- 有关动手实践教程的集合，请参阅 Amazon 社区 上的 EKS 集群设置。
- 有关代码示例，请参阅使用 Amazon SDK 的 Amazon EKS 代码示例。

开始使用 Amazon EKS – EKS 自动模式

与其他 EKS 入门体验一样，使用 EKS 自动模式创建第一个集群会将集群本身的管理委派给 Amazon。但 EKS 自动模式进一步扩展了 EKS 自动化水平，承担了设置工作负载基础设施（节点、网络和各种服务）所需的许多基本服务的责任，从而让您可以更轻松地管理节点以及根据工作负载需求纵向扩展。

选择以下方法之一来创建启用了 EKS 自动模式的集群：

- the section called “ Amazon CLI”：使用 `aws` 命令行界面创建集群。
- the section called “管理控制台”：使用 Amazon Web Services 管理控制台 创建集群。
- the section called “eksctl CLI”：使用 `eksctl` 命令行界面创建集群。

如果要比较创建第一个 EKS 集群的不同方法，您需要知道 EKS 自动模式已将额外的集群管理职责交给 Amazon，包括安装相关组件以完成下列任务：

- 根据工作负载需求的增加和减少启动和扩展节点。
- 定期升级集群本身（控制面板）、节点操作系统以及在节点上运行的服务。
- 选择决定节点存储大小和速度以及容器组网络配置等方面的默认设置。

要详细了解 EKS 自动模式集群的特点，请参阅EKS 自动模式。

开始使用 Amazon EKS – `eksctl`

> **Note**
>
> 本主题介绍在不启用 EKS 自动模式的情况下开始使用。
> EKS 自动模式可自动执行集群计算、存储和联网等方面的例行任务。了解如何开始使用 Amazon EKS 自动模式。

此指南可帮助您使用 `eksctl`（用于在 Amazon EKS 上创建和管理 Kubernetes 集群的简单命令行实用程序）来安装开始使用 Amazon Elastic Kubernetes Service (Amazon EKS) 所需的所有资源。在本教程的最后，将为您提供一个运行中的 Amazon EKS 集群，您可向其部署应用程序。

本指南中的程序可自动为您创建在您使用 Amazon Web Services 管理控制台 创建集群时必须手动创建的一些资源。如果您希望手动创建大多数资源并更好地了解它们之间的交互方式，请使用 Amazon Web Services 管理控制台创建集群和执行计算。有关更多信息，请参阅 the section called “创建集群（控制台和 CLI）”。

先决条件

在开始使用本教程之前，您必须按设置以使用 Amazon EKS 中所述安装并配置 Amazon CLI、kubectl 和 eksctl 工具。

第 1 步：创建 Amazon EKS 集群和节点

> **Important**
>
> 为了尽可能简单快速地入门，本主题包括创建具有原定设置的集群和节点的步骤。在创建用于生产使用的集群和节点之前，我们建议您熟悉所有设置，并使用符合您要求的设置部署集群和节点。有关更多信息，请参阅 the section called “创建集群” 和 管理计算。一些设置仅在创建集群和节点时可以启用。

您可以使用下列节点类型之一创建集群。要了解有关各个类型的更多信息，请参阅 管理计算。部署集群后，您可以添加其他节点类型。

- Fargate – Linux – 如果要在 the section called “ Amazon Fargate” 上运行 Linux 应用程序，请选择此类型的节点。Fargate 是一种无服务器计算引擎，允许部署 Kubernetes Pod 而不管理 Amazon EC2 实例。
- 托管式节点 – Linux – 如果要在 Amazon EC2 实例上运行Amazon Linux 应用程序，请选择此类型的节点。虽然本指南中未作介绍，但您还可以向集群添加 Windows 自托管节点和 Bottlerocket 节点。

使用以下命令创建 Amazon EKS 集群。将 *my-cluster* 替换为您自己的值。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。请将 *region-code* 替换为 Amazon EKS 支持的任何 Amazon 区域。有关 Amazon 区域列表，请参阅《Amazon 一般参考》指南中的 Amazon EKS 服务端点和配额。

Example

Fargate - Linux

```
eksctl create cluster --name my-cluster --region region-code --fargate
```

Managed nodes - Linux

```
eksctl create cluster --name my-cluster --region region-code
```

创建集群需要几分钟时间。在创建过程中，您将看到几行输出。输出的最后一行类似于以下示例行。

```
[...]
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

`eksctl` 在 `~/.kube/config` 中创建了 `kubectl` 配置文件，或在计算机上 `~/.kube/config` 中的现有配置文件中添加了新集群的配置。

完成集群创建后，在 Amazon CloudFormation 控制台 中查看名为 `eksctl-`*`my-cluster`*`-cluster` 的 Amazon CloudFormation 堆栈，查看创建的所有资源。

第 2 步：查看 Kubernetes 资源

1. 查看您的集群节点。

```
kubectl get nodes -o wide
```

示例输出如下。

Example

Fargate - Linux

```
NAME                                                  STATUS   ROLES    AGE
 VERSION              INTERNAL-IP   EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
                CONTAINER-RUNTIME
fargate-ip-192-0-2-0.region-code.compute.internal   Ready    <none>
 8m3s    v1.2.3-eks-1234567   192.0.2.0     <none>        Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
fargate-ip-192-0-2-1.region-code.compute.internal   Ready    <none>
 7m30s   v1.2.3-eks-1234567   192-0-2-1     <none>        Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
```

Managed nodes - Linux

```
NAME                                        STATUS   ROLES    AGE    VERSION
        INTERNAL-IP   EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
   CONTAINER-RUNTIME
ip-192-0-2-0.region-code.compute.internal   Ready    <none>   6m7s
  v1.2.3-eks-1234567   192.0.2.0     192.0.2.2     Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
ip-192-0-2-1.region-code.compute.internal   Ready    <none>   6m4s
  v1.2.3-eks-1234567   192.0.2.1     192.0.2.3     Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
```

如需详细了解输出中显示的内容，请参阅 the section called “访问集群资源”。

2. 查看在集群上运行的工作负载。

```
kubectl get pods -A -o wide
```

示例输出如下。

Example

Fargate - Linux

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE   IP
          NODE                                                NOMINATED NODE
 READINESS GATES
kube-system   coredns-1234567890-abcde   1/1     Running   0          18m
 192.0.2.0   fargate-ip-192-0-2-0.region-code.compute.internal   <none>
 <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          18m
 192.0.2.1   fargate-ip-192-0-2-1.region-code.compute.internal   <none>
 <none>
```

Managed nodes - Linux

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE     IP
          NODE                                          NOMINATED NODE   READINESS
 GATES
kube-system   aws-node-12345             1/1     Running   0          7m43s
 192.0.2.1   ip-192-0-2-1.region-code.compute.internal   <none>           <none>
kube-system   aws-node-67890             1/1     Running   0          7m46s
 192.0.2.0   ip-192-0-2-0.region-code.compute.internal   <none>           <none>
kube-system   coredns-1234567890-abcde   1/1     Running   0          14m
 192.0.2.3   ip-192-0-2-3.region-code.compute.internal   <none>           <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          14m
 192.0.2.4   ip-192-0-2-4.region-code.compute.internal   <none>           <none>
kube-system   kube-proxy-12345           1/1     Running   0          7m46s
 192.0.2.0   ip-192-0-2-0.region-code.compute.internal   <none>           <none>
kube-system   kube-proxy-67890           1/1     Running   0          7m43s
 192.0.2.1   ip-192-0-2-1.region-code.compute.internal   <none>           <none>
```

如需详细了解输出中显示的内容，请参阅 the section called “访问集群资源”。

第 3 步：删除集群和节点

在使用完成针对本教程而创建的集群和节点后，应使用下面的命令删除这些集群和节点，从而将它们清除。如果要在清除该集群前对其执行更多操作，请参阅 the section called “后续步骤”。

```
eksctl delete cluster --name my-cluster --region region-code
```

后续步骤

以下文档主题可帮助您扩展集群的此功能。

- 将示例应用程序部署到您的集群。
- 创建集群的 IAM 主体是唯一可以使用 `kubectl` 或 Amazon Web Services 管理控制台 调用 Kubernetes API 服务器的主体。如果您希望其他 IAM 主体拥有访问您的集群的权限，您需要添加它们。有关更多信息，请参阅the section called “Kubernetes API 访问权限” 和the section called “所需的权限”。
- 我们建议您先熟悉集群和节点的所有设置，然后再部署集群用于生产环境。创建集群时必须进行一些设置（例如启用 SSH 访问 Amazon EC2 节点）。
- 为了提高集群的安全性，请配置 Amazon VPC 容器网络接口插件以将 IAM 角色用于服务账户。

开始使用 Amazon EKS – Amazon Web Services 管理控制台和 Amazon CLI

> **Note**
>
> 本主题介绍在不启用 EKS 自动模式的情况下开始使用。它使用托管式节点组来部署节点。EKS 自动模式可自动执行集群计算、存储和联网等方面的例行任务。了解如何开始使用 Amazon EKS 自动模式。EKS 自动模式是部署节点的首选方法。

本指南帮助您创建通过 Amazon Web Services 管理控制台和 Amazon CLI 开始使用 Amazon Elastic Kubernetes Service（Amazon EKS）所需的所有资源。在本指南中，您将手动创建每个资源。在本教程的最后，将为您提供一个运行中的 Amazon EKS 集群，您可向其部署应用程序。

本指南中的步骤可让您完全了解每个资源的创建方式以及资源之间如何相互交互。如果您希望系统自动为您创建大部分资源，请使用 `eksctl` CLI 创建集群和节点。有关更多信息，请参阅 the section called “创建集群（eksctl）”。

先决条件

在开始使用本教程之前，您必须安装并配置创建和管理 Amazon EKS 集群所需的以下工具和资源。

- Amazon CLI – 与 Amazon 服务（包括 Amazon EKS）结合使用的命令行工具。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的安装。在安装 Amazon CLI 后，建议您还要对其进行配

置。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的使用 aws configure 快速配置。请注意，需要 Amazon CLI v2 才能使用本页中显示的 update-kubeconfig 选项。

- **kubectl** – 用于与 Kubernetes 集群一起使用的命令行工具。有关更多信息，请参阅 the section called “设置 kubectl 和 eksctl”。
- 所需的 IAM 权限 – 您正在使用的 IAM 安全主体必须具有使用 Amazon EKS IAM 角色、服务相关角色、Amazon CloudFormation、VPC 和相关资源的权限。有关更多信息，请参阅《IAM 用户指南》中的操作和编辑服务相关角色。您必须以同一用户身份完成本指南中的所有步骤。要查看当前用户，请运行以下命令：

```
aws sts get-caller-identity
```

我们建议您在 Bash Shell 中完成本主题中的步骤。如果您没有使用 Bash Shell，则某些脚本命令（例如行延续字符以及变量的设置和使用方式）需要调整 Shell。此外，您的 Shell 的引用和转义规则可能有所不同。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的在 Amazon CLI 中将引号和字符串结合使用。

第 1 步：创建 Amazon EKS 集群

> **Important**
>
> 为了尽可能简单快速地入门，本主题包括创建具有原定设置的集群的步骤。创建用于生产用途的集群前，我们建议您熟悉所有设置，并使用符合您要求的设置部署集群。有关更多信息，请参阅 the section called “创建集群”。一些设置仅在创建集群时可以启用。

1. 创建具有公有和私有子网且符合 Amazon EKS 要求的 Amazon VPC。请将 *region-code* 替换为 Amazon EKS 支持的任何 Amazon 区域。有关 Amazon 区域列表，请参阅《Amazon 一般参考》指南中的 Amazon EKS 服务端点和配额。可以将 *my-eks-vpc-stack* 替换为您任选的名称。

```
aws cloudformation create-stack \
  --region region-code \
  --stack-name my-eks-vpc-stack \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

> **Tip**
>
> 如需查看上一个命令创建的所有资源的列表，请打开 Amazon CloudFormation 控制台（https://console.aws.amazon.com/cloudformation/）。选择 *my-eks-vpc-stack* 堆栈，然后选择 Resources（资源）选项卡。

2. 创建一个集群 IAM 角色，并将所需的 Amazon EKS IAM 托管式策略附加到该角色。Amazon EKS 托管的 Kubernetes 集群会代表您调用其他Amazon服务，以管理您用于该服务的资源。

   a. 将以下内容复制到名为 *eks-cluster-role-trust-policy.json* 的文件中。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

   b. 创建角色。

```
aws iam create-role \
  --role-name myAmazonEKSClusterRole \
  --assume-role-policy-document file://"eks-cluster-role-trust-policy.json"
```

   c. 将所需的 Amazon EKS 托管 IAM 策略附加到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name myAmazonEKSClusterRole
```

3. 打开位于 eks/home#/clusters 的 Amazon EKS 控制台。

   确保在 Amazon 控制台右上角显示的区域是您要在其中创建集群的 Amazon 区域。如果不是，请选择 Amazon 区域名称旁边的下拉菜单，然后选择要使用的 Amazon 区域。

4. 选择创建集群。如果没有看到此选项，请先选择左侧导航面板中的集群。

5. 在 Configure cluster（配置集群）页面上，请执行以下操作：
   a. 选择自定义配置并禁用使用 EKS 自动模式。（如果您更喜欢 EKS 自动模式集群，请参阅the section called “管理控制台”。）
   b. 输入集群的名称，例如 *my-cluster*。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   c. 对于 Cluster Service Role（集群服务角色），请选择 *myAmazonEKSClusterRole*。
   d. 其余设置保留为默认值，然后选择 Next（下一步）。
6. 在 Specify networking（指定联网）页面中，请执行以下操作：
   a. 请从 VPC 下拉列表中选择在上一步创建的 VPC ID。它类似于 ** | my-eks-vpc-stack-VPC*。
   b. 请从子网下拉列表中选择在上一步创建的子网。子网类似于 ** | my-eks-vpc-stack-**。
   c. 请从其他安全组下拉列表中选择在上一步创建的安全组。安全组类似于 ** | my-eks-vpc-stack-ControlPlaneSecurityGroup-**。
   d. 其余设置保留为默认值，然后选择 Next（下一步）。
7. 在配置可观测性页面上，选择下一步。
8. 在选择附加组件页中，选择下一个。

   有关附加组件的更多信息，请参阅 the section called “Amazon EKS 附加组件”。
9. 在配置选定的附加组件设置页面上，选择下一个。
10. 在审核和创建页面，选择创建。

    集群名称右侧的集群状态会保持为正在创建几分钟，直至集群调配过程完成。在该状态变为有效之前，请勿继续执行下一步。

> **Note**
>
> 您可能会收到一个错误，指示请求中的可用区之一没有足够容量来创建 Amazon EKS 集群。如果发生这种情况，错误输出将包含可支持新集群的可用区。再次尝试使用至少两个位于您账户中支持的可用区的子网创建集群。有关更多信息，请参阅 the section called “容量不足”。

第 2 步：将计算机配置为与您的集群通信

在本部分中，您将为集群创建一个 kubeconfig 文件。此文件中的设置会启用 kubectl CLI 与您的集群进行通信。

在继续操作之前，请确保您已在步骤 1 中成功完成集群创建。

1. 为集群创建或更新 kubeconfig 文件。将 *region-code* 替换为您要在其中创建集群的 Amazon 区域。将 *my-cluster* 替换为您的集群的名称。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

   默认情况下，config 文件创建在 ~/.kube 中或者新集群的配置已添加到 ~/.kube 的现有 config 文件中。

2. 测试配置。

```
kubectl get svc
```

> **Note**
>
> 如果您收到任何授权或资源类型错误，请参阅故障排除主题中的 the section called “未经授权或访问被拒绝 (kubectl)”。

   示例输出如下。

```
NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   1m
```

第 3 步：创建节点

> **Important**
>
> 为了尽可能简单快速地入门，本主题包含了大部分使用默认设置创建节点的步骤。在创建用于生产用途的节点之前，我们建议您熟悉所有设置，并使用符合您要求的设置部署节点。有关更多信息，请参阅 管理计算。一些设置仅在创建节点时可以启用。

此过程将集群配置为使用托管式节点组来创建节点，指定您在之前步骤中创建的子网和节点 IAM 角色。这让您可以在 Amazon EC2 实例上运行 Amazon Linux 应用程序。

要详细了解在 EKS 中配置节点的不同方法，请参阅管理计算。部署集群后，您可以添加其他节点类型。虽然本指南中未作介绍，但您还可以向集群添加 Windows 自托管节点和 Bottlerocket 节点。

创建 EC2 Linux 托管节点组

1. 创建节点 IAM 角色并向其附加所需的 Amazon EKS IAM 托管策略。Amazon EKS 节点 `kubelet` 守护进程代表您调用 Amazon API。节点通过 IAM 实例配置文件和关联的策略获得这些 API 调用的权限。

   a. 将以下内容复制到名为 `node-role-trust-policy.json` 的文件中。

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "ec2.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

   b. 创建节点 IAM 角色。

   ```
   aws iam create-role \
     --role-name myAmazonEKSNodeRole \
     --assume-role-policy-document file://"node-role-trust-policy.json"
   ```

c. 将所需的托管 IAM policy 附加到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name myAmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name myAmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name myAmazonEKSNodeRole
```

d. 打开位于 eks/home#/clusters 的 Amazon EKS 控制台。

e. 选择您在步骤 1：创建 Amazon EKS 集群中创建的集群的名称，例如 *my-cluster*。

f. 在 ***my-cluster*** 页面上，请执行以下操作：

g. 请选择 Compute（计算）选项卡。

h. 请选择 Add Node Group（添加节点组）。

2. 在 Configure Node Group（配置节点组）页面上，请执行以下操作：

a. 对于名称，输入托管节点组的唯一名称，例如 *my-nodegroup*。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。

b. 对于 Node IAM role name（节点 IAM 角色名称），请选择您在上一步中创建的 *myAmazonEKSNodeRole* 角色。我们建议让每个节点组使用各自的唯一 IAM 角色。

c. 选择下一步。

3. 请在 Set compute and scaling configuration（设置计算和扩缩配置）页面上，接受默认值并选择 Next（下一步）。

4. 请在 Specify networking（指定联网）页面，接受默认值，选择 Next（下一步）。

5. 在 Review and create (审核并创建) 页面上，审核托管节点组配置并选择 Create (创建)。

6. 几分钟后，Node Group configuration（节点组配置）部分中的 Status（状态）将从 Creating（正在创建）变为 Active（有效）。在该状态变为有效之前，请勿继续执行下一步。

第 4 步：查看资源

您可以查看节点和 Kubernetes 工作负载。

1. 在左侧导航窗格中，选择集群。在 Clusters（集群）列表中，选择您创建的集群名称，例如 *my-cluster*。

2. 在 ***my-cluster*** 页面上，选择以下选项：
   a. 计算选项卡：您可以看到为集群部署的节点列表。您可以选择节点的名称以查看有关该节点的详细信息。
   b. 资源选项卡 – 您将看到默认情况下部署到 Amazon EKS 集群的所有 Kubernetes 资源。在控制台中选择任何资源类型以了解有关它的更多信息。

第 5 步：删除资源

在使用完成针对本教程而创建的集群和节点后，应删除这些资源。如果要在删除资源前对此集群执行更多操作，请参阅 the section called “后续步骤”。

1. 删除您创建的任何节点组配置文件。
   a. 打开位于 eks/home#/clusters 的 Amazon EKS 控制台。
   b. 在左侧导航窗格中，选择集群。请在集群列表中，选择 *my-cluster*。
   c. 请选择 Compute（计算）选项卡。
   d. 如果您创建了节点组，请选择 *my-nodegroup* 节点组，然后选择 Delete（删除）。输入 *my-nodegroup*，然后选择删除。
   e. 在节点组配置文件删除后再继续操作。
2. 请删除集群。
   a. 在左侧导航窗格中，选择集群。请在集群列表中，选择 *my-cluster*。
   b. 选择删除集群。
   c. 输入 *my-cluster*，然后选择删除。在集群删除后再继续操作。
3. 删除创建的 VPC Amazon CloudFormation 堆栈。
   a. 通过以下网址打开 CloudFormation 控制台：https://console.aws.amazon.com/cloudformation/。
   b. 请选择 *my-eks-vpc-stack* 堆栈，然后选择 Delete（删除）。
   c. 在删除 ***my-eks-vpc-stack*** 确认对话框中，选择删除堆栈。
4. 删除您创建的 IAM 角色。
   a. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
   b. 在左侧导航窗格中，选择 角色。
   c. 请从列表中选择您创建的每个角色（***myAmazonEKSClusterRole*** 以及 *myAmazonEKSNodeRole*）。请选择 Delete（删除），输入请求的确认文本，然后选择 Delete（删除）。

后续步骤

以下文档主题可帮助您扩展集群的此功能。

- 创建集群的 IAM 主体是唯一可以使用 `kubectl` 或 Amazon Web Services 管理控制台 调用 Kubernetes API 服务器的主体。如果您希望其他 IAM 主体拥有访问您的集群的权限，您需要添加它们。有关更多信息，请参阅the section called “Kubernetes API 访问权限”和the section called “所需的权限”。
- 将示例应用程序部署到您的集群。
- 在部署集群用于生产用途前，我们建议您熟悉集群和节点的所有设置。创建集群时必须进行一些设置（例如启用 SSH 访问 Amazon EC2 节点）。
- 为了提高集群的安全性，请配置 Amazon VPC 容器网络接口插件以将 IAM 角色用于服务账户。

使用 EKS 自动模式实现集群基础设施自动化

EKS 自动模式将 Kubernetes 集群的 Amazon 管理扩展到集群本身之外，从而让 Amazon 还可以设置和管理基础设施，确保工作负载的平稳运行。您可以将关键基础设施决策委派给 Amazon 并利用其专业知识进行日常运营。由 Amazon 管理的集群基础设施将多种 Kubernetes 功能作为核心组件包含在内，而不是作为附加组件，例如计算自动扩缩、容器和服务联网、应用程序负载均衡、集群 DNS、块存储和 GPU 支持等。

要开始使用，您可以部署一个新的 EKS 自动模式集群或在现有集群上启用 EKS 自动模式。您可以使用 eksctl、Amazon CLI、Amazon Web Services 管理控制台、EKS API 或偏好的基础设施即代码工具，部署、升级或修改 EKS 自动模式集群。

使用 EKS 自动模式时，您可以继续使用自己偏好的 Kubernetes 兼容工具。EKS 自动模式与 Amazon EC2、Amazon EBS 和 ELB 等 Amazon 服务集成，利用遵循最佳实践的 Amazon 云资源。这些资源可自动扩展、优化成本并定期更新，有助您尽可能降低运营成本和开销。

特征

EKS 自动模式提供以下高级别功能：

简化 Kubernetes 集群管理：EKS 自动模式通过提供运行开销极小的生产就绪型集群，来简化 EKS 的管理。使用 EKS 自动模式时，您可以放心地运行要求严苛的动态工作负载，无需深入的 EKS 专业知识。

应用程序可用性：EKS 自动模式会根据 Kubernetes 应用程序的需求动态添加或移除 EKS 集群中的节点。这尽可能减少了手动容量规划的需求，确保了应用程序的可用性。

效率：EKS 自动模式旨在优化计算成本，同时符合节点池和工作负载要求所定义的灵活性。此外还会终止未使用的实例，并将工作负载整合到其他节点上，从而提高成本效率。

安全性：EKS 自动模式使用对节点而言不可变的 AMI。这些 AMI 强制使用锁定软件，启用 SELinux 强制访问控制，并提供只读的根文件系统。此外，由 EKS 自动模式启动的节点最长生命周期为 21 天（并且您可以缩短此周期），之后会自动替换为新节点。这种方法通过定期循环节点来增强安全态势，也符合许多客户已经采用的最佳实践。

自动升级：EKS 自动模式使用最新补丁来确保 Kubernetes 集群、节点和相关组件保持最新状态，同时遵守您配置的容器组中断预算（PDB）和节点池中断预算（NDB）。在不超过 21 天最长生命周期的前提下，如果阻止 PDB 或其他配置会妨碍更新，则可能需要进行干预。

托管式组件：EKS 自动模式将 Kubernetes 和 Amazon 云功能等作为核心组件，而这些组件本来会作为附加组件管理。这包括对容器组 IP 地址分配、容器组网络策略、本地 DNS 服务、GPU 插件、运行状况检查器和 EBS CSI 存储的内置支持。

可自定义节点池和节点类：如果工作负载需要更改存储、计算或联网配置，则可以使用 EKS 自动模式创建自定义节点池和节点类。虽然不应编辑默认的节点池和节点类，但可以在默认配置之外添加新的自定义节点池或节点类，来满足自己的特定需求。

自动化组件

EKS 自动模式通过自动管理关键基础设施组件来简化 Amazon EKS 集群的操作。启用 EKS 自动模式可进一步减少 EKS 集群管理任务。

以下是自动管理的数据面板组件列表：

- 计算：对于许多工作负载而言，使用 EKS 自动模式后，您都无需关注 EKS 集群计算的许多方面。这些指令包括：
  - 节点：EKS 自动模式节点在设计上可视为设备。EKS 自动模式可完成以下任务：
    - 选择合适的 AMI 并配置运行工作负载所需的多种服务，无需干预。
    - 使用 SELinux 强制实施模式和只读根文件系统锁定对 AMI 上文件的访问权限。
    - 通过禁用 SSH 或 SSM 访问来禁止直接访问节点。
    - 包含 GPU 支持，以及适用于 NVIDIA 和 Neuron GPU 的单独内核驱动程序和插件，从而支持高性能工作负载。
    - 自动处理 EC2 竞价型实例中断通知和 EC2 实例运行状况事件
  - 自动扩缩：依托 Karpenter 自动扩缩功能，EKS 自动模式可以监控不可调度的容器组，并通过部署新节点来运行这些容器组。当工作负载终止时，EKS 自动模式会动态中断和终止不再需要的节点，从而优化资源使用。
  - 升级：通过接管节点的控制，简化 EKS 自动模式根据需要提供安全补丁以及操作系统和组件升级的能力。这些升级旨在尽可能减少工作负载的中断。EKS 自动模式强制实施最长 21 天的节点生命周期，以确保始终使用最新的软件和 API。
- 负载均衡：EKS 自动模式通过与 Amazon 弹性负载均衡服务集成，自动为 Kubernetes 服务和 Ingress 资源预置和配置负载均衡器，从而简化负载均衡任务。支持应用程序均衡器和网络负载均衡器的高级功能，管理其生命周期，并根据集群需求进行扩展。这种集成提供了符合 Amazon 最佳实践的生产就绪型负载均衡解决方案，使您可以专注于应用程序，而无需关注基础设施管理。
- 存储：EKS 自动模式通过设置卷类型、卷大小、加密策略以及节点终止时的删除策略，来为您配置临时存储。

- 联网：EKS 自动模式可自动执行容器组和服务连接的关键联网任务。这包括 IPv4/IPv6 支持以及使用辅助 CIDR 块来扩展 IP 地址空间。
- 身份和访问权限管理：您无需在 EKS 自动模式集群上安装 EKS 容器组身份代理。

有关这些组件的更多信息，请参阅the section called “工作方式”。

配置

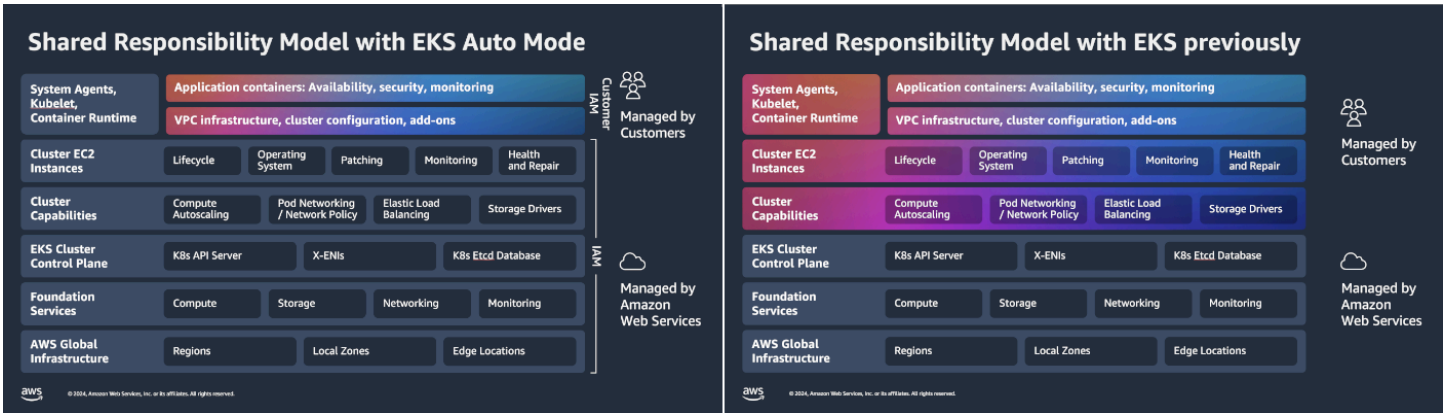
虽然 EKS 自动模式无需您干预即可有效地管理大部分数据面板服务，不过有时您可能需要更改某些服务的行为。您可以通过以下任一方式修改 EKS 自动模式集群的配置：

- Kubernetes 进程守护程序集：您可以使用 Kubernetes 进程守护程序集，而不修改节点上安装的服务。进程守护程序集旨在由 Kubernetes 管理，不过是在集群中的每个节点上运行的。这样，您可以添加特殊服务来监控或以其他方式观察节点。
- 自定义节点池和节点类：默认节点池和节点类由 EKS 自动模式配置，不应编辑。要自定义节点行为，您可以针对应用场景创建其他节点池或节点类，例如：
  - 选择特定的实例类型（例如，加速处理器或 EC2 竞价型实例）。
  - 出于安全或成本跟踪目的隔离工作负载。
  - 配置临时存储设置，例如 IOPS、大小和吞吐量等。
- 负载均衡：EKS 自动模式作为 Kubernetes 对象运行的某些服务（例如负载均衡），可以直接在 EKS 自动模式集群上配置。

要详细了解配置 EKS 自动模式的选项，请参阅the section called “Configure”。

责任共担模式

Amazon 责任共担模式定义了 Amazon 与客户之间的安全与合规责任。下面的图片和文字比较和对比 EKS 自动模式与 EKS 标准模式之间客户与 Amazon 责任的区别。



EKS 自动模式将 Kubernetes 基础设施的大部分共担责任从客户转移到 Amazon。使用 EKS 自动模式后，Amazon 承担了更多云安全责任，而云安全曾经是客户的责任，现在我们与客户共同承担。现在，客户可以更加专注于自己的应用程序，而 Amazon 负责管理底层基础设施。

客户责任

在 EKS 自动模式下，客户继续负责应用程序容器，包括可用性、安全性和监控。其还保留对 VPC 基础设施和 EKS 集群配置的控制。此模式使客户能够专注于应用程序特定的问题，同时将集群基础设施管理委派给 Amazon。通过 Amazon 附加组件可以在集群中加入可选的每节点功能。

Amazon 责任

使用 EKS 自动模式后，与未使用自动模式的 EKS 集群中已管理的组件相比，Amazon 的责任范围扩大到包括管理多个其他关键组件。具体而言，EKS 自动模式接管了启动的 EC2 实例的配置、管理、安全和扩展，以及负载均衡、IP 地址管理、联网策略和块存储等集群功能。在 EKS 自动模式下，以下组件由 Amazon 管理：

- 自动模式启动的 EC2 实例：Amazon 利用 Amazon EC2 托管实例处理节点的整个生命周期。EC2 托管实例负责操作系统配置、修补、监控和运行状况维护。在此模式下，实例本身和在其上运行的客户操作系统均由 Amazon 负责。这些节点使用为运行容器而优化的 Bottlerocket AMI 的变体。Bottlerocket AMI 具有锁定的软件、不可变的根文件系统，以及安全的网络访问（防止通过 SSH 或 SSM 直接通信）。
- 集群功能：Amazon 管理计算自动扩展、具有网络策略强制执行的容器组（pod）联网、Elastic Load Balancing 集成，以及存储驱动程序配置。
- 集群控制面板：与标准 EKS 一样，Amazon 继续管理 Kubernetes API 服务器、跨账户 ENI 和 etcd 数据库。
- 基础服务和全球基础设施：Amazon 负责底层计算、存储、联网和监控服务，以及区域、本地区域和边缘站点的全球基础设施。

创建启用 Amazon EKS 自动模式的集群

本章介绍如何使用各种工具和界面创建启用自动模式的 Amazon EKS 集群。自动模式通过自动配置和管理集群的计算、联网和存储基础设施，来简化集群的创建。您将了解如何使用 Amazon CLI、Amazon Web Services 管理控制台或 eksctl 命令行工具创建自动模式集群。

> **Note**
>
> EKS 自动模式需要使用 Kubernetes 版本 1.29 或更高版本。

根据需要选择偏好的工具：Amazon Web Services 管理控制台提供了直观的界面，非常适合用来了解 EKS 自动模式功能和创建单个集群。Amazon CLI 非常适合脚本化和自动化任务，尤其是在将集群创建集成到现有工作流或 CI/CD 管道时。eksctl CLI 提供了 Kubernetes 原生体验，建议熟悉 Kubernetes 工具并且希望使用合理的默认值来简化命令行操作的用户使用。

在开始操作之前，请确保您已安装并配置必要的先决条件，包括具有创建 EKS 集群的适当 IAM 权限。要了解如何安装 `kubectl`、`aws` 和 `eksctl` 等 CLI 工具，请参阅设置。

您可以使用 Amazon CLI、Amazon Web Services 管理控制台或 eksctl CLI 创建启用 Amazon EKS 自动模式的集群。

主题

- 使用 eksctl CLI 创建 EKS 自动模式集群
- 使用 Amazon CLI 创建 EKS 自动模式集群
- 使用 Amazon Web Services 管理控制台创建 EKS 自动模式集群

使用 eksctl CLI 创建 EKS 自动模式集群

此主题演示如何使用 eksctl 命令行界面（CLI）创建 Amazon EKS 自动模式集群。您可以通过运行单个 CLI 命令或应用 YAML 配置文件来创建自动模式集群。这两种方法提供的功能相同，不过 YAML 方法可以更精细地控制集群设置。

eksctl CLI 通过处理底层 Amazon 资源的创建和配置，来简化创建和管理 EKS 自动模式集群的过程。在开始操作之前，请确保在本地计算机上配置了必要的 Amazon 凭证和权限。本指南假设您熟悉 Amazon EKS 的基本概念，并且已经安装了必需的 CLI 工具。

> **Note**
>
> 您必须安装版本 `0.195.0` 或更高版本的 eksctl。有关更多信息，请参阅 GitHub 上的 eksctl releases。

使用 CLI 命令创建 EKS 自动模式集群

您必须已安装 `aws` 和 `eksctl` 工具。您必须以管理 Amazon 资源所需的足够权限登录 Amazon CLI，包括 EC2 实例、EC2 联网、EKS 集群和 IAM 角色等。有关更多信息，请参阅 设置。

运行以下命令以创建新的 EKS 自动模式集群

```
eksctl create cluster --name=<cluster-name> --enable-auto-mode
```

使用 YAML 文件创建 EKS 自动模式集群

您必须已安装 `aws` 和 `eksctl` 工具。您必须以管理 Amazon 资源所需的足够权限登录 Amazon CLI，包括 EC2 实例、EC2 联网、EKS 集群和 IAM 角色等。有关更多信息，请参阅 设置。

检查以下集群配置示例资源中的 EKS 自动模式配置选项。有关完整的 ClusterConfig 规范，请参阅 eksctl 文档。

Amazon 建议您启用 EKS 自动模式。如果这是您第一次创建 EKS 自动模式集群，请勿指定 `nodeRoleARN`，以便为 EKS 自动模式创建节点 IAM 角色。如果您的 Amazon 账户中已有节点 IAM 角色，Amazon 建议重复使用该角色。

Amazon 建议不要为 `nodePools` 指定任何值。EKS 自动模式将创建默认节点池。您可以使用 Kubernetes API 来创建其他节点池。

```
# cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: <cluster-name>
  region: <aws-region>

iam:
  # ARN of the Cluster IAM Role
  # optional, eksctl creates a new role if not supplied
```

```
  # suggested to use one Cluster IAM Role per account
  serviceRoleARN: <arn-cluster-iam-role>

autoModeConfig:
  # defaults to false
  enabled: boolean
  # optional, defaults to [general-purpose, system].
  # suggested to leave unspecified
  # To disable creation of nodePools, set it to the empty array ([]).
  nodePools: []string
  # optional, eksctl creates a new role if this is not supplied
  # and nodePools are present.
  nodeRoleARN: string
```

将 `ClusterConfig` 文件另存为 `cluster.yaml`，然后使用以下命令创建集群：

```
eksctl create cluster -f cluster.yaml
```

使用 Amazon CLI 创建 EKS 自动模式集群

EKS 自动模式集群可自动执行计算、存储和联网等方面的例行集群管理任务。例如，EKS 自动模式集群会自动检测何时需要增加节点，并相应预置新的 EC2 实例来满足工作负载的需求。

本主题将演示使用 Amazon CLI 创建新的 EKS 自动模式集群，以及（可选）部署示例工作负载的过程。

先决条件

- 已在您的设备上安装并配置了最新版本的 Amazon 命令行界面（Amazon CLI）。要查看当前版本，请使用 `aws --version`。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。
  - 使用创建 Amazon 资源所需的充分 IAM 权限登录 CLI，包括 IAM 策略、IAM 角色和 EKS 集群等。
- 已在您的设备上安装了 kubectl 命令行工具。Amazon 建议使用与 EKS 集群的 Kubernetes 版本相同的 kubectl 版本。要安装或升级 kubectl，请参阅the section called “设置 kubectl 和 eksctl”。

指定 VPC 子网

Amazon EKS 自动模式会将节点部署到 VPC 子网。创建 EKS 集群时，必须指定要部署节点的 VPC 子网。可以使用您 Amazon 账户中的默认 VPC 子网，也可以创建用于关键工作负载的专用 VPC。

- Amazon 建议为集群创建专用 VPC。了解如何the section called “创建 VPC”。
- EKS 控制台可帮助创建新的 VPC。了解如何the section called “管理控制台”。
- 您也可以使用您 Amazon 账户的默认 VPC。按照以下说明查找子网 ID。

查找默认 VPC 的子网 ID

使用 Amazon CLI：

1. 运行以下命令，列出默认 VPC 及其子网：

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$(aws ec2 describe-vpcs
 --query 'Vpcs[?IsDefault==`true`].VpcId' --output text)" --query 'Subnets[*].
{ID:SubnetId,AZ:AvailabilityZone}' --output table
```

2. 保存输出并记下子网 ID。

   示例输出：

```
----------------------------------------
|             DescribeSubnets          |
----------------------------------------
|   SubnetId        |   AvailabilityZone  |
|-------------------|---------------------|
|   subnet-012345678 |   us-west-2a        |
|   subnet-234567890 |   us-west-2b        |
|   subnet-345678901 |   us-west-2c        |
----------------------------------------
```

EKS 自动模式集群的 IAM 角色

集群 IAM 角色

EKS 自动模式需要集群 IAM 角色才能在您的 Amazon 账户中执行操作，例如预置新的 EC2 实例。您必须创建此角色才能向 EKS 授予必要的权限。Amazon 建议将以下 Amazon 托管式策略附加到集群 IAM 角色：

- AmazonEKSComputePolicy
- AmazonEKSBlockStoragePolicy
- AmazonEKSLoadBalancingPolicy

- AmazonEKSNetworkingPolicy
- AmazonEKSClusterPolicy：

节点 IAM 角色

创建 EKS 自动模式集群时，需要指定一个节点 IAM 角色。EKS 自动模式创建节点来处理待处理的工作负载时，将为每个新的 EC2 实例节点分配此节点 IAM 角色。此角色允许节点与 EKS 通信，但节点上运行的工作负载通常无法访问此角色。

要向在节点上运行的工作负载授予权限，请使用 EKS 容器组身份。有关更多信息，请参阅 the section called “容器组身份”。

您必须创建此角色并附加以下 Amazon 托管式策略：

- AmazonEKSWorkerNodeMinimalPolicy
- AmazonEC2ContainerRegistryPullOnly

服务相关角色

EKS 自动模式还需要一个将由 Amazon 自动创建和配置的服务相关角色。有关更多信息，请参阅 AWSServiceRoleForAmazonEKS。

创建 EKS 自动模式集群 IAM 角色

第 1 步：创建信任策略

创建信任策略以允许 Amazon EKS 服务代入该角色。将策略另存为 `trust-policy.json`：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
```

```
  ]
}
```

第 2 步：创建 IAM 角色

使用信任策略创建集群 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoClusterRole \
    --assume-role-policy-document file://trust-policy.json
```

第 3 步：记下角色 ARN

检索并保存新角色的 ARN，以便在后续步骤中使用：

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

第 4 步：附加必需的策略

将以下 Amazon 托管式策略附加到集群 IAM 角色以授予必要的权限：

AmazonEKSClusterPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSComputePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy：

```
aws iam attach-role-policy \
```

```
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSNetworkingPolicy
```

创建 EKS 自动模式 节点 IAM 角色

第 1 步：创建信任策略

创建信任策略以允许 Amazon EKS 服务代入该角色。将策略另存为 `node-trust-policy.json`：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

第 2 步：创建节点 IAM 角色

使用上一步中的 node-trust-policy.json 文件来定义哪些实体可以代入该角色。运行以下命令以创建节点 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

第 3 步：记下角色 ARN

创建角色后，检索并保存节点 IAM 角色的 ARN。在后续步骤中，您将需要此 ARN。使用以下命令来获取 ARN：

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

第 4 步：附加必需的策略

将以下 Amazon 托管式策略附加到节点 IAM 角色，以提供必要的权限：

AmazonEKSWorkerNodeMinimalPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

创建 EKS 自动模式集群

概述

要使用 Amazon CLI 创建 EKS 自动模式集群，您需要以下参数：

- `cluster-name`：集群的名称。
- `k8s-version`：Kubernetes 版本（例如 1.31）。
- `subnet-ids`：之前步骤中确定的子网 ID。
- `cluster-role-arn`：集群 IAM 角色的 ARN。
- `node-role-arn`：节点 IAM 角色的 ARN。

默认集群配置

在创建集群之前，请检查以下默认值和功能：

- `nodePools`：EKS 自动模式包含通用节点池和系统默认节点池。了解有关节点池的更多信息。

注意：EKS 自动模式下的节点池与 Amazon EKS 托管式节点组不同，但可以共存于同一个集群中。

- computeConfig.enabled：自动执行例行计算任务，例如创建和删除 EC2 实例。
- kubernetesNetworkConfig.elasticLoadBalancing.enabled：自动执行负载均衡任务，包括创建和删除弹性负载均衡器。
- storageConfig.blockStorage.enabled：自动执行存储任务，例如创建和删除 Amazon EBS 卷。
- accessConfig.authenticationMode：需要 EKS 访问条目。了解有关 EKS 身份验证模式的更多信息。

运行命令

使用下面的命令创建集群：

```
aws eks create-cluster \
  --region ${AWS_REGION} \
  --cli-input-json \
  "{
      \"name\": \"${CLUSTER_NAME}\",
      \"version\": \"${K8S_VERSION}\",
      \"roleArn\": \"${CLUSTER_ROLE_ARN}\",
      \"resourcesVpcConfig\": {
        \"subnetIds\": ${SUBNETS_JSON},
        \"endpointPublicAccess\": true,
        \"endpointPrivateAccess\": true
      },
      \"computeConfig\": {
        \"enabled\": true,
        \"nodeRoleArn\":\"${NODE_ROLE_ARN}\",
        \"nodePools\": [\"general-purpose\", \"system\"]
      },
      \"kubernetesNetworkConfig\": {
        \"elasticLoadBalancing\": {
          \"enabled\": true
        }
      },
      \"storageConfig\": {
        \"blockStorage\": {
          \"enabled\": true
        }
      },
      \"accessConfig\": {
        \"authenticationMode\": \"API\"
```

```
      }
    }
```

检查集群状态

第 1 步：验证集群创建

运行以下命令来检查集群的状态。创建集群通常需要大约 15 分钟时间：

```
aws eks describe-cluster --name "${CLUSTER_NAME}" --output json
```

第 2 步：更新 kubeconfig

集群准备就绪后，更新本地 kubeconfig 文件以便 `kubectl` 可以与集群通信。此配置使用 Amazon CLI 进行身份验证。

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

第 3 步：验证节点池

使用以下命令列出集群中的节点池：

```
kubectl get nodepools
```

后续步骤

- 了解如何将示例工作负载部署到新的 EKS 自动模式集群。

使用 Amazon Web Services 管理控制台创建 EKS 自动模式集群

与其他选项相比，在 Amazon Web Services 管理控制台中创建 EKS 自动模式集群所需的配置更少。EKS 与 Amazon IAM 和 VPC 联网服务集成，可帮助您创建与 EKS 集群关联的资源。

您可以通过两种方式在控制台中创建集群：

- 快速配置（使用 EKS 自动模式）
- 自定义配置

在本主题中，您将了解如何使用快速配置选项创建 EKS 自动模式集群。

使用快速配置选项创建 EKS 自动模式集群

您必须以管理 Amazon 资源所需的足够权限登录 Amazon Web Services 管理控制台，包括 EC2 实例、EC2 联网、EKS 集群和 IAM 角色等。

1. 导航到 EKS 控制台
2. 单击创建集群
3. 确认已选中快速配置选项
4. 确定以下值，或使用测试集群的默认值。
   - 集群名称
   - Kubernetes 版本
5. 选择集群 IAM 角色。如果您是首次创建 EKS 自动模式集群，请使用创建推荐角色选项。
   - 您也可以将 Amazon 账户中的单个集群 IAM 角色重复用于所有 EKS 自动模式集群。
   - 集群 IAM 角色包含 EKS 自动模式管理资源（包括 EC2 实例、EBS 卷和 EC2 负载均衡器）所需的权限。
   - 创建推荐角色选项会使用推荐值预填充所有字段。选择下一步，然后选择创建。此角色将使用建议的 `AmazonEKSAutoClusterRole` 名称。
   - 如果您最近创建了新角色，请使用刷新图标重新加载角色选择下拉列表。
6. 选择节点 IAM 角色。如果您是首次创建 EKS 自动模式集群，请使用创建推荐角色选项。
   - 您也可以将 Amazon 账户中的单个节点 IAM 角色重复用于所有 EKS 自动模式集群。
   - 此节点 IAM 角色包含自动模式节点连接到集群所需的权限。此节点 IAM 角色必须包含检索容器的 ECR 映像的权限。
   - 创建推荐角色选项会使用推荐值预填充所有字段。选择下一步，然后选择创建。此角色将使用建议的 `AmazonEKSAutoNodeRole` 名称。
   - 如果您最近创建了新角色，请使用刷新图标重新加载角色选择下拉列表。
7. 为您的 EKS 自动模式集群选择 VPC。选择创建 VPC 为 EKS 创建新的 VPC，也可以选择您之前为 EKS 创建的 VPC。
   - 如果您使用 VPC 控制台创建新的 VPC，Amazon 建议您在每个可用区至少创建一个 NAT 网关。否则，您可以使用所有其他默认值。
   - 有关 IPv6 集群要求的更多信息和详情，请参阅the section called “创建 VPC”。
8. （可选）EKS 自动模式会自动填充所选 VPC 的私有子网。您可以移除不需要的子网。
   - EKS 会按照最佳实践自动选择 VPC 中的私有子网。您可以从 VPC 中选择额外的子网，例如公有子网。

9. （可选）选择查看快速配置默认值以查看新集群的所有配置值。该表提示某些值在集群创建后不可编辑。

10选择 Create cluster（创建集群）。请注意，完成集群创建可能需要 15 分钟时间。

后续步骤

- 了解如何将示例工作负载部署到新的 EKS 自动模式集群

在现有 EKS 集群上启用 EKS 自动模式

您可以在现有 EKS 集群上启用 EKS 自动模式。

Amazon 支持下列迁移：

- 从 Karpenter 迁移到 EKS 自动模式节点。有关更多信息，请参阅 the section called “从 Karpenter 迁移”。
- 从 EKS 托管式节点组迁移到 EKS 自动模式节点。有关更多信息，请参阅 the section called “从 MNG 迁移”。
- 从 EKS Fargate 迁移到 EKS 自动模式节点。有关更多信息，请参阅 the section called “从 Fargate 迁移”。

Amazon 不支持下列迁移：

- 将卷从 EBS CSI 控制器（使用 Amazon EKS 附加组件）迁移到 EKS 自动模式 EBS CIS 控制器（由 EKS 自动模式管理）。由不同配置器创建的 PVC 无法互相挂载，因为它们使用的是两个不同的 Kubernetes 卷预调配程序。
  - `eks-auto-mode-ebs-migration-tool`（Amazon Labs 项目）支持在标准 EBS CSI StorageClass (`ebs.csi.aws.com`) 与 EKS 自动模式 EBS CSI StorageClass (`ebs.csi.eks.amazonaws.com`) 之间迁移。请注意，迁移需要删除并重新创建现有的 PersistentVolumeClaim/PersistentVolume 资源，因此在进行迁移之前，必须在非生产环境中进行验证。
- 将负载均衡器从 Amazon 负载均衡器控制器迁移到 EKS 自动模式

  您可以在 Amazon EKS 自动模式集群上安装 Amazon 负载均衡器控制器。使用 `IngressClass` 或 `loadBalancerClass` 选项将服务和 Ingress 资源关联到 负载均衡器控制器或 EKS 自动模式。
- 使用备用 CNI 或其他不受支持的网络配置迁移 EKS 集群

迁移参考

按照以下迁移参考将 Kubernetes 资源配置为由自主管理型控制器或 EKS 自动模式所有。

能力	资源	字段	自主管理型	EKS 自动模式
数据块存储	StorageClass	provisioner	ebs.csi.aws.com	ebs.csi.eks.amazonaws.com
负载均衡	Service	loadBalancerClass	service.k8s.aws/nlb	eks.amazonaws.com/nlb
负载均衡	IngressClass	controller	ingress.k8s.aws/alb	eks.amazonaws.com/alb
负载均衡	IngressClassParams	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
负载均衡	TargetGroupBinding	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
计算	NodeClass	apiVersion	karpenter.sh/v1	eks.amazonaws.com/v1

迁移 EBS 卷

将工作负载迁移到 EKS 自动模式时，由于 CSI 驱动程序预调配程序不同，您需要处理 EBS 卷迁移问题：

- EKS 自动模式预调配程序：ebs.csi.eks.amazonaws.com
- 开源 EBS CSI 预调配程序：ebs.csi.aws.com

请按照以下步骤迁移持久性卷：

1. 修改卷保留策略：将现有平台版本（PV）由 `persistentVolumeReclaimPolicy` 更改为 `Retain`，确保底层 EBS 卷不会删除。
2. 从 Kubernetes 中删除 PV：删除旧的 PV 资源，但保留实际的 EBS 卷。
3. 通过静态预置创建新 PV：创建一个引用相同 EBS 卷但与目标 CSI 驱动程序结合使用的新 PV。
4. 绑定到新 PVC：使用 `volumeName` 字段创建一个专门引用 PV 的新 PVC。

注意事项

- 确保在开始迁移之前应用程序已停止。
- 在开始迁移过程之前，备份数据。
- 需要对每个持久性卷执行此过程。
- 必须更新工作负载才能使用新 PVC。

迁移负载均衡器

不能将现有负载均衡器从自主管理型 Amazon 负载均衡器控制器直接转移到 EKS 自动模式。相反，您必须实施蓝绿部署策略。这涉及维护现有的负载均衡器配置，同时在托管式控制器下创建新的负载均衡器。

为了尽可能减少服务中断，我们建议采用基于 DNS 的流量转移方法。首先使用 EKS 自动模式创建新的负载均衡器，同时保持现有配置正常运行。然后使用 DNS 路由（例如 Route 53）将流量从旧负载均衡器逐渐转移到新负载均衡器。成功迁移流量并验证新配置后，您可以停用旧负载均衡器和自主管理型控制器。

在现有集群上启用 EKS 自动模式

本主题介绍如何在现有的 Amazon EKS 集群上启用 Amazon EKS 自动模式。要在现有集群上启用自动模式，需要更新 IAM 权限并配置核心 EKS 自动模式设置。启用自动模式后，您可以开始迁移现有的计算工作负载，以利用自动模式的简化操作和自动化基础设施管理等优势。

> **Important**
>
> 启用 EKS 自动模式之前，请确认您已安装某些 Amazon EKS 附加组件的最低所需版本。有关更多信息，请参阅 the section called “必需的附加组件版本”。

在开始操作之前，请确保您拥有 Amazon EKS 集群的管理员访问权限以及修改 IAM 角色的权限。本主题中的步骤将演示使用 Amazon Web Services 管理控制台或 Amazon CLI 启用自动模式的过程。

Amazon Web Services 管理控制台

您必须登录 Amazon 控制台并拥有管理 IAM、EKS 和 EC2 资源的权限。

> **Note**
>
> 创建 EKS 集群后，将无法更改其集群 IAM 角色。EKS 自动模式需要此角色拥有额外的权限。您必须为当前角色附加额外的策略。

更新集群 IAM 角色

1. 在 Amazon Web Services 管理控制台中打开集群概述页面。
2. 在集群 IAM 角色 ARN 下，选择在 IAM 中查看。
3. 从添加权限下拉菜单中，选择附加策略。
4. 使用搜索框查找并选中以下策略：
   - `AmazonEKSComputePolicy`
   - `AmazonEKSBlockStoragePolicy`
   - `AmazonEKSLoadBalancingPolicy`
   - `AmazonEKSNetworkingPolicy`
   - `AmazonEKSClusterPolicy`
5. 选择添加权限
6. 从信任关系选项卡中选择编辑信任策略
7. 插入以下集群 IAM 角色信任策略，然后选择更新策略

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
```

```
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

启用 EKS 自动模式

1. 在 Amazon Web Services 管理控制台中打开集群概述页面。
2. 在 EKS 自动模式下，选择管理
3. 将 EKS 自动模式切换为开启。
4. 从 EKS 节点池下拉列表中，选择要创建的默认节点池。
   - 详细了解 EKS 自动模式中的节点池。有关更多信息，请参阅 the section called “创建节点池”。
5. 如果您之前已为此 Amazon 账户创建了 EKS 自动模式节点 IAM 角色，请在节点 IAM 角色下拉列表中选中该角色。如果您之前尚未创建该角色，请选择创建建议角色，然后按照步骤操作。

Amazon CLI

先决条件

- 现有 EKS 集群的集群 IAM 角色必须包含足够的 EKS 自动模式权限，例如以下策略：
  - `AmazonEKSComputePolicy`
  - `AmazonEKSBlockStoragePolicy`
  - `AmazonEKSLoadBalancingPolicy`
  - `AmazonEKSNetworkingPolicy`
  - `AmazonEKSClusterPolicy`
- 集群 IAM 角色必须更新信任策略，其中应包括 `sts:TagSession` 操作。有关创建集群 IAM 角色的更多信息，请参阅the section called “ Amazon CLI”。
- 已安装 aws CLI，已登录，并且版本充分满足要求。您必须拥有管理 IAM、EKS 和 EC2 资源的权限。有关更多信息，请参阅 设置。

过程

使用以下命令在现有集群上启用 EKS 自动模式。

> **Note**
>
> 计算、块存储和负载均衡功能必须在同一请求中全部启用或禁用。

```
aws eks update-cluster-config \
 --name $CLUSTER_NAME \
 --compute-config enabled=true \
 --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": true}}' \
 --storage-config '{"blockStorage":{"enabled": true}}'
```

必需的附加组件版本

如果您计划在现有集群上启用 EKS 自动模式，则可能需要更新某些附加组件。请注意：

- 这仅适用于将现有集群迁移到 EKS 自动模式。
- 创建启用 EKS 自动模式的新集群不需要进行这些更新。

如果您安装了以下任何附加组件，请确保这些附加组件至少达到指定的最低版本：

附加组件名称	最低所需版本
适用于 Kubernetes 的 Amazon VPC CNI 插件	v1.19.0-eksbuild.1
Kube-proxy	• v1.26.15-eksbuild.19 • v1.27.16-eksbuild.14 • v1.28.15-eksbuild.4 • v1.29.10-eksbuild.3 • v1.30.6-eksbuild.3 • v1.31.2-eksbuild.3
Amazon EBS CSI 驱动程序	v1.37.0-eksbuild.1
CSI 快照控制器	v8.1.0-eksbuild.2
EKS 容器组身份代理	v1.3.4-eksbuild.1

有关更多信息，请参阅 the section called “更新附加组件”。

后续步骤

- 要迁移“管理节点组”工作负载，请参阅the section called “从 MNG 迁移”。
- 要从自主管理型 Karpenter 迁移，请参阅the section called “从 Karpenter 迁移”。

使用 kubectl 从 Karpenter 迁移到 EKS 自动模式

本主题将演示使用 kubectl 将工作负载从 Karpenter 迁移到 Amazon EKS 自动模式的过程。迁移可以逐步执行，从而让您按照自己的节奏移动工作负载，同时在整个迁移期间保持集群稳定性和应用程序可用性。

借助下面概述的分步方法，您可以在迁移期间并行运行 Karpenter 和 EKS 自动模式。借助这种双重运行策略，您可以在完全停用 Karpenter 之前验证 EKS 自动模式下的工作负载行为，从而帮助保证平稳过渡。您可以逐一或分组迁移应用程序，从而灵活地适应自己的具体运行要求和风险承受能力。

先决条件

开始迁移之前，请确保您满足以下条件：

- 您的集群已安装 Karpenter 版本 1.1 或更高版本。有关更多信息，请参阅 Karpenter 文档中的 Upgrading to 1.1.0+。
- 已安装 `kubectl` 并连接到集群。有关更多信息，请参阅 设置。

本主题假设您熟悉 Karpenter 和节点池。有关更多信息，请参阅 Karpenter 文档。

第 1 步：在集群上启用 EKS 自动模式

使用 Amazon CLI 或管理控制台在现有集群上启用 EKS 自动模式。有关更多信息，请参阅 the section called “在集群上启用”。

> **Note**
>
> 启用 EKS 自动模式时，不要在迁移过程的这一阶段启用 `general purpose` 节点池。此节点池不是选择性的。
> 有关更多信息，请参阅 the section called “检查内置节点池”。

第 2 步：创建一个受污染的 EKS 自动模式节点池

为 EKS 自动模式创建一个带有污点的新节点池。这样可以确保不会在新的 EKS 自动模式节点上自动调度现有的容器组。此节点池使用 EKS 自动模式中内置的 `default` NodeClass。有关更多信息，请参阅 the section called “创建节点类”。

带有污点的节点池示例：

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      taints:
        - key: "eks-auto-mode"
          effect: "NoSchedule"
```

更新节点池要求以使其与您要迁移的 Karpenter 配置一致。您至少需要一项要求。

第 3 步：更新要迁移的工作负载

识别并更新要迁移到 EKS 自动模式的工作负载。为这些工作负载添加容忍度和节点选择器：

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      tolerations:
      - key: "eks-auto-mode"
        effect: "NoSchedule"
      nodeSelector:
```

```
        eks.amazonaws.com/compute-type: auto
```

实施此更改后，将可以在新的 EKS 自动模式节点上调度工作负载。

EKS 自动模式使用的标签与 Karpenter 不同。与 EC2 托管式实例相关的标签以 `eks.amazonaws.com` 开头。有关更多信息，请参阅 the section called “创建节点池”。

第 4 步：逐步迁移工作负载

对要迁移的每个工作负载重复第 3 步。这样就可以根据自己的要求和风险承受能力逐一或分组移动工作负载。

第 5 步：移除原始 Karpenter 节点池

迁移完所有工作负载后，您可以移除原始 Karpenter 节点池：

```
kubectl delete nodepool <original-nodepool-name>
```

第 6 步：从 EKS 自动模式节点池中移除污点（可选）

要让 EKS 自动模式成为新工作负载的默认模式，您可以从 EKS 自动模式节点池中移除污点：

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      # Remove the taints section
```

第 7 步：从工作负载中移除节点选择器（可选）

如果您已从 EKS 自动模式节点池中移除污点，则可以选择从工作负载中移除节点选择器，因为 EKS 自动模式现已是默认模式：

```
apiVersion: apps/v1
```

```
kind: Deployment
spec:
  template:
    spec:
      # Remove the nodeSelector section
      tolerations:
      - key: "eks-auto-mode"
        effect: "NoSchedule"
```

第 8 步：从集群中卸载 Karpenter

移除 Karpenter 的步骤取决于最初的安装方式。有关更多信息，请参阅 Karpenter install instructions。

从 EKS 托管式节点组迁移到 EKS 自动模式

在将 Amazon EKS 集群转换为使用 EKS 自动模式时，可以使用 eksctl CLI 工具将现有工作负载从托管节点组（MNG）顺利迁移。此过程可确保应用程序的持续可用，同时 EKS 自动模式也可优化计算资源。迁移过程可以尽可能减少对正在运行的应用程序的干扰。

本主题将演示安全地清空现有托管式节点组中的容器组，让 EKS 自动模式在新预置的实例上重新调度这些容器组的步骤。按照此过程，您既可以利用 EKS 自动模式的智能工作负载整合优势，同时还可以在整个迁移过程中保持应用程序的可用性。

先决条件

- 启用 EKS 自动模式的集群
- 已安装 `eksctl` CLI 并连接到集群。有关更多信息，请参阅 设置。
- 集群上未安装 Karpenter。

过程

使用以下 `eksctl` CLI 命令启动清空现有托管式节点组实例中的容器组。EKS 自动模式将创建新节点来支持移位后容器组。

```
eksctl delete nodegroup --cluster=<clusterName> --name=<nodegroupName>
```

您需要为集群中的每个托管式节点组运行此命令。

有关此命令的更多信息，请参阅 eksctl 文档中的 Deleting and draining nodegroups。

从 EKS Fargate 迁移到 EKS 自动模式

本主题将演示使用 kubectl 将工作负载从 EKS Fargate 迁移到 Amazon EKS 自动模式的过程。迁移可以逐步执行，从而让您按照自己的节奏移动工作负载，同时在整个迁移期间保持集群稳定性和应用程序可用性。

借助下面概述的分步方法，您可以在迁移期间并行运行 EKS Fargate 和 EKS 自动模式。借助这种双重运行策略，您可以在完全停用 EKS Fargate 之前验证 EKS 自动模式下的工作负载行为，从而帮助保证平稳过渡。您可以逐一或分组迁移应用程序，从而灵活地适应自己的具体运行要求和风险承受能力。

将 Amazon EKS 自动模式与具有 Amazon Fargate 的 EKS 进行比较

对于希望运行 EKS 的客户来说，尽管具有 Amazon Fargate 的 Amazon EKS 仍是一种选择，但 Amazon EKS 自动模式是未来推荐的首选方法。EKS 自动模式完全符合 Kubernetes 要求，支持所有上游 Kubernetes 原语和平台工具，例如 Fargate 无法支持的 Istio。EKS 自动模式还完全支持所有 EC2 运行时购买选项，包括 GPU 和竞价型实例，有助于客户利用议定的 EC2 折扣以及其他节省机制。但是，将 EKS 与 Fargate 结合使用时，这些功能不可用。

此外，EKS 自动模式允许客户实现与 Fargate 相同的隔离模型，通过标准的 Kubernetes 调度功能确保每个 EC2 实例都运行一个应用程序容器。借助 Amazon EKS 自动模式，客户可以充分享受在 Amazon 上运行 Kubernetes 的全部优势。Kubernetes 是一个完全符合 Kubernetes 要求的平台，可以灵活利用所有 EC2 和购买选项，同时保留 Fargate 提供的易用性以及从基础设施管理中抽象化的能力。

先决条件

开始迁移之前，请确保您满足以下条件

- 设置带有 Fargate 的集群。有关更多信息，请参阅 the section called “开始使用”。
- 已安装 kubectl 并将其连接到集群。有关更多信息，请参阅 设置。

步骤 1：检查 Fargate 集群

1. 检查带有 Fargate 的 EKS 集群是否正在运行：

```
kubectl get node
```

```
NAME STATUS ROLES AGE VERSION
fargate-ip-192-168-92-52.ec2.internal Ready <none> 25m v1.30.8-eks-2d5f260
```

```
fargate-ip-192-168-98-196.ec2.internal Ready <none> 24m v1.30.8-eks-2d5f260
```

2. 检查正在运行的容器组（pod）：

```
kubectl get pod -A
```

```
NAMESPACE NAME READY STATUS RESTARTS AGE
kube-system coredns-6659cb98f6-gxpjz 1/1 Running 0 26m
kube-system coredns-6659cb98f6-gzzsx 1/1 Running 0 26m
```

3. 在名为 deployment_fargate.yaml 的文件中创建部署：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        eks.amazonaws.com/compute-type: fargate
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

4. 应用部署：

```
kubectl apply -f deployment_fargate.yaml
```

```
deployment.apps/nginx-deployment created
```

5. 检查容器组（pod）和部署：

```
kubectl get pod,deploy
```

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-5c7479459b-6trtm   1/1     Running   0          61s
pod/nginx-deployment-5c7479459b-g8ssb   1/1     Running   0          61s
pod/nginx-deployment-5c7479459b-mq4mf   1/1     Running   0          61s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment    3/3     3            3           61s
```

6. 检查节点：

```
kubectl get node -owide
```

```
NAME                                    STATUS  ROLES  AGE VERSION
   INTERNAL-IP     EXTERNAL-IP OS-IMAGE       KERNEL-VERSION
 CONTAINER-RUNTIME
fargate-ip-192-168-111-43.ec2.internal  Ready   <none> 31s v1.30.8-eks-2d5f260
 192.168.111.43  <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
fargate-ip-192-168-117-130.ec2.internal Ready   <none> 36s v1.30.8-eks-2d5f260
 192.168.117.130 <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
fargate-ip-192-168-74-140.ec2.internal  Ready   <none> 36s v1.30.8-eks-2d5f260
 192.168.74.140  <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
```

步骤 2：在集群上启用 EKS 自动模式

1. 使用 Amazon CLI 或管理控制台在现有集群上启用 EKS 自动模式。有关更多信息，请参阅 the section called “在集群上启用”。
2. 检查节点池：

```
kubectl get nodepool
```

```
NAME              NODECLASS   NODES   READY   AGE
general-purpose   default     1       True    6m58s
```

```
system            default     0       True    3d14h
```

第 3 步：更新要迁移的工作负载

识别并更新要迁移到 EKS 自动模式的工作负载。

要将工作负载从 Fargate 迁移到 EKS 自动模式，请应用注释 `eks.amazonaws.com/compute-type: ec2`。这可确保工作负载不会被 Fargate 调度（即使存在 Fargate 配置文件），而是由 EKS 自动模式节点池接管。有关更多信息，请参阅 the section called “创建节点池”。

1. 修改部署（例如 `deployment_fargate.yaml` 文件）以将计算类型更改为 `ec2`：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        eks.amazonaws.com/compute-type: ec2
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

2. 应用部署。实施此更改后，即可在新的 EKS 自动模式节点上调度工作负载：

```
kubectl apply -f deployment_fargate.yaml
```

3. 检查部署是否正在 EKS 自动模式集群中运行：

```
kubectl get pod -o wide
```

```
NAME                                READY   STATUS    RESTARTS   AGE     IP
    NODE                   NOMINATED NODE   READINESS GATES
nginx-deployment-97967b68d-ffxxh   1/1     Running   0          3m31s
 192.168.43.240   i-0845aafcb51630ffb   <none>           <none>
nginx-deployment-97967b68d-mbcgj   1/1     Running   0          2m37s
 192.168.43.241   i-0845aafcb51630ffb   <none>           <none>
nginx-deployment-97967b68d-qpd8x   1/1     Running   0          2m35s
 192.168.43.242   i-0845aafcb51630ffb   <none>           <none>
```

4. 确认 EKS 自动模式托管式节点中没有正在运行的 Fargate 节点和正在运行的部署：

```
kubectl get node -owide
```

```
NAME                 STATUS ROLES  AGE   VERSION              INTERNAL-IP     EXTERNAL-
IP OS-IMAGE                                          KERNEL-VERSION CONTAINER-RUNTIME
i-0845aafcb51630ffb Ready  <none> 3m30s v1.30.8-eks-3c20087 192.168.41.125
 3.81.118.95 Bottlerocket (EKS Auto) 2025.3.14 (aws-k8s-1.30) 6.1.129
 containerd://1.7.25+bottlerocket
```

第 4 步：逐步迁移工作负载

对要迁移的每个工作负载重复第 3 步。这样就可以根据自己的要求和风险承受能力逐一或分组移动工作负载。

步骤 5：删除原始 Fargate 配置文件

迁移完所有工作负载后，可以删除原始 `fargate` 配置文件。将 *<fargate profile name>* 替换为 Fargate 配置文件的名称：

```
aws eks delete-fargate-profile --cluster-name eks-fargate-demo-cluster --fargate-
profile-name <fargate profile name>
```

步骤 6：缩减 CoreDNS

由于 EKS 自动模式可以处理 CoreDNS，因此可将 `coredns` 部署缩减到 0：

```
kubectl scale deployment coredns -n kube-system —-replicas=0
```

在 EKS 自动模式集群中运行示例工作负载

本章提供的示例演示了如何将不同类型的工作负载部署到在自动模式下运行的 Amazon EKS 集群。这些示例演示了关键的工作负载模式，包括示例应用程序、启用负载均衡的 Web 应用程序、使用持久性存储的有状态工作负载以及具有特定节点放置要求的工作负载。每个示例都包含完整的清单和详细分步部署说明，您可以将其用作自己应用程序的模板。

在开始使用示例之前，请确保您有一个在自动模式下运行的 EKS 集群，并且已经安装了 Amazon CLI 和 kubectl。有关更多信息，请参阅 设置。这些示例假设您基本熟悉 Kubernetes 的概念和 kubectl 命令。

您可以使用这些基于应用场景的示例，在 EKS 自动模式集群中运行工作负载。

the section called “部署 inflate 工作负载”

演示如何使用 `kubectl` 命令将示例工作负载部署到 EKS 自动模式集群。

the section called “部署负载均衡器”

演示如何在 Amazon EKS 上部署 2048 游戏的容器化版本。

the section called “部署有状态工作负载”

演示如何将示例有状态应用程序部署到 EKS 自动模式集群。

the section called “部署加速型工作负载”

演示如何将硬件加速的工作负载部署到由 EKS 自动模式管理的节点。

the section called “控制部署”

演示如何使用注释来控制是否将工作负载部署到由 EKS 自动模式管理的节点。

将示例 inflate 工作负载部署到 Amazon EKS 自动模式集群

在本教程中，您将了解如何将示例工作负载部署到 EKS 自动模式集群，并观察其如何自动预置所需的计算资源。您将使用 `kubectl` 命令来观察集群行为，并亲眼见证自动模式如何在 Amazon 上简化 Kubernetes 的操作。完成本教程后，您将了解 EKS 自动模式如何通过自动管理底层计算资源来响应工作负载部署，无需手动配置节点组。

先决条件

- Amazon EKS 自动模式集群。记下集群的名称和 Amazon 区域。
- 一个具有管理联网、计算和 EKS 资源的足够权限的 IAM 主体，例如用户或角色。
  - 有关更多信息，请参阅《IAM 用户指南》中的创建角色并附加策略（控制台）。
- 已使用某个 IAM 身份 安装并配置了 aws CLI。
- 已安装 kubectl CLI 并连接到集群。
  - 有关更多信息，请参阅 设置。

第 1 步：检查现有的计算资源（可选）

首先，使用 kubectl 列出集群上的节点池。

```
kubectl get nodepools
```

示例输出：

```
general-purpose
```

在本教程中，我们将部署一个配置为使用 general-purpose 节点池的工作负载。此节点池已内置在 EKS 自动模式中，包括微服务和 Web 应用程序等一般工作负载的合理默认值。您可以创建自己的节点池。有关更多信息，请参阅 the section called “创建节点池”。

然后，使用 kubectl 列出连接到集群的节点。

```
kubectl get nodes
```

如果您刚刚创建了 EKS 自动模式集群，则将没有任何节点。

在本教程中，您将部署一个示例工作负载。如果没有节点，或者现有节点不能满足工作负载的要求，EKS 自动模式将预置一个新节点。

第 2 步：将示例应用程序部署到集群

检查以下 Kubernetes 部署并将其另存为 inflate.yaml

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: inflate
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate
  template:
    metadata:
      labels:
        app: inflate
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      securityContext:
        runAsUser: 1000
        runAsGroup: 3000
        fsGroup: 2000
      containers:
        - name: inflate
          image: public.ecr.aws/eks-distro/kubernetes/pause:3.7
          resources:
            requests:
              cpu: 1
          securityContext:
            allowPrivilegeEscalation: false
```

请注意，eks.amazonaws.com/compute-type: auto 选择器要求将工作负载部署到 Amazon EKS 自动模式节点上。

将部署应用到集群。

```
kubectl apply -f inflate.yaml
```

第 3 步：观察 Kubernetes 事件

使用以下命令观察 Kubernetes 事件，包括创建新节点。使用 ctrl+c 停止观察事件。

```
kubectl get events -w --sort-by '.lastTimestamp'
```

再次使用 kubectl 列出连接到集群的节点。记下新创建的节点。

```
kubectl get nodes
```

第 4 步：在 Amazon 控制台中查看节点和实例

您可以在 EKS 控制台中查看 EKS 自动模式节点，以及在 EC2 控制台中查看关联的 EC2 实例。

由 EKS 自动模式部署的 EC2 实例存在限制。您不能在 EKS 自动模式节点上运行任意命令。

第 5 步：删除部署

使用 kubectl 删除示例部署

```
kubectl delete -f inflate.yaml
```

如果集群中没有部署其他工作负载，则由 EKS 自动模式创建的节点将为空。

在默认配置中，EKS 自动模式会检测已空三十秒钟的节点，并将其终止。

使用 kubectl 或 EC2 控制台确认关联的实例已被删除。

将示例负载均衡器工作负载部署到 EKS 自动模式

本指南将演示在 Amazon EKS 上部署 2048 游戏的容器化版本，包括负载均衡和互联网访问功能。

先决条件

- 一个 EKS 自动模式集群
- 配置好 kubectl 以与集群进行交互
- 用于创建 ALB 资源的适当 IAM 权限

第 1 步：创建命名空间

首先，为 2048 游戏应用程序创建一个专用的命名空间。

创建一个名为 01-namespace.yaml 的文件：

```
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: game-2048
```

应用命名空间配置：

```
kubectl apply -f 01-namespace.yaml
```

第 2 步：部署应用程序

该应用程序会运行 2048 游戏容器的多个副本。

创建一个名为 `02-deployment.yaml` 的文件：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 5
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          name: app-2048
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: "0.5"
```

> **Note**
>
> 如果加载映像 `public.ecr.aws/l6m2t8p7/docker-2048:latest` 时收到错误消息，请确认节点 IAM 角色是否具有足够的权限从 ECR 提取映像。有关更多信息，请参阅 the section

called “节点 IAM 角色”。此外，示例中的 docker-2048 映像为 x86_64 映像，不会在其他架构上运行。

关键组件：

- 部署 5 个应用程序副本
- 使用一个公有 ECR 映像
- 每个容器组（pod）请求 0.5 个 CPU 核心
- 为 HTTP 流量公开端口 80

应用部署：

```
kubectl apply -f 02-deployment.yaml
```

第 3 步：创建服务

该服务将该部署向集群网络公开。

创建一个名为 03-service.yaml 的文件：

```
apiVersion: v1
kind: Service
metadata:
  namespace: game-2048
  name: service-2048
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    app.kubernetes.io/name: app-2048
```

关键组件：

- 创建一个节点端口服务
- 将端口 80 映射到容器的端口 80

- 使用标签选择器查找容器组

应用服务：

```
kubectl apply -f 03-service.yaml
```

第 4 步：配置负载均衡

您将设置一个 Ingress，以将该应用程序向互联网公开。

首先，创建 IngressClass。创建一个名为 04-ingressclass.yaml 的文件：

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/name: LoadBalancerController
  name: alb
spec:
  controller: eks.amazonaws.com/alb
```

> **Note**
>
> EKS 自动模式需要子网标签来标识公有子网和私有子网。
> 如果使用 eksctl 创建集群，则已包含这些标签。
> 了解如何the section called “为子网添加标签”。

然后创建该 Ingress 资源。创建一个名为 05-ingress.yaml 的文件：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: game-2048
  name: ingress-2048
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
```

```
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: service-2048
                port:
                  number: 80
```

关键组件：

- 创建一个面向互联网的 ALB
- 使用适用于直接容器组路由的 IP 目标类型
- 将所有流量 (/) 路由到游戏服务

应用 Ingress 配置：

```
kubectl apply -f 04-ingressclass.yaml
kubectl apply -f 05-ingress.yaml
```

第 5 步：验证部署

1. 检查所有容器组是否都在运行：

```
kubectl get pods -n game-2048
```

2. 确认服务是否已创建：

```
kubectl get svc -n game-2048
```

3. 获取 ALB 端点：

```
kubectl get ingress -n game-2048
```

Ingress 输出中的 ADDRESS 字段将显示您的 ALB 端点。等待 2-3 分钟，让 ALB 完成预置并注册所有目标。

第 6 步：访问游戏

打开 Web 浏览器并浏览到之前步骤中的 ALB 端点 URL。您应该会看到 2048 游戏界面。

步骤 7：清除

移除本教程中创建的所有资源：

```
kubectl delete namespace game-2048
```

这将删除命名空间中的所有资源，包括部署、服务和 Ingress 资源。

幕后发生了什么

1. 此部署创建了 5 个运行 2048 游戏的容器组
2. 该服务为这些容器组提供稳定的网络访问
3. EKS 自动模式：
   - 在 Amazon 中创建一个应用程序负载均衡器
   - 为容器组配置目标组
   - 设置路由规则以将流量定向到服务

故障排除

如果游戏无法加载：

- 确保所有容器组都在运行：`kubectl get pods -n game-2048`
- 检查 Ingress 状态：`kubectl describe ingress -n game-2048`
- 验证 ALB 运行状况检查：在 Amazon 控制台中检查目标组的运行状况

将示例有状态工作负载部署到 EKS 自动模式

本教程将演示向 EKS 自动模式集群部署示例有状态应用程序的过程。该应用程序会将时间戳写入持久性卷，并演示 EKS 自动模式的自动 EBS 卷预置和持久性功能。

先决条件

- 一个 EKS 自动模式集群

- 以恰当的权限配置了 Amazon CLI
- 安装并配置了 kubectl
  - 有关更多信息，请参阅 设置。

第 1 步：配置环境

1. 设置环境变量：

```
export CLUSTER_NAME=my-auto-cluster
export AWS_REGION="us-west-2"
```

2. 更新 kubeconfig：

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

第 2 步：创建存储类

StorageClass 定义了 EKS 自动模式将如何预置 EBS 卷。

EKS 自动模式不会为您创建 StorageClass。您必须创建引用 ebs.csi.eks.amazonaws.com 的 StorageClass，才能使用 EKS 自动模式的存储功能。

1. 创建一个名为 storage-class.yaml 的文件：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. 应用 StorageClass：

```
kubectl apply -f storage-class.yaml
```

关键组件：

- `provisioner: ebs.csi.eks.amazonaws.com` – 使用 EKS 自动模式
- `volumeBindingMode: WaitForFirstConsumer` – 将卷的创建延迟到容器组需要时
- `type: gp3` – 指定 EBS 卷类型
- `encrypted: "true"` – EBS 将使用默认的 `aws/ebs` 密钥来加密使用该类创建的卷。您可以自由选择，但我们建议您这样做。
- `storageclass.kubernetes.io/is-default-class: "true"` – 默认情况下 Kubernetes 将使用此存储类，除非您在持久性卷声明中指定了其他卷类。如果要从其他存储控制器迁移，请谨慎设置此值。（可选）

第 3 步：创建持久性卷声明

PVC 向 `StorageClass` 请求存储空间。

1. 创建一个名为 `pvc.yaml` 的文件：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: auto-ebs-sc
  resources:
    requests:
      storage: 8Gi
```

2. 应用 PVC：

```
kubectl apply -f pvc.yaml
```

关键组件：

- `accessModes: ReadWriteOnce` – 一次只能由一个节点挂载卷
- `storage: 8Gi` – 请求一个 8 GiB 卷
- `storageClassName: auto-ebs-sc` – 引用我们创建的 `StorageClass`

第 4 步：部署应用程序

部署会运行一个将时间戳写入持久性卷的容器。

1. 创建一个名为 deployment.yaml 的文件：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate-stateful
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate-stateful
  template:
    metadata:
      labels:
        app: inflate-stateful
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      containers:
        - name: bash
          image: public.ecr.aws/docker/library/bash:4.4
          command: ["/usr/local/bin/bash"]
          args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 60;
 done"]
          resources:
            requests:
              cpu: "1"
          volumeMounts:
            - name: persistent-storage
              mountPath: /data
      volumes:
        - name: persistent-storage
          persistentVolumeClaim:
            claimName: auto-ebs-claim
```

2. 应用部署：

```
kubectl apply -f deployment.yaml
```

关键组件：

- 将时间戳写入文件的简单 bash 容器
- 将 PVC 挂载到 /data
- 请求 1 个 CPU 核心
- 使用 EKS 托管式节点的节点选择器

第 5 步：验证设置

1. 检查容器组是否在运行：

```
kubectl get pods -l app=inflate-stateful
```

2. 验证 PVC 是否已绑定：

```
kubectl get pvc auto-ebs-claim
```

3. 检查 EBS 卷：

```
# Get the PV name
PV_NAME=$(kubectl get pvc auto-ebs-claim -o jsonpath='{.spec.volumeName}')
# Describe the EBS volume
aws ec2 describe-volumes \
  --filters Name=tag:CSIVolumeName,Values=${PV_NAME}
```

4. 验证数据是否正在写入：

```
kubectl exec "$(kubectl get pods -l app=inflate-stateful \
  -o=jsonpath='{.items[0].metadata.name}')" -- \
  cat /data/out.txt
```

步骤 6：清除

运行以下命令以移除本教程中创建的所有资源：

```
# Delete all resources in one command
kubectl delete deployment/inflate-stateful pvc/auto-ebs-claim storageclass/auto-ebs-sc
```

幕后发生了什么

1. PVC 向 StorageClass 请求存储空间
2. 当容器组被调度时：
   a. EKS 自动模式会预置一个 EBS 卷
   b. 创建一个持久性卷
   c. 将该卷挂载到节点
3. 容器组会挂载该卷并开始写入时间戳

快照控制器

EKS 自动模式与 Kubernetes CSI Snapshotter（也称为快照控制器）兼容。但是，Amazon EKS 自动模式不包括快照控制器。您负责安装和配置快照控制器。有关更多信息，请参阅 the section called “CSI 快照控制器”。

检查以下引用 EKS 自动模式存储功能的 VolumeSnapshotClass。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: auto-ebs-vsclass
driver: ebs.csi.eks.amazonaws.com
deletionPolicy: Delete
```

了解有关 Kubernetes CSI Snapshotter 的更多信息。

部署加速型工作负载

本教程演示了 Amazon EKS 自动模式如何简化硬件加速型工作负载的启动。Amazon EKS 自动模式通过自动执行各种关键基础设施组件操作，提供可开箱即用的计算、联网、负载均衡、存储以及身份访问和管理功能，从而简化集群本身以外的操作。

Amazon EKS 自动模式包含某些实例类型所需的驱动程序和设备插件，例如 NVIDIA 和 Amazon Neuron 驱动程序。您无需安装或更新这些组件。

EKS 自动模式会自动管理以下加速器的驱动程序：

- Amazon Trainium

- Amazon Inferentia
- Amazon EC2 加速型实例上的 NVIDIA GPU

> **Note**
>
> EKS 自动模式包含适用于 Kubernetes 的 NVIDIA 设备插件。该插件会自动运行，不会像集群中的进程守护程序集一样可见。

其他联网支持：

- Elastic Fabric Adapter（EFA）

Amazon EKS 自动模式消除了繁琐的加速器驱动程序和设备插件管理工作。

您还可以将集群扩展到零，从而节省成本。您可以将 EKS 自动模式配置为在没有工作负载运行时终止实例。这对于基于批处理的推理工作负载十分实用。

以下示例展示了如何使用 Amazon EKS 自动模式启动加速型工作负载。

先决条件

- 一个配置了 Amazon EKS 自动模式的 Kubernetes 集群。
- 一个在启用 `general-purpose` 或 `system` 托管式节点池时创建的 `default` EKS 节点类。

第 1 步：部署 GPU 工作负载

在此示例中，您将为基于 NVIDIA 的工作负载创建一个节点池，需要 45GB GPU 内存。使用 EKS 自动模式时，您可以使用 Kubernetes 调度约束来定义实例要求。

要部署 Amazon EKS 自动模式 `NodePool` 和示例 `workload`，请检查以下节点池和容器组（pod）定义并将其另存为 `nodepool-gpu.yaml` 和 `pod.yaml`：

nodepool-gpu.yaml

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
```

```
  name: gpu
spec:
  disruption:
    budgets:
    - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values:
          - g6e
          - g6
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      terminationGracePeriod: 24h0m0s
```

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  nodeSelector:
    eks.amazonaws.com/compute-type: auto
  restartPolicy: OnFailure
  containers:
  - name: nvidia-smi
```

```
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
    args:
    - "nvidia-smi"
    resources:
      requests:
        memory: "30Gi"
        cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        memory: "30Gi"
        nvidia.com/gpu: 1
  tolerations:
  - key: nvidia.com/gpu
    effect: NoSchedule
    operator: Exists
```

请注意，eks.amazonaws.com/compute-type: auto 选择器要求将工作负载部署到 Amazon EKS 自动模式节点上。该节点池还设置了一个污点，从而只允许调度具有 Nvidia GPU 容忍度的容器组。

将该节点池和工作负载应用到集群：

```
kubectl apply -f nodepool-gpu.yaml
kubectl apply -f pod.yaml
```

您应看到以下输出：

```
nodepool.karpenter.sh/gpu configured created
pod/nvidia-smi created
```

等待几秒钟，然后检查集群中的节点。现在，您应会看到自己的 Amazon EKS 自动模式集群中预置了一个新的节点：

```
> kubectl get nodes

NAME        TYPE           CAPACITY    ZONE         NODE                   READY   AGE
gpu-dnknr   g6e.2xlarge    on-demand   us-west-2b   i-02315c7d7643cdee6   True    76s
```

第 2 步：验证

您可以看到 Amazon EKS 自动模式启动了一个 g6e.2xlarge 实例，而不是一个 g6.2xlarge 实例，这是因为根据以下 Kubernetes 调度约束，该工作负载需要一个搭载 l40s GPU 的实例：

```
...
  nodeSelector:
    eks.amazonaws.com/instance-gpu-name: l40s
...
    requests:
        memory: "30Gi"
        cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        memory: "30Gi"
        nvidia.com/gpu: 1
```

现在，通过运行以下命令来查看容器日志：

```
kubectl logs nvidia-smi
```

示例输出：

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 535.230.02             Driver Version: 535.230.02   CUDA Version: 12.2
 |
|-----------------------------------------+----------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC
 |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M.
 |
|                                         |                      |               MIG M.
 |
|=========================================+======================
+======================|
|   0  NVIDIA L40S                    On  | 00000000:30:00.0 Off |                    0
 |
| N/A   27C    P8              23W / 350W |      0MiB / 46068MiB |      0%      Default
 |
```

```
|                                         |                      |                  N/A
 |
+-----------------------------------------+----------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| Processes:
 |
|  GPU   GI   CI        PID   Type   Process name                             GPU Memory
 |
|        ID   ID                                                              Usage
 |
|
=============================================================================================|
|  No running processes found
 |
+-----------------------------------------------------------------------------------------
+
```

您可以看到容器检测到其正在一个搭载 NVIDIA GPU 的实例上运行，并且您无需安装任何设备驱动程序，因为这些驱动程序均由 Amazon EKS 自动模式管理。

第 3 步：清除

要移除所有创建的对象，请使用 kubectl 删除示例部署和节点池，从而终止该节点：

```
kubectl delete -f nodepool-gpu.yaml
kubectl delete -f pod.yaml
```

节点池引用示例

创建 NVIDIA 节点池

以下节点池定义了如下要求：

- 仅启动 g6e 和 g6 系列的实例
- 合并空置达到 1 小时的节点
  - 使用 1 小时的 consolodateAfter 值，可支持高峰工作负载并减少节点中断。您可以根据工作负载要求微调 consolidateAfter。

搭载 GPU 实例系列的节点池以及合并功能示例

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  disruption:
    budgets:
    - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values:
          - g6e
          - g6
      terminationGracePeriod: 24h0m0s
```

您可以不设置 `eks.amazonaws.com/instance-gpu-name`，而改为使用 `eks.amazonaws.com/instance-family` 来指定实例系列。有关可能影响调度审查的其他已知标签，请参阅the section called “受 EKS 自动模式支持的标签”。

如果您有特定的存储需求，则可以通过在节点池中创建自己的节点类来引用，从而微调节点的临时存储 `iops`、`size` 和 `throughput`。详细了解可配置的节点类选项。

节点类的存储配置示例

```
apiVersion: eks.amazonaws.com/v1
```

```
kind: NodeClass
metadata:
  name: gpu
spec:
  ephemeralStorage:
    iops: 3000
    size: 80Gi
    throughput: 125
```

定义 Amazon Trainium 和 Amazon Inferentia 节点池

以下节点池设的 `eks.amazonaws.com/instance-category` 集规定只能启动 Inferentia 和 Trainium 系列的实例：

```
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values:
            - inf
            - trn
```

配置 EKS 自动模式设置

本章介绍如何配置 Amazon Elastic Kubernetes Service（EKS）自动模式集群的特定方面。虽然 EKS 自动模式会自动管理大多数基础设施组件，不过您也可以自定义某些功能来满足工作负载的要求。

使用本主题中介绍的配置选项，您可以修改联网设置、计算资源和负载均衡行为，同时继续享受自动化基础设施管理的优势。在进行任何配置更改之前，请检查以下各节中的可用选项，以确定哪种方法最适合您的需求。

您要配置哪些功能？	配置选项
节点联网和存储 • 跨公有和私有子网配置节点放置 • 定义用于节点访问控制的自定义安全组 • 自定义网络地址转换（SNAT）策略 • 启用 Kubernetes 网络策略、详细网络策略日志记录和监控 • 设置临时存储参数（大小、IOPS、吞吐量）	the section called “创建节点类”

您要配置哪些功能？	配置选项
• 配置使用自定义 KMS 密钥加密的临时存储 • 将不同子网中的容器组（pod）流量与节点隔离开来	
节点计算资源 • 选择特定的 EC2 实例类型和系列 • 定义 CPU 架构（x86_64、ARM64） • 配置容量类型（按需型、竞价型） • 指定可用区 • 配置节点污点和标签 • 设置 CPU 和内存使用量的资源限制	the section called “创建节点池”
静态容量节点池 • 无论容器组（pod）需求为何，保持固定数量的节点 • 为预留实例配置可预测的容量 • 为一致的基础设施占用空间设置特定的副本数量 • 手动扩展静态节点池 • 监控静态容量节点池状态	the section called “静态容量节点池”
应用程序负载均衡器设置 • 部署面向互联网的负载均衡器 • 配置跨区域负载均衡 • 设置空闲超时期限 • 启用 HTTP/2 和 WebSocket 支持 • 配置运行状况检查参数 • 指定 TLS 证书设置 • 定义目标组属性 • 设置 IP 地址类型（IPv4、双栈）	the section called “创建 IngressClass”

您要配置哪些功能？	配置选项
网络负载均衡器设置 • 配置直接容器组 IP 路由 • 启用跨区域负载均衡 • 设置连接空闲超时 • 配置运行状况检查参数 • 指定子网放置 • 设置 IP 地址类型（IPv4、双栈） • 配置保留客户端源 IP • 定义目标组属性	the section called “创建服务”
存储类设置 • 定义 EBS 卷类型（gp3、io1、io2 等） • 配置卷加密和 KMS 密钥使用情况 • 设置 IOPS 和吞吐量参数 • 设置为默认存储类 • 为预置的卷定义自定义标签	the section called “创建 StorageClass”
控制 ODCR 使用情况 • 配置将工作负载部署到 EC2 按需容量预留 • 按 ID 明确选择特定的 ODCR 以实现目标容量使用 • 使用基于标签的选择来定位 ODCR 组 • 按拥有的 Amazon 账户筛选 ODCR 以支持跨账户场景 • 控制工作负载是否自动使用开放式 ODCR	the section called “控制容量预留”

为 Amazon EKS 创建节点类

Amazon EKS 节点类是模板，可对 EKS 自动模式托管式节点的配置进行精细控制。节点类定义了适用于 EKS 集群中节点组的基础设施级别设置，包括网络配置、存储设置和资源标记等。本主题旨在介绍如何创建和配置节点类以满足您的具体运行要求。

如果需要在默认设置之外自定义 EKS 自动模式预置和配置 EC2 实例的方式，创建节点类可让您精准控制关键基础设施参数。例如，您可以指定私有子网放置来增强安全性，为性能敏感型工作负载配置实例临时存储，或者应用自定义标记以满足成本分配的需要。

创建节点类

要创建 NodeClass，请按以下步骤操作：

1. 创建一个包含节点类配置的 YAML 文件（例如 nodeclass.yaml）
2. 使用 kubectl 将此配置应用到集群
3. 在节点池配置中引用该节点类。有关更多信息，请参阅 the section called “创建节点池”。

您需要已经安装并配置好 kubectl。有关更多信息，请参阅 设置。

基本节点类示例

以下是一个节点类示例：

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: private-compute
spec:
  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
        kubernetes.io/role/internal-elb: "1"
  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"
  ephemeralStorage:
    size: "160Gi"
```

此节点类增加了节点上的临时存储量。

使用以下命令应用此配置：

```
kubectl apply -f nodeclass.yaml
```

然后在节点池配置中引用该节点类。有关更多信息，请参阅 the section called “创建节点池”。

创建节点类访问条目

如果您创建自定义节点类，则需要创建 EKS 访问条目以允许节点加入集群。如果您使用内置节点类和节点池，EKS 会自动创建访问条目。

有关访问条目工作原理的信息，请参阅the section called “访问条目”。

为 EKS 自动模式节点类创建访问条目时，需要使用 EC2 访问条目类型。

使用 CLI 创建访问条目

要为 EC2 节点创建访问条目并关联 EKS 自动节点策略，请执行以下操作：

使用集群名称和节点角色 ARN 更新以下 CLI 命令。在节点类 YAML 中指定节点角色 ARN。

```
# Create the access entry for EC2 nodes
aws eks create-access-entry \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --type EC2

# Associate the auto node policy
aws eks associate-access-policy \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --policy-arn arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy \
  --access-scope type=cluster
```

使用 CloudFormation 创建访问条目

要为 EC2 节点创建访问条目并关联 EKS 自动节点策略，请执行以下操作：

使用集群名称和节点角色 ARN 更新以下 CloudFormation。在节点类 YAML 中指定节点角色 ARN。

```
EKSAutoNodeRoleAccessEntry:
  Type: AWS::EKS::AccessEntry
  Properties:
    ClusterName: <cluster-name>
```

```
    PrincipalArn: <node-role-arn>
    Type: "EC2"
    AccessPolicies:
      - AccessScope:
          Type: cluster
        PolicyArn: arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy
  DependsOn: [ <cluster-name> ] # previously defined in CloudFormation
```

有关部署 CloudFormation 堆栈的信息，请参阅 CloudFormation 入门

节点类规范

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Required fields

  # role and instanceProfile are mutually exclusive fields.
  role: MyNodeRole  # IAM role for EC2 instances
  # NOTE: instance profile names must start with the 'eks' prefix. this is a
  # temporary limitation to be lifted soon.
  # instanceProfile: eks-MyNodeInstanceProfile  # IAM instance-profile for EC2
 instances

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
        kubernetes.io/role/internal-elb: "1"
    # Alternative using direct subnet ID
    # - id: "subnet-0123456789abcdef0"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"
    # Alternative approaches:
    # - id: "sg-0123456789abcdef0"
    # - name: "eks-cluster-security-group"

  # Optional: Pod subnet selector for advanced networking
  podSubnetSelectorTerms:
    - tags:
```

```
        Name: "pod-subnet"
        kubernetes.io/role/pod: "1"
    # Alternative using direct subnet ID
    # - id: "subnet-0987654321fedcba0"
# must include Pod security group selector also
 podSecurityGroupSelectorTerms:
  - tags:
      Name: "eks-pod-sg"
    # Alternative using direct security group ID
    # - id: "sg-0123456789abcdef0"

  # Optional: Selects on-demand capacity reservations and capacity blocks
  # for EKS Auto Mode to prioritize.
  capacityReservationSelectorTerms:
    - id: cr-56fac701cc1951b03
    # Alternative Approaches
    - tags:
        Name: "targeted-odcr"
      # Optional owning account ID filter
      owner: "012345678901"

  # Optional fields
  snatPolicy: Random  # or Disabled

  networkPolicy: DefaultAllow  # or DefaultDeny
  networkPolicyEventLogs: Disabled  # or Enabled

  ephemeralStorage:
    size: "80Gi"    # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
    iops: 3000      # Range: 3000-16000
    throughput: 125 # Range: 125-1000
    # Optional KMS key for encryption
    kmsKeyID: "arn:aws-cn:kms:region:account:key/key-id"
    # Accepted formats:
    # KMS Key ID
    # KMS Key ARN
    # Key Alias Name
    # Key Alias ARN

  advancedNetworking:
    # Optional: Controls whether public IP addresses are assigned to instances that are
 launched with the nodeclass.
    # If not set, defaults to the MapPublicIpOnLaunch setting on the subnet.
    associatePublicIPAddress: false
```

```
    # Optional: Forward proxy, commonly requires certificateBundles as well
    # for EC2, see https://repost.aws/knowledge-center/eks-http-proxy-containerd-
automation
    httpsProxy: http://192.0.2.4:3128 #commonly port 3128 (Squid) or 8080 (NGINX) #Max
 255 characters
    #httpsProxy: http://[2001:db8::4]:3128 # IPv6 address with port, use []
    noProxy: #Max 50 entries
        - localhost #Max 255 characters each
        - 127.0.0.1
        #- ::1 # IPv6 localhost
        #- 0:0:0:0:0:0:0:1 # IPv6 localhost
        - 169.254.169.254 # EC2 Instance Metadata Service
        #- [fd00:ec2::254] # IPv6 EC2 Instance Metadata Service
        # Domains to exclude, put all VPC endpoints here
        - .internal
        - .eks.amazonaws.com

  advancedSecurity:
    # Optional, US regions only: Specifying `fips: true` will cause nodes in the
 nodeclass to run FIPS compatible AMIs.
    fips: false

  # Optional: Custom certificate bundles.
  certificateBundles:
    - name: "custom-cert"
      data: "base64-encoded-cert-data"

  # Optional: Additional EC2 tags (with restrictions)
  tags:
    Environment: "production"
    Team: "platform"
    # Note: Cannot use restricted tags like:
    # - kubernetes.io/cluster/*
    # - karpenter.sh/provisioner-name
    # - karpenter.sh/nodepool
    # - karpenter.sh/nodeclaim
    # - karpenter.sh/managed-by
    # - eks.amazonaws.com/nodeclass
```

注意事项

- 如果要验证实例的本地存储空间大小，您可以描述该节点以查看临时存储资源。

- 卷加密：EKS 使用配置的自定义 KMS 密钥来加密实例的只读根卷和读/写数据卷。
- 替换节点 IAM 角色 – 如果您更改与 NodeClass 关联的节点 IAM 角色，则需要创建一个新的访问条目。在创建集群期间，EKS 会自动为节点 IAM 角色创建一个访问条目。节点 IAM 角色需要具有 AmazonEKSAutoNodePolicy EKS 访问策略。有关更多信息，请参阅 the section called “访问条目”。
- 最大容器组（pod）密度：在 EKS 中，一个节点上最多可以有 110 个容器组（pod）。此限制将在计算现有最大容器组（pod）数量之后应用。有关更多信息，请参阅 the section called “Amazon EC2 实例类型”。
- 标签 – 如果要将标签从 Kubernetes 传播到 EC2，则需要配置额外的 IAM 权限。有关更多信息，请参阅 the section called “身份和访问权限”。
- 默认节点类 – 不要将自定义节点类命名为 default。这是因为 EKS 自动模式包含一个名为 default 的 NodeClass，您启用至少一个内置 NodePool 时，系统会自动预置该节点类。有关启用内置 NodePools 的信息，请参阅the section called “检查内置节点池”。
- 多个子网的 **subnetSelectorTerms** 行为 – 如果存在多个符合 subnetSelectorTerms 条件或您按照 ID 提供的子网，则 EKS 自动模式会创建分布在子网中的节点。
  - 如果子网位于不同的可用区（AZ），则可以使用 Kubernetes 功能 [例如容器组（pod）拓扑分布约束和拓扑感知路由] 分别在各个区域之间分布容器组（pod）和流量。
  - 如果同一个可用区中有多个子网与 subnetSelectorTerms 匹配，则 EKS 自动模式会在分布在该可用区子网中的每个节点上创建容器组（pod）。EKS 自动模式在同一可用区内其他子网中的每个节点上创建辅助网络接口。它根据每个子网中可用 IP 地址的数量进行选择，以便更高效地使用子网。但是，您无法指定 EKS 自动模式为每个容器组（pod）使用哪个子网；如果您需要容器组（pod）在特定子网中运行，请改用 the section called “容器组（pod）的子网选择”。

容器组（pod）的子网选择

podSubnetSelectorTerms 和 podSecurityGroupSelectorTerms 字段允许容器组（pod）在与其节点不同的子网中运行，从而实现高级联网配置。这种分离增强了对网络流量路由和安全策略的控制。请注意，podSecurityGroupSelectorTerms 需与 podSubnetSelectorTerms 一同使用。

使用案例

在需要执行以下操作时使用 podSubnetSelectorTerms：

- 将基础设施流量（节点到节点通信）与应用程序流量 [容器组（pod）到容器组（pod）通信] 分开
- 对节点子网应用与容器组（pod）子网不同的网络配置。
- 为节点和容器组（pod）实施不同的安全策略或路由规则。

- 在不影响容器组（pod）流量的情况下，专门为节点流量配置反向代理或网络筛选。使用 advancedNetworking 和 certificateBundles 定义您的反向代理以及代理的任何自签名证书或私有证书。

示例配置

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  # Subnets for EC2 instances (nodes)
  subnetSelectorTerms:
    - tags:
        Name: "node-subnet"
        kubernetes.io/role/internal-elb: "1"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  # Separate subnets for Pods
  podSubnetSelectorTerms:
    - tags:
        Name: "pod-subnet"
        kubernetes.io/role/pod: "1"

  podSecurityGroupSelectorTerms:
  - tags:
      Name: "eks-pod-sg"
```

容器组（pod）子网选择器的注意事项

- 降低容器组（pod）密度：使用 podSubnetSelectorTerms 时每个节点上可以运行的容器组（pod）更少，因为节点的主网络接口位于节点子网中，不能用于容器组（pod）子网中的容器组（pod）。
- 子网选择器限制：标准 subnetSelectorTerms 和 securityGroupSelectorTerms 配置不适用于容器组（pod）子网选择。

- 网络规划：确保节点和容器组（pod）子网中有足够的 IP 地址空间，以满足工作负载需求。
- 路由配置：验证容器组（pod）子网的路由表和网络访问控制列表（ACL）是否已针对节点和容器组（pod）子网之间的通信进行正确配置。
- 可用区：确认已跨多个可用区创建容器组（pod）子网。如果您使用的是特定容器组（pod）子网，则该子网必须与节点子网可用区位于同一可用区中。

为 EKS 自动模式创建节点池

借助 Amazon EKS 节点池可以灵活地管理 Kubernetes 集群中的计算资源。本主题演示如何使用 Karpenter 创建和配置节点池，Karpenter 是一种有助于优化集群扩展和资源利用率的节点预置工具。借助 Karpenter 的节点池资源，您可以定义计算资源的具体要求，包括实例类型、可用区、架构和容量类型。

您无法修改内置的 `system` 和 `general-purpose` 节点池，只能启用或禁用这些节点池。有关更多信息，请参阅 the section called “检查内置节点池”。

借助节点池规范，您可以通过各种受支持的标签和要求对 EKS 集群的计算资源进行精细控制。其中包括用于指定 EC2 实例类别、CPU 配置、可用区、架构（ARM64/AMD64）和容量类型（竞价型或按需型）的选项。您还可以设置有关 CPU 和内存使用量的资源限制，确保集群始终遵守所需的运行界限。

EKS 自动模式利用通用的 Kubernetes 标签，从而通过一致和标准化的方法来标识节点特征。这些标签（例如用于可用区的 `topology.kubernetes.io/zone` 标签和用于 CPU 架构的 `kubernetes.io/arch` 标签）都遵循约定的 Kubernetes 规范。此外，特定于 EKS 的标签（前缀为 `eks.amazonaws.com/`）通过实例类型、CPU 制造商、GPU 功能和联网规范等 Amazon 特定属性进一步扩展了此功能。这种标准化的标记系统有利于与现有 Kubernetes 工具的无缝集成，同时实现深入的 Amazon 基础设施集成。

创建节点池

按照以下步骤为 Amazon EKS 集群创建节点池：

1. 创建一个包含所需节点池配置的 YAML 文件并将其命名为 `nodepool.yaml`。您可以使用以下示例配置。
2. 将节点池应用到集群：

```
kubectl apply -f nodepool.yaml
```

3. 验证节点池是否已成功创建：

```
kubectl get nodepools
```

4. （可选）监控节点池状态：

```
kubectl describe nodepool default
```

确认节点池引用了集群中存在的有效节点类。节点类定义了计算资源的 Amazon 特定配置。有关更多信息，请参阅 the section called “创建节点类”。

示例节点池

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
    metadata:
      labels:
        billing-team: my-team
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["4", "8", "16", "32"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["arm64", "amd64"]

  limits:
```

```
    cpu: "1000"
    memory: 1000Gi
```

受 EKS 自动模式支持的标签

EKS 自动模式支持以下通用标签。

> **Note**
>
> EKS 自动模式使用的标签与 Karpenter 不同。与 EC2 托管式实例相关的标签以 `eks.amazonaws.com` 开头。

标签	示例	描述
topology.kubernetes.io/zone	us-east-2a	Amazon 区域
node.kubernetes.io/instance-type	g4dn.8xlarge	Amazon 实例类型
kubernetes.io/arch	amd64	架构是由实例上的 GOARCH 值来定义
karpenter.sh/capacity-type	spot	容量类型包括 `spot`、`on-demand`
eks.amazonaws.com/instance-hypervisor	nitro	使用特定虚拟机监控程序的实例类型
eks.amazonaws.com/compute-type	自动	标识 EKS 自动模式托管式节点
eks.amazonaws.com/instance-encryption-in-transit-supported	true	支持（或不支持）传输中加密的实例类型
eks.amazonaws.com/instance-category	g	相同类别的实例类型，通常是代次编号之前的字符串
eks.amazonaws.com/instance-generation	4	实例类别中的实例类型代次编号

标签	示例	描述
eks.amazonaws.com/instance-family	g4dn	属性相似但资源数量不同的实例类型
eks.amazonaws.com/instance-size	8xlarge	资源数量相似但属性不同的实例类型
eks.amazonaws.com/instance-cpu	32	实例上的 CPU 数量
eks.amazonaws.com/instance-cpu-manufacturer	aws	CPU 制造商的名称
eks.amazonaws.com/instance-memory	131072	实例内存兆比字节数
eks.amazonaws.com/instance-ebs-bandwidth	9500	实例可用 EBS 最大兆比特数
eks.amazonaws.com/instance-network-bandwidth	131072	实例可用基线兆比特数
eks.amazonaws.com/instance-gpu-name	t4	实例上 GPU 的名称（如有）
eks.amazonaws.com/instance-gpu-manufacturer	nvidia	CPU 制造商的名称
eks.amazonaws.com/instance-gpu-count	1	实例 GPU 数量
eks.amazonaws.com/instance-gpu-memory	16384	GPU 内存兆比字节数
eks.amazonaws.com/instance-local-nvme	900	实例本地 nvme 存储空间千兆比字节数

> **Note**
>
> EKS 自动模式仅支持某些实例，并对最小大小有要求。有关更多信息，请参阅 the section called “EKS 自动模式支持的实例参考”。

不受 EKS 自动模式支持的标签

EKS 自动模式不支持以下标签。

- EKS 自动模式仅支持 Linux
  - `node.kubernetes.io/windows-build`
  - `kubernetes.io/os`

禁用内置节点池

如果创建自定义节点池，则可以禁用内置节点池。有关更多信息，请参阅 the section called “检查内置节点池”。

不含内置节点池的集群

您可以创建不含内置节点池的集群。如果组织创建了自定义节点池，此功能就大有用处。

> **Note**
>
> 当您创建不含内置节点池的集群时，不会自动预置 `default` NodeClass。您需要创建一个自定义 NodeClass。有关更多信息，请参阅 the section called “创建节点类”。

概述：

1. 创建一个 EKS 集群，`nodePools` 值和 `nodeRoleArn` 值皆为空。
   - 示例 eksctl `autoModeConfig`：

```
autoModeConfig:
  enabled: true
  nodePools: []
  # Do not set a nodeRoleARN
```

有关更多信息，请参阅 the section called “eksctl CLI”。

2. 创建具有节点角色 ARN 的自定义节点类
   - 有关更多信息，请参阅 the section called “创建节点类”。
3. 为自定义节点类创建访问条目
   - 有关更多信息，请参阅 the section called “创建节点类访问条目”。
4. 如上所述，创建自定义节点池。

中断

您可以配置 EKS 自动模式，以多种方式通过 NodePool 中断节点。您可以使用 `spec.disruption.consolidationPolicy`、`spec.disruption.consolidateAfter` 或 `spec.template.spec.expireAfter`。您也可以通过 NodePool 的 `spec.disruption.budgets` 对 EKS 自动模式的中断进行速率限制。您还可以控制时间范围和同时中断的节点数量。有关配置此行为的说明，请参阅 Karpenter 文档中的 Disruption。

您可就以下目的配置节点池中断功能：

- 发现实例未得到充分利用的情况，据此整合工作负载。
- 创建节点池中断预算，对偏移、空置和整合导致的节点终止进行速率限制。

EKS 自动模式默认会：

- 整合未得到充分利用的实例。
- 在 336 小时后终止实例。
- 将单个中断预算设置为 10% 的节点。
- 允许在发布新的自动模式 AMI（约每周一次）时由于漂移而更换节点。

终止宽限期

如果未在 EKS Auto NodePool 上明确定义 `terminationGracePeriod`，系统会自动对关联的 NodeClaim 应用默认的 24 小时终止宽限期。虽然 EKS Auto 客户不会在其自定义 NodePool 配置中看到默认的 `terminationGracePeriod`，但他们会在 NodeClaim 上看到这个默认值。无论是在 NodePool 上明确设置了宽限期还是在 NodeClaim 上设置了默认宽限期，该功能都保持一致，从而确保整个集群中的节点终止行为可预测。

EKS 自动模式中的静态容量节点池

Amazon EKS 自动模式支持静态容量节点池，无论容器组（pod）需求如何，这些节点池都保持固定数量的节点。如果工作负载需要可预测容量、预留实例，或需要保持稳定的基础设施占用空间以满足特定合规性要求，静态容量节点池非常有用。

与基于容器组（pod）调度要求进行扩展的动态节点池不同，静态容量节点池会保留您配置的节点数量。

基本示例

这是一个简单的静态容量节点池，其中维持了 5 个节点：

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-static-nodepool
spec:
  replicas: 5  # Maintain exactly 5 nodes

  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["m", "c"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b"]

  limits:
    nodes: 8
```

配置静态容量节点池

要创建静态容量节点池，请在 NodePool 规范中设置 `replicas` 字段。`replicas` 字段定义节点池将维持的确切节点数。

静态容量节点池限制

静态容量节点池有几个重要的限制和行为：

配置限制：

- 无法切换模式：一旦在节点池上设置了 `replicas`，就无法将其移除。节点池无法在静态和动态模式之间切换。
- 有限的资源限制：限制部分中仅支持 `limits.nodes` 字段。CPU 和内存限制不适用。
- 无权重字段：由于节点选择不是基于优先级，因此无法在静态容量节点池上设置 `weight` 字段。

操作行为：

- 无整合：静态容量池中的节点不会根据利用率被纳入整合范围。
- 扩展操作：扩展操作会绕过节点中断预算，但仍然遵守 PodDisruptionBudgets。
- 节点更换：节点仍会因偏差（例如 AMI 更新）或因过期而被替换，具体方式取决于您的配置。

扩展静态容量节点池

您可以使用 `kubectl scale` 命令更改静态容量节点池中的副本数量：

```
# Scale down to 5 nodes
kubectl scale nodepool static-nodepool --replicas=5
```

进行缩减时，EKS 自动模式将正常终止节点，同时遵守 PodDisruptionBudgets，并允许将正在运行的容器组（pod）重新调度至剩余节点。

监控静态容量节点池

使用以下命令监控静态容量节点池：

```
# View node pool status
kubectl get nodepool static-nodepool

# Get detailed information including current node count
kubectl describe nodepool static-nodepool

# Check the current number of nodes
```

```
kubectl get nodepool static-nodepool -o jsonpath='{.status.nodes}'
```

status.nodes 字段显示节点池当前管理的节点数量，在正常条件下，该数量应与所需的 replicas 数量相匹配。

示例配置

基本静态容量节点池

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: basic-static
spec:
  replicas: 5

  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["m"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a"]

  limits:
    nodes: 8  # Allow scaling up to 8 during operations
```

特定实例类型的静态容量

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: reserved-instances
spec:
  replicas: 20
```

```
  template:
    metadata:
      labels:
        instance-type: reserved
        cost-center: production
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "node.kubernetes.io/instance-type"
          operator: In
          values: ["m5.2xlarge"]  # Specific instance type
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b", "us-west-2c"]

  limits:
    nodes: 25

  disruption:
    # Conservative disruption for production workloads
    budgets:
      - nodes: 10%
```

多可用区静态容量节点池

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: multi-zone-static
spec:
  replicas: 12  # Will be distributed across specified zones

  template:
    metadata:
      labels:
```

```
        availability: high
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["8", "16"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b", "us-west-2c"]
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]

  limits:
    nodes: 15

  disruption:
    budgets:
      - nodes: 25%
```

最佳实践

容量规划。

- 将 `limits.nodes` 设置为高于 `replicas`，以允许在节点替换操作期间进行临时扩展。
- 设置限制时，请考虑节点偏差或 AMI 更新期间所需的最大容量。

实例选择：

- 如果您有预留实例或特定的硬件要求，请使用特定的实例类型。
- 避免要求过于严格，因为这可能会限制扩展期间的实例可用性。

中断管理：

- 配置适当的中断预算，以平衡可用性与维护操作。
- 在设置预算百分比时，请考虑应用程序对节点替换的容忍度。

监控

- 定期监控 `status.nodes` 字段，确保保持所需容量。
- 设置警报，以了解实际节点数何时偏离所需的副本。

可用区分布：

- 为了实现高可用性，请在多个可用区之间分布静态容量。
- 当您创建跨多个可用区的静态容量节点池时，EKS 自动模式会将节点分布到指定可用区，但不能保证分布均匀。
- 为了在可用区之间实现可预测且均匀的分布，请创建单独的静态容量节点池，每个节点池都根据 `topology.kubernetes.io/zone` 要求固定到特定的可用区。
- 如果您需要在 3 个可用区中均匀分布 12 个节点，请创建 3 个节点池，每个节点池包含 4 个副本，而不是在 3 个可用区中仅创建 1 个节点池，其中共 12 个副本。

问题排查

节点未到达所需的副本数：

- 检查 `limits.nodes` 值是否足够
- 确认您的要求不会过度限制实例选择
- 查看您正在使用的实例类型和区域的 Amazon 服务配额

节点替换花费的时间太长：

- 调整中断预算，允许更多并发替换
- 检查 PodDisruptionBudgets 是否阻止了节点终止

节点意外终止：

- 检查 `expireAfter` 和 `terminationGracePeriod` 设置
- 检查是否存在手动节点终止或 Amazon 维护事件

创建 IngressClass 以配置应用程序负载均衡器

EKS 自动模式可自动执行例行负载均衡任务，包括向互联网公开集群应用程序等。

Amazon 建议使用应用程序负载均衡器（ALB）来处理 HTTP 和 HTTPS 流量。应用程序负载均衡器可以根据请求的内容进行路由。有关应用程序负载均衡器的更多信息，请参阅 What is Elastic Load Balancing?

EKS 自动模式会创建和配置应用程序负载均衡器（ALB）。例如，EKS 自动模式会在您创建 `Ingress` Kubernetes 对象时创建一个负载均衡器，并配置为使用该负载均衡器来路由指向集群工作负载的流量。

概述

1. 创建您希望向互联网公开的工作负载。
2. 创建一个 `IngressClassParams` 资源并指定 Amazon 特定的配置值，例如用于 SSL/TLS 和 VPC 子网的证书。
3. 创建一个 `IngressClass` 资源并将 EKS 自动模式指定为该资源的控制器。
4. 创建一个 `Ingress` 资源，以将某个 HTTP 路径和端口与集群工作负载关联。

EKS 自动模式将使用 `IngressClassParams` 资源中指定的负载均衡器配置，创建一个指向 `Ingress` 资源中指定工作负载的应用程序负载均衡器。

先决条件

- 在 Amazon EKS 集群上启用 EKS 自动模式
- 配置好 Kubectl 以连接到集群
  - 您可以使用 `kubectl apply -f <filename>` 将以下示例配置 YAML 文件应用到集群。

> **Note**
>
> EKS 自动模式需要子网标签来标识公有子网和私有子网。
> 如果使用 `eksctl` 创建集群，则已包含这些标签。
> 了解如何the section called “为子网添加标签”。

第 1 步：创建工作负载

首先，创建您希望向互联网公开的工作负载。这可以是任何提供 HTTP 流量的 Kubernetes 资源，例如部署或服务。

此示例使用一个名为 `service-2048`、在端口 `80` 上侦听的简单 HTTP 服务。通过应用以下 `2048-deployment-service.yaml` 清单来创建此服务及其部署：

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-2048
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 2
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          name: app-2048
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: service-2048
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: NodePort
  selector:
    app.kubernetes.io/name: app-2048
```

将配置应用到您的集群：

```
kubectl apply -f 2048-deployment-service.yaml
```

上面列出的资源将在默认命名空间中创建。可以通过运行以下命令进行验证：

```
kubectl get all -n default
```

第 2 步：创建 IngressClassParams

创建一个 IngressClassParams 对象来指定应用程序负载均衡器的 Amazon 特定配置选项。在本示例中，我们创建了一个名为 alb 的 IngressClassParams 资源（您将在下一步中使用该资源），该资源在名为 alb-ingressclassparams.yaml 的文件中将负载均衡器方案指定为 internet-facing。

```
apiVersion: eks.amazonaws.com/v1
kind: IngressClassParams
metadata:
  name: alb
spec:
  scheme: internet-facing
```

将配置应用到您的集群：

```
kubectl apply -f alb-ingressclassparams.yaml
```

第 3 步：创建 IngressClass

在名为 alb-ingressclass.yaml 的文件中创建 IngressClass，它引用 IngressClassParams 资源中设置的 Amazon 特定配置值。记下 IngressClass 的名称。在此示例中，IngressClass 和 IngressClassParams 的名称都为 alb。

使用 is-default-class 注释来控制 Ingress 资源在默认情况下是否应使用该类。

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    # Use this annotation to set an IngressClass as Default
```

```
    # If an Ingress doesn't specify a class, it will use the Default
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  # Configures the IngressClass to use EKS Auto Mode
  controller: eks.amazonaws.com/alb
  parameters:
    apiGroup: eks.amazonaws.com
    kind: IngressClassParams
    # Use the name of the IngressClassParams set in the previous step
    name: alb
```

有关配置选项的更多信息，请参阅 the section called “IngressClassParams 参考”。

将配置应用到您的集群：

```
kubectl apply -f alb-ingressclass.yaml
```

第 4 步：创建 Ingress

在名为 `alb-ingress.yaml` 的文件中创建 `Ingress` 资源。此资源的用途是将应用程序负载均衡器上的路径和端口关联到集群中的工作负载。在本示例中，我们创建了一个名为 `2048-ingress` 的 `Ingress` 资源，该资源将流量路由到端口 80 上名为 `service-2048` 的服务。

有关如何配置此资源的更多信息，请参阅 Kubernetes 文档中的 Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: 2048-ingress
spec:
  # this matches the name of IngressClass.
  # this can be omitted if you have a default ingressClass in cluster: the one with
 ingressclass.kubernetes.io/is-default-class: "true"  annotation
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /*
            pathType: ImplementationSpecific
            backend:
              service:
                name: service-2048
```

```
                port:
                  number: 80
```

将配置应用到您的集群：

```
kubectl apply -f alb-ingress.yaml
```

第 5 步：检查状态

使用 kubectl 来查找 Ingress 的状态。负载均衡器可能需要过几分钟会可用。

使用您在上一步中设置的 Ingress 资源的名称。例如：

```
kubectl get ingress 2048-ingress
```

资源准备就绪后，检索负载均衡器的域名。

```
kubectl get ingress 2048-ingress -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

要在 Web 浏览器中查看服务，请检查 Ingress 抢救文件中指定的端口和路径。

第 6 步：清除

要清理负载均衡器，请使用以下命令：

```
kubectl delete ingress 2048-ingress
kubectl delete ingressclass alb
kubectl delete ingressclassparams alb
```

EKS 自动模式会自动删除您的 Amazon 账户中关联的负载均衡器。

IngressClassParams 参考

下表列举了常用配置选项的快速参考。

字段	说明	示例值
scheme	定义 ALB 是内部专用还是面向互联网	internet-facing

字段	说明	示例值
namespaceSelector	限定可以使用此 IngressClass 的命名空间	environment: prod
group.name	将多个 Ingress 分组以共享单个 ALB	retail-apps
ipAddressType	设置 ALB 的 IP 地址类型	dualstack
subnets.ids	用于 ALB 部署的子网 ID 列表	subnet-xxxx, subnet-yyyy
subnets.tags	用于为 ALB 选择子网的标签筛选条件	Environment: prod
certificateARNs	要使用的 SSL 证书的 ARN	arn:aws-cn:acm:region:account:certificate/id
tags	Amazon 资源的自定义标签	Environment: prod, Team: platform
loadBalancerAttributes	负载均衡器特定的属性	idle_timeout.timeout_seconds: 60

注意事项

- 您不能使用在 IngressClass 上的注释来配置用于 EKS 自动模式的负载均衡器。
- 您无法使用 EKS 自动模式设置 ListenerAttribute。
- 您必须更新集群 IAM 角色才能支持将标签从 Kubernetes 传播到 Amazon 负载均衡器资源。有关更多信息，请参阅 the section called “EKS 自动模式资源的自定义 Amazon 标签”。
- 有关将资源关联到 EKS 自动模式或自主管理型 Amazon 负载均衡器控制器的信息，请参阅the section called “迁移参考”。
- 有关修复负载均衡器问题的信息，请参阅 the section called “故障排除”。
- 有关使用 EKS 自动模式的负载均衡功能时的其他注意事项，请参阅the section called “负载均衡”。

以下表格详细比较了 EKS 自动模式的 IngressClassParams、Ingress 注释和 TargetGroupBinding 配置的更改。这些表格重点介绍了 EKS 自动模式的负载均衡功能与开源负载均衡器控制器之间的主要区别，包括 API 版本更改、已弃用的功能和更新后的参数名称等。

IngressClassParams

旧版	New	说明
`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`	API 版本更改
`spec.certificateArn`	`spec.certificateARNs`	支持多个证书 ARN
`spec.subnets.tags`	`spec.subnets.matchTags`	更改了子网匹配架构
`spec.listeners.listenerAttributes`	不支持	尚未得到 EKS 自动模式的支持

Ingress 注释

旧版	New	说明
`kubernetes.io/ingress.class`	不支持	在 Ingress 对象上使用 `spec.ingressClassName`
`alb.ingress.kubernetes.io/group.name`	不支持	仅指定 IngressClass 中的组
`alb.ingress.kubernetes.io/waf-acl-id`	不支持	改用 WAF v2
`alb.ingress.kubernetes.io/web-acl-id`	不支持	改用 WAF v2

旧版	New	说明
`alb.ingress.kubernetes.io/shield-advanced-protection`	不支持	禁用了 Shield 集成
`alb.ingress.kubernetes.io/auth-type: oidc`	不支持	当前不支持 OIDC 身份验证类型。

TargetGroupBinding

旧版	New	说明
`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`	API 版本更改
`spec.targetType` 可选	`spec.targetType` 必需	显式目标类型规范
`spec.networking.ingress.from`	不支持	不再支持没有安全组的 NLB

要使用自定义 TargetGroupBinding 功能，您必须使用带有集群名称的 eks:eks-cluster-name 标签标记目标组，从而向控制器授予必要的 IAM 权限。请注意，当 TargetGroupBinding 资源或集群被删除时，控制器将删除目标组。

使用服务注释配置网络负载均衡器

了解如何使用 Kubernetes 服务注释在 Amazon EKS 中配置网络负载均衡器（NLB）。本主题介绍 EKS 自动模式支持的用于自定义 NLB 行为的注释，包括互联网可访问性、运行状况检查、SSL/TLS 终止和 IP 目标模式。

在 EKS 自动模式下创建类型为 `LoadBalancer` 的 Kubernetes 服务时，EKS 会根据您指定的注释自动预置和配置 Amazon 网络负载均衡器。借助这种声明式方法，您可以直接通过 Kubernetes 清单来管理负载均衡器配置，确保遵循基础设施即代码实践。

对于所有 LoadBalancer 类型的服务，EKS 自动模式会默认处理网络负载均衡器预置，无需额外的控制器安装或配置。`loadBalancerClass: eks.amazonaws.com/nlb` 规范会自动设置为集群的默认值，这不仅简化了部署过程，同时也保持了与现有 Kubernetes 工作负载的兼容性。

> **Note**
>
> EKS 自动模式需要子网标签来标识公有子网和私有子网。
> 如果使用 `eksctl` 创建集群，则已包含这些标签。
> 了解如何the section called “为子网添加标签”。

示例服务

有关 Kubernetes `Service` 资源的更多信息，请参阅 Kubernetes 文档。

请参阅下面的示例 `Service` 资源：

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
  annotations:
    # Specify the load balancer scheme as internet-facing to create a public-facing
 Network Load Balancer (NLB)
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  selector:
    app: echoserver
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  # Specify the new load balancer class for NLB as part of EKS Auto Mode feature
  # For clusters with Auto Mode enabled, this field can be omitted as it's the default
  loadBalancerClass: eks.amazonaws.com/nlb
```

常用注释

下表列举了 EKS 自动模式支持的常用注释。请注意，EKS 自动模式可能并非支持所有注释。

> **Tip**
>
> 以下所有注释都需要加上前缀 `service.beta.kubernetes.io/`

字段	说明	示例
`aws-load-balancer-type`	指定负载均衡器类型。新部署请使用 `external`。	`external`
`aws-load-balancer-nlb-target-type`	指定是将流量路由到节点实例还是直接路由到容器组 IP。标准部署请使用 `instance`，直接路由到容器组请使用 `ip`。	`instance`
`aws-load-balancer-scheme`	指定负载均衡器是面向内部还是互联网。	`internet-facing`
`aws-load-balancer-healthcheck-protocol`	目标组的运行状况检查协议。常用选项是 TCP（默认）或 HTTP。	`HTTP`
`aws-load-balancer-healthcheck-path`	使用 HTTP/HTTPS 协议时进行运行状况检查的 HTTP 路径。	`/healthz`
`aws-load-balancer-healthcheck-port`	用于运行状况检查的端口。可以是特定的端口号，也可以是 `traffic-port` 。	`traffic-port`
`aws-load-balancer-subnets`	指定要在其中创建负载均衡器的子网。可以使用子网 ID 或名称。	`subnet-xxxx, subnet-yyyy`
`aws-load-balancer-ssl-cert`	来自 Amazon Certication Manager 的适用于 HTTPS/TLS 的 SSL 证书 ARN。	`arn:aws-cn:acm:region:account:certificate/cert-id`

字段	说明	示例
aws-load-balancer-ssl-ports	指定应使用 SSL/TLS 的端口。	443, 8443
load-balancer-source-ranges	被允许访问该负载均衡器的 CIDR 范围。	10.0.0.0/24, 192.168.1.0/24
aws-load-balancer-additional-resource-tags	要应用于负载均衡器和相关资源的其他 Amazon 标签。	Environment=prod,Team=platform
aws-load-balancer-ip-address-type	指定负载均衡器是使用 IPv4 还是双栈（IPv4 + IPv6）地址。	ipv4 或 dualstack

注意事项

- 您必须更新集群 IAM 角色才能支持将标签从 Kubernetes 传播到 Amazon 负载均衡器资源。有关更多信息，请参阅 the section called “EKS 自动模式资源的自定义 Amazon 标签”。
- 有关将资源关联到 EKS 自动模式或自主管理型 Amazon 负载均衡器控制器的信息，请参阅the section called “迁移参考”。
- 有关修复负载均衡器问题的信息，请参阅 the section called “故障排除”。
- 有关使用 EKS 自动模式的负载均衡功能时的其他注意事项，请参阅the section called “负载均衡”。

将负载均衡模式迁移到 EKS 自动模式进时，需要对服务注释和资源配置进行一些更改。下表总结了旧版和新版实现之间的主要区别，包括不支持的选项和建议的替代方案。

服务注释

旧版	New	说明
service.beta.kubernetes.io/load-balancer-source-ranges	不支持	在服务上使用 spec.loadBalancerSourceRanges

旧版	New	说明
service.beta.kubernetes.io/aws-load-balancer-type	不支持	在服务上使用 spec.loadBalancerClass
service.beta.kubernetes.io/aws-load-balancer-internal	不支持	使用 service.beta.kubernetes.io/aws-load-balancer-scheme
service.beta.kubernetes.io/aws-load-balancer-proxy-protocol	不支持	请改用 service.beta.kubernetes.io/aws-load-balancer-target-group-attributes
各种负载均衡器属性	不支持	使用 service.beta.kubernetes.io/aws-load-balancer-attributes
service.beta.kubernetes.io/aws-load-balancer-access-log-enabled	不支持	请改用 service.beta.kubernetes.io/aws-load-balancer-attributes
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name	不支持	请改用 service.beta.kubernetes.io/aws-load-balancer-attributes
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix	不支持	请改用 service.beta.kubernetes.io/aws-load-balancer-attributes

旧版	New	说明
service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled	不支持	请改用 service.beta.kubernetes.io/aws-load-balancer-attributes

要迁移已弃用的负载均衡器属性注释，请将这些设置合并到 service.beta.kubernetes.io/aws-load-balancer-attributes 注释中。此注释接受以逗号分隔的负载均衡器属性键值对列表。例如，要指定访问日志记录和跨区负载均衡，请使用以下格式：

```
service.beta.kubernetes.io/aws-load-balancer-attributes:
 access_logs.s3.enabled=true,access_logs.s3.bucket=my-bucket,access_logs.s3.prefix=my-prefix,load_balancing.cross_zone.enabled=true
```

这种合并后的格式让您可以更加一致、灵活地配置负载均衡器属性，同时减少所需的单独注释数量。检查您现有的服务配置并进行更新以使用此合并后的格式。

TargetGroupBinding

旧版	New	说明
elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1	API 版本更改
spec.targetType 可选	spec.targetType 必需	显式目标类型规范
spec.networking.ingress.from	不支持	不再支持没有安全组的 NLB

注意：要使用自定义 TargetGroupBinding 功能，您必须使用带有集群名称的 eks:eks-cluster-name 标签标记目标组，从而向控制器授予必要的 IAM 权限。请注意，当 TargetGroupBinding 资源或集群被删除时，控制器将删除目标组。

创建存储类

Amazon EKS 自动模式中的 StorageClass 定义了应用程序请求持久性存储时如何自动预置 Amazon EBS 卷。本页介绍如何创建和配置可与 Amazon EKS 自动模式配合使用的 StorageClass，从而预置 EBS 卷。

通过配置 StorageClass，您可以指定 EBS 卷的默认设置，包括卷类型、加密、IOPS 和其他存储参数。您还可以将 StorageClass 配置为使用 Amazon KMS 密钥进行加密管理。

EKS 自动模式不会为您创建 StorageClass。您必须创建引用 ebs.csi.eks.amazonaws.com 的 StorageClass，才能使用 EKS 自动模式的存储功能。

首先创建一个名为 storage-class.yaml 的文件：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
allowedTopologies:
- matchLabelExpressions:
  - key: eks.amazonaws.com/compute-type
    values:
    - auto
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

然后将存储类应用于到集群。

```
kubectl apply -f storage-class.yaml
```

关键组件：

- provisioner: ebs.csi.eks.amazonaws.com – 使用 EKS 自动模式
- allowedTopologies：指定与 eks.amazonaws.com/compute-type:auto 匹配的 matchLabelExpressions 将确保，如果容器组（pod）需要使用自动模式自动预置卷，则不会在非自动节点上调度容器组（pod）。

- `volumeBindingMode: WaitForFirstConsumer` – 将卷的创建延迟到容器组需要时
- `type: gp3` – 指定 EBS 卷类型
- `encrypted: "true"` – EBS 将对使用该 `StorageClass` 创建的所有卷进行加密。EBS 将使用默认的 `aws/ebs` 密钥别名。有关更多信息，请参阅《Amazon EBS 用户指南》中的 How Amazon EBS encryption works。该值是可选的，但建议使用。
- `storageclass.kubernetes.io/is-default-class: "true"` – 默认情况下 Kubernetes 将使用此存储类，除非您在持久性卷声明中指定了其他卷类。该值为可选项。如果要从其他存储控制器迁移，请谨慎设置此值。

使用自主管理型 KMS 密钥加密 EBS 卷

要使用自主管理型 KMS 密钥加密由 EKS 自动模式自动预置的 EBS 卷，您需要完成以下操作：

1. 创建自主管理型 KMS 密钥。
   - 有关更多信息，请参阅《KMS 用户指南》中的 Create a symmetric encryption KMS key 或 How Amazon Elastic Block Store (Amazon EBS) uses KMS。
2. 创建一个允许访问该 KMS 密钥的新策略。
   - 使用以下 IAM 策略示例来创建策略。插入新的自主管理型 KMS 密钥的 ARN。有关更多信息，请参阅《Amazon IAM 用户指南》中的创建角色并附加策略（控制台）。
3. 将此策略附加到 EKS 集群角色。
   - 使用 Amazon 控制台查找 EKS 集群角色的 ARN。角色信息将在概述部分中显示。有关更多信息，请参阅 the section called “集群 IAM 角色”。
4. 更新 `StorageClass` 以在 `parameters.kmsKeyId` 字段中引用该 KMS 密钥 ID。

自主管理型 KMS IAM 策略示例

更新以下策略中的以下值：

- `<account-id>` – Amazon 账户 ID，例如 `111122223333`
- `<aws-region>` – 集群所在的 Amazon 区域，例如 `us-west-2`

```
{
  "Version":"2012-10-17",
  "Id": "key-auto-policy-3",
  "Statement": [
```

```
      {
          "Sid": "Enable IAM User Permissions",
          "Effect": "Allow",
          "Principal": {
              "AWS": "arn:aws:iam::123456789012:root"
          },
          "Action": "kms:*",
          "Resource": "*"
      },
      {
        "Sid": "Allow access through EBS for all principals in the account that are
 authorized to use EBS",
        "Effect": "Allow",
        "Principal": {
            "AWS": "*"
        },
        "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:CreateGrant",
            "kms:DescribeKey"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:CallerAccount": "123456789012",
                "kms:ViaService": "ec2.us-east-1.amazonaws.com"
            }
        }
    }
  ]
}
```

自主管理型 KMS **StorageClass** 示例

```
parameters:
  type: gp3
  encrypted: "true"
  kmsKeyId: <custom-key-arn>
```

StorageClass 参数参考

有关 Kubernetes StorageClass 资源的一般信息，请参阅 Kubernetes 文档中的存储类。

StorageClass 资源的 parameters 部分特定于 Amazon。请使用下表来查看可用的选项。

Parameters	值	默认值	说明
"csi.storage.k8s.io/fstype"	xfs、ext2、ext3、ext4	ext4	将在创建卷时进行格式化的文件系统类型。此参数区分大小写！
"type"	io1、io2、gp2、gp3、sc1、st1、标准、sbp1、sbg1	gp3	EBS 卷类型。
"iopsPerGB"			每 GiB 每秒 I/O 读写操作次数。可以为 IO1、IO2 和 GP3 卷指定。
"allowAutoIOPSPerGBIncrease"	true，false	false	如果为 "true"，CSI 驱动程序将在 iopsPerGB * <volume size> 太小而无法达到 Amazon 支持的 IOPS 范围时增加卷的 IOPS。这可确保动态预置始终会成功，即使用户指定的 PVC 容量或 iopsPerGB 值太小。不过这可能会带来额外的成本，因为此类卷的 IOPS 高于 iopsPerGB 中请求的值。

Parameters	值	默认值	说明
"iops"			每秒 I/O 读写操作次数。可以为 IO1、IO2 和 GP3 卷指定。
"throughput"		125	吞吐量以 MiB/s 为单位。仅在指定 gp3 卷类型时有效。
"encrypted"	true，false	false	是否应对卷进行加密。有效值为“true”或“false”。
"blockExpress"	true，false	false	允许创建 io2 Block Express 卷。
"kmsKeyId"			加密卷时要使用的密钥的完整 ARN。如果未指定，Amazon 将使用该卷所在区域的默认 KMS 密钥。这将是一个自动生成的密钥，名为 `/aws/ebs`（如果未更改）。
"blockSize"			格式化底层文件系统时要使用的块大小。仅支持 linux 节点和文件系统类型 `ext2`、`ext3`、`ext4` 或 `xfs`。

Parameters	值	默认值	说明
"inodeSize"			格式化底层文件系统时要使用的索引节点大小。仅支持 linux 节点和文件系统类型 ext2、ext3、ext4 或 xfs。
"bytesPerInode"			格式化底层文件系统时要使用的 bytes-per-inode 。仅支持 linux 节点和文件系统类型 ext2、ext3 或 ext4。
"numberOfInodes"			格式化底层文件系统时要使用的 number-of-inodes 。仅支持 linux 节点和文件系统类型 ext2、ext3 或 ext4。
"ext4BigAlloc"	true，false	false	通过启用 bigalloc 格式化选项，将 ext4 文件系统更改为使用集群块分配。警告：对于节点的 Linux 内核，bigalloc 可能并非完全受支持。

Parameters	值	默认值	说明
"ext4ClusterSize"			启用 bigalloc 功能后格式化 ext4 文件系统时要使用的集群大小。注意：ext4BigAlloc 参数必须设置为 true。

有关更多信息，请参阅 GitHub 上的 Amazon EBS CSI Driver。

注意事项

> **Note**
>
> 您只能根据 EKS 自动模式节点上的 EKS 自动模式 StorageClasses 部署工作负载。如果您的集群包含混合类型的节点，则需要将工作负载配置为仅在 EKS 自动模式节点上运行。有关更多信息，请参阅 the section called “控制部署”。

EKS 自动模式的块存储功能与 EBS CSI 驱动程序不同。

- 静态预置
  - 要将外部创建的 EBS 卷与 EKS 自动模式结合使用，需要手动添加键为 eks:eks-cluster-name，值为集群名称的 Amazon 标签。
- 节点启动污点
  - 在存储容量准备就绪之前，不能使用节点启动污点功能来阻止容器组调度
- 动态预置卷上的自定义标签
  - 不能使用 extra-tag CLI 标志在动态预置 EBS 卷上配置自定义标签
  - 可以使用 StorageClass 标记功能来添加自定义标签。EKS 自动模式将为关联的 Amazon 资源添加标签。您需要更新自定义标签的集群 IAM 角色。有关更多信息，请参阅 the section called “EKS 自动模式资源的自定义 Amazon 标签”。
- EBS 详细性能指标
  - 您无法访问 Prometheus 指标来了解 EBS 的详细性能

安装 CSI 快照控制器附加组件

EKS 自动模式与 CSI 快照控制器 Amazon EKS 附加组件兼容。

Amazon 建议将此附加组件配置为在内置的 `system` 节点池上运行。

有关更多信息，请参阅：

- the section called “运行关键附加组件”
- the section called “检查内置节点池”
- the section called “CSI 快照控制器”

在系统节点池中安装快照控制器

1. 在 Amazon 控制台中打开 EKS 集群
2. 在附加组件选项卡中，选择获取更多附加组件
3. 选择 CSI 快照控制器，然后选择下一步
4. 在配置选定附加组件的设置页面上，选择可选配置设置以查看附加组件配置架构
   a. 插入以下 yaml 以将快照控制器关联到 `system` 节点池。快照控制器包含对 `CriticalAddonsOnly` 污点的容忍度。

```
{
        "nodeSelector": {
            "karpenter.sh/nodepool": "system"
        }
}
```

   b. 选择下一步
5. 检查附加组件配置，然后选择创建

禁用 EKS 自动模式

您可以在现有 EKS 集群上禁用 EKS 自动模式。这是一种破坏性的操作。

- EKS 将终止所有由 EKS 自动模式运行的 EC2 实例。
- EKS 将删除所有由 EKS 自动模式运行的负载均衡器。
- EKS 不会删除由 EKS 自动模式预置的 EBS 卷。

EKS 自动模式设计为完全托管其创建的资源。手动干预可能会导致在禁用 EKS 自动模式后无法完全清理这些资源。例如，假设您从外部安全组规则中引用某个托管式安全组，但在为集群禁用 EKS 自动模式之前忘记移除该引用，则托管式安全组将会泄露（不会被删除）。以下步骤介绍了在发生安全组泄露时如何移除泄露的安全组。

禁用 EKS 自动模式（Amazon 控制台）

1. 在 Amazon Web Services 管理控制台中打开集群概述页面。
2. 在 EKS 自动模式下，选择管理
3. 将 EKS 自动模式切换为 `off`。

如果在此过程结束后未删除任何托管式安全组，则可以使用删除安全组中的说明手动将其删除。

禁用 EKS 自动模式（Amazon CLI）

使用以下命令在现有集群上禁用 EKS 自动模式。

您需要安装 aws CLI，并以足够的 EKS 集群管理权限登录。有关更多信息，请参阅 设置。

> **Note**
>
> 计算、块存储和负载均衡功能必须在同一请求中全部启用或禁用。

```
aws eks update-cluster-config \
 --name $CLUSTER_NAME \
 --compute-config enabled=false \
 --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": false}}' \
 --storage-config '{"blockStorage":{"enabled": false}}'
```

禁用 EKS 自动模式后，您可以如下所示检查泄露的 EKS 自动模式安全组是否未能删除：

```
aws ec2 describe-security-groups \
    --filters Name=tag:eks:eks-cluster-name,Values=<cluster-Name> Name=tag-
key,Values=ingress.eks.amazonaws.com/resource,service.eks.amazonaws.com/resource --
query "SecurityGroups[*].[GroupName]"
```

然后删除安全组：

```
aws ec2 delete-security-group --group-name=<sg-name>
```

更新 EKS 自动模式集群的 Kubernetes 版本

本主题介绍如何更新自动模式集群的 Kubernetes 版本。自动模式通过处理控制面板更新与节点替换之间的协调来简化版本更新过程，同时通过容器组中断预算来保持工作负载的可用性。

在升级自动模式集群时，许多传统上需要手动更新的组件现在都作为服务的一部分进行管理。了解升级过程的自动化方面以及您需要负责的方面，有助于确保集群顺利完成版本过渡。

了解使用 EKS 自动模式时的更新

启动控制面板升级后，EKS 自动模式将升级集群中的节点。节点过期后，EKS 自动模式会将其替换为新节点。新节点具有相应的新 Kubernetes 版本。EKS 自动模式会在升级节点时观察容器组中断预算。

此外，您不再需要更新组件，例如：

- Amazon VPC CNI
- Amazon Load Balancer Controller
- CoreDNS
- `kube-proxy`
- Karpenter
- Amazon EBS CSI 驱动程序

EKS 自动模式用服务功能取代了这些组件。

您仍需要负责更新下列方面：

- 部署到集群的应用程序和工作负载
- 自主管理型附加组件和控制器
- Amazon EKS 附加组件
  - 了解如何the section called “更新附加组件”

了解集群升级最佳实践

启动集群更新

要开始集群更新，请参阅the section called “更新 Kubernetes 版本”。

启用或禁用内置节点池

EKS 自动模式有两个内置节点池。您可以使用 Amazon 控制台、CLI 或 API 启用或禁用这些节点池。

内置节点池参考

- `system`
  - 此节点池有一个 `CriticalAddonsOnly` 污点。许多 EKS 附加组件（例如 CoreDNS）都容忍此污点。使用此系统节点池来隔离集群关键型应用程序。
  - 同时支持 `amd64` 和 `arm64` 架构。
- `general-purpose`
  - 此节点池提供了为集群中的通用型工作负载启动节点的支持。
  - 仅使用 `amd64` 架构。

同时启用两个内置节点池：

- 使用默认的 EKS 节点类
- 仅使用按需型 EC2 容量
- 使用 C、M 和 R EC2 实例系列
- 需要第 5 代或更高版本的 EC2 实例

> **Note**
>
> EKS 需要启用至少一个内置节点池才能配置“默认”节点类。如果禁用所有内置节点池，则需要创建自定义节点类并配置节点池才能使用它。有关节点类的更多信息，请参阅the section called “创建节点类”。

过程

先决条件

- 已在您的设备上安装并配置了最新版本的 Amazon 命令行界面（Amazon CLI）。要查看当前版本，请使用 `aws --version`。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。

- 使用创建 Amazon 资源所需的充分 IAM 权限登录 CLI，包括 IAM 策略、IAM 角色和 EKS 集群等。

使用 Amazon CLI 启用

使用以下命令同时启用两个内置节点池：

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
    "nodeRoleArn": "<node-role-arn>",
    "nodePools": ["general-purpose", "system"],
    "enabled": true
  }' \
  --kubernetes-network-config '{
  "elasticLoadBalancing":{"enabled": true}
  }' \
  --storage-config '{
  "blockStorage":{"enabled": true}
  }'
```

您可以修改命令以有选择地启用节点池。

使用 Amazon CLI 禁用

使用以下命令同时禁用两个内置节点池：

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
  "enabled": true,
  "nodePools": []
  }' \
  --kubernetes-network-config '{
  "elasticLoadBalancing":{"enabled": true}}' \
  --storage-config '{
  "blockStorage":{"enabled": true}
  }'
```

控制工作负载是否部署在 EKS 自动模式节点上

在启用 EKS 自动模式的 EKS 集群中运行工作负载时，您可能需要控制特定工作负载是在 EKS 自动模式节点上运行还是其他计算类型上运行。本主题介绍如何使用节点选择器和亲和性规则，来确保您的工作负载在需要的计算基础设施上调度。

本主题中的示例演示如何使用 `eks.amazonaws.com/compute-type` 标签要求或阻止在 EKS 自动模式节点上部署工作负载。这在同时运行 EKS 自动模式和其他计算类型（例如自主管理型 Karpenter 预置器或 EKS 托管式节点组）的混合模式集群中特别有用。

EKS 自动模式节点已将标签 `eks.amazonaws.com/compute-type` 的值设置为 `auto`。您可以使用此标签来控制是否将工作负载部署到由 EKS 自动模式管理的节点。

要求将工作负载部署到 EKS 自动模式节点

> **Note**
>
> 此 `nodeSelector` 值对于 EKS 自动模式并非必需。此 `nodeSelector` 值仅当您在混合模式下运行集群，节点类型不由 EKS 自动模式管理时才重要。例如，您可以使用 EKS 托管式节点组将静态计算容量部署到集群，并使用由 EKS 自动模式管理的动态计算容量。

您可以将此 `nodeSelector` 添加到部署或其他工作负载中，以要求 Kubernetes 将其调度到 EKS 自动模式节点上。

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    nodeSelector:
      eks.amazonaws.com/compute-type: auto
```

要求不将工作负载部署到 EKS 自动模式节点

您可以将此 `nodeAffinity` 添加到部署或其他工作负载中，以要求 Kubernetes 不将其调度到 EKS 自动模式节点上。

```
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
```

```
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - auto
```

在专用实例上运行关键附加组件

在本主题中，您将了解如何部署具有 CriticalAddonsOnly 容忍度的工作负载，以便 EKS 自动模式将其调度到 system 节点池中。

EKS 自动模式内置的 system 节点池专为在专用实例上运行关键附加组件而设计。这种隔离可确保关键组件拥有专用资源并与一般工作负载隔离，从而增强集群的整体稳定性和性能。

本指南演示了如何利用 CriticalAddonsOnly 容忍度和适当的节点选择器将附加组件部署到 system 节点池。通过执行这些步骤，您可以利用 EKS 自动模式专用节点池结构提供的隔离和资源分配优势，确保将关键应用程序调度到专用的 system 节点上。

EKS 自动模式有以下两个内置节点池：general-purpose 和 system。有关更多信息，请参阅 the section called “检查内置节点池”。

system 节点池的作用是将关键附加组件隔离到不同的节点上。system 节点池预置的节点有一个 CriticalAddonsOnly Kubernetes 污点。Kubernetes 仅在具有对应的容忍度时才会将容器组调度到这些节点上。有关更多信息，请参阅 Kubernetes 文档中的污点和容忍度。

先决条件

- 启用内置 system 节点池的 EKS 自动模式集群。有关更多信息，请参阅 the section called “检查内置节点池”
- 安装并配置了 kubectl。有关更多信息，请参阅 设置。

过程

参阅下面的示例 yaml。请注意以下配置：

- nodeSelector – 这会将工作负载关联到内置的 system 节点池。必须使用 Amazon API 启用此节点池。有关更多信息，请参阅 the section called “检查内置节点池”。
- tolerations – 此容忍度会克服 system 节点池中节点上的 CriticalAddonsOnly 污点。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      nodeSelector:
        karpenter.sh/nodepool: system
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
      containers:
      - name: app
        image: nginx:latest
        resources:
          requests:
            cpu: "500m"
            memory: "512Mi"
```

要更新要在 system 节点池上运行的工作负载，您需要完成以下操作：

1. 更新现有工作负载以添加上文描述的以下配置：
   - nodeSelector
   - tolerations
2. 使用 kubectl apply 将更新后的工作负载部署到集群

工作负载更新后将在专用节点上运行。

将网络策略与 EKS 自动模式结合使用

概述

随着客户使用 EKS 扩展其应用程序环境，网络流量隔离对于防止未经授权访问集群内外的资源变得越来越重要。对于在集群中同时运行多个不相关工作负载的多租户环境，这一点尤其重要。Kubernetes 网络策略能帮助您改进 Kubernetes 工作负载的网络安全状况，以及它们与集群外部端点的集成。EKS 自动模式支持不同类型的网络策略。

第 3 层和第 4 层隔离

标准 Kubernetes 网络策略在 OSI 网络模型的第 3 层和第 4 层上运行，让您能够控制 Amazon EKS 集群中 IP 地址或端口级别的流量。

使用案例

- 在工作负载之间对网络流量进行分段，确保只有相关的应用程序才能相互通信。
- 使用策略在命名空间级别隔离租户，以强制执行网络分离。

基于 DNS 的执行

客户通常在 EKS 中部署属于更广泛的分布式环境中一部分的工作负载，其中一些工作负载必须与集群外的系统和服务通信（北向流量）。这些系统和服务可以位于 Amazon 云中，也可以完全位于 Amazon 外部。基于域名系统（DNS）的策略允许您采用更稳定、更可预测的方法来防止容器组（pod）未经授权访问集群外部资源或端点，从而改进您的安全状况。这种机制无需手动跟踪并将特定 IP 地址加入允许列表。通过使用基于 DNS 的方法保护资源，您还可以更灵活地更新外部基础设施，而不必在上游服务器和主机发生变化时放松安全状况或修改网络策略。您可以使用完全限定域名（FQDN）或 DNS 域名的匹配模式筛选传送至外部端点的出口流量。这样您便可以更加灵活地将访问权限扩展到与特定集群外部端点关联的多个子域。

使用案例

- 使用基于 DNS 的方法进行标准化，用于筛选从 Kubernetes 环境到集群外部端点的访问权限。
- 在多租户环境中安全访问 Amazon 服务。
- 在混合云环境中管理从容器组（pod）到本地工作负载的网络访问权限。

管理员（或集群范围）规则

在某些情况下，例如多租户场景，客户可能需要强制执行适用于整个集群的网络安全标准。无需为每个命名空间重复定义和维护不同的策略，而是可以使用单个策略来集中管理集群中不同工作负载的网络访问权限控制，无论其命名空间如何。这些类型的策略允许您扩大在第 3 层、第 4 层以及使用 DNS 规则时应用的网络筛选规则的实施范围。

使用案例

- 集中管理 EKS 集群中所有（或部分）工作负载的网络访问权限控制。
- 定义整个集群的默认网络安全状况。
- 以更具运营效率的方式，将组织安全标准扩展到集群范围。

开始使用

先决条件

- 启用了 EKS 自动模式的 Amazon EKS 集群
- 配置好 kubectl 以连接到集群

第 1 步：启用网络策略控制器

要将网络策略与 EKS 自动模式结合使用，您首先需要通过将 ConfigMap 应用到集群来启用网络策略控制器。

1. 使用以下内容创建名为 `enable-network-policy.yaml` 的文件：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy-controller: "true"
```

2. 将 ConfigMap 应用到集群：

```
kubectl apply -f enable-network-policy.yaml
```

第 2 步：在节点类中启用网络策略

您需要首先确保节点类已配置为支持网络策略，然后才能使用这些策略。按照以下步骤进行操作：

1. 创建或编辑包含以下内容的节点类 YAML 文件（例如 `nodeclass-network-policy.yaml`）：

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: network-policy-enabled
spec:
  # Enables network policy support
  networkPolicy: DefaultAllow
  # Optional: Enables logging for network policy events
  networkPolicyEventLogs: Enabled
  # Include other Node Class configurations as needed
```

2. 将节点类配置应用到集群：

```
kubectl apply -f nodeclass-network-policy.yaml
```

3. 验证节点类是否已创建：

```
kubectl get nodeclass network-policy-enabled
```

4. 更新节点池以使用此节点类。有关更多信息，请参阅 the section called “创建节点池”。

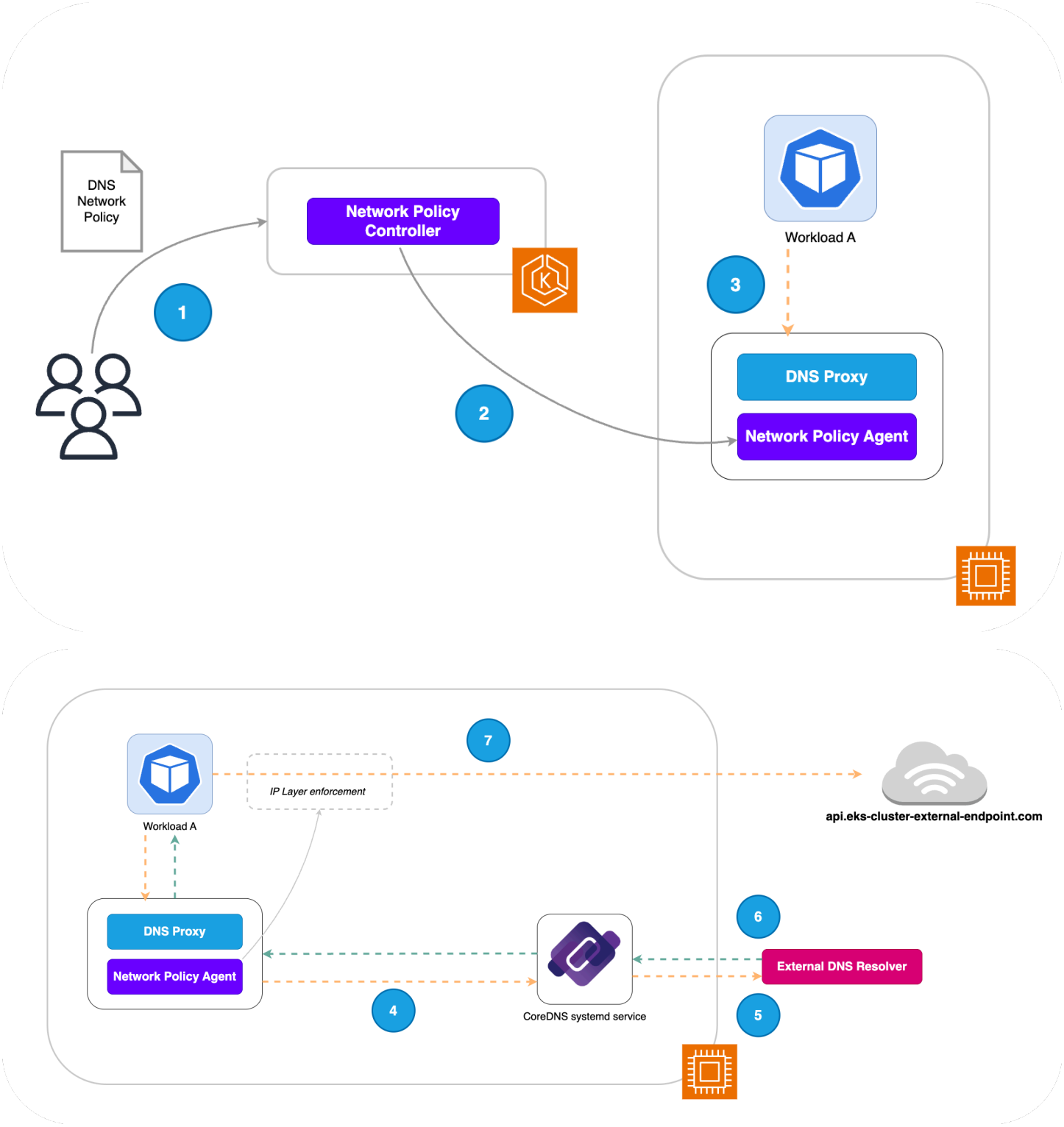
只要您的节点使用此节点类，既可以强制实施网络策略。现在，您可以开始创建和应用网络策略来控制集群内的流量。有关所有节点类配置选项，请参阅the section called “创建节点类”。

第 3 步：创建和测试网络策略

您的 EKS 自动模式集群现已配置为支持 Kubernetes 网络策略。您可以使用 the section called “Stars 策略演示” 进行测试。

如何工作？

基于 DNS 的网络策略



1. 平台团队将基于 DNS 的策略应用于 EKS 集群。

2. 网络策略控制器负责监控集群内策略的创建情况，然后协调策略端点。在此使用案例中，网络策略控制器指示节点代理根据创建的策略中的允许列表域筛选 DNS 请求。使用 FQDN 或与 Kubernetes 资源配置中定义的模式相匹配的域名，将域名列入允许名单。
3. 工作负载 A 尝试解析集群外部端点的 IP。DNS 请求首先通过代理，该代理根据通过网络策略应用的允许列表筛选此类请求。
4. 一旦 DNS 请求通过 DNS 筛选器允许列表，就会被传输到 CoreDNS，
5. 反过来，CoreDNS 会将请求发送到外部 DNS 解析器（Amazon Route 53 Resolver），获取域名背后的 IP 地址列表。
6. 已解析的 IP 连同 TTL 都将在对 DNS 请求的响应中返回。然后，将这些 IP 写入 eBPF 映射中，该映射用于 IP 层执行的下一步。
7. 随后，连接到容器组（pod）veth 接口的 eBPF 探测器将根据现有规则筛选从工作负载 A 到集群外部端点的出口流量。这可确保容器组（pod）只能向已加入允许列表的域 IP 发送集群外部流量。这些 IP 的有效性取决于从外部 DNS 解析器（Amazon Route 53 Resolver）检索到的 TTL。

使用应用程序网络策略

`ApplicationNetworkPolicy` 使用单个自定义资源定义（CRD）在命名空间级别将标准 Kubernetes 网络策略的功能与基于 DNS 的筛选相结合。因此，`ApplicationNetworkPolicy` 可以用于：

1. 使用 IP 块和端口号定义网络堆栈第 3 层和第 4 层的限制。
2. 定义网络堆栈第 7 层运行的规则，并允许您基于 FQDN 筛选流量。

重要说明：使用 `ApplicationNetworkPolicy` 定义的基于 DNS 的规则仅适用于在 EKS 自动模式启动的 EC2 实例中运行的工作负载。

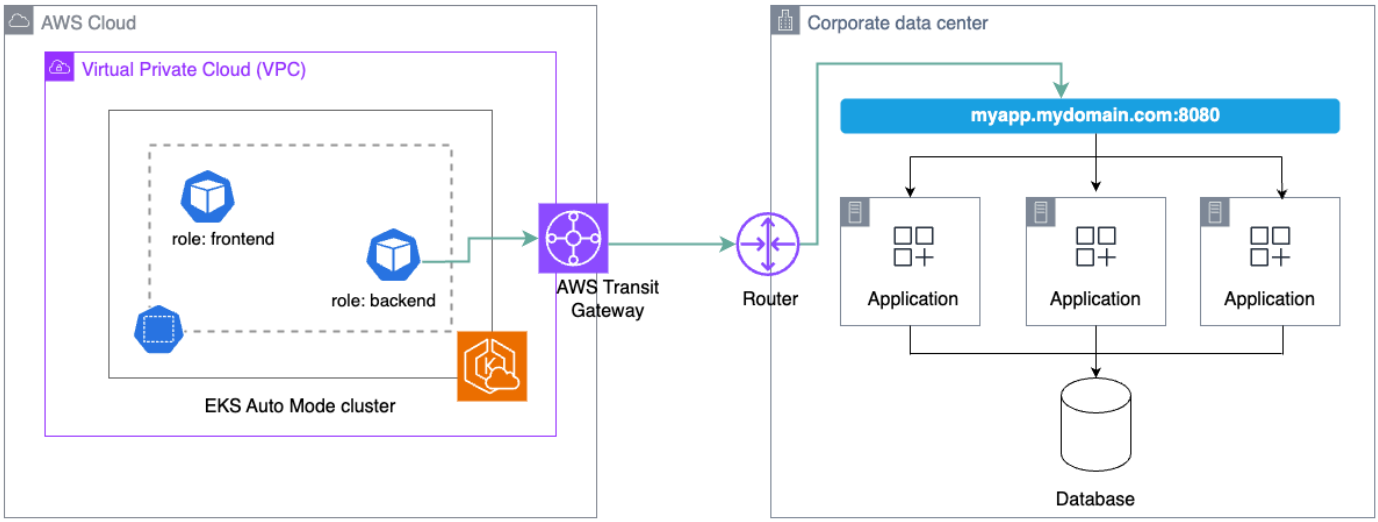
示例

您的 EKS 自动模式集群中有一个工作负载，需要与带有 DNS 名称的负载均衡器背后的本地应用程序通信。您可以使用以下网络策略实现此目的：

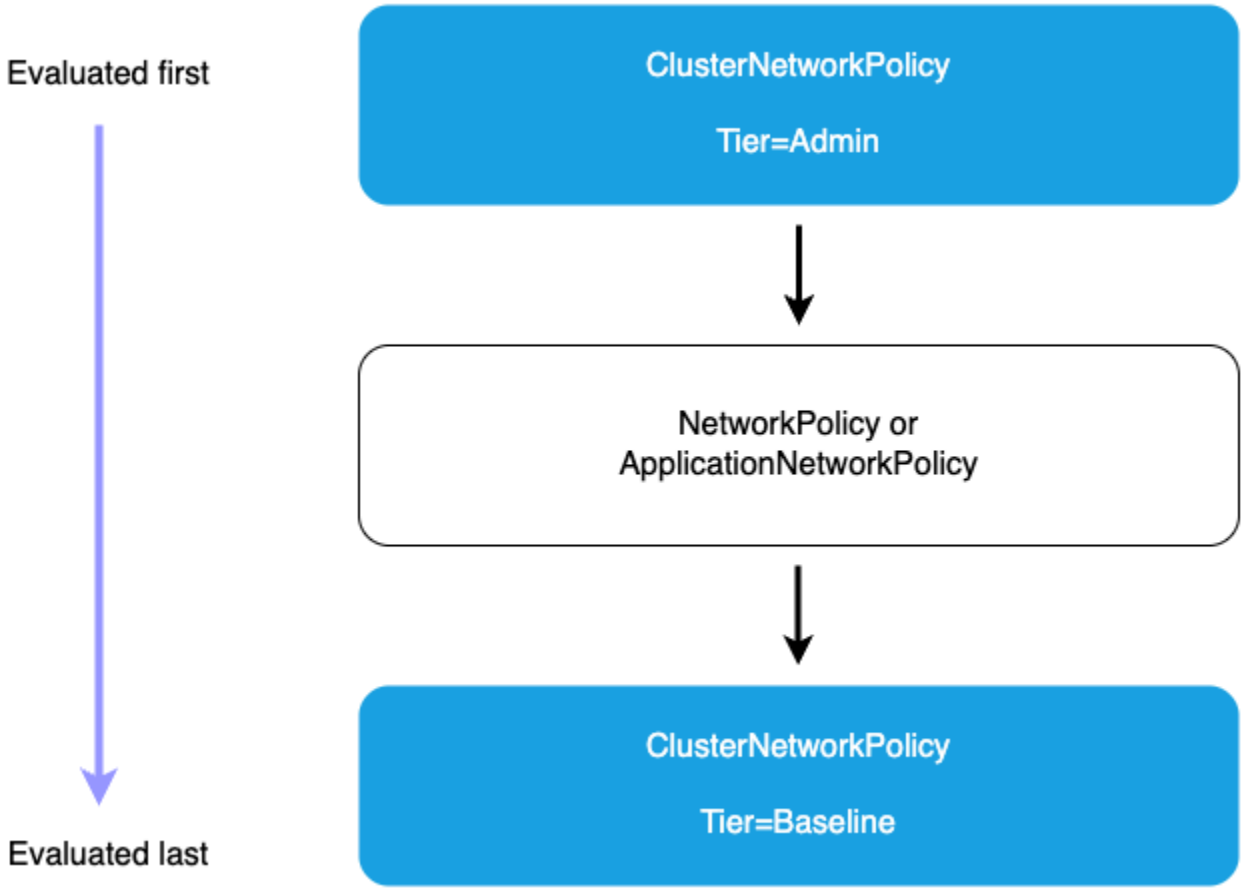
```
apiVersion: networking.k8s.aws/v1alpha1
kind: ApplicationNetworkPolicy
metadata:
  name: my-onprem-app-egress
  namespace: galaxy
```

```
spec:
  podSelector:
    matchLabels:
      role: backend
  policyTypes:
  - Egress
  egress:
  - to:
    - domainNames:
      - "myapp.mydomain.com"
    ports:
    - protocol: TCP
      port: 8080
```

在 Kubernetes 网络级别，这将允许来自标有 `role: backend` 的“galaxy”命名空间中的任何容器组（pod）的出口，通过 TCP 端口 8080 连接到域名 myapp.mydomain.com。此外，您还需要为从 VPC 到公司数据中心的出口流量设置网络连接。



管理员（或集群）网络策略



使用集群网络策略

使用 ClusterNetworkPolicy 时，将首先评估管理员层策略，并且无法被覆盖。在评估管理员层策略后，将使用标准命名空间范围策略来执行应用的网络分段规则。这可以通过使用 ApplicationNetworkPolicy 或 NetworkPolicy 完成。最后，将强制执行定义了集群工作负载默认网络限制的基准层规则。如有需要，这些基准层规则可以被命名空间范围的策略所覆盖。

示例

您的集群中有一个要与其他租户工作负载隔离的应用程序。您可以明确阻止来自其他命名空间的集群流量，防止网络访问敏感的工作负载命名空间。

```
apiVersion: networking.k8s.aws/v1alpha1
kind: ClusterNetworkPolicy
metadata:
  name: protect-sensitive-workload
spec:
```

```
  tier: Admin
  priority: 10
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: earth
  ingress:
    - action: Deny
      from:
      - namespaces:
          matchLabels: {} # Match all namespaces.
      name: select-all-deny-all
```

注意事项

了解策略评估顺序

EKS 中支持的网络策略功能按特定顺序进行评估，以确保流量管理可预测且安全。因此，了解评估流程是非常重要的，这样才能为您的环境设计有效的网络安全状况。

1. 管理员层策略（先进行评估）：首先对所有管理员层 ClusterNetworkPolicies 进行评估，然后再评估任何其他策略。在管理员层中，策略按优先级顺序处理（优先级序号最小者优先）。操作类型决定了接下来的操作。
   - Deny 操作（优先级最高）：当具有 Deny 操作的管理员策略与流量匹配时，无论采用何种其他策略，该流量都会立即被阻止。不再处理其他 ClusterNetworkPolicy 或 NetworkPolicy 规则。这样可以确保组织范围的安全控制不会被命名空间级别的策略所覆盖。
   - Allow 操作：评估 Deny 规则后，将按优先级顺序处理具有 Allow 操作的管理员策略（优先级序号最小者优先）。当 Allow 操作匹配时，流量将被接受，并且不再进行进一步的策略评估。这些策略可以根据标签选择器授予跨多个命名空间的访问权限，从而集中控制哪些工作负载可以访问特定资源。
   - Pass 操作：管理员层策略中的 Pass 操作将决策委托给较低层级。当流量与 Pass 规则匹配时，评估将跳过该流量的所有剩余管理层规则，直接进入 NetworkPolicy 层。这允许管理员将某些流量模式的控制权明确委托给应用程序团队。例如，您可以使用 Pass 规则将命名空间内的流量管理委托给命名空间管理员，同时保持对外部访问权限的严格控制。
2. 网络策略层：如果没有与 Deny 或 Allow 匹配的管理员层策略，或者匹配了 Pass 操作，则接下来将评估命名空间范围的 ApplicationNetworkPolicy 和传统 NetworkPolicy 资源。这些策略在各个命名空间内提供精细的控制，并由应用程序团队管理。命名空间范围的策略只能比管理员策略更严格。它们无法覆盖管理员策略的 Deny 决策，但可以进一步限制管理员策略允许或通过的流量。

3. 基准层管理员策略：如果没有管理员或命名空间范围的策略与流量匹配，则会评估基准层 ClusterNetworkPolicies。它们提供了可以被命名空间范围的策略覆盖的默认安全状况，允许管理员设置组织范围的默认值，同时让团队可以灵活地根据需要进行自定义。基准策略按优先级顺序进行评估（优先级序号最小者优先）。
4. 默认拒绝（如果没有匹配的策略）：这种默认拒绝行为可确保仅允许明确允许的连接，从而保持稳固的安全状况。

应用最低权限原则

- 从限制性策略开始，然后根据需要逐步添加权限：首先在集群级别实施默认拒绝策略，然后在验证合法的连接要求时，逐步添加允许规则。这种方法迫使团队明确证明每个外部连接的合理性，从而打造更加安全且可审计的环境。
- 定期审计并删除未使用的策略规则：随着应用程序的发展，网络策略可能会随着时间的推移而积累，所留下的过时规则会不必要地扩大攻击面。实施定期审查流程，以确定并删除不再需要的策略规则，确保您的安全状况保持严密且可维护。
- 尽可能使用特定的域名而不是宽泛的模式：虽然像 `*.amazonaws.com` 这类通配符模式提供了便利，但这也会授予对各种服务的访问权限。只要可行，请指定如 `s3.us-west-2.amazonaws.com` 等确切的域名，仅允许访问应用程序所需的特定服务，从而降低工作负载遭盗用时横向移动的风险。

在 EKS 中使用基于 DNS 的策略

- 使用 `ApplicationNetworkPolicy` 定义的基于 DNS 的规则仅适用于在 EKS 自动模式启动的 EC2 实例中运行的工作负载。如果您运行的是混合模式集群（由 EKS 自动模式和非 EKS 自动模式 Worker 节点组成），则基于 DNS 的规则仅在 EKS 自动模式 Worker 节点（EC2 托管实例）中有效。

验证 DNS 策略

- 使用与生产网络拓扑结构相同的暂存集群进行测试：您的暂存环境应复制生产的网络架构、外部依赖关系和连接模式，确保进行准确的策略测试。这包括相同的 VPC 配置、DNS 解析行为以及访问生产工作负载所需的相同外部服务。
- 对关键网络路径实施自动测试：构建自动化测试，验证与基本外部服务的连接（作为 CI/CD 管道的一部分）。这些测试应验证在阻止未经授权的连接的同时是否允许合法的流量，从而持续验证随着基础设施的演变，您的网络策略是否仍能保持理想的安全状况。

- 在策略更改后监控应用程序行为：将新的或经修改的网络策略部署到生产环境后，密切监控应用程序日志、错误率和性能指标，以此快速识别任何连接问题。制定明确的回滚程序，以便在策略更改导致意外应用程序行为或服务中断时可以快速还原这些更改。

与 Amazon Route 53 DNS 防火墙互动

当流量启动时，首先在容器组（pod）级别评估 EKS 管理员和网络策略。如果 EKS 网络策略允许出口到特定域，则该容器组（pod）随后会执行 DNS 查询，其随后会到达 Route 53 解析器。此时，将对 Route 53 的 DNS 防火墙规则进行评估。如果 DNS 防火墙阻止域查询，那么即使 EKS 网络策略允许 DNS 解析失败，也无法建立连接。这创建了互补的安全层：基于 EKS DNS 的网络策略为应用程序特定的访问要求和多租户安全边界提供容器组（pod）级别的出口控制，而 DNS 防火墙则提供 VPC 范围内针对已知恶意域的保护，并强制执行组织范围的阻止名单。

为 EKS 自动模式的子网添加标签

如果您使用 EKS 自动模式的负载均衡功能，则需要向 VPC 子网添加 Amazon 标签。

背景

这些标签可将子网标识为与集群相关联，更重要的是能标识出子网是公有子网还是私有子网。

公有子网可以通过互联网网关直接访问互联网。此类子网用于需要可公开访问的资源，例如负载均衡器。

私有子网无法直接访问互联网，需使用 NAT 网关来传输出站流量。此类子网用于内部资源，例如不需要公有 IP 的 EKS 节点。

要了解有关 NAT 网关和互联网网关的更多信息，请参阅《Amazon Virtual Private Cloud（VPC）用户指南》中的将 VPC 连接到其他网络。

要求

目前，EKS 自动模式用于负载平衡的子网需具有以下某个标签。

公有子网

公有子网用于面向互联网的负载均衡器。这些子网必须具有以下标签：

键	值
kubernetes.io/role/elb	1 或 ``

私有子网

私有子网用于内部负载均衡器。这些子网必须具有以下标签：

键	值
kubernetes.io/role/internal-elb	1 或 ``

过程

在开始之前，请确定哪些是公有子网（具有互联网网关访问权限），哪些是私有子网（使用 NAT 网关）。必须具备权限才能修改 VPC 资源。

Amazon Web Services 管理控制台

1. 打开 Amazon VPC 控制台，然后导航至“子网”
2. 选择要添加标签的子网
3. 选择“标签”选项卡，然后选择“添加标签”
4. 添加相应标签：
   - 对于公有子网：Key=kubernetes.io/role/elb
   - 对于私有子网：Key=kubernetes.io/role/internal-elb
5. 将“值”设置为 1 或留空
6. 保存设置，再重复对其余子网执行前述操作

Amazon CLI

对于公有子网：

```
aws ec2 create-tags \
    --resources subnet-ID \
    --tags Key=kubernetes.io/role/elb,Value=1
```

对于私有子网：

```
aws ec2 create-tags \
    --resources subnet-ID \
    --tags Key=kubernetes.io/role/internal-elb,Value=1
```

将 subnet-ID 替换为您的实际子网 ID。

使用 kubectl debug 从 Kubernetes 节点生成 CIS 合规性报告

本主题介绍如何使用 kubectl debug 命令，为 Amazon EKS 节点生成 CIS（Center for Internet Security）合规性报告。该命令允许您在 Kubernetes 节点上临时创建调试容器，并使用 apiclient 工具运行 CIS 合规性检查。apiclient 工具是 Bottlerocket 操作系统（EKS 自动模式节点使用的操作系统）中的一部分。

先决条件

在开始之前，请确保您满足以下条件：

- 访问已配置 kubectl 的 Amazon EKS 集群（版本必须至少为 v1.32.0；键入 kubectl version 以执行检查）。
- 具有相应的节点调试 IAM 权限。
- 具有允许调试操作的有效配置文件（例如 sysadmin）。

有关使用包含 kubectl 的调试配置文件的更多信息，请参阅 Kubernetes 文档中的指定应用配置文件来调试容器组（pod）或节点。

过程

1. 确定要在其上运行报告的节点的 Amazon 实例 ID。使用以下命令列出该集群中的节点。实例 ID 位于名称列中，以 i- 开头：

```
kubectl get nodes
```

```
NAME                  STATUS   ROLES    AGE   VERSION
i-0ea0ba0f8ef9ad609   Ready    <none>   62s   v1.30.10-eks-1a9dacd
```

2. 运行以下命令，将 <instance-id> 替换为要查询的节点的实例 ID：

```
kubectl debug node/<instance-id> -it --profile=sysadmin --image=public.ecr.aws/
amazonlinux/amazonlinux:2023 -- bash -c "yum install -q -y util-linux-core; nsenter -
t 1 -m apiclient report cis --level 1 --format text"
```

该命令的组成部分包括：

- `kubectl debug node/<instance-id>`：在指定的 EC2 实例 ID 上创建调试会话。
- `-it`：分配 TTY（命令行 Shell）并使 stdin 保持打开状态供交互式使用。
- `--profile=sysadmin`：使用具有适当权限的指定 kubectl 配置文件。
- `--image=public.ecr.aws/amazonlinux/amazonlinux:2023`：使用 amazonlinux:2023 作为容器映像进行调试。
- `bash -c "…"` ：在 Bash Shell 中执行以下命令：
  - `yum install -q -y util-linux-core`：静默安装所需的实用工具包。
  - `nsenter -t 1 -m`：运行 nsenter 以进入主机进程（PID 1）的命名空间。
  - `apiclient report cis --level 1 --format text`：在级别 1 运行 CIS 合规性报告，并输出文本。

3. 查看报告文本输出。

解析输出

该命令生成一份基于文本的报告，显示各种 CIS 控件的合规性状态。输出包括：

- 各个 CIS 控件 ID
- 各个控件的描述
- 各项检查的“通过”、“失败”或“跳过”状态
- 解释任何合规性问题的详细信息

以下是在 Bottlerocket 实例上运行的报告输出示例：

```
Benchmark name:  CIS Bottlerocket Benchmark
Version:         v1.0.0
Reference:       https://www.cisecurity.org/benchmark/bottlerocket
Benchmark level: 1
Start time:      2025-04-11T01:40:39.055623436Z

[SKIP] 1.2.1     Ensure software update repositories are configured (Manual)
[PASS] 1.3.1     Ensure dm-verity is configured (Automatic)[PASS] 1.4.1     Ensure
 setuid programs do not create core dumps (Automatic)
[PASS] 1.4.2     Ensure address space layout randomization (ASLR) is enabled
 (Automatic)
[PASS] 1.4.3     Ensure unprivileged eBPF is disabled (Automatic)
[PASS] 1.5.1     Ensure SELinux is configured (Automatic)
```

```
[SKIP] 1.6       Ensure updates, patches, and additional security software are
 installed (Manual)
[PASS] 2.1.1.1   Ensure chrony is configured (Automatic)
[PASS] 3.2.5     Ensure broadcast ICMP requests are ignored (Automatic)
[PASS] 3.2.6     Ensure bogus ICMP responses are ignored (Automatic)
[PASS] 3.2.7     Ensure TCP SYN Cookies is enabled (Automatic)
[SKIP] 3.4.1.3   Ensure IPv4 outbound and established connections are configured
 (Manual)
[SKIP] 3.4.2.3   Ensure IPv6 outbound and established connections are configured
 (Manual)
[PASS] 4.1.1.1   Ensure journald is configured to write logs to persistent disk
 (Automatic)
[PASS] 4.1.2     Ensure permissions on journal files are configured (Automatic)

Passed:          11
Failed:          0
Skipped:         4
Total checks:    15
```

有关基准的信息，请参阅 Center for Internet Security（CIS）的 Kubernetes Benchmark。

相关资源

- Bottlerocket 操作系统文档中的 Bottlerocket CIS Benchmark。
- Kubernetes 文档中的调试运行中的 Pod。
- Center for Internet Security（CIS）中的 Kubernetes Benchmark

在 EKS 自动模式下使用客户自主管理型 KMS 密钥启用 EBS 卷加密

您可以使用客户自主管理型 KMS 密钥对 EKS 自动模式实例的临时根卷进行加密。

在管理 Kubernetes 集群的加密 EBS 卷时，Amazon EKS 自动模式使用服务相关角色向其他 Amazon 服务委派权限。本主题介绍当您为 Amazon EBS 加密指定客户自主管理型密钥时，如何设置所需的密钥策略。

注意事项：

- EKS 自动模式无需额外授权，即可使用默认的 Amazon 托管式密钥来保护您账户中的加密卷。
- 本主题介绍加密临时卷，即 EC2 实例的根卷。有关加密用于工作负载的数据卷的更多信息，请参阅the section called “创建 StorageClass”。

概述

以下 Amazon KMS 密钥可在 EKS 自动模式启动实例时用于加密 Amazon EBS 根卷：

- Amazon 托管式密钥：您账户中由 Amazon EBS 创建、拥有和管理的加密密钥。这是新账户的默认加密密钥。
- 客户托管密钥：您创建、拥有和管理的自定义加密密钥。

> **Note**
>
> 密钥必须是对称的。Amazon EBS 不支持非对称客户托管密钥。

第 1 步：配置密钥策略

您的 KMS 密钥必须具有相应的密钥策略，以允许 EKS 自动模式启动具有使用客户自主管理型密钥加密的 Amazon EBS 卷的实例。

请使用以下结构配置密钥策略：

> **Note**
>
> 此策略仅包含 EKS 自动模式的权限。如果其他身份需要使用密钥或管理授权，则此密钥策略可能需要额外的权限。

```
{
    "Version":"2012-10-17",
    "Id": "MyKeyPolicy",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:role/ClusterServiceRole"
                ]
            },
            "Action": [
                "kms:Encrypt",
```

```
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:role/ClusterServiceRole"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": "true"
                }
            }
        }
    ]
}
```

请务必将 <account-id> 替换为实际的 Amazon 账户 ID。

配置密钥策略时：

- ClusterServiceRole 必须具有使用该 KMS 密钥执行加密操作所需的 IAM 权限
- kms:GrantIsForAWSResource 条件可确保只能为 Amazon 服务创建授权

第 2 步：使用客户自主管理型密钥配置节点类

配置密钥策略后，在 EKS 自动模式的节点类配置中引用该 KMS 密钥：

```
apiVersion: eks.amazonaws.com/v1
```

```
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Insert existing configuration

  ephemeralStorage:
    size: "80Gi"  # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
    iops: 3000    # Range: 3000-16000
    throughput: 125  # Range: 125-1000

    # KMS key for encryption
    kmsKeyID: "arn:aws-cn:kms:<region>:<account-id>:key/<key-id>"
```

请将占位符值替换为实际的值：

- 请将 <region> 替换为您所在的 Amazon 区域
- 请将 <account-id> 替换为您的 Amazon 账户 ID
- 请将 <key-id> 替换为您的 KMS 密钥 ID

您可以使用以下任意一种格式指定 KMS 密钥：

- KMS 密钥 ID：1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
- KMS 密钥 ARN： arn:aws-cn:kms:us-west-2:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
- 密钥别名名称：alias/eks-auto-mode-key
- 密钥别名 ARN： arn:aws-cn:kms:us-west-2:111122223333:alias/eks-auto-mode-key

使用 kubectl 应用节点类配置：

```
kubectl apply -f nodeclass.yaml
```

相关资源

- 为 Amazon EKS 创建节点类
- 有关更多信息，请参阅《Amazon Key Management Service 开发人员指南》。
  - 密钥策略中 Amazon 服务的权限

- 更改密钥策略
- Amazon KMS 中的授权

更新 EKS 自动模式的组织控制

某些组织控制可能会阻止 EKS 自动模式正常运行。如果出现这种情况，则必须更新这些控制，以允许 EKS 自动模式拥有代表您管理 EC2 实例所需的权限。

EKS 自动模式使用服务角色启动支持 EKS 自动模式节点的 EC2 实例。服务角色是在您的账户中创建的 IAM 角色，服务假定该角色代表您执行操作。服务控制策略（SCP）始终适用于使用服务角色执行的操作。这将允许 SCP 禁止自动模式的操作。最常见的情况是使用 SCP 来限制可以启动的亚马逊机器映像（AMI）。要使 EKS 自动模式可以正常工作，请修改 SCP 以允许从 EKS 自动模式账户启动 AMI。

您还可以使用 EC2 允许的 AMI 功能来限制其他账户中 AMI 的可见性。如果您使用此功能，则必须扩展映像标准，以包含感兴趣区域中的 EKS 自动模式 AMI 账户。

示例 SCP：阻止除 EKS 自动模式 AMI 外的所有 AMI

除非 AMI 属于 us-west-2 或 us-east-1 的 EKS 自动模式 AMI 账户，否则以下 SCP 会阻止调用 `ec2:RunInstances`。

> **Note**
>
> 切勿使用 `ec2:Owner` 上下文键。Amazon 拥有 EKS 自动模式 AMI 账户，该键的值将始终为 `amazon`。如果构建一个 SCP，允许当 `ec2:Owner` 为 `amazon` 时启动 AMI，则将允许启动所有 Amazon 拥有的 AMI，而不仅限于 EKS 自动模式的 AMI。*

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAMI",
      "Effect": "Deny",
      "Action": "ec2:RunInstances",
      "Resource": "arn:*:ec2:*::image/ami-*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
```

```
            "767397842682",
            "992382739861"
          ]
        }
      }
    }
  ]
}
```

EKS 自动模式 AMI 账户

因区域而异的 Amazon 账户托管 EKS 自动模式公有 AMI。

Amazon 区域	账户
af-south-1	471112993317
ap-east-1	590183728416
ap-northeast-1	851725346105
ap-northeast-2	992382805010
ap-northeast-3	891377407544
ap-south-1	975049899075
ap-south-2	590183737426
ap-southeast-1	339712723301
ap-southeast-2	58264376476
ap-southeast-3	471112941769
ap-southeast-4	590183863144
ap-southeast-5	654654202513
ap-southeast-7	533267217478
ca-central-1	992382439851

ca-west-1	767397959864
eu-central-1	891376953411
eu-central-2	381492036002
eu-north-1	339712696471
eu-south-1	975049955519
eu-south-2	471112620929
eu-west-1	381492008532
eu-west-2	590184142468
eu-west-3	891376969258
il-central-1	590183797093
me-central-1	637423494195
me-south-1	905418070398
mx-central-1	211125506622
sa-east-1	339712709251
us-east-1	992382739861
us-east-2	975050179949
us-west-1	975050035094
us-west-2	767397842682

关联公有 IP 地址

调用 `ec2:RunInstances` 时，实例启动的 `AssociatePublicIpAddress` 字段将根据实例启动到的子网类型自动确定。可以使用 SCP 强制将该值显式设置为

false，无论实例启动到哪种类型的子网。在这种情况下，也可以将 NodeClass 字段 `spec.advancedNetworking.associatePublicIPAddress` 设置为 false 以满足 SCP 的要求。

```
{
      "Sid": "DenyPublicEC2IPAddesses",
      "Effect": "Deny",
      "Action": "ec2:RunInstances",
      "Resource": "arn:aws:ec2:*:*:network-interface/*",
      "Condition": {
          "BoolIfExists": {
              "ec2:AssociatePublicIpAddress": "true"
          }
      }
  }
```

使用 EKS 自动模式控制将工作负载部署到容量预留中

您可以控制工作负载在容量预留中的部署。EKS 自动模式支持 EC2 按需容量预留（ODCR）和适用于机器学习的 EC2 容量块。

> **Tip**
>
> 默认情况下，EKS 自动模式会在开放式 ODCR 和机器学习容量块中自动启动。在 NodeClass 定义中使用 `capacityReservationSelectorTerms` 时，EKS 自动模式将不再自动使用任何开放容量预留。

EC2 按需容量预留（ODCR）

通过使用 EC2 按需容量预留（ODCR），您可以在特定可用区中为 Amazon EC2 实例预留计算容量达任意持续时间。使用 EKS 自动模式时，您可能想要控制是否将 Kubernetes 工作负载部署到这些预留实例上，以最大限度地利用预先购买的容量，或确保关键工作负载能够访问有保障的资源。

默认情况下，EKS 自动模式会在开放式 ODCR 中自动启动。但是，通过在 NodeClass 上配置 `capacityReservationSelectorTerms`，您可以显式控制工作负载使用的 ODCR。使用已配置的 ODCR 预置的节点将具有 `karpenter.sh/capacity-type: reserved` 并优先于按需和竞价型节点。启用此功能后，EKS 自动模式将不再自动使用开放式 ODCR – 其必须由 NodeClass 显式选择，这样您就可以精确控制集群中的容量预留使用情况。

> **Warning**
>
> 如果在集群中的 NodeClass 上配置了 `capacityReservationSelectorTerms`，则 EKS 自动模式将不再为集群中的任何 NodeClass 自动使用开放式 ODCR。

示例 NodeClass

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
spec:
  # Optional: Selects upon on-demand capacity reservations and capacity blocks
  # for EKS Auto Mode to prioritize.
  capacityReservationSelectorTerms:
    - id: cr-56fac701cc1951b03
    # Alternative Approaches
    - tags:
        app: "my-app"
      # Optional owning account ID filter
      owner: "012345678901"
```

此示例 NodeClass 演示了两种选择 ODCR 的方法。第一种方法通过其 ID（`cr-56fac701cc1951b03`）直接引用特定 ODCR。第二种方法使用基于标签的选择，目标是带有标签 `Name: "targeted-odcr"` 的 ODCR。您还可以选择按拥有预留的 Amazon 账户进行筛选，这在跨账户场景或使用共享容量预留时特别有用。

适用于机器学习的 EC2 容量块

机器学习容量块允许您在未来预留基于 GPU 的加速型计算实例，以支持短期机器学习（ML）工作负载。在容量块内运行的实例会自动紧密放置在 Amazon EC2 UltraClusters 中，以实现低延迟、PB 级非阻塞联网。

有关受支持的平台和实例类型的更多信息，请参阅《EC2 用户指南》中的 ML 容量块。

您可以创建使用机器学习容量块的 EKS 自动模式 NodeClass，类似于 ODCR（如前所述）。

以下示例定义创建了三个资源：

1. 引用容量块预留的 NodeClass
2. 使用 NodeClass 并应用污点的 NodePool
3. 可以容忍污点并请求 GPU 资源的容器组（pod）规范

示例 NodeClass

此 NodeClass 通过其预留 ID 来引用特定机器学习容量块。您可以从 EC2 控制台获取此 ID。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu
spec:
  # Specify your Capacity Block reservation ID
  capacityReservationSelectorTerms:
    - id: cr-56fac701cc1951b03
```

有关更多信息，请参阅 the section called “创建节点类”。

示例 NodePool

此 NodePool 引用 gpu NodeClass 并指定重要的配置：

- 其仅通过设置 `karpenter.sh/capacity-type: reserved` 来使用预留容量
- 其请求适用于机器学习工作负载的特定 GPU 实例系列
- 其应用 `nvidia.com/gpu` 污点，确保只在这些节点上调度 GPU 工作负载

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: gpu
      requirements:
        - key: eks.amazonaws.com/instance-family
          operator: In
          values:
            - g6
            - p4d
            - p4de
```

```
            - p5
            - p5e
            - p5en
            - p6
            - p6-b200
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - reserved
            # Enable other capacity types
            # - on-demand
            # - spot
      taints:
        - effect: NoSchedule
          key: nvidia.com/gpu
```

有关更多信息，请参阅 the section called “创建节点池”。

示例容器组（pod）

此示例容器组（pod）演示了如何将工作负载配置为在容量块节点上运行：

- 其使用 nodeSelector 来定位特定的 GPU 类型（在本例中为 H200 GPU）
- 其包括对 NodePool 应用的 `nvidia.com/gpu` 污点的容忍度
- 其使用 `nvidia.com/gpu` 资源类型明确请求 GPU 资源

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  nodeSelector:
    # Select specific GPU type - uncomment as needed
    # eks.amazonaws.com/instance-gpu-name: l4
    # eks.amazonaws.com/instance-gpu-name: a100
    eks.amazonaws.com/instance-gpu-name: h200
    # eks.amazonaws.com/instance-gpu-name: b200
    eks.amazonaws.com/compute-type: auto
  restartPolicy: OnFailure
  containers:
  - name: nvidia-smi
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
```

```
    args:
    - "nvidia-smi"
    resources:
      requests:
        # Uncomment if needed
        # memory: "30Gi"
        # cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        # Uncomment if needed
        # memory: "30Gi"
        nvidia.com/gpu: 1
  tolerations:
  - key: nvidia.com/gpu
    effect: NoSchedule
    operator: Exists
```

有关更多信息，请参阅 Kubernetes 文档中的 Pod。

相关资源

- 《Amazon EC2 用户指南》中的 ML 容量块
- 《Amazon EC2 用户指南》中的查找和购买容量块
- 管理 Amazon EKS 上人工智能/机器学习工作负载的计算资源
- 《EKS 最佳实践指南》中的 GPU 资源优化和成本管理

将 EKS 自动模式节点部署到 Local Zones

EKS 自动模式通过自动节点预置简化了集群管理。AmazonLocal Zones 将 Amazon 基础设施扩展到离最终用户更近的地理位置，从而减少延迟敏感型应用程序的延迟。本指南将引导您完成将 EKS 自动模式节点部署到 Amazon Local Zones 的过程，您随后便能以更低的延迟为特定地理区域的用户运行容器化应用程序。

本指南还演示了如何使用 Kubernetes 污点和容忍度来确保只有特定的工作负载在 Local Zone 节点上运行，从而帮助您控制成本和优化资源使用。

先决条件

在开始将 EKS 自动模式节点部署 Local Zones 之前，请确保已满足以下先决条件：

- 现有的 EKS 自动模式集群

- 在您的 Amazon 账户中已选择加入 Local Zone

步骤 1：创建 Local Zone 子网

将 EKS 自动模式节点部署到 Local Zone 的第一步是在该 Local Zone 中创建子网。该子网为您的节点提供网络基础设施，并允许它们与 VPC 的其余部分进行通信。按照《Amazon Local Zones 用户指南》中的创建 Local Zone 子网说明，在您选择的 Local Zone 中创建子网。

> **Tip**
>
> 记下您的 Local Zone 子网的名称。

步骤 2：为 Local Zone 子网创建 NodeClass

创建 Local Zone 子网后，您需要定义一个引用该子网的 NodeClass。NodeClass 是一种 Kubernetes 自定义资源，用于指定节点的基础设施属性，包括要使用的子网、安全组和存储配置。在下面的示例中，我们创建了一个名为“local-zone”的 NodeClass，它根据 Local Zone 子网的名称对其进行定位。您也可以使用子网 ID。您需要调整此配置以定位 Local Zone 子网。

有关更多信息，请参阅 the section called “创建节点类”。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: local-zone
spec:
  subnetSelectorTerms:
    - id: <local-subnet-id>
```

步骤 3：使用 NodeClass 和污点创建 NodePool

配置好 NodeClass 后，您现在需要创建一个使用此 NodeClass 的 NodePool。NodePool 定义节点的计算特征，包括实例类型。NodePool 使用 NodeClass 作为参考，以确定在何处启动实例。

在下面的示例中，我们创建了一个引用“local-zone”NodeClass 的 NodePool。我们还将污点添加到节点，确保只有容忍度匹配的容器组（pod）才能在这些 Local Zone 节点上被调度。这对于 Local Zone 节点尤其重要，因为 Local Zone 节点的成本通常更高，只有特别受益于降低延迟的工作负载才应使用此类节点。

有关更多信息，请参阅 the section called “创建节点池”。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
    metadata:
      labels:
        node-type: local-zone
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: local-zone
      taints:
        - key: "aws.amazon.com/local-zone"
          value: "true"
          effect: NoSchedule

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["4", "8", "16", "32"]
```

带有 aws.amazon.com/local-zone 键和 NoSchedule 效果的污点可确保容忍度不匹配的容器组（pod）不会在这些节点上被调度。这样可以防止常规工作负载意外在 Local Zone 中运行，这可能会导致意外成本。

步骤 4：部署具有容忍度和节点亲和性的工作负载

要对 Local Zone 节点上的工作负载放置进行最佳控制，请同时使用污点/容忍度和节点亲和性。此组合方法具有以下优势：

1. 成本控制：污点可确保只有具有明确容忍度的容器组（pod）才能使用可能较为昂贵的 Local Zone 资源。
2. 保证放置：节点亲和性可确保您的延迟敏感型应用程序仅在 Local Zone 中运行，而不是在常规集群节点上运行。

以下是配置为专门在 Local Zone 节点上运行的部署示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: low-latency-app
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: low-latency-app
  template:
    metadata:
      labels:
        app: low-latency-app
    spec:
      tolerations:
      - key: "aws.amazon.com/local-zone"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: "node-type"
                operator: "In"
                values: ["local-zone"]
      containers:
      - name: application
        image: my-low-latency-app:latest
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"
          requests:
            cpu: "500m"
            memory: "512Mi"
```

此部署有两个关键的调度配置：

1. 容忍度允许将容器组（pod）在带有 `aws.amazon.com/local-zone` 污点的节点上调度。
2. 节点亲和性要求可确保这些容器组（pod）只能在带有 `node-type: local-zone` 标签的节点上运行。

它们共同确保您的延迟敏感型应用程序仅在 Local Zone 节点上运行，并且除非明确配置为使用 Local Zone 资源，否则常规应用程序不会消耗 Local Zone 资源。

步骤 5：使用 Amazon 控制台验证

设置 NodeClass、NodePool 和 Deployments 后，您应验证节点是否按预期在 Local Zone 中预置，并且您的工作负载是否正在这些节点上运行。您可以使用 Amazon 管理控制台验证 EC2 实例是否在正确的 Local Zone 子网中启动。

此外，您可以使用 `kubectl get nodes -o wide` 查看 Kubernetes 节点列表，确认这些节点是否以正确的标签和污点加入集群：

```
kubectl get nodes -o wide
kubectl describe node <node-name> | grep -A 5 Taints
```

您还可以验证您的工作负载容器组（pod）是否已在 Local Zone 节点上调度：

```
kubectl get pods -o wide
```

这种方法可确保只有特别容忍 Local Zone 污点的工作负载才会在这些节点上被调度，从而帮助您控制成本并最有效地利用 Local Zone 资源。

了解 EKS 自动模式的工作原理

通过本章了解 Amazon EKS 自动模式集群的组件的工作原理。

主题

- 了解 Amazon EKS 自动模式的托管式实例
- 了解 EKS 自动模式中的身份和访问权限
- 了解 EKS 自动模式下的 VPC 联网和负载均衡

了解 Amazon EKS 自动模式的托管式实例

本主题介绍了 Amazon EKS 自动模式如何管理 EKS 集群中的 Amazon EC2 实例。启用 EKS 自动模式后，集群的计算资源将由 EKS 自动预置和管理，从而改变您与作为集群中节点的 EC2 实例的交互方式。

了解 Amazon EKS 自动模式会如何管理实例，对于规划工作负载部署策略和操作程序至关重要。与传统的 EC2 实例或托管式节点组不同，此类实例遵循不同的生命周期模式，EKS 负责许多操作任务，同时限制某些类型的访问和自定义。

Amazon EKS 自动模式可自动执行创建新 EC2 实例的例行任务，并将其作为节点挂载到 EKS 集群。EKS 自动模式会检测现有节点何时不能满足工作负载的需要，并创建新的 EC2 实例。

Amazon EKS 自动模式负责创建、删除和修补 EC2 实例。您负责实例上部署的容器和容器组。

EKS 自动模式创建的 EC2 实例属于托管式实例，这与其他 EC2 实例不同。此类托管式实例由 EKS 所有，并且受到更多限制。您不能直接访问由 EKS 自动模式管理的实例或在此类实例上安装软件。

Amazon 建议运行 EKS 自动模式或自主管理型 Karpenter。您可以在迁移过程中或在高级配置中同时安装这两者。如果您同时安装了这两者，请配置节点池，以将工作负载关联到 Karpenter 或 EKS 自动模式。

有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 managed instances。

比较表

标准 EC2 实例	EKS 自动模式托管式实例
您负责修补和更新实例。	Amazon 会自动修补和更新实例。
EKS 对实例上的软件不承担任何责任。	EKS 对实例上的某些软件负责，例如 `kubelet`、容器运行时和操作系统。
您可以使用 EC2 API 来删除 EC2 实例。	EKS 负责决定将在您账户中部署的实例数量。如果您删除工作负载，EKS 将会减少您账户中的实例数量。
您可以使用 SSH 访问 EC2 实例。	您可以将容器组和容器部署到托管式实例。
您负责决定操作系统和映像（AMI）。	Amazon 负责决定操作系统和映像。

标准 EC2 实例	EKS 自动模式托管式实例
您可以部署依赖 Windows 或 Ubuntu 功能的工作负载。	您可以部署基于 Linux 但无特定操作系统依赖项的容器。
您负责决定要启动的实例类型和系列。	Amazon 负责决定要启动的实例类型和系列。您可以使用节点池来限制 EKS 自动模式可选择的实例类型。

以下功能对托管式实例和标准 EC2 实例均适用：

- 您可以在 Amazon 控制台中查看实例。
- 您可以将实例存储用作工作负载的临时存储。

AMI 支持

通过 EKS 自动模式，Amazon 可确定用于计算节点的映像（AMI）。Amazon 还会监控新的 EKS 自动模式 AMI 版本的推出。如果遇到与 AMI 版本相关的工作负载问题，请创建支持案例。有关更多信息，请参阅《Amazon Support User Guide》中的 Creating support cases and case management。

通常，EKS 每周都会发布一个包含 CVE 和安全修复的新 AMI。

EKS 自动模式支持的实例参考

EKS 自动模式仅创建属于受支持类型且符合最小大小要求的实例。

EKS 自动模式支持以下实例类型：

系列	实例类型
计算优化型（C）	c8g、c7a、c7g、c7gn、c7gd、c7i、c7i-flex、c6a、c6g、c6i、c6gn、c6id、c6in、c6gd、c5、c5a、c5d、c5ad、c5n、c4
通用型（M）	m8g、m7i、m7a、m7g、m7gd、m7i-flex、m6a、m6i、m6in、m6g、m6idn、m6id、m6gd、m5、m5a、m5ad、m5n、m5dn、m5d、m5zn、m4

系列	实例类型
内存优化型（R）	r8g、r7a、r7iz、r7gd、r7i、r7g、r6a、r6i、r6id、r6in、r6idn、r6g、r6gd、r5、r5n、r5a、r5dn、r5b、r5ad、r5d、r4
突增型（T）	t4g、t3、t3a、t2
内存增强型（Z/X）	z1d、x8g、x2gd
存储优化型（I/D）	i8g、i7ie、i4g、i4i、i3、i3en、is4gen、d3、d3en、im4gn
加速计算型（P/G/Inf/Trn）	p5、p4d、p4de、p3、p3dn、gr6、g6、g6e、g5g、g5、g4dn、inf2、inf1、trn1、trn1n
高性能计算型（X2）	x2iezn、x2iedn、x2idn

此外，EKS 自动模式将仅创建满足以下要求的 EC2 实例：

- 超过 1 个 CPU
- 实例大小并非纳米型、微型或小型

有关更多信息，请参阅 Amazon EC2 实例类型命名约定。

实例元数据服务

- 默认情况下，EKS 自动模式会遵循 Amazon 安全最佳实践，强制执行 IMDSv2（跃点限制为 1）。
- 此默认配置在自动模式下无法修改。
- 对于通常需要 IMDS 访问权限的附加组件，请在安装过程中提供参数（例如 Amazon 区域），以避免 IMDS 查找。有关更多信息，请参阅 the section called “可以自定义的字段”。
- 如果容器组在自动模式下运行时确实需要 IMDS 访问权限，则必须将该容器组配置为可以使用 `hostNetwork: true` 运行。这将允许容器组直接访问实例元数据服务。
- 授予容器组实例元数据访问权限时，应考虑安全影响。

有关 Amazon EC2 实例元数据服务（IMDS）的更多信息，请参阅《Amazon EC2 用户指南》中的配置实例元数据服务选项。

注意事项

- 如果 NodeClass 中配置的临时存储小于实例的 NVMe 本地存储，则 EKS 自动模式会自动执行以下操作，无需手动配置：
  - 使用较小（20 GiB）的 Amazon EBS 数据量来降低成本。
  - 格式化并配置 NVMe 本地存储以供临时数据使用。这包括在具有多个 NVMe 驱动器的情况下设置 RAID 0 阵列。
- 当 `ephemeralStorage.size` 等于或超过本地 NVMe 容量时，会发生以下操作：
  - 自动模式会跳过小型 EBS 卷。
  - NVMe 驱动器将直接公开给工作负载。
- Amazon EKS 自动模式不支持下列 Amazon 故障注入服务操作：
  - `ec2:RebootInstances`
  - `ec2:SendSpotInstanceInterruptions`
  - `ec2:StartInstances`
  - `ec2:StopInstances`
  - `ec2:TerminateInstances`
  - `ec2:PauseVolumeIO`
- Amazon EKS 自动模式支持 Amazon 故障注入服务 EKS 容器组（pod）操作。有关更多信息，请参阅管理故障注入服务试验和《Amazon 韧性监测中心用户指南》中的执行 Amazon FIS aws:eks:pod 操作。
- 不需要在 EKS 自动模式节点上安装 `Neuron Device Plugin`。

  如果集群中有其他类型的节点，则需要将 Neuron Device 插件配置为不在自动模式节点上运行。有关更多信息，请参阅 the section called “控制部署”。

了解 EKS 自动模式中的身份和访问权限

本主题介绍使用 EKS 自动模式所需的 Identity and Access Management（IAM）角色和权限。EKS 自动模式使用两个主要 IAM 角色：一个集群 IAM 角色和一个节点 IAM 角色。这两个角色与 EKS 容器组身份和 EKS 访问条目配合使用，从而为 EKS 集群提供全面的访问管理。

配置 EKS 自动模式时，您需要设置这两个 IAM 角色，为其授予允许 Amazon 服务与集群资源交互所需的特定权限，包括管理计算资源、存储卷、负载均衡器和网络组件的权限。了解这些角色配置对于确保集群的正常运行和安全至关重要。

在 EKS 自动模式下，Amazon IAM 角色会自动通过 EKS 访问条目映射到 Kubernetes 权限，无需手动配置 `aws-auth` ConfigMaps 或自定义绑定。创建新的自动模式集群时，EKS 会使用访问条目自动创建相应的 Kubernetes 权限，从而确保 Amazon 服务和集群组件在 Amazon 和 Kubernetes 授权系统中都具有适当的访问权限级别。这种自动集成可减少配置的复杂性，有助于防止在管理 EKS 集群时常见的权限相关问题。

集群 IAM 角色

集群 IAM 角色是 Amazon EKS 用来管理 Kubernetes 集群权限的 Amazon Identity and Access Management（IAM）角色。此角色向 Amazon EKS 授予代表集群与其他 Amazon 服务进行交互的必要权限，并且使用 EKS 访问条目自动配置了 Kubernetes 权限。

- 您必须将 Amazon IAM 策略附加到此角色。
- EKS 自动模式使用 EKS 访问条目自动为此角色附加 Kubernetes 权限。
- 使用 EKS 自动模式时，Amazon 建议为每个 Amazon 账户创建一个集群 IAM 角色。
- Amazon 建议将此角色命名为 `AmazonEKSAutoClusterRole`。
- 此角色需要具有允许多个 Amazon 服务管理资源的权限，包括 EBS 卷、弹性负载均衡器和 EC2 实例等。
- 此角色的建议配置包括与 EKS 自动模式的不同功能相关的多个 Amazon 托管式 IAM 策略。
  - `AmazonEKSComputePolicy`
  - `AmazonEKSBlockStoragePolicy`
  - `AmazonEKSLoadBalancingPolicy`
  - `AmazonEKSNetworkingPolicy`
  - `AmazonEKSClusterPolicy`

有关集群 IAM 角色和 Amazon 托管式 IAM 策略的更多信息，请参阅：

- the section called “ Amazon 托管策略”
- the section called “集群 IAM 角色”

有关 Kubernetes 访问权限的更多信息，请参阅：

- the section called “检查访问策略”

节点 IAM 角色

节点 IAM 角色是 Amazon EKS 用来管理 Kubernetes 集群中 Worker 节点权限的 Amazon Identity and Access Management（IAM）角色。此角色向作为 Kubernetes 节点运行的 EC2 实例授予与 Amazon 服务和资源进行交互所需的必要权限，并使用 EKS 访问条目自动配置 Kubernetes RBAC 权限。

- 您必须将 Amazon IAM 策略附加到此角色。
- EKS 自动模式使用 EKS 访问条目自动为此角色附加 Kubernetes RBAC 权限。
- Amazon 建议将此角色命名为 `AmazonEKSAutoNodeRole`。
- 使用 EKS 自动模式时，Amazon 建议为每个 Amazon 账户创建一个节点 IAM 角色。
- 此角色具有的权限较为有限。主要权限包括代入容器组身份角色以及从 ECR 中拉取映像的权限。
- Amazon 建议使用以下 Amazon 托管式 IAM 策略：
  - `AmazonEKSWorkerNodeMinimalPolicy`
  - `AmazonEC2ContainerRegistryPullOnly`

有关集群 IAM 角色和 Amazon 托管式 IAM 策略的更多信息，请参阅：

- the section called “ Amazon 托管策略”
- the section called “节点 IAM 角色”

有关 Kubernetes 访问权限的更多信息，请参阅：

- the section called “检查访问策略”

服务相关角色

Amazon EKS 会使用一个服务相关角色（SLR）来执行某些操作。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。服务相关角色是由 Amazon EKS 预定义的，并包含服务代表您调用其它 Amazon 服务所需的所有权限。

Amazon 会自动创建和配置此 SLR。只有首先删除 SLR 的相关资源后才能将其删除。这将保护您的 Amazon EKS 资源，因为您不会无意中删除对资源的访问权限。

SLR 策略会向 Amazon EKS 授予观察和删除核心基础设施组件的权限：EC2 资源（实例、网络接口、安全组）、ELB 资源（负载均衡器、目标组）、CloudWatch 功能（日志记录和指标）以及带有“eks”前缀的 IAM 角色。此策略还支持通过 VPC/托管区关联来实现私有端点联网，并且包括 EventBridge 监控和清理带有 EKS 标签的资源的权限。

有关更多信息，请参阅：

- the section called “Amazon托管策略：AmazonEKSServiceRolePolicy”
- the section called “Amazon EKS 的服务相关角色权限”

EKS 自动模式资源的自定义 Amazon 标签

默认情况下，与 EKS 自动模式相关的托管式策略不允许将用户定义的标签应用于自动模式预置的 Amazon 资源。如果要将用户定义的标签应用于 Amazon 资源，则必须为集群 IAM 角色附加额外的权限，以提供在 Amazon 资源上创建和修改标签的充分权限。以下示例策略将允许不受限制的标记访问权限：

查看自定义标签策略示例

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Compute",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateFleet",
                "ec2:RunInstances",
                "ec2:CreateLaunchTemplate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                },
                "StringLike": {
                    "aws:RequestTag/eks:kubernetes-node-class-name": "*",
                    "aws:RequestTag/eks:kubernetes-node-pool-name": "*"
                }
            }
```

```
        },
        {
            "Sid": "Storage",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVolume",
                "ec2:CreateSnapshot"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:snapshot/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "Networking",
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterface",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                },
                "StringLike": {
                    "aws:RequestTag/eks:kubernetes-cni-node-name": "*"
                }
            }
        },
        {
            "Sid": "LoadBalancer",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateRule",
                "ec2:CreateSecurityGroup"
            ],
```

```
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "ShieldProtection",
            "Effect": "Allow",
            "Action": [
                "shield:CreateProtection"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "ShieldTagResource",
            "Effect": "Allow",
            "Action": [
                "shield:TagResource"
            ],
            "Resource": "arn:aws:shield::*:protection/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        }
    ]
}
```

访问策略参考

要详细了解 EKS 自动模式使用的 Kubernetes 权限，请参阅the section called “检查访问策略”。

了解 EKS 自动模式下的 VPC 联网和负载均衡

本主题介绍如何在 EKS 自动模式下配置虚拟私有云（VPC）联网和负载均衡功能。虽然 EKS 自动模式会自动管理大多数联网组件，但您仍然可以通过 NodeClass 资源和负载均衡器注释来自定义集群联网配置的某些方面。

使用 EKS 自动模式时，Amazon 会管理集群的 VPC 容器网络接口（CNI）配置和负载均衡器预置。您可以通过定义 NodeClass 对象并对服务和 Ingress 资源应用特定的注释来影响联网行为，同时保持 EKS 自动模式提供的自动运行模式优势。

联网功能

EKS 自动模式具有一个能够处理节点和容器组（pod）联网的新联网功能。您可以通过创建 NodeClass Kubernetes 对象来对其进行配置。

以前的 Amazon VPC CNI 配置选项不适用于 EKS 自动模式。

使用 **NodeClass** 配置联网

借助 EKS 自动模式中的 NodeClass 资源，您可以自定义联网功能的某些方面。通过 NodeClass，您可以指定安全组选择，控制 VPC 子网中的节点放置，设置 SNAT 策略，配置网络策略以及启用网络事件日志记录。这种方法既可保持 EKS 自动模式的自动化运行模式，同时又可灵活自定义网络。

您可以使用 NodeClass 来：

- 选择节点的安全组
- 控制节点在 VPC 子网上的放置方式
- 将节点 SNAT 策略设置为 random 或 disabled
- 启用 Kubernetes 网络策略，包括：
  - 将网络策略设置为“默认拒绝”或“默认允许”
  - 启用将网络事件日志记录到文件。
- 通过将容器组（pod）连接到不同的子网，将容器组（pod）流量与节点流量隔离开来。

了解如何创建 Amazon EKS 节点类。

注意事项

EKS 自动模式支持下列功能：

- EKS 网络策略。
- Kubernetes 容器组的 `HostPort` 和 `HostNetwork` 选项。
- 公有子网或私有子网中的节点和容器组（pod）。
- 在节点上缓存 DNS 查询。

EKS 自动模式不支持下列功能：

- 每容器组的安全组（SGPP）。
- `ENIConfig` 中的自定义联网。您可以将容器组（pod）放在多个子网中，也可以使用 the section called “容器组（pod）的子网选择” 专门将它们与节点流量隔离开来。
- 暖 IP、暖前缀和暖 ENI 配置。
- 最低 IP 目标配置。
- 开源 Amazon VPC CNI 支持的其他配置。
- 网络策略配置，例如 conntrack 计时器自定义（默认为 300 秒）。
- 将网络事件日志导出到 CloudWatch。

网络资源管理

EKS 自动模式通过监控用于网络配置的节点类资源来处理前缀、IP 寻址及网络接口管理。该服务会自动执行多项关键操作：

前缀委派

EKS 自动模式会默认使用前缀委托（/28 前缀）进行容器组（pod）联网，并维护可根据调度的容器组（pod）数量进行扩展的预定义 IP 热池。检测到容器组（pod）子网碎片时，自动模式会预置辅助 IP 地址（/32）。由于这种默认的容器组（pod）联网算法，自动模式会根据每种实例类型支持的 ENI 和 IP 数量来计算每个节点的最大容器组（pod）数（假设是最坏的碎片化情况）。有关每个实例类型的最大 ENI 和 IP 数量的更多信息，请参阅《EC2 用户指南》中的每个网络接口的最大 IP 地址数。新一代（Nitro v6 及更高版本）实例系列通常会增加每种实例类型的 ENI 和 IP，自动模式会相应地调整最大容器组（pod）计算。

对于 IPv6 集群，仅使用前缀委派，自动模式始终使用每个节点 110 个容器组（pod）的最大容器组（pod）限制。

冷却管理

该服务为不再使用的前缀或辅助 IPv4 地址设置了冷却池。冷却时间到期后，这些资源将释放回 VPC。但是，如果容器组（pod）在冷却时间内重复使用这些资源，它们将从冷却池中恢复。

IPv6 支持

对于 IPv6 集群，EKS 自动模式会在主网络接口上为每个节点预置 /80 IPv6 前缀。

该服务还可确保对所有网络接口进行适当管理和垃圾回收。

负载均衡

您可以使用服务和 Ingress 资源上的注释来配置由 EKS 自动模式预置的 Amazon 弹性负载均衡器。

有关更多信息，请参阅the section called “创建 IngressClass”或the section called “创建服务”。

EKS 自动模式负载均衡的注意事项

- 默认目标模式是 IP 模式，而不是实例模式。
- EKS 自动模式仅支持网络负载均衡器的安全组模式。
- Amazon 不支持将负载均衡器从自主管理型 Amazon 负载均衡器控制器迁移到由 EKS 自动模式管理。
- 不支持 `TargetGroupBinding` 规范中的 `networking.ingress.ipBlock` 字段。
- 如果 Worker 节点使用自定义安全组（非 `eks-cluster-sg-` 命名模式），则集群角色需要额外的 IAM 权限。默认 EKS 托管式策略仅允许 EKS 修改名为 `eks-cluster-sg-` 的安全组。如果没有修改自定义安全组的权限，EKS 将无法添加允许 ALB/NLB 流量到达容器组所需的 Ingress 规则。

CoreDNS 注意事项

EKS 自动模式不使用传统的 CoreDNS 部署在集群内提供 DNS 解析。相反，自动模式节点利用 CoreDNS 作为系统服务直接在每个节点上运行。如果将传统集群过渡到自动模式，则可以在工作负载移至自动模式节点后，从集群中移除 CoreDNS 部署。

> **Important**
>
> 如果您计划维护同时包含自动模式和非自动模式节点的集群，则必须保留 CoreDNS 部署。非自动模式节点依赖传统的 CoreDNS 容器组（pod）进行 DNS 解析，因为它们无法访问自动模式提供的节点级 DNS 服务。

EKS 自动模式故障排除

使用 EKS 自动模式时，Amazon 对您 Amazon 账户中的 EC2 实例承担更多责任。EKS 负责节点上的容器运行时、节点上的操作系统和某些控制器。这包括块存储控制器、负载均衡控制器和计算控制器。

您必须使用 Amazon 和 Kubernetes API 来排除节点故障。您可以：

- 通过 the section called “节点监控代理” 使用 Kubernetes `NodeDiagnostic` 资源检索节点日志。有关更多步骤，请参阅the section called “获取节点日志”。
- 使用 Amazon EC2 CLI 命令 `get-console-output` 检索节点的控制台输出。有关更多步骤，请参阅the section called “使用 Amazon EC2 CLI 获取 EC2 托管式实例的控制台输出”。
- 使用 Kubernetes 调试容器检索节点日志。有关更多步骤，请参阅the section called “使用调试容器和 `kubectl` CLI 获取节点日志”。

> **Note**
>
> EKS 自动模式使用 EC2 托管式实例。您不能直接访问 EC2 托管式实例，包括通过 SSH。

您可能遇到以下问题，这些问题都有针对 EKS 自动模式组件的解决方案：

- 容器组（pod）卡在 `Pending` 状态，无法调度到自动模式节点。有关解决方案，请参阅the section called “对容器组（pod）无法调度到自动模式节点进行问题排查”。
- EC2 托管实例不作为 Kubernetes 节点加入集群。有关解决方案，请参阅the section called “对节点无法加入集群进行问题排查”。
- `NodePools`、`PersistentVolumes` 以及使用 EKS 自动模式涉及的控制器的 `Services` 方面出现错误和问题。有关解决方案，请参阅the section called “对自动模式涉及的控制器进行问题排查”。
- 容器组（pod）安全性增强后，无法在容器组（pod）之间共享卷。有关解决方案，请参阅the section called “在容器组（pod）之间共享卷”。

您可以使用以下方法对 EKS 自动模式组件进行问题排查：

- the section called “使用 Amazon EC2 CLI 获取 EC2 托管式实例的控制台输出”
- the section called “使用调试容器和 `kubectl` CLI 获取节点日志”
- the section called “在 Amazon 控制台中查看与 EKS 自动模式关联的资源”

- the section called “查看您 Amazon 账户中的 IAM 错误”
- the section called “使用 VPC Reachability Analyzer 检测节点连接问题”

节点监控代理

EKS 自动模式包含 Amazon EKS 节点监控代理。您可以使用此代理来查看有关节点的故障排除和调试信息。节点监控代理会发布 Kubernetes `events` 和节点 `conditions`。有关更多信息，请参阅 the section called “节点运行状况”。

使用 Amazon EC2 CLI 获取 EC2 托管式实例的控制台输出

此过程有助于排除启动时或内核级别的问题。

首先，您需要确定与工作负载关联的实例的 EC2 实例 ID。其次，使用 Amazon CLI 检索控制台输出。

1. 确认您已安装 `kubectl` 并连接到集群
2. （可选）使用 Kubernetes 部署的名称来列出关联的容器组。

```
kubectl get pods -l app=<deployment-name>
```

3. 使用 Kubernetes 容器的名称来确定所关联节点的 EC2 实例 ID。

```
kubectl get pod <pod-name> -o wide
```

4. 使用 EC2 实例 ID 检索控制台输出。

```
aws ec2 get-console-output --instance-id <instance id> --latest --output text
```

使用调试容器和 **kubectl** CLI 获取节点日志

从 EKS 自动模式节点检索日志的推荐方法是使用 `NodeDiagnostic` 资源。有关步骤，请参阅the section called “获取节点日志”。

但是，也可使用 `kubectl debug node` 命令从实例实时流式传输日志。此命令会在要调试的节点上启动一个新的容器组（pod），然后您就能以交互方式使用它。

1. 启动调试容器。以下命令使用 `i-01234567890123456` 作为节点的实例 ID，`-it` 会分配 `tty` 并附加 `stdin` 供交互式使用，并使用 kubeconfig 文件中的 `sysadmin` 配置文件。

```
kubectl debug node/i-01234567890123456 -it --profile=sysadmin --image=public.ecr.aws/
amazonlinux/amazonlinux:2023
```

示例输出如下。

```
Creating debugging pod node-debugger-i-01234567890123456-nxb9c with container
 debugger on node i-01234567890123456.
If you don't see a command prompt, try pressing enter.
bash-5.2#
```

2. 现在，您可以从 Shell 中安装提供 nsenter 命令的 util-linux-core。使用 nsenter 进入主机上 PID 1 (init) 的挂载命名空间，然后运行 journalctl 命令以从 kubelet 流式传输日志：

```
yum install -y util-linux-core
nsenter -t 1 -m journalctl -f -u kubelet
```

为安全起见，默认情况下，Amazon Linux 容器映像不会安装很多个二进制文件。您可以使用 yum whatprovides 命令来确定为提供给定二进制文件而必须安装的软件包。

```
yum whatprovides ps
```

```
Last metadata expiration check: 0:03:36 ago on Thu Jan 16 14:49:17 2025.
procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo        : @System
Matched from:
Filename    : /usr/bin/ps
Provide    : /bin/ps

procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo        : amazonlinux
Matched from:
Filename    : /usr/bin/ps
Provide    : /bin/ps
```

在 Amazon 控制台中查看与 EKS 自动模式关联的资源

您可以使用 Amazon 控制台查看与 EKS 自动模式集群关联的资源的状态。

- EBS 卷

  - 通过搜索标签键 `eks:eks-cluster-name` 查看 EKS 自动模式卷
- 负载均衡器
  - 通过搜索标签键 `eks:eks-cluster-name` 查看 EKS 自动模式负载均衡器
- EC2 实例
  - 通过搜索标签键 `eks:eks-cluster-name` 查看 EKS 自动模式实例

查看您 Amazon 账户中的 IAM 错误

1. 导航到 CloudTrail 控制台
2. 从左侧导航窗格中选择“事件历史记录”
3. 应用错误代码筛选条件：
   - AccessDenied
   - UnauthorizedOperation
   - InvalidClientTokenId

查找与您的 EKS 集群相关的错误。使用错误消息更新 EKS 访问条目、集群 IAM 角色或节点 IAM 角色。您可能需要为这些角色附加一个具有 EKS 自动模式权限的新策略。

对容器组（pod）无法调度到自动模式节点进行问题排查

如果容器组（pod）处于 `Pending` 状态且没有调度到自动模式节点，请确认容器组（pod）或部署清单中是否有 `nodeSelector`。如果有 `nodeSelector`，请确保它在使用 `eks.amazonaws.com/compute-type: auto`，以便在由 EKS 自动模式生成的节点上进行调度。有关 EKS 自动模式所用节点标签的更多信息，请参阅the section called “控制部署”。

对节点无法加入集群进行问题排查

EKS 自动模式会自动使用正确的信息配置新的 EC2 实例以加入集群，包括集群端点和集群证书颁发机构（CA）。但是，这些实例可能仍无法作为节点加入 EKS 集群。运行以下命令以确定无法加入集群的实例：

1. 运行 `kubectl get nodeclaim` 以检查是否存在状态为 `Ready = False` 的 NodeClaims。

```
kubectl get nodeclaim
```

2. 运行 `kubectl describe nodeclaim <node_claim>` 并在状态下检查，找出阻止该节点加入集群的任何问题。

```
kubectl describe nodeclaim <node_claim>
```

常见错误消息：

`Error getting launch template configs`

如果您使用默认集群 IAM 角色权限在 `NodeClass` 中设置自定义标签，则可能会收到此错误消息。请参阅the section called “身份和访问权限”。

`Error creating fleet`

从 EC2 API 调用 `RunInstances` 调用时可能会出现某些授权问题。检查 Amazon CloudTrail 错误，并参阅the section called “自动模式集群 IAM 角色”以了解所需的 IAM 权限。

使用 `VPC Reachability Analyzer` 检测节点连接问题

> **Note**
>
> 您需要为运行 VPC Reachability Analyzer 的每项分析付费。有关定价详细信息，请参阅 Amazon VPC 定价。

实例无法加入集群的原因之一是网络连接问题，导致其无法访问 API 服务器。要诊断此问题，可以使用 VPC Reachability Analyzer 对无法加入集群的节点与 API 服务器之间的连接进行分析。您需要提供两条信息：

- 无法加入集群的节点的实例 ID
- Kubernetes API 服务器端点的 IP 地址

要获取实例 ID，需要在集群上创建工作负载，以使 EKS 自动模式启动 EC2 实例。此操作还会在集群中创建一个具有实例 ID 的 `NodeClaim` 对象。运行 `kubectl get nodeclaim -o yaml` 以打印集群中的所有 `NodeClaims`。每个 `NodeClaim` 都包含实例 ID 作为字段，并且再次包含在 providerID 中：

```
kubectl get nodeclaim -o yaml
```

示例输出如下。

```
    nodeName: i-01234567890123456
    providerID: aws:///us-west-2a/i-01234567890123456
```

您可以通过运行 kubectl get endpoint kubernetes -o yaml 来确定 Kubernetes API 服务器端点。地址位于地址字段中：

```
kubectl get endpoints kubernetes -o yaml
```

示例输出如下。

```
apiVersion: v1
kind: Endpoints
metadata:
  name: kubernetes
  namespace: default
subsets:
- addresses:
  - ip: 10.0.143.233
  - ip: 10.0.152.17
  ports:
  - name: https
    port: 443
    protocol: TCP
```

有了这两条信息，您就可以执行 s 分析了。首先导航到 Amazon Web Services 管理控制台中的 VPC Reachability Analyzer。

1. 单击“创建和分析路径”
2. 提供分析名称（例如“节点加入失败”）
3. 在“源类型”中选择“实例”
4. 输入故障节点的实例 ID 作为“源”
5. 在“路径目标”中选择“IP 地址”
6. 输入 API 服务器的其中一个 IP 地址作为“目标地址”
7. 展开“其他数据包标头配置部分”
8. 输入 443 的“目标端口”

9. 如果尚未选择“协议”，请选择 TCP

10.单击“创建和分析路径”

11.该分析可能需要几分钟才能完成。如果分析结果表明无法访问，则将指示网络路径中发生故障的位置，便于您解决问题。

在容器组（pod）之间共享卷

由于 EKS 自动模式节点在强制执行模式下使用 SELinux 进行配置，因而会将在同一节点上运行的容器组（pod）分隔开。启用 SELinux 后，大多数非特权容器组（pod）将自动应用自己的多类别安全性（MCS）标签。此 MCS 标签对于每个容器组（pod）都是唯一的，旨在确保一个容器组（pod）中的进程无法操作任何其他容器组（pod）或主机上的进程。即使带标签的容器组（pod）以 root 身份运行并有权访问主机文件系统，也无法操作文件、在主机上进行敏感的系统调用、访问容器运行时或获取 kubelet 的密钥材料。

因此，尝试在容器组（pod）之间共享数据时可能会遇到问题。例如，访问模式为 `ReadWriteOnce` 的 `PersistentVolumeClaim` 仍不允许多个容器组（pod）同时访问该卷。

要在容器组（pod）之间实现此共享，可以使用容器组（pod）的 `seLinuxOptions`，在这些容器组（pod）上配置相同的 MCS 标签。在此示例中，我们向容器组（pod）分配三个类别 `c123,c456,c789`。这不会与自动向节点上的容器组（pod）分配的任何类别发生冲突，因为这些容器组只会分配到两个类别。

```
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456,c789"
```

在控制面板日志中查看 Karpenter 事件

对于启用了控制面板日志的 EKS 集群，您可以通过查询日志来深入了解 Karpenter 的操作和决策过程。这对于排查与节点预置、扩展和终止相关的 EKS 自动模式问题特别有用。要查看与 Karpenter 相关的事件，请使用以下 CloudWatch Logs Insights 查询：

```
fields @timestamp, @message
| filter @logStream like /kube-apiserver-audit/
| filter @message like 'DisruptionBlocked'
or @message like 'DisruptionLaunching'
or @message like 'DisruptionTerminating'
```

```
or @message like 'DisruptionWaitingReadiness'
or @message like 'Unconsolidatable'
or @message like 'FailedScheduling'
or @message like 'NoCompatibleInstanceTypes'
or @message like 'NodeRepairBlocked'
or @message like 'Disrupted'
or @message like 'Evicted'
or @message like 'FailedDraining'
or @message like 'TerminationGracePeriodExpiring'
or @message like 'TerminationFailed'
or @message like 'FailedConsistencyCheck'
or @message like 'InsufficientCapacityError'
or @message like 'UnregisteredTaintMissing'
or @message like 'NodeClassNotReady'
| sort @timestamp desc
```

此查询会筛选 kube-apiserver 审计日志中与 Karpenter 相关的特定事件。这些事件包括各种中断状态、调度失败、容量问题和节点相关问题。通过分析这些日志，您可以更好地了解：

- Karpenter 采取某些操作的原因。
- 任何妨碍正常节点预置、扩展或终止的问题。
- 实例类型可能存在的容量或兼容性问题。
- 节点生命周期事件，例如中断、驱逐或终止。

要使用此查询，请执行以下操作：

1. 导航到 CloudWatch 控制台
2. 从左侧导航窗格中选择“Logs Insights”
3. 选择 EKS 集群控制面板日志的日志组
4. 将该查询粘贴到查询编辑器中
5. 根据需要调整时间范围
6. 运行查询

结果将显示与 Karpenter 相关事件的时间线，帮助您排查问题，并了解集群中 EKS 自动模式的行为。要查看特定节点上的 Karpenter 操作，可以在上述查询中添加以下行筛选器，从而指定实例 ID：

```
|filter @message like /[.replaceable]`i-12345678910123456`/
```

> **Note**
>
> 要使用此查询，必须在 EKS 集群上启用控制面板日志记录。如果尚未启用，请参阅the section called “控制面板日志”。

对自动模式涉及的控制器进行问题排查

如果您在使用控制器时遇到问题，请检查以下方面：

- 与该控制器关联的资源是否格式正确且有效。
- 是否为您的集群正确配置了 Amazon IAM 和 Kubernetes RBAC 资源。有关更多信息，请参阅 the section called “身份和访问权限”。

相关资源

阅读 Amazon re:Post 中的以下文章，了解问题排查高级步骤：

- 如何排查 EKS 自动模式中常见的扩展问题？
- 如何排查 Amazon EKS 自动模式下的自定义 nodepool 和 nodeclass 预置问题？
- 如何排查状态未知的 EKS 自动模式内置节点池问题？

查看 EKS 自动模式发布说明

本页面记录了 Amazon EKS 自动模式的更新内容。您可以定期查看此页面，了解有关功能、错误修复、已知问题及已弃用功能的公告。

要接收此特定文档页面的所有源文件更改通知，您可以通过 RSS 阅读器订阅以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/automode/auto-change.adoc.atom
```

2025 年 11 月 19 日

功能：为具有本地 NVMe 存储的 G、P 和 Trn 系列实例启用 Seekable OCI（SOCI）并行拉取和解压缩。这些具有 EKS 自动模式的实例系列始终使用 SOCI 并行拉取和解压缩，并且无需更改配置即可启用该模式。有关 SOCI 的更多信息，请参阅发布博客。

2025 年 11 月 19 日

功能：新增了对保持固定数量节点的静态容量节点池的支持。有关更多信息，请参阅 the section called “静态容量节点池”。

2025 年 10 月 23 日

功能：在美国地区拥有集群的用户现在可以通过在其 NodeClass 定义中指定 `spec.advancedSecurity.fips` 来请求使用兼容 FIPS 的 AMI。

2025 年 10 月 1 日

功能：EKS 自动模式现在支持将节点部署到 Amazon Local Zones。有关更多信息，请参阅 the section called “使用 Local Zones”。

2025 年 9 月 30 日

功能：为 NodeClass `spec.instanceProfile` 添加了对 instanceProfile 的支持，该支持与 `spec.role` 字段互斥。

2025 年 9 月 29 日

EKS 自动模式目前不支持 DRA。

2025 年 9 月 10 日

琐事：从自动模式计算控制器触发的事件现在将使用名称 `eks-auto-mode/compute` 而非 `karpenter`。

2025 年 8 月 24 日

错误修复：若 VPC 使用的 DHCP 选项集包含带有大写字母的自定义域名，会导致节点无法加入集群，原因是会生成无效的主机名。此问题已得到解决，包含大写字母的域名现在可以正常使用。

2025 年 8 月 15 日

错误修复：容器组身份代理现在只能侦听 IPv4 EKS 集群中的 IPv4 链路本地地址，以避免出现容器组（pod）无法访问 IPv6 地址的问题。

2025 年 8 月 6 日

功能：在 NodeClass `spec.advancedNetworking.associatePublicIPAddress` 中新增配置，可用于防止为 EKS 自动模式节点分配公有 IP 地址

2025 年 6 月 30 日

功能：自动模式 NodeClass 现在使用配置的自定义 KMS 密钥，对实例的只读根卷以及读/写数据卷进行加密。以前，自定义 KMS 密钥仅用于加密数据卷。

2025 年 6 月 20 日

功能：支持控制将工作负载部署到 EC2 按需容量预留（ODCR）。此功能向 NodeClass 添加可选密钥 `capacityReservationSelectorTerms`，允许您显式控制工作负载使用的 ODCR。有关更多信息，请参阅 the section called “控制容量预留”。

2025 年 6 月 13 日

功能：对 `NodeClass` 中单独的容器组（pod）子网的支持。这将添加可选键 `` `podSubnetSelectorTerms `` 和 `podSecurityGroupSelectorTerms`，以便为容器组（pod）设置子网和安全组。有关更多信息，请参阅 the section called “容器组（pod）的子网选择”。

2025 年 4 月 30 日

功能：支持 `NodeClass` 中的转发网络代理。这将添加可选的 `advancedNetworking` 键来设置 HTTPS 代理。有关更多信息，请参阅 the section called “节点类规范”。

2025 年 4 月 18 日

功能：支持通过单播 DNS 解析 .local 域（通常为组播 DNS 保留）。

2025 年 4 月 11 日

功能：向 `NodeClass` 新增了 `certificateBundles` 和 `ephemeralStorage.kmsKeyID`。有关更多信息，请参阅 the section called “节点类规范”。

功能：提高了映像拉取速度，特别针对具有本地实例存储的实例类型，可以利用更快的映像解压缩功能。

错误修复：解决了导致 FailedCreatePodSandBox 的争用情况（拨号时出现错误：dial tcp 127.0.0.1:50051: connect: connection refused to sometimes occur for Pods scheduling to a Node immediately at startup. ）

2025 年 4 月 4 日

功能：将 `registryPullQPS` 从 5 提升至 25，将 `registryBurst` 从 10 提升至 50，从而减少了客户端强制执行的映像拉取限制 (`Failed to pull image xyz: pull QPS exceeded`)

2025 年 3 月 31 日

错误修复：修复了一个问题，即当 Core DNS 容器组（pod）在自动模式节点上运行时，该节点上容器组（pod）的 DNS 查询会直接访问 Core DNS 容器组（pod）而不是节点本地 DNS 服务器。来自自动模式节点上容器组（pod）的 DNS 查询将始终指向节点本地 DNS。

2025 年 3 月 21 日

错误修复：现在，自动模式节点可在集群中未安装 `kube-dns` 服务的情况下，正确解析 `kube-dns.kube-system.svc.cluster.local`。GitHub 问题 #2546 得以解决。

2025 年 3 月 14 日

功能：在 IPv6 集群中启用了 IPv4 出口。现在，从 IPv6 自动模式集群输出的 IPv4 流量将自动转换为节点主 ENI 的 v4 地址。

Amazon EKS 集群生命周期与配置

本节提供有关 Kubernetes 集群完整生命周期管理以及配置集群的不同方法的深入指导。内容涵盖集群创建、监控集群运行状况、更新 Kubernetes 版本和删除集群。本节还包括高级主题，内容涉及如何配置端点访问权限、启用/禁用 Windows 支持、设置私有集群、实现自动扩展，以及如何在多可用区设置中使用可用区转移进行流量重定向以增强恢复能力。

主题

- 创建 Amazon EKS 自动模式集群
- 创建一个 Amazon EKS 集群。
- Amazon EKS 预置式控制面板
- 利用集群见解为 Kubernetes 版本升级做好准备并对错误配置进行问题排查
- 将现有集群更新到新的 Kubernetes 版本
- 删除集群
- 集群 API 服务器端点
- 在 EKS 集群上部署 Windows 节点
- 禁用 Windows 支持
- 部署具有有限互联网访问权限的私有集群
- 使用 Karpenter 和 Cluster Autoscaler 扩展集群计算
- 了解 Amazon EKS 中的 Amazon 应用程序恢复控制器（ARC）可用区转移
- 启用 EKS 可用区转移以避开损坏的可用区

创建 Amazon EKS 自动模式集群

本主题提供了有关使用高级配置选项创建 Amazon EKS 自动模式集群的详细说明。其中包括先决条件、联网选项和附加组件配置等。整个过程包括设置 IAM 角色、配置集群设置、指定联网参数和选择附加组件等。用户可以使用 Amazon Web Services 管理控制台或 Amazon CLI 创建集群，这两种方法都有详细的分步指南。

如果用户需要不太复杂的设置过程，请参阅以下内容，了解简化的集群创建步骤：

- the section called “eksctl CLI”
- the section called “ Amazon CLI”
- the section called “管理控制台”

本高级配置指南适用于需要更精细地控制 EKS 自动模式集群设置，并且熟悉 Amazon EKS 概念和要求的用户。在开始高级配置之前，请确保您已满足所有先决条件，并全面了解 EKS 自动模式集群的联网和 IAM 要求。

EKS 自动模式需要额外的 IAM 权限。有关更多信息，请参阅：

- the section called “EKS 自动模式集群的 IAM 角色”
- the section called “身份和访问权限”

> **Note**
>
> 要创建未启用 EKS 自动模式的集群，请参阅the section called “创建集群”。
> 本主题介绍高级配置。如果您需要学习使用 EKS 自动模式，请参阅the section called “创建集群”。

先决条件

- 满足 Amazon EKS 要求的现有 VPC 和子网。在部署集群用于生产用途前，我们建议您彻底了解 VPC 和子网要求。如果您没有 VPC 和子网，则可以使用 Amazon EKS 提供的 Amazon CloudFormation 模板创建它们。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version`。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。
- 一个具有创建和修改 EKS 和 IAM 资源的权限的 IAM 主体。

创建集群 – Amazon 控制台

1. 打开 Amazon EKS 控制台。
2. 请选择 Add cluster（添加集群），然后选择 Create（创建）。

3. 在配置选项下，选择自定义配置。
   - 本主题介绍自定义配置。有关快速配置的信息，请参阅the section called “管理控制台”。
4. 确认已启用使用 EKS 自动模式。
   - 本主题介绍创建启用 EKS 自动模式的集群。有关创建不启用 EKS 自动模式的集群的更多信息，请参阅the section called “创建集群”。
5. 在 Configure cluster（配置集群）页面上，输入以下字段：
   - Name（名称）– 集群的名称。名称只能包含字母数字字符（区分大小写）、连字符和下划线。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   - 集群 IAM 角色 – 选择您创建的 Amazon EKS 集群 IAM 角色，以允许 Kubernetes 控制面板来代表您管理 Amazon 资源。如果您之前尚未为 EKS 自动模式创建集群 IAM 角色，请在 IAM 控制台中选择创建推荐角色按钮，以创建具有所需权限的角色。
   - Kubernetes version（Kubernetes 版本）– 要用于集群的 Kubernetes 的版本。建议选择最新版本，除非您需要早期版本。
   - 升级策略 – 要为集群设置的 Kubernetes 版本策略。如果您希望集群仅根据标准支持版本运行，则可以选择标准。如果您希望集群在某个版本的标准支持终止时进入扩展支持，则可以选择扩展。如果您选择的 Kubernetes 版本当前处于延期支持状态，则无法选择标准支持选项。
6. 在“配置集群”页面的自动模式计算部分中，输入以下字段：
   - 节点池 – 确定是否要使用内置节点池。有关更多信息，请参阅 the section called “检查内置节点池”。
   - 节点 IAM 角色 – 如果您启用任何内置节点池，则需要选择节点 IAM 角色。EKS 自动模式会将此角色分配给新节点。创建集群后，您无法更改此值。如果您之前尚未为 EKS 自动模式创建节点 IAM 角色，请选择“创建推荐角色”按钮，以创建具有所需权限的角色。有关该角色的更多信息，请参阅the section called “身份和访问权限”。
7. 在“配置集群”页面的集群访问权限部分中，输入以下字段：
   - 引导集群管理员访问权限 – 集群创建者自动成为 Kubernetes 管理员。如果要禁用此功能，请选择不允许集群管理员访问权限。
   - 集群身份验证模式 – EKS 自动模式需要 EKS 访问条目，即 EKS API 身份验证模式。您也可以通过选择 EKS API 和 ConfigMap 来启用 `ConfigMap` 身份验证模式。
8. 输入“配置集群”页面上的剩余字段：
   - Secrets encryption（密钥加密）–（可选）选择此选项以使用 KMS 密钥启用 Kubernetes 密钥的密钥加密。您也可以在创建集群后启用此功能。在启用此功能之前，请确保您熟悉the section called “启用密钥加密”中的信息。

- ARC 可用区转移 – EKS 自动模式不支持 ARC 可用区转移。
- Tags（标签）– （可选）向集群添加任何标签。有关更多信息，请参阅 the section called “标记您的资源”。

  完成此页面后，请选择下一步。

9. 在 Specify networking (指定联网) 页面上，为以下字段选择值：

- VPC – 选择符合 Amazon EKS VPC 要求的现有 VPC 以在其中创建集群。在选择 VPC 之前，我们建议您熟悉 the section called “VPC 和子网要求” 中的所有要求和考虑因素。集群创建后，您无法更改要使用的 VPC。如果没有列出任何 VPC，则需要先创建一个。有关更多信息，请参阅 the section called “创建 VPC”。
- Subnets（子网）– 预设情况下，已预先选中在之前字段中指定的 VPC 中的所有可用子网。您必须至少选择两个子网。

  您选择的子网必须符合 Amazon EKS 子网要求。在选择子网之前，我们建议您熟悉所有的 Amazon EKS VPC 以及子网要求和注意事项。

  Security groups（安全组）–（可选）指定您希望 Amazon EKS 将之与其创建的网络接口关联的一个或多个安全组。

  无论您是否选择任何安全组，Amazon EKS 都会创建一个安全组，以实现集群和 VPC 之间的通信。Amazon EKS 将此安全组以及您选择的任何安全组与它创建的网络接口关联起来。有关 Amazon EKS 创建的集群安全组的更多信息，请参阅 the section called “安全组要求”。您可以修改 Amazon EKS 创建的集群安全组中的规则。
- 选择集群 IP 地址系列 - 您可以选择 IPv4 和 IPv6。

  默认情况下，Kubernetes 会将 `IPv4` 地址分配给容器组（pod）和服务。在决定使用 `IPv6` 系列前，请确保您熟悉 VPC 要求和注意事项、the section called “子网要求和注意事项”、the section called “安全组要求”和 the section called “IPv6” 主题中的所有注意事项和要求。如果选择 `IPv6` 系列，则不同于可为其指定地址范围的 `IPv4` 系列，您无法指定从中分配 `IPv6` 服务地址的 Kubernetes 的地址范围。Kubernetes 从唯一的本地地址范围 (`fc00::/7`) 分配服务地址。
- （可选）选择 Configure Kubernetes Service IP address range（配置 Kubernetes 服务 IP 地址范围），然后指定服务 **`IPv4`** 范围。

  指定自己的范围有助于防止 Kubernetes 服务与对等或连接到 VPC 的其他网络之间发生冲突。使用 CIDR 表示法输入范围。例如：`10.2.0.0/16`。

  此 CIDR 块必须满足以下要求：

- 处于以下范围之一：10.0.0.0/8、172.16.0.0/12 或 192.168.0.0/16。
- 具有最小大小 /24 和最大大小 /12。
- 与您的 Amazon EKS 资源的 VPC 范围不重叠。

您只能在使用 IPv4 地址系列和创建集群时指定此选项。如果没有指定此选项，Kubernetes 会从 10.100.0.0/16 或 172.20.0.0/16 CIDR 块分配服务 IP 地址。

- 对于集群端点访问中，选择一个选项。创建集群后，您可以更改此选项。在选择非默认选项之前，请务必熟悉这些选项及其影响。有关更多信息，请参阅 the section called “集群终端节点访问”。

完成此页面后，请选择下一步。

10. （可选）在配置可观测性页面上，选择要开启的指标和控制面板日志记录选项。默认情况下，每种日志类型都处于关闭状态。
    - 有关 Prometheus 指标选项的更多信息，请参阅the section called “第 1 步：开启 Prometheus 指标”。
    - 有关控制面板日志记录选项的更多信息，请参阅 the section called “控制面板日志”。
    - 完成此页面后，请选择下一步。
11. 在 Select add-ons（选择附加组件）页面上，选择要添加到集群的附加组件。您可以根据需要选择任意数量的 Amazon EKS 附加组件和 Amazon Marketplace 附加组件。如果未列出要安装的 Amazon Marketplace 附加组件，则您可以单击页码编号查看更多页面结果或通过在搜索框中输入文本来搜索可用的 Amazon Marketplace 附加组件。您也可以按类别、供应商或定价模式进行搜索，然后从搜索结果中选择附加组件。创建集群时，您可以查看、选择和安装任何支持 EKS 容器组身份的附加组件，详情请参阅the section called “容器组身份”。
    - EKS 自动模式会自动执行某些附加组件的功能。如果您计划将 EKS 托管式节点组部署到 EKS 自动模式集群，请选择其他 Amazon EKS 附加组件并检查选项。可能需要安装诸如 CoreDNS 和 kube-proxy 之类的附加组件。EKS 仅在自主管理型节点和节点组上安装本节中的附加组件。
    - 完成此页面后，请选择下一步。
12. 在配置选定插件设置页面上，选择要安装的版本。创建集群后，您可以随时更新到更高版本。

    对于支持 EKS 容器组身份的附加组件，您可以使用控制台自动生成角色，其中包含专门为该附加组件预先填充的名称、Amazon 托管策略和信任策略。您可以重复使用现有角色或为支持的附加组件创建新角色。有关使用控制台为支持 EKS 容器组身份的附加组件创建角色的步骤，请参阅the section called “创建附加组件（Amazon 控制台）”。如果附加组件不支持 EKS 容器组身份，则系统会显示一条消息，说明如何在创建集群后使用向导为服务账户（IRSA）创建 IAM 角色。

创建集群后，您可以更新每个附加组件的配置。有关配置附加组件的更多信息，请参阅the section called “更新附加组件”。完成此页面后，请选择下一步。

13. 在 Review and create (审核和创建) 页面上，审核您在之前页面输入或选择的信息。如果需要进行更改，请选择 Edit（编辑）。在您感到满意后，选择创建。Status（状态）字段在预置集群时显示 CREATING（正在创建）。

> **Note**
>
> 您可能会收到一个错误，指示请求中的可用区之一没有足够容量来创建 Amazon EKS 集群。如果发生这种情况，错误输出将包含可支持新集群的可用区。再次尝试使用至少两个位于您账户中支持的可用区的子网创建集群。有关更多信息，请参阅 the section called “容量不足”。

集群预配置需要几分钟时间。

创建集群 – Amazon CLI

以下 CLI 说明介绍创建 IAM 资源和创建集群的过程。

创建 EKS 自动模式集群 IAM 角色

第 1 步：创建信任策略

创建信任策略以允许 Amazon EKS 服务代入该角色。将策略另存为 `trust-policy.json`：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
```

```
  ]
}
```

第 2 步：创建 IAM 角色

使用信任策略创建集群 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoClusterRole \
    --assume-role-policy-document file://trust-policy.json
```

第 3 步：记下角色 ARN

检索并保存新角色的 ARN，以便在后续步骤中使用：

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

第 4 步：附加必需的策略

将以下 Amazon 托管式策略附加到集群 IAM 角色以授予必要的权限：

AmazonEKSClusterPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSComputePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy：

```
aws iam attach-role-policy \
```

```
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSNetworkingPolicy
```

创建 EKS 自动模式 节点 IAM 角色

第 1 步：创建信任策略

创建信任策略以允许 Amazon EKS 服务代入该角色。将策略另存为 `node-trust-policy.json`：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

第 2 步：创建节点 IAM 角色

使用上一步中的 node-trust-policy.json 文件来定义哪些实体可以代入该角色。运行以下命令以创建节点 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

第 3 步：记下角色 ARN

创建角色后，检索并保存节点 IAM 角色的 ARN。在后续步骤中，您将需要此 ARN。使用以下命令来获取 ARN：

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

第 4 步：附加必需的策略

将以下 Amazon 托管式策略附加到节点 IAM 角色，以提供必要的权限：

AmazonEKSWorkerNodeMinimalPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

创建集群

1. 使用以下命令创建集群。在运行命令之前，进行以下替换：
   - 将 *region-code* 替换为您要在其中创建集群的 Amazon 区域。
   - 将 *my-cluster* 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）、连字符和下划线。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   - 将 *1.30* 替换为任何 Amazon EKS 支持的版本。
   - 请将 *111122223333* 替换为您的账户 ID
   - 如果您为集群和节点角色创建了其他名称的 IAM 角色，请相应替换 ARN。
   - 将 `subnetIds` 的值替换为您自己的值。您还可以添加其他 ID。您必须指定至少两个子网 ID。

     您选择的子网必须符合 Amazon EKS 子网要求。在选择子网之前，我们建议您熟悉所有的 Amazon EKS VPC 以及子网要求和注意事项。
   - 如果您不想指定安全组 ID，请从命令中删除 `,securityGroupIds=sg-<ExampleID1>`。如果您想指定一个或多个安全组 ID，请将 `securityGroupIds` 的值替换为您自己的值。您还可以添加其他 ID。

     无论您是否选择任何安全组，Amazon EKS 都会创建一个安全组，以实现集群和 VPC 之间的通信。Amazon EKS 将此安全组以及您选择的任何安全组与它创建的网络接口关联起来。有关

Amazon EKS 创建的集群安全组的更多信息，请参阅 the section called “安全组要求”。您可以修改 Amazon EKS 创建的集群安全组中的规则。

```
aws eks create-cluster \
  --region region-code \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --role-arn arn:aws-cn:iam::111122223333:role/AmazonEKSAutoClusterRole \
  --resources-vpc-config '{"subnetIds": ["subnet-ExampleID1","subnet-ExampleID2"],
 "securityGroupIds": ["sg-ExampleID1"], "endpointPublicAccess": true,
 "endpointPrivateAccess": true}' \
  --compute-config '{"enabled": true, "nodeRoleArn": "arn:aws-
cn:iam::111122223333:role/AmazonEKSAutoNodeRole", "nodePools": ["general-purpose",
 "system"]}' \
  --kubernetes-network-config '{"elasticLoadBalancing": {"enabled": true}}' \
  --storage-config '{"blockStorage": {"enabled": true}}' \
  --access-config '{"authenticationMode": "API"}'
```

> **Note**
>
> 您可能会收到一个错误，指示请求中的可用区之一没有足够容量来创建 Amazon EKS 集群。如果发生这种情况，错误输出将包含可支持新集群的可用区。再次尝试使用至少两个位于您账户中支持的可用区的子网创建集群。有关更多信息，请参阅 the section called “容量不足”。

以下是可选设置，如果需要，必须将这些设置添加到上一个命令中。您只能在创建集群时启用这些选项，而不能在创建集群后启用。

- 如果您想指定 Kubernetes 从中分配服务 ID 地址的 IPv4 无类别域间路由（CIDR）块，您必须通过将 `--kubernetes-network-config serviceIpv4Cidr=<cidr-block>` 添加到以下命令中来指定它。

  指定自己的范围有助于防止 Kubernetes 服务与对等或连接到 VPC 的其他网络之间发生冲突。使用 CIDR 表示法输入范围。例如：`10.2.0.0/16`。

  此 CIDR 块必须满足以下要求：

  - 处于以下范围之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。
  - 具有最小大小 `/24` 和最大大小 `/12`。

- 与您的 Amazon EKS 资源的 VPC 范围不重叠。

  您只能在使用 IPv4 地址系列和创建集群时指定此选项。如果没有指定此选项，Kubernetes 会从 10.100.0.0/16 或 172.20.0.0/16 CIDR 块分配服务 IP 地址。

- 如果您创建集群并希望集群分配 IPv6 地址而不是 IPv4 地址给容器组（pod）和服务，请将 --kubernetes-network-config ipFamily=ipv6 添加到以下命令。

  默认情况下，Kubernetes 会将 IPv4 地址分配给容器组（pod）和服务。在决定使用 IPv6 系列前，请确保您熟悉 VPC 要求和注意事项、the section called “子网要求和注意事项”、the section called “安全组要求”和 the section called “IPv6” 主题中的所有注意事项和要求。如果选择 IPv6 系列，则不同于可为其指定地址范围的 IPv4 系列，您无法指定从中分配 IPv6 服务地址的 Kubernetes 的地址范围。Kubernetes 从唯一的本地地址范围 (fc00::/7) 分配服务地址。

2. 预置集群需要几分钟时间。可使用以下命令查询集群的状态。

```
aws eks describe-cluster --region region-code --name my-cluster --query
 "cluster.status"
```

后续步骤

- the section called “使用 kubectl 访问集群”
- the section called “访问条目”
- 为集群启用密钥加密。
- 为集群配置日志记录。
- 将节点添加到集群。

创建一个 Amazon EKS 集群。

> **Note**
>
> 本主题旨在介绍创建未启用 EKS 自动模式的 EKS 集群。
> 有关创建 EKS 自动模式集群的详细说明，请参阅the section called “创建自动集群”。
> 要学习使用 EKS 自动模式，请参阅the section called “创建集群（EKS 自动模式）”。

本主题概述了可用选项，并介绍了创建 Amazon EKS 集群时需要考虑的内容。如果需要创建将本地基础设施作为节点计算的集群，请参阅the section called “创建集群”。如果您是首次创建 Amazon EKS 集群，我们建议您按照 开始使用 中的指南之一操作。这些指南可帮助您创建一个简单的默认集群，而无需扩展到所有可用选项。

先决条件

- 满足 Amazon EKS 要求的现有 VPC 和子网。在部署集群用于生产用途前，我们建议您彻底了解 VPC 和子网要求。如果您没有 VPC 和子网，则可以使用 Amazon EKS 提供的 Amazon CloudFormation 模板创建它们。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 具有 `create` 和 `describe` Amazon EKS 集群权限的 IAM 主体。有关更多信息，请参阅the section called “在 Outpost 上创建本地 Kubernetes 集群”和the section called “列出或描述所有集群”。

第 1 步：创建集群 IAM 角色

1. 如果您已经拥有集群 IAM 角色，或者您将使用 `eksctl` 创建集群，则可以跳过此步骤。默认情况下，`eksctl` 会为您创建角色。
2. 运行以下命令以创建 IAM 信任策略 JSON 文件。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 创建 Amazon EKS 集群 IAM 角色。如有必要，使用您在上一步中将文件写入到的计算机上的路径为 *eks-cluster-role-trust-policy.json* 添加前言。该命令将您在上一步中创建的信任策略与角色关联。要创建 IAM 角色，必须为正在创建角色的 IAM 主体分配 `iam:CreateRole` 操作（权限）。

```
aws iam create-role --role-name myAmazonEKSClusterRole --assume-role-policy-document
 file://"eks-cluster-role-trust-policy.json"
```

4. 您可以分配 Amazon EKS 托管策略或创建自己的自定义策略。有关必须在自定义策略中使用的最低权限，请参阅 the section called “集群 IAM 角色”。

   将名为 AmazonEKSClusterPolicy 的 Amazon EKS 托管 IAM 策略附加到角色。要将 IAM 策略附加到某个 IAM 主体，必须为附加该策略的主体分配以下 IAM 操作（权限）之一：`iam:AttachUserPolicy` 或 `iam:AttachRolePolicy`。

```
aws iam attach-role-policy --policy-arn arn:aws-cn:iam::aws:policy/
AmazonEKSClusterPolicy --role-name myAmazonEKSClusterRole
```

服务相关角色

Amazon EKS 会自动创建一个名为 `AWSServiceRoleForAmazonEKS` 的服务相关角色。

此角色是集群 IAM 角色的补充。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。该角色允许 Amazon EKS 管理您账户中的集群。有关更多信息，请参阅 the section called “集群 角色”。

您用来创建 EKS 集群的 IAM 身份必须具有创建该服务相关角色的权限。这包括 `iam:CreateServiceLinkedRole` 权限。

如果该服务相关角色尚不存在，并且您当前的 IAM 角色没有足够的权限创建该角色，则集群创建操作将会失败。

第 2 步：创建集群

您可以使用以下工具来创建集群：

- eksctl
- Amazon Web Services 管理控制台
- Amazon CLI

创建集群 – eksctl

1. 您需要在设备或 Amazon CloudShell 上安装 `0.215.0` 版或更高版本的 `eksctl` 命令行工具。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的 Installation。
2. 在默认 Amazon 区域中，使用 Amazon EKS 默认 Kubernetes 版本创建 Amazon EKS IPv4 集群。在运行命令之前，进行以下替换：
3. 将 *region-code* 替换为您要在其中创建集群的 Amazon 区域。
4. 将 *my-cluster* 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
5. 将 *1.33* 替换为任何 Amazon EKS 支持的版本。
6. 更改 `vpc-private-subnets` 的值以满足您的要求。您还可以添加其他 ID。您必须指定至少两个子网 ID。如果您想要指定公有子网，您可以将 `--vpc-private-subnets` 更改为 `--vpc-public-subnets`。公有子网有一个与互联网网关的路由相关联的路由，但私有子网没有关联的路由表。我们建议尽可能使用私有子网。

   您选择的子网必须符合 Amazon EKS 子网要求。在选择子网之前，我们建议您熟悉所有的 Amazon EKS VPC 以及子网要求和注意事项。
7. 运行如下命令：

   ```
   eksctl create cluster --name my-cluster --region region-code --version 1.33 --vpc-private-subnets subnet-ExampleID1,subnet-ExampleID2 --without-nodegroup
   ```

   集群预配置需要几分钟时间。在创建集群时，将显示几行输出。输出的最后一行类似于以下示例行。

   ```
   [#]  EKS cluster "my-cluster" in "region-code" region is ready
   ```
8. 继续the section called “第 3 步：更新 kubeconfig”

可选设置

要查看在使用 eksctl 创建集群时可指定的大多数选项，请使用 eksctl create cluster --help 命令。要查看所有可用的选项，请使用 config 文件。有关更多信息，请参阅 eksctl 文档中的使用配置文件和配置文件架构。您可以在 GitHub 上查找配置文件示例。

以下是可选设置，如果需要，必须将这些设置添加到上一个命令中。您只能在创建集群时启用这些选项，而不能在创建集群后启用。如果您需要指定这些选项，则必须使用 eksctl 配置文件创建集群，然后指定设置，而不是使用上一个命令。

- 如果您想指定 Amazon EKS 分配给它创建的网络接口的一个或多个安全组，请指定 securityGroup 选项。

  无论您是否选择任何安全组，Amazon EKS 都会创建一个安全组，以实现集群和 VPC 之间的通信。Amazon EKS 将此安全组以及您选择的任何安全组与它创建的网络接口关联起来。有关 Amazon EKS 创建的集群安全组的更多信息，请参阅 the section called “安全组要求”。您可以修改 Amazon EKS 创建的集群安全组中的规则。
- 如果您想指定 Kubernetes 从中分配服务 IP 地址的 IPv4 无类别域间路由块（CIDR），请指定 serviceIPv4CIDR 选项。

  指定自己的范围有助于防止 Kubernetes 服务与对等或连接到 VPC 的其他网络之间发生冲突。使用 CIDR 表示法输入范围。例如：10.2.0.0/16。

  此 CIDR 块必须满足以下要求：

  - 处于以下范围之一：10.0.0.0/8、172.16.0.0/12 或 192.168.0.0/16。
  - 具有最小大小 /24 和最大大小 /12。
  - 与您的 Amazon EKS 资源的 VPC 范围不重叠。

    您只能在使用 IPv4 地址系列和创建集群时指定此选项。如果没有指定此选项，Kubernetes 会从 10.100.0.0/16 或 172.20.0.0/16 CIDR 块分配服务 IP 地址。
- 如果您要创建集群并希望集群将 IPv6 地址（而不是 IPv4 地址）分配到容器组（pod）和服务，请指定 ipFamily 选项。

  默认情况下，Kubernetes 会将 IPv4 地址分配给容器组（pod）和服务。在决定使用 IPv6 系列前，请确保您熟悉 the section called “VPC 要求和注意事项”、the section called “子网要求和注意事项”、the section called “安全组要求” 和 the section called “IPv6” 主题中的所有考虑因素和要求。如果选择 IPv6 系列，则不同于可为其指定地址范围的 IPv4 系列，您无法指定从中分配 IPv6 服务地址的 Kubernetes 的地址范围。Kubernetes 从唯一的本地地址范围 (fc00::/7) 分配服务地址。

创建集群 – Amazon 控制台

1. 打开 Amazon EKS 控制台。
2. 请选择 Add cluster（添加集群），然后选择 Create（创建）。
3. 在配置选项下，选择自定义配置
   - 有关快速创建启用 EKS 自动模式的集群的信息，请参阅the section called “管理控制台”。
4. 在 EKS 自动模式下，将使用 EKS 自动模式切换为关闭。
   - 有关创建具有自定义配置的 EKS 自动模式集群的信息，请参阅the section called “创建自动集群”。
5. 在 Configure cluster（配置集群）页面上，输入以下字段：
   - Name（名称）– 集群的名称。名称只能包含字母数字字符（区分大小写）、连字符和下划线。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   - 集群 IAM 角色 – 选择您创建的 Amazon EKS 集群 IAM 角色，以允许 Kubernetes 控制面板来代表您管理 Amazon 资源。
   - Kubernetes version（Kubernetes 版本）– 要用于集群的 Kubernetes 的版本。建议选择最新版本，除非您需要早期版本。
   - 支持类型 – 要为集群设置的 Kubernetes 版本策略。如果您希望集群仅根据标准支持版本运行，则可以选择标准支持。如果您希望集群在某个版本的标准支持终止时进入扩展支持，则可以选择扩展支持。如果您选择的 Kubernetes 版本当前处于延期支持状态，则无法选择标准支持选项。
   - Secrets encryption（密钥加密）–（可选）选择此选项以使用 KMS 密钥启用 Kubernetes 密钥的密钥加密。您也可以在创建集群后启用此功能。在启用此功能之前，请确保您熟悉the section called “启用密钥加密”中的信息。
   - Tags（标签）–（可选）向集群添加任何标签。有关更多信息，请参阅 the section called “标记您的资源”。
   - ARC 可用区转移 –（可选）您可以使用 Route53 应用程序恢复控制器来缓解受损的可用区。有关更多信息，请参阅 the section called “了解可用区转移”。
6. 在“配置集群”页面的集群访问权限部分中，输入以下字段：
   - 引导集群管理员访问权限 – 集群创建者自动成为 Kubernetes 管理员。如果要禁用此功能，请选择不允许集群管理员访问权限。
   - 集群身份验证模式 – 确定您希望如何向 IAM 用户和角色授予 Kubernetes API 访问权限。有关更多信息，请参阅 the section called “设置集群身份验证模式”。

     完成此页面后，请选择下一步。

7. 在 Specify networking (指定联网) 页面上，为以下字段选择值：

   - VPC – 选择符合 Amazon EKS VPC 要求的现有 VPC 以在其中创建集群。在选择 VPC 之前，我们建议您熟悉 the section called “VPC 和子网要求” 中的所有要求和考虑因素。集群创建后，您无法更改要使用的 VPC。如果没有列出任何 VPC，则需要先创建一个。有关更多信息，请参阅 the section called “创建 VPC”。
   - Subnets（子网）– 预设情况下，已预先选中在之前字段中指定的 VPC 中的所有可用子网。您必须至少选择两个子网。

     您选择的子网必须符合 Amazon EKS 子网要求。在选择子网之前，我们建议您熟悉所有的 Amazon EKS VPC 以及子网要求和注意事项。

     Security groups（安全组）–（可选）指定您希望 Amazon EKS 将之与其创建的网络接口关联的一个或多个安全组。

     无论您是否选择任何安全组，Amazon EKS 都会创建一个安全组，以实现集群和 VPC 之间的通信。Amazon EKS 将此安全组以及您选择的任何安全组与它创建的网络接口关联起来。有关 Amazon EKS 创建的集群安全组的更多信息，请参阅 the section called “安全组要求”。您可以修改 Amazon EKS 创建的集群安全组中的规则。
   - 选择集群 IP 地址系列 - 您可以选择 IPv4 和 IPv6。

     默认情况下，Kubernetes 会将 `IPv4` 地址分配给容器组（pod）和服务。在决定使用 `IPv6` 系列前，请确保您熟悉 VPC 要求和注意事项、the section called “子网要求和注意事项”、the section called “安全组要求”和 the section called “IPv6” 主题中的所有注意事项和要求。如果选择 `IPv6` 系列，则不同于可为其指定地址范围的 `IPv4` 系列，您无法指定从中分配 `IPv6` 服务地址的 Kubernetes 的地址范围。Kubernetes 从唯一的本地地址范围 (`fc00::/7`) 分配服务地址。
   - （可选）选择 Configure Kubernetes Service IP address range（配置 Kubernetes 服务 IP 地址范围），然后指定服务 **`IPv4`** 范围。

     指定自己的范围有助于防止 Kubernetes 服务与对等或连接到 VPC 的其他网络之间发生冲突。使用 CIDR 表示法输入范围。例如：`10.2.0.0/16`。

     此 CIDR 块必须满足以下要求：

     - 处于以下范围之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。
     - 具有最小大小 `/24` 和最大大小 `/12`。
     - 与您的 Amazon EKS 资源的 VPC 范围不重叠。

您只能在使用 IPv4 地址系列和创建集群时指定此选项。如果没有指定此选项，Kubernetes 会从 10.100.0.0/16 或 172.20.0.0/16 CIDR 块分配服务 IP 地址。

- 对于集群端点访问中，选择一个选项。创建集群后，您可以更改此选项。在选择非默认选项之前，请务必熟悉这些选项及其影响。有关更多信息，请参阅 the section called “集群终端节点访问”。

完成此页面后，请选择下一步。

8. （可选）在配置可观测性页面上，选择要开启的指标和控制面板日志记录选项。默认情况下，每种日志类型都处于关闭状态。
   - 有关 Prometheus 指标选项的更多信息，请参阅the section called “第 1 步：开启 Prometheus 指标”。
   - 有关控制面板日志记录选项的更多信息，请参阅 the section called “控制面板日志”。

   完成此页面后，请选择下一步。
9. 在 Select add-ons（选择附加组件）页面上，选择要添加到集群的附加组件。预先选择特定附加组件。您可以根据需要选择任意数量的 Amazon EKS 附加组件和 Amazon Marketplace 附加组件。如果未列出要安装的 Amazon Marketplace 附加组件，则您可以单击页码编号查看更多页面结果或通过在搜索框中输入文本来搜索可用的 Amazon Marketplace 附加组件。您也可以按类别、供应商或定价模式进行搜索，然后从搜索结果中选择附加组件。创建集群时，您可以查看、选择和安装任何支持 EKS 容器组身份的附加组件，详情请参阅the section called “容器组身份”。

   完成此页面后，请选择下一步。

   系统会默认安装 Amazon VPC CNI、CoreDNS 和 kube-proxy 等插件。如果您禁用任何默认插件，则可能会影响您运行 Kubernetes 应用程序的能力。
10. 在配置选定插件设置页面上，选择要安装的版本。创建集群后，您可以随时更新到更高版本。

    对于支持 EKS 容器组身份的附加组件，您可以使用控制台自动生成角色，其中包含专门为该附加组件预先填充的名称、Amazon 托管策略和信任策略。您可以重复使用现有角色或为支持的附加组件创建新角色。有关使用控制台为支持 EKS 容器组身份的附加组件创建角色的步骤，请参阅the section called “创建附加组件（Amazon 控制台）”。如果附加组件不支持 EKS 容器组身份，则系统会显示一条消息，说明如何在创建集群后使用向导为服务账户（IRSA）创建 IAM 角色。

    创建集群后，您可以更新每个附加组件的配置。有关配置附加组件的更多信息，请参阅the section called “更新附加组件”。完成此页面后，请选择下一步。

11.在 Review and create (审核和创建) 页面上，审核您在之前页面输入或选择的信息。如果需要进行更改，请选择 Edit（编辑）。在您感到满意后，选择创建。Status（状态）字段在预置集群时显示 CREATING（正在创建）。

> **Note**
>
> 您可能会收到一个错误，指示请求中的可用区之一没有足够容量来创建 Amazon EKS 集群。如果发生这种情况，错误输出将包含可支持新集群的可用区。再次尝试使用至少两个位于您账户中支持的可用区的子网创建集群。有关更多信息，请参阅 the section called “容量不足”。

集群预配置需要几分钟时间。

12继续the section called “第 3 步：更新 kubeconfig”

创建集群 – Amazon CLI

1. 使用以下命令创建集群。在运行命令之前，进行以下替换：
   - 将 *region-code* 替换为您要在其中创建集群的 Amazon 区域。
   - 将 *my-cluster* 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）、连字符和下划线。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   - 将 *1.33* 替换为任何 Amazon EKS 支持的版本。
   - 将 *111122223333* 替换为账户 ID，并将 *myAmazonEKSClusterRole* 替换为您的集群 IAM 角色的名称。
   - 将 `subnetIds` 的值替换为您自己的值。您还可以添加其他 ID。您必须指定至少两个子网 ID。

     您选择的子网必须符合 Amazon EKS 子网要求。在选择子网之前，我们建议您熟悉所有的 Amazon EKS VPC 以及子网要求和注意事项。
   - 如果您不想指定安全组 ID，请从命令中删除 `,securityGroupIds=sg-<ExampleID1>`。如果您想指定一个或多个安全组 ID，请将 `securityGroupIds` 的值替换为您自己的值。您还可以添加其他 ID。

     无论您是否选择任何安全组，Amazon EKS 都会创建一个安全组，以实现集群和 VPC 之间的通信。Amazon EKS 将此安全组以及您选择的任何安全组与它创建的网络接口关联起来。有关

Amazon EKS 创建的集群安全组的更多信息，请参阅 the section called “安全组要求”。您可以修改 Amazon EKS 创建的集群安全组中的规则。

```
aws eks create-cluster --region region-code --name my-cluster --kubernetes-version
 1.33 \
   --role-arn arn:aws-cn:iam::111122223333:role/myAmazonEKSClusterRole \
   --resources-vpc-config subnetIds=subnet-ExampleID1,subnet-
ExampleID2,securityGroupIds=sg-ExampleID1
```

> **Note**
>
> 您可能会收到一个错误，指示请求中的可用区之一没有足够容量来创建 Amazon EKS 集群。如果发生这种情况，错误输出将包含可支持新集群的可用区。再次尝试使用至少两个位于您账户中支持的可用区的子网创建集群。有关更多信息，请参阅 the section called “容量不足”。

以下是可选设置，如果需要，必须将这些设置添加到上一个命令中。您只能在创建集群时启用这些选项，而不能在创建集群后启用。

- 默认情况下，EKS 会在创建集群期间安装多个联网插件。这包括 Amazon VPC CNI、CoreDNS 和 kube-proxy 等。

  如果要禁用这些默认联网插件的安装，请使用以下参数。这可以用于 Cilium 等替代 CNI。有关更多信息，请参阅 EKS API 参考。

  ```
  aws eks create-cluster --bootstrapSelfManagedAddons false
  ```

- 如果您想指定 Kubernetes 从中分配服务 ID 地址的 IPv4 无类别域间路由（CIDR）块，您必须通过将 `--kubernetes-network-config serviceIpv4Cidr=<cidr-block>` 添加到以下命令中来指定它。

  指定自己的范围有助于防止 Kubernetes 服务与对等或连接到 VPC 的其他网络之间发生冲突。使用 CIDR 表示法输入范围。例如：`10.2.0.0/16`。

  此 CIDR 块必须满足以下要求：

  - 处于以下范围之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。
  - 具有最小大小 `/24` 和最大大小 `/12`。
  - 与您的 Amazon EKS 资源的 VPC 范围不重叠。

您只能在使用 IPv4 地址系列和创建集群时指定此选项。如果没有指定此选项，Kubernetes 会从 10.100.0.0/16 或 172.20.0.0/16 CIDR 块分配服务 IP 地址。

- 如果您创建集群并希望集群分配 IPv6 地址而不是 IPv4 地址给容器组（pod）和服务，请将 --kubernetes-network-config ipFamily=ipv6 添加到以下命令。

  默认情况下，Kubernetes 会将 IPv4 地址分配给容器组（pod）和服务。在决定使用 IPv6 系列前，请确保您熟悉 VPC 要求和注意事项、the section called “子网要求和注意事项”、the section called “安全组要求”和 the section called “IPv6” 主题中的所有注意事项和要求。如果选择 IPv6 系列，则不同于可为其指定地址范围的 IPv4 系列，您无法指定从中分配 IPv6 服务地址的 Kubernetes 的地址范围。Kubernetes 从唯一的本地地址范围 (fc00::/7) 分配服务地址。

2. 预置集群需要几分钟时间。可使用以下命令查询集群的状态。

```
aws eks describe-cluster --region region-code --name my-cluster --query
 "cluster.status"
```

在返回的输出为 ACTIVE 之前，请勿继续执行下一步。

3. 继续the section called “第 3 步：更新 kubeconfig”

第 3 步：更新 kubeconfig

1. 如果您使用 eksctl 创建了集群，则可以跳过此步骤。这是因为 eksctl 已经为您完成了此步骤。通过向 kubectl config 文件添加新上下文来启用 kubectl 与您的集群通信。有关如何创建和更新文件的更多信息，请参阅 the section called “使用 kubectl 访问集群”。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

示例输出如下。

```
Added new context arn:aws-cn:eks:region-code:111122223333:cluster/my-cluster to /
home/username/.kube/config
```

2. 通过运行以下命令以确认与集群的通信。

```
kubectl get svc
```

示例输出如下。

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
```

第 4 步：集群设置

1. （推荐）要使用某些 Amazon EKS 附加组件，或启用个别 Kubernetes 工作负载以具有特定 Amazon Identity and Access Management（IAM）权限，请为集群创建一个 IAM OpenID Connect（OIDC）提供者。您只需为集群创建一次 IAM OIDC 提供程序。要了解有关 Amazon EKS 附加组件的详情，请参阅 the section called “Amazon EKS 附加组件”。要了解有关将特定 IAM 权限分配给工作负载的详情，请参阅 the section called “包含 IRSA 的凭证”。
2. （推荐）在将 Amazon EC2 节点部署到集群之前，为适用于 Kubernetes 的 Amazon VPC CNI 插件配置集群。默认情况下，该插件随集群一起安装。当您将 Amazon EC2 节点添加到集群时，插件将自动部署到您添加的每个 Amazon EC2 节点上。该插件要求您将以下 IAM 策略之一附加到 IAM 角色。如果集群使用 IPv4 系列，请使用 AmazonEKS_CNI_Policy 托管 IAM 策略。如果您的集群使用 IPv6 系列，请使用您创建的 IAM 策略。

   您将策略附加到的 IAM 角色可以是节点 IAM 角色，也可以是仅用于插件的专用角色。我们建议将策略附加到此角色。有关创建角色的更多信息，请参阅the section called “配置 IRSA”或 Amazon EKS 节点 IAM 角色。
3. 如果您使用 Amazon Web Services 管理控制台 部署了集群，则可以跳过此步骤。默认情况下，Amazon Web Services 管理控制台 会部署适用于 Kubernetes 的 Amazon VPC CNI 插件、CoreDNS 和 kube-proxy Amazon EKS 附加组件。

   如果使用 eksctl 或 Amazon CLI 部署集群，则系统将部署适用于 Kubernetes 的 Amazon VPC CNI 插件、CoreDNS 和 kube-proxy 自主管理型附加组件。您可以将使用集群部署的适用于 Kubernetes 的 Amazon VPC CNI 插件、CoreDNS 和 kube-proxy 自主管理型附加组件迁移到 Amazon EKS 附加组件。有关更多信息，请参阅 the section called “Amazon EKS 附加组件”。
4. （可选）如果您尚未执行此操作，可以为集群启用 Prometheus 指标。有关更多信息，请参阅《Amazon Managed Service for Prometheus 用户指南》中的创建抓取程序。
5. 如果您计划将工作负载部署到使用 Amazon EBS 卷的集群，则在部署工作负载之前，您必须将 Amazon EBS CSI 安装到您的集群。

后续步骤

- 创建集群的 IAM 主体是唯一可以访问集群的主体。向其它 IAM 主体授予权限，以便它们可以访问您的集群。
- 如果创建集群的 IAM 主体仅具有先决条件中引用的最低 IAM 权限，则您可能想要为该主体添加额外的 Amazon EKS 权限。有关向 IAM 主体授予 Amazon EKS 权限的更多信息，请参阅the section called “IAM 参考”。
- 如果您希望创建集群的 IAM 主体或任何其他主体在 Amazon EKS 控制台中查看 Kubernetes 资源，请向实体授予所需权限。
- 如果您希望节点和 IAM 主体从 VPC 内访问您的集群，请为集群启用私有端点。默认情况下，将启用公有端点。如需要，可以在启用私有端点后禁用公有端点。有关更多信息，请参阅 the section called “集群终端节点访问”。
- 为集群启用密钥加密。
- 为集群配置日志记录。
- 将节点添加到集群。

Amazon EKS 预置式控制面板

概述

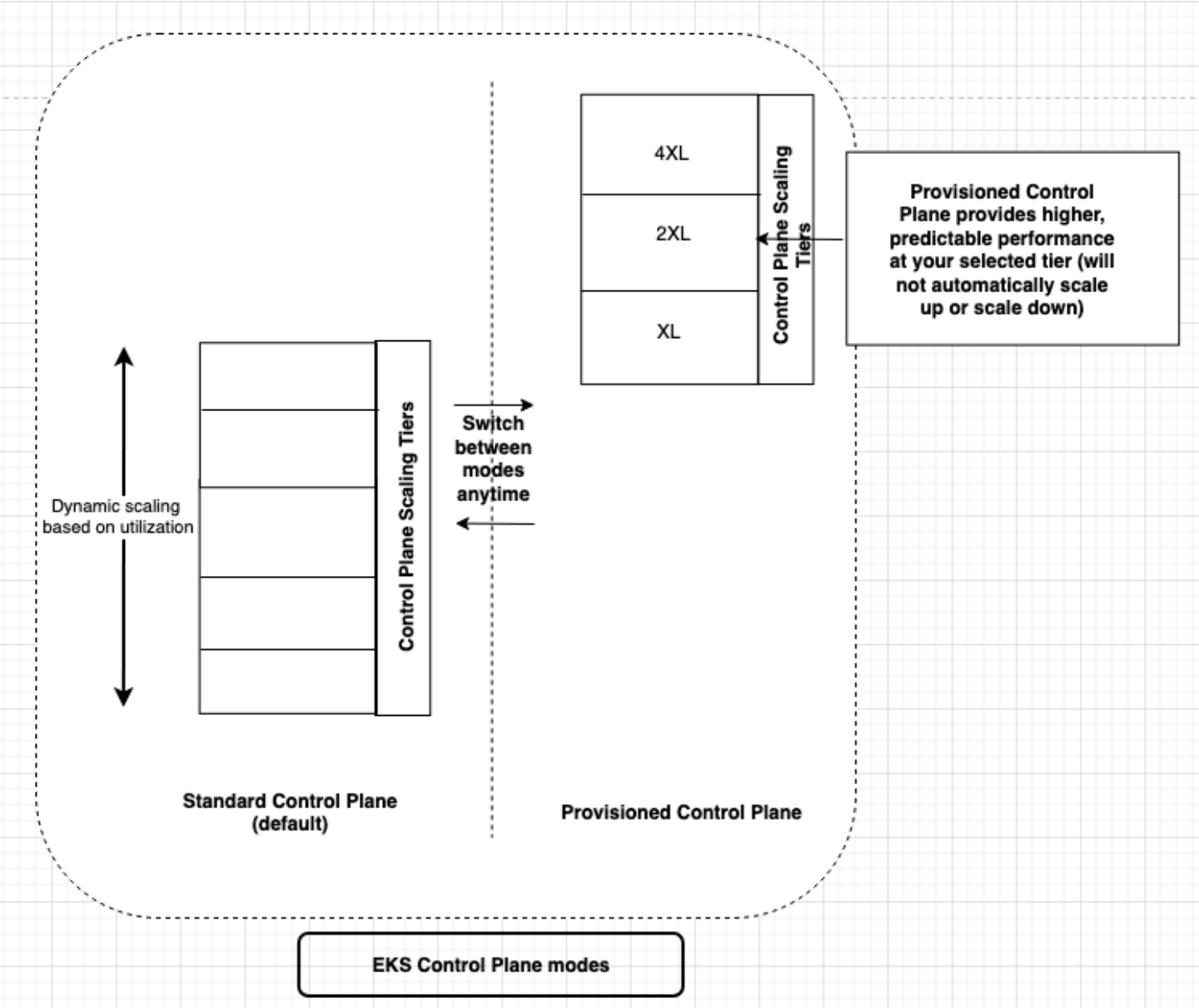
Amazon EKS 预置式控制面板是一项功能，集群管理员可通过它从一系列扩展层级中进行选择，并为集群的控制面板指定所选层级，以此实现极高且可预期的性能表现。借助该功能，集群管理员能够确保控制面板始终按照指定的容量完成预置部署。

Amazon EKS 为集群的控制面板提供两种运行模式。默认情况下，Amazon EKS 集群采用标准模式。在此模式下，控制面板会根据工作负载需求自动进行纵向扩展与缩减。标准模式可动态分配充足的控制面板容量来满足工作负载需求，是适用于大多数使用案例的推荐方案。但对于部分特殊工作负载，比如无法容忍控制面板因扩缩而产生任何性能波动，或需要超大控制面板容量的场景，您可以选择启用预置模式。预置模式支持预先分配控制面板容量，让资源始终处于就绪状态，能够应对高负载的工作负载需求。

> **Note**
>
> 预置模式是一种附加的控制面板运行模式，与默认的标准模式并行存在。预置模式的推出，不会改变标准模式的原有运行逻辑。

借助 EKS 预置式控制面板功能，集群管理员可以提前预置所需的控制面板容量，让集群控制面板始终保持可用状态，同时实现可预期的高性能表现。EKS 预置式控制面板还支持集群管理员在多环境中预置相同规格的控制面板容量，覆盖从测试环境、生产环境到灾难恢复站点的全场景。这一点至关重要，能够确保多环境下的控制面板性能一致且可预期。此外，EKS 预置式控制面板可提供超高规格的控制面板性能，支持您在 Kubernetes 上运行具备海量扩展能力的人工智能工作负载、高性能计算工作负载以及大规模数据处理工作负载。

所有现存及新建的 Amazon EKS 集群，均默认采用标准模式运行。对于对控制面板有高性能、可预期表现需求的集群，您可以选择启用 EKS 预置式控制面板功能。计费方面，除 EKS 标准支持或扩展支持的小时费外，您还需按所选控制面板扩展层级的小时费率支付费用。有关定价的更多信息，请参阅 Amazon EKS 定价。



使用案例

EKS 预置式控制面板功能专为满足部分特定业务场景而设计，在这些场景中，控制面板的高性能与可预期表现对业务运行至关重要。了解这些使用案例，有助于您判断 EKS 预置式控制面板是否为适配自身工作负载的理想方案。

性能敏感型工作负载：对于要求 Kubernetes 控制面板具备极低延迟与极致性能的工作负载，EKS 预置式控制面板可提供充足容量，消除因控制面板扩缩操作引发的性能波动问题。

海量扩展型工作负载：若运行的工作负载具备极强的扩展属性（例如人工智能训练与推理、高性能计算、大规模数据处理），且这类负载需要集群内运行大量节点，EKS 预置式控制面板能够提供所需的控制面板容量，以此支撑这些高压力工作负载的运行。

可预期的高需求事件：当预判将出现控制面板请求量激增的情况时，例如电商大促活动、新品发布、节假日购物季，或是重大体育赛事、文娱活动期间，预置式控制面板允许提前对控制面板容量进行扩展。这种主动配置方式，可确保控制面板提前就绪，从容应对流量高峰，无需等待自动扩缩机制响应需求变化。

环境一致性：预置式控制面板支持您在测试环境与生产环境中配置相同的控制面板容量与性能规格，助力您在业务部署至生产环境前，尽早识别潜在问题。通过在多环境中保持一致的控制面板层级，测试结果能够精准反映生产环境的运行状态，从而降低业务上线时因性能问题引发意外状况的风险。

灾难恢复与业务连续性：在灾难恢复场景下，预置式控制面板允许您为失效转移环境配置与主环境同等水平的控制面板容量。这一配置可确保失效转移切换时，业务中断时间降至最低，且恢复速度更快。因为灾难恢复集群从启动的那一刻起，就具备与生产集群完全一致的控制面板性能特征。

控制面板扩展层级

EKS 预置式控制面板提供的扩展层级采用 T 恤尺码命名方式（XL、2XL、4XL）。每个层级均通过三项 Kubernetes 核心属性定义自身容量规格，而这三项属性决定了集群控制面板的性能特征。理解这些属性，有助于您根据自身工作负载需求选择合适的扩展层级。

API 请求并发量：衡量 Kubernetes 控制面板的 API Server 可同时处理的请求数量，这一指标对于高吞吐量工作负载至关重要。

容器组（pod）调度速率：表示默认 Kubernetes 调度器将容器组（pod）调度到节点的速度，单位为每秒调度容器组（pod）数。

集群数据库容量：表示为 etcd（存储集群状态/元数据的数据库）分配的存储空间大小。

当您通过预置式控制面板功能，为集群控制面板配置某一扩展层级时，Amazon EKS 会确保该控制控制面板维持与所选层级对应的各项限额。控制面板各扩展层级的限额会随 Kubernetes 版本不同而有所差异，具体信息详见下表。

EKS v1.28 及 v1.29

预置式控制面板扩展层级	API 请求并发量（席位）	容器组（pod）调度速率（个/秒）	集群数据库容量（GB）
XL	1700	100	16
2XL	3400	100	16
4XL	6800	100	16

EKS v1.30 及更高版本

预置式控制面板扩展层级	API 请求并发量（席位）	容器组（pod）调度速率（个/秒）	集群数据库容量（GB）
XL	1700	167	16
2XL	3400	283	16
4XL	6800	400	16

监控控制面板扩展层级使用率

Amazon EKS 提供多项指标，有助于您监控控制面板的扩展层级使用率。这些指标会以 Amazon CloudWatch 指标的形式发布，可通过 CloudWatch 控制台与 EKS 控制台进行查看。此外，您可以从 EKS 集群的 Prometheus 端点中采集这些指标（详见此处）。

	Prometheus 指标	CloudWatch 指标
API 请求并发数	apiserver_flowcontrol_current_executing_seats	apiserver_flowcontrol_current_executing_seats

	Prometheus 指标	CloudWatch 指标
容器组（pod）调度速率	scheduler_schedule_attempts_total	scheduler_schedule_attempts_total、scheduler_schedule_attempts_SCHEDULED、scheduler_schedule_attempts_UNSCHEDULABLE
集群数据库大小	apiserver_storage_size_bytes	apiserver_storage_size_bytes

理解层级容量与实际性能的区别

当您选定某一预置式控制面板扩展层级时，该层级的各项属性代表着 Amazon EKS 为集群控制面板配置的底层参数。但您最终获得的实际性能，取决于自身具体的工作负载模式、配置情况以及对 Kubernetes 最佳实践的遵循程度。例如，4XL 层级会将 API 优先级和公平性（APF）配置为支持 6800 个并发请求席位，但控制面板实际能达到的请求吞吐量，会受当前执行的操作类型影响。例如，Kubernetes 对 list 请求的资源消耗要求高于 get 请求，因此控制面板可并发处理的 list 请求有效数量，会低于 get 请求的并发处理数量（详见此处）。同理，虽然 4XL 层级会将默认调度器的每秒查询率（QPS）设置为 400，但实际的容器组（pod）调度速率，还会受节点是否处于就绪、健康的可调度状态等因素影响。若要实现最优性能，请确保应用程序遵循 Kubernetes 最佳实践规范（详见此处），并针对自身工作负载的特征完成合理配置。

注意事项

- 标准模式控制面板容量：EKS 标准模式控制面板具备出色的性价比，是绝大多数使用案例的推荐方案。但对于两类特殊工作负载，无法容忍控制面板因扩缩产生性能波动的负载，或是需要超大控制面板容量的负载，您可以选择启用预置模式。
- 需手动启用：现存集群不会从标准控制面板自动纵向扩展到更高资费的 EKS 预置式控制面板层级。您必须明确启用其中一个全新的 EKS 预置式控制面板扩展层级。
- 回退限制：标准模式控制面板支持的集群数据库（etcd）容量上限为 8 GB。若在预置模式下的集群数据库容量超过 8 GB，则必须先将数据库容量降至 8 GB 以下，才能切换回标准模式。例如，若在预置模式下使用了 14 GB 的数据库存储空间，需先将数据库使用率降至 8 GB 以下，方可回退至标准模式。
- 无自动层级扩缩机制：EKS 预置式控制面板不支持跨层级自动扩缩。一旦选定某一扩展层级，集群控制面板将固定在该层级运行，以此保障性能的一致性与可预期性。不过，您可灵活搭建自定义的自

动扩缩方案：通过监控层级使用率指标，当指标触发设定的阈值时，调用 EKS 预置式控制面板 API 执行纵向扩展或缩减操作，从而完全掌控扩缩策略与成本优化方向。

- 查看当前层级：您可通过 Amazon EKS 控制台、AWS 命令行界面（CLI）或 API，查看当前的控制面板扩展层级。在 CLI 中，可执行 `describe-cluster` 命令：`aws eks describe-cluster --name cluster-name`
- 层级切换耗时：您可通过 Amazon EKS 控制台、API 或 CLI，执行跨层级切换或退出预置模式的操作。Amazon EKS 新增了一种名为 `ScalingTierConfigUpdate` 的集群更新类型，您可通过查看该更新类型的状态，监控层级切换进度。执行层级变更命令后，可列出集群的更新记录，此时会出现一条类型为 `ScalingTierConfigUpdate`、状态为 `Updating` 的新记录。更新完成后，状态会变为 `Successful`；若出现错误，状态则为 `Failed`。更新记录的 error 字段会注明失败原因。层级切换的操作频率不受限制。控制面板切换过程通常需要数分钟完成。
- 选择最优层级：若要为集群选定最优的预置式控制面板扩展层级，可按如下步骤进行压力测试：先将集群预置在最高层级（4XL），再执行压力测试，模拟集群控制面板的峰值负载场景。观察峰值负载下的控制面板层级使用率指标，并以此为依据，选定适合预置模式的扩展层级。
- 预置式控制面板计费规则：集群将按预置式控制面板当前所属扩展层级的小时费率计费。该费用在标准支持或扩展支持的小时费基础上叠加收取。有关详细信息，请参阅 Amazon EKS 定价页面。
- 更大规格扩展层级：若计划将集群运行在高于 4XL 的扩展层级，请联系亚马逊云科技客户经理，咨询额外的定价信息。
- Kubernetes 版本与地域支持：EKS 预置式控制面板在所有亚马逊云科技商业区域、GovCloud 区域及中国区域均提供支持。预置式控制面板适用于 EKS v1.28 及更高版本。

Amazon EKS 预置式控制面板入门

本指南将逐步引导您完成借助 Amazon CLI 及Amazon Web Services 管理控制台，搭建并使用 EKS 预置式控制面板的操作流程。

先决条件

在开始之前，请确保您满足以下条件：

- Amazon CLI – 与 Amazon 服务（包括 Amazon EKS）结合使用的命令行工具。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的安装。在安装 Amazon CLI 后，建议您还要对其进行配置。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的使用 aws configure 快速配置。请注意，需要 Amazon CLI v2 才能使用本页中显示的 update-kubeconfig 选项。
- 所需的 IAM 权限 – 您正在使用的 IAM 安全主体必须具有使用 Amazon EKS IAM 角色、服务相关角色、Amazon CloudFormation、VPC 和相关资源的权限。有关更多信息，请参阅《IAM 用户指

南》中的操作和使用服务相关角色。您必须以同一用户身份完成本指南中的所有步骤。要查看当前用户，请运行以下命令：

```
aws sts get-caller-identity
```

> **Note**
>
> 我们建议您在 Bash Shell 中完成本主题中的步骤。如果您没有使用 Bash Shell，则某些脚本命令（例如行延续字符以及变量的设置和使用方式）需要调整 Shell。此外，您的 Shell 的引用和转义规则可能有所不同。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的在 Amazon CLI 中将引号和字符串结合使用。

EKS 预置式控制面板：Amazon CLI

创建搭载 EKS 预置式控制面板扩展层级的集群

```
aws eks create-cluster --name prod-cluster \
--role-arn arn:aws:iam::012345678910:role/eks-service-role-AWSServiceRoleForAmazonEKS-
J7ONKE3BQ4PI \
--resources-vpc-config subnetIds=subnet-6782e71e,subnet-
e7e761ac,securityGroupIds=sg-6979fe18 \
--control-plane-scaling-config tier=tier-xl
```

响应：

```
{
    "cluster": {
        "name": "my-eks-cluster",
        "arn": "arn:aws:eks:us-east-2:111122223333:cluster/my-eks-cluster",
        "createdAt": "2024-03-14T11:31:44.348000-04:00",
        "version": "1.26",
        "endpoint": "https://JSA79429HJDASKJDJ8223829MNDNASW.yl4.us-
east-2.eks.amazonaws.com",
        "roleArn": "arn:aws:iam::111122223333:role/eksctl-my-eks-cluster-cluster-
ServiceRole-zMF6CBakwwbW",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-0fb75d2d8401716e7",
                "subnet-02184492f67a3d0f9",
```

```
                "subnet-04098063527aab776",
                "subnet-0e2907431c9988b72",
                "subnet-04ad87f71c6e5ab4d",
                "subnet-09d912bb63ef21b9a"
            ],
            "securityGroupIds": [
                "sg-0c1327f6270afbb36"
            ],
            "clusterSecurityGroupId": "sg-01c84d09d70f39a7f",
            "vpcId": "vpc-0012b8e1cc0abb17d",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": true,
            "publicAccessCidrs": [
                "22.19.18.2/32"
            ]
        },
        "controlPlaneScalingConfig": {
            "tier": "tier-xl"
        },
        "kubernetesNetworkConfig": {
            "serviceIpv4Cidr": "10.100.0.0/16",
            "ipFamily": "ipv4"
        },
        "logging": {
            "clusterLogging": [
                {
                    "types": [
                        "api",
                        "audit",
                        "authenticator",
                        "controllerManager",
                        "scheduler"
                    ],
                    "enabled": true
                }
            ]
        },
        "identity": {
            "oidc": {
                "issuer": "https://oidc.eks.us-east-2.amazonaws.com/id/
JSA79429HJDASKJDJ8223829MNDNASW"
            }
        },
        "status": "CREATING",
```

```
        "certificateAuthority": {
            "data": "CA_DATA_STRING..."
        },
        "platformVersion": "eks.14",
        "tags": {
            "aws:cloudformation:stack-name": "eksctl-my-eks-cluster-cluster",
            "alpha.eksctl.io/cluster-name": "my-eks-cluster",
            "karpenter.sh/discovery": "my-eks-cluster",
            "aws:cloudformation:stack-id": "arn:aws:cloudformation:us-
east-2:111122223333:stack/eksctl-my-eks-cluster-cluster/e752ea00-e217-11ee-
beae-0a9599c8c7ed",
            "auto-delete": "no",
            "eksctl.cluster.k8s.io/v1alpha1/cluster-name": "my-eks-cluster",
            "EKS-Cluster-Name": "my-eks-cluster",
            "alpha.eksctl.io/cluster-oidc-enabled": "true",
            "aws:cloudformation:logical-id": "ControlPlane",
            "alpha.eksctl.io/eksctl-version": "0.173.0-dev
+a7ee89342.2024-03-01T03:40:57Z",
            "Name": "eksctl-my-eks-cluster-cluster/ControlPlane"
        },
        "health": {
            "issues": []
        },
        "accessConfig": {
            "authenticationMode": "API_AND_CONFIG_MAP"
        }
    }
}
```

查看集群的控制面板扩展层级

```
aws eks describe-cluster --name prod-cluster
```

响应：

```
{
    "cluster": {
        "name": "my-eks-cluster",
        "arn": "arn:aws:eks:us-east-2:111122223333:cluster/my-eks-cluster",
        "createdAt": "2024-03-14T11:31:44.348000-04:00",
        "version": "1.26",
        "endpoint": "https://JSA79429HJDASKJDJ8223829MNDNASW.yl4.us-
east-2.eks.amazonaws.com",
```

```
        "roleArn": "arn:aws:iam::111122223333:role/eksctl-my-eks-cluster-cluster-
ServiceRole-zMF6CBakwwbW",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-0fb75d2d8401716e7",
                "subnet-02184492f67a3d0f9",
                "subnet-04098063527aab776",
                "subnet-0e2907431c9988b72",
                "subnet-04ad87f71c6e5ab4d",
                "subnet-09d912bb63ef21b9a"
            ],
            "securityGroupIds": [
                "sg-0c1327f6270afbb36"
            ],
            "clusterSecurityGroupId": "sg-01c84d09d70f39a7f",
            "vpcId": "vpc-0012b8e1cc0abb17d",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": true,
            "publicAccessCidrs": [
                "22.19.18.2/32"
            ]
        },
        "controlPlaneScalingConfig": {
            "tier": "tier-xl"
        },
        "kubernetesNetworkConfig": {
            "serviceIpv4Cidr": "10.100.0.0/16",
            "ipFamily": "ipv4"
        },
        "logging": {
            "clusterLogging": [
                {
                    "types": [
                        "api",
                        "audit",
                        "authenticator",
                        "controllerManager",
                        "scheduler"
                    ],
                    "enabled": true
                }
            ]
        },
        "identity": {
```

```
            "oidc": {
                "issuer": "https://oidc.eks.us-east-2.amazonaws.com/id/
JSA79429HJDASKJDJ8223829MNDNASW"
            }
        },
        "status": "ACTIVE",
        "certificateAuthority": {
            "data": "CA_DATA_STRING..."
        },
        "platformVersion": "eks.14",
        "tags": {
            "aws:cloudformation:stack-name": "eksctl-my-eks-cluster-cluster",
            "alpha.eksctl.io/cluster-name": "my-eks-cluster",
            "karpenter.sh/discovery": "my-eks-cluster",
            "aws:cloudformation:stack-id": "arn:aws:cloudformation:us-
east-2:111122223333:stack/eksctl-my-eks-cluster-cluster/e752ea00-e217-11ee-
beae-0a9599c8c7ed",
            "auto-delete": "no",
            "eksctl.cluster.k8s.io/v1alpha1/cluster-name": "my-eks-cluster",
            "EKS-Cluster-Name": "my-eks-cluster",
            "alpha.eksctl.io/cluster-oidc-enabled": "true",
            "aws:cloudformation:logical-id": "ControlPlane",
            "alpha.eksctl.io/eksctl-version": "0.173.0-dev
+a7ee89342.2024-03-01T03:40:57Z",
            "Name": "eksctl-my-eks-cluster-cluster/ControlPlane"
        },
        "health": {
            "issues": []
        },
        "accessConfig": {
            "authenticationMode": "API_AND_CONFIG_MAP"
        }
    }
}
```

更新集群以启用 EKS 预置式控制面板

```
aws eks update-cluster-config --name prod-cluster \
--control-plane-scaling-config tier=tier-2xl
```

响应：

```
{
```

```
    "update": {
        "id": "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd",
        "status": "InProgress",
        "type": "ScalingTierConfigUpdate",
        "params": [
            {
                "type": "UpdatedTier",
                "value": "tier-2xl"
            },
            {
                "type": "PreviousTier",
                "value": "tier-xl"
            }
        ],
        "createdAt": 1565807210.37,
        "errors": []
    }
}
```

查看控制面板扩展配置的更新状态

```
aws eks list-updates --name example
```

响应：

```
{
    "updateIds": [
        "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd1"
    ]
}
```

从预置式控制面板切换到标准控制面板

```
aws eks update-cluster-config --name prod-cluster \
--control-plane-scaling-config tier=standard
```

响应：

```
{
    "update": {
        "id": "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd",
```

```
        "status": "InProgress",
        "type": "ScalingTierConfigUpdate",
        "params": [
            {
                "type": "UpdatedTier",
                "value": "standard"
            },
            {
                "type": "PreviousTier",
                "value": "tier-2xl"
            }
        ],
        "createdAt": 1565807210.37,
        "errors": []
    }
}
```

EKS 预置式控制面板：Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择创建集群。
3. 在配置选项下，选择自定义配置。
4. 向下滚动到控制面板扩展层级选项。勾选启用扩展层级，以此开启预置式控制面板功能。
5. 从各类扩展层级选项（如 XL、2XL、4XL）中，选择要为集群配置的控制面板扩展层级。
6. 根据需求选择其他集群配置项。在最后一步中，点击创建集群。请注意，集群创建过程可能需要数分钟才能完成。

利用集群见解为 Kubernetes 版本升级做好准备并对错误配置进行问题排查

Amazon EKS 集群见解提供问题检测和解决问题的建议，以帮助您管理集群。每个 Amazon EKS 集群都会根据 Amazon EKS 精心策划的见解列表进行自动的定期检查。这些见解检查完全由 Amazon EKS 管理，并就如何解决任何调查发现提供建议。

集群见解类型

- 配置见解：识别 EKS 混合节点设置中可能损害集群或工作负载功能的错误配置。

- 升级见解：识别可能影响您升级到新版 Kubernetes 能力的问题。

注意事项

- 频率：Amazon EKS 每 24 小时刷新一次集群见解，您也可以手动刷新以查看最新状态。例如，您可以在解决问题后手动刷新集群见解，以查看问题是否已解决。
- 权限：Amazon EKS 会为每个 EKS 集群中的集群见解自动创建集群访问条目。此条目会授予 EKS 查看有关集群的信息的权限。Amazon EKS 会利用这些信息来生成见解。有关更多信息，请参阅 the section called “AmazonEKSClusterInsightsPolicy”。

使用案例

Amazon EKS 中的集群见解提供自动检查，可帮助维护 Kubernetes 集群的运行状况、可靠性和最佳配置。以下是集群见解的关键使用案例，包括升级准备情况和配置故障排除。

升级见解

升级见解是集群见解中的一种特定类型的见解检查。这些检查会返回与 Kubernetes 版本升级就绪情况相关的见解。Amazon EKS 会对每个 EKS 集群运行升级见解检查。

> **Important**
>
> Amazon EKS 暂时回滚了一项功能，该功能要求您在遇到某些集群见解问题时使用 `--force` 标志升级集群。有关更多信息，请参阅 GitHub 上的 Temporary rollback of enforcing upgrade insights on update cluster version。
> 有关更新集群的更多信息，请参阅the section called “第 3 步：更新集群控制面板”。

在更新集群 Kubernetes 版本之前，可以使用 Amazon EKS 控制台中可观测性控制面板的升级见解选项卡。如果您的集群已发现问题，请查看它们并进行适当的修复。这些问题包括指向 Amazon EKS 和 Kubernetes 文档的链接。修复问题后，按需刷新集群见解以获取最新见解。如果所有问题都已解决，则请更新您的集群。

Amazon EKS 会返回与 Kubernetes 版本升级就绪情况相关的见解。升级见解可以发现可能影响 Kubernetes 集群升级的可能问题。这样可以最大限度地减少管理员准备升级所需的工作量，并提高新 Kubernetes 版本上应用程序的可靠性。Amazon EKS 会根据可能影响 Kubernetes 版本升级的问题列表自动扫描集群。Amazon EKS 经常根据对每个 Kubernetes 版本中所做更改的审查来更新见解检查列表。

Amazon EKS 升级见解加快了新版本的测试和验证过程。还允许集群管理员和应用程序开发人员通过强调问题和提供补救建议来利用最新 Kubernetes 功能。

配置见解

EKS 集群见解会自动扫描带有混合节点的 Amazon EKS 集群，以发现影响 Kubernetes 控制面板到 Webhook 通信、exec 和日志等 kubectl 命令执行及其他方面的配置问题。配置见解可以发现问题并提供补救建议，从而加快实现混合节点设置完全正常运行的时间。

开始使用

要查看已执行的见解检查列表以及 Amazon EKS 发现的任何相关问题，您可以使用 Amazon Web Services 管理控制台、Amazon CLI、Amazon SDK 和 Amazon EKS `ListInsights` API 操作。要开始使用，请参阅 the section called “查看集群见解”。

查看集群见解

Amazon EKS 提供两种类型的见解：配置见解与升级见解。配置见解可识别 EKS 混合节点设置中可能损害集群或工作负载功能的错误配置。升级见解可识别可能影响您升级到新版 Kubernetes 能力的问题。

要查看已执行的见解检查列表以及 Amazon EKS 发现的任何相关问题，您可以调用 Amazon Web Services 管理控制台、Amazon CLI、Amazon SDK 和 Amazon EKS `ListInsights` API 操作进行查看。

查看配置见解（控制台）

1. 打开 Amazon EKS console 控制台。
2. 从集群列表中，选择您希望查看见解的 Amazon EKS 集群的名称。
3. 选择监控集群。
4. 选择集群运行状况选项卡。
5. 在配置见解表中，您将看到以下列：
   - 名称 – Amazon EKS 对集群执行的检查。
   - 见解状态：状态为 `Error` 的见解表示存在可能影响集群功能的错误配置。状态为 `Warning` 的见解表示配置与记录的方法不匹配，但如果有意配置该集群功能，则可能会起作用。状态为 `Passing` 的见解表示 Amazon EKS 并未在集群中发现与该见解检查相关的任何问题。
   - 版本：适用的版本。
   - 上次刷新时间：此集群上次刷新见解状态的时间。

- 描述 – 来自见解检查的信息，包括警报和建议的补救措施。

查看升级见解（控制台）

1. 打开 Amazon EKS console 控制台。
2. 从集群列表中，选择您希望查看见解的 Amazon EKS 集群的名称。
3. 选择监控集群。
4. 选择升级见解选项卡。
5. 要查看最新数据，请选择刷新见解按钮，然后等待刷新操作完成。
6. 在升级见解表中，您将看到以下列：
   - 名称 – Amazon EKS 对集群执行的检查。
   - 见解状态 – 状态为“错误”的见解通常表示受影响的 Kubernetes 版本是当前集群版本的 N+1，而状态为“警告”的见解表示该见解适用于未来 Kubernetes 版本 N+2 或更高版本。状态为“通过”的见解表示 Amazon EKS 在您的集群中未发现与该见解检查相关的任何问题。状态为“未知”的见解表示 Amazon EKS 无法确定您的集群是否受到此见解检查的影响。
   - 版本 – 见解检查可能存在问题的 Kubernetes 版本。
   - 上次刷新时间：此集群上次刷新见解状态的时间。
   - 上次转换时间：此见解上次更改状态的时间。
   - 描述 – 来自见解检查的信息，包括警报和建议的补救措施。

查看集群见解（Amazon CLI）

1. 要查看最新数据，请刷新指定集群的见解。根据需要对该命令进行以下修改，然后运行修改后的命令。
   - 将 *region-code* 替换为 Amazon 区域的代码。
   - 将 *my-cluster* 替换为您的集群的名称。

```
aws eks start-insight-refresh --region region-code --cluster-name my-cluster
```

2. 要跟踪见解刷新状态，请运行以下命令。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-insights-refresh --cluster-name my-cluster
```

示例输出如下。

```
{
    "message": "Insights refresh is in progress",
    "status": "IN_PROGRESS",
    "startedAt": "2025-07-30T13:36:09-07:00"
}
```

3. 列出指定集群的见解。根据需要对该命令进行以下修改，然后运行修改后的命令。
   - 将 *region-code* 替换为 Amazon 区域的代码。
   - 将 *my-cluster* 替换为您的集群的名称。

```
aws eks list-insights --region region-code --cluster-name my-cluster
```

示例输出如下。

```
{
"insights":
    [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
 for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
 updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
 days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "name": "Kubelet version skew",
            "category": "UPGRADE_READINESS",
```

```
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557309.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for kubelet versions of worker nodes in the
cluster to see if upgrade would cause non compliance with supported Kubernetes
kubelet version skew policy.",
            "insightStatus":
            {
                "status": "UNKNOWN",
                "reason": "Unable to determine status of node kubelet versions.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
            "name": "Cluster health issues",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for any cluster health issues that prevent
successful upgrade to the next Kubernetes version on EKS.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No cluster health issues detected.",
            },
        },
```

```
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
            "name": "EKS add-on version compatibility",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of installed EKS add-ons to ensure they
are compatible with the next version of Kubernetes. ",
            "insightStatus": { "status": "PASSING", "reason": "All installed EKS
add-on versions are compatible with next Kubernetes version."},
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEccccc",
            "name": "kube-proxy version skew",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of kube-proxy in cluster to see if
upgrade would cause non compliance with supported Kubernetes kube-proxy version
skew policy.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "kube-proxy versions match the cluster control plane
version.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEddddd",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
days.",
```

```
            },
        },
    ],
"nextToken": null,
}
```

4. 有关见解的描述性信息，请运行以下命令。根据需要对该命令进行以下修改，然后运行修改后的命令。
   - 将 *region-code* 替换为 Amazon 区域的代码。
   - 请将 *a1b2c3d4-5678-90ab-cdef-EXAMPLE22222* 替换为从集群见解列表中检索到的见解 ID。
   - 将 *my-cluster* 替换为您的集群的名称。

   ```
   aws eks describe-insight --region region-code --id a1b2c3d4-5678-90ab-cdef-
   EXAMPLE22222 --cluster-name my-cluster
   ```

   示例输出如下。

   ```
   {
     "insight":
       {
         "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
         "name": "Kubelet version skew",
         "category": "UPGRADE_READINESS",
         "kubernetesVersion": "1.27",
         "lastRefreshTime": 1734557309.000,
         "lastTransitionTime": 1734557309.000,
         "description": "Checks for kubelet versions of worker nodes in the cluster
    to see if upgrade would cause non compliance with supported Kubernetes kubelet
    version skew policy.",
         "insightStatus":
           {
             "status": "UNKNOWN",
             "reason": "Unable to determine status of node kubelet versions.",
           },
         "recommendation": "Upgrade your worker nodes to match the Kubernetes version
    of your cluster control plane.",
         "additionalInfo":
           {
             "Kubelet version skew policy": "https://kubernetes.io/releases/version-
   skew-policy/#kubelet",
   ```

```
          "Updating a managed node group": "https://docs.aws.amazon.com/eks/latest/
userguide/update-managed-node-group.html",
        },
      "resources": [],
      "categorySpecificSummary":
        { "deprecationDetails": [], "addonCompatibilityDetails": [] },
    },
}
```

将现有集群更新到新的 Kubernetes 版本

当 Amazon EKS 中有新的 Kubernetes 版本可用时，您可以将 Amazon EKS 集群更新到最新版本。

> **Important**
>
> 升级集群后，就无法降级到以前的版本。在更新到新的 Kubernetes 版本之前，建议您先查看了解 EKS 上的 Kubernetes 版本生命周期中的信息，以及本主题中的更新步骤。

新的 Kubernetes 版有时会引入重大更改。因此，我们建议您先针对新的 Kubernetes 版本测试您的应用程序的行为，然后再在生产集群上执行更新。您可以通过构建持续集成工作流来测试应用程序行为，然后再移至新的 Kubernetes 版本来执行此操作。

更新过程包含 Amazon EKS 随 Kubernetes 的更新版本推出新的 API 服务器节点，以取代现有此类节点。Amazon EKS 对这些新节点上的网络流量执行标准基础设施和就绪运行状况检查，以确认它们是否按预期工作。但是，一旦开始集群升级，就不能暂停或停止。如果任意一项检查失败，Amazon EKS 都将恢复基础设施部署，且您的集群保留为先前的 Kubernetes 版本。正在运行的应用程序不会受影响，并且您的集群绝不会处于不确定性或不可恢复的状态。Amazon EKS 会定期备份所有托管的集群，并且具有在必要时恢复集群的机制。我们会不断评估和改进我们的 Kubernetes 基础设施管理流程。

为了升级集群，Amazon EKS 需要您在创建集群时指定的子网中的最多 5 个可用的 IP 地址。Amazon EKS 在您指定的任何子网中创建新的集群弹性网络接口（网络接口）。网络接口可能在不同于现有网络接口所在的子网中创建，因此请确保您的安全组规则允许您对创建集群时指定的任何子网进行所需的集群通信。如果您在创建集群时指定的任何子网不存在，没有足够的可用 IP 地址，或者没有允许必要的集群通信的安全组规则，则更新可能会失败。

为确保集群的 API 服务器端点始终可访问，Amazon EKS 提供了高度可用的 Kubernetes 控制面板，并且会在更新操作期间执行 API 服务器实例的滚动更新。由于支持 Kubernetes API 服务器端点的 API

服务器实例 IP 地址会持续变化，您必须确保自己的 API 服务器客户端能够有效地管理重新连接。最新版本的 kubectl 和 Kubernetes 客户端库已经获得正式支持，能够透明地执行此重新连接过程。

> **Note**
>
> 要了解有关集群更新内容的更多信息，请参阅《EKS Best Practices Guide》中的 Best Practices for Cluster Upgrades。此资源有助于您规划升级，了解集群升级策略。

Amazon EKS 自动模式的注意事项

- Amazon EKS 自动模式的计算能力控制节点的 Kubernetes 版本。升级控制面板后，EKS 自动模式将开始逐步更新托管式节点。EKS 自动模式会遵守容器组中断预算。
- 您无需手动升级 Amazon EKS 自动模式的功能，包括计算自动扩缩、块存储和负载均衡功能。

摘要

下面简要概述了 Amazon EKS 集群升级过程：

1. 确保集群处于支持升级的状态。这包括检查部署到集群中的资源所使用的 Kubernetes API，确保集群没有任何运行状况问题。在评估集群的升级准备情况时，您应使用 Amazon EKS 升级见解。
2. 将控制面板升级到下一个次要版本（例如，从 1.32 升级到 1.33）。
3. 升级数据面板中的节点以匹配控制面板中的节点。
4. 升级集群上运行的所有其他应用程序（例如 cluster-autoscaler）。
5. 升级 Amazon EKS 提供的附加组件，例如默认包含的附加组件：
   - Amazon VPC CNI 推荐版本
   - CoreDNS 推荐版本
   - kube-proxy 推荐版本
6. 升级与集群通信的所有客户端（例如 kubectl）。

第 1 步：准备升级

比较集群控制层面的 Kubernetes 版本与节点的 Kubernetes 版本。

- 获取集群控制面板的 Kubernetes 版本。

```
kubectl version
```

- 获取您的节点的 Kubernetes 版本。此命令会返回所有自主管理型和托管式 Amazon EC2、Fargate 和混合节点。每个 Fargate 容器组（pod）都作为其自身的节点列出。

```
kubectl get nodes
```

将控制面板升级到新的 Kubernetes 版本之前，确保集群中的托管节点和 Fargate 节点的 Kubernetes 次要版本与控制面板的版本相同。例如，如果您的控制面板正在运行 1.29 版本，且其中一个节点正在运行 1.28 版本，则您必须将节点更新为 1.29 版本，然后将控制面板更新为 1.30。我们还建议在更新控制面板之前，将自主管理型节点和混合节点更新到与控制面板相同的版本。有关更多信息，请参阅 the section called “更新”、the section called “更新方法” 和 the section called “升级混合节点”。如果 Fargate 节点的次要版本低于控制面板版本，请先删除节点所表示的容器组（pod）。然后更新您的控制面板。剩余的容器组（pod）将在重新部署之后更新到新版本。

第 2 步：检查升级注意事项

Amazon EKS 集群见解会根据可能影响 Kubernetes 版本升级的问题（例如已弃用的 API 使用问题）列表自动扫描集群。Amazon EKS 会根据对 Kubernetes 项目变更的评估，定期更新待执行的见解检查列表。每当 Amazon EKS 服务引入更改以及新版本后，Amazon EKS 也会更新见解检查列表。有关更多信息，请参阅 the section called “集群见解”。

查看 Kubernetes 文档中的已弃用 API 的迁移指南。

查看升级见解

使用 Amazon EKS 升级见解来识别问题。有关更多信息，请参阅 the section called “查看升级见解（控制台）”。

详细注意事项

- 由于 Amazon EKS 运行高度可用的控制面板，您一次只能更新一个次要版本。有关此要求的更多信息，请参阅 Kubernetes 版本和版本偏差支持策略。假设集群的当前版本为版本 1.28，而且您想将其更新到版本 1.30。您必须先将版本 1.28 集群更新为版本 1.29，然后将版本 1.29 集群更新为版本 1.30。
- 查看节点上 Kubernetes `kube-apiserver` 和 `kubelet` 之间的版本偏差。

  - 从 Kubernetes 版本 1.28 开始，kubelet 最多可能有三个早于 kube-apiserver 的次要版本。请参阅 Kubernetes 上游版本偏差策略。
  - 如果您的托管节点和 Fargate 节点上的 kubelet 为 Kubernetes 版本 1.25 或更新版本，则无需更新 kubelet 版本即可将集群最多提前更新三个版本。例如，如果 kubelet 为版本 1.25，则可以在 kubelet 保持版本 1.25 的情况下，将您的 Amazon EKS 集群版本从 1.25 更新到 1.27、1.26 和 1.28。
- 作为开始更新之前的最佳实践，请确保节点上的 kubelet 与控制面板具有相同的 Kubernetes 版本。
- 如果集群配置有早于 1.8.0 的适用于 Kubernetes 的 Amazon VPC CNI 插件版本，建议将插件更新为最新版本，然后更新集群。要更新插件，请参阅 the section called “Amazon VPC CNI”。
- 您可以备份 Amazon EKS 集群，以便在升级过程中出现故障时恢复 Amazon EKS 集群状态和持久性存储。请参阅 the section called “ Amazon Backup”。

第 3 步：更新集群控制面板

> **Important**
>
> Amazon EKS 暂时回滚了一项功能，该功能要求您在遇到某些集群见解问题时使用 --force 标志升级集群。有关更多信息，请参阅 GitHub 上的 Temporary rollback of enforcing upgrade insights on update cluster version。
> Amazon EKS 会在“上次刷新时间”后 24 小时刷新集群见解。您可以将问题解决时间与集群见解的“上次刷新时间”进行比较。
> 此外，解决已弃用的 API 使用问题后，见解状态最多需要 30 天才能更新。升级见解始终会在 30 天滚动窗口内查找已弃用的 API 使用问题。

您可以使用以下方式提交升级 EKS 控制面板版本的请求：

- eksctl
- Amazon 控制台
- Amazon CLI

更新集群 – eksctl

此过程需要 eksctl 版本 0.215.0 或更高版本。可以使用以下命令来查看您的版本：

```
eksctl version
```

有关安装和更新 eksctl 的说明，请参阅 eksctl 文档中的 Installation。

更新 Amazon EKS 控制面板的 Kubernetes 版本。将 <cluster-name> 替换为您的集群名称。将 <version-number> 替换为想要将集群更新到的 Amazon EKS 支持的版本号。有关支持的版本号列表，请参阅 Amazon EKS 支持的版本。

```
eksctl upgrade cluster --name <cluster-name> --version <version-number> --approve
```

更新过程可能需要几分钟才能完成。

继续the section called “第 4 步：更新集群组件”。

更新集群 – Amazon 控制台

1. 打开 Amazon EKS 控制台。
2. 对于想要升级的集群，选择立即升级。
3. 选择要将集群更新到的版本，然后选择升级。
4. 更新过程可能需要几分钟才能完成。继续the section called “第 4 步：更新集群组件”。

更新集群– Amazon CLI

1. 确认已安装 Amazon CLI 并且您已登录。有关更多信息，请参阅安装或更新到最新版本的 Amazon CLI。
2. 使用以下 Amazon CLI 命令更新 Amazon EKS 集群。替换要升级的集群的 <cluster-name> 和 <region-code>。将 <version-number> 替换为想要将集群更新到的 Amazon EKS 支持的版本号。有关支持的版本号列表，请参阅 Amazon EKS 支持的版本。

   ```
   aws eks update-cluster-version --name <cluster-name> \
     --kubernetes-version <verion-number> --region <region-code>
   ```

   示例输出如下。

   ```
   {
       "update": {
           "id": "<update-id>",
           "status": "InProgress",
   ```

```
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "<version-number>"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
[...]
        "errors": []
    }
```

3. 更新过程可能需要几分钟才能完成。使用以下命令监控集群更新的状态。除了使用相同的 <cluster-name> 和 <region-code> 之外，还要使用上一条命令返回的 <update-id>。

```
aws eks describe-update --name <cluster-name> \
   --region <region-code> --update-id <update-id>
```

   当 Successful 状态显示时，更新完成。

4. 继续the section called “第 4 步：更新集群组件”。

第 4 步：更新集群组件

1. 集群更新完成后，将节点更新为与已更新的集群相同的 Kubernetes 版本。有关更多信息，请参阅 the section called “更新方法”、the section called “更新” 和 the section called “升级混合节点”。在 Fargate 上启动的任何新容器组（pod）都具有与您的集群版本匹配的 kubelet 版本。现有 Fargate Pod 不会更改。
2. （可选）如果您在更新集群之前将 Kubernetes Cluster Autoscaler 部署到了集群，请将 Cluster Autoscaler 更新为与您升级后的 Kubernetes 主版本和次要版本匹配的最新版本。
   a. 在 Web 浏览器中打开 Cluster Autoscaler 版本页面，查找与您集群的 Kubernetes 主版本和次要版本相匹配的最新 Cluster Autoscaler 版本。例如，如果您集群的 Kubernetes 版本是 1.30，则查找以 1.30 开头的最新 Cluster Autoscaler 版本。记录该版本的语义版本号（例如 1.30.n）以在下一步中使用。
   b. 使用以下命令，将 Cluster Autoscaler 映像标签设置为您在上一步中记录的版本。如有必要，将 X.XX.X 替换为您自己的值。

```
kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-
autoscaler=registry.k8s.io/autoscaling/cluster-autoscaler:vX.XX.X
```

3. （仅限具有 GPU 节点的集群）如果您的集群具有支持 GPU 的节点组（例如 p3.2xlarge），则您必须在集群上更新适用于 Kubernetes 的 NVIDIA 设备插件 DaemonSet。将 <vX.X.X> 替换为您需要的 NVIDIA/k8s-device-plugin 版本，然后运行以下命令。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/<vX.X.X>/
deployments/static/nvidia-device-plugin.yml
```

4. 更新适用于 Kubernetes 的 Amazon VPC CNI 插件、CoreDNS 和 kube-proxy 附加组件。建议将附件组件更新为服务账户令牌中列出的最低版本。
   - 如果您正在使用 Amazon EKS 附加组件，请在 Amazon EKS 控制台中选择 Clusters（集群），然后在左侧导航窗格中选择您更新的集群的名称。控制台中显示通知。这些通知将告知您有一个新版本可用于每个具有可用更新的附加组件。要更新附加组件，请选择附加组件选项卡。在具有可用更新的附加组件的其中一个框中，选择立即更新，选择一个可用版本，然后选择更新。
   - 此外，您也可以使用 Amazon CLI 或 eksctl 更新附加组件。有关更多信息，请参阅 the section called “更新附加组件”。
5. 如有必要，请更新您的 kubectl 版本。您必须使用与 Amazon EKS 集群控制面板不同的次要版本内的 kubectl 版本。

降级 Amazon EKS 集群的 Kubernetes 版本

无法降级 Amazon EKS 集群的 Kubernetes。而应根据之前的 Amazon EKS 版本创建一个新集群并迁移工作负载。

删除集群

使用完 Amazon EKS 集群后，应删除与其关联的资源，这样便不会产生任何不必要的费用。

您可以使用 eksctl、Amazon Web Services 管理控制台或 Amazon CLI 删除集群。

注意事项

- 如果您因为已删除集群创建者而收到错误，请参阅这篇文章解决。

- 适用于 Prometheus 的 Amazon 托管服务资源不在集群生命周期内，需要独立于集群进行维护。删除集群时，请务必同时删除所有适用的抓取器以停止适用的费用。有关更多信息，请参阅《Amazon Managed Service for Prometheus 用户指南》中的查找和删除抓取程序。
- 要删除连接的集群，请参阅 the section called “注销集群”

EKS 自动模式注意事项

- 所有 EKS 自动模式节点都将被删除，包括 EC2 托管式实例
- 所有负载均衡器都将被删除

有关更多信息，请参阅 the section called “禁用 EKS 自动模式”。

删除集群（eksctl）

此过程需要 `eksctl` 版本 `0.215.0` 或更高版本。可以使用以下命令来查看您的版本：

```
eksctl version
```

有关安装或升级 `eksctl` 的说明，请参阅 `eksctl` 文档中的 Installation。

1. 列出集群中运行的所有服务。

   ```
   kubectl get svc --all-namespaces
   ```

   a. 删除具有关联的 `EXTERNAL-IP` 值的任何服务。这些服务的前面配置了一个 Elastic Load Balancing 负载均衡器，您必须从 Kubernetes 中将其删除才能释放负载均衡器和关联资源。请根据描述将 *`service-name`* 替换为所列每项服务的名称。

      ```
      kubectl delete svc service-name
      ```

2. 使用以下命令（将 *`prod`* 替换为您的集群名称）删除集群及其关联的节点。

   ```
   eksctl delete cluster --name prod
   ```

   输出：

   ```
   [#]  using region region-code
   ```

```
[#]  deleting EKS cluster "prod"
[#]  will delete stack "eksctl-prod-nodegroup-standard-nodes"
[#]  waiting for stack "eksctl-prod-nodegroup-standard-nodes" to get deleted
[#]  will delete stack "eksctl-prod-cluster"
[#]  the following EKS cluster resource(s) for "prod" will be deleted: cluster. If in
 doubt, check CloudFormation console
```

删除集群（Amazon 控制台）

1. 列出集群中运行的所有服务。

```
kubectl get svc --all-namespaces
```

2. 删除具有关联的 EXTERNAL-IP 值的任何服务。这些服务的前面配置了一个 Elastic Load Balancing 负载均衡器，您必须从 Kubernetes 中将其删除才能释放负载均衡器和关联资源。请根据描述将 *service-name* 替换为所列每项服务的名称。

```
kubectl delete svc service-name
```

3. 删除所有节点组和 Fargate 配置文件。
   a. 打开 Amazon EKS 控制台。
   b. 请在左侧导航窗格中，选择 Amazon EKS Clusters（集群），然后在集群的选项卡列表中，选择要删除的集群的名称。
   c. 选择 Compute（计算）选项卡，然后选择要删除的节点组。选择 Delete（删除），输入节点组的名称，然后选择 Delete（删除）。删除集群中的所有节点组。

   > **Note**
   >
   > 只会列出托管节点组。

   d. 选择要删除的 Fargate Profile（Fargate 配置文件），选择 Delete（删除），输入配置文件的名称，然后选择 Delete（删除）。删除集群中的所有 Fargate 配置文件。
4. 删除所有自行管理的节点 Amazon CloudFormation 堆栈。
   a. 打开 Amazon CloudFormation 控制台。
   b. 请选择要删除的节点堆栈，然后选择 Delete（删除）。
   c. 在 Delete stack（删除堆栈）确认对话框中，请选择 Delete stack（删除堆栈）。删除集群中的所有自行管理的节点堆栈。

5. 请删除集群。
   a. 打开 Amazon EKS 控制台。
   b. 选择要删除的集群并选择 Delete（删除）。
   c. 在删除集群确认屏幕上，选择 Delete (删除)。
6. （可选）删除 VPC Amazon CloudFormation 堆栈。
   a. 打开 Amazon CloudFormation 控制台。
   b. 请选择要删除的 VPC 堆栈，然后选择 Delete（删除）。
   c. 在 Delete stack（删除堆栈）确认对话框中，请选择 Delete stack（删除堆栈）。

删除集群（Amazon CLI）

1. 列出集群中运行的所有服务。

```
kubectl get svc --all-namespaces
```

2. 删除具有关联的 EXTERNAL-IP 值的任何服务。这些服务的前面配置了一个 Elastic Load Balancing 负载均衡器，您必须从 Kubernetes 中将其删除才能释放负载均衡器和关联资源。请根据描述将 *service-name* 替换为所列每项服务的名称。

```
kubectl delete svc service-name
```

3. 删除所有节点组和 Fargate 配置文件。
   a. 使用以下命令列出集群中的节点组。

```
aws eks list-nodegroups --cluster-name my-cluster
```

> **Note**
> 只会列出托管节点组。

   b. 使用以下命令删除每个节点组。删除集群中的所有节点组。

```
aws eks delete-nodegroup --nodegroup-name my-nodegroup --cluster-name my-cluster
```

   c. 使用以下命令列出集群中的 Fargate 配置文件。

```
aws eks list-fargate-profiles --cluster-name my-cluster
```

d. 使用以下命令删除每个 Fargate 配置文件。删除集群中的所有 Fargate 配置文件。

```
aws eks delete-fargate-profile --fargate-profile-name my-fargate-profile --
cluster-name my-cluster
```

4. 删除所有自行管理的节点 Amazon CloudFormation 堆栈。

a. 使用以下命令列出您的可用 Amazon CloudFormation 堆栈。在生成的输出中查找节点模板名称。

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

b. 使用以下命令（将 *node-stack* 替换为节点堆栈名称）删除每个节点堆栈。删除集群中的所有自行管理的节点堆栈。

```
aws cloudformation delete-stack --stack-name node-stack
```

5. 使用以下命令删除集群，同时将 *my-cluster* 替换为您的集群名称。

```
aws eks delete-cluster --name my-cluster
```

6. （可选）删除 VPC Amazon CloudFormation 堆栈。

a. 使用以下命令列出您的可用 Amazon CloudFormation 堆栈。在生成的输出中查找 VPC 模板名称。

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

b. 使用以下命令删除 VPC 堆栈，同时将 *my-vpc-stack* 替换为您的 VPC 堆栈名称。

```
aws cloudformation delete-stack --stack-name my-vpc-stack
```

防止意外删除 EKS 集群

意外删除 EKS 集群可能会影响 Kubernetes 集群的正常运行。

现在，您可以防止意外删除 EKS 集群。如果在集群上启用了删除保护，必须先禁用删除保护，然后才能删除该集群。

删除保护的目的是防止意外操作。您应谨慎限制有权删除集群的人员。

如果您尝试删除已启用删除保护的活动集群，将收到 `InvalidRequestException`。

> **Important**
>
> 如果您在集群上启用了删除保护，则必须同时拥有 UpdateClusterConfig 和 DeleteCluster IAM 权限才能先移除删除保护，并最终删除该集群。

> **Note**
>
> 如果集群状态为正在创建、失败或正在删除，即使已启用删除保护，您仍可以删除该集群。

为现有集群启用删除保护

您只能在处于活动状态的集群上运行此命令。

```
aws eks update-cluster-config --name <cluster-name> --region <aws-region> --deletion-protection
```

为现有集群禁用删除保护

```
aws eks update-cluster-config --name <cluster-name> --region <aws-region> --no-deletion-protection
```

集群 API 服务器端点

本主题可帮助您为 Amazon EKS 集群的 Kubernetes API 服务器端点启用私有访问，并限制或完全禁用通过互联网进行公有访问。

在创建新集群时，Amazon EKS 将为您用于与集群进行通信的托管 Kubernetes API 服务器（使用 Kubernetes 管理工具，如 `kubectl`）创建端点。默认情况下，此 API 服务器端点对于互联网是公有的，对 API 服务器的访问将使用 Amazon Identity and Access Management（IAM）与本机 Kubernetes 基于角色的访问控制（RBAC）相结合的方式加以保护。此端点称为集群公共端点。还有一个集群私有端点。有关集群私有端点的更多信息，请参阅以下部分：the section called “集群私有端点”。

IPv6 集群端点格式

EKS 按以下格式为 2024 年 10 月之后创建的新 IPv6 集群创建唯一的双堆栈端点。IPv6 集群是您在群集的 IP 系列 (ipFamily) 设置中选择了 IPv6 的集群。

Example

Amazon

EKS 集群公共/私有端点：`eks-cluster.region.api.aws`

Amazon GovCloud (US)

EKS 集群公共/私有端点：`eks-cluster.region.api.aws`

Amazon Web Services in China

EKS 集群公共/私有端点：`eks-cluster.region.api.amazonwebservices.com.cn`

> **Note**
>
> 双堆栈集群端点已于 2024 年 10 月推出。有关 IPv6 的更多信息，请参阅 the section called "IPv6"。在 2024 年 10 月之前创建的集群，请改用以下端点格式。

IPv4 集群端点格式

EKS 采用以下格式为在集群的 IP 系列（ipFamily）设置中选择了 IPv4 的每个集群创建一个唯一的端点：

Example

Amazon

EKS 集群公共/私有端点 `eks-cluster.region.eks.amazonaws.com`

Amazon GovCloud (US)

EKS 集群公共/私有端点 `eks-cluster.region.eks.amazonaws.com`

Amazon Web Services in China

EKS 集群公共/私有端点 `eks-cluster.region.amazonwebservices.com.cn`

> **Note**
>
> 在 2024 年 10 月之前，IPv6 集群也使用此端点格式。对于这些集群，公共端点和私有端点都只有从此端点解析的 IPv4 地址。

集群私有端点

您可以启用对 Kubernetes API 服务器的私有访问，以便节点与 API 服务器之间的所有通信都在 VPC 内。您可以限制从 Internet 访问 API 服务器的 IP 地址，或完全禁用对 API 服务器的 Internet 访问。

> **Note**
>
> 由于此端点用于 Kubernetes API 服务器，而不是用于与 Amazon API 通信的传统 Amazon PrivateLink 端点，因而不会在 Amazon VPC 控制台中显示为端点。

当您为集群启用端点私有访问时，Amazon EKS 将代表您创建一个 Route 53 私有托管区域，并将它与您的集群的 VPC 关联。这个私有托管区域由 Amazon EKS 管理，它不会出现在您的账户的 Route 53 资源中。为了使私有托管区域正确地将流量路由到您的 API 服务器，您的 VPC 必须将 `enableDnsHostnames` 和 `enableDnsSupport` 设置为 `true`，而且为 VPC 设置的 DHCP 选项必须在其域名服务器列表中包含 `AmazonProvidedDNS`。有关更多信息，请参阅 Amazon VPC 用户指南中的更新 VPC 的 DNS 支持。

您可以在创建新集群时定义 API 服务器终端节点的访问要求，并且可以随时更新集群的 API 服务器终端节点访问。

修改集群终端节点访问

使用本节中的过程来修改现有集群的终端节点访问。下表显示了受支持的 API 服务器终端节点访问组合及其关联的行为。

终端节点公有访问	终端节点私有访问	行为
已启用	已禁用	• 这是新 Amazon EKS 集群的默认行为。

终端节点公有访问	终端节点私有访问	行为
		• 源自集群的 VPC 内的 Kubernetes API 请求（如控制面板通信的节点）会脱离 VPC，但不会脱离 Amazon 网络。 • 集群 API 服务器可从 Internet 访问。您可以选择使用公共访问权限 CIDR 列表，通过 CIDR 数据块列表来控制对公有端点的访问权限。如果限制对特定 CIDR 数据块的访问，则建议您还启用私有端点，或者确保指定的 CIDR 数据块包括节点和 Fargate 容器组（pod）[如果您使用这些容器组（pod）] 用于访问公有端点的地址。
已启用	已启用	• 集群的 VPC 内的 Kubernetes API 请求（如控制面板通信的节点）使用私有 VPC 端点。您可以使用集群安全组上的规则来控制对私有端点的访问。 • 集群 API 服务器可从 Internet 访问。您可以选择使用公共访问权限 CIDR 列表，通过 CIDR 数据块列表来控制对公有端点的访问权限。 • 如果在 Amazon EKS 集群中使用混合节点，则不建议同时启用公有和私有端点访问权限。由于混合节点在 VPC 之外运行，因此会将集群端点解析为公有 IP 地址。对于具有混合节点的集群，建议使用公有或私有端点访问权限。

终端节点公有访问	终端节点私有访问	行为
已禁用	已启用	• 传输到集群 API 服务器的所有流量都必须来自您的集群的 VPC 或连接的网络中。 • 没有来自 Internet 的对 API 服务器的公有访问。所有 kubectl 命令都必须来自 VPC 或连接的网络中。对于连接选项，请参阅 the section called “访问私有 API 服务器”。 • 您可以使用集群安全组上的规则来控制对私有端点的访问。请注意，公共访问权限 CIDR 不会影响私有端点。 • 公有 DNS 服务器将集群的 API 服务器端点解析为 VPC 中的私有 IP 地址。过去，终端节点只能在 VPC 内部解析。 如果您的终端节点未解析为现有集群的 VPC 内的私有 IP 地址，您可以： • 启用公有访问，然后重新禁用。您只需为集群执行一次该操作，终端节点就将从该时间点开始解析为私有 IP 地址。 • 更新您的集群。

端点访问控制

请注意，以下用于控制端点访问的各个方法都只会影响相应的端点。

集群安全组

集群安全组控制两种类型的连接：与 kubelet API 的连接和私有端点的连接。与 kubelet API 的连接用于 kubectl attach、kubectl cp、kubectl exec、kubectl logs 和 kubectl port-forward 命令中。集群安全组不会影响公有端点。

公共访问权限 CIDR

公共访问权限 CIDR 通过 CIDR 数据块列表来控制对公有端点的访问权限。请注意，公共访问权限 CIDR 不会影响私有端点。公共访问权限 CIDR 在 IPv6 集群和 IPv4 集群上的行为会有所不同，具体取决于它们的创建日期，如下所述：

公有端点（**IPv6** 集群）中的 CIDR 块

您可以将 IPv6 和 IPv4 CIDR 块添加到 IPv6 集群的公有端点，因为公有端点是双堆栈端点。这仅适用于您在 2024 年 10 月或之后创建的且将 ipFamily 设置为 IPv6 的新集群。您可以通过新的端点域名 api.aws 来识别这些集群。

公有端点（**IPv4** 集群）中的 CIDR 块

您可以将 IPv4 CIDR 块添加到 IPv4 集群的公有端点。您无法将 IPv6 CIDR 块添加到 IPv4 集群的公有端点。如果您尝试添加，EKS 将返回以下错误消息：The following CIDRs are invalid in publicAccessCidrs

公有端点中的 CIDR 块（2024 年 10 月之前创建的 **IPv6** 集群）

您可以将 IPv4 CIDR 块添加到您在 2024 年 10 月之前创建的旧 IPv6 集群的公有端点。您可以通过 eks.amazonaws.com 端点识别这些集群。您无法将 IPv6 CIDR 块添加到您在 2024 年 10 月之前创建的旧 IPv6 集群的公有端点。如果您尝试添加，EKS 将返回以下错误消息：The following CIDRs are invalid in publicAccessCidrs

访问私有 API 服务器

如果已禁用集群的 Kubernetes API 服务器端点的公有访问，则只能从 VPC 或连接的网络内访问 API 服务器。以下是访问 Kubernetes API 服务器终端节点的部分可行方法：

连接的网络

使用 Amazon Transit Gateway 或其它连接选项将您的网络连接到 VPC，然后使用连接的网络中的计算机。必须确保 Amazon EKS 控制面板安全组包含允许来自所连接网络的入口流量通过端口 443 的规则。

Amazon EC2 堡垒主机

您可以在集群的 VPC 中将 Amazon EC2 实例启动到公共子网中，然后通过 SSH 登录到该实例来运行 kubectl 命令。有关更多信息，请参阅Amazon上的 Linux 堡垒主机。必须确保 Amazon EKS 控制面板安全组包含允许来自堡垒主机的入口流量通过端口 443 的规则。有关更多信息，请参阅 the section called “安全组要求”。

在为堡垒主机配置 kubectl 时，请确保使用已映射到集群的 RBAC 配置的 Amazon 凭证，或在移除端点公共访问之前添加您的堡垒机将用于 RBAC 配置的 IAM 主体。有关更多信息，请参阅the section called “Kubernetes API 访问权限”和the section called “未经授权或访问被拒绝 (kubectl)”。

AmazonCloud9 IDE

Amazon Cloud9 是一种基于云的集成式开发环境（IDE），您只需要一个浏览器，即可编写、运行和调试代码。您可以在集群的 VPC 中创建 Amazon Cloud9 IDE，然后使用 IDE 来与集群进行通信。有关更多信息，请参阅在 Amazon Cloud9 中创建环境。必须确保 Amazon EKS 控制面板安全组包含允许来自 IDE 安全组的入口流量通过端口 443 的规则。有关更多信息，请参阅 the section called “安全组要求”。

在为 Amazon Cloud9 IDE 配置 `kubectl` 时，请确保使用已映射到集群的 RBAC 配置的 Amazon 凭证，或在移除端点公共访问之前添加您的 IDE 将用于 RBAC 配置的 IAM 主体。有关更多信息，请参阅the section called “Kubernetes API 访问权限”和the section called “未经授权或访问被拒绝 (`kubectl`)”。

在 GitHub 上编辑此页面

配置对集群 API 服务器端点的网络访问权限

您可以在以下部分中使用 Amazon Web Services 管理控制台或 Amazon CLI 修改集群 API 服务器端点访问。

配置端点访问 – Amazon 控制台

1. 打开 Amazon EKS console（Amazon EKS 控制台）。
2. 选择集群的名称可以显示集群信息。
3. 选择联网选项卡，然后选择管理端点访问。
4. 对于私有访问，选择是启用还是禁用集群的 Kubernetes API 服务器端点的私有访问。如果启用私有访问，源自集群的 VPC 内的 Kubernetes API 请求将使用私有 VPC 端点。您必须启用私有访问以禁用公有访问。
5. 对于公有访问，选择是启用还是禁用集群的 Kubernetes API 服务器端点的公有访问。如果禁用公有访问，集群的 Kubernetes API 服务器只能接收来自集群 VPC 内的请求。
6. （可选）如果您已启用公有访问，则可以指定互联网中的哪些地址可以与公共端点通信。选择 Advanced Settings (高级设置)。输入 CIDR 块，例如 *`203.0.113.5/32`*。该块不能包含预留地址。您可以通过选择 Add Source (添加源) 来输入其他块。您可以指定的 CIDR 块存在最大数量限制。有关更多信息，请参阅 the section called “服务配额”。如果未指定任何块，则公有 API 服务器端点将接收来自双堆栈 `IPv6` 集群的所有 `IPv4` (`0.0.0.0/0`) 和 `IPv6` (`::/0`) IP 地址的请求。如果您使用 CIDR 块限制对公有端点的访问，建议您还启用私有端点访问，以便节点和 Fargate Pods（如果您使用它们）可以与集群进行通信。在未启用私有终端节点的情况下，您的公有访问终

端节点 CIDR 源必须包含来自 VPC 的出口源。例如，如果您在私有子网中有一个节点，该节点通过 NAT 网关与 Internet 通信，则您需要将 NAT 网关的出站 IP 地址添加作为公有端点上的容许 CIDR 块的一部分。

7. 选择 Update (更新) 完成操作。

配置端点访问 – Amazon CLI

使用 Amazon CLI 版本 `1.27.160` 或更高版本完成以下步骤。您可以使用 `aws --version` 检查您的当前版本。要安装或升级 Amazon CLI，请参阅安装 Amazon CLI。

1. 使用下面的 Amazon CLI 命令更新集群 API 服务器端点访问。替换您的集群名称和所需的终端节点访问值。如果设置了 `endpointPublicAccess=true`，则可以（可选）输入单个 CIDR 块，或者输入 `publicAccessCidrs` 的用逗号分隔的 CIDR 块列表。块不能包含预留地址。如果您指定 CIDR 块，则公有 API 服务器终端节点将只接收来自列出的块的请求。您可以指定的 CIDR 块存在最大数量限制。有关更多信息，请参阅 the section called “服务配额”。如果您使用 CIDR 块限制对公有端点的访问，建议您还启用私有端点访问，以便节点和 Fargate Pods（如果您使用这些 Pod）可以与集群进行通信。在未启用私有终端节点的情况下，您的公有访问终端节点 CIDR 源必须包含来自 VPC 的出口源。例如，如果您在私有子网中有一个节点，该节点通过 NAT 网关与 Internet 通信，则您需要将 NAT 网关的出站 IP 地址添加作为公有端点上的容许 CIDR 块的一部分。如果未指定任何 CIDR 块，则公有 API 服务器端点将接收来自双堆栈 IPv6 集群的所有 (0.0.0.0/0) 和 IPv6 (`::/0`) 的 IP 地址请求。

> **Note**
>
> 以下命令为 API 服务器终端节点启用来自单个 IP 地址的私有访问和公有访问。将 *203.0.113.5/32* 替换为要限制网络访问的单个 CIDR 块或用逗号分隔的 CIDR 块列表。

```
aws eks update-cluster-config \
    --region region-code \
    --name my-cluster \
    --resources-vpc-config
 endpointPublicAccess=true,publicAccessCidrs="203.0.113.5/32",endpointPrivateAccess=true
```

示例输出如下。

```
{
```

```
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "InProgress",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\"203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

2. 使用以下命令通过上一命令返回的集群名称和更新 ID 监控您的终端节点访问更新的状态。当状态显示为 Successful 时，您的更新将完成。

```
aws eks describe-update \
    --region region-code \
    --name my-cluster \
    --update-id e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000
```

示例输出如下。

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "Successful",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
```

```
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\"203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

在 GitHub 上编辑此页面

在 EKS 集群上部署 Windows 节点

了解如何为 Amazon EKS 集群启用和管理 Windows 支持，以便同时运行 Windows 容器和 Linux 容器。

注意事项

在部署 Windows 节点之前，请注意以下注意事项。

- EKS 自动模式不支持 Windows 节点
- 您可以使用 `HostProcess` 容器组（pod）在 Windows 节点上使用主机网络。有关更多信息，请参阅 Kubernetes 文档中的创建 Windows HostProcessPod。
- Amazon EKS 集群必须包含一个或多个 Linux 或 Fargate 节点，才能运行仅在 Linux 上运行的核心系统容器组（pod），如 CoreDNS。
- `kubelet` 和 `kube-proxy` 事件日志将重定向到 `EKS Windows` 事件日志，并设置为 200MB 限制。
- 您不能对在 Windows 节点上运行的容器组（pod）使用将安全组分配到单个容器组（pod）。
- 您不能对 Windows 节点使用自定义联网。
- 您不能将 `IPv6` 与 Windows 节点结合使用。
- Windows 节点支持每个节点一个弹性网络接口。默认情况下，每个 Windows 节点可以运行的容器组（pod）数等于每个弹性网络接口可用于节点实例类型的 IP 地址数减 1。有关更多信息，请参阅《Amazon EC2 用户指南》中的每种实例类型的每个网络接口的 IP 地址数。

- 在 Amazon EKS 集群中，采用负载均衡器的单个服务最多可支持 1024 个后端容器组（pod）。每个 Pod 都有自己的唯一 IP 地址。对于从 OS 内部版本 17763.2746 开始的 Windows 服务器更新，之前的 64 个 容器组（pod）限制已经没有了。
- Fargate 上的 Amazon EKS 容器组（pod）不支持 Windows 容器。
- 不能使用将 Windows 作为主机操作系统的 Amazon EKS 混合节点。
- 无法从 `vpc-resource-controller` 容器组（pod）检索日志。以前在将此控制器部署到数据面板时则可以检索。
- 在将 IPv4 地址分配给新容器组（pod）之前，会有一段“冷却”时间。这样可以防止流量因过时的 `kube-proxy` 规则而流向具有相同 IPv4 地址的旧容器组（pod）。
- 控制器的源代码在 GitHub 上进行管理。要为该控制器贡献代码或提交针对该控制器的问题，请访问 GitHub 上的相应项目。
- 为 Windows 托管节点组指定自定义 AMI ID 时，请将 `eks:kube-proxy-windows` 添加到您的 Amazon IAM 身份验证器配置映射中。有关更多信息，请参阅 the section called “指定 AMI ID 时的限制和条件”。
- 如果保留可用的 IPv4 地址对您的子网至关重要，请参阅 EKS Best Practices Guide - Windows Networking IP Address Management 以获取指导。
- EKS 访问条目注意事项
  - 用于 Windows 节点的访问条目所需类型为 `EC2_WINDOWS`。有关更多信息，请参阅 the section called “创建访问条目”。

    要为 Windows 节点创建访问条目，请执行以下操作：

    ```
    aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws-cn:iam::111122223333:role/<role-name> --type EC2_Windows
    ```

先决条件

- 现有集群。
- 您的集群必须至少有一个（我们建议至少有两个）Linux 节点或 Fargate 容器组（pod）才能运行 CoreDNS。如果您启用旧版 Windows 支持，则必须使用 Linux 节点而非 Fargate 容器组（pod）来运行 CoreDNS。
- 现有 Amazon EKS 集群 IAM 角色。

启用 Windows 支持

1. 如果集群中没有 Amazon Linux 节点，并且使用容器组（pod）的安全组，请跳至下一步。否则，请确认 AmazonEKSVPCResourceController 托管策略是否已附加到您的集群角色。请将 *eksClusterRole* 替换为集群角色的名称。

```
aws iam list-attached-role-policies --role-name eksClusterRole
```

示例输出如下。

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonEKSClusterPolicy",
            "PolicyArn": "arn:aws-cn:iam::aws:policy/AmazonEKSClusterPolicy"
        },
        {
            "PolicyName": "AmazonEKSVPCResourceController",
            "PolicyArn": "arn:aws-cn:iam::aws:policy/AmazonEKSVPCResourceController"
        }
    ]
}
```

如果像前面的输出中那样已经附加了策略，请跳过下一步。

2. 将 AmazonEKSVPCResourceController 托管策略附加到您的 Amazon EKS 集群 IAM 角色。请将 *eksClusterRole* 替换为集群角色的名称。

```
aws iam attach-role-policy \
  --role-name eksClusterRole \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSVPCResourceController
```

3. 更新 VPC CNI ConfigMap 以启用 Windows IPAM。请注意，如果使用 Helm 图表或作为 Amazon EKS 附加组件将 VPC CNI 安装在您的集群上，则可能无法直接修改 ConfigMap。有关配置 Amazon EKS 附加组件的信息，请参阅the section called “可以自定义的字段”。

   a. 创建名为 *vpc-resource-controller-configmap.yaml* 的文件，具有以下内容。

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-windows-ipam: "true"
```

b. 将 ConfigMap 应用于集群。

```
kubectl apply -f vpc-resource-controller-configmap.yaml
```

4. 如果集群已将身份验证模式设置为启用 aws-auth Configmap：

- 验证 aws-auth ConfigMap 是否包含 Windows 节点的实例角色映射，以包含 eks:kube-proxy-windows RBAC 权限组。您可以通过运行以下命令进行验证。

```
kubectl get configmap aws-auth -n kube-system -o yaml
```

示例输出如下。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      - eks:kube-proxy-windows # This group is required for Windows DNS resolution
 to work
      rolearn: arn:aws-cn:iam::111122223333:role/eksNodeRole
      username: system:node:{{EC2PrivateDNSName}}
[...]
```

您应该会看到组下面列出的 eks:kube-proxy-windows。如果未指定组，则需要更新 ConfigMap 或进行创建，以包含所需的组。有关 aws-auth ConfigMap 的更多信息，请参阅 the section called “将 aws-authConfigMap 应用到集群”。

5. 如果集群已将身份验证模式设置为禁用 aws-auth Configmap，则可使用 EKS 访问条目。创建一个用于 Windows 实例的新节点角色，之后 EKS 将自动创建 EC2_WINDOWS 类型的访问条目。

部署 Windows 容器组（pod）

将容器组（pod）部署到集群时，如果您运行的是多种不同类型的节点，则需要指定这些容器组（pod）所用的操作系统。

对于 Linux 容器组（pod），请使用清单中的以下节点选择器文本。

```
nodeSelector:
        kubernetes.io/os: linux
        kubernetes.io/arch: amd64
```

对于 Windows 容器组（pod），请使用清单中的以下节点选择器文本。

```
nodeSelector:
        kubernetes.io/os: windows
        kubernetes.io/arch: amd64
```

可以部署一款示例应用程序，以查看正在使用的节点选择器。

在 Windows 节点上支持更高的容器组（pod）密度

在 Amazon EKS 中，每个容器组（pod）都会从您的 VPC 分配一个 IPv4 地址。因此，即使有足够的资源可以在节点上运行更多容器组（pod），可以部署到该节点的容器组（pod）数量也受到可用 IP 地址的限制。由于 Windows 节点仅支持一个弹性网络接口，因此默认情况下，Windows 节点上可用 IP 地址的最大数量等于：

```
Number of private IPv4 addresses for each interface on the node - 1
```

一个 IP 地址用作网络接口的主要 IP 地址，因此无法将其分配给容器组（pod）。

通过启用 IP 前缀委派，可以在 Windows 节点上启用更高的容器组（pod）密度。此功能使您可以为主网络接口分配 /28 IPv4 前缀，而不是分配辅助 IPv4 地址。分配 IP 前缀会将节点上的最大可用 IPv4 地址数增加到：

```
(Number of private IPv4 addresses assigned to the interface attached to the node - 1) *
 16
```

由于可用 IP 地址的数量要多得多，可用的 IP 地址不应限制您在节点上扩展容器组（pod）数量的能力。有关更多信息，请参阅 the section called “增加 IP 地址”。

禁用 Windows 支持

1. 如果您的集群包含 Amazon Linux 节点，并且您对这些节点使用容器组（pod）安全组，请跳过此步骤。

   从您的集群角色中删除 `AmazonVPCResourceController` 这项托管的 IAM policy。请将 *`eksClusterRole`* 替换为集群角色的名称。

   ```
   aws iam detach-role-policy \
       --role-name eksClusterRole \
       --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSVPCResourceController
   ```

2. 在 `amazon-vpc-cni` ConfigMap 中禁用 Windows IPAM。

   ```
   kubectl patch configmap/amazon-vpc-cni \
                       -n kube-system \
                       --type merge \
                       -p '{"data":{"enable-windows-ipam":"false"}}'
   ```

部署具有有限互联网访问权限的私有集群

本主题将介绍如何部署部署在 Amazon 云上但没有出站互联网访问权限的 Amazon EKS 集群。如果您在 Amazon 上有一个本地集群，请参阅the section called “Nodes”而不是本主题。

如果您不熟悉 Amazon EKS 联网，请参阅 Amazon EKS 工作线程节点的集群联网解密。如果集群没有出站互联网访问权限，则其必须满足以下要求：

集群架构要求

- 您的集群必须从 VPC 中的容器注册表中拉取映像。您可以在 VPC 中创建 Amazon Elastic Container Registry，并将容器映像复制到其中，以供节点拉取。有关更多信息，请参阅 the section called “将镜像复制到存储库”。
- 集群必须启用端点私有访问权限。需要启用该访问权限，节点才能注册到集群端点。终端节点公有访问权限是可选的。有关更多信息，请参阅 the section called “集群终端节点访问”。

节点要求

- 自主管理型 Linux 和 Windows 节点必须包含以下引导参数才能启动。这些实际参数绕过 Amazon EKS 自检，且无需从 VPC 内访问 Amazon EKS API。

  a. 使用以下命令确定集群端点的值。将 *my-cluster* 替换为您的集群的名称。

  ```
  aws eks describe-cluster --name my-cluster --query cluster.endpoint --output text
  ```

  示例输出如下。

  ```
  https://EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-code.eks.amazonaws.com
  ```

  b. 使用以下命令确定集群证书颁发机构的值。将 *my-cluster* 替换为您的集群的名称。

  ```
  aws eks describe-cluster --name my-cluster --query cluster.certificateAuthority --
  output text
  ```

  返回的输出是一个长字符串。

  c. 请将 NodeConfig 对象中的 `apiServerEndpoint` 和 `certificateAuthority` 值替换为以上命令的输出中返回的值。有关在启动 Amazon Linux 2023 自主管理型节点时指定引导参数的更多信息，请参阅the section called “Amazon Linux”和the section called “Windows”。

  - 对于 Linux 节点：

  ```
  ---
  MIME-Version: 1.0
  Content-Type: multipart/mixed; boundary="BOUNDARY"

  --BOUNDARY
  Content-Type: application/node.eks.aws

  ---
  apiVersion: node.eks.aws/v1alpha1
  kind: NodeConfig
  spec:
    cluster:
      name: my-cluster
      apiServerEndpoint: [.replaceable]https://
  EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-code.eks.amazonaws.com
      certificateAuthority: [.replaceable]Y2VydGlmaWNhdGVBdXRob3JpdHk=
  ```

```
...
```

有关更多参数，请参阅 GitHub 上的引导启动脚本。

- 对于 Windows 节点：

> **Note**
>
> 如果您使用自定义服务 CIDR，则需要使用 `-ServiceCIDR` 参数进行指定。否则，集群中的容器组（pod）DNS 解析将失败。

```
-APIServerEndpoint cluster-endpoint -Base64ClusterCA certificate-authority
```

有关更多参数，请参阅 the section called “引导脚本配置参数”。

- 必须从 VPC 内创建集群的 `aws-auth ConfigMap`。有关在 `aws-auth ConfigMap` 中创建和添加条目的更多信息，请在终端中输入 `eksctl create iamidentitymapping --help`。如果您的服务器上不存在 `ConfigMap`，则 `eksctl` 将在您使用命令添加身份映射时创建。

容器组（pod）要求

- 容器组身份：配置了 EKS 容器组身份的容器组（pod）从 EKS Auth API 获取凭证。如果没有出站互联网访问权限，则必须创建并使用一个用于 EKS Auth API 的 VPC 端点：`com.amazonaws.region-code.eks-auth`。有关 EKS 和 EKS Auth VPC 端点的更多信息，请参阅the section called “ Amazon PrivateLink”。
- IRSA：配置了服务账户的 IAM 角色的容器组（pod）通过 Amazon Security Token Service（Amazon STS）API 调用获取凭证。如果没有出站互联网访问权限，则您必须在 VPC 中创建并使用 Amazon STS VPC 端点。默认情况下，大多数 Amazon v1 SDK 均使用全局 Amazon STS 端点（`sts.amazonaws.com`），而此端点不使用 Amazon STS VPC 端点。要使用 Amazon STS VPC 端点，您可能需要将 SDK 配置为使用区域 Amazon STS 端点（`sts.region-code.amazonaws.com`）。有关更多信息，请参阅 the section called “STS 端点”。
- 集群的 VPC 子网必须具有容器组（pod）需要访问的任何 Amazon 服务的 VPC 接口端点。有关更多信息，请参阅使用接口 VPC 端点访问 Amazon 服务。下表列出了一些常用的服务和端点。有关端点的完整列表，请参阅《Amazon PrivateLink 指南》https://docs.amazonaws.cn/vpc/latest/privatelink/中的与 Amazon PrivateLink 集成的 Amazon 服务。

我们建议您为 VPC 端点启用私有 DNS 名称，这样工作负载即可继续正常使用公共 Amazon 服务端点。

服务	端点
Amazon EC2	com.amazonaws.*region-code* .ec2
Amazon Elastic Container Registry（用于拉取容器映像）	com.amazonaws.*region-code* .ecr.api、com.amazonaws.*region-code* .ecr.dkr 和 com.amazonaws.*region-code* .s3
Amazon 应用程序负载均衡器和网络负载均衡器	com.amazonaws.*region-code* .elasticloadbalancing
（可选）Amazon X-Ray（将跟踪数据发送到 Amazon X-Ray 时需要）	com.amazonaws.*region-code* .xray
（可选）Amazon SSM（SSM Agent 执行节点管理任务时需要，可作为 SSH 的替代方案）	com.amazonaws.*region-code* .ssm
Amazon CloudWatch Logs [将节点和容器组（pod）日志发送到 Amazon CloudWatch Logs 时需要]	com.amazonaws.*region-code* .logs
Amazon Security Token Service（将 IAM 角色用于服务账户时为必需）	com.amazonaws.*region-code* .sts
Amazon EKS 身份验证（使用容器组身份关联时为必需）	com.amazonaws.*region-code* .eks-auth
Amazon EKS	com.amazonaws.*region-code* .eks

- 任何自行管理的节点都必须部署到具有您需要的 VPC 接口端点的子网中。如果您创建托管节点组，则 VPC 接口端点安全组必须允许子网的 CIDR，或者您必须将已创建的节点安全组添加到 VPC 接口端点安全组。
- EFS 存储：如果容器组（pod）使用 Amazon EFS 卷，在部署使用 Amazon EFS 存储弹性文件系统之前，必须更改驱动程序的 kustomization.yaml 文件，才能将容器映像设置为使用与 Amazon EKS 集群相同的 Amazon 区域。

- Route53 不支持 Amazon PrivateLink。您无法从私有 Amazon EKS 集群管理 Route53 DNS 记录。这会影响 Kubernetes external-dns。
- 如果您使用 EKS 优化版 AMI，则应启用上表中的 `ec2` 端点。您也可以手动设置节点 DNS 名称。优化版 AMI 使用 EC2 API 来自动设置节点 DNS 名称。
- 您可以使用 Amazon Load Balancer Controller 将 Amazon Application Load Balancers（ALB）和 Network Load Balancers 部署到私有集群。部署时，您应该使用命令行标记将 `enable-shield`、`enable-waf` 和 `enable-wafv2` 设置为 false。带有来自 Ingress 对象中的主机名的证书发现不受支持。这是因为控制器需要访问没有 VPC 接口端点的 Amazon Certificate Manager。

  该控制器支持带有 IP 目标的网络负载均衡器，结合 Fargate 一起使用时需要网络负载均衡器。有关更多信息，请参阅the section called “应用程序负载均衡”和the section called “创建网络负载均衡器”。
- 支持 Cluster Autoscaler。在部署 Cluster Autoscaler 容器组（pod）时，请确保命令行包括 `--aws-use-static-instance-list=true`。有关更多信息，请参阅 GitHub 上的使用静态实例列表。Worker 节点 VPC 还必须包括 Amazon STS VPC 端点和弹性伸缩 VPC 端点。
- 一些容器软件产品使用 API 调用来访问监控使用情况的 Amazon Marketplace Metering Service。私有集群不允许这些调用，因此您不能在私有集群中使用这些容器类型。

使用 Karpenter 和 Cluster Autoscaler 扩展集群计算

Autoscaling 是一项功能，可以自动扩缩资源以满足您不断变化的需求。若没有此项重要的 Kubernetes 功能，则需要耗费大量的人力资源来手动执行这些工作。

EKS 自动模式

Amazon EKS 自动模式会自动扩展集群计算资源。如果现有节点无法满足容器组的需要，EKS 自动模式会创建一个新节点。EKS 自动模式还会整合工作负载并删除节点。EKS 自动模式是以 Karpenter 为基础构建的。

有关更多信息，请参阅：

- EKS 自动模式
- the section called “创建节点池”
- the section called “部署 inflate 工作负载”

其他解决方案

Amazon EKS 支持另外两个自动扩缩产品：

Karpenter

Karpenter 是一款灵活的高性能 Kubernetes 集群自动扩缩器，可帮助提高应用程序可用性和集群效率。Karpenter 只需不到一分钟时间，即可启动适当规模的计算资源（例如 Amazon EC2 实例）来响应不断变化的应用程序负载。通过将 Kubernetes 与 Amazon 相集成，Karpenter 可以即时调配精准满足工作负载需求的计算资源。Karpenter 会根据集群工作负载的具体需求来自动调配新的计算资源。这包括计算、存储、加速和调度需求。Amazon EKS 支持使用 Karpenter 的集群，但 Karpenter 可以与任何合规的 Kubernetes 集群配合使用。有关更多信息，请参阅 Karpenter 文档。

> **Important**
>
> Karpenter 是一种开源软件，Amazon 客户负责在其 Kubernetes 集群中安装、配置和管理。Amazon 在 Amazon EKS 集群中使用兼容版本运行未经修改的 Karpenter 时提供技术支持。客户在升级 Karpenter 控制器或运行它的 Kubernetes 集群时，必须维护 Karpenter 控制器的可用性和安全性以及适当的测试程序，就像任何其它客户管理的软件一样。Karpenter 没有 Amazon 服务水平协议（SLA），客户有责任确保 Karpenter 启动的 EC2 实例满足其业务需求。

Cluster Autoscaler

当 Pod 失败或被重新安排到其他节点时，Kubernetes Cluster Autoscaler 会自动调整集群中的节点数。Cluster Autoscaler 使用自动扩缩组。有关更多信息，请参阅 Amazon 上的 Cluster Autoscaler。

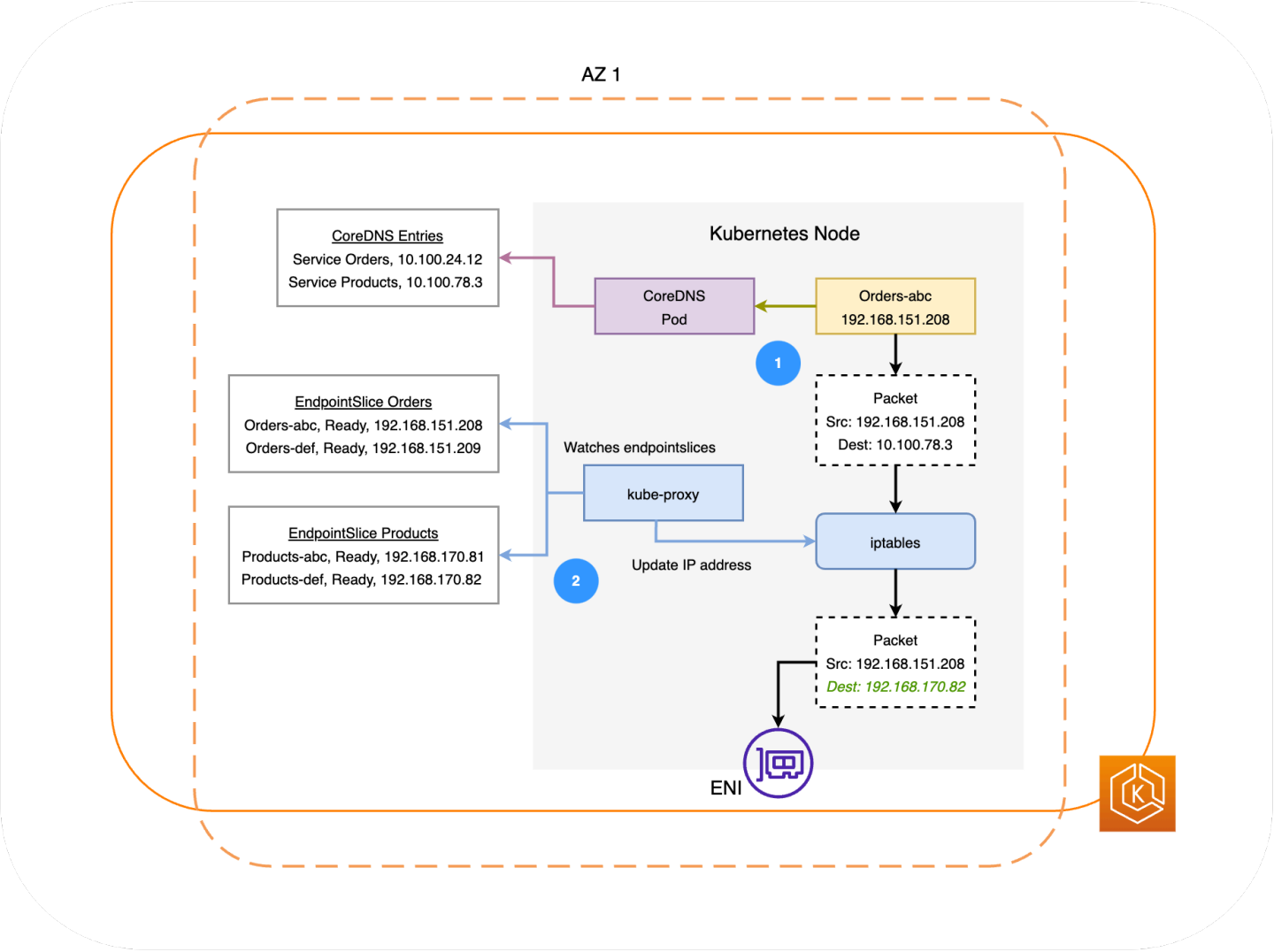
了解 Amazon EKS 中的 Amazon 应用程序恢复控制器（ARC）可用区转移

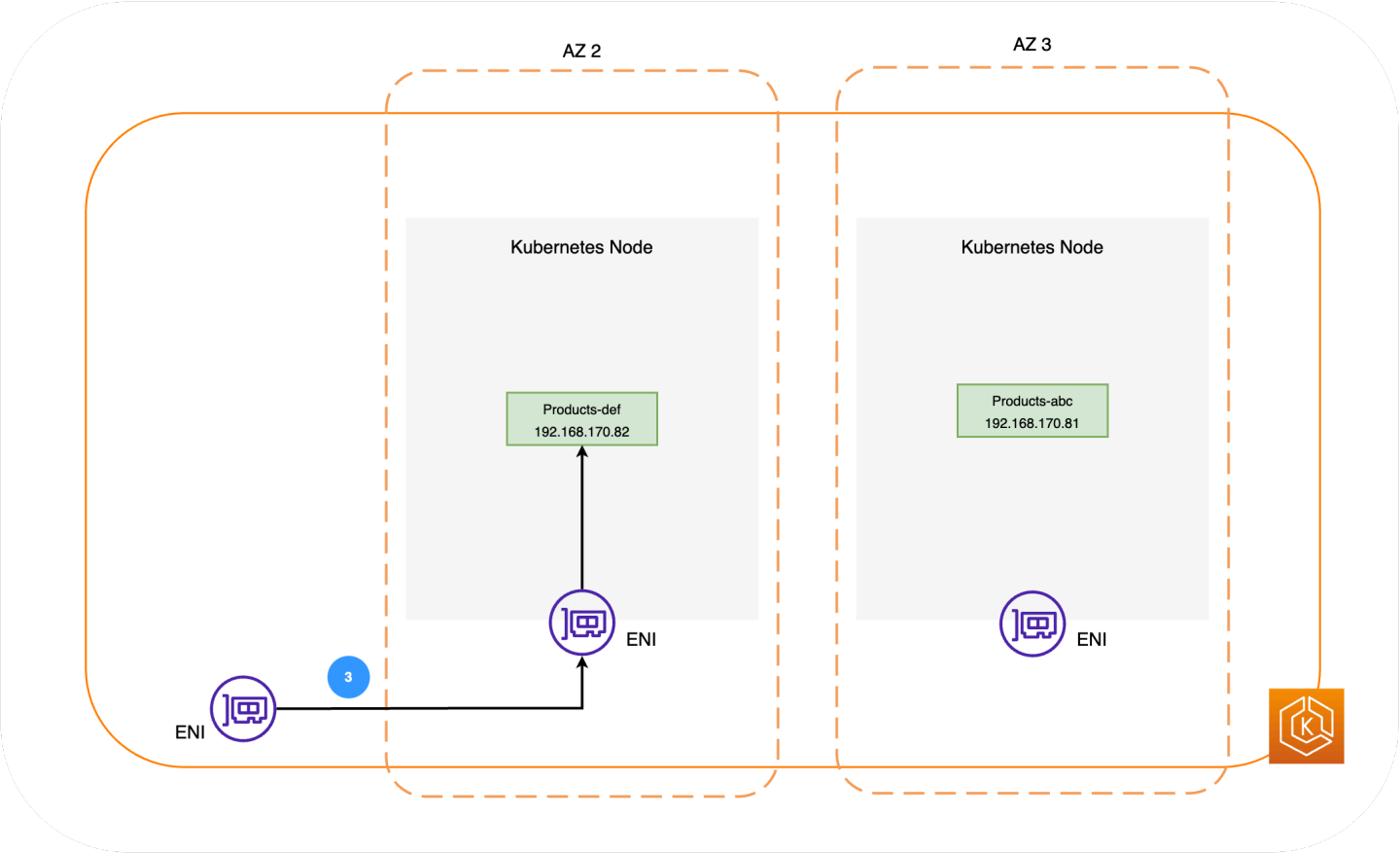
Kubernetes 具有原生功能，可让您的应用程序对可用区（AZ）运行状况下降或损坏等事件更具弹性。在 Amazon EKS 集群中运行工作负载时，您可以使用 Amazon 应用程序恢复控制器（ARC）的可用区转移或可用区自动转移，进一步改善应用程序环境的容错能力和应用程序恢复。ARC 可用区转移旨在作为一种临时措施，可让您将资源流量从损坏的可用区转出，直到可用区转移到期或您取消它。如有必要，可以延长可用区转移。

您可以为 EKS 集群启动可用区转移，也可以通过启用可用区自动转移来让 Amazon 为您执行可用区转移。这种转移会更新集群中东-西网络流量，使其仅考虑在运行状况良好的可用区中的 Worker 节点上运行的容器组（pod）的网络端点。此外，任何处理 EKS 集群中应用程序入口流量的 ALB 或 NLB 都会自动将流量路由到运行状况良好的可用区中的目标。对于那些寻求最高可用性目标的客户来说，在可用区损坏的情况下，能够将所有流量从损坏的可用区转移出去可能很重要，直到可用区恢复为止。为此，您还可以启用带有 ARC 可用区转移的 ALB 或 NLB。

了解容器组（pod）之间的东-西网络流量

下图说明了两个示例工作负载，即“订单”和“产品”。此示例的目的是展示不同可用区中的工作负载和容器组（pod）是如何通信的。





1. 要使“订单”与“产品”通信，它必须首先解析目标服务的 DNS 名称。“订单”将与 CoreDNS 通信以获取该服务的虚拟 IP 地址（集群 IP）。“订单”解析“产品”服务名称后，它会将流量发送到该目标 IP 地址。
2. kube-proxy 在集群中的每个节点上运行，并持续监视 EndpointSlices 的服务。创建服务时，EndpointSlice 控制器将在后台创建和管理 EndpointSlice。每个 EndpointSlice 都有一个端点列表或表格，其中包含容器组（pod）地址的子集以及它们在其中正在运行的节点。kube-proxy 在节点上为每个容器组（pod）端点设置 `iptables` 路由规则。kube-proxy 还负责一种基本的负载均衡，也就是将发往服务集群 IP 地址的流量重定向为直接发送到容器组（pod）的 IP 地址。kube-proxy 通过重写传出连接上的目标 IP 地址来实现此目的。
3. 然后，网络数据包通过相应节点上的 ENI 发送到可用区 2 中的“产品”容器组（pod）（如前面的图表所示）。

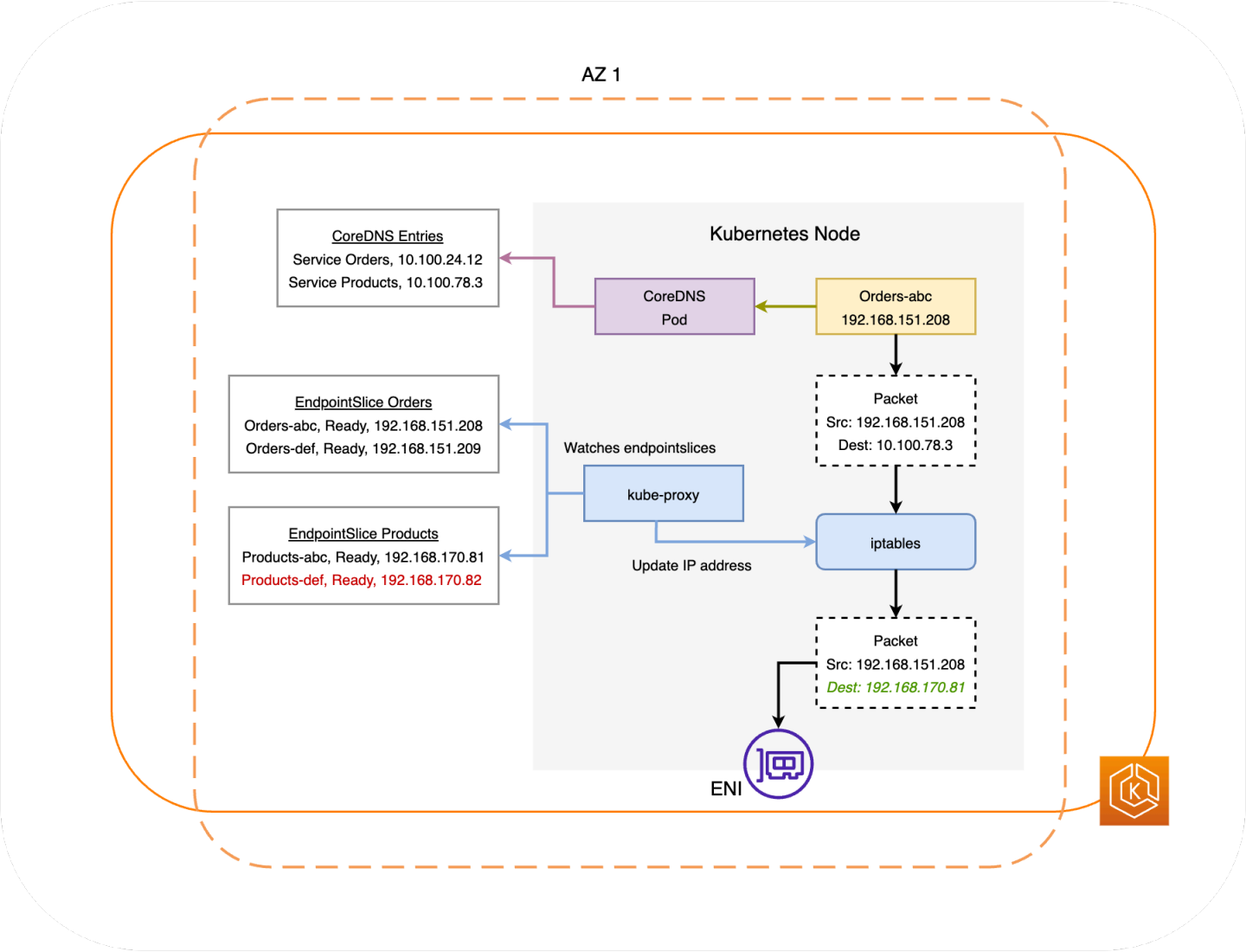
了解 Amazon EKS 中的 ARC 可用区转移

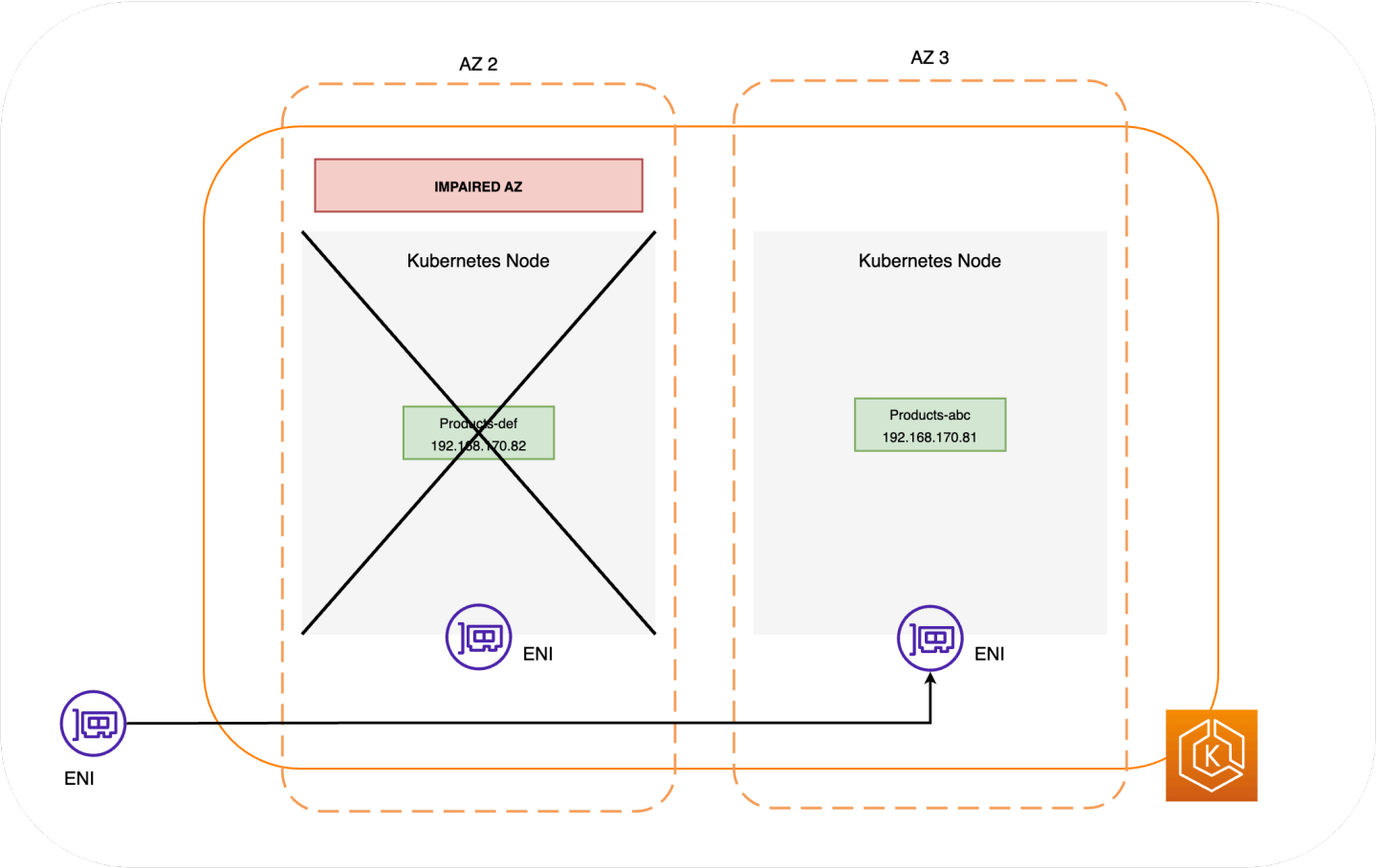
如果您的环境中存在可用区损坏，则可以为您的 EKS 集群环境启动可用区转移。或者，您可以允许 Amazon 使用可用区自动转移来为您管理流量转移。使用可用区自动转移，Amazon 可监控整体可用区运行状况，并通过自动将流量从集群环境中损坏的可用区转移出去来应对潜在的可用区损坏。

使用 ARC 启用 Amazon EKS 集群可用区转移后，您可以使用 ARC 控制台、Amazon CLI 或可用区转移和可用区自动转移 API，启动可用区转移或启用可用区自动转移。在 EKS 可用区转移期间，将自动执行以下操作：

- 受影响可用区中的所有节点都被封锁。这将防止 Kubernetes 调度器将新容器组（pod）调度到运行状况不佳的可用区中的节点上。
- 如果您使用的是托管节点组，则可用区重新平衡将被暂停，并且您的自动扩缩组也将更新，以确保新的 EKS 数据面板节点仅在运行状况良好的可用区中启动。
- 运行状况不佳的可用区中的节点不会被终止，容器组（pod）也不会被逐出这些节点。这可确保当可用区转移到期或被取消时，您的流量可以安全地返回到可用区以获得完整容量。
- EndpointSlice 控制器将在损坏的可用区中找到所有容器组（pod）端点，并将它们从相关的 EndpointSlices 中移除。这将确保只有运行状况良好的可用区中的容器组（pod）端点才会成为接收网络流量的目标。当可用区转移被取消或到期时，EndpointSlice 控制器将更新 EndpointSlices 以包括已恢复的可用区中的端点。

下图简要概述了 EKS 可用区转移如何确保集群环境中仅针对运行状况良好的容器组（pod）端点。


AZ 1
CoreDNS Entries
Service Orders, 10.100.24.12
Service Products, 10.100.78.3
Kubernetes Node
CoreDNS
Pod
Orders-abc
192.168.151.208
Packet
Src: 192.168.151.208
Dest: 10.100.78.3
EndpointSlice Orders
Orders-abc, Ready, 192.168.151.208
Orders-def, Ready, 192.168.151.209
Watches endpointslices
kube-proxy
EndpointSlice Products
Products-abc, Ready, 192.168.170.81
Products-def, Ready, 192.168.170.82
iptables
Update IP address
Packet
Src: 192.168.151.208
Dest: 192.168.170.81
ENI



EKS 可用区转移要求

要在 EKS 中成功进行可用区转移，您必须事先设置集群环境，使其能够抵御可用区损坏。以下是有助于确保韧性的配置选项列表。

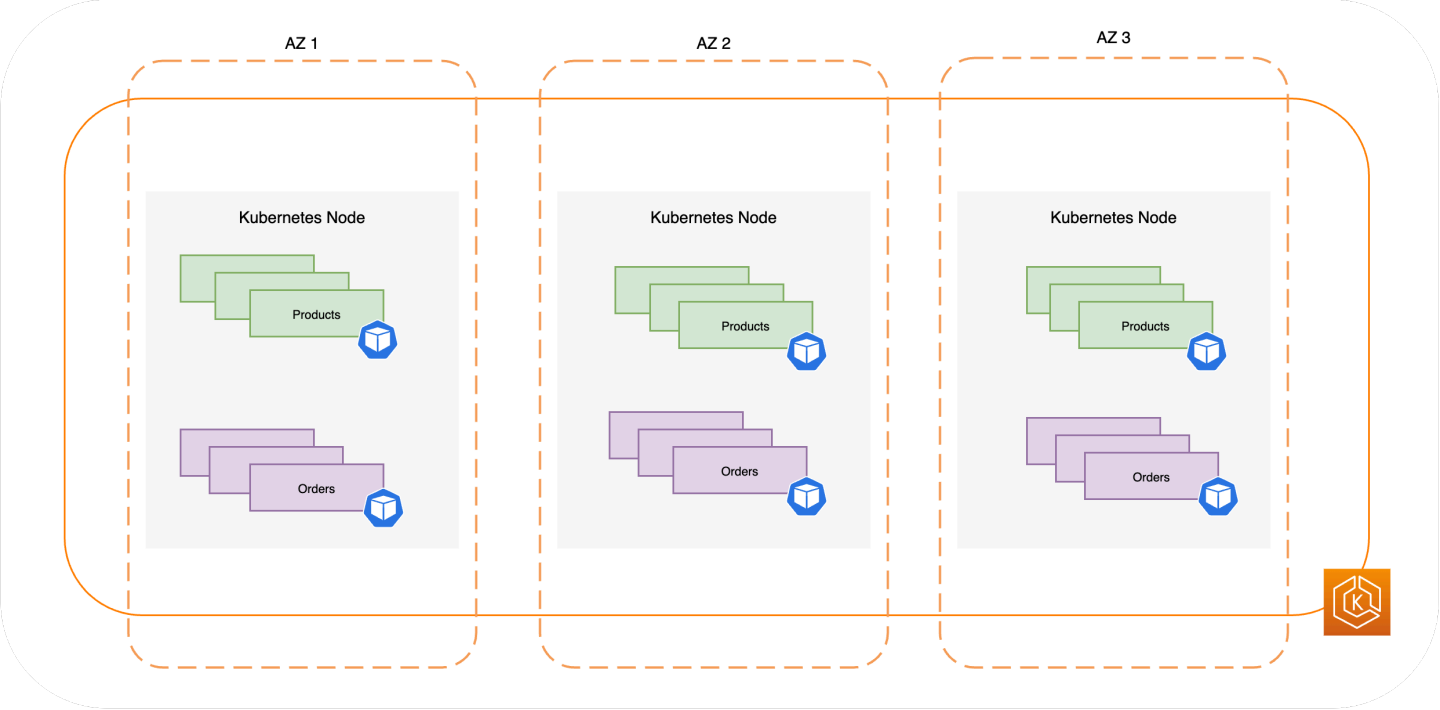
- 跨多个可用区配置集群的 Worker 节点
- 预调配足够的计算容量，以适应单个可用区的移除
- 在每个可用区预先扩展容器组（pod）（包括 CoreDNS）
- 将多个容器组（pod）副本分布在所有可用区中，帮助确保从单个可用区转出后，仍会给您留下足够的容量
- 将相互依赖或相关的容器组（pod）托管在同一可用区中
- 通过手动启动从可用区的可用区转移，测试集群环境是否能在没有可用区的情况下按预期运行。或者，您可以启用可用区自动转移并在自动转移练习运行时回复。要在 EKS 中进行可用区转移，手动测试或进行可用区转移并非强制要求，但强烈建议执行此操作。

跨多个可用区预调配您的 EKS Worker 节点

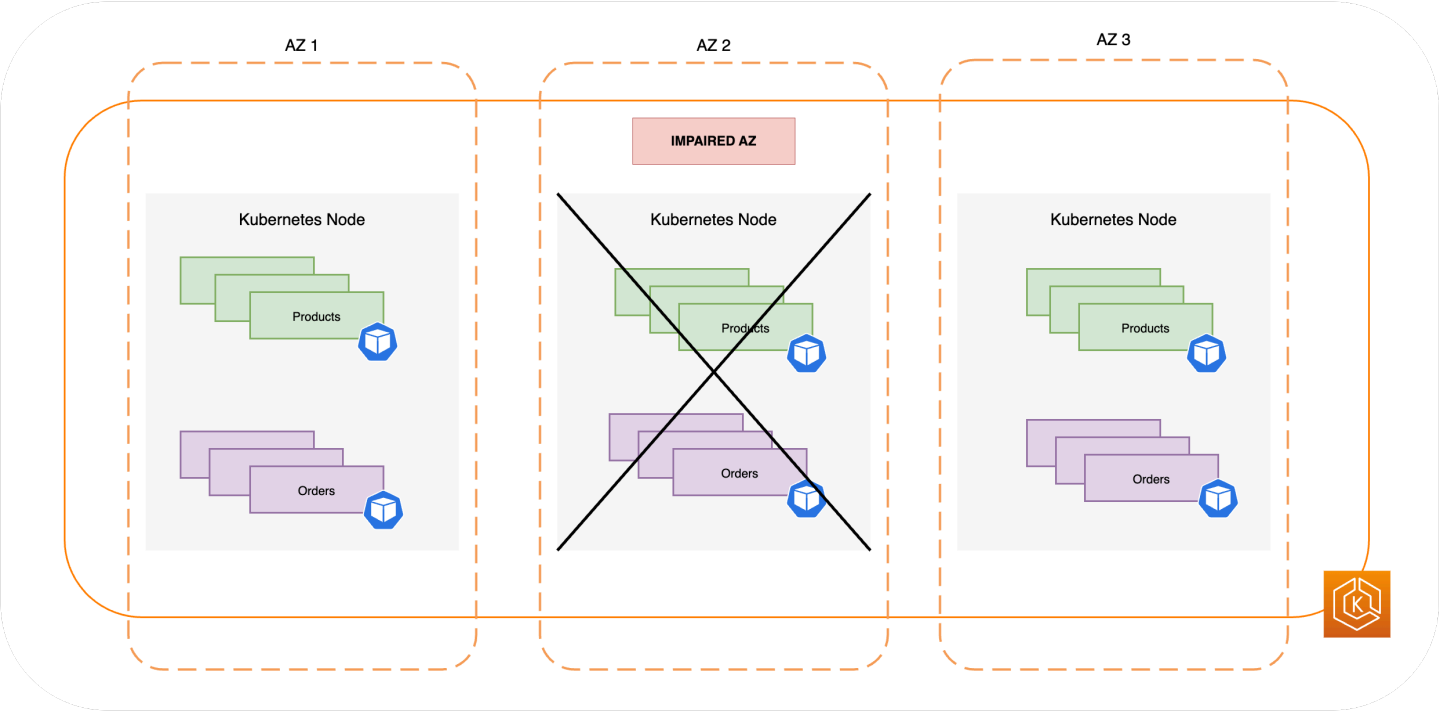
Amazon 区域有多个独立的地点，其物理数据中心称为可用区（AZ)。可用区在物理上相互隔离，以避免同时影响整个区域的冲击。在预调配 EKS 集群时，建议将 Worker 节点部署到一个区域中的多个可用区。这有助于使您的集群环境更能抵御单个可用区的损坏，并让您能够保持在其它可用区中运行的应用程序的高可用性（HA）。当您开始从受影响的可用区启动可用区转移时，EKS 环境的集群内网络将自动更新为仅使用运行状况良好的可用区，以帮助保持集群的高可用性。

确保为 EKS 环境设置多可用区将提高系统的整体可靠性。但是，多可用区环境会影响应用程序数据的传输和处理方式，这反过来又会影响您环境的网络费用。特别是，频繁的跨可用区出站流量（流量分布在可用区之间）可能会对您的网络相关成本产生重大影响。您可以应用不同的策略来控制 EKS 集群中容器组（pod）之间的跨可用区流量并降低相关成本。有关在运行高可用性 EKS 环境时如何优化网络成本的更多信息，请参阅这些最佳实践。

下图说明了具有三个运行状况良好的可用区的高可用性 EKS 环境。



下图说明了具有三个可用区的 EKS 环境如何抵御可用区损坏，并且由于剩余两个可用区运行状况良好，因此仍保持高可用性。



预调配足够的计算容量，以适应单个可用区的移除

为了优化 EKS 数据面板中计算基础设施的资源利用率和成本，最佳做法是使计算容量与工作负载要求保持一致。但是，如果您的所有 Worker 节点都已满负荷，则在调度新的容器组（pod）之前，您只能依赖将新的 Worker 节点添加到 EKS 数据面板中。在运行关键工作负载时，通常最好使用冗余容量在线运行，以应对负载突然增加、节点运行状况问题等意外情况。如果您计划使用可用区转移，则要计划在出现损坏时移除整个可用区的容量。这意味着您必须调整冗余计算容量，使其足以处理负载，即使其中一个可用区处于离线状态也不例外。

扩展计算资源时，向 EKS 数据面板添加新节点的过程需要一些时间。这可能会对应用程序的实时性能和可用性产生影响，尤其是在出现可用区损坏的情况下。您的 EKS 环境应能够承受丢失一个可用区所带来的负载，而不会导致最终用户或客户体验降级。这意味着要最大限度地减少或消除需要新容器组（pod）的时间与实际调度到 Worker 节点上的时间之间的延迟。

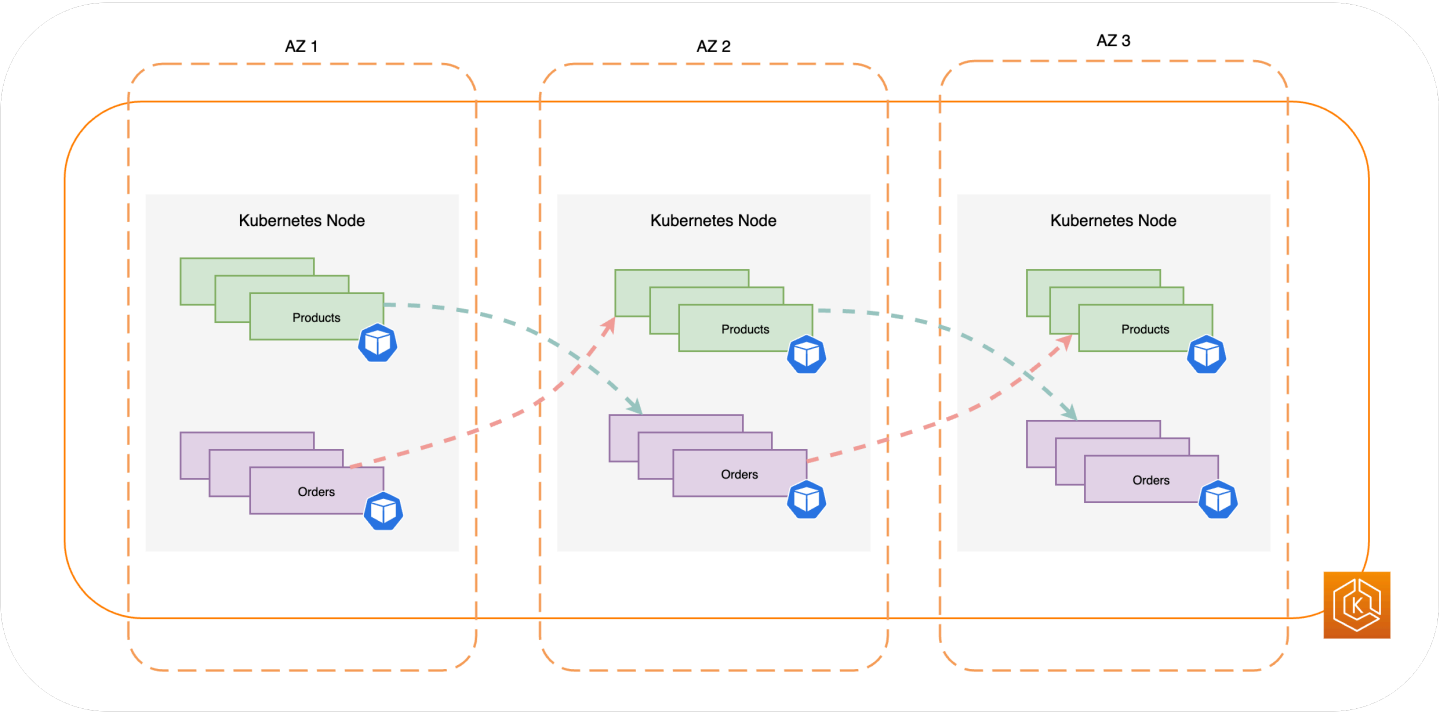
此外，如果出现可用区损坏，您应旨在降低遇到计算容量限制的风险，这将阻止在运行状况良好的可用区中将新需要的节点添加到您的 EKS 数据面板中。

为降低这些潜在负面影响的风险，建议您在每个可用区的某些 Worker 节点中过度预调配计算容量。通过执行此操作，Kubernetes 调度器有预先存在的容量可用于放置新容器组（pod），当您在环境中丢失其中一个可用区时，这一点尤为重要。

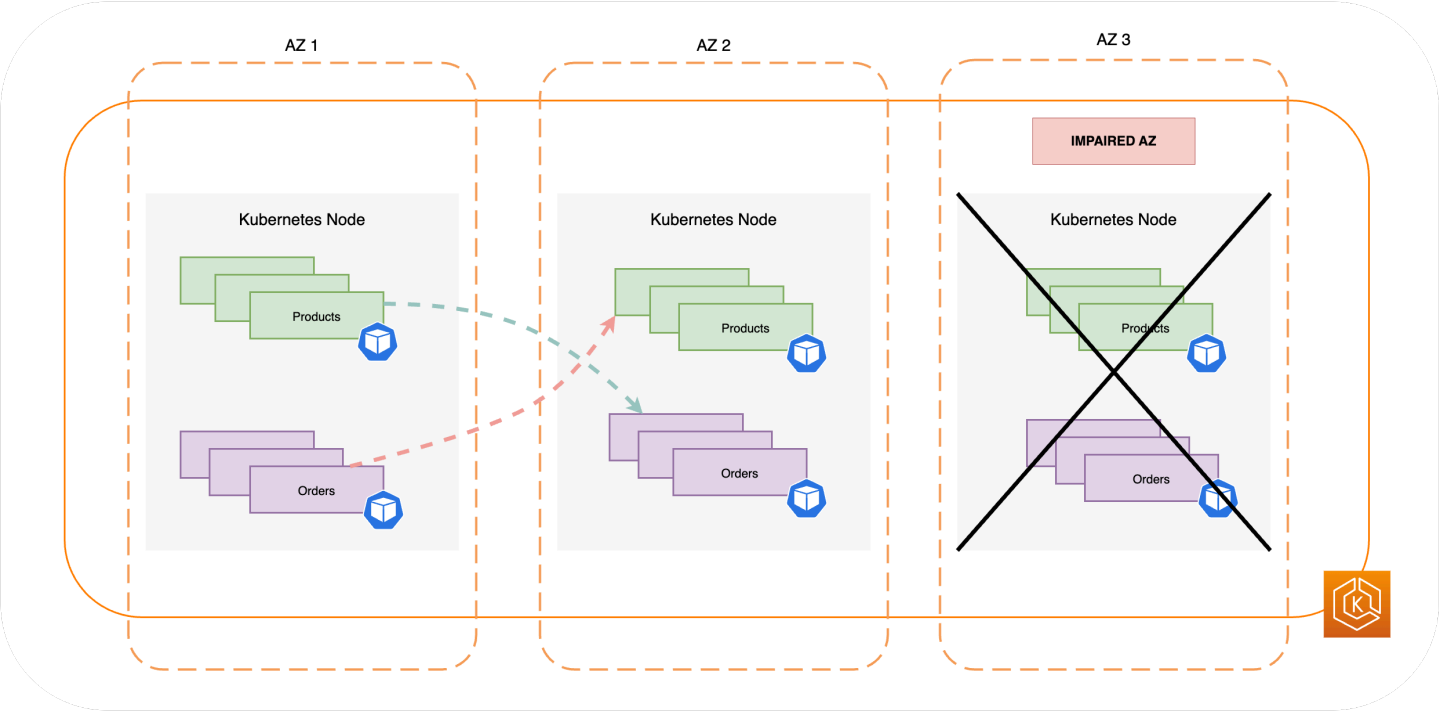
跨可用区运行和分布多个容器组（pod）副本

Kubernetes 允许您通过运行单个应用程序的多个实例（容器组（pod）副本）来预先扩展工作负载。为一个应用程序运行多个容器组（pod）副本可以消除单点故障，并通过减少单个副本的资源紧张来提高其整体性能。但是，为了使应用程序具有高可用性和更好的容错能力，建议在不同的故障域（也称为拓扑域）中运行应用程序的多个副本并将它们分散到不同的故障域。这种情况下的故障域是可用区。通过使用拓扑分布约束，您可以将应用程序设置为具有预先存在的静态稳定性。然后，当出现可用区损坏情况时，您的环境将在运行状态良好的可用区中具有足够的副本，可立即应对任何流量峰值或激增。

下图展示了当所有可用区都运行状况良好时，具有东-西流量流向的 EKS 环境。



下图展示了具有东-西流量流向的 EKS 环境，其中单个可用区出现故障，且您已启动可用区转移。



以下代码片段是如何在 Kubernetes 中使用多个副本设置工作负载的示例。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders
spec:
  replicas: 9
  selector:
    matchLabels:
      app:orders
  template:
    metadata:
      labels:
        app: orders
        tier: backend
    spec:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: "topology.kubernetes.io/zone"
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            app: orders
```

最重要的是，您应该运行 DNS 服务器软件（CoreDNS/kube-dns）的多个副本，如果默认情况下尚未配置这些副本，则应用类似的拓扑分布约束。这有助于确保在出现单个可用区损坏时，您在运行状况良好的可用区中有足够的 DNS 容器组（pod），以便继续处理集群中其他通信容器组（pod）的服务发现请求。CoreDNS EKS 附加组件具有 CoreDNS 容器组（pod）默认设置，可确保如果多个可用区中有节点可用，则 CoreDNS 容器组（pod）将分布在集群的可用区域中。您也可以根据需要将这些默认设置替换为自己的自定义配置。

使用 Helm 安装 CoreDNS 时，您可以更新 values.yaml 文件中的 `replicaCount`，以确保每个可用区中有足够的副本。此外，为确保这些副本分布在集群环境中的不同可用区中，请确保更新同一个 `values.yaml` 文件中的 `topologySpreadConstraints` 属性。以下代码片段说明了如何配置 CoreDNS 来执行此操作。

CoreDNS Helm values.yaml

```
replicaCount: 6
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

如果出现可用区损坏，您可以使用 CoreDNS 的自动缩放系统来承受 CoreDNS 容器组（pod）上增加的负载。您需要的 DNS 实例数取决于集群中运行的工作负载数量。CoreDNS 受 CPU 约束，允许它使用 Horizontal Pod Autoscaler（HPA）基于 CPU 进行扩展。以下是您可以修改以满足需求的示例。

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: coredns
  namespace: default
spec:
  maxReplicas: 20
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: coredns
  targetCPUUtilizationPercentage: 50
```

另外，EKS 可以在 CoreDNS 的 EKS 附加组件版本中管理 CoreDNS 部署的自动扩缩。此 CoreDNS 自动扩缩器会持续监控集群状态，包括节点数量和 CPU 核心数量。控制器会根据这些信息，动态调整 EKS 集群中 CoreDNS 部署的副本数量。

要在 CoreDNS EKS 附加组件中启用自动扩缩配置，请使用以下配置设置：

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

您也可以使用 NodeLocal DNS 或集群比例自动扩缩器来扩展 CoreDNS。有关更多信息，请参阅水平扩展 CoreDNS。

将相互依赖的容器组（pod）托管在同一个可用区中

通常，应用程序有不同的工作负载，这些工作负载需要相互通信才能成功完成端到端流程。如果这些不同的应用程序分布在不同的可用区中，但托管在同一个可用区中，则单个可用区损坏可能会影响端到端流程。例如，如果应用程序 A 在可用区 1 和可用区 2 中有多个副本，但应用程序 B 的所有副本都在可用区 3 中，则可用区 3 丢失将影响两个工作负载（应用程序 A 和应用程序 B）之间的端到端流程。如果您将拓扑分布约束与容器组（pod）亲和性相结合，则可以通过将容器组（pod）分布到所有可用区来增强应用程序的韧性。此外，这还配置了某些容器组（pod）之间的关系，确保它们已托管。

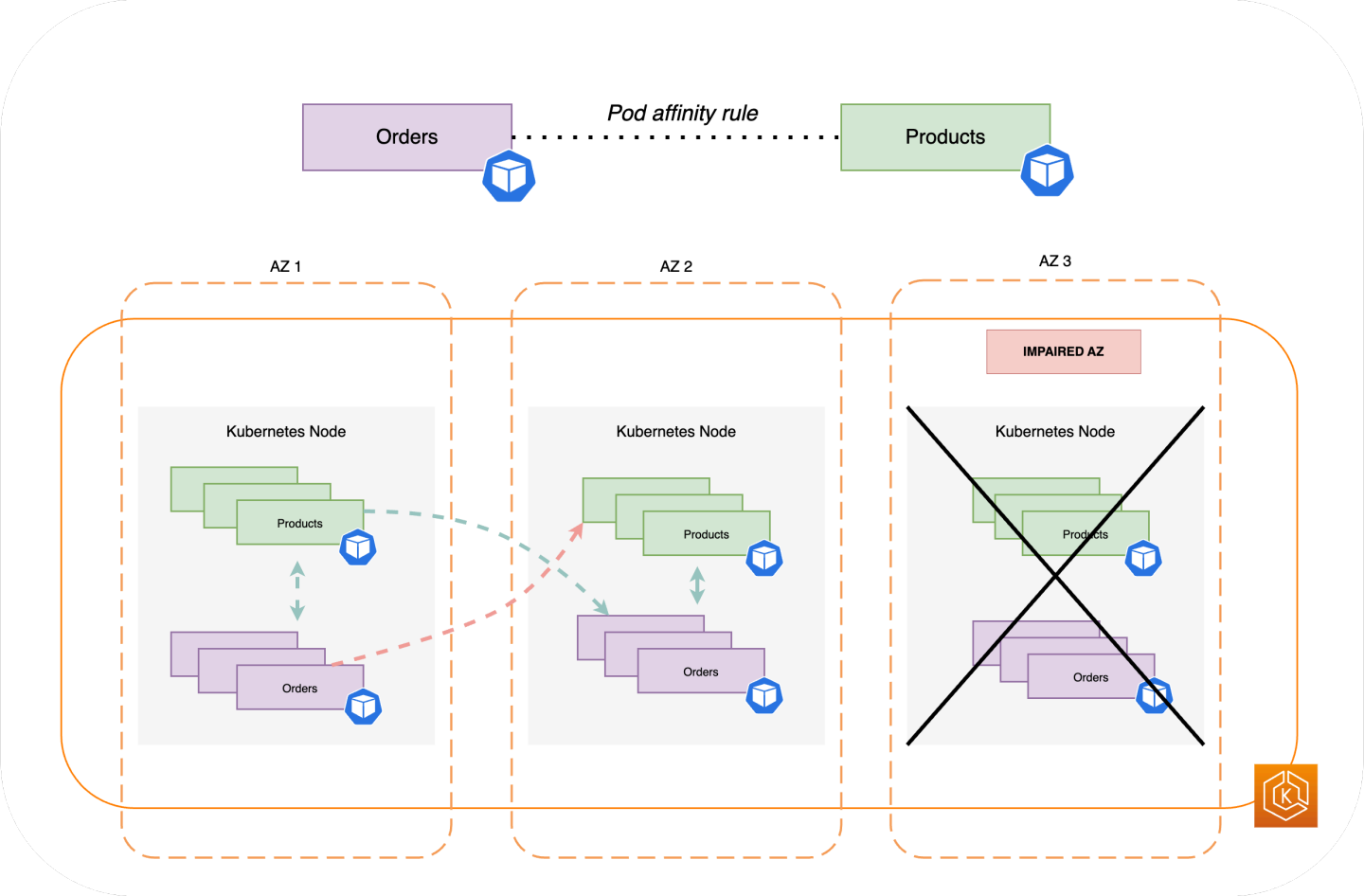
借助容器组（pod）亲和性规则，您可以定义工作负载之间的关系以影响 Kubernetes 调度器的行为，使其将容器组（pod）托管在同一个 Worker 节点或同一个可用区中。您还可以配置调度约束的严格程度。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products
  namespace: ecommerce
  labels:
    app.kubernetes.io/version: "0.1.6"

    spec:
      serviceAccountName: graphql-service-account
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
```

```
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - orders
          topologyKey: "kubernetes.io/hostname"
```

下图展示了使用容器组（pod）亲和性规则托管在同一节点上的多个容器组（pod）。



测试您的集群环境能否应对可用区的丢失

完成之前章节所述的要求后，下一个步骤是测试您是否有足够的计算和工作负载容量来应对可用区的丢失。您可以通过在 EKS 中手动启动可用区转移来实现此目的。或者，您可以启用可用区自动转移并配置练习运行，这种运行还可测试在集群环境中少一个可用区的情况下，您的应用程序是否按预期运行。

常见问题

为什么应该使用此功能？

通过在 EKS 集群中使用 ARC 可用区转移或可用区自动转移，您可以自动执行将集群内网络流量从损坏的可用区转移出去的快速恢复过程，从而更好地维护 Kubernetes 应用程序的可用性。使用 ARC，您可以避免漫长而复杂的步骤，这些步骤会导致在可用区损坏事件期间延长恢复期。

此功能如何与其它 Amazon 服务结合使用？

EKS 与 ARC 集成，后者为您提供了在 Amazon 中完成恢复操作所需的主要接口。为了确保集群内流量从损坏的可用区适当路由，EKS 对在 Kubernetes 数据面板中运行的容器组（pod）的网络端点列表进行了修改。如果您使用弹性负载均衡将外部流量路由到集群，则可以向 ARC 注册您的负载均衡器，并对其启动可用区转移，以防止流量流入降级的可用区。可用区转移还适用于由 EKS 托管节点组创建的 Amazon EC2 Auto Scaling 组。为了防止损坏的可用区被用于新的 Kubernetes 容器组（pod）或节点启动，EKS 会将损坏的可用区从自动扩缩组中移除。

此功能与默认的 Kubernetes 保护有何不同？

此功能与多个 Kubernetes 内置保护结合使用，可帮助客户应用程序保持韧性。您可以配置容器组（pod）就绪性和存活性探测器，以决定容器组（pod）何时应接入流量。当这些探测器失败时，Kubernetes 会移除将这些容器组（pod）作为服务的目标，流量将不再发送到这些容器组（pod）。虽然这很有用，但对于客户来说，配置这些运行状况检查以确保在可用区降级时它们会失败并非易事。ARC 可用区转移功能提供了额外的安全网，可以帮助您在 Kubernetes 的原生保护不足时完全隔离已降级的可用区。可用区转移还为用户提供了一种测试架构的运行就绪性和韧性的简便方法。

Amazon 可以代表我启动可用区转移吗？

可以。如果您想以一种全自动的方式使用 ARC 可用区转移，则可以启用 ARC 可用区自动转移。借助可用区自动转移，您可以依靠 Amazon 来监控 EKS 集群的可用区运行状况，并在检测到可用区损坏时自动启动可用区转移。

如果我使用此功能，但我的 Worker 节点和工作负载未预先扩展，会发生什么情况？

如果您没有预先扩展，并且依赖于在可用区转移期间预调配其他节点或容器组（pod），则可能会面临恢复延迟的风险。向 Kubernetes 数据面板添加新节点的过程将需要一些时间，这可能会对应用程序的实时性能和可用性产生影响，尤其是在出现可用区损坏的情况下。此外，如果出现可用区损坏，您可能会遇到潜在的计算容量约束，这会阻止将新需要的节点添加到运行状况良好的可用区。

如果您的工作负载未预先扩展并分布在集群中的所有可用区，则可用区损坏可能会影响仅在受影响可用区的 Worker 节点上运行的应用程序的可用性。为了降低应用程序可用性完全中断的风险，如果工作负载的所有端点都位于运行状况不佳的可用区中，EKS 提供了故障安全功能，可以将流量发送到损坏的可用区中的容器组（pod）端点。但是，强烈建议您预先扩展应用程序并将其分散到所有可用区，以便在出现可用区问题时保持可用性。

如果我运行的是有状态的应用程序会怎样？

如果您运行的是有状态的应用程序，则须根据使用案例和架构评测其容错能力。如果您采用主动/备用架构或模式，则可能存在主动服务器位于损坏的可用区中的情况。在应用程序级别，如果未激活备用副本，则您的应用程序可能会遇到问题。在运行状况良好的可用区中启动新的 Kubernetes 容器组（pod）时，您也可能会遇到问题，因为它们将无法附加到损坏的可用区的持久卷。

此功能适用于 Karpenter 吗？

在 EKS 中，ARC 可用区转移和可用区自动转移目前不支持 Karpenter。如果可用区损坏，则可以通过移除运行状况不佳的可用区来调整相关的 Karpenter NodePool 配置，这样新的 Worker 节点只能在其他可用区中启动。

此功能是否可以与 EKS Fargate 结合使用？

此功能不能与 EKS Fargate 结合使用。默认情况下，当 EKS Fargate 识别出可用区运行状况值事件时，容器组（pod）将更愿意在其它可用区中运行。

EKS 托管的 Kubernetes 控制面板是否会受到影响？

不会。默认情况下，Amazon EKS 跨多个可用区运行和扩展 Kubernetes 控制面板以确保高可用性。ARC 可用区转移和可用区自动转移只能在 Kubernetes 数据面板上起作用。

这项新功能有什么相关费用吗？

您可以在 EKS 集群中使用 ARC 可用区转移和可用区自动转移，无需额外付费。但是，您将继续为预调配实例付费，因此强烈建议您在使用此功能之前预先扩展 Kubernetes 数据面板。您应该考虑在成本和应用程序可用性之间取得平衡。

其他资源

- 可用区转移的工作原理
- ARC 中的可用区转移最佳实践
- 可用区转移和可用区自动转移支持的资源和场景
- 在 Amazon EKS 上运行弹性工作负载
- 通过容器组（pod）优先级和超额预置消除 Kubernetes 节点扩展延迟
- 扩展 CoreDNS 容器以获得高 DNS 流量

启用 EKS 可用区转移以避开损坏的可用区

Amazon 应用程序恢复控制器（ARC）可帮助您管理和协调跨可用区（AZ）的应用程序恢复，并可与包括 Amazon EKS 在内的许多服务结合使用。借助对 ARC 可用区转移的 EKS 支持，您可以将集群内的网络流量从损坏的可用区转移出去。您还可以授权 Amazon 监控可用区的运行状况，并代表您暂时将网络流量从运行状况不佳的可用区转移出去。

如何使用 EKS 可用区转移：

1. 使用 Amazon 应用程序恢复控制器（ARC）启用您的 EKS 集群。此操作通过使用 Amazon EKS 控制台、Amazon CLI、CloudFormation 或 eksctl 在集群级别完成。
2. 启用后，您可以使用 ARC 控制台、Amazon CLI 或可用区转移和可用区自动转移 API 来管理可用区转移或可用区自动转移。

请注意，向 ARC 注册 EKS 集群后，您仍然需要配置 ARC。例如，您可以使用 ARC 控制台来配置可用区自动转移。

有关 EKS 可用区转移的工作原理以及如何设计工作负载以处理损坏的可用区的更多详细信息，请参阅the section called “了解可用区转移”。

注意事项

- EKS 自动模式不支持 Amazon 应用程序恢复控制器、可用区转移和可用区自动转移。
- 我们建议在两次可用区转移操作之间至少等待 60 秒，以确保正确处理每个请求。

  尝试快速连续执行可用区转移（彼此相隔在 60 秒以内）时，Amazon EKS 服务可能无法正确处理所有转移请求。这是由于当前的轮询机制会更新集群的可用区状态。如果您需要执行多次可用区转移，则请确保两次操作之间有足够的时间让系统处理每次变更。

什么是 Amazon 应用程序恢复控制器？

Amazon 应用程序恢复控制器（ARC）可帮助您为在 Amazon 上运行的应用程序做好恢复准备并更快地完成恢复。可用区转移通过暂时将支持的资源的流量从可用区（AZ）转移到 Amazon 区域中的运行状况良好的可用区，从而可帮助您从可用区损坏中快速恢复。

了解有关 Amazon 应用程序恢复控制器（ARC）的更多信息

什么是可用区转移？

可用区转移是 ARC 中的一项功能，它允许您将资源（例如 EKS 集群或 Elastic Load Balancer）的流量从某个 Amazon 区域的可用区移走，从而快速缓解问题并快速恢复应用程序。由于部署不当导致延迟问题或者由于可用区损坏等原因，您可能会选择转移流量。可用区转移不需要高级配置步骤。

详细了解 ARC 可用区转移

什么是可用区自动转移？

可用区自动转移是 ARC 中的一项功能，您可以启用该功能，以授权 Amazon 代表您将流量从受支持资源的可用区转移到 Amazon 区域中运行状况良好的可用区。当内部遥测功能指示区域中的可用区存在可能会影响客户的损坏时，Amazon 会开始自动转移。内部遥测纳入了多个来源的指标，包括 Amazon 网络、Amazon EC2 和 Elastic Load Balancing 服务。

当指示器显示不再存在问题或潜在问题时，Amazon 将结束自动转移。

详细了解 ARC 区域自动转移

EKS 在自动转移期间会做什么？

EKS 更新网络配置以避免将流量定向到损坏的可用区。此外，如果您使用的是托管节点组，EKS 将仅在可用区转移期间在运行状况良好的可用区中启动新节点。当转移到期或被取消时，联网配置将恢复为包括先前检测为运行状况不佳的可用区。

详细了解 EKS 可用区转移。

向 Amazon 应用程序恢复控制器（ARC）（Amazon 控制台）注册 EKS 集群

1. 找到您要向 ARC 注册的 EKS 集群的名称和区域。
2. 导航到该区域的 EKS 控制台，然后选择您的集群。
3. 在集群信息页面上，选择概述选项卡。
4. 在可用区转移标题下，选择管理按钮。
5. 为 EKS 可用区转移选择启用或禁用。

现在，您的 EKS 集群已向 ARC 注册。

如果您希望 Amazon 检测并避开损坏的可用区，则需要配置 ARC 可用区自动转移。例如，您可以在 ARC 控制台中执行此操作。

后续步骤

- 了解如何启用可用区自动转移。
- 了解如何手动启动可用区转移。

了解 Amazon EKS 中的访问控制工作原理

了解如何管理对您的 Amazon EKS 集群的访问权限。使用 Amazon EKS 需要了解 Kubernetes 和 Amazon Identity and Access Management（Amazon IAM）如何处理访问控制。

本部分包括：

the section called “Kubernetes API 访问权限” – 了解如何允许应用程序或用户对 Kubernetes API 进行身份验证。您可以使用访问条目、aws-auth ConfigMap 或外部 OIDC 提供商。

the section called “访问集群资源” – 了解如何将 Amazon Web Services 管理控制台 配置为与 Amazon EKS 集群进行通信。使用控制台查看集群中的 Kubernetes 资源，例如命名空间、节点和容器组（pod）。

the section called “使用 kubectl 访问集群” – 了解如何将 kubectl 配置为与 Amazon EKS 集群进行通信。请使用 Amazon CLI 创建 kubeconfig 文件。

the section called “Amazon 工作负载访问权限 ” – 了解如何将 Kubernetes 服务账户与 Amazon IAM 角色关联。您可以将容器组身份或 IAM 角色用于服务账户的（IRSA）。

常见任务

- 向开发人员授予对 Kubernetes API 的访问权限。在 Amazon Web Services 管理控制台中查看 Kubernetes 资源。
  - 解决方案：使用访问条目将 Kubernetes RBAC 权限与 Amazon IAM 用户或角色相关联。
- 将 kubectl 配置为使用 Amazon 凭证与 Amazon EKS 集群通信。
  - 解决方案使用 Amazon CLI 创建 kubeconfig 文件。
- 使用外部身份提供者（例如 Ping Identity）对 Kubernetes API 进行用户身份验证。
  - 解决方案：链接外部 OIDC 提供商。
- 授予 Kubernetes 集群上的工作负载调用 Amazon API 的能力。
  - 解决方案：使用容器组身份将 Amazon IAM 角色与 Kubernetes 服务账户关联。

背景

- 了解 Kubernetes 服务账户的工作原理。
- 查看 Kubernetes 基于角色的访问控制（RBAC）模型

- 有关管理对 Amazon 资源的访问权限的更多信息，请参阅《Amazon IAM 用户指南》https://docs.amazonaws.cn/IAM/latest/UserGuide/intro-structure.html。或者，请参加有关使用 Amazon IAM 的免费入门培训。

EKS 自动模式注意事项

EKS 自动模式与 EKS 容器组身份和 EKS 访问条目集成。

- EKS 自动模式使用访问条目向 EKS 控制面板授予 Kubernetes 权限。例如，访问策略允许 EKS 自动模式读取有关网络端点和服务的信息。
  - 您不能在 EKS 自动模式集群上禁用访问条目。
  - 您可以选择启用 `aws-auth ConfigMap`。
  - EKS 自动模式的访问条目是自动配置的。您可以查看这些访问条目，但不能对其进行修改。
  - 如果您使用节点类创建自定义节点 IAM 角色，则需要使用 AmazonEKSAutoNodePolicy 访问策略为该角色创建访问条目。
- 要向工作负载授予 Amazon 服务的权限，请使用 EKS 容器组身份。
  - 您无需在 EKS 自动模式集群上安装容器组身份代理。

向 IAM 用户和角色授予对 Kubernetes API 的访问权限

集群有一个 Kubernetes API 端点。Kubectl 使用此 API。您可以使用两种类型的身份对此 API 进行身份验证：

- Amazon Identity and Access Management（IAM）主体（角色或用户）– 此类型需要对 IAM 进行身份验证。用户可以使用通过身份源提供的凭证以 IAM 用户或联合身份登录到 Amazon。如果您的管理员以前使用 IAM 角色设置了身份联合验证，则用户只能使用联合身份登录。当用户使用联合身份验证访问 Amazon 时，他们就是在间接分派角色。当用户使用此类身份时，您：
  - 可以为用户分配 Kubernetes 权限，以便他们使用集群上的 Kubernetes 对象。有关如何为 IAM 主体分配权限以便他们能够访问集群上的 Kubernetes 对象的更多信息，请参阅the section called “访问条目”。
  - 可以为他们分配 IAM 权限，以便其使用 Amazon EKS API、Amazon CLI、Amazon、Amazon Web Services 管理控制台 CloudFormation 或 `eksctl` 使用您的 Amazon EKS 集群及其资源。有关更多信息，请参阅《服务授权参考》中的 Amazon Elastic Kubernetes Service 定义的操作。
  - 节点通过分派 IAM 角色加入您的集群。使用 IAM 主体访问集群的能力由适用于 Kubernetes 的 Amazon IAM 身份验证器提供，该工具在 Amazon EKS 控制面板上运行。

- 您自己的 OpenID Connect（OIDC）提供者中的用户 – 此类型需要对您的 OIDC 提供者进行身份验证。有关使用 Amazon EKS 集群设置自己的 OIDC 提供者的更多信息，请参阅the section called “关联 OIDC 提供者”。当用户使用此类身份时，您：
  - 可以为用户分配 Kubernetes 权限，以便他们使用集群上的 Kubernetes 对象。
  - 无法向他们分配 IAM 权限，使他们能够使用 Amazon EKS API、Amazon CLI、Amazon CloudFormation、Amazon Web Services 管理控制台或 `eksctl` 使用您的 Amazon EKS 集群及其资源。

您可以在集群中使用这两种类型的身份。IAM 身份验证方法无法禁用。OIDC 身份验证方法是可选的。

将 IAM 身份与 Kubernetes 权限相关联

适用于 Kubernetes 的 Amazon IAM 身份验证器安装在集群的控制面板上。该工具使您允许的 Amazon Identity and Access Management（IAM）主体（角色和用户）能够访问集群上的 Kubernetes 资源。您可以使用以下某一种方法，允许 IAM 主体访问集群上的 Kubernetes 对象：

- 创建访问条目 – 如果集群的 Kubernetes 版本等于或高于先决条件部分所列的平台版本，建议使用此选项。

  使用访问条目管理集群外的 IAM 主体的 Kubernetes 权限。您可以使用 EKS API、Amazon 命令行界面、Amazon SDK、Amazon CloudFormation 和 Amazon Web Services 管理控制台来添加和管理对集群的访问权限。这意味着您可以使用与创建集群相同的工具来管理用户。

  首先将身份验证模式改为使用访问条目，然后将现有 aws-auth ConfigMap 条目迁移到访问条目。
- 向 **`aws-auth ConfigMap`** 中添加条目 – 如果集群的平台版本早于先决条件部分中列出的版本，则必须使用此选项。对于集群的 Kubernetes 版本，如果集群的平台版本等于或高于先决条件部分所列的平台版本，并且您已向 `ConfigMap` 添加了条目，建议将这些条目迁移到访问条目。但是，您无法迁移 Amazon EKS 添加到 `ConfigMap` 的条目，例如与托管节点组结合使用的 IAM 角色的条目或 Fargate 配置文件。有关更多信息，请参阅 the section called “Kubernetes API 访问权限”。
  - 如果必须使用 `aws-auth ConfigMap` 选项，则可以使用 `eksctl create iamidentitymapping` 命令向 `ConfigMap` 中添加条目。有关更多信息，请参阅 `eksctl` 文档中的管理 IAM 用户和角色。

设置集群身份验证模式

每个集群都有一种身份验证模式。身份验证模式决定了您可以使用哪些方法来允许 IAM 主体访问集群上的 Kubernetes 对象。有三种身份验证模式。

> **Important**
>
> 一旦启用了访问条目方法，就无法将其禁用。
> 如果在创建集群时未启用 ConfigMap 方法，则以后无法启用它。在引入访问条目之前创建的所有集群都启用了 ConfigMap 方法。
> 如果您在集群中使用混合节点，则必须使用 API 或 API_AND_CONFIG_MAP 集群身份验证模式。

集群内部的 aws-auth ConfigMap

这是 Amazon EKS 集群的原始身份验证模式。创建集群的 IAM 主体是可以使用 kubectl 访问集群的初始用户。初始用户必须将其他用户添加到 aws-auth ConfigMap 中的列表中，并分配影响集群内其他用户的权限。这些其它用户无法管理或移除初始用户，因为 ConfigMap 中没有要管理的条目。

ConfigMap 和访问条目

使用这种身份验证模式，您可以使用这两种方法向集群添加 IAM 主体。请注意，每种方法都存储单独的条目；例如，如果您从 Amazon CLI 中添加访问条目，aws-auth ConfigMap 则不会更新。

仅访问条目

使用这种身份验证模式，您可以使用 EKS API、Amazon 命令行界面、Amazon SDK、Amazon CloudFormation 和 Amazon Web Services 管理控制台来管理 IAM 主体对集群的访问权限。

每个访问条目都有一种类型，您可以使用访问范围将主体限制在特定的命名空间和访问策略的组合来设置预先配置且可重复使用的权限策略。您也可以使用 STANDARD 类型和 Kubernetes RBAC 组来分配自定义权限。

身份验证模式	方法
仅 ConfigMap （CONFIG_MAP ）	aws-auth ConfigMap
EKS API 和 ConfigMap （API_AND_CONFIG_MAP ）	访问 EKS API、Amazon 命令行界面、Amazon SDK、Amazon CloudFormation Amazon Web Services 管理控制台 和 aws-auth ConfigMap 中的条目

身份验证模式	方法
仅 EKS API (API)	访问 EKS API、Amazon 命令行界面、Amazon SDK、Amazon CloudFormation 和 Amazon Web Services 管理控制台中的条目

> **Note**
>
> Amazon EKS 自动模式需要使用访问条目。

使用 EKS 访问条目向 IAM 用户授予 Kubernetes 访问权限

本节旨在向您展示如何使用访问条目和策略管理 IAM 主体对 Amazon Elastic Kubernetes Service（EKS）中 Kubernetes 集群的访问权限。您将找到有关更改身份验证模式；从旧版 aws-auth ConfigMap 条目迁移；创建、更新和删除访问条目；将策略与条目关联；查看预定义策略权限，以及安全访问管理的关键先决条件和注意事项的详细信息。

概览

EKS 访问条目是向用户授予 Kubernetes API 访问权限的最佳方式。例如，您可以使用访问条目向开发人员授予使用 kubectl 的权限。从根本上讲，EKS 访问条目会将一组 Kubernetes 权限与某个 IAM 身份（例如 IAM 角色）关联起来。例如，开发人员可以担任某个 IAM 角色并使用该角色对 EKS 集群进行身份验证。

特征

- 集中式身份验证和授权：直接通过 Amazon EKS API 控制对 Kubernetes 集群的访问，无需在 Amazon 和 Kubernetes API 之间切换即可获得用户权限。
- 精细权限管理：使用访问条目和策略为 Amazon IAM 主体定义精细权限，包括修改或撤销创建者的集群管理员访问权限。
- IaC 工具集成：支持 Amazon CloudFormation、Terraform 和 Amazon CDK 等基础设施即代码工具，以便在集群创建期间定义访问配置。
- 错误配置恢复：允许通过 Amazon EKS API 恢复集群访问权限，而无需直接访问 Kubernetes API。
- 降低开销并增强安全性：集中运营以降低开销，同时利用 Amazon IAM 功能（如 CloudTrail 审计日志和多重身份验证）提升安全性。

如何附加权限

您可以通过两种方式将 Kubernetes 权限附加到访问条目：

- 使用访问策略。访问策略是由 Amazon 维护的预定义 Kubernetes 权限模板。有关更多信息，请参阅 the section called “检查访问策略”。
- 引用 Kubernetes 组。如果您将某个 IAM 身份与某个 Kubernetes 组关联，则可以创建向该组授予权限 Kubernetes 资源。有关更多信息，请参阅 Kubernetes 文档中的使用 RBAC 授权。

注意事项

在现有集群上启用 EKS 访问条目时，请注意以下事项：

- 旧版集群行为：对于在引入访问条目之前创建的集群（初始平台版本早于平台版本要求中指定的版本），EKS 会自动创建一个反映先前已存在权限的访问条目。此条目包括最初创建集群的 IAM 身份以及在创建集群期间授予该身份的管理权限。
- 处理旧版 **`aws-auth`** ConfigMap：如果您的集群依赖旧版 `aws-auth` ConfigMap 进行访问管理，则启用访问条目后，仅自动创建原始集群创建者的访问条目。添加到 ConfigMap 的其他角色或权限（例如，开发人员或服务的自定义 IAM 角色）不会自动迁移。要解决此问题，请手动创建相应的访问条目。

开始使用

1. 确定您要使用的 IAM 身份和访问策略。
   - the section called “检查访问策略”
2. 在集群上启用 EKS 访问条目。确认您有受支持的平台版本。
   - the section called “身份验证模式”
3. 创建一个访问条目，将某个 IAM 身份与 Kubernetes 权限关联起来。
   - the section called “创建访问条目”
4. 使用该 IAM 身份向集群进行身份验证。
   - the section called “设置 Amazon CLI”
   - the section called “设置 kubectl 和 eksctl”

将访问策略与访问条目关联起来

您可以为类型为 STANDARD 的访问条目分配一个或多个访问策略。Amazon EKS 会自动向其他类型的访问条目授予在您的集群中正常运行所需的权限。Amazon EKS 访问策略包含 Kubernetes 权限，但不包含 IAM 权限。在将访问策略关联到访问条目之前，请确保您熟悉每个访问策略中包含的 Kubernetes 权限。有关更多信息，请参阅 the section called “检查访问策略”。如果所有访问策略都不符合您的要求，则不要将访问策略与访问条目相关联。而是为访问条目指定一个或多个组名称，然后创建和管理 Kubernetes 基于角色的访问控制对象。有关更多信息，请参阅 the section called “创建访问条目”。

- 现有的访问条目。要创建该文件，请参阅 the section called “创建访问条目”。
- 具有以下权限的 Amazon Identity and Access Management 角色或用户：`ListAccessEntries`、`DescribeAccessEntry`、`UpdateAccessEntry`、`ListAccessPolicie` 和 `DisassociateAccessPolicy`。有关更多信息，请参阅《服务授权参考》中的 Actions defined by Amazon Elastic Kubernetes Service。

在将访问策略与访问条目关联之前，请考虑以下要求：

- 您可以将多个访问策略关联到每个访问条目，但只能将每个策略与一个访问条目关联一次。如果您关联多个访问策略，则访问条目的 IAM 主体拥有所有关联访问策略中包含的所有权限。
- 您可以将访问策略的范围限定为集群上的所有资源，也可以指定一个或多个 Kubernetes 命名空间的名称。您可以为命名空间名称使用通配符。例如，如果要将访问策略的范围限定为以 `dev-` 开头的所有命名空间，则可以指定 `dev-*` 为命名空间名称。请确保您的集群上存在命名空间，并且拼写与集群上的实际命名空间名称相匹配。Amazon EKS 不会确认您的集群上命名空间的拼写或是否存在。
- 将访问策略与访问条目关联后，您可以更改访问策略的访问范围。如果已将访问策略的范围限定为 Kubernetes 命名空间，则可根据需要为关联添加和移除命名空间。
- 如果您将访问策略关联到同时指定了组名称的访问条目，则 IAM 主体拥有所有关联访问策略中的所有权限。该主体还拥有在指定组名称的任何 Kubernetes `Role` 和 `RoleBinding` 对象中指定的任何 Kubernetes `Role` 或 `ClusterRole` 对象中的所有权限。
- 如果运行 `kubectl auth can-i --list` 命令，则不会看到访问策略分配的任何 Kubernetes 权限，这些权限与您在运行命令时使用的 IAM 主体的访问条目相关联。只有已经在 Kubernetes `Role` 或 `ClusterRole` 对象中授予 Kubernetes 权限，并且已经将这些权限绑定到为访问条目指定的组名称或用户名时，该命令才会显示这些权限。
- 如果您在与集群上的 Kubernetes 对象交互时伪装成 Kubernetes 用户或组，例如使用带 `--as` *`username`* 或 `--as-group` *`group-name`* 的 `kubectl` 命令，则表示您强制使用 Kubernetes RBAC 授权。因此，IAM 主体没有与访问条目关联的任何访问策略所分配的权限。IAM 主体伪装的用户或组拥有的唯一 Kubernetes 权限是您在 Kubernetes `Role` 或 `ClusterRole` 对象中向

其授予的 Kubernetes 权限，这些权限已绑定到组名称或用户名。要使 IAM 主体拥有相关访问策略中的权限，请不要伪装成 Kubernetes 用户或组。IAM 主体仍将拥有您在 Kubernetes `Role` 或 `ClusterRole` 对象中向其授予的任何权限，这些权限已绑定到为访问条目指定的组名称或用户名。有关更多信息，请参阅 Kubernetes 文档中的用户伪装。

您可以使用 Amazon Web Services 管理控制台 或 Amazon CLI 将访问策略与访问条目相关联。

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择具有要与访问策略关联的访问条目的集群的名称。
3. 选择访问选项卡。
4. 如果访问条目的类型为标准，则可以关联或取消关联 Amazon EKS 访问策略。如果您的访问条目类型不是标准，则此选项不可用。
5. 选择关联访问策略。
6. 对于策略名称，选择具有您希望 IAM 主体拥有的权限的策略。要查看每个策略包含的权限，请参阅 the section called “检查访问策略”。
7. 对于访问范围，选择一个访问范围。如果选择集群，则访问策略中的权限将授予 IAM 主体以访问所有 Kubernetes 命名空间中的资源。如果选择 Kubernetes 命名空间，则可选择添加新命名空间。可在显示的命名空间字段中，输入集群上 Kubernetes 命名空间的名称。如果您希望 IAM 主体拥有跨多个命名空间的权限，则可以输入多个命名空间。
8. 选择添加访问策略。

Amazon CLI

1. 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
2. 查看可用的访问策略。

```
aws eks list-access-policies --output table
```

示例输出如下。

```
-------------------------------------------------------------------------------------------------
|                                          ListAccessPolicies
                  |
+-----------------------------------------------------------------------------------------------
+
||                                            accessPolicies
                 ||
|+-------------------------------------------------------------------
+-------------------------------+|
||                                arn                                |
 name              ||
|+-------------------------------------------------------------------
+-------------------------------+|
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSAdminPolicy        |
 AmazonEKSAdminPolicy         ||
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy |
 AmazonEKSClusterAdminPolicy  ||
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSEditPolicy         |
 AmazonEKSEditPolicy          ||
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSViewPolicy         |
 AmazonEKSViewPolicy          ||
|+-------------------------------------------------------------------
+-------------------------------+|
```

要查看每个策略包含的权限，请参阅 the section called “检查访问策略”。

3. 查看您现有的访问条目。将 *my-cluster* 替换为您的集群的名称。

```
aws eks list-access-entries --cluster-name my-cluster
```

示例输出如下。

```
{
    "accessEntries": [
        "arn:aws-cn:iam::111122223333:role/my-role",
        "arn:aws-cn:iam::111122223333:user/my-user"
    ]
}
```

4. 将访问策略与访问条目关联。以下示例将 AmazonEKSViewPolicy 访问策略与访问条目关联。每当 *my-role* IAM 角色尝试访问集群上的 Kubernetes 对象时，Amazon EKS 都将授权该角色使用策略中的权限仅访问 *my-namespace1* 和 *my-namespace2* Kubernetes 命名空间中的 Kubernetes 对象。将 *my-cluster* 替换为集群的名称，将 *111122223333* 替换为 Amazon 账户 ID，将 *my-role* 替换为您希望 Amazon EKS 授权其访问 Kubernetes 集群对象的 IAM 角色的名称。

```
aws eks associate-access-policy --cluster-name my-cluster --principal-arn arn:aws-
cn:iam::111122223333:role/my-role \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
 arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSViewPolicy
```

如果您希望 IAM 主体拥有整个集群的权限，请将 type=namespace,namespaces=*my-namespace1*,*my-namespace2* 替换为 type=cluster。如果要将多个访问策略关联到访问条目，请多次运行该命令，每个访问条目都有唯一的访问策略。每个关联的访问策略都有自己的范围。

> **Note**
>
> 如果以后要更改关联访问策略的范围，请使用新的范围再次运行前面的命令。例如，如果您想移除 *my-namespace2*，则只能使用 type=namespace,namespaces=*my-namespace1* 再次运行该命令。如果要将范围从 namespace 更改为 cluster，则可以使用 type=cluster 再次运行该命令并移除 type=namespace,namespaces=*my-namespace1*,*my-namespace2* 。

5. 确定哪些访问策略与访问条目关联。

```
aws eks list-associated-access-policies --cluster-name my-cluster --principal-arn
 arn:aws-cn:iam::111122223333:role/my-role
```

示例输出如下。

```
{
    "clusterName": "my-cluster",
    "principalArn": "arn:aws-cn:iam::111122223333",
    "associatedAccessPolicies": [
        {
            "policyArn": "arn:aws-cn:eks::aws:cluster-access-policy/
AmazonEKSViewPolicy",
            "accessScope": {
```

```
                "type": "cluster",
                "namespaces": []
            },
            "associatedAt": "2023-04-17T15:25:21.675000-04:00",
            "modifiedAt": "2023-04-17T15:25:21.675000-04:00"
        },
        {
            "policyArn": "arn:aws-cn:eks::aws:cluster-access-policy/
AmazonEKSAdminPolicy",
            "accessScope": {
                "type": "namespace",
                "namespaces": [
                    "my-namespace1",
                    "my-namespace2"
                ]
            },
            "associatedAt": "2023-04-17T15:02:06.511000-04:00",
            "modifiedAt": "2023-04-17T15:02:06.511000-04:00"
        }
    ]
}
```

在前面的示例中，此访问条目的 IAM 主体拥有集群上所有命名空间的查看权限，以及对两个 Kubernetes 命名空间的管理员权限。

6. 取消访问策略与访问条目的关联。在此示例中，AmazonEKSAdminPolicy 策略与访问条目取消关联。但是，IAM 主体保留 AmazonEKSViewPolicy 访问策略中对 *my-namespace1* 和 *my-namespace2* 命名空间中对象的权限，因为该访问策略并未与访问条目取消关联。

```
aws eks disassociate-access-policy --cluster-name my-cluster --principal-arn arn:aws-
cn:iam::111122223333:role/my-role \
    --policy-arn arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy
```

要列出可用的访问策略，请参阅the section called “检查访问策略”。

将现有 aws-auth ConfigMap 条目迁移至访问条目

如果您已向集群的 aws-auth ConfigMap 中添加了条目，我们建议您为 aws-auth ConfigMap 中的现有条目创建访问条目。创建访问条目后，您可以从 ConfigMap 中移除这些条目。不能将访问策略关联到 aws-auth ConfigMap 中的条目。如果您想将访问策略与您的 IAM 主体相关联，请创建访问条目。

> **Important**
>
> - 当集群处于 API_AND_CONFIGMAP 身份验证模式并且在 aws-auth ConfigMap 和访问条目中都存在同一 IAM 角色的映射时，该角色将使用访问条目的映射进行身份验证。对于同一 IAM 主体，访问条目优先于 ConfigMap 条目。
> - 在移除 Amazon EKS 为托管节点组创建的现有 aws-auth ConfigMap 条目或集群的 Fargate 配置文件之前，请仔细检查您的 Amazon EKS 集群中是否存在这些特定资源的正确访问条目。如果您移除 Amazon EKS 在 ConfigMap 中创建的条目而没有等效的访问条目，您的集群将无法正常运行。

先决条件

- 熟悉访问条目和访问策略。有关更多信息，请参阅the section called “访问条目”和the section called “关联访问策略”。
- 一个平台版本等于或高于the section called “访问条目”主题中“先决条件”部分所列版本的现有版本。
- 您的设备或 Amazon CloudShell 上安装 0.215.0 版或更高版本的 eksctl 命令行工具。要安装或更新 eksctl，请参阅 eksctl 文档中的 Installation。
- 用于修改 kube-system 命名空间中的 aws-auth ConfigMap 的 Kubernetes 权限。
- 具有以下权限的 Amazon Identity and Access Management 角色或用户：CreateAccessEntry 和 ListAccessEntries。有关更多信息，请参阅《服务授权参考》中的 Amazon Elastic Kubernetes Service 定义的操作。

eksctl

1. 查看您的 aws-auth ConfigMap 中的现有条目。将 *my-cluster* 替换为您的集群的名称。

```
eksctl get iamidentitymapping --cluster my-cluster
```

示例输出如下。

```
ARN
            USERNAME                                GROUPS
                        ACCOUNT
arn:aws-cn:iam::111122223333:role/EKS-my-cluster-Admins
                Admins                                  system:masters
```

```
arn:aws-cn:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers
              my-namespace-Viewers                     Viewers
arn:aws-cn:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1
                              system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
arn:aws-cn:iam::111122223333:user/my-user
              my-user
arn:aws-cn:iam::111122223333:role/EKS-my-cluster-fargateprofile1
                              system:node:{{SessionName}}
 system:bootstrappers,system:nodes,system:node-proxier
arn:aws-cn:iam::111122223333:role/EKS-my-cluster-managed-ng
                              system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
```

2. 为您创建的并在上一个输出中返回的任何 ConfigMap 条目the section called “创建访问条目”。创建访问条目时，请确保为输出中返回的 ARN、USERNAME、GROUPS 和 ACCOUNT 指定相同的值。在此示例输出中，您将为除最后两个条目外的其他所有条目创建访问条目，因为最后两个条目是由 Amazon EKS 为 Fargate 配置文件和托管节点组创建的。
3. 从 ConfigMap 中删除您创建的任何访问条目对应的条目。如果您不从 ConfigMap 中删除条目，则 IAM 主体 ARN 的访问条目设置将覆盖 ConfigMap 条目。将 *111122223333* 替换为您的 Amazon 账户 ID，将 *EKS-my-cluster-my-namespace-Viewers* 替换为您的 ConfigMap 中相应条目的角色名称。如果您要移除的条目是用于 IAM 用户而不是 IAM 角色的，请将 role 替换为 user，将 *EKS-my-cluster-my-namespace-Viewers* 替换为用户名。

```
eksctl delete iamidentitymapping --arn arn:aws-cn:iam::111122223333:role/EKS-my-
cluster-my-namespace-Viewers --cluster my-cluster
```

检查访问策略权限

访问策略包括包含 Kubernetes verbs（权限）和 resources 的 rules。访问策略不包括 IAM 权限或资源。访问策略与 Kubernetes Role 和 ClusterRole 对象类似，仅包括 allow rules。您无法修改访问策略的内容。您无法创建自己的访问策略。如果访问策略中的权限不符合需求，请创建 Kubernetes RBAC 对象并为访问条目指定组名称。有关更多信息，请参阅 the section called “创建访问条目”。访问策略中包含的权限与 Kubernetes 面向用户的集群角色中的权限类似。有关更多信息，请参阅 Kubernetes 文档中的面向用户的角色。

列出所有策略

使用此页面上列出的任何一个访问策略，或者使用 Amazon CLI 检索所有可用访问策略的列表：

```
aws eks list-access-policies
```

预期输出应如下所示（为简洁起见已缩写）：

```
{
    "accessPolicies": [
        {
            "name": "AmazonAIOpsAssistantPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonAIOpsAssistantPolicy"
        },
        {
            "name": "AmazonARCRegionSwitchScalingPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/
AmazonARCRegionSwitchScalingPolicy"
        },
        {
            "name": "AmazonEKSAdminPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"
        },
        {
            "name": "AmazonEKSAdminViewPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminViewPolicy"
        },
        {
            "name": "AmazonEKSAutoNodePolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy"
        }
        // Additional policies omitted
    ]
}
```

AmazonEKSAdminPolicy

此访问策略包括授予 IAM 主体对资源的大部分权限的权限。当与访问条目关联时，其访问范围通常是一个或多个 Kubernetes 命名空间。如果您希望 IAM 主体拥有集群上所有资源的管理员访问权限，请改为将 the section called “AmazonEKSClusterAdminPolicy” 访问策略关联到您的访问条目。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy`

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicasets , replicasets/scale, statefulsets , statefulsets/scale	create, delete, deletecollection , patch, update
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
authorization.k8s.io	localsubjectaccessreviews	create
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
batch	cronjobs, jobs	create, delete, deletecollection , patch, update

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicasets , replicasets/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
rbac.authorization.k8s.io	rolebindings , roles	create, delete, deletecollection , get, list, patch, update, watch
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get,list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
	pods, pods/attach , pods/exec , pods/port forward , pods/proxy	create, delete, deletecollection , patch, update
	serviceaccounts	impersonate
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get,list, watch

AmazonEKSClusterAdminPolicy

此访问策略包括授予 IAM 主体对集群的管理员访问权限的权限。当与访问条目关联时，其访问范围通常是集群，而不是 Kubernetes 命名空间。如果您希望 IAM 主体的管理范围更有限，请考虑将 the section called “AmazonEKSAdminPolicy” 访问策略与您的访问条目相关联。

ARN – arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy

Kubernetes API 组	Kubernetes 非资源 URL	Kubernetes 资源	Kubernetes 动词（权限）
*		*	*
	*		*

AmazonEKSAdminViewPolicy

此访问策略包括授予 IAM 主体列出/查看集群中所有资源的权限。请注意，此项包括 Kubernetes 密钥。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSAdminViewPolicy`

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
*	*	get, list, watch

AmazonEKSEditPolicy

此访问策略包括允许 IAM 主体编辑大多数 Kubernetes 资源的权限。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSEditPolicy`

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
apps	daemonsets，deployments，deployments/rollback，deployments/scale，replicasets，replicasets/scale, statefulsets，statefulsets/scale	create, delete, deletecollection，patch, update
apps	controllerrevisions，daemonsets，daemonsets/status，deployments，deployments/scale，deployments/status，replicasets，replicasets/scale，replicasets/status，statefulsets，statefulsets/scale，statefulsets/status	get, list, watch

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
autoscaling	horizontalpodautoscalers ，horizontalpodautoscalers/status	get, list, watch
autoscaling	horizontalpodautoscalers	create, delete, deletecollection ，patch, update
batch	cronjobs, jobs	create, delete, deletecollection ，patch, update
batch	cronjobs, cronjobs/status ，jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets ，deployments ，deployments/rollback ，deployments/scale ，ingresses ，networkpolicies ，replicasets ，replicasets/scale ，replicationcontrollers/scale	create, delete, deletecollection ，patch, update

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
	namespaces	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	serviceaccounts	impersonate

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
	pods, pods/attach , pods/exec , pods/port forward , pods/proxy	create, delete, deletecollection , patch, update
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch

AmazonEKSViewPolicy

此访问策略包括允许 IAM 主体查看大多数 Kubernetes 资源的权限。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSViewPolicy`

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale ,	get, list, watch

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
	replicasets/status ，replicationcontrollers/scale	
networking.k8s.io	ingresses ，ingresses/status ，networkpolicies	get, list, watch
policy	poddisruptionbudgets ，poddisruptionbudgets/status	get, list, watch
	configmaps ，endpoints ，persistentvolumeclaims ，persistentvolumeclaims/status ，pods, replicationcontrollers ，replicationcontrollers/scale ，serviceaccounts ，services, services/status	get, list, watch
	bindings、events、limitranges 、namespaces/status、pods/log、pods/status 、replicationcontrollers/status 、resourcequotas 、resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSSecretReaderPolicy

此访问策略包括允许 IAM 主体读取 Kubernetes 密钥的权限。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSSecretReaderPolicy`

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
	`secrets`	`get`, `list`, `watch`

AmazonEKSAutoNodePolicy

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy`

此策略包含允许 Amazon EKS 组件完成以下任务的下列权限：

- `kube-proxy` – 监控网络端点和服务，并管理相关事件。这将启用集群范围的网络代理功能。
- `ipamd` – 管理 Amazon VPC 联网资源和容器网络接口（CNI）。这将允许 IP 地址管理进程守护程序处理容器组联网。
- `coredns` – 访问端点和服务等服务发现资源。这将在集群内启用 DNS 解析。
- `ebs-csi-driver` – 使用 Amazon EBS 卷的存储相关资源。这将允许动态预置和挂载持久性卷。
- `neuron` – 监控 Amazon Neuron设备的节点和容器组。这将有助于 Amazon Inferentia 和 Trainium 加速器的管理。
- `node-monitoring-agent` – 访问节点诊断和事件。这将启用集群运行状况监控和诊断收集。

每个组件都使用一个专用的服务账户，并且仅限于其特定功能所需的权限。

如果您在 NodeClass 中手动指定节点 IAM 角色，则需要创建一个访问条目，将新的节点 IAM 角色与该访问策略关联。

AmazonEKSBlockStoragePolicy

> **Note**
>
> 此策略仅适用于 Amazon 服务相关角色，不能与客户自行管理的角色一起使用。

ARN – arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSBlockStoragePolicy

此策略包含允许 Amazon EKS 管理用于存储操作的领导副本选择和协调资源的权限：

- coordination.k8s.io – 创建和管理领导副本选择的租赁对象。这将使 EKS 存储组件能够通过领导副本选择机制来协调其在集群中的活动。

此策略的范围限于 EKS 存储组件使用的特定租赁资源，以防止对集群中其他协调资源的访问权限冲突。

启用自动模式后，Amazon EKS 会自动使用此访问策略为集群 IAM 角色创建访问条目，从而确保具备必要的权限来支持块存储功能的正常运行。

AmazonEKSLoadBalancingPolicy

ARN – arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingPolicy

此策略包含允许 Amazon EKS 管理用于负载均衡的领导副本选择的权限：

- coordination.k8s.io – 创建和管理领导副本选择的租赁对象。这让 EKS 负载均衡组件能够通过领导副本选择来协调多个副本之间的活动。

此策略的范围特别限于负载均衡租赁资源，以确保适当的协调，同时防止访问集群中的其他租赁资源。

启用自动模式后，Amazon EKS 会自动使用此访问策略为集群 IAM 角色创建访问条目，从而确保具备必要的权限来支持联网功能的正常运行。

AmazonEKSNetworkingPolicy

ARN – arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSNetworkingPolicy

此策略包含允许 Amazon EKS 管理用于联网的领导副本选择的权限：

- coordination.k8s.io – 创建和管理领导副本选择的租赁对象。这让 EKS 网络组件能够通过领导副本选择来协调 IP 地址分配活动。

此策略的范围特别限于联网租赁资源，以确保适当的协调，同时防止访问集群中的其他租赁资源。

启用自动模式后，Amazon EKS 会自动使用此访问策略为集群 IAM 角色创建访问条目，从而确保具备必要的权限来支持联网功能的正常运行。

AmazonEKSComputePolicy

> **Note**
> 此策略仅适用于 Amazon 服务相关角色，不能与客户自行管理的角色一起使用。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSComputePolicy`

此策略包含允许 Amazon EKS 管理用于计算操作的领导副本选择资源的权限：

- `coordination.k8s.io` – 创建和管理领导副本选择的租赁对象。这让 EKS 计算组件能够通过领导副本选择来协调节点扩缩活动。

此策略的范围特别限于计算管理租赁资源，同时允许对集群中所有租赁资源的基本读取访问权限（`get`、`watch`）。

启用自动模式后，Amazon EKS 会自动使用此访问策略为集群 IAM 角色创建访问条目，从而确保具备必要的权限来支持联网功能的正常运行。

AmazonEKSBlockStorageClusterPolicy

> **Note**
> 此策略仅适用于 Amazon 服务相关角色，不能与客户自行管理的角色一起使用。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSBlockStorageClusterPolicy`

此策略将授予 Amazon EKS 自动模式块存储功能的必要权限，有助于对 Amazon EKS 集群中块存储资源的高效管理。此策略包含以下权限：

CSI 驱动程序管理：

- 创建、读取、更新和删除 CSI 驱动程序，尤其是针对块存储。

卷管理：

- 列出、观察、创建、更新、修补和删除持久性卷。
- 列出、观察和更新持久性卷声明。
- 修补持久性卷声明状态。

节点和容器组交互：

- 读取节点和容器组信息。
- 管理与存储操作相关的事件。

存储类和属性：

- 读取存储类和 CSI 节点。
- 读取卷属性类。

卷挂载：

- 列出、观察和修改卷挂载及其状态。

快照操作：

- 管理卷快照、快照内容和快照类别。
- 处理卷组快照和相关资源的操作。

此策略适用于在自动模式下运行的 Amazon EKS 集群，旨在支持全面的块存储管理。此策略将各种操作的权限进行组合，包括块存储卷的预置、挂载、大小调整和快照创建等。

启用自动模式后，Amazon EKS 会自动使用此访问策略为集群 IAM 角色创建访问条目，从而确保具备必要的权限来支持块存储功能的正常运行。

AmazonEKSComputeClusterPolicy

> **Note**
>
> 此策略仅适用于 Amazon 服务相关角色，不能与客户自行管理的角色一起使用。

ARN – arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSComputeClusterPolicy

此策略将授予 Amazon EKS 自动模式计算管理功能的必要权限，有助于在 Amazon EKS 集群中实现高效的计算资源编排和扩缩。此策略包含以下权限：

节点管理：

- 创建、读取、更新、删除和管理节点池和节点声明的状态。
- 管理节点类，包括创建、修改和删除。

调度和资源管理：

- 对容器组、节点、持久性卷、持久性卷声明、复制控制器和命名空间的读取权限。
- 对存储类、CSI 节点和卷挂载的读取权限。
- 列出和观察部署、进程守护程序集、副本集和有状态集。
- 读取容器组中断预算。

事件处理：

- 创建、读取和管理集群事件。

节点取消预置和容器组弹出：

- 更新、修补和删除节点。
- 创建容器组弹出并在必要时删除容器组。

自定义资源定义（CRD）管理：

- 创建新建 CRD。
- 管理与节点管理相关的特定 CRD（节点类、节点池、节点声明和节点诊断）。

此策略适用于在自动模式下运行的 Amazon EKS 集群，旨在支持全面的计算管理。此策略将各种操作的权限进行组合，包括节点预置、调度、扩缩和资源优化。

启用自动模式后，Amazon EKS 会自动使用此访问策略为集群 IAM 角色创建访问条目，从而确保具备必要的权限来支持计算管理功能的正常运行。

AmazonEKSLoadBalancingClusterPolicy

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingClusterPolicy`

此策略将授予 Amazon EKS 自动模式负载均衡功能的必要权限，有助于对 Amazon EKS 集群中负载均衡资源的高效管理和配置。此策略包含以下权限：

事件和资源管理：

- 创建和修补事件。
- 对容器组、节点、端点和命名空间的读取权限。
- 更新容器组状态。

服务和 Ingress 管理：

- 全面管理服务及其状态。
- 全面控制 Ingress 及其状态。
- 对端点切片和 Ingress 类的读取权限。

目标组绑定：

- 创建和修改目标组绑定及其状态。
- 对 Ingress 类参数的读取权限。

自定义资源定义（CRD）管理：

- 创建并读取所有 CRD。
- 对 targetgroupbindings.eks.amazonaws.com 和 ingressclassparams.eks.amazonaws.com CRD 的具体管理。

Webhook 配置：

- 创建和读取变异和验证性 Webhook 配置。
- 管理 eks-load-balancing-webhook 配置。

此策略适用于在自动模式下运行的 Amazon EKS 集群，旨在支持全面的负载均衡管理。此策略将各种操作的权限进行组合，包括服务公开、Ingress 路由以及与 Amazon 负载均衡服务的集成。

启用自动模式后，Amazon EKS 会自动使用此访问策略为集群 IAM 角色创建访问条目，从而确保具备必要的权限来支持负载均衡功能的正常运行。

AmazonEKSNetworkingClusterPolicy

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSNetworkingClusterPolicy`

AmazonEKSNetworkingClusterPolicy

此策略将授予 Amazon EKS 自动模式联网功能的必要权限，有助于对 Amazon EKS 集群中联网资源的高效管理和配置。此策略包含以下权限：

节点和容器组管理：

- 对节点类及其状态的读取权限。
- 对节点声明及其状态的读取权限。
- 对容器组的读取权限。

CNI 节点管理：

- CNINodes 及其状态的权限，包括创建、读取、更新、删除和修补。

自定义资源定义（CRD）管理：

- 创建并读取所有 CRD。
- cninodes.eks.amazonaws.com CRD 的具体管理（更新、修补、删除）。

事件管理：

- 创建和修补事件。

此策略适用于在自动模式下运行的 Amazon EKS 集群，旨在支持全面的联网管理。此策略将各种操作的权限进行组合，包括节点联网配置、CNI（容器网络接口）管理和相关的自定义资源处理。

此策略允许联网组件与节点相关资源进行交互，管理特定于 CNI 的节点配置，以及处理对集群中联网操作至关重要的自定义资源。

启用自动模式后，Amazon EKS 会自动使用此访问策略为集群 IAM 角色创建访问条目，从而确保具备必要的权限来支持联网功能的正常运行。

AmazonEKSHybridPolicy

> **Note**
>
> 此策略仅适用于 Amazon 服务相关角色，不能与客户自行管理的角色一起使用。

此访问策略包含授予 EKS 访问集群节点的权限。当与访问条目关联时，其访问范围通常是集群，而不是 Kubernetes 命名空间。此策略由 Amazon EKS 混合节点功能使用。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSHybridPolicy`

Kubernetes API 组	Kubernetes 非资源 URL	Kubernetes 资源	Kubernetes 动词（权限）
*		nodes	list

AmazonEKSClusterInsightsPolicy

> **Note**
>
> 此策略仅适用于 Amazon 服务相关角色，不能与客户自行管理的角色一起使用。

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSClusterInsightsPolicy`

此策略授予 Amazon EKS 集群见解功能的只读权限。此策略包含以下权限：

节点访问权限：– 列出并查看集群节点 – 读取节点状态信息

DaemonSet 访问权限：– 读取 kube-proxy 配置的访问权限

此策略由 EKS 集群见解服务自动管理。有关更多信息，请参阅 the section called “集群见解”。

AWSBackupFullAccessPolicyForBackup

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AWSBackupFullAccessPolicyForBackup`

AWSBackupFullAccessPolicyForBackup

此策略授予 Amazon Backup 管理和创建 EKS 集群备份所需的权限。该策略包含以下权限：

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
*	*	`list`, `get`

AWSBackupFullAccessPolicyForRestore

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AWSBackupFullAccessPolicyForRestore`

AWSBackupFullAccessPolicyForRestore

此策略授予 Amazon Backup 管理和还原 EKS 集群备份所需的权限。该策略包含以下权限：

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
*	*	`list`, `get`, `create`

AmazonEKSACKPolicy

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSACKPolicy`

此策略授予 Amazon Controllers for Kubernetes（ACK）功能所需的权限，以管理来自 Kubernetes 的 Amazon 资源。此策略包含以下权限：

ACK 自定义资源管理：

- 对 50 多项 Amazon 服务的所有 ACK 服务自定义资源的完全访问权限，包括 S3、RDS、DynamoDB、Lambda、EC2 等。
- 创建、读取、更新和删除 ACK 自定义资源定义。

命名空间访问权限：

- 读取命名空间，用于整理资源。

主节点选举：

- 创建和读取协调租约，用于主节点选举。
- 更新和删除特定的 ACK 服务控制器租约。

事件管理：

- 创建和修补事件，用于 ACK 操作。

此策略旨在通过 Kubernetes API 支持全面的 Amazon 资源管理。创建 ACK 功能时，Amazon EKS 会自动使用此访问策略为所提供的功能 IAM 角色创建访问条目。

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
	`namespaces`	`get`, `watch`, `list`
`services.k8s.aws` , `acm.services.k8s.aws` , `acmpca.services.k8s.aws` , `apigateway.services.k8s.aws` , `apigatewayv2.services.k8s.aws` , `applicationautoscaling.services.k8s.aws` , `athena.services.k8s.aws` , `bedrock.services.k8s.aws` , `bedrockagent.services.k8s.aws` , `bedrockagentcorecontrol.services.k8s.aws` ,	*	*

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
cloudfront.services.k8s.aws ，cloudtrail.services.k8s.aws ，cloudwatch.services.k8s.aws ，cloudwatchlogs.services.k8s.aws ，codeartifact.services.k8s.aws ，cognitoidentityprovider.services.k8s.aws ，documentdb.services.k8s.aws ，dynamodb.services.k8s.aws ，ec2.services.k8s.aws ，ecr.services.k8s.aws ，ecrpublic.services.k8s.aws ，ecs.services.k8s.aws ，efs.services.k8s.aws ，eks.services.k8s.aws ，elasticache.services.k8s.aws ，elbv2.services.k8s.aws ，emrcontainers.services.k8s.aws ，eventbridge.services.k8s.aws ，iam.services.k8s.aws ，kafka.services.k8s.aws ，keyspaces.services.k8s.aws ，kinesis.services.k8s.aws ，kms.servi		

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
ces.k8s.aws ，lambda.services.k8s.aws ，memorydb.services.k8s.aws ，mq.services.k8s.aws ，networkfirewall.services.k8s.aws ，opensearchservice.services.k8s.aws ，organizations.services.k8s.aws ，pipes.services.k8s.aws ，prometheusservice.services.k8s.aws ，ram.services.k8s.aws ，rds.services.k8s.aws ，recyclebin.services.k8s.aws ，route53.services.k8s.aws ，route53resolver.services.k8s.aws ，s3.services.k8s.aws ，s3control.services.k8s.aws ，sagemaker.services.k8s.aws ，secretsmanager.services.k8s.aws ，ses.services.k8s.aws ，sfn.services.k8s.aws ，sns.services.k8s.aws ，sqs.services.k8s.aws ，ssm.services.k8s.aws ，		

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
wafv2.services.k8s.aws		
coordination.k8s.io	leases	create, get, list, watch
coordination.k8s.io	leases（仅限特定的 ACK 服务控制器租约）	delete, update, patch
	events	create, patch
apiextensions.k8s.io	customresourcedefinitions	*

AmazonEKSArgoCDClusterPolicy

ARN – arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSArgoCDClusterPolicy

此策略授予 Argo CD 功能发现资源和管理集群范围对象所需的集群级权限。此策略包含以下权限：

命名空间管理：

- 创建、读取、更新和删除命名空间，用于管理应用程序命名空间。

自定义资源定义管理：

- 管理特定于 Argo CD 的 CRD（Applications、AppProjects、ApplicationSets）。

API 发现：

- 读取 Kubernetes API 端点，用于资源发现。

此策略旨在支持集群级 Argo CD 操作，包括管理命名空间和安装 CRD。创建 Argo CD 功能时，Amazon EKS 会自动使用此访问策略为所提供的功能 IAM 角色创建访问条目。

Kubernetes API 组	Kubernetes 非资源 URL	Kubernetes 资源	Kubernetes 动词（权限）
		namespaces	create, get, update, patch, delete
apiextensions.k8s.io		customresourcedefinitions	create
apiextensions.k8s.io		customresourcedefinitions （仅限 Argo CD CRD）	get, update, patch, delete
	/api, /api/*, /apis, /apis/*		get

AmazonEKSArgoCDPolicy

ARN – arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSArgoCDPolicy

此策略授予 Argo CD 功能部署和管理应用程序所需的命名空间级权限。此策略包含以下权限：

密钥管理：

- 对密钥的完全访问权限，用于 Git 凭证和集群密钥。

ConfigMap 访问权限：

- 读取 ConfigMap，以便在客户尝试使用不支持的 Argo CD ConfigMap 时发送警告。

事件管理：

- 读取和创建事件，用于应用程序生命周期跟踪。

Argo CD 资源管理：

- 对 Applications、ApplicationSets 和 AppProjects 的完全访问权限。
- 管理 Argo CD 资源的终结器和状态。

此策略旨在支持命名空间级 Argo CD 操作，包括部署和管理应用程序。创建 Argo CD 功能时，Amazon EKS 会自动使用此访问策略为所提供的功能 IAM 角色创建访问条目，作用范围为 Argo CD 命名空间。

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
	`secrets`	`*`
	`configmaps`	`get`, `list`, `watch`
	`events`	`get`, `list`, `watch`, `patch`, `create`
`argoproj.io`	`applications`，`applications/finalizers`，`applications/status`，`applicationsets`，`applicationsets/finalizers`，`applicationsets/status`，`appprojects`，`appprojects/finalizers`，`appprojects/status`	`*`

AmazonEKSKROPolicy

ARN – `arn:aws-cn:eks::aws:cluster-access-policy/AmazonEKSKROPolicy`

此策略授予 kro（Kube Resource Orchestrator）功能创建和管理自定义 Kubernetes API 所需的权限。此策略包含以下权限：

kro 资源管理：

- 对所有 kro 资源的完全访问权限，包括 ResourceGraphDefinitions 和自定义资源实例。

自定义资源定义管理：

- 创建、读取、更新和删除由 ResourceGraphDefinitions 定义的自定义 API 的 CRD。

主节点选举：

- 创建和读取协调租约，用于主节点选举。
- 更新和删除 kro 控制器租约。

事件管理：

- 创建和修补事件，用于 kro 操作。

此策略旨在通过 kro 支持全面的资源组合和自定义 API 管理。创建 kro 功能时，Amazon EKS 会自动使用此访问策略为所提供的功能 IAM 角色创建访问条目。

Kubernetes API 组	Kubernetes 资源	Kubernetes 动词（权限）
kro.run	*	*
apiextensions.k8s.io	customresourcedefinitions	*
coordination.k8s.io	leases	create, get, list, watch
coordination.k8s.io	leases（仅限 kro 控制器租约）	delete, update, patch
	events	create, patch

访问策略更新

查看有关自引入访问策略以来更新的详细信息。要获得有关此页面更改的自动提示，请订阅 文档历史记录 中的 RSS 源。

更改	描述	日期
添加适用于 EKS 功能的策略	发布用于管理 EKS 功能的 AmazonEKSACKPolicy 、AmazonEKSArgoCDClusterPolicy 、AmazonEKSArgoCDPolicy 和 AmazonEKSKROPolicy	2025 年 11 月 22 日
添加 AmazonEKSSecretReaderPolicy	添加用于密钥只读访问的新策略	2025 年 11 月 6 日
为 EKS 集群见解添加策略	发布 AmazonEKSClusterInsightsPolicy	2024 年 12 月 2 日
增加了适用于 Amazon EKS 混合节点功能的策略	发布 AmazonEKSHybridPolicy	2024 年 12 月 2 日
增加了适用于 Amazon EKS 自动模式的策略	这些访问策略授予集群 IAM 角色和节点 IAM 角色调用 Kubernetes API 的权限。Amazon 使用这些策略来自动执行存储、计算和联网资源的例行任务。	2024 年 12 月 2 日
添加 AmazonEKSAdminViewPolicy	添加新策略以扩展查看权限，包括 Secrets 等资源。	2024 年 4 月 23 日
引入访问策略。	Amazon EKS 引入访问策略。	2023 年 5 月 29 日

更改身份验证模式以使用访问条目

要开始使用访问条目，必须将集群的身份验证模式更改为 API_AND_CONFIG_MAP 或 API 模式。这为访问条目添加了 API。

Amazon 控制台

1. 打开 Amazon EKS 控制台。

2. 选择要在其中创建访问条目的集群的名称。
3. 选择访问选项卡。
4. 身份验证模式会显示集群的当前身份验证模式。如果模式显示 EKS API，您已经可以添加访问条目了，可以跳过其余的步骤。
5. 选择管理访问。
6. 对于集群身份验证模式，请使用 EKS API 选择一种模式。请注意，您不能将身份验证模式更改回移除 EKS API 和访问条目的模式。
7. 选择保存更改。Amazon EKS 开始更新集群，集群的状态更改为“更新中”，更改记录在更新历史记录选项卡中。
8. 等待集群的状态恢复为“活动”。当集群处于“活动”状态时，您可以按照the section called “创建访问条目”中的步骤为 IAM 主体添加对集群的访问权限。

Amazon CLI

1. 按照《Amazon 命令行界面用户指南》中的安装部分的描述安装 Amazon CLI。
2. 运行以下命令。将 *my-cluster* 替换为您的集群的名称。如果要永久禁用 `ConfigMap` 方法，请将 `API_AND_CONFIG_MAP` 替换为 `API`。

   Amazon EKS 开始更新集群，集群的状态更改为“更新中”，更改记录在 aws eks list-updates 中。

   ```
   aws eks update-cluster-config --name my-cluster --access-config
    authenticationMode=API_AND_CONFIG_MAP
   ```

3. 等待集群的状态恢复为“活动”。当集群处于“活动”状态时，您可以按照the section called “创建访问条目”中的步骤为 IAM 主体添加对集群的访问权限。

所需平台版本

要使用访问条目，集群的平台版本必须与下表中所列的版本相同或更高，或者为高于下表中所列版本的 Kubernetes 版本。如果您的 Kubernetes 版本未列出，则所有平台版本都支持访问条目。

Kubernetes 版本	平台版本
未列出	全部受支持
`1.30`	`eks.2`

Kubernetes 版本	平台版本
1.29	eks.1
1.28	eks.6

有关更多信息，请参阅 platform-versions。

创建访问条目

在创建访问条目之前，请考虑以下事项：

- 身份验证模式设置正确。请参阅the section called “身份验证模式”。
- 访问条目包含一个（且仅限一个）现有 IAM 主体的 Amazon 资源名称（ARN）。一个 IAM 主体不能包含在多个访问条目中。您指定的 ARN 的其他注意事项：
  - IAM 最佳实践建议使用具有短期凭证的 IAM 角色而不是具有长期凭证的 IAM 用户访问您的集群。有关更多信息，请参阅《IAM 用户指南》中的要求人类用户使用带有身份提供商的联合身份验证才能使用临时凭证访问 Amazon。
  - 如果 ARN 适用于 IAM 角色，则可以包含路径。aws-auth ConfigMap 条目中的 ARN 不能包含路径。例如，您的 ARN 可以是 arn:aws-cn:iam::<111122223333>:role/<development/apps/my-role> 或 arn:aws-cn:iam::<111122223333>:role/<my-role>。
  - 如果访问条目的类型不是 STANDARD（请参阅下文关于类型的注意事项），则 ARN 必须位于与您的集群相同的 Amazon 账户。如果类型为 STANDARD，则 ARN 可以位于与您的集群所在的账户相同或不同的 Amazon 账户。
  - 在创建访问条目后，您将无法更改 IAM 主体。
  - 如果您删除了具有此 ARN 的 IAM 主体，则访问条目不会自动删除。对于已删除的 IAM 主体，我们建议您删除具有 ARN 的访问条目。如果您不删除访问条目并重新创建 IAM 主体，即使该访问条目具有相同的 ARN，也将无法运行。这是因为尽管重新创建的 IAM 主体的 ARN 相同，但对于重新创建的 IAM 主体，roleID 或 userID（您可以用 aws sts get-caller-identity Amazon CLI 命令查看）与原始 IAM 主体不同。即使您看不到 IAM 主体的 roleID 或 userID 访问条目，Amazon EKS 也会将其与访问条目一起存储。
- 每个访问条目都有一种类型。访问条目的类型取决于与其关联的资源类型，并且不定义权限。如果不指定类型，Amazon EKS 会自动将类型设置为 STANDARD
  - EC2_LINUX：适用于与 Linux 或 Bottlerocket 自主管理型节点一起使用的 IAM 角色

  - EC2_WINDOWS：适用于与 Windows 自主管理型节点一起使用的 IAM 角色
  - FARGATE_LINUX：适用于与 Amazon Fargate（Fargate）一起使用的 IAM 角色
  - HYBRID_LINUX：适用于与混合节点一起使用的 IAM 角色
  - STANDARD：若未指定，则采用默认类型
  - EC2：适用于 EKS 自动模式自定义节点类。有关更多信息，请参阅 the section called “创建节点类访问条目”。
  - 在创建访问条目后，您将无法更改类型。
- 无需为用于托管式节点组或 Fargate 配置文件的 IAM 角色创建访问条目。EKS 将创建访问条目（如果启用），或者更新身份验证配置映射（如果访问条目不可用）
- 如果访问条目的类型为 STANDARD，则可以为该访问条目指定用户名。如果您没有为用户名指定值，Amazon EKS 会根据访问条目的类型以及您指定的 IAM 主体是 IAM 角色还是 IAM 用户，为您设置以下值之一。除非您出于特定原因要指定自己的用户名，否则我们建议您不要指定用户名，而是让 Amazon EKS 为您自动生成用户名。如果您指定自己的用户名：
  - 不能以 system:、eks:、aws:、amazon: 或 iam: 开头。
  - 如果该用户名用于 IAM 角色，我们建议您在用户名的末尾添加 {{SessionName}} 或 {{SessionNameRaw}}。如果您在用户名中添加 {{SessionName}} 或 {{SessionNameRaw}}，则该用户名必须在 {{SessionName}} 之前加一个冒号。如果担任此角色，则担任此角色时指定的 Amazon STS 会话名称将自动传递到集群，并显示在 CloudTrail 日志中。例如，您不能将用户名设置为 john{{SessionName}}。用户名必须是 :john{{SessionName}} 或 jo:hn{{SessionName}}。冒号只能在 {{SessionName}} 前面。下表中由 Amazon EKS 生成的用户名包含一个 ARN。由于 ARN 包含冒号，因此符合此要求。如果用户名中未包含 {{SessionName}}，则不需要使用冒号。请注意，会话名称 {{SessionName}} 中的特殊字符“@”已替换为“-”。{{SessionNameRaw}} 保留会话名称中的所有特殊字符。

IAM 主体类型	类型	Amazon EKS 自动设置的用户名值
用户	STANDARD	用户的 ARN。示例：arn:aws-cn:iam::<111122223333>:user/<my-user>
角色	STANDARD	分派角色时此角色的 STS ARN。Amazon EKS 会将

IAM 主体类型	类型	Amazon EKS 自动设置的用户名值
		{{SessionName}} 附加到该角色中。 示例： arn:aws-cn:sts::<111122223333>:assumed-role/<my-role>/{{SessionName}} 如果您指定的角色的 ARN 包含路径，Amazon EKS 会在生成的用户名中将其移除。
角色	EC2_LINUX 或 EC2_Windows	system:node:{{EC2PrivateDNSName}}
角色	FARGATE_LINUX	system:node:{{SessionName}}
角色	HYBRID_LINUX	system:node:{{SessionName}}

创建访问条目后，您可以更改用户名。

- 如果访问条目的类型为 STANDARD，并且您想使用 Kubernetes RBAC 授权，则可以在访问条目中添加一个或多个组名称。创建访问条目后，您可以添加和移除组名。要让 IAM 主体能够访问集群上的 Kubernetes 对象，您必须创建和管理 Kubernetes 基于角色的授权（RBAC）对象。在集群上创建 Kubernetes RoleBinding 或 ClusterRoleBinding 对象，将组名称指定为 kind: Group 的 subject。Kubernetes 授权 IAM 主体访问您已在 Kubernetes Role 或 ClusterRole 对象中指定的任何集群对象（您也在绑定的 roleRef 中指定了这些对象）。如果指定组名称，建议熟悉 Kubernetes 基于角色的授权（RBAC）对象。有关更多信息，请参阅 Kubernetes 文档中的使用 RBAC 授权。

> **Important**
>
> Amazon EKS 不会确认您集群上存在的任何 Kubernetes RBAC 对象是否包含您指定的任何组名称。例如，假设您为某个当前不存在的组创建访问条目，EKS 将会创建该组而不是返回错误。

除了 Kubernetes 授权 IAM 主体访问集群上的 Kubernetes 对象之外，您还可以将 Amazon EKS 访问策略关联到访问条目。Amazon EKS 授权 IAM 主体使用访问策略中的权限访问您集群上的 Kubernetes 对象。您可以将访问策略的权限范围限定到自己指定的 Kubernetes 命名空间。使用访问策略不需要您管理 Kubernetes RBAC 对象。有关更多信息，请参阅 the section called “关联访问策略”。

- 如果您创建类型为 `EC2_LINUX` 或 `EC2_Windows` 的访问条目，则创建访问条目的 IAM 主体必须拥有 `iam:PassRole` 权限。有关更多信息，请参阅 IAM 用户指南中的向用户授予将角色传递给 Amazon 服务的权限。
- 与标准 IAM 行为类似，访问条目创建和更新最终是一致的，并且在初始 API 调用成功返回后可能需要几秒钟才能生效。您在设计应用程序时，必须考虑到这些可能的延迟。在应用程序的关键、高可用性代码路径中，我们不建议创建或更新访问条目。而应在不常运行的、单独的初始化或设置例程中进行 更改。另外，在生产工作流程依赖这些更改之前，请务必验证更改已传播。
- 访问条目不支持服务相关角色。如果主体 ARN 是服务相关角色，则无法创建访问条目。您可以通过服务相关角色的 ARN（格式为 `arn:aws-cn:iam::*:role/aws-service-role/*`）来识别它们。

您可以使用 Amazon Web Services 管理控制台或 Amazon CLI 创建访问条目。

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择要在其中创建访问条目的集群的名称。
3. 选择访问选项卡。
4. 选择创建访问条目。
5. 对于 IAM 主体，请选择现有的 IAM 角色或用户。IAM 最佳实践建议使用具有短期凭证的 IAM 角色而不是具有长期凭证的 IAM 用户访问您的集群。有关更多信息，请参阅《IAM 用户指南》中的要求人类用户使用带有身份提供商的联合身份验证才能使用临时凭证访问 Amazon。

6. 对于类型，如果访问条目是用于自行管理的 Amazon EC2 节点的节点角色，请选择 EC2 Linux 或 EC2 Windows。否则，请接受默认值（标准）。
7. 如果您选择的类型是标准，并且想要指定用户名，请输入用户名。
8. 如果您选择的类型是标准，并且您想对 IAM 主体使用 Kubernetes RBAC 授权，请为组指定一个或多个名称。如果您未指定任何组名并想使用 Amazon EKS 授权，则可以在后续步骤中或创建访问条目之后关联访问策略。
9. （可选）对于标签，为访问条目分配标签。例如，为了更轻松地查找所有具有相同标签的资源。
10选择下一步。
11.在添加访问策略页面上，如果您选择的类型是标准，并且希望 Amazon EKS 授权 IAM 主体拥有对集群上 Kubernetes 对象的权限，请完成以下步骤。否则，请选择下一步。
    a. 对于策略名称，选择访问策略。您无法查看访问策略的权限，但这些策略包含的权限与 Kubernetes 面向用户的 `ClusterRole` 对象中的权限类似。有关更多信息，请参阅 Kubernetes 文档中的面向用户的角色。
    b. 请选择以下选项之一：
        - 集群 – 如果您希望 Amazon EKS 授权 IAM 主体拥有集群上所有 Kubernetes 对象的访问策略的权限，请选择此选项。
        - Kubernetes 命名空间 – 如果您希望 Amazon EKS 授权 IAM 主体拥有集群上特定 Kubernetes 命名空间内所有 Kubernetes 对象的访问策略权限，请选择此选项。对于命名空间，输入集群上 Kubernetes 命名空间的名称。如果要添加其他命名空间，请选择添加新命名空间并输入命名空间名称。
    c. 如果要添加其他策略，请选择添加策略。您可以对每个策略设定不同的范围，但每个策略只能添加一次。
    d. 选择下一步。
12查看访问条目的配置。如果有任何内容看起来不正确，请选择上一步以返回步骤并更正错误。如果配置正确，请选择创建。

Amazon CLI

1. 按照《Amazon 命令行界面用户指南》中的安装中所述安装 Amazon CLI。
2. 创建访问条目 您可以使用以下任意示例来创建访问条目：
    - 为自行管理的 Amazon EC2 Linux 节点组创建访问条目。请将 *`my-cluster`* 替换为您的集群名称，将 *`111122223333`* 替换为您的 Amazon 账户 ID，将 *`EKS-my-cluster-self-managed-ng-1`* 替换为您的节点 IAM 角色的名称。如果节点组是 Windows 节点组，则将 *`EC2_LINUX`* 替换为 `EC2_Windows`。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws-
cn:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1 --type EC2_LINUX
```

当您指定除 STANDARD 之外的类型时，不能使用 `--kubernetes-groups` 选项。您无法将访问策略与该访问条目相关联，因为其类型的值不是 STANDARD。

- 创建一个允许 IAM 角色的访问条目，该角色不用于 Amazon EC2 自主管理型节点组，您想要 Kubernetes 通过该角色授权访问您的集群。将 *`my-cluster`* 替换为您的集群的名称，将 *`111122223333`* 替换为您的 Amazon 账户 ID，将 *`my-role`* 替换为您的 IAM 角色的名称。将 *`Viewers`* 替换为您在集群上的 Kubernetes `RoleBinding` 或 `ClusterRoleBinding` 对象中指定的组的名称。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws-
cn:iam::111122223333:role/my-role --type STANDARD --user Viewers --kubernetes-
groups Viewers
```

- 创建允许 IAM 用户对集群进行身份验证的访问条目。之所以提供此示例，是因为尽管 IAM 最佳实践建议使用具有短期凭证的 IAM 角色而不是具有长期凭证的 IAM 用户访问集群，但这仍是可行的。有关更多信息，请参阅《IAM 用户指南》中的要求人类用户使用带有身份提供商的联合身份验证才能使用临时凭证访问 Amazon。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws-
cn:iam::111122223333:user/my-user --type STANDARD --username my-user
```

如果您希望此用户对您的集群拥有的访问权限超过 Kubernetes API 发现角色中的权限，则需要将访问策略与访问条目相关联，因为未使用 `--kubernetes-groups` 选项。有关更多信息，请参阅the section called “关联访问策略”，以及 Kubernetes 文档中的 API 发现角色。

更新访问条目

您可以使用 Amazon Web Services 管理控制台 或 Amazon CLI 更新访问条目。

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择要在其中创建访问条目的集群的名称。
3. 选择访问选项卡。
4. 选择要更新的访问条目。

5. 选择编辑。
6. 对于用户名，您可以更改现有值。
7. 对于组，您可以移除现有的组名称或添加新的组名称。如果存在以下组名称，请不要将其移除：system:nodes 或 system:bootstrappers。移除这些组可能会导致集群无法正常运行。如果您未指定任何组名称并想使用 Amazon EKS 授权，请在后续步骤中关联访问策略。
8. 对于标签，您可以为访问条目分配标签。例如，为了更轻松地查找所有具有相同标签的资源。您也可以移除现有标签。
9. 选择 Save changes（保存更改）。
10. 如果要将访问策略与条目关联，请参阅 the section called “关联访问策略”。

Amazon CLI

1. 按照《Amazon 命令行界面用户指南》中的安装中所述安装 Amazon CLI。
2. 更新访问条目 将 *my-cluster* 替换为您的集群的名称，将 *111122223333* 替换为您的 Amazon 账户 ID，将 *EKS-my-cluster-my-namespace-Viewers* 替换为 IAM 角色的名称。

```
aws eks update-access-entry --cluster-name my-cluster --principal-arn arn:aws-
cn:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --kubernetes-groups
 Viewers
```

如果访问条目的类型值不是 STANDARD，则不能使用 `--kubernetes-groups` 选项。您也不能将访问策略与访问条目关联到 STANDARD 类型之外的访问条目。

删除访问条目

如果您发现自己误删了访问条目，则可以随时重新创建。如果您要删除的访问条目与任何访问策略相关联，则关联会自动删除。在删除访问条目之前，您不必取消访问策略与访问条目的关联。

您可以使用 Amazon Web Services 管理控制台 或 Amazon CLI 删除访问条目。

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择要从中删除访问条目的集群的名称。
3. 选择访问选项卡。
4. 在访问条目列表中，选择要删除的访问条目。

5. 选择 Delete。
6. 在确认对话框中，选择删除。

Amazon CLI

1. 按照《Amazon 命令行界面用户指南》中的安装中所述安装 Amazon CLI。
2. 要删除访问条目，将 *my-cluster* 替换为您的集群名称，将 *111122223333* 替换为您的 Amazon 账户 ID，将 *my-role* 替换为您不想再访问集群的 IAM 角色的名称。

```
aws eks delete-access-entry --cluster-name my-cluster --principal-arn arn:aws-
cn:iam::111122223333:role/my-role
```

为 EKS 访问条目设置自定义用户名

在为 Amazon EKS 创建访问条目时，可以使用自动生成的用户名，也可以指定自定义用户名。本页介绍这两个选项，并将指导您设置自定义用户名。

概览

访问条目中的用户名用于识别 Kubernetes 日志和审计跟踪记录中的 IAM 主体。默认情况下，Amazon EKS 会根据 IAM 身份的 ARN 生成用户名，但您可以根据需要指定自定义用户名。

默认用户名生成

如果不指定用户名的值，Amazon EKS 会根据 IAM 身份自动生成用户名：

- 对于 IAM 用户：
  - EKS 会将 Kubernetes 用户名设置为 IAM 用户的 ARN
  - 示例：

```
{arn-aws}iam::<111122223333>:user/<my-user>
```

- 对于 IAM 角色：
  - EKS 会根据 IAM 角色的 ARN 设置 Kubernetes 用户名
  - 分派角色时此角色的 STS ARN。Amazon EKS 会将 {{SessionName}} 附加到该角色中。如果您指定的角色的 ARN 包含路径，Amazon EKS 会在生成的用户名中将其移除。
  - 示例：

```
{arn-aws}sts::<111122223333>:assumed-role/<my-role>/{{SessionName}}
```

除非您出于特定原因要指定自己的用户名，否则我们建议您不要指定用户名，而是让 Amazon EKS 为您自动生成用户名。

设置自定义用户名

创建访问条目时，可以使用 `--username` 参数指定自定义用户名：

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD --username <custom-username>
```

自定义用户名的要求

如果指定自定义用户名：

- 用户名不能以 `system:`、`eks:`、`aws:`、`amazon:` 或 `iam:` 开头。
- 如果该用户名用于 IAM 角色，我们建议您在用户名的末尾添加 `{{SessionName}}` 或 `{{SessionNameRaw}}`。
  - 如果您在用户名中添加 `{{SessionName}}` 或 `{{SessionNameRaw}}`，则该用户名必须在 {{SessionName}} 之前加一个冒号。

使用访问策略和 Amazon CLI 为 IAM 角色或用户创建访问条目

创建使用 Amazon 托管的 EKS 访问策略的 Amazon EKS 访问条目，以向 IAM 身份授予访问和管理 Kubernetes 集群的标准化权限。

概览

Amazon EKS 中的访问条目定义了 IAM 身份（用户和角色）如何访问 Kubernetes 集群并与之交互。通过创建使用 EKS 访问策略的访问条目，您可以执行以下操作：

- 向特定的 IAM 用户或角色授予访问 EKS 集群的权限
- 使用 Amazon 托管的 EKS 访问策略控制权限，这种策略提供标准化的预定义权限集
- 将权限范围限定为特定的命名空间或集群范围
- 无需修改 `aws-auth` ConfigMap 或创建 Kubernetes RBAC 资源即可简化访问管理

- 使用 Amazon 集成的 Kubernetes 访问控制方法，该方法涵盖常见使用案例，同时保持安全最佳实践

建议在大多数使用案例中使用此方法，因为其无需手动配置 Kubernetes RBAC 即可提供 Amazon 托管的标准化权限。EKS 访问策略无需手动配置 Kubernetes RBAC 资源，并提供涵盖常见使用案例的预定义权限集。

先决条件

- 必须将集群的身份验证模式配置为启用访问条目。有关更多信息，请参阅 the section called “身份验证模式”。
- 按照《Amazon 命令行界面用户指南》的安装中所述，安装和配置 Amazon CLI。

步骤 1：定义访问条目

1. 找到要向其授予权限的 IAM 身份（例如用户或角色）的 ARN。
   - 每个 IAM 身份只能有一个 EKS 访问条目。
2. 确定您希望 Amazon EKS 访问策略权限仅应用于特定的 Kubernetes 命名空间，还是应用于整个集群。
   - 如果要将权限限制为特定的命名空间，请记下命名空间名称。
3. 为 IAM 身份选择所需的 EKS 访问策略。此策略会授予集群内权限。记下策略的 ARN。
   - 有关策略列表，请参阅可用的访问策略。
4. 确定自动生成的用户名是否适用于访问条目，或者是否需要手动指定用户名。
   - Amazon 会根据 IAM 身份自动生成此值。您可以设置自定义用户名。这会显示在 Kubernetes 日志中。
   - 有关更多信息，请参阅 the section called “设置自定义用户名”。

步骤 2：创建访问条目

规划好访问条目后，使用 Amazon CLI 进行创建。

以下示例涵盖了大多数使用案例。查看 CLI 参考以了解所有配置选项。

您将在下一步中附加访问策略。

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD
```

步骤 3：关联访问策略

根据您是否希望将策略限制为指定的 Kubernetes 命名空间，命令会有所不同。

您需要访问策略的 ARN。查看可用的访问策略。

创建没有命名空间作用域的策略

```
aws eks associate-access-policy --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --policy-arn <access-policy-arn>
```

创建具有命名空间作用域的策略

```
aws eks associate-access-policy --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
 <access-policy-arn>
```

后续步骤

- 创建 kubeconfig，以便将 kubectl 与 IAM 身份结合使用

使用 Amazon CLI 通过 Kubernetes 组创建访问条目

创建 Amazon EKS 访问条目，这些条目使用 Kubernetes 组进行授权且需要手动配置 RBAC。

> **Note**
>
> 对于大多数使用案例，建议使用 EKS 访问策略，而不是本页介绍的 Kubernetes 组方法。EKS 访问策略提供了一种更简单、与 Amazon 更集成的方式来管理访问权限，无需手动配置 RBAC。仅当您需要比 EKS 访问策略所提供之控制更精细的控制时，才可使用 Kubernetes 组方法。

概览

访问条目定义了 IAM 身份（用户和角色）如何访问 Kubernetes 集群。Kubernetes 组方法通过标准的 Kubernetes RBAC 组，向 IAM 用户或角色授予访问 EKS 集群的权限。此方法需要创建和管理 Kubernetes RBAC 资源（Roles、RoleBindings、ClusterRoles 和 ClusterRoleBindings），如果您需

要高度自定义的权限集、复杂的授权要求或想要在混合 Kubernetes 环境中保持一致的访问控制模式，建议使用此方法。

本主题不介绍为用于 Amazon EC2 实例加入 EKS 集群的 IAM 身份创建访问条目。

先决条件

- 必须将集群的身份验证模式配置为启用访问条目。有关更多信息，请参阅 the section called “身份验证模式”。
- 按照《Amazon 命令行界面用户指南》的安装中所述，安装和配置 Amazon CLI。
- 建议熟悉 Kubernetes RBAC。有关更多信息，请参阅 Kubernetes 文档中的使用 RBAC 授权。

步骤 1：定义访问条目

1. 找到要向其授予权限的 IAM 身份（例如用户或角色）的 ARN。
   - 每个 IAM 身份只能有一个 EKS 访问条目。
2. 确定要与此 IAM 身份关联的 Kubernetes 组。
   - 您需要创建或使用引用了这些组的现有 Kubernetes `Role/ClusterRole` 和 `RoleBinding/ClusterRoleBinding` 资源。
3. 确定自动生成的用户名是否适用于访问条目，或者是否需要手动指定用户名。
   - Amazon 会根据 IAM 身份自动生成此值。您可以设置自定义用户名。这会显示在 Kubernetes 日志中。
   - 有关更多信息，请参阅 the section called “设置自定义用户名”。

步骤 2：通过 Kubernetes 组创建访问条目

规划好访问条目后，使用 Amazon CLI 通过相应的 Kubernetes 组创建该条目。

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD --kubernetes-groups <groups>
```

进行如下替换：

- 将 `<cluster-name>` 替换为 EKS 集群名称
- 将 `<iam-identity-arn>` 替换为 IAM 用户或角色的 ARN
- 将 `<groups>` 替换为以逗号分隔的 Kubernetes 组列表（例如“system:developers,system:readers”）

查看 CLI 参考以了解所有配置选项。

步骤 3：配置 Kubernetes RBAC

要让 IAM 主体能够访问集群上的 Kubernetes 对象，必须创建和管理 Kubernetes 基于角色的访问控制（RBAC）对象：

1. 创建定义权限的 Kubernetes `Role` 或 `ClusterRole` 对象。
2. 在集群上创建 Kubernetes `RoleBinding` 或 `ClusterRoleBinding` 对象，将组名称指定为 `kind: Group` 的 `subject`。

有关在 Kubernetes 中配置组和权限的详细信息，请参阅 Kubernetes 文档中的使用 RBAC 鉴权。

后续步骤

- 创建 kubeconfig，以便将 kubectl 与 IAM 身份结合使用

向 IAM 用户授予使用 ConfigMap 访问 Kubernetes 的权限

> **Important**
>
> `aws-auth ConfigMap` 已弃用。有关管理 Kubernetes API 访问权限的推荐方法，请参阅the section called “访问条目”。

在 Amazon EKS 控制面板上运行的 Amazon IAM Authenticator for Kubernetes 支持使用 IAM 主体访问集群。身份验证程序从 `aws-auth ConfigMap` 获取配置信息。对于所有 `aws-auth ConfigMap` 设置，请参阅 GitHub 上的完整配置格式。

将 IAM 主体添加到 Amazon EKS 集群

创建 Amazon EKS 集群时，将为创建集群的 IAM 主体自动授予 Amazon EKS 控制面板中基于集群角色的访问控制（RBAC）配置中的 `system:masters` 权限。该主体不会显示在任何可见配置中，因此请确保跟踪最初创建集群的主体。要授予其他 IAM 主体与集群进行交互的能力，请编辑 Kubernetes 中的 `aws-auth ConfigMap`，创建 Kubernetes `rolebinding` 或 `clusterrolebinding`，名为 `aws-auth ConfigMap` 中指定的 `group`。

> **Note**
>
> 有关 Kubernetes 基于角色的访问控制（RBAC）配置的更多信息，请参阅 Kubernetes 文档中的使用 RBAC 鉴权。

1. 确定 kubectl 用来访问集群的凭据。可以在计算机上使用下面的命令查看 kubectl 使用的凭据。如果不使用默认路径，请将 *~/.kube/config* 替换为 kubeconfig 文件的路径。

```
cat ~/.kube/config
```

示例输出如下。

```
[...]
contexts:
- context:
    cluster: my-cluster.region-code.eksctl.io
    user: admin@my-cluster.region-code.eksctl.io
  name: admin@my-cluster.region-code.eksctl.io
current-context: admin@my-cluster.region-code.eksctl.io
[...]
```

在上一个示例输出中，为 *my-cluster* 集群配置名为 *admin* 的用户凭证。如果这是创建了集群的用户，那么该用户有权访问您的集群。如果它不是创建集群的用户，则需要完成剩余步骤才能让其它 IAM 主体有权访问集群。IAM 最佳实践建议您向角色而不是用户授予权限。您可以使用以下命令查看哪些其他主体当前有权访问您的集群：

```
kubectl describe -n kube-system configmap/aws-auth
```

示例输出如下。

```
Name:         aws-auth
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
mapRoles:
```

```
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws-cn:iam::111122223333:role/my-node-role
  username: system:node:{{EC2PrivateDNSName}}


BinaryData
====

Events:  <none>
```

上一个示例是默认的 aws-auth ConfigMap。只有节点实例角色才有权访问集群。

2. 请确保拥有现有的 Kubernetes roles 和 rolebindings，或者可将 IAM 主体映射到其中的 clusterroles 和 clusterrolebindings。有关这些资源的更多信息，请参阅 Kubernetes 文档中的使用 RBAC 授权。

   a. 查看您现有的 Kubernetes roles 或 clusterroles。Roles 范围限定为 namespace，但 clusterroles 范围则限定为集群。

   ```
   kubectl get roles -A
   ```

   ```
   kubectl get clusterroles
   ```

   b. 查看之前的输出中返回的任何 role 或 clusterrole 的详细信息，并确认它具有您希望 IAM 主体在集群中拥有的权限（rules）。

   将 *role-name* 替换为在上一个命令的输出中返回的 role 名称。将 *kube-system* 替换为 role 的命名空间。

   ```
   kubectl describe role role-name -n kube-system
   ```

   将 *cluster-role-name* 替换为在上一个命令的输出中返回的 clusterrole 名称。

   ```
   kubectl describe clusterrole cluster-role-name
   ```

   c. 查看您现有的 Kubernetes rolebindings 或 clusterrolebindings。Rolebindings 范围限定为 namespace，但 clusterrolebindings 范围则限定为集群。

```
kubectl get rolebindings -A
```

```
kubectl get clusterrolebindings
```

d. 查看任何 rolebinding 或 clusterrolebinding 的详细信息，并确认它具有在上一步中列为 roleRef 的 role 或 clusterrole，以及为 subjects 列出的组名称。

将 *role-binding-name* 替换为在上一个命令的输出中返回的 rolebinding 名称。将 *kube-system* 替换为 rolebinding 的 namespace。

```
kubectl describe rolebinding role-binding-name -n kube-system
```

示例输出如下。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eks-console-dashboard-restricted-access-role-binding
  namespace: default
subjects:
- kind: Group
  name: eks-console-dashboard-restricted-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-console-dashboard-restricted-access-role
  apiGroup: rbac.authorization.k8s.io
```

将 *cluster-role-binding-name* 替换为在上一个命令的输出中返回的 clusterrolebinding 名称。

```
kubectl describe clusterrolebinding cluster-role-binding-name
```

示例输出如下。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks-console-dashboard-full-access-binding
```

```
subjects:
- kind: Group
  name: eks-console-dashboard-full-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: eks-console-dashboard-full-access-clusterrole
  apiGroup: rbac.authorization.k8s.io
```

3. 编辑 aws-auth ConfigMap。您可以使用 eksctl 之类的工具更新 ConfigMap，或者可以通过编辑它来进行手动更新。

> **Important**
>
> 我们建议使用 eksctl 或者其他工具来编辑 ConfigMap。有关您可以使用的其它工具的信息，请参阅《Amazon EKS 最佳实践指南》中的使用工具对 aws-authConfigMap 进行更改。格式不正确的 aws-auth ConfigMap 可能会导致您失去对集群的访问权限。

- 查看使用 eksctl 编辑 configmap 的步骤。
- 查看手动编辑 configmap 的步骤。

使用 Eksctl 编辑 Configmap

1. 您需要在设备或 Amazon CloudShell 上安装 0.215.0 版或更高版本的 eksctl 命令行工具。要安装或更新 eksctl，请参阅 eksctl 文档中的 Installation。
2. 查看 ConfigMap 中的当前映射。将 *my-cluster* 替换为您的集群的名称。将 *region-code* 替换为您的集群所在的 Amazon 区域。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

示例输出如下。

```
ARN
                USERNAME                                GROUPS
 ACCOUNT
arn:aws-cn:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
```

3. 为角色添加映射。将 *my-role* 替换为您的角色名称。将 *eks-console-dashboard-full-access-group* 替换为在 Kubernetes RoleBinding 或 ClusterRoleBinding 对象中指定的组名称。请将 *111122223333* 替换为您的账户 ID。您可以将 *admin* 替换为您选择的任何名称。

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws-cn:iam::111122223333:role/my-role --username admin --group eks-
console-dashboard-full-access-group \
    --no-duplicate-arns
```

> **Important**
>
> 角色 ARN 不能包含 role/my-team/developers/my-role 等路径。ARN 的格式必须为 arn:aws-cn:iam::*111122223333*:role/*my-role* 。在此示例中，my-team/developers/ 需要删除。

示例输出如下。

```
[...]
2022-05-09 14:51:20 [#]  adding identity "{arn-aws}iam::111122223333:role/my-role" to
 auth ConfigMap
```

4. 为用户添加映射。IAM 最佳实践建议您向角色而不是用户授予权限。将 *my-user* 替换为您的用户名。将 *eks-console-dashboard-restricted-access-group* 替换为在 Kubernetes RoleBinding 或 ClusterRoleBinding 对象中指定的组名称。请将 *111122223333* 替换为您的账户 ID。您可以将 *my-user* 替换为您选择的任何名称。

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws-cn:iam::111122223333:user/my-user --username my-user --group eks-
console-dashboard-restricted-access-group \
    --no-duplicate-arns
```

示例输出如下。

```
[...]
2022-05-09 14:53:48 [#]  adding identity "arn:aws-cn:iam::111122223333:user/my-user"
 to auth ConfigMap
```

5. 再次查看 ConfigMap 中的映射。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

示例输出如下。

```
ARN
           USERNAME                                GROUPS
      ACCOUNT
arn:aws-cn:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
arn:aws-cn:iam::111122223333:role/admin
                my-role                                 eks-console-dashboard-full-
access-group
arn:aws-cn:iam::111122223333:user/my-user
                   my-user                                 eks-console-dashboard-
restricted-access-group
```

手动编辑 Configmap

1. 打开 ConfigMap 文件进行编辑。

```
kubectl edit -n kube-system configmap/aws-auth
```

> **Note**
>
> 如果您收到错误指示“Error from server (NotFound): configmaps "aws-auth" not found”，请使用将 aws-auth ConfigMap 应用于您的集群中的过程来应用库存 ConfigMap。

2. 将您的 IAM 主体添加到 ConfigMap。IAM 组不是 IAM 主体，因此无法将其添加到 ConfigMap。
   - 添加 IAM 角色（例如，对于联合身份用户）：将角色详细信息添加到 data 下 ConfigMap 的 mapRoles 部分。如果此部分在文件中尚不存在，请添加它。每个条目支持以下参数：
     - rolearn：要添加的 IAM 角色的 ARN。此值不能包含路径。例如，您无法指定 ARN，例如 arn:aws-cn:iam::*111122223333*:role/my-team/developers/*role-name* 。ARN 需要为 arn:aws-cn:iam::*111122223333*:role/*role-name* 。
     - username：Kubernetes 内要映射到 IAM 角色的用户名。

- 组：要将角色映射到的组或 Kubernetes 组列表。该组可以是默认组，也可以是 `clusterrolebinding` 或 `rolebinding` 中指定的组。有关更多信息，请参阅 Kubernetes 文档中的默认角色和角色绑定。

- 要添加 IAM 用户：IAM 最佳实践建议您向角色而不是用户授予权限。将用户详细信息添加到 `data` 下 `ConfigMap` 的 `mapUsers` 部分。如果此部分在文件中尚不存在，请添加它。每个条目支持以下参数：
  - userarn：要添加的 IAM 用户的 ARN。
  - username：Kubernetes 内要映射到 IAM 用户的用户名。
  - 组：要将用户映射到的组或 Kubernetes 组列表。该组可以是默认组，也可以是 `clusterrolebinding` 或 `rolebinding` 中指定的组。有关更多信息，请参阅 Kubernetes 文档中的默认角色和角色绑定。

3. 例如，下面的 YAML 块包含：

- 一个 `mapRoles` 部分，此部分将 IAM 节点实例映射到 Kubernetes 组，以便节点可以自行注册到集群，和映射到可以查看所有集群的所有 Kubernetes 资源的 Kubernetes 组的 `my-console-viewer-role` IAM 角色。有关 `my-console-viewer-role` IAM 角色所需的 IAM 和 Kubernetes 组权限的列表，请参阅 the section called “所需的权限”。
- 一个 `mapUsers` 部分，此部分将默认 Amazon 账户中的 `admin` IAM 用户映射到 `system:masters` Kubernetes 组，以及映射到可以查看特定命名空间的 Kubernetes 资源的 Kubernetes 组的另一个 Amazon 账户的 `my-user` 用户。有关 `my-user` IAM 用户所需的 IAM 和 Kubernetes 组权限的列表，请参阅 the section called “所需的权限”。

  根据需要添加或删除行并将所有 example values 替换为您自己的值。

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
 will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws-cn:iam::111122223333:role/my-role
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - eks-console-dashboard-full-access-group
```

```
    rolearn: arn:aws-cn:iam::111122223333:role/my-console-viewer-role
    username: my-console-viewer-role
mapUsers: |
  - groups:
    - system:masters
    userarn: arn:aws-cn:iam::111122223333:user/admin
    username: admin
  - groups:
    - eks-console-dashboard-restricted-access-group
    userarn: arn:aws-cn:iam::444455556666:user/my-user
    username: my-user
```

4. 保存文件并退出文本编辑器。

将 **aws-authConfigMap** 应用到集群

使用 eksctl 创建托管节点组时或创建节点组时自动创建 aws-authConfigMap 并应用于集群。最初创建它的目的是允许节点加入您的集群，也可以使用 ConfigMap 为 IAM 主体添加基于角色的访问控制（RBAC）。如果您尚未启动自行管理的节点并且未将 aws-auth ConfigMap 应用到集群，则可以按照下面的过程执行此操作。

1. 检查您是否已经应用了 aws-auth ConfigMap。

```
kubectl describe configmap -n kube-system aws-auth
```

   如果您收到错误指示“Error from server (NotFound): configmaps "aws-auth" not found”，则继续以下步骤应用库存 ConfigMap。

2. 下载、编辑和应用 Amazon 身份验证器配置映射。

   a. 下载配置映射。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

   b. 在 aws-auth-cm.yaml 文件中，将 rolearn 设置为与您的节点关联的 IAM 角色的 Amazon 资源名称（ARN）。您可以使用文本编辑器或者通过替换 *my-node-instance-role* 和运行以下命令来执行此操作：

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

请勿修改此文件中的任何其它行。

> **Important**
>
> 角色 ARN 不能包含 `role/my-team/developers/my-role` 等路径。ARN 的格式必须为 `arn:aws-cn:iam::111122223333:role/my-role` 。在此示例中，`my-team/developers/` 需要删除。

您可以检查节点组的 Amazon CloudFormation 堆栈输出，并查找以下值：

- InstanceRoleARN – 对于使用 `eksctl` 创建的节点组
- NodeInstanceRole – 对于在 Amazon Web Services 管理控制台 中使用 Amazon EKS 提供的 Amazon CloudFormation 模板创建的节点组

c. 应用配置。此命令可能需要几分钟才能完成。

```
kubectl apply -f aws-auth-cm.yaml
```

> **Note**
>
> 如果您收到任何授权或资源类型错误，请参阅故障排除主题中的 the section called “未经授权或访问被拒绝 (kubectl)”。

3. 查看节点的状态并等待它们达到 `Ready` 状态。

```
kubectl get nodes --watch
```

输入 `Ctrl+C` 以返回到 Shell 提示符。

通过外部 OIDC 提供者向用户授予 Kubernetes 访问权限

Amazon EKS 支持使用 OpenID Connect (OIDC) 身份提供商作为对您的集群的用户进行身份验证的方法。OIDC 身份提供者可与 Amazon Identity and Access Management（IAM）结合使用或作为其替代方法。有关使用 IAM 的更多信息，请参阅the section called “Kubernetes API 访问权限”。配置集群身份验证后，您可以创建 Kubernetes `roles` 和 `clusterroles` 以将权限分配给角色，然后使用

Kubernetes `rolebindings` 和 `clusterrolebindings` 将角色绑定到身份。有关更多信息，请参阅 Kubernetes 文档中的使用 RBAC 授权。

- 您可以将一个 OIDC 身份提供商与您的集群关联。
- Kubernetes 不提供 OIDC 身份提供者。您可以使用现有的公共 OIDC 身份提供商，也可以运行您自己的身份提供商。有关经认证提供商的列表，请参阅 OpenID 网站上的 OpenID 认证。
- OIDC 身份提供商的发布者 URL 必须可公开访问，以便 Amazon EKS 能够发现签名密钥。Amazon EKS 不支持拥有自签名证书的 OIDC 身份提供者。
- 您不能禁用向集群进行 IAM 身份验证，因为在将节点加入集群时仍需要使用。
- Amazon EKS 集群仍必须由 Amazon IAM 主体创建，而不是由 OIDC 身份提供商用户创建。这是因为集群创建者与 Amazon EKS API 进行交互，而不是与 Kubernetes API 进行交互。
- 如果为控制面板启用了 CloudWatch 日志，则将在集群的审核日志中列出经 OIDC 身份提供者验证的用户。有关更多信息，请参阅 the section called “启用或禁用控制面板日志”。
- 您无法使用来自 OIDC 提供者的账户登录 Amazon Web Services 管理控制台。您只能通过使用 Amazon Identity and Access Management 账户登录 Amazon Web Services 管理控制台，从而the section called “访问集群资源”。

关联 OIDC 身份提供商

您需要提供商提供以下信息，然后才能将 OIDC 身份提供商与集群关联：

发布者 URL

OIDC 身份提供者的 URL，该 URL 允许 API 服务器发现用于验证令牌的公共签名密钥。该 URL 必须以 `https://` 开头，并应与提供者的 OIDC ID 令牌中的 `iss` 声明相对应。根据 OIDC 标准，允许使用路径组件，但不允许使用查询参数。通常，URL 只包含一个主机名称，如 `https://server.example.org` 或 `https://example.com`。该 URL 应指向 `.well-known/openid-configuration` 以下的级别，并且必须可通过 Internet 网公开访问。

客户端 ID（也称为受众）

向 OIDC 身份提供商发出身份验证请求的客户端应用程序的 ID。

您可以使用 `eksctl` 或 Amazon Web Services 管理控制台 关联身份提供商。

使用 eksctl 关联身份提供者

1. 使用以下内容创建名为 `associate-identity-provider.yaml` 的文件。将 example values 替换为您自己的值。`identityProviders` 部分中的值是从 OIDC 身份提供商处获取的。仅 `identityProviders` 下的 `name`、`type`、`issuerUrl` 和 `clientId` 设置需要相应值。

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: your-region-code

identityProviders:
  - name: my-provider
    type: oidc
    issuerUrl: https://example.com
    clientId: kubernetes
    usernameClaim: email
    usernamePrefix: my-username-prefix
    groupsClaim: my-claim
    groupsPrefix: my-groups-prefix
    requiredClaims:
      string: string
    tags:
      env: dev
```

> **Important**
>
> 不要为 `groupsPrefix` 或 `usernamePrefix` 指定 `system:` 或该字符串的任何部分。

2. 创建提供商。

```
eksctl associate identityprovider -f associate-identity-provider.yaml
```

3. 要使用 `kubectl` 以与集群和 OIDC 身份提供者结合使用，请参阅 Kubernetes 文档中的使用 kubectl。

您可以使用 Amazon 控制台关联身份提供者。

1. 打开 Amazon EKS 控制台。
2. 选择您的集群，然后选择访问选项卡。
3. 在 OIDC 身份提供者部分中，选择“关联身份提供者”。
4. 在 Associate OIDC Identity Provider（关联 OIDC 身份提供商）页面上，输入或选择以下选项，然后选择 Associate（关联）。
   - 对于 Name（名称），为提供商输入一个唯一的名称。
   - 对于 Issuer URL（发布者 URL），输入您的提供商的 URL。此 URL 必须可通过 Internet 进行访问。
   - 在客户端 ID 中，输入 OIDC 身份提供者的客户端 ID（也称为受众）。
   - 对于 Username claim（用户名声明），输入要用作用户名的声明。
   - 对于组声明，输入要用作用户的组的声明。
   - （可选）选择 Advanced options（高级选项），输入或选择以下信息。
     - Username prefix（用户名前缀）– 输入要加在用户名声明前面的前缀。该前缀加在用户名声明前面，以防止与现有名称发生冲突。如果您未提供值，并且用户名为 `email` 之外的值，则前缀默认为 Issuer URL（发布者 URL）的值。您可以使用值 `-` 来禁用所有前缀。不要指定 `system:` 或该字符串的任何部分。
     - 组前缀 – 输入要加在组声明前面的前缀。该前缀加在组声明前面，以防止与现有名称（如 `system: groups`）发生冲突。例如，值 `oidc:` 会创建组名称，如 `oidc:engineering` 和 `oidc:infra`。不要指定 `system:` 或该字符串的任何部分。
     - Required claims（所需声明）– 选择 Add claim（添加声明），然后在客户端 ID 令牌中输入描述所需声明的一个或多个键值对。这些键值对描述了 ID 令牌中所需的声明。如果设置，则验证每个声明是否存在于具有匹配值的 ID 令牌中。
       a. 要使用 `kubectl` 以与集群和 OIDC 身份提供者结合使用，请参阅 Kubernetes 文档中的使用 kubectl。

示例 IAM policy

如果要阻止 OIDC 身份提供商与集群关联，请创建以下 IAM policy 并将其与 Amazon EKS 管理员的 IAM 账户关联。有关更多信息，请参阅《IAM 用户指南》中的创建 IAM 策略和添加 IAM 身份权限以及《服务授权参考》中的 Actions。

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "denyOIDC",
            "Effect": "Deny",
            "Action": [
                "eks:AssociateIdentityProviderConfig"
            ],
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/*"

        },
        {
            "Sid": "eksAdmin",
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        }
    ]
}
```

如果 clientID 为 kubernetes 而 issuerUrl 为 https://cognito-idp.us-west-2amazonaws.com/* ，则以下示例策略允许进行 OIDC 身份提供商关联。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCognitoOnly",
            "Effect": "Deny",
            "Action": "eks:AssociateIdentityProviderConfig",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
            "Condition": {
                "StringNotLikeIfExists": {
                    "eks:issuerUrl": "https://cognito-idp.us-west-2.amazonaws.com/*"
                }
            }
        },
        {
            "Sid": "DenyOtherClients",
            "Effect": "Deny",
            "Action": "eks:AssociateIdentityProviderConfig",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
```

```
            "Condition": {
                "StringNotEquals": {
                    "eks:clientId": "kubernetes"
                }
            }
        },
        {
            "Sid": "AllowOthers",
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
        }
    ]
}
```

解除 OIDC 身份提供商与集群的关联

如果您解除 OIDC 身份提供商与集群的关联，则提供商中包含的用户将无法再访问该集群。但是，您仍然可以使用 IAM 主体访问该集群。

1. 打开 Amazon EKS 控制台。
2. 在 OIDC Identity Providers（OIDC 身份提供商）部分中，选择 Disassociate（解除关联），输入身份提供商名称，然后选择 `Disassociate`。

在 Amazon Web Services 管理控制台中查看 Kubernetes 资源

您可以使用 Amazon Web Services 管理控制台 查看部署到您的集群的 Kubernetes 资源。您无法使用 Amazon CLI 或 eksctl 查看 Kubernetes 资源。要使用命令行工具查看 Kubernetes 资源，请使用 kubectl。

> **Note**
>
> 要在 Amazon Web Services 管理控制台中查看资源选项卡和计算选项卡上的节点部分，您使用的 IAM 主体必须具有特定的 IAM 和 Kubernetes 权限。有关更多信息，请参阅 the section called “所需的权限”。

1. 打开 Amazon EKS 控制台。
2. 在 Clusters（集群）列表中，选择包含要查看的 Kubernetes 资源的集群。

3. 选择资源选项卡。
4. 选择您要查看其资源的 Resource type（资源类型）组，例如 Workloads（工作负载）。您可以看到该组中的资源类型列表。
5. 选择资源类型，例如 Workloads（工作负载）组中的 Deployments（部署）。您可以看到资源类型的描述、Kubernetes 文档的链接以获取有关资源类型的更多信息，以及在集群上部署的该类型资源的列表。如果列表为空，则表示您的集群中没有部署此类型的资源。
6. 选择一种资源以查看关于该资源的更多信息。请尝试以下示例：
   - 依次选择 Workloads（工作负载）组、Deployments（部署）资源类型和 coredns 资源。当您选择资源时，您默认情况下处于 Structured view（结构化视图）中。对于某些资源类型，您会在 Structured view（结构化视图）中看到 Pods（容器组（pod））部分。本节列出了由工作负载管理的容器组（pod）。您可以选择列出的任何容器组（pod），查看有关容器组（pod）的信息。并非所有资源类型都在结构化视图中显示信息。如果您在该资源的页面的右上角选择 Raw view（原始视图），您可以看到来自 Kubernetes API 对该资源的完整 JSON 响应。
   - 选择 Cluster（集群）组，然后选择 Nodes（节点）资源类型。您可以看到集群中所有节点的列表。节点可以是任意 Amazon EKS 节点类型。这与您在为集群选择 Compute（计算）选项时在 Nodes（节点）部分中看到的列表相同。从列表中选择节点资源。在结构化视图中，您还会看到 Pods（容器组（pod））部分。此部分会显示在节点上运行的所有容器组（pod）。

所需的权限

要在 Amazon Web Services 管理控制台中查看资源选项卡和计算选项卡上的节点部分，您使用的 IAM 主体必须具有特定的最低 IAM 和 Kubernetes 权限。完成以下步骤以将所需的权限分配给您的 IAM 主体。

1. 确保向您正在使用的 IAM 主体分配 `eks:AccessKubernetesApi` 以及查看 Kubernetes 资源所需的其他 IAM 权限。有关如何为 IAM 主体编辑权限的更多信息，请参阅《IAM 用户指南》中的控制主体的访问权限。有关如何编辑角色权限策略的信息，请参阅《IAM 用户指南》中的修改角色权限策略（控制台）。

   以下示例策略包含主体查看账户中所有集群的 Kubernetes 资源所需的权限。将 *111122223333* 替换为您的 Amazon 账户 ID。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "eks:ListFargateProfiles",
                "eks:DescribeNodegroup",
                "eks:ListNodegroups",
                "eks:ListUpdates",
                "eks:AccessKubernetesApi",
                "eks:ListAddons",
                "eks:DescribeCluster",
                "eks:DescribeAddonVersions",
                "eks:ListClusters",
                "eks:ListIdentityProviderConfigs",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:GetParameter",
            "Resource": "arn:aws:ssm:*:111122223333:parameter/*"
        }
    ]
}
```

要查看已连接集群中的节点，Amazon EKS 连接器 IAM 角色应能够模拟集群中的主体。这让 Amazon EKS Connector 将主体映射到 Kubernetes 用户。

2. 创建绑定到具有必要权限以查看 Kubernetes 资源的 Kubernetes `role` 或 `clusterrole` 的 Kubernetes `rolebinding` 或 `clusterrolebinding`。要了解有关 Kubernetes 角色和角色绑定的更多信息，请参阅 Kubernetes 文档中的使用 RBAC 授权。您可以将以下清单之一应用于使用必要的 Kubernetes 权限创建 `role` 或 `rolebinding` 或者 `clusterrole` 和 `clusterrolebinding` 的集群。

   查看所有命名空间中的 Kubernetes 资源

   - 文件中的组名为 `eks-console-dashboard-full-access-group`。使用以下命令将清单应用于集群：

     ```
     kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
     ```

   查看特定命名空间中的 Kubernetes 资源

   - 此文件中的命名空间为 `default`。文件中的组名为 `eks-console-dashboard-restricted-access-group`。使用以下命令将清单应用于集群：

```
kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

如果您需要更改 Kubernetes 组名称、命名空间、权限或文件中的任何其他配置，请先下载文件并对其进行编辑，然后再将其应用于集群：

a. 使用下面的命令之一下载文件：

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
```

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

b. 根据需要编辑文件。

c. 使用以下命令之一将清单应用于集群：

```
kubectl apply -f eks-console-full-access.yaml
```

```
kubectl apply -f eks-console-restricted-access.yaml
```

3. 将 IAM 主体映射到 aws-auth ConfigMap 中的 Kubernetes 用户或组。您可以使用 eksctl 之类的工具更新 ConfigMap，或者可以通过编辑它来进行手动更新。

> **Important**
>
> 我们建议使用 eksctl 或者其他工具来编辑 ConfigMap。有关您可以使用的其它工具的信息，请参阅《Amazon EKS 最佳实践指南》中的使用工具对 aws-authConfigMap 进行更改。格式不正确的 aws-auth ConfigMap 可能会导致您失去对集群的访问权限。

使用 eksctl 编辑

1. 您需要在设备或 Amazon CloudShell 上安装 0.215.0 版或更高版本的 eksctl 命令行工具。要安装或更新 eksctl，请参阅 eksctl 文档中的 Installation。
2. 查看 ConfigMap 中的当前映射。将 *my-cluster* 替换为您的集群的名称。将 *region-code* 替换为您的集群所在的 Amazon 区域。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

示例输出如下。

```
ARN
            USERNAME                                GROUPS
 ACCOUNT
arn:aws-cn:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
```

3. 为角色添加映射。此示例假设您已在第一步中将 IAM 权限附加到名为 *my-console-viewer-role* 的角色。请将 *111122223333* 替换为您的账户 ID。

```
eksctl create iamidentitymapping \
    --cluster my-cluster \
    --region=region-code \
    --arn arn:aws-cn:iam::111122223333:role/my-console-viewer-role \
    --group eks-console-dashboard-full-access-group \
    --no-duplicate-arns
```

> **Important**
>
> 角色 ARN 不能包含 `role/my-team/developers/my-role` 等路径。ARN 的格式必须为 `arn:aws-cn:iam::111122223333:role/my-role` 。在此示例中，`my-team/developers/` 需要删除。

示例输出如下。

```
[...]
2022-05-09 14:51:20 [#]  adding identity "arn:aws-cn:iam::111122223333:role/my-
console-viewer-role" to auth ConfigMap
```

4. 为用户添加映射。IAM 最佳实践建议您向角色而不是用户授予权限。此示例假设您已在第一步中将 IAM 权限附加到名为 *my-user* 的用户。请将 *111122223333* 替换为您的账户 ID。

```
eksctl create iamidentitymapping \
    --cluster my-cluster \
```

```
    --region=region-code \
    --arn arn:aws-cn:iam::111122223333:user/my-user \
    --group eks-console-dashboard-restricted-access-group \
    --no-duplicate-arns
```

示例输出如下。

```
[...]
2022-05-09 14:53:48 [#]  adding identity "arn:aws-cn:iam::111122223333:user/my-user"
 to auth ConfigMap
```

5. 再次查看 ConfigMap 中的映射。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

示例输出如下。

```
ARN
           USERNAME                                GROUPS
      ACCOUNT
arn:aws-cn:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
arn:aws-cn:iam::111122223333:role/my-console-viewer-role
                                                        eks-console-dashboard-full-
access-group
arn:aws-cn:iam::111122223333:user/my-user
                                                           eks-console-dashboard-
restricted-access-group
```

手动编辑 ConfigMap

有关将用户或角色添加到 aws-auth ConfigMap 中的更多信息，请参阅the section called “将 IAM 主体添加到 Amazon EKS 集群”。

1. 打开 aws-auth ConfigMap 进行编辑。

```
kubectl edit -n kube-system configmap/aws-auth
```

2. 将映射添加到 aws-auth ConfigMap，但不要替换任何现有的映射。以下示例添加了在第一步中添加权限的 IAM 主体与在上一步中创建的 Kubernetes 组之间的映射：

   - *my-console-viewer-role* 角色和 eks-console-dashboard-full-access-group。
   - *my-user* 用户和 eks-console-dashboard-restricted-access-group。

   这些例假设您已在第一步中将 IAM 权限附加到名为 *my-console-viewer-role* 的角色和名为 *my-user* 的用户。将 *111122223333* 替换为您的 Amazon 账户 ID。

   ```
   apiVersion: v1
   data:
   mapRoles: |
     - groups:
       - eks-console-dashboard-full-access-group
       rolearn: arn:aws-cn:iam::111122223333:role/my-console-viewer-role
       username: my-console-viewer-role
   mapUsers: |
     - groups:
       - eks-console-dashboard-restricted-access-group
       userarn: arn:aws-cn:iam::111122223333:user/my-user
       username: my-user
   ```

   > **Important**
   >
   > 角色 ARN 不能包含 role/my-team/developers/my-console-viewer-role 等路径。ARN 的格式必须为 arn:aws-cn:iam::*111122223333*:role/*my-console-viewer-role* 。在此示例中，my-team/developers/ 需要删除。

3. 保存文件并退出文本编辑器。

通过创建 kubeconfig 文件将 kubectl 连接到 EKS 集群

在本主题中，您将为您的集群创建 kubeconfig 文件（或更新现有文件）。

kubectl 命令行工具使用 kubeconfig 文件中的配置信息与集群的 API 服务器通信。有关更多信息，请参阅 Kubernetes 文档中的使用 kubeconfig 文件组织集群访问。

Amazon EKS 使用带 kubectl 的 aws eks get-token 命令进行集群身份验证。默认情况下，Amazon CLI 使用以下命令返回的相同凭证：

```
aws sts get-caller-identity
```

- 现有 Amazon EKS 集群。要部署一个角色，请参阅开始使用。
- 您的设备或 Amazon CloudShell 上安装了 kubectl 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 1.29，则可以将 kubectl 的 1.28、1.29 或 1.30 版本与之配合使用。要安装或升级 kubectl，请参阅 the section called “设置 kubectl 和 eksctl”。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 2.12.3 或更高版本，或版本 1.27.160 或更高版本。要查看当前版本，请使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 有权将 eks:DescribeCluster API 操作用于您指定的集群的 IAM 用户或角色。有关更多信息，请参阅 the section called “基于身份的策略”。如果您使用自己的 OpenID Connect 提供者提供的身份来访问集群，请参阅 Kubernetes 文档中的使用 kubectl 来创建或更新 kube config 文件。

自动创建 kubeconfig 文件

- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 2.12.3 或更高版本，或版本 1.27.160 或更高版本。要查看当前版本，请使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 将 eks:DescribeCluster API 操作用于您指定的集群的权限。有关更多信息，请参阅 the section called “基于身份的策略”。

1. 为集群创建或更新 kubeconfig 文件。将 *region-code* 替换为您的集群所在的 Amazon 区域，并将 *my-cluster* 替换为您的集群的名称。

   ```
   aws eks update-kubeconfig --region region-code --name my-cluster
   ```

预设情况下，在主目录的原定设置 kubeconfig 路径 (.kube) 中创建得到的配置文件，或者或与该位置的现有 config 合并。您可以使用 --kubeconfig 选项指定其他路径。

发出 kubectl 命令时，可以使用 --role-arn 选项指定 IAM 角色 ARN 供身份验证使用。否则，将使用默认 Amazon CLI 或 SDK 凭证链中的 IAM 主体。通过运行 aws sts get-caller-identity 命令可以查看默认 Amazon CLI 或 SDK 标识。

对于所有可用选项，运行 aws eks update-kubeconfig help 命令，或请参阅《Amazon CLI 命令参考》中的 update-kubeconfig。

2. 测试配置。

```
kubectl get svc
```

示例输出如下。

```
NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   1m
```

如果您收到任何授权或资源类型错误，请参阅故障排除主题中的 the section called “未经授权或访问被拒绝 (kubectl)”。

使用 Kubernetes 服务账户授予 Kubernetes 工作负载访问 Amazon 的权限

A Kubernetes service account provides an identity for processes that run in a Pod. For more information see 管理服务账户 in the Kubernetes documentation. If your Pod needs access to Amazon services, you can map the service account to an Amazon Identity and Access Management identity to grant that access. For more information, see the section called “包含 IRSA 的凭证” or the section called “容器组身份”.

服务账户令牌

在 Kubernetes 版本中默认启用 BoundServiceAccountTokenVolume 功能。此功能通过允许在 Kubernetes 上运行工作负载来提高服务账户令牌的安全性，以请求与受众、时间和密钥绑定的 JSON Web 令牌。服务账户令牌有效期为 1 小时。在较低的 Kubernetes 版本中，令牌并无到期限制。这意味着依赖这些令牌的客户端必须在一小时内刷新令牌。以下 Kubernetes 客户端 SDK 会在所需的时间范围内自动刷新令牌：

- Go 版本 `0.15.7` 和更高版本
- Python 版本 `12.0.0` 和更高版本
- Java 版本 `9.0.0` 和更高版本
- JavaScript 版本 `0.10.3` 和更高版本
- Ruby `master` 分支
- Haskell 版本 `0.3.0.0`
- C# 版本 `7.0.5` 和更高版本

如果您的工作负载使用的是早期客户端，则必须予以更新。为了使客户顺利迁移到更新的有时限的服务账户令牌，Kubernetes 在默认一小时内向服务账户令牌添加延长的到期期限。对于 Amazon EKS 集群，延长到期期限为 90 天。Amazon EKS 集群的 Kubernetes API 服务器会拒绝令牌超过 90 天的请求。我们建议您检查应用程序及其依赖项，以确保 Kubernetes 客户端开发工具包版本等于或高于之前列出的版本。

当 API 服务器收到令牌超过一小时的请求时，它会使用 `annotations.authentication.k8s.io/stale-token` 注释 API 审核日志事件。注释的值与以下示例类似：

```
subject: system:serviceaccount:common:fluent-bit, seconds after warning threshold:
 4185802.
```

如果您的集群启用了控制面板日志记录，则注释在审计日志内。您可以使用以下 CloudWatch Logs Insights 查询，识别 Amazon EKS 集群中使用过时令牌的所有容器组（pod）：

```
fields @timestamp
|filter @logStream like /kube-apiserver-audit/
|filter @message like /seconds after warning threshold/
|parse @message "subject: *, seconds after warning threshold:*\"" as subject,
 elapsedtime
```

`subject` 指的是容器组（pod）使用的服务账户。`elapsedtime` 表示读取最新令牌后的时间（以秒为单位）。对 API 服务器的请求在 `elapsedtime` 超过 90 天（7,776,000 秒）时被拒绝。您应该主动更新应用程序的 Kubernetes 客户端开发工具包，以使用前面列出的其中一个自动刷新令牌的版本。如果使用的服务账户令牌接近 90 天，并且您没有足够的时间在令牌到期之前更新客户端开发工具包版本，您可以终止现有容器组（pod）并创建新的容器组（pod）。这将导致重新获取服务账户令牌，从而为您提供额外 90 天的时间来更新客户端版本开发工具包。

如果容器组（pod）是部署的一部分，在保持高可用性的同时终止容器组（pod）的建议方法是使用以下命令执行部署。将 *my-deployment* 替换为您的部署的名称。

```
kubectl rollout restart deployment/my-deployment
```

集群附加组件

已更新以下集群附加组件，以使用自动重新获取服务账户令牌的 Kubernetes 客户端开发工具包。我们建议确保已在集群上安装列出的版本或更高版本。

- 适用于 Kubernetes 的 Amazon VPC CNI 插件和指标帮助程序插件版本 `1.8.0` 和更高版本。要查看当前版本或进行更新，请参阅 the section called “Amazon VPC CNI” 和 cni-metrics-helper。
- CoreDNS `1.8.4` 及更高版本。要查看当前版本或更新它，请参阅 the section called “CoreDNS”。
- Amazon 负载均衡器控制器版本 `2.0.0` 和更高版本。要查看当前版本或更新它，请参阅 the section called “ Amazon 负载均衡器控制器”。
- 当前 `kube-proxy` 版本。要查看当前版本或更新它，请参阅 the section called “kube-proxy”。
- Amazon for Fluent Bit 版本 `2.25.0` 或更高版本。要更新当前版本，请参阅 GitHub 上的发布。
- Fluentd 镜像版本 1.14.6-1.2 或更高版本以及适用于 Kubernetes 元数据的 Fluentd 筛选器插件版本 2.11.1 或更高版本。

向 Amazon Elastic Kubernetes Service 集群上的工作负载授予 Amazon Identity and Access Management 权限

对于向 Amazon EKS 集群中运行的工作负载授予 Amazon Identity and Access Management 权限，Amazon EKS 提供了两种方法：服务账户的 IAM 角色和 EKS 容器组身份。

服务账户的 IAM 角色

> 服务账户的 IAM 角色（IRSA）可为 Amazon 上运行的 Kubernetes 应用程序配置细粒度 IAM 权限，使其能够访问 Amazon S3 存储桶、Amazon DynamoDB 表等各种其他 Amazon 资源。您可以在同一 Amazon EKS 集群中同时运行多个应用程序，并确保每个应用程序仅拥有所需的最低权限集。IRSA 旨在支持 Amazon 支持的各种 Kubernetes 部署选项，例如 Amazon EKS、Amazon EKS Anywhere、Amazon 上的 Red Hat OpenShift 服务，以及 Amazon EC2 实例上的自主管理型 Kubernetes 集群。因此，IRSA 是使用基础 Amazon 服务（如 IAM）构建的，不直接依赖 Amazon EKS 服务和 EKS API。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。

EKS 容器组身份

EKS 容器组身份为集群管理员提供了一个简化的工作流，用于对应用程序进行身份验证，以访问各种其他 Amazon 资源，例如 Amazon S3 存储桶、Amazon DynamoDB 表等。EKS 容器组身份仅适用于 EKS，因此简化了集群管理员配置 Kubernetes 应用程序以获取 IAM 权限的方式。现在可以直接通过 Amazon Web Services 管理控制台、EKS API 和 Amazon CLI，用更少的步骤轻松配置这些权限，无需对集群内的任何 Kubernetes 对象执行任何操作。集群管理员无需在 EKS 和 IAM 服务之间切换，也无需使用特权 IAM 操作来配置应用程序所需的权限。现在可以跨多个集群使用 IAM 角色，无需在创建新集群时更新角色信任策略。EKS 容器组身份提供的 IAM 凭证包括角色会话标签，以及集群名称、命名空间、服务账户名称等属性。角色会话标签允许管理员根据匹配的标签访问 Amazon 资源，从而使管理员能够创建可跨服务账户使用的单一角色。有关更多信息，请参阅 the section called “容器组身份”。

比较 EKS 容器组身份和 IRSA

总的来说，EKS 容器组身份和 IRSA 都允许您向在 Kubernetes 集群上运行的应用程序授予 IAM 权限。但是，二者在配置方式、支持的限制和启用的功能方面有根本不同。下面，我们将比较这两种解决方案的一些关键方面。

> **Note**
>
> Amazon 建议尽可能使用 EKS 容器组（pod）身份授予您的容器组（pod）访问 Amazon 资源的权限。有关更多信息，请参阅 the section called “容器组身份”。

属性	EKS 容器组身份	IRSA
角色可扩展性	您必须对每个角色设置一次，才能与新引入的 Amazon EKS 服务主体 `pods.eks.amazonaws.com` 建立信任。完成此一次性步骤后，每次在新集群中使用角色时，都无需更新该角色的信任策略。	每次要在新集群中使用 IAM 角色时，都必须使用新的 EKS 集群 OIDC 提供者端点，来更新该角色的信任策略。

属性	EKS 容器组身份	IRSA
集群可扩展性	EKS 容器组身份不需要用户设置 IAM OIDC 提供者，因此这一限制并不适用。	每个 EKS 集群都具有与其关联的 OpenID Connect（OIDC）发布者 URL。要使用 IRSA，需要在 IAM 中为每个 EKS 集群创建一个唯一的 OpenID Connect 提供者。IAM 对每个 Amazon 账户的默认全局限制为 100 个 OIDC 提供者。如果计划为每个使用 IRSA 的 Amazon 账户设置超过 100 个 EKS 集群，则会达到 IAM OIDC 提供者限制。
角色可扩展性	EKS 容器组身份不要求用户在信任策略中定义 IAM 角色和服务账户之间的信任关系，因此这一限制并不适用。	在 IRSA 中，您可以在角色的信任策略中定义 IAM 角色和服务账户之间的信任关系。默认情况下，信任策略大小的长度为 2048。这意味着您通常可以在单个信任策略中定义 4 个信任关系。虽然您可以增加信任策略长度限制，但单个信任策略中通常最多只能有 8 个信任关系。
STS API 配额使用	EKS 容器组身份简化了向容器组（pod）交付 Amazon 凭证的过程，无需您的代码使用 Amazon Security Token Service（STS）直接调用。EKS 服务会处理角色代入，并向在容器组（pod）中使用 Amazon SDK 编写的应用程序提供凭证，而无需您的容器组（pod）与 Amazon STS 通信或使用 STS API 配额。	在 IRSA 中，容器组（pod）中使用 Amazon SDK 编写的应用程序会使用令牌在 Amazon Security Token Service（STS）上调用 `AssumeRoleWithWebIdentity` API。根据您在 Amazon SDK 上的代码逻辑，您的代码可能会对 Amazon STS 进行不必要的调用并收到节流错误。

属性	EKS 容器组身份	IRSA
角色可重用性	EKS 容器组身份提供的 Amazon STS 临时凭证包括角色会话标签，例如集群名称、命名空间、服务账户名称。角色会话标签使管理员能够创建单个 IAM 角色，该角色可用于具有不同有效权限的多个服务账户，方法是允许根据附加到资源的 Amazon 标签来访问资源，也称为基于属性的访问权限控制（ABAC）。有关更多信息，请参阅 the section called “授予容器组（pod）访问权限”。	不支持 Amazon STS 会话标签。您可以在集群之间重用角色，但每个容器组都会获得该角色的所有权限。
支持的环境	EKS 容器组身份仅在 Amazon EKS 上可用。	IRSA 可用于 Amazon EKS、Amazon EKS Anywhere、Amazon 上的 Red Hat OpenShift 服务，以及 Amazon EC2 实例上的自主管理型 Kubernetes 集群。
支持的 EKS 版本	所有受支持的 EKS 集群版本。有关特定平台版本，请参阅 the section called “EKS 容器组身份集群版本”。	所有受支持的 EKS 集群版本。

服务账户的 IAM 角色

容器组（pod）的容器中的应用程序可以使用 Amazon SDK 或 Amazon CLI 以向使用 Amazon Identity and Access Management（IAM）权限的 Amazon 服务 发出 API 请求。应用程序必须通过 Amazon 凭证签署 Amazon API 请求。服务账户的 IAM 角色（IRSA）可用于管理供应用程序使用的凭证，这与 Amazon EC2 实例配置文件为 Amazon EC2 实例提供凭证的方式类似。您可以将 IAM 角色与 Kubernetes 服务账户关联并配置容器组（pod）来使用服务账户，而不是创建 Amazon 凭证并将其分

配到容器或使用 Amazon EC2 实例的角色。您不能将服务账户的 IAM 角色用于 Amazon Outposts 上的 Amazon EKS 本地集群。

服务账户的 IAM 角色提供下列优势：

- 最低权限 – 您可以将 IAM 权限范围限定到服务账户，并且只有使用该服务账户的容器组（pod）可以访问这些权限。此功能还消除了对 `kiam` 或 `kube2iam` 等第三方解决方案的需求。
- 凭证隔离：当对 Amazon EC2 实例元数据服务（IMDS）的访问受到限制时，容器组的容器只能检索与该容器所使用服务账户关联的 IAM 角色的凭证。容器永远无法访问其他容器组（pod）中其他容器所使用的凭证。如果 IMDS 不受限制，则容器组的容器还可以访问 Amazon EKS 节点 IAM 角色，并且容器可能能够访问同一节点上其他容器组 IAM 角色的凭证。有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

> **Note**
>
> 配置为 `hostNetwork: true` 的容器组将始终具有 IMDS 访问权限，但 Amazon SDK 和 CLI 在启用后将使用 IRSA 凭证。

- 可审核性 – 可通过 Amazon CloudTrail 进行访问和事件日志记录以帮助确保可追溯性审核。

> **Important**
>
> 容器并非安全边界，使用服务账户的 IAM 角色也不会改变这一点。分配给同一个节点的容器组将共享一个内核，并可能共享其他资源，具体视您的容器组配置而定。虽然在不同节点上运行的容器组将在计算层被隔离，但有些节点应用程序在 Kubernetes API 中拥有超出单个实例范围的额外权限。一些示例包括 `kubelet`、`kube-proxy`、CSI 存储驱动程序或您自己的 Kubernetes 应用程序。

通过完成下列步骤为服务账户启用 IAM 角色：

1. 为集群创建 IAM OIDC 提供者 – 您只需为集群完成一次此操作。

> **Note**
>
> 如果启用了 EKS VPC 端点，则无法从该 VPC 内部访问 EKS OIDC 服务端点。因此，您在 VPC 中使用 `eksctl` 创建 OIDC 提供者等操作将不起作用，并且在尝试请求 `https://oidc.eks.region.amazonaws.com` 时将导致超时。错误消息示例如下：
>
> ```
> server cant find oidc.eks.region.amazonaws.com: NXDOMAIN
> ```
>
> 要完成此步骤，您可以在 VPC 外部运行该命令，例如在 Amazon CloudShell 中或在连接到互联网的计算机上。您也可在 VPC 中创建水平分割条件解析器（例如 Route 53 Resolver），以便为 OIDC 发布者 URL 使用不同的解析器，而不对其使用 VPC DNS。有关 CoreDNS 中条件转发的示例，请参阅 GitHub 上的 Amazon EKS feature request。

2. 将 IAM 角色分配给 Kubernetes 服务账户 – 针对您希望应用程序拥有的每组唯一权限完成此步骤。
3. 将容器组（pod）配置为使用 Kubernetes 服务账户 – 为需要访问 Amazon 服务的每个 容器组（pod）完成此过程。
4. 将 IRSA 与 Amazon SDK 结合使用 – 确认工作负载使用支持版本的 Amazon SDK，并且工作负载使用默认凭证链。

IAM、Kubernetes 和 OpenID Connect（OIDC）背景信息

2014 年，Amazon Identity and Access Management 使用 OpenID Connect（OIDC）增加了对联合身份验证的支持。此功能允许您通过支持的身份提供商对 Amazon API 调用进行身份验证，并获得有效的 OIDC JSON Web 令牌 (JWT)。您可以将此令牌传递到 Amazon STS `AssumeRoleWithWebIdentity` API 操作并接收 IAM 临时角色凭证。您可以使用这些凭证与任意 Amazon 服务交互，包括 Amazon S3 和 DynamoDB。

每个 JWT 令牌均由签名密钥对签名。密钥由 Amazon EKS 管理的 OIDC 提供者提供，私钥每 7 天轮换一次。Amazon EKS 会保留公钥直至其过期。如果您连接外部 OIDC 客户端，请注意您需要在公钥过期之前刷新签名密钥。了解如何获取签名密钥以验证 OIDC 令牌。

Kubernetes 长期以来将服务账户用作其内部身份系统。Pod 可以使用自动装载的令牌（这是非 OIDC JWT，只有 Kubernetes API 服务器可以验证）进行 Kubernetes API 服务器的身份验证。这些旧服务账户令牌不会过期，轮换签名密钥是一个困难的过程。在 Kubernetes `1.12` 版本中，添加了对新 `ProjectedServiceAccountToken` 功能的支持。此功能是 OIDC JSON Web 令牌，包含了服务账户身份，且支持可配置的受众。

Amazon EKS 为包含 ProjectedServiceAccountToken JSON Web 令牌的签名密钥的每个集群托管公有 OIDC 发现端点，这样 IAM 等外部系统就可以验证和接收 Kubernetes 颁发的 OIDC 令牌。

为集群创建 IAM OIDC 提供商

您的集群具有与其关联的 OpenID Connect（OIDC）发布者 URL。要将 Amazon Identity and Access Management（IAM）角色用于服务账户，您集群的 OIDC 发布者 URL 必须存在 IAM OIDC 提供者。

先决条件

- 现有 Amazon EKS 集群。要部署一个角色，请参阅开始使用。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 2.12.3 或更高版本，或版本 1.27.160 或更高版本。要查看当前版本，请使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 您的设备或 Amazon CloudShell 上安装了 kubectl 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 1.29，则可以将 kubectl 的 1.28、1.29 或 1.30 版本与之配合使用。要安装或升级 kubectl，请参阅 the section called “设置 kubectl 和 eksctl”。
- 包含集群配置的现有 kubectl config 文件。要创建 kubectl config 文件，请参阅 the section called “使用 kubectl 访问集群”。

您可以使用 eksctl 或 Amazon Web Services 管理控制台为集群创建 IAM OIDC 提供者。

创建 OIDC 提供者（eksctl）

1. 您的设备或 Amazon CloudShell 上安装 0.215.0 版或更高版本的 eksctl 命令行工具。要安装或更新 eksctl，请参阅 eksctl 文档中的 Installation。
2. 确定集群的 OIDC 发布者 ID。

   检索集群的 OIDC 发布者 ID 并将其存储在变量中。将 <my-cluster> 替换为您自己的值。

```
cluster_name=<my-cluster>
oidc_id=$(aws eks describe-cluster --name $cluster_name --query
 "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

```
echo $oidc_id
```

3. 确定账户中是否已存在具有集群发布者 ID 的 IAM OIDC 提供者。

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

如果返回了输出，则表示集群已经有 IAM OIDC 提供者，您可以跳过下一步。如果没有返回输出，则您必须为集群创建 IAM OIDC 提供者。

4. 使用以下命令为您的集群创建 IAM OIDC 身份提供商。

```
eksctl utils associate-iam-oidc-provider --cluster $cluster_name --approve
```

> **Note**
>
> 如果启用了 EKS VPC 端点，则无法从该 VPC 内部访问 EKS OIDC 服务端点。因此，您在 VPC 中使用 `eksctl` 创建 OIDC 提供者等操作将不起作用，并将导致超时。错误消息示例如下：
>
> ```
> ** server cant find oidc.eks.<region-code>.amazonaws.com: NXDOMAIN
> ```

要完成此步骤，您可以在 VPC 外部运行该命令，例如在 Amazon CloudShell 中或在连接到互联网的计算机上。您也可在 VPC 中创建水平分割条件解析器（例如 Route 53 Resolver），以便为 OIDC 发布者 URL 使用不同的解析器，而不对其使用 VPC DNS。有关 CoreDNS 中条件转发的示例，请参阅 GitHub 上的 Amazon EKS feature request。

创建 OIDC 提供者（Amazon 控制台）

1. 打开 Amazon EKS 控制台。
2. 在左侧窗格中，选择 Clusters（集群），然后在 Clusters（集群）页面上选择集群的名称。
3. 在 Overview（概述）选项卡上的 Details（详细信息）部分中，记下 OpenID Connect provider URL（OpenID Connect 提供商 URL）的值。
4. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
5. 请在左侧导航窗格中，选择 Access management（访问管理）下方的 Identity Providers（标识提供程序）。如果列出的 Provider（提供商）与集群的 URL 匹配，那么您的集群已经有了提供商。如果未列出与集群 URL 匹配的提供者，则必须创建一个提提供者。

6. 要创建提供商，请选择 Add Provider（添加提供商）。

7. 对于提供者类型，选择 OpenID Connect。

8. 对于提供者 URL，输入集群的 OIDC 提供者 URL。

9. 对于受众,，输入 `sts.amazonaws.com`。

10.（可选）添加任何标签，例如用于标识哪个集群用于该提供商的标签。

11选择添加提供程序。

下一步： the section called “分配 IAM 角色”

为 Kubernetes 服务账户分配 IAM 角色

本主题介绍如何配置 Kubernetes 服务账户，以担任 Amazon Identity and Access Management（IAM）角色。然后，配置为使用服务账户的任何容器组（pod）都可以访问该角色有权访问的任何 Amazon 服务。

先决条件

- 现有集群。如果您没有，可以按照开始使用 中的指南之一创建一个。
- 您的集群的现有 IAM OpenID Connect (OIDC) 提供商。要了解您是否已拥有一个（IAM）角色或如何创建一个（IAM）角色，请参阅 the section called “IAM OIDC 提供者”。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 包含集群配置的现有 `kubectl config` 文件。要创建 `kubectl config` 文件，请参阅 the section called “使用 kubectl 访问集群”。

步骤 1：创建 IAM 策略

要将现有 IAM 策略关联到您的 IAM 角色，请跳至下一步。

1. 创建一个 IAM 策略。您可以创建自己的策略，也可以复制已授予您部分所需权限的 Amazon 托管策略，并根据您的特定要求对其进行自定义。有关更多信息，请参阅《IAM 用户指南》中的创建 IAM 策略。
2. 创建一个文件，其中包含您希望容器组（pod）访问的 Amazon 服务所需的权限。有关所有 Amazon 服务的所有操作的列表，请参阅服务授权参考。

   您可以运行以下命令创建一个示例策略文件，以实现对 Amazon S3 存储桶的只读访问权限。您可以选择将配置信息或引导脚本存储在此存储桶中，并且容器组（pod）中的容器可以从存储桶读取文件并将其加载到应用程序中。如果您要创建此示例策略，请将以下内容复制到您的设备。将 *my-pod-secrets-bucket* 替换为您的存储桶名称并运行该命令。

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "s3:GetObject",
               "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
           }
       ]
   }
   ```

3. 创建 IAM 策略。

   ```
   aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
   ```

步骤 2：创建和关联 IAM 角色

创建 IAM 角色并将其与 Kubernetes 服务账户关联。您可以使用 `eksctl` 或 Amazon CLI。

创建和关联角色（eksctl）

此 `eksctl` 命令会在指定的命名空间中创建一个 Kubernetes 服务账户，创建具有指定名称的 IAM 角色（如果其不存在），将现有的 IAM 策略 ARN 附加到该角色，并使用 IAM 角色 ARN 为该服务账户添加注释。请务必使用您的具体值替换此命令中的示例占位符值。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的 Installation。

```
eksctl create iamserviceaccount --name my-service-account --namespace default --cluster
 my-cluster --role-name my-role \
    --attach-policy-arn arn:aws-cn:iam::111122223333:policy/my-policy --approve
```

> **Important**
>
> 如果角色或服务账户已经存在，则之前的命令可能会失败。在这些情况下，eksctl 提供不同的选项供您使用。有关更多信息，请运行 eksctl create iamserviceaccount --help。

创建和关联角色（Amazon CLI）

如果您有要担任 IAM 角色的现有 Kubernetes 服务账户，则可跳过此步骤。

1. 创建一个 Kubernetes 服务账户。将以下内容复制到您的设备。将 *my-service-account* 替换为所需的名称，如有必要，将 *default* 替换为其他命名空间。如更改 *default*，则命名空间必须已经存在。

   ```
   cat >my-service-account.yaml <<EOF
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: my-service-account
     namespace: default
   EOF
   kubectl apply -f my-service-account.yaml
   ```

2. 使用以下命令将 Amazon 账户 ID 设置为环境变量。

   ```
   account_id=$(aws sts get-caller-identity --query "Account" --output text)
   ```

3. 使用以下命令将集群的 OIDC 身份提供者设置为环境变量。将 *my-cluster* 替换为您的集群的名称。

   ```
   oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --
   query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
   ```

4. 为服务账户的命名空间和名称设置变量。将 *my-service-account* 替换为要担任角色的 Kubernetes 服务账户。将 *default* 替换为服务账户的命名空间。

```
export namespace=default
export service_account=my-service-account
```

5. 运行以下命令为 IAM 角色创建信任策略文件。若要允许命名空间内的所有服务账户使用该角色，请将以下内容复制到您的设备。将 *StringEquals* 替换为 StringLike，并将 *$service_account* 替换为 *。您可在 StringEquals 和 StringLike 条件中添加多个条目，以允许多个服务账户或命名空间分派角色。要允许其它 Amazon 账户中的角色（而不是您的集群所在账户中的角色）分派该角色户，请参阅the section called “跨账户 IAM”获取更多信息。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
```

6. 创建角色。将 *my-role* 替换为您的 IAM 角色的名称，并将 *my-role-description* 替换为您的角色的描述。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-
relationship.json --description "my-role-description"
```

7. 将 IAM 策略附加到您的角色。将 *my-role* 替换为您的 IAM 角色的名称，并将 *my-policy* 替换为您创建的现有策略的名称。

```
aws iam attach-role-policy --role-name my-role --policy-arn=arn:aws-cn:iam::
$account_id:policy/my-policy
```

8. 请使用要让服务账户分派的 IAM 角色的 Amazon 资源名称（ARN）注释您的服务账户。将 *my-role* 替换为您的现有 IAM 角色的名称。假设您在上一步中允许其它 Amazon 账户中的角色（而不是您的集群所在账户中的角色）分派该角色户。那么，请确保指定其它账户中的 Amazon 账户和角色。有关更多信息，请参阅the section called “跨账户 IAM”。

```
kubectl annotate serviceaccount -n $namespace $service_account eks.amazonaws.com/
role-arn=arn:aws-cn:iam::$account_id:role/my-role
```

9. （可选）为服务账户配置 Amazon Security Token Service 端点。Amazon 建议使用区域级 Amazon STS 端点而不是全局端点。这可以减少延迟，提供内置冗余并提高会话令牌的有效性。

步骤 3：确认配置

1. 确认 IAM 角色的信任策略配置正确。

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

示例输出如下。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:default:my-service-
account",
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
```

```
}
```

2. 确认您在上一步中附加到角色的策略已附加到该角色。

```
aws iam list-attached-role-policies --role-name my-role --query
 "AttachedPolicies[].PolicyArn" --output text
```

示例输出如下。

```
            arn:aws-cn:iam::111122223333:policy/my-policy
```

3. 设置一个变量以存储要使用策略的 Amazon 资源名称（ARN）。将 *my-policy* 替换为要确认其权限的策略的名称。

```
export policy_arn=arn:aws-cn:iam::111122223333:policy/my-policy
```

4. 查看该策略的默认版本。

```
aws iam get-policy --policy-arn $policy_arn
```

示例输出如下。

```
{
    "Policy": {
        "PolicyName": "my-policy",
        "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
        "Arn": "arn:aws-cn:iam::111122223333:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        [...]
    }
}
```

5. 查看策略内容以确保该策略包含容器组（pod）所需的所有权限。如有必要，将以下命令中的 *1* 替换为上一个输出中返回的版本。

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

示例输出如下。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
```

如果您在上一步中创建了示例策略，则您的输出与示例相同。如果您创建了其他策略，则与##内容不同。

6. 确认已使用角色为 Kubernetes 服务账户添加注释。

```
kubectl describe serviceaccount my-service-account -n default
```

示例输出如下。

```
Name:                my-service-account
Namespace:           default
Annotations:         eks.amazonaws.com/role-arn: arn:aws-cn:iam::111122223333:role/
my-role
Image pull secrets:  <none>
Mountable secrets:   my-service-account-token-qqjfl
Tokens:              my-service-account-token-qqjfl
[...]
```

后续步骤

- the section called “分配给容器组（pod）”

配置容器组（pod）使用 Kubernetes 服务账户

如果容器组（pod）需要访问 Amazon 服务，则必须配置它来使用 Kubernetes 服务账户。服务账户必须关联到有权访问 Amazon 服务的 Amazon Identity and Access Management（IAM）角色。

- 现有集群。如果还没有，可以使用开始使用 中的指南之一创建一个。

- 您的集群的现有 IAM OpenID Connect (OIDC) 提供商。要了解您是否已拥有一个（IAM）角色或如何创建一个（IAM）角色，请参阅 the section called “IAM OIDC 提供者”。
- 与 IAM 角色关联的现有 Kubernetes 服务账户。必须使用 IAM 角色的 Amazon 资源名称（ARN）注释服务账户。该角色必须具有关联的 IAM 策略，其中包含您希望容器组（pod）必须具有的权限，以便使用 Amazon 服务。有关如何创建和配置服务账户和角色的更多信息，请参阅 the section called “分配 IAM 角色”。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 包含集群配置的现有 `kubectl config` 文件。要创建 `kubectl config` 文件，请参阅 the section called “使用 kubectl 访问集群”。

1. 使用以下命令创建部署清单，您可以使用该部署清单部署容器组（pod）来确认配置。将示例值替换为您自己的值。

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
```

```
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. 将清单部署到集群。

```
kubectl apply -f my-deployment.yaml
```

3. 确认容器组（pod）具有所需的环境变量。

   a. 按上一步部署后，查看部署的容器组（pod）。

   ```
   kubectl get pods | grep my-app
   ```

   示例输出如下。

   ```
   my-app-6f4dfff6cb-76cv9   1/1     Running   0          3m28s
   ```

   b. 查看容器组（pod）使用的 IAM 角色的 ARN。

   ```
   kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_ROLE_ARN:
   ```

   示例输出如下。

   ```
   AWS_ROLE_ARN:                 arn:aws-cn:iam::111122223333:role/my-role
   ```

   角色 ARN 必须与您用于注释现有服务账户的角色 ARN 匹配。有注释服务账户的更多信息，请参阅 the section called “分配 IAM 角色”。

   c. 确认容器组（pod）有 Web 身份令牌文件挂载。

   ```
   kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_WEB_IDENTITY_TOKEN_FILE:
   ```

   示例输出如下。

   ```
   AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/serviceaccount/
   token
   ```

   kubelet 代表容器组（pod）请求并存储令牌。默认情况下，如果令牌早于其总生存时间的 80%，或者令牌大于 24 小时，则 kubelet 会刷新令牌。您可以使用容器组（pod）规范中的

设置修改任何账户（默认服务账户除外）的到期期限。有关更多信息，请参阅 Kubernetes 文档中的服务账户令牌卷预测。

集群上的 Amazon EKS 容器组身份 Webhook 监控使用具有以下注释的服务账户：

```
eks.amazonaws.com/role-arn: arn:aws-cn:iam::111122223333:role/my-role
```

Webhook 将之前的环境变量应用于这些容器组（pod）。您的集群无需使用 Webhook 来配置环境变量和令牌文件挂载。您可以手动配置容器组（pod）来包含这些环境变量。Amazon 开发工具包支持的版本首先在凭证链提供商中查找这些环境变量。角色凭证用于满足这些标准的容器组（pod）。

4. 确认您的容器组（pod）可以使用您在附加到角色的 IAM 策略中分配的权限与 Amazon 服务进行交互。

> **Note**
>
> 当容器组（pod）使用来自与服务账户关联的 IAM 角色的 Amazon 凭证时，该容器组（pod）中的 Amazon CLI 或其他 SDK 包仅使用该角色提供的凭证。如果您不限制对提供给 Amazon EKS 节点 IAM 角色的凭证的访问，容器组（pod）仍然可以访问这些凭证。有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

如果您的容器组（pod）无法按预期与服务进行交互，请完成以下步骤，确认所有配置都正确。

a. 确认您的容器组（pod）使用支持通过 OpenID Connect Web 身份令牌文件担任 IAM 角色的 Amazon SDK 版本。有关更多信息，请参阅 the section called “受支持的 SDK”。

b. 确认部署正在使用服务账户。

```
kubectl describe deployment my-app | grep "Service Account"
```

示例输出如下。

```
Service Account:  my-service-account
```

c. 如果您的容器组（pod）仍然无法访问服务，请查看为 Kubernetes 服务账户分配 IAM 角色中描述的步骤，确认角色和服务账户的配置都正确。

为服务账户配置 Amazon Security Token Service 端点

如果将 Kubernetes 服务账户与服务账户的 IAM 角色结合使用，则可配置服务账户使用的 Amazon Security Token Service 端点类型。

Amazon 建议使用区域 Amazon STS 端点而不是全局端点。这可以减少延迟，提供内置冗余并提高会话令牌的有效性。Amazon Security Token Service 必须在容器组（pod）运行的 Amazon 区域处于活动状态。此外，您的应用程序必须内置冗余功能，以便在该 Amazon 区域的服务出现故障时选择其它 Amazon 区域。有关更多信息，请参阅《IAM 用户指南》中的在 Amazon 区域中管理 Amazon STS。

- 现有集群。如果还没有，可以使用 开始使用 中的指南之一创建一个。
- 集群的现有 IAM OIDC 提供商。有关更多信息，请参阅 the section called “IAM OIDC 提供者”。
- 现有的 Kubernetes 服务账户配置为与适用于服务账户的 Amazon EKS IAM 功能结合使用。

以下示例全部使用 Amazon VPC CNI 插件所使用的 aws-node Kubernetes 服务账户。可将*###*替换为您自己的服务账户、容器组（pod）、命名空间和其他资源。

1. 选择一个使用要更改其端点的服务账户的容器组（pod）。确定容器组（pod）在其中运行的 Amazon 区域。将 *aws-node-6mfgv* 替换为容器组（pod）的名称，并将 *kube-system* 替换为容器组（pod）的命名空间。

   ```
   kubectl describe pod aws-node-6mfgv -n kube-system |grep Node:
   ```

   示例输出如下。

   ```
   ip-192-168-79-166.us-west-2/192.168.79.166
   ```

   在之前的输出中，容器组（pod）在 us-west-2 Amazon 区域中的节点上运行。
2. 确定容器组（pod）的服务账户使用的端点类型。

   ```
   kubectl describe pod aws-node-6mfgv -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
   ```

   示例输出如下。

   ```
   AWS_STS_REGIONAL_ENDPOINTS: regional
   ```

如果当前端点是全局端点，则输出中将返回 `global`。如果没有返回输出，则默认端点类型正在使用中且尚未被覆盖。

3. 如果您的集群或平台版本与表中列出的版本相同或晚于该版本，则可以使用以下命令之一将服务账户使用的端点类型从默认类型更改为其他类型。将 *aws-node* 替换为您的服务账户的名称，并将 *kube-system* 替换为您的服务账户的命名空间。

   - 如果您的默认或当前端点类型为全局端点类型，并且您想将其更改为区域性端点：

     ```
     kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=true
     ```

     如果您使用服务账户的 IAM 角色，在容器组（pod）的容器中运行的应用程序中生成预签名 S3 URL，则区域端点的 URL 格式与以下示例类似：

     ```
     https://bucket.s3.us-west-2.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
     ```

   - 如果您的默认或当前端点类型为区域性端点类型，并且您想将其更改为全局性端点：

     ```
     kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=false
     ```

     如果您的应用程序明确向 Amazon STS 全局端点提出请求，且您不会覆盖在 Amazon EKS 集群中使用区域端点的默认行为，则请求将失败，并出现错误。有关更多信息，请参阅 the section called “容器组（pod）容器收到以下错误：An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region”。

     如果您使用服务账户的 IAM 角色，在容器组（pod）的容器中运行的应用程序中生成预签名 S3 URL，则全局端点的 URL 格式与以下示例类似：

     ```
     https://bucket.s3.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
     ```

   如果您的自动化需要特定格式的预签名 URL，或者如果您的应用程序或使用预签名 URL 的下游依赖关系对目标 Amazon 区域有期望，则进行必要的更改以使用适当的 Amazon STS 端点。

4. 删除并重新创建任何与服务账户关联的现有容器组（pod），以应用凭证环境变量。变更 Webhook 不会将其应用到已经在运行的容器组（pod）。您可以将 *Pods*、*kube-system* 和 *-l k8s-app=aws-node* 替换为设置注释的容器组（pod）信息。

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

5. 确认容器组（pod）已全部重新启动。

```
kubectl get Pods -n kube-system -l k8s-app=aws-node
```

6. 查看其中一个容器组（pod）的环境变量。验证 AWS_STS_REGIONAL_ENDPOINTS 值是您在上一步中将其设置的值。

```
kubectl describe pod aws-node-kzbtr -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

示例输出如下。

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

使用 IRSA 对另一个账户进行身份验证

您可以通过从其它账户的集群创建身份提供者或者使用链接的 AssumeRole 操作，配置跨账户 IAM 权限。在以下示例中，账户 A 拥有一个支持服务账户的 IAM 角色的 Amazon EKS 集群。在该集群上运行的容器组（pod）需要承担来自账户 B 的 IAM 权限。

Example 从其它账户的集群创建身份提供者

Example

在此示例中，账户 A 从其集群向账户 B 提供 OpenID Connect（OIDC）颁发者 URL。账户 B 遵循为集群创建 IAM OIDC 提供者和the section called “分配 IAM 角色”中的说明，使用来自账户 A 集群的 OIDC 发布者 URL。然后，集群管理员注释账户 A 的集群中的服务账户以使用来自账户 B 的角色（*444455556666*）。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws-cn:iam::444455556666:role/account-b-role
```

Example 使用链接 **`AssumeRole`** 操作

Example

在此示例中，账户 B 创建一个 IAM 策略，其中包括授予账户 A 集群中容器组（pod）的权限。账户 B（*444455556666*）将该策略附加到具有信任关系的 IAM 角色，允许对账户 A（*111122223333*）的 `AssumeRole` 权限。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

账户 A 创建具有信任策略的角色，该策略从使用集群的 OIDC 颁发者地址创建的身份提供者处获取凭证。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
        }
      }
    }
```

```
  ]
}
```

账户 A 将策略附加到具有以下权限的角色，以代入账户 B 创建的角色。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::444455556666:role/account-b-role"
        }
    ]
}
```

容器组（pod）的应用程序代码使用两个配置文件担任账户 B 的角色：`account_b_role` 和 `account_a_role`。`account_b_role` 配置文件使用 `account_a_role` 配置文件作为其源。对于 Amazon CLI，`~/.aws/config` 文件类似于以下内容。

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws-cn:iam::444455556666:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws-cn:iam::111122223333:role/account-a-role
```

要为其它 Amazon SDK 指定链接的配置文件，请参考所使用 SDK 的文档。有关更多信息，请参阅用于在 Amazon 上进行构建的工具。

将 IRSA 与 Amazon SDK 结合使用

使用凭证

要使用服务账户的 IAM 角色（IRSA）凭证，代码可以使用任何 Amazon SDK 为具有 SDK 的 Amazon 服务创建客户端。默认情况下，SDK 会在一系列位置中搜索要使用的 Amazon Identity and Access Management 凭证。如果您在创建客户端或者初始化 SDK 时未指定凭证提供程序，则将使用服务账户凭证的 IAM 角色。

由于服务账户的 IAM 角色已添加为默认凭证链中的一个步骤，因此该操作可行。如果您的工作负载当前使用凭证链中较早的证书，则即使您为同一工作负载的服务账户配置了 IAM 角色，这些凭证也将继续使用。

SDK 使用 `AssumeRoleWithWebIdentity` 操作自动将服务账户 OIDC 令牌交换为 Amazon Security Token Service 中的临时凭证。Amazon EKS 和此 SDK 操作会在临时凭证到期前进行续订，从而继续轮换临时凭证。

在使用服务账户的 IAM 角色时，容器组（pod）中的容器必须使用支持通过 OpenID Connect Web 身份令牌文件担任 IAM 角色的 Amazon SDK 版本。请确保为您的 Amazon SDK 使用以下版本或更高版本：

- Java（版本 2）– 2.10.11
- Java – 1.12.782
- Amazon SDK for Go v1 – 1.23.13
- Amazon SDK for Go v2 – 所有版本均支持
- Python (Boto3) – 1.9.220
- Python (botocore) – 1.12.200
- Amazon CLI – 1.16.232
- 节点：2.525.0 和 3.27.0
- Ruby – 3.58.0
- C++ – 1.7.174
- .NET：3.3.659.1，您必须还包括 `AWSSDK.SecurityToken`。
- PHP – 3.110.7

许多流行的 Kubernetes 附加组件（例如 Cluster Autoscaler、使用 Amazon 负载均衡器控制器的路由互联网流量和适用于 Kubernetes 的 Amazon VPC CNI 插件）支持服务账户的 IAM 角色。

为了确保您使用的是受支持的 SDK，请在构建容器时按照用于在 Amazon 上进行构建的工具中针对您的首选 SDK 的安装说明操作。

注意事项

Java

使用 Java 时，必须在类路径中包括 `sts` 模块。有关更多信息，请参阅 Java SDK 文档中的 WebIdentityTokenFileCredentialsProvider。

获取签名密钥以验证 OIDC 令牌

Kubernetes 向每个 Kubernetes 服务账户发放一个 `ProjectedServiceAccountToken`。此令牌是一个 OIDC 令牌，更是一种 JSON Web 令牌（JWT）。Amazon EKS 为包含令牌的签名密钥的每个集群托管公有 OIDC 端点，这样外部系统就可以验证。

您需要获取 OIDC 公有签名密钥 [又称 JSON Web 密钥集（简称 JWKS）] 才能验证 `ProjectedServiceAccountToken`。在应用程序中使用这些密钥来验证令牌。例如，您可以使用 PyJWT Python 库来验证使用这些密钥的令牌。有关 `ProjectedServiceAccountToken` 的更多信息，请参阅 the section called “IAM、Kubernetes 和 OpenID Connect（OIDC）背景信息”。

先决条件

- 适用于集群的现有 Amazon Identity and Access Management（IAM）OpenID Connect（OIDC）提供者。要确定您是否已经拥有一个或是否要创建一个，请参阅the section called “IAM OIDC 提供者”。
- Amazon CLI – 与 Amazon 服务（包括 Amazon EKS）结合使用的命令行工具。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的安装。在安装 Amazon CLI 后，建议您还要对其进行配置。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的使用 aws configure 快速配置。

过程

1. 使用 Amazon CLI 检索适用于 Amazon EKS 集群的 OIDC URL。

```
$ aws eks describe-cluster --name my-cluster --query 'cluster.identity.oidc.issuer'
"https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE"
```

2. 使用 curl 或类似工具检索公有签名密钥。结果是 JSON Web 密钥集（JWKS）。

> **Important**
>
> Amazon EKS 会限制对 OIDC 端点的调用。您应该缓存公共签名密钥。请遵守响应中包含的 `cache-control` 标题。

> **Important**
>
> Amazon EKS 每七天轮换一次 OIDC 签名密钥。

```
$ curl https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE/keys
{"keys":
[{"kty":"RSA","kid":"2284XXXX4a40","use":"sig","alg":"RS256","n":"wklbXXXXMVfQ","e":"AQAB"}]
```

了解 EKS 容器组身份如何向容器组（pod）授予对 Amazon 服务的访问权限

容器组（pod）的容器中的应用程序可以使用 Amazon SDK 或 Amazon CLI 以向使用 Amazon Identity and Access Management（IAM）权限的 Amazon 服务 发出 API 请求。应用程序必须通过 Amazon 凭证签署 Amazon API 请求。

Amazon EKS 容器组身份提供管理应用程序凭证的功能，类似于 Amazon EC2 实例配置文件为 Amazon EC2 实例提供凭证的方式。您可以将 IAM 角色与 Kubernetes 服务账户关联并配置容器组（pod）来使用服务账户，而不是创建 Amazon 凭证并将其分配到容器或使用 Amazon EC2 实例的角色。

每个 EKS 容器组身份关联都将角色映射到指定集群命名空间中的服务账户。如果您在多个集群中使用相同的应用程序，则可以在每个集群中进行相同的关联，而无需修改角色的信任策略。

如果某个容器组使用具有关联的服务账户，Amazon EKS 会在容器组的容器中设置环境变量。环境变量配置 Amazon SDK（包括 Amazon CLI）使用 EKS 容器组身份凭证。

EKS 容器组身份的优势

EKS 容器组身份具有以下优势：

- 最低权限 – 您可以将 IAM 权限范围限定到服务账户，并且只有使用该服务账户的容器组（pod）可以访问这些权限。此功能还消除了对 `kiam` 或 `kube2iam` 等第三方解决方案的需求。
- 凭证隔离：当对 Amazon EC2 实例元数据服务（IMDS）的访问受到限制时，容器组的容器只能检索与该容器所使用服务账户关联的 IAM 角色的凭证。容器永远无法访问其他容器组（pod）中其他容器所使用的凭证。如果 IMDS 不受限制，则容器组的容器还可以访问 Amazon EKS 节点 IAM 角色，并且容器可能能够访问同一节点上其他容器组 IAM 角色的凭证。有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

> **Note**
>
> 配置为 `hostNetwork: true` 的容器组将始终具有 IMDS 访问权限，但 Amazon SDK 和 CLI 在启用后将使用容器组身份凭证。

- 可审计性 – 可通过 Amazon CloudTrail 进行访问和事件日志记录，帮助确保追溯性审计。

> **Important**
>
> 容器并非安全边界，使用任务容器组身份也不能改变这一点。分配给同一个节点的容器组将共享一个内核，并可能共享其他资源，具体视您的容器组配置而定。虽然在不同节点上运行的容器组将在计算层被隔离，但有些节点应用程序在 Kubernetes API 中拥有超出单个实例范围的额外权限。一些示例包括 `kubelet`、`kube-proxy`、CSI 存储驱动程序或您自己的 Kubernetes 应用程序。

EKS 容器组身份是一种比 the section called “包含 IRSA 的凭证” 更简单的方法，因为此方法不使用 OIDC 身份提供者。EKS 容器组身份具有以下增强功能：

- 独立操作：在许多组织中，不同的团队负责创建 OIDC 身份提供者，而不是管理 Kubernetes 集群。EKS 容器组身份有明确的职责分工，EKS 容器组身份关联的所有配置都在 Amazon EKS 中完成，而 IAM 权限的所有配置都在 IAM 中完成。
- 可重用性：EKS 容器组身份使用单个 IAM 主体，而不是服务账户 IAM 角色使用的每个集群的单独主体。IAM 管理员将以下主体添加到任何角色的信任策略中，使其可供 EKS 容器组身份使用。

```
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            }
```

- 可扩展性：每组临时凭证均由 EKS 容器组身份中的 EKS Auth 服务使用，而不是由每个容器组（pod）中运行的每个 Amazon SDK 使用。然后，在每个节点上运行的 Amazon EKS 容器组身份代理向 SDK 发放凭证。因此，每个节点的负载减少至一次，而且不会在每个容器组中重复。有关该过程的更多详细信息，请参阅 the section called “工作方式”。

有关比较两种替代方案的更多信息，请参阅 the section called “Amazon 工作负载访问权限 ”。

EKS 容器组身份设置概述

完成以下过程，打开 EKS 容器组身份：

1. the section called “设置代理” – 对于每个集群，您只需完成一次此步骤。如果您的集群启用了 EKS 自动模式，则无需完成此步骤。
2. the section called “分配 IAM 角色” – 针对您希望应用程序拥有的每组唯一权限完成此步骤。
3. the section called “容器组（pod）服务账户”：为需要访问 Amazon 服务的每个容器组（pod）完成此步骤。
4. the section called “受支持的 SDK” – 确认工作负载使用支持版本的 Amazon SDK，并且工作负载使用默认凭证链。

限制

- 每个集群最多支持 5000 个 EKS 容器组身份关联，用于将 IAM 角色映射到 Kubernetes 服务账户。

注意事项

- IAM 角色关联：集群中的每个 Kubernetes 服务账户都可以与集群相同的 Amazon 账户中的一个 IAM 角色相关联。要更改该角色，请编辑 EKS 容器组身份关联。要进行跨账户访问，请使用 IAM 角色向该角色委派访问权限。要了解更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色委托跨 Amazon 账户的访问权限。
- EKS 容器组身份代理：使用 EKS 容器组身份需要 EKS 容器组身份代理。该代理作为 Kubernetes `DaemonSet` 在集群节点上运行，仅向同一节点上的容器组提供凭证。其使用节点的 `hostNetwork`，占用链路本地地址上的端口 `80` 和 `2703`（IPv4 为 `169.254.170.23`，IPv6 为 `[fd00:ec2::23]`）。如果您的集群中禁用 IPv6，则请为容器组身份代理禁用 IPv6。要了解更多信息，请参阅在 EKS 容器组身份代理中禁用 IPv6。
- 最终一致性：EKS 容器组身份关联最终保持一致，API 调用后可能会有几秒钟的延迟。避免在关键的高可用性代码路径中创建或更新关联。相反，应在单独的、不太频繁的初始化或设置例程中执行这些操作。要了解更多信息，请参阅《EKS Best Practices Guide》中的 Security Groups Per Pod。
- 代理和安全组注意事项：对于使用代理的容器组，请将 `169.254.170.23`（IPv4）和 `[fd00:ec2::23]`（IPv6）添加到 `no_proxy/NO_PROXY` 环境变量，以防止对 EKS 容器组身份代理的请求失败。如果容器组的安全组与 Amazon VPC CNI 一起使用，则请将 `ENABLE_POD_ENI` 标志设置为“true”，`POD_SECURITY_GROUP_ENFORCING_MODE` 标志设置为“standard”。要了解更多信息，请参阅将安全组分配给各个容器组。

EKS 容器组身份集群版本

要使用 EKS 容器组身份，集群的平台版本必须与下表中所列的版本相同或更高，或者 Kubernetes 版本必须比下表中所列的版本更高。要查找适用于 Kubernetes 版本的建议 Amazon EKS 容器组身份代理版本，请参阅the section called “验证兼容性”。

Kubernetes 版本	平台版本
未列出 Kubernetes 版本	支持所有平台版本
1.28	eks.4

EKS 容器组身份限制

EKS 容器组身份适用于：

- 上一个主题 the section called “EKS 容器组身份集群版本” 中列出的 Amazon EKS 集群版本。
- 集群中属于 Linux Amazon EC2 实例的 Worker 节点。

EKS 容器组身份不适用于：

- Amazon Outposts。
- Amazon EKS Anywhere。
- 您在 Amazon EC2 上创建和运行的 Kubernetes 集群。EKS 容器组身份组件仅在 Amazon EKS 上可用。

您不能将 EKS 容器组身份用于：

- 在 Linux Amazon EC2 实例之外的任何位置运行的容器组。不支持在 Amazon Fargate（Fargate）上运行的 Linux 和 Windows 容器组。不支持 Windows Amazon EC2 实例上运行的容器组。

了解 EKS 容器组身份的工作原理

Amazon EKS 容器组身份关联提供管理应用程序凭证的功能，类似于 Amazon EC2 实例配置文件为 Amazon EC2 实例提供凭证的方式。

Amazon EKS 容器组身份通过额外的 EKS Auth API，以及在每个节点上运行的代理容器组为您的工作负载提供凭证。

在您的插件中，例如，Amazon EKS 插件和自我管理控制器、运算符和其他插件，创建者需要更新软件才能使用最新的 Amazon SDK。有关 EKS 容器组身份与 Amazon EKS 插件之间的兼容性列表，请参阅上一节 the section called “EKS 容器组身份限制”。

在代码中使用 EKS 容器组身份

在您的代码中，您可以使用 Amazon SDK 访问 Amazon 服务。您可以编写代码，为使用 SDK 的 Amazon 服务创建客户端，默认情况下，SDK 会在一系列位置中搜索要使用的 Amazon Identity and Access Management 凭证。找到有效凭证后，搜索停止。有关使用的默认位置的更多信息，请参阅《Amazon SDK 和工具参考指南》中的凭证提供程序链。

EKS 容器组身份已添加到容器凭证提供程序，可在默认凭证链的一个步骤中搜索。如果您的工作负载当前使用凭证链中较早的证书，则即使您为同一工作负载配置了 EKS 容器组身份关联，这些凭证也将继续使用。这样，在删除旧凭证之前，您可以先创建关联，从而安全地从其他类型的凭证迁移。

容器凭证提供程序从每个节点上运行的代理处提供临时凭证。在 Amazon EKS 中，代理是 Amazon EKS 容器组身份，而在 Amazon Elastic Container Service 中，代理则是 `amazon-ecs-agent`。SDK 使用环境变量来定位要连接的代理。

相比之下，服务账户的 IAM 角色提供了一个 Web 身份令牌，Amazon SDK 必须使用 `AssumeRoleWithWebIdentity` 与 Amazon Security Token Service 进行交换。

EKS 容器组身份代理如何使用容器组（pod）

1. 当 Amazon EKS 启动一个新的容器组（pod），而该容器组使用与 EKS 容器组身份关联的服务账户时，集群将添加以下内容到容器组清单中：

```
    env:
    - name: AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE
      value: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-
identity-token"
    - name: AWS_CONTAINER_CREDENTIALS_FULL_URI
      value: "http://169.254.170.23/v1/credentials"
    volumeMounts:
    - mountPath: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/"
      name: eks-pod-identity-token
  volumes:
  - name: eks-pod-identity-token
    projected:
```

```
      defaultMode: 420
      sources:
      - serviceAccountToken:
          audience: pods.eks.amazonaws.com
          expirationSeconds: 86400 # 24 hours
          path: eks-pod-identity-token
```

2. Kubernetes 选择要在哪个节点上运行容器组（pod）。然后，节点上的 Amazon EKS 容器组身份代理使用 AssumeRoleForPodIdentity 操作从 EKS Auth API 检索临时凭证。
3. EKS 容器组身份代理可为在容器中运行的 Amazon SDK 提供这些凭证。
4. 在应用程序中使用 SDK 时，无需指定凭证提供程序使用默认凭证链。或者，您可以指定容器凭证提供程序。有关使用的默认位置的更多信息，请参阅《Amazon SDK 和工具参考指南》中的凭证提供程序链。
5. SDK 使用环境变量连接到 EKS 容器组身份代理并检索凭证。

> **Note**
>
> 如果您的工作负载当前使用凭证链中较早的证书，则即使您为同一工作负载配置了 EKS 容器组身份关联，这些凭证也将继续使用。

设置 Amazon EKS 容器组身份代理

Amazon EKS 容器组身份关联提供管理应用程序凭证的功能，类似于 Amazon EC2 实例配置文件为 Amazon EC2 实例提供凭证的方式。

Amazon EKS 容器组身份通过额外的 EKS Auth API，以及在每个节点上运行的代理容器组为您的工作负载提供凭证。

> **Tip**
>
> 您无需在 EKS 自动模式集群上安装 EKS 容器组身份代理。此功能已内置在 EKS 自动模式中。

注意事项

- 默认情况下，EKS 容器组身份代理预安装在 EKS 自动模式集群上。要了解更多信息，请参阅EKS 自动模式。

- 默认情况下，EKS 容器组身份代理会侦听 IPv4 和 IPv6 地址，以便为容器组请求凭证。对于 IPv4，代理会使用环回（本地主机）IP 地址 169.254.170.23，对于 IPv6，则会使用本地主机 IP 地址 [fd00:ec2::23]。
- 如果禁用 IPv6 地址或以其它方式阻止本地主机 IPv6 IP 地址，代理将无法启动。要在无法使用 IPv6 的节点上启动代理，请按照 the section called “禁用 IPv6” 中的步骤禁用 IPv6 配置。

创建 Amazon EKS 容器组身份代理

代理先决条件

- 现有 Amazon EKS 集群。要部署一个角色，请参阅开始使用。集群版本和平台版本必须与 EKS 容器组身份集群版本中列出的版本相同或更高。
- 节点角色有权让代理在 EKS Auth API 中执行 AssumeRoleForPodIdentity 操作。您可以使用 Amazon 托管式策略：AmazonEKSWorkerNodePolicy，或添加类似如下的自定义策略：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks-auth:AssumeRoleForPodIdentity"
            ],
            "Resource": "*"
        }
    ]
}
```

  可通过标签来限制此操作，以限制使用代理的容器组可以分派哪些角色。
- 节点可以访问 Amazon ECR 并从中下载映像。该附加组件的容器映像位于查看 Amazon EKS 附加组件的 Amazon 容器映像注册表中列出的注册表中。

  请注意，您可以在 Amazon Web Services 管理控制台 中的可选配置设置，以及 Amazon CLI 中的 --configuration-values 更改映像位置并为 EKS 附加组件提供 imagePullSecrets。
- 节点可以访问 Amazon EKS Auth API。对于私有集群，Amazon PrivateLink 中的 eks-auth 端点是必需的。

使用 Amazon 控制台设置代理

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群，然后为您要配置 EKS 容器组身份代理插件的集群选择集群名称。
3. 选择附加组件选项卡。
4. 选择获取更多附加组件。
5. 选择 EKS 容器组身份代理插件框右上角的框，然后选择下一步。
6. 在配置选定插件设置页面上，从版本下拉列表中选择任意版本。
7. （可选）展开可选配置设置以输入其他配置。例如，您可以提供备用容器映像位置和 `ImagePullSecrets`。带有已接受键的 JSON 架构显示在附加组件配置架构中。

   在配置值中输入配置键和值。
8. 选择下一步。
9. 确认 EKS 容器组身份代理容器组正在您的集群上运行。

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

示例输出如下。

```
eks-pod-identity-agent-gmqp7                                          1/1     Running
   1 (24h ago)   24h
eks-pod-identity-agent-prnsh                                          1/1     Running
   1 (24h ago)   24h
```

现在，您可以在集群中使用 EKS 容器组身份关联。有关更多信息，请参阅 the section called “分配 IAM 角色”。

使用 Amazon CLI 设置代理

1. 运行以下 Amazon CLI 命令。将 `my-cluster` 替换为您的集群名称。

```
aws eks create-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent --
addon-version v1.0.0-eksbuild.1
```

> **Note**
>
> EKS 容器组身份代理不将 `service-account-role-arn` 用于服务账户的 IAM 角色。您必须为 EKS 容器组身份代理提供节点角色的权限。

2. 确认 EKS 容器组身份代理容器组正在您的集群上运行。

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

示例输出如下。

```
eks-pod-identity-agent-gmqp7                                          1/1     Running
   1 (24h ago)   24h
eks-pod-identity-agent-prnsh                                          1/1     Running
   1 (24h ago)   24h
```

现在，您可以在集群中使用 EKS 容器组身份关联。有关更多信息，请参阅 the section called “分配 IAM 角色”。

向 Kubernetes 服务账户分配 IAM 角色

本主题介绍如何配置 Kubernetes 服务账户，以便使用 EKS 容器组身份担任 Amazon Identity and Access Management（IAM）角色。然后，配置为使用服务账户的任何容器组（pod）都可以访问该角色有权访问的任何 Amazon 服务。

要创建 EKS 容器组身份关联，只需一步；您可以通过 Amazon Web Services 管理控制台、Amazon CLI、Amazon SDK、Amazon CloudFormation，以及其它工具在 EKS 中创建关联。任何 Kubernetes 对象中都没有关于集群内关联的任何数据或元数据，也不需要向服务账户添加任何注释。

先决条件

- 现有集群。如果您没有，可以按照开始使用 中的指南之一创建一个。
- 创建关联的 IAM 主体必须具有 `iam:PassRole`。
- 在您的设备或 Amazon CLI 上安装和配置的最新版本 Amazon CloudShell。您可以使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1` 检查您的当前版本。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon

CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。

- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 包含集群配置的现有 `kubectl config` 文件。要创建 `kubectl config` 文件，请参阅 the section called “使用 kubectl 访问集群”。

创建容器组身份关联（Amazon 控制台）

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群，然后为您要配置 EKS 容器组身份代理插件的集群选择集群名称。
3. 选择访问选项卡。
4. 在容器组身份关联中，选择创建。
5. 对于 IAM 角色，选择具有工作负载所需权限的 IAM 角色。

> **Note**
>
> 该列表仅包含具有以下信任策略的角色，该策略允许 EKS 容器组身份使用这些角色。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
```

```
}
```

`sts:AssumeRole` – 在将临时凭证传递给您的容器组之前，EKS 容器组身份使用 `AssumeRole` 分派 IAM 角色。

`sts:TagSession` – EKS 容器组身份使用 `TagSession` 在对 Amazon STS 的请求中包含会话标签。

您可以在信任策略的条件键中使用这些标签，来限制哪些服务账户、命名空间和集群可以使用此角色。

有关 Amazon EKS 条件键的列表，请参阅《Service Authorization Reference》中的 Conditions defined by Amazon Elastic Kubernetes Service。要了解您可以对哪些操作和资源使用条件键，请参阅 Actions defined by Amazon Elastic Kubernetes Service。

6. 对于 Kubernetes 命名空间，选择包含服务账户和工作负载的 Kubernetes 命名空间。或者，您可以按名称指定集群中不存在的命名空间。
7. 对于 Kubernetes 服务账户，选择要使用的 Kubernetes 服务账户。Kubernetes 工作负载清单必须指定此服务账户。或者，您可以按名称指定集群中不存在的服务账户。
8. （可选）对于标签，选择添加标签以在键值对中添加元数据。这些标签将应用于关联，并可在 IAM 策略中使用。

   您可以重复此步骤以添加多个标签。
9. 选择创建。

创建容器组身份关联（Amazon CLI）

1. 要将现有 IAM 策略关联到您的 IAM 角色，请跳至下一步。

   创建一个 IAM 策略。您可以创建自己的策略，也可以复制已授予您部分所需权限的 Amazon 托管策略，并根据您的特定要求对其进行自定义。有关更多信息，请参阅《IAM 用户指南》中的创建 IAM 策略。

   a. 创建一个文件，其中包含您希望容器组（pod）访问的 Amazon 服务所需的权限。有关所有 Amazon 服务的所有操作的列表，请参阅服务授权参考。

      您可以运行以下命令创建一个示例策略文件，以实现对 Amazon S3 存储桶的只读访问权限。您可以选择将配置信息或引导脚本存储在此存储桶中，并且容器组（pod）中的容器可以从存储桶

读取文件并将其加载到应用程序中。如果您要创建此示例策略，请将以下内容复制到您的设备。将 *my-pod-secrets-bucket* 替换为您的存储桶名称并运行该命令。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
```

b. 创建 IAM 策略。

```
aws iam create-policy --policy-name my-policy --policy-document file://my-
policy.json
```

2. 创建 IAM 角色并将其与 Kubernetes 服务账户关联。

a. 如果您有要担任 IAM 角色的现有 Kubernetes 服务账户，则可跳过此步骤。

创建一个 Kubernetes 服务账户。将以下内容复制到您的设备。将 *my-service-account* 替换为所需的名称，如有必要，将 *default* 替换为其他命名空间。如更改 *default*，则命名空间必须已经存在。

```
cat >my-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

运行如下命令。

```
kubectl apply -f my-service-account.yaml
```

b. 运行以下命令为 IAM 角色创建信任策略文件。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

c. 创建角色。将 *my-role* 替换为您的 IAM 角色的名称，并将 *my-role-description* 替换为您的角色的描述。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://
trust-relationship.json --description "my-role-description"
```

d. 将 IAM 策略附加到您的角色。将 *my-role* 替换为您的 IAM 角色的名称，并将 *my-policy* 替换为您创建的现有策略的名称。

```
aws iam attach-role-policy --role-name my-role --policy-arn=arn:aws-
cn:iam::111122223333:policy/my-policy
```

> **Note**
>
> 与服务账户的 IAM 角色不同，EKS 容器组身份不在服务账户上使用注释。

e. 运行以下命令以创建关联。将 `my-cluster` 替换为集群的名称，将 *my-service-account* 替换为您所需的名称，并根据需要将 *default* 替换为其他命名空间。

```
aws eks create-pod-identity-association --cluster-name my-cluster --role-arn
 arn:aws-cn:iam::111122223333:role/my-role --namespace default --service-account
 my-service-account
```

示例输出如下。

```
{
    "association": {
        "clusterName": "my-cluster",
        "namespace": "default",
        "serviceAccount": "my-service-account",
        "roleArn": "arn:aws-cn:iam::111122223333:role/my-role",
        "associationArn": "arn:aws-cn::111122223333:podidentityassociation/my-
cluster/a-abcdefghijklmnop1",
        "associationId": "a-abcdefghijklmnop1",
        "tags": {},
        "createdAt": 1700862734.922,
        "modifiedAt": 1700862734.922
    }
}
```

> **Note**
>
> 您可以按名称指定集群中不存在的命名空间和服务账户。您必须创建命名空间、服务账户，以及使用服务账户的工作负载，EKS 容器组身份关联才能正常工作。

确认配置

1. 确认 IAM 角色的信任策略配置正确。

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

示例输出如下。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow EKS Auth service to assume this role for Pod Identities",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
```

```
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

2. 确认您在上一步中附加到角色的策略已附加到该角色。

```
aws iam list-attached-role-policies --role-name my-role --query
 'AttachedPolicies[].PolicyArn' --output text
```

示例输出如下。

```
            arn:aws-cn:iam::111122223333:policy/my-policy
```

3. 设置一个变量以存储要使用策略的 Amazon 资源名称（ARN）。将 *my-policy* 替换为要确认其权限的策略的名称。

```
export policy_arn=arn:aws-cn:iam::111122223333:policy/my-policy
```

4. 查看该策略的默认版本。

```
aws iam get-policy --policy-arn $policy_arn
```

示例输出如下。

```
{
    "Policy": {
        "PolicyName": "my-policy",
        "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
        "Arn": "arn:aws-cn:iam::111122223333:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        [...]
    }
}
```

5. 查看策略内容以确保该策略包含容器组（pod）所需的所有权限。如有必要，将以下命令中的 *1* 替换为上一个输出中返回的版本。

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

示例输出如下。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
```

如果您在上一步中创建了示例策略，则您的输出与示例相同。如果您创建了其他策略，则与##内容不同。

后续步骤

the section called “容器组（pod）服务账户”

使用 EKS 容器组身份目标 IAM 角色访问 Amazon 资源

在 Amazon Elastic Kubernetes Service（Amazon EKS）上运行应用程序时，您可能需要访问位于不同 Amazon 账户中的 Amazon 资源。本指南向您介绍如何使用 EKS 容器组身份设置跨账户访问，以便 Kubernetes 容器组（pod）能够使用目标角色访问其他 Amazon 资源。

先决条件

在开始之前，请确保您已完成以下步骤：

- 设置 Amazon EKS 容器组身份代理
- 创建 EKS 容器组身份角色

工作方式

容器组身份通过名为角色链的流程，使 EKS 集群中的应用程序能够跨账户访问 Amazon 资源。

创建容器组身份关联时，可以提供两个 IAM 角色：一个是与 EKS 集群位于同一账户的 EKS 容器组（pod）身份角色，另一个是包含您希望访问的 Amazon 资源（例如 S3 存储桶或 RDS 数据库）的账户中的目标 IAM 角色。因 IAM PassRole 要求，EKS 容器组身份角色必须位于 EKS 集群所在账户中，而目标 IAM 角色可以位于任何 Amazon 账户中。PassRole 允许 Amazon 实体将角色代入委派给其他服务。EKS 容器组身份通过 PassRole 将角色连接到 Kubernetes 服务账户，同时要求角色及传递该角色的身份都必须与 EKS 集群位于同一 Amazon 账户中。如果应用程序容器组（pod）需要访问 Amazon 资源，它会向容器组身份请求凭证。随后，容器组身份会按顺序自动执行两次角色代入：首先代入 EKS 容器组身份角色，然后使用该凭证代入目标 IAM 角色。此过程可为容器组（pod）提供临时凭证（这些凭证具有目标角色中定义的权限），从而安全访问其他 Amazon 账户中的资源。

缓存注意事项

由于缓存机制，对现有容器组身份关联中 IAM 角色的更新可能不会立即在 EKS 集群上运行的容器组（pod）中生效。容器组身份代理会根据获取凭证时的关联配置来缓存 IAM 凭证。如果关联仅包含 EKS 容器组身份角色而不包含目标 IAM 角色，则缓存凭证有效期为 6 小时。如果关联同时包含 EKS 容器组身份角色 ARN 和目标 IAM 角色，则缓存凭证有效期为 59 分钟。修改现有关联（例如更新 EKS 容器组身份角色 ARN 或添加目标 IAM 角色）不会重置现有缓存。因此，只有在刷新缓存凭证后，代理才会识别更新。如需立即应用更改，可重新创建现有容器组（pod）；否则需等待缓存过期。

第 1 步：创建并关联目标 IAM 角色

在此步骤中，您将通过创建和配置目标 IAM 角色来建立安全的信任链。出于演示目的，我们将创建新的目标 IAM 角色，以在两个 Amazon 账户间建立信任链：位于 EKS 集群 Amazon 账户中的 EKS 容器组身份角色（例如 `eks-pod-identity-primary-role`）将获得代入目标账户中目标 IAM 角色（例如 `eks-pod-identity-aws-resources`）的权限，从而实现对 Amazon S3 存储桶等 Amazon 资源的访问。

创建目标 IAM 角色

1. 打开 Amazon IAM 控制台。
2. 在顶部导航栏中，确认您已登录包含目标 IAM 角色所需 Amazon 资源（例如 S3 存储桶或 DynamoDB 表）的账户。
3. 在左侧导航窗格中，选择 角色。
4. 选择创建角色按钮，然后选择“可信实体类型”下的 Amazon 账户。

5. 选择其他 Amazon 账户，输入 Amazon 账号（EKS 容器组身份角色所在的账户账号），然后选择下一步。
6. 添加要与该角色关联的权限策略（例如 AmazonS3FullAccess），然后选择下一步。
7. 输入 MyCustomIAMTargetRole 等角色名称，然后选择创建角色。

更新目标 IAM 角色的信任策略

1. 创建角色后，您将返回到角色列表。查找并选择您在上一步中创建的新角色（例如 MyCustomIAMTargetRole）。
2. 选择信任关系选项卡。
3. 点击右侧的编辑信任策略。
4. 在策略编辑器中，将默认 JSON 替换为信任策略。将 IAM 角色 ARN 中角色名称和 111122223333 的占位符值替换为托管 EKS 集群的 Amazon 账户 ID。您也可以选择在角色信任策略中使用 PrincipalTags，以仅授权来自给定集群和命名空间的特定服务账户来代入目标角色。例如：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/eks-cluster-arn": "arn:aws:eks:us-
east-1:111122223333:cluster/example-cluster",
          "aws:RequestTag/kubernetes-namespace": "ExampleNameSpace",
          "aws:RequestTag/kubernetes-service-account": "ExampleServiceAccountName"
        },
        "ArnEquals": {
          "aws:PrincipalARN": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-
role"
        }
      }
```

```
    }
  ]
}
```

上述策略允许来自 Amazon 账户 111122223333，并带有相关 EKS 容器组身份会话标签的角色 eks-pod-identeity-primary-role 代入此角色。

如果您在 EKS 容器组身份中禁用会话标签，则在代入目标角色时，EKS 容器组身份还会设置 sts:ExternalId，其中包含有关容器组（pod）集群、命名空间和服务账户的信息。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "region/111122223333/cluster-name/namespace/service-
account-name"
        },
        "ArnEquals": {
          "aws:PrincipalARN": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-
role"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:TagSession"
    }
  ]
}
```

上述策略有助于确保只有预期的集群、命名空间和服务账户才能代入目标角色。

更新 EKS 容器组身份角色的权限策略

在此步骤中，您将通过添加目标 IAM 角色 ARN 作为资源，来更新 EKS 容器组身份角色（与 Amazon EKS 集群关联）的权限策略。

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群，然后选择 EKS 集群名称。
3. 选择访问选项卡。
4. 在容器组身份关联下，选择 EKS 容器组身份角色。
5. 依次选择权限、添加权限，然后选择创建内联策略。
6. 选择右侧的 JSON。
7. 在策略编辑器中，将默认 JSON 替换为权限策略。将 IAM 角色 ARN 中角色名称和 222233334444 的占位符值替换为目标 IAM 角色。例如：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Resource": "arn:aws:iam::222233334444:role/eks-pod-identity-aws-resources"
        }
    ]
}
```

第 2 步：将 IAM 角色关联到 Kubernetes 服务账户

在此步骤中，您将在目标 IAM 角色与 EKS 集群中的 Kubernetes 服务账户之间创建关联。

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群，然后选择要向其添加关联的集群名称。
3. 选择访问选项卡。
4. 在容器组身份关联中，选择创建。
5. 在 IAM 角色中选择供工作负载代入的 EKS 容器组身份角色。

6. 在目标 IAM 角色中选择将由 EKS 容器组身份角色代入的目标 IAM 角色。
7. 在 Kubernetes 命名空间字段中，输入要在其中创建关联的命名空间名称（例如 `my-app-namespace`），这将定义服务账户所在的位置。
8. 在 Kubernetes 服务账户字段中，输入将使用 IAM 凭证的服务账户名称（例如 `my-service-account`）。这会将 IAM 角色链接到服务账号。
9. 选择创建以创建关联。

配置容器组（pod）使用服务账户访问 Amazon 服务

如果容器组（pod）需要访问 Amazon 服务，则必须配置它来使用 Kubernetes 服务账户。服务账户必须关联到有权访问 Amazon 服务的 Amazon Identity and Access Management（IAM）角色。

- 现有集群。如果还没有，可以使用开始使用 中的指南之一创建一个。
- 现有 Kubernetes 服务账户和 EKS 容器组身份关联（将服务账户与 IAM 角色关联）。该角色必须具有关联的 IAM 策略，其中包含您希望容器组（pod）必须具有的权限，以便使用 Amazon 服务。有关如何创建和配置服务账户和角色的更多信息，请参阅 the section called “分配 IAM 角色”。
- 在您的设备或 Amazon CLI 上安装和配置的最新版本 Amazon CloudShell。您可以使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1` 检查您的当前版本。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 包含集群配置的现有 `kubectl config` 文件。要创建 `kubectl config` 文件，请参阅 the section called “使用 kubectl 访问集群”。
  1. 使用以下命令创建部署清单，您可以使用该部署清单部署容器组（pod）来确认配置。将示例值替换为您自己的值。

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
```

```
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. 将清单部署到集群。

```
kubectl apply -f my-deployment.yaml
```

3. 确认容器组（pod）具有所需的环境变量。

   a. 按上一步部署后，查看部署的容器组（pod）。

   ```
   kubectl get pods | grep my-app
   ```

   示例输出如下。

   ```
   my-app-6f4dfff6cb-76cv9   1/1     Running   0          3m28s
   ```

   b. 确认容器组（pod）已挂载服务账户令牌。

   ```
   kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep
    AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:
   ```

   示例输出如下。

   ```
   AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:  /var/run/secrets/
   pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token
   ```

4. 确认您的容器组（pod）可以使用您在附加到角色的 IAM 策略中分配的权限与 Amazon 服务进行交互。

> **Note**
>
> 当容器组（pod）使用来自与服务账户关联的 IAM 角色的 Amazon 凭证时，该容器组（pod）中的 Amazon CLI 或其他 SDK 包仅使用该角色提供的凭证。如果您不限制对提供给 Amazon EKS 节点 IAM 角色的凭证的访问，容器组（pod）仍然可以访问这些凭证。有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

如果您的容器组（pod）无法按预期与服务进行交互，请完成以下步骤，确认所有配置都正确。

a. 确认容器组（pod）使用的 Amazon SDK 版本支持通过 EKS 容器组身份关联担任 IAM 角色。有关更多信息，请参阅 the section called “受支持的 SDK”。

b. 确认部署正在使用服务账户。

```
kubectl describe deployment my-app | grep "Service Account"
```

示例输出如下。

```
Service Account:  my-service-account
```

基于标签授予容器组（pod）对 Amazon 资源的访问权限

基于属性的访问权限控制（ABAC）通过将各种属性组合在一起的策略向用户授予权限。EKS 容器组身份将标签附加到每个容器组（pod）的临时凭证上，其中包含集群名称、命名空间、服务账户名称等属性。这些角色会话标签允许管理员根据匹配的标签访问 Amazon 资源，从而使管理员能够创建可跨服务账户使用的单一角色。通过添加对角色会话标签的支持，您可以在集群之间和集群内的工作负载之间实施更严格的安全边界，同时重复使用相同的 IAM 角色和 IAM 策略。

带标签的示例策略

以下是一个 IAM 策略示例，该策略会在相应对象标有 EKS 集群名称时授予 `s3:GetObject` 权限。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectTagging"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/eks-cluster-name": "${aws:PrincipalTag/eks-
cluster-name}"
                }
            }
        }
    ]
}
```

启用或禁用会话标签

EKS 容器组身份会在代入角色时添加一组预定义的会话标签。这些会话标签允许管理员根据匹配的标签访问 Amazon 资源，从而使管理员能够创建可跨资源使用的单一角色。

启用会话标签

会话标签将通过 EKS 容器组身份自动启用，您无需执行任何操作。默认情况下，EKS 容器组身份会将一组预定义标签附加到会话。要在策略中引用这些标签，请使用语法 `${aws:PrincipalTag/`，后跟标签键。例如 `${aws:PrincipalTag/kubernetes-namespace}`。

- `eks-cluster-arn`
- `eks-cluster-name`
- `kubernetes-namespace`
- `kubernetes-service-account`
- `kubernetes-pod-name`
- `kubernetes-pod-uid`

禁用会话标签

Amazon 会将内联会话策略、托管策略 ARN 和会话标签压缩为具有单独限制的打包二进制格式。如果您收到 `PackedPolicyTooLarge` 错误消息，表明打包二进制格式超出了大小限制，则可通过禁用由 EKS 容器组身份添加的会话标签来减小大小。要禁用这些会话标签，请执行以下步骤：

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群，然后选择要修改的集群名称。
3. 选择访问选项卡。
4. 在容器组身份关联中，在关联 ID 中选择要修改的关联 ID，然后选择编辑。
5. 在会话标签下，选择禁用会话标签。
6. 选择保存更改。

跨账户标签

EKS 容器组身份添加的所有会话标签都是可传递的；将标签键和值传递到任何 `AssumeRole` 操作，以便您的工作负载用于将角色切换到其他账户。您可以在其他账户的策略中使用这些标签，来限制跨账户场景中的访问。有关更多信息，请参阅《IAM 用户指南》中的使用会话标签链接角色。

自定义标签

EKS 容器组身份无法在其执行的 `AssumeRole` 操作中添加其它自定义标签。但是，应用于 IAM 角色的标签始终可通过相同格式引用：`${aws:PrincipalTag/` 后跟键，例如 `${aws:PrincipalTag/MyCustomTag}`。

> **Note**
>
> 如果发生冲突，通过 `sts:AssumeRole` 请求添加到会话中的标签优先。例如，假设：
>
> - EKS 代入客户角色时，Amazon EKS 向会话添加键 `eks-cluster-name` 和值 `my-cluster`
> - 您向 IAM 角色添加了一个值为 `my-own-cluster` 的 `eks-cluster-name` 标签。
>
> 在这种情况下，前者优先，`eks-cluster-name` 标签的值为 `my-cluster`。

将容器组身份与 Amazon SDK 结合使用

使用 EKS 容器组身份凭证

要使用来自 EKS 容器组身份关联的凭证，您的代码可以使用任何 Amazon SDK 为具有 SDK 的 Amazon 服务创建客户端，默认情况下，SDK 会在一系列位置中搜索要使用的 Amazon Identity and Access Management 凭证。如果您在创建客户端或者初始化 SDK 时未指定凭证提供程序，则将使用 EKS 容器组身份凭证。

由于 EKS 容器组身份已添加到容器凭证提供程序，可在默认凭证链的一个步骤中搜索，因此该操作可行。如果您的工作负载当前使用凭证链中较早的证书，则即使您为同一工作负载配置了 EKS 容器组身份关联，这些凭证也将继续使用。

有关 EKS 容器组身份如何工作的更多信息，请参阅 the section called “工作方式”。

在使用了解 EKS 容器组身份如何向容器组（pod）授予对 Amazon 服务的访问权限时，容器组中的容器必须使用支持从 EKS 容器组身份代理担任 IAM 角色的 Amazon SDK 版本。请确保为您的 Amazon SDK 使用以下版本或更高版本：

- Java（版本 2）：2.21.30
- Java – 1.12.746
- Go v1：v1.47.11
- Go v2：release-2023-11-14
- Python（Boto3）– 1.34.41
- Python (botocore) – 1.34.41
- Amazon CLI – 1.30.0

  Amazon CLI – 2.15.0
- JavaScript v2 – 2.1550.0
- JavaScript v3 – v3.458.0
- Kotlin – v1.0.1
- Ruby：3.188.0
- Rust – release-2024-03-13
- C++ – 1.11.263
- .NET – 3.7.734.0
- PowerShell – 4.1.502

- PHP：3.289.0

为了确保您使用的是受支持的 SDK，请在构建容器时按照用于在 Amazon 上进行构建的工具中针对您的首选 SDK 的安装说明操作。

有关支持 EKS 容器组身份的附加组件列表，请参阅 the section called “容器组身份支持参考”。

在 EKS 容器组身份代理中禁用 IPv6

Amazon Web Services 管理控制台

1. 要在 EKS 容器组身份代理中禁用 IPv6，请将以下配置添加到 EKS 附加组件的可选配置设置中。
   a. 打开 Amazon EKS 控制台。
   b. 在左侧导航窗格中，选择 Clusters（集群），然后选择要为其配置附加组件的集群的名称。
   c. 选择附加组件选项卡。
   d. 选择 EKS 容器组身份代理附加组件框右上角的框，然后选择编辑。
   e. 在配置 EKS 容器组身份代理页面上：
      i. 选择您想使用的版本。我们建议您保留与上一步相同的版本，并通过单独的操作更新版本和配置。
      ii. 展开可选配置设置。
      iii. 在配置值中输入 JSON 键 "agent":，以及键为 "additionalArgs": 的嵌套 JSON 对象的值。生成的文本必须是有效的 JSON 对象。如果此键和值是文本框中的唯一数据，请用大括号 { } 将键和值括起来。以下示例显示网络策略已启用：

```
{
    "agent": {
        "additionalArgs": {
            "-b": "169.254.170.23"
        }
    }
}
```

      此配置会将 IPv4 地址设置为代理所使用的唯一地址。
   f. 要通过替换 EKS 容器组身份代理容器组来应用新配置，请选择保存更改。

      Amazon EKS 通过推出适用于 EKS 容器组身份代理的 Kubernetes DaemonSet，将更改应用到 EKS 附加组件。可在 Amazon Web Services 管理控制台 中的附加组件更新历史记录中跟

踪推出状态和 `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system`。

`kubectl rollout` 具有以下命令：

```
$ kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

如果推出时间过长，Amazon EKS 将撤销推出，并且会将类型为附加组件更新且状态为失败的消息添加到附加组件的更新历史记录中。要调查任何问题，请先查看推出历史记录，然后在 EKS 容器组身份代理容器组上运行 `kubectl logs`，进而查看 EKS 容器组身份代理的日志。

2. 如果更新历史记录中的新条目状态为成功，则表示推出已完成，并且附加组件正在所有 EKS 容器组身份代理容器组中使用新配置。

Amazon CLI

1. 要在 EKS 容器组身份代理中禁用 IPv6，请将以下配置添加到 EKS 附加组件的配置值中。

运行以下 Amazon CLI 命令。将 `my-cluster` 替换为集群的名称，并将 IAM 角色 ARN 替换为您正在使用的角色。

```
aws eks update-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent \
    --resolve-conflicts PRESERVE --configuration-values '{"agent":{"additionalArgs":
 { "-b": "169.254.170.23"}}}'
```

此配置会将 IPv4 地址设置为代理所使用的唯一地址。

Amazon EKS 通过推出适用于 EKS 容器组身份代理的 Kubernetes DaemonSet，将更改应用到 EKS 附加组件。可在 Amazon Web Services 管理控制台 中的附加组件更新历史记录中跟踪推出状态和 `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system`。

`kubectl rollout` 具有以下命令：

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

如果推出时间过长，Amazon EKS 将撤销推出，并且会将类型为附加组件更新且状态为失败的消息添加到附加组件的更新历史记录中。要调查任何问题，请先查看推出历史记录，然后在 EKS 容器组身份代理容器组上运行 `kubectl logs`，进而查看 EKS 容器组身份代理的日志。

创建具有 EKS 容器组身份所需信任策略的 IAM 角色

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

`sts:AssumeRole`

在将临时凭证传递给您的容器组之前，EKS 容器组身份使用 `AssumeRole` 分派 IAM 角色。

`sts:TagSession`

EKS 容器组身份使用 `TagSession` 在对 Amazon STS 的请求中包含会话标签。

设置条件

您可以在信任策略的条件键中使用这些标签，来限制哪些服务账户、命名空间和集群可以使用此角色。有关容器组身份添加的请求标签列表，请参阅the section called “启用或禁用会话标签”。

例如，您可以通过在以下信任策略中添加 Condition，将可以担任某个容器组身份 IAM 角色的容器组（pod）限制为特定的 ServiceAccount 和 Namespace：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/kubernetes-namespace": [
                        "Namespace"
                    ],
                    "aws:RequestTag/kubernetes-service-account": [
                        "ServiceAccount"
                    ]
                }
            }
        }
    ]
}
```

有关 Amazon EKS 条件键的列表，请参阅《Service Authorization Reference》中的 Conditions defined by Amazon Elastic Kubernetes Service。要了解您可以对哪些操作和资源使用条件键，请参阅 Actions defined by Amazon Elastic Kubernetes Service。

使用节点来管理计算资源

Kubernetes 节点是运行容器化应用程序的机器。每个节点包含下列组件：

- 容器运行时 – 负责运行容器的软件。
- kubelet – 确保容器运行状况良好且在关联的容器组（pod）中运行。
- kube-proxy – 维护允许与容器组（pod）通信的网络规则。

有关更多信息，请参阅 Kubernetes 文档中的节点。

Amazon EKS 集群可以按照任意 EKS 自动模式托管节点、自主管理型节点、Amazon EKS 托管节点组、Amazon Fargate 以及 Amazon EKS 混合节点功能组合调度容器组（pod）。要了解集群中部署的节点的更多信息，请参阅 the section called “访问集群资源”。

> **Note**
>
> 除混合节点外，节点必须位于与您创建集群时所选子网相同的 VPC 中。不过，节点不需要在相同子网中。

比较计算选项

下表提供了在决定哪些选项最符合您的要求时需要评估的几个标准。另一个选项是自主管理型节点，此选项支持所列的所有标准，但需要更多的手动维护。有关更多信息，请参阅 the section called “自行管理的节点”。

> **Note**
>
> Bottlerocket 与此表中的一般信息有一些具体的区别。有关更多信息，请参阅 GitHub 上的 Bottlerocket 文档。

标准	EKS 托管节点组	EKS 自动模式	Amazon EKS 混合节点功能
可以部署到 Amazon Outposts	否	否	否
可以部署到 Amazon 本地区域	是	否	否
可以运行需要 Windows 的容器	是	否	否
可以运行需要 Linux 的容器	支持	是	是
可以运行需要 Inferentia 芯片的工作负载	是 – 仅限 Amazon Linux 节点	是	否
可以运行需要 GPU 的工作负载	是 – 仅限 Amazon Linux 节点	支持	是
可以运行需要 ARM 处理器的工作负载	是	是	是
可以运行 Amazon Bottlerocket	支持	是	否
容器组之间共享 CPU、内存、存储和网络资源。	支持	是	是
必须部署和管理 Amazon EC2 实例	是	否 – 了解 EC2 托管式实例	是 – 本地物理计算机或虚拟机由您使用自己选择的工具进行管理。
必须保护、维护和修补 Amazon EC2 实例的操作系统	是	否	是 – 在物理计算机或虚拟机上运行的操作

标准	EKS 托管节点组	EKS 自动模式	Amazon EKS 混合节点功能
			系统由您使用自己选择的工具进行管理。
可以在部署节点时提供引导参数，例如额外的 kubelet 实际参数。	是 – 使用 eksctl 或将启动模板与自定义 AMI 结合使用。	否 – 使用 NodeClass 配置节点	是 – 可以使用 nodeadm 自定义引导参数。请参阅 the section called “混合节点 nodeadm”。
可以从与分配给节点的 IP 地址不同的 CIDR 块为容器组（pod）分配 IP 地址。	是 – 结合使用启动模板和自定义 AMI。有关更多信息，请参阅 the section called “启动模板”。	否	是 – 请参阅the section called “配置 CNI”。
可以通过 SSH 访问节点	是	否 – 了解如何进行节点故障排除	是
可以将您自己的自定义 AMI 部署到节点	是 – 使用启动模板	否	是
可以将自己的自定义 CNI 部署到节点	是 – 结合使用启动模板和自定义 AMI	否	是
必须自己更新节点 AMI	是 – 如果部署 Amazon EKS 优化版 AMI，更新可用时，Amazon EKS 控制台会通知您。可以在控制台中单击一键执行更新。如果部署自定义 AMI，更新可用时，Amazon EKS 控制台不会通知您。必须自己执行更新。	否	是 – 在物理计算机或虚拟机上运行的操作系统由您使用自己选择的工具进行管理。请参阅 the section called “准备操作系统”。

标准	EKS 托管节点组	EKS 自动模式	Amazon EKS 混合节点功能
必须自己更新节点 Kubernetes 版本	是 – 如果部署 Amazon EKS 优化版 AMI，更新可用时，Amazon EKS 控制台会通知您。可以在控制台中单击一键执行更新。如果部署自定义 AMI，更新可用时，Amazon EKS 控制台不会通知您。必须自己执行更新。	否	是 – 您可以使用自己选择的工具或使用 `nodeadm` 来管理混合节点升级。请参阅 the section called “升级混合节点”。
可以为容器组（pod）使用 Amazon EBS 存储	是	是，作为一项集成功能。了解如何创建存储类。	否
可以为容器组（pod）使用 Amazon EFS 存储	是	是	否
可以为容器组（pod）使用适用于 Lustre 的 Amazon FSx 存储	是	是	否
可以对服务使用 Network Load Balancer	是	是	是 – 必须使用目标类型 `ip`。
Pod 可以在公有子网中运行	支持	是	否 – 容器组在本地环境中运行。
可以将不同的 VPC 安全组分配给各个容器组（pod）	是 – 仅限 Linux 节点	否	否

标准	EKS 托管节点组	EKS 自动模式	Amazon EKS 混合节点功能
可以运行 Kubernetes DaemonSets	支持	是	是
支持容器组（pod）清单中的 `HostPort` 和 `HostNetwork`	支持	是	是
Amazon 区域可用性	所有 Amazon EKS 支持的区域	所有 Amazon EKS 支持的区域	所有支持 Amazon EKS 的区域，Amazon GovCloud（美国）区域和中国区域除外。
可以在 Amazon EC2 专属主机上运行容器	是	否	否
定价	运行多个容器组（pod）的 Amazon EC2 实例的成本。有关更多信息，请参阅 Amazon EC2定价。	在集群中启用 EKS 自动模式后，除标准的 EC2 实例费用外，您还需要为使用自动模式的计算能力启动的实例单独付费。具体金额因启动的实例类型和集群所在 Amazon 区域而异。有关更多信息，请参阅 Amazon EKS 定价。	每小时的混合节点 vCPU 成本。有关更多信息，请参阅 Amazon EKS 定价。

使用托管式节点组简化节点生命周期

Amazon EKS 托管节点组为 Amazon EKS Kubernetes 集群自动对节点（Amazon EC2 实例）进行预置和生命周期管理。

使用 Amazon EKS 托管节点组，您无需单独预置或注册 Amazon EC2 实例以提供计算容量来运行 Kubernetes 应用程序。您可以通过单个操作为集群创建、自动更新或终止节点。节点的更新和终止操作会自动耗尽节点，以确保您的应用程序保持为可用的状态。

所有托管节点均作为由 Amazon EKS 为您管理的 Amazon EC2 Auto Scaling 组的一部分进行预置。所有资源（包括实例和自动扩缩组在内）都在您的 Amazon 账户中运行。每个节点组跨您定义的多个可用区运行。

托管式节点组还可以选择利用节点自动修复功能，从而持续监控节点的运行状况。此功能还会自动处理检测到的问题，并在可能的情况下替换节点。这有助于在尽可能减少手动干预的情况下提高集群的整体可用性。有关更多信息，请参阅 the section called “节点运行状况”。

您可以使用 Amazon EKS 控制台、`eksctl`、Amazon CLI、Amazon API 或基础设施即代码工具（包括 Amazon CloudFormation），将托管式节点组添加到新的或现有集群中。作为托管节点组一部分启动的节点会自动进行标记，以便 Kubernetes Cluster Autoscaler 能自动发现这些节点。您可以使用节点组将 Kubernetes 标签应用到节点并随时更新它们。

使用 Amazon EKS 托管节点组没有额外的费用，您只需为预置的 Amazon 资源付费。这些资源包括 Amazon EC2 实例、Amazon EBS 卷、Amazon EKS 集群小时数和任何其他 Amazon 基础设施。无最低费用，无预先承诺。

要开始使用新 Amazon EKS 集群和托管节点组，请参阅 the section called “创建集群（控制台和 CLI）”。

要将托管节点组添加到现有集群，请参阅 the section called “创建”。

托管节点组概念

- Amazon EKS 托管节点组为您创建和管理 Amazon EC2 实例。
- 所有托管节点均作为由 Amazon EKS 为您管理的 Amazon EC2 Auto Scaling 组的一部分进行预置。此外，包括 Amazon EC2 实例和自动扩缩组在内的所有资源都在您的 Amazon 账户中运行。
- 托管节点组的弹性伸缩组涵盖您在创建组时指定的所有子网。
- Amazon EKS 标记托管节点组资源，以便将其配置为使用 Kubernetes Cluster Autoscaler。

> **Important**
>
> 如果要使用 Kubernetes Cluster Autoscaler 在由 Amazon EBS 卷支持的多个可用区中运行有状态应用程序，则应该配置多个节点组，且每个节点组的范围都限定为一个可用区。此外，您还应该启用 `--balance-similar-node-groups` 功能。

- 在部署托管节点时，可以使用自定义启动模板以获得更大灵活性和自定义性。例如，您可以指定额外的 kubelet 参数并使用自定义 AMI。有关更多信息，请参阅 the section called “启动模板”。如果首次创建托管节点组时没有使用自定义启动模板，则会有自动生成的启动模板。不要手动修改此自动生成的模板，否则会发生错误。
- Amazon EKS 在托管节点组上遵循 CVE 和安全补丁的责任共担模式。当托管节点运行 Amazon EKS 优化版 AMI 时，Amazon EKS 负责在报告 Bug 或问题时构建 AMI 的修补版本。我们可以发布修复程序。但是，您负责将这些修补的 AMI 版本部署到托管节点组。当托管节点运行自定义 AMI 时，您要负责在报告 Bug 或问题时构建 AMI 的修补版本，然后部署 AMI。有关更多信息，请参阅 the section called “更新”。
- Amazon EKS 托管节点组既可以在公有子网也可以在私有子网中启动。如果您在不早于 2020 年 4 月 22 日启动了公有子网中的托管节点组，则子网必须将 MapPublicIpOnLaunch 设置为 true，这些实例才能成功加入集群。如果公有子网是使用 eksctl 或 Amazon EKS 发布的 Amazon CloudFormation 模板并在 2020 年 3 月 26 日或之后创建的，则此设置已设置为 true。如果在 2020 年 3 月 26 日之前创建了公有子网，则必须手动更改设置。有关更多信息，请参阅修改子网的公有 IPv4 寻址属性。
- 在私有子网中部署托管节点组时，必须确保它可以访问 Amazon ECR 来拉取容器映像。您可以通过将 NAT 网关连接到子网的路由表或添加以下 Amazon PrivateLink VPC 端点来实现此目的：
  - Amazon ECR API 端点接口 – com.amazonaws.*region-code*.ecr.api
  - Amazon ECR Docker 注册表 API 端点接口 – com.amazonaws.*region-code*.ecr.dkr
  - Amazon S3 网关端点 – com.amazonaws.*region-code*.s3

  有关其他常用服务和端点的信息，请参阅 the section called “私有集群”。
- 托管节点组无法部署在 Amazon Outposts 上或 Amazon Wavelength 中。托管节点组现已可在 Amazon Local Zones 中创建。有关更多信息，请参阅 the section called “ Amazon Local Zones”。
- 您可以在单个集群中创建多个托管节点组。例如，您可以为某些工作负载使用标准的 Amazon EKS 优化版 Amazon Linux AMI 创建一个节点组，为需要 GPU 支持的工作负载使用 GPU 变体创建另一个节点组。
- 如果您的托管节点组遇到 Amazon EC2 实例状况检查失败，则 Amazon EKS 会返回错误代码来帮助您诊断问题。有关更多信息，请参阅 the section called “托管节点组错误代码”。
- Amazon EKS 将 Kubernetes 标签添加到托管节点组实例。Amazon EKS 提供的这些标签带有前缀 eks.amazonaws.com。
- 在终止或更新期间，Amazon EKS 使用 Kubernetes API 自动耗尽节点。
- 使用 AZRebalance 终止节点或减少所需节点数量时，未遵守容器组（pod）中断预算。这些操作试图驱逐节点上的容器组（pod）。但如果超过 15 分钟，则无论节点上的所有容器组（pod）是否都

被终止，该节点都会终止。要将时间延长至节点终止，请向自动扩缩组添加生命周期挂钩。有关更多信息，请参阅《Amazon EC2 Auto Scaling 用户指南》中的添加生命周期挂钩。

- 为了在收到 Spot 中断通知或容量再平衡通知后正确运行耗尽进程，必须将 `CapacityRebalance` 设置为 `true`。
- 更新托管节点组遵循您为容器组（pod）设置的容器组（pod）中断预算。有关更多信息，请参阅 the section called “更新行为详细信息”。
- 使用 Amazon EKS 托管节点组不会产生额外的费用。您仅需为预置的Amazon资源付费。
- 如果要为节点加密 Amazon EBS 卷，可以使用启动模板部署节点。要在不使用启动模板的情况下部署包含加密 Amazon EBS 卷的托管节点，必须对账户中创建的所有新 Amazon EBS 卷进行加密。有关更多信息，请参阅 Amazon EC2 用户指南中的默认加密。

托管节点组容量类型

创建托管节点组时，您可以选择按需容量或 Spot 容量类型。Amazon EKS 使用 Amazon EC2 Auto Scaling 组部署托管节点组，该组只包含按需实例或只包含 Amazon EC2 竞价型实例。您可以将容错应用程序的容器组（pod）调度到 Spot 托管节点组，将容错应用程序调度到单个 Kubernetes 集群中的按需节点组。预设情况下，托管节点组部署按需 Amazon EC2 实例。

按需

使用按需实例，您按秒为计算容量支付费用，无需长期订阅。

预设情况下，如果您没有指定容量类型，则会使用按需实例预置托管节点组。托管节点组会代表您配置 Amazon EC2 Auto Scaling 组，并应用以下设置：

- 预置按需容量的分配策略设置为 `prioritized`。托管节点组使用实例类型在 API 中的传入顺序，确定在满足按需容量时首先使用哪种实例类型。例如，您可以按以下顺序指定三种实例类型：`c5.large`、`c4.large` 和 `c3.large`。在启动您的按需实例时，托管节点组满足按需容量的顺序是从 `c5.large` 开始，然后是 `c4.large`，再然后是 `c3.large`。有关更多信息，请参阅 Amazon EC2 Auto Scaling 用户指南中的 Amazon EC2 Auto Scaling 组。
- Amazon EKS 会向您指定了容量类型的托管节点组中的所有节点添加以下 Kubernetes 标注：`eks.amazonaws.com/capacityType: ON_DEMAND`。您可以使用此标注在按需节点上调度有状态或不容错的应用程序。

Spot

Amazon EC2 Spot 实例是备用的 Amazon EC2 容量，价格相当于在按需实例价格上打了巨大折扣。当 EC2 需要收回容量时，Amazon EC2 Spot 实例可能中断并发出一个两分钟的中断通知。有关更多信息，请参阅 Amazon EC2 用户指南中的竞价型实例。您可以使用 Amazon EC2 Spot 实例配置托管节点组，以优化 Amazon EKS 集群中运行的计算节点的成本。

要在托管节点组中使用 Spot 实例，请将容量类型设置为 `spot`，创建托管节点组。托管节点组使用以下 Spot 最佳实践代表您配置 Amazon EC2 Auto Scaling 组：

- 为确保在最佳 Spot 容量池中预置 Spot 节点，将分配策略设置为以下策略之一：
  - `price-capacity-optimized`（PCO）– 在 Kubernetes `1.28` 版本或更高版本的集群中创建新节点组时，将分配策略设置为 `price-capacity-optimized`。但是，对于在 Amazon EKS 托管节点组开始支持 PCO 之前使用 `capacity-optimized` 创建的节点组，分配策略不会更改。
  - `capacity-optimized`（CO）– 在 Kubernetes `1.27` 版本或更低版本的集群中创建新节点组时，将分配策略设置为 `capacity-optimized`。

  要增加可用于分配容量的 Spot 容量池的数量，请将托管节点组配置为使用多个实例类型。
- 启用了 Amazon EC2 Spot 容量再平衡，以便 Amazon EKS 能够顺畅地耗尽和重新平衡您的 Spot 节点，从而在 Spot 节点中断风险增加时最大限度地减少应用程序中断。有关更多信息，请参阅 Amazon EC2 Auto Scaling 用户指南中的 Amazon EC2 Auto Scaling 容量再平衡。
  - Spot 节点收到再平衡建议时，Amazon EKS 会自动尝试启动新的替换 Spot 节点。
  - 如果 Spot 两分钟中断通知在替换 Spot 节点进入 `Ready` 状态之前到达时间，Amazon EKS 将开始耗尽收到再平衡建议的 Spot 节点。Amazon EKS 会尽最大努力耗尽节点。因此，无法保证 Amazon EKS 会等待替换节点加入集群后再耗尽现有节点。
  - 当替换 Spot 节点引导启动并在 Kubernetes 中处于 `Ready` 状态后，Amazon EKS 将封锁并耗尽收到再平衡建议的 Spot 节点。封锁 Spot 节点可确保服务控制器不会向此 Spot 节点发送任何新请求。它还将其从其运行状况良好的活动 Spot 节点列表中删除。耗尽 Spot 节点可确保正在运行的容器组（pod）被自然地逐出。
- Amazon EKS 会向您指定了容量类型的托管节点组中的所有节点添加以下 Kubernetes 标注：`eks.amazonaws.com/capacityType: SPOT`。您可以使用此标注在 Spot 节点上安排容错应用程序。

> **Important**
>
> EC2 会在终止您的竞价型实例前两分钟发出竞价型实例中断通知。但是，竞价型实例节点上的容器组（pod）可能无法获得完整的 2 分钟正常关闭时段。当 EC2 发出通知时，Amazon

EKS 开始驱逐容器组（pod）之前会有延迟。驱逐按顺序进行，以保护 Kubernetes API 服务器，因此在同时进行多次竞价型实例回收期间，某些容器组（pod）可能会收到延迟驱逐的通知。容器组（pod）可能会在未接收到终止信号的情况下被强制终止，尤其是在高密度的容器组（pod）节点上、并发回收期间或使用较长的终止宽限期时。对于竞价型实例工作负载，建议将应用程序设计为可承受中断，使用 30 秒或更短的终止宽限期，避免长时间运行的 PreStop 钩子，并监控容器组（pod）驱逐指标以了解集群中的实际宽限期。对于需要保证正常终止的工作负载，建议改用按需容量。

在决定是部署具有按需容量还是 Spot 容量的节点组时，应考虑以下条件：

- Spot 实例适合无状态、容错且灵活的应用程序。其中包括批处理和机器学习培训工作负载、大数据 ETL（例如 Apache Spark）、队列处理应用程序和无状态 API 端点。由于 Spot 是备用 Amazon EC2 容量，这些容量可能会随时间变化，因此我们建议您对可容忍中断的工作负载使用 Spot 容量。更具体地说，Spot 容量适合可以容忍所需容量不可用的时段的工作负载。
- 我们建议您为不容错的应用程序使用 On-Demand。这包括集群管理工具，例如监控和操作工具，需要 `StatefulSets` 的部署，以及有状态的应用程序，如数据库。
- 为了在使用 Spot 实例时最大限度地提高应用程序的可用性，我们建议您将 Spot 托管节点组配置为使用多个实例类型。我们建议在使用多个实例类型时应用以下规则：
  - 在托管节点组中，如果使用 Cluster Autoscaler，我们建议使用具有相同数量 vCPU 和内存资源的灵活实例类型集。这是为了确保集群中的节点按预期进行扩缩。例如，如果需要 4 个 vCPU 和 8 GiB 内存，请使用 `c3.xlarge`、`c4.xlarge`、`c5.xlarge`、`c5d.xlarge`、`c5a.xlarge`、`c5n.xlarge` 或其他类似实例类型。
  - 为了提高应用程序可用性，我们建议部署多个 Spot 托管节点组。为此，每个组应使用一组具有相同 vCPU 和内存资源的灵活实例类型。例如，如果您需要 4 个 vCPU 和 8 GiB 内存，我们建议您使用 `c3.xlarge`、`c4.xlarge`、`c5.xlarge`、`c5d.xlarge`、`c5a.xlarge`、`c5n.xlarge` 或其他类似的实例类型创建一个托管节点组，并使用 `m3.xlarge`、`m4.xlarge`、`m5.xlarge`、`m5d.xlarge`、`m5a.xlarge`、`m5n.xlarge` 或其他类似的实例类型创建第二个托管节点组。
  - 用使用自定义启动模板的 Spot 容量类型部署节点组时，请使用 API 传递多个实例类型。不要通过启动模板传递单个实例类型。有关使用启动模板部署节点组的更多信息，请参阅 the section called “启动模板”。

为集群创建托管式节点组

本主题介绍了如何启动向 Amazon EKS 集群注册的节点的 Amazon EKS 托管节点组。在这些节点加入集群后，您可以向其部署 Kubernetes 应用程序。

如果这是您首次启动 Amazon EKS 托管节点组，建议您改为遵循开始使用中的指南之一。这些指南提供了有关创建包含节点的 Amazon EKS 集群的演练。

> **Important**
>
> - Amazon EKS 节点是标准的 Amazon EC2 实例。将根据正常的 Amazon EC2 价格向您计费。有关更多信息，请参阅 Amazon EC2 定价。
> - 您无法在启用了 Amazon Outposts 或 Amazon Wavelength 的 Amazon 区域创建托管节点。您可以创建自行管理的节点。有关更多信息，请参阅 the section called “Amazon Linux”、the section called “Windows” 和 the section called “Bottlerocket”。您还可以在 Outpost 上创建自行管理的 Amazon Linux 节点组。有关更多信息，请参阅 the section called “Nodes”。
> - 如果您没有为包含在 Amazon EKS 优化版 Linux 或 Bottlerocket 中的 `bootstrap.sh` 文件指定 AMI ID，则托管节点组会对 `maxPods` 的值强制实施最大数量。对于 vCPU 少于 30 个的实例，最大数量为 `110`。对于 vCPU 大于 30 个的实例，最大数量将跳至 `250`。这些数字基于 Kubernetes 可扩展性阈值以及内部 Amazon EKS 可扩展性团队测试的推荐设置。有关更多信息，请参阅博客文章 Amazon VPC CNI 插件提高每个节点的容器组限制。

- 现有 Amazon EKS 集群。要部署一个角色，请参阅the section called “创建集群”。
- 供节点使用的现有 IAM 角色。要创建该文件，请参阅 the section called “节点 IAM 角色”。如果此角色没有 VPC CNI 的任一策略，则需要为 VPC CNI Pod 使用随后的单独角色。
- （可选，但推荐）适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件已配置自己的 IAM 角色，并附加了必要的 IAM 策略。有关更多信息，请参阅 the section called “配置 IRSA”。
- 熟悉选择最优的 Amazon EC2 节点实例类型中列出的注意事项。根据您选择的实例类型，您的集群和 VPC 可能还有其他先决条件。
- 要添加 Windows 托管节点组，必须先启用对集群的 Windows 支持。有关更多信息，请参阅 the section called “启用 Windows 支持”。

您可以使用以下任一途径创建托管节点组：

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”

eksctl

使用 eksctl 创建托管节点组

此过程需要 eksctl 版本 0.215.0 或更高版本。可以使用以下命令来查看您的版本：

```
eksctl version
```

有关安装或升级 eksctl 的说明，请参阅 eksctl 文档中的 Installation。

1. （可选）如果 AmazonEKS_CNI_Policy 托管的 IAM 策略附加到 Amazon EKS 节点 IAM 角色，我们建议将其分配给您与 Kubernetes aws-node 服务账户关联的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。
2. 使用或不使用自定义启动模板创建托管节点组。手动指定启动模板可允许对节点组进行更好的自定义。例如，它可以允许部署自定义 AMI 或向 Amazon EKS 优化的 AMI 中的 boostrap.sh 脚本提供参数。要查看所有可用选项和原定设置的完整列表，请输入以下命令。

   ```
   eksctl create nodegroup --help
   ```

   在以下命令中，将 *my-cluster* 替换为您的集群名称，并将 *my-mng* 替换为您的节点组名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。

   > **Important**
   >
   > 如果首次创建托管节点组时没有使用自定义启动模板，则以后不要对节点组使用模板。如果没有指定自定义启动模板，系统会自动生成启动模板，我们不建议您手动修改该模板。手动修改此自动生成的启动模板可能会导致错误。

不使用启动模板

eksctl 在您的账户中创建默认的 Amazon EC2 启动模板，并使用它根据您指定的选项创建的启动模板来部署节点组。在为 --node-type 指定值之前，请参阅 the section called “Amazon EC2 实例类型”。

将 *ami-family* 替换为允许的关键字。有关更多信息，请参阅 eksctl 文档中的 Setting the node AMI Family（设置节点 AMI 系列）。将 *my-key* 替换为您的 Amazon EC2 密钥对或公有密钥的名称。此密钥用于在节点启动后通过 SSH 进入节点。

> **Note**
>
> 对于 Windows，此命令不会启用 SSH。反之，它会将 Amazon EC2 密钥对与实例关联，并允许您 RDP 到实例中。

如果还没有 Amazon EC2 密钥对，可以在 Amazon Web Services 管理控制台 中创建一个。有关 Linux 信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对和 Linux 实例。有关 Windows 信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对和 Windows 实例。

如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：

- 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
- 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

有关更多信息，请参阅限制对分配给工作节点的实例配置文件的访问。

如果要阻止容器组（pod）访问 IMDS，请将 `--disable-pod-imds` 选项添加到以下命令。

```
eksctl create nodegroup \
  --cluster my-cluster \
  --region region-code \
  --name my-mng \
  --node-ami-family ami-family \
  --node-type m5.large \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 4 \
  --ssh-access \
  --ssh-public-key my-key
```

实例可以选择为容器组（pod）分配更多 IP 地址，为其他 CIDR 块（而不是实例的 CIDR 块）中的容器组（pod）分配 IP 地址，并将其部署到没有互联网访问的集群。有关更多信息，请参阅 the section

called “增加 IP 地址”、the section called “自定义联网” 和 the section called “私有集群”，以获取要添加到上一个命令中的其他选项。

托管节点组将根据实例类型计算并应用单个值，以作为可以在节点组的每个节点上运行的最大容器组（pod）数量。如果创建具有不同实例类型的节点组，则在所有实例类型中计算得出的最小值将应用为可以在节点组中每种实例类型上运行的最大容器组（pod）数量。托管节点组会使用 Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量中引用的脚本计算值。

使用启动模板

启动模板必须已存在，并且必须满足启动模板配置基础知识中指定的要求。如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：

- 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
- 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

有关更多信息，请参阅限制对分配给工作节点的实例配置文件的访问。

如果要阻止容器组（pod）访问 IMDS，请在启动模板中指定必要的设置。

a. 将以下内容复制到您的设备。替换示例值，然后运行修改后的命令以创建 `eks-nodegroup.yaml` 文件。在不使用启动模板的情况下进行部署时指定的多个设置将移动到启动模板中。如果未指定 `version`，则使用模板的默认版本。

```
cat >eks-nodegroup.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code
managedNodeGroups:
- name: my-mng
  launchTemplate:
    id: lt-id
    version: "1"
EOF
```

有关 eksctl 配置文件设置的完整列表，请参阅 eksctl 文档中的配置文件架构。实例可以选择将更多的 IP 地址分配到容器组（pod），将 IP 地址分配到其他 CIDR 块（而不是实例的 CIDR 块）中的容器组（pod），并将其部署到没有出站互联网访问的集群。有关更多信息，请参阅the section called “增加 IP 地址”、the section called “自定义联网”和the section called “私有集群”，以获取添加到配置文件的其他选项。

如果没有在启动模板中指定 AMI ID，则托管节点组将根据实例类型计算并应用单个值，以作为可以在节点组的每个节点上运行的最大容器组（pod）数量。如果创建具有不同实例类型的节点组，则在所有实例类型中计算得出的最小值将应用为可以在节点组中每种实例类型上运行的最大容器组（pod）数量。托管节点组会使用 Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量中引用的脚本计算值。

如果在启动模板中指定了 AMI ID，请指定可以在节点组的每个节点上运行的最大容器组（pod）数量（如果您使用的是自定义网络或者想要增加分配给实例的 IP 地址数量）。有关更多信息，请参阅the section called “Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量”。

b. 使用以下命令部署节点组。

```
eksctl create nodegroup --config-file eks-nodegroup.yaml
```

Amazon Web Services 管理控制台

使用 Amazon Web Services 管理控制台创建托管节点组

1. 等待集群状态显示为 ACTIVE。无法为状态尚未处于 ACTIVE 的集群创建托管节点组。
2. 打开 Amazon EKS 控制台。
3. 选择要在其中创建托管节点组的集群的名称。
4. 选择 Compute（计算）选项卡。
5. 请选择 Add node group（添加节点组）。
6. 在 Configure node group (配置节点组) 页面上，填写相应参数，然后选择 Next (下一步)。
   - 名称 – 为托管节点组输入唯一名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。
   - 节点 IAM 角色 – 选择要与节点组一起使用的节点实例角色。有关更多信息，请参阅 the section called “节点 IAM 角色”。

> **Important**
>
> - 您不能使用创建任何集群时使用的相同角色。
> - 我们建议使用任何自行管理节点组当前未使用的角色。否则，计划与新的自行管理节点组配合使用。有关更多信息，请参阅 the section called “删除”。

- 使用启动模板 -（可选）选择是否要使用现有启动模板。选择 Launch Template Name（启动模板名称）。然后，选择 Launch template version（启动模板版本）。如果您未选择版本，Amazon EKS 将使用模板的默认版本。启动模板允许您对节点组进行更多自定义，例如允许您部署自定义 AMI、将更多的 IP 地址分配给容器组（pod）、将 IP 地址分配给其他 CIDR 块（而不是实例的 CIDR 块）中的容器组（pod），并将节点部署到没有出站互联网访问的集群。有关更多信息，请参阅 the section called “增加 IP 地址”、the section called “自定义联网” 和 the section called “私有集群”。

  启动模板必须满足中的使用启动模板自定义托管式节点中的要求。如果您不使用自己的启动模板，Amazon EKS API 会在您的账户中创建默认 Amazon EC2 启动模板，并使用默认启动模板部署节点组。

  如果实施服务账户的 IAM 角色，请将必要的权限直接分配给需要访问 Amazon 服务的所有容器组（pod）。如果集群中没有容器组（pod）出于其他原因（例如检索当前 Amazon 区域）而需要访问 IMDS，您还可以在启动模板中为不使用主机网络的容器组（pod）禁用 IMDS 的访问权限。有关更多信息，请参阅限制对分配给工作节点的实例配置文件的访问。
- Kubernetes 标签 –（可选）您可以选择对托管节点组中的节点应用 Kubernetes 标签。
- Kubernetes 污点 –（可选）您可以选择对托管节点组中的节点应用 Kubernetes 污点。Effect（效果）菜单中的可用选项包括 **`NoSchedule`**、**`NoExecute`** 和 **`PreferNoSchedule`**。有关更多信息，请参阅 the section called “防止在特定节点上调度容器组（pod）”。
- 标签 –（可选）您可以选择对 Amazon EKS 托管节点组进行标记。这些标签不会传播到节点组中的其它资源，例如自动扩缩组或实例。有关更多信息，请参阅 the section called “标记您的资源”。

7. 在 Set compute and scaling configuration（设置计算和扩展配置）页面上，填写相应参数，然后选择 Next（下一步）。
   - AMI 类型 – 选择一个 AMI 类型。如果要部署 Arm 实例，请务必在部署前查看 Amazon EKS 优化版 Arm Amazon Linux AMI 中的注意事项。

如果您在上一页指定了启动模板，并在启动模板中指定了 AMI，则无法选择值。此时将显示模板中的值。模板中指定的 AMI 必须满足指定 AMI 中的要求。

- 容量类型 – 选择容量类型。有关选择容量类型的更多信息，请参阅 the section called “托管节点组容量类型”。不能在同一节点组中混合使用不同的容量类型。如果要同时使用这两种容量类型，请创建单独的节点组，每个节点组都有自己的容量和实例类型。有关预置和扩展由 GPU 加速的 Worker 节点的信息，请参阅为托管节点组预留 GPU。
- 实例类型 – 默认指定一个或多个实例类型。要删除默认实例类型，请选择实例类型右侧的 X。选择要在托管节点组中使用的实例类型。有关更多信息，请参阅 the section called “Amazon EC2 实例类型”。

  控制台显示一组常用的实例类型。如果需要使用未显示的实例类型创建托管节点组，请使用 `eksctl`、Amazon CLI、Amazon CloudFormation 或 SDK 创建节点组。如果在上一页指定了启动模板，则无法选择值，因为必须在启动模板中指定实例类型。将显示启动模板中的值。如果为容量类型选择了 Spot 实例，我们建议您指定多个实例类型以增强可用性。
- 磁盘大小 – 输入要用于节点根卷的磁盘大小（单位为 GiB）。

  如果在上一页指定了启动模板，则无法选择值，因为必须在启动模板中指定该值。
- 所需大小 – 指定托管节点组在启动时应当维持的当前节点数量。

  > **Note**
  >
  > Amazon EKS 不会自动扩展或缩减节点组。但是，您可以配置 Kubernetes Cluster Autoscaler 来为您执行此操作。有关更多信息，请参阅 Amazon 上的 Cluster Autoscaler。
- 最小大小 – 指定托管节点组可以横向缩减到的最小节点数量。
- 最大大小 – 指定托管节点组可以横向扩展到的最大节点数量。
- 节点组更新配置 – （可选）您可以选择要并行更新的节点的数量或百分比。这些节点在更新期间将不可用。对于最大不可用，选择下列选项之一，然后指定一个值：
  - Number（数字）– 选择并指定节点组中可以并行更新的节点数。
  - Percentage（百分比）– 选择并指定节点组中可并行更新的节点的百分比。如果您的节点组中有大量节点，这将非常有用。
- 节点自动修复配置 – （可选）如果您激活了启用节点自动修复复选框，Amazon EKS 将在检测到问题时自动替换节点。有关更多信息，请参阅 the section called “节点运行状况”。

8. 在 Specify networking（指定联网）页面上，相应填写参数，然后选择 Next（下一步）。

- 子网 – 选择要在其中启动托管节点的子网。

  > **Important**
  >
  > 如果要使用 Kubernetes Cluster Autoscaler 在由 Amazon EBS 卷支持的多个可用区中运行有状态应用程序，则应该配置多个节点组，且每个节点组的范围都限定为一个可用区。此外，您还应该启用 `--balance-similar-node-groups` 功能。

  > **Important**
  >
  - 如果您选择公有子网，并且您的集群仅启用公有 API 服务器端点，则子网必须将 `MapPublicIPOnLaunch` 设置为 `true`，实例才能成功加入集群。如果子网是使用 `eksctl` 或 Amazon EKS 发布的 Amazon CloudFormation 模板在 2020 年 3 月 26 日或之后创建的，则此设置已设置为 `true`。如果子网是在 2020 年 3 月 26 日之前使用 `eksctl` 或 Amazon CloudFormation 模板创建的，则需要手动更改设置。有关更多信息，请参阅修改子网的公有 IPv4 寻址属性。
  - 如果使用启动模板并指定多个网络接口，即使 `MapPublicIpOnLaunch` 设置 `true`，Amazon EC2 也不会自动分配公有 `IPv4` 地址。在这种情况下，要让节点加入集群，您必须启用集群的私有 API 服务器端点，或者在具有出站 Internet 访问的私有子网中启动节点（Internet 访问通过如 NAT 网关等其它方法提供）。有关更多信息，请参阅 Amazon EC2 用户指南中的 Amazon EC2 实例 IP 寻址。

- Configure SSH access to nodes（配置对节点的 SSH 访问）（可选）。启用 SSH 后，如果出现问题，您可以连接到实例并收集诊断信息。我们强烈建议您在创建节点组时启用远程访问。创建节点组后，将无法启用远程访问。

  如果您选择使用启动模板，则不会显示此选项。要启用对节点的远程访问，请在启动模板中指定密钥对，并确保为您在启动模板中指定的安全组中的节点打开正确的端口。有关更多信息，请参阅 the section called “使用自定义安全组”。

  > **Note**
  >
  > 对于 Windows，此命令不会启用 SSH。反之，它会将 Amazon EC2 密钥对与实例关联，并允许您 RDP 到实例中。

- 对于 SSH 密钥对（可选），请选择要使用的 Amazon EC2 SSH 密钥。有关 Linux 信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对和 Linux 实例。有关 Windows 信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对和 Windows 实例。如果您选择使用启动模板，则无法选择密钥对。使用 Bottlerocket AMI 为节点组提供 Amazon EC2 SSH 密钥后，还启用管理容器。有关更多信息，请参阅 GitHub 上的 Admin 容器。
- 对于 Allow SSH remote access from（允许来自以下的远程访问），如果要限制对特定实例的访问，请选择与这些实例关联的安全组。如果没有选择特定的安全组，则允许从 Internet 上的任何位置进行 SSH 访问（`0.0.0.0/0`）。

9. 在 Review and create (审核并创建) 页面上，审核托管节点组配置并选择 Create (创建)。

   如果节点无法加入集群，则请参阅故障排除一章中的 the section called “节点未能加入集群”。

10查看节点的状态并等待它们达到 `Ready` 状态。

```
kubectl get nodes --watch
```

11.（仅限 GPU 节点）如果选择 GPU 实例类型和 Amazon EKS 优化版加速型 AMI，则必须将适用于 Kubernetes 的 NVIDIA 设备插件用作集群上的 DaemonSet。将 *vX.X.X* 替换为您需要的 NVIDIA/k8s-device-plugin 版本，然后运行以下命令。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

安装 Kubernetes 附加组件

现在，您已有使用节点运行的 Amazon EKS 集群，那么就可以准备开始安装 Kubernetes 附加组件并将应用程序部署到集群。以下文档主题可帮助您扩展集群的此功能。

- 创建集群的 IAM 主体是唯一可以使用 `kubectl` 或 Amazon Web Services 管理控制台 调用 Kubernetes API 服务器的主体。如果您希望其他 IAM 主体拥有访问您的集群的权限，您需要添加它们。有关更多信息，请参阅the section called “Kubernetes API 访问权限”和the section called “所需的权限”。
- 如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：
  - 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
  - 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

- 配置 Kubernetes Cluster Autoscaler 以自动调整节点组中的节点数。
- 将示例应用程序部署到您的集群。
- 使用管理集群的重要工具来组织和监控集群资源。

更新集群的托管式节点组

启动托管节点组更新后，Amazon EKS 会自动更新您的节点，完成了解节点更新的各个阶段中列出的步骤。如果您使用的是 Amazon EKS 优化版 AMI，Amazon EKS 会自动将最新的安全补丁和操作系统更新应用到您的节点，作为最新 AMI 版本的一部分。

在几种情况下，需要更新 Amazon EKS 托管节点组的版本或配置：

- 您已更新 Amazon EKS 集群的 Kubernetes 版本，并且您希望更新节点以使用相同 Kubernetes 版本。
- 新 AMI 发行版本可用于您的托管节点组。有关 AMI 版本的更多信息，请参阅以下章节：
  - the section called “获取版本信息”
  - the section called “Bottlerocket”
  - the section called “获取版本信息”
- 您希望调整托管节点组中的最少、最多或所需实例数。
- 您希望对托管节点组中的实例添加或删除 Kubernetes 标签。
- 您希望对托管节点组添加或删除 Amazon 标签。
- 您需要部署包含配置更改（如更新的自定义 AMI）的最新版本启动模板。
- 您已部署了版本 `1.9.0` 或更高版本的 Amazon VPC CNI 附加组件，启用了前缀委派附加组件，并希望节点组中的新 Amazon Nitro 实例支持数量显著增加的容器组（pod）。有关更多信息，请参阅 the section called “增加 IP 地址”。
- 您已为 Windows 节点启用了 IP 前缀委派，并希望节点组中的新 Amazon Nitro System 实例支持数量显著增加的容器组（pod）。有关更多信息，请参阅 the section called “增加 IP 地址”。

如果有相比您托管节点组的 Kubernetes 版本更新的 AMI 发行版，则可以将您节点组的版本更新为使用较新的 AMI 版本。同理，如果集群运行的 Kubernetes 版本比节点组更新，则可以将节点组更新为使用与集群 Kubernetes 版本匹配的最新 AMI 发行版。

如果托管节点组中的节点因扩展操作或更新而终止，则会先耗尽该节点中的容器组（pod）。有关更多信息，请参阅 the section called “更新行为详细信息”。

更新节点组版本

您可以使用以下任一方式更新节点组版本：

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”

您更新到的版本不能高于控制面板的版本。

eksctl

使用 eksctl 更新托管节点组

使用以下命令，将托管节点组更新到节点上当前部署的相同 Kubernetes 版本的最新 AMI 版本。将所有 *example value* 替换为您自己的值。

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
  --region=region-code
```

> **Note**
>
> 如果要将使用启动模板部署的节点组升级到新的启动模板版本，请将 `--launch-template-version` *version-number* 添加到上述命令。启动模板必须满足使用启动模板自定义托管节点中所述的要求。如果启动模板包含自定义 AMI，则 AMI 必须满足指定 AMI 中的要求。将节点组升级到启动模板的更新版本后，将回收所有节点，以匹配指定的启动模板版本的新配置。

您不能将未使用启动模板部署的节点组直接升级到新的启动模板版本。相反，您必须使用启动模板部署新的节点组，以将节点组更新为新的启动模板版本。

您可以将节点组升级到与控制面板的 Kubernetes 版本相同的版本。例如，如果您的集群运行 Kubernetes `1.33`，则您可以使用以下命令，将当前运行 Kubernetes `1.32` 的节点升级到版本 `1.33`。

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
  --region=region-code \
  --kubernetes-version=1.33
```

Amazon Web Services 管理控制台

要使用 Amazon Web Services 管理控制台创建托管节点组

1. 打开 Amazon EKS 控制台。
2. 选择包含要更新的节点组的集群。
3. 如果至少有一个节点组具有可用更新，则页面顶部将出现一个提示框，通知存在可用的更新。如果选择计算选项卡，则在具有可用更新的节点组的节点组表中，AMI 发行版本列会显示立即更新。要更新节点组，请选择 Update now（立即更新）。

   对于使用自定义 AMI 部署的节点组，不会显示通知。如果您的节点是使用自定义 AMI 部署的，请完成以下步骤以部署更新的自定义 AMI。

   a. 创建 AMI 的新版本。

   b. 使用新 AMI ID 创建新的启动模板版本。

   c. 将节点升级到新版本的启动模板。
4. 在 Update node group version（更新节点组版本）对话框中，激活或停用以下选项：
   - Update node group version（更新节点组版本）：如果您部署了自定义 AMI，或者您的 Amazon EKS 优化版 AMI 当前位于集群的最新版本上，则此选项不可用。
   - Change launch template version（更改启动模板版本）：如果部署节点组时没有使用自定义启动模板，则此选项不可用。您只能更新已使用自定义启动模板部署的节点组的启动模板版本。选择要将节点组更新到的 Launch template version（启动模板版本）。如果您的节点组配置了自定义 AMI，则您选择的版本还必须指定 AMI。升级到更新版本的启动模板后，所有节点都会被回收，以匹配指定的启动模板版本的新配置。
5. 对于 Update strategy（更新策略），选择下列选项之一：
   - 滚动更新 – 此选项会考虑集群的容器组（pod）中断预算。如果容器组（pod）中断预算问题导致 Amazon EKS 无法正常耗尽在此节点组上运行的容器组（pod），则更新将失败。
   - 强制更新 – 此选项不考虑容器组（pod）中断预算。通过强制节点重新启动，无论是否存在容器组（pod）中断预算问题，都会发生更新。
6. 选择更新。

编辑节点组配置

您可以修改托管节点组的某些配置。

1. 打开 Amazon EKS 控制台。
2. 选择包含要编辑的节点组的集群。
3. 选择计算选项卡。
4. 选择要编辑的节点组，然后选择 Edit（编辑）。
5. （可选）在编辑节点组页面上，执行以下操作：
   a. 编辑 Node group scaling configuration（节点组扩展配置）。
      - 所需大小 – 指定托管节点组应当维持的当前节点数量。
      - 最小大小 – 指定托管节点组可以横向缩减到的最小节点数量。
      - 最大大小 – 指定托管节点组可以横向扩展到的最大节点数量。有关节点组中支持的最大节点数，请参阅 the section called “服务配额”。
   b. （可选）向节点组中的节点添加或删除 Kubernetes 标签。此处显示的标签仅为已应用于 Amazon EKS 的标签。节点上可能存在此处未显示的其它标签。
   c. （可选）向节点组中的节点添加或删除 Kubernetes 污点。添加的污点可以具有 **`NoSchedule`**、**`NoExecute`** 或 **`PreferNoSchedule`** 效果。有关更多信息，请参阅 the section called “防止在特定节点上调度容器组（pod）”。
   d. （可选）向节点组资源添加或删除 Tags（标签）。这些标签仅应用于 Amazon EKS 节点组。这些标签不会传播到其它资源，例如节点组中的子网或 Amazon EC2 实例。
   e. （可选）编辑节点组更新配置。选择数字或百分比。
      - Number（数字）– 选择并指定节点组中可以并行更新的节点数。这些节点在更新过程中将不可用。
      - Percentage（百分比）– 选择并指定节点组中可并行更新的节点的百分比。这些节点在更新过程中将不可用。如果您的节点组中有大量节点，这将非常有用。
   f. 编辑完成后，选择保存更改。

> **Important**
>
> 更新节点组配置时，修改 `NodegroupScalingConfig` 不会考虑容器组（pod）中断预算（PDB）。与更新节点组进程（在升级阶段耗尽节点并考虑 PDB）不同，更新扩展配置会导致通过自动扩缩组（ASG）缩减调用立即终止节点。这种情况在不考虑 PDB 的情况下发生，

无论缩减到什么目标大小。这意味着，当您减少 Amazon EKS 托管节点组的 `desiredSize` 时，只要节点终止，容器组（pod）就会立即被驱逐，而不考虑任何 PDB。

了解节点更新的各个阶段

Amazon EKS 托管 Worker 节点升级策略有四个不同的阶段，将在以下各节中介绍。

设置阶段

设置阶段有以下步骤：

1. 为与节点组关联的自动扩缩组创建新的 Amazon EC2 启动模板版本。新的启动模板版本使用目标 AMI 或自定义启动模板版本进行更新。
2. 对自动扩缩组进行更新，以便使用最新的启动模板版本。
3. 使用节点组的 `updateConfig` 属性确定要并行升级的节点最大数量。最大不可用的配额为 100 个节点。原定设置值为一个节点。有关更多信息，请参阅《Amazon EKS API 参考》中的 updateConfig 属性。

扩展阶段

升级托管节点组中的节点时，已升级的节点将在与正在升级的节点在相同可用区中启动。为了保证这一点，我们使用 Amazon EC2 的可用区重新平衡。有关更多信息，请参阅 Amazon EC2 Auto Scaling 用户指南的可用区重新平衡。为了满足此要求，将为托管节点组中的每个可用区启动最多两个实例。

扩展阶段有以下步骤：

1. 它增加自动扩缩组的最大大小和所需大小，增加幅度为以下两者中的较大者：
   - 部署自动扩缩组可用区数量的最多两倍。
   - 升级不可用的最大值。

   例如，如果节点组有五个可用区，`maxUnavailable` 为 1，则升级过程可以启动最多 10 个节点。但是如果 `maxUnavailable` 为 20（或者任何高于 10 的值），则该过程将启动 20 个新节点。
2. 扩展 Auto Scaling 组后，将检查节点组中是否存在使用最新配置的节点。只有在满足以下标准时，此步骤才会成功：
   - 在节点所在的每个可用区中启动至少一个新节点。

- 每个新节点都应该处于 Ready 状态。
- 新节点应该有应用 Amazon EKS 的标签。

  以下是常规节点组中的 Worker 节点上应用 Amazon EKS 的标签：

  - eks.amazonaws.com/nodegroup-image=$amiName
  - eks.amazonaws.com/nodegroup=$nodeGroupName

  以下是自定义启动模板或 AMI 节点组中的 Worker 节点上应用 Amazon EKS 的标签：

  - eks.amazonaws.com/nodegroup-image=$amiName
  - eks.amazonaws.com/nodegroup=$nodeGroupName
  - eks.amazonaws.com/sourceLaunchTemplateId=$launchTemplateId
  - eks.amazonaws.com/sourceLaunchTemplateVersion=$launchTemplateVersion

3. 将节点标记为不可调度，避免调度新的容器组（pod）。它还使用 node.kubernetes.io/exclude-from-external-load-balancers=true 标记节点，以便在终止节点之前从负载均衡器中移除旧节点。

下面是导致这一阶段 NodeCreationFailure 出现错误的已知原因：

可用区中容量不足

可用区可能没有请求的实例类型的容量。建议在创建托管节点组时配置多种实例类型。

账户的 EC2 实例限制

可能需要使用服务配额增加账户可以同时运行的 Amazon EC2 实例数量。有关更多信息，请参阅适用于 Linux 实例的 Amazon Elastic Compute Cloud 用户指南中的 EC2 Service Quotas。

自定义用户数据

自定义用户数据有时会中断引导过程。这种情况可能导致节点不启动 kubelet，或者节点上没有获得预期的 Amazon EKS 标签。有关更多信息，请参阅 the section called “指定 AMI”。

使节点不正常或未就绪的任何更改

节点磁盘压力、内存压力和类似情况可能导致节点无法进入 Ready 状态。

每个节点的引导时间最多不超过 15 分钟

如果任何节点引导和加入集群的时间超过 15 分钟，则会导致升级超时。这是从需要新节点开始到新节点加入集群为止测量的引导新节点的总运行时间。升级托管节点组时，时间计数器会在自动扩缩组的大小增加后启动。

升级阶段

升级阶段有两种行为方式，具体取决于更新策略。更新策略分为默认策略和最小策略这两种。

建议在大多数情况下使用默认策略。该策略会在终止旧节点之前创建新节点，这样就能在升级阶段保持可用容量。在资源或成本受限的情况下，例如使用 GPU 等硬件加速器，最小策略非常有用。该策略会在创建新节点之前终止旧节点，这样总容量永远不会超过配置的数量。

默认更新策略包含如下步骤：

1. 增加自动扩缩组中的节点数量（所需数量），导致该节点组创建更多节点。
2. 它随机选择一个需要更新的节点，最多为该节点组配置的最大不可用性。
3. 耗尽节点的容器组（pod）。如果容器组（pod）在 15 分钟内没有离开节点并且没有强制标志，则升级阶段将失败并显示 PodEvictionFailure 错误。对于这种情况，您可以使用 update-nodegroup-version 请求应用强制标志来删除容器组（pod）。
4. 在每个容器组（pod）被驱逐后封锁节点并等待 60 秒钟。这样做是为了防止服务控制器向此节点发送任何新请求，并将此节点从活动节点列表中删除。
5. 它向封锁节点的自动扩缩组发送终止请求。
6. 它重复以前的升级步骤，直到节点组中没有使用早期版本启动模板部署的节点。

最小更新策略包括如下步骤：

1. 首先会封锁节点组中的所有节点，确保服务控制器不会向这些节点发送任何新请求。
2. 它随机选择一个需要更新的节点，最多为该节点组配置的最大不可用性。
3. 耗尽选定节点的容器组（pod）。如果容器组（pod）在 15 分钟内没有离开节点并且没有强制标志，则升级阶段将失败并显示 PodEvictionFailure 错误。对于这种情况，您可以使用 update-nodegroup-version 请求应用强制标志来删除容器组（pod）。
4. 将所有容器组弹出并等待 60 秒后，它会向选定节点的自动扩缩组发出一条终止请求。该自动扩缩组会创建新节点（数量与选定的节点数相同）来替换缺失的容量。
5. 它重复以前的升级步骤，直到节点组中没有使用早期版本启动模板部署的节点。

升级期间的 **PodEvictionFailure** 错误

下面是导致这一阶段 PodEvictionFailure 出现错误的已知原因：

积极的 PDB

积极的 PDB 在容器组（pod）上定义，或者有多个 PDB 指向同一个容器组（pod）。

容忍所有污点的部署

一旦每个容器组（pod）被逐出，预计该节点将为空，因为该节点在前面的步骤中受到污染。但是，如果部署容忍每个污点，则该节点更有可能是非空的，导致容器组（pod）驱逐失败。

缩减阶段

缩减阶段将自动扩缩组的最大大小和所需大小减小一，返回更新开始之前的值。

如果升级工作流确定 Cluster Autoscaler 在工作流缩减阶段扩展节点组，则会立即退出，而不会使节点组恢复原始大小。

使用启动模板自定义托管式节点

为实现最高级别的自定义，您可以根据此页面上的步骤使用自己的启动模板部署托管节点。使用启动模板可以实现以下功能：在部署节点期间提供引导参数（例如，额外的 kubelet 参数），从与分配给节点的 IP 地址不同的 CIDR 数据块为容器组（pod）分配 IP 地址，将自己的自定义 AMI 部署到节点，或者将自己的自定义 CNI 部署到节点。

如果您在首次创建托管节点组时提供自己的启动模板，则以后也将具有更大的灵活性。只要使用自己的启动模板部署托管节点组，您就可以使用同一启动模板的不同版本对其进行迭代更新。将节点组更新为启动模板的其他版本时，将回收组中的所有节点，以匹配指定启动模板版本的新配置。

托管式节点组始终部署用于 Amazon EC2 Auto Scaling 组的启动模板。当您不提供启动模板时，Amazon EKS API 会在您的账户中自动创建一个具有默认值的模板。但是，我们建议不要修改自动生成的启动模板。而且，无法直接更新不使用自定义启动模板的现有节点组。相反，您必须使用自定义启动模板创建新的节点组才能执行此操作。

启动模板配置基础知识

您可以使用 Amazon Web Services 管理控制台、Amazon CLI 或 Amazon SDK 创建 Amazon EC2 Auto Scaling 启动模板。有关更多信息，请参阅《Amazon EC2 Auto Scaling 用户指南》中的为自动扩缩组创建启动模板。启动模板中的某些设置类似于用于托管节点配置的设置。使用启动模板部署或更新节点组时，必须在节点组配置或启动模板中指定某些设置。不要在两个位置都指定一个设置。如果某个设置出现在错误的位置，则创建或更新节点组之类的操作将失败。

下表列出启动模板中禁止的设置，它还列出托管节点组配置中所需的类似设置（如果有）。列出的设置是显示在控制台中的设置。它们在 Amazon CLI 和 SDK 中可能具有相似但不同的名称。

启动模板 - 已禁止	Amazon EKS 节点组配置
Network interfaces（网络接口）（Add network interface（添加网络接口））下的Subnet（子网）	Specify networking（指定联网）页面上 Node group network configuration（节点组网络配置）下的 Subnets（子网）
Advanced details（高级详细信息）下的 IAM instance profile（IAM 实例配置文件）	Configure Node group（配置节点组）页面上 Node group configuration（节点组配置）下的 Node IAM role（节点 IAM 角色）
Advanced details（高级详细信息）下的 Shutdown behavior（关机行为）和 Stop - Hibernate behavior（停止 - 休眠行为）。在启动模板中，对于两个设置都保留默认的不包含在启动模板设置中。	无等效项 Amazon EKS 必须控制实例生命周期，而不是自动扩缩组。

下表列出托管节点组配置中禁止的设置，还列出启动模板中所需的类似设置（如果有）。列出的设置是显示在控制台中的设置。它们在 Amazon CLI 和 SDK 中可能有类似的名称。

Amazon EKS 节点组配置 – 已禁止	启动模板
（仅当在启动模板中指定了自定义 AMI 时）设置计算和扩缩配置页面上节点组计算配置下的 AMI 类型 – 控制台显示已在启动模板中指定和指定的 AMI ID。 如果未在启动模板中指定应用程序和操作系统映像（Amazon 机器映像），则可以在节点组配置中选择 AMI。	Launch template contents（启动模板内容）下的 Application and OS Images (Amazon Machine Image)（应用程序和操作系统映像（Amazon 机器映像））– 如果您有以下任一要求，则必须指定 ID： • 使用自定义 AMI。如果您指定的 AMI 不满足指定 AMI 中列出的要求，节点组部署将失败。 • 需要提供用户数据以将实际参数传递给随 Amazon EKS 优化版 AMI 一起提供的 `bootstrap.sh` 文件。可以允许实例为容器组（pod）分配明显更多的 IP 地址，为其他

Amazon EKS 节点组配置 – 已禁止	启动模板
	CIDR 块（而不是实例的 CIDR 块）中的容器组（pod）分配 IP 地址，或将其部署到没有出站网络访问权限的私有集群。有关更多信息，请参阅以下主题： • 为带前缀的 Amazon EKS 节点分配更多 IP 地址 • 使用自定义网络在备用子网中部署容器组（pod） • 部署具有有限互联网访问权限的私有集群 • 指定 AMI
Set compute and scaling configuration（设置计算和扩展配置）页面上 Node group compute configuration（节点组计算配置）下的 Disk size（磁盘大小）– 控制台显示 Specified in launch template（已在启动模板中指定）。	Storage (Volumes)（存储（卷））（Add new volume（添加新卷））下的 Size（大小）。您必须在启动模板中指定此项。
Specify Networking（指定网络）页面上 Node group configuration（节点组配置）下的 SSH key pair（SSH 密钥对）– 控制台显示在启动模板中指定的密钥或显示 Not specified in launch template（未在启动模板中指定）。	Key pair (login)（密钥对（登录））下的 Key pair name（密钥对名称。
在使用启动模板时，您无法指定允许远程访问的源安全组。	实例的 Network settings（网络设置）下的 Security groups（安全组），或者 Network interfaces（网络接口）（Add network interface（添加网络接口））下的 Security groups（安全组），但不能同时兼具。有关更多信息，请参阅 the section called “使用自定义安全组”。

> **Note**
>
> • 如果使用启动模板部署节点组，请在启动模板的 Launch template contents（启动模板内容）下指定 0 或 1 个 Instance type（实例类型）。您也可以为控制台 Set compute and

scaling configuration（设置计算和扩缩配置）页面的 Instance types（实例类型）指定 0-20 个实例类型。或者，您可以使用其他使用 Amazon EKS API 的工具来执行此操作。如果您在启动模板中指定实例类型，并使用该启动模板部署节点组，则无法在控制台中或通过使用 Amazon EKS API 的其它工具指定任何实例类型。如果没有在启动模板、控制台中或通过使用 Amazon EKS API 的其它工具指定实例类型，则使用 `t3.medium` 实例类型。如果您的节点组使用 Spot 容量类型，我们建议您使用控制台指定多个实例类型。有关更多信息，请参阅 the section called “托管节点组容量类型”。

- 如果部署到节点组的任何容器使用实例元数据服务版本 2，请确保在启动模板中将Metadata response hop limit（元数据响应跃点限制）设置为 2。有关更多信息，请参阅《Amazon EC2 用户指南》中的实例元数据和用户数据。
- 启动模板不支持 `InstanceRequirements` 功能，借助该功能可以灵活选择实例类型。

为 Amazon EC2 实例添加标签

您可以使用启动模板的 `TagSpecification` 参数来指定将哪些标签应用于节点组中的 Amazon EC2 实例。调用 `CreateNodegroup` 或 `UpdateNodegroupVersion` API 的 IAM 实体必须具有 `ec2:RunInstances` 和 `ec2:CreateTags` 的权限，并且必须将标签添加到启动模板中。

使用自定义安全组

您可以使用启动模板指定自定义 Amazon EC2 安全组以应用到节点组中的实例。这可以在实例级安全组参数中，也可以作为网络接口配置参数的一部分。但是，您无法创建同时指定实例级安全组和网络接口安全组的启动模板。请考虑以下适用于为托管节点组使用自定义安全组的条件：

- 使用 Amazon Web Services 管理控制台时，Amazon EKS 仅允许使用具有单个网络接口规范的启动模板。
- 预设情况下，Amazon EKS 将集群安全组应用于节点组中的实例，以便于节点和控制层面之间的通信。如果您使用前面提到的任一选项在启动模板中指定自定义安全组，则 Amazon EKS 不会添加集群安全组。因此，您必须确保安全组的入站和出站规则启用了与集群端点的通信。如果您的安全组规则不正确，则 worker 节点无法加入集群。有关安全组规则的更多信息，请参阅 the section called “安全组要求”。
- 如果需要对节点组中的实例进行 SSH 访问，请确保包含允许该访问的安全组。

Amazon EC2 用户数据

启动模板包括自定义用户数据的部分。您可以在此部分为节点组指定配置设置，而无需手动创建单个自定义 AMI。有关 Bottlerocket 可用设置的更多信息，请参阅 GitHub 上的使用用户数据。

在启动实例时，您可以使用 `cloud-init` 在启动模板中提供 Amazon EC2 用户数据。有关更多信息，请参阅 cloud-init 文档。您的用户数据可用于执行常见配置操作。其中包括以下操作：

- 包含用户或组
- 安装程序包

启动模板中用于托管节点组的 Amazon EC2 用户数据必须采用 MIME 分段归档格式（用于 Amazon Linux AMI）和 TOML 格式（用于 Bottlerocket AMI）。这是因为您的用户数据与节点加入集群所需的 Amazon EKS 用户数据合并。不要在用户数据中指定任何启动或修改 `kubelet` 的命令。这是作为 Amazon EKS 合并的用户数据的一部分执行的。某些 `kubelet` 参数（例如节点上的设置标签）可以直接通过托管节点组 API 进行配置。

> **Note**
>
> 如果需要高级 `kubelet` 自定义，包括手动启动或传入自定义配置参数，请参阅 the section called “指定 AMI”。如果在启动模板中指定自定义 AMI ID，Amazon EKS 不会合并用户数据。

以下详细信息提供有关用户数据部分的更多信息。

Amazon Linux 2 用户数据

您可以将多个用户数据块合并到一个 MIME 分段文件中。例如，您可以将配置 Docker 守护进程的云 Boothook 与安装自定义软件包的用户数据 Shell 脚本合并。MIME 分段文件包含以下组成部分：

- 内容类型和段边界声明 - `Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="`
- MIME 版本声明 - `MIME-Version: 1.0`
- 一个或多个用户数据块，其包含以下组成部分：
  - 开口边界，表示用户数据块的开头 - `--==MYBOUNDARY==`
  - 数据块的内容类型声明：`Content-Type: text/cloud-config; charset="us-ascii"`。有关内容类型的更多信息，请参阅 cloud-init 文档。
  - 用户数据的内容（例如 Shell 命令或 `cloud-init` 指令的列表）。

- 封闭边界，表示 MIME 分段文件的结尾：--==MYBOUNDARY==--

以下是 MIME 分段文件的示例，您可以用它来创建您自己的文件。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
echo "Running custom user data script"

--==MYBOUNDARY==--
```

Amazon Linux 2023 用户数据

Amazon Linux 2023（AL2023）引入了使用 YAML 配置架构的新节点初始化流程 nodeadm。如果您使用的是自行管理的节点组或带有启动模板的 AMI，则在创建新节点组时，现在需要明确提供其它集群元数据。以下是最低必需参数的示例，其中 apiServerEndpoint、certificateAuthority 和服务 cidr 是必需的：

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
```

通常，您将在用户数据中设置此配置，无论是按原样设置还是嵌入 MIME 多部分文档中：

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
```

```
kind: NodeConfig spec: [...]

--BOUNDARY--
```

在 AL2 中，来自这些参数的元数据是从 Amazon EKS `DescribeCluster` API 调用中发现的。使用 AL2023，这种行为发生了变化，因为在大型节点纵向扩展期间，额外的 API 调用有节流风险。如果您使用的是没有启动模板的托管节点组或者 Karpenter，则此更改不会对您产生影响。有关 `certificateAuthority` 和服务 `cidr` 的更多信息，请参阅《Amazon EKS API 参考》中的 `DescribeCluster`。

以下是一个完整的 AL2023 用户数据示例，该实例将用于自定义节点（例如安装程序包或预缓存容器映像）的 shell 脚本与所需的 nodeadm 配置进行组合。该示例展示了常见的自定义设置，包括：* 安装附加系统程序包 * 预缓存容器影响以缩短容器组（pod）启动时间 * 设置 HTTP 代理配置 * 配置用于节点标记的 `kubelet` 标志

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset

# Install additional packages
yum install -y htop jq iptables-services

# Pre-cache commonly used container images
nohup docker pull public.ecr.aws/eks-distro/kubernetes/pause:3.2 &

# Configure HTTP proxy if needed
cat > /etc/profile.d/http-proxy.sh << 'EOF'
export HTTP_PROXY="http://proxy.example.com:3128"
export HTTPS_PROXY="http://proxy.example.com:3128"
export NO_PROXY="localhost,127.0.0.1,169.254.169.254,.internal"
EOF

--BOUNDARY
Content-Type: application/node.eks.aws
```

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
  kubelet:
    config:
      clusterDNS:
      - 10.100.0.10
    flags:
    - --node-labels=app=my-app,environment=production

--BOUNDARY--
```

Bottlerocket 用户数据

Bottlerocket 采用 TOML 格式构建用户数据。您可以提供要与 Amazon EKS 提供的用户数据合并的用户数据。例如，您可以提供额外的 `kubelet` 设置。

```
[settings.kubernetes.system-reserved]
cpu = "10m"
memory = "100Mi"
ephemeral-storage= "1Gi"
```

有关支持设置的更多信息，请参阅 GitHub 上的 Bottlerocket 文档。您可以在用户数据中配置节点标签和污点。但是，我们建议在节点组中对其进行配置。在您这样做时，Amazon EKS 应用这些配置。

合并用户数据时，不保留格式，但内容保持不变。您在用户数据中提供的配置将覆盖 Amazon EKS 配置的任何设置。所以，如果设置 `settings.kubernetes.max-pods` 或 `settings.kubernetes.cluster-dns-ip`，用户数据中的这些值将应用到节点。

Amazon EKS 不支持所有有效的 TOM。以下是已知不受支持的格式的列表：

- 引用的键中的引号：`'quoted "value"' = "value"`
- 值中的转义引号：`str = "I'm a string. \"You can quote me\""`
- 混合浮点数和整数：`numbers = [ 0.1, 0.2, 0.5, 1, 2, 5 ]`
- 数组中的混合类型：`contributors = ["foo@example.com", { name = "Baz", email = "baz@example.com" }]`

- 带引用键的括号标题：[foo."bar.baz"]

Windows 用户数据

Windows 用户数据使用 PowerShell 命令。创建托管节点组时，您的自定义用户数据与 Amazon EKS 托管用户数据结合使用。您的 PowerShell 命令排在首位，然后是托管用户数据命令，所有这些命令都在一个 <powershell></powershell> 标签内。

> **Important**
>
> 创建 Windows 节点组时，Amazon EKS 会更新 aws-auth ConfigMap，以便允许基于 Linux 的节点加入集群。该服务不会自动配置 Windows AMI 权限。如果使用的是 Windows 节点，则需要通过访问条目 API 或通过直接更新 aws-auth ConfigMap 来管理访问权限。有关更多信息，请参阅 the section called “启用 Windows 支持”。

> **Note**
>
> 如果在启动模板中未指定 AMI ID，请勿在用户数据中使用 Windows Amazon EKS 引导脚本来配置 Amazon EKS。

用户数据示例如下所示。

```
<powershell>
Write-Host "Running custom user data script"
</powershell>
```

指定 AMI

如果您具有以下任一要求，请在启动模板的 ImageId 字段中指定一个 AMI ID。选择您对其他信息的要求。

提供用户数据，将参数传递给随 Amazon EKS 优化版 Linux/Bottlerocket AMI 一起提供的 **bootstrap.sh** 文件

Bootstrapping（引导启动）是一个术语，用于描述添加可以在实例启动时运行的命令。例如，引导允许使用额外的 kubelet 参数。您可以使用 eksctl 将参数传递给 bootstrap.sh 脚本，无需指定启动模板。或者，您可以在启动模板的用户数据部分中指定信息来实现此目标。

eksctl 无需指定启动模板

创建一个名为 *my-nodegroup.yaml* 的文件，其中包含以下内容。将所有 *example value* 替换为您自己的值。--apiserver-endpoint、--b64-cluster-ca 和 --dns-cluster-ip 参数是可选的。但是，定义它们可使 bootstrap.sh 脚本避免进行 describeCluster 调用。在私有集群设置或频繁扩展节点的集群中，这是非常有用的。有关 bootstrap.sh 脚本的更多信息，请参阅 GitHub 上的 bootstrap.sh 文件。

- 唯一必需的参数是集群名称（*my-cluster*）。
- 要检索 ami-*1234567890abcdef0* 的优化版 AMI ID，请参阅以下部分：
  - 检索建议的 Amazon Linux AMI ID
  - 检索建议的 Bottlerocket AMI ID
  - 检索建议的 Microsoft Windows AMI ID
- 要检索您的集群的 *certificate-authority*，请运行以下命令。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
 --name my-cluster --region region-code
```

- 要检索您的集群的 *api-server-endpoint*，请运行以下命令。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- --dns-cluster-ip 的值是您的服务 CIDR，末尾为 .10。要检索您的集群的 *service-cidr*，请运行以下命令。例如，如果返回值为 ipv4 10.100.0.0/16，则您的值为 *10.100.0.10*。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
 --output text --name my-cluster --region region-code
```

- 此示例使用包含在 Amazon EKS 优化版 AMI 中的 bootstrap.sh 脚本提供一个 kubelet 参数来设置一个自定义 max-pods 值。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。有关选择 *my-max-pods-value* 的帮助，请参阅 the section called “Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量”。

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
```

```
metadata:
  name: my-cluster
  region: region-code

managedNodeGroups:
  - name: my-nodegroup
    ami: ami-1234567890abcdef0
    instanceType: m5.large
    privateNetworking: true
    disableIMDSv1: true
    labels: { x86-al2-specified-mng }
    overrideBootstrapCommand: |
      #!/bin/bash
      /etc/eks/bootstrap.sh my-cluster \
        --b64-cluster-ca certificate-authority \
        --apiserver-endpoint api-server-endpoint \
        --dns-cluster-ip service-cidr.10 \
        --kubelet-extra-args '--max-pods=my-max-pods-value' \
        --use-max-pods false
```

对于每一个可用的 eksctl config 文件选项，请参阅 eksctl 文档中的配置文件架构。eksctl 实用程序仍会为您创建启动模板，并使用您在 config 文件中提供的数据填充其用户数据。

使用以下命令创建节点组。

```
eksctl create nodegroup --config-file=my-nodegroup.yaml
```

启动模板中的用户数据

在启动模板的用户数据部分指定以下信息。将所有 *example value* 替换为您自己的值。--apiserver-endpoint、--b64-cluster-ca 和 --dns-cluster-ip 参数是可选的。但是，定义它们可使 bootstrap.sh 脚本避免进行 describeCluster 调用。在私有集群设置或频繁扩展节点的集群中，这是非常有用的。有关 bootstrap.sh 脚本的更多信息，请参阅 GitHub 上的 bootstrap.sh 文件。

- 唯一必需的参数是集群名称（*my-cluster*）。
- 要检索您的集群的 *certificate-authority*，请运行以下命令。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
 --name my-cluster --region region-code
```

- 要检索您的集群的 *api-server-endpoint*，请运行以下命令。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- `--dns-cluster-ip` 的值是您的服务 CIDR，末尾为 `.10`。要检索您的集群的 *service-cidr*，请运行以下命令。例如，如果返回值为 `ipv4 10.100.0.0/16`，则您的值为 *10.100.0.10*。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
 --output text --name my-cluster --region region-code
```

- 此示例使用包含在 Amazon EKS 优化版 AMI 中的 `bootstrap.sh` 脚本提供一个 `kubelet` 参数来设置一个自定义 `max-pods` 值。有关选择 *my-max-pods-value* 的帮助，请参阅 the section called “Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量”。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -ex
/etc/eks/bootstrap.sh my-cluster \
  --b64-cluster-ca certificate-authority \
  --apiserver-endpoint api-server-endpoint \
  --dns-cluster-ip service-cidr.10 \
  --kubelet-extra-args '--max-pods=my-max-pods-value' \
  --use-max-pods false

--==MYBOUNDARY==--
```

提供用户数据，将参数传递给随 Amazon EKS 优化版 Windows AMI 一起提供的 `Start-EKSBootstrap.ps1` 文件

Bootstrapping（引导启动）是一个术语，用于描述添加可以在实例启动时运行的命令。您可以使用 `eksctl` 将参数传递给 `Start-EKSBootstrap.ps1` 脚本，无需指定启动模板。或者，您可以在启动模板的用户数据部分中指定信息来实现此目标。

如果您想指定自定义 Windows AMI ID，请记住以下注意事项：

- 您必须使用启动模板并在用户数据部分提供所需的引导命令。要检索所需的 Windows ID，您可以使用使用优化版 Windows AMI 创建节点中的表。
- 有一些限制和条件。例如，您必须将 eks:kube-proxy-windows 添加到您的 Amazon IAM 身份验证器配置映射中。有关更多信息，请参阅 the section called “指定 AMI ID 时的限制和条件”。

在启动模板的用户数据部分指定以下信息。将所有 *example value* 替换为您自己的值。-APIServerEndpoint、-Base64ClusterCA 和 -DNSClusterIP 参数是可选的。但是，定义它们可使 Start-EKSBootstrap.ps1 脚本避免进行 describeCluster 调用。

- 唯一必需的参数是集群名称（*my-cluster*）。
- 要检索您的集群的 *certificate-authority*，请运行以下命令。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text --
name my-cluster --region region-code
```

- 要检索您的集群的 *api-server-endpoint*，请运行以下命令。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-cluster
 --region region-code
```

- --dns-cluster-ip 的值是您的服务 CIDR，末尾为 .10。要检索您的集群的 *service-cidr*，请运行以下命令。例如，如果返回值为 ipv4 10.100.0.0/16，则您的值为 *10.100.0.10*。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr" --
output text --name my-cluster --region region-code
```

- 有关更多参数，请参阅 the section called “引导脚本配置参数”。

> **Note**
>
> 如果您使用自定义服务 CIDR，则需要使用 -ServiceCIDR 参数进行指定。否则，集群中的容器组（pod）DNS 解析将失败。

```
<powershell>
[string]$EKSBootstrapScriptFile = "$env:ProgramFiles\Amazon\EKS\Start-EKSBootstrap.ps1"
& $EKSBootstrapScriptFile -EKSClusterName my-cluster `
  -Base64ClusterCA certificate-authority `
  -APIServerEndpoint api-server-endpoint `
```

```
  -DNSClusterIP service-cidr.10
</powershell>
```

由于特定的安全性、合规性或内部策略要求，运行自定义 AMI

有关更多信息，请参阅《Amazon EC2 用户指南》中的亚马逊机器映像（AMI）。Amazon EKS AMI 构建规范包含用于构建基于 Amazon Linux 的自定义 Amazon EKS AMI 的资源和配置脚本。有关更多信息，请参阅 GitHub 上的 Amazon EKS AMI 构建规范。要构建与其他操作系统一起安装的自定义 AMI，请参阅 GitHub 上的 Amazon EKS 示例自定义 AMI。

在与托管式节点组一起使用的启动模板中，AMI ID 不能使用动态参数引用。

> **Important**
>
> 指定 AMI 时，Amazon EKS 不会合并任何用户数据。实际上，您要负责提供节点加入集群所需的 `bootstrap` 命令。如果您的节点无法加入集群，那么 Amazon EKS `CreateNodegroup` 和 `UpdateNodegroupVersion` 操作也会失败。

指定 AMI ID 时的限制和条件

使用托管节点组指定 AMI ID 所涉及的限制和条件如下：

- 您必须创建一个新的节点组，以便在在启动模板中指定 AMI ID 和不指定 AMI ID 之间进行切换。
- 当有较新的 AMI 版本可用时，控制台中不会通知您。要将节点组更新为更新的 AMI 版本，需要使用更新的 AMI ID 创建新版本的启动模板。然后需要使用新的启动模板版本更新节点组。
- 如果您指定 AMI ID，则无法在 API 中设置以下字段：
  - `amiType`
  - `releaseVersion`
  - `version`
- 如果您指定 AMI ID，则会异步应用 API 中的任何 `taints` 设置。要在节点加入集群之前应用污点，必须使用 `--register-with-taints` 命令行标志将污点传递到用户数据中的 `kubelet`。有关更多信息，请参阅 Kubernetes 文档中的 kubelet。
- 为 Windows 托管节点组指定自定义 AMI ID 时，请将 `eks:kube-proxy-windows` 添加到您的 Amazon IAM 身份验证器配置映射中。需要执行此操作 DNS 才能正常运行。

  a. 打开 Amazon IAM 身份验证器配置映射进行编辑。

```
kubectl edit -n kube-system cm aws-auth
```

b. 将此条目添加到每个与 Windows 节点关联的 `rolearn` 下的 `groups` 列表中。您的配置图应该看起来像 aws-auth-cm-windows.yaml。

```
- eks:kube-proxy-windows
```

c. 保存文件并退出文本编辑器。

- 对于使用自定义启动模板的任何 AMI，托管节点组的默认 `HttpPutResponseHopLimit` 均设置为 2。

从集群中删除托管式节点组

本主题介绍如何删除 Amazon EKS 托管节点组。删除托管式节点组时，Amazon EKS 会首先将自动扩缩组的最小、最大和所需大小设置为零。然后，这会导致节点组缩小。

在每个实例终止前，Amazon EKS 会发送一个信号对该节点执行排空操作。在节点排空流程中，Kubernetes 会对节点上的每个容器组（pod）执行以下操作：运行所有已配置的 `preStop` 生命周期挂钩，向容器发送 `SIGTERM` 信号，随后等待 `terminationGracePeriodSeconds` 完成正常关闭。如果 5 分钟后节点仍未完成排空，Amazon EKS 会允许自动扩缩组继续执行实例的强制终止操作。终止所有实例后，将删除自动扩缩组。

> **Important**
>
> 如果您删除的托管节点组使用的节点 IAM 角色未由集群中任何其它托管节点组使用，则该角色将从 `aws-auth ConfigMap` 中移除。如果集群中的任何自行托管节点组使用相同的节点 IAM 角色，则自行管理的节点将变为 `NotReady` 状态。此外，集群操作也被中断。如果您的集群的平台版本至少为使用 EKS 访问条目向 IAM 用户授予对 Kubernetes 的访问权限 的先决条件部分中列出的最低版本，要为仅用于自行管理的节点组的角色添加映射，请参阅the section called “创建访问条目”。如果您的平台版本早于访问条目所需的最低版本，则可以将该条目重新添加到 `aws-auth ConfigMap`。要了解更多信息，请在您的终端中输入 `eksctl create iamidentitymapping --help`。

您可以使用以下工具删除托管节点组：

- the section called “`eksctl`”

- the section called “Amazon Web Services 管理控制台”
- the section called “Amazon CLI”

eksctl

用 **eksctl** 删除托管节点组

输入以下命令。将每个 <example value> 替换为您自己的值。

```
eksctl delete nodegroup \
  --cluster <my-cluster> \
  --name <my-mng> \
  --region <region-code>
```

有关更多选项，请参阅 eksctl 文档中的删除和清空节点组。

Amazon Web Services 管理控制台

用 Amazon Web Services 管理控制台删除托管节点组

1. 打开 Amazon EKS 控制台。
2. 在集群页面上，请选择包含要删除的节点组的集群。
3. 在所选集群页面上，请选择计算选项卡。
4. 在 Node groups（节点组）部分中，选择要删除的节点组。然后选择删除。
5. 在删除节点组确认对话框中，请输入节点组的名称。然后选择删除。

Amazon CLI

用 Amazon CLI 删除托管节点组

1. 输入如下命令。将每个 <example value> 替换为您自己的值。

```
aws eks delete-nodegroup \
  --cluster-name <my-cluster> \
  --nodegroup-name <my-mng> \
  --region <region-code>
```

2. 若 CLI 配置中已设置 cli_pager=，则使用键盘上的方向键滚动查看响应输出内容。完成后按 q 键。

有关更多选项，请参阅 Amazon CLI 命令参考中的 `delete-nodegroup` 命令。

使用自行管理型节点来自行维护节点

一个集群包含一个或多个在其上调度了容器组（pod）的 Amazon EC2 节点。Amazon EKS 节点在 Amazon 账户中运行并通过集群 API 服务器端点连接到集群的控制层面。您需要根据 Amazon EC2 价格收取费用。有关更多信息，请参阅 Amazon EC2 定价。

一个集群可以包含多个节点组。每个节点组都包含在 Amazon EC2 Auto Scaling 组中部署的一个或多个节点。组内节点的实例类型可能会有所不同，例如将基于属性的实例类型选择与 Karpenter 结合使用时。节点组中的所有实例都必须使用 Amazon EKS 节点 IAM 角色。

Amazon EKS 提供一个专用亚马逊机器映像（AMI），称为 Amazon EKS 优化版 AMI。AMI 配置为与 Amazon EKS 配合使用。其组件包括 `containerd`、`kubelet` 和 Amazon IAM 身份验证器。此 AMI 还包含专用引导脚本，从而允许它自动发现并连接到您的集群控制面板。

如果使用 CIDR 块限制对集群的公有端点的访问，建议您还启用私有端点访问。目的是为了让节点可以与集群通信。在未启用私有终端节点的情况下，您指定用于公有访问的 CIDR 块必须包含来自 VPC 的出口源。有关更多信息，请参阅 the section called “集群终端节点访问”。

要向您的 Amazon EKS 集群添加自行管理的节点，请参阅以下主题。如果手动启动自主管理型节点，则向每个节点添加以下标签，同时确保 `<cluster-name>` 与集群匹配。有关更多信息，请参阅为单个资源添加和删除标签。如果按照下面指南中的步骤操作，则会将您所需的标签自动添加到节点。

键	值
`kubernetes.io/cluster/<cluster-name>`	`owned`

Important

Amazon EC2 实例元数据服务（IMDS）中的标签与 EKS 节点不兼容。启用实例元数据标签后，将禁止在标签值中使用正斜杠（“/”）。不执行此限制可能会导致实例启动失败，尤其是在使用 Karpenter 或 Cluster Autoscaler 等节点管理工具时，因为这些服务能否正常运行取决于标签是否包含正斜杠。

有关节点的更多信息（从一般 Kubernetes 角度看），请参阅 Kubernetes 文档中的节点。

主题

- 创建自行管理的 Amazon Linux 节点
- 创建自主管理型 Bottlerocket 节点
- 创建自主管理型 Microsoft Windows 节点
- 创建自主管理型 Ubuntu Linux 节点
- 更新集群的自行管理型节点

创建自行管理的 Amazon Linux 节点

本主题介绍如何启动向 Amazon EKS 集群注册的 Linux 节点的自动扩缩组。在这些节点加入集群后，您可以向其部署 Kubernetes 应用程序。您可以使用 `eksctl` 或 Amazon Web Services 管理控制台 启动自行管理的 Amazon Linux 节点。如果您需要在 Amazon Outposts 上启动节点，请参阅 the section called “Nodes”。

- 现有 Amazon EKS 集群。要部署一个角色，请参阅the section called “创建集群”。如果您在启用了 Amazon Outposts、Amazon Wavelength 或 Amazon 本地区域的 Amazon 区域中拥有子网，则这些子网不得在您创建集群时就已传入。
- 供节点使用的现有 IAM 角色。要创建该文件，请参阅 the section called “节点 IAM 角色”。如果此角色没有 VPC CNI 的任一策略，则需要为 VPC CNI Pod 使用随后的单独角色。
- （可选，但推荐）适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件已配置自己的 IAM 角色，并附加了必要的 IAM 策略。有关更多信息，请参阅 the section called “配置 IRSA”。
- 熟悉选择最优的 Amazon EC2 节点实例类型中列出的注意事项。根据您选择的实例类型，您的集群和 VPC 可能还有其他先决条件。

您可以使用以下任一选项来启动自行管理的 Linux 节点：

- the section called “`eksctl`”
- the section called “Amazon Web Services 管理控制台”

`eksctl`

使用 **`eksctl`** 启动自主管理型 Linux 节点

1. 在您的设备或 Amazon CloudShell 上安装 `0.215.0` 版或更高版本的 `eksctl` 命令行工具。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的 Installation。
2. （可选）如果 AmazonEKS_CNI_Policy 托管的 IAM 策略附加到 Amazon EKS 节点 IAM 角色，我们建议将其分配给您与 Kubernetes `aws-node` 服务账户关联的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。
3. 以下命令在现有集群中创建节点组。将 *`al-nodes`* 替换为您的节点组名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。将 *`my-cluster`* 替换为您的集群的名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。将其余的 *`example value`* 替换为您自己的值。预设情况下，将使用与控制层面相同的 Kubernetes 版本创建节点。

   在为 `--node-type` 选择值之前，请查看选择最优的 Amazon EC2 节点实例类型。

   将 *`my-key`* 替换为您的 Amazon EC2 密钥对或公有密钥的名称。此密钥用于在节点启动后通过 SSH 进入节点。如果还没有 Amazon EC2 密钥对，可以在 Amazon Web Services 管理控制台 中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。

   使用以下命令创建您的节点组。

   > **Important**
   >
   > 如果要将节点组部署到 Amazon Outposts、Wavelength 或本地区域子网，还有其它注意事项：
   >
   > - 创建集群时，不得传入子网。
   > - 您必须使用配置文件创建节点组，指定子网和 `volumeType: gp2`。有关更多信息，请参阅 `eksctl` 文档中的从配置文件创建节点组和 Config 文件架构。

```
eksctl create nodegroup \
  --cluster my-cluster \
  --name al-nodes \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --ssh-access \
  --managed=false \
```

```
  --ssh-public-key my-key
```

部署符合以下条件的节点组：

- 可以将明显高出默认配置数量的 IP 地址分配给容器组（pod），请参阅the section called “增加 IP 地址”。
- 可以将 IPv4 地址分配给来自与实例不同的 CIDR 块的容器组（pod），请参阅the section called “自定义联网”。
- 可以将 IPv6 地址分配给容器组（pod）和服务，请参阅the section called “IPv6”。
- 没有出站互联网访问权限，请参阅the section called “私有集群”。

要查看所有可用选项和默认设置的完整列表，请输入以下命令。

```
eksctl create nodegroup --help
```

如果节点无法加入集群，则请参阅故障排除一章中的 the section called “节点未能加入集群”。

示例输出如下。创建节点时会输出几行。输出的最后几行类似于以下示例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

4. （可选）部署示例应用程序以测试集群和 Linux 节点。
5. 如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：
   - 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
   - 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

   有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

Amazon Web Services 管理控制台

第 1 步：使用 Amazon Web Services 管理控制台 启动自主管理型 Linux 节点

1. 下载最新版本的 Amazon CloudFormation 模板。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2025-11-26/
amazon-eks-nodegroup.yaml
```

2. 等待集群状态显示为 ACTIVE。如果在集群处于活动状态之前启动节点，则节点将无法向集群注册，您必须重新启动节点。
3. 打开 Amazon CloudFormation 控制台。
4. 选择 Create stack（创建堆栈），然后选择 With new resources (standard)（使用新资源（标准））。
5. 对于 Specify template（指定模板），选择 Upload a template file（上传模板文件），然后选择Choose file（选择文件）。
6. 选择您下载的 amazon-eks-nodegroup.yaml 文件。
7. 选择下一步。
8. 在 Specify stack details（指定堆栈详细信息）页面上，相应填写以下参数，然后选择 Next（下一步）：
   - 堆栈名称：为 Amazon CloudFormation 堆栈选择堆栈名称。例如，您可以将其称为 *my-cluster-nodes*。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   - ClusterName：输入您在创建 Amazon EKS 集群时使用的名称。此名称必须与集群名称完全匹配，否则您的节点无法加入该集群。
   - ClusterControlPlaneSecurityGroup：从您在创建 VPC 时生成的 Amazon CloudFormation 输出中，选择 SecurityGroups 值。

     以下步骤显示了检索适用组的一种操作。

     a. 打开 Amazon EKS 控制台。

     b. 选择集群的名称。

     c. 选择 Networking（联网）选项卡。

     d. 从 ClusterControlPlaneSecurityGroup 下拉列表中选择时使用 Additional security groups（其他安全组）值作为参考。
   - ApiServerEndpoint：输入 EKS 集群的 API 服务器端点。可以在 EKS 集群控制台的“详细信息”部分中找到此信息
   - CertificateAuthorityData：输入 base64 编码的证书颁发机构数据，这些数据也可以在 EKS 集群控制台的“详细信息”部分中找到。
   - ServiceCidr：输入用于为集群内的 Kubernetes 服务分配 IP 地址的 CIDR 范围。可以在 EKS 集群控制台的“网络”选项卡中找到此信息。

- AuthenticationMode：查看 EKS 集群控制台中的访问选项卡，选择 EKS 集群中正在使用的身份验证模式。
- NodeGroupName：输入节点组的名称。稍后您可以使用此名称来标识为您的节点创建的自动扩缩节点组。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。
- NodeAutoScalingGroupMinSize：输入您的节点自动扩缩组可横向缩减到的最小节点数。
- NodeAutoScalingGroupDesiredCapacity：输入创建堆栈时要扩展到的所需节点数目。
- NodeAutoScalingGroupMaxSize：输入您的节点自动扩缩组可横向扩展到的最大节点数。
- NodeInstanceType：选择节点的实例类型。有关更多信息，请参阅 the section called “Amazon EC2 实例类型”。
- NodeImageIdSSMParam：使用用于某个变量 Kubernetes 版本最近的 Amazon EKS 优化版 Amazon Linux 2023 AMI 的 Amazon EC2 Systems Manager 参数进行预填充。要使用 Amazon EKS 支持的其他 Kubernetes 次要版本，可以将 *1.XX* 替换为不同的支持版本。我们建议您指定与您的集群相同的 Kubernetes 版本。

  您也可以将 *amazon-linux-2023* 替换为其它 AMI 类型。有关更多信息，请参阅 the section called “获取最新 ID”。

  > **Note**
  >
  > Amazon EKS 节点 AMI 基于 Amazon Linux。您可以在 Amazon Linux 安全中心跟踪 Amazon Linux 2023 的安全和隐私事件，或订阅关联的 RSS 源。安全和隐私事件包括问题的概述、受影响的程序包以及如何更新实例以解决问题。

- NodeImageId：（可选）如果使用自定义 AMI（而不是 Amazon EKS 优化型 AMI），则输入您 Amazon 区域的节点 AMI ID。如果您在此处指定值，它会覆盖 NodeImageIdSSMParam 字段中的任意值。
- NodeVolumeSize：指定您的节点的根卷大小（以 GiB 为单位）。
- NodeVolumeType：指定您的节点的根卷类型。
- KeyName：输入 Amazon EC2 SSH 密钥对的名称，您可使用该密钥对来在节点启动后使用 SSH 连接到这些节点。如果还没有 Amazon EC2 密钥对，可以在 Amazon Web Services 管理控制台中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。
- VpcId：输入您创建的 VPC 的 ID。
- Subnets（子网）：选择您为 VPC 创建的子网。如果您使用为您的 Amazon EKS 集群创建 Amazon VPC 中描述的步骤创建了 VPC，则在 VPC 中仅指定私有子网以供您的节点启动到其

中。您可以通过从集群的 Networking（联网）选项卡中打开每个子网链接来查看哪些子网是私有的。

> **Important**
>
> - 如果其中的任何子网是公有子网，则其必须启用自动公有 IP 地址分配设置。如果没有为该公有子网启用该设置，则您部署到该公有子网的任何节点都不会分配到公有 IP 地址，也无法与集群或其它 Amazon 服务进行通信。如果子网是在 2020 年 3 月 26 日之前使用任一 Amazon EKS Amazon CloudFormation VPC 模板部署的，或者是使用 `eksctl` 部署的，则会为这些公有子网禁用自动公有 IP 地址分配。有关如何为子网启用公有 IP 地址分配的信息，请参阅修改子网的公有 IPv4 寻址属性。如果节点部署到私有子网，则可以通过 NAT 网关与集群和其它 Amazon 服务进行通信。
> - 如果子网没有 Internet 访问权限，请确保您了解部署具有有限互联网访问权限的私有集群中的注意事项和额外步骤。
> - 如果您选择 Amazon Outposts、Wavelength 或本地区域子网，则这些子网不得在您创建集群时就已传入。

9. 在 Configure stack options（配置堆栈选项）页面上选择所需选项，然后选择 Next（下一步）。
10. 选择我了解 Amazon CloudFormation 可能会创建 IAM 资源。左侧的复选框，然后选择创建堆栈。
11. 完成创建堆栈后，在控制台中选中它，然后选择 Outputs（输出）。如果您使用的是 `EKS API` 或 `EKS API and ConfigMap` 身份验证模式，则这是最后一步。
12. 如果您使用的是 `ConfigMap` 身份验证模式，请记录已创建的节点组的 NodeInstanceRole。

第 2 步：使节点能够加入集群

> **Note**
>
> 只有在 EKS 集群内使用 Configmap 身份验证模式时，才需要执行以下两个步骤。此外，如果您在没有出站 Internet 访问的情况下在私有 VPC 内启动了节点，请确保使节点能够从 VPC 中加入您的集群。

1. 检查您是否已经应用拥有 `aws-auth ConfigMap`。

```
kubectl describe configmap -n kube-system aws-auth
```

2. 如果您看到的是 `aws-auth ConfigMap`，则请根据需要对其进行更新。

a. 打开 ConfigMap 文件进行编辑。

```
kubectl edit -n kube-system configmap/aws-auth
```

b. 根据需要添加新的 mapRoles 条目。将 rolearn 值设置为您在上一个步骤中记录的 NodeInstanceRole 值。

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
[...]
```

c. 保存文件并退出文本编辑器。

3. 如果您收到错误提示 "Error from server (NotFound): configmaps "aws-auth" not found，则请使用库存 ConfigMap。

a. 下载配置映射。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

b. 在 aws-auth-cm.yaml 文件中，将 rolearn 值设置为您在上一个步骤中记录的 NodeInstanceRole 值。您可以使用文本编辑器或者通过替换 *my-node-instance-role* 和运行以下命令来执行此操作：

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

c. 应用配置。此命令可能需要几分钟才能完成。

```
kubectl apply -f aws-auth-cm.yaml
```

4. 查看节点的状态并等待它们达到 Ready 状态。

```
kubectl get nodes --watch
```

输入 Ctrl+C 以返回到 Shell 提示符。

> **Note**
>
> 如果您收到任何授权或资源类型错误，请参阅故障排除主题中的 the section called “未经授权或访问被拒绝 (kubectl)”。

如果节点无法加入集群，则请参阅故障排除一章中的 the section called “节点未能加入集群”。

5. （仅限 GPU 节点）如果选择 GPU 实例类型和 Amazon EKS 优化版加速型 AMI，则必须在集群上应用适用于 Kubernetes 的 NVIDIA 设备插件作为 DaemonSet。将 *vX.X.X* 替换为您需要的 NVIDIA/k8s-device-plugin 版本，然后运行以下命令。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

第 3 步：其它操作

1. （可选）部署示例应用程序以测试集群和 Linux 节点。
2. （可选）如果 AmazonEKS_CNI_Policy 托管 IAM 策略（如果您拥有 IPv4 集群）或 *AmazonEKS_CNI_IPv6_Policy*（如果您拥有 IPv6 集群时自行创建）附加到 Amazon EKS 节点 IAM 角色，我们建议将其分配给关联到 Kubernetes aws-node 服务账户的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。
3. 如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：
   - 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
   - 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

   有关更多信息，请参阅限制对分配给工作节点的实例配置文件的访问。

创建自主管理型 Bottlerocket 节点

> **Note**
>
> 托管节点组可能会为您的使用案例带来一些优势。有关更多信息，请参阅 the section called “托管节点组”。

本主题介绍了如何启动向 Amazon EKS 集群注册的 Bottlerocket 节点的自动扩缩组。Bottlerocket 是 Amazon 的基于 Linux 的开源操作系统，用于在虚拟机或裸机主机上运行容器。在这些节点加入集群后，您可以向其部署 Kubernetes 应用程序。有关 Bottlerocket 的更多信息，请参阅 GitHub 上的将 Bottlerocket AMI 与 Amazon EKS 一起使用和 `eksctl` 文档中的自定义 AMI 支持。

有关就地升级的信息，请参阅 GitHub 上的 Bottlerocket Update Operator。

> **Important**
>
> - Amazon EKS 节点是标准的 Amazon EC2 实例，您需要基于常规的 Amazon EC2 实例价格为其付费。有关更多信息，请参阅 Amazon EC2 定价。
> - 您可以在 Amazon Outpost 上的 Amazon EKS 扩展集群中启动 Bottlerocket 节点，但您无法在 Amazon Outpost 上的本地集群中启动它们。有关更多信息，请参阅 Amazon EKS on Amazon Outposts。
> - 您可以部署到采用 x86 或 Arm 处理器的 Amazon EC2 实例。但是，您无法部署到具有 Inferentia 芯片的实例。
> - Bottlerocket 与 Amazon CloudFormation 兼容。但是，没有可以复制用于为 Amazon EKS 部署 Bottlerocket 节点的官方 CloudFormation 模板。
> - Bottlerocket 映像不附带 SSH 服务器或 Shell。您可以使用带外访问方法来允许 SSH 启用管理员容器，并执行一些带有用户数据的引导启动配置步骤。有关更多信息，请参阅 GitHub 上的 bottlerocket README.md：
>   - 探索
>   - 管理员容器
>   - Kubernetes 设置

此过程需要 `eksctl` 版本 `0.215.0` 或更高版本。可以使用以下命令来查看您的版本：

```
eksctl version
```

有关如何安装或升级 eksctl 的说明，请参阅 eksctl 文档中的安装。注意：此过程仅适用于使用 eksctl 创建的集群。

1. 将以下内容复制到您的设备。将 *my-cluster* 替换为您的集群的名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。将 *ng-bottlerocket* 替换为您的节点组名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。要在 Arm 实例上部署，请将 *m5.large* 替换为 Arm 实例类型。将 *my-ec2-keypair-name* 替换为 Amazon EC2 SSH 密钥对的名称，您可使用该密钥对来在节点启动后使用 SSH 连接到这些节点。如果还没有 Amazon EC2 密钥对，可以在 Amazon Web Services 管理控制台 中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。将所有剩余 example values 替换为您自己的值。完成替换后，运行修改后的命令以创建 bottlerocket.yaml 文件。

   如果指定了 Arm Amazon EC2 实例类型，请在部署前查看 Amazon EKS 优化版 Arm Amazon Linux AMI 中的注意事项。有关使用自定义 AMI 进行部署的说明，请参阅 GitHub 上的构建 Bottlerocket 以及 eksctl 文档中的自定义 AMI 支持。要部署托管节点组，请使用启动模板部署自定义 AMI。有关更多信息，请参阅 the section called “启动模板”。

   > **Important**
   >
   > 要将节点组部署到 Amazon Outposts、Amazon Wavelength 或 Amazon Local Zones 子网，就不要在创建集群时传递 Amazon Outposts、Amazon Wavelength 或 Amazon Local Zones 子网。您必须在以下示例中指定子网。有关更多信息，请参阅 eksctl 文档中的从配置文件创建节点组和 Config 文件架构。将 *region-code* 替换为您的集群所在的 Amazon 区域。

```
cat >bottlerocket.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
```

```
  version: '1.33'

iam:
  withOIDC: true

nodeGroups:
  - name: ng-bottlerocket
    instanceType: m5.large
    desiredCapacity: 3
    amiFamily: Bottlerocket
    ami: auto-ssm
    iam:
       attachPolicyARNs:
          - arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodePolicy
          - arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
          - arn:aws-cn:iam::aws:policy/AmazonSSMManagedInstanceCore
          - arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy
    ssh:
        allow: true
        publicKeyName: my-ec2-keypair-name
EOF
```

2. 使用以下命令部署您的节点。

```
eksctl create nodegroup --config-file=bottlerocket.yaml
```

示例输出如下。

创建节点时会输出几行。输出的最后几行类似于以下示例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. （可选）使用 Amazon EBS CSI 插件在 Bottlerocket 节点上创建 Kubernetes 持久性卷。默认 Amazon EBS 驱动程序依赖于一些 Bottlerocket 中不包含的文件系统工具。有关使用驱动程序创建存储类的更多信息，请参阅 the section called “Amazon EBS”。

4. （可选）kube-proxy 默认将 nf_conntrack_max 内核参数设置为默认值，该值可能与 Bottlerocket 最初在启动时设置的值不同。要保留 Bottlerocket 的默认设置，请使用以下命令编辑 kube-proxy 配置。

```
kubectl edit -n kube-system daemonset kube-proxy
```

将 `--conntrack-max-per-core` 和 `--conntrack-min` 添加到以下示例中的 `kube-proxy` 参数。设置为 `0` 意味着没有变化。

```
      containers:
      - command:
        - kube-proxy
        - --v=2
        - --config=/var/lib/kube-proxy-config/config
        - --conntrack-max-per-core=0
        - --conntrack-min=0
```

5. （可选）部署示例应用程序来测试您的 Bottlerocket 节点。
6. 如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：
   - 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
   - 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

   有关更多信息，请参阅限制对分配给工作节点的实例配置文件的访问。

创建自主管理型 Microsoft Windows 节点

本主题介绍了如何启动向 Amazon EKS 集群注册的 Windows 节点的自动扩缩组。在这些节点加入集群后，您可以向其部署 Kubernetes 应用程序。

> **Important**
>
> - Amazon EKS 节点是标准的 Amazon EC2 实例，您需要基于常规的 Amazon EC2 实例价格为其付费。有关更多信息，请参阅 Amazon EC2 定价。
> - 您可以在 Amazon Outpost 上的 Amazon EKS 扩展集群中启动 Windows 节点，但您无法在 Amazon Outpost 上的本地集群中启动它们。有关更多信息，请参阅 Amazon EKS on Amazon Outposts。

为集群启用 Windows 支持。我们建议您在启动 Windows 节点组之前查看重要的注意事项。有关更多信息，请参阅 the section called “启用 Windows 支持”。

您可以通过以下任一方法启动自主管理型 Windows 节点：

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”

eksctl

使用 **eksctl** 启动自主管理型 Windows 节点

此过程要求您已安装 eksctl，且您的 eksctl 版本至少为 0.215.0。可以使用以下命令来查看您的版本。

```
eksctl version
```

有关安装或升级 eksctl 的说明，请参阅 eksctl 文档中的 Installation。

> **Note**
>
> 此过程仅适用于使用 eksctl 创建的集群。

1. （可选）如果 AmazonEKS_CNI_Policy 托管 IAM 策略（如果您拥有 IPv4 集群）或 *AmazonEKS_CNI_IPv6_Policy*（如果您拥有 IPv6 集群时自行创建）附加到 Amazon EKS 节点 IAM 角色，我们建议将其分配给关联到 Kubernetes aws-node 服务账户的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。
2. 此过程假设您已经有一个现有集群。如果还没有可添加 Windows 节点组的 Amazon EKS 集群和 Amazon Linux 节点组，则建议按照the section called “创建集群（eksctl）” 中的说明操作。该指南提供使用 Amazon Linux 节点创建 Amazon EKS 集群的完整演练。

   使用以下命令创建您的节点组。将 *region-code* 替换为您的集群所在的 Amazon 区域。将 *my-cluster* 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。将 *ng-windows* 替换为您的节点组名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。您可以将 *2019* 替换为 2022，以使用 Windows Server 2022。将 example values 的剩余部分替换为您自己的值。

> **Important**
>
> 要将节点组部署到 Amazon Outposts、Amazon Wavelength 或 Amazon 本地区域子网，在创建集群时不要传递 Amazon Outposts、Wavelength 或本地区域子网。使用配置文件创建节点组，指定 Amazon Outposts、Wavelength 或本地区域子网。有关更多信息，请参阅 `eksctl` 文档中的从配置文件创建节点组和 Config 文件架构。

```
eksctl create nodegroup \
    --region region-code \
    --cluster my-cluster \
    --name ng-windows \
    --node-type t2.large \
    --nodes 3 \
    --nodes-min 1 \
    --nodes-max 4 \
    --managed=false \
    --node-ami-family WindowsServer2019FullContainer
```

> **Note**
>
> - 如果节点无法加入集群，请参阅《故障排除指南》中的 the section called “节点未能加入集群”。
> - 要查看 `eksctl` 命令可用选项，请输入以下命令。
>
>   ```
>   eksctl command -help
>   ```

示例输出如下。创建节点时会输出几行。输出的最后几行类似于以下示例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. （可选）部署示例应用程序以测试您的集群和 Windows 节点。
4. 如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：
   - 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。

- 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

Amazon Web Services 管理控制台

先决条件

- 现有 Amazon EKS 集群和 Linux 节点组。如果您没有这些资源，我们建议您按照我开始使用中的指南之一创建它们。这些指南介绍如何使用 Linux 节点创建 Amazon EKS 集群。
- 符合 Amazon EKS 集群要求的现有 VPC 和安全组。有关更多信息，请参阅the section called “VPC 和子网要求”和the section called “安全组要求”。开始使用 中的指南创建符合要求的 VPC。或者，您可以按照为 Amazon EKS 集群创建一个 Amazon VPC 中的说明，手动创建一个。
- 现有 Amazon EKS 集群，它使用符合 Amazon EKS 集群要求的 VPC 和安全组。有关更多信息，请参阅 the section called “创建集群”。如果您在启用了 Amazon Outposts、Amazon Wavelength 或 Amazon 本地区域的 Amazon 区域中拥有子网，则这些子网不得在您创建集群时就已传入。

第 1 步：使用 Amazon Web Services 管理控制台 启动自行管理 Windows 节点

1. 等待集群状态显示为 `ACTIVE`。如果在集群处于活动状态之前启动节点，则节点将无法向集群注册，您需要重新启动节点。
2. 打开 Amazon CloudFormation 控制台
3. 选择创建堆栈。
4. 对于 Specify template（指定模板），选择 Amazon S3 URL。
5. 复制以下 URL 并将其粘贴到 Amazon S3 URL 中。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2023-02-09/amazon-eks-windows-nodegroup.yaml
```

6. 选择 Next（下一步）两次。
7. 在 Quick create stack（快速创建堆栈）页面上，相应地填写以下参数：
   - 堆栈名称：为 Amazon CloudFormation 堆栈选择堆栈名称。例如，您可以将其命名为 `my-cluster-nodes`。
   - ClusterName：输入您在创建 Amazon EKS 集群时使用的名称。

> **Important**
>
> 此名称必须与在第 1 步：创建您的 Amazon EKS 集群中使用的名称完全匹配。否则，您的节点无法加入集群。

- ClusterControlPlaneSecurityGroup：从您在创建 VPC 时生成的 Amazon CloudFormation 输出中，选择安全组。以下步骤显示了检索适用组的一种方法。

  a. 打开 Amazon EKS 控制台。

  b. 选择集群的名称。

  c. 选择 Networking（联网）选项卡。

  d. 从 ClusterControlPlaneSecurityGroup 下拉列表中选择时使用 Additional security groups（其他安全组）值作为参考。

- NodeGroupName：输入节点组的名称。稍后您可以使用此名称来标识为您的节点创建的自动扩缩节点组。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。
- NodeAutoScalingGroupMinSize：输入您的节点自动扩缩组可横向缩减到的最小节点数。
- NodeAutoScalingGroupDesiredCapacity：输入创建堆栈时要扩展到的所需节点数目。
- NodeAutoScalingGroupMaxSize：输入您的节点自动扩缩组可横向扩展到的最大节点数。
- NodeInstanceType：选择节点的实例类型。有关更多信息，请参阅 the section called “Amazon EC2 实例类型”。

> **Note**
>
> GitHub 的 vpc_ip_resource_limit.go 列出适用于 Kubernetes 的 Amazon VPC CNI 插件最新版本支持的实例类型。您可能需要更新 CNI 版本以使用最新的受支持实例类型。有关更多信息，请参阅 the section called “Amazon VPC CNI”。

- NodeImageIdSSMParam：预先填充了当前推荐的 Amazon EKS 优化版 Windows Core AMI ID 的 Amazon EC2 Systems Manager 参数。要使用完整版本的 Windows，请将 `Core` 替换为 `Full`。
- NodeImageId：（可选）如果使用自定义 AMI（而不是 Amazon EKS 优化型 AMI），则输入您 Amazon 区域的节点 AMI ID。如果您为此字段指定值，它会覆盖 NodeImageIdSSMParam 字段中的任意值。
- NodeVolumeSize：指定您的节点的根卷大小（以 GiB 为单位）。

- KeyName：输入 Amazon EC2 SSH 密钥对的名称，您可使用该密钥对来在节点启动后使用 SSH 连接到这些节点。如果还没有 Amazon EC2 密钥对，可以在 Amazon Web Services 管理控制台中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。

  > **Note**
  >
  > 如果此处不提供密钥对，Amazon CloudFormation 堆栈创建将失败。

- BootstrapArguments：指定要传递给节点引导脚本的所有可选参数，如使用 `-KubeletExtraArgs` 的其他 `kubelet` 实际参数。
- DisableIMDSv1：预设情况下，每个节点支持实例元数据服务版本 1 (IMDSv1) 和 IMDSv2。您可以禁用 IMDSv1。要防止节点组中的未来节点和容器组（pod）使用 MDSv1，请将 DisableIMDSv1 设置为 true。有关 IMDS 的更多信息，请参阅配置实例元数据服务。
- VpcId：选择创建的 VPC 的 ID。
- NodeSecurityGroups：选择在您创建 VPC 时为您的 Linux 节点组创建的安全组。如果您的 Linux 节点附加了多个安全组，请指定所有安全组。例如，如果 Linux 节点组是使用创建的 `eksctl`。
- 子网：选择创建的子网。如果您使用为 Amazon EKS 集群创建 Amazon VPC 中的步骤创建了 VPC，则在 VPC 中仅指定私有子网以供您的节点启动到其中。

  > **Important**
  >
  > - 如果其中的任何子网是公有子网，则其必须启用自动公有 IP 地址分配设置。如果没有为该公有子网启用该设置，则您部署到该公有子网的任何节点都不会分配到公有 IP 地址，也无法与集群或其它 Amazon 服务进行通信。如果子网是在 2020 年 3 月 26 日之前使用任一 Amazon EKS Amazon CloudFormation VPC 模板部署的，或者是使用 `eksctl` 部署的，则会为这些公有子网禁用自动公有 IP 地址分配。有关如何为子网启用公有 IP 地址分配的信息，请参阅修改子网的公有 IPv4 寻址属性。如果节点部署到私有子网，则可以通过 NAT 网关与集群和其它 Amazon 服务进行通信。
  > - 如果子网没有 Internet 访问权限，请确保您了解部署具有有限互联网访问权限的私有集群中的注意事项和额外步骤。
  > - 如果您选择 Amazon Outposts、Wavelength 或本地区域子网，则这些子网不得在您创建集群时就已传入。

8. 确认堆栈可创建 IAM 资源，然后选择 Create stack（创建堆栈）。
9. 完成创建堆栈后，在控制台中选中它，然后选择 Outputs（输出）。
10. 记录已创建的节点组的 NodeInstanceRole。您在配置 Amazon EKS Windows 节点时需要此值。

第 2 步：使节点能够加入集群

1. 检查您是否已经应用拥有 aws-auth ConfigMap。

```
kubectl describe configmap -n kube-system aws-auth
```

2. 如果您看到的是 aws-auth ConfigMap，则请根据需要对其进行更新。

   a. 打开 ConfigMap 文件进行编辑。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 根据需要添加新的 mapRoles 条目。将 rolearn 值设置为您在前面的步骤中记录的 NodeInstanceRole 值。

```
[...]
data:
  mapRoles: |
- rolearn: <ARN of linux instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: <ARN of windows instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
        - eks:kube-proxy-windows
[...]
```

   c. 保存文件并退出文本编辑器。

3. 如果您收到错误提示 "Error from server (NotFound): configmaps "aws-auth" not found，则请使用库存 ConfigMap。

   a. 下载配置映射。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm-windows.yaml
```

   b. 在 aws-auth-cm-windows.yaml 文件中，将 rolearn 值设置为您在前面的步骤中记录的 NodeInstanceRole 值。您可以使用文本编辑器或者通过替换 example values 和运行以下命令来执行此操作：

```
sed -i.bak -e 's|<ARN of linux instance role (not instance profile)>|my-node-linux-instance-role|' \
    -e 's|<ARN of windows instance role (not instance profile)>|my-node-windows-instance-role|' aws-auth-cm-windows.yaml
```

> **Important**
>
> - 请勿修改此文件中的任何其它行。
> - 不要对 Windows 和 Linux 节点使用相同的 IAM 角色。

c. 应用配置。此命令可能需要几分钟才能完成。

```
kubectl apply -f aws-auth-cm-windows.yaml
```

4. 查看节点的状态并等待它们达到 Ready 状态。

```
kubectl get nodes --watch
```

输入 Ctrl+C 以返回到 Shell 提示符。

> **Note**
>
> 如果您收到任何授权或资源类型错误，请参阅故障排除主题中的 the section called “未经授权或访问被拒绝 (kubectl)”。

如果节点无法加入集群，则请参阅故障排除一章中的 the section called “节点未能加入集群”。

第 3 步：其它操作

1. （可选）部署示例应用程序以测试您的集群和 Windows 节点。
2. （可选）如果 AmazonEKS_CNI_Policy 托管 IAM 策略（如果您拥有 IPv4 集群）或 *AmazonEKS_CNI_IPv6_Policy*（如果您拥有 IPv6 集群时自行创建）附加到 Amazon EKS 节点 IAM 角色，我们建议将其分配给关联到 Kubernetes aws-node 服务账户的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。
3. 如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：

- 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
- 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

有关更多信息，请参阅限制对分配给工作节点的实例配置文件的访问。

创建自主管理型 Ubuntu Linux 节点

> **Note**
>
> 托管节点组可能会为您的使用案例带来一些优势。有关更多信息，请参阅 the section called "托管节点组"。

本主题介绍了如何启动向 Amazon EKS 集群注册的适用于 Amazon Elastic Kubernetes Service（EKS）的 Ubuntu 节点自动扩缩组，或适用于 Amazon Elastic Kubernetes Service（EKS）的 Ubuntu Pro 节点的自动扩缩组。适用于 EKS 的 Ubuntu 和 Ubuntu Pro 基于官方的 Ubuntu Minimal LTS，包括与 Amazon 共同开发并专门为 EKS 构建的自定义 Amazon 内核。Ubuntu Pro 通过支持 EKS 延期支持期、内核 livepatch、FIPS 合规性以及运行无限制 Pro 容器的功能，增加了额外的安全覆盖。

在这些节点加入集群后，您可以向其部署容器化应用程序。有关更多信息，请访问有关 Amazon 上的 Ubuntu 的文档和 `eksctl` 文档中的自定义 AMI 支持。

> **Important**
>
> - Amazon EKS 节点是标准的 Amazon EC2 实例，您需要基于常规的 Amazon EC2 实例价格为其付费。有关更多信息，请参阅 Amazon EC2 定价。
> - 您可以在 Amazon Outposts 上的 Amazon EKS 扩展集群中启动 Ubuntu 节点，但您无法在 Amazon Outposts 上的本地集群中启动这些节点。有关更多信息，请参阅 Amazon EKS on Amazon Outposts。
> - 您可以部署到采用 x86 或 Arm 处理器的 Amazon EC2 实例。但是，具有 Inferentia 芯片的实例可能需要先安装 Neuron SDK。

此过程需要 `eksctl` 版本 `0.215.0` 或更高版本。可以使用以下命令来查看您的版本：

```
eksctl version
```

有关如何安装或升级 eksctl 的说明，请参阅 eksctl 文档中的安装。注意：此过程仅适用于使用 eksctl 创建的集群。

1. 将以下内容复制到您的设备。将 my-cluster 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母字符开头，且不得超过 100 个字符。将 ng-ubuntu 替换为您的节点组名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。要在 Arm 实例上部署，请将 m5.large 替换为 Arm 实例类型。将 my-ec2-keypair-name 替换为 Amazon EC2 SSH 密钥对的名称，您可使用该密钥对来在节点启动后使用 SSH 连接到这些节点。如果还没有 Amazon EC2 密钥对，可以在 Amazon Web Services 管理控制台 中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。将所有剩余 example values 替换为您自己的值。完成替换后，运行修改后的命令以创建 ubuntu.yaml 文件。

   > **Important**
   >
   > 要将节点组部署到 Amazon Outposts、Amazon Wavelength 或 Amazon Local Zones 子网，就不要在创建集群时传递 Amazon Outposts、Amazon Wavelength 或 Amazon Local Zones 子网。您必须在以下示例中指定子网。有关更多信息，请参阅 eksctl 文档中的从配置文件创建节点组和 Config 文件架构。将 *region-code* 替换为您的集群所在的 Amazon 区域。

```
cat >ubuntu.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.33'

iam:
  withOIDC: true

nodeGroups:
  - name: ng-ubuntu
```

```
    instanceType: m5.large
    desiredCapacity: 3
    amiFamily: Ubuntu2204
    iam:
       attachPolicyARNs:
          - arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodePolicy
          - arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
          - arn:aws-cn:iam::aws:policy/AmazonSSMManagedInstanceCore
          - arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy
    ssh:
        allow: true
        publicKeyName: my-ec2-keypair-name
EOF
```

要创建 Ubuntu Pro 节点组，只需将 `amiFamily` 值更改为 `UbuntuPro2204`。

2. 使用以下命令部署您的节点。

```
eksctl create nodegroup --config-file=ubuntu.yaml
```

示例输出如下。

创建节点时会输出几行。输出的最后几行类似于以下示例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. （可选）部署示例应用程序来测试 Ubuntu 节点。
4. 如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：
   - 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
   - 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

   有关更多信息，请参阅限制对分配给工作节点的实例配置文件的访问。

更新集群的自行管理型节点

当发布新的 Amazon EKS 优化版 AMI 时，考虑将您自行管理节点组中的节点替换为这一新的 AMI。同样，如果为 Amazon EKS 集群更新 Kubernetes 版本，则还应更新节点将其用于同一 Kubernetes 版本。

> **Important**
>
> 本主题介绍适用于自行管理的节点的节点更新。如果使用的是托管式节点组，请参阅the section called “更新”。

有两种基本方法可以更新集群中自行管理的节点组以使用新 AMI：

将应用程序迁移到新的节点组

创建一个新的节点组并将您的容器组（pod）迁移到该组。迁移到新节点组比简单地在现有 Amazon CloudFormation 堆栈中更新 AMI ID 更好。这是因为迁移过程会将旧节点组污染为 `NoSchedule`，并在新堆栈准备好接受现有容器组（pod）工作负载之后耗尽节点。

更新 Amazon CloudFormation 节点堆栈

更新现有节点组的 Amazon CloudFormation 堆栈以使用新 AMI。使用 `eksctl` 创建的节点组不支持此方法。

将应用程序迁移到新的节点组

本主题介绍如何创建新的节点组，将您的现有应用程序自然地迁移到新组，然后从您的集群中删除旧的节点组。您可以使用 `eksctl` 或 Amazon Web Services 管理控制台 迁移到新的节点组。

- the section called “`eksctl`”
- the section called “Amazon Web Services 管理控制台 和 Amazon CLI”

`eksctl`

使用 **`eksctl`** 将您的应用程序迁移到新的节点组

有关使用 eksctl 进行迁移的更多信息，请参阅 `eksctl` 文档中的非托管节点组。

此过程需要 `eksctl` 版本 `0.215.0` 或更高版本。可以使用以下命令来查看您的版本：

```
eksctl version
```

有关安装或升级 `eksctl` 的说明，请参阅 `eksctl` 文档中的 Installation。

> **Note**
>
> 此过程仅适用于使用 `eksctl` 创建的集群和节点组。

1. 检索您现有节点组的名称，同时将 *my-cluster* 替换为您的集群名称。

```
eksctl get nodegroups --cluster=my-cluster
```

示例输出如下。

```
CLUSTER      NODEGROUP          CREATED               MIN SIZE      MAX SIZE
 DESIRED CAPACITY     INSTANCE TYPE     IMAGE ID
default      standard-nodes   2019-05-01T22:26:58Z  1             4             3
               t3.medium         ami-05a71d034119ffc12
```

2. 用 `eksctl` 和下面的命令启动新节点组。将命令中的所有 *example value* 替换为您自己的值。版本号不能高于控制面板的 Kubernetes 版本，也不能比控制面板的 Kubernetes 版本低两个以上的次要版本。我们建议您使用与控制面板相同的版本。

   如果满足以下条件，我们建议阻止容器组（pod）访问 IMDS：

   - 您计划将 IAM 角色分配给所有 Kubernetes 服务账户，以便容器组（pod）仅具有其所需的最低权限。
   - 集群中没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 Amazon EC2 实例元数据服务（IMDS）。

     有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

     要阻止容器组（pod）对 IMDS 的访问，请将 `--disable-pod-imds` 选项添加到以下命令。

     > **Note**
     >
     > 有关更多可用标志及其说明，请参阅 https://eksctl.io/。

```
eksctl create nodegroup \
  --cluster my-cluster \
  --version 1.33 \
  --name standard-nodes-new \
  --node-type t3.medium \
```

```
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --managed=false
```

3. 当上一个命令完成时，使用以下命令验证您的所有节点是否已达到 Ready 状态：

```
kubectl get nodes
```

4. 使用以下命令删除原始节点组。在命令中，将所有 *example value* 替换为集群和节点组名称：

```
eksctl delete nodegroup --cluster my-cluster --name standard-nodes-old
```

Amazon Web Services 管理控制台 和 Amazon CLI

使用 Amazon Web Services 管理控制台和 Amazon CLI 将您的应用程序迁移到新的节点组。

1. 执行创建自行管理的 Amazon Linux 节点中概述的步骤，启动新的节点组。
2. 完成创建堆栈后，在控制台中选中它，然后选择 Outputs（输出）。
3. 记录已创建的节点组的 NodeInstanceRole。您需要它来将新的 Amazon EKS 节点添加到集群。

> **Note**
>
> 如果您已将任何其他 IAM 策略附加到旧节点组 IAM 角色，则应将这些相同的策略附加到新节点组 IAM 角色以在新组上保持该功能。这适用于为 Kubernetes Cluster Autoscaler 添加权限的情况。

4. 同时更新两个节点组的安全组，以便它们可以相互通信。有关更多信息，请参阅 the section called “安全组要求”。
   a. 记下两个节点组的安全组 ID。这在 Amazon CloudFormation 堆栈输出中显示为 NodeSecurityGroup 值。

      您可以使用以下 Amazon CLI 命令从堆栈名称中获取安全组 ID。在这些命令中，oldNodes 是您的较旧节点堆栈的 Amazon CloudFormation 堆栈名称，newNodes 是要迁移到的堆栈的名称。将所有 *example value* 替换为您自己的值。

      ```
      oldNodes="old_node_CFN_stack_name"
      newNodes="new_node_CFN_stack_name"
      ```

```
oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType==`Amazon::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
--query 'StackResources[?
ResourceType==`Amazon::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
```

b. 向每个节点安全组添加入口规则，以便它们接受彼此的流量。

以下 Amazon CLI 命令向每个安全组添加入站规则，以允许来自另一个安全组的所有协议上的所有流量。通过此配置，在您将工作负载迁移到新组时，每个节点组中的容器组（pod）都可以相互通信。

```
aws ec2 authorize-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 authorize-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

5. 编辑 aws-auth configmap 以在 RBAC 中映射新的节点实例角色。

```
kubectl edit configmap -n kube-system aws-auth
```

为新的节点组添加新的 mapRoles 条目。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: ARN of instance role (not instance profile)
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
    - rolearn: arn:aws-cn:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

将 *ARN of instance role (not instance profile)* 代码段替换为在上一步中记录的 NodeInstanceRole 值。保存并关闭该文件以应用更新后的 configmap。

6. 查看节点的状态并等待新节点加入您的集群并达到 Ready 状态。

```
kubectl get nodes --watch
```

7. （可选）如果使用 Kubernetes Cluster Autoscaler，请将部署缩减到 0 个副本以避免相互冲突的扩缩操作。

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

8. 使用以下命令，对要使用 NoSchedule 删除的每个节点执行 Taint 操作。如此一来，就不会在要替换的节点上调用或重新调用新的容器组（pod）。有关更多信息，请参阅 Kubernetes 文档中的 Taints and Tolerations。

```
kubectl taint nodes node_name key=value:NoSchedule
```

如果要将节点升级到新的 Kubernetes 版本，则可使用以下代码段来标识并污染特定 Kubernetes 版本（此示例中为 1.31 版本）的所有节点。版本号不能高于控制面板的 Kubernetes 版本，也不能比控制面板的 Kubernetes 版本低两个以上的次要版本。我们建议您使用与控制面板相同的版本。

```
K8S_VERSION=1.31
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Tainting $node"
    kubectl taint nodes $node key=value:NoSchedule
done
```

9. 确定集群的 DNS 提供商。

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

示例输出如下。此集群使用 CoreDNS 解析 DNS，但您的集群可能会返回 kube-dns）：

```
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
coredns   1         1         1            1           31m
```

10.如果您的当前部署所运行的副本少于 2 个，请将部署扩展到 2 个副本。如果您的上一个命令输出返回了该项，请将 *coredns* 替换为kubedns。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

11.使用以下命令耗尽要从集群中删除的每个节点：

```
kubectl drain node_name --ignore-daemonsets --delete-local-data
```

如果您要将节点升级到新的 Kubernetes 版本，请使用以下代码段标识并耗尽特定 Kubernetes 版本（此示例中为 *1.31* 版本）的所有节点。

```
K8S_VERSION=1.31
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-local-data
done
```

12.在旧节点耗尽后，请撤销您之前授权的安全组入口规则。然后删除 Amazon CloudFormation 堆栈以终止实例。

> **Note**
>
> 如果您已将任何其它 IAM 策略附加到旧节点组 IAM 角色（例如，为 Kubernetes Cluster Autoscaler 添加权限），请先将这些附加策略与该角色分离，然后才能删除 Amazon CloudFormation 堆栈。

a. 撤销您之前为节点安全组创建的入站规则。在这些命令中，oldNodes 是您的较旧节点堆栈的 Amazon CloudFormation 堆栈名称，newNodes 是要迁移到的堆栈的名称。

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType==`Amazon::EC2::SecurityGroup`].PhysicalResourceId' \
```

```
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
--query 'StackResources[?
ResourceType==`Amazon::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
aws ec2 revoke-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 revoke-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

b. 打开 Amazon CloudFormation 控制台。

c. 选择您的旧节点堆栈。

d. 选择删除。

e. 在 Delete stack（删除堆栈）确认对话框中，请选择 Delete stack（删除堆栈）。

13编辑 aws-auth configmap 以从 RBAC 中删除旧节点实例角色。

```
kubectl edit configmap -n kube-system aws-auth
```

删除旧节点组的 mapRoles 条目。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws-cn:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
W70725MZQFF8
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws-cn:iam::111122223333:role/nodes-1-15-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
```

保存并关闭该文件以应用更新后的 configmap。

14.（可选）如果您使用的是 Kubernetes Cluster Autoscaler，请将部署缩减为 1 个副本。

> **Note**
>
> 您还必须适当地标记新的自动扩缩组（例如，`k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/my-cluster`）并将您的 Cluster Autoscaler 部署命令更新为指向新标记的自动扩缩组。有关更多信息，请参阅 Amazon 上的 Cluster Autoscaler。

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

15. （可选）确认您使用的是最新版本的 Kubernetes 的 Amazon VPC CNI 插件。您可能需要更新 CNI 版本以使用最新的受支持实例类型。有关更多信息，请参阅 the section called “Amazon VPC CNI”。
16. 如果您的集群使用适用于 DNS 解析的 `kube-dns`（请参阅[migrate-determine-dns-step]），请将 `kube-dns` 部署横向缩减为 1 个副本。

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

更新 Amazon CloudFormation 节点堆栈

本主题介绍如何使用新 AMI 更新现有 Amazon CloudFormation 自行管理的节点堆栈。在集群更新后，可以使用此过程将节点更新为新版本的 Kubernetes。或者，您可以更新到针对现有 Kubernetes 版本的最新 Amazon EKS 优化 AMI。

> **Important**
>
> 本主题介绍适用于自行管理的节点的节点更新。有关使用托管式节点组简化节点生命周期的信息，请参阅the section called “更新”。

最新的默认 Amazon EKS 节点 Amazon CloudFormation 模板将配置为使用新 AMI 在集群中启动实例，然后再删除旧 AMI，一次删除一个。此配置可确保您在滚动更新期间始终具有集群中自动扩缩组所需的活动实例计数。

> **Note**
>
> 使用 eksctl 创建的节点组不支持此方法。如果您使用 eksctl 创建了集群或节点组，请参阅 the section called “迁移”。

1. 确定集群的 DNS 提供商。

   ```
   kubectl get deployments -l k8s-app=kube-dns -n kube-system
   ```

   示例输出如下。此集群使用 CoreDNS 解析 DNS，但您的集群可能会返回 kube-dns。您的输出可能看起来有所不同，具体取决于您使用的 kubectl 版本。

   ```
   NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   coredns   1         1         1            1           31m
   ```

2. 如果您的当前部署所运行的副本少于 2 个，请将部署扩展到 2 个副本。如果您的上一个命令输出返回了该项，请将 *coredns* 替换为kube-dns。

   ```
   kubectl scale deployments/coredns --replicas=2 -n kube-system
   ```

3. （可选）如果使用 Kubernetes Cluster Autoscaler，请将部署缩减到 0 个副本以避免相互冲突的扩缩操作。

   ```
   kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
   ```

4. 确定当前节点组的实例类型和所需的实例计数。以后更新该组的 Amazon CloudFormation 模板时将输入这些值。

   a. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。

   b. 请在左侧导航窗格中，选择 Launch Configurations（启动配置），然后记下现有节点启动配置的实例类型。

   c. 请在左侧导航窗格中，选择自动扩缩组，并记下现有节点自动扩缩组的所需实例计数。

5. 打开 Amazon CloudFormation 控制台。

6. 选择您的节点组堆栈，然后选择 Update（更新）。

7. 选择替换当前模板，然后选择 Amazon S3 URL。

8. 对于 Amazon S3 URL，请将以下 URL 粘贴到文本区域中以确保您使用的是最新版本的节点 Amazon CloudFormation 模板。接下来，选择 Next（下一步）：

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/amazon-eks-nodegroup.yaml
```

9. 在 Specify stack details (指定堆栈详细信息) 页面上，填写以下参数，然后选择 Next (下一步)：

   - NodeAutoScalingGroupDesiredCapacity - 输入在上一步记录的所需实例计数。或者输入更新堆栈时要扩展到的所需节点数目。
   - NodeAutoScalingGroupMaxSize：输入您的节点自动扩缩组可横向扩展到的最大节点数。此值必须比所需容量多至少一个节点。这样更新期间可以对节点进行滚动更新而不会减少节点数。
   - NodeInstanceType - 选择在上一步记录的实例类型。也可以为节点选择不同的实例类型。在选择其它实例类型之前，请查看选择最优的 Amazon EC2 节点实例类型。每种 Amazon EC2 实例类型支持最大数量的弹性网络接口（网络接口），每个网络接口都支持最大数量的 IP 地址。由于为每个 Worker 节点和容器组（pod）分配了自己的 IP 地址，请务必选择一个实例类型，该实例类型可以支持您希望在每个 Amazon EC2 节点上运行的最大数量的容器组（pod）。有关实例类型支持的网络接口和 IP 地址数量的列表，请参阅每种实例类型的每个网络接口的 IP 地址数。例如，`m5.large` 实例类型最多支持 Worker 节点和容器组（pod）的 30 个 IP 地址。

     > **Note**
     >
     > 最新版本适用于 Kubernetes 的 Amazon VPC CNI 插件支持的实例类型在 GitHub 上的 vpc_ip_resource_limit.go 中显示。您可能需要更新适用于 Kubernetes 的 Amazon VPC CNI 插件版本，才能使用最新的受支持实例类型。有关更多信息，请参阅 the section called “Amazon VPC CNI”。

     > **Important**
     >
     > 一些实例类型并非在所有 Amazon 区域中都可用。

   - NodeImageIdSSMParam – 要更新到的 AMI ID 的 Amazon EC2 Systems Manager 参数。以下值为 Kubernetes 版本 `1.33` 使用最新 Amazon EKS 优化版 AMI。

     ```
     /aws/service/eks/optimized-ami/1.33/amazon-linux-2/recommended/image_id
     ```

     您可以将 *`1.33`* 替换为相同的 platform-version。或者，它应该比控制面板运行的 Kubernetes 版本高最多一个版本。我们建议您将节点保持在与控制面板相同的版本。您也可以将 *`amazon-linux-2`* 替换为其它 AMI 类型。有关更多信息，请参阅 the section called “获取最新 ID”。

> **Note**
>
> 使用 Amazon EC2 Systems Manager 参数让您可以在以后更新节点而无需查找和指定 AMI ID。如果您的 Amazon CloudFormation 堆栈使用此值，则任何堆栈更新将为您指定的 Kubernetes 版本始终启动最新建议的 Amazon EKS 优化版 AMI。即使不更改模板中的任何值，情况也是如此。

- NodeImageId – 要使用您的自定义 AMI，请输入要使用的 AMI 的 ID。

> **Important**
>
> 此值覆盖为 NodeImageIdSSMParam 指定的任意值。如果您要使用 NodeImageIdSSMParam 值，请确保 NodeImageId 的值为空白。

- DisableIMDSv1 - 每个节点默认支持实例元数据服务版本 1 (IMDSv1) 和 IMDSv2。但可以禁用 IMDSv1。如果不希望节点组中调度的任何节点或任何容器组（pod）使用 IMDSv1，请选择 true。有关 IMDS 的更多信息，请参阅配置实例元数据服务。如果已实施服务账户的 IAM 角色，请将所需权限直接分配给需要访问 Amazon 服务的所有容器组（pod）。这样，集群中就没有任何容器组（pod）需要出于其他原因（例如检索当前 Amazon 区域）访问 IMDS。然后，对于不使用主机网络的容器组（pod），您还可以禁用 IMDSv2 的访问权限。有关更多信息，请参阅限制对分配给工作节点的实例配置文件的访问。

10. （可选）在 Options (选项) 页面上，为您的堆栈资源添加标签。选择下一步。
11. 在 Review（审核）页面上审核您的信息，确认堆栈可创建 IAM 资源，然后选择 Update stack（更新堆栈）。

> **Note**
>
> 集群中每个节点的更新需要几分钟时间。等待所有节点更新完成，然后再执行后续步骤。

12. 如果您的集群的 DNS 提供商是 `kube-dns`，请将 `kube-dns` 部署横向缩减为 1 个副本。

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

13. （可选）如果您使用的是 Kubernetes Cluster Autoscaler，请将部署缩减为所需数量的副本。

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

14. （可选）确认您使用的是最新版本的 Kubernetes 的 Amazon VPC CNI 插件。您可能需要更新适用于 Kubernetes 的 Amazon VPC CNI 插件版本，才能使用最新的受支持实例类型。有关更多信息，请参阅 the section called “Amazon VPC CNI”。

使用 Amazon Fargate 简化计算管理

本主题讨论使用 Amazon EKS 在 Amazon Fargate 上运行 Kubernetes 容器组（pod）。Fargate 是一种为容器按需提供大小合适的计算容量的技术。使用 Fargate，您不必再自己预置、配置或扩展虚拟机组即可运行容器。您无需再选择服务器类型、确定扩展节点组的时间和优化集群打包。

您可以控制要在 Fargate 上启动的容器组（pod）及其如何利用 Fargate 配置文件运行。Fargate 配置文件被定义为 Amazon EKS 集群的一部分。Amazon EKS 使用由 Amazon 构建的控制器（使用 Kubernetes 提供的上游可扩展模型）将 Kubernetes 与 Fargate 集成。这些控制器作为 Amazon EKS 托管的 Kubernetes 控制面板的一部分运行，负责将本机 Kubernetes 容器组（pod）调度到 Fargate 上。除了若干转换和验证准入控制器外，Fargate 控制器还包括一个与默认 Kubernetes 调度器一起运行的新调度器。在启动满足 Fargate 上的运行条件的容器组（pod）时，集群中运行的 Fargate 控制器会识别、更新容器组（pod）并将其安排到 Fargate 上。

本主题介绍 Fargate 上运行的容器组（pod）的不同组件，还列出了将 Fargate 与 Amazon EKS 结合使用时的特别注意事项。

Amazon Fargate 注意事项

以下是使用 Amazon EKS 上的 Fargate 时要考虑的一些事项。

- 在 Fargate 上运行的每个容器组都有自己的计算边界，不与其他容器组（pod）共享底层内核、CPU 资源、内存资源或弹性网络接口。
- 网络负载均衡器和 Application Load Balancer (ALB) 只能与具有 IP 目标的 Fargate 一起使用。有关更多信息，请参阅the section called “创建网络负载均衡器”和the section called “应用程序负载均衡”。
- Fargate 暴露的服务仅在目标类型 IP 模式下运行，而不是在节点 IP 模式下运行。推荐通过服务名称进行连接来检查托管节点上运行的服务和 Fargate 上运行服务的连接性。
- 在对容器组（pod）进行安排时，它们必须与 Fargate 配置文件匹配，才能在 Fargate 上运行。与 Fargate 配置文件不匹配的容器组可能会卡在 `Pending` 状态。如果存在匹配的 Fargate 配置文件，您可以删除已创建的待处理容器组（pod），将之重新调度到 Fargate 上。
- Fargate 上不支持 Daemonset。如果应用程序需要进程守护程序，则应将该进程守护程序重新配置为在容器组（pod）中作为边车容器运行。

- Fargate 上不支持特权容器。
- 在 Fargate 上运行的容器组（pod）无法在容器组（pod）清单中指定 `HostPort` 或 `HostNetwork`。
- 对于 Fargate 容器组（pod），默认 `nofile` 和 `nproc` 软限制为 1024，硬限制为 65535。
- GPU 目前在 Fargate 上不可用。
- 在 Fargate 上运行的容器组（pod）仅在私有子网上受支持（对 Amazon 服务具有 NAT 网关访问权限，但没有到互联网网关的直接路由），因此您的集群的 VPC 必须具有可用的私有子网。对于没有出站 Internet 访问的集群，请参阅 the section called “私有集群”。
- 您可以按照使用 Vertical Pod Autoscaler 调整容器组（pod）资源来设置 Fargate 容器组（pod）的正确初始 CPU 和内存大小，然后按照使用 Horizontal Pod Autoscaler 扩展容器组（pod）部署来扩展那些容器组（pod）。如果希望 Vertical Pod Autoscaler 自动将容器组（pod）重新部署到具有更大 CPU 和内存组合的 Fargate，请将 Vertical Pod Autoscaler 的模式设置为 `Auto` 或 `Recreate`，确保功能正常运行。有关更多信息，请参阅 GitHub 上的 Vertical Pod Autoscaler 文档。
- 必须为 VPC 启用 DNS 解析和 DNS 主机名。有关详细信息，请参阅查看和更新您的 VPC 的 DNS 支持。
- Amazon EKS Fargate 通过将虚拟机 (VM) 中的每个容器组隔离出来，从而为 Kubernetes 应用程序添加深度防御。此 VM 边界可防止在容器逃离情况下访问其他容器组使用的基于主机的资源，容器逃离是攻击容器化应用程序和获取容器外部资源访问权限的常见方法。

  使用 Amazon EKS 不会更改您在责任共担模式下的责任。您应该仔细考虑集群安全和治理控制的配置。隔离应用程序的最安全方法始终是在单独的集群中运行该程序。
- Fargate 配置文件支持从 VPC 辅助 CIDR 块指定子网。您可能想要指定辅助 CIDR 块。这是因为子网中可用 IP 地址的数量是有限的。因此，可在集群中创建的容器组（pod）数量受到限制。对容器组（pod）使用不同的子网，您可以增加可用 IP 地址的数量。有关更多信息，请参阅向 VPC 中添加 IPv4 CIDR 块。
- 部署到 Fargate 节点的容器组（pod）无法使用 Amazon EC2 实例元数据服务（IMDS）。如果在 Fargate 部署了需要 IAM 凭证的容器组（pod），请使用服务账户的 IAM 角色将之分配到容器组（pod）。如果容器组（pod）需要通过 IMDS 访问其他可用信息，则必须将此信息硬编码到容器组（pod）规范中。这包括容器组（pod）部署到的 Amazon 区域或可用区。
- 您不能将 Fargate 容器组（pod）部署到 Amazon Outposts、Amazon Wavelength 或 Amazon Local Zones。
- Amazon EKS 必须定期修补 Fargate 容器组（pod）来确保它们的安全。我们以减少影响的方式尝试更新，但是有时候，如果没有成功驱逐容器组（pod），必须删除它们。您可以采取一些措施来尽量减少中断。有关更多信息，请参阅 the section called “操作系统补丁事件”。

- 适用于 Amazon EKS 的 Amazon VPC CNI 插件安装在 Fargate 节点上。您不能将适用于 Amazon EKS 集群的 Alternate CNI 插件用于 Fargate 节点。
- Fargate 上运行的容器组（pod）会自动挂载 Amazon EFS 文件系统，无需手动执行驱动程序安装步骤。您不能将动态持久性卷预置与 Fargate 节点结合使用，但可以使用静态预置。
- Amazon EKS 不支持 Fargate Spot。
- 您无法将 Amazon EBS 卷挂载到 Fargate 容器组（pod）。
- 您可以在 Fargate 节点上运行 Amazon EBS CSI 控制器，但 Amazon EBS CSI 节点 DaemonSet 只能在 Amazon EC2 实例上运行。
- 在 Kubernetes 作业被标记为 `Completed` 或 `Failed` 后，作业创建的容器组（pod）通常会继续存在。此行为允许您查看日志和结果，但在使用 Fargate 的情况下，如果您过后不清理作业，则将会产生费用。

  要在作业完成或失败后自动删除相关的容器组（pod），您可以使用生存时间（TTL）控制器指定时间段。以下示例显示了在作业清单中指定 `.spec.ttlSecondsAfterFinished`。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: busybox
spec:
  template:
    spec:
      containers:
      - name: busybox
        image: busybox
        command: ["/bin/sh", "-c", "sleep 10"]
      restartPolicy: Never
  ttlSecondsAfterFinished: 60 # <-- TTL controller
```

Fargate 比较表

标准	Amazon Fargate
可以部署到 Amazon Outposts	否
可以部署到 Amazon 本地区域	否

标准	Amazon Fargate
可以运行需要 Windows 的容器	否
可以运行需要 Linux 的容器	是
可以运行需要 Inferentia 芯片的工作负载	否
可以运行需要 GPU 的工作负载	否
可以运行需要 ARM 处理器的工作负载	否
可以运行 Amazon Bottlerocket	否
容器组（pod）之间共享内核运行时环境	否 – 每个容器组（pod）拥有专用的内核
容器组之间共享 CPU、内存、存储和网络资源。	否 – 每个容器组（pod）都有专用资源，并且可以独立调整大小，能够最大限度地提高资源利用率。
容器组（pod）可以使用多于容器组（pod）规范中所请求的硬件和内存	否 – 但是，可以使用更大的 vCPU 和内存配置重新部署容器组（pod）。
必须部署和管理 Amazon EC2 实例	否
必须保护、维护和修补 Amazon EC2 实例的操作系统	否
可以在部署节点时提供引导参数，例如额外的 kubelet 实际参数。	否
可以从与分配给节点的 IP 地址不同的 CIDR 块为容器组（pod）分配 IP 地址。	否
可以通过 SSH 访问节点	否 – 没有主机操作系统要通过 SSH 访问的节点。
可以将您自己的自定义 AMI 部署到节点	否
可以将自己的自定义 CNI 部署到节点	否

标准	Amazon Fargate
必须自己更新节点 AMI	否
必须自己更新节点 Kubernetes 版本	否 – 您不管理节点。
可以为容器组（pod）使用 Amazon EBS 存储	否
可以为容器组（pod）使用 Amazon EFS 存储	是
可以为容器组（pod）使用适用于 Lustre 的 Amazon FSx 存储	否
可以对服务使用 Network Load Balancer	是，当使用创建网络负载均衡器时
Pod 可以在公有子网中运行	否
可以将不同的 VPC 安全组分配给各个容器组（pod）	是
可以运行 Kubernetes DaemonSets	否
支持容器组（pod）清单中的 `HostPort` 和 `HostNetwork`	否
Amazon 区域可用性	部分 Amazon EKS 支持的区域
可以在 Amazon EC2 专属主机上运行容器	否
定价	单个 Fargate 内存和 CPU 配置的成本。每个容器组（pod）都有自己的成本。有关更多信息，请参阅 Amazon Fargate 定价。

开始将 Amazon Fargate 用于集群

本主题介绍如何通过 Amazon EKS 集群开始在 Amazon Fargate 上运行容器组（pod）。

如果使用 CIDR 块限制对集群的公有端点的访问，建议您还启用私有端点访问。通过这种方式，Fargate 容器组（pod）可以与集群通信。在未启用私有端点的情况下，您指定用于公有访问的

CIDR 块必须包含来自 VPC 的出站源。有关更多信息，请参阅 the section called “集群终端节点访问”。

先决条件

现有集群。如果您还没有 Amazon EKS 集群，请参阅开始使用。

第 1 步：确保现有节点可以与 Fargate 容器组（pod）进行通信

如果您使用的是没有节点的新集群，或是只有托管节点组的集群（请参阅the section called “托管节点组”），可以跳至the section called “第 2 步：创建 Fargate 容器组（pod）执行角色”。

假设您正在使用已有关联节点的现有集群。确保这些节点上的容器组（pod）可以与 Fargate 上运行的容器组（pod）自由通信。在 Fargate 上运行的容器组（pod）会自动配置为使用与其关联的集群的集群安全组。确保集群中的任何现有节点都可以向集群安全组发送流量以及接收来自集群安全组的流量。托管节点组也会自动配置为使用集群安全组，所以您无需修改或检查它们是否具有此兼容性（请参阅the section called “托管节点组”）。

针对已使用 `eksctl` 或者 Amazon EKS 管理 Amazon CloudFormation 模板创建的现有节点组，您可以手动将集群安全组添加到节点。您也可以修改节点组的自动扩缩组启动模板，以便将集群安全组附加到实例。有关更多信息，请参阅《Amazon VPC 用户指南》中的更改实例的安全组。

您可以在 Amazon Web Services 管理控制台 中集群的 Networking（联网）部分下，检查您集群的安全组。或者，您可以使用下面的 Amazon CLI 命令进行这项操作。使用此命令时，请将 `<my-cluster>` 替换为您的集群名称。

```
aws eks describe-cluster --name <my-cluster> --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

第 2 步：创建 Fargate 容器组（pod）执行角色

当您的集群在 Amazon Fargate 上创建容器组（pod）时，在 Fargate 基础设施上运行的组件必须代表您调用 Amazon API。Amazon EKS 容器组（pod）执行角色提供执行此操作的 IAM 权限。要创建 Amazon Fargate 容器组（pod）执行角色，请参阅 the section called “容器执行 IAM 角色”。

> **Note**
>
> 如果您使用 `--fargate` 选项通过 `eksctl` 创建了集群，则集群已具有容器组（pod）执行角色，您可以使用模式 `eksctl-my-cluster-FargatePodExecutionRole-`

ABCDEFGHIJKL 在 IAM 控制台中找到该角色。同理，如果您使用 eksctl 创建 Fargate 配置文件，则 eksctl 会创建您的容器组（pod）执行角色（如果尚未创建）。

第 3 步：为集群创建 Fargate 配置文件

您必须先定义一个 Fargate 配置文件，指定在启动时哪些容器组（pod）使用 Fargate，然后才能调度在集群中的 Fargate 上运行的容器组（pod）。有关更多信息，请参阅 the section called “定义配置文件”。

> **Note**
>
> 如果您使用 --fargate 选项通过 eksctl 创建了集群，则已经为您的集群创建了 Fargate 配置文件，而且其包含 kube-system 和 default 命名空间中所有容器组（pod）的选择器。使用以下程序为您想要用于 Fargate 的任何其他命名空间创建 Fargate 配置文件。

您可以使用以下任一工具创建 Fargate 配置文件：

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”

eksctl

此过程需要 eksctl 版本 0.215.0 或更高版本。可以使用以下命令来查看您的版本：

```
eksctl version
```

有关安装或升级 eksctl 的说明，请参阅 eksctl 文档中的 Installation。

使用 eksctl 创建 Fargate 配置文件

使用以下 eksctl 命令创建 Fargate 配置文件，并将所有 <example value> 替换为您自己的值。您需要指定命名空间。但是，--labels 选项不是必选。

```
eksctl create fargateprofile \
    --cluster <my-cluster> \
    --name <my-fargate-profile> \
```

```
    --namespace <my-kubernetes-namespace> \
    --labels <key=value>
```

您可以将某些通配符用于 <my-kubernetes-namespace> 和 <key=value> 标签。有关更多信息，请参阅 the section called “Fargate 配置文件通配符”。

Amazon Web Services 管理控制台

使用 Amazon Web Services 管理控制台 创建 Fargate 配置文件

1. 打开 Amazon EKS 控制台。
2. 选择要为其创建 Fargate 配置文件的集群。
3. 请选择 Compute（计算）选项卡。
4. 在 Fargate profiles（Fargate 配置文件）下，选择 Add Fargate profile（添加 Fargate 配置文件）。
5. 在 Configure Fargate profile（配置 Fargate 配置文件）页面上，执行以下操作：
   a. 对于 Name（名称），为 Fargate 配置文件输入名称。名称必须唯一。
   b. 对于容器组（pod）执行角色，选择要用于您 Fargate 配置文件的容器组（pod）执行角色。将仅显示具有 eks-fargate-pods.amazonaws.com 服务委托人的 IAM 角色。如果您未看到列出的任何角色，则必须创建一个角色。有关更多信息，请参阅 the section called “容器执行 IAM 角色”。
   c. 根据需要修改选定的子网。

      > **Note**
      >
      > Fargate 上运行的容器组（pod）仅支持私有子网。

   d. 对于 Tags（标签），您可以自行选择是否为 Fargate 配置文件添加标签。这些标签不会传播到与配置文件关联的其他资源，如容器组（pod）。
   e. 选择下一步。
6. 在配置容器组（pod）选择页面上，执行以下操作：
   a. 对于命名空间，输入与容器组（pod）匹配的命名空间。
      - 您可以使用匹配的特定命名空间，例如 kube-system 或 default。
      - 您可以使用某些通配符（例如 prod-*）以匹配多个命名空间（例如，prod-deployment 和 prod-test)。有关更多信息，请参阅 the section called “Fargate 配置文件通配符”。
   b. （可选）将 Kubernetes 标签添加到选择器中。特别是将它们添加到指定命名空间中的容器组（pod）需要匹配的那个。

- 您可以将标签 `infrastructure: fargate` 添加到选择器中，以便只有指定命名空间中也具有 `infrastructure: fargate` Kubernetes 标签的容器组（pod）与选择器匹配。
- 您可以使用某些通配符（例如 `key?: value?`）以匹配多个命名空间（例如，`keya: valuea` 和 `keyb: valueb`）。有关更多信息，请参阅 the section called “Fargate 配置文件通配符”。

c. 选择下一步。

7. 在 Review and create（查看和创建）页面上，查看 Fargate 配置文件的信息，然后选择 Create（创建）。

第 4 步：更新 CoreDNS

预设情况下，CoreDNS 配置为在 Amazon EKS 集群的 Amazon EC2 基础设施上运行。如果仅在集群中的 Fargate 上运行容器组（pod），请完成以下步骤。

> **Note**
>
> 如果使用 `--fargate` 选项用 `eksctl` 创建集群，则可以跳至 the section called “后续步骤”。

1. 使用以下命令为 CoreDNS 创建 Fargate 配置文件。将 `<my-cluster>` 替换为您的集群名称，将 `<111122223333>` 替换为您的账户 ID，将 `<AmazonEKSFargatePodExecutionRole>` 替换为您的容器组（pod）执行角色名称，并将 `<000000000000000a>`、`<000000000000000b>` 和 `<000000000000000c>` 替换为您的私有子网 ID。如果没有容器组（pod）执行角色，则必须先创建一个（请参阅the section called “第 2 步：创建 Fargate 容器组（pod）执行角色”）。

> **Important**
>
> 角色 ARN 不能包含 `/` 之外的路径。例如，如果您的角色名称为 `development/apps/AmazonEKSFargatePodExecutionRole`，则需要在为该角色指定 ARN 时将其更改为 `AmazonEKSFargatePodExecutionRole`。角色 ARN 的格式必须为 `arn:aws-cn:iam::<111122223333>:role/<AmazonEKSFargatePodExecutionRole>`。

```
aws eks create-fargate-profile \
    --fargate-profile-name coredns \
    --cluster-name <my-cluster> \
```

```
    --pod-execution-role-arn arn:aws-cn:iam::<111122223333>:role/
<AmazonEKSFargatePodExecutionRole> \
    --selectors namespace=kube-system,labels={k8s-app=kube-dns} \
    --subnets subnet-<000000000000000a> subnet-<000000000000000b> subnet-
<000000000000000c>
```

2. 触发 coredns 部署的推出。

```
kubectl rollout restart -n kube-system deployment coredns
```

后续步骤

- 您可以开始使用以下工作流迁移现有应用程序来在 Fargate 上运行现有应用程序。
  a. the section called “创建 Fargate 配置文件”，该配置文件与您应用程序的 Kubernetes 命名空间和 Kubernetes 标签匹配。
  b. 删除并重新创建所有现有的容器组（pod），以便可以在 Fargate 上调度容器组。修改 <namespace> 和 <deployment-type> 以更新您的特定容器组（pod）。

  ```
  kubectl rollout restart -n <namespace> deployment <deployment-type>
  ```

- 部署 the section called “应用程序负载均衡”，允许在 Fargate 上运行的容器组（pod）的入口对象。
- 您可以使用 the section called “Vertical Pod Autoscaler” 为 Fargate 容器组（pod）设置正确的初始 CPU 和内存大小，然后使用 the section called “Horizontal Pod Autoscaler” 来扩展这些容器组（pod）。如果您希望 Vertical Pod Autoscaler 自动将容器组（pod）重新部署到具有更高 CPU 和内存组合的 Fargate，请将 Vertical Pod Autoscaler 的模式设置为 Auto 或 Recreate。这是为了确保正确的功能。有关更多信息，请参阅 GitHub 上的 Vertical Pod Autoscaler 文档。
- 您可以依照这些说明设置 Amazon Distro for OpenTelemetry (ADOT) 收集器，用于监控应用程序。

定义启动时将使用 Amazon Fargate 的容器组（pod）

您必须先定义至少一个 Fargate 配置文件，指定在启动时哪些容器组（pod）使用 Fargate，然后才能调度集群中的 Fargate 上的容器组（pod）。

作为管理员，您可以使用 Fargate 配置文件声明哪些容器组（pod）在 Fargate 上运行。您可以通过配置文件的选择器执行此操作。您最多可以为每个配置文件添加五个选择器。每个选择器都必须包含一个命名空间。选择器还可以包含标签。标注字段由多个可选键值对组成。与选择器匹配的容器组（pod）被安排在 Fargate 上。容器组（pod）使用选择器中指定的命名空间和标签进行匹配。如果定义了没

有标签的命名空间选择器，Amazon EKS 会尝试使用配置文件将在该命名空间中运行的所有容器组（pod）调度到 Fargate。如果待调度的容器组（pod）与 Fargate 配置文件中的任意一个选择器都匹配，则该容器组（pod）将被调度到 Fargate 上。

如果容器组（pod）匹配多个 Fargate 配置文件，您可以指定容器组（pod）使用的配置文件，方法为将以下 Kubernetes 标签添加到容器组（pod）规范：`eks.amazonaws.com/fargate-profile: my-fargate-profile`。但是，容器组（pod）仍必须匹配该配置文件中的选择器，才能被调度到 Fargate 上。Kubernetes 亲和性/反亲和性规则不适用，Amazon EKS Fargate 容器组（pod）并不需要这些规则。

创建 Fargate 配置文件时，必须指定容器组（pod）执行角色。此执行角色适用于使用配置文件在 Fargate 基础设施上运行的 Amazon EKS 组件。该角色将被添加到集群的 Kubernetes 基于角色的访问控制（RBAC）以进行授权。这样，Fargate 基础设施上运行的 `kubelet` 可以注册到您的 Amazon EKS 集群，并在您的集群中作为节点显示。容器组（pod）执行角色还提供对 Fargate 基础设施的 IAM 权限，以允许对 Amazon ECR 映像存储库进行读取访问。有关更多信息，请参阅 the section called “容器执行 IAM 角色”。

您无法更改 Fargate 配置文件。但是，您可以创建新的更新配置文件来替换现有配置文件，然后删除原始配置文件。

> **Note**
>
> 删除配置文件后，使用 Fargate 配置文件运行的任何容器组（pod）都会停止并进入待处理状态。

如果集群中有任何 Fargate 配置文件处于 `DELETING` 状态，您必须等待删除该 Fargate 配置文件后，才能在该集群中创建其他配置文件。

> **Note**
>
> Fargate 目前不支持 Kubernetes topologySpreadConstraints。

Amazon EKS 和 Fargate 将容器组（pod）分布在 Fargate 配置文件定义的每个子网中。但是，最终可能会出现不均匀的分布。如果您必须获得均匀的分布，请使用两个 Fargate 配置文件。在您希望部署两个副本并且不希望造成任何停机的方案中，均匀的分布非常重要。我们建议每个配置文件只有一个子网。

Fargate 配置文件组件

Fargate 配置文件中包含以下组件。

Pod 执行角色

当您的集群在 Amazon Fargate 上创建容器组（pod）时，在 Fargate 基础设施上运行的 `kubelet` 必须代表您调用 Amazon API。例如，它需要调用才能从 Amazon ECR 提取容器镜像。Amazon EKS 容器组（pod）执行角色提供执行此操作的 IAM 权限。

创建 Fargate 配置文件时，必须指定要用于容器组（pod）的容器组（pod）执行角色。此角色将被添加到集群的 Kubernetes 基于角色的访问控制（RBAC）以进行授权。这允许在 Fargate 基础设施上运行的 `kubelet` 注册到您的 Amazon EKS 集群，以便它可以作为节点显示在您的集群中。有关更多信息，请参阅 the section called “容器执行 IAM 角色”。

子网

将容器组（pod）启动到其中的使用此配置文件的子网的 ID。目前，在 Fargate 上运行的容器组（pod）没有分配公有 IP 地址。因此，此参数仅接受私有子网（没有到互联网网关的直接路由）。

选择器

要匹配容器组（pod）以使用此 Fargate 配置文件的选择器。您最多可以在 Fargate 配置文件中指定五个选择器。选择器具有以下组件：

- 命名空间 – 您必须为选择器指定命名空间。选择器仅匹配在此命名空间中创建的容器组（pod）。但是，您可以创建多个选择器来定位多个命名空间。
- 标签 – 您可以选择指定 Kubernetes 标签来匹配选择器。选择器只匹配具有在选择器中指定的所有标签的容器组（pod）。

Fargate 配置文件通配符

除了 Kubernetes 允许的字符外，您还可以在命名空间、标签键和标签值的选择器标准中使用 * 和 ?：

- * 表示无、一个或多个字符。例如，`prod*` 可以表示 `prod` 和 `prod-metrics`。
- ? 表示单个字符（例如，`value?` 可以表示 `valuea`）。但是，它不能表示 `value` 和 `value-a`，因为 ? 只能准确地表示一个字符。

这些通配符可以在任何位置组合使用（例如，`prod*`、`*dev` 以及 `frontend*?`）。不支持其他通配符和模式匹配形式，例如正则表达式。

如果容器组（pod）规范中有多个配置文件与命名空间和标签匹配，Fargate 会根据配置文件名称的字母数字排序来选取配置文件。例如，如果配置文件 A（名称为 `beta-workload`）和配置文件 B（名称为 `prod-workload`）都有匹配的选择器可供要启动的容器组（pod）使用，Fargate 会为容器组（pod）选取配置文件 A (`beta-workload`)。容器组（pod）具有标签，显示容器组（pod）上有配置文件 A（例如 `eks.amazonaws.com/fargate-profile=beta-workload`）。

如果要将现有 Fargate 容器组（pod）迁移到使用通配符的新配置文件中，有两种方法可执行此操作：

- 使用匹配的选择器创建一个新的配置文件，然后删除旧的配置文件。标有旧配置文件的容器组（pod）将被重新安排到新的匹配配置文件中。
- 若要迁移工作负载，但不确定每个 Fargate 容器组（pod）上有哪些 Fargate 标签，则可以使用以下方法。创建一个新的配置文件，首先在同一个集群上的配置文件中按字母数字排序配置文件名称。然后，回收需要迁移到新配置文件的 Fargate 容器组（pod）。

创建 Fargate 配置文件

这一部分介绍如何创建 Fargate 配置文件。您还必须已经创建了要用于 Fargate 配置文件的容器组（pod）执行角色。有关更多信息，请参阅 the section called “容器执行 IAM 角色”。在 Fargate 上运行的容器组（pod）仅在私有子网上受支持（对 Amazon 服务具有 NAT 网关访问权限，但没有到互联网网关的直接路由）。因此，您的集群的 VPC 必须有可用的私有子网。

您可以使用以下工具创建配置文件。

- the section called “`eksctl`”
- the section called “Amazon Web Services 管理控制台”

`eksctl`

使用 **`eksctl`** 创建 Fargate 配置文件

使用以下 `eksctl` 命令创建 Fargate 配置文件，并将所有示例值替换为您自己的值。您需要指定命名空间。但是，`--labels` 选项不是必选。

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --name my-fargate-profile \
    --namespace my-kubernetes-namespace \
    --labels key=value
```

您可以将某些通配符用于 my-kubernetes-namespace 和 key=value 标签。有关更多信息，请参阅 the section called “Fargate 配置文件通配符”。

Amazon Web Services 管理控制台

使用 Amazon Web Services 管理控制台 创建 Fargate 配置文件

1. 打开 Amazon EKS 控制台。
2. 选择要为其创建 Fargate 配置文件的集群。
3. 请选择 Compute（计算）选项卡。
4. 在 Fargate profiles（Fargate 配置文件）下，选择 Add Fargate profile（添加 Fargate 配置文件）。
5. 在 Configure Fargate profile（配置 Fargate 配置文件）页面上，执行以下操作：
   a. 对于 Name（名称），为 Fargate 配置文件输入唯一名称，例如 my-profile。
   b. 对于容器组（pod）执行角色，选择要用于您 Fargate 配置文件的容器组（pod）执行角色。将仅显示具有 eks-fargate-pods.amazonaws.com 服务委托人的 IAM 角色。如果您未看到列出的任何角色，则必须创建一个角色。有关更多信息，请参阅 the section called “容器执行 IAM 角色”。
   c. 根据需要修改选定的子网。

      > **Note**
      > Fargate 上运行的容器组（pod）仅支持私有子网。

   d. 对于 Tags（标签），您可以自行选择是否为 Fargate 配置文件添加标签。这些标签不会传播到与配置文件关联的其他资源，例如容器组（pod）。
   e. 选择下一步。
6. 在配置容器组（pod）选择页面上，执行以下操作：
   a. 对于命名空间，输入与容器组（pod）匹配的命名空间。
      - 您可以使用匹配的特定命名空间，例如 kube-system 或 default。
      - 您可以使用某些通配符（例如 prod-*）以匹配多个命名空间（例如，prod-deployment 和 prod-test）。有关更多信息，请参阅 the section called “Fargate 配置文件通配符”。
   b. （可选）将 Kubernetes 标签添加到选择器中。特别是将它们添加到指定命名空间中的容器组（pod）需要匹配的那个。
      - 您可以将标签 infrastructure: fargate 添加到选择器中，以便只有指定命名空间中也具有 infrastructure: fargate Kubernetes 标签的容器组（pod）与选择器匹配。

- 您可以使用某些通配符（例如 key?: value?）以匹配多个命名空间（例如，keya: valuea 和 keyb: valueb)。有关更多信息，请参阅 the section called “Fargate 配置文件通配符”。

c. 选择下一步。

7. 在 Review and create（查看和创建）页面上，查看 Fargate 配置文件的信息，然后选择 Create（创建）。

删除 Fargate 配置文件

本主题介绍如何删除 Fargate 配置文件。当您删除 Fargate 配置文件时，使用该配置文件调度到 Fargate 的所有容器组（pod）都将被删除。如果这些容器组（pod）与另一个 Fargate 配置文件匹配，则将使用该配置文件在 Fargate 上调度这些容器组（pod）。如果它们不再匹配任何 Fargate 配置文件，则不会被安排到 Fargate，并仍可能处于待处理状态。

集群中一次只能有一个 Fargate 配置文件处于 DELETING 状态。等待 Fargate 配置文件完成删除，然后才能删除该集群中的任何其他配置文件。

可以使用以下任一工具删除配置文件：

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”
- the section called “Amazon CLI”

eksctl

使用 **eksctl** 删除 Fargate 配置文件

使用以下命令从集群中删除配置文件。将所有 *example value* 替换为您自己的值。

```
eksctl delete fargateprofile  --name my-profile --cluster my-cluster
```

Amazon Web Services 管理控制台

使用 Amazon Web Services 管理控制台删除 Fargate 配置文件

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。在集群列表中，选择要从中删除 Fargate 配置文件的集群。

3. 请选择 Compute（计算）选项卡。
4. 选择要删除的 Fargate 配置文件，然后选择 Delete（删除）。
5. 在 Delete Fargate profile（删除 Fargate 配置文件）页面上，输入配置文件的名称，然后选择 Delete（删除）。

Amazon CLI

使用 Amazon CLI 删除 Fargate 配置文件

使用以下命令从集群中删除配置文件。将所有 *example value* 替换为您自己的值。

```
aws eks delete-fargate-profile --fargate-profile-name my-profile --cluster-name my-cluster
```

了解 Fargate 容器组（pod）配置详细信息

本节介绍在 Amazon Fargate 上运行 Kubernetes 容器组（pod）的一些唯一容器组（pod）配置详细信息。

Pod CPU 和内存

使用 Kubernetes，您可以定义请求、最低 vCPU 量以及分配给容器组（pod）中每个容器的内存资源。Kubernetes 会调度容器组（pod），确保至少每个容器组（pod）的请求资源可用于计算资源。有关更多信息，请参阅 Kubernetes 文档中的管理容器的计算资源。

> **Note**
>
> 由于 Amazon EKS Fargate 对每个节点只运行一个容器组（pod），因此在资源较少的情况下不会出现驱逐容器组（pod）的情况。所有 Amazon EKS Fargate 容器组（pod）都以担保优先级运行，因此请求的 CPU 和内存必须等于所有容器的限制。有关更多信息，请参阅 Kubernetes 文档中的为 Pod（一组容器）配置服务质量。

在 Fargate 上调度容器组（pod）后，容器组（pod）规格中的 vCPU 和内存预留将确定为容器组（pod）预置的 CPU 和内存量。

- 超出所有 Init 容器的最大请求用于确定 Init 请求 vCPU 和内存要求。

- 将所有长时间运行的容器的请求相加来确定长时间运行的请求的 vCPU 和内存要求。
- 然后为要用于容器组（pod）的 vCPU 和内存请求选择先前两个值中较大的值。
- Fargate 会为所需 Kubernetes 组件（`kubelet`、`kube-proxy` 和 `containerd`）的每个容器组（pod）的内存预留增加 256 MB。

Fargate 向上舍入到下列最接近 vCPU 与内存请求总和的计算配置，确保容器组（pod）始终拥有运行所需的资源。

如果未指定 vCPU 和内存组合，则使用最小的可用组合（0.25 vCPU 和 0.5GB 内存）。

下表显示了 Fargate 上运行的容器组（pod）可以使用的 vCPU 和内存组合。

vCPU 值	内存值
.25 vCPU	0.5GB、1GB、2GB
.5 vCPU	1GB、2GB、3GB、4GB
1 个 vCPU	2GB、3GB、4GB、5GB、6GB、7GB、8GB
2 个 vCPU	4GB 到 16GB 之间（以 1GB 为增量）
4 个 vCPU	8GB 到 30GB 之间（以 1GB 为增量）
8 个 vCPU	介于 16GB 到 60GB 之间（以 4GB 为增量）
16 个 vCPU	介于 32GB 到 120GB 之间（以 8GB 为增量）

为 Kubernetes 组件保留的额外内存可能会导致 Fargate 任务的 vCPUs 数量超过请求预置的 vCPU 数量。例如，由于没有具有 1 个 vCPU 和 9 GB 内存的任务可用，因此如果请求内容为 1 个 vCPU 和 8 GB 内存，则会为其内存请求增加 256 MB，并会为 Fargate 任务预置 2 个 vCPU 和 9 GB 内存。

Fargate 上运行的容器组（pod）大小与 Kubernetes 使用 `kubectl get nodes` 报告的节点大小之间没有相关性。报告的节点大小通常大于容器组（pod）的容量。您可以使用以下命令验证容器组（pod）容量。将 *`default`* 替换为容器组（pod）的命名空间，并将 *`pod-name`* 替换为容器组（pod）的名称。

```
kubectl describe pod --namespace default pod-name
```

示例输出如下。

```
[...]
annotations:
    CapacityProvisioned: 0.25vCPU 0.5GB
[...]
```

`CapacityProvisioned` 注释表示强制执行的容器组（pod）容量，决定了在 Fargate 上运行的容器组（pod）的成本。有关这些计算配置的定价信息，请参阅 Amazon Fargate 定价。

Fargate 存储

Fargate 上运行的容器组（pod）会自动挂载 Amazon EFS 文件系统，无需手动执行驱动程序安装步骤。您不能将动态持久性卷预置与 Fargate 节点结合使用，但可以使用静态预置。有关更多信息，请参阅 GitHub 上的 Amazon EFS CSI 驱动程序。

完成预置后，在 Fargate 上运行的每个容器组（pod）都会获得 20 GiB 的默认临时存储空间。这种类型的存储将在容器组（pod）停止后被删除。在 Fargate 上启动的新容器组（pod）会默认启用临时存储卷加密。使用 Amazon Fargate 托管密钥通过 AES-256 加密算法对临时容器组（pod）存储进行加密。

> **Note**
>
> 在 Fargate 上运行的 Amazon EKS 容器组（pod）的默认可用存储空间值小于 20 GiB。这是因为部分空间会被 `kubelet` 以及 Kubernetes 内部加载的其他容器组（pod）模块使用。

您最高可以将临时存储总量增加到 175GiB。要使用 Kubernetes 配置大小，请指定一个容器组（pod）中每个容器的 `ephemeral-storage` 资源请求数。在调度容器组（pod）时，Kubernetes 会确保每个容器组（pod）的资源请求总和小于 Fargate 任务的容量。有关更多信息，请参阅 Kubernetes 文档中的为 Pod 和容器管理资源。

Amazon EKS Fargate 预置的临时存储空间会高于请求的容量，以满足系统使用的需要。例如，假设您请求 100GiB 的空间，则系统会为 Fargate 任务预置 115GiB 的临时存储空间。

为 Amazon Fargate 操作系统补丁事件设置操作

Amazon EKS 定期修补 Amazon Fargate 节点的操作系统以确保它们的安全。作为修补过程的一部分，我们会回收节点来安装操作系统补丁。尝试更新的方式对您的服务造成的影响最小。但是，如果容

器组（pod）没有成功被驱逐，有时必须删除容器组。以下是您可以采取的最大限度减少潜在中断的操作：

- 设置适当的容器组（pod）中断预算（PDB），控制同时关闭的容器组（pod）数量。
- 创建 Amazon EventBridge 规则，在删除容器组（pod）之前处理失败的驱逐。
- 在您所收通知中发布的驱逐日期之前，手动重启受影响的容器组（pod）。
- 在 Amazon 用户通知中创建通知配置。

Amazon EKS 与 Kubernetes 社区密切合作，以便尽快提供错误修复和安全补丁。所有 Fargate 容器组（pod）都从最新的 Kubernetes 修补程序版本开始，可从集群的 Kubernetes 版本的 Amazon EKS 中获取。如果您的容器组（pod）具有较旧的修补程序版本，Amazon EKS 可能会回收容器组，以便将其更新到最新版本。这可以确保您的容器组（pod）配备了最新的安全更新。这样一来，如果存在关键常见漏洞和风险（CVE）问题，您可以随时了解最新信息以降低安全风险。

Amazon Fargate 操作系统更新后，Amazon EKS 将向您发送通知，其中包含受影响的资源和即将驱逐容器组（pod）的日期。如果提供的日期不便进行驱逐，您可以在通知中于选择发布的驱逐日期之前，手动重启受影响的容器组（pod）。系统将驱逐您在收到通知之前创建的任何容器组（pod）。有关如何手动重启容器组（pod）的更多说明，请参阅 Kubernetes 文档。

要限制在回收容器组（pod）时同时停机的容器组数量，您可以设置容器组（pod）中断预算（PDB）。您可以使用 PDB 根据每个应用程序的要求定义最低可用性，同时仍允许进行更新。PDB 的最低可用性必须低于 100%。有关更多信息，请参阅 Kubernetes 文档中的为应用程序设置干扰预算。

Amazon EKS 使用驱逐 API 在遵守您为应用程序设置的 PDB 的同时安全地耗尽容器组（pod）。容器组（pod）被可用区驱逐出来，以最大限度地减少影响。如果驱逐成功，新容器组（pod）将获得最新的补丁，无需进一步操作。

当容器组（pod）的驱逐失败时，Amazon EKS 会向您的账户发送一个事件，其中包含驱逐失败的容器组（pod）的详细信息。您可以在计划的终止时间之前对消息采取行动。具体时间根据补丁的紧迫性而有所不同。时间一到，Amazon EKS 会尝试再次驱逐容器组（pod）。但是，如果驱逐失败，这次不会发送新的事件。如果驱逐再次失败，则会定期删除您现有的容器组（pod），以便新容器组（pod）可以拥有最新的补丁。

以下是在容器组（pod）驱逐失败时收到的示例事件。其中包含有关集群、容器组（pod）名称、容器组（pod）命名空间、Fargate 配置文件和计划终止时间的详细信息。

```
{
    "version": "0",
```

```
    "id": "12345678-90ab-cdef-0123-4567890abcde",
    "detail-type": "EKS Fargate Pod Scheduled Termination",
    "source": "aws.eks",
    "account": "111122223333",
    "time": "2021-06-27T12:52:44Z",
    "region": "region-code",
    "resources": [
        "default/my-database-deployment"
    ],
    "detail": {
        "clusterName": "my-cluster",
        "fargateProfileName": "my-fargate-profile",
        "podName": "my-pod-name",
        "podNamespace": "default",
        "evictErrorMessage": "Cannot evict pod as it would violate the pod's disruption
 budget",
        "scheduledTerminationTime": "2021-06-30T12:52:44.832Z[UTC]"
    }
}
```

此外，将多个 PDB 与一个容器组（pod）关联可能会导致驱逐失败事件。此事件将返回以下错误消息。

```
"evictErrorMessage": "This pod has multiple PodDisruptionBudget, which the eviction
 subresource does not support",
```

您可以基于此事件创建所需的操作。例如，要控制驱逐容器组（pod）的方式，您可以调整容器组（pod）中断预算（PDB）。更具体地说，假设您从指定可用容器组（pod）的目标百分比的 PDB 开始。在升级期间强制终止容器组（pod）之前，您可以将 PDB 调整为不同百分比的容器组（pod）。要接收此事件，您必须在 Amazon 账户和集群所属的 Amazon 区域中创建 Amazon EventBridge 规则。该规则必须使用以下自定义模式。有关更多信息，请参阅《Amazon EventBridge 用户指南》中的创建对事件作出反应的 Amazon EventBridge 规则。

```
{
  "source": ["aws.eks"],
  "detail-type": ["EKS Fargate Pod Scheduled Termination"]
}
```

可以为事件设置合适的目标来捕获它。有关可用目标的完整列表，请参阅《Amazon EventBridge 用户指南》中的 Amazon EventBridge 目标。您还可以在 Amazon 用户通知中创建通知配置。使用 Amazon Web Services 管理控制台创建通知时，在事件规则下，为 Amazon 服务名称选择 Elastic

Kubernetes Service（EKS），为事件类型选择 EKS Fargate 容器组（pod）计划的终止。有关更多信息，请参阅《Amazon 用户通知用户指南》中的Amazon 用户通知入门。

有关 EKS 容器组（pod）驱逐的常见问题，请参阅 Amazon re:Post 中的常见问题解答：Fargate 容器组（pod）驱逐通知。

收集 Amazon Fargate 应用程序和使用情况指标

您可以收集系统指标和 Amazon Fargate 的 CloudWatch 使用量指标。

应用程序指标

对于在 Amazon EKS 和 Amazon Fargate 上运行的应用程序，您可以使用适用于 OpenTelemetry 的 Amazon Distro（ADOT）。ADOT 允许您收集系统指标并将其发送到 CloudWatch Container Insights 控制面板。要开始为在 Fargate 上运行的应用程序使用 ADOT，请参阅 ADOT 文档中的将 CloudWatch Container Insights 与 Amazon Distro for OpenTelemetry 结合使用。

使用情况指标

您可以使用 CloudWatch 使用情况指标来提供账户资源使用情况的可见性。这些指标可在 CloudWatch 图表和控制面板上直观呈现当前的服务使用情况。

Amazon Fargate 用量指标与 Amazon 服务配额对应。您可以配置警报，以在用量接近服务配额时向您发出警报。有关 Fargate 服务配额的更多信息，请参阅 the section called “服务配额”。

Amazon Fargate 在 `Amazon/Usage` 命名空间中发布以下指标。

指标	描述
`ResourceCount`	您账户中运行的指定资源的总数量。资源由与指标关联的维度定义。

以下维度用于优化由 Amazon Fargate 发布的使用情况指标。

维度	描述
`Service`	包含该资源的 Amazon 服务的名称。对于 Amazon Fargate 用量指标，此维度的值为 `Fargate`。

维度	描述
Type	正在报告的实体的类型。目前，Amazon Fargate 用量指标的唯一有效值为 Resource。
Resource	正在运行的资源的类型。 目前，Amazon Fargate 会返回有关 Fargate 按需使用情况的信息。Fargate 按需使用情况的资源值为 OnDemand。 [注意] ==== Fargate 按需使用情况结合了使用 Fargate 的 Amazon EKS Pod、使用 Fargate 启动类型的 Amazon ECS 任务和使用 FARGATE 容量提供程序的 Amazon ECS 任务。 ====
Class	所跟踪的资源的类。目前，Amazon Fargate 不使用类维度。

创建 CloudWatch 警报以监控 Fargate 资源使用情况指标

Amazon Fargate 提供 CloudWatch 使用情况指标，这些指标与 Fargate 按需资源使用情况的 Amazon 服务配额相对应。在 Service Quotas 控制台中，您可以在图表上可视化您的使用情况。还可以配置警报，以在用量接近服务配额时向您发出警报。有关更多信息，请参阅 the section called “收集指标”。

使用以下步骤根据 Fargate 资源使用情况指标创建 CloudWatch 警报。

1. 访问 https://console.aws.amazon.com/servicequotas/，打开服务配额控制台。
2. 在左侧导航窗格中，选择 Amazon 服务。
3. 从 Amazon 服务列表中，搜索并选择 Amazon Fargate。
4. 在 Service quotas（服务配额）列表中，选择要为其创建警报的 Fargate 使用情况配额。
5. 在 Amazon CloudWatch 警报部分中，选择 Create（创建）。
6. 对于警报阈值，选择要设置为警报值的适用配额值的百分比。
7. 对于警报名称，输入警报名称，然后选择创建。

为集群启动 Amazon Fargate 日志记录

Fargate 上的 Amazon EKS 提供了一个基于 Fluent Bit 的内置日志路由器。这意味着您没有明确地将 Fluent Bit 容器作为边车容器运行，但 Amazon 会为您运行该容器。您只需配置日志路由器即可。通过必须满足以下条件的专用 `ConfigMap` 进行配置：

- 名为 `aws-logging`
- 在名为 `aws-observability` 的专用命名空间中创建
- 不能超过 5300 个字符。

一旦您创建了 `ConfigMap`，Fargate 上的 Amazon EKS 会自动检测到它并使用它来配置日志路由器。Fargate 使用 Amazon for Fluent Bit 的版本是 Fluent Bit 的一种上游合规发行版，由Amazon托管。有关更多信息，请参阅 GitHub 上的 Amazon for Fluent Bit。

日志路由器允许您使用广泛的Amazon服务进行日志分析和存储。您可以将日志从 Fargate 直接流式传输到 Amazon CloudWatch、Amazon OpenSearch Service。您还可以通过 Amazon Data Firehose 将日志流式传输到 Amazon S3、Amazon Kinesis Data Streams 和合作伙伴工具等目标。

- 现有的 Fargate 配置文件，用于指定您将 Fargate 容器组（pod）部署到的现有 Kubernetes 命名空间。有关更多信息，请参阅 the section called “第 3 步：为集群创建 Fargate 配置文件”。
- 现有的 Fargate 容器组（pod）执行角色。有关更多信息，请参阅 the section called “第 2 步：创建 Fargate 容器组（pod）执行角色”。

日志路由器配置

> **Important**
>
> 要成功发布日志，必须允许集群所在的 VPC 对日志目标的网络访问权限。这主要涉及用户为其 VPC 自定义出口规则。有关使用 CloudWatch 的示例，请参阅《Amazon CloudWatch Logs 用户指南》中的 Using CloudWatch Logs with interface VPC endpoints。

在以下步骤中，将所有 *`example value`* 替换为您自己的值。

1. 创建一个名为 `aws-observability` 的专用 Kubernetes 命名空间。
   a. 将以下内容保存到计算机上名为 `aws-observability-namespace.yaml` 的文件中。`name` 的值必须为 `aws-observability`，并且 `aws-observability: enabled` 标注是必需的。

```
kind: Namespace
apiVersion: v1
metadata:
  name: aws-observability
  labels:
    aws-observability: enabled
```

b. 创建命名空间。

```
kubectl apply -f aws-observability-namespace.yaml
```

2. 创建带有 Fluent Conf 数据值的 ConfigMap，以将容器日志发送到某个目标。Fluent Conf 为 Fluent Bit，是一种快速轻量级日志处理器配置语言，用于将容器日志路由到您选择的日志目标。有关更多信息，请参阅 Fluent Bit 文档中的配置文件。

> **Important**
>
> 典型 Fluent Conf 中包含的主要部分为 Service、Input、Filter 和 Output。但是，Fargate 日志路由器仅接受：
>
> - Filter 和 Output 部分。
> - Parser 部分。
>
> 如果您提供任何其他部分，将被拒绝。

Fargate 日志路由器管理 Service 和 Input 部分。它包含以下 Input 部分，无法修改，并且您的 ConfigMap 也不需要该部分。但是，您可以从中获取见解，例如内存缓冲区限制和应用于日志的标签。

```
[INPUT]
    Name tail
    Buffer_Max_Size 66KB
    DB /var/log/flb_kube.db
    Mem_Buf_Limit 45MB
    Path /var/log/containers/*.log
    Read_From_Head On
    Refresh_Interval 10
    Rotate_Wait 30
    Skip_Long_Lines On
    Tag kube.*
```

在创建 ConfigMap 时，请考虑 Fargate 用于验证字段的以下规则：

- [FILTER]、[OUTPUT] 和 [PARSER] 应该在每个相应的键下指定。例如，[FILTER] 必须在 filters.conf 下。在 filters.conf 下可以有一个或多个 [FILTER]。[OUTPUT] 和 [PARSER] 部分也应在其相应的键下。通过指定多个 [OUTPUT] 部分，您可以同时将日志路由到不同的目标。
- Fargate 会验证每个部分所需的键。Name 和 match 是每个 [FILTER] 和 [OUTPUT] 所必需的。Name 和 format 是每个 [PARSER] 所必需的。键不区分大小写。
- 在 ConfigMap 中不允许使用环境变量（例如 ${ENV_VAR}）。
- 对于每个 filters.conf、output.conf 和 parsers.conf 中的指令或键值对，缩进必须是相同的。键值对的缩进必须多于指令。
- Fargate 根据以下受支持的筛选条件进行验证：grep、parser、record_modifier、rewrite_tag、throttle、nest、modify 和 kubernetes。
- Fargate 根据以下受支持的输出进行验证：es、firehose、kinesis_firehose、cloudwatch、cloudwatch_logs 和 kinesis。
- ConfigMap 中必须至少提供一个受支持的 Output 插件才能启用日志记录。不需要 Filter 和 Parser 即可启用日志记录。

您还可以使用所需的配置在 Amazon EC2 上运行 Fluent Bit，以对验证过程中出现的任何问题进行故障排除。使用以下示例之一创建您的 ConfigMap。

> **Important**
>
> Amazon EKS Fargate 日志记录不支持 ConfigMap 的动态配置。对 ConfigMap 所做的任何更改都只会应用于新容器组（pod）。不会将更改应用于现有容器组（pod）。

使用所需日志目标的示例创建 ConfigMap。

> **Note**
>
> 您也可以将 Amazon Kinesis Data Streams 用作您的日志目的地。如果您使用 Kinesis Data Streams，则请确保容器组（pod）执行角色已被授予 kinesis:PutRecords 权

限。有关更多信息，请参阅《Fluent Bit：官方手册》中的 Amazon Kinesis Data Streams 权限。

Example

CloudWatch

使用 CloudWatch 视时，您有两个输出选项：

- 用 C 语言编写的输出插件
- 用 Golang 编写的输出插件

以下示例为您展示了如何使用 `cloudwatch_logs` 插件将日志发送到 CloudWatch。

a. 将以下内容保存到名为 `aws-logging-cloudwatch-configmap.yaml` 的文件中。将 *`region-code`* 替换为您的集群所在的 Amazon 区域。`[OUTPUT]` 下的参数是必需的。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  flb_log_cw: "false"  # Set to true to ship Fluent Bit process logs to
 CloudWatch.
  filters.conf: |
    [FILTER]
        Name parser
        Match *
        Key_name log
        Parser crio
    [FILTER]
        Name kubernetes
        Match kube.*
        Merge_Log On
        Keep_Log Off
        Buffer_Size 0
        Kube_Meta_Cache_TTL 300s
  output.conf: |
    [OUTPUT]
        Name cloudwatch_logs
        Match   kube.*
        region region-code
```

```
        log_group_name my-logs
        log_stream_prefix from-fluent-bit-
        log_retention_days 60
        auto_create_group true
  parsers.conf: |
    [PARSER]
        Name crio
        Format Regex
        Regex ^(?<time>[^ ]+) (?<stream>stdout|stderr) (?<logtag>P|F) (?
<log>.*)$
        Time_Key    time
        Time_Format %Y-%m-%dT%H:%M:%S.%L%z
```

b. 将清单应用于集群。

```
kubectl apply -f aws-logging-cloudwatch-configmap.yaml
```

Amazon OpenSearch Service

如果您想要将日志发送到 Amazon OpenSearch Service，则可以使用 es 输出，这是一个用 C 语言编写的插件。以下示例介绍如何使用该插件将日志发送到 OpenSearch。

a. 将以下内容保存到名为 `aws-logging-opensearch-configmap.yaml` 的文件中。将所有 *`example value`* 替换为您自己的值。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
      Name  es
      Match *
      Host  search-example-gjxdcilagiprbglqn42jsty66y.region-
code.es.amazonaws.com
      Port  443
      Index example
      Type  example_type
      AWS_Auth On
      AWS_Region region-code
      tls   On
```

b. 将清单应用于集群。

```
kubectl apply -f aws-logging-opensearch-configmap.yaml
```

Firehose

将日志发送到 Firehose 时，您有两个输出选项：

- kinesis_firehose – 用 C 语言编写的输出插件。
- firehose – 用 Golang 语言编写的输出插件。

以下示例向您展示了如何使用 `kinesis_firehose` 插件将日志发送到 Firehose。

a. 将以下内容保存到名为 `aws-logging-firehose-configmap.yaml` 的文件中。将 *region-code* 替换为您的集群所在的 Amazon 区域。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
     Name  kinesis_firehose
     Match *
     region region-code
     delivery_stream my-stream-firehose
```

b. 将清单应用于集群。

```
kubectl apply -f aws-logging-firehose-configmap.yaml
```

3. 为 Fargate 容器组执行角色设置向目标发送日志的权限。

a. 将目标的 IAM 策略下载到您的计算机。

Example

CloudWatch

将 CloudWatch IAM policy 下载到您的计算机。您还可以在 GitHub 上查看策略。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-
logging-examples/mainline/examples/fargate/cloudwatchlogs/permissions.json
```

Amazon OpenSearch Service

将 OpenSearch IAM policy 下载到您的计算机。您还可以在 GitHub 上查看策略。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-
fluent-logging-examples/mainline/examples/fargate/amazon-elasticsearch/
permissions.json
```

确保 OpenSearch 控制面板的访问控制配置正确。OpenSearch 控制面板中的 `all_access role` 需要映射 Fargate 容器组（pod）执行角色和 IAM 角色。必须为 `security_manager` 角色执行同样的映射。您可以添加以前的映射，方法为：选择 `Menu`，然后依次选择 `Security` 和 `Roles`，再选择相应的角色。有关更多信息，请参阅如何对 CloudWatch Logs 进行故障排除，以便将其流式传输到我的 Amazon ES 域？

Firehose

将 Firehose IAM 策略下载到您的计算机。您还可以在 GitHub 上查看策略。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-
logging-examples/mainline/examples/fargate/kinesis-firehose/permissions.json
```

b. 使用下载的策略文件创建一个 IAM 策略。

```
aws iam create-policy --policy-name eks-fargate-logging-policy --policy-document
 file://permissions.json
```

c. 使用以下命令将 IAM policy 附加到为 Fargate 配置文件指定的容器组（pod）执行角色。请将 *111122223333* 替换为您的账户 ID。将 *AmazonEKSFargatePodExecutionRole* 替换为容器组（pod）执行角色（有关更多信息，请参阅the section called “第 2 步：创建 Fargate 容器组（pod）执行角色”）。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::111122223333:policy/eks-fargate-logging-policy \
  --role-name AmazonEKSFargatePodExecutionRole
```

Kubernetes 筛选器支持

Fluent Bit Kubernetes 筛选器允许您将 Kubernetes 元数据添加到日志文件中。有关筛选器的更多信息，请参阅 Fluent Bit 文档中的 Kubernetes。您可以使用 API 服务器终端节点应用筛选器。

```
filters.conf: |
    [FILTER]
        Name                kubernetes
        Match               kube.*
        Merge_Log           On
        Buffer_Size         0
        Kube_Meta_Cache_TTL 300s
```

Important

- `Kube_URL`、`Kube_CA_File`、`Kube_Token_Command` 和 `Kube_Token_File` 是服务拥有的配置参数，不能指定。Amazon EKS Fargate 填充这些值。
- `Kube_Meta_Cache_TTL` 是 Fluent Bit 与 API 服务器通信以获取最新元数据等待的时间。如果未指定 `Kube_Meta_Cache_TTL`，则 Amazon EKS Fargate 会追加原定设置值 30 分钟，以减轻 API 服务器的负载。

将 Fluent Bit 进程日志发送到您的账户

您可以选择使用下面的 `ConfigMap`，将 Fluent Bit 进程日志发送到 Amazon CloudWatch。将 Fluent Bit 进程日志发送到 CloudWatch 需要额外的日志摄取和存储成本。将 *`region-code`* 替换为您的集群所在的 Amazon 区域。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  # Configuration files: server, input, filters and output
  # ======================================================
  flb_log_cw: "true"  # Ships Fluent Bit process logs to CloudWatch.

  output.conf: |
```

```
    [OUTPUT]
        Name cloudwatch
        Match kube.*
        region region-code
        log_group_name fluent-bit-cloudwatch
        log_stream_prefix from-fluent-bit-
        auto_create_group true
```

日志位于集群所在 Amazon 区域的 CloudWatch 中。日志组名称为 *my-cluster*-fluent-bit-logs，Fluent Bit 日志流名称为 fluent-bit-*podname*-*pod-namespace* 。

> **Note**
>
> - 仅当 Fluent Bit 流程成功开始后，才会发送流程日志。如果启动 Fluent Bit 时出现故障，则会丢失流程日志。您只能将流程日志发送到 CloudWatch。
> - 要调试将流程日志传输到您的账户，可以应用以前的 ConfigMap 获取流程日志。Fluent Bit 启动失败通常是启动时 Fluent Bit 未解析或接受 ConfigMap 导致的。

停止发送 Fluent Bit 进程日志

将 Fluent Bit 进程日志发送到 CloudWatch 需要额外的日志摄取和存储成本。要排除现有 ConfigMap 设置中的进程日志，请执行以下步骤。

1. 在启用 Fargate 日志记录后，找到为 Amazon EKS 集群的 Fluent Bit 进程日志自动创建的 CloudWatch 日志组。它遵循格式 *my-cluster*-fluent-bit-logs。
2. 删除 CloudWatch 日志组中为每个容器组（pod）的进程日志创建的现有 CloudWatch 日志流。
3. 编辑 ConfigMap 并设置 flb_log_cw: "false"。
4. 重启集群中的任何现有容器组（pod）。

测试应用程序

1. 部署示例容器组（pod）。
   a. 将以下内容保存到计算机上名为 sample-app.yaml 的文件中。

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: sample-app
  namespace: same-namespace-as-your-fargate-profile
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - name: http
              containerPort: 80
```

b. 将清单应用于集群。

```
kubectl apply -f sample-app.yaml
```

2. 使用您在 ConfigMap 中配置的目标查看 NGINX 日志。

大小注意事项

我们建议您为日志路由器规划最多 50MB 的内存。如果您希望应用程序以非常高的吞吐量生成日志，那么您应该规划高达 100MB 的内存。

故障排除

要确认日志记录功能是否因某种原因（例如无效 ConfigMap）而启用还是禁用，以及无效的原因，请使用 `kubectl describe pod pod-name` 检查您的容器组（pod）事件。输出可能包含澄清是否已启用日志记录的容器组（pod）事件，例如以下示例输出。

```
[...]
Annotations:          CapacityProvisioned: 0.25vCPU 0.5GB
                      Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
[...]
Events:
```

```
  Type     Reason           Age        From
              Message
  ----     ------           ----       ----
              -------
  Warning  LoggingDisabled  <unknown>  fargate-scheduler
               Disabled logging because aws-logging configmap was not found. configmap
"aws-logging" not found
```

容器组（pod）事件是短暂的，时间长短取决于设置。您也可以使用 `kubectl describe pod` *pod-name* 查看容器组（pod）的注释。容器组（pod）注释中包含有关日志记录功能是已启用还是已禁用状态以及对应原因的信息。

选择最优的 Amazon EC2 节点实例类型

Amazon EC2 提供 Worker 节点的各种不同的实例类型。每种实例类型提供不同的计算、内存、存储和网络功能。每个实例也按照这些功能分组到实例系列。有关列表，请参阅《Amazon EC2 用户指南》中的可用实例类型。Amazon EKS 发布了多种 Amazon EC2 AMI 的变体以启用支持。要确保您选择的实例类型与 Amazon EKS 兼容，请考虑以下标准。

- 所有 Amazon EKS AMI 当前均不支持 `mac` 系列。
- Arm 和非加速 Amazon EKS AMI 不支持 `g3`、`g4`、`inf` 和 `p` 系列。
- 加速 Amazon EKS AMI 不支持 `a`、`c`、`hpc`、`m` 和 `t` 系列。
- 对于基于 Arm 的实例，Amazon Linux 2023（AL2023）仅支持使用 Graviton2 或更高版本处理器的实例类型。AL2023 不支持 `A1` 实例。

在选择 Amazon EKS 支持的实例类型时，请考虑每种类型的以下功能。

节点组中的实例数

一般来说，数量较少、规模较大的实例更好，当您有很多 Daemonset 时更是如此。每个实例都需要对 API 服务器进行 API 调用，因此您拥有的实例越多，API 服务器上的负载就越多。

操作系统

查看 Linux、Windows 和 Bottlerocket 支持的实例类型。创建 Windows 实例之前，请查看在 EKS 集群上部署 Windows 节点。

硬件架构

您需要 x86 还是 Arm？ 部署 Arm 实例之前，请查看 Amazon EKS 优化版 Arm Amazon Linux AMI。您需要基于 Nitro System（Linux 或 Windows）构建的实例或是拥有加速功能的实例吗？ 如果您需要加速功能，则只能将 Linux 与 Amazon EKS 结合使用。

最大容器组（pod）数量

由于每个容器组（pod）都分配到了自己的 IP 地址，因此实例类型支持的 IP 地址数量是决定可以在实例上运行的容器组（pod）数量的因素之一。要手动确定实例类型支持多少个容器组（pod），请参阅the section called “Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量”。

> **Note**
>
> 如果您使用的是 `v20220406` 版本或更高版本的 Amazon EKS 优化版 Amazon Linux 2 AMI，则无需升级至最新 AMI 也能使用新的实例类型。对于这些 AMI，如果需要的 `max-pods` 值未在 eni-max-pods.txt 文件中列出，AMI 会自动计算该值。默认情况下，Amazon EKS 可能不支持当前处于预览状态的实例类型。这些类型的 `max-pods` 的值仍然需要添加到我们的 AMI 中的 `eni-max-pods.txt`。

Amazon Nitro System 实例类型可选择性地支持比非 Nitro System 实例类型多得多的 IP 地址。但是，并非为实例分配的所有 IP 地址都可用于容器组（pod）。要为您的实例分配大量的 IP 地址，您必须在集群中安装 `1.9.0` 版或更高版本的 Amazon VPC CNI 附加组件并进行适当配置。有关更多信息，请参阅 the section called “增加 IP 地址”。要为实例分配最大数量的 IP 地址，您必须在集群中安装 `1.10.1` 版或更高版本的 Amazon VPC CNI 附加组件，然后使用 `IPv6` 系列部署集群。

IP 系列

您可以在将 `IPv4` 系列用于集群时使用任何受支持的实例类型，以便集群将私有 `IPv4` 地址分配到容器组（pod）和服务。但是，如果您想将 `IPv6` 系列用于集群，则您必须使用 Amazon Nitro System 实例类型或裸机实例类型。Windows 实例仅支持 `IPv4`。您的集群必须运行 `1.10.1` 版或更高版本的 Amazon VPC CNI 附加组件。有关使用 `IPv6` 的更多信息，请参阅the section called “IPv6”。

您正在运行的 Amazon VPC CNI 附加组件的版本

适用于 Kubernetes 的最新版本的 Amazon VPC CNI 插件支持这些实例类型。您可能需要更新 Amazon VPC CNI 附加组件版本来利用最新的受支持的实例类型。有关更多信息，请参阅 the section called “Amazon VPC CNI”。最新版本支持与 Amazon EKS 一起使用的最新功能。以前版本并不支持所有功能。您可以在 GitHub 上的 Changelog 中查看不同版本支持的功能。

您在其中创建节点的 Amazon 区域

并非所有实例类型在所有 Amazon 区域中都可用。

是否将安全组用于容器组（pod）

如果将安全组用于容器组（pod），则仅支持特定的实例类型。有关更多信息，请参阅 the section called “适用于容器组（pod）的安全组”。

Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量

由于每个容器组（pod）都分配到了自己的 IP 地址，因此实例类型支持的 IP 地址数量是决定可以在实例上运行的容器组（pod）数量的因素之一。Amazon EKS 提供了一个脚本，您可以下载并运行该脚本，以确定 Amazon EKS 建议在每种实例类型上运行的最大容器组数量。该脚本使用每个实例的硬件属性和配置选项来确定最大容器组数量。您可以使用这些步骤中返回的数字来启用诸如将 IP 地址分配给不同于实例子网的容器组（pod）和显著增加实例的 IP 地址数量等功能。如果您使用的是具有多种实例类型的托管节点组，请使用适用于所有实例类型的值。

1. 下载一个您可以用来计算每种实例类型的最大容器组数量的脚本。

```
curl -O https://raw.githubusercontent.com/awslabs/amazon-eks-ami/master/templates/
al2/runtime/max-pods-calculator.sh
```

2. 将脚本标记为您计算机上的可执行文件。

```
chmod +x max-pods-calculator.sh
```

3. 运行脚本，请将 *m5.large* 替换为您计划部署的实例类型，并将 *1.9.0-eksbuild.1* 替换为您的 Amazon VPC CNI 附加组件版本。要确定附加组件版本，请参阅使用 Amazon VPC CNI 将 IP 分配给容器组（pod）中的更新过程。

```
./max-pods-calculator.sh --instance-type m5.large --cni-version 1.9.0-eksbuild.1
```

示例输出如下。

```
29
```

您可以向脚本添加以下选项，以查看使用可选功能时支持的最大容器组数量。

- `--cni-custom-networking-enabled` – 如果要从不同于实例的子网分配 IP 地址，请使用此选项。有关更多信息，请参阅 the section called “自定义联网”。将此选项添加到具有相同示例值的上一个脚本中会生成 `20`。
- `--cni-prefix-delegation-enabled`：如果要为每个弹性网络接口分配数量显著增加的 IP 地址，请使用此选项。此功能需要在 Nitro System 上运行的 Amazon Linux 实例和 `1.9.0` 或更高版本的 Amazon VPC CNI 附加组件。有关更多信息，请参阅 the section called “增加 IP 地址”。将此选项添加到具有相同示例值的上一个脚本中会生成 `110`。

您也可以使用运行带有 `--help` 选项的脚本以查看所有可用的选项。

> **Note**
>
> 最大容器组（pod）计算器脚本根据 Kubernetes 可扩展性阈值和推荐设置将返回值限制为 `110`。如果您的实例类型超过 30 个 vCPU，则此限制将跳至 `250`，这是基于内部 Amazon EKS 可扩展性团队测试得出的数字。有关更多信息，请参阅博客文章 Amazon VPC CNI 插件提高每个节点的容器组限制。

EKS 自动模式注意事项

EKS 自动模式会将节点上的容器组（pod）数量限制为以下两项中较小的一个：

- 110 个容器组硬上限
- 上述最大容器组数量的计算结果。

使用预先构建的优化型映像创建节点

使用托管式节点组或自主管理型节点时，您可以通过预构建的 Amazon EKS 优化型亚马逊机器映像（AMI）或您自己的自定义 AMI 来部署节点。如果您运行的是混合节点，请参阅the section called “准备操作系统”。有关每种类型的 Amazon EKS 优化版 AMI 的信息，请参阅以下主题之一。有关如何创建您自己的自定义 AMI 的说明，请参阅 the section called “构建自定义 EKS 优化版 Linux AMI”。

使用 Amazon EKS 自动模式时，EKS 负责管理 EC2 实例，包括选择和更新 AMI。

主题

- EKS AL2 和 AL2 加速型 AMI 过渡功能指南

- 使用优化型 Amazon Linux AMI 创建节点
- 使用优化版 Bottlerocket AMI 创建节点
- 使用优化版 Ubuntu Linux AMI 创建节点
- 使用优化版 Windows AMI 创建节点

EKS AL2 和 AL2 加速型 AMI 过渡功能指南

> **Warning**
>
> Amazon EKS 已于 2025 年 11 月 26 日停止发布 EKS 优化型 Amazon Linux 2（AL2）AMI。基于 AL2023 和 Bottlerocket 的 Amazon EKS AMI 适用于所有支持的 Kubernetes 版本（包括 1.33 和更高版本）。

Amazon 将于 2025 年 11 月 26 日终止对 EKS AL2 优化型和 AL2 加速型 AMI 的支持。尽管您可以在终止支持（EOS）日期（2025 年 11 月 26 日）之后继续使用 EKS AL2 AMI，但在该日期之后，EKS 将不再为 AL2 AMI 发布任何新的 Kubernetes 版本或更新，包括次要版本、补丁和错误修复。我们建议升级到 Amazon Linux 2023（AL2023）或 Bottlerocket AMI：

- AL2023 启用了默认安全方法，包括预配置的安全策略、允许模式下的 SELinux、默认启用仅 IMDSv2 模式、优化的启动时间以及改进的程序包管理，增强了安全性和性能，非常适合需要大量自定义（例如直接操作系统级访问或大量节点更改）的基础设施。要了解更多信息，请参阅 AL2023 常见问题，或查阅我们的详细迁移指引（the section called “升级至 AL2023”）。
- Bottlerocket 采用专门构建的容器优化设计，可增强安全性、缩短启动时间和缩小攻击面，从而提高效率，非常适合节点自定义极少的容器原生方法。要了解更多信息，请参阅 Bottlerocket 常见问题，或查阅我们的详细迁移指引（the section called “Bottlerocket”）。

另外，您可以在 EOS 日期（2025 年 11 月 26 日）之前，the section called “构建自定义 EKS 优化版 Linux AMI”。此外，您可以在 Amazon Linux 2 EOS 日期（2026 年 6 月 30 日）之前，使用 Amazon Linux 2 基础实例构建自定义 AMI。

迁移和支持常见问题

如何从 AL2 迁移到 AL2023 AMI？

我们建议您制定和实施迁移计划，其中应包括全面的应用程序工作负载测试和正式记录的回滚程序，然后按照 EKS 官方文档中从 Amazon Linux 2 升级到 Amazon Linux 2023 中的分步说明进行操作。

我能否在 EKS 终止支持（EOS）日期之后用 EKS 优化型 AL2 AMI 构建自定义 AL2 AMI？

虽然我们建议迁移到官方支持和发布的 EKS 优化型 AL2023 或 Bottlerocket AMI，不过您可以在 AL2 AMI EOS 日期（2025 年 11 月 26 日）之前构建自定义 EKS AL2 优化型和 AL2 加速型 AMI。您也可以在 Amazon Linux 2 EOS 日期（2026 年 6 月 30 日）之前使用 Amazon Linux 2 基础实例构建自定义 AMI。有关构建自定义 EKS AL2 优化型和 AL2 加速型 AMI 的分步说明，请参阅 EKS 官方文档中的构建自定义 Amazon Linux AMI。

EKS Kubernetes 版本支持政策是否适用于 Amazon Linux 发行版？

否。EKS AL2 优化型和 AL2 加速型 AMI 的 EOS 日期与 EKS 对 Kubernetes 版本的标准支持和扩展支持时间表无关。即使您使用 EKS 扩展支持，也需要迁移到 AL2023 或 Bottlerocket。

从 cgroupv1 迁移到 cgroupv2 会如何影响我们的迁移？

Kubernetes 社区已将（AL2 使用的）`cgroupv1` 支持改为维护模式，这意味着不会添加任何新功能，只会提供关键的安全和重要错误修复。要在 Kubernetes 中采用 `cgroupv2`，需要确保操作系统、内核、容器运行时和 Kubernetes 组件之间兼容。这需要使用默认启用 `cgroupv2` 的 Linux 发行版，例如 AL2023、Bottlerocket、Red Hat Enterprise Linux（RHEL）9+、Ubuntu 22.04+ 或 Debian 11+。这些发行版附带的内核版本 ≥5.8，即在 Kubernetes 中支持 `cgroupv2` 的最低要求。要了解详情，请参阅 About cgroup v2。

如果我的自定义 AL2 AMI 需要使用 Neuron，该怎么办？

您无法在基于 AL2 的 AMI 上以原生方式运行完全由 Neuron 驱动的应用程序。要在 AL2 AMI 上使用 Amazon Neuron，您必须使用支持 Neuron 且具有非 AL2 Linux 发行版（例如 Ubuntu 22.04、Amazon Linux 2023 等）的容器对应用程序进行容器化，然后将这些容器部署到安装了 Neuron Driver (`aws-neuronx-dkms`) 的 AL2 AMI 上。

EKS AL2 AMI EOS 日期（2025 年 11 月 26 日）之后，我是否应该切换到原生 Amazon Linux 2 基础实例？

切换至原生 Amazon Linux 2 基础实例，会缺失官方 EKS AL2 优化版 AMI 与 AL2 加速型 AMI 所提供的专属优化、容器运行时配置及定制化内容。若必须继续使用基于 AL2 的方案，建议通过the section called “构建自定义 EKS 优化版 Linux AMI” 或 Amazon EKS AMI Build Specification 中的 EKS AMI 构建模板，来制作自定义 AMI。此举可确保与现有工作负载的兼容性，并可在 Amazon Linux 2 EOS 日期（2026 年 6 月 30 日）前，持续获取 AL2 内核更新。

在 EKS AL2 AMI EOS 日期（2025 年 11 月 26 日）之后，若通过 EKS AMI GitHub 存储库构建自定义 AL2 AMI，来自 amzn2-core 与 amzn2extra-docker 等软件源的软件包可获得何种支持？

Amazon EKS AMI Build Specification 中的 EKS AMI 构建模板，会通过 YUM 工具从 amzn2-core、amzn2extra-docker 等标准 Amazon Linux 2 软件源拉取软件包。在 EKS AL2 AMI EOS 日期（2025 年 11 月 26 日）之后，上述软件源仍会持续提供支持，直至 Amazon Linux 2 的整体 EOS 日期（2026 年 6 月 30 日）。需注意，此期间的支持范围仅限内核更新，意即需手动管理并应用其他软件包更新、安全补丁及所有非内核依赖项，以保障系统的安全性与兼容性。

在搭载 AL2023 的 Amazon EKS 上运行且使用旧版 JDK8 的 Java 应用，为何可能出现内存不足（OOM）异常与容器组（pod）重启问题？该如何解决此问题？

在 Amazon EKS AL2023 节点上运行时，依赖 `jdk8u372` 之前版本 JDK 8 的 Java 应用，可能会引发 OOM 异常与容器组（pod）重启，原因在于这些版本的 JVM 与 `cgroupv2` 不兼容。此问题的具体成因是：在 Amazon Linux 2023 的默认配置下，JVM 无法通过 `cgroupv2` 机制识别容器的内存限制。因此，JVM 会基于节点的总内存进行堆内存分配，而非基于容器组（pod）中定义的内存限制。该问题的根源是 `cgroupv2` 变更了内存限制数据的存储路径，导致旧版 Java 无法正确读取可用内存信息，误将节点级别的资源当作容器可用资源。以下是几种可行的解决办法：

- 升级 JDK 版本：升级到 `jdk8u372` 或更高版本，或升级到完全支持 `cgroupv2` 的新版 JDK，即可解决此问题。有关完全兼容 `cgroupv2` 的 Java 版本列表，请参阅关于 cgroup v2。
- 构建自定义 AMI：若必须继续使用基于 AL2 的方案，可通过the section called “构建自定义 EKS 优化版 Linux AMI” 或 Amazon EKS AMI Build Specification，构建基于 AL2 的自定义 AMI（支持至 2025 年 11 月 26 日）。例如，您可构建基于 AL2 的 v1.33 AMI（支持到 2025 年 11 月 26 日）。Amazon EKS 会在 EKS AL2 EOS 日期（2025 年 11 月 26 日）之前，持续提供基于 AL2 的 AMI。在 EOS 日期（2025 年 11 月 26 日）之后，您需自行构建 AMI。
- 启用 cgroupv1：若必须继续使用 `cgroupv1`，可在 EKS AL2023 AMI 上启用 `cgroupv1`。执行命令 `sudo grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=0"`，重启系统（例如运行 Amazon Linux 2023 的 EC2 实例或节点），即可完成启用。此操作会修改系统的启动参数（例如，在 GRUB 配置中添加内核参数 systemd.unified_cgroup_hierarchy=0，指示 systemd 使用传统的 `cgroupv1` 层级架构），从而启用 `cgroupv1`。需注意，执行该 grubby 命令后，系统内核的启动配置会被修改为启用 `cgroupv1` 且禁用 `cgroupv2`。节点上仅会采用其中一种 cgroup 版本进行活跃的资源管理。此模式与“启用 `cgroupv2` 并兼容 `cgroupv1` API 向后兼容”的模式并不相同。

> **Warning**
>
> 我们不建议继续使用 cgroupv1，而是推荐迁移到 cgroupv2。Kubernetes 社区已将 cgroupv1 的支持（即 AL2 所采用的版本）调整为维护模式。这意味着该版本将不再新增功能或推送更新，仅会针对严重安全漏洞与重大程序缺陷提供修复补丁。cgroupv1 的支持功能预计将在后续版本中被完全移除，不过官方尚未公布具体的移除日期。若在使用 cgroupv1 时遇到相关问题，Amazon 将无法提供技术支持，同时会建议升级到 cgroupv2。

兼容性与版本

AL2 AMI 支持的 Kubernetes 版本

Kubernetes 版本 1.32 是 Amazon EKS 将为其发布 AL2（Amazon Linux 2）AMI 的最后一个版本。对于不高于 1.32 的受支持 Kubernetes 版本，EKS 将在 2025 年 11 月 26 日之前继续发布 AL2 AMI（AL2_ARM_64、AL2_x86_64）和 AL2 加速型 AMI（AL2_x86_64_GPU）。在该日期之后，EKS 将停止为所有 Kubernetes 版本发布 AL2 优化型和 AL2 加速型 AMI。请注意，EKS AL2 优化型和 AL2 加速型 AMI 的 EOS 日期与 EKS 对 Kubernetes 版本的标准支持和扩展支持时间表无关。

AL2、AL2023 和 Bottlerocket AMI 支持的驱动程序和 Linux 内核版本比较

组件	EKS AL2 AMI	EKS AL2023 AMI	EKS Bottlerocket AMI
基础操作系统兼容性	RHEL7/CentOS 7	Fedora/CentOS 9	不适用
CUDA 用户模式驱动程序	12.x	12.x、13.x	12.x、13.x
NVIDIA GPU 驱动程序	R570	R580	R570、R580
Amazon Neuron 驱动程序	2.20+	2.20+	2.20+
Linux 内核	5.10	6.1、6.12	6.1、6.12

有关 NVIDIA 驱动程序及 CUDA 兼容性的更多信息，请参阅 NVIDIA 文档。

Amazon Neuron 与 AL2 AMI 的兼容性

从 Amazon Neuron 版本 2.20 起，基于 EKS AL 的 AMI 所用 Neuron 运行时 (`aws-neuronx-runtime-lib`) 不再支持 Amazon Linux 2（AL2）。Neuron 驱动程序 (`aws-neuronx-dkms`) 现在是唯一支持 Amazon Linux 2 的 Amazon Neuron 程序包。这意味着您无法在基于 AL2 的 AMI 上以原生方式运行由 Neuron 驱动的应用程序。要在 AL2023 AMI 上安装 Neuron，请参阅 Amazon Neuron 安装指南。

Kubernetes 与 AL2 AMI 的兼容性

Kubernetes 社区已将（AL2 使用的）`cgroupv1` 支持改为维护模式。这意味着不会添加任何新功能，只会提供关键的安全和重要错误修复。任何依赖于 cgroupv2 的 Kubernetes 功能（例如 MemoryQoS 和增强的资源隔离），都不支持 AL2。此外，Amazon EKS Kubernetes 1.32 版本是最后一个支持 AL2 AMI 的版本。为保持与最新 Kubernetes 版本的兼容性，我们建议迁移到 AL2023 或 Bottlerocket，因为其已默认启用 `cgroupv2`。

Linux 版本与 AL2 AMI 的兼容性

在终止支持（EOS）日期（2026 年 6 月 30 日）之前，Amazon 将为 Amazon Linux 2（AL2）提供支持。但随着 AL2 的过时，整个 Linux 社区对新应用程序和功能的支持变得越来越少。AL2 AMI 基于 Linux 内核 5.10，而 AL2023 使用 Linux 内核 6.1。与 AL2023 不同，整个 Linux 社区对 AL2 的支持十分有限。这意味着许多上游 Linux 程序包和工具需要向后移植才能兼容 AL2 使用的早期内核版本；同时由于内核版本过时，无法支持某些现代 Linux 功能和安全改进，许多开源项目已弃用或限制了对较早内核版本（如 5.10）的支持。

AL2023 中未包含已弃用的程序包

AL2023 中未包含或已更改的一些最常见程序包包括：

- Amazon Linux 2023 不再包含 Amazon Linux 2 中的某些源二进制包
- Amazon Linux 在 AL2023 中更改了支持不同版本程序包（例如 amazon-linux-extras system）的方式
- AL2023 不支持适用于 Enterprise Linux 的附加程序包（EPEL）
- AL2023 不支持 32 位应用程序

要了解更多信息，请参阅 Comparing AL2 and AL2023。

AL2、AL2023 和 Bottlerocket 的 FIPS 验证方式比较

Amazon Linux 2（AL2）、Amazon Linux 2023（AL2023）和 Bottlerocket 提供了对《联邦信息处理标准》（FIPS）合规性的支持。

- AL2 通过了 FIPS 140-2 认证，AL2023 通过了 FIPS 140-3 认证。要在 AL2023 上启用 FIPS 模式，请在您的 Amazon EC2 实例上安装必要的程序包，然后按照 Enable FIPS Mode on AL2023 中的说明完成配置步骤即可。要了解更多信息，请参阅 AL2023 常见问题。
- Bottlerocket 提供了特别针对 FIPS 构建的专用变体，将内核和用户空间组件限制为使用已提交给 FIPS 140-3 加密模块验证计划的加密模块。

EKS AMI 驱动程序和版本更改日志

有关所有 EKS AMI 组件及其版本的完整列表，请参阅 GitHub 上的 Amazon EKS AMI Release Notes。

使用优化型 Amazon Linux AMI 创建节点

Amazon Elastic Kubernetes Service（Amazon EKS）提供了专用的亚马逊机器映像（AMI），这类映像针对 Kubernetes Worker 节点的运行需求做了优化处理。这些 EKS 优化版 Amazon Linux（AL）AMI，已预配置各类核心组件，例如 `kubelet`、AWS IAM 身份验证器及 `containerd`，以此保障与集群的无缝集成，同时提升集群的安全性。本指南详细列出了当前可用的 AMI 版本，并概述了针对加速计算架构及基于 Arm 架构的专属配置方案。

注意事项

- 您可以在 Amazon Linux 安全中心通过选择所需版本的选项卡来跟踪 Amazon Linux 的安全和隐私事件。您也可以订阅相应的 RSS 源。安全和隐私事件包括问题的概述、受影响的程序包以及如何更新实例以解决问题。
- 在部署加速型 AMI 或 Arm AMI 之前，请先查看 Amazon EKS 优化版加速型 Amazon Linux AMI 和 the section called “Amazon EKS 优化版 Arm Amazon Linux AMI” 中的相关信息。
- Amazon EKS 不支持 Amazon EC2 P2 实例，因为它们需要 `NVIDIA` 驱动程序版本 470 或更早版本。
- 在版本 `1.30` 或更高版本中新创建的任何托管节点组都将自动默认使用 AL2023 作为节点操作系统。

Amazon EKS 优化版加速型 Amazon Linux AMI

Amazon EKS 优化版加速型 Amazon Linux（AL）AMI 基于标准的 Amazon EKS 优化版 Amazon Linux AMI 构建。并配置作为 Amazon EKS 节点的可选映像，用来支持基于 GPU、Inferentia 和 Trainium 的工作负载。

有关更多信息，请参阅 the section called “使用 EKS Linux GPU AMI”。

Amazon EKS 优化版 Arm Amazon Linux AMI

Arm 实例可以为横向扩展和基于 Arm 的应用程序（例如 Web 服务器、容器化微服务、缓存队列以及分布式数据存储）节省大量成本。当将 Arm 节点添加到集群时，请查看以下注意事项。

- 如果您的集群是在 2020 年 8 月 17 日之前部署的，则必须对关键集群附加组件清单进行一次性升级。这样，Kubernetes 可以为集群中使用的每个硬件架构提取正确映像。有关更新集群附加组件的更多信息，请参阅 the section called “第 1 步：准备升级”。如果您在 2020 年 8 月 17 日当天或之后部署了集群，则 CoreDNS、`kube-proxy` 以及适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件已经具备多架构功能。
- 部署到 Arm 节点的应用程序必须针对 Arm 进行编译。
- 如果您在现有集群中部署了 DaemonSet，或者希望将之部署到同时要在其中部署 Arm 节点的新集群中，请验证 DaemonSet 是否可以在集群中的所有硬件架构上运行。
- 您可以在同一集群中运行 Arm 节点组和 x86 节点组。如果要这样做，请考虑将多架构容器映像部署到容器存储库（如 Amazon Elastic Container Registry），然后将节点选择器添加到清单中，以便 Kubernetes 知道要将容器组（pod）部署到哪个硬件架构。有关更多信息，请参阅 Amazon ECR 用户指南中的推送多架构映像和 Amazon ECR 的多架构容器映像简介博客文章。

更多信息

有关使用 Amazon EKS 优化版 Amazon Linux AMI 的更多信息，请参阅以下部分：

- 要将 Amazon Linux 与托管节点组一起使用，请参阅 the section called “托管节点组”。
- 要启动自行管理的 Amazon Linux 节点，请参阅 the section called “获取最新 ID”。
- 有关版本信息，请参阅 the section called “获取版本信息”。
- 要获取 Amazon EKS 优化版 Amazon Linux AMI 的最新 ID，请参阅the section called “获取最新 ID”。
- 要获取用于构建 Amazon EKS 优化版 AMI 的开源脚本，请参阅the section called “构建自定义 EKS 优化版 Linux AMI”。

从 Amazon Linux 2 升级到 Amazon Linux 2023

> **Warning**
>
> Amazon EKS 已于 2025 年 11 月 26 日停止发布 EKS 优化型 Amazon Linux 2（AL2）AMI。基于 AL2023 和 Bottlerocket 的 Amazon EKS AMI 适用于所有支持的 Kubernetes 版本（包括 1.33 和更高版本）。

AL2023 是一款基于 Linux 的操作系统，旨在为云应用程序提供安全、稳定和高性能的环境。它是 Amazon Web Services 推出的下一代 Amazon Linux，适用于所有支持的 Amazon EKS 版本。

AL2023 比 AL2 提供了多项改进。有关完整比较，请参阅《Amazon Linux 2023 用户指南》中的比较 AL2 和 Amazon Linux 2023。已在 AL2 中添加、升级和移除了多个程序包。强烈建议在升级之前使用 AL2023 测试您的应用程序。有关 AL2023 中所有程序包更改的列表，请参阅《Amazon Linux 2023 发行说明》中的 Amazon Linux 2023 的程序包更改。

除了这些更改之外，您还应了解以下事项：

- AL2023 引入了使用 YAML 配置架构的新节点初始化流程 `nodeadm`。如果您使用的是自行管理的节点组或带有启动模板的 AMI，则在创建新节点组时，现在需要明确提供其它集群元数据。以下是最低必需参数的示例，其中 `apiServerEndpoint`、`certificateAuthority` 和服务 `cidr` 是必需的：

  ```
  ---
  apiVersion: node.eks.aws/v1alpha1
  kind: NodeConfig
  spec:
    cluster:
      name: my-cluster
      apiServerEndpoint: https://example.com
      certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
      cidr: 10.100.0.0/16
  ```

  在 AL2 中，来自这些参数的元数据是从 Amazon EKS `DescribeCluster` API 调用中发现的。使用 AL2023，这种行为发生了变化，因为在大型节点纵向扩展期间，额外的 API 调用有节流风险。如果您使用的是没有启动模板的托管节点组或者 Karpenter，则此更改不会对您产生影响。有关 `certificateAuthority` 和服务 `cidr` 的更多信息，请参阅《Amazon EKS API 参考》中的 `DescribeCluster`。

- 对于 AL2023，nodeadm 也可以使用 NodeConfigSpec 来更改格式，将参数应用于每个节点的 kubelet。而在 AL2 中，这通过 --kubelet-extra-args 参数完成。这通常用于向节点添加标签和污点。以下示例显示了如何将 maxPods 和 --node-labels 应用于节点。

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: test-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
  kubelet:
    config:
      maxPods: 110
    flags:
      - --node-labels=karpenter.sh/capacity-type=on-demand,karpenter.sh/nodepool=test
```

- AL2023 需要 Amazon VPC CNI 版本 1.16.2 或更高版本。
- 默认情况下，AL2023 需要 IMDSv2。IMDSv2 有多项益处，可助于改善安全状况。它使用面向会话的身份验证方法，需要在简单的 HTTP PUT 请求中创建密钥令牌才能启动会话。会话令牌的有效时间可以介于 1 秒到 6 小时之间。有关如何从 IMDSv1 转换到 IMDSv2 的更多信息，请参阅转换到使用实例元数据服务版本 2 和获取 IMDSv2 的全部优势并在 Amazon 基础设施中禁用 IMDSv1。如果您想使用 IMDSv1，则您仍然可以通过使用实例元数据选项启动属性手动覆盖设置来做到这一点。

> **Note**
>
> 对于使用 AL2023 的 IMDSv2，托管节点组的默认跳数可能会有所不同：
>
> - 不使用启动模板时，默认值设置为 1。这意味着，容器无法使用 IMDS 访问节点的凭证。如果您需要容器访问节点的凭证，仍然可以通过使用自定义 Amazon EC2 启动模板来实现。
> - 在启动模板中使用自定义 AMI 时，默认 HttpPutResponseHopLimit 设置为 2。您可以手动覆盖启动模板中的 HttpPutResponseHopLimit。
>
> 或者，您可以使用 Amazon EKS 容器组身份来提供凭证，而不是 IMDSv2。

- AL2023 采用下一代统一控制组层次结构 (cgroupv2)。cgroupv2 用于实施容器运行时，并由 systemd 实施。虽然 AL2023 仍然包含可以使用 cgroupv1 使系统运行的代码，但这不是建议或支持的配置。此配置将在未来的 Amazon Linux 主要版本中完全移除。

- `eksctl` 需要 `eksctl` 版本 `0.176.0` 或更高版本才能支持 AL2023。

对于以前存在的托管节点组，您可以执行就地升级或蓝/绿升级，具体取决于您所使用的启动模板的方式：

- 如果您在托管节点组中使用自定义 AMI，则可以通过交换启动模板中的 AMI ID 来执行就地升级。在执行此升级策略之前，您应确保您的应用程序和所有用户数据首先传输到 AL2023。
- 如果您将托管节点组与标准启动模板或未指定 AMI ID 的自定义启动模板结合使用，则需要使用蓝/绿策略升级。蓝/绿升级通常更复杂，需要创建一个全新的节点组，在其中指定 AL2023 作为 AMI 类型。然后，需要小心配置新的节点组，以确保来自 AL2 节点组的所有自定义数据都与新操作系统兼容。在应用程序中对新节点组进行测试和验证后，容器组（pod）可以从旧节点组迁移到新的节点组。迁移完成之后，您就可以删除旧节点组。

如果您使用的是 Karpenter 且希望使用 AL2023，则需使用 AL2023 修改 `EC2NodeClass` `amiFamily` 字段。默认情况下，偏差在 Karpenter 中启用。这意味着，`amiFamily` 字段更改后，Karpenter 会自动将 Worker 节点更新为最新的 AMI（如果有）。

检索 Amazon Linux AMI 版本信息

Amazon EKS 优化版 Amazon Linux AMI 按 Kubernetes 版本和 AMI 的发布日期进行版本控制，格式如下：

```
k8s_major_version.k8s_minor_version.k8s_patch_version-release_date
```

每个 AMI 发行版都包含若干 kubelet、Linux 内核及 containerd 版本。加速型 AMI 还包括 NVIDIA 驱动程序的各种版本。您可以在 GitHub 上的 Changelog 中找到该版本信息。

检索建议的 Amazon Linux AMI ID

部署节点时，您可以为预先构建的 Amazon EKS 优化版亚马逊机器映像（AMI）指定 ID。要检索符合所需配置的 AMI ID，请查询 Amazon Systems Manager Parameter Store API。使用此 API，您无需手动查找 Amazon EKS 优化版 AMI ID。有关更多信息，请参阅 GetParameter。您使用的 IAM 主体必须具有 `ssm:GetParameter` IAM 权限才能检索 Amazon EKS 优化版 AMI 元数据。

您可以使用以下命令检索最新推荐 Amazon EKS 优化版 AMI 的映像 ID，该命令使用子参数 `image_id`。根据需要对该命令进行以下修改，然后运行修改后的命令：

- 将 `<kubernetes-version>` 替换为 Amazon EKS 支持的版本。

- 将 *ami-type* 替换为以下选项之一。有关 Amazon EC2 实例类型的信息，请参阅 Amazon EC2 实例类型。
  - 将 *amazon-linux-2023/x86_64/standard* 用于基于 Amazon Linux 2023（AL2023）x86 的实例。
  - 将 *amazon-linux-2023/arm64/standard* 用于 AL2023 ARM 实例，例如基于 Amazon Graviton 的实例。
  - 将 *amazon-linux-2023/x86_64/nvidia* 用于最新批准的基于 AL2023 NVIDIA x86 的实例。
  - 将 *amazon-linux-2023/arm64/nvidia* 用于最新批准的基于 AL2023 NVIDIA arm64 的实例。
  - 将 *amazon-linux-2023/x86_64/neuron* 用于最新的 AL2023 Amazon Neuron 实例。
- 请将 <region-code> 替换为您需要 AMI ID 的 Amazon EKS 支持的 Amazon 区域。

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/<kubernetes-version>/<ami-type>/recommended/image_id \
    --region <region-code> --query "Parameter.Value" --output text
```

以下是替换占位符后的命令示例。

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.31/amazon-linux-2023/x86_64/standard/recommended/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

示例输出如下。

```
ami-1234567890abcdef0
```

构建自定义 EKS 优化版 Linux AMI

> **Warning**
>
> Amazon EKS 已于 2025 年 11 月 26 日停止发布 EKS 优化型 Amazon Linux 2（AL2）AMI。基于 AL2023 和 Bottlerocket 的 Amazon EKS AMI 适用于所有支持的 Kubernetes 版本（包括 1.33 和更高版本）。

Amazon EKS 在 Amazon EKS AMI Build Specification 存储库中提供开源构建脚本，您可通过这些脚本查看 kubelet、运行时环境及适用于 Kubernetes 的 Amazon IAM 身份验证器的相关配置，同时也能从头构建基于 AL 的自有 AMI。

此存储库包含适用于 AL2 的专用引导脚本，以及在开机阶段运行的适用于 AL2023 的 nodeadm 工具。这些脚本可配置实例的证书数据、控制面板端点、集群名称等内容。这些脚本是 Amazon EKS 优化版 AMI 生成的可信来源，因此您可关注 GitHub 存储库以监控对 AMI 所做的更改。

以 EKS 优化版 AMI 为基础构建自定义 AMI 时，不建议且不提供技术支持运行操作系统升级（即 dnf upgrade），亦不建议升级 EKS 优化版 AMI 中预装的任何 Kubernetes 或 GPU 相关软件包，因为此类操作可能破坏组件间的兼容性。若仍选择对 EKS 优化版 AMI 中的操作系统或软件包执行升级操作，建议先在开发或预发布环境中开展全面测试，再部署至生产环境。

针对 GPU 实例构建自定义 AMI 时，建议为计划运行的各代实例类型及实例系列分别构建独立的自定义 AMI。EKS 优化版加速型 AMI 会在运行时阶段，根据底层实例的代次与系列选择性安装驱动程序及软件包。有关安装及运行时的详细信息，请参阅 EKS AMI 脚本。

先决条件

- 安装 Amazon CLI。
- 安装 HashiCorp Packer v1.9.4+
- 安装 GNU Make

快速入门

本快速入门指南将演示在 Amazon 账户中创建自定义 AMI 所需执行的命令。要了解有关可用于自定义 AMI 的配置的更多信息，请参阅 Amazon Linux 2023 页面上的模板变量。

先决条件

安装所需的 Amazon Q 插件。例如：

```
packer plugins install github.com/hashicorp/amazon
```

步骤 1：设置您的环境

克隆或复刻 Amazon EKS AMI 官方存储库。例如：

```
git clone https://github.com/awslabs/amazon-eks-ami.git
cd amazon-eks-ami
```

确认已安装 Packer：

```
packer --version
```

步骤 2：创建自定义 AMI

以下是用于各种自定义 AMI 的示例命令。

基本 NVIDIA AL2 AMI：

```
make k8s=1.31 os_distro=al2 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=560 \
  enable_efa=true
```

基本 NVIDIA AL2023 AMI：

```
make k8s=1.31 os_distro=al2023 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=560 \
  enable_efa=true
```

符合 STIG 标准的 Neuron AL2023 AMI：

```
make k8s=1.31 os_distro=al2023 \
  enable_accelerator=neuron \
  enable_fips=true \
  source_ami_id=ami-0abcd1234efgh5678 \
  kms_key_id=alias/aws-stig
```

运行这些命令后，Packer 将执行以下操作：* 启动临时 Amazon EC2 实例。* 安装 Kubernetes 组件、驱动程序和配置。在 Amazon 账户中创建 AMI。

预期输出应如下所示：

```
==> Wait completed after 8 minutes 42 seconds

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9
```

```
--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9

--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9
```

步骤 3：查看默认值

要查看默认值和其他选项，请运行以下命令：

```
make help
```

使用优化版 Bottlerocket AMI 创建节点

Bottlerocket 是由 Amazon 赞助和支持的开源 Linux 发行版。Bottlerocket 专为托管容器工作负载而构建。您可以借助 Bottlerocket 实现容器基础设施自动更新，由此提高容器化部署的可用性，降低运营成本。Bottlerocket 仅包含运行容器所需的必备软件，可提高资源利用率、减少安全威胁并降低管理开销。Bottlerocket AMI 包括 `containerd`、`kubelet` 和 Amazon IAM 身份验证器。除了托管节点组和自主管理型节点外，Bottlerocket 还受 Karpenter 支持。

优点

将 Bottlerocket 与 Amazon EKS 集群结合使用有以下优势：

- 正常运行时间更长，运营成本和管理复杂性更低 – 与其他 Linux 发行版相比，Bottlerocket 资源占用量更小，启动时间更短，更不容易受到安全威胁。Bottlerocket 占用量更小，因此可通过使用更少的存储、计算和网络资源来帮助用户降低成本。
- 自动操作系统更新可提高安全性 – Bottlerocket 的更新作为单个单元应用，必要时可以回滚。这消除了更新损坏或失败的风险，这些情况可能会使系统处于不可用状态。借助 Bottlerocket，安全更新能在可用时以中断最少的方式自动应用，并在出现故障时回滚。
- Premium Support – Amazon 在 Amazon EC2 上提供的 Bottlerocket 发行版包含在相同的 Amazon Support 计划中，这些计划还涵盖 Amazon EC2、Amazon EKS 和 Amazon ECR 等 Amazon 服务。

注意事项

将 Bottlerocket 用于 AMI 类型时，请考虑以下事项：

- Bottlerocket 支持采用 `x86_64` 和 `arm64` 处理器的 Amazon EC2 实例。

- Bottlerocket 系统支持搭载 GPU 的 Amazon EC2 实例。有关更多信息，请参阅 the section called “使用 EKS Linux GPU AMI”。
- Bottlerocket 映像不附带 SSH 服务器或 Shell。您可以使用带外访问方法来允许 SSH。这些方法会启用管理员容器，并利用用户数据来执行一些引导配置步骤。有关更多信息，请参阅 GitHub 上 Bottlerocket OS 一文中的以下章节：
  - 探索
  - 管理员容器
  - Kubernetes 设置
- Bottlerocket 使用不同的容器类型：
  - 默认情况下，将启用控制容器。该容器运行 Amazon Systems Manager 代理，可以用于在 Amazon EC2 Bottlerocket 实例上运行命令或启动 Shell 会话。有关更多信息，请参阅 Amazon Systems Manager 用户指南中的设置会话管理器。
  - 如果创建节点组时提供 SSH 密钥，则会启用管理员容器。我们建议仅将管理员容器用于开发和测试场景。我们建议不要在生产环境中使用此模式。有关更多信息，请参阅 GitHub 上的 Admin 容器。

更多信息

有关使用 Amazon EKS 优化版 Bottlerocket AMI 的更多信息，请参阅以下章节：

- 有关 Bottlerocket 的详细信息，请参阅 Bottlerocket Documentation。
- 有关版本信息资源，请参阅 the section called “获取版本信息”。
- 要将 Bottlerocket 与托管节点组一起使用，请参阅the section called “托管节点组”。
- 要启动自主管理型 Bottlerocket 节点，请参阅the section called “Bottlerocket”。
- 要检索 Amazon EKS 优化版 Bottlerocket AMI 的最新 ID，请参阅the section called “获取最新 ID”。
- 有关合规支持的详细信息，请参阅 the section called “合规支持”。

检索 Bottlerocket AMI 版本信息

每个 Bottlerocket AMI 发行版都包含若干 kubelet、Bottlerocket 内核及 containerd 版本。加速型 AMI 版本还包括若干 NVIDIA 驱动程序版本。您可以在 Bottlerocket 文档的操作系统主题中找到此版本信息。在此页面上，导航到相应的版本信息子主题。

Bottlerocket 文档中的信息有时会落后于 GitHub 上提供的版本。您可以在 GitHub 上的 releases 中找到最新版本更改的列表。

检索建议的 Bottlerocket AMI ID

部署节点时，您可以为预先构建的 Amazon EKS 优化版亚马逊机器映像（AMI）指定 ID。要检索符合所需配置的 AMI ID，请查询 Amazon Systems Manager Parameter Store API。使用此 API，您无需手动查找 Amazon EKS 优化版 AMI ID。有关更多信息，请参阅 GetParameter。您使用的 IAM 主体必须具有 `ssm:GetParameter` IAM 权限才能检索 Amazon EKS 优化版 AMI 元数据。

您可以使用以下 Amazon CLI 命令检索最新推荐 Amazon EKS 优化版 Bottlerocket AMI 的映像 ID，该命令使用子参数 `image_id`。根据需要对该命令进行以下修改，然后运行修改后的命令：

- 将 *kubernetes-version* 替换为支持的 platform-version。
- 将 *-flavor* 替换为以下选项之一。
  - 对于没有 GPU 的变体，请移除 *-flavor*。
  - 将 *-nvidia* 用于支持 GPU 的变体。
  - 对启用 FIPS 的变体使用 *-fips*。
- 将 *architecture* 替换为以下选项之一。
  - 将 *x86_64* 用于基于 x86 的实例。
  - 将 *arm64* 用于 ARM 实例。
- 将 *region-code* 替换为您需要 AMI ID 的 Amazon EKS 支持的 Amazon 区域。

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-kubernetes-version-
flavor/architecture/latest/image_id \
    --region region-code --query "Parameter.Value" --output text
```

以下是替换占位符后的命令示例。

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-1.31/x86_64/latest/
image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

示例输出如下。

```
ami-1234567890abcdef0
```

使用 Bottlerocket 满足合规性要求

Bottlerocket 符合多个组织制定的建议：

- 相关人士定义了适用于 Bottlerocket 的 CIS 基准。在默认配置中，Bottlerocket 映像具有 CIS 1 级配置配置文件所要求的大部分控件。您可以实现 CIS 2 级配置配置文件所要求的控制。有关更多信息，请参阅 Amazon 博客上的根据 CIS 基准验证 Amazon EKS 优化版 Bottlerocket AMI。
- 经优化的功能集和更小的攻击面意味着 Bottlerocket 实例只需更少的配置便可满足 PCI DSS 要求。适用于 Bottlerocket 的 CIS 基准是固化指南的理想资源，可支持您达到要求，实现符合 PCI DSS 要求 2.2 规定的安全配置标准。您还可以利用 Fluent Bit 来支持您满足要求，实现符合 PCI DSS 要求 10.2 规定的操作系统级别审核日志记录。Amazon 会定期发布新的（经修补的）Bottlerocket 实例，帮助您满足 PCI DSS 要求 6.2（适用于 v3.2.1）和要求 6.3.3（适用于 v4.0）。
- Bottlerocket 是一项符合 HIPAA 要求的功能，已获授权用于 Amazon EC2 和 Amazon EKS 的受监管工作负载。有关更多信息，请参阅符合 HIPAA 要求的服务参考。
- 预配置为使用经过 FIPS 140-3 验证的加密模块的 Bottlerocket AMI 已可供使用。这包括 Amazon Linux 2023 内核加密 API 加密模块和 Amazon-LC 加密模块。有关更多信息，请参阅 the section called “Bottlerocket FIPS AMI”。

使用 Bottlerocket FIPS AMI 让 Worker 节点进入 FIPS 就绪状态

“联邦信息处理标准（FIPS）第 140-3 号出版物”是美国和加拿大的一项政府标准，其中规定了对保护敏感信息的加密模块的安全要求。Bottlerocket 提供具有 FIPS 内核的 AMI，以便更容易地符合 FIPS 标准。

这些 AMI 已预配置为使用经过 FIPS 140-3 验证的加密模块。这包括 Amazon Linux 2023 内核加密 API 加密模块和 Amazon-LC 加密模块。

使用 Bottlerocket FIPS AMI 可使 Worker 节点进入“FIPS 就绪”状态，但不会自动使其“符合 FIPS 标准”。有关更多信息，请参阅美国联邦信息处理标准（FIPS）140-3。

注意事项

- 如果集群使用隔离子网，可能无法访问 Amazon ECR FIPS 端点。这可能会导致节点引导失败。确保网络配置允许访问必要的 FIPS 端点。有关更多信息，请参阅《Amazon PrivateLink 指南》中的通过资源 VPC 端点访问资源。
- 如果集群使用带有 PrivateLink 的子网，则映像提取将失败，因为 Amazon ECR FIPS 端点无法通过 PrivateLink 进行访问。

使用 Bottlerocket FIPS AMI 创建托管式节点组

Bottlerocket FIPS AMI 提供两种支持工作负载的变体：

- `BOTTLEROCKET_x86_64_FIPS`
- `BOTTLEROCKET_ARM_64_FIPS`

要使用 Bottlerocket FIPS AMI 创建托管式节点组，请在创建过程中选择适用的 AMI 类型。有关更多信息，请参阅 the section called “创建”。

有关选择启用 FIPS 的变体的更多信息，请参阅the section called “获取最新 ID”。

为不受支持的 Amazon 区域禁用 FIPS 端点

美国，包括 Amazon GovCloud（美国）区域，直接支持 Bottlerocket FIPS AMI。对于提供 AMI 但不直接支持的 Amazon 区域，仍可通过启动模板创建托管式节点组来使用 AMI。

Bottlerocket FIPS AMI 在引导期间依赖于 Amazon ECR FIPS 端点，而该端点通常在美国以外地区不可用。要在不支持 Amazon ECR FIPS 端点的 Amazon 区域中使用 AMI 作为其 FIPS 内核，请执行以下步骤以禁用 FIPS 端点：

1. 创建包含以下内容的新配置文件，或将内容合并到现有的配置文件中。

```
[default]
use_fips_endpoint=false
```

1. 将文件内容以 Base64 格式编码。
2. 在启动模板的 `UserData` 中，使用 TOML 格式添加以下编码字符串：

```
[settings.aws]
config = "<your-base64-encoded-string>"
```

有关其他设置，请参阅 GitHub 上的 Bottlerocket Description of settings。

以下是启动模板中 `UserData` 的示例：

```
[settings]
```

```
motd = "Hello from eksctl!"
[settings.aws]
config = "W2RlZmF1bHRdCnVzZV9maXBzX2VuZHBvaW50PWZhbHNlCg==" # Base64-encoded string.
[settings.kubernetes]
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
cluster-name = "<cluster-name>"
...<other-settings>
```

有关创建包含用户数据的启动模板的更多信息，请参阅the section called “启动模板”。

使用优化版 Ubuntu Linux AMI 创建节点

Canonical 还与 Amazon EKS 合作创建了可在您的集群中使用的节点 AMI。

Canonical 提供专用的 Kubernetes 节点操作系统映像。这是针对 Amazon EKS 优化的最小化 Ubuntu 镜像，包含与Amazon联合开发的自定义Amazon内核。有关更多信息，请参阅 Amazon Elastic Kubernetes Service（EKS）上的 Ubuntu 和the section called “Ubuntu Linux”。有关支持的更多信息，请参阅 Amazon Premium Support 常见问题解答中的第三方软件部分。

使用优化版 Windows AMI 创建节点

Windows Amazon EKS 优化版 AMI 基于 Windows Server 2019 和 Windows Server 2022 构建。它们被配置作为 Amazon EKS 节点的基本映像。默认情况下，AMI 包括以下组件：

- kubelet
- kube-proxy
- 适用于 Kubernetes 的 Amazon IAM 身份验证器
- csi-proxy
- containerd

> **Note**
>
> 您可以使用 Microsoft 安全更新指南跟踪 Windows Server 的安全或隐私事件。

Amazon EKS 提供了已针对 Windows 容器进行了优化的 AMI，包括以下变体：

- Amazon EKS 优化版 Windows Server 2019 Core AMI
- Amazon EKS 优化版 Windows Server 2019 Full AMI
- Amazon EKS 优化版 Windows Server 2022 Core AMI
- Amazon EKS 优化版 Windows Server 2022 Full AMI

Important

- Amazon EKS 优化版 Windows Server 20H2 Core AMI 已弃用。不会发布此 AMI 的任何新版本。
- 为了确保您在默认情况下安装了最新的安全更新，Amazon EKS 将维持最近 4 个月的优化版 Windows AMI。自首次发布之日起，每个新 AMI 的可用期为 4 个月。在此期限之后，较旧的 AMI 将变为私有且不能再访问。我们鼓励使用最新的 AMI，以避免出现安全漏洞，避免无法访问已达到其支持生命周期尽头的旧 AMI。虽然我们不能保证可以提供对已设为私有的 AMI 的访问权限，但您可以通过向 Amazon Support 提交服务单来请求访问权限。

发布日历

下表列出了 Amazon EKS 上的 Windows 版本的发布日期和支持终止日期。如果终止日期为空，则是因为相应版本仍受支持。

Windows 版本	Amazon EKS 版本	Amazon EKS 支持终止
Windows Server 2022 Core	2022/10/17	
Windows Server 2022 Full	2022/10/17	
Windows Server 20H2 Core	2021/8/12	2022/8/9
Windows Server 2004 Core	2020/8/19	2021/12/14
Windows Server 2019 Core	2019/10/7	
Windows Server 2019 Full	2019/10/7	
Windows Server 1909 Core	2019/10/7	2020/12/8

引导脚本配置参数

创建 Windows 节点时，节点上有一个允许配置不同参数的脚本。根据您的设置，可以在节点上类似于以下位置：C:\Program Files\Amazon\EKS\Start-EKSBootstrap.ps1 找到此脚本。您可以通过将自定义参数值指定为引导脚本的参数。例如，您可以更新启动模板中的用户数据。有关更多信息，请参阅 the section called “Amazon EC2 用户数据”。

此脚本包含以下命令行参数：

- -EKSClusterName：指定此 Worker 节点要加入的 Amazon EKS 集群名称。
- -KubeletExtraArgs：为 kubelet 指定额外的参数（可选）。
- -KubeProxyExtraArgs：为 kube-proxy 指定额外的参数（可选）。
- -APIServerEndpoint：指定 Amazon EKS 集群 API 服务器端点（可选）。仅在与 -Base64ClusterCA 一起使用时才有效。绕过调用 Get-EKSCluster。
- -Base64ClusterCA：指定 base64 编码的集群 CA 内容（可选）。仅在与 -APIServerEndpoint 一起使用时才有效。绕过调用 Get-EKSCluster。
- -DNSClusterIP：覆盖用于集群内 DNS 查询的 IP 地址（可选）。基于主接口的 IP 地址，默认值为 10.100.0.10 或 172.20.0.10。
- -ServiceCIDR – 覆盖从中寻址集群服务的 Kubernetes 服务 IP 地址范围。基于主接口的 IP 地址，默认值为 172.20.0.0/16 或 10.100.0.0/16。
- -ExcludedSnatCIDRs – 要从源网络地址转换（SNAT）中排除的 IPv4 CIDR 列表。这意味着，VPC 可寻址的容器组（pod）私有 IP 不会转换为用于出站流量的实例 ENI 主 IPv4 地址的 IP 地址。默认情况下，系统会添加 Amazon EKS Windows 节点的 VPC 的 IPv4 CIDR。为该参数指定 CIDR 还会另外排除所指定的 CIDR。有关更多信息，请参阅 the section called “出站流量”。

除了命令行参数之外，您还可以指定一些环境变量参数。指定命令行参数时，它优先于相应的环境变量。环境变量应定义为机器（或系统）作用范围，因为引导脚本只会读取机器范围的变量。

该脚本考虑以下环境变量：

- SERVICE_IPV4_CIDR — 有关定义，请参阅 ServiceCIDR 命令行参数。
- EXCLUDED_SNAT_CIDRS — 应为以逗号分隔的字符串。有关定义，请参阅 ExcludedSnatCIDRs 命令行参数。

gMSA 身份验证支持

Amazon EKS Windows 容器组（pod）允许不同类型的组托管服务账户（gMSA）身份验证。

- Amazon EKS 支持使用 Active Directory 域身份进行身份验证。有关加入域的 gMSA 的更多信息，请参阅 Amazon 博客上的 Windows Authentication on Amazon EKS Windows pods。
- Amazon EKS 提供一个插件，可让未加入域的 Windows 节点使用移动用户标识号检索 gMSA 凭证。有关无域 gMSA 的更多信息，请参阅 Amazon 博客上的 Domainless Windows Authentication for Amazon EKS Windows pods。

缓存的容器映像

Amazon EKS Windows 优化版 AMI 缓存了某些容器映像，用于 `containerd` 运行时。使用 Amazon 托管的构建组件构建自定义 AMI 时，系统会缓存容器映像。有关更多信息，请参阅 the section called “使用 Amazon 托管的构建组件”。

以下缓存的容器映像适用于 `containerd` 运行时系统：

- `amazonaws.com/eks/pause-windows`
- `mcr.microsoft.com/windows/nanoserver`
- `mcr.microsoft.com/windows/servercore`

更多信息

有关使用 Amazon EKS 优化版 Windows AMI 的更多信息，请参阅以下章节：

- 有关在 Amazon EKS 优化加速型 Windows AMI 上运行工作负载的详细信息，请参阅the section called “设置 Windows GPU AMI”。
- 要将 Windows 与托管节点组一起使用，请参阅the section called “托管节点组”。
- 要启动自主管理型 Windows 节点，请参阅the section called “Windows”。
- 有关版本信息，请参阅 the section called “获取版本信息”。
- 要检索 Amazon EKS 优化版 Windows AMI 的最新 ID，请参阅the section called “获取最新 ID”。
- 要使用 Amazon EC2 Image Builder 创建自定义 Amazon EKS 优化版 Windows AMI，请参阅the section called “自定义构建”。
- 有关最佳实践，请参阅 EKS 最佳实践指南中的Amazon EKS 优化版 Windows AMI 管理。

使用 `eksctl` 创建自主管理型 Windows Server 2022 节点

您可以将以下 `test-windows-2022.yaml` 用作参考，创建自主管理型 Windows Server 2022 节点。将所有 *`example value`* 替换为您自己的值。

> **Note**
>
> 您必须使用 `eksctl` 版本 0.116.0 或更高版本来运行自主管理型 Windows Server 2022 节点。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: windows-2022-cluster
  region: region-code
  version: '1.33'

nodeGroups:
  - name: windows-ng
    instanceType: m5.2xlarge
    amiFamily: WindowsServer2022FullContainer
    volumeSize: 100
    minSize: 2
    maxSize: 3
  - name: linux-ng
    amiFamily: AmazonLinux2
    minSize: 2
    maxSize: 3
```

然后，可以使用以下命令创建节点组。

```
eksctl create cluster -f test-windows-2022.yaml
```

检索 Windows AMI 版本信息

本主题列出了 Amazon EKS 优化版 Windows AMI 的版本及其 kubelet、containerd 和 csi-proxy 的相应版本。

每个变体的 Amazon EKS 优化版 AMI 元数据（包括 AMI ID）均可通过编程方式检索。有关更多信息，请参阅 the section called “获取最新 ID”。

AMI 按 Kubernetes 版本和 AMI 的发布日期进行版本控制，格式如下：

```
k8s_major_version.k8s_minor_version-release_date
```

> **Note**
>
> Amazon EKS 托管节点组支持 2022 年 11 月及更高版本的 Windows AMI。

要接收此特定文档页面的所有源文件更改通知，您可以通过 RSS 阅读器订阅以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/nodes/eks-ami-versions-windows.adoc.atom
```

Amazon EKS 优化版 Windows Server 2022 Core AMI

以下各表列出了 Amazon EKS 优化版 Windows Server 2022 Core AMI 的当前版本和先前版本。

Example

Kubernetes 版本 1.34

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes 1.33 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes 1.32 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	已将 container

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
				d 升级到 1.7.27。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.0	1.7.20	1.1.3	

Kubernetes 1.31 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	已将 container

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
				d 升级到 1.7.27。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	包括 CVE-2024-9042 的补丁。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes 1.30 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	包括 CVE-2024-5321 的补丁。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	已将 containerd 升级到 1.7.14。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes 1.29 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	包括 CVE-2024-5321 的补丁。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	已将 containerd 升级到 1.7.11。已将 kubelet 升级到 1.29.3。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	已将 containerd 升级到 1.6.28。已使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	修复了以下错误：kubelet 垃圾回收进程错误删除了暂停映像。
1.29-2024.01.11	1.29.0	1.6.18	1.1.2	不包括 Windows Server 2022 Core AMI 上的独立 Windows 更新 KB5034439。知识库仅适用于具有单独 WinRE 分区的 Windows 安装，不包含在我们的任何 Amazon EKS 优化版 Windows AMI 中。

Kubernetes 1.28 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.15	1.28.15	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	包括 CVE-2024-5321 的补丁。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	已将 containerd 升级到 1.7.11。
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	已将 containerd 升级到 1.6.28。已将 kubelet 升级到 1.28.8。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	已将 containerd 升级到 1.6.25。已使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.11	1.28.5	1.6.18	1.1.2	不包括 Windows Server 2022 Core AMI 上的独立 Windows 更新 KB5034439。知识库仅适用于具有单独 WinRE 分区的 Windows 安装，不包含在我们的任何 Amazon EKS 优化版 Windows AMI 中。
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	包括 CVE-2023-5528 的补丁。
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	已将 containerd 升级到 1.6.18。添加了新的引导脚本环境变量（SERVICE_IPV4_CIDR 和 EXCLUDED_SNAT_CIDRS ）。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	修复了 kubelet 中的安全通告。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 优化版 Windows Server 2022 Full AMI

以下各表列出了 Amazon EKS 优化版 Windows Server 2022 Full AMI 的当前版本和先前版本。

Example

Kubernetes 版本 1.34

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes 1.33 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes 1.32 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	已将 containrd 升级到 1.7.27
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.01	1.32.0	1.7.20	1.1.3	

Kubernetes 1.31 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	包括 CVE-2024-9042 的补丁。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes 1.30 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	已将 contianerd 升级到 1.7.20。
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	包括 CVE-2024-5321 的补丁。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	已将 containerd 升级到 1.7.14。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes 1.29 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	包括 CVE-2024-5321 的补丁。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	已将 containerd 升级到 1.7.11。已将 kubelet 升级到 1.29.3。
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	已将 containerd 升级到 1.6.28。已使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	修复了以下错误：kubelet 垃圾回收进程错误删除了暂停映像。
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes 1.28 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.15	1.28.15	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2025.01.01	1.28.15	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	包括 CVE-2024-5321 的补丁。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	已将 containerd 升级到 1.7.11。
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	已将 containerd 升级到 1.6.28。已将 kubelet 升级到 1.28.8。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	已将 containerd 升级到 1.6.25。已使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	包括 CVE-2023-5528 的补丁。
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	已将 containerd 升级到 1.6.18。添加了新的引导脚本环境变量（SERVICE_IPV4_CIDR 和 EXCLUDED_SNAT_CIDRS ）。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	修复了 kubelet 中的安全通告。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 优化版 Windows Server 2019 Core AMI

以下各表列出了 Amazon EKS 优化版 Windows Server 2019 Core AMI 的当前版本和先前版本。

Example

Kubernetes 版本 1.34

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes 1.33 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes 1.32 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.4	1.7.20	1.1.3	

Kubernetes 1.31 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	包括 CVE-2024-9042 的补丁。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes 1.30 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.30-2025-02-15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	包括 CVE-2024-5321 的补丁。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	已将 containerd 升级到 1.7.14。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes 1.29 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	包括 CVE-2024-5321 的补丁。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	已将 containerd 升级到 1.7.11。已将 kubelet 升级到 1.29.3。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	已将 containerd 升级到 1.6.28。已使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	修复了以下错误：kubelet 垃圾回收进程错误删除了暂停映像。
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes 1.28 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.20	1.2.1	已将 containerd 升级到 1.7.27。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	包括 CVE-2024-5321 的补丁。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	已将 containerd 升级到 1.7.11。
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	已将 containerd 升级到 1.6.28。已将 kubelet 升级到 1.28.8。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	已将 containerd 升级到 1.6.25。已使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	包括 CVE-2023-5528 的补丁。
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	已将 containerd 升级到 1.6.18。添加了新的引导脚本环境变量（SERVICE_IPV4_CIDR 和 EXCLUDED_SNAT_CIDRS ）。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	修复了 kubelet 中的安全通告。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 优化版 Windows Server 2019 Full AMI

以下各表列出了 Amazon EKS 优化版 Windows Server 2019 Full AMI 的当前版本和先前版本。

Example

Kubernetes 版本 1.34

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes 1.33 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes 1.32 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.0	1.7.20	1.1.3	

Kubernetes 1.31 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	包括 CVE-2024-9042 的补丁。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes 1.30 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	包括 CVE-2024-5321 的补丁。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	已将 containerd 升级到 1.7.14。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes 1.29 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	包括 CVE-2024-5321 的补丁。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	已将 containerd 升级到 1.7.11。已将 kubelet 升级到 1.29.3。
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	已将 containerd 升级到 1.6.28。已使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	修复了以下错误：kubelet 垃圾回收进程错误删除了暂停映像。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes 1.28 版

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	将 GMSA 插件日志变更为 Windows 事件
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	已将 containerd 升级到 1.7.27。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	已将 containerd 升级到 1.7.20。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	包括 CVE-2024-9042 的补丁。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	包括 CVE-2024-5321 的补丁。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	已将 containerd 升级到 1.7.11。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	已将 containerd 升级到 1.6.28。已将 kubelet 升级到 1.28.8。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	已将 containerd 升级到 1.6.25。已使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	包括 CVE-2023-5528 的补丁。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	发行说明
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	已将 containerd 升级到 1.6.18。添加了新的引导脚本环境变量（SERVICE_IPV4_CIDR 和 EXCLUDED_SNAT_CIDRS ）。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	修复了 kubelet 中的安全通告。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

检索建议的 Microsoft Windows AMI ID

部署节点时，您可以为预先构建的 Amazon EKS 优化版亚马逊机器映像（AMI）指定 ID。要检索符合所需配置的 AMI ID，请查询 Amazon Systems Manager Parameter Store API。使用此 API，您无需手动查找 Amazon EKS 优化版 AMI ID。有关更多信息，请参阅 GetParameter。您使用的 IAM 主体必须具有 `ssm:GetParameter` IAM 权限才能检索 Amazon EKS 优化版 AMI 元数据。

您可以使用以下命令检索最新推荐 Amazon EKS 优化版 Windows AMI 的映像 ID，该命令使用子参数 `image_id`。根据需要对该命令进行以下修改，然后运行修改后的命令：

- 将 *`release`* 替换为以下选项之一。
  - 将 *`2022`* 用于 Windows Server 2022。
  - 将 *`2019`* 用于 Windows Server 2019。
- 将 *`installation-option`* 替换为以下选项之一。有关更多信息，请参阅 Windows Server 中服务器核心安装选项简介。
  - 将 *`Core`* 用于攻击面更小的最低限度安装。

  - 使用 *Full* 包含 Windows 桌面体验。
- 将 *kubernetes-version* 替换为支持的 platform-version。
- 将 *region-code* 替换为您需要 AMI ID 的 Amazon EKS 支持的 Amazon 区域。

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-release-
English-installation-option-EKS_Optimized-kubernetes-version/image_id \
    --region region-code --query "Parameter.Value" --output text
```

以下是替换占位符后的命令示例。

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2022-
English-Core-EKS_Optimized-k8s-n-2/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

示例输出如下。

```
ami-1234567890abcdef0
```

使用 Image Builder 构建自定义 Windows AMI

您可以通过以下任一方式，使用 EC2 Image Builder 创建自定义 Amazon EKS 优化版 Windows AMI：

- 使用 Amazon EKS 优化版 Windows AMI 作为基础
- 使用 Amazon 托管的构建组件

无论采用哪种方式，您必须创建自己的 Image Builder 配方。有关更多信息，请参阅《Image Builder 用户指南》中的创建映像配方的新版本。

> **Important**
>
> 以下 Amazon 托管的 eks 组件包括 CVE-2024-5321 的补丁。
>
> - 1.28.2 和更高版本
> - 1.29.2 和更高版本
> - 1.30.1 和更高版本

- 所有 Kubernetes 1.31 及更高版本

使用 Amazon EKS 优化版 Windows AMI 作为基础

推荐使用该选项构建自定义 Windows AMI。我们提供的 Amazon EKS 优化版 Windows AMI 比 Amazon 托管的构建组件更新得更频繁。

1. 开启新的 Image Builder 配方。
   a. 打开位于 https://console.aws.amazon.com/imagebuilder的 EC2 Image Builder 控制台
   b. 在左侧导航窗格中，选择映像配方。
   c. 选择创建映像配方。
2. 在配方详细信息部分中，输入名称和版本。
3. 在基础映像部分指定 Amazon EKS 优化版 Windows AMI 的 ID。
   a. 选择输入自定义 AMI ID。
   b. 检索所需 Windows 操作系统版本的 AMI ID。有关更多信息，请参阅 the section called “获取最新 ID”。
   c. 输入自定义 AMI ID。如果找不到 AMI ID，请确保 AMI ID 的 Amazon 区域与控制台右上角显示的 Amazon 区域相匹配。
4. （可选）要获取最新的安全更新，请在构建组件 - 部分中添加 `update-windows` 组件。
   a. 在按名称查找组件搜索框右侧的下拉列表中，选择 Amazon 托管。
   b. 在按名称查找组件搜索框中，输入 `update-windows`。
   c. 选中 **`update-windows`** 搜索结果的复选框。此组件包括操作系统的最新 Windows 补丁。
5. 使用所需配置完成剩余的映像配方输入。有关更多信息，请参阅《Image Builder 用户指南》中的创建新的映像配方版本（控制台）。
6. 选择创建配方。
7. 在新的或现有的映像管道中使用新的映像配方。映像管道成功运行后，您的自定义 AMI 将作为输出映像列出并随时可用。有关更多信息，请参阅使用 EC2 Image Builder 控制台向导创建映像管道。

使用 Amazon 托管的构建组件

当使用 Amazon EKS 优化版 Windows AMI 作为基本操作系统不可行时，您可以改用 Amazon 托管的构建组件。此选项可能落后于支持的最新 Kubernetes 版本。

1. 开启新的 Image Builder 配方。

a. 打开位于 https://console.aws.amazon.com/imagebuilder的 EC2 Image Builder 控制台

b. 在左侧导航窗格中，选择映像配方。

c. 选择创建映像配方。

2. 在配方详细信息部分中，输入名称和版本。

3. 在基础映像部分中确定您将使用哪个选项来创建自定义 AMI：

- 选择托管映像 – 为您的映像操作系统（OS）选择 Windows。然后，为映像来源选择以下选项之一。
  - 快速入门（Amazon 托管）– 在映像名称下拉列表中，选择一个 Amazon EKS 支持的 Windows Server 版本。有关更多信息，请参阅 the section called “Windows”。
  - 我拥有的映像 – 对于映像名称，请选择具有自带许可证的自有映像的 ARN。您提供的镜像尚未安装 Amazon EKS 组件。
- 输入自定义 AMI ID – 对于 AMI ID，请输入具有自带许可证的 AMI 的 ID。您提供的镜像尚未安装 Amazon EKS 组件。

4. 在构建组件 – Windows 部分中，执行以下操作：

a. 在按名称查找组件搜索框右侧的下拉列表中，选择 Amazon 托管。

b. 在按名称查找组件搜索框中，输入 `eks`。

c. 选中**`eks-optimized-ami-windows`** 搜索结果的复选框，即使返回的结果可能不是您想要的版本。

d. 在按名称查找组件搜索框中，输入 `update-windows`。

e. 选中 update-windows 搜索结果的复选框。此组件包括操作系统的最新 Windows 补丁。

5. 在选定的组件部分中，执行以下操作：

a. 选择 **`eks-optimized-ami-windows`** 的版本控制选项。

b. 选择指定组件版本。

c. 在组件版本字段中，输入 *`version.x`*，将 *`version`* 替换为支持的 Kubernetes 版本。在版本号部分输入 *`x`* 表示使用与您明确定义的版本相应部分也相符的最新组件版本。注意控制台输出，因为它会就您所需版本是否作为托管组件提供给出建议。请记住，最新 Kubernetes 版本可能不适用于构建组件。有关可用版本的更多信息，请参阅 the section called “检索有关 `eks-optimized-ami-windows` 组件版本的信息”。

6. 使用所需配置完成剩余的映像配方输入。有关更多信息，请参阅《Image Builder 用户指南》中的创建新的映像配方版本（控制台）。

7. 选择创建配方。

8. 在新的或现有的映像管道中使用新的映像配方。映像管道成功运行后，您的自定义 AMI 将作为输出映像列出并随时可用。有关更多信息，请参阅使用 EC2 Image Builder 控制台向导创建映像管道。

检索有关 **eks-optimized-ami-windows** 组件版本的信息

您可以检索有关每个组件所安装内容的特定信息。例如，您可以验证安装了哪个 kubelet 版本。这些组件在 Amazon EKS 支持的 Windows 操作系统版本上进行功能测试。有关更多信息，请参阅 the section called “发布日历”。未列为受支持或已进入终止支持状态的任何其他 Windows 操作系统版本可能与组件不兼容。

1. 打开位于 https://console.aws.amazon.com/imagebuilder的 EC2 Image Builder 控制台
2. 在左侧导航窗格中选择组件。
3. 在按名称查找组件搜索框右侧的下拉列表中，将我拥有的更改为快速入门（Amazon 托管）。
4. 在 Find components by name（按名称查找组件）框中，输入 eks。
5. （可选）如果您使用的是最新版本，请选择两次，按降序对版本列进行排序。
6. 选择具有所需版本的 **eks-optimized-ami-windows** 链接。

结果页面中的描述显示了具体信息。

启用节点自动修复并调查节点运行状况问题

节点运行状况是指节点的运行状态以及有效运行工作负载的能力。运行正常的节点可以保持预期的连接，拥有充分的资源，并且可以在不中断的情况下成功运行容器组。要了解如何获取节点详细信息，请参阅the section called “查看节点运行状况”和the section called “获取节点日志”。

为帮助保持运行正常的节点，Amazon EKS 提供了节点监控代理和节点自动修复功能。

> **Important**
>
> 节点监控代理和节点自动修复功能仅支持 Linux。这些功能不支持 Windows。

节点监控代理

节点监控代理会自动读取节点日志以检测某些运行状况问题。此代理会解析节点日志以检测故障，并显示有关 Worker 节点的各种状态信息。对于检测到的每类问题（例如存储和联网问题），都会在

Worker 节点上应用一个专门的 NodeCondition。对于检测到的运行状况问题，相关描述可以在可观测性控制面板中查看。有关更多信息，请参阅 the section called “节点运行状况问题”。

节点监控代理是所有 Amazon EKS 自动模式集群中都包含的一项功能。对于其他集群类型，您可以将监控代理作为 Amazon EKS 附加组件添加。有关更多信息，请参阅 the section called “创建附加组件”。

节点自动修复

节点自动修复是一项附加功能，可以持续监控节点的运行状况，自动对检测到的问题做出反应，并在可能的情况下更换节点。这有助于在尽可能减少手动干预的情况下提高集群的整体可用性。如果运行状况检查失败，则会自动封锁该节点，从而确保不会在该节点上调度新的容器组。

节点自动修复功能本身可以对 kubelet 的 Ready 状况以及任何手动删除的节点对象做出反应。与节点监控代理配合使用时，节点自动修复功能可以对更多原本无法检测到的状况做出反应。这些额外的状况包括 KernelReady、NetworkingReady 和 StorageReady。

这种自动节点恢复功能可自动解决间歇性节点问题，例如加入集群失败、kubelet 无响应以及加速器（设备）错误增加等。由于提高了可靠性，因此有助于减少应用程序停机时间并优化集群运行。节点自动修复无法处理报告的某些问题，例如 DiskPressure、MemoryPressure 和 PIDPressure。Amazon EKS 会等待 10 分钟后才针对 AcceleratedHardwareReady NodeConditions 执行操作，对于所有其他状况则会等待 30 分钟。

出于安全考虑，托管式节点组在这种两种情况下还将自动禁用节点修复。在这两种情况下，以前正在进行的任何修复操作都将继续执行。

- 如果通过应用程序恢复控制器（ARC）触发了集群可用区转移，则所有后续修复操作都将停止。
- 如果节点组的节点数量超过五个，并且节点组中超过 20% 的节点处于运行不正常状态，则修复操作也将停止。

您可以在创建或编辑托管式节点组时启用节点自动修复。

- 使用 Amazon EKS 控制台时，激活托管式节点组的启用节点自动修复复选框即可。有关更多信息，请参阅 the section called “创建”。
- 使用 Amazon CLI 时，请将 `--node-repair-config enabled=true` 添加到 `eks create nodegroup` 或 `eks update-nodegroup-config` 命令中。
- 有关使用启用了节点自动修复功能的托管式节点组的示例 eksctl ClusterConfig，请参阅 GitHub 上的 44-node-repair.yaml。

Amazon EKS 通过以下方式对节点自动修复行为提供更精细的控制：

- `maxUnhealthyNodeThresholdCount` 和 `maxUnhealthyNodeThresholdPercentage`
  - 这些字段允许您指定运行状况不佳节点的计数或百分比阈值，超过该阈值时，节点自动修复操作将停止。这可以更好地控制节点自动修复的“影响范围”。
  - 您可以设置绝对计数或百分比，但不能同时设置两者。
- `maxParallelNodesRepairedCount` 和 `maxParallelNodesRepairedPercentage`
  - 这些字段允许您指定可以同时或并行修复的最大节点数，以所有运行状况不佳的节点的计数或百分比表示。这让您可以更精细地控制节点更换的速度。
  - 与运行状况不佳的节点阈值一样，您可以设置绝对计数或百分比，但不能同时设置两者。
- `nodeRepairConfigOverrides`
  - 这是一个复杂的结构，允许您为特定修复操作设置精细的覆盖。这些覆盖可控制节点被视为符合修复条件之前的修复操作和修复延迟时间。
  - 此结构中的特定字段包括：
    - `nodeMonitoringCondition`：节点监控代理报告的运行状况不佳。
    - `nodeUnhealthyReason`：节点监控代理将节点识别为运行状况不佳的原因。
    - `minRepairWaitTimeMins`：在节点符合修复条件之前，修复状况和运行状况不佳原因必须持续存在的最短时间（以分钟为单位）。
    - `repairAction`：当满足上述条件时，修复系统应采取的措施。
  - 如果使用此字段，则必须指定结构中的所有字段。您还可以提供这些覆盖的列表。
  - `nodeMonitoringCondition` 和 `nodeUnhealthyReason` 是手动文本输入，您将其设置为表示要偏离系统的默认行为。
  - `minRepairWaitTimeMins` 和 `repairAction` 字段允许您指定与系统默认行为的偏差。

节点运行状况问题

下表介绍了节点监控代理可能会检测到的节点运行状况问题。问题有两种类型：

- 状况 – 需要采取补救措施（例如替换实例或重启实例）的终端问题。启用自动修复后，Amazon EKS 将执行修复操作，可能是替换或重启节点。有关更多信息，请参阅 the section called “节点状况”。
- 事件 – 暂时性问题或节点配置不佳问题。不会执行任何自动修复操作。有关更多信息，请参阅 the section called “节点事件”。

AcceleratedHardware 节点运行状况问题

监控条件是下表中严重性为“状况”的问题的 `AcceleratedHardwareReady`。

如果启用了自动修复，则列出的修复操作将在检测到问题 10 分钟后开始。有关 XID 错误的更多信息，请参阅《NVIDIA GPU 部署和管理文档》中的 Xid Errors。有关单个 XID 消息的更多信息，请参阅《NVIDIA GPU 部署和管理文档》中的 Understanding Xid Messages。

名称	严重性	说明
DCGMDiagnosticFailure	条件	DCGM 主动诊断测试套件中的测试用例失败。
DCGMError	条件	到 DCGM 主机进程的连接已断开或无法建立。
DCGMFieldError[Code]	事件	DCGM 通过字段标识符检测到 GPU 降级。
DCGMHealthCode[Code]	事件	DCGM 运行状况检查以非致命方式失败。
DCGMHealthCode[Code]	条件	DCGM 运行状况检查以致命方式失败。
NeuronDMAError	条件	DMA 引擎遇到不可恢复的错误。
NeuronHBMUncorrectableError	条件	HBM 遇到无法纠正的错误并产生了不正确的结果。
NeuronNCUncorrectableError	条件	检测到无法纠正的 Neuron Core 内存错误。
NeuronSRAMUncorrectableError	条件	片上 SRAM 遇到奇偶校验错误并产生了不正确的结果。
NvidiaDeviceCountMismatch	事件	通过 NVML 可见的 GPU 数量与文件系统中的 NVIDIA 设备计数不一致。

名称	严重性	说明
NvidiaDoubleBitError	条件	GPU 驱动程序产生了双比特错误。
NvidiaNCCLError	事件	NVIDIA Collective Communications 库中出现段错误 (`libnccl`)。
NvidiaNVLinkError	条件	GPU 驱动程序报告了 NVLink 错误。
NvidiaPCIeError	事件	触发 PCIe 重播是为了从传输错误中恢复过来。
NvidiaPageRetirement	事件	GPU 驱动程序已将内存页标记为停用。如果出现单个双比特错误或在同一地址遇到两个单比特错误，则可能会发生这种情况。
NvidiaPowerError	事件	GPU 的功率利用率超过了允许的阈值。
NvidiaThermalError	事件	GPU 的散热状态超过了允许的阈值。
NvidiaXID[Code]Error	条件	出现严重的 GPU 错误。
NvidiaXID[Code]Warning	事件	出现了非严重的 GPU 错误。

ContainerRuntime 节点运行状况问题

监控条件是下表中严重性为“状况”的问题的 `ContainerRuntimeReady`。

名称	严重性	说明
ContainerRuntimeFailed	事件	容器运行时创建容器失败，如果反复发生，则可能与任何报告的问题有关。
DeprecatedContainerdConfiguration	事件	使用已弃用映像清单版本 2、架构 1 的容器映像最近通过 `containerd` 拉取到了节点上。
KubeletFailed	事件	kubelet 处于某个失败状态。
LivenessProbeFailures	事件	检测到存活探针失败，如果反复发生，则可能指示应用程序代码问题或超时值不足。
PodStuckTerminating	条件	容器组正在或曾经过长时间停滞在正在终止状态，这可能是因 CRI 错误阻止容器组状态进展造成的。
ReadinessProbeFailures	事件	检测到就绪探针失败，如果反复发生，则可能指示应用程序代码问题或超时值不足。
[Name]RepeatedRestart	事件	systemd 单元频繁重启。
ServiceFailedToStart	事件	系统单元启动失败。

节点内核运行状况问题

监控条件是下表中严重性为“状况”的问题的 `KernelReady`。

名称	严重性	说明
AppBlocked	事件	任务被长时间禁止调度，这通常是由于在输入或输出时被屏蔽所致。
AppCrash	事件	节点上的应用程序已崩溃。
ApproachingKernelPidMax	事件	根据当前 `kernel.pid_max` 设置，进程数量已接近可用 PID 的最大数量，之后将无法启动更多进程。
ApproachingMaxOpenFiles	事件	按照当前内核设置，打开的文件数已接近可能打开的最大文件数，之后打开新文件将失败。
ConntrackExceededKernel	事件	连接跟踪超出内核的最大值且无法建立新连接可能会导致数据包丢失。
ExcessiveZombieProcesses	事件	无法完全回收的进程正在大量积累，这指示存在应用程序问题，并且可能导致达到系统进程限制。
ForkFailedOutOfPIDs	条件	由于系统进程 ID 或内存不足，fork 或 exec 调用失败，这可能是僵尸进程或物理内存耗尽造成的。
KernelBug	事件	Linux 内核本身检测到并报告了内核错误，但这有时可能是由节点的 CPU 或内存使用率过高，导致事件处理延迟造成的。

名称	严重性	说明
LargeEnvironment	事件	此进程的环境变量数大于预期，这可能是因许多将 `enableServiceLinks` 设置为 true 的服务造成的，因为这可能导致性能问题。
RapidCron	事件	cron 作业在此节点上的运行速度超过每五分钟一次，如果该作业消耗大量资源，则可能会影响性能。
SoftLockup	事件	CPU 在给定时间停滞。

节点联网运行状况问题

监控条件是下表中严重性为“状况”的问题的 `NetworkingReady`。

名称	严重性	说明
BandwidthInExceeded	事件	因入站总带宽超过实例的最大值，导致数据包已排队或被丢弃。
BandwidthOutExceeded	事件	因出站总带宽超过实例的最大值，导致数据包已排队或被丢弃。
ConntrackExceeded	事件	连接跟踪超出实例的最大值且无法建立新连接，这可能会导致数据包丢失。
IPAMDInconsistentState	事件	磁盘上 IPAMD 检查点的状态不反映容器运行时中的 IP。
IPAMDNoIPs	事件	IPAMD 的 IP 地址已用完。

名称	严重性	说明
IPAMDNotReady	条件	IPAMD 无法连接到 API 服务器。
IPAMDNotRunning	条件	未发现 Amazon VPC CNI 进程正在运行。
IPAMDRepeatedlyRestart	事件	IPAMD 服务已多次重启。
InterfaceNotRunning	条件	此接口似乎未运行或存在网络问题。
InterfaceNotUp	条件	此接口似乎未启动或存在网络问题。
KubeProxyNotReady	事件	Kube-proxy 无法观察或列出资源。
LinkLocalExceeded	事件	由于到本地代理服务的流量 PPS 超出网络接口的最大值，数据包被丢弃。
MACAddressPolicyMisconfigured	事件	systemd-networkd 链接配置的值 `MACAddressPolicy` 不正确。
MissingDefaultRoutes	事件	缺少默认路由规则。
MissingIPRoutes	事件	容器组（pod）IP 缺少路由。
MissingIPRules	事件	容器组（pod）IP 缺少规则。
MissingLoopbackInterface	条件	此实例缺少环回接口，导致服务失败，具体取决于本地连接。
NetworkSysctl	事件	此节点的网络 `sysctl` 设置可能不正确。

名称	严重性	说明
PPSExceeded	事件	因双向 PPS 超过实例的最大值，数据包已排队或被丢弃。
PortConflict	事件	如果容器组（pod）使用 hostPort，则可能写入会覆盖主机已经绑定端口的 iptables 规则，这可能会阻止 API 服务器访问 kubelet。
UnexpectedRejectRule	事件	在 iptables 中发现了可能阻止预期流量的异常 REJECT 或 DROP 规则。

节点存储运行状况问题

监控条件是下表中严重性为“状况”的问题的 StorageReady。

名称	严重性	说明
EBSInstanceIOPSExceeded	事件	已超过实例的最大 IOPS。
EBSInstanceThroughputExceeded	事件	已超过实例的最大吞吐量。
EBSVolumeIOPSExceeded	事件	已超过特定 EBS 卷的最大 IOPS。
EBSVolumeThroughputExceeded	事件	已超过特定 Amazon EBS 卷的最大吞吐量。
EtcHostsMountFailed	事件	由于在 kubelet-container 操作过程中进行了用户数据重新挂载 /var/lib/kubelet/pods ，因此挂载 kubelet 生成的 /etc/hosts 失败。

名称	严重性	说明
IODelays	事件	在进程中检测到输入或输出延迟，如果过长，则可能表明预调配的输入输出读写操作次数不足。
KubeletDiskUsageSlow	事件	kubelet 报告在尝试访问文件系统时磁盘使用率降低。这可能表示磁盘输入输出不足或存在文件系统问题。
XFSSmallAverageClusterSize	事件	XFS 平均集群大小很小，这表示可用空间碎片过多。这可能会阻止文件创建，尽管有索引节点或可用空间可用。

查看节点运行状况检查状态

本主题介绍可用于监控 Amazon EKS 集群中节点运行状况检查状态的工具和方法。这些信息涵盖有助您识别和诊断节点级别问题的节点状况、事件和检测案例。使用本文描述的命令和模式来检查节点运行状况资源、解释状态条件，以及分析节点事件以进行运行故障排除。

您可以对所有节点使用 Kubernetes 命令来获取节点的某些运行状况信息。如果您通过 Amazon EKS 自动模式或 Amazon EKS 托管式附加组件使用节点监控代理，则将可获得更多种类的节点信号来帮助进行故障排除。对于节点监控代理检测到的运行状况问题，相关描述也可以在可观测性控制面板中查看。有关更多信息，请参阅 the section called “节点运行状况”。

节点状况

节点状况表示需要采取补救措施（例如更换实例或重启实例）的终端问题。

获取所有节点的状况：

```
kubectl get nodes -o 'custom-
columns=NAME:.metadata.name,CONDITIONS:.status.conditions[*].type,STATUS:.status.conditions[*].
```

获取特定节点的详细状况

```
kubectl describe node node-name
```

正常节点的状况输出示例：

```
  - lastHeartbeatTime: "2024-11-21T19:07:40Z"
    lastTransitionTime: "2024-11-08T03:57:40Z"
    message: Monitoring for the Networking system is active
    reason: NetworkingIsReady
    status: "True"
    type: NetworkingReady
```

出现网络问题的不正常节点的状况示例：

```
  - lastHeartbeatTime: "2024-11-21T19:12:29Z"
    lastTransitionTime: "2024-11-08T17:04:17Z"
    message: IPAM-D has failed to connect to API Server which could be an issue with
      IPTable rules or any other network configuration.
    reason: IPAMDNotReady
    status: "False"
    type: NetworkingReady
```

节点事件

节点事件指示暂时性问题或配置不佳问题。

获取节点监控代理报告的所有事件

当节点监控代理可用时，您可以运行以下命令。

```
kubectl get events --field-selector=reportingComponent=eks-node-monitoring-agent
```

示例输出：

```
LAST SEEN   TYPE      REASON       OBJECT
 MESSAGE
4s          Warning   SoftLockup   node/ip-192-168-71-251.us-west-2.compute.internal
 CPU stuck for 23s
```

获取所有节点的事件

```
kubectl get events --field-selector involvedObject.kind=Node
```

获取特定节点的事件

```
kubectl get events --field-selector involvedObject.kind=Node,involvedObject.name=node-name
```

实时观察事件

```
kubectl get events -w --field-selector involvedObject.kind=Node
```

事件输出示例：

```
LAST SEEN   TYPE      REASON           OBJECT         MESSAGE
2m          Warning   MemoryPressure   Node/node-1    Node experiencing memory pressure
5m          Normal    NodeReady        Node/node-1    Node became ready
```

常用故障排除命令

```
# Get comprehensive node status
kubectl get node node-name -o yaml

# Watch node status changes
kubectl get nodes -w

# Get node metrics
kubectl top node
```

使用 kubectl 和 S3 检索托管式节点的节点日志

了解如何检索具有节点监控代理的 Amazon EKS 托管式节点的节点日志。

先决条件

确保您已满足以下条件：

- 一个具有节点监控代理的现有 Amazon EKS 集群。有关更多信息，请参阅 the section called “节点运行状况”。

- 已安装并配置好 kubectl 命令行工具，以便与您的集群通信。
- 已安装 Amazon CLI 并登录，并且拥有创建 S3 存储桶和对象的充分权限。
- 已安装 Python 3 的最新版本
- 已安装适用于 Python 3、Boto 3 的 Amazon SDK。

第 1 步：创建 S3 存储桶目标（可选）

如果您还没有用来存储日志的 S3 存储桶，请创建一个。使用以下 Amazon CLI 命令。存储桶默认为 private 访问控制列表。将 *bucket-name* 替换为您选择的唯一存储桶名称。

```
aws s3api create-bucket --bucket <bucket-name>
```

第 2 步：创建用于 HTTP Put 的预签名 S3 URL

Amazon EKS 通过对您指定的 URL 执行 HTTP PUT 操作来返回节点日志。在本教程中，我们将生成一个预签名的 S3 HTTP PUT URL。

日志将以 gzip 压缩包的形式返回，扩展名为 .tar.gz。

> **Note**
>
> 您必须使用 Amazon API 或 SDK 创建预签名的 S3 上传 URL，以便 EKS 上传日志文件。您无法使用 Amazon CLI 创建预签名的 S3 上传 URL。

1. 确定要用于存储桶日志的位置。例如，可能将 *2024-11-12/logs1.tar.gz* 作为键。
2. 将以下 Python 代码保存到 *presign-upload.py* 文件中。替换 *<bucket-name>* 和 *<key>*。键应以 .tar.gz 结尾。

```
import boto3; print(boto3.client('s3').generate_presigned_url(
   ClientMethod='put_object',
   Params={'Bucket': '[.replaceable]`<bucket-name>`', 'Key':
 '[.replaceable]`<key>`'},
   ExpiresIn=[.replaceable]`1000`
))
```

3. 运行脚本

```
python presign-upload.py
```

4. 记下 URL 输出。在下一步中将该值作为 `http-put-destination`。

有关更多信息，请参阅适用于 Python 的 Amazon Boto3 SDK 文档中的 Generate a presigned URL to upload a file。

第 3 步：创建节点诊断资源

确定需要从中收集日志的节点的名称。

创建一个 `NodeDiagnostic` 清单，将节点名称作为资源名称，并提供 HTTP PUT URL 目标位置。

```
apiVersion: eks.amazonaws.com/v1alpha1
kind: NodeDiagnostic
metadata:
    name: <node-name>
spec:
    logCapture:
        destination: http-put-destination
```

将清单应用于集群。

```
kubectl apply -f nodediagnostic.yaml
```

您可以通过描述 `NodeDiagnostic` 资源来检查收集状态：

- 如果状态为 `Success` 或 `SuccessWithErrors`，则表示任务已完成且日志已上传到提供的目标位置（`SuccessWithErrors` 表示可能缺失某些日志）
- 如果状态为“失败”，请确认上传 URL 格式正确且未过期。

```
kubectl describe nodediagnostics.eks.amazonaws.com/<node-name>
```

第 4 步：从 S3 下载日志

等待大约一分钟，然后再尝试下载日志。然后使用 S3 CLI 下载日志。

```
# Once NodeDiagnostic shows Success status, download the logs
aws s3 cp s3://<bucket-name>/key ./<path-to-node-logs>.tar.gz
```

第 5 步：清除节点诊断资源

- NodeDiagnostic 资源不会被自动删除。获取日志构件后，您需要自行清理这些文件

```
# Delete the NodeDiagnostic resource
kubectl delete nodediagnostics.eks.amazonaws.com/<node-name>
```

Amazon EKS 混合节点功能概述

借助 Amazon EKS 混合节点功能，您可以将本地和边缘基础设施用作 Amazon EKS 集群中的节点。Amazon 负责管理由 Amazon 托管的 Amazon EKS 集群的 Kubernetes 控制面板，而您负责管理在本地或边缘环境中运行的混合节点。这样就实现了跨环境的统一 Kubernetes 管理，将本地和边缘应用程序的 Kubernetes 控制面板管理工作转移到 Amazon。

Amazon EKS 混合节点功能可支持任何本地硬件或虚拟机，从而在任何需要运行应用程序的位置享受到 Amazon EKS 的效率、可扩展性和可用性。您可以将各种 Amazon EKS 功能与 Amazon EKS 混合节点功能结合使用，包括 Amazon EKS 附加组件、Amazon EKS 容器组身份、集群访问条目、集群及见解和 Kubernetes 版本延期支持。Amazon EKS 混合节点功能原生集成了多项 Amazon 服务，包括 Amazon Systems Manager、Amazon IAM Roles Anywhere、Amazon Managed Service for Prometheus 和 Amazon CloudWatch，用于集中监控、日志记录和身份管理。

使用 Amazon EKS 混合节点功能时，无预付承诺或最低费用要求，并且在将混合节点挂载到 Amazon EKS 集群时，您只需按小时为混合节点的 vCPU 资源付费。有关详细定价信息，请参阅 Amazon EKS 定价。

特征

EKS 混合节点具有以下高级功能：

- 托管式 Kubernetes 控制面板：Amazon 负责管理 EKS 集群中由 Amazon 托管的 Kubernetes 控制面板，您负责管理在本地或边缘环境中运行的混合节点。这样就实现了跨环境的统一 Kubernetes 管理，将本地和边缘应用程序的 Kubernetes 控制面板管理工作转移到 Amazon。通过将 Kubernetes

控制面板转移到 Amazon，您可以将本地容量留给应用程序，并且 Kubernetes 控制平面会根据工作负载的需要而扩展。

- 一致的 EKS 体验：EKS 混合节点功能支持大多数 EKS 功能，可在本地和云环境中提供一致的 EKS 体验，包括 EKS 附加组件、EKS 容器组身份、集群访问条目、集群见解、扩展 Kubernetes 版本支持等。有关 EKS 混合节点功能支持的 EKS 附加组件的更多信息，请参阅the section called “配置附加组件”。
- 集中式可观测性和身份管理：EKS 混合节点功能可与多项 Amazon 服务原生集成，包括 Amazon Systems Manager、Amazon IAM Roles Anywhere、Amazon Managed Service for Prometheus 和 Amazon CloudWatch 等，从而满足集中监控、日志记录和身份管理等需求。
- 通过云端满足突发需求或增加本地容量：单个 EKS 集群可用于运行混合节点以及位于 Amazon 区域、Amazon Local Zones 或 Amazon Outposts 中的节点，从而通过云端满足突发需求，也可为 EKS 集群添加本地容量。有关更多信息，请参阅混合模式集群注意事项。
- 灵活基础设施：EKS 混合节点功能遵循自带基础设施的方法，并且与您用于混合节点的基础设施无关。您可以在物理计算机或虚拟机上运行混合节点，可以使用 x86 和 ARM 架构，从而跨不同的基础设施类型迁移在混合节点上运行的本地工作负载。
- 灵活联网：使用 EKS 混合节点功能时，EKS 控制面板与混合节点之间的通信通过您在集群创建期间传递的 VPC 和子网进行路由，此方法基于 EKS 中控制面板用于节点联网的现有机制。您可以灵活选择将本地网络连接到 Amazon 中的 VPC 的首选方法。有多个有文档记录的选项可供选择，包括 Amazon Site-to-Site VPN、Amazon Direct Connect 或您自己的 VPN 解决方案，您可以根据自己的应用场景选择最适合的方法。

限制

- 每个集群最多支持将 15 个 CIDR 用于远程节点网络，以及将 15 个 CIDR 用于远程容器组网络。

注意事项

- EKS 混合节点功能可与新的或现有的 EKS 集群结合使用。
- 除 Amazon GovCloud（美国）区域和 Amazon 中国区域外，EKS 混合节点功能已在所有 Amazon 区域开放。
- 要使用 EKS 混合节点功能，必须在您的本地环境和 Amazon 之间建立可靠的连接。EKS 混合节点功能不适合连接断开、中断、间歇性或受限（DDIL）的环境。如果在 DDIL 环境中运行，可以考虑使用 Amazon EKS Anywhere。有关混合节点在网络断开连接情况下行为的信息，请参阅 Best Practices for EKS Hybrid Nodes。

- 不支持在云基础设施（包括 Amazon 区域、Amazon Local Zones、Amazon Outposts 或其他云）中运行 EKS 混合节点功能。如果您在 Amazon EC2 实例上运行混合节点，则需要支付混合节点功能使用费。
- 混合节点功能使用费从节点加入 EKS 集群时起开始计费，并在从集群中移除节点时停止计费。如果您不使用混合节点功能，请务必将其从 EKS 集群中移除。

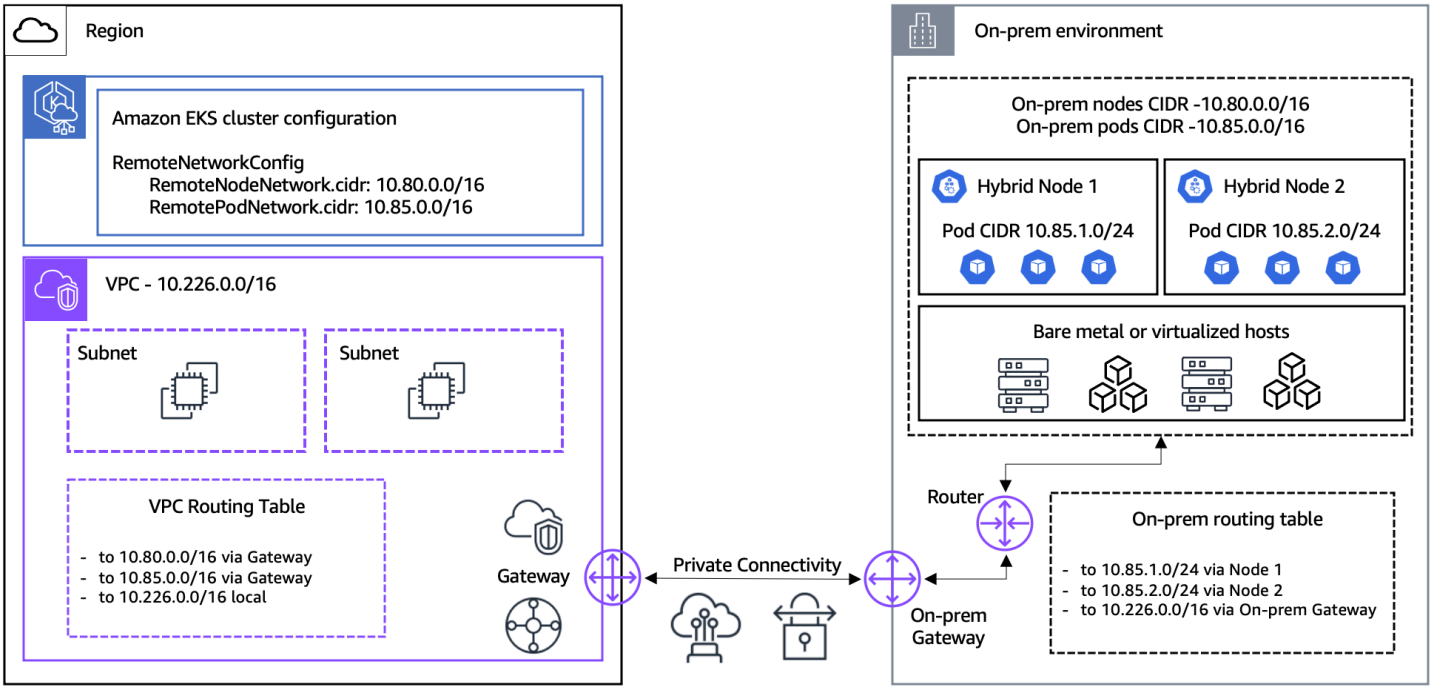
其他资源

- EKS Hybrid Nodes workshop：在演示环境中部署 EKS 混合节点的分步说明。
- Amazon re:Invent: EKS Hybrid Nodes：Amazon re:Invent 会议介绍了 EKS 混合节点发布会，一位客户展示了其如何在自己的环境中使用 EKS 混合节点。
- Amazon re:Post: Cluster networking for EKS Hybrid Nodes：文章说明了为 EKS 混合节点设置网络的各种方法。
- Amazon 博客：Run GenAI inference across environments with EKS Hybrid Nodes：博客文章展示了如何使用 EKS 混合节点跨环境运行 GenAI 推理。

混合节点的先决条件设置

要使用 Amazon EKS 混合节点功能，您必须拥有在本地环境与 Amazon、具有支持操作系统的裸机服务器或虚拟机之间往来传输的私有连接，并且配置了 Amazon IAM Roles Anywhere 或 Amazon Systems Manager（SSM）混合激活。您负责在整个混合节点生命周期中管理这些先决条件。

- 在本地环境与 Amazon 之间往来传输的混合网络连接
- 物理计算机或虚拟机形式的基础设施
- 与混合节点功能兼容的操作系统
- 已配置好本地 IAM 凭证提供者



混合网络连接

Amazon EKS 控制面板和混合节点之间的通信通过您在集群创建期间传递的 VPC 和子网进行路由，此方法以 Amazon EKS 中控制面板到节点联网的现有机制为基础。有多个有文档记录的选项可供选择，以在本地环境与 VPC 之间建立连接，包括 Amazon Site-to-Site VPN 和 Amazon Direct Connect。有关如何使用这些解决方案实现混合网络连接的更多信息，请参阅 Amazon Site-to-Site VPN 和 Amazon Direct Connect 用户指南。

为确保最佳体验，我们建议您从混合节点到 Amazon 区域的可靠网络连接至少达到 100 Mbps，并且往返延迟最大 200 毫秒。这是适用于大多数使用案例的一般指导，但并非严格要求。带宽和延迟要求可能因混合节点的数量和工作负载特征而异，例如应用程序映像大小，应用程序弹性、监控和日志记录配置，以及需要访问其他 Amazon 服务中所存储数据的应用程序依赖项。我们建议在部署到生产环境之前使用自己的应用程序和环境进行测试，从而验证您的网络设置是否满足工作负载的要求。

本地网络配置

您必须启用从 Amazon EKS 控制面板到本地环境的入站网络访问权限，以允许 Amazon EKS 控制面板与在混合节点上运行的 `kubelet` 进行通信，此外也可以选择与在混合节点上运行的 Webhook 进行通信。此外，您必须为混合节点以及在其上运行的组件启用出站网络访问权限，确保其能与 Amazon EKS 控制面板进行通信。您可以将此通信配置为完全限定于您的 Amazon Direct Connect、Amazon Site-to-Site VPN 或您自己的 VPN 连接。

用于本地节点和容器组网络的无类别域间路由（CIDR）范围必须使用 IPv4 RFC-1918 地址范围。本地路由器必须配置指向本地节点以及（可选）容器组的路由。请参阅the section called “本地联网配置”以了解有关本地网络要求的更多信息，包括防火墙和本地环境中必须启用的所需端口和协议的完整列表。

EKS 集群配置

为尽可能减少延迟，我们建议您在距离本地或边缘环境最近的 Amazon 区域创建 Amazon EKS 集群。您需要在创建 Amazon EKS 集群期间通过以下两个 API 字段来传递本地节点和容器组 CIDR：`RemoteNodeNetwork` 和 `RemotePodNetwork`。您可能需要与本地网络团队讨论，以确定本地节点和容器组 CIDR。节点 CIDR 是从本地网络分配的，如果您为 CNI 使用叠加网络，则容器组 CIDR 是从您使用的容器网络接口（CNI）分配的。默认情况下，Cilium 和 Calico 使用叠加网络。

您通过 `RemoteNodeNetwork` 和 `RemotePodNetwork` 字段配置的本地节点和容器组 CIDR 可用于配置 Amazon EKS 控制面板，以通过您的 VPC 将流量路由到 `kubelet` 以及在混合节点上运行的容器组。您的本地节点和容器组 CIDR 不能相互重叠，不能与您在创建集群时传递的 VPC CIDR 重叠，也不能与 Amazon EKS 集群的服务 IPv4 配置重叠。此外，每个 EKS 集群的容器组（pod）CIDR 必须是唯一的，这样本地路由器才能路由流量。

我们建议为 Amazon EKS Kubernetes API 服务器端点使用公有或私有端点访问权限。如果您选择“公有和私有”，Amazon EKS Kubernetes API 服务器端点将始终解析为在您的 VPC 之外运行的混合节点公有 IP，这可能会阻止您的混合节点加入集群。使用公有端点访问权限时，Kubernetes API 服务器端点将解析为公有 IP，并且从混合节点到 Amazon EKS 控制面板的通信将通过互联网路由。当您选择私有端点访问权限时，Kubernetes API 服务器端点将解析为私有 IP，并且从混合节点到 Amazon EKS 控制面板的通信将通过您的私有连接链路（大部分情况下为 Amazon Direct Connect 或 Amazon Site-to-Site VPN）进行路由。

VPC 配置

您必须在创建 Amazon EKS 集群期间配置 VPC，并在其路由表中包含本地节点的路由，您还可以配置容器组网络，将您的虚拟专用网关（VGW）或中转网关（TGW）作为目标。下面显示了一个示例。将 `REMOTE_NODE_CIDR` 和 `REMOTE_POD_CIDR` 替换为本地网络的相应值。

目标位置	目标	描述
10.226.0.0/16	本地	到 VPC 的流量将在 VPC 内部路由
REMOTE_NODE_CIDR	tgw-abcdef123456	本地节点 CIDR，将流量路由到 TGW

目标位置	目标	描述
REMODE_POD_CIDR	tgw-abcdef123456	本地容器组 CIDR，将流量路由到 TGW

安全组配置

在创建集群时，Amazon EKS 将创建一个名为 `eks-cluster-sg-<cluster-name>-<uniqueID>` 的安全组。您无法更改此集群安全组的入站规则，但可以限制出站规则。您必须向集群添加一个额外的安全组，以确保在混合节点上运行的 kubelet 和（可选）Webhook 能够与 Amazon EKS 控制面板通信。对于此额外的安全组，必需的入站规则如下所示。将 `REMOTE_NODE_CIDR` 和 `REMOTE_POD_CIDR` 替换为本地网络的相应值。

名称	安全组规则 ID	IP 版本	类型	协议	端口范围	来源
本地节点入站	sgr-abcdef123456	IPv4	HTTPS	TCP	443	REMOTE_NODE_CIDR
本地容器组入站	sgr-abcdef654321	IPv4	HTTPS	TCP	443	REMOTE_POD_CIDR

基础设施

您必须有可用作混合节点的裸机服务器或虚拟机可用。混合节点与底层基础设施无关，x86 和 ARM 架构均受支持。Amazon EKS 混合节点功能采用“自带基础设施”的方法，即您需要自行预置和管理用作混合节点的裸机服务器或虚拟机。虽然没有严格的最低资源要求，但对于混合节点，我们建议您使用至少具有 1 个 vCPU 和 1GiB RAM 的主机。

操作系统

Bottlerocket、Amazon Linux 2023（AL2023）、Ubuntu 和 RHEL 均已经过持续验证，可用作混合节点的节点操作系统。Amazon 仅在 VMware vSphere 环境中支持 Bottlerocket。在 Amazon EC2 之外运行时，AL2023 不属于 Amazon Support 计划的范围。AL2023 只能在本地虚拟化环境中使用；有关更多信息，请参阅《Amazon Linux 2023 User Guide》https://docs.amazonaws.cn/linux/al2023/ug/outside-ec2.html。Amazon 支持混合节点与 Ubuntu 和 RHEL 操作系统的集成，但不提供对操作系统本身的支持。

操作系统的预置和管理由您负责。首次测试混合节点时，最简单的方法是在一台已预置好的主机上运行 Amazon EKS 混合节点功能 CLI (nodeadm)。对于生产部署，建议在黄金操作系统映像中包含 nodeadm 并将其配置为作为 systemd 服务运行，从而在主机启动时自动将主机加入 Amazon EKS 集群。

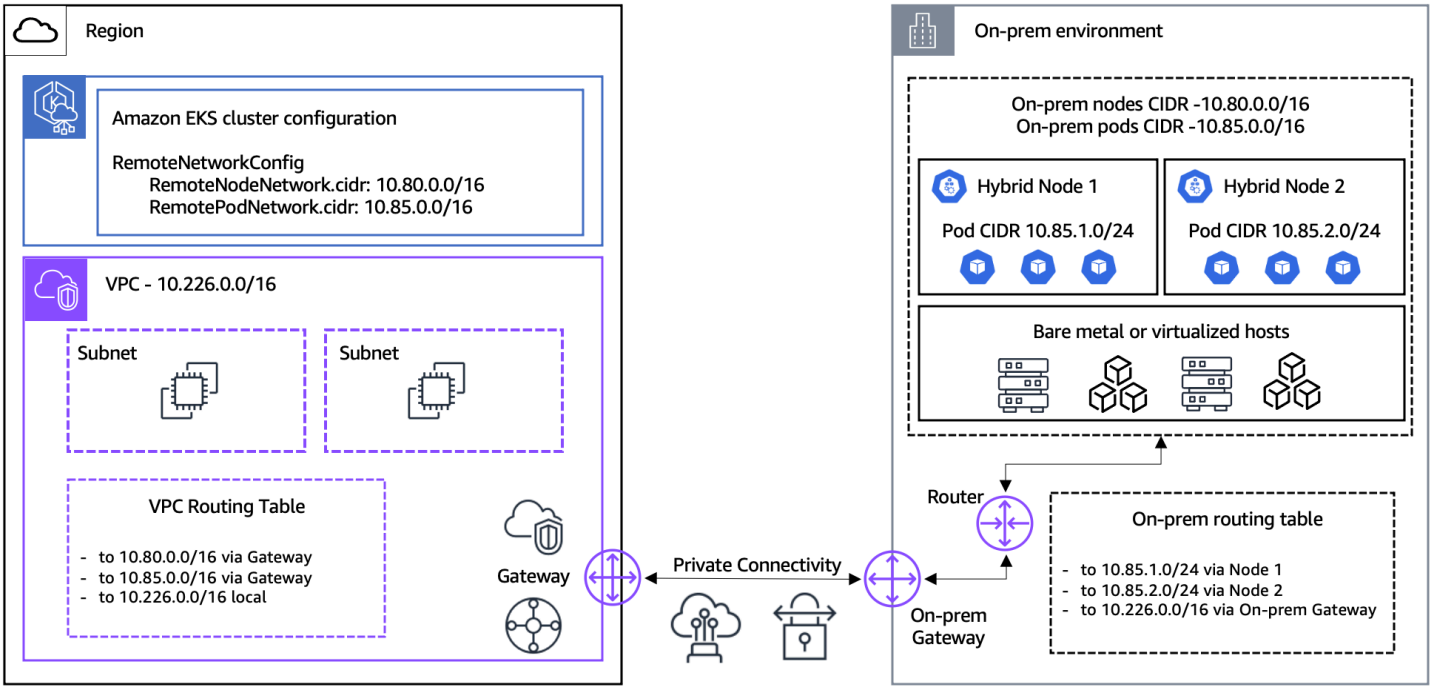
本地 IAM 凭证提供者

Amazon EKS 混合节点功能使用由 Amazon SSM 混合激活或 Amazon IAM Roles Anywhere 预置的临时 IAM 凭证，来进行 Amazon EKS 集群身份验证。您必须使用 Amazon SSM 混合激活，或将 Amazon IAM Roles Anywhere 与 Amazon EKS 混合节点功能 CLI (nodeadm) 配合使用。如果您没有具有证书颁发机构（CA）的现有公钥基础设施（PKI）和本地环境的证书，我们建议您使用 Amazon SSM 混合激活。如果您在确实具有现有的 PKI 和本地证书，请使用 Amazon IAM Roles Anywhere。

与适用于在 Amazon EC2 上运行的节点的 the section called “节点 IAM 角色” 类似，您将创建一个混合节点 IAM 角色，具有将混合节点加入到 Amazon EKS 集群所需的权限。如果您使用 Amazon IAM Roles Anywhere，请配置一个信任策略来允许 Amazon IAM Roles Anywhere 代入该混合节点 IAM 角色，此外还需要配置您的 Amazon IAM Roles Anywhere 配置文件，将该混合节点 IAM 角色作为假设角色。如果您使用 Amazon SSM，请配置一个信任策略来允许 Amazon SSM 代入混合节点 IAM 角色并使用混合节点 IAM 角色创建混合激活。有关如何创建具有必需权限的混合节点 IAM 角色的信息，请参阅the section called “准备凭证”。

准备混合节点的联网

本主题概述创建 Amazon EKS 集群和挂载混合节点之前必须配置的联网设置。本指南假设您已满足使用 Amazon Site-to-Site VPN、Amazon Direct Connect 或您自己的 VPN 解决方案进行混合网络连接的先决条件。



本地联网配置

最低网络要求

为确保最佳体验，我们建议您从混合节点到 Amazon 区域的可靠网络连接至少达到 100 Mbps，并且往返延迟最大 200 毫秒。这是适用于大多数使用案例的一般指导，但并非严格要求。带宽和延迟要求可能因混合节点的数量和工作负载特征而异，例如应用程序映像大小，应用程序弹性、监控和日志记录配置，以及需要访问其他 Amazon 服务中所存储数据的应用程序依赖项。我们建议在部署到生产环境之前使用自己的应用程序和环境进行测试，从而验证您的网络设置是否满足工作负载的要求。

本地节点和容器组 CIDR

确定您将用于混合节点以及在其上运行的工作负载的节点和容器组 CIDR。节点 CIDR 是从本地网络分配的，如果您为 CNI 使用叠加网络，则容器组 CIDR 是从您的容器网络接口（CNI）分配的。创建 EKS 集群时，您可以使用 `RemoteNodeNetwork` 和 `RemotePodNetwork` 字段将本地节点 CIDR 和容器组 CIDR 作为输入传递。本地节点 CIDR 必须可在本地网络上路由。有关本地容器组 CIDR 可路由性的信息，请参阅以下一节。

本地节点和容器组 CIDR 块必须满足以下要求：

1. 处于以下 IPv4 RFC-1918范围之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。
2. 不相互重叠，也不与 EKS 集群的 VPC CIDR 或 Kubernetes 服务 IPv4 CIDR 重叠。

本地容器组网络路由

使用 EKS 混合节点时，我们通常建议您将本地容器组 CIDR 设置为在本地网络上可路由，以便在云和本地环境之间实现完整的集群通信和功能。

可路由的容器组网络

如果您能够将容器组网络设置为在本地网络上可路由，则请按照以下指导操作。

1. 使用本地容器 CIDR 配置 EKS 集群的 `RemotePodNetwork` 字段，使用本地容器 CIDR 配置您的 VPC 路由表，以及使用本地容器 CIDR 配置您的 EKS 集群安全组。
2. 您可以通过多种方法来使本地容器组 CIDR 可在本地网络上路由，包括边界网关协议（BGP）、静态路由或其他自定义路由解决方案。我们推荐使用 BGP 解决方案，因为与需要自定义或手动路由配置的备选解决方案相比，此方案具有更好的可扩展性并且更易于管理。Amazon 支持使用 Cilium 和 Calico 的 BGP 功能来公开容器组 CIDR，有关更多信息，请参阅the section called “配置 CNI”和the section called “可路由的远程容器组（pod）CIDR”。
3. Webhook 可以在混合节点上运行，因为 EKS 控制面板能够与分配给 Webhook 的容器组 IP 地址进行通信。
4. 在云节点上运行的工作负载能够直接与同一 EKS 集群中混合节点上运行的工作负载进行通信。
5. 其他 Amazon 服务（例如 Amazon 应用程序负载均衡器和 Amazon Managed Service for Prometheus）能够与在混合节点上运行的工作负载进行通信，以平衡网络流量并抓取容器组指标。

不可路由的容器组网络

如果您无法将容器组网络设置为在本地网络上可路由，则请按照以下指导操作。

1. Webhook 无法在混合节点上运行，因为 Webhook 需要从 EKS 控制面板连接到分配给 Webhook 的容器组 IP 地址。在这种情况下，我们建议您在与混合节点位于同一 EKS 集群中的云节点上运行 Webhook；有关更多信息，请参阅the section called “配置 Webhook”。
2. 对云节点使用 VPC CNI 并对混合节点使用 Cilium 或 Calico 时，在云节点上运行的工作负载无法直接与在混合节点上运行的工作负载进行通信。
3. 使用服务流量分布将流量保持在其源区域的本地。有关服务流量分布的更多信息，请参阅the section called “配置服务流量分布”。
4. 将 CNI 配置为在离开本地主机时对容器组流量使用出口伪装或网络地址转换（NAT）。此项在 Cilium 中默认为启用状态。Calico 需要将 `natOutgoing` 设置为 `true`。
5. 其他 Amazon 服务（例如 Amazon 应用程序负载均衡器和 Amazon Managed Service for Prometheus）能够与在混合节点上运行的工作负载进行通信。

混合节点安装和升级期间必需的访问权限

在安装过程中，您必须具有以下域的访问权限，以便在主机上安装混合节点依赖项。此过程可以在生成操作系统映像时一次执行，也可以在运行时中在每台主机上完成。这包括初始安装以及升级混合节点的 Kubernetes 版本时。

某些软件包使用操作系统的默认软件包管理器进行安装。对于 AL2023 和 RHEL，使用 `yum` 命令安装 `containerd`、`ca-certificates`、`iptables` 和 `amazon-ssm-agent`。对于 Ubuntu，使用 `apt` 安装 `containerd`、`ca-certificates` 和 `iptables`，并使用 `snap` 安装 `amazon-ssm-agent`。

组件	URL	协议	端口
EKS 节点构件（S3）	https://hybrid-assets.eks.amazonaws.com	HTTPS	443
EKS 服务端点	https://eks.*region*.amazonaws.com	HTTPS	443
ECR 服务端点	https://api.ecr.*region*.amazonaws.com	HTTPS	443
EKS ECR 端点	区域性端点请参阅the section called “查看 Amazon 映像注册表”。	HTTPS	443
SSM 二进制端点 [1]	https://amazon-ssm-*region*.s3.*region*.amazonaws.com	HTTPS	443
SSM 服务端点 [1]	https://ssm.*region*.amazonaws.com	HTTPS	443
IAM Anywhere 二进制端点 [2]	https://rolesanywhere.amazonaws.com	HTTPS	443

组件	URL	协议	端口
IAM Anywhere 服务端点 [2]	https://rolesanywhere.*region*.amazonaws.com	HTTPS	443
操作系统软件包管理器端点	软件包存储库端点是特定于操作系统的端点，并且可能因地理区域而异。	HTTPS	443

Note

[1] 仅在将 Amazon SSM 混合激活用作本地 IAM 凭证提供者时，才需要 Amazon SSM 端点访问权限。
[2] 仅在将 Amazon IAM Roles Anywhere 用作本地 IAM 凭证提供者时，才需要 Amazon IAM 端点访问权限。

正在进行的集群操作需要的访问权限

正在进行的集群操作需要拥有本地防火墙的以下网络访问权限。

Important

根据您选择的 CNI，您需要为 CNI 端口配置其他网络访问规则。有关详细信息，请参阅 Cilium 文档和 Calico 文档。

类型	协议	Direction	端口	来源	目标	用法
HTTPS	TCP	出站	443	远程节点 CIDR	EKS 集群 IP [1]	Kubelet 到 Kubernetes API 服务器
HTTPS	TCP	出站	443	远程容器组 CIDR	EKS 集群 IP [1]	容器组（pod）到

类型	协议	Direction	端口	来源	目标	用法
						Kubernetes API 服务器
HTTPS	TCP	出站	443	远程节点 CIDR	SSM 服务端点	SSM 混合激活凭证刷新和 SSM 心跳信号发送每隔 5 分钟进行一次
HTTPS	TCP	出站	443	远程节点 CIDR	IAM Anywhere 服务端点	IAM Roles Anywhere 凭证刷新
HTTPS	TCP	出站	443	远程容器组 CIDR	STS 区域性端点	容器组（pod）到 STS 端点，仅对 IRSA 为必需
HTTPS	TCP	出站	443	远程节点 CIDR	Amazon EKS 身份验证服务端点	节点到 Amazon EKS 身份验证端点，仅对 Amazon EKS 容器组身份为必需
HTTPS	TCP	入站	10250	EKS 集群 IP [1]	远程节点 CIDR	Kubernetes API 服务器到 kubelet

类型	协议	Direction	端口	来源	目标	用法
HTTPS	TCP	入站	Webhook 端口	EKS 集群 IP [1]	远程容器组 CIDR	Kubernetes API 服务器到 Webhook
HTTPS	TCP、UDP	入站、出站	53	远程容器组 CIDR	远程容器组 CIDR	容器组（pod）到 CoreDNS。如果您在云端运行至少 1 个 CoreDNS 副本，则必须允许指向运行 CoreDNS 的 VPC 的 DNS 流量。
用户定义	用户定义	入站、出站	应用程序端口	远程容器组 CIDR	远程容器组 CIDR	容器组（pod）到容器组（pod）

> **Note**
>
> [1] EKS 集群的 IP。请参阅以下有关 Amazon EKS 弹性网络接口的章节。

Amazon EKS 网络接口

Amazon EKS 将网络接口附加到您在集群创建期间传递的 VPC 中的子网，从而实现 EKS 控制面板与您的 VPC 之间的通信。Amazon EKS 创建的网络接口可以在集群创建后通过 Amazon EC2 控制台查看，也可以通过 Amazon CLI 查看。对 EKS 集群应用更改（例如 Kubernetes 版本升级）时，系统会

删除原始网络接口并创建新的网络接口。您可以在集群创建期间传递子网时使用受限子网大小来限制 Amazon EKS 网络接口的 IP 范围，从而更轻松地配置本地防火墙，允许与已知受限 IP 集的入站/出站连接。要控制要在其中创建网络接口的子网，您可以在创建集群时限制指定的子网数，或者在创建集群后更新子网。

由 Amazon EKS 预置的网络接口的描述格式为 `Amazon EKS your-cluster-name` 。有关可用于查找 Amazon EKS 所预置网络接口的 IP 地址的 Amazon CLI 命令，请参阅以下示例。请将 `VPC_ID` 替换为您在创建集群时传递的 VPC 的 ID。

```
aws ec2 describe-network-interfaces \
--query 'NetworkInterfaces[?(VpcId == VPC_ID && contains(Description,Amazon
 EKS))].PrivateIpAddress'
```

Amazon VPC 和子网设置

Amazon EKS 的现有 VPC 和子网要求适用于使用混合节点的集群。此外，VPC CIDR 不能与本地节点和容器组 CIDR 重叠。必须在 VPC 路由表中为本地节点以及（可选）容器组 CIDR 配置路由。这些路由必须设置为将流量路由到用于混合网络连接的网关，这通常是虚拟专用网关（VGW）或中转网关（TGW）。如果您使用 TGW 或 VGW 来连接 VPC 和本地环境，则必须为您的 VPC 创建 TGW 或 VGW 连接。VPC 必须具有 DNS 主机名和 DNS 解析支持。

以下步骤会用到 Amazon CLI。您还可以在 Amazon Web Services 管理控制台中或使用其他接口（例如 Amazon CloudFormation、Amazon CDK 或 Terraform）创建这些资源。

第 1 步：创建 VPC

1. 运行以下命令以创建一个 VPC。请将 `VPC_CIDR` 替换为某个 IPv4 RFC-1918（私有）或非 RFC-1918（公共）CIDR 范围（例如 `10.0.0.0/16`）。注意：VPC 会默认启用 DNS 解析，这是 EKS 的一项要求。

   ```
   aws ec2 create-vpc --cidr-block VPC_CIDR
   ```

2. 为 VPC 启用 DNS 主机名。VPC 会默认启用 DNS 解析。请将 `VPC_ID` 替换为在上一步中创建的 VPC 的 ID。

   ```
   aws ec2 modify-vpc-attribute --vpc-id VPC_ID --enable-dns-hostnames
   ```

第 2 步：创建子网

至少创建 2 个子网。Amazon EKS 会将这些子网作为集群网络接口。有关更多信息，请参阅子网要求和注意事项。

1. 您可以使用以下命令找到 Amazon 区域的可用区。请将 us-west-2 替换为您的区域。

```
aws ec2 describe-availability-zones \
     --query 'AvailabilityZones[?(RegionName == us-west-2)].ZoneName'
```

2. 创建子网。请将 VPC_ID 替换为 VPC 的 ID。请将 SUBNET_CIDR 替换为子网的 CIDR 块（例如 10.0.1.0/24）。请将 AZ 替换为将在其中创建子网的可用区（例如 us-west-2a）。您创建的子网必须至少位于 2 个不同的可用区。

```
aws ec2 create-subnet \
    --vpc-id VPC_ID \
    --cidr-block SUBNET_CIDR \
    --availability-zone AZ
```

（可选）第 3 步：使用 Amazon VPC Transit Gateway（TGW）或 Amazon Direct Connect 虚拟专用网关（VGW）附加 VPC

如果使用 TGW 或 VGW，请将您的 VPC 连接到该 TGW 或 VGW。有关更多信息，请参阅 Amazon VPC attachments in Amazon VPC Transit Gateways 或 Amazon Direct Connect virtual private gateway associations。

Transit Gateway

运行以下命令连接某个中转网关。请将 VPC_ID 替换为 VPC 的 ID。请将 SUBNET_ID1 和 SUBNET_ID2 替换为在上一步中创建的子网的 ID。请将 TGW_ID 替换为 TGW 的 ID。

```
aws ec2 create-transit-gateway-vpc-attachment \
    --vpc-id VPC_ID \
    --subnet-ids SUBNET_ID1 SUBNET_ID2 \
    --transit-gateway-id TGW_ID
```

虚拟专用网关

运行以下命令连接某个中转网关。请将 VPN_ID 替换为 VGW 的 ID。请将 VPC_ID 替换为 VPC 的 ID。

```
aws ec2 attach-vpn-gateway \
    --vpn-gateway-id VPN_ID \
    --vpc-id VPC_ID
```

（可选）第 4 步：创建路由表

您可以修改 VPC 的主路由表，也可以创建自定义路由表。以下步骤将创建一个自定义路由表，其中包含本地节点和容器组 CIDR 的路由。有关更多信息，请参阅子网路由表。请将 VPC_ID 替换为 VPC 的 ID。

```
aws ec2 create-route-table --vpc-id VPC_ID
```

第 5 步：创建本地节点和容器组的路由

在路由表中创建每个本地远程节点的路由。您可以修改 VPC 的主路由表，也可以使用您在上一步中创建的自定义路由表。

以下示例展示了如何创建本地节点和容器组 CIDR 的路由。这些示例使用中转网关（TGW）将 VPC 连接到本地环境。如果您有多个本地节点和容器组 CIDR，请对每个 CIDR 重复这些步骤。

- 如果您使用互联网网关或虚拟专用网关（VGW），请将 `--transit-gateway-id` 替换为 `--gateway-id`。
- 请将 RT_ID 替换为在上一步中创建的路由表的 ID。
- 请将 REMOTE_NODE_CIDR 替换为将用于混合节点的 CIDR 范围。
- 请将 REMOTE_POD_CIDR 替换为将在混合节点上运行的容器组的 CIDR 范围。容器组 CIDR 范围对应于容器网络接口（CNI）配置，后者最常使用的是叠加本地网络。有关更多信息，请参阅 the section called “配置 CNI”。
- 请将 TGW_ID 替换为 TGW 的 ID。

远程节点网络

```
aws ec2 create-route \
    --route-table-id RT_ID \
    --destination-cidr-block REMOTE_NODE_CIDR \
```

```
    --transit-gateway-id TGW_ID
```

远程容器组（pod）网络

```
aws ec2 create-route \
    --route-table-id RT_ID \
    --destination-cidr-block REMOTE_POD_CIDR \
    --transit-gateway-id TGW_ID
```

（可选）第 6 步：将子网关联到路由表

如果在上一步中创建了自定义路由表，请将您在上一步中创建的每个子网关联到您的自定义路由表。如果您选择修改 VPC 主路由表，则子网会自动关联到 VPC 的主路由表，因此可以跳过此步骤。

为您在之前步骤中创建的每个子网运行以下命令。请将 RT_ID 替换为在上一步中创建的路由表。请将 SUBNET_ID 替换为子网的 ID。

```
aws ec2 associate-route-table --route-table-id RT_ID --subnet-id SUBNET_ID
```

集群安全组配置

EKS 集群安全组需要具有以下访问权限才能进行持续集群操作。当您创建或更新配置了远程节点和容器组（pod）网络的集群时，Amazon EKS 会自动为混合节点创建所需的入站安全组规则。默认情况下，安全组允许所有出站流量，因此 Amazon EKS 不会自动为混合节点修改集群安全组的出站规则。如果要自定义集群安全组，可将流量限制在下表中的规则范围内。

类型	协议	Direction	端口	来源	目标	用法
HTTPS	TCP	入站	443	远程节点 CIDR	不适用	Kubelet 到 Kubernetes API 服务器
HTTPS	TCP	入站	443	远程容器组 CIDR	不适用	当 CNI 未使用 NAT 来处理容器组流量时，容器组需

类型	协议	Direction	端口	来源	目标	用法
						要访问 K8s API 服务器的权限。
HTTPS	TCP	出站	10250	不适用	远程节点 CIDR	Kubernetes API 服务器到 Kubelet
HTTPS	TCP	出站	Webhook 端口	不适用	远程容器组 CIDR	Kubernetes API 服务器到 Webhook（如果在混合节点上运行 Webhook）

Important

安全组规则限制：Amazon EC2 安全组默认最多有 60 条入站规则。如果集群安全组接近此限制，则安全组入站规则可能无法生效。在这种情况下，可能需要手动添加缺少的入站规则。
CIDR 清理责任：如果您从 EKS 集群中移除远程节点或容器组（pod）网络，EKS 不会自动移除相应的安全组规则。您负责从安全组规则中手动移除未使用的远程节点或容器组（pod）网络。

有关 Amazon EKS 创建的集群安全组的更多信息，请参阅 the section called “安全组要求”。

（可选）手动安全组配置

如果您需要创建其他安全组或修改自动创建的规则，可以参考以下命令。默认情况下，以下命令会创建一个允许所有出站访问的安全组。您可以将出站访问范围限定为仅包含上述规则。如果您考虑限制出站规则，我们建议您首先全面测试所有应用程序和容器组，然后再将更改后的规则应用于生产集群。

- 请将第一个命令中的 `SG_NAME` 替换为安全组的名称
- 请将第一个命令中的 `VPC_ID` 替换为在上一步中创建的 VPC 的 ID
- 请将第二个命令中的 `SG_ID` 替换为您在第一个命令中创建的安全组的 ID

- 请将第二个命令中的 REMOTE_NODE_CIDR 和 REMOTE_POD_CIDR 替换为混合节点和本地网络的相应值。

```
aws ec2 create-security-group \
    --group-name SG_NAME \
    --description "security group for hybrid nodes" \
    --vpc-id VPC_ID
```

```
aws ec2 authorize-security-group-ingress \
    --group-id SG_ID \
    --ip-permissions '[{"IpProtocol": "tcp", "FromPort": 443, "ToPort": 443,
 "IpRanges": [{"CidrIp": "REMOTE_NODE_CIDR"}, {"CidrIp": "REMOTE_POD_CIDR"}]}]'
```

为混合节点准备操作系统

Bottlerocket、Amazon Linux 2023（AL2023）、Ubuntu 和 RHEL 均已经过持续验证，可用作混合节点的节点操作系统。Amazon 仅在 VMware vSphere 环境中支持 Bottlerocket。在 Amazon EC2 之外运行时，AL2023 不属于 Amazon Support 计划的范围。AL2023 只能在本地虚拟化环境中使用；有关更多信息，请参阅《Amazon Linux 2023 User Guide》https://docs.amazonaws.cn/linux/al2023/ug/outside-ec2.html。Amazon 支持混合节点与 Ubuntu 和 RHEL 操作系统的集成，但不提供对操作系统本身的支持。

操作系统的预置和管理由您负责。首次测试混合节点时，最简单的方法是在一台已预置好的主机上运行 Amazon EKS 混合节点功能 CLI (nodeadm)。对于生产部署，建议在操作系统映像中包含 nodeadm 并将其配置为作为 systemd 服务运行，从而在主机启动时自动将主机加入 Amazon EKS 集群。如果在 vSphere 上将 Bottlerocket 作为节点操作系统使用，则无需使用 nodeadm，因为 Bottlerocket 已经包含混合节点功能所需的依赖项，并且会在主机启动时自动连接到您配置的集群。

版本兼容性

下表列举了经过验证可用作混合节点节点操作系统的兼容操作系统版本。如果使用此表中未包含的其他操作系统变体或版本，则混合节点与相关操作系统变体或版本的兼容性不属于 Amazon Support 的范围。混合节点与底层基础设施无关，x86 和 ARM 架构均受支持。

操作系统	版本
Amazon Linux	Amazon Linux 2023（AL2023）

操作系统	版本
Bottlerocket	运行 Kubernetes v1.28 及更高版本的 v1.37.0 及以上版本的 VMware 变体
Ubuntu	Ubuntu 20.04、Ubuntu 22.04、Ubuntu 24.04
Red Hat Enterprise Linux	RHEL 8、RHEL 9

操作系统注意事项

General

- Amazon EKS 混合节点功能 CLI (nodeadm) 可用于简化混合节点组件和依赖项的安装和配置。您可以在操作系统映像生成管道期间运行 `nodeadm install` 进程，也可以在每台本地主机的运行时中运行该进程。有关 nodeadm 将安装的组件的更多信息，请参阅the section called “混合节点 nodeadm”。
- 如果您使用本地环境的代理来访问互联网，则安装和升级过程需要完成额外的操作系统配置，才能将软件包管理器配置为使用该代理。有关说明，请参阅the section called “配置代理”：

Bottlerocket

- 连接 Bottlerocket 节点的步骤和工具与连接其他操作系统节点的步骤不同，将在the section called “连接使用 Bottlerocket 的混合节点”中单独介绍，而不是使用the section called “连接混合节点”中的步骤。
- 适用于 Bottlerocket 的步骤不使用混合节点 CLI 工具 `nodeadm`。
- 仅支持将 Bottlerocket 版本 v1.37.0 及更高版本的 VMware 变体与 EKS 混合节点结合使用。Bottlerocket 的 VMware 变体支持 Kubernetes 版本 1.28 及更高版本。不支持将其他 Bottlerocket 变体用作混合节点操作系统。注意：Bottlerocket 的 VMware 变体仅支持 x86_64 架构。

Containerd

- Containerd 是标准的 Kubernetes 容器运行时，也是混合节点以及所有 Amazon EKS 节点计算类型的一个依赖项。Amazon EKS 混合节点功能 CLI (nodeadm) 会尝试在 `nodeadm install` 进程中安装 containerd。您可以使用 `--containerd-source` 命令行选项在 `nodeadm install`

运行时中配置 containerd 安装。有效选项为 `none`、`distro`、和 `docker`。如果使用 RHEL，则 `distro` 不是有效选项，并且您可以将 `nodeadm` 配置为按照 Docker 存储库中的 containerd 版本，也可以手动安装 containerd。使用 AL2023 或 Ubuntu 时，`nodeadm` 默认为安装操作系统发行版中的 containerd。如果不希望 nodeadm 安装 containerd，请使用 `--containerd-source none` 选项。

Ubuntu

- 如果您使用的是 Ubuntu 24.04，则可能需要更新 containerd 版本或更改 AppArmor 配置，以采用一个可让容器组（pod）正确终止的修复程序，请参阅 Ubuntu #2065423。需要重启后才能将更改应用于 AppArmor 配置文件。最新版本 Ubuntu 24.04 的包管理器中包含更新的 containerd 版本，其中具有该修复程序（containerd 版本 1.7.19+）。

ARM

- 如果使用的是 ARM 硬件，则需要带有加密扩展（ARMv8.2+crypto）的 ARMv8.2 兼容处理器，才能运行 1.31 版本及更高版本的 EKS kube-proxy 附加组件。Raspberry Pi 5 之前的所有 Raspberry Pi 系统以及基于 Cortex-A72 的处理器都不符合此要求。要解决此问题，可以继续使用 1.30 版本的 EKS kube-proxy 附加组件，直到其延期支持在 2026 年 7 月终止（请参阅 Kubernetes 发布日历）；也可以使用来自上游的自定义 kube-proxy 映像。
- Kube-proxy 日志中的以下错误消息表明存在这样的不兼容性问题：

```
Fatal glibc error: This version of Amazon Linux requires a newer ARM64 processor
 compliant with at least ARM architecture 8.2-a with Cryptographic extensions. On EC2
 this is Graviton 2 or later.
```

生成操作系统映像

Amazon EKS 提供了示例 Packer 模板，您可以使用这些模板来创建包含 `nodeadm` 的操作系统映像，并将其配置为在主机启动时运行。建议使用此过程以避免在每台主机上分别拉取混合节点的依赖项，并自动完成混合节点的引导过程。您可以将示例 Packer 模板与 Ubuntu 22.04、Ubuntu 24.04、RHEL 8 或 RHEL 9 ISO 映像一起使用，并且可以输出以下格式的映像：OVA、Qcow2 或 Raw。

先决条件

在使用示例 Packer 模板之前，必须在运行 Packer 的计算机上安装以下项目。

- Packer 版本 1.11.0 或更高版本。有关安装 Packer 的说明，请参阅 Packer 文档中的 Install Packer。
- 如果生成 OVA，则必须已安装 VMware vSphere 插件 1.4.0 或更高版本
- 如果生成 Qcow2 或原始映像，则必须已安装 QEMU 插件版本 1.x

设置环境变量

在运行此 Packer 生成版本之前，请在运行 Packer 的计算机上设置以下环境变量。

General

要生成各种操作系统和输出格式的映像，必须设置以下环境变量。

环境变量	类型	说明
PKR_SSH_PASSWORD	字符串	Packer 使用 `ssh_username` 和 `ssh_password` 变量以在预置时通过 SSH 连接到创建的计算机。这需要与在相应操作系统的 kickstart 文件或 user-data 文件中创建初始用户时使用的密码一致。默认设置为“builder器”或“ubuntu”，具体因操作系统而异。设置密码时，请务必在相应的 `ks.cfg` 或 `user-data` 文件中进行更改以确保一致。
ISO_URL	字符串	要使用的 ISO URL。这可以是从服务器下载的 Web 链接，也可以是本地文件的绝对路径
ISO_CHECKSUM	字符串	所提供 ISO 的相关校验和。
CREDENTIAL_PROVIDER	字符串	混合节点的凭证提供者。SSM 混合激活的有效值为 `ssm`（默认），IAM Roles Anywhere 的有效值为 `iam`

环境变量	类型	说明
K8S_VERSION	字符串	适用于混合节点的 Kubernetes 版本（例如 `1.31`）。有关支持的 Kubernetes 版本，请参阅 Amazon EKS 支持的版本。
NODEADM_ARCH	字符串	适用于 `nodeadm install` 的架构。选择 `amd` 或 `arm`。

RHEL

如果使用 RHEL，则必须设置以下环境变量。

环境变量	类型	说明
RH_USERNAME	字符串	RHEL 订阅管理员用户名
RH_PASSWORD	字符串	RHEL 订阅管理器密码
RHEL_VERSION	字符串	使用的 Rhel iso 版本。有效值为 8 或 9。

Ubuntu -

没有特定于 Ubuntu 的环境变量。

vSphere

如果构建 VMware vSphere OVA，则必须设置以下环境变量。

环境变量	类型	说明
VSPHERE_SERVER	字符串	vSphere 服务器地址
VSPHERE_USER	字符串	vSphere 用户名
VSPHERE_PASSWORD	字符串	vSphere 密码

环境变量	类型	说明
VSPHERE_DATACENTER	字符串	vSphere 数据中心名称
VSPHERE_CLUSTER	字符串	vSphere 集群名称
VSPHERE_DATASTORE	字符串	vSphere 数据存储名称
VSPHERE_NETWORK	字符串	vSphere 网络名称
VSPHERE_OUTPUT_FOLDER	字符串	模板的 vSphere 输出文件夹

QEMU

环境变量	类型	说明
PACKER_OUTPUT_FORMAT	字符串	QEMU 生成器的输出格式。有效值为 `qcow2` 和 `raw`。

验证模版

在运行生成的版本之前，请在设置环境变量后使用以下命令验证模板。如果您的模板使用其他名称，请相应替换 `template.pkr.hcl`。

```
packer validate template.pkr.hcl
```

生成映像

使用以下命令生成映像，并使用 `-only` 标志来指定映像的目标和操作系统。如果您的模板使用其他名称，请相应替换 `template.pkr.hcl`。

vSphere OVA

> **Note**
>
> 如果将 RHEL 与 vSphere 结合使用，则需要将 kickstart 文件转换为 OEMDRV 映像，然后将其传递为要用于启动的 ISO。有关更多信息，请参阅 EKS 混合节点 GitHub 存储库中的 Packer Readme。

Ubuntu 22.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu24 template.pkr.hcl
```

RHEL 8 OVA

```
packer build -only=general-build.vsphere-iso.rhel8 template.pkr.hcl
```

RHEL 9 OVA

```
packer build -only=general-build.vsphere-iso.rhel9 template.pkr.hcl
```

QEMU

> **Note**
>
> 如果要为其生成映像的特定主机 CPU 生成与生成器主机不匹配，请参阅 QEMU 文档以获取与主机 CPU 匹配的名称，并在运行以下命令时将 `-cpu` 标志与该主机 CPU 的名称结合使用。

Ubuntu 22.04 Qcow2 / Raw

```
packer build -only=general-build.qemu.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 Qcow2 / Raw

```
packer build -only=general-build.qemu.ubuntu24 template.pkr.hcl
```

RHEL 8 Qcow2 / Raw

```
packer build -only=general-build.qemu.rhel8 template.pkr.hcl
```

RHEL 9 Qcow2 / Raw

```
packer build -only=general-build.qemu.rhel9 template.pkr.hcl
```

通过 user-data 传递 nodeadm 配置

您可以通过 cloud-init 在 user-data 中传递 nodeadm 的配置，从而配置混合节点并在主机启动时自动将其连接到 EKS 集群。以下示例演示了如何在将 VMware vSphere 作为混合节点的基础设施时实现这一点。

1. 按照 GitHub 上 govc readme 中的说明安装 govc CLI。
2. 运行上一节中的 Packer 版本并预置模板后，您可以通过使用以下方法克隆模板来创建多个不同的节点。您必须为要创建的用于混合节点的每个新 VM 克隆该模板。请将以下命令中的变量替换为您的环境的相应值。当您通过 metadata.yaml 文件注入 VM 的名称时，以下命令中的 VM_NAME 将用作 NODE_NAME。

   ```
   govc vm.clone -vm "/PATH/TO/TEMPLATE" -ds="YOUR_DATASTORE" \
       -on=false -template=false -folder=/FOLDER/TO/SAVE/VM "VM_NAME"
   ```

3. 在为每个新 VM 克隆该模板后，为您的 VM 创建一个 userdata.yaml 和 metadata.yaml。VM 可以共享同一 userdata.yaml 和 metadata.yaml，并且您将在以下步骤中为每个 VM 填充这些参数。nodeadm 配置是在 userdata.yaml 的 write_files 部分中创建和定义的。以下示例将 Amazon SSM 混合激活作为混合节点的本地凭证提供者。有关 nodeadm 配置的更多信息，请参阅the section called “混合节点 nodeadm”。

   userdata.yaml：

   ```
   #cloud-config
   users:
     - name: # username for login. Use 'builder' for RHEL or 'ubuntu' for Ubuntu.
       passwd: # password to login. Default is 'builder' for RHEL.
       groups: [adm, cdrom, dip, plugdev, lxd, sudo]
       lock-passwd: false
       sudo: ALL=(ALL) NOPASSWD:ALL
       shell: /bin/bash
   ```

```
write_files:
  - path: /usr/local/bin/nodeConfig.yaml
    permissions: '0644'
    content: |
      apiVersion: node.eks.aws/v1alpha1
      kind: NodeConfig
      spec:
          cluster:
              name: # Cluster Name
              region: # Amazon region
          hybrid:
              ssm:
                  activationCode: # Your ssm activation code
                  activationId: # Your ssm activation id

runcmd:
  - /usr/local/bin/nodeadm init -c file:///usr/local/bin/nodeConfig.yaml >> /var/log/
nodeadm-init.log 2>&1
```

metadata.yaml：

为您的环境创建一个 metadata.yaml。保留文件中的 "$NODE_NAME" 变量格式，因为这将使用后续步骤中的值填充。

```
instance-id: "$NODE_NAME"
local-hostname: "$NODE_NAME"
network:
  version: 2
  ethernets:
    nics:
      match:
        name: ens*
      dhcp4: yes
```

4. 使用以下命令将 userdata.yaml 和 metadata.yaml 文件添加为 gzip+base64 字符串。应为要创建的每个 VM 运行以下命令。请将 VM_NAME 替换为要更新的 VM 的名称。

```
export NODE_NAME="VM_NAME"
export USER_DATA=$(gzip -c9 <userdata.yaml | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata="${USER_DATA}"
```

```
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata.encoding=gzip+base64

envsubst '$NODE_NAME' < metadata.yaml > metadata.yaml.tmp
export METADATA=$(gzip -c9 <metadata.yaml.tmp | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata="${METADATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata.encoding=gzip+base64
```

5. 将新 VM 开机，这时应会自动连接到您配置的 EKS 集群。

```
govc vm.power -on "${NODE_NAME}"
```

准备用于混合节点的凭证

Amazon EKS 混合节点功能使用由 Amazon SSM 混合激活或 Amazon IAM Roles Anywhere 预置的临时 IAM 凭证，来进行 Amazon EKS 集群身份验证。您必须使用 Amazon SSM 混合激活，或将 Amazon IAM Roles Anywhere 与 Amazon EKS 混合节点功能 CLI (nodeadm) 配合使用。您不应同时使用 Amazon SSM 混合激活和 Amazon IAM Roles Anywhere。如果您没有具有证书颁发机构（CA）的现有公钥基础设施（PKI）和本地环境的证书，我们建议您使用 Amazon SSM 混合激活。如果您在确实具有现有的 PKI 和本地证书，请使用 Amazon IAM Roles Anywhere。

混合节点 IAM 角色

您必须首先为您的混合节点凭证创建一个将与 Amazon SSM 混合激活或 Amazon IAM Roles Anywhere 结合使用的 IAM 角色，然后才能将混合节点连接到 Amazon EKS 集群。创建集群后，您将使用此角色和某个 Amazon EKS 访问条目或 aws-auth ConfigMap 条目，来将该 IAM 角色映射到 Kubernetes 基于角色的访问控制（RBAC）。有关将混合节点 IAM 角色与 Kubernetes RBAC 关联的更多信息，请参阅the section called “准备集群访问权限”。

混合节点 IAM 角色必须具有以下权限。

- 需为 nodeadm 配置权限，使其能够调用 eks:DescribeCluster 操作，以获取待接入混合节点的目标集群相关信息。若未启用 eks:DescribeCluster 操作权限，则必须在传入 nodeadm init 命令的节点配置中，手动指定 Kubernetes API 端点、集群 CA 证书包以及服务 IPv4 CIDR。
- 需为 nodeadm 配置权限，使其能够调用 eks:ListAccessEntries 操作，以列出待接入混合节点的目标集群中的访问条目。若未启用 eks:ListAccessEntries 操作权限，则在执行 nodeadm init 命令时，必须传入 --skip cluster-access-validation 标记。

- 如 AmazonEC2ContainerRegistryPullOnly 策略所定义，kubelet 使用来自 Amazon Elastic Container Registry（Amazon ECR）的容器映像的权限。
- 若使用 Amazon SSM，需为 `nodeadm init` 配置权限，使其能够按照 aws-managed-policy/latest/reference/AmazonSSMManagedInstanceCore.html 中定义的规则，执行 Amazon SSM 混合激活操作。
- 如果使用 Amazon SSM，则为使用 `nodeadm uninstall` 的 `ssm:DeregisterManagedInstance` 操作和 `ssm:DescribeInstanceInformation` 操作注销实例的权限。
- （可选）Amazon EKS 容器组身份代理使用 `eks-auth:AssumeRoleForPodIdentity` 操作检索容器组凭证的权限。

设置 Amazon SSM 混合激活

必须首先创建并配置一个混合节点 IAM 角色，然后才能设置 Amazon SSM 混合激活。有关更多信息，请参阅 the section called “创建混合节点 IAM 角色”。按照《Amazon Systems Manager 用户指南》中创建混合激活以将节点注册到 Systems Manager 部分的说明，为您的混合节点创建一个 Amazon SSM 混合激活。将主机作为混合节点注册到 Amazon EKS 集群时，您收到的激活码和 ID 将与 `nodeadm` 结合使用。在为混合节点创建并准备好了 Amazon EKS 集群后，您可以稍后再回到此步骤。

> **Important**
>
> 根据您创建激活的方式，Systems Manager 立即将激活码和 ID 返回到控制台或命令窗口。复制此信息并将其存储在安全位置。如果您离开该控制台或关闭命令窗口，可能会丢失此信息。如果您丢失对应信息，则必须创建一个新激活。

默认情况下，Amazon SSM 混合激活会处于活动状态 24 小时。您也可以用时间戳格式（例如 `2024-08-01T00:00:00`）来指定 `--expiration-date` 何时创建混合激活。当凭证提供者为 Amazon SSM 时，混合节点的节点名称将不可配置，而是由 Amazon SSM 自动生成的。您可以在 Amazon Systems Manager 控制台中的 Fleet Manager 下查看和管理 Amazon SSM 托管式实例。每个账户在每个 Amazon 区域最多可以免费注册 1000 个标准混合激活节点。但是，注册超过 1000 个混合节点需要激活高级实例套餐。使用高级实例套餐将会产生费用，并且该费用未包含在 Amazon EKS 混合节点功能定价中。有关更多信息，请参阅 Amazon Systems Manager 定价。

请参阅以下示例，了解如何使用混合节点 IAM 角色创建 Amazon SSM 混合激活。将 Amazon SSM 混合激活作为混合节点凭证使用时，混合节点名称的格式将为 `mi-012345678abcdefgh`，Amazon

SSM 预置的临时凭证有效期为 1 小时。当凭证提供者为 Amazon SSM 时，您无法更改节点名称或凭证有效期。临时凭证将由 Amazon SSM 自动轮换，但此轮换不会影响节点或应用程序的状态。

我们建议您为每个 EKS 集群使用一个 Amazon SSM 混合激活来限制混合节点 IAM 角色的 Amazon SSM `ssm:DeregisterManagedInstance` 权限，使其只能注销与您的 Amazon SSM 混合激活关联的实例。本页中的示例使用了一个具有 EKS 集群 ARN 的标签，该标签可用于将您的 Amazon SSM 混合激活映射到该 EKS 集群。您也可以根据自己的权限边界和要求，使用自己偏好的标签和方法来确定 Amazon SSM 权限的范围。以下命令中的 `REGISTRATION_LIMIT` 选项是一个整型，用于限制可以使用 Amazon SSM 混合激活的计算机数量（例如 `10`）

```
aws ssm create-activation \
     --region AWS_REGION \
     --default-instance-name eks-hybrid-nodes \
     --description "Activation for EKS hybrid nodes" \
     --iam-role AmazonEKSHybridNodesRole \
     --tags Key=EKSClusterARN,Value=arn:aws-cn:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/
CLUSTER_NAME \
     --registration-limit REGISTRATION_LIMIT
```

请参阅创建混合激活以将节点注册到 Systems Manager 部分的说明，以详细了解有关 Amazon SSM 混合激活的可用配置设置。

设置 Amazon IAM Roles Anywhere

按照《IAM Roles Anywhere 用户指南》中 Getting started with IAM Roles Anywhere 部分的说明，设置将用作混合节点 IAM 角色的临时 IAM 凭证的信任锚和配置文件。创建配置文件时，您可以仅创建配置文件而不添加任何角色。您可以首先创建此配置文件，然后返回这些步骤来创建混合节点 IAM 角色，最后再将创建的角色添加到您的配置文件。您也可以使用本页后面的 Amazon CloudFormation 步骤来完成混合节点的 IAM Roles Anywhere 设置。

将混合节点 IAM 角色添加到配置文件时，请在 Amazon IAM Roles Anywhere 控制台中编辑配置文件页面底部的自定义角色会话名称面板中，选择接受自定义角色会话名称。这对应于 `CreateProfile` API 的 acceptRoleSessionName 字段。选择此选项后，您可以在引导阶段传递到 `nodeadm` 的配置中为混合节点提供自定义节点名称。在 `nodeadm init` 过程中必须传递一个自定义节点名称。创建配置文件后，您可以通过更新配置文件来接受自定义角色会话名称。

您可以通过 Amazon IAM Roles Anywhere 配置文件的 durationSeconds 字段，来配置 Amazon IAM Roles Anywhere 的凭证有效期。默认有效期为 1 小时，最长可为 12 小时。混合节点 IAM 角色的 `MaxSessionDuration` 设置必须大于 Amazon IAM Roles Anywhere 配置文件中的

durationSeconds 设置。有关 MaxSessionDuration 的更多信息，请参阅 UpdateRole API 文档。

您通过证书颁发机构（CA）为每台计算机生成的证书和密钥必须放置在每个混合节点的 /etc/iam/pki 目录中，并且证书的文件名必须为 server.pem，密钥的文件名必须为 server.key。

创建混合节点 IAM 角色

要运行本节中的步骤，使用 Amazon 控制台或 Amazon CLI 的 IAM 主体必须具有以下权限。

- iam:CreatePolicy
- iam:CreateRole
- iam:AttachRolePolicy
- 如果使用的是 Amazon IAM Roles Anywhere
  - rolesanywhere:CreateTrustAnchor
  - rolesanywhere:CreateProfile
  - iam:PassRole

Amazon CloudFormation

安装并配置 Amazon CLI（如果尚未执行此操作）。请参阅安装或更新到最新版本的 Amazon CLI。

Amazon SSM 混合激活的步骤

CloudFormation 堆栈将创建具有上述权限的混合节点 IAM 角色。CloudFormation 模板不会创建 Amazon SSM 混合激活。

1. 下载适用于混合节点的 Amazon SSM CloudFormation 模板：

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/hybrid-ssm-cfn.yaml'
```

2. 使用以下选项创建 cfn-ssm-parameters.json：
   a. 将 ROLE_NAME 替换为混合节点 IAM 角色的名称。如果您未指定名称，则默认情况下 CloudFormation 模板将使用 AmazonEKSHybridNodesRole 作为其创建的角色的名称。
   b. 请将 TAG_KEY 替换为您在创建 Amazon SSM 混合激活时使用的 Amazon SSM 资源标签键。在条件中使用标签键和标签值组合，从而让 ssm:DeregisterManagedInstance 仅允许混合节点 IAM 角色注销与您的 Amazon SSM 混合激活关联的 Amazon SSM 托管式实例。在 CloudFormation 模板中，TAG_KEY 默认为 EKSClusterARN。

c. 请将 TAG_VALUE 替换为您在创建 Amazon SSM 混合激活时使用的 Amazon SSM 资源标签值。在条件中使用标签键和标签值组合，从而让 ssm:DeregisterManagedInstance 仅允许混合节点 IAM 角色注销与您的 Amazon SSM 混合激活关联的 Amazon SSM 托管式实例。如果您使用 EKSClusterARN 的默认值 TAG_KEY，则请将 EKS 集群 ARN 传递为 TAG_VALUE。EKS 集群 ARN 的格式为 arn:aws-cn:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME。

```
{
  "Parameters": {
    "RoleName": "ROLE_NAME",
    "SSMDeregisterConditionTagKey": "TAG_KEY",
    "SSMDeregisterConditionTagValue": "TAG_VALUE"
  }
}
```

3. 部署 CloudFormation 堆栈。请将 STACK_NAME 替换 CloudFormation 堆栈的名称。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --template-file hybrid-ssm-cfn.yaml \
    --parameter-overrides file://cfn-ssm-parameters.json \
    --capabilities CAPABILITY_NAMED_IAM
```

Amazon IAM Roles Anywhere 的步骤

CloudFormation 堆栈会使用您配置的证书颁发机构（CA）来创建 Amazon IAM Roles Anywhere 信任锚，创建 Amazon IAM Roles Anywhere 配置文件，并创建具有前文所述权限的混合节点 IAM 角色。

1. 设置证书颁发机构（CA）

   a. 要使用 Amazon 私有 CA 资源，请打开 Amazon Private Certificate Authority 控制台。按照 Amazon Private CA 用户指南中的说明进行操作。

   b. 要使用外部 CA，请按照相关 CA 提供的说明进行操作。您需要在后续步骤中提供证书正文。

   c. 公有 CA 颁发的证书不能用作信任锚。

2. 下载适用于混合节点的 Amazon IAM Roles Anywhere CloudFormation 模板

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/hybrid-ira-cfn.yaml'
```

3. 使用以下选项创建 cfn-iamra-parameters.json：

   a. 将 ROLE_NAME 替换为混合节点 IAM 角色的名称。如果您未指定名称，则默认情况下 CloudFormation 模板将使用 AmazonEKSHybridNodesRole 作为其创建的角色的名称。

   b. 请将 CERT_ATTRIBUTE 替换为用于唯一标识主机的每台计算机的证书属性。使用的证书属性必须与您在将混合节点连接到集群时用于 nodeadm 配置的 nodeName 一致。有关更多信息，请参阅the section called “混合节点 nodeadm”。默认情况下，CloudFormation 模板将 ${aws:PrincipalTag/x509Subject/CN} 作为 CERT_ATTRIBUTE，这将对应于每台计算机的证书的 CN 字段。您也可以将 $(aws:PrincipalTag/x509SAN/Name/CN} 传递为您的 CERT_ATTRIBUTE。

   c. 请将 CA_CERT_BODY 替换为 CA 的证书正文，不带换行符。CA_CERT_BODY 必须采用隐私增强邮件（PEM）格式。如果您具有 PEM 格式的 CA 证书，请在将 CA 证书正文放入 cfn-iamra-parameters.json 文件之前移除换行符和 BEGIN CERTIFICATE 和 END CERTIFICATE 行。

```
{
  "Parameters": {
    "RoleName": "ROLE_NAME",
    "CertAttributeTrustPolicy": "CERT_ATTRIBUTE",
    "CABundleCert": "CA_CERT_BODY"
  }
}
```

4. 部署 CloudFormation 模板。请将 STACK_NAME 替换 CloudFormation 堆栈的名称。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --template-file hybrid-ira-cfn.yaml \
    --parameter-overrides file://cfn-iamra-parameters.json
    --capabilities CAPABILITY_NAMED_IAM
```

Amazon CLI

安装并配置 Amazon CLI（如果尚未执行此操作）。请参阅安装或更新到最新版本的 Amazon CLI。

创建 EKS 描述集群策略

1. 使用以下内容创建名为 eks-describe-cluster-policy.json 的文件：

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster"
            ],
            "Resource": "*"
        }
    ]
}
```

2. 使用以下命令创建此策略：

```
aws iam create-policy \
    --policy-name EKSDescribeClusterPolicy \
    --policy-document file://eks-describe-cluster-policy.json
```

Amazon SSM 混合激活的步骤

1. 使用以下内容创建名为 eks-hybrid-ssm-policy.json 的文件。此策略将授予对 ssm:DescribeInstanceInformation 和 ssm:DeregisterManagedInstance 这两个操作的权限。此策略将根据您在信任策略中指定的资源标签，将 ssm:DeregisterManagedInstance 权限范围限定为与您的 Amazon SSM 混合激活关联的 Amazon SSM 托管式实例。

   a. 请将 AWS_REGION 替换为您的 Amazon SSM 混合激活的 Amazon 区域。

   b. 将 AWS_ACCOUNT_ID 替换为您的Amazon账户 ID。

   c. 请将 TAG_KEY 替换为您在创建 Amazon SSM 混合激活时使用的 Amazon SSM 资源标签键。在条件中使用标签键和标签值组合，从而让 ssm:DeregisterManagedInstance 仅允许混合节点 IAM 角色注销与您的 Amazon SSM 混合激活关联的 Amazon SSM 托管式实例。在 CloudFormation 模板中，TAG_KEY 默认为 EKSClusterARN。

   d. 请将 TAG_VALUE 替换为您在创建 Amazon SSM 混合激活时使用的 Amazon SSM 资源标签值。在条件中使用标签键和标签值组合，从而让 ssm:DeregisterManagedInstance 仅允许混合节点 IAM 角色注销与您的 Amazon SSM 混合激活关联的 Amazon SSM 托管式实例。如果您使用 EKSClusterARN 的默认值 TAG_KEY，则请将 EKS 集群 ARN 传递为 TAG_VALUE。EKS 集群 ARN 的格式为 arn:aws-cn:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME。

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ssm:DescribeInstanceInformation",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:DeregisterManagedInstance",
            "Resource": "arn:aws:ssm:us-east-1:123456789012:managed-instance/*",
            "Condition": {
                "StringEquals": {
                    "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
                }
            }
        }
    ]
}
```

2. 使用以下命令创建此策略

```
aws iam create-policy \
    --policy-name EKSHybridSSMPolicy \
    --policy-document file://eks-hybrid-ssm-policy.json
```

3. 创建一个名为 eks-hybrid-ssm-trust.json的文件。请将 AWS_REGION 替换为您的 Amazon SSM 混合激活所在的 Amazon 区域，将 AWS_ACCOUNT_ID 替换为您的 Amazon 账户 ID。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"",
         "Effect":"Allow",
         "Principal":{
            "Service":"ssm.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            },
            "ArnEquals":{
```

```
                "aws:SourceArn":"arn:aws:ssm:us-east-1:123456789012:*"
            }
        }
    }
  ]
}
```

4. 使用以下命令创建此角色。

```
aws iam create-role \
    --role-name AmazonEKSHybridNodesRole \
    --assume-role-policy-document file://eks-hybrid-ssm-trust.json
```

5. 附加您在之前步骤中创建的 EKSDescribeClusterPolicy 和 EKSHybridSSMPolicy。将 AWS_ACCOUNT_ID 替换为您的Amazon账户 ID。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws-cn:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws-cn:iam::AWS_ACCOUNT_ID:policy/EKSHybridSSMPolicy
```

6. 附加 AmazonEC2ContainerRegistryPullOnly 和 AmazonSSMManagedInstanceCore Amazon 托管式策略。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonSSMManagedInstanceCore
```

Amazon IAM Roles Anywhere 的步骤

要使用 Amazon IAM Roles Anywhere，您必须在创建混合节点 IAM 角色之前设置 Amazon IAM Roles Anywhere信任锚。有关说明，请参阅the section called “设置 Amazon IAM Roles Anywhere”：

1. 创建一个名为 eks-hybrid-iamra-trust.json的文件。请将 TRUST_ANCHOR ARN 替换为您在“the section called “设置 Amazon IAM Roles Anywhere””步骤中创建的信任锚的 ARN。只有当角色会话名称与混合节点上安装的 x509 证书中的 CN 一致时，此信任策略中的条件才会将 Amazon IAM Roles Anywhere 代入混合节点 IAM 角色的权限范围限定为仅交换临时 IAM 凭证。您也可以使用其他证书属性来唯一标识节点。您在信任策略中使用的证书属性必须与您在 nodeadm 配置中设置的 nodeName 对应。有关更多信息，请参阅the section called “混合节点 nodeadm”。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:TagSession",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}",
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
```

```
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        }
    ]
}
```

2. 使用以下命令创建此角色。

```
aws iam create-role \
    --role-name AmazonEKSHybridNodesRole \
    --assume-role-policy-document file://eks-hybrid-iamra-trust.json
```

3. 附加您在之前步骤中创建的 EKSDescribeClusterPolicy。将 AWS_ACCOUNT_ID 替换为您的 Amazon账户 ID。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws-cn:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

4. 附加 AmazonEC2ContainerRegistryPullOnly Amazon 托管式策略

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

Amazon Web Services 管理控制台

创建 EKS 描述集群策略

1. 打开 Amazon IAM 控制台
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 在策略页面上，选择创建策略。
4. 在“指定权限”页面的“选择服务”面板中，选择 EKS。
   a. 筛选 DescribeCluster 的操作，然后选择 DescribeCluster 读取操作。
   b. 选择下一步。
5. 在检查并创建页面上

a. 输入策略的策略名称，如 EKSDescribeClusterPolicy。

b. 选择创建策略。

Amazon SSM 混合激活的步骤

1. 打开 Amazon IAM 控制台
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 在策略页面上，选择创建策略。
4. 在指定权限页面右上角的策略编辑器导航中，选择 JSON。粘贴以下代码段。请将 AWS_REGION 替换为您的 Amazon SSM 混合激活所在的 Amazon 区域，并将 AWS_ACCOUNT_ID 替换为您的 Amazon 账户 ID。请将 TAG_KEY 和 TAG_VALUE 替换为您在创建 Amazon SSM 混合激活时使用的 Amazon SSM 资源标签键。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ssm:DescribeInstanceInformation",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:DeregisterManagedInstance",
            "Resource": "arn:aws:ssm:us-east-1:123456789012:managed-instance/*",
            "Condition": {
                "StringEquals": {
                    "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
                }
            }
        }
    ]
}
```

a. 选择下一步。

5. 在检查并创建页面上，

a. 输入策略的策略名称，如 EKSHybridSSMPolicy

b. 选择创建策略。

6. 在左侧导航窗格中，选择 Roles（角色）。
7. 在 Roles（角色）页面上，选择 Create role（创建角色）。
8. 在 Select trusted entity（选择受信任的实体）页面上，请执行以下操作：
   a. 在可信实体类型部分中，选择自定义信任策略。将以下策略复制粘贴到自定义信任策略编辑器中。请将 AWS_REGION 替换为您的 Amazon SSM 混合激活所在的 Amazon 区域，将 AWS_ACCOUNT_ID 替换为您的 Amazon 账户 ID。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"",
         "Effect":"Allow",
         "Principal":{
            "Service":"ssm.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:ssm:us-east-1:123456789012:*"
            }
         }
      }
   ]
}
```

   b. 选择下一步。
9. 在添加权限页面上，附加自定义策略或执行以下操作：
   a. 在筛选策略对话框中，输入 EKSDescribeClusterPolicy 或您上面创建的策略的名称。选中搜索结果中您的策略左侧的复选框。
   b. 在筛选策略对话框中，输入 EKSHybridSSMPolicy 或您上面创建的策略的名称。选中搜索结果中您的策略左侧的复选框。
   c. 在 Filter policies (筛选器策略) 框中，输入 AmazonEC2ContainerRegistryPullOnly。选中搜索结果中 AmazonEC2ContainerRegistryPullOnly 左侧的复选框。
   d. 在 Filter policies (筛选器策略) 框中，输入 AmazonSSMManagedInstanceCore。选中搜索结果中 AmazonSSMManagedInstanceCore 左侧的复选框。

e. 选择下一步。

10.在 Name, review, and create（命名、查看和创建）页面中，请执行以下操作：

a. 对于 Role name (角色名称)，为角色输入唯一名称，例如 `AmazonEKSHybridNodesRole`。

b. 对于 Description（说明），请将当前文本替换为描述性文本，例如 `Amazon EKS - Hybrid Nodes role`。

c. 选择创建角色。

Amazon IAM Roles Anywhere 的步骤

要使用 Amazon IAM Roles Anywhere，您必须在创建混合节点 IAM 角色之前设置 Amazon IAM Roles Anywhere信任锚。有关说明，请参阅the section called “设置 Amazon IAM Roles Anywhere”：

1. 打开 Amazon IAM 控制台
2. 在左侧导航窗格中，选择 Roles（角色）。
3. 在 Roles（角色）页面上，选择 Create role（创建角色）。
4. 在 Select trusted entity（选择受信任的实体）页面上，请执行以下操作：

a. 在可信实体类型部分中，选择自定义信任策略。将以下策略复制粘贴到自定义信任策略编辑器中。请将 `TRUST_ANCHOR ARN` 替换为您在“the section called “设置 Amazon IAM Roles Anywhere””步骤中创建的信任锚的 ARN。只有当角色会话名称与混合节点上安装的 x509 证书中的 CN 一致时，此信任策略中的条件才会将 Amazon IAM Roles Anywhere 代入混合节点 IAM 角色的权限范围限定为仅交换临时 IAM 凭证。您也可以使用其他证书属性来唯一标识节点。您在信任策略中使用的证书属性必须与您在 nodeadm 配置中设置的 nodeName 对应。有关更多信息，请参阅the section called “混合节点 nodeadm”。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:TagSession",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
```

```
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}",
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        }
    ]
}
```

b. 选择下一步。

5. 在添加权限页面上，附加自定义策略或执行以下操作：

   a. 在筛选策略对话框中，输入 EKSDescribeClusterPolicy 或您上面创建的策略的名称。选中搜索结果中您的策略左侧的复选框。

   b. 在 Filter policies (筛选器策略) 框中，输入 AmazonEC2ContainerRegistryPullOnly。选中搜索结果中 AmazonEC2ContainerRegistryPullOnly 左侧的复选框。

   c. 选择下一步。

6. 在 Name, review, and create（命名、查看和创建）页面中，请执行以下操作：

   a. 对于 Role name (角色名称)，为角色输入唯一名称，例如 AmazonEKSHybridNodesRole。

b. 对于 Description（说明），请将当前文本替换为描述性文本，例如 `Amazon EKS - Hybrid Nodes role`。

c. 选择创建角色。

创建具有混合节点的 Amazon EKS 集群

本主题概述了创建启用混合节点功能的 Amazon EKS 集群时可用的选项，并介绍了相关注意事项。EKS 混合节点功能提供的 Kubernetes 版本支持与具有云端节点的 Amazon EKS 集群相同，包括标准和扩展支持。

如果您不计划使用 EKS 混合节点功能，请参阅有关 Amazon EKS 创建集群的主文档：the section called “创建集群”。

先决条件

- 已完成the section called “先决条件”。在创建启用混合节点功能的集群之前，必须确定本地节点和可选的容器组 CIDR，根据 EKS 要求和混合节点要求创建 VPC 和子网，并且具有包含本地入站规则和（可选）容器组 CIDR 的安全组。有关这些先决条件的更多信息，请参阅the section called “准备联网”。
- 已在您的设备上安装并配置了最新版本的 Amazon 命令行界面（Amazon CLI）。要查看当前版本，请使用 `aws --version`。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装或更新到最新版本的 Amazon CLI 和配置 Amazon CLI 设置。
- 一个具有创建 IAM 角色和附加策略以及创建和描述 EKS 集群的权限的 IAM 主体

注意事项

- 集群的集群身份验证模式必须为 `API` 或 `API_AND_CONFIG_MAP`。
- 集群必须使用 IPv4 地址系列。
- 集群必须使用公有或私有集群端点连接。集群不能使用“公有和私有”集群端点连接，因为 Amazon EKS Kubernetes API 服务器端点将解析为在您的 VPC 之外运行的混合节点公有 IP。
- 具有混合节点的 Amazon EKS 集群支持 OIDC 身份验证。
- 您可以从现有集群中添加、更改或移除混合节点配置。有关更多信息，请参阅 the section called “现有集群”。

第 1 步：创建集群 IAM 角色

如果您已经拥有集群 IAM 角色，或者您将使用 `eksctl` 或Amazon CloudFormation 创建集群，则可以跳过此步骤。默认情况下，`eksctl` 和 Amazon CloudFormation 模板会为您创建集群 IAM 角色。

1. 运行以下命令以创建 IAM 信任策略 JSON 文件。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 创建 Amazon EKS 集群 IAM 角色。如有必要，使用您在上一步中将文件写入到的计算机上的路径为 eks-cluster-role-trust-policy.json 添加前言。该命令将您在上一步中创建的信任策略与角色关联。要创建 IAM 角色，必须为正在创建角色的 IAM 主体分配 `iam:CreateRole` 操作（权限）。

```
aws iam create-role \
    --role-name myAmazonEKSClusterRole \
    --assume-role-policy-document file://"eks-cluster-role-trust-policy.json"
```

3. 您可以分配 Amazon EKS 托管策略或创建自己的自定义策略。有关必须在自定义策略中使用的最低权限，请参阅 the section called “节点 IAM 角色”。将名为 `AmazonEKSClusterPolicy` 的 Amazon EKS 托管 IAM 策略附加到角色。要将 IAM 策略附加到某个 IAM 主体，必须为附加该策略的主体分配以下 IAM 操作（权限）之一：`iam:AttachUserPolicy` 或 `iam:AttachRolePolicy`。

```
aws iam attach-role-policy \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSClusterPolicy \
    --role-name myAmazonEKSClusterRole
```

第 2 步：创建启用混合节点功能的集群

您可以使用以下工具来创建集群：

- eksctl
- Amazon CloudFormation
- Amazon CLI
- Amazon Web Services 管理控制台

创建启用混合节点功能的集群 – eksctl

您需要安装最新版本的 `eksctl` 命令行工具。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的 Installation。

1. 创建 `cluster-config.yaml` 以定义启用混合节点功能的 Amazon EKS IPv4 集群。在 `cluster-config.yaml` 中进行以下替换。有关完整的设置列表，请参阅 eksctl 文档。
   a. 将 `CLUSTER_NAME` 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   b. 请将 `AWS_REGION` 替换为要在其中创建集群的 Amazon 区域。
   c. 将 `K8S_VERSION` 替换为 Amazon EKS 支持的版本。
   d. 请根据您在 the section called “准备凭证” 步骤中配置的凭证提供者，将 `CREDS_PROVIDER` 替换为 `ssm` 或 `ira`。
   e. 如果凭证提供者设置为 `ira`，请替换 CA_BUNDLE_CERT，这会将 Amazon IAM Roles Anywhere 作为凭证提供者。CA_BUNDLE_CERT 是证书颁发机构（CA）的证书正文，具体取决于您选择的 CA。该证书必须采用隐私增强邮件（PEM）格式。
   f. 请将 `GATEWAY_ID` 替换为要连接到 VPC 的虚拟专用网关或中转网关的 ID。
   g. 请将 `REMOTE_NODE_CIDRS` 替换为混合节点的本地节点 CIDR。
   h. 对于在混合节点上运行的工作负载，请将 `REMOTE_POD_CIDRS` 替换为本地容器组 CIDR；如果您不在混合节点上运行 Webhook，则请从配置中移除该行。如果您的 CNI 在容器组流量离开本地主机时不使用网络地址转换（NATI）或容器组 IP 地址伪装，则必须配置 `REMOTE_POD_CIDRS`。如果在混合节点上运行 Webhook，则必须配置 `REMOTE_POD_CIDRS`，更多信息请参阅the section called “配置 Webhook”。
   i. 本地节点和容器组 CIDR 块必须满足以下要求：
      i. 处于以下 IPv4 RFC–1918 范围之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。
      ii. 不相互重叠，也不与集群的 VPC CIDR 或 Kubernetes 服务 IPv4 CIDR 重叠

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: CLUSTER_NAME
  region: AWS_REGION
  version: "K8S_VERSION"

remoteNetworkConfig:
  iam:
    provider: CREDS_PROVIDER # default SSM, can also be set to IRA
    # caBundleCert: CA_BUNDLE_CERT
  vpcGatewayID: GATEWAY_ID
  remoteNodeNetworks:
  - cidrs: ["REMOTE_NODE_CIDRS"]
  remotePodNetworks:
  - cidrs: ["REMOTE_POD_CIDRS"]
```

2. 运行如下命令：

```
eksctl create cluster -f cluster-config.yaml
```

集群预配置需要几分钟时间。在创建集群时，将显示几行输出。输出的最后一行类似于以下示例行。

```
[#]  EKS cluster "CLUSTER_NAME" in "REGION" region is ready
```

3. 继续the section called “第 3 步：更新 kubeconfig”。

创建启用混合节点功能的集群 – Amazon CloudFormation

CloudFormation 堆栈使用您指定的 RemoteNodeNetwork 和 RemotePodNetwork 来创建 EKS 集群 IAM 角色和 EKS 集群。如果您需要自定义未在 CloudFormation 模板中公开的 EKS 集群设置，请修改 CloudFormation 模板。

1. 下载 CloudFormation 模板文件。

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-eks-cfn.yaml'
```

2. 创建 cfn-eks-parameters.json 并指定每个值的配置。

   a. CLUSTER_NAME：要创建的 EKS 集群的名称

   b. CLUSTER_ROLE_NAME：要创建的 EKS 集群 IAM 角色的名称。模板中的默认值为“EKSClusterRole”。

   c. SUBNET1_ID：您在先决条件步骤中创建的第一个子网的 ID

   d. SUBNET2_ID：您在先决条件步骤中创建的第二个子网的 ID

   e. SG_ID：您在先决条件步骤中创建的安全组 ID

   f. REMOTE_NODE_CIDRS：混合节点的本地节点 CIDR

   g. REMOTE_POD_CIDRS：在混合节点上运行的工作负载的本地容器组 CIDR。如果您的 CNI 在容器组流量离开本地主机时不使用网络地址转换（NATI）或容器组 IP 地址伪装，则必须配置 REMOTE_POD_CIDRS。如果在混合节点上运行 Webhook，则必须配置 REMOTE_POD_CIDRS，更多信息请参阅the section called “配置 Webhook”。

   h. 本地节点和容器组 CIDR 块必须满足以下要求：

      i. 处于以下 IPv4 RFC–1918 范围之一：10.0.0.0/8、172.16.0.0/12 或 192.168.0.0/16。

      ii. 不相互重叠，也不与集群的 VPC CIDR 或 Kubernetes 服务 IPv4 CIDR 重叠。

   i. CLUSTER_AUTH：集群的集群身份验证模式。有效值为 API 和 API_AND_CONFIG_MAP。模版中的默认值为 API_AND_CONFIG_MAP。

   j. CLUSTER_ENDPOINT：集群的集群端点连接。有效值为“公有”或“私有”。模板中的默认值为“私有”，因此只能从 VPC 内连接到 Kubernetes API 端点。

   k. K8S_VERSION：要用于集群的 Kubernetes 版本。请参阅 Amazon EKS 支持的版本。

```
{
  "Parameters": {
    "ClusterName": "CLUSTER_NAME",
    "ClusterRoleName": "CLUSTER_ROLE_NAME",
    "SubnetId1": "SUBNET1_ID",
    "SubnetId2": "SUBNET2_ID",
    "SecurityGroupId" "SG_ID",
    "RemoteNodeCIDR": "REMOTE_NODE_CIDRS",
    "RemotePodCIDR": "REMOTE_POD_CIDRS",
    "ClusterAuthMode": "CLUSTER_AUTH",
    "ClusterEndpointConnectivity": "CLUSTER_ENDPOINT",
    "K8sVersion": "K8S_VERSION"
  }
 }
```

3. 部署 CloudFormation 堆栈。请将 STACK_NAME 替换为 CloudFormation 堆栈的名称，并将 AWS_REGION 替换为将会创建集群的 Amazon 区域。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --region AWS_REGION \
    --template-file hybrid-eks-cfn.yaml \
    --parameter-overrides file://cfn-eks-parameters.json \
    --capabilities CAPABILITY_NAMED_IAM
```

集群预配置需要几分钟时间。您可以使用以下命令检查堆栈的状态。请将 STACK_NAME 替换为 CloudFormation 堆栈的名称，并将 AWS_REGION 替换为将会创建集群的 Amazon 区域。

```
aws cloudformation describe-stacks \
    --stack-name STACK_NAME \
    --region AWS_REGION \
    --query 'Stacks[].StackStatus'
```

4. 继续the section called “第 3 步：更新 kubeconfig”。

创建启用混合节点功能的集群 – Amazon CLI

1. 运行以下命令创建启用混合节点功能的 EKS 集群。在运行此命令之前，请将以下参数替换为相应的设置。有关完整的设置列表，请参阅the section called “创建集群”文档。
   a. CLUSTER_NAME：要创建的 EKS 集群的名称
   b. AWS_REGION：将创建集群的 Amazon 区域。
   c. K8S_VERSION：要用于集群的 Kubernetes 版本。请参阅“Amazon EKS 支持的版本”。
   d. ROLE_ARN：您为集群配置的 Amazon EKS 集群角色。有关更多信息，请参阅“Amazon EKS 集群 IAM 角色”。
   e. SUBNET1_ID：您在先决条件步骤中创建的第一个子网的 ID
   f. SUBNET2_ID：您在先决条件步骤中创建的第二个子网的 ID
   g. SG_ID：您在先决条件步骤中创建的安全组 ID
   h. 您可以将 API 和 API_AND_CONFIG_MAP 作为集群访问身份验证模式。在下面的命令中，集群访问身份验证模式设置为 API_AND_CONFIG_MAP。

i. 您可以使用 endpointPublicAccess 和 endpointPrivateAccess 参数来启用或禁用对集群 Kubernetes API 服务器端点的公有和私有访问。在下面的命令中，endpointPublicAccess 设置为 false，endpointPrivateAccess 设置为 true。

j. REMOTE_NODE_CIDRS：混合节点的本地节点 CIDR。

k. REMOTE_POD_CIDRS（可选）：在混合节点上运行的工作负载的本地容器组 CIDR。

l. 本地节点和容器组 CIDR 块必须满足以下要求：

i. 处于以下 IPv4 RFC–1918 范围之一：10.0.0.0/8、172.16.0.0/12 或 192.168.0.0/16。

ii. 不相互重叠，也不与 Amazon EKS 集群的 VPC CIDR 或 Kubernetes 服务 IPv4 CIDR 重叠。

```
aws eks create-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --kubernetes-version K8S_VERSION \
    --role-arn ROLE_ARN \
    --resources-vpc-config
 subnetIds=SUBNET1_ID,SUBNET2_ID,securityGroupIds=SG_ID,endpointPrivateAccess=true,endpo
 \
    --access-config authenticationMode=API_AND_CONFIG_MAP \
    --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. 预置集群需要几分钟时间。可使用以下命令查询集群的状态。请将 CLUSTER_NAME 替换为要创建的集群的名称，并将 AWS_REGION 替换为要创建集群的 Amazon 区域。在返回的输出为 ACTIVE 之前，请勿继续执行下一步。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

3. 继续the section called “第 3 步：更新 kubeconfig”。

创建启用混合节点功能的集群 – Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。

2. 请选择 Add cluster（添加集群），然后选择 Create（创建）。

3. 在 Configure cluster（配置集群）页面上，输入以下字段：

a. Name（名称）– 集群的名称。名称只能包含字母数字字符（区分大小写）、连字符和下划线。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。

b. 集群 IAM 角色 – 选择您创建的 Amazon EKS 集群 IAM 角色，以允许 Kubernetes 控制面板来代表您管理 Amazon 资源。

c. Kubernetes version（Kubernetes 版本）– 要用于集群的 Kubernetes 的版本。建议选择最新版本，除非您需要早期版本。

d. 升级策略 – 选择“扩展”或“标准”。

   i. 扩展：选中此选项后，将从发布之日起为 Kubernetes 版本提供 26 个月的支持。扩展支持期会额外按小时收取费用，从标准支持期结束后开始计算。扩展支持终止时，集群将自动升级到下一个版本。

   ii. 标准：选中此选项后，将从发布之日起为 Kubernetes 版本提供 14 个月的支持。此选项不会产生额外的成本。标准支持终止时，集群将自动升级到下一个版本。

e. 集群访问权限 – 选择允许还是禁止集群管理员访问权限并选择身份验证模式。启用混合节点功能的集群支持以下身份验证模式。

   i. EKS API：集群将仅从 EKS 访问条目 API 中获取已通过身份验证的 IAM 主体。

   ii. EKS API 和 ConfigMap：集群将从 EKS 访问条目 API 和 `aws-auth` ConfigMap 中获取已通过身份验证的 IAM 主体。

f. Secrets encryption（密钥加密）–（可选）选择此选项以使用 KMS 密钥启用 Kubernetes 密钥的密钥加密。您也可以在创建集群后启用此功能。在启用此功能之前，请确保您熟悉the section called “启用密钥加密”中的信息。

g. ARC 可用区转移：若启用该功能，EKS 会将集群注册到 ARC 可用区转移服务，使您能够借助可用区流量转移能力，将应用流量从某一可用区（AZ）中迁出。

h. Tags（标签）–（可选）向集群添加任何标签。有关更多信息，请参阅 the section called “标记您的资源”。

i. 完成此页面后，请选择下一步。

4. 在 Specify networking (指定联网) 页面上，为以下字段选择值：

   a. VPC – 选择符合the section called “VPC 和子网要求”和 Amazon EKS 混合节点功能要求的现有 VPC。在选择 VPC 之前，我们建议您熟悉“查看有关 VPC、子网和混合节点的 Amazon EKS 联网要求”中的所有要求和注意事项。集群创建后，您无法更改要使用的 VPC。如果没有列出任何 VPC，则需要先创建一个。有关更多信息，请参阅the section called “创建 VPC”和 Amazon EKS 混合节点功能联网要求。

b. Subnets（子网）– 预设情况下，已预先选中在之前字段中指定的 VPC 中的所有可用子网。您必须至少选择两个子网。

c. Security groups（安全组）–（可选）指定您希望 Amazon EKS 将之与其创建的网络接口关联的一个或多个安全组。您指定的安全组中必须至少有一个含有本地节点和（可选）容器组 CIDR 的入站规则。有关更多信息，请参阅 Amazon EKS 混合节点功能联网要求。无论您是否选择任何安全组，Amazon EKS 都会创建一个安全组，以实现集群和 VPC 之间的通信。Amazon EKS 将此安全组以及您选择的任何安全组与它创建的网络接口关联起来。要详细了解 Amazon EKS 创建的集群安全组，请参阅the section called “安全组要求”。您可以修改 Amazon EKS 创建的集群安全组中的规则。

d. 选择集群 IP 地址系列 – 对于启用混合节点功能的集群必须选择 IPv4。

e. （可选）选择配置 Kubernetes 服务 IP 地址范围，然后指定一个服务 IPv4 范围。

f. 选择配置远程网络以启用混合节点，并为混合节点指定本地节点和容器组 CIDR。

g. 如果您的 CNI 在容器组流量离开本地主机时不使用网络地址转换（NATI）或容器组 IP 地址伪装，则必须配置远程容器组 CIDR。如果在混合节点上运行 Webhook，则必须配置远程容器组 CIDR。

h. 本地节点和容器组 CIDR 块必须满足以下要求：

   i. 处于以下 IPv4 RFC–1918 范围之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。

   ii. 不相互重叠，也不与集群的 `VPC CIDR` 或 Kubernetes 服务 IPv4 CIDR 重叠

i. 对于集群端点访问中，选择一个选项。创建集群后，您可以更改此选项。对于启用混合节点功能的集群，您必须选择“公有”或“私有”。在选择非默认选项之前，请务必熟悉这些选项及其影响。有关更多信息，请参阅 the section called “集群终端节点访问”。

j. 完成此页面后，请选择下一步。

5. （可选）在配置可观测性页面上，选择要开启的指标和控制面板日志记录选项。默认情况下，每种日志类型都处于关闭状态。

   a. 有关 Prometheus 指标选项的更多信息，请参阅the section called “Prometheus 指标”。

   b. 有关 EKS 控制面板日志记录选项的更多信息，请参阅the section called “控制面板日志”。

   c. 完成此页面后，请选择下一步。

6. 在 Select add-ons（选择附加组件）页面上，选择要添加到集群的附加组件。

   a. 您可以根据需要选择任意数量的 Amazon EKS 附加组件和 Amazon Marketplace 附加组件。与混合节点功能不兼容的 Amazon EKS 附加组件带有“与混合节点功能不兼容”标志，并且此类附加组件使用反亲和性规则来防止其在混合节点上运行。有关更多信息，请参阅“为混合节点配置附加组件”。如果未列出要安装的 Amazon Marketplace 附加组件，则可以通过在搜索框中输入文本来搜

索可用的 Amazon Marketplace 附加组件。您也可以按 category（类别）、vendor（供应商）或 pricing model（定价模式）进行搜索，然后从搜索结果中选择附加组件。

b. CoreDNS 和 kube-proxy 等附加组件会默认安装。如果您禁用任何默认插件，则可能会影响您运行 Kubernetes 应用程序的能力。

c. 完成此页中步骤后，请选择 Next。

7. 在配置选定插件设置页面上，选择要安装的版本。

a. 创建集群后，您可以随时更新到更高版本。创建集群后，您可以更新每个附加组件的配置。有关配置附加组件的更多信息，请参阅the section called “更新附加组件”。有关兼容混合节点功能的附加组件版本，请参阅the section called “配置附加组件”。

b. 完成此页面后，请选择下一步。

8. 在 Review and create (审核和创建) 页面上，审核您在之前页面输入或选择的信息。如果需要进行更改，请选择 Edit（编辑）。在您感到满意后，选择创建。Status（状态）字段在预置集群时显示 CREATING（正在创建）。集群预配置需要几分钟时间。

9. 继续the section called “第 3 步：更新 kubeconfig”。

第 3 步：更新 kubeconfig

如果您使用 `eksctl` 创建了集群，则可以跳过此步骤。这是因为 `eksctl` 已经为您完成了此步骤。通过向 `kubectl` 文件添加新上下文，从而启用 `kubectl` 与集群之间的通信。有关如何创建和更新文件的更多信息，请参阅 the section called “使用 kubectl 访问集群”。

```
aws eks update-kubeconfig --name CLUSTER_NAME --region AWS_REGION
```

示例输出如下。

```
Added new context arn:aws-cn:eks:AWS_REGION:111122223333:cluster/CLUSTER_NAME to /home/
username/.kube/config
```

通过运行以下命令以确认与集群的通信。

```
kubectl get svc
```

示例输出如下。

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
```

```
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
```

第 4 步：集群设置

然后请参阅the section called “准备集群访问权限”以启用让混合节点能够加入集群的访问权限。

在现有 Amazon EKS 集群上启用混合节点或修改配置

本主题概述了添加、更改或移除 Amazon EKS 集群的混合节点配置时可用的选项，并介绍了相关注意事项。

要让 Amazon EKS 集群能够使用混合节点，请在 RemoteNetworkConfig 配置中添加本地节点的 IP 地址 CIDR 范围以及可选的容器组（pod）网络。EKS 使用此 CIDR 列表来实现集群与本地网络之间的连接。有关更新集群配置时选项的完整列表，请参阅《Amazon EKS API Reference》中的 UpdateClusterConfig。

您可以对集群中的 EKS 混合节点网络配置执行以下任一操作：

- 添加远程网络配置，以在现有集群中启用 EKS 混合节点。
- 在现有集群中添加、更改或移除远程节点网络或远程容器组（pod）网络。
- 移除所有远程节点网络 CIDR 范围，以在现有集群中禁用 EKS 混合节点。

先决条件

- 在为混合节点启用 Amazon EKS 集群之前，请确保您的环境满足 the section called “先决条件” 中概述的要求以及 the section called “准备联网”、the section called “准备操作系统” 和 the section called “准备凭证” 中详细介绍的要求。
- 集群必须使用 IPv4 地址系列。
- 集群的集群身份验证模式必须为 API 或 API_AND_CONFIG_MAP。the section called “身份验证模式” 中介绍了修改集群身份验证模式的流程。
- 我们建议为 Amazon EKS Kubernetes API 服务器端点使用公有或私有端点访问权限，但不要同时启用这两者。如果您选择“公有和私有”，Amazon EKS Kubernetes API 服务器端点将始终解析为在您的 VPC 之外运行的混合节点公有 IP，这可能会阻止您的混合节点加入集群。the section called “集群终端节点访问” 中介绍了修改集群网络访问权限的流程。
- 已在您的设备上安装并配置了最新版本的 Amazon 命令行界面（Amazon CLI）。要查看当前版本，请使用 aws --version。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装或更新到最新版本的 Amazon CLI 和配置 Amazon CLI 设置。

- 有权限在您的 Amazon EKS 集群上调用 UpdateClusterConfig 的 IAM 主体。
- 将附加组件更新到兼容混合节点的版本。有关兼容混合节点功能的附加组件版本，请参阅the section called “配置附加组件”。
- 如果您正在运行与混合节点不兼容的附加组件，请确保附加组件 DaemonSet 或 Deployment 具有以下亲和性规则来防止部署到混合节点。如果以下亲和性规则尚不存在，请添加该规则。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

注意事项

`remoteNetworkConfig` JSON 对象在更新期间具有以下行为：

- 您未指定的任何现有配置部分均保持不变。如果您未指定 `remoteNodeNetworks` 或 `remotePodNetworks` 中的任何一个，则该部分将保持不变。
- 如果要修改 CIDR 的 `remoteNodeNetworks` 或 `remotePodNetworks` 列表，则必须在最终配置中指定所需的 CIDR 的完整列表。如果您指定了对 `remoteNodeNetworks` 或 `remotePodNetworks` CIDR 列表的更改，EKS 将在更新期间替换原始列表。
- 本地节点和容器组 CIDR 块必须满足以下要求：
  1. 在 IPv4 RFC-1918 范围之一内：10.0.0.0/8、172.16.0.0/12 或 192.168.0.0/16。
  2. 不相互重叠，也不与 Amazon EKS 集群的所有 VPC CIDR 或 Kubernetes 服务 IPv4 CIDR 重叠。

在现有集群上启用混合节点

您可以使用以下方法在现有集群中启用 EKS 混合节点：

- Amazon CloudFormation
- Amazon CLI
- Amazon Web Services 管理控制台

在现有集群中启用 EKS 混合节点 – Amazon CloudFormation

1. 要在集群中启用 EKS 混合节点，请将 RemoteNodeNetwork 和（可选）RemotePodNetwork 添加到您的 CloudFormation 模板并更新堆栈。请注意，RemoteNodeNetwork 列表最多包含一个 Cidrs 项目，而 Cidrs 列表可包含多个 IP CIDR 范围。

```
RemoteNetworkConfig:
  RemoteNodeNetworks:
    - Cidrs: [RemoteNodeCIDR]
  RemotePodNetworks:
    - Cidrs: [RemotePodCIDR]
```

2. 继续the section called “准备集群访问权限”。

在现有集群中启用 EKS 混合节点 – Amazon CLI

1. 运行以下命令，为 EKS 集群启用 EKS 混合节点的 RemoteNetworkConfig。在运行此命令之前，请将以下参数替换为相应的设置。有关设置的完整列表，请参阅《Amazon EKS API Reference》中的 UpdateClusterConfig。

   a. CLUSTER_NAME：要更新的 EKS 集群的名称。

   b. AWS_REGION：EKS 集群正在运行的 Amazon 区域。

   c. REMOTE_NODE_CIDRS：混合节点的本地节点 CIDR。

   d. REMOTE_POD_CIDRS（可选）：在混合节点上运行的工作负载的本地容器组 CIDR。

```
aws eks update-cluster-config \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. 更新集群需要几分钟时间。可使用以下命令查询集群的状态。请将 CLUSTER_NAME 替换为要修改的集群的名称，并将 AWS_REGION 替换为运行集群的 Amazon 区域。在返回的输出为 ACTIVE 之前，请勿继续执行下一步。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

3. 继续the section called “准备集群访问权限”。

在现有集群中启用 EKS 混合节点 – Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择集群的名称可以显示集群信息。
3. 选择联网选项卡，然后选择更新。
4. 在下拉列表中，选择远程网络。
5. 选择配置远程网络以启用混合节点，并为混合节点指定本地节点和容器组 CIDR。
6. 选择 Save changes（保存更改）以完成操作。等待集群状态恢复为活动。
7. 继续the section called “准备集群访问权限”。

更新现有集群中的混合节点配置

您可以使用以下任何一种方法修改现有混合集群中的 `remoteNetworkConfig`：

- Amazon CloudFormation
- Amazon CLI
- Amazon Web Services 管理控制台

更新现有集群中的混合配置 – Amazon CloudFormation

1. 使用新的网络 CIDR 值更新您的 CloudFormation 模板。

```
RemoteNetworkConfig:
  RemoteNodeNetworks:
    - Cidrs: [NEW_REMOTE_NODE_CIDRS]
  RemotePodNetworks:
    - Cidrs: [NEW_REMOTE_POD_CIDRS]
```

> **Note**
>
> 更新 `RemoteNodeNetworks` 或 `RemotePodNetworks` CIDR 列表时，请包含全部 CIDR（新的和现有的）。更新期间，EKS 会替换整个列表。在更新请求中省略这些字段会保留其现有配置。

2. 使用修改后的模板更新您的 CloudFormation 堆栈，然后等待堆栈更新完成。

更新现有集群中的混合配置 – Amazon CLI

1. 要修改远程网络 CIDR，请运行以下命令。将这些值替换为相应的设置：

```
aws eks update-cluster-config
--name CLUSTER_NAME
--region AWS_REGION
--remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["NEW_REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":
["NEW_REMOTE_POD_CIDRS"]}]}'
```

> **Note**
>
> 更新 `remoteNodeNetworks` 或 `remotePodNetworks` CIDR 列表时，请包含全部 CIDR（新的和现有的）。更新期间，EKS 会替换整个列表。在更新请求中省略这些字段会保留其现有配置。

2. 等待集群状态恢复为“ACTIVE”，然后再继续。

更新现有集群中的混合配置 – Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择集群的名称可以显示集群信息。
3. 选择联网选项卡，然后选择更新。
4. 在下拉列表中，选择远程网络。
5. 根据需要更新 `Remote node networks` 和 `Remote pod networks - Optional` 下方的 CIDR。
6. 选择保存更改，然后等待集群状态恢复为活动。

在现有集群中禁用混合节点

您可以使用以下方法在现有集群中禁用 EKS 混合节点：

- Amazon CloudFormation
- Amazon CLI

- Amazon Web Services 管理控制台

在现有集群中禁用 EKS 混合节点 – Amazon CloudFormation

1. 要在集群中禁用 EKS 混合节点，请在 CloudFormation 模板中将 RemoteNodeNetworks 和 RemotePodNetworks 设置为空数组，然后更新堆栈。

```
RemoteNetworkConfig:
  RemoteNodeNetworks: []
  RemotePodNetworks: []
```

在现有集群中禁用 EKS 混合节点 – Amazon CLI

1. 运行以下命令，从 EKS 集群中移除 RemoteNetworkConfig。在运行此命令之前，请将以下参数替换为相应的设置。有关设置的完整列表，请参阅《Amazon EKS API Reference》中的 UpdateClusterConfig。
   a. CLUSTER_NAME：要更新的 EKS 集群的名称。
   b. AWS_REGION：EKS 集群正在运行的 Amazon 区域。

```
aws eks update-cluster-config \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --remote-network-config '{"remoteNodeNetworks":[],"remotePodNetworks":[]}'
```

2. 更新集群需要几分钟时间。可使用以下命令查询集群的状态。请将 CLUSTER_NAME 替换为要修改的集群的名称，并将 AWS_REGION 替换为运行集群的 Amazon 区域。在返回的输出为 ACTIVE 之前，请勿继续执行下一步。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

在现有集群中禁用 EKS 混合节点 – Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择集群的名称可以显示集群信息。

3. 选择联网选项卡，然后选择更新。
4. 在下拉列表中，选择远程网络。
5. 选择配置远程网络以启用混合节点，然后移除 `Remote node networks` 和 `Remote pod networks - Optional` 下的所有 CIDR。
6. 选择 Save changes（保存更改）以完成操作。等待集群状态恢复为活动。

准备混合节点的集群访问权限

在将混合节点连接到 Amazon EKS 集群之前，您必须为混合节点 IAM 角色授予加入集群的 Kubernetes 权限。有关如何创建混合节点 IAM 角色的信息，请参阅the section called “准备凭证”。Amazon EKS 支持通过两种方式将 IAM 主体关联到 Kubernetes 基于角色的访问控制（RBAC）、Amazon EKS 访问条目和 `aws-auth` ConfigMap。有关 Amazon EKS 访问权限管理的更多信息，请参阅the section called “Kubernetes API 访问权限”。

按照以下过程将混合节点 IAM 角色与 Kubernetes 权限关联起来。要使用 Amazon EKS 访问条目，集群必须是使用 `API` 或 `API_AND_CONFIG_MAP` 身份验证模式创建的。要使用 `aws-auth` ConfigMap，集群必须是使用 `API_AND_CONFIG_MAP` 身份验证模式创建的。启用混合节点功能的 Amazon EKS 集群不支持仅 `CONFIG_MAP` 身份验证模式。

将 Amazon EKS 访问条目用于混合节点 IAM 角色

一种名为 HYBRID_LINUX 的混合节点 Amazon EKS 访问条目类型可以与 IAM 角色一起使用。使用此访问条目类型，用户名将被自动设置为 system:node:{{SessionName}}。有关创建访问条目的更多信息，请参阅the section called “创建访问条目”。

Amazon CLI

1. 您必须已在设备上安装和配置最新版本的 Amazon CLI。要查看当前版本，请使用 `aws --version`。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的“安装”和“使用 aws configure 快速配置”。
2. 使用以下命令创建访问条目。请将 CLUSTER_NAME 替换为集群的名称，将 HYBRID_NODES_ROLE_ARN 替换为您在the section called “准备凭证”步骤中创建的角色的 ARN。

```
aws eks create-access-entry --cluster-name CLUSTER_NAME \
    --principal-arn HYBRID_NODES_ROLE_ARN \
```

```
    --type HYBRID_LINUX
```

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择已启用混合节点功能的集群的名称。
3. 选择访问选项卡。
4. 选择创建访问条目。
5. 对于 IAM 主体，请选择您在the section called “准备凭证”步骤中创建的混合节点 IAM 角色。
6. 对于类型，选择 Hybrid Linux。
7. （可选）对于标签，为访问条目分配标签。例如，为了更轻松地查找所有具有相同标签的资源。
8. 选择跳过以检查并创建。不能向 Hybrid Linux 访问条目添加策略或更改其访问权限范围。
9. 查看访问条目的配置。如果有任何内容看起来不正确，请选择上一步以返回步骤并更正错误。如果配置正确，请选择创建。

将 aws-auth ConfigMap 用于混合节点 IAM 角色

在以下步骤中，您将使用您在the section called “准备凭证”步骤中创建的混合节点 IAM 角色的 ARN 来创建或更新 `aws-auth` ConfigMap。

1. 检查您的集群是否已有现有的 `aws-auth` ConfigMap。请注意如果您使用特定的 `kubeconfig` 文件，请使用 `--kubeconfig` 标志。

```
kubectl describe configmap -n kube-system aws-auth
```

2. 如果您看到有 `aws-auth` ConfigMap，则请根据需要对其进行更新。

   a. 打开该 ConfigMap 进行编辑。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 根据需要添加新的 `mapRoles` 条目。将 `HYBRID_NODES_ROLE_ARN` 替换为混合节点 IAM 角色的 ARN。注意，`{{SessionName}}` 是保存在 ConfigMap 中的正确模板格式。请勿将其替换为其他值。

```
data:
```

```
  mapRoles: |
  - groups:
    - system:bootstrappers
    - system:nodes
    rolearn: HYBRID_NODES_ROLE_ARN
    username: system:node:{{SessionName}}
```

c. 保存文件并退出文本编辑器。

3. 如果集群没有现有的 aws-auth ConfigMap，请使用以下命令创建一个。将 HYBRID_NODES_ROLE_ARN 替换为混合节点 IAM 角色的 ARN。注意，{{SessionName}} 是保存在 ConfigMap 中的正确模板格式。请勿将其替换为其他值。

```
kubectl apply -f=/dev/stdin <<-EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
  - groups:
    - system:bootstrappers
    - system:nodes
    rolearn: HYBRID_NODES_ROLE_ARN
    username: system:node:{{SessionName}}
EOF
```

在混合节点上运行本地工作负载

在启用了混合节点功能的 EKS 集群中，您可以使用与 Amazon 云端相同的 Amazon EKS 集群、功能和工具，在自己的基础设施上运行本地和边缘应用程序。

以下章节包含有关使用混合节点的分步指导。

主题

- 连接混合节点
- 连接使用 Bottlerocket 的混合节点
- 升级集群的混合节点
- 混合节点的安全更新修补

- 移除混合节点

连接混合节点

> **Note**
>
> 以下步骤适用于运行除 Bottlerocket 以外的兼容操作系统的混合节点。有关连接运行 Bottlerocket 的混合节点的步骤，请参阅the section called “连接使用 Bottlerocket 的混合节点”。

本主题介绍如何将混合节点连接到 Amazon EKS 集群。混合节点加入集群后，将在 Amazon EKS 控制台和兼容 Kubernetes 的工具（例如 kubectl）中显示并且状态为“未就绪”。完成本页介绍的步骤后，the section called “配置 CNI” 以确保混合节点可以随时用于运行应用程序。

先决条件

在将混合节点连接到 Amazon EKS 集群之前，请确保您已完成先决步骤。

- 您已建立从本地环境到托管 Amazon EKS 集群的 Amazon 区域的连接。请参阅the section called “准备联网”了解更多信息。
- 您已在本地主机上安装了适用于混合节点的兼容操作系统。请参阅the section called “准备操作系统”了解更多信息。
- 您已经创建了混合节点 IAM 角色并设置了本地凭证提供者（Amazon Systems Manager 混合激活或 Amazon IAM Roles Anywhere）。请参阅the section called “准备凭证”了解更多信息。
- 您已经创建启用了混合节点的 Amazon EKS 集群。请参阅the section called “创建集群”了解更多信息。
- 您已为您的混合节点 IAM 角色关联了 Kubernetes 基于角色的访问控制（RBAC）权限。请参阅the section called “准备集群访问权限”了解更多信息。

第 1 步：在每台本地主机上安装混合节点 CLI (**nodeadm**)

如果您预构建的操作系统映像已经包含 Amazon EKS 混合节点功能 CLI (nodeadm)，则可以跳过此步骤。有关 nodeadm 混合节点版本的更多信息，请参阅the section called “混合节点 nodeadm”。

nodeadm 的混合节点版本在 Amazon S3 中托管，由 Amazon CloudFront 分发。要在每台本地主机上安装 nodeadm，您可以在本地主机上运行以下命令。

x86_64 主机：

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/
nodeadm'
```

ARM 主机

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/
nodeadm'
```

将可执行文件权限添加到每台主机上下载的二进制文件。

```
chmod +x nodeadm
```

第 2 步：使用 **nodeadm** 安装混合节点依赖项

如果在预构建的操作系统映像中安装混合节点依赖项，则可以跳过此步骤。`nodeadm install` 命令可用于安装混合节点必需的所有依赖项。混合节点依赖项包括 containerd、kubelet、kubectl 以及 Amazon SSM 或 Amazon IAM Roles Anywhere 组件。要详细了解 the section called “混合节点 nodeadm” 安装的组件和文件位置，请参阅`nodeadm install`。要详细了解必须在本地防火墙中为 `nodeadm install` 进程允许的域，请参阅the section called “准备联网”。

运行以下命令，从而在本地主机上安装混合节点依赖项。运行以下命令的用户必须具有您主机上的 sudo/root 访问权限。

> **Important**
>
> 运行混合节点 CLI (nodeadm) 的用户必须具有您主机上的 sudo/root 访问权限。

- 请 `K8S_VERSION` 替换为 Amazon EKS 集群的 Kubernetes 次要版本，例如 `1.31`。有关支持的 Kubernetes 版本的列表，请参阅 Amazon EKS 支持的版本。
- 请将 `CREDS_PROVIDER` 替换为您使用的本地凭证提供者。Amazon SSM 的有效值为 `ssm`，Amazon IAM Roles Anywhere 的有效值为 `iam-ra`。

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER
```

第 3 步：将混合节点连接到集群

在将混合节点连接到集群之前，请确保您的本地防火墙和集群安全组中已经允许在 Amazon EKS 控制面板与混合节点之间进行通信所必需的权限。此步骤中出现的大多数问题都与防火墙配置、安全组配置或混合节点 IAM 角色配置有关。

> **Important**
>
> 运行混合节点 CLI (nodeadm) 的用户必须具有您主机上的 sudo/root 访问权限。

1. 在每台主机上创建一个包含适用于部署的值的 `nodeConfig.yaml` 文件。有关可用配置设置的完整说明，请参阅the section called “混合节点 nodeadm”。如果混合节点 IAM 角色没有执行 `eks:DescribeCluster` 操作的权限，则必须在 `nodeConfig.yaml` 的集群部分中传递 Kubernetes API 端点、集群 CA 捆绑和 Kubernetes 服务 IPv4 CIDR。
   a. 如果您的本地凭证提供者为 Amazon SSM 混合激活，请使用以下 `nodeConfig.yaml` 示例。
      i. 将 `CLUSTER_NAME` 替换为您的集群名称。
      ii. 请将 Amazon 替换为集群所在的 `AWS_REGION` 区域。例如 `us-west-2`。
      iii. 请将 `ACTIVATION_CODE` 替换为创建 Amazon SSM 混合激活时收到的激活码。请参阅the section called “准备凭证”了解更多信息。
      iv. 请将 `ACTIVATION_ID` 替换为创建 Amazon SSM 混合激活时收到的激活 ID。您可以通过 Amazon Systems Manager 控制台或 Amazon CLI `aws ssm describe-activations` 命令查看此信息。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    ssm:
      activationCode: ACTIVATION_CODE
      activationId: ACTIVATION_ID
```

   b. 如果您的本地凭证提供者为 Amazon IAM Roles Anywhere，请使用以下 `nodeConfig.yaml` 示例。
      i. 将 `CLUSTER_NAME` 替换为您的集群名称。

ii. 请将 Amazon 替换为集群所在的 AWS_REGION 区域。例如 us-west-2。

iii. 请将 NODE_NAME 替换为节点的名称。如果您为混合节点 IAM 角色配置的信任策略使用 "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}" 资源条件，则节点名称必须与主机上证书的 CN 一致。您使用的 nodeName 长度不得超过 64 个字符。

iv. 请将 TRUST_ANCHOR_ARN 替换为您在“准备用于混合节点的凭证”步骤中配置的信任锚的 ARN。

v. 请将 PROFILE_ARN 替换为您在“the section called “准备凭证””步骤中配置的信任锚的 ARN。

vi. 将 ROLE_ARN 替换为混合节点 IAM 角色的 ARN。

vii. 请将 CERTIFICATE_PATH 替换为磁盘中指向节点证书的路径。如果不指定此参数，则默认值为 /etc/iam/pki/server.pem。

viii. 请将 KEY_PATH 替换为磁盘中指向证书私有密钥的路径。如果不指定此参数，则默认值为 /etc/iam/pki/server.key。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    iamRolesAnywhere:
      nodeName: NODE_NAME
      trustAnchorArn: TRUST_ANCHOR_ARN
      profileArn: PROFILE_ARN
      roleArn: ROLE_ARN
      certificatePath: CERTIFICATE_PATH
      privateKeyPath: KEY_PATH
```

2. 使用您的 nodeConfig.yaml 运行 nodeadm init 命令，从而将混合节点连接到 Amazon EKS 集群。

```
nodeadm init -c file://nodeConfig.yaml
```

如果上述命令成功完成，则表示混合节点已加入您的 Amazon EKS 集群。您可以在 Amazon EKS 控制台中导航到集群的“计算”选项卡（确保 IAM 主体具有查看权限）或使用 kubectl get nodes 来验证此信息。

> **Important**
>
> 您的节点将处于 `Not Ready` 状态，此状态符合预期，是因混合节点上缺少正在运行的 CNI 所致。如果节点未加入集群，请参阅the section called “问题排查”。

第 4 步：为混合节点配置 CNI

要使混合节点能够随时运行应用程序，请继续完成the section called “配置 CNI”中的步骤。

连接使用 Bottlerocket 的混合节点

本主题介绍如何将运行 Bottlerocket 的混合节点连接到 Amazon EKS 集群。Bottlerocket 是由 Amazon 赞助并提供支持的开源 Linux 发行版。Bottlerocket 专为托管容器工作负载而构建。您可以借助 Bottlerocket 实现容器基础设施自动更新，由此提高容器化部署的可用性，降低运营成本。Bottlerocket 仅包含运行容器所需的必备软件，可提高资源利用率、减少安全威胁并降低管理开销。

仅支持将 Bottlerocket 版本 v1.37.0 及更高版本的 VMware 变体与 EKS 混合节点结合使用。Bottlerocket 的 VMware 变体支持 Kubernetes 版本 1.28 及更高版本。这些变体的操作系统映像包括 kubelet、containerd、aws-iam-authenticator 以及适用于 EKS 混合节点功能的其他软件先决条件。您可以使用 Bottlerocket 设置文件来配置这些组件，该文件包含用于 Bottlerocket 引导容器和管理容器的 base64 编码用户数据。通过配置这些设置，可让 Bottlerocket 使用您的混合节点凭证提供程序来完成集群对混合节点的身份验证。混合节点加入集群后，将在 Amazon EKS 控制台以及 `kubectl` 等兼容 Kubernetes 的工具中显示并且状态为 `Not Ready`。完成本页介绍的步骤后，the section called “配置 CNI” 以确保混合节点可以随时用于运行应用程序。

先决条件

在将混合节点连接到 Amazon EKS 集群之前，请确保您已完成先决步骤。

- 您已建立从本地环境到托管 Amazon EKS 集群的 Amazon 区域的连接。请参阅the section called “准备联网”了解更多信息。
- 您已经创建了混合节点 IAM 角色并设置了本地凭证提供者（Amazon Systems Manager 混合激活或 Amazon IAM Roles Anywhere）。请参阅the section called “准备凭证”了解更多信息。
- 您已经创建启用了混合节点的 Amazon EKS 集群。请参阅the section called “创建集群”了解更多信息。
- 您已为您的混合节点 IAM 角色关联了 Kubernetes 基于角色的访问控制（RBAC）权限。请参阅the section called “准备集群访问权限”了解更多信息。

第 1 步：创建 Bottlerocket 设置 TOML 文件

要为混合节点配置 Bottlerocket，您需要创建一个包含必要配置的 `settings.toml` 文件。TOML 文件的内容因您使用的凭证提供程序（SSM 或 IAM Roles Anywhere）而异。在预置 Bottlerocket 实例时，此文件将作为用户数据传递。

> **Note**
>
> 下方提供的 TOML 文件仅包含将 Bottlerocket VMWare 机器初始化为 EKS 集群节点所需的最低配置项。Bottlerocket 提供了丰富的配置项，可满足多种不同的使用案例。因此，若需配置混合节点初始化之外的更多功能，请查阅 Bottlerocket 文档，获取所使用版本的完整配置项列表（例如，此处列出了 Bottlerocket 1.51.x 版本的全部可用配置项）。

SSM

如果将 Amazon Systems Manager 作为凭证提供程序，请创建一个包含以下内容的 `settings.toml` 文件：

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"
cloud-provider = ""
server-tls-bootstrap = true

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"

[settings.kubernetes.credential-providers.ecr-credential-provider]
enabled = true
cache-duration = "12h"
image-patterns = [
    "*.dkr.ecr.*.amazonaws.com",
    "*.dkr.ecr.*.amazonaws.com.cn",
    "*.dkr.ecr.*.amazonaws.eu",
```

```
    "*.dkr.ecr-fips.*.amazonaws.com",
    "*.dkr.ecr-fips.*.amazonaws.eu",
    "public.ecr.aws"
]

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "ssm"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"

[settings.host-containers.control]
enabled = true
```

将占位符替换为以下值：

- <cluster-name>：Amazon EKS 集群的名称。
- <api-server-endpoint>：集群的 API 服务器端点。
- <cluster-certificate-authority>：集群的 base64 编码 CA 捆绑包。
- <region>：托管集群的 Amazon 区域，例如“us-east-1”。
- <hostname>：Bottlerocket 实例的主机名，这也将被配置为节点名称。这可以是您选择的任何唯一值，但必须符合 Kubernetes 对象命名惯例。此外，使用的主机名称长度不能超过 64 个字符。注意：使用 SSM 提供程序时，在将实例注册到 SSM 后，该主机名和节点名称将替换为托管实例 ID（例如，mi-* ID）。
- <base64-encoded-admin-container-userdata>：Bottlerocket 管理容器配置的 base64 编码内容。启用管理容器后，您将能够通过 SSH 连接到 Bottlerocket 实例，以便进行系统探索和调试。虽然这不是必需设置，但为了便于故障排除，我们建议将其启用。有关使用管理容器进行身份验证的更多信息，请参阅 Bottlerocket 管理容器文档。例如，管理容器接受 JSON 格式的 SSH 用户和密钥输入。

```
{
  "user": "<ssh-user>",
  "ssh": {
```

```
    "authorized-keys": [
      "<ssh-authorized-key>"
    ]
  }
}
```

- `<base64-encoded-bootstrap-container-userdata>`：Bottlerocket 引导容器配置的 base64 编码内容。有关其配置的更多信息，请参阅 Bottlerocket 引导容器文档。引导容器负责将实例注册为 Amazon SSM 托管式实例，并将其作为 Kubernetes 节点加入 Amazon EKS 集群。传递到引导容器的用户数据采用命令调用的形式，这会将您之前创建的 SSM 混合激活码和 ID 作为输入接受：

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-code=<activation-
code> --region=<region>
```

IAM Roles Anywhere

如果将 Amazon IAM Roles Anywhere 作为凭证提供程序，请创建一个包含以下内容的 `settings.toml` 文件：

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"
cloud-provider = ""
server-tls-bootstrap = true

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"
config = "<base64-encoded-aws-config-file>"

[settings.kubernetes.credential-providers.ecr-credential-provider]
enabled = true
cache-duration = "12h"
image-patterns = [
```

```
    "*.dkr.ecr.*.amazonaws.com",
    "*.dkr.ecr.*.amazonaws.com.cn",
    "*.dkr.ecr.*.amazonaws.eu",
    "*.dkr.ecr-fips.*.amazonaws.com",
    "*.dkr.ecr-fips.*.amazonaws.eu",
    "public.ecr.aws"
]

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "iam-ra"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"
```

将占位符替换为以下值：

- <cluster-name>：Amazon EKS 集群的名称。
- <api-server-endpoint>：集群的 API 服务器端点。
- <cluster-certificate-authority>：集群的 base64 编码 CA 捆绑包。
- <region>：托管集群的 Amazon 区域，例如“us-east-1”
- <hostname>：Bottlerocket 实例的主机名，这也将被配置为节点名称。这可以是您选择的任何唯一值，但必须符合 Kubernetes 对象命名惯例。此外，使用的主机名称长度不能超过 64 个字符。注意：使用 IAM-RA 提供程序时，如果您为混合节点 IAM 角色配置的信任策略使用 "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}" 资源条件，则节点名称必须与主机上证书的 CN 一致。
- <base64-encoded-aws-config-file>：Amazon 配置文件的 Base64 编码内容。该文件的内容应会如下所示：

```
[default]
credential_process = aws_signing_helper credential-process --certificate /root/.aws/
node.crt --private-key /root/.aws/node.key --profile-arn <profile-arn> --role-arn
 <role-arn> --trust-anchor-arn <trust-anchor-arn> --role-session-name <role-session-
name>
```

- <base64-encoded-admin-container-userdata>：Bottlerocket 管理容器配置的 base64 编码内容。启用管理容器后，您将能够通过 SSH 连接到 Bottlerocket 实例，以便进行系统探索和调试。虽然这不是必需设置，但为了便于故障排除，我们建议将其启用。有关使用管理容器进行身份验证的更多信息，请参阅 Bottlerocket 管理容器文档。例如，管理容器接受 JSON 格式的 SSH 用户和密钥输入。

```
{
  "user": "<ssh-user>",
  "ssh": {
    "authorized-keys": [
      "<ssh-authorized-key>"
    ]
  }
}
```

- <base64-encoded-bootstrap-container-userdata>：Bottlerocket 引导容器配置的 base64 编码内容。有关其配置的更多信息，请参阅 Bottlerocket 引导容器文档。引导容器负责在实例上创建 IAM Roles Anywhere 主机证书和证书私有密钥文件。然后，aws_signing_helper 将会使用这些证书和文件来获取临时证书，以用于 Amazon EKS 集群身份验证。传递到引导容器的用户数据采用命令调用的形式，这会将您之前创建的证书和私有密钥内容作为输入接受：

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key>
```

第 2 步：预置 Bottlerocket vSphere 虚拟机及用户数据

构造 TOML 文件后，在创建 vSphere 虚拟机期间将其传递为用户数据。请注意，在虚拟机首次开机之前，必须配置用户数据。因此，您需要在创建实例时提供用户数据；如果您希望先行创建虚拟机，则在您为其配置用户数据前，虚拟机必须处于关机状态。例如，假设使用 govc CLI：

首次创建虚拟机

```
govc vm.create \
  -on=true \
  -c=2 \
  -m=4096 \
  -net.adapter=<network-adapter> \
  -net=<network-name> \
  -e guestinfo.userdata.encoding="base64" \
```

```
  -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
  -template=<template-name> \
  <vm-name>
```

更新现有虚拟机的用户数据

```
govc vm.create \
    -on=false \
    -c=2 \
    -m=4096 \
    -net.adapter=<network-adapter> \
    -net=<network-name> \
    -template=<template-name> \
    <vm-name>

govc vm.change
    -vm <vm-name> \
    -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
    -e guestinfo.userdata.encoding="base64"

govc vm.power -on <vm-name>
```

在以上章节中，`-e guestinfo.userdata.encoding="base64"` 选项用于指定用户数据采用 base64 编码。`-e guestinfo.userdata` 选项将 `settings.toml` 文件的 base64 编码内容作为用户数据传递给 Bottlerocket 实例。将占位符替换为您的具体值，例如 Bottlerocket OVA 模板和联网详细信息。

第 3 步：验证混合节点连接

Bottlerocket 实例启动后，将会尝试加入您的 Amazon EKS 集群。您可以在 Amazon EKS 控制台中导航到集群的“计算”选项卡或运行以下命令，从而验证连接：

```
kubectl get nodes
```

> **Important**
>
> 您的节点将处于 `Not Ready` 状态，此状态符合预期，是因混合节点上缺少正在运行的 CNI 所致。如果节点未加入集群，请参阅the section called “问题排查”。

第 4 步：为混合节点配置 CNI

要使混合节点能够随时运行应用程序，请继续完成the section called “配置 CNI”中的步骤。

升级集群的混合节点

升级混合节点的指南与在 Amazon EC2 中运行的自主管理型 Amazon EKS 节点类似。我们建议您在目标 Kubernetes 版本上创建新的混合节点，从容地将现有应用程序迁移到新 Kubernetes 版本上的混合节点，然后再从集群中移除旧 Kubernetes 版本上的混合节点。在开始升级之前，请务必检查 Amazon EKS 升级最佳实践。Amazon EKS 混合节点功能支持的 Kubernetes 版本与具有云节点的 Amazon EKS 集群相同，包括标准和扩展支持。

Amazon EKS 混合节点功能使用的节点版本偏斜策略与上游 Kubernetes 相同。Amazon EKS 混合节点功能不能使用比 Amazon EKS 控制面板更高的版本，混合节点的 Kubernetes 次要版本最多可比 Amazon EKS 控制面板早三个次要版本。

如果您没有剩余容量可用来创建具有目标 Kubernetes 版本的新混合节点，以满足割接迁移升级策略的需要，则可以使用 Amazon EKS 混合节点 CLI (`nodeadm`) 就地升级混合节点的 Kubernetes 版本。

> **Important**
>
> 如果您使用 `nodeadm` 就地升级混合节点，则在关闭旧版本的 Kubernetes 组件以及安装和启动新的 Kubernetes 版本组件的过程中，节点会有停机时间。

先决条件

在升级之前，请确保您已满足以下先决条件。

- 升级混合节点的目标 Kubernetes 版本必须等于或早于 Amazon EKS 控制面板的版本。
- 如果您遵循割接迁移升级策略，则在目标 Kubernetes 版本上安装的新混合节点必须满足the section called “先决条件”要求。这包括拥有在创建 Amazon EKS 集群期间传递的远程节点网络 CIDR 中的 IP 地址。
- 无论是割接迁移升级还是就地升级，混合节点必须有权访问所需的域才能拉取混合节点依赖项的新版本。
- 您必须已在本地计算机或用于与 Amazon EKS Kubernetes API 端点进行交互的实例上安装 kubectl。
- 您的 CNI 版本必须支持要升级到的 Kubernetes 版本。如果不支持，请在升级混合节点之前先升级 CNI 版本。请参阅the section called “配置 CNI”了解更多信息。

割接迁移（蓝绿迁移）升级

割接迁移升级是指首先在新主机上创建具有目标 Kubernetes 版本的新混合节点，从容地将现有应用程序迁移到目标 Kubernetes 版本的新混合节点，然后再从集群中移除旧 Kubernetes 版本的混合节点的过程。这种策略也称为蓝绿迁移。

1. 按照 the section called “连接混合节点” 步骤将新主机作为混合节点连接。运行 `nodeadm install` 命令时，请使用目标 Kubernetes 版本。
2. 启用目标 Kubernetes 版本的新混合节点与旧 Kubernetes 版本的混合节点之间的通信。此配置让容器组可在将工作负载迁移到目标 Kubernetes 版本的混合节点后进行相互通信。
3. 确认目标 Kubernetes 版本的混合节点已成功加入集群并处于“就绪”状态。
4. 使用以下命令，将要移除的每个节点标记为不可调度。这样可确保新容器组不会在要替换的节点上调度或重新调度。有关更多信息，请参阅 Kubernetes 文档中的 kubectl cordon。请将 `NODE_NAME` 替换为旧 Kubernetes 版本混合节点的名称。

   ```
   kubectl cordon NODE_NAME
   ```

   您可以使用以下代码段标识并封锁使用特定 Kubernetes 版本（此示例中为版本 `1.28`）的所有节点。

   ```
   K8S_VERSION=1.28
   for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
    "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
    match("\($K8S_VERSION)")).metadata.name')
   do
       echo "Cordoning $node"
       kubectl cordon $node
   done
   ```

5. 如果混合节点上当前部署运行的 CoreDNS 副本少于 2 个，请将部署横向扩展到至少 2 个副本。我们建议您在混合节点上运行至少两个 CoreDNS 副本，以确保正常运行期间的恢复能力。

   ```
   kubectl scale deployments/coredns --replicas=2 -n kube-system
   ```

6. 使用以下命令清空要从集群中删除的每个使用旧版本 Kubernetes 的混合节点。有关清空节点的更多信息，请参阅 Kubernetes 文档中的安全地清空一个节点。请将 `NODE_NAME` 替换为旧 Kubernetes 版本混合节点的名称。

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

您可以使用以下代码段标识并清空特定 Kubernetes 版本（此示例中为版本 1.28）的所有节点。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
 "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
 match("\($K8S_VERSION)")).metadata.name')
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-emptydir-data
done
```

7. 您可以使用 nodeadm 停止混合节点构件并将其从主机中移除。您必须使用具有 root/sudo 权限的用户运行 nodeadm。默认情况下，如果节点上还有容器组，则 nodeadm uninstall 不会执行。有关更多信息，请参阅 the section called “混合节点 nodeadm”。

```
nodeadm uninstall
```

8. 停止并卸载混合节点构件后，从集群中移除节点资源。

```
kubectl delete node node-name
```

您可以使用以下代码段标识并删除特定 Kubernetes 版本（此示例中为版本 1.28）的所有节点。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
 "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
 match("\($K8S_VERSION)")).metadata.name')
do
    echo "Deleting $node"
    kubectl delete node $node
done
```

9. 运行上述步骤后，您的混合节点上可能会剩下一些构件，具体取决于您选择的 CNI。请参阅the section called “配置 CNI”了解更多信息。

就地升级

就地升级过程是指使用 `nodeadm upgrade` 升级混合节点的 Kubernetes 版本，而不使用新的物理或虚拟主机和割接迁移策略。`nodeadm upgrade` 进程会关闭在混合节点上运行的现有较早版本的 Kubernetes 组件，卸载现有较早版本的 Kubernetes 组件，安装新的目标 Kubernetes 组件，然后启动新的目标 Kubernetes 组件。强烈建议一次升级一个节点，以尽可能减少对混合节点上运行的应用程序的影响。此过程的持续时间取决于网络带宽和延迟。

1. 使用以下命令将要升级的节点标记为不可调度。这样可确保新容器组不会在要升级的节点上调度或重新调度。有关更多信息，请参阅 Kubernetes 文档中的 kubectl cordon。请将 `NODE_NAME` 替换为要升级的混合节点的名称

   ```
   kubectl cordon NODE_NAME
   ```

2. 使用以下命令清空您要升级的节点。有关清空节点的更多信息，请参阅 Kubernetes 文档中的安全地清空一个节点。请将 `NODE_NAME` 替换为要升级的混合节点的名称。

   ```
   kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
   ```

3. 在要升级的混合节点上运行 `nodeadm upgrade`。您必须使用具有 root/sudo 权限的用户运行 `nodeadm`。无论是使用 Amazon SSM 还是 Amazon IAM Roles Anywhere 凭证提供者，节点名称都会通过升级保留。在升级过程中，您无法更改凭证提供者。有关 `nodeConfig.yaml` 的配置值，请参阅 the section called “混合节点 nodeadm”。请将 `K8S_VERSION` 替换为要升级到的目标 Kubernetes 版本。

   ```
   nodeadm upgrade K8S_VERSION -c file://nodeConfig.yaml
   ```

4. 要允许在升级后在节点上调度容器组（pod），请键入以下内容。请将 `NODE_NAME` 替换为节点的名称。

   ```
   kubectl uncordon NODE_NAME
   ```

5. 观察混合节点的状态，等待节点关闭并在新的 Kubernetes 版本上重新启动并进入“就绪”状态。

   ```
   kubectl get nodes -o wide -w
   ```

混合节点的安全更新修补

本主题介绍为在混合节点上运行的特定程序包和依赖项执行就地安全更新修补的过程。作为一项最佳实践，我们建议您定期更新混合节点以获取 CVE 和安全补丁。

有关升级 Kubernetes 版本的步骤，请参阅the section called “升级混合节点”。

可能需要安装安全补丁的软件的一个例子是 `containerd`。

Containerd

`containerd` 是适用于 EKS 混合节点功能的标准 Kubernetes 容器运行时和核心依赖项，用于管理容器生命周期，包括提取映像和管理容器执行等。在混合节点上，您可以通过 nodeadm CLI 来安装 `containerd`，也可以手动安装。`nodeadm` 将从操作系统分发的程序包或 Docker 程序包安装 `containerd`，具体取决于节点使用的操作系统。

`containerd` 中的 CVE 发布后，您可以使用以下方案，在混合节点上升级到 `containerd` 的修补版本。

第 1 步：检查补丁是否已发布

您可以参考相应的安全公告，检查 `containerd` CVE 补丁是否已发布到相应的各个操作系统程序包管理器：

- Amazon Linux 2023
- RHEL
- Ubuntu 20.04
- Ubuntu 22.04
- Ubuntu 24.04

如果将该 Docker 存储库作为 `containerd` 的来源，您可以检查 Docker 安全公告，来确定已修补版本是否在 Docker 存储库中可用。

第 2 步：选择安装补丁的方法

在节点上就地修补和安装安全升级的方法有三种。具体可以使用的方法，取决于程序包管理器中的操作系统是否提供补丁：

1. 使用 `nodeadm upgrade` 发布到程序包管理器的安装补丁，详情请参阅第 2 a 步。
2. 直接使用程序包管理器安装补丁，详情请参见第 2 b 步。

3. 安装未在程序包管理器中发布的自定义补丁。对于第 2 c 步 中 containerd 的自定义补丁，有一些特殊注意事项需要注意。

第 2 a 步：使用 **nodeadm upgrade** 修补

确认 containerd CVE 补丁已发布到操作系统或 Docker 存储库（Apt 或 RPM）后，您可以使用 nodeadm upgrade 命令升级到最新版本的 containerd。由于这不属于 Kubernetes 版本升级，因此您必须将当前的 Kubernetes 版本传递给 nodeadm 升级命令。

```
nodeadm upgrade K8S_VERSION --config-source file:///root/nodeConfig.yaml
```

第 2 b 步：使用操作系统程序包管理器修补

您也可以通过相应的程序包管理器来获取更新，并使用该更新来升级 containerd 程序包，如下所示。

Amazon Linux 2023

```
sudo yum update -y
sudo yum install -y containerd
```

RHEL

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-
ce.repo
sudo yum update -y
sudo yum install -y containerd
```

Ubuntu

```
sudo mkdir -p /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/
docker.asc
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
 https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update -y
```

```
sudo apt install -y --only-upgrade containerd.io
```

第 2 c 步：**Containerd** CVE 补丁未在程序包管理器中发布

如果修补后的 containerd 版本只能通过其他方式获取，而未在程序包管理器中提供，例如对于 GitHub 发行版，可以从 GitHub 官方网站安装 containerd。

1. 如果计算机已作为混合节点加入集群，则需要运行 nodeadm uninstall 命令。
2. 安装官方 containerd 二进制文件。您可以使用 GitHub 上的官方安装步骤。
3. 运行 nodeadm install 命令并将 --containerd-source 参数设置为 none，这将跳过通过 nodeadm 安装 containerd 的过程。对于节点运行的任何操作系统，您可以在 containerd 来源中使用 none 的值。

   ```
   nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --containerd-source
    none
   ```

移除混合节点

本主题介绍如何删除 Amazon EKS 集群中的混合节点。必须使用您选择的兼容 Kubernetes 的工具（例如 kubectl）来删除混合节点。从 Amazon EKS 集群中移除节点对象后，将停止收取混合节点的费用。有关混合节点定价的更多信息，请参阅 Amazon EKS 定价。

> **Important**
>
> 移除节点对于在节点上运行的工作负载而言是破坏性的。在删除混合节点之前，我们建议您首先清空节点以将容器组转移到其他活动节点。有关清空节点的更多信息，请参阅 Kubernetes 文档中的安全地清空一个节点。

在用于与 Amazon EKS 集群的 Kubernetes API 端点进行交互的本地计算机或实例上运行以下 kubectl 步骤。如果您使用特定的 kubeconfig 文件，请使用 --kubeconfig 标志。

第 1 步：列出节点

```
kubectl get nodes
```

第 2 步：清空节点

有关 kubectl drain 命令的更多信息，请参阅 Kubernetes 文档中的 kubectl drain。

```
kubectl drain --ignore-daemonsets <node-name>
```

第 3 步：停止并卸载混合节点构件

您可以使用 Amazon EKS 混合节点功能 CLI (nodeadm) 来停止混合节点构件并将其从主机中移除。您必须使用具有 root/sudo 权限的用户运行 nodeadm。默认情况下，如果节点上还有容器组，则 nodeadm uninstall 不会执行。如果您将 Amazon Systems Manager（SSM）作为凭证提供者，则 nodeadm uninstall 命令会将主机从 Amazon SSM 托管式实例中注销。有关更多信息，请参阅 the section called “混合节点 nodeadm”。

```
nodeadm uninstall
```

第 4 步：删除集群中的节点

停止并卸载混合节点构件后，从集群中移除节点资源。

```
kubectl delete node <node-name>
```

第 5 步：检查剩余的构件

运行上述步骤后，您的混合节点上可能会剩下一些构件，具体取决于您选择的 CNI。参阅 the section called “配置 CNI” 了解更多信息。

为混合节点配置应用程序联网、附加组件和 Webhook

为混合节点创建 EKS 集群后，请为应用程序联网（CNI、BGP、入口、负载均衡、网络策略）、附加组件、Webhook 和代理设置配置附加功能。有关兼容混合节点功能的 EKS 和社区附加组件的完整列表，请参阅the section called “配置附加组件”。

EKS 集群见解 EKS 包括见解检查，可检查混合节点设置中可能损害集群或工作负载功能的错误配置。有关集群见解的更多信息，请参阅the section called “集群见解”。

以下列出了可用于混合节点的常用功能和附加组件：

- 容器网络接口（CNI）：Amazon 支持 Cilium 作为混合节点的 CNI。有关更多信息，请参阅 the section called “配置 CNI”。注意：Amazon VPC CNI 不能用于混合节点。
- 混合节点加入 EKS 集群时会自动安装 CoreDNS 以及 **kube-proxy**：CoreDNS 和 kube-proxy。这些附加组件可以在集群创建后作为 EKS 附加组件进行管理。
- 入口和负载均衡：您可以将 Amazon 负载均衡器控制器和应用程序负载均衡器（ALB）或目标类型为 ip 的网络负载均衡器（NLB）用于在混合节点上运行的工作负载。对于在混合节点上运行的工作

负载，Amazon 支持 Cilium 的内置入口、网关和 Kubernetes 服务负载均衡功能。有关更多信息，请参阅the section called “配置入口”和the section called “配置 LoadBalancer 服务”。

- 指标：您可以将 Amazon Managed Service for Prometheus（AMP）无代理抓取程序、Amazon Distro for Open Telemetry（ADOT）以及 Amazon CloudWatch 可观测性代理与混合节点结合使用。要将 AMP 无代理抓取程序用于混合节点的容器组（pod）指标，相关容器组必须可以从您用于 EKS 集群的 VPC 进行访问。
- 日志：您可以为启用混合节点功能的集群启用 EKS 控制面板日志记录。您可以将 ADOT EKS 附加组件和 Amazon CloudWatch 可观测性代理 EKS 附加组件用于混合节点和容器组（pod）日志记录。
- 容器组身份和 IRSA：您可以将 Amazon EKS 容器组身份和服务账户 IAM 角色（IRSA）与在混合节点上运行的应用程序配合使用，从而为使用其他 Amazon 服务在混合节点上运行的容器组（pod）实现精细访问控制。
- Webhook：如果运行 Webhook，请参阅the section called “配置 Webhook”，以了解在无法将本地容器组网络设置为可路由时，选择在云节点上运行 Webhook 的注意事项和步骤。
- 代理：如果在本地环境中使用代理服务器来处理离开数据中心或边缘环境的流量，则需要将混合节点和集群配置为使用您的代理服务器。有关更多信息，请参阅 the section called “配置代理”。

主题

- 为混合节点配置 CNI
- 为混合节点配置附加组件
- 为混合节点配置 Webhook
- 为混合节点功能配置代理
- 为混合节点配置 Cilium BGP
- 为混合节点配置 Kubernetes 入口
- 为混合节点配置 LoadBalancer 类型服务
- 为混合节点配置 Kubernetes 网络策略

为混合节点配置 CNI

Cilium 是 Amazon 支持的 Amazon EKS 混合节点功能容器网络接口（CNI）。您必须为混合节点安装 CNI 才能使其做好服务工作负载的准备。在 CNI 开始运行之前，混合节点的状态会显示为 `Not Ready`。您可以使用自己选择的工具（例如 Helm）来管理该 CNI。本页上的说明涵盖了 Cilium 生命周期管理（安装、升级、删除）。有关如何配置 Cilium 的入口、负载均衡和网络策略的信息，请参阅the

section called “Cilium 入口和 Cilium 网关概述”、the section called “服务类型 LoadBalancer”和the section called “配置网络策略”。

在 Amazon 云中的节点上运行时，Amazon 不支持 Cilium。Amazon VPC CNI 与混合节点功能不兼容，并且 VPC CNI 使用 `eks.amazonaws.com/compute-type: hybrid` 标签配置了反亲和性。

本页之前的 Calico 文档已移至 EKS Hybrid Examples Repository。

版本兼容性

对于 Amazon EKS 所支持的全部 Kubernetes 版本，其 EKS 混合节点均兼容 Cilium `v1.17.x` 与 `v1.18.x` 版本。

> **Note**
>
> Cilium v1.18.3 内核版本要求：由于该版本存在内核版本限制（要求 Linux 内核版本 ≥ 5.10），因此 Cilium v1.18.3 无法在以下节点上使用：

- Ubuntu 20.04
- Red Hat Enterprise Linux (RHEL) 8

有关系统要求，请参阅 Cilium 系统要求。

有关 Amazon EKS 支持的 Kubernetes 版本，请参阅 Kubernetes 版本支持。EKS 混合节点功能提供的 Kubernetes 版本支持与具有云端节点的 Amazon EKS 集群相同。

支持的功能

Amazon 维护基于开源 Cilium 项目的适用于 EKS 混合节点的 Cilium 构建版本。要获得 Amazon 对 Cilium 的支持，您必须使用由 Amazon 维护的 Cilium 构建版本和支持的 Cilium 版本。

Amazon 为 Cilium 以下功能的默认配置提供技术支持，以用于 EKS 混合节点。如果您计划使用不属于 Amazon 支持范围的功能，建议您获得 Cilium 的替代商业支持，或者利用内部专业知识对 Cilium 项目进行问题排查并提供修复。

Cilium 功能	由 Amazon 支持
Kubernetes 网络合规性	是

Cilium 功能	由 Amazon 支持
核心集群连接性	是
IP 系列	IPv4
生命周期管理	Helm
联网模式	VXLAN 封装
IP 地址管理（IPAM）	Cilium IPAM 集群范围
网络策略	Kubernetes 网络策略
边界网关协议（BGP）	Cilium BGP 控制面板
Kubernetes 入口	Cilium 入口、Cilium 网关
服务 LoadBalancer IP 分配	Cilium 负载均衡器 IPAM
服务 LoadBalancer IP 地址通告	Cilium BGP 控制面板
kube-proxy 替换	是

Cilium 注意事项

- Helm 存储库：Amazon 在 Amazon EKS Cilium/Cilium 的 Amazon Elastic Container Registry Public（Amazon ECR Public）中托管 Cilium Helm 图表。可用版本包括：
  - Cilium v1.17.9：`oci://public.ecr.aws/eks/cilium/cilium:1.17.9-0`
  - Cilium v1.18.3：`oci://public.ecr.aws/eks/cilium/cilium:1.18.3-0`

  本主题中的命令使用此存储库。请注意，某些 `helm repo` 命令对于 Amazon ECR Public 中的 Helm 存储库无效，因此您无法从本地 Helm 存储库名称中引用此存储库。因此请在大多数命令中使用完整的 URI。
- 默认情况下，Cilium 配置为在覆盖/隧道模式下运行，并以 VXLAN 为封装方法。此模式对底层物理网络的要求最低。
- 默认情况下，对于离开集群的所有容器组流量，Cilium 会将其源 IP 地址伪装为该节点的 IP 地址。如果禁用伪装功能，则您的容器组（pod）CIDR 必须可在本地网络上路由。

- 如果在混合节点上运行 Webhook，则您的容器组（pod）CIDR 必须可在本地网络上路由。如果容器组 CIDR 无法在本地网络上路由，则建议在同一集群中的云端节点上运行 Webhook。有关更多信息，请参阅 the section called “配置 Webhook” 和 the section called “准备联网”。
- Amazon 建议使用 Cilium 的内置 BGP 功能，使您的容器组（pod）CIDR 可在本地网络上路由。有关如何为 Cilium BGP 配置混合节点的更多信息，请参阅the section called “配置 BGP”。
- Cilium 中的默认 IP 地址管理（IPAM）称为集群范围，其中 Cilium 管理器根据用户配置的容器组（pod）CIDR 为每个节点分配 IP 地址。

在混合节点上安装 Cilium

过程

1. 创建一个名为 `cilium-values.yaml` 的 YAML 文件。以下示例通过设置 Cilium 代理和管理器的 `eks.amazonaws.com/compute-type: hybrid` 标签的亲和性，将 Cilium 配置为仅在混合节点上运行。
   - 使用为 EKS 集群的远程容器组（pod）网络配置的相同 容器组（pod）CIDR，对 `clusterPoolIpv4PodCIDRList` 进行配置。例如 `10.100.0.0/24`。Cilium 管理器从配置的 `clusterPoolIpv4PodCIDRList` IP 空间内分配 IP 地址切片。容器组（pod）CIDR 不得与本地节点 CIDR、VPC CIDR 或 Kubernetes 服务 CIDR 重叠。
   - 根据每个节点所需的容器组数量配置 `clusterPoolIpv4MaskSize`。例如，25 对应于分段大小 /25，每个节点 128 个容器组。
   - 在集群上部署 Cilium 之后，请勿更改 `clusterPoolIpv4PodCIDRList` 或 `clusterPoolIpv4MaskSize`，请参阅 Expanding the cluster pool 了解更多信息。
   - 如果在 kube-proxy 替换模式下运行 Cilium，请在 Helm 值中设置 `kubeProxyReplacement: "true"`，并确保在与 Cilium 相同的节点上没有运行现有的 kube-proxy 部署。
   - 以下示例禁用了 Cilium 用于 L7 网络策略和入口的 Envoy 第 7 层（L7）代理。有关更多信息，请参阅the section called “配置网络策略”和the section called “Cilium 入口和 Cilium 网关概述”。
   - 以下示例配置 `loadBalancer.serviceTopology: true`，以便在您为服务配置了服务流量分发功能的情况下，该功能可以正常运行。有关更多信息，请参阅 the section called “配置服务流量分布”。
   - 有关适用于 Cilium 的完整 Helm 值列表，请参阅 Cilium 文档中的 Helm reference。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
```

```
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: In
          values:
          - hybrid
ipam:
  mode: cluster-pool
  operator:
    clusterPoolIPv4MaskSize: 25
    clusterPoolIPv4PodCIDRList:
    - POD_CIDR
loadBalancer:
  serviceTopology: true
operator:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
              - hybrid
  unmanagedPodWatcher:
    restart: false
loadBalancer:
  serviceTopology: true
envoy:
  enabled: false
kubeProxyReplacement: "false"
```

2. 在集群上安装 Cilium。

- 将 `CILIUM_VERSION` 替换为某个 Cilium 版本（例如 `1.17.9-0` 或 `1.18.3-0`）。建议使用 Cilium 次要版本的最新补丁版本。
- 确保节点符合所选版本的内核版本要求。Cilium v1.18.3 要求 Linux 内核版本 ≥ 5.10。
- 如果使用特定的 kubeconfig 文件，请在 Helm install 命令中使用 `--kubeconfig` 标志。

```
helm install cilium oci://public.ecr.aws/eks/cilium/cilium \
    --version CILIUM_VERSION \
    --namespace kube-system \
    --values cilium-values.yaml
```

3. 使用以下命令确认 Cilium 安装是否成功。您应会看到 `cilium-operator` 部署以及在每个混合节点上运行的 `cilium-agent`。此外，混合节点现在的状态应为 `Ready`。有关如何配置 Cilium BGP 以向本地网络通告您的容器组（pod）CIDR 的信息，请转到the section called “配置 BGP”。

```
kubectl get pods -n kube-system
```

```
NAME                              READY   STATUS    RESTARTS   AGE
cilium-jjjn8                      1/1     Running   0          11m
cilium-operator-d4f4d7fcb-sc5xn   1/1     Running   0          11m
```

```
kubectl get nodes
```

```
NAME                   STATUS   ROLES    AGE   VERSION
mi-04a2cf999b7112233   Ready    <none>   19m   v1.31.0-eks-a737599
```

在混合节点上升级 Cilium

在升级 Cilium 部署之前，请仔细查看 Cilium 升级文档和升级说明，以了解目标 Cilium 版本中的更改。

1. 确保已在命令行环境中安装了 `helm` CLI。有关安装说明，请参阅 Helm 文档。
2. 运行 Cilium 升级预检。请将 `CILIUM_VERSION` 替换为您的目标 Cilium 版本。建议运行 Cilium 次要版本的最新补丁版本。您可以在 Cilium 文档的 Stable Releases section 部分找到给定次要 Cilium 版本的最新补丁版本。

```
helm install cilium-preflight oci://public.ecr.aws/eks/cilium/cilium --version
 CILIUM_VERSION \
  --namespace=kube-system \
  --set preflight.enabled=true \
  --set agent=false \
  --set operator.enabled=false
```

3. 应用 `cilium-preflight.yaml` 后，请确保状态为 `READY` 的容器组（pod）数量与正在运行的 Cilium 容器组（pod）数量相同。

```
kubectl get ds -n kube-system | sed -n '1p;/cilium/p'
```

```
NAME                      DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
cilium                    2         2         2       2            2           <none>
          1h20m
cilium-pre-flight-check   2         2         2       2            2           <none>
          7m15s
```

4. 一旦状态为 READY 的容器组数量相等，请确保 Cilium 执行前部署也标记为 READY 1/1。如果显示为 READY 0/1，请先查阅 CNP Validation 部分并解决部署问题，然后再继续升级。

```
kubectl get deployment -n kube-system cilium-pre-flight-check -w
```

```
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
cilium-pre-flight-check   1/1     1            0           12s
```

5. 删除预检

```
helm uninstall cilium-preflight --namespace kube-system
```

6. 在运行 `helm upgrade` 命令之前，请将部署值保留在 `existing-cilium-values.yaml` 中，或在运行升级命令时为您的设置使用 `--set` 命令行选项。升级操作会覆盖 Cilium ConfigMap，因此在升级时传递配置值至关重要。

```
helm get values cilium --namespace kube-system -o yaml > existing-cilium-values.yaml
```

7. 在正常的集群操作期间，所有 Cilium 组件都应运行相同的版本。以下步骤介绍了如何将所有组件从一个稳定版本升级到更高的稳定版本。从一个次要版本升级到另一个次要版本时，建议先升级到现有 Cilium 次要版本的最新补丁版本。为尽可能减少中断，请将 `upgradeCompatibility` 选项设置为此集群中安装的初始 Cilium 版本。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium --version CILIUM_VERSION \
  --namespace kube-system \
  --set upgradeCompatibility=1.X \
  -f existing-cilium-values.yaml
```

8. （可选）如果由于问题出现需要回滚升级，请运行以下命令。

```
helm history cilium --namespace kube-system
helm rollback cilium [REVISION] --namespace kube-system
```

从混合节点中删除 Cilium

1. 运行以下命令从集群中卸载所有 Cilium 组件。请注意，卸载 CNI 可能会影响节点和容器组（pod）的运行状况，因此不应在生产集群上执行卸载。

```
helm uninstall cilium --namespace kube-system
```

   从集群中移除 CNI 时，Cilium 配置的接口和路由默认不会移除，有关更多信息，请参阅 GitHub issue。
2. 要清理磁盘上的配置文件和资源，如果使用的是标准配置目录，则可以移除 GitHub Cilium 存储库中 cni-uninstall.sh 脚本所显示的文件。
3. 要从集群中移除 Cilium 自定义资源定义（CRD），可以运行以下命令。

```
kubectl get crds -oname | grep "cilium" | xargs kubectl delete
```

为混合节点配置附加组件

本页介绍通过 Amazon EKS 混合节点功能运行 Amazon 附加组件和社区附加组件的注意事项。要详细了解 Amazon EKS 附加组件以及在集群中创建、升级和移除附加组件的过程，请参阅 the section called “Amazon EKS 附加组件”。除本页中另有说明的外，对于具有混合节点的 Amazon EKS 集群，创建、升级和删除 Amazon EKS 附加组件的过程与使用在 Amazon 云端运行的节点的 Amazon EKS 集群相同。仅本页包含的组件已通过 Amazon EKS 混合节点功能兼容性验证。

下列 Amazon 附加组件与 Amazon EKS 混合节点功能兼容。

Amazon 附加组件	兼容的附加组件版本
kube-proxy	v1.25.14-eksbuild.2 及更高版本
CoreDNS	v1.9.3-eksbuild.7 及更高版本
Amazon Distro for OpenTelemetry (ADOT)	v0.102.1-eksbuild.2 及更高版本
CloudWatch 可观测性代理	v2.2.1-eksbuild.1 及更高版本
EKS 容器组身份代理	• v1.3.3-eksbuild.1 及更高版本，Bottlerocket 除外

Amazon 附加组件	兼容的附加组件版本
	• v1.3.7-eksbuild.2 及更高版本，适用于 Bottlerocket
节点监控代理	v1.2.0-eksbuild.1 及更高版本
CSI 快照控制器	v8.1.0-eksbuild.1 及更高版本
适用于 Kubernetes 的 Amazon Private CA Connector	v1.6.0-eksbuild.1 及更高版本

下列社区附加组件与 Amazon EKS 混合节点功能兼容。要了解有关社区组件的更多信息，请参阅the section called “社区附加组件”。

社区附加组件	兼容的附加组件版本
Kubernetes Metrics Server	v0.7.2-eksbuild.1 及更高版本
cert-manager	v1.17.2-eksbuild.1 及更高版本
Prometheus Node Exporter	v1.9.1-eksbuild.2 及更高版本
kube-state-metrics	v2.15.0-eksbuild.4 及更高版本
External DNS	v0.19.0-eksbuild.1 及更高版本

除以上表格所列的 Amazon EKS 附加组件外，Amazon Managed Service for Prometheus Collector，以及用于应用程序入口（HTTP）和负载均衡（TCP/UDP）的 Amazon 负载均衡器控制器也与混合节点功能兼容。

有些 Amazon 附加组件和社区附加组件与 Amazon EKS 混合节点功能不兼容。这些附加组件的最新版本存在针对混合节点所应用默认 `eks.amazonaws.com/compute-type: hybrid` 标签的反亲和性规则。这会在集群中部署有混合节点时阻止这些附加组件在混合节点上运行。如果集群既有混合节点又有在 Amazon 云端运行的节点，则可以在集群中将这些附加组件部署到在 Amazon 云端运行的节点。Amazon VPC CNI 与混合节点功能不兼容，Amazon EKS 混合节点功能支持将 Cilium 和 Calico 作为容器网络接口（CNI）。请参阅the section called “配置 CNI”了解更多信息。

Amazon 附加组件

后续章节介绍了与其他 Amazon EKS 计算类型相比，在混合节点上运行兼容的 Amazon 附加组件的区别。

kube-proxy 和 CoreDNS

使用 Amazon API 和 Amazon SDK（包括通过 Amazon CLI）创建 EKS 集群时，EKS 会默认将 kube-proxy 和 CoreDNS 安装为自主管理型附加组件。您可以在集群创建后使用 Amazon EKS 附加组件覆盖这些附加组件。有关the section called “kube-proxy”和the section called “CoreDNS”的详细信息，请参阅 EKS 文档。如果您运行混合模式集群，既有混合节点，也有位于 Amazon 云的节点，则 Amazon 建议混合节点上至少有一个 CoreDNS 副本，在 Amazon 云的节点上也至少有一个 CoreDNS 副本。有关配置步骤，请参阅the section called “配置 CoreDNS 副本”。

CloudWatch 可观测性代理

CloudWatch 可观测性代理操作符使用 Webhook。如果在混合节点上运行此操作符，则本地容器组 CIDR 必须可在本地网络上路由，并且必须为您的 EKS 集群配置远程容器组网络。有关更多信息，请参阅为混合节点配置 Webhook。

混合节点不提供节点级别指标，因为 CloudWatch Container Insights 仅在实例元数据服务（IMDS）可用的情况下才能提供节点级别指标。混合节点提供集群、工作负载、容器组和容器级别的指标。

按照使用 Amazon CloudWatch Observability 安装 CloudWatch 代理中描述的步骤安装附加组件后，必须先更新附加组件清单，然后才能在混合节点上成功运行代理。编辑集群上的 `amazoncloudwatchagents` 资源以添加 `RUN_WITH_IRSA` 环境变量，如下所示。

```
kubectl edit amazoncloudwatchagents -n amazon-cloudwatch cloudwatch-agent
```

```
apiVersion: v1
items:
- apiVersion: cloudwatch.aws.amazon.com/v1alpha1
  kind: AmazonCloudWatchAgent
  metadata:
    ...
    name: cloudwatch-agent
    namespace: amazon-cloudwatch
    ...
  spec:
    ...
    env:
```

```
    - name: RUN_WITH_IRSA # <-- Add this
      value: "True" # <-- Add this
    - name: K8S_NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
          ...
```

适用于混合节点的 Amazon Managed Prometheus 托管式收集器

Amazon Managed Service for Prometheus（AMP）托管式收集器由一个抓取程序组成，用于发现和收集 Amazon EKS 集群中资源的指标。AMP 负责为您管理抓取程序，无需您自行管理任何实例、代理或抓取程序。

您可以在不提供任何混合节点特定附加配置的情况下使用 AMP 托管式收集器。不过对于混合节点上的应用程序，指标端点必须可以从 VPC 访问，包括从 VPC 到远程容器组网络 CIDR 的路由以及本地防火墙中打开的端口。此外，集群必须具有私有集群端点访问权限。

按照《Amazon Managed Service for Prometheus 用户指南》中 Using an Amazon managed collector 部分说明的步骤进行操作。

Amazon Distro for OpenTelemetry (ADOT)

您可以使用 Amazon Distro for OpenTelemetry（ADOT）附加组件，来收集在混合节点上运行的应用程序的指标、日志和跟踪数据。ADOT 使用准入 Webhook 更改和验证收集器自定义资源请求。如果在混合节点上运行 ADOT 操作符，则本地容器组 CIDR 必须可在本地网络上路由，并且必须为您的 EKS 集群配置远程容器组网络。有关更多信息，请参阅为混合节点配置 Webhook。

按照《Amazon Distro for OpenTelemetry》文档中 Getting Started with Amazon Distro for OpenTelemetry using EKS Add-Ons 部分说明的步骤进行操作。

Amazon 负载均衡器控制器

您可以将 Amazon 负载均衡器控制器和应用程序负载均衡器（ALB）或目标类型为 `ip` 的网络负载均衡器（NLB）用于混合节点上的工作负载。与 ALB 或 NLB 一起使用的 IP 目标必须可从 Amazon 路由。Amazon 负载均衡器控制器也使用 Webhook。如果在混合节点上运行 Amazon 负载均衡器控制器操作符，则本地容器组 CIDR 必须可在本地网络上路由，并且必须为您的 EKS 集群配置远程容器组网络。有关更多信息，请参阅为混合节点配置 Webhook。

要安装 Amazon 负载均衡器控制器，请按照the section called “Amazon 应用程序负载均衡器”或the section called “Amazon 网络负载均衡器”部分说明的步骤操作。

对于使用 ALB 的 Ingress，您必须指定下面的注释。请参阅the section called “应用程序负载均衡”了解更多信息。

```
alb.ingress.kubernetes.io/target-type: ip
```

对于使用 NLB 的负载均衡，您必须指定下面的注释。请参阅the section called “网络负载均衡”了解更多信息。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

EKS 容器组身份代理

> **Note**
>
> 要在运行 Bottlerocket 的混合节点上成功部署 EKS 容器组身份代理附加组件，请确保您的 Bottlerocket 版本至少为 v1.39.0。混合节点环境中较早的 Bottlerocket 版本不支持容器组身份代理。

原始 Amazon EKS 容器组身份代理 DaemonSet 仅在节点上的 EC2 IMDS 可用时才能获取必需的 Amazon 凭证。由于 IMDS 在混合节点上不可用，因此从版本 1.3.3-eksbuild.1 开始，容器组身份代理附加组件可以选择部署挂载所需凭证的 DaemonSet。运行 Bottlerocket 的混合节点需要不同的方法来挂载凭证，从 1.3.7-eksbuild.2 版本开始，容器组身份代理附加组件可以选择部署特别以 Bottlerocket 混合节点为目标的 DaemonSet。以下各节介绍启用可选 DaemonSet 的过程。

Ubuntu/RHEL/AL2023

1. 要在 Ubuntu/RHEL/Al2023 混合节点上使用容器组身份代理，请按如下所示，在 nodeadm 配置的混合部分设置 `enableCredentialsFile: true`：

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
    hybrid:
        enableCredentialsFile: true # <-- Add this
```

这将配置 nodeadm 以创建一个将在 /eks-hybrid/.aws/credentials 下的节点上配置的凭证文件，以供 eks-pod-identity-agent 容器组使用。此凭证文件包含将会定期刷新的临时 Amazon 凭证。

2. 在每个节点上更新 nodeadm 配置后，使用您的 nodeConfig.yaml 运行以下 nodeadm init 命令，从而将混合节点加入 Amazon EKS 集群。如果节点之前已经加入集群，仍可再次运行 nodeadm init 命令。

```
nodeadm init -c file://nodeConfig.yaml
```

3. 使用 Amazon CLI 或 Amazon Web Services 管理控制台安装 eks-pod-identity-agent，并启用混合节点支持。

   a. Amazon CLI：在您用于管理集群的计算机上，运行以下命令安装 eks-pod-identity-agent 并启用混合节点支持。将 my-cluster 替换为您的集群名称。

```
aws eks create-addon \
    --cluster-name my-cluster \
    --addon-name eks-pod-identity-agent \
    --configuration-values '{"daemonsets":{"hybrid":{"create": true}}}'
```

   b. Amazon Web Services 管理控制台：如果您要通过 Amazon 控制台安装容器组身份代理附加组件，请在可选配置中添加以下内容，以部署将混合节点作为目标的 DaemonSet。

```
{"daemonsets":{"hybrid":{"create": true}}}
```

Bottlerocket

1. 要在 Bottlerocket 混合节点上使用容器组身份代理，请在用于 Bottlerocket 引导容器用户数据的命令中添加 --enable-credentials-file=true 标志，如 the section called “连接使用 Bottlerocket 的混合节点” 中所述。

   a. 如果您使用的是 SSM 凭证提供程序，则命令应如下所示：

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-
code=<activation-code> --region=<region> --enable-credentials-file=true
```

   b. 如果您使用的是 IAM Roles Anywhere 凭证提供商，则命令应如下所示：

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key> --enable-
credentials-file=true
```

这将配置引导脚本，以便在 /var/eks-hybrid/.aws/credentials 下在节点上创建凭证文件，以供 eks-pod-identity-agent 容器组（pod）使用。此凭证文件包含将会定期刷新的临时 Amazon 凭证。

2. 使用 Amazon CLI 或 Amazon Web Services 管理控制台安装 eks-pod-identity-agent 并启用 Bottlerocket 混合节点支持。
   a. Amazon CLI：在您用于管理集群的计算机上，运行以下命令安装 eks-pod-identity-agent 并启用 Bottlerocket 混合节点支持。将 my-cluster 替换为您的集群名称。

```
aws eks create-addon \
    --cluster-name my-cluster \
    --addon-name eks-pod-identity-agent \
    --configuration-values '{"daemonsets":{"hybrid-bottlerocket":{"create":
 true}}}'
```

   b. Amazon Web Services 管理控制台：如果您要通过 Amazon 控制台安装容器组身份代理附加组件，请在可选配置中添加以下内容，以部署将 Bottlerocket 混合节点作为目标的 DaemonSet。

```
{"daemonsets":{"hybrid-bottlerocket":{"create": true}}}
```

CSI 快照控制器

从版本 v8.1.0-eksbuild.2 起，CSI 快照控制器附加组件执行针对混合节点的软反亲和性规则，偏向让控制器 deployment 在与 Amazon EKS 控制面板相同的 Amazon 区域中的 EC2 上运行。将 deployment 与 Amazon EKS 控制面板托管在同一 Amazon 区域有助于减少延迟。

社区附加组件

后续章节介绍了与其他 Amazon EKS 计算类型相比，在混合节点上运行兼容的社区附加组件的区别。

Kubernetes Metrics Server

控制面板需要能够连接到 Metrics Server 的容器组 IP（如果启用了 hostNetwork，则需要能够连接到节点 IP）。因此，除非在 hostNetwork 模式下运行 Metrics Server，否则在创建 Amazon EKS 集群时必须配置远程容器组网络，并且必须确保您的容器 IP 地址可路由。要使容器组 IP 地址可路由，一种常用的方法是使用 CNI 实现边界网关协议（BGP）。

cert-manager

`cert-manager` 可使用 Webhook。如果在混合节点上运行 `cert-manager`，则本地容器组 CIDR 必须可在本地网络上路由，并且必须为您的 EKS 集群配置远程容器组网络。有关更多信息，请参阅为混合节点配置 Webhook。

为混合节点配置 Webhook

本页详细介绍了使用混合节点运行 Webhook 的注意事项。在 Kubernetes 应用程序和开源项目（例如 Amazon 负载均衡器控制器和 CloudWatch 可观测性代理）中，Webhook 用于执行运行时更改和验证功能。

可路由的容器组网络

如果您能够在本地网络上路由本地容器组 CIDR，则可以在混合节点上运行 Webhook。您可以通过多种方法来使本地容器组 CIDR 可在本地网络上路由，包括边界网关协议（BGP）、静态路由或其他自定义路由解决方案。我们推荐使用 BGP 解决方案，因为与需要自定义或手动路由配置的备选解决方案相比，此方案具有更好的可扩展性并且更易于管理。Amazon 支持使用 Cilium 和 Calico 的 BGP 功能来公开容器组 CIDR，有关更多信息，请参阅the section called “配置 CNI”和the section called “可路由的远程容器组（pod）CIDR”。

不可路由的容器组网络

如果无法在本地网络上路由本地容器组 CIDR，但又需要运行 Webhook，我们建议您在与混合节点位于同一 EKS 集群中的云节点上运行所有 Webhook。

混合模式集群的注意事项

混合模式集群是指为既有混合节点又有在 Amazon 云端运行的节点的 EKS 集群。运行混合模式集群时应注意以下建议：

- 在 Amazon 云端节点上运行 VPC CNI，在混合节点上运行 Cilium 或 Calico。在 Amazon 云端节点上运行时，Amazon 不支持 Cilium 和 Calico。
- 配置要在 Amazon 云端节点上运行的 Webhook。请参阅the section called “为附加组件配置 Webhook”，了解如何为 Amazon 和社区附加组件配置 webhook。
- 如果应用程序要求在 Amazon Cloud 节点上运行的容器组直接与在混合节点上运行的容器组通信（“东西向通信”），并且您在 Amazon Cloud 节点上使用 VPC CNI，在混合节点上使用 Cilium 或 Calico，则必须使本地容器组 CIDR 可在本地网络上路由。
- 在 Amazon Cloud 中的节点上运行至少一个 CoreDNS 副本，在混合节点上运行至少一个 CoreDNS 副本。

- 配置服务流量分布，使服务流量保持在其源区域的本地。有关服务流量分布的更多信息，请参阅the section called “配置服务流量分布”。
- 如果使用 Amazon 应用程序负载均衡器（ALB）或网络负载均衡器（NLB）来处理在混合节点上运行的工作负载流量，则用于 ALB 或 NLB 的 IP 目标必须可以从 Amazon 路由。
- Metrics Server 附加组件需要能够从 EKS 控制面板连接到 Metrics Server 容器组 IP 地址。如果在混合节点上运行 Metrics Server 附加组件，则本地容器组 CIDR 必须可在本地网络上路由。
- 要使用 Amazon Managed Service for Prometheus（AMP）托管式收集器来收集混合节点的指标，本地容器组 CIDR 必须可在本地网络上路由。您也可以将 AMP 托管式收集器用于 EKS 控制面板指标和在 Amazon Cloud 中运行的资源，以及使用 Amazon Distro for OpenTelemetry（ADOT）附加组件来收集混合节点的指标。

配置混合模式集群

要查看集群上运行的更改和验证 Webhook，可以通过集群的 EKS 控制台的资源面板查看扩展资源类型，也可以使用以下命令。EKS 还会在集群可观测性控制面板中报告 Webhook 指标，有关更多信息，请参阅the section called “可观测性仪表板”。

```
kubectl get mutatingwebhookconfigurations
```

```
kubectl get validatingwebhookconfigurations
```

配置服务流量分布

运行混合模式集群时，我们建议您使用服务流量分布将服务流量保持在其源区域的本地。推荐解决方案是服务流量分配功能（在 Kubernetes 1.31 及更高版本的 EKS 中提供此功能），而不建议使用拓扑感知路由功能，因为前者的可预测性更好。使用服务流量分配功能时，分区中运行正常的端点将会接收指向该分区的所有流量。使用拓扑感知路由功能时，每项服务必须满足该分区中的多个条件才能应用自定义路由，否则会将流量平均路由到所有端点。

如果将 Cilium 作为 CNI，则必须在运行该 CNI 时将 `enable-service-topology` 设置为 `true`，才能启用服务流量分配功能。您可以使用 Helm 安装标志 `--set loadBalancer.serviceTopology=true` 传递此配置，也可以使用 Cilium CLI 命令 `cilium config set enable-service-topology true` 更新现有安装。更新现有安装的配置后，必须重新启动在每个节点上运行的 Cilium 代理。

以下部分显示如何为 CoreDNS 服务配置服务流量分布的示例，我们建议您为集群中的所有服务启用相同的配置，以避免意外的跨环境流量。

配置 CoreDNS 副本

如果运行混合模式集群，既有混合节点，也有位于 Amazon 云端的节点，则建议混合节点上至少有一个 CoreDNS 副本，在 Amazon 云端的节点上也至少有一个 CoreDNS 副本。为防止混合模式集群设置中的延迟和网络问题，可以使用服务流量分配功能，将 CoreDNS 服务配置为首选距离最近的 CoreDNS 副本。

推荐解决方案是服务流量分配功能（在 Kubernetes 1.31 及更高版本的 EKS 中提供此功能），而不建议使用拓扑感知路由功能，因为前者的可预测性更好。使用服务流量分配功能时，分区中运行正常的端点将会接收指向该分区的所有流量。而使用拓扑感知路由功能时，每项服务必须满足该分区中的多个条件才能应用自定义路由，否则会将流量平均路由到所有端点。以下是配置服务流量分配功能的步骤。

如果将 Cilium 作为 CNI，则必须在运行该 CNI 时将 `enable-service-topology` 设置为 `true`，才能启用服务流量分配功能。您可以使用 Helm 安装标志 `--set loadBalancer.serviceTopology=true` 传递此配置，也可以使用 Cilium CLI 命令 `cilium config set enable-service-topology true` 更新现有安装。更新现有安装的配置后，必须重新启动在每个节点上运行的 Cilium 代理。

1. 为每个混合节点添加一个拓扑分区标签，例如 `topology.kubernetes.io/zone: onprem`。您也可以通过在 nodeadm 配置中指定标签，从而在 `nodeadm init` 阶段设置该标签，具体请参阅the section called “自定义 kubelet 的节点配置（可选）”。请注意，在 Amazon 云端运行的节点会自动应用一个拓扑分区标签，与节点的可用区（AZ）相对应。

```
kubectl label node hybrid-node-name topology.kubernetes.io/zone=zone
```

2. 使用拓扑分区键将 `podAntiAffinity` 添加到 CoreDNS 部署。您也可以在安装过程中使用 EKS 附加组件配置 CoreDNS 部署。

```
kubectl edit deployment coredns -n kube-system
```

```
spec:
  template:
    spec:
      affinity:
       ...
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
```

```
                matchExpressions:
                - key: k8s-app
                  operator: In
                  values:
                  - kube-dns
              topologyKey: kubernetes.io/hostname
            weight: 100
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: k8s-app
                  operator: In
                  values:
                  - kube-dns
              topologyKey: topology.kubernetes.io/zone
            weight: 50
      ...
```

3. 将设置 `trafficDistribution: PreferClose` 添加到 kube-dns 服务配置以启用服务流量分布。

```
kubectl patch svc kube-dns -n kube-system --type=merge -p '{
  "spec": {
    "trafficDistribution": "PreferClose"
  }
}'
```

4. 您可以通过查看 kube-dns 服务的端点切片来确认已启用服务流量功能。端点切片必须显示拓扑分区标签为 `hints`，从而确认服务流量分配功能已启用。如果没有看到每个端点地址为 `hints`，则说明服务流量分配功能尚未启用。

```
kubectl get endpointslice -A | grep "kube-dns"
```

```
kubectl get endpointslice [.replaceable]`kube-dns-<id>`  -n kube-system -o yaml
```

```
addressType: IPv4
apiVersion: discovery.k8s.io/v1
endpoints:
- addresses:
  - <your-hybrid-node-pod-ip>
  hints:
```

```
    forZones:
    - name: onprem
  nodeName: <your-hybrid-node-name>
  zone: onprem
- addresses:
  - <your-cloud-node-pod-ip>
  hints:
    forZones:
    - name: us-west-2a
  nodeName: <your-cloud-node-name>
  zone: us-west-2a
```

为附加组件配置 Webhook

以下附加组件使用 Webhook，并支持将 Webhook 用于混合节点。

- Amazon Load Balancer Controller
- CloudWatch 可观测性代理
- Amazon Distro for OpenTelemetry (ADOT)
- `cert-manager`

请参阅以下章节，了解如何配置这些附加组件使用的 Webhook，使其在 Amazon 云端的节点上运行。

Amazon Load Balancer Controller

要在混合模式集群设置中使用 Amazon 负载均衡器控制器，您必须在 Amazon 云端的节点上运行控制器。为此，请将以下内容添加到 Helm 值配置中，或者使用 EKS 附加组件配置来指定值。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

CloudWatch 可观测性代理

CloudWatch 可观测性代理附加组件有一个使用 Webhook 的 Kubernetes 操作符。要在混合模式集群设置下在 Amazon 云端节点上运行此操作符，请编辑 CloudWatch 可观测性代理操作符配置。无法在安装过程中使用 Helm 和 EKS 附加组件配置操作符亲和性（请参阅 containers-roadmap 问题 #2431）。

```
kubectl edit -n amazon-cloudwatch deployment amazon-cloudwatch-observability-
controller-manager
```

```
spec:
  ...
  template:
    ...
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: NotIn
                values:
                - hybrid
```

Amazon Distro for OpenTelemetry (ADOT)

Amazon Distro for OpenTelemetry（ADOT）附加组件有一个使用 Webhook 的 Kubernetes 操作符。要在混合模式集群设置中的 Amazon 云端节点上运行此操作符，请将以下内容添加到 Helm 值配置中，或者使用 EKS 附加组件配置来指定值。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

如果容器组 CIDR 无法在本地网络上路由，则 ADOT 收集器必须在混合节点上运行，以便可以从混合节点以及在其上运行的工作负载中抓取指标。为此，请编辑自定义资源定义（CRD）。

```
kubectl -n opentelemetry-operator-system edit opentelemetrycollectors.opentelemetry.io
 adot-col-prom-metrics
```

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
            - hybrid
```

您可以通过在 ADOT 收集器 CRD 配置中将以下 `relabel_configs` 添加到每个 `scrape_configs` 中，从而将 ADOT 收集器配置为仅从混合节点以及在混合节点上运行的资源中抓取指标。

```
relabel_configs:
  - action: keep
    regex: hybrid
    source_labels:
    - __meta_kubernetes_node_label_eks_amazonaws_com_compute_type
```

ADOT 附加组件的一个先决条件是为 ADOT 操作符使用的 TLS 证书安装 `cert-manager`。`cert-manager` 也会运行 Webhook，并且您可以使用以下 Helm 值配置将其配置为在 Amazon 云端的节点上运行。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
webhook:
  affinity:
```

```
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
cainjector:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
startupapicheck:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
```

cert-manager

cert-manager 附加组件会运行 Webhook，并且您可以使用以下 Helm 值配置将其配置为在 Amazon 云端的节点上运行。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

```
webhook:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
cainjector:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
startupapicheck:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
```

为混合节点功能配置代理

如果在本地环境中使用代理服务器来处理离开数据中心或边缘环境的流量，则需要单独将节点和集群配置为使用您的代理服务器。

集群

在集群上，您需要将 kube-proxy 配置为使用您的代理服务器。创建 Amazon EKS 集群后，您必须配置 kube-proxy。

Nodes

在节点上，您必须将操作系统、containerd、kubelet 和 Amazon SSM Agent 配置为使用您的代理服务器。您可以在操作系统映像构建过程中进行这些更改，也可在每个混合节点上运行 nodeadm init 之前进行这些更改。

节点级别配置

您必须在操作系统映像中或在每个混合节点上运行 nodeadm init 之前应用以下配置。

containerd 代理配置

containerd 是 Kubernetes 的默认容器管理运行时。如果使用代理来访问互联网，则必须配置 containerd，以确保其能够提取 Kubernetes 和 Amazon EKS 所需的容器镜像。

在每个混合节点的 /etc/systemd/system/containerd.service.d 目录中，创建一个名为 http-proxy.conf 的文件，其中应包含以下内容。请将 proxy-domain 和 port 替换为环境的相应值。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

来自用户数据的 **containerd** 配置

需要为此文件创建 containerd.service.d 目录。您要重新加载 systemd，才能在不重新启动的情况下获取配置文件。在 AL2023 中，该服务在脚本执行时可能已经在运行，因此还需要重新启动服务。

```
mkdir -p /etc/systemd/system/containerd.service.d
echo '[Service]' > /etc/systemd/system/containerd.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/containerd.service.d/
http-proxy.conf
systemctl daemon-reload
systemctl restart containerd
```

kubelet 代理配置

kubelet 是在每个 Kubernetes 节点上运行的 Kubernetes 节点代理，负责管理该节点以及在该节点上运行的容器组（pod）。如果在本地环境中使用代理，则必须配置 kubelet，以确保其能够与 Amazon EKS 集群的公共或私有端点通信。

在每个混合节点的 /etc/systemd/system/kubelet.service.d/ 目录中，创建一个名为 http-proxy.conf 的文件，其中应包含以下内容。请将 proxy-domain 和 port 替换为环境的相应值。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

来自用户数据的 **kubelet** 配置

必须为此文件创建 kubelet.service.d 目录。您要重新加载 systemd，才能在不重新启动的情况下获取配置文件。在 AL2023 中，该服务在脚本执行时可能已经在运行，因此还需要重新启动服务。

```
mkdir -p /etc/systemd/system/kubelet.service.d
echo '[Service]' > /etc/systemd/system/kubelet.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/kubelet.service.d/http-
proxy.conf
systemctl daemon-reload
systemctl restart kubelet
```

ssm 代理配置

ssm 是可用于初始化混合节点的凭证提供程序之一。ssm 负责完成 Amazon 身份验证并生成将由 kubelet 使用的临时凭证。如果在本地环境中使用代理，并在节点上将 ssm 作为凭证提供程序，则必须配置 ssm，以确保其能够与 Amazon SSM 服务端点通信。

在每个混合节点的以下路径中创建一个名为 http-proxy.conf 文件，具体取决于操作系统

- Ubuntu - /etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d/http-proxy.conf

- Amazon Linux 2023 和 Red Hat Enterprise Linux – `/etc/systemd/system/amazon-ssm-agent.service.d/http-proxy.conf`

使用以下内容填充该文件。请将 `proxy-domain` 和 `port` 替换为环境的相应值。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

来自用户数据的 **ssm** 配置

必须为该文件创建 ssm systemd 服务文件目录。目录路径取决于节点上使用的操作系统。

- Ubuntu - `/etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d`
- Amazon Linux 2023 和 Red Hat Enterprise Linux – `/etc/systemd/system/amazon-ssm-agent.service.d`

请替换以下重启命令中的 systemd 服务名称，具体取决于节点上使用的操作系统

- Ubuntu - `snap.amazon-ssm-agent.amazon-ssm-agent`
- Amazon Linux 2023 和 Red Hat Enterprise Linux – `amazon-ssm-agent`

```
mkdir -p systemd-service-file-directory
echo '[Service]' > [.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://[.replaceable]#proxy-domain:port"' >> systemd-
service-file-directory/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://[.replaceable]#proxy-domain:port"' >>
 [.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> [.replaceable]#systemd-service-file-
directory/http-proxy.conf
systemctl daemon-reload
systemctl restart [.replaceable]#systemd-service-name
```

操作系统代理配置

如果使用代理来访问互联网，则必须对操作系统进行配置，以确保能够从操作系统的软件包管理器中提取混合节点依赖项。

Ubuntu

1. 通过以下命令配置 snap 以使用代理：

```
sudo snap set system proxy.https=http://proxy-domain:port
sudo snap set system proxy.http=http://proxy-domain:port
```

2. 要为 apt 启用代理，请在 /etc/apt/ 目录中创建一个名为 apt.conf 的文件。请将代理域和端口替换为环境的相应值。

```
Acquire::http::Proxy "http://proxy-domain:port";
Acquire::https::Proxy "http://proxy-domain:port";
```

Amazon Linux 2023

1. 配置 dnf 以使用代理。创建一个具有环境的代理域和端口值的文件 /etc/dnf/dnf.conf。

```
proxy=http://proxy-domain:port
```

Red Hat Enterprise Linux

1. 配置 yum 以使用代理。创建一个具有环境的代理域和端口值的文件 /etc/yum.conf。

```
proxy=http://proxy-domain:port
```

IAM Roles Anywhere 代理配置

在使用带有 enableCredentialsFile 标志的 IAM Roles Anywhere 时，IAM Roles Anywhere 凭证提供商服务负责刷新凭证（请参阅the section called “EKS 容器组身份代理”）。如果在本地环境中使用代理，则必须配置该服务，以确保其能够与 IAM Roles Anywhere 端点通信。

在 /etc/systemd/system/aws_signing_helper_update.service.d/ 目录中，创建一个名为 http-proxy.conf 的包含以下内容的文件。请将 proxy-domain 和 port 替换为环境的相应值。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

集群范围配置

本节中的配置必须在创建 Amazon EKS 集群之后并且在每个混合节点上运行 `nodeadm init` 之前应用。

kube-proxy 代理配置

当混合节点加入集群时，Amazon EKS 会自动在每个混合节点上将 `kube-proxy` 作为 DaemonSet 安装。`kube-proxy` 支持在 Amazon EKS 集群上跨容器组支持的服务进行路由。要配置每台主机，`kube-proxy` 需要对 Amazon EKS 集群端点进行 DNS 解析。

1. 使用以下命令编辑 `kube-proxy` DaemonSet

   ```
   kubectl -n kube-system edit ds kube-proxy
   ```

   这将在您配置的编辑器中打开 `kube-proxy` DaemonSet 定义。

2. 添加 `HTTP_PROXY` 和 `HTTPS_PROXY` 的环境变量。请注意，您的配置中应该已经存在 `NODE_NAME` 环境变量。请将 `proxy-domain` 和 `port` 替换为环境的相应值。

   ```
   containers:
     - command:
       - kube-proxy
       - --v=2
       - --config=/var/lib/kube-proxy-config/config - --hostname-override=$(NODE_NAME)
       env:
       - name: HTTP_PROXY
         value: http://proxy-domain:port
       - name: HTTPS_PROXY
         value: http://proxy-domain:port
       - name: NODE_NAME
         valueFrom:
           fieldRef:
             apiVersion: v1
             fieldPath: spec.nodeName
   ```

为混合节点配置 Cilium BGP

本主题介绍如何为 Amazon EKS 混合节点功能配置 Cilium 边界网关协议（BGP）。Cilium 的 BGP 功能称为 Cilium BGP 控制面板，可用于向本地网络通告容器组（pod）和服务地址。有关将容器组（pod）CIDR 设置为可在本地网络上路由的替代方法，请参阅the section called “可路由的远程容器组（pod）CIDR”。

配置 Cilium BGP

先决条件

- 按照the section called “配置 CNI”中的说明安装 Cilium。

过程

1. 要将 BGP 与 Cilium 结合使用，以向本地网络通告容器组（pod）或服务地址，必须已经安装了 Cilium 并配置了 `bgpControlPlane.enabled: true`。如果要为现有 Cilium 部署启用 BGP，则如果之前未启用 BGP，则必须重新启动 Cilium 管理器才能应用 BGP 配置。可以在 Helm 值中将 `operator.rollOutPods` 设置为 `true`，以便在 Helm 安装/升级过程中重新启动 Cilium 管理器。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  --set bgpControlPlane.enabled=true
```

2. 确认 Cilium 管理器和代理已重启并正在运行。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-grwlc                       1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl   1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz   1/1     Running   0          4m20s
cilium-pnxcv                       1/1     Running   0          6m29s
cilium-r7qkj                       1/1     Running   0          4m12s
cilium-wxhfn                       1/1     Running   0          4m1s
cilium-z7hlb                       1/1     Running   0          6m30s
```

3. 创建带有 `CiliumBGPClusterConfig` 定义的名为 `cilium-bgp-cluster.yaml` 的文件。您可能需要从网络管理员获取以下信息：

- 使用 ASN 为运行 Cilium 的节点配置 `localASN`。
- 使用 ASN 为本地路由器配置 `peerASN`。
- 使用运行 Cilium 的每个节点将与之对等的本地路由器 IP 对 `peerAddress` 进行配置。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPClusterConfig
metadata:
  name: cilium-bgp
spec:
  nodeSelector:
    matchExpressions:
    - key: eks.amazonaws.com/compute-type
      operator: In
      values:
      - hybrid
  bgpInstances:
  - name: "rack0"
    localASN: NODES_ASN
    peers:
    - name: "onprem-router"
      peerASN: ONPREM_ROUTER_ASN
      peerAddress: ONPREM_ROUTER_IP
      peerConfigRef:
        name: "cilium-peer"
```

4. 将 Cilium BGP 集群配置应用于集群。

```
kubectl apply -f cilium-bgp-cluster.yaml
```

5. 使用定义 BGP 对等配置的 `CiliumBGPPeerConfig` 资源创建名为 `cilium-bgp-peer.yaml` 的文件。多个对等体可以共享相同的配置并引用共同的 `CiliumBGPPeerConfig` 资源。有关配置选项的完整列表，请参阅 Cilium 文档中的 BGP Peer Configuration。

   以下 Cilium 对等设置的值必须与您正在与之对等的本地路由器的值相匹配。

   - 配置 `holdTimeSeconds`，其决定了 BGP 对等体在宣布会话关闭之前等待保持连接或更新消息的时间。默认值为 90 秒。
   - 配置 `keepAliveTimeSeconds`，其决定了 BGP 对等体是否仍可访问以及 BGP 会话是否处于活动状态。默认值为 30 秒。
   - 配置 `restartTimeSeconds`，其决定了 Cilium 的 BGP 控制面板在重启后预计重新建立 BGP 会话的时间。默认值为 120 秒。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeerConfig
metadata:
  name: cilium-peer
spec:
  timers:
    holdTimeSeconds: 90
    keepAliveTimeSeconds: 30
  gracefulRestart:
    enabled: true
    restartTimeSeconds: 120
  families:
    - afi: ipv4
      safi: unicast
      advertisements:
        matchLabels:
          advertise: "bgp"
```

6. 将 Cilium BGP 对等体配置应用于集群。

```
kubectl apply -f cilium-bgp-peer.yaml
```

7. 使用 `CiliumBGPAdvertisement` 资源创建名为 `cilium-bgp-advertisement-pods.yaml` 的文件，用于向本地网络通告容器组（pod）CIDR。
   - `CiliumBGPAdvertisement` 资源用于定义各种通告类型以及与之相关的属性。以下示例将 Cilium 配置为仅通告容器组（pod）CIDR。有关配置 Cilium 以通告服务地址的更多信息，请参阅the section called “服务类型 LoadBalancer”和the section called “Cilium 集群内负载均衡”中的示例。
   - 每个运行 Cilium 代理的混合节点都与启用了 BGP 的上游路由器对等。如下方示例所示，当 Cilium 的 `advertisementType` 设置为 `PodCIDR` 时，每个节点都会通告其拥有的容器组（pod）CIDR 范围。有关更多信息，请参阅 Cilium 文档中的 BGP Advertisements configuration。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-pods
  labels:
    advertise: bgp
spec:
```

```
  advertisements:
    - advertisementType: "PodCIDR"
```

8. 将 Cilium BGP 广播配置应用到集群。

```
kubectl apply -f cilium-bgp-advertisement-pods.yaml
```

9. 您可以通过 Cilium CLI，使用 `cilium bgp peers` 命令来确认 BGP 对等连接是否有效。您应会在输出中看到您环境的正确值，会话状态为 `established`。有关故障排除的更多信息，请参阅 Cilium 文档中的 Troubleshooting and Operations Guide。

   在下面的示例中，有五个混合节点在运行 Cilium 代理，每个节点都会通告其拥有的容器组（pod）CIDR 范围。

```
cilium bgp peers
```

```
Node                   Local AS    Peer AS               Peer Address        Session
 State   Uptime     Family         Received   Advertised
mi-026d6a261e355fba7   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h18m58s   ipv4/unicast   1
      2
mi-082f73826a163626e   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h19m12s   ipv4/unicast   1
      2
mi-09183e8a3d755abf6   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h18m47s   ipv4/unicast   1
      2
mi-0d78d815980ed202d   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h19m12s   ipv4/unicast   1
      2
mi-0daa253999fe92daa   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h18m58s   ipv4/unicast   1
      2
```

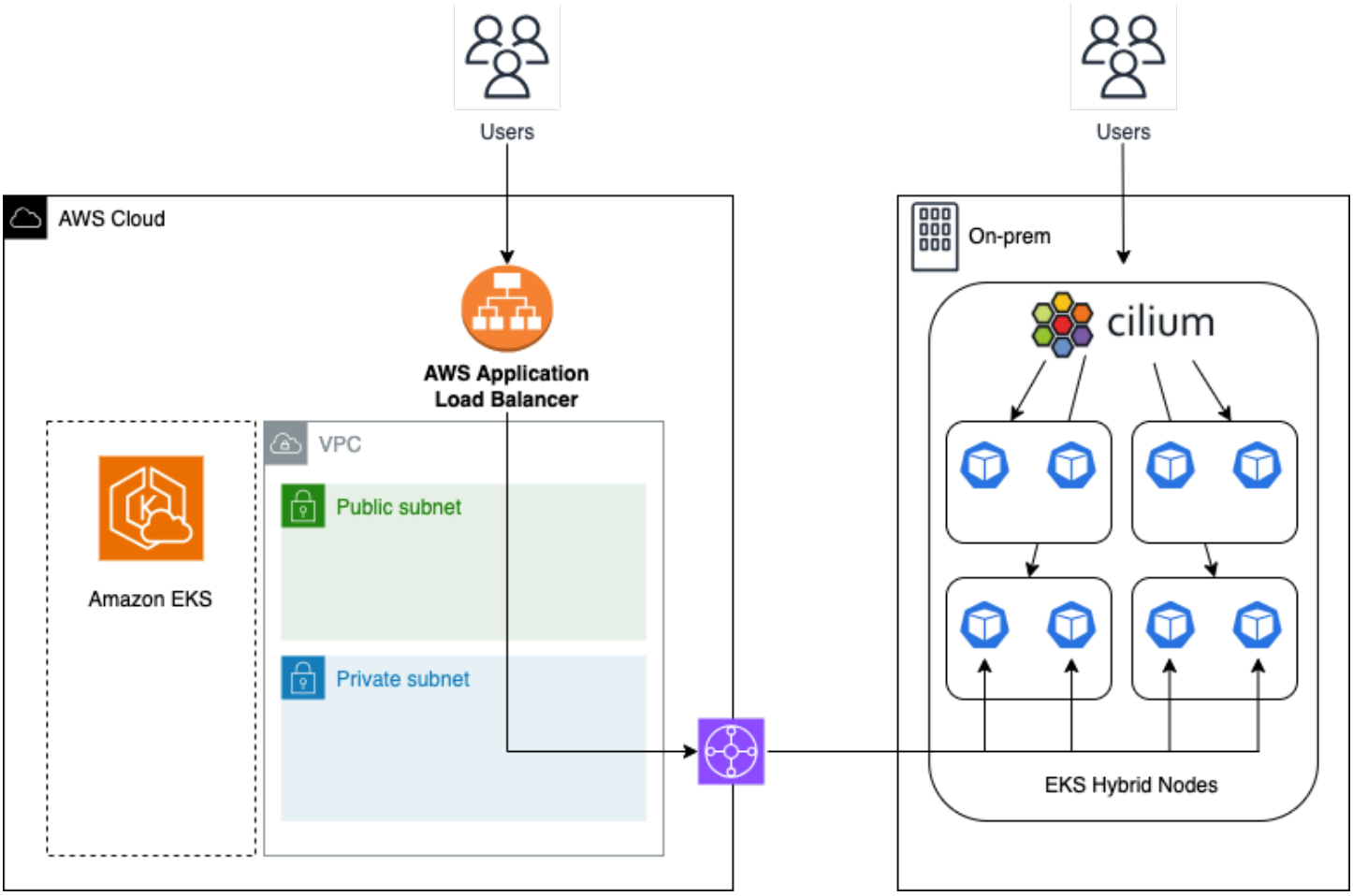
```
cilium bgp routes
```

```
Node                    VRouter       Prefix           NextHop   Age         Attrs
mi-026d6a261e355fba7    NODES_ASN     10.86.2.0/26     0.0.0.0   1h16m46s    [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-082f73826a163626e    NODES_ASN     10.86.2.192/26   0.0.0.0   1h16m46s    [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-09183e8a3d755abf6    NODES_ASN     10.86.2.64/26    0.0.0.0   1h16m46s    [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-0d78d815980ed202d    NODES_ASN     10.86.2.128/26   0.0.0.0   1h16m46s    [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-0daa253999fe92daa    NODES_ASN     10.86.3.0/26     0.0.0.0   1h16m46s    [{Origin:
 i} {Nexthop: 0.0.0.0}]
```

为混合节点配置 Kubernetes 入口

本主题介绍如何为在 Amazon EKS 混合节点上运行的工作负载配置 Kubernetes 入口。Kubernetes 入口向集群内的服务公开来自集群外部的 HTTP 和 HTTPS 路由。要使用入口资源，需要使用 Kubernetes 入口控制器来设置联网基础设施以及为网络流量提供服务的组件。

Amazon 支持 Amazon 应用程序负载均衡器（ALB）和适用于 Kubernetes 入口的 Cilium，可用于在 EKS 混合节点上运行的工作负载。使用 ALB 还是适用于入口的 Cilium，取决于应用程序流量的来源。如果应用程序流量来自某个 Amazon 区域，Amazon 建议使用 Amazon ALB 和 Amazon 负载均衡器控制器。如果应用程序流量来自本地或边缘环境，Amazon 建议使用 Cilium 的内置入口功能，无论是否在环境中使用负载均衡器基础设施，均可使用该功能。



Amazon 应用程序负载均衡器

您可以将 Amazon 负载均衡器控制器和目标类型为 `ip` 的应用程序负载均衡器（ALB）用于在混合节点上运行的工作负载。使用目标类型 `ip` 时，ALB 会绕过服务层网络路径，直接将流量转发到容器组（pod）。为了让 ALB 到达混合节点上的容器组（pod）IP 目标，本地容器组（pod）CIDR 必须可在本地网络中路由。此外，Amazon 负载均衡器控制器使用 Webhook，需要来自 EKS 控制面板的直接通信。有关更多信息，请参阅 the section called “配置 Webhook”。

注意事项

- 有关 Amazon 应用程序负载均衡器和 Amazon 负载均衡器控制器的更多信息，请参阅the section called “应用程序负载均衡”和the section called “使用 Helm 安装”。
- 有关如何在 Amazon 应用程序负载均衡器和 Amazon 网络负载均衡器之间做出选择的信息，请参阅 Best Practices for Load Balancing。
- 有关可以使用 Amazon 应用程序负载均衡器为入口资源配置的注释列表，请参阅 Amazon Load Balancer Controller Ingress annotations。

先决条件

- 按照the section called “配置 CNI”中的说明安装 Cilium。
- 按照the section called “配置 BGP” 中的说明启用 Cilium BGP 控制面板。如果您不想使用 BGP，则必须使用其他方法，使本地容器组（pod）CIDR 可在本地网络中路由。如果您不将本地容器组（pod）CIDR 设置为可路由，ALB 将无法注册或联系您的容器组（pod）IP 目标。
- 在命令行环境中安装 Helm，有关更多信息，请参阅安装 Helm 说明。
- 在命令行环境中安装 eksctl，有关更多信息，请参阅 eksctl 安装说明。

过程

1. 下载Amazon负载均衡器控制器的 IAM 策略，该策略允许负载均衡器代表您调用 Amazon API。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/refs/heads/main/docs/install/iam_policy.json
```

2. 使用上一步中下载的策略创建一个 IAM 策略。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

3. 将集群名称 (CLUSTER_NAME)、Amazon 区域 (AWS_REGION) 和 Amazon 账户 ID (AWS_ACCOUNT_ID) 的值替换为您的设置，然后运行以下命令。

```
eksctl create iamserviceaccount \
    --cluster=CLUSTER_NAME \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
    --attach-policy-arn=arn:aws:iam::AWS_ACCOUNT_ID:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region AWS_REGION \
    --approve
```

4. 添加 eks-charts Helm 图表存储库并更新您的本地 Helm 存储库，确保您拥有最新的图表。

```
helm repo add eks https://aws.github.io/eks-charts
```

```
helm repo update eks
```

5. 安装Amazon负载均衡器控制器。将集群名称 (CLUSTER_NAME)、Amazon 区域 (AWS_REGION)、VPC ID (VPC_ID) 和 Amazon 负载均衡器控制器 Helm 图表版本 (AWS_LBC_HELM_VERSION) 的值替换为您的设置，并运行下列命令。如果您运行的混合模式集群同时包含混合节点和 Amazon 云中的节点，则可以按照the section called “Amazon Load Balancer Controller” 中的说明，在云端节点上运行 Amazon 负载均衡器控制器。
   - 您可以通过运行 `helm search repo eks/aws-load-balancer-controller --versions` 找到最新版本的 Helm 图表。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --version AWS_LBC_HELM_VERSION \
  --set clusterName=CLUSTER_NAME \
  --set region=AWS_REGION \
  --set vpcId=VPC_ID \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

6. 验证 Amazon 负载均衡器控制器是否成功安装。

```
kubectl get -n kube-system deployment aws-load-balancer-controller
```

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

7. 创建示例应用程序。以下示例使用 Istio Bookinfo 示例微服务应用程序。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

8. 使用以下内容创建名为 `my-ingress-alb.yaml` 的文件。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    alb.ingress.kubernetes.io/load-balancer-name: "my-ingress-alb"
```

```
    alb.ingress.kubernetes.io/target-type: "ip"
    alb.ingress.kubernetes.io/scheme: "internet-facing"
    alb.ingress.kubernetes.io/healthcheck-path: "/details/1"
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

9. 将入口配置应用于集群。

```
kubectl apply -f my-ingress-alb.yaml
```

10.为您的入口资源预置 ALB 可能需要几分钟时间。预置 NLB 后，将为您的入口资源分配一个与 ALB 部署的 DNS 名称相对应的地址。该地址将采用以下格式：<alb-name>-<random-string>.<region>.elb.amazonaws.com。

```
kubectl get ingress my-ingress
```

```
NAME         CLASS   HOSTS   ADDRESS
    PORTS   AGE
my-ingress   alb     *       my-ingress-alb-<random-
string>.<region>.elb.amazonaws.com   80      23m
```

11.使用 ALB 的地址访问服务。

```
curl -s http//my-ingress-alb-<random-string>.<region>.elb.amazonaws.com:80/details/1
 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
```

```
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
  "details": "This is the details page"
}
```

Cilium 入口和 Cilium 网关概述

Cilium 的入口功能内置在 Cilium 的架构中，可以使用 Kubernetes 入口 API 或网关 API 进行管理。如果您没有现有的入口资源，Amazon 建议您从网关 API 开始，因为它是定义和管理 Kubernetes 联网资源的一种更具表现力和灵活性的方式。Kubernetes 网关 API 旨在标准化在 Kubernetes 集群中定义和管理入口、负载均衡和服务网格的联网资源的方式。

启用 Cilium 的入口或网关功能时，Cilium 管理器会协调集群中的入口/网关对象，每个节点上的 Envoy 代理负责处理第 7 层（L7）网络流量。Cilium 不直接预置入口/网关基础设施，例如负载均衡器。如果您计划将 Cilium 入口/网关与负载均衡器一起使用，则必须使用负载均衡器的工具（通常是入口或网关控制器）来部署和管理负载均衡器的基础设施。

对于入口/网关流量，Cilium 会处理核心网络流量和 L3/L4 策略执行，集成的 Envoy 代理负责处理第 7 层网络流量。借助 Cilium 入口/网关，Envoy 负责应用 L7 路由规则、策略和请求操作、高级流量管理（例如流量拆分和镜像）以及 TLS 终止和发起。默认情况下，Cilium 的 Envoy 代理作为单独的 DaemonSet (`cilium-envoy`) 部署，这使得可以单独对 Envoy 和 Cilium 代理进行更新、扩展和管理。

有关 Cilium 入口和 Cilium 网关如何工作的更多信息，请参阅 Cilium 文档中的 Cilium Ingress 和 Cilium Gateway 页面。

Cilium 入口和网关比较

下表汇总了截至 Cilium 1.17.x 版本的 Cilium 入口和 Cilium 网关功能。

功能	入口	Gateway
服务类型 LoadBalancer	支持	是
服务类型 NodePort	是	否[1]
主机网络	支持	是

功能	入口	Gateway
共享负载均衡器	支持	是
专用负载均衡器	是	否[2]
网络策略	支持	是
协议	第 7 层 [HTTP（S）、gRPC]	第 7 层 [HTTP（S）、gRPC][3]
TLS 传递	支持	是
流量管理	路径和主机路由	路径和主机路由、URL 重定向和重写、流量拆分、标头修改

[1] Cilium 1.18.x 版本计划提供 NodePort 服务的 Cilium 网关支持（#27273）

[2] 针对专用负载均衡器的 Cilium 网关支持（#25567）

[3] 针对 TCP/UDP 的 Cilium 网关支持（#21929）

安装 Cilim 网关

注意事项

- 配置 Cilium 时，必须将 `nodePort.enabled` 设置为 `true`，如下列示例所示。如果您使用的是 Cilium 的 kube-proxy 替换功能，则无需将 `nodePort.enabled` 设置为 `true`。
- 配置 Cilium 时，必须将 `envoy.enabled` 设置为 `true`，如下列示例所示。
- Cilium 网关可以在负载均衡器（默认）或主机网络模式下部署。
- 在负载均衡器模式下使用 Cilium 网关时，必须在网关资源上设置 `service.beta.kubernetes.io/aws-load-balancer-type: "external"` 注释，以防止传统 Amazon 云提供商为 Cilium 为网关资源创建的 LoadBalancer 类型服务创建经典负载均衡器。
- 在主机网络模式下使用 Cilium 网关时，将禁用 LoadBalancer 模式类型的服务。主机网络模式对于没有负载均衡器基础设施的环境非常有用，有关更多信息，请参阅the section called “主机网络”。

先决条件

1. 在命令行环境中安装 Helm，请参阅安装 Helm 说明。
2. 按照the section called “配置 CNI”中的说明安装 Cilium。

过程

1. 安装 Kubernetes 网关 API 自定义资源定义（CRD）。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_gatewayclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_gateways.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_httproutes.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_referencegrants.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_grpcroutes.yaml
```

2. 创建以下内容的名为 `cilium-gateway-values.yaml` 的文件。以下示例将 Cilium 网关配置为使用默认负载均衡器模式，并为配置为仅在混合节点上运行的 Envoy 代理使用单独的 `cilium-envoy` DaemonSet。

```
gatewayAPI:
  enabled: true
  # uncomment to use host network mode
  # hostNetwork:
  #   enabled: true
nodePort:
  enabled: true
envoy:
  enabled: true
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
```

```
            - hybrid
```

3. 将 Helm 值文件应用于集群。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  --values cilium-gateway-values.yaml
```

4. 确认 Cilium 管理器、代理和 Envoy 容器组（pod）正在运行。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-envoy-5pgnd                 1/1     Running   0          6m31s
cilium-envoy-6fhg4                 1/1     Running   0          6m30s
cilium-envoy-jskrk                 1/1     Running   0          6m30s
cilium-envoy-k2xtb                 1/1     Running   0          6m31s
cilium-envoy-w5s9j                 1/1     Running   0          6m31s
cilium-grwlc                       1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl   1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz   1/1     Running   0          4m20s
cilium-pnxcv                       1/1     Running   0          6m29s
cilium-r7qkj                       1/1     Running   0          4m12s
cilium-wxhfn                       1/1     Running   0          4m1s
cilium-z7hlb                       1/1     Running   0          6m30s
```

配置 Cilim 网关

通过将 `gatewayClassName` 设置为 `cilium`，可以在网关对象上启用 Cilium 网关。Cilium 为网关资源创建的服务可以使用网关对象上的字段进行配置。可以使用网关对象的 `infrastructure` 字段，配置网关控制器用于配置负载均衡器基础设施的常用注释。使用 Cilium 的 LoadBalancer IPAM（请参阅the section called “服务类型 LoadBalancer”中的示例）时，可以在网关对象的 `addresses` 字段上配置用于 LoadBalancer 类型服务的 IP 地址。有关网关配置的更多信息，请参阅 Kubernetes Gateway API specification。

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
```

```
  name: my-gateway
spec:
  gatewayClassName: cilium
  infrastructure:
    annotations:
      service.beta.kubernetes.io/...
      service.kuberentes.io/...
  addresses:
  - type: IPAddress
    value: <LoadBalancer IP address>
  listeners:
  ...
```

Cilium 和 Kubernetes 网关规范支持 GatewayClass、网关、httpRoute、grpcRoute 和 ReferenceGrant 资源。

- 有关可用字段的列表，请参阅 HTTPRoute 和 GRPCRoute 规范。
- 有关如何使用和配置这些资源，请参阅以下the section called “部署 Cilium 网关”部分中的示例和 Cilium 文档中的示例。

部署 Cilium 网关

1. 创建示例应用程序。以下示例使用 Istio Bookinfo 示例微服务应用程序。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

2. 确认应用程序在成功运行。

```
kubectl get pods
```

```
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-766844796b-9965p       1/1     Running   0          81s
productpage-v1-54bb874995-jmc8j   1/1     Running   0          80s
ratings-v1-5dc79b6bcd-smzxz       1/1     Running   0          80s
reviews-v1-598b896c9d-vj7gb       1/1     Running   0          80s
reviews-v2-556d6457d-xbt8v        1/1     Running   0          80s
reviews-v3-564544b4d6-cpmvq       1/1     Running   0          80s
```

3. 使用以下内容创建名为 my-gateway.yaml 的文件。以下示例使用 service.beta.kubernetes.io/aws-load-balancer-type: "external" 注释，防止传统 Amazon 云提供商为 Cilium 为网关资源创建的 LoadBalancer 类型服务创建经典负载均衡器。

```
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: cilium
  infrastructure:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "external"
  listeners:
  - protocol: HTTP
    port: 80
    name: web-gw
    allowedRoutes:
      namespaces:
        from: Same
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: http-app-1
spec:
  parentRefs:
  - name: my-gateway
    namespace: default
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /details
    backendRefs:
    - name: details
      port: 9080
```

4. 将网关资源应用于集群。

```
kubectl apply -f my-gateway.yaml
```

5. 确认网关资源和相应的服务已创建。在此阶段，预计网关资源的 ADDRESS 字段中没有填充 IP 地址或主机名，网关资源的 LoadBalancer 类型服务同样没有分配 IP 地址或主机名。

```
kubectl get gateway my-gateway
```

```
NAME         CLASS    ADDRESS   PROGRAMMED   AGE
my-gateway   cilium             True         10s
```

```
kubectl get svc cilium-gateway-my-gateway
```

```
NAME                          TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
    AGE
cilium-gateway-my-gateway   LoadBalancer   172.16.227.247   <pending>     80:30912/
TCP   24s
```

6. 继续执行the section called “服务类型 LoadBalancer”，将网关资源配置为使用 Cilium 负载均衡器 IPAM 分配的 IP 地址，并执行the section called “服务类型 NodePort”或the section called “主机网络”，将网关资源配置为使用 NodePort 或主机网络地址。

安装 Cilium 入口

注意事项

- 配置 Cilium 时，必须将 nodePort.enabled 设置为 true，如下列示例所示。如果您使用的是 Cilium 的 kube-proxy 替换功能，则无需将 nodePort.enabled 设置为 true。
- 配置 Cilium 时，必须将 envoy.enabled 设置为 true，如下列示例所示。
- 将 ingressController.loadbalancerMode 设置为 dedicated 后，Cilium 会为每个入口资源创建专用服务。将 ingressController.loadbalancerMode 设置为 shared 后，Cilium 会为集群中的所有入口资源创建 LoadBalancer 类型的共享服务。使用 shared 负载均衡器模式时，共享服务的设置（例如 labels、annotations、type 和 loadBalancerIP）在 Helm 值的 ingressController.service 部分中进行配置。有关更多信息，请参阅 Cilium Helm values reference。
- 将 ingressController.default 设置为 true 后，Cilium 将被配置为集群的默认入口控制器并会创建入口条目（即使未在入口资源上指定 ingressClassName，也是如此）。

- Cilium 入口可以在负载均衡器（默认）、节点端口或主机网络模式下部署。在主机网络模式下安装 Cilium 时，将禁用 LoadBalancer 类型服务和 NodePort 类型服务的模式。请参阅the section called “主机网络”了解更多信息。
- 在 Helm 值中，始终将 `ingressController.service.annotations` 设置为 `service.beta.kubernetes.io/aws-load-balancer-type: "external"`，以防止传统 Amazon 云提供商为 Cilium Helm 图表创建的默认 `cilium-ingress` 服务创建经典负载均衡器。

先决条件

1. 在命令行环境中安装 Helm，请参阅安装 Helm 说明。
2. 按照the section called “配置 CNI”中的说明安装 Cilium。

过程

1. 创建以下内容的名为 `cilium-ingress-values.yaml` 的文件。以下示例将 Cilium 入口配置为使用默认负载均衡器 `dedicated` 模式，并对配置为仅在混合节点上运行的 Envoy 代理使用单独的 `cilium-envoy` DaemonSet。

```
ingressController:
  enabled: true
  loadbalancerMode: dedicated
  service:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "external"
nodePort:
  enabled: true
envoy:
  enabled: true
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
            - hybrid
```

2. 将 Helm 值文件应用于集群。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  --values cilium-ingress-values.yaml
```

3. 确认 Cilium 管理器、代理和 Envoy 容器组（pod）正在运行。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-envoy-5pgnd                 1/1     Running   0          6m31s
cilium-envoy-6fhg4                 1/1     Running   0          6m30s
cilium-envoy-jskrk                 1/1     Running   0          6m30s
cilium-envoy-k2xtb                 1/1     Running   0          6m31s
cilium-envoy-w5s9j                 1/1     Running   0          6m31s
cilium-grwlc                       1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl   1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz   1/1     Running   0          4m20s
cilium-pnxcv                       1/1     Running   0          6m29s
cilium-r7qkj                       1/1     Running   0          4m12s
cilium-wxhfn                       1/1     Running   0          4m1s
cilium-z7hlb                       1/1     Running   0          6m30s
```

配置 Cilium 入口

通过将 `ingressClassName` 设置为 `cilium`，可以在入口对象上启用 Cilium 入口。Cilium 为入口资源创建的服务在使用 `dedicated` 负载均衡器模式时可以在入口对象上使用注释进行配置，在使用 `shared` 负载均衡器模式时可以在 Cilium/Helm 配置中进行配置。入口控制器通常使用这些注释来配置负载均衡器基础设施或服务的其他属性，例如服务类型、负载均衡器模式、端口和 TLS 传递。下面介绍了关键注释。有关支持的注释的完整列表，请参阅 Cilium 文档中的 Cilium Ingress annotations。

注释	描述
`ingress.cilium.io/loadbalancer-mode`	`dedicated` ：每个入口资源的 LoadBalancer 类型的专用服务（默认）。

注释	描述
	shared：所有入口资源的 LoadBalancer 类型的单一服务。
ingress.cilium.io/service-type	LoadBalancer ：该服务将为 LoadBalancer 类型（默认） NodePort：该服务将为 NodePort 类型。
service.beta.kubernetes.io/aws-load-balancer-type	"external" ：防止传统 Amazon 云提供商为 LoadBalancer 类型服务预置经典负载均衡器。
lbipam.cilium.io/ips	要从 Cilium LoadBalancer IPAM 分配的 IP 地址列表

Cilium 和 Kubernetes 入口规范支持入口路径的精确、前缀和特定于实现的匹配规则。Cilium 支持正则表达式作为其特定于实现的匹配规则。有关更多信息，请参阅 Cilium 文档中的 Ingress path types and precedence 和 Path types examples，以及本页the section called “部署 Cilium 入口”部分中的示例。

以下显示了示例 Cilium 入口对象。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    service.beta.kuberentes.io/...
    service.kuberentes.io/...
spec:
  ingressClassName: cilium
  rules:
  ...
```

部署 Cilium 入口

1. 创建示例应用程序。以下示例使用 Istio Bookinfo 示例微服务应用程序。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

2. 确认应用程序在成功运行。

```
kubectl get pods
```

```
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-766844796b-9965p       1/1     Running   0          81s
productpage-v1-54bb874995-jmc8j   1/1     Running   0          80s
ratings-v1-5dc79b6bcd-smzxz       1/1     Running   0          80s
reviews-v1-598b896c9d-vj7gb       1/1     Running   0          80s
reviews-v2-556d6457d-xbt8v        1/1     Running   0          80s
reviews-v3-564544b4d6-cpmvq       1/1     Running   0          80s
```

3. 使用以下内容创建名为 my-ingress.yaml 的文件。以下示例使用 service.beta.kubernetes.io/aws-load-balancer-type: "external" 注释，防止传统 Amazon 云提供商为 Cilium 为入口资源创建的 LoadBalancer 类型服务创建经典负载均衡器。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "external"
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

4. 将入口资源应用于集群。

```
kubectl apply -f my-ingress.yaml
```

5. 确认入口资源和相应的服务已创建。在此阶段，预计入口资源的 ADDRESS 字段中没有填充 IP 地址或主机名，同样，入口资源的 LoadBalancer 类型的共享或专用服务也没有分配 IP 地址或主机名。

```
kubectl get ingress my-ingress
```

```
NAME         CLASS    HOSTS   ADDRESS   PORTS   AGE
my-ingress   cilium   *                 80      8s
```

适用于负载均衡器模式 shared

```
kubectl -n kube-system get svc cilium-ingress
```

```
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
      AGE
cilium-ingress   LoadBalancer   172.16.217.48   <pending>     80:32359/TCP,443:31090/
TCP   10m
```

适用于负载均衡器模式 dedicated

```
kubectl -n default get svc cilium-ingress-my-ingress
```

```
NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
                  AGE
cilium-ingress-my-ingress    LoadBalancer   172.16.193.15   <pending>     80:32088/
TCP,443:30332/TCP   25s
```

6. 继续执行the section called “服务类型 LoadBalancer”，将入口资源配置为使用 Cilium 负载均衡器 IPAM 分配的 IP 地址，并执行the section called “服务类型 NodePort”或the section called “主机网络”，将入口资源配置为使用 NodePort 或主机网络地址。

服务类型 LoadBalancer

现有的负载均衡器基础设施

默认情况下，对于 Cilium 入口和 Cilium 网关，Cilium 会为入口/网关资源创建 LoadBalancer 类型的 Kubernetes 服务。Cilium 创建的服务的属性可以通过入口和网关资源进行配置。在创建入口或网关资源时，入口或网关的对外部公开的 IP 地址或主机名是从负载均衡器基础设施进行分配，而负载均衡器基础设施通常由入口或网关控制器预置。

许多入口和网关控制器都使用注释来检测和配置负载均衡器基础设施。这些入口和网关控制器的注释是在入口或网关资源上配置，如上面的示例所示。请参阅入口或网关控制器的文档，了解其支持的注释，并参阅 Kubernetes 入口文档和 Kubernetes 网关文档，获取常用控制器列表。

> **Important**
>
> 在具有 EKS 混合节点的情况下，Cilium 入口和网关不能与 Amazon 负载均衡器控制器和 Amazon 网络负载均衡器（NLB）一起使用。尝试将它们一起使用会导致目标未注册，因为当 NLB 的 `target-type` 设置为 `ip` 时（在 EKS 混合节点上运行工作负载时使用 NLB 的要求），NLB 会尝试直接连接到支持 LoadBalancer 类型服务的容器组（pod）IP。

没有负载均衡器基础设施

如果您的环境中没有负载均衡器基础设施和相应的入口/网关控制器，则可以将入口/网关资源和相应的 LoadBalancer 类型服务配置为使用 Cilium 的 负载均衡器 IP 地址管理（LB IPAM）分配的 IP 地址。Cilium LB IPAM 可以使用本地环境中的已知 IP 地址范围进行配置，并且可以使用 Cilium 的内置边界网关协议（BGP）支持或 L2 通告向您的本地网络通告 LoadBalancer IP 地址。

以下示例说明如何使用用于入口/网关资源的 IP 地址配置 Cilium 的 LB IPAM，以及如何配置 Cilium BGP 控制面板以向本地网络通告 LoadBalancer IP 地址。默认情况下，Cilium 的 LB IPAM 功能处于启用状态，但在创建 `CiliumLoadBalancerIPPool` 资源后才会激活。

先决条件

- 按照the section called “安装 Cilium 入口”或the section called “安装 Cilim 网关”中的说明安装 Cilium 入口或网关。
- 按照the section called “部署 Cilium 入口”或the section called “部署 Cilium 网关”中的说明部署带有示例应用程序的 Cilium 入口或网关资源。
- 按照the section called “配置 BGP” 中的说明启用 Cilium BGP 控制面板。如果您不想使用 BGP，则可以跳过此先决条件，但是在 Cilium LB IPAM 分配的 LoadBalancer IP 地址可在本地网络上路由之前，您无法访问您的入口或网关资源。

过程

1. 可选择修补入口或网关资源，以请求用于 LoadBalancer 类型服务的特定 IP 地址。如果您不请求特定 IP 地址，Cilium 将在后续步骤中从 `CiliumLoadBalancerIPPool` 资源中配置的 IP 地址范围

分配一个 IP 地址。在下面的命令中，将 LB_IP_ADDRESS 替换为请求 LoadBalancer 类型服务的 IP 地址。

网关

```
kubectl patch gateway -n default my-gateway --type=merge -p '{
  "spec": {
    "addresses": [{"type": "IPAddress", "value": "LB_IP_ADDRESS"}]
  }
}'
```

入口

```
kubectl patch ingress my-ingress --type=merge -p '{
  "metadata": {"annotations": {"lbipam.cilium.io/ips": "LB_IP_ADDRESS"}}
}'
```

2. 使用 CiliumLoadBalancerIPPool 资源创建一个名为 cilium-lbip-pool-ingress.yaml 的文件，用于为您的入口/网关资源配置负载均衡器 IP 地址范围。

   - 如果使用的是 Cilium 入口，Cilium 会自动将 cilium.io/ingress: "true" 标签应用于它为入口资源创建的服务。您可以在 CiliumLoadBalancerIPPool 资源定义的 serviceSelector 字段中使用此标签来选择符合 LB IPAM 条件的服务。
   - 如果您使用的是 Cilium 网关，则可以使用 CiliumLoadBalancerIPPool 资源定义 serviceSelector 字段中的 gateway.networking.k8s.io/gateway-name 标签来选择符合 LB IPAM 条件的网关资源。
   - 将 LB_IP_CIDR 替换为用于负载均衡器 IP 地址的 IP 地址范围。要选择单个 IP 地址，请使用 /32 CIDR。有关更多信息，请参阅 Cilium 文档中的 LoadBalancer IP Address Management。

   ```
   apiVersion: cilium.io/v2alpha1
   kind: CiliumLoadBalancerIPPool
   metadata:
     name: bookinfo-pool
   spec:
     blocks:
     - cidr: "LB_IP_CIDR"
     serviceSelector:
       # if using Cilium Gateway
       matchExpressions:
         - { key: gateway.networking.k8s.io/gateway-name, operator: In, values: [ my-
   gateway ] }
   ```

```
    # if using Cilium Ingress
    matchLabels:
      cilium.io/ingress: "true"
```

3. 将 CiliumLoadBalancerIPPool 资源应用于集群。

```
kubectl apply -f cilium-lbip-pool-ingress.yaml
```

4. 确认已从 Cilium LB IPAM 为入口/网关资源分配了 IP 地址。

   网关

```
kubectl get gateway my-gateway
```

```
NAME         CLASS    ADDRESS        PROGRAMMED   AGE
my-gateway   cilium   LB_IP_ADDRESS    True         6m41s
```

   入口

```
kubectl get ingress my-ingress
```

```
NAME         CLASS    HOSTS   ADDRESS        PORTS   AGE
my-ingress   cilium   *       LB_IP_ADDRESS   80      10m
```

5. 使用 CiliumBGPAdvertisement 资源创建一个名为 cilium-bgp-advertisement-ingress.yaml 的文件，用于通告入口/网关资源的 LoadBalancer IP 地址。如果您不使用 Cilium BGP，则可跳过此步骤。用于入口/网关资源的 LoadBalancer IP 地址必须可在本地网络上路由，这样您才能在下一步中查询该服务。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-lb-ip
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
```

```
          - LoadBalancerIP
      selector:
        # if using Cilium Gateway
        matchExpressions:
          - { key: gateway.networking.k8s.io/gateway-name, operator: In, values:
 [ my-gateway ] }
        # if using Cilium Ingress
        matchLabels:
          cilium.io/ingress: "true"
```

6. 将 CiliumBGPAdvertisement 资源应用于集群。

```
kubectl apply -f cilium-bgp-advertisement-ingress.yaml
```

7. 使用从 Cilium LB IPAM 分配的 IP 地址访问该服务。

```
curl -s http://LB_IP_ADDRESS:80/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

服务类型 NodePort

如果您的环境中没有负载均衡器基础设施和相应的入口控制器，或者您要自行管理负载均衡器基础设施或使用基于 DNS 的负载均衡，则可以将 Cilium 入口配置成为入口资源创建 NodePort 类型服务。将 NodePort 与 Cilium 入口配合使用时，NodePort 类型服务会在端口范围为 30000-32767 的每个节点的端口上公开。在此模式下，当流量到达 NodePort 上集群中的任何节点时，它会被转发到支持该服务的容器组（pod），而该容器组可能位于同一节点或不同节点上。

> **Note**
>
> Cilium 1.18.x 版本计划提供 NodePort 服务的 Cilium 网关支持（#27273）

先决条件

- 按照the section called “安装 Cilium 入口”中的说明安装Cilium 入口。
- 按照the section called “部署 Cilium 入口”中的说明部署带有示例应用程序的 Cilium 入口资源。

过程

1. 修补现有入口资源 `my-ingress`，将其从 LoadBalancer 类型服务更改为 NodePort 类型服务。

```
kubectl patch ingress my-ingress --type=merge -p '{
    "metadata": {"annotations": {"ingress.cilium.io/service-type": "NodePort"}}
}'
```

如果您尚未创建入口资源，则可以通过将以下入口定义应用于集群来创建它。请注意，下面的入口定义使用了the section called “部署 Cilium 入口”中所述的 Istio Bookinfo 示例应用程序。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "external"
    "ingress.cilium.io/service-type": "NodePort"
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
```

```
        pathType: Prefix
```

2. 确认入口资源的服务已更新为使用 NodePort 类型服务。记下输出中 HTTP 协议的端口。在下面的示例中，此 HTTP 端口为 32353，将在后续步骤中使用它来查询服务。将 Cilium 入口与 NodePort 类型服务配合使用的好处是，您可以为入口流量应用基于路径和主机的路由，以及网络策略，而对于没有入口的 NodePort 类型的标准服务，则无法做到这一点。

```
kubectl -n default get svc cilium-ingress-my-ingress
```

```
NAME                         TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
             AGE
cilium-ingress-my-ingress    NodePort   172.16.47.153   <none>        80:32353/
TCP,443:30253/TCP   27m
```

3. 获取集群节点的 IP 地址。

```
kubectl get nodes -o wide
```

```
NAME                   STATUS   ROLES    AGE   VERSION               INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. 使用您的节点的 IP 地址和上面捕获的 NodePort 访问 NodePort 类型服务。在下面的示例中，使用的节点 IP 地址是 `10.80.146.32`，NodePort 是 32353。请将这些值替换为您环境的相应值。

```
curl -s http://10.80.146.32:32353/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
```

```
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

主机网络

与 NodePort 类型服务类似，如果您没有负载均衡器基础设施和入口或网关控制器，或者您要使用外部负载均衡器自行管理负载均衡，则可以将 Cilium 入口和 Cilium 网关配置为直接在主机网络上公开入口和网关资源。当为入口或网关资源启用主机网络模式时，会自动禁用 LoadBalancer 类型服务和 NodePort 类型服务的模式，主机网络模式与每个入口或网关资源的这些替代模式相互排斥。与 NodePort 类型服务模式相比，主机网络模式为可使用的端口范围提供了更大灵活性（不限于 30000-32767 NodePort 范围），您可以配置 Envoy 代理在主机网络上运行的节点子集。

先决条件

- 按照the section called “安装 Cilium 入口”或the section called “安装 Cilim 网关”中的说明安装 Cilium 入口或网关。

过程

Gateway

1. 使用以下内容创建名为 `cilium-gateway-host-network.yaml` 的文件。

```
gatewayAPI:
  enabled: true
  hostNetwork:
    enabled: true
    # uncomment to restrict nodes where Envoy proxies run on the host network
    # nodes:
    #   matchLabels:
    #     role: gateway
```

2. 将主机网络 Cilium 网关配置应用于集群。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
```

```
  --reuse-values \
  --set operator.rollOutPods=true \
  -f cilium-gateway-host-network.yaml
```

如果您尚未创建网关资源，则可以通过将以下网关定义应用于集群来创建它。下面的网关定义使用了the section called “部署 Cilium 网关”中所述的 Istio Bookinfo 示例应用程序。在下面的示例中，网关资源配置为使用 HTTP 侦听器的 8111 端口，该端口是在主机网络上运行的 Envoy 代理的共享侦听器端口。如果您将特权端口（低于 1023）用于网关资源，请参阅 Cilium 文档获取相关说明。

```
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: cilium
  listeners:
  - protocol: HTTP
    port: 8111
    name: web-gw
    allowedRoutes:
      namespaces:
        from: Same
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: http-app-1
spec:
  parentRefs:
  - name: my-gateway
    namespace: default
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /details
    backendRefs:
    - name: details
      port: 9080
```

您可以使用以下命令观察应用的 Cilium Envoy 配置。

```
kubectl get cec cilium-gateway-my-gateway -o yaml
```

您可以使用以下命令获取 cilium-gateway-my-gateway 服务的 Envoy 侦听器端口。在此示例中，共享侦听器端口为 8111。

```
kubectl get cec cilium-gateway-my-gateway -o jsonpath={.spec.services[0].ports[0]}
```

3. 获取集群节点的 IP 地址。

```
kubectl get nodes -o wide
```

```
NAME                   STATUS   ROLES    AGE   VERSION              INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. 使用节点的 IP 地址和 cilium-gateway-my-gateway 资源的侦听器端口访问服务。在下面的示例中，使用的节点 IP 地址是 10.80.146.32，侦听器端口是 8111。请将这些值替换为您环境的相应值。

```
curl -s http://10.80.146.32:8111/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
```

```
  "ISBN-13": "123-1234567890"
}
```

入口

由于上游 Cilium 问题（#34028），主机网络模式下的 Cilium 入口需要使用 `loadbalancerMode: shared`，这会为集群中的所有入口资源创建 ClusterIP 类型的单一服务。如果您将特权端口（低于 1023）用于入口资源，请参阅 Cilium 文档获取相关说明。

1. 使用以下内容创建名为 `cilium-ingress-host-network.yaml` 的文件。

```
ingressController:
  enabled: true
  loadbalancerMode: shared
  # This is a workaround for the upstream Cilium issue
  service:
    externalTrafficPolicy: null
    type: ClusterIP
  hostNetwork:
    enabled: true
    # ensure the port does not conflict with other services on the node
    sharedListenerPort: 8111
    # uncomment to restrict nodes where Envoy proxies run on the host network
    # nodes:
    #   matchLabels:
    #     role: ingress
```

2. 将主机网络 Cilium 入口配置应用于集群。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  -f cilium-ingress-host-network.yaml
```

如果您尚未创建入口资源，则可以通过将以下入口定义应用于集群来创建它。下面的入口定义使用了the section called “部署 Cilium 入口”中所述的 Istio Bookinfo 示例应用程序。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: my-ingress
  namespace: default
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

您可以使用以下命令观察应用的 Cilium Envoy 配置。

```
kubectl get cec -n kube-system cilium-ingress -o yaml
```

您可以使用以下命令获取 `cilium-ingress` 服务的 Envoy 侦听器端口。在此示例中，共享侦听器端口为 8111。

```
kubectl get cec -n kube-system cilium-ingress -o
 jsonpath={.spec.services[0].ports[0]}
```

3. 获取集群节点的 IP 地址。

```
kubectl get nodes -o wide
```

```
NAME                   STATUS   ROLES    AGE   VERSION               INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. 使用节点的 IP 地址和 `cilium-ingress` 资源的 `sharedListenerPort` 访问服务。在下面的示例中，使用的节点 IP 地址是 `10.80.146.32`，侦听器端口是 `8111`。请将这些值替换为您环境的相应值。

```
curl -s http://10.80.146.32:8111/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

为混合节点配置 LoadBalancer 类型服务

本主题介绍了如何为在 Amazon EKS 混合节点上运行的应用程序配置第 4 层（L4）负载均衡。LoadBalancer 类型的 Kubernetes 服务用于向集群外部公开 Kubernetes 应用程序。LoadBalancer 类型服务通常与云或本地环境中的物理负载均衡器基础设施结合使用，以处理工作负载的流量。这种负载均衡器基础设施通常通过特定于环境的控制器进行预置。

对于在 EKS 混合节点上运行的 LoadBalancer 类型服务，Amazon 支持使用 Amazon 网络负载均衡器（NLB）和 Cilium。使用 NLB 还是 Cilium 取决于应用程序流量的来源。如果应用程序流量来自某个 Amazon 区域，Amazon 建议使用 Amazon NLB 和 Amazon 负载均衡器控制器。如果应用程序流量来自本地或边缘环境，Amazon 建议使用 Cilium 的内置负载均衡功能，无论是否在环境中使用负载均衡器基础设施，均可使用该功能。

有关第 7 层（L7）应用程序流量负载均衡的信息，请参阅the section called “配置入口”。有关使用 EKS 实现负载均衡的一般信息，请参阅 Best Practices for Load Balancing。

Amazon 网络负载均衡器

您可以将 Amazon 负载均衡器控制器和目标类型为 `ip` 的 NLB 用于在混合节点上运行的工作负载。使用目标类型 `ip` 时，NLB 会绕过服务层网络路径，直接将流量转发到容器组（pod）。为了让 NLB 到达混合节点上的容器组（pod）IP 目标，本地容器组（pod）CIDR 必须可在本地网络中路由。此

外，Amazon 负载均衡器控制器使用 Webhook，需要来自 EKS 控制面板的直接通信。有关更多信息，请参阅 the section called “配置 Webhook”。

- 有关子网配置要求，请参阅the section called “网络负载均衡”；有关 Amazon 网络负载均衡器及 Amazon 负载均衡器控制器的其他信息，请参阅the section called “使用 Helm 安装” 和 Best Practices for Load Balancing。
- 有关可应用于使用 Amazon 网络负载均衡器的 `LoadBalancer` 类型服务的配置，请参阅 Amazon Load Balancer Controller NLB configurations。

先决条件

- 按照the section called “配置 CNI”中的说明安装 Cilium。
- 按照the section called “配置 BGP” 中的说明启用 Cilium BGP 控制面板。如果您不想使用 BGP，则必须使用其他方法，使本地容器组（pod）CIDR 可在本地网络中路由，有关更多信息，请参阅the section called “可路由的远程容器组（pod）CIDR”。
- 在命令行环境中安装 Helm，请参阅安装 Helm 说明。
- 在命令行环境中安装 eksctl，请参阅安装 eksctl 说明。

过程

1. 下载Amazon负载均衡器控制器的 IAM 策略，该策略允许负载均衡器代表您调用 Amazon API。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/refs/heads/main/docs/install/iam_policy.json
```

2. 使用上一步中下载的策略创建一个 IAM 策略。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

3. 将集群名称 (`CLUSTER_NAME`)、Amazon 区域 (`AWS_REGION`) 和 Amazon 账户 ID (`AWS_ACCOUNT_ID`) 的值替换为您的设置，然后运行以下命令。

```
eksctl create iamserviceaccount \
    --cluster=CLUSTER_NAME \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
```

```
    --attach-policy-arn=arn:aws:iam::AWS_ACCOUNT_ID:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region AWS_REGION \
    --approve
```

4. 添加 eks-charts Helm 图表存储库。Amazon 会在 GitHub 上维护此存储库。

```
helm repo add eks https://aws.github.io/eks-charts
```

5. 更新本地 Helm 存储库，确保拥有最新的图表。

```
helm repo update eks
```

6. 安装Amazon负载均衡器控制器。将集群名称 (CLUSTER_NAME)、Amazon 区域 (AWS_REGION)、VPC ID (VPC_ID) 和 Amazon 负载均衡器控制器 Helm 图表版本 (AWS_LBC_HELM_VERSION) 的值替换为您的设置。您可以通过运行 `helm search repo eks/aws-load-balancer-controller --versions` 找到最新版本的 Helm 图表。如果您运行的混合模式集群同时包含混合节点和 Amazon 云中的节点，则可以按照the section called “Amazon Load Balancer Controller” 中的说明，在云端节点上运行 Amazon 负载均衡器控制器。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --version AWS_LBC_HELM_VERSION \
  --set clusterName=CLUSTER_NAME \
  --set region=AWS_REGION \
  --set vpcId=VPC_ID \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

7. 验证 Amazon 负载均衡器控制器是否成功安装。

```
kubectl get -n kube-system deployment aws-load-balancer-controller
```

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

8. 在名为 `tcp-sample-app.yaml` 的文件中定义示例应用程序。以下示例使用带有 TCP 端口的简单 NGINX 部署。

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: tcp-sample-app
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: tcp
              containerPort: 80
```

9. 将部署应用于集群。

```
kubectl apply -f tcp-sample-app.yaml
```

10.在名为 `tcp-sample-service.yaml` 的文件中为部署定义 LoadBalancer 类型服务。

```
apiVersion: v1
kind: Service
metadata:
  name: tcp-sample-service
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
```

```
    app: nginx
```

11.将服务配置应用于集群。

```
kubectl apply -f tcp-sample-service.yaml
```

12.为服务预置 NLB 可能需要几分钟时间。预置 NLB 后，将为服务分配一个与 NLB 部署的 DNS 名称相对应的地址。

```
kubectl get svc tcp-sample-service
```

```
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP
                                                PORT(S)        AGE
tcp-sample-service   LoadBalancer   172.16.115.212   k8s-default-tcpsampl-xxxxxxxxxx-
xxxxxxxxxxxxxxxx.elb.<region>.amazonaws.com   80:30396/TCP   8s
```

13.使用 NLB 的地址访问服务。

```
curl k8s-default-tcpsampl-xxxxxxxxxx-xxxxxxxxxxxxxxxx.elb.<region>.amazonaws.com
```

输出示例如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

14.清理您创建的 资源。

```
kubectl delete -f tcp-sample-service.yaml
kubectl delete -f tcp-sample-app.yaml
```

Cilium 集群内负载均衡

Cilium 可用作在 EKS 混合节点上运行的工作负载的集群内负载均衡器，这对于没有负载均衡器基础设施的环境非常有用。Cilium 的负载均衡功能基于多种 Cilium 功能构建而成，包括 kube-proxy 替换、负载均衡器 IP 地址管理（IPAM）和 BGP 控制面板。这些功能的职责详述如下：

- Cilium kube-proxy 替换：负责将服务流量路由到后端容器组（pod）。

- Cilium 负载均衡器 IPAM：管理可分配给 LoadBalancer 类型服务的 IP 地址。
- Cilium BGP 控制面板：向本地网络通告负载均衡器 IPAM 分配的 IP 地址。

如果不使用 Cilium 的 kube-proxy 替换，您仍然可以使用 Cilium 负载均衡器 IPAM 和 BGP 控制面板为 LoadBalancer 类型服务分配和指定 IP 地址。如果不使用 Cilium 的 kube-proxy 替换，则在默认情况下，服务到后端容器组（pod）的负载均衡将由 EKS 中的 kube-proxy 和 iptables 规则处理。

先决条件

- 无论是否启用 kube-proxy 替换，都已按照the section called “配置 CNI” 中的说明安装 Cilium。Cilium 的 kube-proxy 替换需要运行 Linux 内核至少为 v4.19.57、v5.1.16 或 v5.2.0 的最新操作系统。除 Red Hat Enterprise Linux（RHEL）8.x 外，所有支持用于混合节点的操作系统的最新版本都符合此标准。
- 按照the section called “配置 BGP” 中的说明启用 Cilium BGP 控制面板。如果您不想使用 BGP，则必须使用其他方法，使本地容器组（pod）CIDR 可在本地网络中路由，有关更多信息，请参阅the section called “可路由的远程容器组（pod）CIDR”。
- 在命令行环境中安装 Helm，请参阅安装 Helm 说明。

过程

1. 创建一个名为 cilium-lbip-pool-loadbalancer.yaml 的文件，其中包含 CiliumLoadBalancerIPPool 资源，用于为 LoadBalancer 类型服务配置负载均衡器 IP 地址范围。
   - 将 LB_IP_CIDR 替换为用于负载均衡器 IP 地址的 IP 地址范围。要选择单个 IP 地址，请使用 /32 CIDR。有关更多信息，请参阅 Cilium 文档中的 LoadBalancer IP Address Management。
   - 配置 serviceSelector 字段，使其与将在后续步骤中创建的服务名称相匹配。此配置完成后，此池中的 IP 将仅分配给名为 tcp-sample-service 的服务。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: tcp-service-pool
spec:
  blocks:
  - cidr: "LB_IP_CIDR"
  serviceSelector:
    matchLabels:
```

```
      io.kubernetes.service.name: tcp-sample-service
```

2. 将 CiliumLoadBalancerIPPool 资源应用于集群。

```
kubectl apply -f cilium-lbip-pool-loadbalancer.yaml
```

3. 确认池中至少有一个可用的 IP 地址。

```
kubectl get ciliumloadbalancerippools.cilium.io
```

```
NAME               DISABLED   CONFLICTING   IPS AVAILABLE   AGE
tcp-service-pool   false      False         1               24m
```

4. 创建一个名为 cilium-bgp-advertisement-loadbalancer.yaml 的文件，其中包含 CiliumBGPAdvertisement 资源，用于通告将在下一步中创建的服务的负载均衡器 IP 地址。如果您不使用 Cilium BGP，则可跳过此步骤。用于服务的负载均衡器 IP 地址必须可在本地网络中路由，以便您可以在最后一步中查询服务。
   - 将 advertisementType 字段设置为 Service，service.addresses 设置为 LoadBalancerIP，以仅通告 LoadBalancer 类型服务的 LoadBalancerIP。
   - 配置 selector 字段，使其与将在后续步骤中创建的服务名称相匹配。此配置完成后，仅会通告名为 tcp-sample-service 的服务的 LoadBalancerIP。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-tcp-service
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - LoadBalancerIP
      selector:
        matchLabels:
          io.kubernetes.service.name: tcp-sample-service
```

5. 将 CiliumBGPAdvertisement 资源应用于集群。如果您不使用 Cilium BGP，则可跳过此步骤。

```
kubectl apply -f cilium-bgp-advertisement-loadbalancer.yaml
```

6. 在名为 `tcp-sample-app.yaml` 的文件中定义示例应用程序。以下示例使用带有 TCP 端口的简单 NGINX 部署。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tcp-sample-app
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: tcp
              containerPort: 80
```

7. 将部署应用于集群。

```
kubectl apply -f tcp-sample-app.yaml
```

8. 在名为 `tcp-sample-service.yaml` 的文件中为部署定义 LoadBalancer 类型服务。

   - 您可以使用服务对象上的 `lbipam.cilium.io/ips` 注释，从负载均衡器 IP 池请求特定 IP 地址。如果您不想为服务请求特定 IP 地址，可以删除此注释。
   - `loadBalancerClass` 规范字段为必填字段，用于防止传统 Amazon 云提供商为服务创建经典负载均衡器。在以下示例中，将此字段配置为 `io.cilium/bgp-control-plane`，以将 Cilium 的 BGP 控制面板用作负载均衡器类。也可将此字段配置为 `io.cilium/l2-announcer`，以使用 Cilium 的 L2 通告功能（目前处于测试阶段，尚未得到 Amazon 正式支持）。

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: tcp-sample-service
  namespace: default
  annotations:
    lbipam.cilium.io/ips: "LB_IP_ADDRESS"
spec:
  loadBalancerClass: io.cilium/bgp-control-plane
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

9. 将服务应用于集群。服务将使用外部 IP 地址创建，您可以使用该地址访问应用程序。

```
kubectl apply -f tcp-sample-service.yaml
```

10确认服务是否已成功创建，是否为其分配了来自上一步创建的 CiliumLoadBalancerIPPool 的 IP。

```
kubectl get svc tcp-sample-service
```

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
 AGE
tcp-sample-service   LoadBalancer   172.16.117.76   LB_IP_ADDRESS   80:31129/TCP
 14m
```

11如果您在 kube-proxy 替换模式下使用 Cilium，则可通过运行以下命令来确认 Cilium 是否正在处理服务的负载均衡。在以下输出中，10.86.2.x 地址是服务后端容器组（pod）的容器组（pod）IP 地址。

```
kubectl -n kube-system exec ds/cilium -- cilium-dbg service list
```

```
ID   Frontend               Service Type   Backend
...
41   LB_IP_ADDRESS:80/TCP   LoadBalancer   1 => 10.86.2.76:80/TCP (active)
                                           2 => 10.86.2.130:80/TCP (active)
                                           3 => 10.86.2.141:80/TCP (active)
```

12确认 Cilium 正在通过 BGP 向本地网络通告 IP 地址。在以下示例中，有五个混合节点，每个节点都将向本地网络通告 tcp-sample-service 服务的 LB_IP_ADDRESS。

```
Node                    VRouter      Prefix             NextHop   Age     Attrs
mi-026d6a261e355fba7    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-082f73826a163626e    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-09183e8a3d755abf6    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-0d78d815980ed202d    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-0daa253999fe92daa    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
```

13使用分配的负载均衡器 IP 地址访问服务。

```
curl LB_IP_ADDRESS
```

输出示例如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

14清理您创建的 资源。

```
kubectl delete -f tcp-sample-service.yaml
kubectl delete -f tcp-sample-app.yaml
kubectl delete -f cilium-lb-ip-pool.yaml
kubectl delete -f cilium-bgp-advertisement.yaml
```

为混合节点配置 Kubernetes 网络策略

如果将 Cilium 用作 EKS 混合节点的 CNI，则 Amazon 支持通过 Kubernetes 网络策略（第 3 层/第 4 层）来控制容器组（pod）的入口和出口流量。如果运行的 EKS 集群包含 Amazon 云中的节点，则 Amazon 支持使用 Amazon VPC CNI 来实现 Kubernetes 网络策略。

本主题介绍如何通过 EKS 混合节点配置 Cilium 和 Kubernetes 网络策略。有关 Kubernetes 网络策略的详细信息，请参阅 Kubernetes 文档中的 Kubernetes Network Policies。

配置网络策略

注意事项

- Amazon 支持通过上游 Kubernetes 网络策略及其规范，控制容器组（pod）的入口和出口流量，但 Amazon 目前不支持 `CiliumNetworkPolicy` 或 `CiliumClusterwideNetworkPolicy`。
- `policyEnforcementMode` Helm 可用于控制 Cilium 的默认策略执行行为。默认行为允许所有出口和入口流量。如果一个端点被某个网络策略选中，则会转换为默认拒绝状态，仅允许明确允许的流量。有关默认策略模式和策略执行模式的更多信息，请参阅 Cilium 文档。
- 如果要为现有的 Cilium 安装更改 `policyEnforcementMode`，则必须重启 Cilium 代理 DaemonSet，这样才能应用新的策略执行模式。
- 使用 `namespaceSelector` 和 `podSelector` 以允许或拒绝进出带有匹配标签的命名空间和容器组（pod）的流量。`namespaceSelector` 和 `podSelector` 可与 `matchLabels` 或 `matchExpressions` 结合使用，以根据标签选择命名空间和容器组（pod）。
- 使用 `ingress.ports` 和 `egress.ports` 以允许或拒绝进出端口和协议的流量。
- `ipBlock` 字段不能用于选择性地允许或拒绝进出容器组（pod）IP 地址的流量（#9209）。对节点 IP 使用 `ipBlock` 选择器是 Cilium 中的一项测试功能，不受 Amazon 支持。
- 有关 Kubernetes 网络策略可用字段的信息，请参阅 Kubernetes 文档中的 NetworkPolicy resource。

先决条件

- 按照the section called “配置 CNI”中的说明安装 Cilium。
- 在命令行环境中安装 Helm，请参阅安装 Helm 说明。

过程

以下过程为示例微服务应用程序设置了网络策略，使得各个组件只能与应用程序正常运行所需的其他组件通信。该过程使用 Istio Bookinfo 示例微服务应用程序。

Bookinfo 应用程序由四项独立的微服务组成，它们具有以下关系：

- productpage。productpage 微服务会调用 details 和 reviews 微服务来填充页面。
- details。details 微服务包含包含书籍信息。
- reviews。reviews 微服务包含书评，还会调用 ratings 微服务。
- ratings。ratings 微服务包含随书评附带的书籍排名信息。

1. 创建示例应用程序。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

2. 确认应用程序成功运行，并记下 productpage 微服务的容器组（pod）IP 地址。您将在后续步骤中使用此容器组（pod）IP 地址查询每项微服务。

```
kubectl get pods -o wide
```

```
NAME                              READY   STATUS    RESTARTS   AGE   IP
 NODE
details-v1-766844796b-9wff2       1/1     Running   0          7s    10.86.3.7
 mi-0daa253999fe92daa
productpage-v1-54bb874995-lwfgg   1/1     Running   0          7s    10.86.2.193
 mi-082f73826a163626e
ratings-v1-5dc79b6bcd-59njm       1/1     Running   0          7s    10.86.2.232
 mi-082f73826a163626e
reviews-v1-598b896c9d-p2289       1/1     Running   0          7s    10.86.2.47
 mi-026d6a261e355fba7
reviews-v2-556d6457d-djktc        1/1     Running   0          7s    10.86.3.58
 mi-0daa253999fe92daa
reviews-v3-564544b4d6-g8hh4       1/1     Running   0          7s    10.86.2.69
 mi-09183e8a3d755abf6
```

3. 创建一个容器组（pod），用于全程测试网络策略。请注意，该容器组（pod）是在 `default` 命名空间中创建，带有 `access: true` 标签。

```
kubectl run curl-pod --image=curlimages/curl -i --tty --labels=access=true --
namespace=default --overrides='{"spec": { "nodeSelector": {"eks.amazonaws.com/
compute-type": "hybrid"}}}' -- /bin/sh
```

4. 测试对 productpage 微服务的访问权限。在以下示例中，我们使用 productpage 容器组（pod）的容器组（pod）IP 地址 (10.86.2.193) 查询该微服务。请将其替换为环境中 productpage 容器组（pod）的容器组（pod）IP 地址。

```
curl -s http://10.86.2.193:9080/productpage | grep -o "<title>.*</title>"
```

```
<title>Simple Bookstore App</title>
```

5. 您可以通过输入 `exit` 来退出测试 curl 容器组（pod），也可以通过运行以下命令重新附加到该容器组（pod）。

```
kubectl attach curl-pod -c curl-pod -i -t
```

6. 为演示后续步骤中网络策略的效果，我们首先创建了一个拒绝所有流向 BookInfo 微服务之流量的网络策略。创建一个名为 `network-policy-deny-bookinfo.yaml` 的文件，用于定义拒绝网络策略。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-bookinfo
  namespace: default
spec:
  podSelector:
    matchExpressions:
    - key: app
      operator: In
      values: ["productpage", "details", "reviews", "ratings"]
  policyTypes:
  - Ingress
  - Egress
```

7. 将拒绝网络策略应用于集群。

```
kubectl apply -f network-policy-default-deny-bookinfo.yaml
```

8. 测试对 BookInfo 应用程序的访问权限。在以下示例中，我们使用 productpage 容器组（pod）的容器组（pod）IP 地址 (10.86.2.193) 查询该微服务。请将其替换为环境中 productpage 容器组（pod）的容器组（pod）IP 地址。

```
curl http://10.86.2.193:9080/productpage --max-time 10
```

```
curl: (28) Connection timed out after 10001 milliseconds
```

9. 创建一个名为 network-policy-productpage.yaml 的文件，用于定义 productpage 网络策略。该策略具有以下规则：
   - 允许来自带有 access: true 标签的容器组（pod）[即上一步中创建的 curl 容器组（pod）] 的入口流量
   - 允许端口 9080 上的出口 TCP 流量流向 details、reviews 和 ratings 微服务
   - 允许端口 53 上的出口 TCP/UDP 流量流向在 kube-system 命名空间中运行的 CoreDNS

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: productpage-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: productpage
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
  egress:
  - to:
    - podSelector:
        matchExpressions:
        - key: app
          operator: In
          values: ["details", "reviews", "ratings"]
    ports:
```

```
    - port: 9080
      protocol: TCP
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - port: 53
      protocol: UDP
    - port: 53
      protocol: TCP
```

10.将 productpage 网络策略应用于集群。

```
kubectl apply -f network-policy-productpage.yaml
```

11.连接到 curl 容器组（pod）并测试对 Bookinfo 应用程序的访问权限。现在允许访问 productpage 微服务，但其他微服务仍被拒绝，因为它们仍然受到拒绝网络策略的约束。在以下示例中，我们使用 productpage 容器组（pod）的容器组（pod）IP 地址 (`10.86.2.193`) 查询该微服务。请将其替换为环境中 productpage 容器组（pod）的容器组（pod）IP 地址。

```
kubectl attach curl-pod -c curl-pod -i -t
```

```
curl -s http://10.86.2.193:9080/productpage | grep -o "<title>.*</title>"
<title>Simple Bookstore App</title>
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1
{"error": "Sorry, product details are currently unavailable for this book."}
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1/reviews
{"error": "Sorry, product reviews are currently unavailable for this book."}
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1/ratings
{"error": "Sorry, product ratings are currently unavailable for this book."}
```

12.创建一个名为 `network-policy-details.yaml` 的文件，用于定义 details 网络策略。该策略仅允许来自 productpage 微服务的入口流量。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: details-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: details
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: productpage
```

13.创建一个名为 network-policy-reviews.yaml 的文件，用于定义 reviews 网络策略。该策略仅允许来自 productpage 微服务的入口流量，并且仅允许流向 ratings 微服务和 CoreDNS 的出口流量。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: reviews-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: reviews
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: productpage
  egress:
  - to:
    - podSelector:
        matchLabels:
```

```
          app: ratings
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - port: 53
      protocol: UDP
    - port: 53
      protocol: TCP
```

14.创建一个名为 network-policy-ratings.yaml 的文件，用于定义 ratings 网络策略。该策略仅允许来自 productpage 和 reviews 微服务的入口流量。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ratings-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: ratings
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchExpressions:
        - key: app
          operator: In
          values: ["productpage", "reviews"]
```

15.将 details、reviews 和 ratings 网络策略应用于集群。

```
kubectl apply -f network-policy-details.yaml
kubectl apply -f network-policy-reviews.yaml
kubectl apply -f network-policy-ratings.yaml
```

16连接到 curl 容器组（pod）并测试对 Bookinfo 应用程序的访问权限。在以下示例中，我们使用 productpage 容器组（pod）的容器组（pod）IP 地址 (10.86.2.193) 查询该微服务。请将其替换为环境中 productpage 容器组（pod）的容器组（pod）IP 地址。

```
kubectl attach curl-pod -c curl-pod -i -t
```

测试 details 微服务。

```
curl -s http://10.86.2.193:9080/api/v1/products/1
```

```
{"id": 1, "author": "William Shakespeare", "year": 1595, "type": "paperback",
 "pages": 200, "publisher": "PublisherA", "language": "English", "ISBN-10":
 "1234567890", "ISBN-13": "123-1234567890"}
```

测试 reviews 微服务。

```
curl -s http://10.86.2.193:9080/api/v1/products/1/reviews
```

```
{"id": "1", "podname": "reviews-v1-598b896c9d-p2289", "clustername": "null",
 "reviews": [{"reviewer": "Reviewer1", "text": "An extremely entertaining play
 by Shakespeare. The slapstick humour is refreshing!"}, {"reviewer": "Reviewer2",
 "text": "Absolutely fun and entertaining. The play lacks thematic depth when
 compared to other plays by Shakespeare."}]}
```

测试 ratings 微服务。

```
curl -s http://10.86.2.193:9080/api/v1/products/1/ratings
```

```
{"id": 1, "ratings": {"Reviewer1": 5, "Reviewer2": 4}}
```

17清理您在此过程中创建的资源。

```
kubectl delete -f network-policy-deny-bookinfo.yaml
kubectl delete -f network-policy-productpage.yaml
kubectl delete -f network-policy-details.yaml
kubectl delete -f network-policy-reviews.yaml
kubectl delete -f network-policy-ratings.yaml
```

```
kubectl delete -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
kubectl delete pod curl-pod
```

混合节点的概念

借助 Amazon EKS 混合节点功能，您可以将在本地或边缘环境中运行的物理计算机或虚拟机加入到在 Amazon Cloud 中运行的 Amazon EKS 集群。这种方法提供了诸多优势，同时也为熟悉在单一网络环境中运行 Kubernetes 集群的用户引入了新的网络概念和架构。

以下各节将深入探讨 Kubernetes 和 EKS 混合节点的网络概念，并详细介绍流量如何流经混合架构。这些章节要求您熟悉基本的 Kubernetes 网络知识，例如容器组（pod）、节点、服务、Kubernetes 控制面板、kubelet 以及 kube-proxy 的概念。

建议按顺序阅读这些页面，从 the section called “联网概念” 开始，然后是 the section called “Kubernetes 概念”，最后是 the section called “流量”。

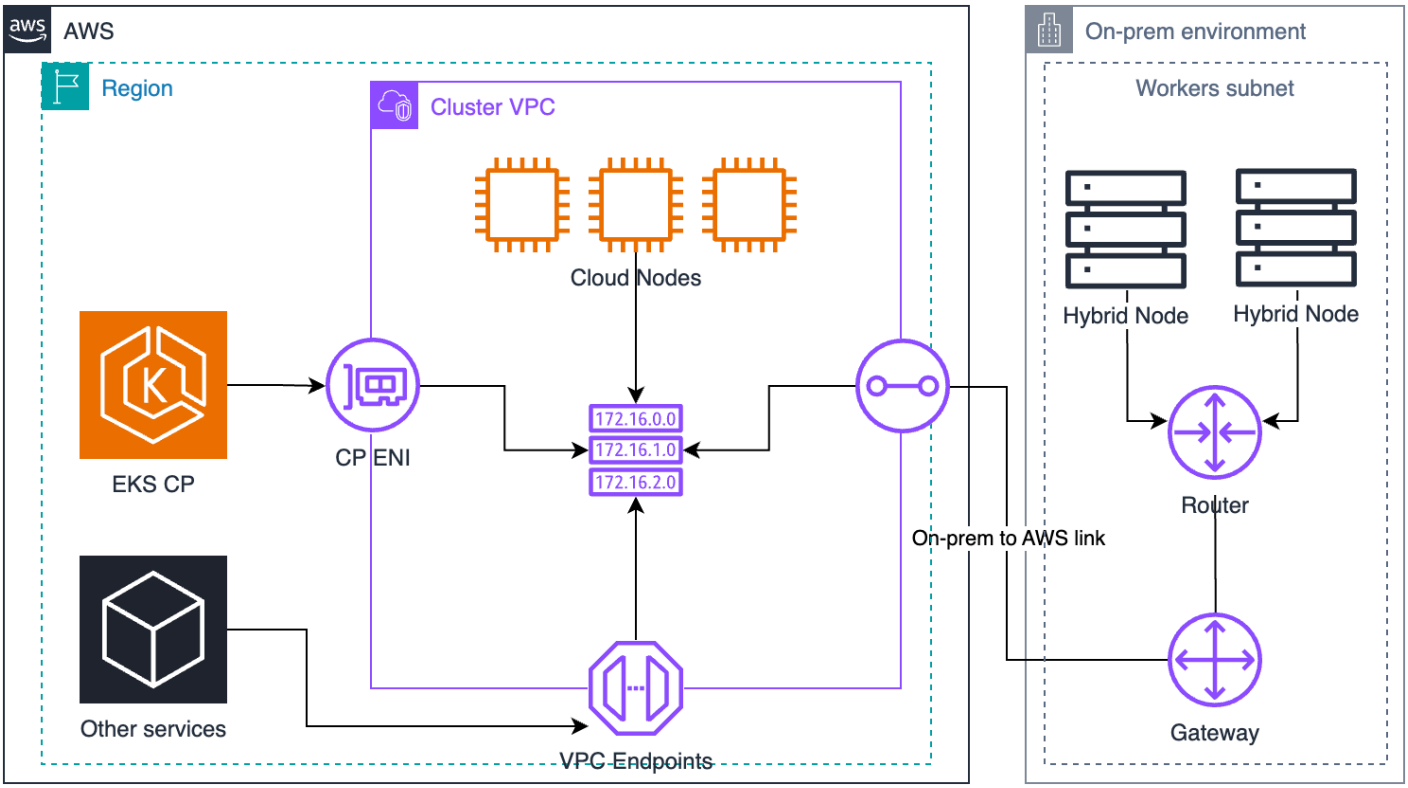
主题

- 混合节点的联网概念
- 混合节点的 Kubernetes 概念
- 混合节点的网络流量

混合节点的联网概念

本节详细介绍了核心联网概念以及在为 EKS 混合节点设计网络拓扑时必须考虑的限制。

EKS 混合节点的联网概念



作为网络中心的 VPC

所有穿过云边界的流量都会通过您的 VPC 进行路由。这包括从 EKS 控制面板或 Amazon 中运行的容器组（pod）流向混合节点或这些节点上运行的容器组（pod）的流量。您可以将集群的 VPC 视为混合节点和集群其余部分之间的网络中心。此架构让您可以完全控制流量及其路由，但您因此也有责任为 VPC 正确配置路由、安全组和防火墙。

EKS 控制面板到 VPC

EKS 控制面板将弹性网络接口（ENI）附加到您的 VPC 中。这些 ENI 处理流入和流出 EKS API 服务器的流量。在配置集群时，您可以控制 EKS 控制面板 ENI 的位置，因为 EKS 会将 ENI 附加到您在集群创建期间传递的子网。

EKS 将安全组与 EKS 附加到子网的 ENI 相关联。这些安全组允许通过 ENI 流入和流出 EKS 控制面板的流量。这对 EKS 混合节点很重要，因为您必须允许混合节点及其上运行的容器组（pod）的流量流向 EKS 控制面板 ENI。

远程节点网络

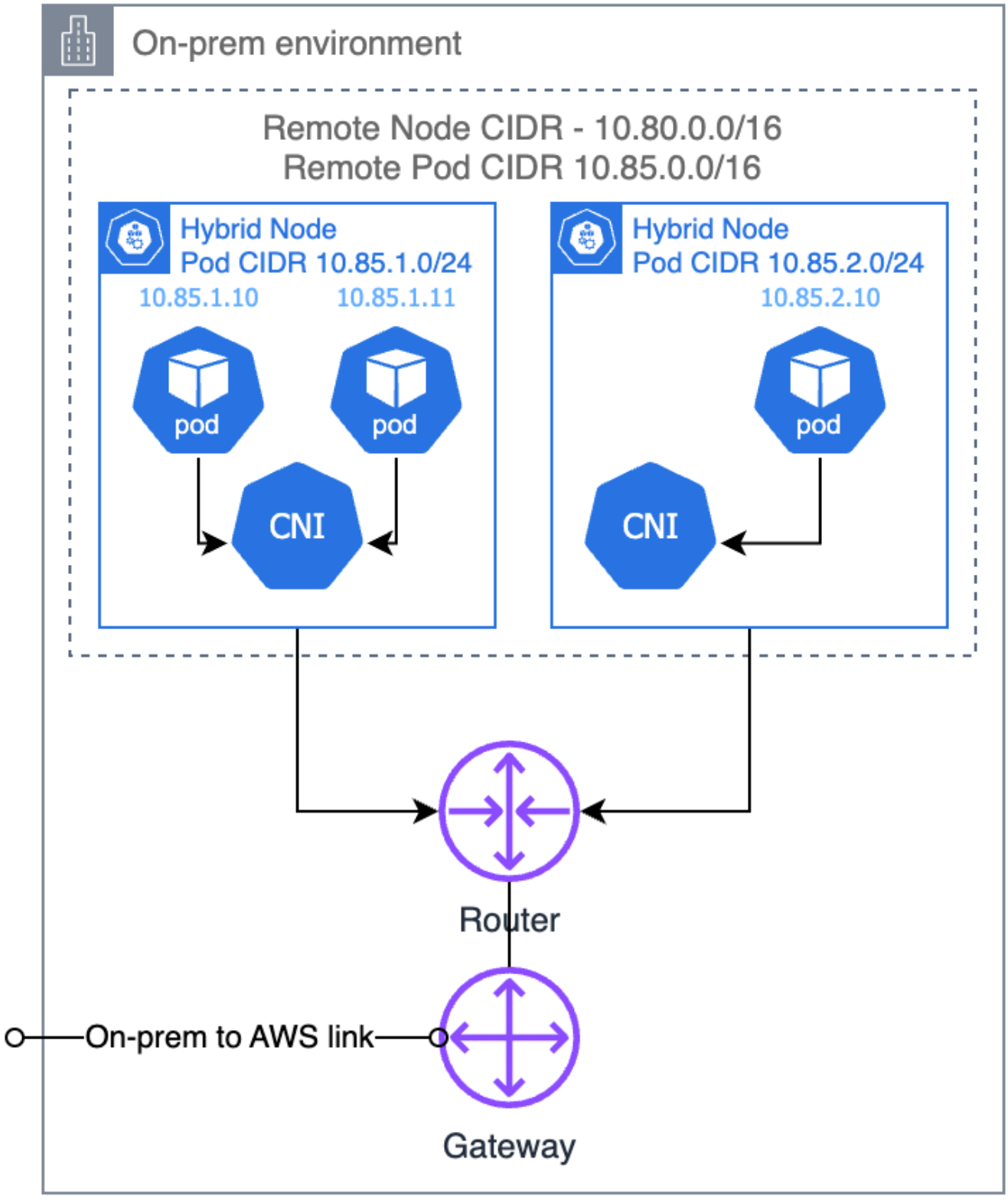
远程节点网络，特别是远程节点 CIDR，是分配给作为混合节点使用的计算机的 IP 范围。您预置混合节点时，它们位于您的本地数据中心或边缘站点，这与 EKS 控制面板和 VPC 属于不同的网络域。每个混合节点都有一个或多个来自远程节点 CIDR 的 IP 地址，该地址与 VPC 中的子网不同。

您可以使用这些远程节点 CIDR 配置 EKS 集群，这样 EKS 就会通过集群 VPC 路由所有发往混合节点 IP 的流量，例如对 kubelet API 的请求。与 `kubelet` API 的连接用于 `kubectl attach`、`kubectl cp`、`kubectl exec`、`kubectl logs` 和 `kubectl port-forward` 命令中。

远程容器组（pod）网络

远程容器组（pod）网络是分配给在混合节点上运行的容器组（pod）的 IP 范围。通常，您可以使用这些范围来配置 CNI，而 CNI 的 IP 地址管理（IPAM）功能负责将这些范围的切片分配给每个混合节点。您创建容器组（pod）时，CNI 会从分配给节点 [容器组（pod）在其中计划] 的切片中为该容器组分配 IP。

您可以使用这些远程容器组（pod）CIDR 配置 EKS 集群，这样 EKS 控制面板就会通过集群的 VPC 来路由所有发往混合节点上运行的容器组（pod）的流量，例如与 Webhook 的通信。


On-prem environment
Remote Node CIDR - 10.80.0.0/16
Remote Pod CIDR 10.85.0.0/16
Hybrid Node
Pod CIDR 10.85.1.0/24
10.85.1.10
10.85.1.11
pod
pod
CNI
Hybrid Node
Pod CIDR 10.85.2.0/24
10.85.2.10
pod
CNI
Router
On-prem to AWS link
Gateway

本地到 VPC

用于混合节点的本地网络必须路由到用于 EKS 集群的 VPC。有多种网络到 Amazon VPC 的连接选项可用来将本地网络连接到 VPC。您还可以使用自己的 VPN 解决方案。

务必在 VPC 和本地网络中的 Amazon 云端正确配置路由，这样两个网络便会通过各自的连接路由正确的流量。

在 VPC 中，所有流向远程节点和远程容器组（pod）网络的流量都必须通过连接路由到本地网络（称为“网关”）。如果您的某些子网具有不同的路由表，则必须使用混合节点的路由和在这些节点上运行的容器组（pod）来配置每个路由表。这适用于 ENI 附加到 EKS 控制面板的子网，以及包含必须与混合节点通信的 EC2 节点或容器组（pod）的子网。

在您的本地网络中，您必须将网络配置为允许流入和流出 EKS 集群 VPC 以及混合节点所需的其他 Amazon 服务的流量。EKS 集群的流量会双向穿过网关。

网络限制

完全路由的网络

主要限制在于 EKS 控制面板和所有节点，无论是云节点还是混合节点，都需要形成一个完全路由的网络。这意味着所有节点都必须能够通过 IP 地址在第三层相互访问。

EKS 控制面板和云节点已经可以相互访问，因为它们位于扁平化网络（VPC）中。但是，混合节点位于不同的网络域中。因此，您需要在 VPC 和本地网络中配置其他路由，以便在混合节点和集群的其余部分之间路由流量。如果混合节点可以相互访问并可从 VPC 访问，则您的混合节点可以位于一个扁平化网络中，也可以位于多个分段网络中。

可路由的远程容器组（pod）CIDR

为了让 EKS 控制面板与在混合节点（例如 Webhook 或 Metrics Server）上运行的容器组（pod）通信，或者让在云节点上运行的容器组（pod）与在混合节点上运行的容器组（pod）通信（工作负载东西向通信），远程容器组（pod）CIDR 必须可以从 VPC 路由。这意味着 VPC 必须能够通过网关将流量路由到容器组（pod）CIDR 再到本地网络，并且本地网络必须能够将容器组（pod）的流量路由到正确的节点。

请务必注意 VPC 和本地容器组（pod）路由要求之间的区别。VPC 只需要知道任何流向远程容器组（pod）的流量都应通过网关即可。如果您只有一个远程容器组（pod）CIDR，则只需要一条路由。

此要求适用于本地网络中的所有跃点，包括与混合节点位于同一子网中的本地路由器。这是唯一需要知道分配给每个节点的 pod CIDR 切片的路由器，这能确保特定容器组（pod）的流量传送到计划了容器组（pod）的节点。

您可以选择将本地容器组（pod）CIDR 的这些路由从本地路由器传播到 VPC 路由表，但这不是必需的。如果您的本地容器组（pod）CIDR 频繁更改，并且需要更新 VPC 路由表以反映不断变化的容器组（pod）CIDR，我们建议您将本地容器组（pod）CIDR 传播到 VPC 路由表，但这种情况并不常见。

注意，本地容器组（pod）CIDR 可路由的限制条件是可选的。如果您不需要在混合节点上运行 Webhook，也不需要让云节点上的容器组（pod）与混合节点上的容器组（pod）通信，则无需在本地网络上为容器组（pod）CIDR 配置路由。

为何本地容器组（pod）CIDR 需要可通过混合节点进行路由？

在云节点上使用 EKS 和 VPC CNI 时，VPC CNI 会直接从 VPC 向容器组（pod）分配 IP。这意味着无需任何特殊路由，因为云容器组（pod）和 EKS 控制面板都可以直接访问 Pod IP。

在本地运行（以及使用云中的其他 CNI 运行）时，容器组（pod）通常在隔离的覆盖网络中运行，CNI 负责在容器组之间传送流量。这通常通过封装来完成：CNI 将容器组（pod）到容器组（pod）的流量转换为节点到节点的流量，同时负责两端的封装和解封工作。这样一来，就无需在节点和路由器上进行额外配置。

与混合节点的联网是独一无二的，因为它结合了两种拓扑：EKS 控制面板和云节点（使用 VPC CNI）需要一个包含节点和容器组（pod）的扁平化网络，而在混合节点上运行的容器组（pod）则使用 VXLAN 进行封装（默认情况下在 Cilium 中），以处于叠加网络中。假设本地网络可以路由到 VPC，则在混合节点上运行的容器组（pod）可以访问 EKS 控制面板以及在云节点上运行的容器组（pod）。但是，如果本地网络上的容器组（pod）CIDR 没有路由，若网络不知道如何访问叠加网络并路由到正确的节点，则返回本地容器组（pod）IP 的任何流量最终都将被丢弃。

混合节点的 Kubernetes 概念

本页面详细介绍了支持 EKS 混合节点系统架构的关键 Kubernetes 概念。

VPC 中的 EKS 控制面板

EKS 控制面板 ENI 的 IP 存储在 `default` 命名空间的 `kubernetes` `Endpoints` 对象中。EKS 创建新的 ENI 或移除较旧的 ENI 时，EKS 会更新此对象，因此 IP 列表始终是最新的。

您可以通过 `kubernetes` 服务使用这些端点，也可以在 `default` 命名空间中使用。`ClusterIP` 类型的这种服务总是会获分配集群服务 CIDR 的第一个 IP。例如，对于服务 CIDR `172.16.0.0/16`，服务 IP 将为 `172.16.0.1`。

通常，这就是容器组（pod）（无论是在云端还是混合节点中运行）访问 EKS Kubernetes API 服务器的方式。容器组（pod）使用服务 IP 作为目标 IP，然后将其转换为其中一个 EKS 控制面板 ENI 的实际 IP。主要例外是 `kube-proxy`，因为它设置了转换。

EKS API 服务器端点

kubernetes 服务 IP 并不是访问 EKS API 服务器的唯一途径。您创建集群时，EKS 还会创建一个 Route53 DNS 名称。这是调用 EKS DescribeCluster API 操作时您的 EKS 集群的 endpoint 字段。

```
{
    "cluster": {
        "endpoint": "https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com",
        "name": "my-cluster",
        "status": "ACTIVE"
    }
}
```

在公有端点访问或公有和私有端点访问集群中，您的混合节点默认会将此 DNS 名称解析为可通过互联网路由的公有 IP。在私有端点访问集群中，DNS 名称解析为 EKS 控制面板 ENI 的私有 IP。

这就是 kubelet 和 kube-proxy 访问 Kubernetes API 服务器的方式。如果您希望所有 Kubernetes 集群流量都通过 VPC，则需要在私有访问模式下配置集群，或者修改本地 DNS 服务器，将 EKS 集群端点解析为 EKS 控制面板 ENI 的私有 IP。

kubelet 端点

kubelet 公开了多个 REST 端点，允许系统的其他部分与每个节点交互并从每个节点收集信息。在大多数集群中，流向 kubelet 服务器的大部分流量来自控制面板，但某些监控代理也可能与该服务器交互。

通过此接口，kubelet 可以处理各种请求：获取日志 (kubectl logs)、在容器内执行命令 (kubectl exec) 以及端口转发流量 (kubectl port-forward)。这些请求中的每一个都通过 kubelet 与底层容器运行时进行交互，在集群管理员和开发人员看来，这是一个无缝的过程。

此 API 最常见的使用器是 Kubernetes API 服务器。您使用前面提到的任何 kubectl 命令时，kubectl 会向 API 服务器发出 API 请求，然后服务器调用运行容器组（pod）的节点的 kubelet API。这就是需要从 EKS 控制面板访问节点 IP 的主要原因，也解释了为什么即使您的容器组（pod）正在运行，但如果节点路由配置错误，您也无法访问它们的日志或 exec。

节点 IP

EKS 控制面板与节点通信时，它会使用 Node 对象状态 (status.addresses) 中报告的地址之一。

对于 EKS 云节点，kubelet 通常会在节点注册期间将 EC2 实例的私有 IP 报告为 `InternalIP`。然后，云控制器管理器（CCM）会验证此 IP，确保它属于 EC2 实例。此外，CCM 通常会将实例的公有 IP（作为 `ExternalIP`）和 DNS 名称（`InternalDNS` 和 `ExternalDNS`）添加到节点状态。

但是，没有适用于混合节点的 CCM。您向 EKS 混合节点 CLI (`nodeadm`) 注册混合节点时，它会将 kubelet 配置为直接以节点状态报告计算机的 IP，而不使用 CCM。

```
apiVersion: v1
kind: Node
metadata:
  name: my-node-1
spec:
  providerID: eks-hybrid:///us-west-2/my-cluster/my-node-1
status:
  addresses:
  - address: 10.1.1.236
    type: InternalIP
  - address: my-node-1
    type: Hostname
```

如果您的计算机有多个 IP，kubelet 会按照自己的逻辑选择其中一个 IP。您可以使用 `--node-ip` 标志控制选定的 IP，并且可以在 `spec.kubelet.flags` 的 `nodeadm` 配置中传入该标志。只有 `Node` 对象中报告的 IP 需要来自 VPC 的路由。您的计算机可能有无法从云端访问的其他 IP。

kube-proxy

`kube-proxy` 负责在每个节点的网络层实现服务抽象。它充当流向 Kubernetes 服务的流量的网络代理和负载均衡器。通过持续监控 Kubernetes API 服务器中与服务和端点相关的更改，`kube-proxy` 会动态更新底层主机的网络规则，确保流量得到正确引导。

在 `iptables` 模式下，`kube-proxy` 会对多个 `netfilter` 链进行编程以处理服务流量。这些规则构成以下层次结构：

1. KUBE-SERVICES 链：所有服务流量的入口点。它具有与每项 `ClusterIP` 服务和端口匹配的规则。
2. KUBE-SVC-XXX 链：特定于服务的链对每项服务都有负载均衡规则。
3. KUBE-SEP-XXX 链：特定于端点的链具有实际的 `DNAT` 规则。

让我们来看看 default 命名空间中的 test-server 服务会发生什么：* 服务 ClusterIP：172.16.31.14 * 服务端口：80 * 支持性容器组（pod）：10.2.0.110、10.2.1.39 和 10.2.2.254

检查 iptables 规则（使用 iptables-save 0 grep -A10 KUBE-SERVICES）时：

1. 在 KUBE-SERVICES 链中，我们找到了一条与该服务匹配的规则：

```
-A KUBE-SERVICES -d 172.16.31.14/32 -p tcp -m comment --comment "default/test-server
 cluster IP" -m tcp --dport 80 -j KUBE-SVC-XYZABC123456
```

   - 此规则与发往 172.16.31.14:80 的数据包匹配
   - 注释指出了此规则的用途：default/test-server cluster IP
   - 匹配的数据包跳转到 KUBE-SVC-XYZABC123456 链
2. KUBE-SVC-XYZABC123456 链具有基于概率的负载均衡规则：

```
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.33333333349 -j
 KUBE-SEP-POD1XYZABC
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.50000000000 -j
 KUBE-SEP-POD2XYZABC
-A KUBE-SVC-XYZABC123456 -j KUBE-SEP-POD3XYZABC
```

   - 第一条规则：有 33.3% 的概率跳转到 KUBE-SEP-POD1XYZABC
   - 第二条规则：剩余流量（占总流量的 33.3%）有 50% 的概率跳转到 KUBE-SEP-POD2XYZABC
   - 最后一条规则：所有剩余流量（占总流量的 33.3%）都将跳转到 KUBE-SEP-POD3XYZABC
3. 各个 KUBE-SEP-XXX 链都执行 DNAT（目标 NAT）：

```
-A KUBE-SEP-POD1XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.0.110:80
-A KUBE-SEP-POD2XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.1.39:80
-A KUBE-SEP-POD3XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.2.254:80
```

   - 这些 DNAT 规则会重写目标 IP 和端口，将流量引导至特定容器组（pod）。
   - 每条规则处理大约 33.3% 的流量，从而在 10.2.0.110、10.2.1.39 和 10.2.2.254 之间提供均衡的负载均衡。

这种多级链结构让 kube-proxy 能够通过内核级别的数据包操作高效地实现服务负载均衡和重定向，而无需在数据路径中使用代理进程。

对 Kubernetes 操作的影响

节点上损坏的 kube-proxy 会阻止该节点正确路由服务流量，导致依赖集群服务的容器组（pod）超时或连接失败。首次注册节点时，这可能会造成特别大的干扰。CNI 需要先与 Kubernetes API 服务器通信以获取信息，例如节点的容器组（pod）CIDR，然后才能配置任何容器组（pod）网络。为此，它使用 kubernetes 服务 IP。但是，如果 kube-proxy 无法启动或未能设置正确的 iptables 规则，则发往 kubernetes 服务 IP 的请求不会转换为 EKS 控制面板 ENI 的实际 IP。因此，CNI 将进入崩溃循环，所有的容器组（pod）都无法正常运行。

我们知道容器组（pod）使用 kubernetes 服务 IP 与 Kubernetes API 服务器通信，但 kube-proxy 需要先设置 iptables 规则才能使其正常工作。

kube-proxy 如何与 API 服务器通信？

kube-proxy 必须配置为使用 Kubernetes API 服务器的实际 IP 或解析为它们的 DNS 名称。对于 EKS，EKS 将默认 kube-proxy 配置为指向您在创建集群时 EKS 创建的 Route53 DNS 名称。您可以在 kube-system 命名空间的 kube-proxy ConfigMap 中看到此值。此 ConfigMap 的内容是注入到 kube-proxy 容器组（pod）中的 kubeconfig，因此请寻找 clusters0.cluster.server 字段。此值将与 EKS 集群的 endpoint 字段匹配（在调用 EKS DescribeCluster API 时）。

```
apiVersion: v1
data:
  kubeconfig: |-
    kind: Config
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        server: https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com
      name: default
    contexts:
    - context:
        cluster: default
        namespace: default
        user: default
      name: default
    current-context: default
    users:
    - name: default
      user:
        tokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
```

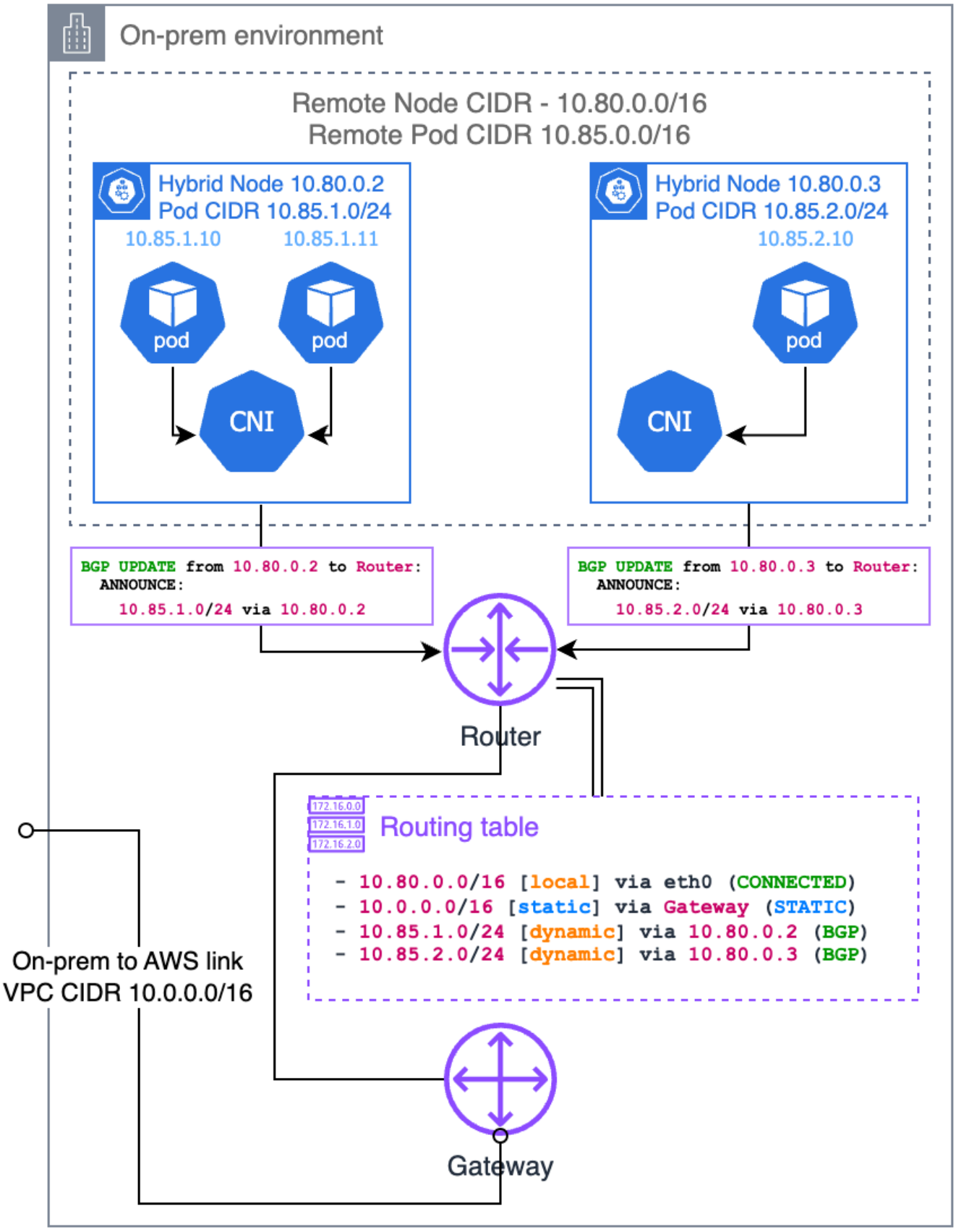
```
kind: ConfigMap
metadata:
  name: kube-proxy
  namespace: kube-system
```

可路由的远程容器组（pod）CIDR

the section called “联网概念” 页面详细说明了在混合节点上运行 Webhook 或让在云节点上运行的容器组（pod）与在混合节点上运行的容器组（pod）通信的要求。关键要求是，本地路由器需要知道哪个节点负责特定容器组（pod）IP。有多种方法可以实现这一点，包括边界网关协议（BGP）、静态路由和地址解析协议（ARP）代理。以下部分将介绍这些方法。

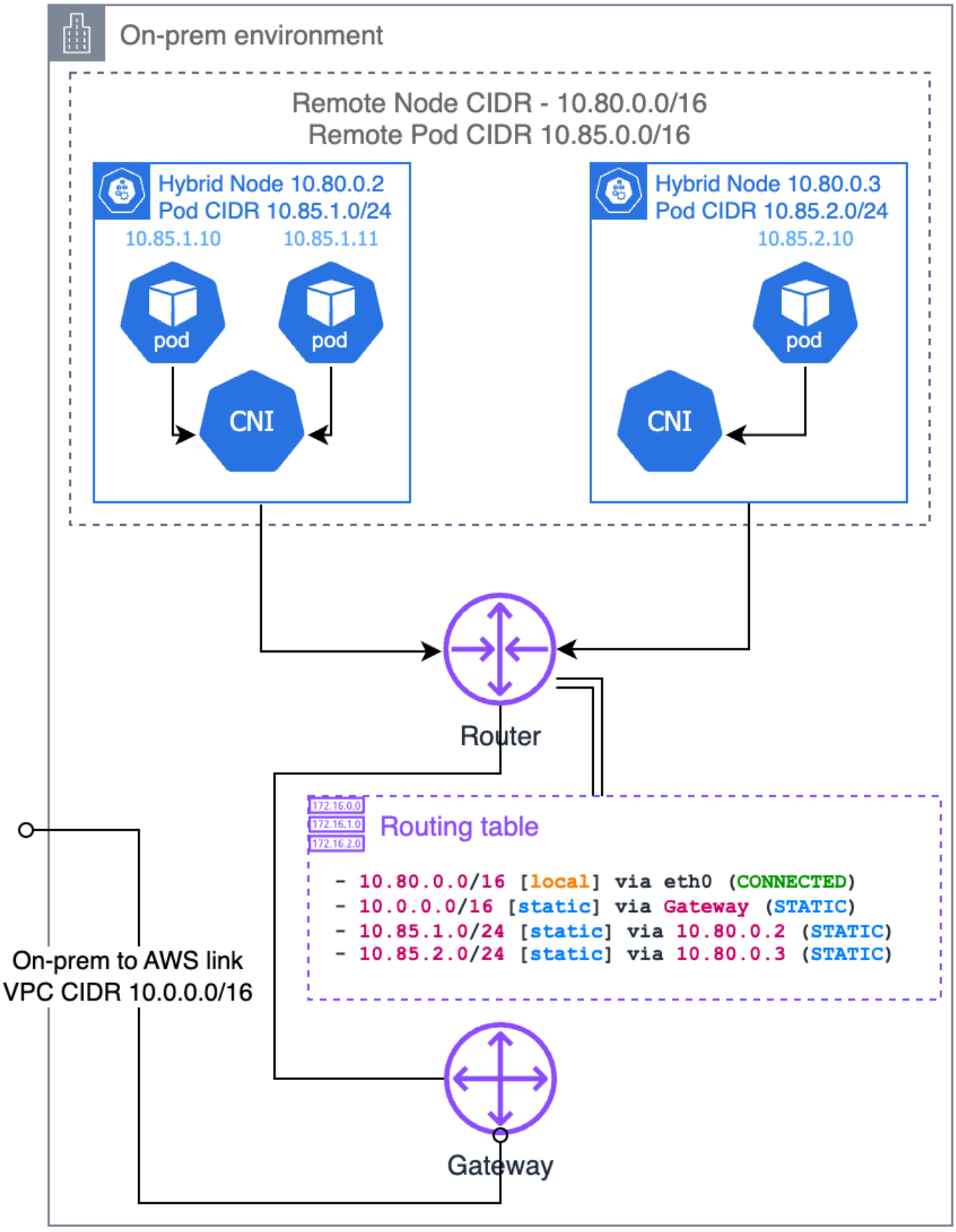
边界网关协议（BGP）

如果您的 CNI 支持这种协议（例如 Cilium 和 Calico），您可以使用 CNI 的 BGP 模式，将每节点容器组（pod）CIDR 的路由从节点传播到本地路由器。使用 CNI 的 BGP 模式时，您的 CNI 充当虚拟路由器，因此本地路由器认为容器组（pod）CIDR 属于不同的子网，而您的节点是该子网的网关。


On-prem environment
Remote Node CIDR - 10.80.0.0/16
Remote Pod CIDR 10.85.0.0/16
Hybrid Node 10.80.0.2
Pod CIDR 10.85.1.0/24
10.85.1.10
10.85.1.11
pod
pod
CNI
Hybrid Node 10.80.0.3
Pod CIDR 10.85.2.0/24
10.85.2.10
pod
CNI
BGP UPDATE from 10.80.0.2 to Router:
ANNOUNCE:
10.85.1.0/24 via 10.80.0.2
BGP UPDATE from 10.80.0.3 to Router:
ANNOUNCE:
10.85.2.0/24 via 10.80.0.3
Router
172.16.0.0
172.16.1.0
172.16.2.0
Routing table
- 10.80.0.0/16 [local] via eth0 (CONNECTED)
- 10.0.0.0/16 [static] via Gateway (STATIC)
- 10.85.1.0/24 [dynamic] via 10.80.0.2 (BGP)
- 10.85.2.0/24 [dynamic] via 10.80.0.3 (BGP)
On-prem to AWS link
VPC CIDR 10.0.0.0/16
Gateway

静态路由

您也可以在本地路由器中配置静态路由。这是将本地容器组（pod）CIDR 路由到您的 VPC 的最简单方法，但这也是最容易出错且最难维护的方法。您需要确保现有节点及其分配的容器组（pod）CIDR 的路由始终是最新的。如果您的节点数量很少且基础设施是静态的，那么这是一个可行的选择，并且无需在路由器中支持 BGP。如果您选择这样做，建议使用要分配给每个节点的容器组（pod）CIDR 切片来配置 CNI，而不是让其 IPAM 决定。


On-prem environment
Remote Node CIDR - 10.80.0.0/16
Remote Pod CIDR 10.85.0.0/16
Hybrid Node 10.80.0.2
Pod CIDR 10.85.1.0/24
10.85.1.10
10.85.1.11
pod
pod
CNI
Hybrid Node 10.80.0.3
Pod CIDR 10.85.2.0/24
10.85.2.10
pod
CNI
Router
172.16.0.0
172.16.1.0
172.16.2.0
Routing table
- 10.80.0.0/16 [local] via eth0 (CONNECTED)
- 10.0.0.0/16 [static] via Gateway (STATIC)
- 10.85.1.0/24 [static] via 10.80.0.2 (STATIC)
- 10.85.2.0/24 [static] via 10.80.0.3 (STATIC)
On-prem to AWS link
VPC CIDR 10.0.0.0/16
Gateway

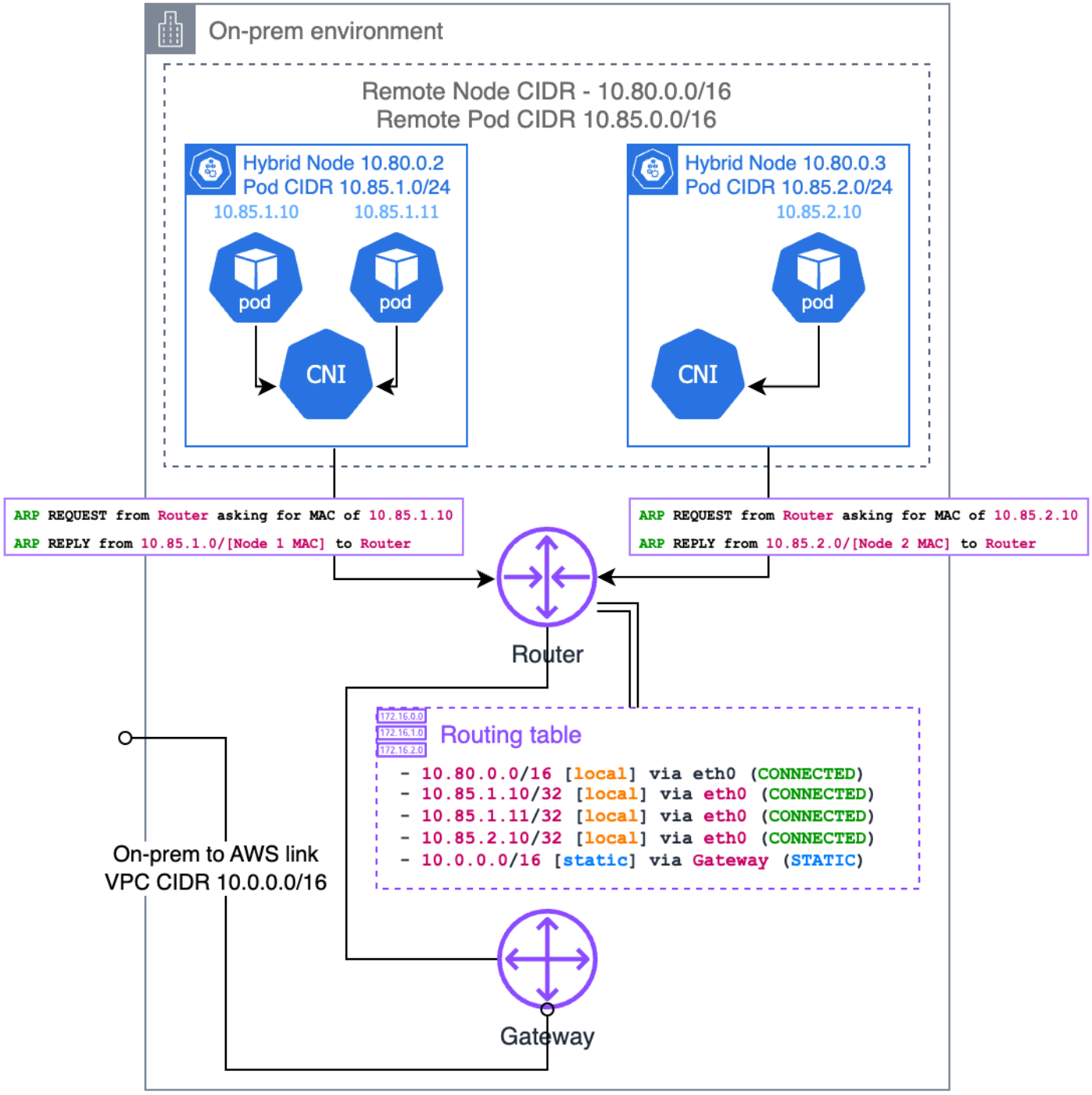
地址解析协议（ARP）代理

ARP 代理是使本地容器组（pod）IP 可路由的另一种方法，当混合节点与本地路由器位于同一个第 2 层网络上时，这种方法特别有用。启用 ARP 代理后，节点会响应其托管的容器组（pod）IP 的 ARP 请求，即使这些 IP 属于不同的子网，也会如此。

本地网络上的设备尝试访问容器组（pod）IP 时，它会首先发送 ARP 请求，询问“谁拥有此 IP？”。托管该容器组（pod）的混合节点将使用自己的 MAC 地址进行响应，表示“我可以处理该 IP 的流量。”这无需配置路由器，即可在本地网络上的设备和容器组（pod）之间创建直接路径。

要使此方法起作用，您的 CNI 必须支持代理 ARP 功能。Cilium 内置了对代理 ARP 的支持，您可以通过配置启用该功能。关键考虑因素是容器组（pod）CIDR 不得与环境中的任何其他网络重叠，因为这可能会导致路由冲突。

这种方法有几个优点：* 无需为路由器配置 BGP，也无需维护静态路由 * 在路由器配置不由您控制的环境中效果良好



容器组（pod）到容器组（pod）封装

在本地环境中，CNI 通常使用封装协议来创建叠加网络，这种网络无需重新配置即可在物理网络之上运行。此部分介绍了这种封装的工作原理。请注意，根据您使用的 CNI，某些细节可能会有所不同。

封装将原始容器组（pod）网络数据包包装在另一个网络数据包中，该数据包可以通过底层物理网络路由。这允许容器组（pod）在运行相同 CNI 的节点之间进行通信，物理网络不必理解如何路由这些容器组（pod）CIDR。

Kubernetes 中最常用的封装协议是虚拟可扩展局域网（VXLAN），但也可使用其他封装协议（例如 Geneve），具体取决于您的 CNI。

VXLAN 封装

VXLAN 将第 2 层以太网帧封装在 UDP 数据包中。当一个容器组（pod）向不同节点上的另一个容器组（pod）发送流量时，CNI 会执行以下操作：

1. CNI 拦截来自容器组（pod）A 的数据包
2. CNI 将原始数据包封装在 VXLAN 标头中
3. 然后，这个封装后的数据包通过节点的常规网络堆栈发送到目标节点
4. 目标节点上的 CNI 解开数据包并将其传送到容器组（pod）B

以下是 VXLAN 封装期间，数据包结构的情况：

容器组（pod）到容器组（pod）的原始数据包：

```
+-----------------+---------------+-------------+-----------------+
| Ethernet Header | IP Header     | TCP/UDP     | Payload         |
| Src: Pod A MAC  | Src: Pod A IP | Src Port    |                 |
| Dst: Pod B MAC  | Dst: Pod B IP | Dst Port    |                 |
+-----------------+---------------+-------------+-----------------+
```

VXLAN 封装后：

```
+-----------------+-------------+--------------+------------
+---------------------------+
| Outer Ethernet  | Outer IP    | Outer UDP    | VXLAN      | Original Pod-to-Pod
 |
| Src: Node A MAC | Src: Node A | Src: Random  | VNI: xx    | Packet (unchanged
 |
| Dst: Node B MAC | Dst: Node B | Dst: 4789    |            | from above)
 |
+-----------------+-------------+--------------+------------
+---------------------------+
```

VXLAN 网络标识符（VNI）用于区分不同的叠加网络。

容器组（pod）通信场景

同一个混合节点上的容器组（pod）

同一个混合节点上的容器组（pod）通信时，通常不需要封装。CNI 设置本地路由，通过节点的内部虚拟接口引导容器组（pod）之间的流量：

```
Pod A -> veth0 -> node's bridge/routing table -> veth1 -> Pod B
```

数据包永远不会离开节点，也不需要封装。

不同混合节点上的容器组（pod）

不同混合节点上的容器组（pod）之间的通信需要封装：

```
Pod A -> CNI -> [VXLAN encapsulation] -> Node A network -> router or gateway -> Node B
 network -> [VXLAN decapsulation] -> CNI -> Pod B
```

这样，容器组（pod）流量就可以遍历物理网络基础架构，而物理网络不必理解容器组（pod）IP 路由。

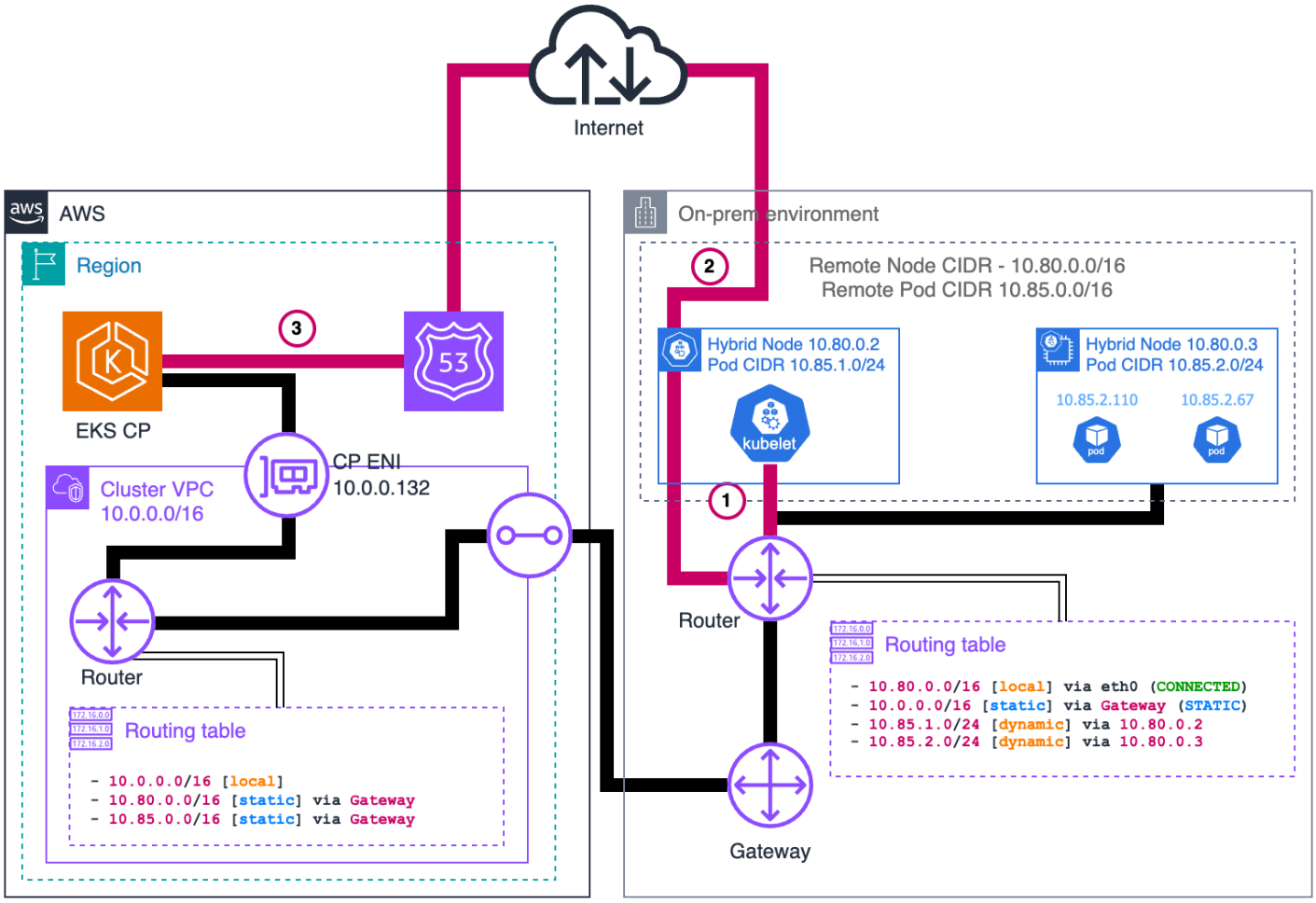
混合节点的网络流量

本页面详细介绍了 EKS 混合节点的网络流量，其中的图表显示了不同流量类型的端到端网络路径。

涵盖以下流量：

- the section called “混合节点 kubelet 到 EKS 控制面板”
- the section called “EKS 控制面板到混合节点（kubelet 服务器）”
- the section called “混合节点上运行的容器组（pod）到 EKS 控制面板”
- the section called “EKS 控制面板到混合节点上运行的容器组（pod）（Webhook）”
- the section called “在混合节点上运行的容器组（pod）到容器组（pod）”
- the section called “云节点上的容器组（pod）到混合节点上的容器组（pod）（东西向流量）”

混合节点 **kubelet** 到 EKS 控制面板



请求

1**kubelet**. 发起请求

混合节点上的 kubelet 需要与 EKS 控制面板通信 [例如，报告节点状态或获取容器组（pod）规格] 时，它将使用节点注册期间提供的 kubeconfig 文件。此 kubeconfig 具有 API 服务器端点 URL（Route53 DNS 名称），而非直接 IP 地址。

kubelet 对端点执行 DNS 查找（例如 https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-west-2.eks.amazonaws.com ）。在公有访问集群中，这会解析为属于 Amazon 中运行的 EKS 服务的某个公有 IP 地址（如 54.239.118.52 ）。然后，kubelet 创建针对此端点的安全 HTTPS 请求。初始数据包类似于以下示例：

```
+--------------------+---------------------+-----------------+
```

```
| IP Header           | TCP Header           | Payload         |
| Src: 10.80.0.2      | Src: 52390 (random) |                 |
| Dst: 54.239.118.52 | Dst: 443             |                 |
+--------------------+----------------------+-----------------+
```

2. 本地路由器路由

目标 IP 是公有 IP 地址而不是本地网络的一部分，因此 kubelet 会将此数据包发送到其默认网关（本地路由器）。路由器检查目标 IP 并确定它是公有 IP 地址。

对于公共流量，路由器通常会将数据包转发到处理互联网出站流量的互联网网关或边界路由器。图表中省略了这一点，并且这取决于您的本地网络设置方式。数据包会穿过您的本地网络基础设施，最终到达互联网服务提供商的网络。

3. 分发到 EKS 控制面板

数据包通过公共互联网和传输网络进行传输，直至到达 Amazon 网络。Amazon 网络将数据包路由到相应区域的 EKS 服务端点。数据包到达 EKS 服务时，它会被转发到集群的实际 EKS 控制面板。

这种通过公共互联网进行的路由不同于我们在其他流量流中看到的私有 VPC 路由路径。主要区别在于，在使用公共访问模式时，从本地 kubelet [尽管不是从容器组（pod）] 到 EKS 控制面板的流量不会通过您的 VPC，而是使用全球互联网基础设施。

响应

在 EKS 控制面板处理 kubelet 请求后，它会发送回响应：

3. EKS 控制面板发送响应

EKS 控制面板创建响应数据包。此数据包以该公有 IP 为源，并以混合节点的 IP 为目标：

```
+--------------------+----------------------+-----------------+
| IP Header           | TCP Header           | Payload         |
| Src: 54.239.118.52 | Src: 443             |                 |
| Dst: 10.80.0.2      | Dst: 52390           |                 |
+--------------------+----------------------+-----------------+
```

2. 互联网路由

响应数据包沿着互联网服务提供商确定的路由路径，通过互联网返回，直至到达本地网络边缘路由器。

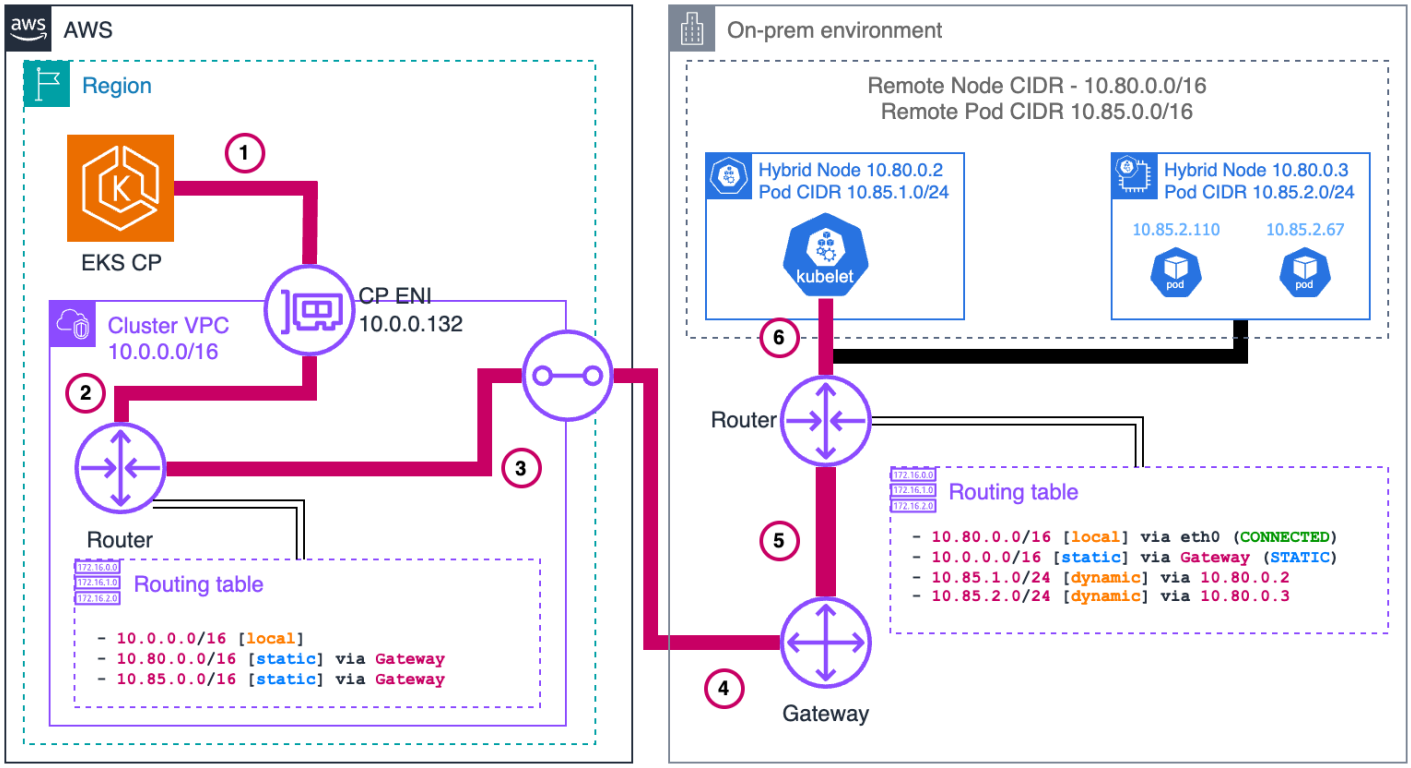
1。本地分发

本地路由器会收到数据包并将目标 IP (10.80.0.2) 识别为属于本地网络。它通过本地网络基础设施转发数据包，直至到达目标混合节点，kubelet 在该节点接收和处理响应。

混合节点 **kube-proxy** 到 EKS 控制面板

如果为集群启用了公有端点访问权限，则返回流量会使用公共互联网。此流量从混合节点上的 kube-proxy 出发流向 EKS 控制面板，其路径与从 kubelet 流向 EKS 控制面板的流量相同。

EKS 控制面板到混合节点（**kubelet** 服务器）



请求

1。EKS Kubernetes API 服务器启动请求

EKS Kubernetes API 服务器从节点对象的状态中检索节点的 IP 地址 (10.80.0.2)。然后通过 VPC 中的 ENI 路由此请求，因为目标 IP 属于配置的远程节点 CIDR (10.80.0.0/16)。初始数据包类似于以下示例：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 67493 (random) |                 |
| Dst: 10.80.0.2  | Dst: 10250          |                 |
+-----------------+---------------------+-----------------+
```

2. VPC 网络处理

数据包离开 ENI 并进入 VPC 网络层，在此处它会被引导到子网的网关进行进一步路由。

3. VPC 路由表查找

包含 EKS 控制面板 ENI 的子网的 VPC 路由表具有远程节点 CIDR 的专用路由（图中的第二条路由）。根据此路由规则，数据包将被引导至“VPC 到本地”网关。

4. 跨边界传输

网关通过您建立的连接（例如 Direct Connect 或 VPN）将数据包跨云边界传输到本地网络。

5. 本地网络接收

数据包到达本地路由器，该路由器处理混合节点所在子网的流量。

6. 最终分发

本地路由器确定目标 IP (`10.80.0.2`) 地址属于其直连网络，并将数据包直接转发到目标混合节点，而 `kubelet` 在该节点接收和处理请求。

响应

在混合节点的 `kubelet` 处理请求后，它会按照相同的路径反向发送响应：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 10250          |                 |
| Dst: 10.0.0.132 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

6. **`kubelet`** 发送响应

混合节点 (`10.80.0.2`) 上的 `kubelet` 创建以原始源 IP 为目标的响应数据包。该目标不属于本地网络，因此它会发送到主机的默认网关，即本地路由器。

5. 本地路由器路由

路由器确定目标 IP (`10.0.0.132`) 属于 `10.0.0.0/16`，其中具有指向连接到 Amazon 的网关的路由。

4. 跨边界返回

数据包通过同一个本地与 VPC 的连接（例如 Direct Connect 或 VPN）返回，朝相反的方向穿过云边界。

3. VPC 路由

数据包到达 VPC 时，路由表确定目标 IP 属于某个 VPC CIDR。数据包将在该 VPC 内路由。

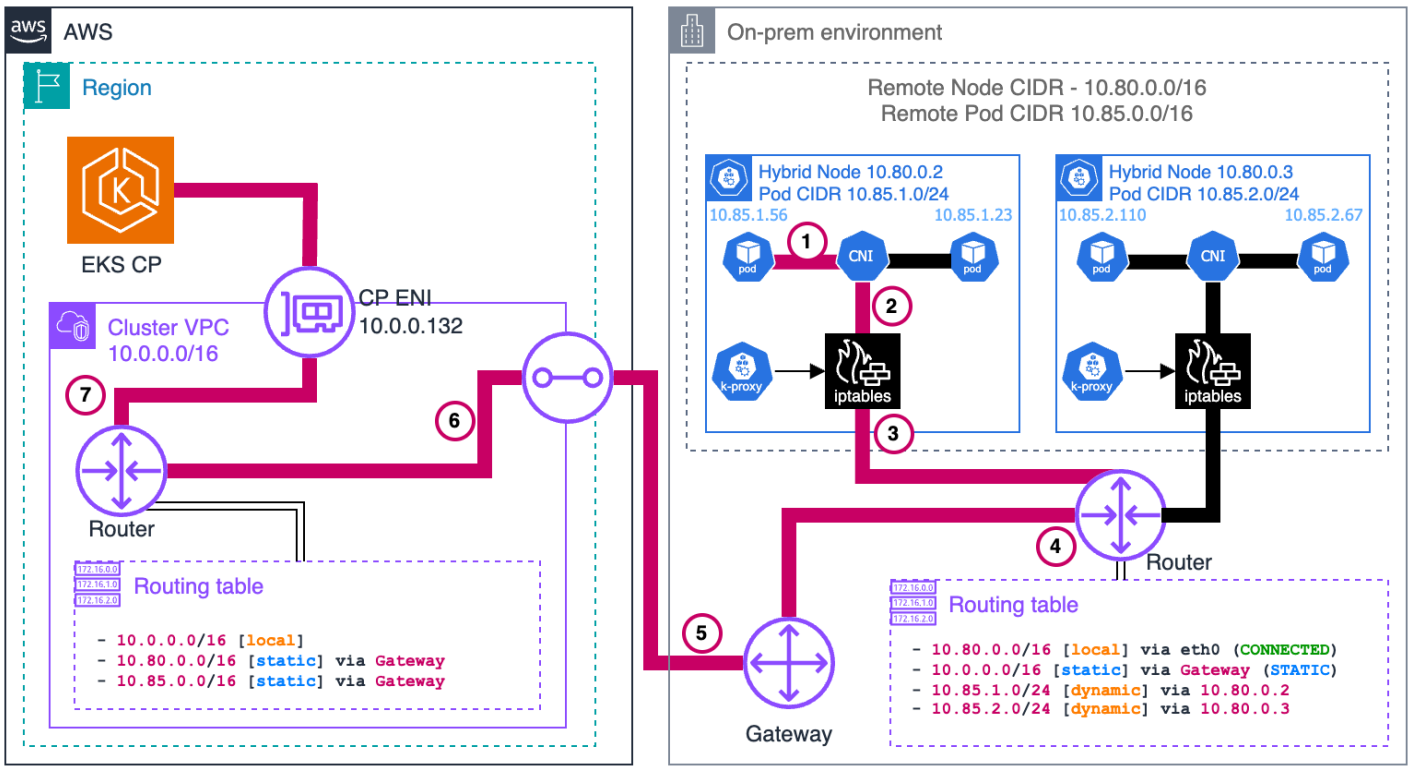
2. VPC 网络分发

VPC 网络层使用 EKS 控制面板 ENI (`10.0.0.132`) 将数据包转发到子网。

1。ENI 接收

数据包到达附加到 Kubernetes API 服务器的 EKS 控制面板 ENI，从而完成往返行程。

混合节点上运行的容器组（pod）到 EKS 控制面板



不使用 CNI NAT

请求

容器组（pod）通常通过 `kubernetes` 服务与 Kubernetes API 服务器通信。服务 IP 是集群的服务 CIDR 的第一个 IP。此惯例允许需要在 CoreDNS 可用之前运行的容器组（pod）访问 API 服务器，例

如 CNI。请求离开以服务 IP 为目标的容器组（pod）。例如，如果服务 CIDR 是 172.16.0.0/16，则服务 IP 将为 172.16.0.1。

1。容器组（pod）发起请求

容器组（pod）从随机源端口向 API 服务器端口（443）上的 kubernetes 服务 IP（172.16.0.1）发送请求。数据包类似于以下示例：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

2. CNI 处理

CNI 检测到目标 IP 不属于其管理的任何容器组（pod）CIDR。由于传出 NAT 已禁用，CNI 会将数据包传递到主机网络堆栈，而不会对其进行修改。

3. 节点网络处理

数据包进入节点的网络堆栈，netfilter 钩子在该堆栈中触发 kube-proxy 设置的 iptables 规则。有多条规则按以下顺序适用：

1. 数据包首先到达 KUBE-SERVICES 链，链中包含与每项服务的 ClusterIP 和端口匹配的规则。
2. 匹配的规则跳转到包含负载均衡规则的 kubernetes 服务（发往 172.16.0.1:443 的数据包）的 KUBE-SVC-XXX 链。
3. 负载均衡规则会为控制面板 ENI IP（10.0.0.132 或 10.0.1.23）随机选择一条 KUBE-SEP-XXX 链。
4. 所选 KUBE-SEP-XXX 链具有将目标 IP 从服务 IP 更改为所选 IP 的实际规则。这称为目标网络地址转换（DNAT）。

应用这些规则后，假设所选 EKS 控制面板 ENI 的 IP 是 10.0.0.132，则数据包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 10.0.0.132 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

由于目标 IP 不在本地网络中，节点会将数据包转发到其默认网关。

4. 本地路由器路由

本地路由器确定目标 IP (10.0.0.132) 属于 VPC CIDR (10.0.0.0/16)，并将其转发到连接至 Amazon 的网关。

5. 跨边界传输

数据包通过您建立的连接（例如 Direct Connect 或 VPN）跨云边界传输到 VPC。

6. VPC 网络分发

VPC 网络层将数据包路由到 EKS 控制面板 ENI (10.0.0.132) 所在的正确子网。

7. ENI 接收

数据包到达附加到 Kubernetes API 服务器的 EKS 控制面板 ENI。

响应

在 EKS 控制面板处理请求后，它会将响应发送回容器组（pod）：

7. API 服务器发送响应

EKS Kubernetes API 服务器会创建以原始源 IP 为目标的响应数据包。数据包类似于以下示例：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

由于目标 IP 属于配置的远程容器组 CIDR (10.85.0.0/16)，它通过 VPC 中的 ENI 将其发送，并将子网的路由器作为下一个跃点。

6. VPC 路由

VPC 路由表包含远程容器组 CIDR (10.85.0.0/16) 的条目，该条目将此流量引导至“VPC 到本地”网关。

5. 跨边界传输

网关通过您建立的连接（例如 Direct Connect 或 VPN）将数据包跨云边界传输到本地网络。

4. 本地网络接收

数据包到达您的本地路由器。

3. 分发到节点

路由器的表中有一个将 10.80.0.2 作为下一个跃点的 10.85.1.0/24 条目，该条目将数据包分发到我们的节点。

2. 节点网络处理

当数据包由节点的网络堆栈处理时，conntrack（netfilter 的一部分）会将数据包与容器组最初建立的连接进行比对。由于最初应用了 DNAT，因此 conntrack 会通过将源 IP 从 EKS 控制面板 ENI 的 IP 重写为 kubernetes 服务 IP，从而逆转 DNAT：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

1。CNI 处理

CNI 确定目标 IP 属于其网络中的某个容器组（pod），并将数据包分发到正确的容器组（pod）网络命名空间。

此流程展示了为何远程容器组（pod）CIDR 必须能够从 VPC 正确路由到托管每个容器组（pod）的特定节点 – 整个返回路径取决于云和本地网络中容器组（pod）IP 的正确路由。

使用 CNI NAT

此流程与不使用 CNI NAT 的流程非常相似，但有一个主要区别：CNI 会在将数据包发送到节点的网络堆栈之前对数据包应用源 NAT（SNAT）。这会将数据包的源 IP 更改为节点的 IP，从而无需进行额外的路由配置即可将数据包路由回节点。

请求

1。容器组（pod）发起请求

容器组（pod）从随机源端口向 EKS Kubernetes API 服务器端口（443）上的 kubernetes 服务 IP（172.16.0.1）发送请求。数据包类似于以下示例：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

2. CNI 处理

CNI 检测到目标 IP 不属于其管理的任何容器组（pod）CIDR。由于传出 NAT 已启用，CNI 会对数据包应用 SNAT，将源 IP 更改为节点的 IP，然后再将其传递到节点的网络堆栈：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

注意：为清楚起见，CNI 和 iptables 在示例中显示为单独的块，但实际上，某些 CNI 可能使用 iptables 来应用 NAT。

3. 节点网络处理

在这里，由 kube-proxy 设置的 iptables 规则的行为与之前的示例相同，即将数据包负载均衡到其中一个 EKS 控制面板 ENI。现在的数据包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 67493 (random) |                 |
| Dst: 10.0.0.132 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

由于目标 IP 不在本地网络中，节点会将数据包转发到其默认网关。

4. 本地路由器路由

本地路由器确定目标 IP (10.0.0.132) 属于 VPC CIDR (10.0.0.0/16)，并将其转发到连接至 Amazon 的网关。

5. 跨边界传输

数据包通过您建立的连接（例如 Direct Connect 或 VPN）跨云边界传输到 VPC。

6. VPC 网络分发

VPC 网络层将数据包路由到 EKS 控制面板 ENI (10.0.0.132) 所在的正确子网。

7. ENI 接收

数据包到达附加到 Kubernetes API 服务器的 EKS 控制面板 ENI。

响应

在 EKS 控制面板处理请求后，它会将响应发送回容器组（pod）：

7. API 服务器发送响应

EKS Kubernetes API 服务器会创建以原始源 IP 为目标的响应数据包。数据包类似于以下示例：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 443            |                 |
| Dst: 10.80.0.2  | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

由于目标 IP 属于配置的远程节点 CIDR (10.80.0.0/16)，它通过 VPC 中的 ENI 将其发送，并将子网的路由器作为下一个跃点。

6. VPC 路由

VPC 路由表包含远程节点 CIDR (10.80.0.0/16) 的条目，该条目将此流量引导至“VPC 到本地”网关。

5. 跨边界传输

网关通过您建立的连接（例如 Direct Connect 或 VPN）将数据包跨云边界传输到本地网络。

4. 本地网络接收

数据包到达您的本地路由器。

3. 分发到节点

本地路由器确定目标 IP (10.80.0.2) 地址属于其直连网络，并将数据包直接转发到目标混合节点。

2. 节点网络处理

当数据包由节点的网络堆栈处理时，conntrack（netfilter 的一部分）会将数据包与容器组（pod）最初建立的连接进行匹配，并且由于最初应用了 DNAT，所以可通过将源 IP 从 EKS 控制面板 ENI 的 IP 重写到 kubernetes 服务 IP 来反转这一过程：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.80.0.2  | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

1。CNI 处理

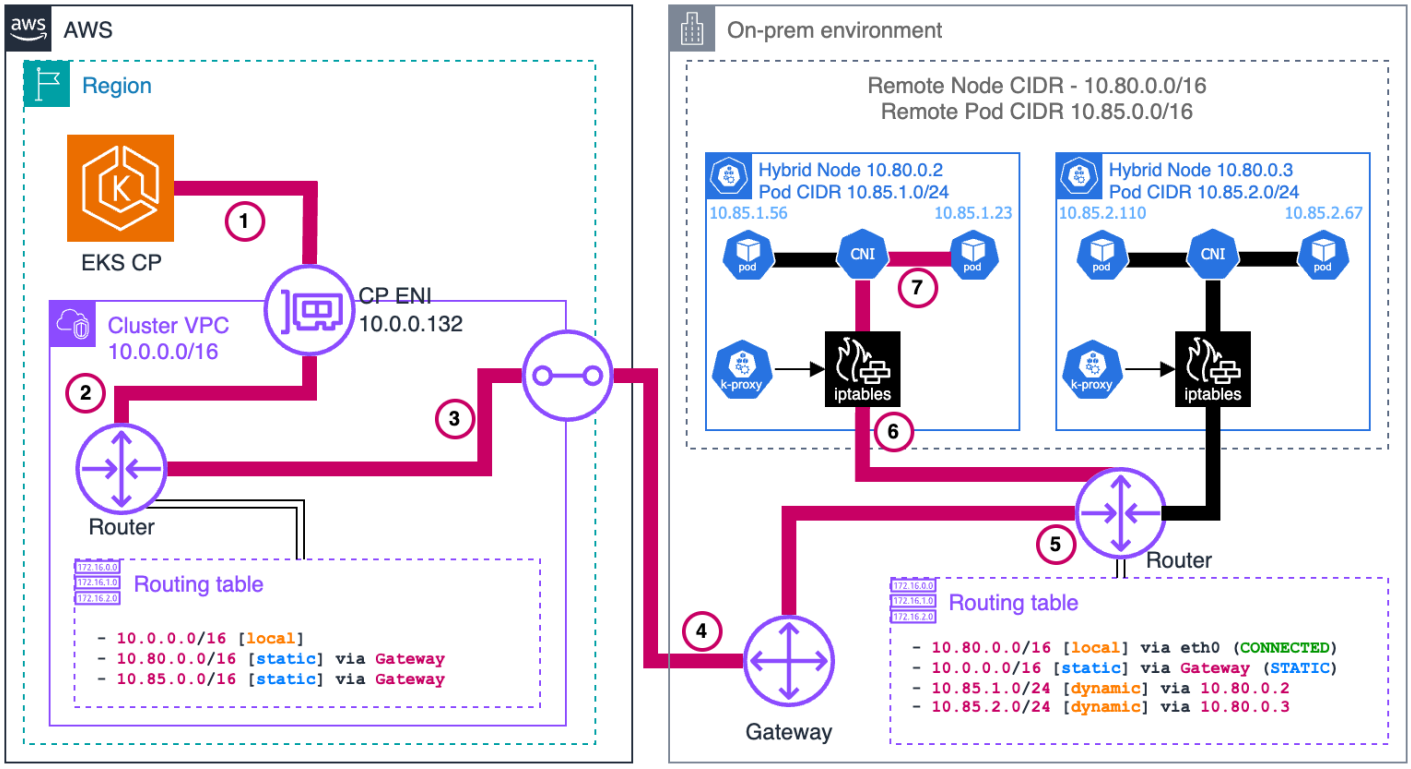
CNI 确定此数据包属于它之前应用过 SNAT 的连接。它会反转 SNAT，将目标 IP 改回容器组（pod）的 IP：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

CNI 检测到目标 IP 属于其网络中的某个容器组（pod），并将数据包分发到正确的容器组（pod）网络命名空间。

此流程展示了 CNI NAT 操作如何允许将数据包路由回节点，而无需为容器组（pod）CIDR 进行额外路由，从而简化配置。

EKS 控制面板到混合节点上运行的容器组（pod）（Webhook）



这种流量模式最常见于 Webhook，其中 EKS 控制面板需要直接启动与在混合节点上的容器组（pod）中运行的 Webhook 服务器的连接。示例包括验证和变更准入 Webhook，它们由 API 服务器在资源验证或变更过程中调用。

请求

1。EKS Kubernetes API 服务器启动请求

如果集群中配置了 Webhook 并且相关的 API 操作触发了它，EKS Kubernetes API 服务器需要直接连接到 Webhook 服务器容器组（pod）。API 服务器首先从与 Webhook 关联的服务或端点资源中查找容器组（pod）的 IP 地址。

假设 Webhook 容器组在 IP 为 `10.85.1.23` 的混合节点上运行，则 EKS Kubernetes API 服务器会创建一个针对 Webhook 端点的 HTTPS 请求。初始数据包通过 VPC 中的 EKS 控制面板 ENI 发送，因为目标 IP `10.85.1.23` 属于配置的远程容器组 CIDR (`10.85.0.0/16`)。数据包类似于以下示例：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 41892 (random) |                 |
| Dst: 10.85.1.23 | Dst: 8443           |                 |
```

```
+-----------------+--------------------+-----------------+
```

2. VPC 网络处理

数据包离开 EKS 控制面板 ENI，进入 VPC 网络层，子网的路由器作为下一个跃点。

3. VPC 路由表查找

包含 EKS 控制面板 ENI 的子网的 VPC 路由表包含远程容器组 CIDR (`10.85.0.0/16`) 的专用路由。此路由规则将数据包引导至“VPC 到本地”网关（例如，用于 Direct Connect 或 VPN 连接的虚拟专用网关）：

```
Destination     Target
10.0.0.0/16     local
10.85.0.0/16    vgw-id (VPC-to-onprem gateway)
```

4. 跨边界传输

网关通过您建立的连接（例如 Direct Connect 或 VPN）将数据包跨云边界传输到本地网络。数据包在穿过此连接时会保留其原始源和目标 IP 地址。

5. 本地网络接收

数据包到达您的本地路由器。路由器会查询其路由表以确定如何访问 10.85.1.23 地址。为此，您的本地网络必须拥有容器组（pod）CIDR 的路由，这些路由可以将流量引导到相应的混合节点。

在这种情况下，路由器的路由表中将包含一个条目，指示可通过 IP 为 `10.80.0.2` 的混合节点访问 `10.85.1.0/24` 子网：

```
Destination     Next Hop
10.85.1.0/24    10.80.0.2
```

6. 分发到节点

根据路由表条目，路由器将数据包转发到混合节点 (`10.80.0.2`)。数据包到达节点时，它看起来与 EKS Kubernetes API 服务器发送它时相同，目标 IP 仍然是容器组（pod）的 IP。

7. CNI 处理

节点的网络堆栈接收数据包，由于目标 IP 不是节点自己的 IP，因此将其传递给 CNI 进行处理。CNI 确定目标 IP 属于在此节点上本地运行的容器组（pod），并通过相应的虚拟接口将数据包转发到正确的容器组（pod）：

```
Original packet -> node routing -> CNI -> Pod's network namespace
```

容器组（pod）中的 Webhook 服务器接收并处理请求。

响应

在 Webhook 容器组（pod）处理请求后，它会按照相同的路径反向发送响应：

7. 容器组（pod）发送响应

Webhook 容器组（pod）创建响应数据包，并以自己的 IP 作为源，原始请求者（EKS 控制面板 ENI）作为目标：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.23 | Src: 8443           |                 |
| Dst: 10.0.0.132 | Dst: 41892          |                 |
+-----------------+---------------------+-----------------+
```

CNI 识别此数据包发送到外部网络（不是本地容器组），并将数据包传递到节点的网络堆栈，同时保留原始源 IP。

6. 节点网络处理

节点确定目标 IP (`10.0.0.132`) 不在本地网络中，并将数据包转发到其默认网关（本地路由器）。

5. 本地路由器路由

本地路由器会查询其路由表并确定目标 IP (`10.0.0.132`) 属于 VPC CIDR (`10.0.0.0/16`)。它会将数据包转发到连接到 Amazon 的网关。

4. 跨边界传输

数据包通过同一个本地到 VPC 的连接返回，朝相反的方向穿过云边界。

3. VPC 路由

数据包到达 VPC 时，路由表确定目标 IP 属于该 VPC 内的子网。数据包将在 VPC 内相应路由。

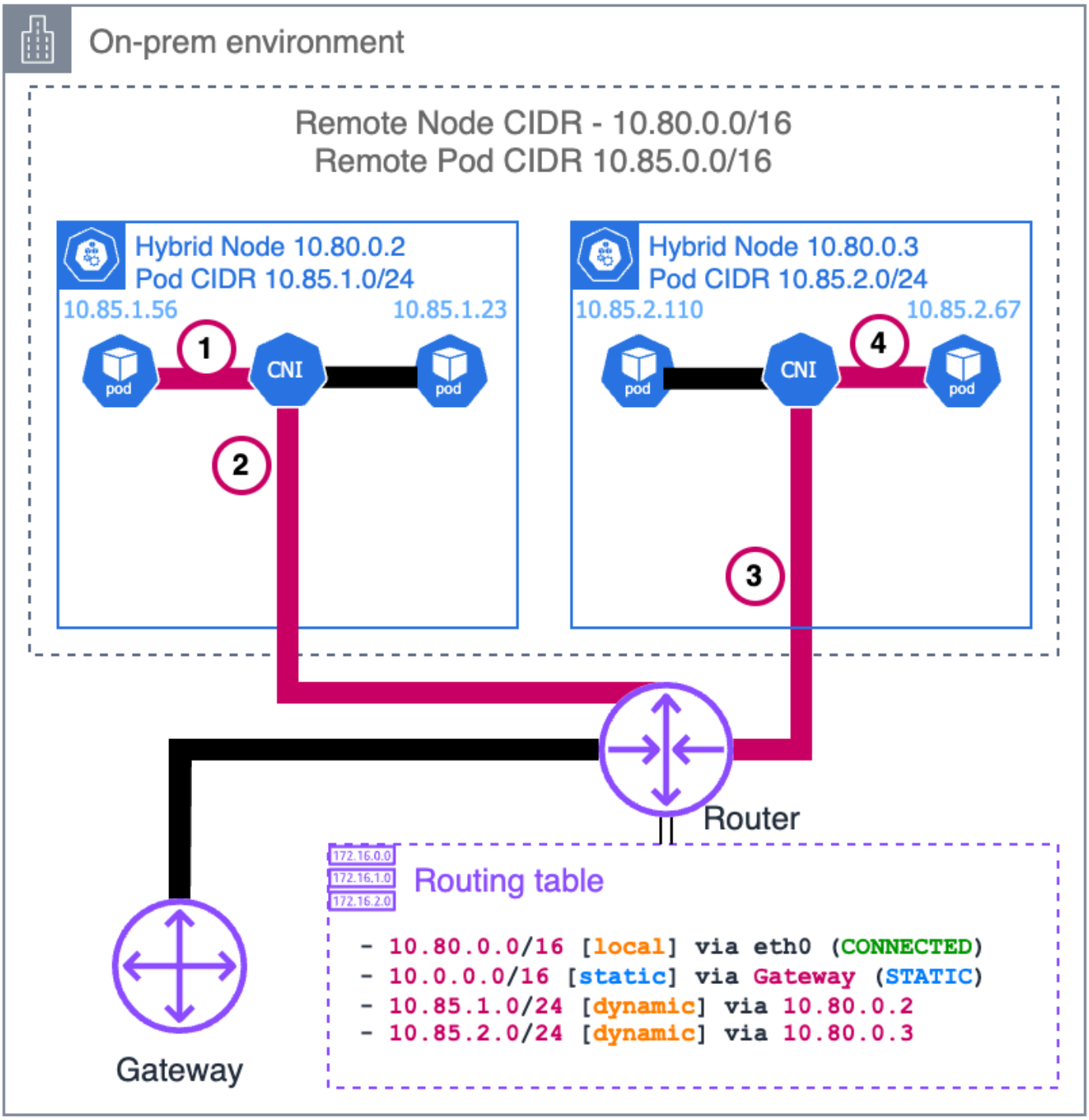
2. 和 1. EKS 控制面板 ENI 接收

数据包到达附加到 Kubernetes API 服务器的 ENI，从而完成往返行程。API 服务器接收 Webhook 响应，并根据此响应继续处理原始 API 请求。

此流量说明了为什么必须正确配置和路由远程容器组（pod）CIDR：

- VPC 必须具有指向本地网关的远程容器组（pod）CIDR 的路由
- 您的本地网络必须具有容器组（pod）CIDR 的路由，这些路由可以将流量引导到托管这些容器组（pod）的特定节点
- 如果没有此路由配置，则无法从 EKS 控制面板访问混合节点上的容器组（pod）中运行的 Webhook 和其他类似服务。

在混合节点上运行的容器组（pod）到容器组（pod）



本节介绍在不同混合节点上运行的容器组（pod）如何相互通信。此示例假设您的 CNI 使用 VXLAN 进行封装，这在 Cilium 或 Calico 等 CNI 中很常见。其他封装协议（例如 Geneve 或 IP-in-IP）的整体过程与示例相似。

请求

1。容器组（pod）A 启动通信

节点 1 上的容器组 A (10.85.1.56) 想要向节点 2 上的容器组 B (10.85.2.67) 发送流量。初始数据包类似于以下示例：

```
+------------------+-----------------+-------------+-----------------+
| Ethernet Header  | IP Header       | TCP/UDP     | Payload         |
| Src: Pod A MAC   | Src: 10.85.1.56 | Src: 43721  |                 |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080   |                 |
+------------------+-----------------+-------------+-----------------+
```

2. CNI 拦截并处理数据包

容器组（pod）A 的数据包离开其网络命名空间时，CNI 会拦截它。CNI 会查询其路由表并确定：– 目标 IP (10.85.2.67) 属于容器组 CIDR – 此 IP 不在本地节点上，但属于节点 2 (10.80.0.3) – 数据包需要用 VXLAN 进行封装。

封装的决定至关重要，因为底层物理网络不知道如何直接路由容器组（pod）CIDR，它只知道如何在节点 IP 之间路由流量。

CNI 将整个原始数据包封装在 VXLAN 帧中。这实际上创建了一个带有新标头的“数据包中的数据包”：

```
+-----------------+---------------+-------------+-----------
+--------------------------+
| Outer Ethernet  | Outer IP      | Outer UDP   | VXLAN     | Original Pod-to-Pod
    |
| Src: Node1 MAC  | Src: 10.80.0.2 | Src: Random  | VNI: 42    | Packet (unchanged
    |
| Dst: Router MAC | Dst: 10.80.0.3 | Dst: 8472    |            | from above)
    |
+-----------------+---------------+-------------+-----------
+--------------------------+
```

关于此封装的要点：– 外部数据包从节点 1 (10.80.0.2) 寻址到节点 2 (10.80.0.3) – UDP 端口 8472 是 Cilium 默认使用的 VXLAN 端口 – VXLAN 网络标识符（VNI）确定此数据包属于哪个叠加网络 – 整个原始数据包（以容器组 A 的 IP 作为源，容器组 B 的 IP 作为目标）完好无损地保留在内部

封装后的数据包现在进入节点 1 的常规网络堆栈，并按照与任何其他数据包相同的方式处理：

1. 节点网络处理：节点 1 的网络堆栈根据其目标路由数据包 (10.80.0.3)

2. 本地网络分发：
   - 如果两个节点都在同一个第 2 层网络上，则数据包将直接发送到节点 2
   - 如果它们位于不同的子网上，则首先将数据包转发到本地路由器
3. 路由器处理：路由器根据其路由表转发数据包，然后将其分发到节点 2

3. 接收节点处理

当封装后的数据包到达节点 2 (`10.80.0.3`) 时：

1. 该节点的网络堆栈接收它并将其标识为 VXLAN 数据包（UDP 端口 4789）
2. 数据包传递到 CNI 的 VXLAN 接口进行处理

4. VXLAN 解封装

节点 2 上的 CNI 处理 VXLAN 数据包：

1. 去掉外部标头（以太网、IP、UDP 和 VXLAN）
2. 提取原始的内部数据包
3. 数据包现在恢复为原始形式：

```
+------------------+-----------------+-------------+-----------------+
| Ethernet Header  | IP Header       | TCP/UDP     | Payload         |
| Src: Pod A MAC   | Src: 10.85.1.56 | Src: 43721  |                 |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080   |                 |
+------------------+-----------------+-------------+-----------------+
```

节点 2 上的 CNI 会检查目标 IP (`10.85.2.67`)，并且：

1. 确定此 IP 属于本地容器组（pod）
2. 通过相应的虚拟接口路由数据包
3. 将数据包分发到容器组（pod）B 的网络命名空间

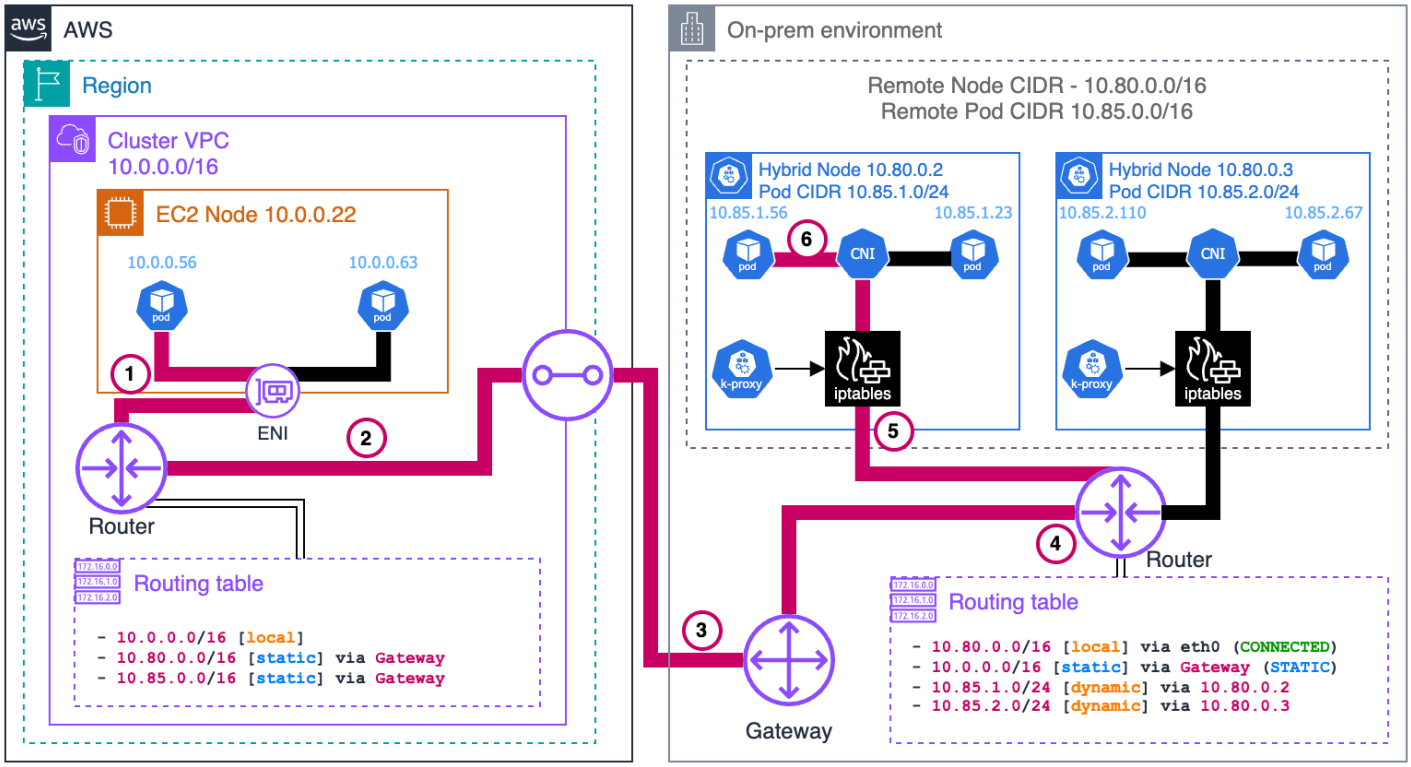
响应

容器组（pod）B 响应容器组（pod）A 时，整个过程是相反的：

4. 容器组 B 向容器组 A 发送数据包 (`10.85.1.56`)

5. 节点 2 的 CNI 使用 VXLAN 将其封装，并将目标设置为节点 1 (10.80.0.2)
6. 封装后的数据包分发到节点 1
7. 节点 1 的 CNI 将其解封并将原始响应分发给容器组（pod）A

云节点上的容器组（pod）到混合节点上的容器组（pod）（东西向流量）



请求

1。容器组（pod）A 启动通信

EC2 节点上的容器组 A (10.0.0.56) 想要向混合节点上的容器组 B (10.85.1.56) 发送流量。初始数据包类似于以下示例：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.56  | Src: 52390 (random) |                 |
| Dst: 10.85.1.56 | Dst: 8080           |                 |
+-----------------+---------------------+-----------------+
```

使用 VPC CNI 时，容器组（pod）A 拥有来自 VPC CIDR 的 IP，并直接附加到 EC2 实例上的 ENI。容器组（pod）的网络命名空间连接到 VPC 网络，因此数据包直接进入 VPC 路由基础设施。

2. VPC 路由

VPC 路由表包含远程容器组 CIDR (10.85.0.0/16) 的专用路由，该路由将此流量引导至“VPC 到本地”网关：

```
Destination     Target
10.0.0.0/16     local
10.85.0.0/16    vgw-id (VPC-to-onprem gateway)
```

根据此路由规则，数据包将被引导至连接到您的本地网络的网关。

3. 跨边界传输

网关通过您建立的连接（例如 Direct Connect 或 VPN）将数据包跨云边界传输到本地网络。在此整个传输过程中，数据包会保留其原始源和目标 IP 地址。

4. 本地网络接收

数据包到达您的本地路由器。路由器会查询其路由表，确定到达 10.85.1.56 地址的下一个跃点。您的本地路由器必须拥有容器组（pod）CIDR 的路由，这些路由可以将流量引导到相应的混合节点。

路由器的路由表中将包含一个条目，指示可通过 IP 为 10.80.0.2 的混合节点访问 10.85.1.0/24 子网：

```
Destination     Next Hop
10.85.1.0/24    10.80.0.2
```

5. 节点网络处理

路由器将数据包转发到混合节点 (10.80.0.2)。当数据包到达节点时，它仍将容器组（pod）A 的 IP 作为源，将容器组（pod）B 的 IP 作为目标。

6. CNI 处理

节点的网络堆栈接收数据包，由于目标 IP 不是自己的 IP，因此将其传递给 CNI 进行处理。CNI 确定目标 IP 属于在此节点上本地运行的容器组（pod），并通过相应的虚拟接口将数据包转发到正确的容器组（pod）：

```
Original packet -> node routing -> CNI -> Pod B's network namespace
```

容器组（pod）B 接收数据包并根据需要进行处理。

响应

6. 容器组（pod）B 发送响应

容器组（pod）B 创建一个以自己的 IP 作为源、容器组（pod）A 的 IP 作为目标的响应数据包：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 8080           |                 |
| Dst: 10.0.0.56  | Dst: 52390          |                 |
+-----------------+---------------------+-----------------+
```

CNI 确定此数据包发往外部网络，并将其传递到节点的网络堆栈。

5. 节点网络处理

节点确定目标 IP (`10.0.0.56`) 不属于本地网络，并将数据包转发到其默认网关（本地路由器）。

4. 本地路由器路由

本地路由器会查询其路由表并确定目标 IP (`10.0.0.56`) 属于 VPC CIDR (`10.0.0.0/16`)。它会将数据包转发到连接到 Amazon 的网关。

3. 跨边界传输

数据包通过同一个本地到 VPC 的连接返回，朝相反的方向穿过云边界。

2. VPC 路由

数据包到达 VPC 时，路由系统确定目标 IP 属于该 VPC 内的子网。数据包通过 VPC 网络路由到托管容器组（pod）A 的 EC2 实例。

1。容器组（pod）A 收到响应

数据包到达 EC2 实例，并通过其附加的 ENI 直接分发到容器组（pod）A。由于 VPC CNI 不对 VPC 中的容器组（pod）使用叠加网络，因此无需进行额外的解封 – 数据包到达时其原始标头保持不变。

此东西向流量说明了为什么必须正确配置远程容器组（pod）CIDR 并可从两个方向路由：

- VPC 必须具有指向本地网关的远程容器组（pod）CIDR 的路由

- 您的本地网络必须具有容器组（pod）CIDR 的路由，这些路由可以将流量引导到托管这些容器组（pod）的特定节点。

混合节点 nodeadm 参考

Amazon EKS 混合节点功能 CLI (nodeadm) 简化了混合节点组件的安装、配置、注册和卸载。您可以将 nodeadm 包含在操作系统映像中以自动执行混合节点引导过程，有关更多信息，请参阅the section called “准备操作系统”。

用于混合节点的 nodeadm 版本，与用于引导 Amazon EC2 实例以作为 Amazon EKS 集群中节点的 nodeadm 版本不同。请按照相应 nodeadm 版本的文档和参考进行操作。此文档页面适用于混合节点 nodeadm 版本。

https://github.com/aws/eks-hybrid GitHub 存储库发布了混合节点 nodeadm 的源代码。

> **Important**
>
> 您必须使用具有 root/sudo 权限的用户运行 nodeadm。

下载 nodeadm

nodeadm 的混合节点版本在 Amazon S3 中托管，由 Amazon CloudFront 分发。要在每台本地主机上安装 nodeadm，您可以在本地主机上运行以下命令。

x86_64 主机：

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/nodeadm'
```

ARM 主机

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/nodeadm'
```

将可执行文件权限添加到每台主机上下载的二进制文件。

```
chmod +x nodeadm
```

nodeadm install

nodeadm install 命令用于安装运行和将混合节点加入 Amazon EKS 集群所需的构件和依赖项。nodeadm install 命令可以在每个混合节点上单独运行，也可以在映像生成管道期间运行，以便在操作系统映像中预装混合节点依赖项。

用法

```
nodeadm install [KUBERNETES_VERSION] [flags]
```

定位参数

（必需）KUBERNETES_VERSION 要安装的 EKS Kubernetes 的主要.次要版本，例如 1.32

Flags

名称	必需	说明
-p, --credential-provider	TRUE	要安装的凭证提供者。支持的值为 iam-ra 和 ssm。请参阅the section called “准备凭证”了解更多信息。
-s, --containerd-source	FALSE	containerd 的来源。nodeadm 支持从操作系统发行版、Docker 软件包安装 containerd 以及跳过 containerd 安装。 值 distro：此为默认值。nodeadm 会安装由节点操作系统分发的、与 EKS Kubernetes 版本兼容的最新 containerd 软件包。distro 并非 Red Hat Enterprise Linux（RHEL）操作系统的有效值。

名称	必需	说明
		docker：nodeadm 会安装由 Docker 编译并分发的、与 EKS Kubernetes 版本兼容的最新 containerd 软件包。docker 并非 Amazon Linux 2023 的有效值。 none：nodeadm 不会安装 containerd 软件包。您必须首先手动安装 containerd ，然后再运行 nodeadm init。
-r, --region	FALSE	指定用于下载 SSM Agent 等构件的 Amazon 区域。默认值为 us-west-2 。
-t, --timeout	FALSE	安装命令的最大持续时间。输入遵循持续时间格式。例如 1h23m。安装命令的默认下载超时设置为 20 分钟。
-h, --help	FALSE	显示带有可用标志、子命令和位置值参数的帮助消息。

示例

安装 Kubernetes 版本 1.32，并将 Amazon Systems Manager（SSM）作为凭证提供者

```
nodeadm install 1.32 --credential-provider ssm
```

安装 Kubernetes 版本 1.32，并将 Amazon Systems Manager（SSM）作为凭证提供者，将 Docker 作为 containerd 源，下载超时设置为 20 分钟。

```
nodeadm install 1.32 --credential-provider ssm --containerd-source docker --timeout 20m
```

安装 Kubernetes 版本 1.32，并将 Amazon IAM Roles Anywhere 作为凭证提供者

```
nodeadm install 1.32 --credential-provider iam-ra
```

nodeadm config check

nodeadm config check 命令会检查提供的节点配置是否存在错误。此命令可用于确认和验证混合节点配置文件的正确性。

用法

```
nodeadm config check [flags]
```

Flags

名称	必需	说明
-c, --config-source	TRUE	nodeadm 配置的来源。对于混合节点，输入应遵循带文件架构的 URI。
-h, --help	FALSE	显示带有可用标志、子命令和位置值参数的帮助消息。

示例

```
nodeadm config check -c file://nodeConfig.yaml
```

nodeadm init

nodeadm init 命令会启动混合节点并将其连接到配置的 Amazon EKS 集群。有关如何配置 nodeConfig.yaml 文件的详细信息，请参阅the section called “SSM 混合激活的节点配置”或the section called “IAM Roles Anywhere 的节点配置”。

用法

```
nodeadm init [flags]
```

Flags

名称	必需	说明
-c, --config-source	TRUE	nodeadm 配置的来源。对于混合节点，输入应遵循带文件架构的 URI。
-s, --skip	FALSE	要跳过的 init 阶段。除非有助于解决问题，否则不建议跳过任何阶段。 值 install-validation 会跳过检查之前的安装命令是否成功运行。 如果节点上启用了防火墙，则 cni-validation 会跳过检查 Cilium 或 Calico CNI 的 VXLAN 端口是否已打开 如果节点 IP 在远程节点网络的 CIDR 范围内，node-ip-validation 将会跳过检查。
-h, --help	FALSE	显示带有可用标志、子命令和位置值参数的帮助消息。

示例

```
nodeadm init -c file://nodeConfig.yaml
```

nodeadm upgrade

nodeadm upgrade 命令会将所有已安装的构件升级到最新版本，启动节点以配置升级后的构件并加入 Amazon 上的 EKS 集群。对于在节点上运行的工作负载，升级是一个中断性命令。在运行升级之前，请将您的工作负载转移至其他节点。

用法

```
nodeadm upgrade [KUBERNETES_VERSION] [flags]
```

定位参数

（必需）KUBERNETES_VERSION 要安装的 EKS Kubernetes 的主要.次要版本，例如 1.32

Flags

名称	必需	说明
-c, --config-source	TRUE	nodeadm 配置的来源。对于混合节点，输入应遵循带文件架构的 URI。
-t, --timeout	FALSE	下载构件超时。输入遵循持续时间格式。例如，1h23m。升级命令的默认下载超时设置为 10 分钟。
-s, --skip	FALSE	要跳过的升级阶段。除非有助于解决问题，否则不建议跳过任何阶段。 值 pod-validation 会跳过检查除进程守护程序集和静态容器组外，是否节点上的所有容器组都在运行。 node-validation 会跳过检查节点是否已被封锁。 init-validation 会跳过检查节点在运行升级之前是否已成功初始化。

名称	必需	说明
		`containerd-major-version-upgrade` 可避免在节点升级期间对 containerd 执行大版本升级操作。
`-h, --help`	FALSE	显示带有可用标志、子命令和位置值参数的帮助消息。

示例

```
nodeadm upgrade 1.32 -c file://nodeConfig.yaml
```

```
nodeadm upgrade 1.32 -c file://nodeConfig.yaml --timeout 20m
```

`nodeadm uninstall`

`nodeadm uninstall` 命令会停止并移除在 `nodeadm install` 期间安装的构件 nodeadm，包括 kubelet 和 containerd。请注意，卸载命令不会从集群中清空或删除混合节点。您必须分别运行清空和删除操作，有关更多信息，请参阅the section called “删除混合节点”。默认情况下，如果节点上还有容器组，则 `nodeadm uninstall` 不会执行。同样，`nodeadm uninstall` 也不会移除 CNI 依赖项或您在集群上运行的其他 Kubernetes 附加组件的依赖项。要从主机上完全移除 CNI 安装，请参阅the section called “配置 CNI” 中的说明。如果将 Amazon SSM 混合激活作为本地凭证提供者，则 `nodeadm uninstall` 命令将从 Amazon SSM 托管式实例中注销您的主机。

用法

```
nodeadm uninstall [flags]
```

Flags

名称	必需	说明
`-s, --skip`	FALSE	要跳过的卸载阶段。除非有助于解决问题，否则不建议跳过任何阶段。

名称	必需	说明
		值 pod-validation 会跳过检查除进程守护程序集和静态容器组外，是否节点上的所有容器组都在运行。 node-validation 会跳过检查节点是否已被封锁。 init-validation 会跳过检查节点在运行卸载之前是否已成功初始化。
-h, --help	FALSE	显示带有可用标志、子命令和位置值参数的帮助消息。

名称	必需	说明
`-f`, `--force`	FALSE	强制删除可能包含 Kubernetes 和 CNI 组件中剩余文件的其他目录。 WARNING 此操作将删除默认 Kubernetes 和 CNI 目录（`/var/lib/cni` 、`/etc/cni/net.d` 等）中的所有内容。如果您将自己的数据存储在这些位置，请不要使用此标志。 从 nodeadm `v1.0.9` 开始，`./nodeadm uninstall --skip node-validation,pod-validation --force` 命令不再删除 `/var/lib/kubelet` 目录。这是因为该目录可能包含容器组（pod）卷和 volume-subpath 目录，这些目录有时会包含已挂载的节点文件系统。 安全处理提示 - 删除已挂载的路径可能会导致意外删除实际挂载的节点文件系统。在手动删除 `/var/lib/kubelet` 目录之前，请仔细检查所有活动的挂载并安全地卸载卷，以免丢失数据。

示例

```
nodeadm uninstall
```

```
nodeadm uninstall --skip node-validation,pod-validation
```

nodeadm debug

nodeadm debug 命令可用于对运行不正常或配置有错误的混合节点进行故障排除。此命令可以验证以下要求是否已经满足。

- 该节点具有必需 Amazon API 的网络访问权限以获取凭证，
- 该节点能够获取所配置混合节点 IAM 角色的 Amazon 凭证，
- 该节点具有对 EKS Kubernetes API 端点的网络访问权限，并且 EKS Kubernetes API 端点证书的有效，
- 该节点能够通过 EKS 集群身份验证，其在集群中的身份有效，并且该节点可以通过为 EKS 集群配置的 VPC 访问 EKS 集群。

如果发现错误，该命令的输出会提供故障排除步骤建议。某些验证步骤会显示子进程。如果这些子进程失败，则输出将显示在验证错误下的 stderr 部分中。

用法

```
nodeadm debug [flags]
```

Flags

名称	必需	说明
-c, --config-source	TRUE	nodeadm 配置的来源。对于混合节点，输入应遵循带文件架构的 URI。
--no-color	FALSE	禁用彩色输出。对自动化很有用。
-h, --help	FALSE	显示带有可用标志、子命令和位置值参数的帮助消息。

示例

```
nodeadm debug -c file://nodeConfig.yaml
```

nodeadm 文件位置

nodeadm 安装

运行 `nodeadm install` 时，将配置以下文件和文件位置。

Artifact	路径
IAM Roles Anywhere CLI	/usr/local/bin/aws_signing_helper
Kubelet 二进制包	/usr/bin/kubelet
Kubectl 二进制包	usr/local/bin/kubectl
ECR 凭证提供者	/etc/eks/image-credential-provider/ecr-credential-provider
Amazon IAM 身份验证器	/usr/local/bin/aws-iam-authenticator
SSM 安装 CLI	/opt/ssm/ssm-setup-cli
SSM Agent	在 Ubuntu 上：/snap/amazon-ssm-agent/current/amazon-ssm-agent 在 RHEL 和 AL2023 上：/usr/bin/amazon-ssm-agent
Containerd	在 Ubuntu 和 AL2023 上：/usr/bin/containerd 在 RHEL 上：/bin/containerd
Iptables	在 Ubuntu 和 AL2023 上：/usr/sbin/iptables 在 RHEL 上：/sbin/iptables
CNI 插件	/opt/cni/bin

Artifact	路径
安装的构件跟踪器	/opt/nodeadm/tracker

nodeadm init

运行 `nodeadm init` 时，将配置以下文件和文件位置。

名称	路径
Kubelet kubeconfig	/var/lib/kubelet/kubelet/kubecconfig
Kubelet config	/etc/kubernetes/kubelet/config.json
Kubelet systemd 单元	/etc/systemd/system/kubelet.service
映像凭证提供者配置	/etc/eks/image-credential-provider/config.json
Kubelet 环境文件	/etc/eks/kubelet/environment
Kubelet 证书	/etc/kubernetes/pki/ca.crt
Containerd 配置	/etc/containerd/config.toml
Containerd 内核模块配置	/etc/modules-load.d/contianerd.conf
Amazon 配置文件	/etc/aws/hybrid/config
Amazon 凭证文件（如果启用了凭证文件）	/eks-hybrid/.aws/credentials
Amazon 签名助手系统单元	/etc/systemd/system/aws_signing_helper_update.service
Sysctl conf 文件	/etc/sysctl.d/99-nodeadm.conf
Ca-certificates	/etc/ssl/certs/ca-certificates.crt
Gpg 密钥文件	/etc/apt/keyrings/docker.asc
Docker 存储库源文件	/etc/apt/sources.list.d/docker.list

SSM 混合激活的节点配置

以下是将 Amazon SSM 混合激活作为混合节点凭证的示例 nodeConfig.yaml。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:              # Name of the EKS cluster
    region:            # Amazon Region where the EKS cluster resides
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

IAM Roles Anywhere 的节点配置

以下是将 Amazon IAM Roles Anywhere 作为混合节点凭证的示例 nodeConfig.yaml。

如果将 Amazon IAM Roles Anywhere 作为本地凭证提供者，您在 nodeadm 配置中使用的 nodeName 必须与您为混合节点 IAM 角色设定的权限范围一致。例如，假设混合节点 IAM 角色的权限仅允许 Amazon IAM Roles Anywhere 在角色会话名称等于主机证书的 CN 时代入该角色，则 nodeadm 配置中的 nodeName 必须与证书的 CN 相同。您使用的 nodeName 长度不能超过 64 个字符。有关更多信息，请参阅 the section called “准备凭证”。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:               # Name of the EKS cluster
    region:             # Amazon Region where the EKS cluster resides
  hybrid:
    iamRolesAnywhere:
      nodeName:         # Name of the node
      trustAnchorArn:   # ARN of the IAM Roles Anywhere trust anchor
      profileArn:       # ARN of the IAM Roles Anywhere profile
      roleArn:          # ARN of the Hybrid Nodes IAM role
      certificatePath: # Path to the certificate file to authenticate with the IAM
 Roles Anywhere trust anchor
      privateKeyPath:  # Path to the private key file for the certificate
```

自定义 kubelet 的节点配置（可选）

您可以在 nodeadm 配置中传递 kubelet 配置和标志。有关如何添加额外节点标签 `abc.amazonaws.com/test-label` 和配置，以将 `shutdownGracePeriod` 设置为 30 秒的信息，请参阅以下示例。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # Amazon Region where the EKS cluster resides
  kubelet:
    config:           # Map of kubelet config and values
       shutdownGracePeriod: 30s
    flags:            # List of kubelet flags
       - --node-labels=abc.company.com/test-label=true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

自定义 containerd 的节点配置（可选）

您可以在 nodeadm 配置中传递自定义 containerd 配置。nodeadm containerd 配置接受内联 TOML。要了解如何配置 containerd 以禁用删除 containerd 内容存储中未打包的映像层，请参阅以下示例。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # Amazon Region where the EKS cluster resides
  containerd:
    config: |         # Inline TOML containerd additional configuration
       [plugins."io.containerd.grpc.v1.cri".containerd]
       discard_unpacked_layers = false
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

> **Note**
>
> containerd 1.x 与 2.x 采用不同的配置文件格式。containerd 1.x 使用配置版本 2，而 containerd 2.x 使用配置版本 3。虽然 containerd 2.x 对配置版本 2 仍保持向下兼容，但为实现最优性能，建议使用配置版本 3。您可通过 `containerd --version` 命令查看 containerd 版本，也可查阅 `nodeadm` 安装日志。有关配置版本控制的更多详细信息，请参阅 https://containerd.io/releases/

您还可以使用 containerd 配置来实现 SELinux 支持。在 containerd 上启用 SELinux 后，确保在节点上调度的容器组启用了正确的 securityContext 和 seLinuxOptions。有关配置安全上下文的更多信息，请参阅 Kubernetes 文档。

> **Note**
>
> Red Hat Enterprise Linux（RHEL）8 和 RHEL 9 在主机上默认启用了 SELinux 并设置为严格模式。Amazon Linux 2023 默认启用 SELinux 并设置为宽松模式。当主机上的 SELinux 设置为宽容模式时，在 containerd 上启用此功能不会阻止请求，但会根据主机上的 SELinux 配置记录请求。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # Amazon Region where the EKS cluster resides
  containerd:
    config: |         # Inline TOML containerd additional configuration
       [plugins."io.containerd.grpc.v1.cri"]
       enable_selinux = true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

混合节点故障排除

本主题介绍在使用 Amazon EKS 混合节点功能时可能遇到的一些常见错误以及修复方法。有关其他故障排除信息，请参阅故障排除以及 Amazon re:Post 上有关 Amazon EKS 的知识中心标签。如果您无法解决问题，请联系 Amazon Support。

使用 **nodeadm debug** 对节点进行故障排除 您可以从混合节点运行 nodeadm debug 命令来验证是否满足联网和凭证要求。有关 nodeadm debug 命令的更多信息，请参阅 the section called “混合节点 nodeadm”。

使用集群见解检测混合节点的问题 Amazon EKS 集群见解包括见解检查，可检测集群中 EKS 混合节点配置的常见问题。您可以从 Amazon Web Services 管理控制台、Amazon CLI 和 Amazon SDK 中查看所有见解检查的结果。有关集群见解的更多信息，请参阅 the section called “集群见解”。

混合节点安装故障排除

以下故障排除主题与使用 nodeadm install 命令在主机上安装混合节点依赖项有关。

nodeadm命令失败，**must run as root**

运行 nodeadm install 命令的用户必须具有主机上的根权限或 sudo 权限。如果以没有根权限或 sudo 权限的用户身份运行 nodeadm install，则会在 nodeadm 输出中看到以下错误。

```
"msg":"Command failed","error":"must run as root"
```

无法连接到依赖项

nodeadm install 命令会安装混合节点必需的依赖项。混合节点依赖项包括 containerd、kubelet、kubectl 和 Amazon SSM 或 Amazon IAM Roles Anywhere 组件。您必须具有从运行 nodeadm install 的位置访问的权限才能下载这些依赖项。要详细了解您必须具有访问权限的位置列表，请参阅the section called “准备联网”。如果您没有访问权限，则会在 nodeadm install 输出中看到与以下类似的错误。

```
"msg":"Command failed","error":"failed reading file from url: ...: max retries achieved
 for http request"
```

更新软件包管理器失败

nodeadm install 命令会在安装混合节点依赖项之前运行 apt update 或 yum update 或 dnf update。如果此步骤未成功，您可能会看到与以下类似的错误。要修复错误，您可以首先运行 apt

update 或 yum update 或 dnf update，然后再运行 nodeadm install，也可以尝试重新运行 nodeadm install。

```
failed to run update using package manager
```

超时或超过上下文截止时间

运行 nodeadm install 时，如果您在安装过程的若干阶段看到问题，并错误消息指示超时或超过上下文截止时间，则可能说明连接速度较慢，导致无法在超时之前安装混合节点依赖项。要解决这些问题，您可以尝试在 nodeadm 中使用 --timeout 标志来延长下载依赖项的超时期限。

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --timeout 20m0s
```

混合节点连接故障排除

本节中的故障排除主题与使用 nodeadm init 命令将混合节点连接到 EKS 集群的过程有关。

操作错误或架构不受支持

运行 nodeadm init 时，如果您看到与 operation error 或 unsupported scheme 相关的错误，请检查 nodeConfig.yaml 以确保其格式正确并且已传递给 nodeadm。有关 nodeConfig.yaml 格式和选项的更多信息，请参阅the section called “混合节点 nodeadm”。

```
"msg":"Command failed","error":"operation error ec2imds: GetRegion, request canceled,
 context deadline exceeded"
```

混合节点 IAM 角色缺少 **eks:DescribeCluster** 操作的权限

运行 nodeadm init 时，nodeadm 会尝试通过调用 EKS DescribeCluster 操作来收集有关 EKS 集群的信息。若混合节点 IAM 角色不具备 eks:DescribeCluster 操作的权限，则在执行 nodeadm init 时，您必须在传入 nodeadm 的节点配置中，手动指定 Kubernetes API 端点、集群 CA 证书包以及服务 IPv4 CIDR。请参阅the section called “准备凭证”，以了解有关混合节点 IAM 角色所需权限的更多信息。

```
"msg":"Command failed","error":"operation error EKS: DescribeCluster, https response
 error StatusCode: 403 ... AccessDeniedException"
```

混合节点 IAM 角色缺少 **eks:ListAccessEntries** 操作的权限

执行 nodeadm init 时，nodeadm 会尝试调用 EKS ListAccessEntries 操作，验证 EKS 集群是否存在一个与该混合节点 IAM 角色绑定的、类型为 HYBRID_LINUX 的访问条目。若混合节点 IAM 角色不具备 eks:ListAccessEntries 操作的权限，那么在执行 nodeadm init 命令时，您必须传入 --skip cluster-access-validation 标记。请参阅the section called “准备凭证”，以了解有关混合节点 IAM 角色所需权限的更多信息。

```
"msg":"Command failed","error":"operation error EKS: ListAccessEntries, https response
 error StatusCode: 403 ... AccessDeniedException"
```

节点 IP 不在远程节点网络 CIDR 范围内

运行 nodeadm init 时，如果节点 IP 地址不在指定远程节点网络 CIDR 范围内，则可能会出现错误。此错误将与以下示例类似：

```
node IP 10.18.0.1 is not in any of the remote network CIDR blocks [10.0.0.0/16
 192.168.0.0/16]
```

此示例显示一个 IP 为 10.18.0.1 的节点正在尝试加入一个远程网络 CIDR 为 10.0.0.0/16 和 192.168.0.0/16 的集群。之所以出现此错误，是因为这两个 CIDR 均不包含 10.18.0.1。

确认您已正确配置 RemoteNodeNetworks，使其包含所有节点 IP 地址。有关网络配置的更多信息，请参阅the section called “准备联网”。

- 在集群所在区域运行以下命令，检查您的 RemoteNodeNetwork 配置。验证输出中列出的 CIDR 块是否包含您的节点 IP 范围，并且与错误消息中列出的 CIDR 块相同。如果不一致，请确认 nodeConfig.yaml 中的集群名称和区域与预期的集群一致。

```
aws eks describe-cluster --name CLUSTER_NAME --region REGION_NAME --query
 cluster.remoteNetworkConfig.remoteNodeNetworks
```

- 确认您使用的是预期的节点：
  - 检查节点的主机名和 IP 地址是否与您计划注册到集群的主机名和 IP 地址一致，从而确认节点是否正确。
  - 确认此节点所在的本地网络是否正确（其 CIDR 范围在集群设置期间注册为 RemoteNodeNetwork）。

如果节点 IP 仍不符合预期，请检查以下内容：

- 如果您使用的是 IAM Roles Anywhere，kubelet 将对 IAM Roles Anywhere nodeName 执行 DNS 查找，并使用与节点名称关联的 IP（如果可用）。如果您负责维护节点的 DNS 条目，请确认这些条目指向远程节点网络 CIDR 内的 IP。
- 如果节点有多个网络接口，kubelet 选择的默认接口的 IP 地址可能在远程节点网络 CIDR 范围外。要使用其他接口，请使用 nodeConfig.yaml 中的 --node-ip kubelet 标志指定其 IP 地址。有关更多信息，请参阅 the section called “混合节点 nodeadm”。您可以通过在节点上运行以下命令，来查看节点的网络接口及其 IP 地址：

```
ip addr show
```

混合节点未在 EKS 集群中出现

如果运行 nodeadm init，但在运行完成后，您的混合节点未在集群中出现，则说明混合节点与 EKS 控制面板之间的网络连接可能存在问题，您可能没有配置所需的安全组权限，或者可能没有将混合节点 IAM 角色映射到 Kubernetes 基于角色的访问控制（RBAC）。您可以使用以下命令检查 kubelet 和 kubelet 日志的状态，从而启动调试进程。从加入集群失败的混合节点上运行以下命令。

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

无法与集群通信

如果混合节点无法与集群控制面板通信，则可能会看到与以下类似的日志。

```
"Failed to ensure lease exists, will retry" err="Get ..."
```

```
"Unable to register node with API server" err="Post ..."
```

```
Failed to contact API server when waiting for CSINode publishing ... dial tcp <ip
 address>: i/o timeout
```

如果您看到此类消息，请检查以下内容以确保其符合the section called “准备联网”中详述的混合节点要求。

- 确认传递给 EKS 集群的 VPC 具有指向本地节点以及（可选）容器组 CIDR 的中转网关（TGW）或虚拟专用网关（VGW）的路由。

- 确认您的 EKS 集群还有一个额外的安全组，其中包含针对本地节点 CIDR 和（可选）容器组 CIDR 的入站规则。
- 确认您的本地路由器已配置为允许进出 EKS 控制面板的流量。

未授权

如果混合节点能够与 EKS 控制面板通信但无法注册，则可能会看到与以下类似的日志。请注意，以下日志消息的主要区别在于 Unauthorized 错误。这表明该节点无法执行其任务，因为没有所需的权限。

```
"Failed to ensure lease exists, will retry" err="Unauthorized"
```

```
"Unable to register node with API server" err="Unauthorized"
```

```
Failed to contact API server when waiting for CSINode publishing: Unauthorized
```

如果您看到此类消息，请检查以下内容以确保其符合the section called “准备凭证”和the section called “准备集群访问权限”中详述的混合节点要求。

- 确认混合节点的身份与您预期的混合节点 IAM 角色相符。这可以通过在混合节点上运行 `sudo aws sts get-caller-identity` 来完成。
- 确认混合节点 IAM 角色具有必需的权限。
- 确认集群中是否有混合节点 IAM 角色的 EKS 访问条目，或者确认 `aws-auth` ConfigMap 中是否有混合节点 IAM 角色的条目。如果使用 EKS 访问条目，请确认混合节点 IAM 角色的访问条目具有 `HYBRID_LINUX` 访问类型。如果使用 `aws-auth` ConfigMap，请确认您输入的混合节点 IAM 角色符合the section called “准备集群访问权限”中详述的要求和格式。

混合节点已在 EKS 集群中注册但显示的状态为 **`Not Ready`**

如果混合节点已成功注册到您的 EKS 集群，但混合节点显示状态为 `Not Ready`，则首先要检查的是容器网络接口（CNI）状态。如果您尚未安装 CNI，则混合节点状态预期将为 `Not Ready`。一旦 CNI 安装并成功运行，节点的状态就会更新为 `Ready`。如果您尝试安装 CNI 但未成功运行，请参阅本页上的the section called “混合节点 CNI 故障排除”。

证书签名请求（CSR）停滞在待处理状态

将混合节点连接到 EKS 集群后，如果您发现混合节点有待处理的 CSR，则表示您的混合节点不满足自动批准的要求。如果混合节点的 CSR 是由带有 `eks.amazonaws.com/compute-type: hybrid` 标签的节点创建的，并且 CSR 具有以下主体备用名称（SAN），则会自动批准和颁发混合节点的 CSR：至少有一个 DNS SAN 等于节点名称且 IP SAN 属于远程节点网络 CIDR。

混合配置文件已经存在

如果您更改了 nodeadm 配置并尝试使用新配置重新注册节点，则可能会看到一条错误消息，指示混合配置文件已经存在，但其内容已更改。请勿在配置更改之间运行 `nodeadm init`，而应运行 `nodeadm uninstall`，然后再运行 `nodeadm install` 和 `nodeadm init`。这样可以确保根据配置更改进行正确的清理。

```
"msg":"Command failed","error":"hybrid profile already exists at /etc/aws/hybrid/config
 but its contents do not align with the expected configuration"
```

混合节点解析私有 API 失败

运行 `nodeadm init` 后，如果您在 `kubelet` 日志中看到某个错误，显示由于存在 `no such host` 而无法连接到 EKS Kubernetes API 服务器，则可能需要在本地网络或主机级别更改 EKS Kubernetes API 端点的 DNS 条目。请参阅《Amazon Route53 文档》中的 Forwarding inbound DNS queries to your VPC。

```
Failed to contact API server when waiting for CSINode publishing: Get ... no such host
```

无法在 EKS 控制台中查看混合节点

如果您已注册了混合节点，但无法在 EKS 控制台中查看，请检查您用于查看控制台的 IAM 主体的权限。您使用的 IAM 主体必须具有特定的最低 IAM 和 Kubernetes 权限，才能在控制台中查看资源。有关更多信息，请参阅 the section called “访问集群资源”。

混合节点运行故障排查

如果混合节点已注册到 EKS 集群并且状态为 `Ready`，但状态很快变为 `Not Ready`，则可能存在多种导致运行不正常的问题，例如节点缺少足够的 CPU、内存或可用磁盘空间资源，或者节点与 EKS 控制面板之间的连接已断开等。您可以使用以下步骤对节点进行故障排除，如果无法解决问题，请联系 Amazon Support。

从您的混合节点运行 `nodeadm debug` 以验证是否满足网络和凭证要求。有关 `nodeadm debug` 命令的更多信息，请参阅 the section called “混合节点 nodeadm”。

获取节点状态

```
kubectl get nodes -o wide
```

检查节点条件和事件

```
kubectl describe node NODE_NAME
```

获取容器组状态

```
kubectl get pods -A -o wide
```

检查容器组状况和事件

```
kubectl describe pod POD_NAME
```

检查容器组日志

```
kubectl logs POD_NAME
```

检查 **kubectl** 日志

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

容器组存活探针失败或 Webhook 不起作用

如果在混合节点上运行的应用程序、附加组件或 Webhook 未正常启动，则可能存在网络问题，阻碍了与容器组的通信。要让 EKS 控制面板连接到在混合节点上运行的 Webhook，您必须将 EKS 集群配置为远程容器组网络，并在 VPC 路由表中包含本地容器组 CIDR 的路由，以中转网关（TGW）、虚拟专用网关（VPW）或您用于将 VPC 与本地网络连接的其他网关为目标。有关混合节点联网要求的更多信息，请参阅the section called “准备联网”。此外，您还必须在本地防火墙中允许此流量，并确保您的路

由器可以正确路由到您的容器组。要详细了解在混合节点上运行 Webhook 的要求，请参阅the section called “配置 Webhook”。

下方显示了此场景的一条常见容器组日志消息，其中 ip-address 是 Kubernetes 服务的集群 IP。

```
dial tcp <ip-address>:443: connect: no route to host
```

`kubectl logs` 或者 `kubectl exec` 命令不起作用（`kubelet` API 命令）

如果 `kubectl attach`、`kubectl cp`、`kubectl exec`、`kubectl logs` 和 `kubectl port-forward` 命令超时，而其他 `kubectl` 命令能够成功运行，则问题可能与远程网络配置有关。这些命令通过集群连接到节点上的 `kubelet` 端点。有关更多信息，请参阅 the section called “`kubelet` 端点”。

确认节点 IP 和容器组 IP 位于为集群配置的远程节点网络和远程容器组网络 CIDR 范围内。使用以下命令检查 IP 分配。

```
kubectl get nodes -o wide
```

```
kubectl get pods -A -o wide
```

将这些 IP 与您配置的远程网络 CIDR 进行比较，确保路由正确。有关网络配置要求，请参阅the section called “准备联网”。

混合节点 CNI 故障排除

如果您在首次使用混合节点启动 Cilium 或 Calico 时遇到问题，则通常是由于混合节点或在混合节点上运行的 CNI 容器组与 EKS 控制面板之间的网络问题所致。确保您的环境符合“准备混合节点的联网”中的要求。将问题分解成若干部分会非常实用。

EKS 集群配置

- RemoteNodeNetwork 和 RemotePodNetwork 的配置是否正确？

VPC 配置

- VPC 路由表中是否有 RemoteNodeNetwork 和 RemotePodNetwork 的路由，并以中转网关或虚拟专用网关为目标？

安全组配置

- 是否有 RemoteNodeNetwork 和 RemoteNetwork 的入站和出站规则？

本地网络

是否存在往返于 EKS 控制面板、混合节点以及在混合节点上运行的容器组的路由和访问权限？

CNI 配置

如果使用叠加网络，CNI 的 IP 池配置是否与使用 Webhook 时为 EKS 集群配置的 RemotePodNetwork 相匹配？

混合节点的状态为 **Ready**，但未安装 CNI

如果混合节点显示状态为 Ready，但您尚未在集群上安装 CNI，则说明混合节点上可能存在旧的 CNI 构件。默认情况下，当您使用 Helm 等工具卸载 Cilium 和 Calico 时，磁盘上的资源不会从物理计算机或虚拟机中移除。此外，在旧安装中，这些 CNI 的自定义资源定义（CRD）可能仍存在于集群上。有关更多信息，请参阅the section called “配置 CNI”中的 Cilium 和 Delete Calico 部分。

Cilium 故障排除

如果您在混合节点上运行 Cilium 时遇到问题，请参阅 Cilium 文档中的故障排除步骤。以下章节介绍了在混合节点上部署 Cilium 时可能遇到的特有问题。

Cilium 未启动

如果在每个混合节点上运行的 Cilium 代理都未启动，请检查 Cilium 代理容器组的日志中是否存在错误。Cilium 代理需要连接到 EKS Kubernetes API 端点才能启动。如果此连接配置不正确，Cilium 代理将无法启动。在这种情况下，您将在 Cilium 代理容器组日志中看到与以下类似的日志消息。

```
msg="Unable to contact k8s api-server"
level=fatal msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/
kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
```

Cilium 代理在主机网络上运行。您的 EKS 集群必须将 Cilium 连接配置为 RemoteNodeNetwork。确认您的 EKS 集群还有一个额外的安全组，其中包含 RemoteNodeNetwork 的入站规则，您的 VPC 中有 RemoteNodeNetwork 的路由，并且您的本地网络配置正确，从而能够连接到 EKS 控制面板。

如果 Cilium 操作符正在运行，并且某些但不是全部 Cilium 代理正在运行，请确认您有可用的容器组 IP 可以分配给集群中的所有节点。在 Cilium 配置中使用带 clusterPoolIPv4PodCIDRList 的集群池 IPAM 时，您需要配置可分配的容器组 CIDR 的大小。每节点 CIDR 大小是使用 Cilium 配置中的 clusterPoolIPv4MaskSize 设置来配置的。有关更多信息，请参阅 Cilium 文档中的 Expanding the cluster pool。

Cilium BGP 不起作用

如果您使用 Cilium BGP 控制面板向本地网络公开您的容器组或服务地址，则可以使用以下 Cilium CLI 命令来检查 BGP 是否在公开指向您的资源的路由。有关安装 Cilium CLI 的步骤，请参阅 Cilium 文档中的 Install the Cilium CLI。

如果 BGP 工作正常，则应在输出中看到会话状态为 `established` 的混合节点。您可能需要联系网络团队，确定环境的本地 AS、对等连接 AS 和对等连接地址的正确值。

```
cilium bgp peers
```

```
cilium bgp routes
```

如果您使用 Cilium BGP 来公开 `LoadBalancer` 类型的服务 IP，则您的 `CiliumLoadBalancerIPPool` 和服务上必须使用相同的标签，该标签应在 `CiliumBGPAdvertisement` 的选择器中使用。下面显示了一个示例。注意，如果您使用 Cilium BGP 来公开 LoadBalancer 类型服务的 IP，则在 Cilium 代理重启期间，BGP 路由可能会中断。有关更多信息，请参阅 Cilium 文档中的 Failure Scenarios。

服务：

```
kind: Service
apiVersion: v1
metadata:
  name: guestbook
  labels:
    app: guestbook
spec:
  ports:
  - port: 3000
    targetPort: http-server
  selector:
    app: guestbook
  type: LoadBalancer
```

CiliumLoadBalancerIPPool

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: guestbook-pool
  labels:
    app: guestbook
```

```
spec:
  blocks:
  - cidr: <CIDR to advertise>
  serviceSelector:
    matchExpressions:
      - { key: app, operator: In, values: [ guestbook ] }
```

CiliumBGPAdvertisement

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisements-guestbook
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - ExternalIP
          - LoadBalancerIP
      selector:
        matchExpressions:
          - { key: app, operator: In, values: [ guestbook ] }
```

Calico 故障排除

如果您在混合节点上运行 Calico 时遇到问题，请参阅 Calico 文档中的故障排除步骤。以下章节介绍了在混合节点上部署 Calico 时可能遇到的特有问题。

下表总结了 Calico 的组件以及这些组件默认情况下是在节点网络还是在容器组网络上运行。如果将 Calico 配置为使用 NAT 来处理传出的容器组流量，则必须将本地网络配置为将流量路由到本地节点 CIDR，并且 VPC 路由表必须配置本地节点 CIDR 的路由，并将中转网关（TGW）或虚拟专用网关（VGW）作为目标。如果未将 Calico 配置为使用 NAT 来处理传出的容器组流量，则必须将本地网络配置为将流量路由到本地容器组 CIDR，并且 VPC 路由表必须配置本地容器组 CIDR 的路由，并将中转网关（TGW）或虚拟专用网关（VGW）作为目标。

组件	Network
Calico API 服务器	节点

组件	Network
适用于 Kubernetes 的 Calico 控制器	Pod
Calico 节点代理	节点
Calico typha	节点
Calico CSI 节点驱动程序	Pod
Calico 操作符	节点

Calico 资源在封锁的节点上调度或运行

默认情况下，不作为 DaemonSet 运行的 Calico 资源具有灵活的容忍度，可以将其调度到尚未准备好调度或运行容器组的封锁节点上。您可以通过更改操作符安装来包括以下内容，从而收紧对非 DaemonSet Calico 资源的容忍度。

```
installation:
  ...
  controlPlaneTolerations:
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  calicoKubeControllersDeployment:
    spec:
      template:
        spec:
          tolerations:
          - effect: NoExecute
            key: node.kubernetes.io/unreachable
            operator: Exists
            tolerationSeconds: 300
          - effect: NoExecute
            key: node.kubernetes.io/not-ready
            operator: Exists
            tolerationSeconds: 300
```

```
  typhaDeployment:
    spec:
      template:
        spec:
          tolerations:
          - effect: NoExecute
            key: node.kubernetes.io/unreachable
            operator: Exists
            tolerationSeconds: 300
          - effect: NoExecute
            key: node.kubernetes.io/not-ready
            operator: Exists
            tolerationSeconds: 300
```

凭证故障排除

对于 Amazon SSM 混合激活和 Amazon IAM Roles Anywhere，您都可以通过在混合节点上运行以下命令来验证混合节点上的混合节点 IAM 角色凭证配置是否正确。确认节点名称和混合节点 IAM 角色名称是否符合预期。

```
sudo aws sts get-caller-identity
```

```
{
    "UserId": "ABCDEFGHIJKLM12345678910:<node-name>",
    "Account": "<aws-account-id>",
    "Arn": "arn:aws-cn:sts::<aws-account-id>:assumed-role/<hybrid-nodes-iam-role/<node-name>"
}
```

AmazonSystems Manager（SSM）故障排除

如果将 Amazon SSM 混合激活用作混合节点凭证，请注意以下将由 nodeadm 在混合节点上安装的 SSM 目录和构件。有关更多信息，请参阅《Amazon Systems Manager 用户指南》中的使用 SSM Agent。

说明	位置
SSM 代理	Ubuntu – /snap/amazon-ssm-agent/current/amazon-ssm-agent RHEL 和 AL2023 – /usr/bin/amazon-ssm-agent

说明	位置
SSM 代理日志	`/var/log/amazon/ssm`
Amazon 凭证	`/root/.aws/credentials`
SSM 安装 CLI	`/opt/ssm/ssm-setup-cli`

重新启动 SSM Agent

有些问题可以通过重新启动 SSM Agent 来解决。您可以使用以下命令重新启动此代理。

AL2023 和其他操作系统

```
systemctl restart amazon-ssm-agent
```

Ubuntu -

```
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

检查与 SSM 端点的连接

确认您可以从混合节点连接到 SSM 端点。有关 SSM 端点列表，请参阅 Amazon Systems Manager endpoints and quotas。对于 Amazon SSM 混合激活，请将以下命令中的 `us-west-2` 替换为 Amazon 区域。

```
ping ssm.us-west-2.amazonaws.com
```

查看已注册 SSM 实例的连接状态

您可以使用以下 Amazon CLI 命令检查通过 SSM 混合激活注册的实例的连接状态。请将 machine ID 替换为实例的机器 ID。

```
aws ssm get-connection-status --target mi-012345678abcdefgh
```

SSM 安装 CLI 校验和不匹配

运行 nodeadm install 时，如果您发现 ssm-setup-cli 校验和不匹配问题，应确认主机上没有早期版本的现有 SSM 安装。如果您的主机上有早期版本的 SSM 安装，请将其移除并重新运行 nodeadm install 以解决问题。

```
Failed to perform agent-installation/on-prem registration: error while verifying
 installed ssm-setup-cli checksum: checksum mismatch with latest ssm-setup-cli.
```

SSM **InvalidActivation**

如果您在将实例注册到 Amazon SSM 时看到错误，请确认 nodeConfig.yaml 中的 region、activationCode 和 activationId 正确无误。EKS 集群的 Amazon 区域必须与 SSM 混合激活区域一致。如果这些值的配置错误，则可能会看到与以下类似的错误。

```
ERROR Registration failed due to error registering the instance with Amazon SSM.
 InvalidActivation
```

SSM **ExpiredTokenException**：请求中包含的安全令牌已过期

如果 SSM Agent 无法刷新凭证，您可能会看到 ExpiredTokenException。在这种情况下，如果能够从混合节点连接到 SSM 端点，则可能需要重新启动 SSM Agent 以强制刷新凭证。

```
"msg":"Command failed","error":"operation error SSM: DescribeInstanceInformation, https
 response error StatusCode: 400, RequestID: eee03a9e-f7cc-470a-9647-73d47e4cf0be, api
 error ExpiredTokenException: The security token included in the request is expired"
```

运行注册计算机命令时出现 SSM 错误

如果在将计算机注册到 SSM 时看到错误，则可能需要重新运行 nodeadm install 以确保正确安装所有 SSM 依赖项。

```
"error":"running register machine command: , error: fork/exec /opt/aws/ssm-setup-cli:
 no such file or directory"
```

SSM **ActivationExpired**

运行 nodeadm init 时，如果在将实例注册到 SSM 时因激活已过期而出现错误，则需要创建新的 SSM 混合激活，使用新 SSM 混合激活的 activationCode 和 activationId 更新 nodeConfig.yaml，然后重新运行 nodeadm init。

```
"msg":"Command failed","error":"SSM activation expired. Please use a valid activation"
```

```
ERROR Registration failed due to error registering the instance with Amazon SSM.
 ActivationExpired
```

SSM 刷新缓存的凭证失败

如果发现无法刷新缓存的凭证，则说明 `/root/.aws/credentials` 文件可能已从主机中删除。首先检查您的 SSM 混合激活并确保其处于活动状态，并且混合节点已正确配置为使用该激活。检查 `/var/log/amazon/ssm` 中的 SSM Agent 日志，并在解决 SSM 端的问题后重新运行 `nodeadm init` 命令。

```
"Command failed","error":"operation error SSM: DescribeInstanceInformation, get
 identity: get credentials: failed to refresh cached credentials"
```

清理 SSM

要从主机上移除 SSM Agent，可以运行以下命令。

```
dnf remove -y amazon-ssm-agent
sudo apt remove --purge amazon-ssm-agent
snap remove amazon-ssm-agent
rm -rf /var/lib/amazon/ssm/Vault/Store/RegistrationKey
```

AmazonIAM Roles Anywhere 故障排除

如果将 Amazon IAM Roles Anywhere 用作混合节点凭证，请注意将由 `nodeadm` 在混合节点上安装的以下目录和构件。有关 IAM Roles Anywhere 故障排除的更多信息，请参阅《Amazon IAM Roles Anywhere 用户指南》中的 Troubleshooting Amazon IAM Roles Anywhere identity and access。

说明	位置
IAM Roles Anywhere CLI	`/usr/local/bin/aws_signing_helper`
默认证书位置和名称	`/etc/iam/pki/server.pem`
默认密钥位置和名称	`/etc/iam/pki/server.key`

IAM Roles Anywhere 刷新缓存的证书失败

如果您发现刷新缓存的证书失败，请检查 /etc/aws/hybrid/config 的内容并确认 nodeadm 配置中正确配置了 IAM Roles Anywhere。确认 /etc/iam/pki 存在。每个节点都必须具有唯一的证书和密钥。默认情况下，凭证提供者为 IAM Roles Anywhere 作时，nodeadm 将 /etc/iam/pki/server.pem 作为证书位置和名称，并将 /etc/iam/pki/server.key 作为私有密钥。在将证书和密钥放入目录之前，您可能需要首先使用 sudo mkdir -p /etc/iam/pki 创建目录。您可以使用以下命令验证证书的内容。

```
openssl x509 -text -noout -in server.pem
```

```
open /etc/iam/pki/server.pem: no such file or directory
could not parse PEM data
Command failed {"error": "... get identity: get credentials: failed to refresh cached
 credentials, process provider error: error in credential_process: exit status 1"}
```

IAM Roles Anywhere 无权执行 **sts:AssumeRole**

在 kubelet 日志中，如果您在使用 IAM Roles Anywhere 时看到 sts:AssumeRole 操作出现访问被拒绝的问题，请检查混合节点 IAM 角色的信任策略，以确认允许 IAM Roles Anywhere 服务主体代入该混合节点 IAM 角色。此外，请确认在混合节点 IAM 角色信任策略中正确配置了信任锚 ARN，并且已将混合节点 IAM 角色添加到 IAM Roles Anywhere 配置文件中。

```
could not get token: AccessDenied: User: ... is not authorized to perform:
 sts:AssumeRole on resource: ...
```

IAM Roles Anywhere 无权设置 **roleSessionName**

在 kubelet 日志中，如果您发现设置 roleSessionName 时出现访问被拒绝的问题，请确认您已将 IAM Roles Anywhere 配置文件的 acceptRoleSessionName 设置为 true。

```
AccessDeniedException: Not authorized to set roleSessionName
```

操作系统故障排除

RHEL

授权或订阅管理器注册失败

如果您运行的是 nodeadm install，但由于授权注册问题而无法安装混合节点依赖项，请确保在主机上正确设置了 Red Hat 用户名和密码。

```
This system is not registered with an entitlement server
```

Ubuntu

找不到 GLIBC

如果您的操作系统为 Ubuntu 操作系统，混合节点的凭证提供者为 IAM Roles Anywhere，但出现找不到 GLIBC 的问题，则可以通过手动安装该依赖项来解决问题。

```
GLIBC_2.32 not found (required by /usr/local/bin/aws_signing_helper)
```

运行以下命令安装依赖项：

```
ldd --version
sudo apt update && apt install libc6
sudo apt install glibc-source
```

Bottlerocket

如果启用了 Bottlerocket 管理容器，则可以使用 SSH 访问该容器，从而使用更高的权限进行高级调试和故障排除。以下章节包含需要在 Bottlerocket 主机的上下文中运行的命令。进入管理容器后，您可以运行 `sheltie` 来获取 Bottlerocket 主机中的完整根 Shell。

```
sheltie
```

此外还可以通过在各个命令前加上前缀 `sudo chroot /.bottlerocket/rootfs`，从而在管理容器 Shell 中运行以下章节中的命令。

```
sudo chroot /.bottlerocket/rootfs <command>
```

使用 logdog 收集日志

Bottlerocket 提供了 `logdog` 实用程序，可用来高效地收集日志和系统信息，以满足故障排除的需要。

```
logdog
```

`logdog` 实用程序可从 Bottlerocket 主机的不同位置收集日志，然后组合成压缩包。默认情况下，该压缩包将在 `/var/log/support/bottlerocket-logs.tar.gz` 位置创建，并且可以从主机容器的 `/.bottlerocket/support/bottlerocket-logs.tar.gz` 位置访问。

使用 journalctl 访问系统日志

您可以使用以下命令检查各种系统服务（例如 kubelet、containerd 等）的状态以及查看其日志。-f 标志会实时跟踪日志。

要检查 kubelet 服务状态和检索 kubelet 日志，您可以运行以下命令：

```
systemctl status kubelet
journalctl -u kubelet -f
```

要检查 containerd 服务状态和检索已编排的 containerd 实例的日志，您可以运行以下命令：

```
systemctl status containerd
journalctl -u containerd -f
```

要检查 host-containerd 服务状态和检索主机 containerd 实例的日志，您可以运行以下命令：

```
systemctl status host-containerd
journalctl -u host-containerd -f
```

要检索引导容器和主机容器的日志，您可以运行以下命令：

```
journalctl _COMM=host-ctr -f
```

使用集群的应用程序数据存储

您可以在 Amazon EKS 中使用多种 Amazon 存储服务，来满足应用程序的存储需求。通过由 Amazon 支持的各种容器存储接口（CSI）驱动程序，您可以轻松使用 Amazon EBS、Amazon S3、Amazon EFS、Amazon FSX 和 Amazon File Cache 来满足 Amazon EKS 上所运行应用程序的存储需求。要管理 Amazon EKS 集群的备份，请参阅 Amazon EKS 的 Amazon 备份支持。

本章介绍 Amazon EKS 集群的存储选项。

主题

- 将 Kubernetes 卷存储与 Amazon EBS 结合使用
- 将弹性文件系统存储与 Amazon EFS 结合使用
- 将高性能应用程序存储与适用于 Lustre 的 Amazon FSx 结合使用
- 将适用于 NetApp ONTAP 的 FSx 与高性能应用程序存储结合使用
- 将适用于 OpenZFS 的 Amazon FSx 与数据存储结合使用
- 使用 Amazon File Cache 减少延迟
- 使用适用于 Amazon S3 的 Mountpoint CSI 驱动程序访问适用于 Amazon S3 对象
- 为 CSI 卷启用快照功能

将 Kubernetes 卷存储与 Amazon EBS 结合使用

> **Note**
>
> 新增：Amazon EKS 自动模式可自动执行块存储例行任务。了解如何the section called “部署有状态工作负载”。

Amazon Elastic Block Store（Amazon EBS）Container Storage Interface（CSI）驱动程序管理作为您所创建 Kubernetes 卷存储的 Amazon EBS 卷的生命周期。Amazon EBS CSI 驱动程序为以下类型的 Kubernetes 卷制作 Amazon EBS 卷：通用临时卷和持久性卷。

注意事项

- 您无需在 EKS 自动模式集群上安装 Amazon EBS CSI 控制器。

- 您无法将 Amazon EBS 卷挂载到 Fargate 容器组（pod）。
- 您可以在 Fargate 节点上运行 Amazon EBS CSI 控制器，但要在 Amazon EBS CSI 节点 `DaemonSet` 只能在 Amazon EC2 实例上运行。
- Amazon EBS 卷和 Amazon EBS CSI 驱动程序与 Amazon EKS 混合节点功能不兼容。
- 将为最新的附加组件版本和一个先前版本提供支持。在最新版本中发现的错误或漏洞将在新的次要版本中向后移植到先前的版本。
- EKS 自动模式需要存储类使用 `ebs.csi.eks.amazonaws.com` 作为置备程序。标准 Amazon EBS CSI 驱动程序 (`ebs.csi.aws.com`) 单独管理自己的卷。要在 EKS 自动模式下使用现有卷，请使用卷快照将其迁移到使用自动模式置备程序的存储类。

> **Important**
>
> 要使用 Amazon EBS CSI 驱动程序的快照功能，必须先安装 CSI 快照控制器。有关更多信息，请参阅 the section called “CSI 快照控制器”。

先决条件

- 现有集群。要查看所需的平台版本，请运行以下命令。

```
aws eks describe-addon-versions --addon-name aws-ebs-csi-driver
```

- EBS CSI 驱动程序需要 Amazon IAM 权限。
  - Amazon 建议使用 EKS 容器组身份。有关更多信息，请参阅 the section called “EKS 容器组身份设置概述”。
  - 有关服务账户 IAM 角色的信息，请参阅the section called “IAM OIDC 提供者”。

第 1 步：创建 IAM 角色

Amazon EBS CSI 插件需要 IAM 权限才能代表您调用 Amazon API。如果您未完成这些步骤，则在尝试安装附加组件并运行 `kubectl describe pvc` 时，将会显示 `failed to provision volume with StorageClass` 以及 `could not create volume in EC2: UnauthorizedOperation` 错误。有关更多信息，请参阅 GitHub 上的设置驱动程序权限。

> **Note**
>
> 容器组（pod）将有权访问分配给 IAM 角色的权限，除非您阻止对 IMDS 的访问。有关更多信息，请参阅 the section called “最佳实践”。

以下过程为您演示了如何创建 IAM 角色并向其附加 Amazon 托管策略。要实现此过程，可以使用以下过程之一：

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”
- the section called “Amazon CLI”

> **Note**
>
> 您可创建一项权限范围最小化的自主管理型策略。审阅 AmazonEBSCSIDriverPolicy 并创建一项权限精简的自定义 IAM 策略。

> **Note**
>
> 此过程中的具体步骤是为将驱动程序用作 Amazon EKS 附加组件而编写的。要将驱动程序用作自行管理的附加组件，需要遵循不同的步骤。有关更多信息，请参阅 GitHub 上的 Set up driver permissions。

eksctl

1. 创建 IAM 角色并附加到策略。Amazon 维护 Amazon 托管策略，或者您可创建自己的自定义策略。您可以使用以下命令创建 IAM 角色并将 Amazon 托管策略附加到其上。将 *my-cluster* 替换为您的集群的名称。此命令将部署 Amazon CloudFormation 堆栈，该堆栈将创建 IAM 角色，并会将 IAM 策略附加到该堆栈。

```
eksctl create iamserviceaccount \
        --name ebs-csi-controller-sa \
        --namespace kube-system \
        --cluster my-cluster \
```

```
        --role-name AmazonEKS_EBS_CSI_DriverRole \
        --role-only \
        --attach-policy-arn arn:aws-cn:iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicy \
        --approve
```

2. 如果您不使用自定义 KMS 密钥，可以跳过此步骤。如果在 Amazon EBS 卷上使用一个密钥进行加密，请根据需要自定义 IAM 角色。例如，执行以下操作：

   a. 复制并在新的 kms-key-for-encryption-on-ebs.json 文件中粘贴以下代码。将 *custom-key-arn* 替换为自定义 KMS 密钥 ARN。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
        }
      ]
```

```
    }
```

b. 创建策略。您可以将 *KMS_Key_For_Encryption_On_EBS_Policy* 更改为其它名称。但是，如果更改了名称，请确保在后面的步骤中也进行相应的更改。

```
aws iam create-policy \
      --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
      --policy-document file://kms-key-for-encryption-on-ebs.json
```

c. 使用以下命令将 IAM policy 附加到该角色。请将 *111122223333* 替换为您的账户 ID。

```
aws iam attach-role-policy \
      --policy-arn arn:aws-cn:iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

Amazon Web Services 管理控制台

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 Roles（角色）。
3. 在 Roles（角色）页面上，选择 Create role（创建角色）。
4. 在 Select trusted entity（选择受信任的实体）页面上，请执行以下操作：
   a. 在 Trusted entity type（受信任的实体类型）部分中，选择 Web identity（Web 身份）。
   b. 对于 Identity provider（身份提供商），为您的集群选择 OpenID Connect provider URL（OpenID Connect 提供程序 URL）（如 Amazon EKS 中的 Overview（概述）下所示）。
   c. 对于 Audience (受众)，请选择 `sts.amazonaws.com`。
   d. 选择下一步。
5. 在 Add permissions（添加权限）页面上，请执行以下操作：
   a. 在 Filter policies (筛选器策略) 框中，输入 `AmazonEBSCSIDriverPolicy`。
   b. 选中搜索返回的 `AmazonEBSCSIDriverPolicy` 左侧的复选框。
   c. 选择下一步。
6. 在 Name, review, and create（命名、查看和创建）页面中，请执行以下操作：
   a. 对于角色名称，为角色输入唯一名称，例如 *AmazonEKS_EBS_CSI_DriverRole*。
   b. 在添加标签（可选）下，将标签作为键值对附加，以将元数据添加到角色。有关在 IAM 中使用标签的更多信息，请参阅《IAM 用户指南》 中的标记 IAM 资源。

c. 选择创建角色。

7. 创建角色后，在控制台中选择角色以将其打开进行编辑。
8. 选择 Trust relationships（信任关系）选项卡，然后选择 Edit trust policy（编辑信任策略）。
9. 该行看起来类似于以下行：

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

在上一行末尾添加逗号，然后在前一行后面添加以下行。将 *region-code* 替换为您的集群所在的 Amazon 区域。将 *EXAMPLED539D4633E53DE1B71EXAMPLE* 替换为您的集群的 OIDC 提供者 ID。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:ebs-csi-controller-sa"
```

10选择 Update policy（更新策略）以完成操作。

11如果在 Amazon EBS 卷上使用自定义 KMS 密钥进行加密，请根据需要自定义 IAM 角色。例如，执行以下操作：

a. 在左侧导航窗格中，选择 Policies（策略）。

b. 在策略页面上，选择 Create a policy (创建策略)。

c. 在创建策略页面上，选择 JSON 选项卡。

d. 将以下代码复制并粘贴到编辑器中，并将 *custom-key-arn* 替换为自定义 KMS 密钥 ARN。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
```

```
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
        }
      ]
    }
```

e. 选择下一步：标签。

f. 在 Add tags (Optional)（添加标签（可选））页面上，选择 Next: Review（下一步：审核）。

g. 在名称中，输入策略的唯一名称（例如，*KMS_Key_For_Encryption_On_EBS_Policy*）。

h. 选择创建策略。

i. 在左侧导航窗格中，选择 角色。

j. 在控制台中选择 ***AmazonEKS_EBS_CSI_DriverRole*** 以打开它进行编辑。

k. 从添加权限下拉列表中，选择附加策略。

l. 在筛选策略框中，输入 *KMS_Key_For_Encryption_On_EBS_Policy*。

m. 选中搜索中返回的 *KMS_Key_For_Encryption_On_EBS_Policy* 左侧的复选框。

n. 选择附加策略。

Amazon CLI

1. 查看集群的 OIDC 提供者 URL。将 *my-cluster* 替换为您的集群名称。如果命令的输出为 None，请查看先决条件。

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

示例输出如下。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. 创建 IAM 角色并向其授予 AssumeRoleWithWebIdentity 操作。

   a. 将以下内容复制到名为 aws-ebs-csi-driver-trust-policy.json 的文件中。请将 *111122223333* 替换为您的账户 ID。将 *EXAMPLED539D4633E53DE1B71EXAMPLE* 和 *region-code* 替换为上一步中返回的值。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
          },
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {
            "StringEquals": {
              "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
              "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:ebs-csi-
controller-sa"
            }
          }
        }
      ]
    }
```

   b. 创建角色。您可以将 *AmazonEKS_EBS_CSI_DriverRole* 更改为其它名称。如果更改了名称，请确保在后面的步骤中也进行相应的更改。

```
aws iam create-role \
      --role-name AmazonEKS_EBS_CSI_DriverRole \
      --assume-role-policy-document file://"aws-ebs-csi-driver-trust-policy.json"
```

3. 附加策略。Amazon 维护 Amazon 托管策略，或者您可创建自己的自定义策略。使用以下命令以将 Amazon 托管策略附加到角色。

```
aws iam attach-role-policy \
```

```
    --policy-arn arn:aws-cn:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
    --role-name AmazonEKS_EBS_CSI_DriverRole
```

4. 如果在 Amazon EBS 卷上使用自定义 KMS 密钥进行加密，请根据需要自定义 IAM 角色。例如，执行以下操作：

   a. 复制并在新的 kms-key-for-encryption-on-ebs.json 文件中粘贴以下代码。将 *custom-key-arn* 替换为自定义 KMS 密钥 ARN。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
        }
      ]
    }
```

   b. 创建策略。您可以将 *KMS_Key_For_Encryption_On_EBS_Policy* 更改为其它名称。但是，如果更改了名称，请确保在后面的步骤中也进行相应的更改。

```
aws iam create-policy \
      --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
      --policy-document file://kms-key-for-encryption-on-ebs.json
```

c. 使用以下命令将 IAM policy 附加到该角色。请将 *111122223333* 替换为您的账户 ID。

```
aws iam attach-role-policy \
      --policy-arn arn:aws-cn:iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

现在您已经创建了 Amazon EBS CSI 驱动程序 IAM 角色，可以前往完成下一部分。使用此 IAM 角色部署附加组件后，该附加组件会创建一个名为 `ebs-csi-controller-sa` 的服务账户，并配置为使用该服务账户。服务账户绑定到被分配了所需的 Kubernetes 权限的 Kubernetes `clusterrole`。

第 2 步：获取 Amazon EBS CSI 驱动程序

我们建议您通过 Amazon EKS 附加组件来安装 Amazon EBS CSI 驱动程序，从而提高安全性并减少工作量。要将 Amazon EKS 附加组件添加到您的集群，请参阅 the section called “创建附加组件”。有关附加组件的更多信息，请参阅 the section called “Amazon EKS 附加组件”。

Important

在将 Amazon EBS 驱动程序添加为 Amazon EKS 附加组件之前，请确认您的集群上没有安装该驱动程序的自行管理版本。如果安装了这样的版本，请参阅 GitHub 上的 Uninstalling a self-managed Amazon EBS CSI driver。

Note

默认情况下，EBS CSI 所使用的 RBAC 角色拥有对节点执行变更操作的权限，以支持其污点移除功能。受限于 Kubernetes RBAC 的机制，该权限同时也允许此角色对集群内的其余节点执行变更操作。Helm 图表中设有一个参数 (`node.serviceAccount.disableMutation`)，可禁用 ebs-csi-node 服务账户的节点变更类 RBAC 权限。启用此参数后，污点移除等驱动功能将无法正常运行。

如果要对 Amazon EBS CSI 驱动程序进行自主管理型安装，请参阅 GitHub 上的 Installation。

第 3 步：部署示例应用程序

您可以部署各种示例应用程序并根据需要对其进行修改。有关更多信息，请参阅 GitHub 上的 Kubernetes Examples。

将弹性文件系统存储与 Amazon EFS 结合使用

Amazon Elastic File System（Amazon EFS）提供无服务器的完全弹性文件存储，因此，您无需预置或管理存储容量和性能，即可共享文件数据。Amazon EFS Container Storage Interface (CSI) 驱动程序提供了一个 CSI 接口，允许在Amazon上运行的 Kubernetes 集群管理 Amazon EFS 文件系统的生命周期。本主题介绍了如何部署 Amazon EFS CSI 驱动程序到您的 Amazon EKS 集群。

注意事项

- Amazon EFS CSI 驱动程序与基于 Windows 的容器映像不兼容。
- 不能将持久性卷的动态预置与 Fargate 节点结合使用，但可以使用静态预置。
- 动态预置需要 1.2 或更高版本的驱动程序。您可以在任何支持的 Amazon EKS 集群版本上通过 `1.1` 版本的驱动程序，对持久性卷使用静态预置（请参阅 Amazon EKS 支持的版本）。
- 此驱动程序的 1.3.2 版或更高版本支持 Arm64 架构，包括基于 Amazon EC2 Graviton 的实例。
- 版本 1.4.2 或更高版本的此驱动程序支持使用 FIPS 装载文件系统。
- 注意 Amazon EFS 的资源配额。例如，可以为每个 Amazon EFS 文件系统创建 1000 个接入点的配额。有关更多信息，请参阅您无法更改的 Amazon EFS 资源配额。
- 从 2.0.0 版本开始，此驱动程序使用 `stunnel` 切换到 `efs-proxy` 进行 TLS 连接。使用 `efs-proxy` 时，它将打开多个线程，其数量等于一加上它正在运行的节点的内核数。
- Amazon EFS CSI 驱动程序与 Amazon EKS 混合节点功能不兼容。

先决条件

- Amazon EFS CSI 驱动程序需要 Amazon Identity and Access Management（IAM）权限。
  - Amazon 建议使用 EKS 容器组身份。有关更多信息，请参阅 the section called “EKS 容器组身份设置概述”。
  - 有关服务账户的 IAM 角色以及为您的集群设置 IAM OpenID Connect（OIDC）提供者的信息，请参阅the section called “IAM OIDC 提供者”。

- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 2.12.3 或更高版本，或版本 1.27.160 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 1.29，则可以将 `kubectl` 的 1.28、1.29 或 1.30 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。

> **Note**
>
> Fargate 上运行的容器组（pod）会自动挂载 Amazon EFS 文件系统，无需手动执行驱动程序安装步骤。

第 1 步：创建 IAM 角色

Amazon EFS CSI 驱动程序需要 IAM 权限才能与您的文件系统进行交互。创建 IAM 角色并向其附加所需的 Amazon 托管策略。要实现此过程，可以使用以下过程之一：

- the section called “`eksctl`”
- the section called “Amazon Web Services 管理控制台”
- the section called “Amazon CLI”

> **Note**
>
> 此过程中的具体步骤是为将驱动程序用作 Amazon EKS 附加组件而编写的。有关自主管理型安装的详细信息，请参阅 GitHub 上的 Set up driver permission。

eksctl

如果使用容器组身份

使用 eksctl 运行以下命令以创建 IAM 角色和容器组身份。将 my-cluster 替换为您的集群名称。您也可以将 AmazonEKS_EFS_CSI_DriverRole 替换为其他名称。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create podidentityassociation \
    --service-account-name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --permission-policy-arns arn:aws-cn:iam::aws:policy/service-role/
AmazonEFSCSIDriverPolicy
```

如果对服务账户使用 IAM 角色

使用 eksctl 运行以下命令以创建 IAM 角色。将 my-cluster 替换为您的集群名称。您也可以将 AmazonEKS_EFS_CSI_DriverRole 替换为其他名称。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --role-only \
    --attach-policy-arn arn:aws-cn:iam::aws:policy/service-role/
AmazonEFSCSIDriverPolicy \
    --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
    sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
 "$TRUST_POLICY"
```

Amazon Web Services 管理控制台

使用 Amazon Web Services 管理控制台 运行以下命令以创建 IAM 角色。

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 Roles（角色）。
3. 在 Roles（角色）页面上，选择 Create role（创建角色）。
4. 在 Select trusted entity（选择受信任的实体）页面上，请执行以下操作：
   a. 如果使用 EKS 容器组身份：
      i. 在可信实体类型部分中，选择 Amazon 服务。
      ii. 在服务或应用场景下拉列表中，选择 EKS。
      iii. 在应用场景部分中，选择 EKS – 容器组身份。
      iv. 选择下一步。
   b. 如果对服务账户使用 IAM 角色：
      i. 在 Trusted entity type（受信任的实体类型）部分中，选择 Web identity（Web 身份）。
      ii. 对于 Identity provider（身份提供商），为您的集群选择 OpenID Connect provider URL（OpenID Connect 提供程序 URL）（如 Amazon EKS 中的 Overview（概述）下所示）。
      iii. 对于 Audience (受众)，请选择 `sts.amazonaws.com`。
      iv. 选择下一步。
5. 在 Add permissions（添加权限）页面上，请执行以下操作：
   a. 在 Filter policies (筛选器策略) 框中，输入 `AmazonEFSCSIDriverPolicy`。
   b. 选中搜索返回的 `AmazonEFSCSIDriverPolicy` 左侧的复选框。
   c. 选择下一步。
6. 在 Name, review, and create（命名、查看和创建）页面中，请执行以下操作：
   a. 对于 Role name (角色名称)，为角色输入唯一名称，例如 `AmazonEKS_EFS_CSI_DriverRole`。
   b. 在添加标签（可选）下，将标签作为键值对附加，以将元数据添加到角色。有关在 IAM 中使用标签的更多信息，请参阅《IAM 用户指南》 中的标记 IAM 资源。
   c. 选择创建角色。
7. 创建角色后：
   a. 如果使用 EKS 容器组身份：
      i. 打开 Amazon EKS 控制台。
      ii. 在左侧导航窗格中，选择集群，然后为您要配置 EKS 容器组身份关联的集群选择名称。
      iii. 选择访问选项卡。

iv. 在容器组身份关联中，选择创建。

v. 选择 IAM 角色下拉列表，然后选择您新创建的角色。

vi. 选择 Kubernetes 命名空间字段并输入 `kube-system`。

vii.选择 Kubernetes 服务账户字段并输入 `efs-csi-controller-sa`。

viii选择创建。

ix. 有关创建容器组身份关联的更多信息，请参阅the section called “创建容器组身份关联（Amazon 控制台）”。

b. 如果对服务账户使用 IAM 角色：

i. 选择要打开进行编辑的角色。

ii. 选择 Trust relationships（信任关系）选项卡，然后选择 Edit trust policy（编辑信任策略）。

iii. 该行看起来类似于以下行：

```
"oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud":
 "sts.amazonaws.com"
```

将以下行添加到上一行的上方。将 `<region-code>` 替换为集群所在的 Amazon 区域。将 `<EXAMPLED539D4633E53DE1B71EXAMPLE>` 替换为集群的 OIDC 提供者 ID。

```
"oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-*",
```

iv. 将 `Condition` 运算符从 `"StringEquals"` 修改为 `"StringLike"`。

v. 选择 Update policy（更新策略）以完成操作。

Amazon CLI

使用 Amazon CLI 运行以下命令以创建 IAM 角色。

如果使用容器组身份

1. 创建向 `pods.eks.amazonaws.com` 服务授予 `AssumeRole` 和 `TagSession` 操作权限的 IAM 角色。

a. 将以下内容复制到名为 `aws-efs-csi-driver-trust-policy-pod-identity.json` 的文件中。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

b. 创建角色。将 my-cluster 替换为您的集群名称。您也可以将 AmazonEKS_EFS_CSI_DriverRole 替换为其他名称。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws iam create-role \
  --role-name $role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-identity.json"
```

2. 使用以下命令以将所需的 Amazon 托管策略附加到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --role-name $role_name
```

3. 运行以下命令以创建容器组身份关联。将 arn:aws-cn:iam::<111122223333>:role/my-role 替换为在之前步骤中创建的角色。

```
aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn {arn-aws}iam::<111122223333>:role/my-role --namespace kube-system --service-account efs-csi-controller-sa
```

4. 有关创建容器组身份关联的更多信息，请参阅the section called “创建容器组身份关联（Amazon 控制台）”。

如果对服务账户使用 IAM 角色

1. 查看集群的 OIDC 提供者 URL。将 my-cluster 替换为您的集群名称。您也可以将 AmazonEKS_EFS_CSI_DriverRole 替换为其他名称。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws eks describe-cluster --name $cluster_name --query "cluster.identity.oidc.issuer"
 --output text
```

示例输出如下。

```
https://oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>
```

如果命令的输出为 None，请查看先决条件。

2. 创建授予 AssumeRoleWithWebIdentity 操作权限的 IAM 角色。

a. 将以下内容复制到名为 aws-efs-csi-driver-trust-policy.json 的文件中。请将 <111122223333> 替换为您的账户 ID。将 <EXAMPLED539D4633E53DE1B71EXAMPLE> 和 <region-code> 替换为在上一步中返回的值。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:efs-csi-
*",
```

```
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

b. 创建角色。

```
aws iam create-role \
  --role-name $role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy.json"
```

3. 使用以下命令以将所需的 Amazon 托管策略附加到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --role-name $role_name
```

第 2 步：获取 Amazon EFS CSI 驱动程序

我们建议您通过 Amazon EKS 附加组件方式来安装 Amazon EFS CSI 驱动程序。要将 Amazon EKS 附加组件添加到您的集群，请参阅 the section called “创建附加组件”。有关附加组件的更多信息，请参阅 the section called “Amazon EKS 附加组件”。如果您无法使用 Amazon EKS 附加组件，我们鼓励您向容器路线图 GitHub 存储库提交有关您为什么无法使用的问题。

> **Important**
>
> 在将 Amazon EFS 驱动程序添加为 Amazon EKS 附加组件之前，请确认集群上没有安装该驱动程序的自行管理版本。如果安装了这样的版本，请参阅 GitHub 上的 Uninstalling the Amazon EFS CSI Driver。

如果要对 Amazon EFS CSI 驱动程序进行自主管理型安装，请参阅 GitHub 上的 Installation。

第 3 步：创建 Amazon EFS 文件系统

要创建 Amazon EFS 文件系统，请参阅 GitHub 上的为 Amazon EKS 创建 Amazon EKS 文件系统。

第 4 步：部署示例应用程序

您可以部署各种示例应用程序并根据需要对其进行修改。有关更多信息，请参阅 GitHub 上的示例。

将高性能应用程序存储与适用于 Lustre 的 Amazon FSx 结合使用

Amazon FSx for Lustre Container Storage Interface (CSI) 驱动程序提供了一个 CSI 接口，允许 Amazon EKS 集群管理 Amazon FSx for Lustre 文件系统的生命周期。有关更多信息，请参阅适用于 Lustre 的 Amazon FSx 用户指南。

要详细了解如何将适用于 Lustre 的 Amazon FSx CSI 驱动程序部署到 Amazon EKS 集群并验证其是否起作用，请参阅the section called “部署驱动程序”。

部署适用于 Lustre 的 FSx 驱动程序

本主题演示如何将适用于 Lustre 的 FSx CSI 驱动程序部署到 Amazon EKS 集群，以及验证驱动程序是否正常工作。建议使用最新版本的驱动程序。有关可用版本，请参阅 GitHub 上的 CSI Specification Compatibility Matrix（CSI 规范兼容性矩阵）。

> **Note**
>
> Fargate 或 Amazon EKS 混合节点功能均不支持该驱动程序。

有关可用参数的详细说明和演示驱动程序功能的完整示例，请参阅 GitHub 上的 FSx for Lustre Container Storage Interface (CSI) driver 项目。

先决条件

- 现有集群。
- Amazon FSx CSI 驱动程序 EKS 附加组件需要 EKS 容器组身份代理进行身份验证。如果没有此组件，附加组件将因错误 `Amazon EKS Pod Identity agent is not installed in the cluster` 而失败，从而无法进行卷操作。在部署 FSx CSI 驱动程序附加组件之前或之后安装容器组身份代理。有关更多信息，请参阅 the section called “设置代理”。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令

行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。

- 您的设备或 Amazon CloudShell 上安装 `0.215.0` 版或更高版本的 `eksctl` 命令行工具。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的 Installation。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。

第 1 步：创建 IAM 角色

Amazon FSx CSI 插件需要 IAM 权限才能代表您调用 Amazon API。

> **Note**
>
> 容器组（pod）将有权访问分配给 IAM 角色的权限，除非您阻止对 IMDS 的访问。有关更多信息，请参阅 the section called “最佳实践”。

以下过程为您演示了如何创建 IAM 角色并向其附加 Amazon 托管策略。

1. 使用以下命令创建 IAM 角色并附加 Amazon 托管策略。将 `my-cluster` 替换为您要使用的集群的名称。此命令将部署 Amazon CloudFormation 堆栈，该堆栈将创建 IAM 角色，并会将 IAM 策略附加到该堆栈。

   ```
   eksctl create iamserviceaccount \
       --name fsx-csi-controller-sa \
       --namespace kube-system \
       --cluster my-cluster \
       --role-name AmazonEKS_FSx_CSI_DriverRole \
       --role-only \
       --attach-policy-arn arn:aws-cn:iam::aws:policy/AmazonFSxFullAccess \
       --approve
   ```

   创建服务账户时，您将看到几行输出。输出的最后一行类似于以下行。

   ```
   [#]  1 task: {
   ```

```
    2 sequential sub-tasks: {
        create IAM role for serviceaccount "kube-system/fsx-csi-controller-sa",
        create serviceaccount "kube-system/fsx-csi-controller-sa",
    } }
[#]  building iamserviceaccount stack "eksctl-my-cluster-addon-iamserviceaccount-
kube-system-fsx-csi-controller-sa"
[#]  deploying stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-
controller-sa"
[#]  waiting for CloudFormation stack "eksctl-my-cluster-addon-iamserviceaccount-
kube-system-fsx-csi-controller-sa"
[#]  created serviceaccount "kube-system/fsx-csi-controller-sa"
```

记录已部署的 Amazon CloudFormation 堆栈的名称。在上面的示例输出中，堆栈的名称为 `eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`。

现在您已经创建了 Amazon FSx CSI 驱动程序 IAM 角色，可以前往完成下一部分。使用此 IAM 角色部署附加组件后，该附加组件会创建一个名为 `fsx-csi-controller-sa` 的服务账户，并配置为使用该服务账户。服务账户绑定到被分配了所需的 Kubernetes 权限的 Kubernetes `clusterrole`。

第 2 步：安装 Amazon FSx CSI 驱动程序

我们建议您通过 Amazon EKS 附加组件来安装 Amazon FSx CSI 驱动程序，从而提高安全性并减少工作量。要将 Amazon EKS 附加组件添加到您的集群，请参阅 the section called “创建附加组件”。有关附加组件的更多信息，请参阅 the section called “Amazon EKS 附加组件”。

Important

集群中预先安装的 Amazon FSx CSI 驱动程序可能会导致附加组件安装失败。当您在存在非 EKS FSx CSI 驱动程序的情况下尝试安装 Amazon EKS 附加组件版本时，由于资源冲突，安装将失败。在安装过程中使用 `OVERWRITE` 标志解决此问题。

```
aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
 --resolve-conflicts OVERWRITE
```

如果要对 Amazon FSx CSI 驱动程序进行自主管理型安装，请参阅 GitHub 上的 Installation。

第 3 步：部署存储类、持久卷声明和示例应用程序

此过程利用 FSx for Lustre Container Storage Interface（CSI）驱动程序 GitHub 存储库来使用动态预置的 FSx for Lustre 卷。

1. 注意集群的安全组。您可以在 Amazon Web Services 管理控制台中的联网部分下或通过使用以下 Amazon CLI 命令来查看它。将 `my-cluster` 替换为您要使用的集群的名称。

```
aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

2. 根据《Amazon FSx for Lustre 用户指南》中的 Amazon VPC 安全组显示的标准，为您的 Amazon FSx 文件系统创建安全组。对于 VPC，选择 Networking（联网）部分下显示的集群的 VPC。对于“与 Lustre 客户端关联的安全组”，请使用您的集群安全组。您可以单独保留出站规则以允许所有流量。
3. 使用下面的命令下载存储类清单。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/
examples/kubernetes/dynamic_provisioning/specs/storageclass.yaml
```

4. 编辑 `storageclass.yaml` 文件的参数部分。将所有 example value 替换为您自己的值。

```
parameters:
  subnetId: subnet-0eabfaa81fb22bcaf
  securityGroupIds: sg-068000ccf82dfba88
  deploymentType: PERSISTENT_1
  automaticBackupRetentionDays: "1"
  dailyAutomaticBackupStartTime: "00:00"
  copyTagsToBackups: "true"
  perUnitStorageThroughput: "200"
  dataCompressionType: "NONE"
  weeklyMaintenanceStartTime: "7:09:00"
  fileSystemTypeVersion: "2.12"
```

- **`subnetId`** – 应在其中创建 Amazon FSx for Lustre 文件系统的子网 ID。并非所有可用区都支持 Amazon FSx for Lustre。打开 https://console.aws.amazon.com/fsx/ 中 Amazon FSx for Lustre 控制台，确认您要使用的子网是否位于支持的可用区中。该子网可以包含您的节点，也可以是不同的子网或 VPC：
  - 您可以通过在 Compute（计算）部分下选择节点组来在 Amazon Web Services 管理控制台 中检查节点子网。

  - 如果您指定的子网不是节点所在的子网，则必须已连接 VPC，并且必须确保已在您的安全组中打开必要的端口。
- **securityGroupIds** – 您为文件系统创建的安全组的 ID。
- **deploymentType**（可选）– 文件系统部署类型。有效值为 SCRATCH_1、SCRATCH_2、PERSISTENT_1 和 PERSISTENT_2。有关部署类型的更多信息，请参阅创建 Amazon FSx for Lustre 文件系统。
- 其他参数（可选）– 有关其他参数的信息，请参阅 GitHub 上的编辑 StorageClass。

5. 创建存储类清单。

```
kubectl apply -f storageclass.yaml
```

示例输出如下。

```
storageclass.storage.k8s.io/fsx-sc created
```

6. 下载持久卷注册清单。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/
examples/kubernetes/dynamic_provisioning/specs/claim.yaml
```

7. （可选）编辑 claim.yaml 文件。根据您的存储需求和上一步选择的 deploymentType，将 1200Gi 更改为下面列出的增量值之一。

```
storage: 1200Gi
```

- SCRATCH_2 和 PERSISTENT – 1.2 TiB、2.4 TiB，或 2.4TiB 之上 2.4TiB 的增量。
- SCRATCH_1 和 1.2 TiB – 2.4 TiB、3.6 TiB，或 3.6TiB 之上 3.6TiB 的增量。

8. 创建持久卷注册。

```
kubectl apply -f claim.yaml
```

示例输出如下。

```
persistentvolumeclaim/fsx-claim created
```

9. 确认已预配置文件系统。

```
kubectl describe pvc
```

示例输出如下。

```
Name:          fsx-claim
Namespace:     default
StorageClass:  fsx-sc
Status:        Bound
[...]
```

> **Note**
>
> Status 可能会在 5-10 分钟内显示为 Pending，然后才会更改为 Bound。在 Status 变成 Bound 之前，请勿继续执行下一步。如果 Status 显示 Pending 10 分钟以上，使用 Events 中的警告消息作为解决任何问题的参考。

10部署示例应用程序。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/pod.yaml
```

11验证示例应用程序正在运行。

```
kubectl get pods
```

示例输出如下。

```
NAME      READY   STATUS              RESTARTS   AGE
fsx-app   1/1     Running             0          8s
```

12验证应用程序是否正确挂载了文件系统。

```
kubectl exec -ti fsx-app -- df -h
```

示例输出如下。

```
Filesystem                  Size  Used Avail Use% Mounted on
overlay                      80G  4.0G   77G   5% /
```

```
tmpfs                          64M     0   64M   0% /dev
tmpfs                         3.8G     0  3.8G   0% /sys/fs/cgroup
192.0.2.0@tcp:/abcdef01       1.1T  7.8M  1.1T   1% /data
/dev/nvme0n1p1                 80G  4.0G   77G   5% /etc/hosts
shm                            64M     0   64M   0% /dev/shm
tmpfs                         6.9G   12K  6.9G   1% /run/secrets/kubernetes.io/
serviceaccount
tmpfs                         3.8G     0  3.8G   0% /proc/acpi
tmpfs                         3.8G     0  3.8G   0% /sys/firmware
```

13验证示例应用程序已将数据写入 FSx for Lustre 文件系统。

```
kubectl exec -it fsx-app -- ls /data
```

示例输出如下。

```
out.txt
```

此示例输出显示示例应用程序成功编写了 `out.txt` 文件到文件系统。

> **Note**
>
> 删除集群之前，请务必删除 FSx for Lustre 文件系统。有关更多信息，请参阅《适用于 Lustre 的 FSx 用户指南》中的清理资源。

适用于 Lustre 的 FSx 的性能调整

在 Amazon EKS 中使用适用于 Lustre 的 FSx 时，您可以通过在节点初始化期间应用 Lustre 调整来优化性能。建议使用启动模板用户数据来确保所有节点的配置一致。

这些调优包括：

- 网络和 RPC 优化
- Lustre 模块管理
- LRU（锁定资源单位）调优：
- 客户端缓存控制设置
- OST 和 MDC 的 RPC 控件

有关实现这些性能调优的详细说明：

- 要优化标准节点（非 EFA）的性能，请参阅the section called “优化（非 EFA）”，了解可添加到启动模板用户数据的完整脚本。
- 要优化启用 EFA 的节点的性能，请参阅the section called “优化（EFA）”。

优化节点上的适用于 Lustre 的 Amazon FSx 性能（EFA）

本主题介绍如何使用 Amazon EKS 和适用于 Lustre 的 Amazon FSx 设置 Elastic Fabric Adapter（EFA）调优。

> **Note**
>
> - 有关创建和部署 FSx for Lustre CSI 驱动程序的信息，请参阅the section called “部署驱动程序”。
> - 要优化没有 EFA 的标准节点，请参阅the section called “优化（非 EFA）”。

步骤 1：创建 EKS 集群

使用提供的配置文件创建一个集群：

```
# Create cluster using efa-cluster.yaml
eksctl create cluster -f efa-cluster.yaml
```

示例：`efa-cluster.yaml`

```
#efa-cluster.yaml

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: csi-fsx
  region: us-east-1
  version: "1.30"

iam:
  withOIDC: true
```

```
availabilityZones: ["us-east-1a", "us-east-1d"]

managedNodeGroups:
  - name: my-efa-ng
    instanceType: c6gn.16xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    availabilityZones: ["us-east-1b"]
    volumeSize: 300
    privateNetworking: true
    amiFamily: Ubuntu2204
    efaEnabled: true
    preBootstrapCommands:
      - |
        #!/bin/bash
        eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print $1}')"
        efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]//g')
        min_efa_version="2.12.1"

        if [[ "$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version" ]]; then
            sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
            tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer
            echo "Installing EFA driver"
            sudo apt-get update && apt-get upgrade -y
            sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms
            sudo ./efa_installer.sh -y
            modinfo efa
        else
            echo "Using EFA driver version $efa_version"
        fi

        echo "Installing Lustre client"
        sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/fsx-
ubuntu-public-key.asc | gpg --dearmor | sudo tee /usr/share/keyrings/fsx-ubuntu-public-
key.gpg > /dev/null
        sudo echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
 https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" > /etc/apt/
sources.list.d/fsxlustreclientrepo.list
        sudo apt update | tail
```

```
        sudo apt install -y lustre-client-modules-$(uname -r) amazon-ec2-utils | tail
        modinfo lustre

        echo "Loading Lustre/EFA modules..."
        sudo /sbin/modprobe lnet
        sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
        sudo /sbin/modprobe ksocklnd
        sudo lnetctl lnet configure

        echo "Configuring TCP interface..."
        sudo lnetctl net del --net tcp 2> /dev/null
        sudo lnetctl net add --net tcp --if $eth_intf

        # For P5 instance type which supports 32 network cards,
        # by default add 8 EFA interfaces selecting every 4th device (1 per PCI bus)
        echo "Configuring EFA interface(s)..."
        instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
        num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
        echo "Found $num_efa_devices available EFA device(s)"

        if [[ "$instance_type" == "p5.48xlarge" || "$instance_type" ==
 "p5e.48xlarge" ]]; then
           for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
               sudo lnetctl net add --net efa --if $intf --peer-credits 32
          done
        else
        # Other instances: Configure 2 EFA interfaces by default if the instance
 supports multiple network cards,
        # or 1 interface for single network card instances
        # Can be modified to add more interfaces if instance type supports it
            sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband | head -
n1) --peer-credits 32
            if [[ $num_efa_devices -gt 1 ]]; then
               sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband | tail
 -n1) --peer-credits 32
            fi
        fi

        echo "Setting discovery and UDSP rule"
        sudo lnetctl set discovery 1
        sudo lnetctl udsp add --src efa --priority 0
        sudo /sbin/modprobe lustre

        sudo lnetctl net show
```

```
        echo "Added $(sudo lnetctl net show | grep -c '@efa') EFA interface(s)"
```

步骤 2：创建节点组

创建一个启用 EFA 的节点组：

```
# Create node group using efa-ng.yaml
eksctl create nodegroup -f efa-ng.yaml
```

> **Important**
>
> === 在 # 5. Mount FSx filesystem 部分中应根据环境调整这些值。

```
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
```

===

示例：efa-ng.yaml

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: final-efa
  region: us-east-1

managedNodeGroups:
  - name: ng-1
    instanceType: c6gn.16xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    availabilityZones: ["us-east-1a"]
    volumeSize: 300
    privateNetworking: true
    amiFamily: Ubuntu2204
    efaEnabled: true
    preBootstrapCommands:
```

```
      - |
        #!/bin/bash
        exec 1> >(logger -s -t $(basename $0)) 2>&1


################################################################################################
        #                                      Configuration Section
         #

################################################################################################

        # File System Configuration
        FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
        MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
        MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.

        # Lustre Tuning Parameters
        LUSTRE_LRU_MAX_AGE=600000
        LUSTRE_MAX_CACHED_MB=64
        LUSTRE_OST_MAX_RPC=32
        LUSTRE_MDC_MAX_RPC=64
        LUSTRE_MDC_MOD_RPC=50

        # File paths
        FUNCTIONS_SCRIPT="/usr/local/bin/lustre_functions.sh"
        TUNINGS_SCRIPT="/usr/local/bin/apply_lustre_tunings.sh"
        SERVICE_FILE="/etc/systemd/system/lustre-tunings.service"

        #EFA
        MIN_EFA_VERSION="2.12.1"

        # Function to check if a command was successful
        check_success() {
            if [ $? -eq 0 ]; then
                echo "SUCCESS: $1"
            else
                echo "FAILED: $1"
                return 1
            fi
        }

        echo "********Starting FSx for Lustre configuration********"

        # 1. Install Lustre client
```

```
        if grep -q '^ID=ubuntu' /etc/os-release; then
            echo "Detected Ubuntu, proceeding with Lustre setup..."
            # Add Lustre repository
            sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/
fsx-ubuntu-public-key.asc | sudo gpg --dearmor | sudo tee /usr/share/keyrings/fsx-
ubuntu-public-key.gpg > /dev/null

            echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
 https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" | sudo tee /etc/
apt/sources.list.d/fsxlustreclientrepo.list

            sudo apt-get update
            sudo apt-get install -y lustre-client-modules-$(uname -r)
            sudo apt-get install -y lustre-client
        else
            echo "Not Ubuntu, exiting"
            exit 1
        fi

        check_success "Install Lustre client"

        # Ensure Lustre tools are in the PATH
        export PATH=$PATH:/usr/sbin

        # 2. Apply network and RPC tunings
        echo "********Applying network and RPC tunings********"
        if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/
modprobe.conf; then
            echo "options ptlrpc ptlrpcd_per_cpt_max=64" | sudo tee -a /etc/modprobe.d/
modprobe.conf
            check_success "Set ptlrpcd_per_cpt_max"
        else
            echo "ptlrpcd_per_cpt_max already set in modprobe.conf"
        fi

        if ! grep -q "options ksocklnd credits" /etc/modprobe.d/modprobe.conf; then
            echo "options ksocklnd credits=2560" | sudo tee -a /etc/modprobe.d/
modprobe.conf
            check_success "Set ksocklnd credits"
        else
            echo "ksocklnd credits already set in modprobe.conf"
        fi

        # 3. Load Lustre modules
```

```
        manage_lustre_modules() {
            echo "Checking for existing Lustre modules..."
            if lsmod | grep -q lustre; then
                echo "Existing Lustre modules found."

                # Check for mounted Lustre filesystems
                echo "Checking for mounted Lustre filesystems..."
                if mount | grep -q "type lustre"; then
                    echo "Found mounted Lustre filesystems. Attempting to unmount..."
                    mounted_fs=$(mount | grep "type lustre" | awk '{print $3}')
                    for fs in $mounted_fs; do
                        echo "Unmounting $fs"
                        sudo umount $fs
                        check_success "Unmount filesystem $fs"
                    done
                else
                    echo "No Lustre filesystems mounted."
                fi

                # After unmounting, try to remove modules
                echo "Attempting to remove Lustre modules..."
                sudo lustre_rmmod
                if [ $? -eq 0 ]; then
                    echo "SUCCESS: Removed existing Lustre modules"
                else
                    echo "WARNING: Could not remove Lustre modules. They may still be
in use."
                    echo "Please check for any remaining Lustre processes or mounts."
                    return 1
                fi
            else
                echo "No existing Lustre modules found."
            fi

            echo "Loading Lustre modules..."
            sudo modprobe lustre
            check_success "Load Lustre modules" || exit 1

            echo "Checking loaded Lustre modules:"
            lsmod | grep lustre
        }

        # Managing Lustre kernel modules
        echo "********Managing Lustre kernel modules********"
```

```
        manage_lustre_modules

        # 4. Initializing Lustre networking
        echo "********Initializing Lustre networking********"
        sudo lctl network up
        check_success "Initialize Lustre networking" || exit 1

        # 4.5 EFA Setup and Configuration
        setup_efa() {
            echo "********Starting EFA Setup********"

            # Get interface and version information
            eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print
 $1}')"
            efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]//
g')
            min_efa_version=$MIN_EFA_VERSION

            # Install or verify EFA driver
            if [[ "$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version" ]]; then
                echo "Installing EFA driver..."
                sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
                tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer

                # Install dependencies
                sudo apt-get update && apt-get upgrade -y
                sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms

                # Install EFA
                sudo ./efa_installer.sh -y
                modinfo efa
            else
                echo "Using existing EFA driver version $efa_version"
            fi
        }

        configure_efa_network() {
            echo "********Configuring EFA Network********"

            # Load required modules
            echo "Loading network modules..."
```

```
            sudo /sbin/modprobe lnet
            sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
            sudo /sbin/modprobe ksocklnd

            # Initialize LNet
            echo "Initializing LNet..."
            sudo lnetctl lnet configure

            # Configure TCP interface
            echo "Configuring TCP interface..."
            sudo lnetctl net del --net tcp 2> /dev/null
            sudo lnetctl net add --net tcp --if $eth_intf

            # For P5 instance type which supports 32 network cards,
            # by default add 8 EFA interfaces selecting every 4th device (1 per PCI
bus)
            echo "Configuring EFA interface(s)..."
            instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
            num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
            echo "Found $num_efa_devices available EFA device(s)"

            if [[ "$instance_type" == "p5.48xlarge" || "$instance_type" ==
"p5e.48xlarge" ]]; then
                # P5 instance configuration
                for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
                    sudo lnetctl net add --net efa --if $intf --peer-credits 32
                done
            else
                # Standard configuration
                # Other instances: Configure 2 EFA interfaces by default if the
instance supports multiple network cards,
                # or 1 interface for single network card instances
                # Can be modified to add more interfaces if instance type supports it
                sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband |
head -n1) --peer-credits 32
                if [[ $num_efa_devices -gt 1 ]]; then
                    sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband |
tail -n1) --peer-credits 32
                fi
            fi

            # Configure discovery and UDSP
            echo "Setting up discovery and UDSP rules..."
            sudo lnetctl set discovery 1
```

```
            sudo lnetctl udsp add --src efa --priority 0
            sudo /sbin/modprobe lustre

            # Verify configuration
            echo "Verifying EFA network configuration..."
            sudo lnetctl net show
            echo "Added $(sudo lnetctl net show | grep -c '@efa') EFA interface(s)"
        }

        # Main execution
        setup_efa
        configure_efa_network

        # 5. Mount FSx filesystem
        if [ ! -z "$FSX_DNS" ] && [ ! -z "$MOUNT_NAME" ]; then
            echo "********Creating mount point********"
            sudo mkdir -p $MOUNT_POINT
            check_success "Create mount point"

            echo "Mounting FSx filesystem..."
            sudo mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
            check_success "Mount FSx filesystem"
        else
            echo "Skipping FSx mount as DNS or mount name is not provided"
        fi

        # 6. Applying Lustre performance tunings
        echo "********Applying Lustre performance tunings********"

        # Get number of CPUs
        NUM_CPUS=$(nproc)

        # Calculate LRU size (100 * number of CPUs)
        LRU_SIZE=$((100 * NUM_CPUS))

        #Apply LRU tunings
        echo "Apply LRU tunings"
        sudo lctl set_param ldlm.namespaces.*.lru_max_age=${LUSTRE_LRU_MAX_AGE}
        check_success "Set lru_max_age"
        sudo lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
        check_success "Set lru_size"

        # Client Cache Control
        sudo lctl set_param llite.*.max_cached_mb=${LUSTRE_MAX_CACHED_MB}
```

```
        check_success "Set max_cached_mb"

        # RPC Controls
        sudo lctl set_param osc.*OST*.max_rpcs_in_flight=${LUSTRE_OST_MAX_RPC}
        check_success "Set OST max_rpcs_in_flight"

        sudo lctl set_param mdc.*.max_rpcs_in_flight=${LUSTRE_MDC_MAX_RPC}
        check_success "Set MDC max_rpcs_in_flight"

        sudo lctl set_param mdc.*.max_mod_rpcs_in_flight=${LUSTRE_MDC_MOD_RPC}
        check_success "Set MDC max_mod_rpcs_in_flight"

        # 7. Verify all tunings
        echo "********Verifying all tunings********"

        # Function to verify parameter value
        verify_param() {
            local param=$1
            local expected=$2
            local actual=$3

            if [ "$actual" == "$expected" ]; then
                echo "SUCCESS: $param is correctly set to $expected"
            else
                echo "WARNING: $param is set to $actual (expected $expected)"
            fi
        }

        echo "Verifying all parameters:"

        # LRU tunings
        actual_lru_max_age=$(lctl get_param -n ldlm.namespaces.*.lru_max_age | head -1)
        verify_param "lru_max_age" "600000" "$actual_lru_max_age"

        actual_lru_size=$(lctl get_param -n ldlm.namespaces.*.lru_size | head -1)
        verify_param "lru_size" "$LRU_SIZE" "$actual_lru_size"

        # Client Cache
        actual_max_cached_mb=$(lctl get_param -n llite.*.max_cached_mb | grep
"max_cached_mb:" | awk '{print $2}')
        verify_param "max_cached_mb" "64" "$actual_max_cached_mb"

        # RPC Controls
        actual_ost_rpcs=$(lctl get_param -n osc.*OST*.max_rpcs_in_flight | head -1)
```

```
        verify_param "OST max_rpcs_in_flight" "32" "$actual_ost_rpcs"

        actual_mdc_rpcs=$(lctl get_param -n mdc.*.max_rpcs_in_flight | head -1)
        verify_param "MDC max_rpcs_in_flight" "64" "$actual_mdc_rpcs"

        actual_mdc_mod_rpcs=$(lctl get_param -n mdc.*.max_mod_rpcs_in_flight | head -1)
        verify_param "MDC max_mod_rpcs_in_flight" "50" "$actual_mdc_mod_rpcs"

        # Network and RPC configurations from modprobe.conf
        actual_ptlrpc=$(grep "ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf
| awk '{print $3}')
        verify_param "ptlrpcd_per_cpt_max" "ptlrpcd_per_cpt_max=64" "$actual_ptlrpc"

        actual_ksocklnd=$(grep "ksocklnd credits" /etc/modprobe.d/modprobe.conf | awk
'{print $3}')
        verify_param "ksocklnd credits" "credits=2560" "$actual_ksocklnd"

        # 8. Setup persistence
        setup_persistence() {
            # Create functions file
            cat << EOF > $FUNCTIONS_SCRIPT
        #!/bin/bash

        apply_lustre_tunings() {
            local NUM_CPUS=\$(nproc)
            local LRU_SIZE=\$((100 * NUM_CPUS))

            echo "Applying Lustre performance tunings..."
            lctl set_param ldlm.namespaces.*.lru_max_age=$LUSTRE_LRU_MAX_AGE
            lctl set_param ldlm.namespaces.*.lru_size=\$LRU_SIZE
            lctl set_param llite.*.max_cached_mb=$LUSTRE_MAX_CACHED_MB
            lctl set_param osc.*OST*.max_rpcs_in_flight=$LUSTRE_OST_MAX_RPC
            lctl set_param mdc.*.max_rpcs_in_flight=$LUSTRE_MDC_MAX_RPC
            lctl set_param mdc.*.max_mod_rpcs_in_flight=$LUSTRE_MDC_MOD_RPC
        }
        EOF

            # Create tuning script
            cat << EOF > $TUNINGS_SCRIPT
        #!/bin/bash
        exec 1> >(logger -s -t \$(basename \$0)) 2>&1

        source $FUNCTIONS_SCRIPT
```

```
        # Function to check if Lustre is mounted
        is_lustre_mounted() {
            mount | grep -q "type lustre"
        }

        # Function to mount Lustre
        mount_lustre() {
            echo "Mounting Lustre filesystem..."
            mkdir -p $MOUNT_POINT
            mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} $MOUNT_POINT
            return \$?
        }

        # Main execution
        # Try to mount if not already mounted
        if ! is_lustre_mounted; then
            echo "Lustre filesystem not mounted, attempting to mount..."
            mount_lustre
        fi

        # Wait for successful mount (up to 5 minutes)
        for i in {1..30}; do
            if is_lustre_mounted; then
                echo "Lustre filesystem mounted, applying tunings..."
                apply_lustre_tunings
                exit 0
            fi
            echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
            sleep 10
        done

        echo "Timeout waiting for Lustre filesystem mount"
        exit 1
        EOF

        # Create systemd service

        # Create systemd directory if it doesn't exist
        sudo mkdir -p /etc/systemd/system/

            # Create service file directly for Ubuntu
            cat << EOF > $SERVICE_FILE
        [Unit]
        Description=Apply Lustre Performance Tunings
```

```
        After=network.target remote-fs.target

        [Service]
        Type=oneshot
        ExecStart=/bin/bash -c 'source $FUNCTIONS_SCRIPT && $TUNINGS_SCRIPT'
        RemainAfterExit=yes

        [Install]
        WantedBy=multi-user.target
        EOF


            # Make scripts executable and enable service
            sudo chmod +x $FUNCTIONS_SCRIPT
            sudo chmod +x $TUNINGS_SCRIPT
            systemctl enable lustre-tunings.service
            systemctl start lustre-tunings.service
        }

        echo "********Setting up persistent tuning********"
        setup_persistence

        echo "FSx for Lustre configuration completed."
```

（可选）第 3 步。验证 EFA 设置

通过 SSH 连接到节点：

```
# Get instance ID from EKS console or {aws} CLI
ssh -i /path/to/your-key.pem ec2-user@<node-internal-ip>
```

验证 EFA 配置：

```
sudo lnetctl net show
```

检查设置日志：

```
sudo cat /var/log/cloud-init-output.log
```

`lnetctl net show` 的预期输出示例如下：

```
net:
```

```
    - net type: tcp
      ...
    - net type: efa
      local NI(s):
        - nid: xxx.xxx.xxx.xxx@efa
          status: up
```

示例部署

a. 创建 claim.yaml

```
#claim.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim-efa
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 4800Gi
  volumeName: fsx-pv
```

应用声明：

```
kubectl apply -f claim.yaml
```

b. 创建 pv.yaml

更新 <replaceable-placeholders>：

```
#pv.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  name: fsx-pv
spec:
```

```
  capacity:
    storage: 4800Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  mountOptions:
    - flock
  persistentVolumeReclaimPolicy: Recycle
  csi:
    driver: fsx.csi.aws.com
    volumeHandle: fs-<1234567890abcdef0>
    volumeAttributes:
      dnsname: fs-<1234567890abcdef0>.fsx.us-east-1.amazonaws.com
      mountname: <abcdef01>
```

应用持久性卷：

```
kubectl apply -f pv.yaml
```

c. 创建 pod.yaml

```
#pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: fsx-efa-app
spec:
  containers:
  - name: app
    image: amazonlinux:2
    command: ["/bin/sh"]
    args: ["-c", "while true; do dd if=/dev/urandom bs=100M count=20 > data/test_file;
 sleep 10; done"]
    resources:
      requests:
        vpc.amazonaws.com/efa: 1
      limits:
        vpc.amazonaws.com/efa: 1
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
```

```
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: fsx-claim-efa
```

应用容器组（pod）：

```
kubectl apply -f pod.yaml
```

其他验证命令

验证容器组（pod）是否挂载到并写入文件系统：

```
kubectl exec -ti fsx-efa-app -- df -h | grep data
# Expected output:
# <192.0.2.0>@tcp:/<abcdef01>  4.5T  1.2G  4.5T   1% /data

kubectl exec -ti fsx-efa-app -- ls /data
# Expected output:
# test_file
```

通过 SSH 连接到节点来验证流量是否通过 EFA：

```
sudo lnetctl net show -v
```

预期输出将显示 EFA 接口及流量统计数据。

相关信息

- the section called “部署驱动程序”
- the section called “优化（非 EFA）”
- 适用于 Lustre 的 Amazon FSx 性能
- Elastic Fabric Adapter

优化节点上的适用于 Lustre 的 Amazon FSx 性能（非 EFA）

您可以在节点初始化期间使用启动模板用户数据应用调优参数，从而优化适用于 Lustre 的 Amazon FSx 性能。

> **Note**
>
> - 有关创建和部署 FSx for Lustre CSI 驱动程序的信息，请参阅the section called “部署驱动程序”。要优化启用 EFA 的节点的性能，请参阅the section called “优化（EFA）”。

为什么要使用启动模板用户数据？

- 在节点初始化期间自动应用调优。
- 可确保所有节点的配置一致。
- 无需手动配置节点。

示例脚本概览

本主题中定义的示例脚本会执行以下操作：

 # 1. Install Lustre client

- 自动检测 Amazon Linux（AL）操作系统版本。
- 安装相应的 Lustre 客户端程序包。

 # 2. Apply network and RPC tunings

- 设置 `ptlrpcd_per_cpt_max=64` 以实现并行 RPC 处理。
- 配置 `ksocklnd credits=2560` 以优化网络缓冲区。

 # 3. Load Lustre modules

- 安全地移除现有 Lustre 模块（如有）。
- 处理现有文件系统的卸载。
- 加载全新的 Lustre 模块。

 # 4. Lustre Network Initialization

- 初始化 Lustre 联网配置。
- 设置所需的网络参数。

 # 5. Mount FSx filesystem

- 必须在这一部分中根据环境调整值。

 # 6. Apply tunings

- LRU（锁定资源单位）调优：
  - `lru_max_age=600000`
  - 根据 CPU 计数计算的 `lru_size`
- 客户端缓存控制：`max_cached_mb=64`
- RPC 控制：
  - OST `max_rpcs_in_flight=32`
  - MDC `max_rpcs_in_flight=64`
  - MDC `max_mod_rpcs_in_flight=50`

 # 7. Verify tunings

- 验证所有已应用的调优。
- 报告每个参数成功或发出警告。

 # 8. Setup persistence

- 在这一部分中还必须根据环境调整值。
- 自动检测操作系统版本（AL2023），确定要应用的 Systemd 服务。
- 系统启动。
- Systemd 会启动 `lustre-tunings` 服务（由于 `WantedBy=multi-user.target`）。
- 服务会运行 `apply_lustre_tunings.sh`，这将：
  - 检查文件系统是否已挂载。
  - 如果未挂载，则将挂载文件系统。
  - 等待挂载成功（最长五分钟）。
  - 成功挂载后应用调优参数。
- 在重新启动之前，设置一直处于活动状态。
- 服务会在脚本完成后退出。

  - Systemd 将服务标记为“活动（已退出）”。
- 下次重启时会重复此过程。

创建启动模板

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 选择启动模板。
3. 选择Create launch template（创建启动模板）。
4. 在高级详细信息中，找到用户数据部分。
5. 粘贴下面的脚本，并根据需要更新任何内容。

> **Important**
>
> 在 `# 5. Mount FSx filesystem` 部分中以及在 `# 8. Setup persistence` 部分 `apply_lustre_tunings.sh` 步骤的 `setup_persistence()` 函数中，根据环境调整这些值：
>
> ```
> FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
> MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
> MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
> ```

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="
--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
# Function definitions
check_success() {
    if [ $? -eq 0 ]; then
        echo "SUCCESS: $1"
    else
        echo "FAILED: $1"
        return 1
    fi
}
apply_tunings() {
```

```
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))
    local params=(
        "ldlm.namespaces.*.lru_max_age=600000"
        "ldlm.namespaces.*.lru_size=$LRU_SIZE"
        "llite.*.max_cached_mb=64"
        "osc.*OST*.max_rpcs_in_flight=32"
        "mdc.*.max_rpcs_in_flight=64"
        "mdc.*.max_mod_rpcs_in_flight=50"
    )
    for param in "${params[@]}"; do
        lctl set_param $param
        check_success "Set ${param%%=*}"
    done
}
verify_param() {
    local param=$1
    local expected=$2
    local actual=$3

    if [ "$actual" == "$expected" ]; then
        echo "SUCCESS: $param is correctly set to $expected"
    else
        echo "WARNING: $param is set to $actual (expected $expected)"
    fi
}
verify_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))
    local params=(
        "ldlm.namespaces.*.lru_max_age:600000"
        "ldlm.namespaces.*.lru_size:$LRU_SIZE"
        "llite.*.max_cached_mb:64"
        "osc.*OST*.max_rpcs_in_flight:32"
        "mdc.*.max_rpcs_in_flight:64"
        "mdc.*.max_mod_rpcs_in_flight:50"
    )
    echo "Verifying all parameters:"
    for param in "${params[@]}"; do
        name="${param%%:*}"
        expected="${param#*:}"
        actual=$(lctl get_param -n $name | head -1)
        verify_param "${name##*.}" "$expected" "$actual"
    done
```

```
}
setup_persistence() {
    # Create functions file
    cat << 'EOF' > /usr/local/bin/lustre_functions.sh
#!/bin/bash
apply_lustre_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))

    echo "Applying Lustre performance tunings..."
    lctl set_param ldlm.namespaces.*.lru_max_age=600000
    lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
    lctl set_param llite.*.max_cached_mb=64
    lctl set_param osc.*OST*.max_rpcs_in_flight=32
    lctl set_param mdc.*.max_rpcs_in_flight=64
    lctl set_param mdc.*.max_mod_rpcs_in_flight=50
}
EOF
    # Create tuning script
    cat << 'EOF' > /usr/local/bin/apply_lustre_tunings.sh
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
# Source the functions
source /usr/local/bin/lustre_functions.sh
# FSx details
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
# Function to check if Lustre is mounted
is_lustre_mounted() {
    mount | grep -q "type lustre"
}
# Function to mount Lustre
mount_lustre() {
    echo "Mounting Lustre filesystem..."
    mkdir -p ${MOUNT_POINT}
    mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
    return $?
}
# Main execution
# Try to mount if not already mounted
if ! is_lustre_mounted; then
    echo "Lustre filesystem not mounted, attempting to mount..."
    mount_lustre
```

```
fi
# Wait for successful mount (up to 5 minutes)
for i in {1..30}; do
    if is_lustre_mounted; then
        echo "Lustre filesystem mounted, applying tunings..."
        apply_lustre_tunings
        exit 0
    fi
    echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
    sleep 10
done
echo "Timeout waiting for Lustre filesystem mount"
exit 1
EOF
    # Create systemd service
    cat << 'EOF' > /etc/systemd/system/lustre-tunings.service
[Unit]
Description=Apply Lustre Performance Tunings
After=network.target remote-fs.target
StartLimitIntervalSec=0
[Service]
Type=oneshot
ExecStart=/usr/local/bin/apply_lustre_tunings.sh
RemainAfterExit=yes
Restart=on-failure
RestartSec=30
[Install]
WantedBy=multi-user.target
EOF
    chmod +x /usr/local/bin/lustre_functions.sh
    chmod +x /usr/local/bin/apply_lustre_tunings.sh
    systemctl enable lustre-tunings.service
    systemctl start lustre-tunings.service
}
echo "Starting FSx for Lustre configuration..."
# 1. Install Lustre client
if grep -q 'VERSION="2"' /etc/os-release; then
    amazon-linux-extras install -y lustre
elif grep -q 'VERSION="2023"' /etc/os-release; then
    dnf install -y lustre-client
fi
check_success "Install Lustre client"
# 2. Apply network and RPC tunings
export PATH=$PATH:/usr/sbin
```

```
echo "Applying network and RPC tunings..."
if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf; then
    echo "options ptlrpc ptlrpcd_per_cpt_max=64" | tee -a /etc/modprobe.d/
modprobe.conf
    echo "options ksocklnd credits=2560" | tee -a /etc/modprobe.d/modprobe.conf
fi
# 3. Load Lustre modules
modprobe lustre
check_success "Load Lustre modules" || exit 1
# 4. Lustre Network Initialization
lctl network up
check_success "Initialize Lustre networking" || exit 1
# 5. Mount FSx filesystem
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
if [ ! -z "$FSX_DNS" ] && [ ! -z "$MOUNT_NAME" ]; then
    mkdir -p $MOUNT_POINT
    mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
    check_success "Mount FSx filesystem"
fi
# 6. Apply tunings
apply_tunings
# 7. Verify tunings
verify_tunings
# 8. Setup persistence
setup_persistence
echo "FSx for Lustre configuration completed."
--==MYBOUNDARY==--
```

6. 创建 Amazon EKS 节点组时，请选择此启动模板。有关更多信息，请参阅 the section called “创建”。

相关信息

- the section called “部署驱动程序”
- the section called “优化（EFA）”
- 适用于 Lustre 的 Amazon FSx 性能

将适用于 NetApp ONTAP 的 FSx 与高性能应用程序存储结合使用

NetApp Trident 使用符合容器存储接口（CSI）的驱动程序提供动态存储编排。该服务允许 Amazon EKS 集群管理适用于 NetApp ONTAP 的 Amazon FSx 文件系统提供支持的持久卷（PV）的生命周期。请注意，适用于 NetApp ONTAP 的 Amazon FSx CSI 驱动程序与 Amazon EKS 混合节点功能不兼容。要开始使用，请参阅 NetApp Trident 文档中的 Use Trident with Amazon FSx for NetApp ONTAP。

Amazon FSx for NetApp ONTAP 是一种存储服务，允许您在云中启动和运行完全托管的 ONTAP 文件系统。ONTAP 是 NetApp 的文件系统技术，提供了一套采用广泛的数据访问和数据管理功能。FSx for ONTAP 提供本地 NetApp 文件系统的功能、性能和 API，具有完全托管式 Amazon 服务的灵活性、可扩展性和简单性。有关更多信息，请参阅《FSx for ONTAP 用户指南》。

Important

如果您将适用于 NetApp ONTAP 的 Amazon FSx 和 Amazon EBS CSI 驱动程序结合使用来预置 EBS 卷，则必须在 `multipath.conf` 文件中指定不要使用 EBS 设备。有关支持的方法，请参阅配置文件黑名单。见下列。

```
defaults {
       user_friendly_names yes
       find_multipaths no
     }
     blacklist {
       device {
         vendor "NVME"
         product "Amazon Elastic Block Store"
       }
     }
```

将适用于 OpenZFS 的 Amazon FSx 与数据存储结合使用

Amazon FSx for OpenZFS 是一项完全托管式的文件存储服务，可让您轻松地将数据从本地 ZFS 或其他基于 Linux 的文件服务器移动到 Amazon。您可以在不更改应用程序代码或数据管理方式的情况下执行此操作。它基于开源 OpenZFS 文件系统提供高度可靠、可扩展、高效且功能丰富的文件存储。它将这些功能与完全托管式 Amazon 服务的灵活性、可扩展性和简单性相结合。有关更多信息，请参阅《Amazon FSx for OpenZFS 用户指南》https://docs.amazonaws.cn/fsx/latest/OpenZFSGuide/what-is-fsx.html。

适用于 OpenZFS 的 FSx 容器存储接口（CSI）驱动程序提供了一个 CSI 接口，从而方便 Amazon EKS 集群管理适用于 OpenZFS 的 FSx 卷的生命周期。请注意，适用于 OpenZFS 的 Amazon FSx CSI 驱动程序与 Amazon EKS 混合节点功能不兼容。要将适用于 OpenZFS 的 FSx CSI 驱动程序部署到 Amazon EKS 集群，请参阅 GitHub 上的 aws-fsx-openzfs-csi-driver。

使用 Amazon File Cache 减少延迟

Amazon File Cache 是 Amazon 上一个完全托管的高速缓存，用于处理文件数据，无论数据存储在何处。Amazon File Cache 会在首次访问数据时自动将数据加载到缓存中，并在不使用时释放数据。有关更多信息，请参阅《Amazon File Cache 用户指南》。

Amazon File Cache Storage Interface (CSI) 驱动程序提供了一个 CSI 接口，允许 Amazon EKS 集群管理 Amazon 文件缓存的生命周期。请注意，Amazon File Cache CSI 驱动程序与 Amazon EKS 混合节点功能不兼容。要将 Amazon File Cache CSI 驱动程序部署到您的 Amazon EKS 集群，请参阅 GitHub 上的 aws-file-cache-csi-driver。

使用适用于 Amazon S3 的 Mountpoint CSI 驱动程序访问适用于 Amazon S3 对象

借助适用于 Amazon S3 的 Mountpoint 容器存储接口（CSI）驱动程序，您的 Kubernetes 应用程序可以通过文件系统接口访问 Amazon S3 对象，从而在不更改任何应用程序代码的情况下实现高聚合吞吐量。基于适用于 Amazon S3 的 Mountpoint 构建的 CSI 驱动程序，将 Amazon S3 存储桶以卷的形式呈现，可供 Amazon EKS 和自主管理型 Kubernetes 集群中的容器访问。

注意事项

- 适用于 Amazon S3 的 Mountpoint CSI 驱动程序目前与基于 Windows 的容器映像不兼容。
- 适用于 Amazon S3 的 Mountpoint CSI 驱动程序目前与 Amazon EKS 混合节点功能不兼容。
- 适用于 Amazon S3 的 Mountpoint CSI 驱动程序不支持 Amazon Fargate，但支持在 Amazon EC2 中运行的容器（无论是采用 Amazon EKS 还是自定义 Kubernetes 安装）。
- 适用于 Amazon S3 的 Mountpoint CSI 驱动程序仅支持静态预置。不支持动态预置或创建新存储桶。

> **Note**
>
> 静态预调配是指使用现有的 Amazon S3 存储桶，该存储桶被指定为 `PersistentVolume` 对象中 `volumeAttributes` 中的 `bucketName`。有关更多信息，请参阅 GitHub 上的 Static Provisioning。

- 使用适用于 Amazon S3 的 Mountpoint CSI 驱动程序挂载的卷，不支持所有 POSIX 文件系统功能。有关文件系统行为的详细信息，请参阅 GitHub 上的 Mountpoint for Amazon S3 file system behavior。

有关部署此驱动程序的详细信息，请参阅the section called “部署驱动程序”。有关移除此驱动程序的详细信息，请参阅the section called “移除驱动程序”。

部署适用于 Amazon S3 的 Mountpoint 驱动程序

借助适用于 Amazon S3 的 Mountpoint 容器存储接口（CSI）驱动程序，您的 Kubernetes 应用程序可以通过文件系统接口访问 Amazon S3 对象，从而在不更改任何应用程序代码的情况下实现高聚合吞吐量。

此过程将演示如何部署适用于 Amazon S3 的 Mountpoint CSI Amazon EKS 驱动程序。在继续操作之前，请查看注意事项。

先决条件

- 适用于集群的现有 Amazon Identity and Access Management（IAM）OpenID Connect（OIDC）提供者。要确定您是否已经拥有一个或是否要创建一个，请参阅the section called “IAM OIDC 提供者”。
- 您的设备或 Amazon CloudShell 上安装并配置了 2.12.3 或更高版本的 Amazon CLI。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。

步骤 1：创建 IAM policy

适用于 Amazon S3 的 Mountpoint CSI 驱动程序需要 Amazon S3 权限才能与文件系统交互。本节将介绍如何创建 IAM 策略来授予必要的权限。

以下示例策略遵循 Mountpoint 的 IAM 权限建议。您也可以使用 Amazon 托管式策略 AmazonS3FullAccess，但此托管式策略授予的权限超出了 Mountpoint 所需的权限。

有关 Mountpoint 建议权限的更多信息，请参阅 GitHub 上的 Mountpoint IAM permissions。

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 在策略页面上，选择创建策略。
4. 对于策略编辑器，选择 JSON。
5. 在策略编辑器下，复制并粘贴以下内容：

> **Important**
>
> 将 amzn-s3-demo-bucket1 替换为您自己的 Amazon S3 存储桶名称。

```
{
   "Version":"2012-10-17",
   "Statement": [
        {
            "Sid": "MountpointFullBucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1"
            ]
        },
        {
            "Sid": "MountpointFullObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1/*"
            ]
```

```
        }
    ]
}
```

与 Amazon S3 Express One Zone 存储类一起推出的目录存储桶，使用不同于一般用途存储桶的身份验证机制。您应该使用 `s3express:CreateSession` 操作，而非使用 `s3:*` 操作。有关目录存储桶的更多信息，请参阅《Amazon S3 用户指南》中的目录存储桶。

以下是您将对目录存储桶使用的最低权限策略的示例。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3express:CreateSession",
            "Resource": "arn:aws:s3express:us-west-2:111122223333:bucket/amzn-s3-
demo-bucket1--usw2-az1--x-s3"
        }
    ]
}
```

6. 选择下一步。
7. 在查看和创建页面上，为您的策略命名。此示例演练使用名称 `AmazonS3CSIDriverPolicy`。
8. 选择创建策略。

步骤 2：创建 IAM 角色

适用于 Amazon S3 的 Mountpoint CSI 驱动程序需要 Amazon S3 权限才能与文件系统交互。本节将介绍如何创建 IAM 角色来委派这些权限。要创建此角色，您可以使用以下工具之一：

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”
- the section called “Amazon CLI”

> **Note**
>
> 在上一节中创建了 IAM 策略 `AmazonS3CSIDriverPolicy`。

eksctl

使用 **`eksctl`** 创建适用于 Amazon S3 的 Mountpoint CSI 驱动程序 IAM 角色

要创建 IAM 角色和 Kubernetes 服务账户，请运行以下命令。这些命令还会将 `AmazonS3CSIDriverPolicy` IAM 策略附加到角色，使用 IAM 角色的 Amazon 资源名称（ARN）对 Kubernetes 服务账户 (`s3-csi-controller-sa`) 进行注释，并将 Kubernetes 服务账户名称添加到 IAM 角色的信任策略。

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
    --name s3-csi-driver-sa \
    --namespace kube-system \
    --cluster $CLUSTER_NAME \
    --attach-policy-arn $POLICY_ARN \
    --approve \
    --role-name $ROLE_NAME \
    --region $REGION \
    --role-only
```

Amazon Web Services 管理控制台

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 Roles（角色）。
3. 在 Roles（角色）页面上，选择 Create role（创建角色）。
4. 在 Select trusted entity（选择受信任的实体）页面上，请执行以下操作：
   a. 在 Trusted entity type（受信任的实体类型）部分中，选择 Web identity（Web 身份）。
   b. 对于 Identity provider（身份提供商），为您的集群选择 OpenID Connect provider URL（OpenID Connect 提供程序 URL）（如 Amazon EKS 中的 Overview（概述）下所示）。

      如果未显示 URL，请查看先决条件部分。
   c. 对于 Audience (受众)，请选择 `sts.amazonaws.com`。
   d. 选择下一步。
5. 在 Add permissions（添加权限）页面上，请执行以下操作：
   a. 在筛选策略框中，输入 AmazonS3CSIDriverPolicy。

> **Note**
>
> 在上一节中创建了此策略。

b. 选中搜索返回的 AmazonS3CSIDriverPolicy 结果左侧的复选框。

c. 选择下一步。

6. 在 Name, review, and create（命名、查看和创建）页面中，请执行以下操作：

a. 在角色名称中为角色输入唯一名称，例如 AmazonEKS_S3_CSI_DriverRole。

b. 在添加标签（可选）下，将标签作为键值对附加，以将元数据添加到角色。有关在 IAM 中使用标签的更多信息，请参阅《IAM 用户指南》 中的标记 IAM 资源。

c. 选择创建角色。

7. 创建角色后，在控制台中选择角色以将其打开进行编辑。

8. 选择 Trust relationships（信任关系）选项卡，然后选择 Edit trust policy（编辑信任策略）。

9. 该行看起来类似于以下行：

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

在上一行末尾添加逗号，然后在逗号后添加下一行。将 *region-code* 替换为您的集群所在的 Amazon 区域。将 *EXAMPLED539D4633E53DE1B71EXAMPLE* 替换为您的集群的 OIDC 提供者 ID。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:s3-csi-driver-sa"
```

10确保将 Condition 运算符设置为 "StringEquals"。

11选择 Update policy（更新策略）以完成操作。

Amazon CLI

1. 查看集群的 OIDC 提供者 URL。将 *my-cluster* 替换为您的集群的名称。如果命令的输出为 None，请查看先决条件。

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

示例输出如下。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. 创建 IAM 角色，向 Kubernetes 服务账户授予 AssumeRoleWithWebIdentity 操作权限。

   a. 将以下内容复制到名为 aws-s3-csi-driver-trust-policy.json 的文件中。请将 *111122223333* 替换为您的账户 ID。将 *EXAMPLED539D4633E53DE1B71EXAMPLE* 和 *region-code* 替换为上一步中返回的值。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:s3-csi-
driver-sa",
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

   b. 创建角色。您可以将 *AmazonEKS_S3_CSI_DriverRole* 更改为其它名称，但如果更改，请确保在后续步骤中也做出相应更改。

```
aws iam create-role \
  --role-name AmazonEKS_S3_CSI_DriverRole \
  --assume-role-policy-document file://"aws-s3-csi-driver-trust-policy.json"
```

3. 使用以下命令将之前创建的 IAM 策略附加到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonS3CSIDriverPolicy \
  --role-name AmazonEKS_S3_CSI_DriverRole
```

> **Note**
>
> 在上一节中创建了 IAM 策略 AmazonS3CSIDriverPolicy。

4. 如果您将驱动程序作为 Amazon EKS 附加组件安装，则跳过此步骤。对于驱动程序自主管理型安装，请创建 Kubernetes 服务账户并使用创建的 IAM 角色的 ARN 进行注释。

   a. 将以下内容保存到名为 mountpoint-s3-service-account.yaml 的文件中。请将 *111122223333* 替换为您的账户 ID。

   ```
   ---
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     labels:
       app.kubernetes.io/name: aws-mountpoint-s3-csi-driver
     name: mountpoint-s3-csi-controller-sa
     namespace: kube-system
     annotations:
       eks.amazonaws.com/role-arn: arn:aws-cn:iam::111122223333:role/
   AmazonEKS_S3_CSI_DriverRole
   ```

   b. 在集群上创建 Kubernetes 服务账户。使用您创建的名为 *AmazonEKS_S3_CSI_DriverRole* 的 IAM 角色对 Kubernetes 服务账户 (mountpoint-s3-csi-controller-sa) 进行注释。

   ```
   kubectl apply -f mountpoint-s3-service-account.yaml
   ```

   > **Note**
   >
   > 在此过程中部署插件后，系统会创建一个名为 s3-csi-driver-sa 的服务账户，并将其配置为使用该服务账户。

第 3 步：安装适用于 Amazon S3 的 Mountpoint CSI 驱动程序

您可以通过 Amazon EKS 附加组件，安装适用于 Amazon S3 的 Mountpoint CSI 驱动程序。您可以使用以下工具将附加组件添加到您的集群。

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”
- the section called “Amazon CLI”

您也可以将适用于 Amazon S3 的 Mountpoint CSI 驱动程序作为自主管理型安装进行安装。有关执行自主管理型安装的说明，请参阅 GitHub 上的 Installation。

从 `v1.8.0` 开始，您可以为 CSI 驱动程序的容器组（pod）配置可容忍的污点。为此，要么使用 `node.tolerations` 指定一组要容忍的自定义污点，要么使用 `node.tolerateAllTaints` 容忍所有污点。有关更多信息，请参阅 Kubernetes 文档中的 Taints and Tolerations。

eksctl

使用 **`eksctl`** 添加 Amazon S3 CSI 附加组件

运行如下命令。将 *`my-cluster`* 替换为您的集群的名称，将 *`111122223333`* 替换为您的账户 ID，并将 *`AmazonEKS_S3_CSI_DriverRole`* 替换为之前创建的 IAM 角色的名称。

```
eksctl create addon --name aws-mountpoint-s3-csi-driver --cluster my-cluster \
  --service-account-role-arn arn:aws-cn:iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole --force
```

如果您删除 *`--force`* 选项，并且任何 Amazon EKS 附加组件设置与您的现有设置冲突，那么更新 Amazon EKS 附加组件将会失败，而且您会收到一条帮助您解决冲突的错误消息。在指定此选项之前，请确保 Amazon EKS 附加组件不会管理您需要管理的设置，因为这些设置会被此选项覆盖。有关此设置其他选项的更多信息，请参阅 `eksctl` 文档中的 Addons（附加组件）。有关 Amazon EKS kubernetes 字段管理的更多信息，请参阅the section called “可以自定义的字段”。

您可以通过配置文件对 `eksctl` 进行自定义。有关更多信息，请参阅 `eksctl` 文档中的使用配置值。以下示例演示了如何容忍所有污点。

```
# config.yaml
...

addons:
```

```
- name: aws-mountpoint-s3-csi-driver
  serviceAccountRoleARN: arn:aws-cn:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
  configurationValues: |-
    node:
      tolerateAllTaints: true
```

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 选择要为其配置适用于 Amazon S3 的 Mountpoint CSI 附加组件的集群名称。
4. 选择附加组件选项卡。
5. 选择获取更多附加组件。
6. 在选择附加组件页面上，执行以下操作：
   a. 在 Amazon EKS 附加组件部分中，选中适用于 Amazon S3 的 Mountpoint CSI 驱动程序复选框。
   b. 选择下一步。
7. 在配置选定的附加组件设置页面上，执行以下操作：
   a. 选择您想使用的版本。
   b. 对于选择 IAM 角色，选择要将适用于 Amazon S3 的 Mountpoint CSI 驱动程序 IAM 策略附加到的 IAM 角色的名称。
   c. （可选）展开可选配置设置后，更新冲突解决方法。如果选择覆盖，则可能用 Amazon EKS 附加组件设置覆盖现有附加组件的一个或多个设置。如果不启用此选项，并且与现有设置存在冲突，则操作将失败。您可以使用生成的错误消息对冲突进行故障排除。在选择此选项之前，请确保 Amazon EKS 附加组件不会管理您需要自行管理的设置。
   d. （可选）展开可选配置设置后，在配置值字段中配置容忍度。
   e. 选择下一步。
8. 在查看和添加页面上，选择创建。附加组件安装完成后，您将看到已安装的附加组件。

Amazon CLI

使用 Amazon CLI 添加适用于 Amazon S3 的 Mountpoint CSI 附加组件

运行如下命令。将 *my-cluster* 替换为您的集群的名称，将 *111122223333* 替换为您的账户 ID，并将 *AmazonEKS_S3_CSI_DriverRole* 替换为之前创建的 IAM 角色的名称。

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver \
  --service-account-role-arn arn:aws-cn:iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole
```

您可以使用 `--configuration-values` 标志自定义命令。以下替代示例演示了如何容忍所有污点。

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver \
  --service-account-role-arn arn:aws-cn:iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole \
  --configuration-values '{"node":{"tolerateAllTaints":true}}'
```

第 4 步：配置适用于 Amazon S3 的 Mountpoint

在大多数情况下，您可以仅使用存储桶名称来配置适用于 Amazon S3 的 Mountpoint。有关配置适用于 Amazon S3 的 Mountpoint 的说明，请参阅 GitHub 上的 Configuring Mountpoint for Amazon S3。

第 5 步：部署示例应用程序

您可以将静态预置部署到现有 Amazon S3 存储桶上的驱动程序。有关更多信息，请参阅 GitHub 上的 Static provisioning。

移除适用于 Amazon S3 的 Mountpoint Amazon EKS 附加组件

您可以通过两种方法来移除适用于 Amazon S3 的 Mountpoint CSI 驱动程序。

- 将附加组件保留在您的集群上 - 此方法将移除 Amazon EKS 对任何设置的管理。还移除 Amazon EKS 通知您更新以及在您启动更新后自动更新 Amazon EKS 附加组件的功能。但是，此方法会保留集群上的附加组件软件。此选项会将附加组件作为自行管理安装使用，而不是作为 Amazon EKS 附加组件使用。使用此方法，附加组件不停机。此过程中的命令使用此选项。
- 从集群中完全删除附加组件 - 我们建议只有当集群中没有资源依赖于附加组件时，才从集群移除 Amazon EKS 附加组件。要执行此选项，请从您在此过程中使用的命令中删除 `--preserve`。

如果附加组件有与其关联的 IAM 账户，则不会移除该 IAM 账户。

您可以使用以下工具移除 Amazon S3 CSI 附加组件：

- the section called “eksctl”

- the section called “Amazon Web Services 管理控制台”
- the section called “Amazon CLI”

eksctl

使用 **eksctl** 移除 Amazon S3 CSI 附加组件

将 *my-cluster* 替换为您的集群的名称，然后运行以下命令。

```
eksctl delete addon --cluster my-cluster --name aws-mountpoint-s3-csi-driver --preserve
```

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 选择要为其移除 Amazon EBS CSI 附加组件的集群名称。
4. 选择附加组件选项卡。
5. 选择适用于 Amazon S3 的 Mountpoint CSI 驱动程序。
6. 选择移除 。
7. 在删除：aws-mountpoint-s3-csi-driver 确认对话框中，执行以下操作：
   a. 如果希望 Amazon EKS 停止管理附加组件设置，请选择在集群上保留。如果要在集群上保留附加组件软件，请执行此操作。这样您可以自行管理附加组件的所有设置。
   b. 输入 aws-mountpoint-s3-csi-driver。
   c. 选择移除。

Amazon CLI

使用 Amazon CLI 移除 Amazon S3 CSI 附加组件

将 *my-cluster* 替换为您的集群的名称，然后运行以下命令。

```
aws eks delete-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver --preserve
```

为 CSI 卷启用快照功能

可借助快照功能创建数据的时间点副本。要在 Kubernetes 中使用此功能，您不仅需要具有支持快照的 CSI 驱动程序（例如 Amazon EBS CSI 驱动程序），同时还需要 CSI 快照控制器。快照控制器可作为 Amazon EKS 托管式附加组件使用，也可以作为自行管理型安装使用。

以下是使用 CSI 快照控制器时需要考虑的一些事项。

- 快照控制器必须与具有快照功能的 CSI 驱动程序一起安装。有关 Amazon EBS CSI 驱动程序的安装说明，请参阅 the section called “Amazon EBS”。
- Kubernetes 不支持通过 CSI 迁移提供的卷的快照，例如使用 `StorageClass` 和预置器 `kubernetes.io/aws-ebs` 的 Amazon EBS 卷。创建卷时必须使用引用 CSI 驱动程序预置器 `ebs.csi.aws.com` 的 `StorageClass` 创建卷。
- Amazon EKS 自动模式不包括快照控制器。EKS 自动模式的存储功能与快照控制器兼容。

我们建议通过 Amazon EKS 托管的插件安装 CSI 快照控制器。此附加组件包括在 Amazon EKS 上创建和管理快照所需的自定义资源定义（CRD）。要将 Amazon EKS 附加组件添加到您的集群，请参阅 the section called “创建附加组件”。有关附加组件的更多信息，请参阅 the section called “Amazon EKS 附加组件”。

或者，如果您想对 CSI 快照控制器进行自主管理型安装，请在 GitHub 上参阅上游 Kubernetes `external-snapshotter` 中的 Usage。

为 Amazon EKS 集群配置网络

您的 Amazon EKS 集群是在 VPC 中创建的。对于在 Amazon 基础设施上运行的容器组（pod），容器组联网由 Amazon VPC 容器网络接口（CNI）插件提供。如果您在自己的基础设施上运行节点，请参阅the section called “配置 CNI”。本章包含可以进一步了解集群联网的以下主题。

主题

- 通过管理控制台将现有 VPC 子网添加到 Amazon EKS 集群
- 查看 Amazon EKS 对 VPC 和子网的联网要求
- 为您的 Amazon EKS 集群创建 Amazon VPC
- 查看集群的 Amazon EKS 安全组要求
- 管理 Amazon EKS 集群的联网附加组件

通过管理控制台将现有 VPC 子网添加到 Amazon EKS 集群

1. 在管理控制台中导航至集群。
2. 从联网选项卡中，选择管理 VPC 资源
3. 从子网下拉列表中，选择集群 VPC 的额外子网。

创建新 VPC 子网：

- 查看 EKS 子网要求
- 请参阅《Amazon Virtual Private Cloud 用户指南》中的创建子网。

查看 Amazon EKS 对 VPC 和子网的联网要求

在创建集群时，您将指定 VPC 以及位于不同可用区中的至少两个子网。本主题概述了 Amazon EKS 对于您用于集群的 VPC 和子网的特定要求和注意事项。如果您没有要用于 Amazon EKS 的 VPC，请参阅the section called “创建 VPC”。如果您要在 Amazon Outposts 上创建本地或扩展集群，请参阅the section called “创建 VPC 和子网”而不是本主题。本主题中的内容适用于具有混合节点的 Amazon EKS 集群。有关混合节点的其他联网要求，请参阅the section called “准备联网”。

VPC 要求和注意事项

在创建集群时，您指定的 VPC 必须满足以下要求和注意事项：

- VPC 必须有足够数量的 IP 地址来供集群、任何节点和您想要创建的其他 Kubernetes 资源使用。如果要使用的 VPC 没有足够数量的 IP 地址，请尝试增加可用 IP 地址的数量。

  为此，您可以更新集群以更改该集群使用的子网和安全组。您可以从 Amazon Web Services 管理控制台、Amazon CLI 的最新版本、Amazon CloudFormation 以及 `eksctl` 版本 `v0.164.0-rc.0` 或更高版本进行更新。您可能需要执行此操作，为子网提供更多可用 IP 地址，以便成功升级集群版本。

  > **Important**
  >
  > 您添加的所有子网均必须位于与创建集群时最初提供的相同一组可用区中。新子网必须满足所有其他要求，例如，它们必须有足够的 IP 地址。
  > 例如，假设您创建一个集群并指定四个子网。按照您指定的顺序，第一个子网位于 `us-west-2a` 可用区中，第二个和第三个子网位于 `us-west-2b` 可用区中，第四个子网位于 `us-west-2c` 可用区中。如果要更改子网，则必须在三个可用区中各提供至少一个子网，并且这些子网必须与原始子网位于同一个 VPC 中。

  如果您需要的 IP 地址多于 VPC 中的 CIDR 块，则可以通过将其他无类别域间路由（CIDR）块与您的 VPC 关联来添加其他 CIDR 块。您可以在创建集群之前或之后将私有 (RFC 1918) 和公有（非 RFC 1918）CIDR 块关联到您的 VPC。

  添加新的 CIDR 块后，您可以立即添加使用该 CIDR 块的节点。但是，由于控制面板仅在协调完成后才会识别新的 CIDR 块，因此集群最多可能需要一个小时才能识别与 VPC 关联的 CIDR 块。然后，您可以对新 CIDR 数据块中的节点和容器组（pod）运行 `kubectl attach`、`kubectl cp`、`kubectl exec`、`kubectl logs` 和 `kubectl port-forward` 命令（这些命令使用 `kubelet API`）。此外，如果您有作为 Webhook 后端运行的容器组（pod），则必须等待控制面板协调完成。

- 通过 Transit Gateway、VPC 对等连接或其他联网配置将 EKS 集群连接到其他 VPC 时，请避免 IP 地址范围重叠。当 EKS 集群的服务 CIDR 与所连接 VPC 的 CIDR 重叠时，就会发生 CIDR 冲突。在这些情况下，ServiceIP 地址优先于具有相同 IP 地址的已连接 VPC 中的资源，但流量路由可能会变得不可预测，并且应用程序可能无法连接到预期资源。

为避免 CIDR 冲突，请确保您的 EKS 服务 CIDR 不与任何已连接 VPC CIDR 重叠，并维护所有 CIDR 分配的集中记录。如果您遇到 CIDR 重叠，您可以将中转网关与共享服务 VPC 结合使用。有关更多信息，请参阅具有共享服务的隔离 VPC 和混合网络中的 Amazon EKS VPC 可路由 IP 地址保护模式。另请参阅《EKS Best Practices Guide》中 VPC and Subnet Considerations 页面的“Communication across VPCs”部分。

- 如果想让 Kubernetes 分配 IPv6 地址给容器组（pod）和服务，请将 IPv6 CIDR 块与您的 VPC 关联。有关更多信息，请参阅 Amazon VPC 用户指南中的关联 IPv6 CIDR 块与 VPC。不能将 IPv6 地址用于在混合节点上运行的容器组和服务，也不能将混合节点用于配置了 IPv6 IP 地址系列的集群。
- VPC 必须具有 DNS 主机名和 DNS 解析支持。否则，节点无法注册到集群。有关更多信息，请参阅《Amazon VPC 用户指南》中的 VPC 的 DNS 属性。
- VPC 可能需要使用 Amazon PrivateLink 的 VPC 节点。有关更多信息，请参阅 the section called “子网要求和注意事项”。

如果您使用 Kubernetes 1.14 或更早版本创建了集群，Amazon EKS 将以下标签添加到您的 VPC 中：

键	值
kubernetes.io/cluster/ *my-cluster*	owned

此标签仅由 Amazon EKS 使用。您可以在不影响服务的情况下删除标签。它不适用于版本 1.15 或更高版本的集群。

子网要求和注意事项

在创建集群时，Amazon EKS 将在您指定的子网中创建 2–4 个弹性网络接口。这些网络接口可实现集群和 VPC 之间的通信。这些网络接口还支持使用 kubelet API 的 Kubernetes 功能。与 kubelet API 的连接用于 kubectl attach、kubectl cp、kubectl exec、kubectl logs 和 kubectl port-forward 命令中。每个 Amazon EKS 创建的网络接口的描述中都有 Amazon EKS *cluster-name* 文本。

当您创建集群时，Amazon EKS 可以在您指定的任何子网中创建其网络接口。创建集群后，您可以更改 Amazon EKS 在哪些子网中创建其网络接口。当您更新集群的 Kubernetes 版本时，Amazon EKS 会删除它创建的原始网络接口，然后创建新的网络接口。这些网络接口可以在与原始网络接口相同的子

网中创建，也可以在与原始网络接口不同的子网中创建。要控制在哪些子网中创建网络接口，可以在创建集群时将指定的子网数限制为只有两个子网，或者在创建集群后更新子网。

集群的子网要求

您在创建或更新集群时指定的子网必须满足以下要求：

- 每个子网必须至少有六个 IP 地址以供 Amazon EKS 使用。但是，我们建议至少使用 16 个 IP 地址。
- 子网必须至少位于两个不同的可用区。
- 子网不能驻留在 Amazon Outposts 或 Amazon Wavelength 中。但是，如果您的 VPC 中有这些子网，则可以在这些类型的子网中部署自行管理的节点和 Kubernetes 资源。有关自主管理型节点的更多信息，请参阅the section called “自行管理的节点”。
- 子网可以是公有子网或私有子网。但是，如果可能，我们建议您指定私有子网。公有子网是指包含路由表的子网，这个路由表中包含指向互联网网关的路由，而私有子网是指具有一个其中不包含指向互联网网关的路由的路由表的子网。
- 子网不能位于以下可用区中：

Amazon 区域	区域名称	不允许使用的可用区 ID
us-east-1	美国东部（弗吉尼亚北部）	use1-az3
us-west-1	美国西部（北加利福尼亚）	usw1-az2
ca-central-1	加拿大（中部）	cac1-az3

按组件划分的 IP 地址系列使用情况

下表包含 Amazon EKS 的每个组件所使用的 IP 地址系列。您可以使用网络地址转换（NAT）或其他兼容性系统，从具有表条目“No”值的系列中的源 IP 地址连接到这些组件。

根据集群的 IP 系列 (ipFamily) 设置，功能可能会有所不同。此设置会更改 Kubernetes 分配给服务的 CIDR 块所使用的 IP 地址类型。设置值为 IPv4 的集群被称为 IPv4 集群，设置值为 IPv6 的集群被称为 IPv6 集群。

组件	IPv4 地址	IPv6 地址	双堆栈地址
EKS API 公用端点	是 [1,3]	是 [1,3]	是 [1,3]
EKS API VPC 端点	是	否	否
EKS Auth API 公有端点（EKS 容器组身份）	是[1]	是[1]	是[1]
EKS Auth API VPC 端点（EKS 容器组身份）	是[1]	是[1]	是[1]
IPv4 Kubernetes 集群公有端点 2	是	否	否
IPv4 Kubernetes 集群私有端点 2	是	否	否
IPv6 Kubernetes 集群公有端点 2	是[1,4]	是[1,4]	是[4]
IPv6 Kubernetes 集群私有端点 2	是[1,4]	是[1,4]	是[4]
Kubernetes 集群子网	是[2]	否	是[2]
节点主 IP 地址	是[2]	否	是[2]
服务 IP 地址的集群 CIDR 范围	是[2]	是[2]	否
来自 VPC CNI 的容器组（pod）IP 地址	是[2]	是[2]	否
IRSA OIDC 发布者 URL	是 [1,3]	是 [1,3]	是 [1,3]

Note

[1] 端点是双堆栈，同时包含 IPv4 和 IPv6 地址。Amazon 外部的应用程序、集群的节点以及集群内的容器组（pod）可以通过 IPv4 或 IPv6 到达此端点。
[2]在创建集群时，您可以在集群的 IP 系列 (ipFamily) 设置中的 IPv4 集群和 IPv6 集群之间进行选择，这是无法更改的。相反，在创建另一个集群并迁移工作负载时，您必须选择不同的设置。
[3] 双栈端点已于 2024 年 8 月推出。要将双堆栈端点与 Amazon CLI 结合使用，请参阅《Amazon SDK 和工具参考指南》中的双堆栈和 FIPS 端点配置。以下列出了新的端点：

EKS API 公用端点

```
eks.region.api.aws
```

IRSA OIDC 发布者 URL

```
oidc-eks.region.api.aws
```

[4] 双堆栈集群端点已于 2024 年 10 月推出。EKS 为在此日期之后创建并在集群的 IP 系列（ipFamily）设置中选择 IPv6 的新集群创建以下端点：

EKS 集群公有/私有端点

```
eks-cluster.region.api.aws
```

节点的子网要求

您可以将节点和 Kubernetes 资源部署到您在创建集群时指定的相同子网。但是，这并不是必要的。这是因为，您还可以将节点和 Kubernetes 资源部署到您在创建集群时未指定的子网。如果您将节点部署到不同的子网，Amazon EKS 不会在这些子网中创建集群网络接口。您向其中部署节点和 Kubernetes 资源的任何子网必须满足以下要求：

- 子网必须有足够的可用 IP 地址才能将所有节点和 Kubernetes 资源部署到其中。
- 如果想让 Kubernetes 分配 IPv6 地址到容器组（pod）和服务，您必须拥有与子网关联的一个 IPv6 CIDR 块和一个 IPv4 CIDR 块。有关更多信息，请参阅《Amazon VPC 用户指南》中的将 IPv6 CIDR 块与您的子网关联。与子网关联的路由表必须包含到 IPv4 和 IPv6 地址的路由。有关更多信息，请参阅《Amazon VPC 用户指南》中的路由。容器组（pod）只分配了 IPv6 地址。但是，Amazon EKS 为您的集群和您的节点创建的网络接口被分配了 IPv4 和 IPv6 地址。

- 如果您需要从互联网入站访问自己的容器组（pod），请确保至少有一个公有子网，且该公有子网具有足够的可用 IP 地址，以便将负载均衡器和入口部署到其中。您可以将负载均衡器部署到公有子网。负载均衡器可以对私有或公有子网中的容器组（pod）进行负载均衡。我们建议尽可能将节点部署到私有子网。
- 如果您计划将节点部署到公有子网，则子网必须自动分配 IPv4 公有地址或 IPv6 地址。如果将节点部署到具有关联的 IPv6 CIDR 块的私有子网，私有子网还必须自动分配 IPv6 地址。如果在 2020 年 3 月 26 日之后使用 Amazon EKS 提供的 Amazon CloudFormation 模板部署 VPC，则此设置已经启用。如果您在此日期之前使用模板部署 VPC，或者您使用自己的 VPC，则必须手动启用此设置。有关模板的信息，请参阅the section called “创建 VPC”。有关更多信息，请参阅 Amazon VPC 用户指南中的修改子网的公有 IPv4 寻址属性和修改子网的 IPv6 寻址属性。
- 如果您将节点部署到的子网是私有子网，且其路由表不包含到网络地址转换（NAT）设备 (IPv4) 或仅限出口的网关 (IPv6) 的路由，则添加使用 Amazon PrivateLink 的 VPC 端点到您的 VPC。节点和容器组（pod）要通信的所有 Amazon 服务都需要 VPC 端点。示例包括 Amazon ECR、Elastic Load Balancing、Amazon CloudWatch、Amazon Security Token Service 和 Amazon Simple Storage Service（Amazon S3）。端点必须包括节点所在的子网。并非所有 Amazon 服务都支持 VPC 端点。有关更多信息，请参阅什么是 Amazon PrivateLink？和与 Amazon PrivateLink 集成的 Amazon 服务。有关更多 Amazon EKS 要求的列表，请参阅 the section called “私有集群”。
- 如果要将负载均衡器部署到子网，则子网必须具有以下标签：
  - 私有子网

键	值
kubernetes.io/role/internal-elb	1

  - 公有子网

键	值
kubernetes.io/role/elb	1

在创建 1.18 及更低版本的 Kubernetes 集群时，Amazon EKS 会将以下标签添加到指定的所有子网。

键	值
kubernetes.io/cluster/ *my-cluster*	shared

在创建新 Kubernetes 集群时，Amazon EKS 不会将该标签添加到子网。如果标签位于以前版本低于 1.19 的集群使用的子网上，则当该集群更新为较新版本时，不会自动从子网上移除该标签。版本 2.1.1 或更低版本的 Amazon 负载均衡器控制器需要此标签。如果您使用的是较新版本的负载均衡器控制器，则可以删除该标签，而不会中断您的服务。有关此控制器的更多信息,请参阅the section called “ Amazon 负载均衡器控制器”。

如果您使用 eksctl 或者任一 Amazon EKS Amazon CloudFormation VPC 模板部署了 VPC，则以下情况适用：

- 2020 年 3 月 26 日或之后 – 公有子网将公有 IPv4 地址自动分配给部署到公有子网的新节点。
- 2020 年 3 月 26 之前 – 公有子网不会将公有 IPv4 地址自动分配给部署到公有子网的新节点。

此更改通过以下方式影响部署到公有子网的新节点组：

- 托管节点组 – 如果节点组在 2020 年 4 月 22 日或之后部署到公有子网，则该公有子网必须启用自动分配公有 IP 地址。有关更多信息，请参阅修改子网的公有 IPv4 寻址属性。
- Linux、Windows 或 Arm 自行管理节点组 – 如果节点组在 2020 年 3 月 26 日或之后部署到公有子网，则该公有子网必须启用自动分配公有 IP 地址。或者，节点必须使用公有 IP 地址启动。有关更多信息，请参阅修改子网的公有 IPv4 寻址属性或在实例启动期间分配公有 IPv4 地址。

共享子网要求和注意事项

您可以使用 VPC 共享与同一 Amazon Organizations 中的其它 Amazon 账户共享子网。您可以在共享子网中创建 Amazon EKS 集群，但要注意以下几点：

- VPC 子网的拥有者必须与参与者账户共享一个子网，该账户才能在其中创建 Amazon EKS 集群。
- 您不能使用 VPC 的默认安全组启动资源，因为此安全组属于拥有者。此外，参与者无法使用其他参与者或拥有者拥有的安全组启动资源。
- 在共享子网中，参与者和拥有者分别控制各自账户中的安全组。子网拥有者可以看到参与者创建的安全组，但不能对其执行任何操作。如果子网拥有者想要删除或修改安全组，则创建安全组的参与者必须执行该操作。

- 如果集群是由参与者创建的，则要注意以下几点：
  - 必须在该账户中创建集群 IAM 角色和节点 IAM 角色。有关更多信息，请参阅the section called “集群 IAM 角色”和the section called “节点 IAM 角色”。
  - 所有节点必须由同一个参与者创建，包括托管节点组。
- 共享 VPC 拥有者无法查看、更新或删除参与者在共享子网中创建的集群。这是对每个账户具有不同访问权限的 VPC 资源的补充。有关更多信息，请参阅 Amazon VPC 用户指南中的拥有者和参与者的责任和权限。
- 如果您使用适用于 Kubernetes 的 Amazon VPC CNI 插件的自定义网络功能，则需要使用拥有者账户中列出的可用区 ID 映射来创建每个 `ENIConfig`。有关更多信息，请参阅 the section called “自定义联网”。

有关 VPC 子网共享的更多信息，请参阅 Amazon VPC 用户指南中的与其他账户共享 VPC。

为您的 Amazon EKS 集群创建 Amazon VPC

您可以使用 Amazon Virtual Private Cloud（Amazon VPC）将 Amazon 资源启动到您定义的虚拟网络中。此虚拟网络与您在自己的数据中心中运行的传统网络极为相似。但是，它带有使用 Amazon Web Services 的可扩展基础设施的优势。我们建议，在部署生产 Amazon EKS 集群之前，深入了解 Amazon VPC 服务。有关更多信息，请参阅《Amazon VPC 用户指南》。

Amazon EKS 集群、节点和 Kubernetes 资源均部署到 VPC 中。如果您想要将现有 VPC 与 Amazon EKS 一起使用，则该 VPC 必须满足 the section called “VPC 和子网要求” 中所述的要求。本主题介绍如何使用 Amazon EKS 提供的 Amazon CloudFormation 模板创建符合 Amazon EKS 要求的 VPC。部署模板后，您可以查看该模板创建的资源，以确切了解它创建了哪些资源以及这些资源的配置。如果使用混合节点，则 VPC 的路由表中必须包含本地网络的路由。有关混合节点联网要求的更多信息，请参阅the section called “准备联网”。

先决条件

要为 Amazon EKS 创建 VPC，您必须拥有必要的 IAM 权限才能创建 Amazon VPC 资源。这些资源包括 VPC、子网、安全组、路由表和路由以及互联网和 NAT 网关。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建带有公有子网示例策略的 VPC 和服务授权参考中的 操作的完整列表。

您可以创建带有公有子网和私有子网、仅公有子网或仅私有子网的 VPC。

公有子网和私有子网

此 VPC 有两个公有子网和两个私有子网。公有子网的关联路由表具有指向互联网网关的路由。但是，私有子网的路由表没有指向互联网网关的路由。一个公有子网和一个私有子网部署到同一个可用区。其它公有子网和私有子网部署到同一 Amazon 区域的另一个可用区。我们建议大多数部署使用此选项。

使用此选项，您可以将节点部署到私有子网。此选项允许 Kubernetes 将负载均衡器部署到公有子网，这些负载均衡器可以对在私有子网的节点上运行的容器组（pod）进行流量负载均衡。系统自动向部署到公有子网的节点分配公有 IPv4 地址，但公有 IPv4 地址不会分配给部署到私有子网的节点。

您还可以将 IPv6 地址分配给公有子网和私有子网中的节点。私有子网中的节点可以与集群和其他 Amazon 服务通信。容器组（pod）可以通过使用 IPv4 地址的 NAT 网关或使用部署在各个可用区的 IPv6 地址的仅出站互联网网关与互联网通信。部署的安全组包含拒绝来自集群或节点以外的来源的所有入站流量但允许所有出站流量的规则。对子网进行标记，以便 Kubernetes 可以向它们部署负载均衡器。

a. 打开 Amazon CloudFormation 控制台。

b. 从导航栏中，选择支持 Amazon EKS 的 Amazon 区域。

c. 依次选择创建堆栈和使用新资源（标准）。

d. 在 Prerequisite - Prepare template（先决条件 - 准备模板）下，请确保选中 Template is ready（模板就绪），然后在 Specify template（指定模板）下选择 Amazon S3 URL。

e. 您可以创建仅支持 IPv4 的 VPC，或支持 IPv4 和 IPv6 的 VPC。请将以下 URL 之一粘贴到 Amazon S3 URL 下方的文本区域，然后选择 Next（下一步）：

- IPv4

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

- IPv4 和 IPv6

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

a. 在 Specify stack details（指定堆栈详细信息）页面上，修改参数，然后选择 Next（下一步）。

- 堆栈名称：为 Amazon CloudFormation 堆栈选择堆栈名称。例如，可以使用上一步中使用的模板名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
- VpcBlock：为您的 VPC 选择 IPv4 CIDR 范围。您部署的每个节点、容器组（pod）和负载均衡器都会分配有此块中的一个 IPv4 地址。默认 IPv4 值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。有关更多信息，请参阅 Amazon VPC 用户指南中的 VPC 和子网大小调整。您还可以在 VPC 创建后向其添加额外的 CIDR 块。如果您要创建 IPv6 VPC，则系统从 Amazon 的 Global Unicast Address 空间自动为您分配 IPv6 CIDR 范围。
- PublicSubnet01Block：指定公有子网 1 的 IPv4 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。如果您要创建 IPv6 VPC，则在模板中为您指定此区块。
- PublicSubnet02Block：指定公有子网 2 的 IPv4 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。如果您要创建 IPv6 VPC，则在模板中为您指定此区块。
- PrivateSubnet01Block：指定私有子网 1 的 IPv4 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。如果您要创建 IPv6 VPC，则在模板中为您指定此区块。
- PrivateSubnet02Block：指定私有子网 2 的 IPv4 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。如果您要创建 IPv6 VPC，则在模板中为您指定此区块。

b. （可选）在 Configure stack options（配置堆栈选项）页面上，为堆栈资源添加标签，然后选择 Next（下一步）。

c. 在 Review（查看）页面中，请选择 Create stack（创建堆栈）。

d. 创建堆栈后，在控制台中选中它，然后选择 Outputs (输出)。

e. 记录所创建 VPC 的 VpcId。在您创建和节点时需要此功能。

f. 记录已创建子网的 SubnetIds，以及您将其作为公有子网还是私有子网创建。在您创建和节点时至少需要其中的两个功能。

g. 如果您创建了 IPv4 VPC，请跳过此步骤。如果您创建了 IPv6 VPC，则必须为模板创建的公有子网启用自动分配 IPv6 地址选项。私有子网的该设置已启用。要启用该设置，请完成以下步骤：

   i. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。

   ii. 请在左侧导航窗格中，选择 Subnets（子网）

iii. 请选择其中一个公有子网（*stack-name*/SubnetPublic01 或 *stack-name*/SubnetPublic02，包含 public 字样），然后依次选择操作、编辑子网设置。

iv. 请选中 Enable auto-assign IPv6 address（启用自动分配 IPv6 地址）复选框，然后选择 Save（保存）。

v. 为另一个公有子网再次完成上述步骤。

仅公有子网

此 VPC 有三个部署到 Amazon 区域中的不同可用区的公有子网。所有节点都会自动分配公有 IPv4 地址，并且可以通过互联网网关发送和接收互联网流量。部署了安全组，拒绝所有入站流量并允许所有出站流量。对子网进行标记，以便 Kubernetes 可以向它们部署负载均衡器。

a. 打开 Amazon CloudFormation 控制台。

b. 从导航栏中，选择支持 Amazon EKS 的 Amazon 区域。

c. 依次选择创建堆栈和使用新资源（标准）。

d. 在 Prepare template（准备模板）下，确保 Template is ready（模板已就绪）处于选中状态，然后在 Template source（模板源）下选择 Amazon S3 URL。

e. 将以下 URL 粘贴到 Amazon S3 URL 下方的文本区域，然后选择 Next（下一步）：

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-sample.yaml
```

a. 在 Specify Details（指定详细信息）页面上，输入参数，然后选择 Next（下一步）。

- 堆栈名称：为 Amazon CloudFormation 堆栈选择堆栈名称。例如，您可以将其称为 *amazon-eks-vpc-sample*。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
- VpcBlock：为您的 VPC 选择 CIDR 块。您部署的每个节点、容器组（pod）和负载均衡器都会分配有此块中的一个 IPv4 地址。默认 IPv4 值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。有关更多信息，请参阅 Amazon VPC 用户指南中的 VPC 和子网大小调整。您还可以在 VPC 创建后向其添加额外的 CIDR 块。
- Subnet01Block：指定子网 1 的 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。

- Subnet02Block：指定子网 2 的 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。
- Subnet03Block：指定子网 3 的 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。

b. （可选）在 Options (选项) 页面上，为您的堆栈资源添加标签。选择下一步。

c. 在 Review 页面上，选择 Create 。

d. 创建堆栈后，在控制台中选中它，然后选择 Outputs (输出)。

e. 记录所创建 VPC 的 VpcId。在您创建和节点时需要此功能。

f. 记录已创建的子网的 SubnetIds。在您创建和节点时至少需要其中的两个功能。

g. （可选）您部署到此 VPC 的任何集群都能够将私有 IPv4 地址分配给您的容器组（pod）和服务。如果想要将集群部署到此 VPC，以便将私有 IPv6 地址分配给您的容器组（pod）和服务，则必须对您的 VPC、子网、路由表和安全组进行更新。有关更多信息，请参阅 Amazon VPC 用户指南中的将现有 VPC 从 IPv4 迁移到 IPv6。Amazon EKS 要求您的子网启用 Auto-assign IPv6 地址选项。该选项默认已禁用。

仅私有子网

此 VPC 有三个部署到 Amazon 区域中的不同可用区的私有子网。部署到子网的资源无法访问互联网，互联网也无法访问子网中的资源。该模板使用 Amazon PrivateLink 为节点通常需要访问的几个 Amazon 服务创建 VPC 端点。如果您的节点需要出站互联网访问，您可以在创建 VPC 后在每个子网的可用区中添加公有 NAT 网关。安全组被创建的用于拒绝所有入站流量，但部署到子网的资源除外。安全组还允许所有出站流量。对子网进行标记，以便 Kubernetes 可以向它们部署内部负载均衡器。如果您正在使用此配置创建 VPC，请参阅 the section called “私有集群” 以了解其它要求和注意事项。

a. 打开 Amazon CloudFormation 控制台。

b. 从导航栏中，选择支持 Amazon EKS 的 Amazon 区域。

c. 依次选择创建堆栈和使用新资源（标准）。

d. 在 Prepare template（准备模板）下，确保 Template is ready（模板已就绪）处于选中状态，然后在 Template source（模板源）下选择 Amazon S3 URL。

e. 将以下 URL 粘贴到 Amazon S3 URL 下方的文本区域，然后选择 Next（下一步）：

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-fully-private-vpc.yaml
```

a. 在 Specify Details（指定详细信息）页面上，输入参数，然后选择 Next（下一步）。

- 堆栈名称：为 Amazon CloudFormation 堆栈选择堆栈名称。例如，您可以称之为 *amazon-eks-fully-private-vpc*。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
- VpcBlock：为您的 VPC 选择 CIDR 块。您部署的每个节点、容器组（pod）和负载均衡器都会分配有此块中的一个 IPv4 地址。默认 IPv4 值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。有关更多信息，请参阅 Amazon VPC 用户指南中的 VPC 和子网大小调整。您还可以在 VPC 创建后向其添加额外的 CIDR 块。
- PrivateSubnet01Block：指定子网 1 的 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。
- PrivateSubnet02Block：指定子网 2 的 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。
- PrivateSubnet03Block：指定子网 3 的 CIDR 块。原定设置值为大多数实现提供了足够的 IP 地址，但如果没有提供，那么您可以对其进行更改。

b. （可选）在 Options (选项) 页面上，为您的堆栈资源添加标签。选择下一步。

c. 在 Review 页面上，选择 Create 。

d. 创建堆栈后，在控制台中选中它，然后选择 Outputs (输出)。

e. 记录所创建 VPC 的 VpcId。在您创建和节点时需要此功能。

f. 记录已创建的子网的 SubnetIds。在您创建和节点时至少需要其中的两个功能。

g. （可选）您部署到此 VPC 的任何集群都能够将私有 IPv4 地址分配给您的容器组（pod）和服务。如果想要将集群部署到此 VPC，以便将私有 IPv6 地址分配给您的容器组（pod）和服务，则必须对您的 VPC、子网、路由表和安全组进行更新。有关更多信息，请参阅 Amazon VPC 用户指南中的将现有 VPC 从 IPv4 迁移到 IPv6。Amazon EKS 要求您的子网启用 Auto-assign IPv6 地址选项（该选项默认为禁用）。

查看集群的 Amazon EKS 安全组要求

本主题介绍 Amazon EKS 集群的安全组要求。

默认集群安全组

在创建集群时，Amazon EKS 将创建一个名为 eks-cluster-sg-*my-cluster-uniqueID* 的安全组。该安全组具有以下默认规则：

Rule type	协议	端口	来源	目标
入站	全部	全部	自身	
出站	全部	全部		0.0.0.0/0 (IPv4) 或 ::/0 (IPv6)
出站	全部	全部		自身（用于 EFA 流量）

默认安全组包含一条出站规则，该规则允许目标为同一安全组的 Elastic Fabric Adapter（EFA）流量。这将在集群内启用 EFA 流量，有利于 AI/ML 和高性能计算（HPC）工作负载。有关更多信息，请参阅《Amazon Elastic Compute Cloud 用户指南》中的 适用于 Amazon EC2 上 AI/ML 和 HPC 工作负载的 Elastic Fabric Adapter。

> **Important**
>
> 如果您的集群不需要出站规则，则可以将其删除。如果将其删除，您仍须遵守限制集群流量中列出的最少规则。如果您删除入站规则，则每当集群更新时，Amazon EKS 都会重新创建该规则。

Amazon EKS 会向安全组添加以下标签。如果您删除标签，则每当集群更新时，Amazon EKS 都会将其重新向安全组添加标签。

键	值
kubernetes.io/cluster/ *my-cluster*	owned
aws:eks:cluster-name	*my-cluster*
Name	eks-cluster-sg- *my-cluster* -*uniqueid*

Amazon EKS 会自动将此安全组关联到它还创建的以下资源：

- 在您创建集群时创建的 2–4 个弹性网络接口（在本文档其余部分中称为网络接口）。

- 您创建的任何托管节点组中节点的网络接口。

默认规则允许所有流量在集群和节点之间自由流动，并允许所有出站流量到任何目的地。在创建集群时，您可以（选择性地）指定您自己的安全组。如果您这样做，那么 Amazon EKS 还会将您指定的安全组与它为集群创建的网络接口关联起来。但是，它不会将它们与您创建的任何节点组关联起来。

您可以在Amazon Web Services 管理控制台中集群的联网部分下，确定您的集群安全组的 ID。或者，您可以使用以下 Amazon CLI 命令进行这项操作。

```
aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

限制集群流量

如果您需要限制集群与节点之间的开放端口，则可以删除默认出站规则并添加集群所需的以下最少规则。如果您删除默认入站规则，则每当集群更新时，Amazon EKS 都会重新创建该规则。

Rule type	协议	端口	目标
出站	TCP	443	集群安全组
出站	TCP	10250	集群安全组
出站（DNS）	TCP 和 UDP	53	集群安全组

您还必须为以下流量添加规则：

- 您预计节点要用于节点间通信的任何协议和端口。
- 出站互联网访问，以便节点可以访问 Amazon EKS API 以在启动时进行集群自检和节点注册。如果您的节点没有互联网访问权限，请查看部署具有有限互联网访问权限的私有集群了解其它注意事项。
- 从它们需要从中拉取镜像的 Amazon ECR 或其他容器注册表 API（例如 DockerHub）中拉取容器镜像的节点访问权限。有关更多信息，请参阅Amazon一般参考中的Amazon IP 地址范围。
- 节点访问 Amazon S3。
- IPv4 和 IPv6 地址需要单独的规则。
- 如果您使用的是混合节点，则必须向集群添加额外的安全组，以允许与本地节点和容器组进行通信。有关更多信息，请参阅 the section called “准备联网”。

如果要考虑限制规则，建议全面测试所有容器组（pod），然后再将更改的规则应用于生产集群。

如果您最初使用了 Kubernetes `1.14` 和 `eks.3` 或更早版本的平台部署了集群，则考虑以下事项：

- 您可能还拥有控制面板和节点安全组。创建这些组时，它们包括上表中列出的限制规则。这些安全组不再需要，可以删除。但是，您需要确保集群安全组包含这些组所包含的规则。
- 如果您直接使用 API 部署集群，或者使用 Amazon CLI 或 Amazon CloudFormation 之类的工具创建集群且在创建集群时您未指定安全组，则 VPC 的默认安全组应用到 Amazon EKS 创建的集群网络接口。

共享安全组

Amazon EKS 支持共享安全组。

- 安全组 VPC 关联将安全组与同一账户和区域中的多个 VPC 关联起来。
  - 在《Amazon VPC 用户指南》中了解如何将安全组与多个 VPC关联起来。
- 共享安全组允许您与其他 Amazon 账户共享安全组。账户必须位于同一 Amazon 组织。
  - 在《Amazon VPC 用户指南》中了解如何与组织共享安全组。
- 安全组始终仅限于单个 Amazon 区域。

Amazon EKS 注意事项

- EKS 对共享或多 VPC 安全组的要求与标准安全组相同。

管理 Amazon EKS 集群的联网附加组件

有多个联网附加组件可用于 Amazon EKS 集群。

内置附加组件

> **Note**
>
> 当您创建 EKS 集群时：
>
> - 使用 Amazon 控制台：内置附加组件（例如 CoreDNS、kube-proxy 等）会自动安装为 Amazon EKS 附加组件。通过 Amazon 控制台、CLI 或 SDK 可以轻松配置和更新这些附加组件。

- 使用其他方法（CLI、SDK 等）：安装的内置附加组件与作为常规 Kubernetes 部署运行的自行管理版本相同。由于这些附加组件无法通过 Amazon 工具进行管理，因此需要手动配置和更新。

我们建议使用 Amazon EKS 附加组件而不是自行管理版本，以简化附加组件管理并通过 Amazon 服务实现集中配置和更新。

适用于 Kubernetes 的 Amazon VPC CNI 插件

此 CNI 附加组件会创建弹性网络接口并将其附加到您的 Amazon EC2 节点。附加组件还会将 VPC 中的私有 `IPv4` 或 `IPv6` 地址分配给每个容器组（pod）和服务。您的集群上默认安装此附加组件。有关更多信息，请参阅 the section called “Amazon VPC CNI”。如果您使用混合节点功能，则默认情况下仍会安装 VPC CNI，不过会使用反亲和性规则阻止其在混合节点上运行。要详细了解有关混合节点的 CNI 选项，请参阅the section called “配置 CNI”。

CoreDNS

CoreDNS 是一个灵活、可扩展的 DNS 服务器，可用作 Kubernetes 集群 DNS。CoreDNS 为集群中的所有容器组（pod）提供名称解析。您的集群上默认安装此附加组件。有关更多信息，请参阅 the section called “CoreDNS”。

`kube-proxy`

此附加组件负责维护 Amazon EC2 节点上的网络规则，实现与容器组（pod）的网络通信。您的集群上默认安装此附加组件。有关更多信息，请参阅 the section called “kube-proxy”。

可选 Amazon 联网附加组件

Amazon负载均衡器控制器

当您部署 `loadbalancer` 类型的 Kubernetes 服务对象时，控制器会创建 Amazon 网络负载均衡器。创建 Kubernetes 入口对象时，控制器会创建 Amazon 应用程序负载均衡器。我们建议使用此控制器来预置网络负载均衡器，而不是使用 Kubernetes 内置的旧版云提供商控制器。有关更多信息，请参阅 Amazon Load Balancer Controller 文档。

Amazon网关 API 控制器

借助此控制器，您可以使用 Kubernetes 网关 API 跨多个 Kubernetes 集群连接服务。控制器使用 Amazon VPC Lattice 服务连接在 Amazon EC2 实例、容器和无服务器功能上运行的 Kubernetes 服务。有关更多信息，请参阅 Amazon Gateway API 控制器文档。

有关附加组件的更多信息，请参阅 the section called “Amazon EKS 附加组件”。

使用 Amazon VPC CNI 将 IP 分配给容器组（pod）

> **Tip**
>
> 使用 Amazon EKS 自动模式时，您无需安装或升级联网附加组件。自动模式包含容器组联网和负载均衡功能。
> 有关更多信息，请参阅 EKS 自动模式。

适用于 Kubernetes 的 Amazon VPC CNI 插件附加组件部署在 Amazon EKS 集群中的每个 Amazon EC2 节点上。附加组件会创建弹性网络接口并将其附加到 Amazon EC2 节点。该附加组件还会将 VPC 中的私有 IPv4 或 IPv6 地址分配给每个容器组（pod）。

附加组件版本随集群中的每个 Fargate 节点一起部署，但您无需在 Fargate 节点上对其进行更新。
其他兼容的 CNI 插件也可用于 Amazon EKS 集群，但对于在 Amazon 基础设施上运行的节点，这是 Amazon EKS 唯一支持的 CNI 插件。有关其它兼容的 CNI 插件的更多信息，请参阅the section called “备用 CNI 插件”。不支持将 VPC CNI 用于混合节点。要详细了解有关混合节点的 CNI 选项，请参阅the section called “配置 CNI”。

下表列出了每个 Kubernetes 版本的 Amazon EKS 附加组件类型的最新可用版本。

Amazon VPC CNI 版本

Kubernetes 版本	Amazon EKS 类型的 VPC CNI 版本
1.34	v1.20.4-eksbuild.1
1.33	v1.20.4-eksbuild.1
1.32	v1.20.4-eksbuild.1

Kubernetes 版本	Amazon EKS 类型的 VPC CNI 版本
1.31	v1.20.4-eksbuild.1
1.30	v1.20.4-eksbuild.1
1.29	v1.20.4-eksbuild.1
1.28	v1.20.4-eksbuild.1

Important

如果您自行管理此附加组件，则表中的版本可能与可用的自行管理版本不同。有关更新此附加组件的自行管理类型的更多信息，请参阅the section called “更新（自主管理型）”。

Important

要升级到 VPC CNI v1.12.0 或更高版本，必须先升级到 VPC CNI v1.7.0。建议您一次更新一个次要版本。

注意事项

以下是使用该功能的注意事项。

- 版本指定为 `major-version.minor-version.patch-version-eksbuild.build-number`。
- 检查每个功能的版本兼容性。适用于 Kubernetes 的 Amazon VPC CNI 插件的每个版本的某些功能，都需要特定的 Kubernetes 版本。使用不同的 Amazon EKS 功能时，如果需要特定版本的附加组件，则会在功能文档中注明。除非您出于某个特定原因需要运行早期版本，否则建议您运行最新版本。

创建 Amazon VPC CNI（Amazon EKS 附加组件）

使用以下步骤创建适用于 Kubernetes Amazon EKS 附加组件的 Amazon VPC CNI 插件。

开始之前，请查看注意事项。有关更多信息，请参阅 the section called “注意事项”。

先决条件

以下是适用于 Kubernetes Amazon EKS 附加组件的 Amazon VPC CNI 插件的先决条件。

- 现有 Amazon EKS 集群。要部署一个角色，请参阅开始使用。
- 适用于集群的现有 Amazon Identity and Access Management（IAM）OpenID Connect（OIDC）提供者。要确定您是否已经拥有一个或是否要创建一个，请参阅the section called “IAM OIDC 提供者”。
- 附加有 AmazonEKS_CNI_Policy IAM 策略（如果集群使用 IPv4 系列）或 IPv6 策略（如果集群使用 IPv6 系列）的 IAM 角色。有关 VPC CNI 角色的更多信息，请参阅the section called “配置 IRSA”。有关 IPv6 策略的信息，请参阅the section called “为使用 IPv6 系列的集群创建 IAM policy”。

> **Important**
>
> 适用于 Kubernetes 的 Amazon VPC CNI 插件版本 `v1.16.0` 到 `v1.16.1` 实现了 CNI 规范版本 `v1.0.0`。有关 CNI 规范 `v1.0.0` 的更多信息，请参阅 GitHub 上的 Container Network Interface (CNI) Specification。

过程

完成先决条件后，请使用以下步骤创建附加组件。

1. 查看集群上当前安装的附加组件版本。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

示例输出如下。

```
v1.16.4-eksbuild.2
```

2. 查看集群上当前安装的附加组件类型。根据您创建集群时使用的工具，您的集群上目前可能没有安装 Amazon EKS 附加组件类型。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 addon.addonVersion --output text
```

如果返回来的是版本号，则表明您的集群上安装有 Amazon EKS 类型的附加组件，而且此流程中其余的步骤，您也不需要走完。如果返回来的是一个错误，则表明您的集群上没有安装 Amazon EKS 类型的附加组件。要安装，就需完成此流程中其余的步骤。

3. 保存您当前安装的附加组件的配置。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

4. 使用 Amazon CLI 创建附加组件。如果要使用 Amazon Web Services 管理控制台 或 `eksctl` 来创建附加组件，请参阅 the section called “创建附加组件” 并指定 `vpc-cni` 为附加组件名称。将以下命令复制到您的设备。根据需要对该命令进行以下修改，然后运行修改后的命令。

   - 将 *`my-cluster`* 替换为您的集群的名称。
   - 将 *`v1.20.3-eksbuild.1`* 替换为适合您的集群版本的最新版本表中列出的最新版本。有关最新版本表，请参阅 the section called “Amazon VPC CNI 版本”。
   - 将 *`111122223333`* 替换为您的账户 ID，并将 *`AmazonEKSVPCCNIRole`* 替换为您创建的现有 IAM 角色的名称。指定角色需要您的集群具有 IAM OpenID Connect（OIDC）提供者。要确定您的集群是否具有此提供程序，或者要创建此提供程序，请参阅the section called “IAM OIDC 提供者”。

   ```
   aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
    v1.20.3-eksbuild.1 \
       --service-account-role-arn arn:aws-cn:iam::111122223333:role/
   AmazonEKSVPCCNIRole
   ```

   如果您对当前附加组件应用的自定义设置与 Amazon EKS 附加组件的默认设置相冲突，则创建可能会失败。如果创建失败，您会收到一条可以帮助您解决问题的错误信息。或者，您可以将 `--resolve-conflicts OVERWRITE` 添加到前面的命令中。这样一来，附加组件会覆盖任何现有的自定义设置。创建附加组件后，您可以使用自定义设置对其进行更新。

5. 确认集群的 Kubernetes 版本的附加组件最新版本已添加到集群。将 *`my-cluster`* 替换为您的集群的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 addon.addonVersion --output text
```

附加组件创建可能需要几秒钟才能完成。

示例输出如下。

```
v1.20.3-eksbuild.1
```

6. 如果您在创建 Amazon EKS 附加组件之前对原始附加组件进行了自定义设置，则请使用您在上一步中保存的配置，以使用您的自定义设置更新 EKS 附加组件。按照 the section called “更新（EKS 附加组件）” 中的步骤操作。
7. （可选）将 `cni-metrics-helper` 安装到您的集群。它会抓取弹性网络接口和 IP 地址信息，在集群级别聚合这些信息，并将指标发布到 Amazon CloudWatch。有关更多信息，请参阅 GitHub 上的 cni-metrics-helper。

更新 Amazon VPC CNI（Amazon EKS 附加组件）

更新适用于 Kubernetes 的 Amazon VPC CNI 插件的附加组件 Amazon EKS 类型。如果您尚未将 Amazon EKS 类型的附加组件添加到集群，您可以通过按照the section called “创建”中的说明来按照它。或者，按照the section called “更新（自主管理型）”更新其它类型的 VPC CNI 安装。

1. 查看集群上当前安装的附加组件版本。将 *my-cluster* 替换为您的集群名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 "addon.addonVersion" --output text
```

示例输出如下。

```
v1.20.0-eksbuild.1
```

将版本与 the section called “Amazon VPC CNI 版本”中的最新版本表进行比较。如果返回的版本与最新版本表中集群的 Kubernetes 版本相同，则表示集群上已经安装最新版本，您无需完成此过程的其余部分。如果您在输出中收到错误信息而不是版本号，则您的集群上没有安装 Amazon EKS 类型的附加组件。您需要先创建附加组件，然后才能使用此过程对其进行更新。要创建 Amazon EKS 类型的 VPC CNI 附加组件，可按照the section called “创建”中的说明操作。

2. 保存您当前安装的附加组件的配置。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

3. 使用 Amazon CLI 更新您的附加组件。如果您想要使用 Amazon Web Services 管理控制台 或 `eksctl` 更新附加组件，则请参阅 the section called “更新附加组件”。将以下命令复制到您的设备。根据需要对该命令进行以下修改，然后运行修改后的命令。

- 将 *my-cluster* 替换为您的集群的名称。
- 将 *v1.20.0-eksbuild.1* 替换为适合您的集群版本的最新版本表中列出的最新版本。
- 将 *111122223333* 替换为您的账户 ID，并将 *AmazonEKSVPCCNIRole* 替换为您创建的现有 IAM 角色的名称。要为 VPC CNI 创建 IAM 角色，请参阅the section called “步骤 1：创建适用于 Kubernetes 的 Amazon VPC CNI 插件 IAM 角色”。指定角色需要您的集群具有 IAM OpenID Connect（OIDC）提供者。要确定您的集群是否具有此提供程序，或者要创建此提供程序，请参阅 the section called “IAM OIDC 提供者”。
- `--resolve-conflicts PRESERVE` 选项保留附加组件的现有配置值。如果您为附加组件设置设定了自定义值，但未使用此选项，则 Amazon EKS 会使用其默认值覆盖您的值。如果您使用此选项，那么我们建议您在更新生产集群上的附加组件之前，先测试非生产集群上所有更改的字段和值。如果您将该值改为 `OVERWRITE`，则所有设置都将更改为 Amazon EKS 的默认值。如果您为任何设置设定了自定义值，这些值可能会被 Amazon EKS 的默认值覆盖。如果您将该值改为 `none`，Amazon EKS 不会更改任何设置的值，但更新可能会失败。如果更新失败，您会收到一条帮助您解决冲突的错误消息。
- 如果您没有更新配置设置，则请从命令中移除 `--configuration-values '{*"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}*}'`。如果您更新配置设置，则将 *"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}* 替换为您想要设置的设置。在此示例中，`AWS_VPC_K8S_CNI_EXTERNALSNAT` 环境变量设置为 `true`。您指定的值必须对配置架构有效。如果您不知道配置架构，请运行 `aws eks describe-addon-configuration --addon-name vpc-cni --addon-version` *v1.20.0-eksbuild.1* ，并将 *v1.20.0-eksbuild.1* 替换为您要查看配置的附加组件的版本号。将在输出中返回架构。如果您有任何现有的自定义配置，想要将其全部删除，并将所有设置的值设置回 Amazon EKS 的默认值，请从命令中删除 *"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}*，这样就可以有空的 `{}`。有关每项设置的说明，请参阅 GitHub 上的 CNI 配置变量。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.20.3-eksbuild.1 \
    --service-account-role-arn arn:aws-cn:iam::111122223333:role/
AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"env":
{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}}'
```

可能需要几秒钟才能完成更新。

4. 确认附加组件版本已更新。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

可能需要几秒钟才能完成更新。

示例输出如下。

```
{
    "addon": {
        "addonName": "vpc-cni",
        "clusterName": "my-cluster",
        "status": "ACTIVE",
        "addonVersion": "v1.20.3-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws-cn:eks:region:111122223333:addon/my-cluster/vpc-
cni/74c33d2f-b4dc-8718-56e7-9fdfa65d14a9",
        "createdAt": "2023-04-12T18:25:19.319000+00:00",
        "modifiedAt": "2023-04-12T18:40:28.683000+00:00",
        "serviceAccountRoleArn": "arn:aws-cn:iam::111122223333:role/
AmazonEKSVPCCNIRole",
        "tags": {},
        "configurationValues": "{\"env\":{\"AWS_VPC_K8S_CNI_EXTERNALSNAT\":\"true
\"}}"
    }
}
```

更新 Amazon VPC CNI（自主管理型附加组件）

> **Important**
>
> 建议您向集群添加 Amazon EKS 类型的附加组件，而不是自行管理类型的附加组件。如果不熟悉这些类型之间的区别，请参阅 the section called “Amazon EKS 附加组件”。有关向集群中添加 Amazon EKS 附加组件的更多信息，请参阅 the section called “创建附加组件”。如果您无法使用 Amazon EKS 附加组件，我们鼓励您向容器路线图 GitHub 存储库提交有关您为什么无法使用的问题。

1. 确认已在您的集群上安装 Amazon EKS 类型的附加组件。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 addon.addonVersion --output text
```

如果返回错误消息，则表明您的集群上安装有 Amazon EKS 类型的附加组件。要自行管理附加组件，请完成此过程中的剩余步骤以更新附加组件。如果返回版本号，则表明集群上安装有 Amazon EKS 类型的附加组件。要对其进行更新，请使用 the section called “更新附加组件” 中的过程，而不是此过程。如果不熟悉这些附加组件类型之间的区别，请参阅 the section called “Amazon EKS 附加组件”。

2. 查看集群上当前安装的容器映像版本。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

示例输出如下。

```
v1.20.0-eksbuild.1
```

输出可能不包含版本号。

3. 备份当前设置，以便在更新版本后可以配置相同设置。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

要查看可用版本并熟悉要更新到的版本中的更改，请参阅 GitHub 上的 releases。请注意，我们建议更新到最新可用版本表中列出的相同 major.minor.patch 版本，即使 GitHub 上提供了更新版本。有关最新可用版本表，请参阅 the section called “Amazon VPC CNI 版本”。表中列出的构建版本并未在 GitHub 上列出的自行管理版本中指定。通过完成以下选项之一中的任务来更新您的版本：

- 如果您的附加组件没有任何自定义设置，请在 GitHub 上针对要更新到的版本运行 `To apply this release:` 标题下的命令。
- 如果有自定义设置，请使用以下命令下载清单文件。将 *https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.20.0/config/master/aws-k8s-cni.yaml* 更改为您要更新到的 GitHub 上的版本的 URL。

```
curl -O https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.20.3/config/
master/aws-k8s-cni.yaml
```

如有必要，使用您在上一步中创建的备份中的自定义设置修改清单，然后将修改后的清单应用到集群。如果节点无法访问从中提取映像的私有 Amazon EKS Amazon ECR 存储库（参阅清单中以 `image:` 开头的行），则您必须下载映像，将其复制到自己的存储库，然后修改清单以从存储库中提取映像。有关更多信息，请参阅 the section called “将镜像复制到存储库”。

```
kubectl apply -f aws-k8s-cni.yaml
```

4. 确认新版本现已安装在集群上。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

示例输出如下。

```
v1.20.3
```

5. （可选）将 `cni-metrics-helper` 安装到您的集群。它会抓取弹性网络接口和 IP 地址信息，在集群级别聚合这些信息，并将指标发布到 Amazon CloudWatch。有关更多信息，请参阅 GitHub 上的 cni-metrics-helper。

配置 Amazon VPC CNI 插件以使用 IRSA

适用于 Kubernetes 的 Amazon VPC CNI 插件是在 Amazon EKS 集群中用于容器组（pod）联网的联网插件。该插件负责向 Kubernetes 容器组分配 VPC IP 地址并为每个节点上的容器组配置所需网络。

> **Note**
>
> Amazon VPC CNI 插件还支持 Amazon EKS 容器组身份。有关更多信息，请参阅 the section called “分配 IAM 角色”。

该插件：

- 需要 Amazon Identity and Access Management（IAM）权限。如果集群使用 `IPv4` 系列，则权限在 `AmazonEKS_CNI_Policy` Amazon 托管式策略中指定。如果集群使用 `IPv6` 系列，则必须将权限添加到您创建的 IAM 策略中；有关说明，请参阅the section called “为使用 IPv6 系列的集群创建 IAM policy”。您可以将该策略附加到 Amazon EKS 节点 IAM 角色或单独 IAM 角色。有关将策略附

加到 Amazon EKS 节点 IAM 角色的说明，请参阅the section called “节点 IAM 角色”。我们建议您按照本主题中的详细说明，将其分配给一个单独的角色。

- 在部署时创建并配置为使用名为 aws-node 的 Kubernetes 服务账户。服务账户绑定到被分配了所需的 Kubernetes 权限的名为 aws-node 的 Kubernetes clusterrole。

> **Note**
>
> 适用于 Kubernetes 的 Amazon VPC CNI 插件容器组（pod）有权访问分配给 Amazon EKS 节点 IAM 角色的权限，除非您阻止对 IMDS 的访问。有关更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。

- 需要现有 Amazon EKS 集群。要部署一个角色，请参阅开始使用。
- 需要适用于集群的现有 Amazon Identity and Access Management（IAM）OpenID Connect（OIDC）提供者。要确定您是否已经拥有一个或是否要创建一个，请参阅the section called “IAM OIDC 提供者”。

步骤 1：创建适用于 Kubernetes 的 Amazon VPC CNI 插件 IAM 角色

1. 确定您的集群的 IP 系列。

```
aws eks describe-cluster --name my-cluster | grep ipFamily
```

示例输出如下。

```
"ipFamily": "ipv4"
```

输出可能会返回 ipv6。

2. 创建 IAM 角色。您可以使用 eksctl 或 kubectl 和 Amazon CLI 以创建 IAM 角色。

eksctl

- 使用与您的集群的 IP 系列匹配的命令创建 IAM 角色并将 IAM 策略附加到该角色。此命令创建并部署一个创建 IAM 角色的 Amazon CloudFormation 堆栈，向其附加您指定的策略，并使用所创建 IAM 角色的 ARN 对现有 aws-node Kubernetes 服务账户添加注释。
  - IPv4

将 *my-cluster* 替换为您自己的值。

```
eksctl create iamserviceaccount \
    --name aws-node \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKSVPCCNIRole \
    --attach-policy-arn arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy \
    --override-existing-serviceaccounts \
    --approve
```

- IPv6

  将 *my-cluster* 替换为您自己的值。将 *111122223333* 替换为您的账户 ID 并将 *AmazonEKS_CNI_IPv6_Policy* 替换为您的 IPv6 策略的名称。如果没有 IPv6 策略，请参阅the section called “为使用 IPv6 系列的集群创建 IAM policy”创建一个。要将 IPv6 用于您的集群，它必须满足多项要求。有关更多信息，请参阅 the section called “IPv6”。

  ```
  eksctl create iamserviceaccount \
      --name aws-node \
      --namespace kube-system \
      --cluster my-cluster \
      --role-name AmazonEKSVPCCNIRole \
      --attach-policy-arn arn:aws-cn:iam::111122223333:policy/
  AmazonEKS_CNI_IPv6_Policy \
      --override-existing-serviceaccounts \
      --approve
  ```

kubectl 和 Amazon CLI

i. 查看集群的 OIDC 提供者 URL。

```
aws eks describe-cluster --name my-cluster --query
 "cluster.identity.oidc.issuer" --output text
```

示例输出如下。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

如果没有返回输出，则您必须为集群创建 IAM OIDC 提供程序。

ii. 将以下内容复制到名为 *vpc-cni-trust-policy.json* 的文件中。将 *111122223333* 替换为您的账户 ID，将 *EXAMPLED539D4633E53DE1B71EXAMPLE* 替换为在上一步中返回的输出。将 *region-code* 替换为您的集群所在的 Amazon 区域。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.us-east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
node"
                }
            }
        }
    ]
}
```

iii. 创建角色。您可以将 *AmazonEKSVPCCNIRole* 替换为选择的任何名称。

```
aws iam create-role \
  --role-name AmazonEKSVPCCNIRole \
  --assume-role-policy-document file://"vpc-cni-trust-policy.json"
```

iv. 将所需的 IAM 策略附加到角色。运行与集群的 IP 系列匹配的命令。

- IPv4

  ```
  aws iam attach-role-policy \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy \
    --role-name AmazonEKSVPCCNIRole
  ```

- IPv6

将 *111122223333* 替换为您的账户 ID 并将 *AmazonEKS_CNI_IPv6_Policy* 替换为您的 IPv6 策略的名称。如果没有 IPv6 策略，请参阅the section called “为使用 IPv6 系列的集群创建 IAM policy”创建一个。要将 IPv6 用于您的集群，它必须满足多项要求。有关更多信息，请参阅 the section called “IPv6”。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
 \
  --role-name AmazonEKSVPCCNIRole
```

v. 运行以下命令以使用您以前创建的 IAM 角色的 ARN 对 aws-node 服务账户添加注释。将示例值替换为您自己的值。

```
kubectl annotate serviceaccount \
    -n kube-system aws-node \
    eks.amazonaws.com/role-arn=arn:aws-cn:iam::111122223333:role/
AmazonEKSVPCCNIRole
```

3. （可选）配置 Kubernetes 服务账户使用的 Amazon Security Token Service 端点类型。有关更多信息，请参阅 the section called “STS 端点”。

步骤 2：重新部署适用于 Kubernetes 的 Amazon VPC CNI 插件容器组（pod）

1. 删除并重新创建任何与服务账户关联的现有容器组（pod），以应用凭证环境变量。注释未应用于目前在没有注释的情况下运行的容器组（pod）。以下命令将删除现有的 aws-node DaemonSet 容器组（pod）并使用服务账户注释对其进行部署。

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

2. 确认容器组（pod）已全部重新启动。

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

3. 描述其中一个容器组（pod）并确保 AWS_WEB_IDENTITY_TOKEN_FILE 和 AWS_ROLE_ARN 环境变量存在。将 *cpjw7* 替换为上一步输出中返回的其中一个容器组（pod）的名称。

```
kubectl describe pod -n kube-system aws-node-cpjw7 | grep 'AWS_ROLE_ARN:\|
AWS_WEB_IDENTITY_TOKEN_FILE:'
```

示例输出如下。

```
AWS_ROLE_ARN:                 arn:aws-cn:iam::111122223333:role/AmazonEKSVPCCNIRole
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
      AWS_ROLE_ARN:                           arn:aws-cn:iam::111122223333:role/
AmazonEKSVPCCNIRole
      AWS_WEB_IDENTITY_TOKEN_FILE:            /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
```

由于容器组（pod）包含两个容器，因此会返回两组重复的结果。两个容器具有相同的值。

如果容器组（pod）正在使用 Amazon 区域级端点，之前的输出中也将返回以下行。

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

步骤 3：从节点 IAM 角色中删除 CNI 策略

如果 Amazon EKS 节点 IAM 角色当前附加有 AmazonEKS_CNI_Policy IAM (IPv4) 策略或 IPv6 策略，而且您已另行创建了 IAM 角色，已将该策略附加到该 IAM 角色，并将其分配给了 aws-node Kubernetes 服务账户，则我们建议您使用与集群的 IP 系列匹配的 Amazon CLI 命令，从节点角色中删除该策略。将 *AmazonEKSNodeRole* 替换为您的节点角色的名称。

- IPv4

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn arn:aws-
cn:iam::aws:policy/AmazonEKS_CNI_Policy
```

- IPv6

  将 *111122223333* 替换为您的账户 ID 并将 *AmazonEKS_CNI_IPv6_Policy* 替换为您的 IPv6 策略的名称。

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn arn:aws-
cn:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
```

为使用 **IPv6** 系列的集群创建 IAM policy

如果您创建了使用 IPv6 系列的集群，并且集群配置了版本 1.10.1 或更高版本的适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件，则需要创建一个可以将其分配给 IAM 角色的 IAM 策略。如果您现有的集群在创建时没有使用 IPv6 系列进行配置，则必须创建一个新集群才能使用 IPv6。有关集群使用 IPv6 的详细信息，请参阅 the section called “IPv6”。

1. 复制以下文本并将其保存到名为 vpc-cni-ipv6-policy.json 的文件。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
```

2. 创建 IAM 策略。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
 file://vpc-cni-ipv6-policy.json
```

了解 VPC CNI 模式和配置

适用于 Kubernetes 的 Amazon VPC CNI 插件为容器组（pod）提供了联网。使用下表了解有关可用网络功能的详细信息。

网络功能	了解更多
配置集群以向集群、容器组（pod）和服务分配 IPv6 地址	the section called “IPv6”
将 IPv4 源网络地址转换用于容器组（pod）	the section called “出站流量”
限制流入和流出容器组（pod）的网络流量	the section called “限制流量”
自定义节点中的辅助网络接口	the section called “自定义联网”
增加节点的 IP 地址	the section called “增加 IP 地址”
将安全组用于容器组（pod）网络流量	the section called “适用于容器组（pod）的安全组”
使用容器组（pod）的多个网络接口	the section called “多个接口”

了解如何将 IPv6 地址分配给集群、容器组（pod）和服务

适用对象：使用 Amazon EC2 实例的容器组（pod）和 Fargate 容器组（pod）

默认情况下，Kubernetes 会将 IPv4 地址分配给容器组（pod）和服务。您可以将集群配置为向其分配 IPv6 地址，而不是将 IPv4 地址分配给容器组（pod）和服务。Amazon EKS 不支持双堆栈容器组（pod）或服务，即使 Kubernetes 支持也是如此。因此，您无法将 IPv4 和 IPv6 地址同时分配给容器组（pod）和服务。

创建集群时，您可以选择要用于集群的 IP 系列。集群在创建之后无法更改系列。

有关部署 Amazon EKS IPv6 集群的教程，请参阅the section called “部署”。

以下是使用该功能的注意事项：

IPv6 功能支持

- 不支持 Windows：不支持 Windows 容器组（pod）和服务。

- 需要基于 Nitro 的 EC2 节点：您只能将 Amazon 与 IPv6 基于 Nitro 的 Amazon EC2 节点或 Fargate 节点结合使用。
- 支持 EC2 和 Fargate 节点：您可以将 IPv6 与具有 Amazon EC2 节点和 Fargate 节点的 the section called “适用于容器组（pod）的安全组” 结合使用。
- 不支持 Outposts：您不能将 IPv6 与 Amazon EKS on Amazon Outposts 结合使用。
- 不支持 FSx for Lustre：不支持 the section called “Amazon FSx for Lustre”。
- 不支持自定义联网：如果您以前使用过the section called “自定义联网”帮助缓解 IP 地址耗尽，您可以改用 IPv6。您不能将自定义联网与 IPv6 结合使用。如果您使用自定义联网进行网络隔离，则可能需要为集群继续使用自定义联网和 IPv4 系列。

IP 地址分配

- Kubernetes 服务：仅为 Kubernetes 服务分配一个 IPv6 地址。系统不会为其分配 IPv4 地址。
- 容器组（pod）：将为容器组（pod）分配一个 IPv6 地址和一个主机本地 IPv4 地址。主机本地 IPv4 地址是使用与 VPC CNI 关联的主机本地 CNI 插件分配的，并且该地址不会报告给 Kubernetes 控制面板。仅在有容器组需要与其他 Amazon VPC 或互联网中的外部 IPv4 资源通信时才会使用该地址。主机本地 IPv4 地址将（由 VPC CNI）SNAT 转换为 Worker 节点的主 ENI 的主 IPv4 地址。
- 容器组和服务：容器组和服务只接收 IPv6 地址，不接收 IPv4 地址。当容器组需要与外部 IPv4 端点通信时，它们会在节点本身上使用 NAT。这种内置的 NAT 功能无需 DNS64 和 NAT64。对于需要公共互联网访问的流量，容器组的流量将源网络地址转换为公有 IP 地址。
- 路由地址：当容器组与 VPC 外部通信时，其原始 IPv6 地址将被保留（不会转换为节点的 IPv6 地址）。此流量将直接通过互联网网关或仅出口互联网网关进行路由。
- 节点：所有节点均已分配 IPv4 和 IPv6 地址。
- Fargate 容器组（pod）：每个 Fargate 容器组（pod）都会收到来自为其部署的子网指定的 CIDR 的 IPv6 地址。运行 Fargate 容器组（pod）的底层硬件单元从分配给部署硬件单元的子网的 CIDR 获取唯一的 IPv4 和 IPv6 地址。

如何将 **IPv6** 与 EKS 结合使用。

- 创建新集群：您必须创建新集群并指定该集群要使用 IPv6 系列。您无法为从前一版本更新的集群启用 IPv6 系列。有关如何创建新集群的说明，请参阅“注意事项”。
- 使用最新的 VPC CNI：部署 Amazon VPC CNI 版本 1.10.1 或更高版本。默认情况下，会部署此版本或更高版本。部署附加组件后，您无法在不先删除集群所有节点组中的所有节点的情况下将 Amazon VPC CNI 附加组件版本降级到 1.10.1 版以下。

- 为 **IPv6** 配置 VPC CNI：如果您使用 Amazon EC2 节点，则必须使用 IP 前缀委派和 IPv6 配置 Amazon VPC CNI 附加组件。如果您在创建集群时选择 IPv6 系列，则 1.10.1 版的附加组件默认采用此配置。自行管理或 Amazon EKS 附加组件均为属于此种情况。有关 IP 前缀委派的更多信息，请参阅 the section called “增加 IP 地址”。
- 配置 **IPv4** 和 **IPv6** 地址：创建集群时，您指定的 VPC 和子网必须具有分配给您指定的 VPC 和子网的 IPv6 CIDR 块。您还必须为其分配一个 IPv4 CIDR 块。这是因为，即使您只想使用 IPv6，VPC 仍然需要 IPv4 CIDR 块才能正常工作。有关更多信息，请参阅 Amazon VPC 用户指南中的关联 IPv6 CIDR 块与 VPC。
- 自动将 IPv6 地址分配非节点：创建节点时，必须指定配置为自动分配 IPv6 地址的子网。否则，您无法部署节点。默认情况下，将禁用此配置。有关更多信息，请参阅 Amazon VPC 用户指南中的修改子网的 IPv6 寻址属性。
- 设置路由表以使用 **IPv6**：分配给子网的路由表必须有 IPv6 地址的路由。有关更多信息，请参阅 Amazon VPC 用户指南中的迁移到 IPv6。
- 为 **IPv6** 设置安全组：您的安全组必须允许 IPv6 地址。有关更多信息，请参阅 Amazon VPC 用户指南中的迁移到 IPv6。
- 设置负载均衡器：使用 Amazon 负载均衡器控制器版本 2.3.1 或更高版本，对 HTTP 应用程序（通过使用 the section called “应用程序负载均衡”）或流向 IPv6 容器组（pod）的网络流量（通过在 IP 模式下借助负载均衡器使用 the section called “网络负载均衡”，不支持实例模式）进行负载均衡。有关更多信息，请参阅 the section called “ Amazon 负载均衡器控制器”。
- 添加 **IPv6** IAM 策略：您必须将 IPv6 IAM 策略附加到节点 IAM 或 CNI IAM 角色。在两者之间，我们建议您将其附加到 CNI IAM 角色。有关更多信息，请参阅the section called “为使用 IPv6 系列的集群创建 IAM policy”和the section called “步骤 1：创建适用于 Kubernetes 的 Amazon VPC CNI 插件 IAM 角色”。
- 评估所有组件：在部署 IPv6 集群之前对集成的应用程序、Amazon EKS 附加组件和 Amazon 服务进行全面的评估。这是为了确保 IPv6 的所有功能按预期运行。

部署 Amazon EKS **IPv6** 集群和 Amazon Linux 托管节点

在本教程中，您将部署 IPv6 Amazon VPC、具有 IPv6 系列的 Amazon EKS 集群以及具有 Amazon EC2 Amazon Linux 节点的托管节点组。您无法在 IPv6 集群中部署 Amazon EC2 Windows 节点。您还可以将 Fargate 节点部署到集群，但为了简单起见，本主题中没有提供这些说明。

先决条件

开始教程前，请完成以下任务：

安装并配置创建和管理 Amazon EKS 集群所需的以下工具和资源。

- 我们建议您熟悉所有设置，并使用符合您要求的设置部署集群。有关更多信息，请参阅the section called “创建集群”、the section called “托管节点组”和本主题中的注意事项。一些设置仅在创建集群时才能启用。
- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 您正在使用的 IAM 安全主体必须具有使用 Amazon EKS IAM 角色、服务相关角色、Amazon CloudFormation、VPC 和相关资源的权限。有关更多信息，请参阅《IAM 用户指南》中的操作和编辑服务相关角色。
- 如果您使用 eksctl，请在计算机上安装版本 `0.215.0` 或更高版本。要安装或更新该工具，请参阅 `eksctl` 文档中的安装。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。如果您使用 Amazon CloudShell，则可能需要安装版本 2.12.3 或更高版本或者 1.27.160 版或更高版本的 Amazon CLI，因为 Amazon CloudShell 中安装的默认 Amazon CLI 版本可能是较早的版本。

您可以使用 eksctl 或 CLI 来部署 `IPv6` 集群。

使用 eksctl 部署 IPv6 集群

a. 创建 `ipv6-cluster.yaml` 文件。将以下命令复制到您的设备。根据需要对该命令进行以下修改，然后运行修改后的命令：
   - 将 *`my-cluster`* 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   - 请将 *`region-code`* 替换为 Amazon EKS 支持的任何 Amazon 区域。有关 Amazon 区域列表，请参阅《Amazon 一般参考》指南中的 Amazon EKS 服务端点和配额。
   - `version` 的值与您的集群版本有关。有关更多信息，请参阅 Amazon EKS 支持的版本。

- 将 *my-nodegroup* 替换为您的节点组的名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。
- 将 *t3.medium* 替换为任何 Amazon Nitro 系统实例类型。

```
cat >ipv6-cluster.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: "X.XX"

kubernetesNetworkConfig:
  ipFamily: IPv6

addons:
  - name: vpc-cni
    version: latest
  - name: coredns
    version: latest
  - name: kube-proxy
    version: latest

iam:
  withOIDC: true

managedNodeGroups:
  - name: my-nodegroup
    instanceType: t3.medium
EOF
```

b. 创建您的集群。

```
eksctl create cluster -f ipv6-cluster.yaml
```

创建集群需要几分钟时间。在看到输出的最后一行（与以下输出类似）之前，不要继续操作。

```
[...]
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

c. 确认默认容器组（pod）分配到了 IPv6 地址。

```
kubectl get pods -n kube-system -o wide
```

示例输出如下。

```
NAME                       READY   STATUS    RESTARTS   AGE     IP
                    NODE                                              NOMINATED NODE
 READINESS GATES
aws-node-rslts             1/1     Running   1          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
aws-node-t74jh             1/1     Running   0          5m32s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-cw7w2   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::                ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-tx6n8   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::1               ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
kube-proxy-btpbk           1/1     Running   0          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
kube-proxy-jjk2g           1/1     Running   0          5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

d. 确认默认服务已分配 IPv6 地址。

```
kubectl get services -n kube-system -o wide
```

示例输出如下。

```
NAME       TYPE        CLUSTER-IP          EXTERNAL-IP   PORT(S)         AGE
 SELECTOR
kube-dns   ClusterIP   fd30:3087:b6c2::a   <none>        53/UDP,53/TCP   57m   k8s-
app=kube-dns
```

e. （可选）部署示例应用程序或部署 Amazon 负载均衡器控制器和示例应用程序，对 HTTP 应用程序（通过the section called “应用程序负载均衡”）或流向 IPv6 容器组（pod）的网络流量（通过the section called “网络负载均衡”）进行负载均衡。

f. 在使用完成针对本教程而创建的集群和节点后，应使用以下命令清理创建的资源。

```
eksctl delete cluster my-cluster
```

使用 Amazon CLI 部署 IPv6 集群

> **Important**
>
> - 您必须以同一用户身份完成此程序中的所有步骤。要查看当前用户，请运行以下命令：
>
>   ```
>   aws sts get-caller-identity
>   ```
> - 您必须在同一 Shell 中完成此程序中的所有步骤。有几个步骤使用了之前步骤中设置的变量。如果在不同的 Shell 中设置变量值，则使用这些变量的步骤将无法正常运行。如果使用 Amazon CloudShell 完成以下程序，请记住，如果您在大约 20–30 分钟内没有使用键盘或指针与其交互，则 Shell 会话将结束。正在运行的进程不算作交互。
> - 这些说明是针对 Bash Shell 编写的，在其他 Shell 中可能需要调整。

将此程序步骤中的所有示例值替换为您自己的值。

a. 运行以下命令以设置在后面的步骤中使用的一些变量。将 *region-code* 替换为您想要在其中部署资源的 Amazon 区域。该值可以是 Amazon EKS 支持的任何 Amazon 区域。有关 Amazon 区域列表，请参阅《Amazon 一般参考》指南中的 Amazon EKS 服务端点和配额。将 *my-cluster* 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。将 *my-nodegroup* 替换为您的节点组的名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。请将 *111122223333* 替换为您的账户 ID。

```
export region_code=region-code
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
```

b. 创建具有公有和私有子网且符合 Amazon EKS 和 IPv6 要求的 Amazon VPC。

i. 运行以下命令设置 Amazon CloudFormation 堆栈名称的变量。您可以将 *my-eks-ipv6-vpc* 替换为选择的任何名称。

```
export vpc_stack_name=my-eks-ipv6-vpc
```

ii. 使用 Amazon CloudFormation 模板创建 IPv6 VPC。

```
aws cloudformation create-stack --region $region_code --stack-name $vpc_stack_name
 \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

堆栈需要几分钟的时间来创建。运行如下命令。在命令的输出变成 CREATE_COMPLETE 之前，请勿继续执行下一步。

```
aws cloudformation describe-stacks --region $region_code --stack-name
 $vpc_stack_name --query Stacks[].StackStatus --output text
```

iii. 检索创建的公有子网的 ID。

```
aws cloudformation describe-stacks --region $region_code --stack-name
 $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
 text
```

示例输出如下。

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE
```

iv. 为创建的公有子网启用自动分配 IPv6 地址选项。

```
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
 subnet-0a1a56c486EXAMPLE --assign-ipv6-address-on-creation
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
 subnet-099e6ca77aEXAMPLE --assign-ipv6-address-on-creation
```

v. 从部署的 Amazon CloudFormation 堆栈中检索模板创建的子网和安全组的名称，将其存储在变量中，以供后续步骤使用。

```
security_groups=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SecurityGroups`].OutputValue' --output
 text)

public_subnets=$(aws cloudformation describe-stacks --region $region_code --stack-
name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
 text)

private_subnets=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPrivate`].OutputValue' --output
 text)

subnets=${public_subnets},${private_subnets}
```

c. 创建一个集群 IAM 角色，并将所需的 Amazon EKS IAM 托管式策略附加到该角色。Amazon EKS 托管的 Kubernetes 集群会代表您调用其他Amazon服务，以管理您用于该服务的资源。

i. 运行以下命令以创建 `eks-cluster-role-trust-policy.json` 文件。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ii. 运行以下命令设置角色名称的变量。您可以将 *myAmazonEKSClusterRole* 替换为选择的任何名称。

```
export cluster_role_name=myAmazonEKSClusterRole
```

iii. 创建角色。

```
aws iam create-role --role-name $cluster_role_name --assume-role-policy-document
 file://"eks-cluster-role-trust-policy.json"
```

iv. 检索 IAM 角色的 ARN 并将其存储在变量中以供后续步骤使用。

```
CLUSTER_IAM_ROLE=$(aws iam get-role --role-name $cluster_role_name --
query="Role.Arn" --output text)
```

v. 将所需的 Amazon EKS 托管 IAM 策略附加到角色。

```
aws iam attach-role-policy --policy-arn arn:aws-cn:iam::aws:policy/
AmazonEKSClusterPolicy --role-name $cluster_role_name
```

d. 创建您的集群。

```
aws eks create-cluster --region $region_code --name $cluster_name --kubernetes-
version 1.XX \
   --role-arn $CLUSTER_IAM_ROLE --resources-vpc-config subnetIds=
$subnets,securityGroupIds=$security_groups \
   --kubernetes-network-config ipFamily=ipv6
```

i. 注意：您可能会收到一个错误，指示请求中的可用区之一没有足够容量来创建 Amazon EKS 集群。如果发生这种情况，错误输出将包含可支持新集群的可用区。再次尝试使用至少两个位于您账户中支持的可用区的子网创建集群。有关更多信息，请参阅 the section called “容量不足”。

创建集群需要几分钟时间。运行如下命令。在命令的输出变成 ACTIVE 之前，请勿继续执行下一步。

```
aws eks describe-cluster --region $region_code --name $cluster_name --query
 cluster.status
```

e. 为集群创建或更新 kubeconfig 文件，以便您可以与集群通信。

```
aws eks update-kubeconfig --region $region_code --name $cluster_name
```

默认情况下，config 文件创建在 ~/.kube 中或者新集群的配置已添加到 ~/.kube 的现有 config 文件中。

f. 创建节点 IAM 角色。

i. 运行以下命令以创建 vpc-cni-ipv6-policy.json 文件。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
```

ii. 创建 IAM 策略。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
 file://vpc-cni-ipv6-policy.json
```

iii. 运行以下命令以创建 node-role-trust-relationship.json 文件。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
```

```
    }
  ]
}
```

iv. 运行以下命令设置角色名称的变量。您可以将 *AmazonEKSNodeRole* 替换为选择的任何名称。

```
export node_role_name=AmazonEKSNodeRole
```

v. 创建 IAM 角色。

```
aws iam create-role --role-name $node_role_name --assume-role-policy-document
 file://"node-role-trust-relationship.json"
```

vi. 将 IAM 策略附加到 IAM 角色。

```
aws iam attach-role-policy --policy-arn arn:aws-cn:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy \
    --role-name $node_role_name
```

> **Important**
>
> 为简单起见，在本教程中，该策略附加到此 IAM 角色。但是，在生产集群中，我们建议将该策略附加到单独的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。

vii.将两个所需的 IAM 托管策略附加到 IAM 角色。

```
aws iam attach-role-policy --policy-arn arn:aws-cn:iam::aws:policy/
AmazonEKSWorkerNodePolicy \
  --role-name $node_role_name
aws iam attach-role-policy --policy-arn arn:aws-cn:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly \
  --role-name $node_role_name
```

viii检索 IAM 角色的 ARN 并将其存储在变量中以供后续步骤使用。

```
node_iam_role=$(aws iam get-role --role-name $node_role_name --query="Role.Arn" --
output text)
```

g. 创建托管节点组。

i. 查看在上一步中创建的子网的 ID。

```
echo $subnets
```

示例输出如下。

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE,subnet-0377963d69EXAMPLE,subnet-0c05f819
```

ii. 创建节点组。将 *0a1a56c486EXAMPLE*、*099e6ca77aEXAMPLE*、*0377963d69EXAMPLE* 和 *0c05f819d5EXAMPLE* 替换为上一步输出中返回的值。请务必在以下命令之前的输出中删除子网 ID 之间的逗号。您可以将 *t3.medium* 替换为任何 Amazon Nitro 系统实例类型。

```
aws eks create-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
    --subnets subnet-0a1a56c486EXAMPLE subnet-099e6ca77aEXAMPLE
 subnet-0377963d69EXAMPLE subnet-0c05f819d5EXAMPLE \
    --instance-types t3.medium --node-role $node_iam_role
```

创建节点组需要几分钟的时间。运行如下命令。在返回的输出为 ACTIVE 之前，请勿继续执行下一步。

```
aws eks describe-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
    --query nodegroup.status --output text
```

h. 确认默认容器组（pod）分配到了 IP 列中的 IPv6 地址。

```
kubectl get pods -n kube-system -o wide
```

示例输出如下。

```
NAME                       READY   STATUS    RESTARTS   AGE     IP
                    NODE                                                NOMINATED NODE
 READINESS GATES
aws-node-rslts             1/1     Running   1          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
aws-node-t74jh             1/1     Running   0          5m32s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

```
coredns-85d5b4454c-cw7w2   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::                ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-tx6n8   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::1               ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
kube-proxy-btpbk           1/1     Running   0          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
kube-proxy-jjk2g           1/1     Running   0          5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

i. 确认默认服务已分配 IP 列中的 IPv6 地址。

```
kubectl get services -n kube-system -o wide
```

示例输出如下。

```
NAME       TYPE        CLUSTER-IP          EXTERNAL-IP   PORT(S)         AGE
 SELECTOR
kube-dns   ClusterIP   fd30:3087:b6c2::a   <none>        53/UDP,53/TCP   57m   k8s-
app=kube-dns
```

j. （可选）部署示例应用程序或部署 Amazon 负载均衡器控制器和示例应用程序，对 HTTP 应用程序（通过the section called “应用程序负载均衡”）或流向 IPv6 容器组（pod）的网络流量（通过the section called “网络负载均衡”）进行负载均衡。

k. 在使用完成针对本教程而创建的集群和节点后，应使用以下命令清理创建的资源。删除之前，确保您没有使用本教程之外的任何资源。

i. 如果您在与之前步骤不同的 Shell 中完成此步骤，请设置之前步骤中使用的所有变量的值，然后将 example values 替换为完成之前步骤时指定的值。如果在完成之前步骤的同一 Shell 中完成此步骤，请跳至下一步。

```
export region_code=region-code
export vpc_stack_name=my-eks-ipv6-vpc
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
export node_role_name=AmazonEKSNodeRole
export cluster_role_name=myAmazonEKSClusterRole
```

ii. 删除您的节点组。

```
aws eks delete-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name
```

删除需要花费几分钟的时间。运行如下命令。如果返回任何输出，请勿继续执行下一步。

```
aws eks list-nodegroups --region $region_code --cluster-name $cluster_name --query
 nodegroups --output text
```

iii. 请删除集群。

```
aws eks delete-cluster --region $region_code --name $cluster_name
```

删除集群需要几分钟时间。在继续之前，请确保使用以下命令删除集群。

```
aws eks describe-cluster --region $region_code --name $cluster_name
```

在输出与以下输出类似之前，请勿继续执行下一步。

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
 operation: No cluster found for name: my-cluster.
```

iv. 删除您创建的 IAM 资源。如果您选择的名称与与之前步骤中使用的名称不同，则将 *AmazonEKS_CNI_IPv6_Policy* 替换为您选择的名称。

```
aws iam detach-role-policy --role-name $cluster_role_name --policy-arn arn:aws-
cn:iam::aws:policy/AmazonEKSClusterPolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws-
cn:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws-
cn:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws-
cn:iam::$account_id:policy/AmazonEKS_CNI_IPv6_Policy
aws iam delete-policy --policy-arn arn:aws-cn:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy
aws iam delete-role --role-name $cluster_role_name
aws iam delete-role --role-name $node_role_name
```

v. 删除创建 VPC 的 Amazon CloudFormation 堆栈。

```
aws cloudformation delete-stack --region $region_code --stack-name $vpc_stack_name
```

为容器组（pod）启用出站互联网访问权限

适用对象：Linux IPv4 Fargate 节点以及带有 Amazon EC2 实例的 Linux 节点

如果您使用 IPv6 系列部署集群，那么本主题中的信息不适用于您的集群，因为 IPv6 地址未进行网络转换。有关集群使用 IPv6 的详细信息，请参阅 the section called “IPv6”。

默认情况下，集群中的每个容器组（pod）都从无类别域间路由（CIDR）块中分配到一个私有 IPv4 地址，该块与容器组（pod）部署在其中的 VPC 关联。相同 VPC 中的容器组（pod）使用这些私有 IP 地址作为端点来相互通信。当容器组（pod）与不在与您 VPC 关联的 CIDR 块内的任何 IPv4 地址通信时，Amazon VPC CNI 插件（适用于 Linux 或 Windows）默认会将容器组（pod）的 IPv4 地址转换为容器组（pod）运行所在的节点的主弹性网络接口的主私有 IPv4 地址*。

> **Note**
>
> 对于 Windows 节点，还有其他详细信息需要考虑。默认情况下，适用于 Windows 的 VPC CNI 插件使用网络配置进行定义，该配置下 SNAT 不包括同一 VPC 内向目标发送的流量。这意味着内部 VPC 通信已禁用 SNAT，分配给容器组（pod）的 IP 地址可在 VPC 内路由。但是向 VPC 以外目标发送的流量会将源容器组（pod）IP 经 SNAT 处理到实例 ENI 的主 IP 地址。适用于 Windows 的此默认配置可确保容器组（pod）能像主机实例一样访问您 VPC 之外的网络。

由于此行为：

- 只有当运行互联网资源的节点分配了公有或弹性 IP 地址并且位于公有子网中时，您的容器组（pod）才能与互联网资源通信。公有子网的关联路由表具有指向互联网网关的路由。我们建议尽可能将节点部署到私有子网。
- 对于 1.8.0 之前的插件版本，如果网络或 VPC 中的资源使用 VPC 对等、中转 VPC 或 Amazon Direct Connect 连接到您的集群 VPC，则无法在辅助弹性网络接口后启动与容器组（pod）的通信。但是，您的容器组（pod）可以启动与这些资源的通信并接收其响应。

如果以下任一陈述在您的环境中成立，则使用以下命令更改默认配置。

- 网络或 VPC 中有资源使用 VPC 对等、中转 VPC 或 Amazon Direct Connect 连接到您的集群 VPC，它们需要使用 IPv4 地址启动与容器组（pod）的通信，并且您的插件版本早于 1.8.0。
- 您的容器组（pod）位于私有子网中，需要向互联网进行出站通信。子网具有通往 NAT 网关的路由。

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

> **Note**
>
> AWS_VPC_K8S_CNI_EXTERNALSNAT 和 AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS CNI 配置变量不适用于 Windows 节点。Windows 不支持禁用 SNAT。至于从 SNAT 中排除 IPv4 CIDR 列表，您可以通过在 Windows 引导脚本中指定 ExcludedSnatCIDRs 参数来定义。有关使用该参数的更多信息，请参阅 the section called “引导脚本配置参数”。

主机网络

*如果容器组（pod）规范包含 hostNetwork=true（默认值为 false），其 IP 地址就不会转换为其他地址。对于在集群上运行的 kube-proxy 和适用于 Kubernetes 的 Amazon VPC CNI 插件容器组（pod），默认情况便是如此。对于这些容器组（pod），IP 地址与节点的主 IP 地址相同，因此容器组（pod）IP 地址未转换。有关容器组（pod）hostNetwork 设置的更多信息，请参阅 Kubernetes API 参考中的 PodSpec v1 core。

通过 Kubernetes 网络策略限制容器组（pod）流量

概述

默认情况下，Kubernetes 对集群中任何容器组（pod）之间或容器组（pod）与任何其他网络中的资源之间的 IP 地址、端口或连接没有限制。可以使用 Kubernetes 网络策略来限制进出容器组（pod）的网络流量。有关更多信息，请参阅 Kubernetes 文档中的网络策略。

标准网络策略

您可以使用 NetworkPolicy 标准在集群中分段容器组（pod）间流量。这些网络策略在 OSI 网络模型的第 3 层和第 4 层上运行，让您能够控制 Amazon EKS 集群中 IP 地址或端口级别的流量。标准网络策略的范围限定为命名空间级别。

使用案例

- 在工作负载之间对网络流量进行分段，确保只有相关的应用程序才能相互通信。
- 使用策略在命名空间级别隔离租户，以强制执行网络分离。

示例

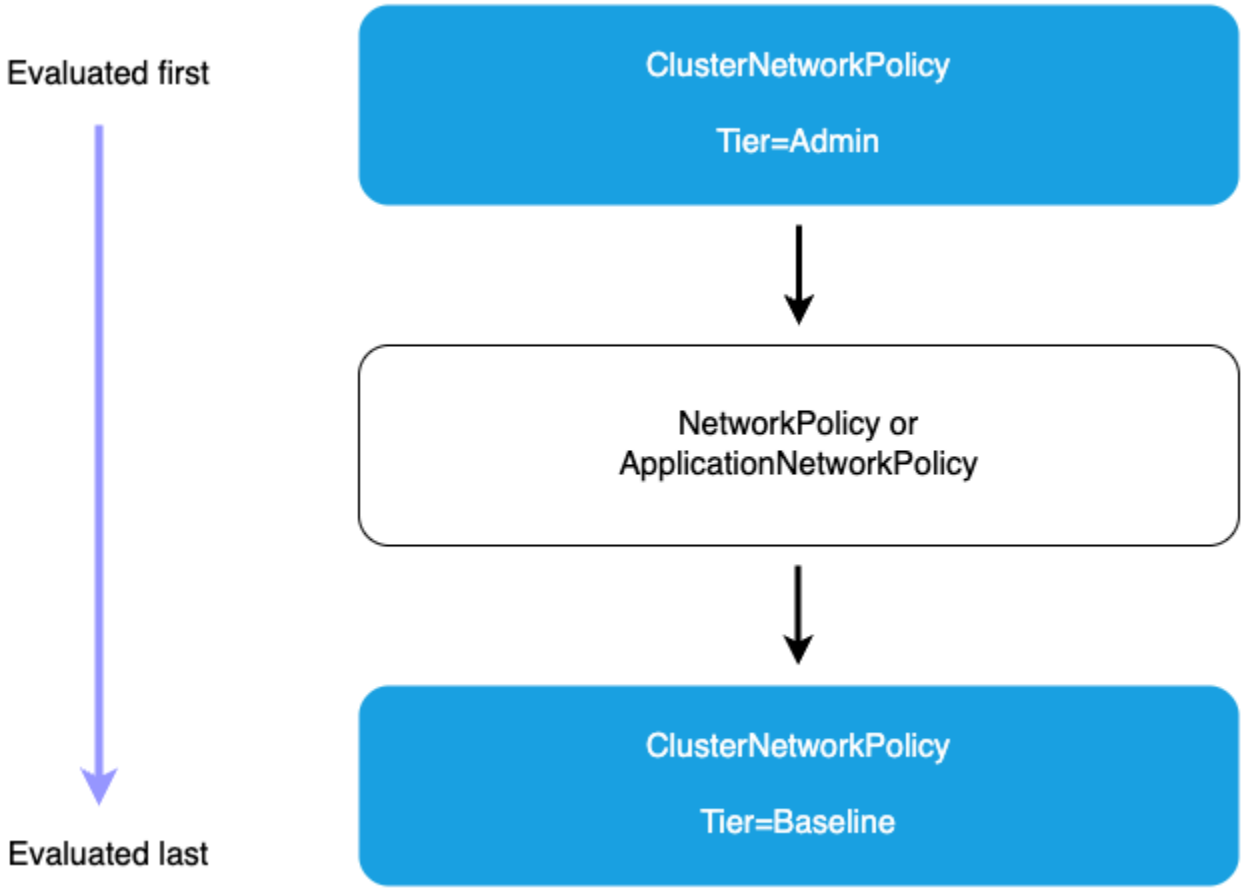
在以下策略中，来自 sun 命名空间中的 webappp容器组（pod）的出口流量受到限制。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: webapp-egress-policy
  namespace: sun
spec:
  podSelector:
    matchLabels:
      role: webapp
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: moon
      podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 8080
  - to:
    - namespaceSelector:
        matchLabels:
          name: stars
      podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 8080
```

该策略适用于 sun 命名空间中带有 role: webapp 标签的容器组（pod）。

- 允许的流量：在 TCP 端口 8080 的 moon 命名空间中带有 role: frontend 标签的容器组（pod）
- 允许的流量：在 TCP 端口 8080 的 stars 命名空间中带有 role: frontend 标签的容器组（pod）
- 被阻止的流量：所有其他来自 webapp 容器组（pod）的出站流量都被隐式拒绝

管理员（或集群）网络策略



您可以使用 ClusterNetworkPolicy 来强制执行适用于整个集群的网络安全标准。无需为每个命名空间重复定义和维护不同的策略，而是可以使用单个策略来集中管理集群中不同工作负载的网络访问权限控制，无论其命名空间如何。

使用案例

- 集中管理 EKS 集群中所有（或部分）工作负载的网络访问权限控制。
- 定义整个集群的默认网络安全状况。

- 以更具运营效率的方式，将组织安全标准扩展到集群范围。

示例

在以下策略中，您可以明确阻止来自其他命名空间的集群流量，以防止网络访问敏感的工作负载命名空间。

```
apiVersion: networking.k8s.aws/v1alpha1
kind: ClusterNetworkPolicy
metadata:
  name: protect-sensitive-workload
spec:
  tier: Admin
  priority: 10
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: earth
  ingress:
    - action: Deny
      from:
      - namespaces:
          matchLabels: {} # Match all namespaces.
      name: select-all-deny-all
```

重要提示

以下配置支持适用于 Kubernetes 的 Amazon VPC CNI 插件中的网络策略。

- 适用于标准和管理员网络策略的 Amazon VPC CNI 插件 1.21.0 版本（或更高版本）。
- 为 IPv4 或 IPv6 地址配置的集群。
- 您可以将网络策略与容器组（pod）的安全组结合使用。通过网络策略，您可以控制集群内的所有通信。借助容器组（pod）的安全组，您可以控制从容器组（pod）中的应用程序对 Amazon 服务的访问。
- 您可以将网络策略与自定义网络和前缀委派结合使用。

注意事项

架构

- 当使用适用于 Kubernetes 的 Amazon VPC CNI 插件，将其网络策略应用于集群时，您只能将策略应用于 Amazon EC2 Linux 节点。无法将策略应用于 Fargate 或 Windows 节点。
- 网络策略仅适用于其中一个 IPv4 或 IPv6 地址，但不适用于两者。在 IPv4 集群中，VPC CNI 为容器组（pod）分配 IPv4 地址并应用 IPv4 策略。在 IPv6 集群中，VPC CNI 为容器组（pod）分配 IPv6 地址并应用 IPv6 策略。系统将忽略应用于 IPv6 集群的任何 IPv4 网络策略规则。系统将忽略应用于 IPv4 集群的任何 IPv6 网络策略规则。

网络策略

- 网络策略仅适用于属于部署构成部分的容器组（pod）。不能将网络策略应用于未设置 `metadata.ownerReferences` 的独立容器组（pod）。
- 可以对同一容器组（pod）应用多个网络策略。当配置两个或多个选择同一容器组（pod）的策略时，所有策略都将应用于该容器组（pod）。
- 在所有网络策略中，单个 IP 地址范围（CIDR）的最大端口和协议组合数量为 24 个。`namespaceSelector` 等选择器会解析为一个或多个 CIDR。如果多个选择器解析为单个 CIDR，或者您在相同或不同的网络策略中多次指定了相同的直接 CIDR，这些都会计入此限制。
- 对于任何 Kubernetes 服务，服务端口必须与容器端口相同。如果您使用的是命名端口，也请在服务规范中使用相同的名称。

管理员网络策略

1. 管理员层策略（先进行评估）：首先对所有管理员层 ClusterNetworkPolicies 进行评估，然后再评估任何其他策略。在管理员层中，策略按优先级顺序处理（优先级序号最小者优先）。操作类型决定了接下来的操作。
   - Deny 操作（优先级最高）：当具有 Deny 操作的管理员策略与流量匹配时，无论采用何种其他策略，该流量都会立即被阻止。不再处理其他 ClusterNetworkPolicy 或 NetworkPolicy 规则。这样可以确保组织范围的安全控制不会被命名空间级别的策略所覆盖。
   - Allow 操作：评估 Deny 规则后，将按优先级顺序处理具有 Allow 操作的管理员策略（优先级序号最小者优先）。当 Allow 操作匹配时，流量将被接受，并且不再进行进一步的策略评估。这些策略可以根据标签选择器授予跨多个命名空间的访问权限，从而集中控制哪些工作负载可以访问特定资源。
   - Pass 操作：管理员层策略中的 Pass 操作将决策委托给较低层级。当流量与 Pass 规则匹配时，评估将跳过该流量的所有剩余管理层规则，直接进入 NetworkPolicy 层。这允许管理员将某些流量模式的控制权明确委托给应用程序团队。例如，您可以使用 Pass 规则将命名空间内的流量管理委托给命名空间管理员，同时保持对外部访问权限的严格控制。

2. 网络策略层：如果没有与 Deny 或 Allow 匹配的管理员层策略，或者匹配了 Pass 操作，则接下来将评估命名空间范围的 NetworkPolicy 资源。这些策略在各个命名空间内提供精细的控制，并由应用程序团队管理。命名空间范围的策略只能比管理员策略更严格。它们无法覆盖管理员策略的 Deny 决策，但可以进一步限制管理员策略允许或通过的流量。
3. 基准层管理员策略：如果没有管理员或命名空间范围的策略与流量匹配，则会评估基准层 ClusterNetworkPolicies。它们提供了可以被命名空间范围的策略覆盖的默认安全状况，允许管理员设置组织范围的默认值，同时让团队可以灵活地根据需要进行自定义。基准策略按优先级顺序进行评估（优先级序号最小者优先）。
4. 默认拒绝（如果没有匹配的策略）：这种默认拒绝行为可确保仅允许明确允许的连接，从而保持稳固的安全状况。

迁移

- 如果集群当前使用第三方解决方案来管理 Kubernetes 网络策略，则可将这些策略与适用于 Kubernetes 的 Amazon VPC CNI 插件结合使用。但是，您必须删除现有的解决方案，使其不再管理相同的策略。

> **Warning**
>
> 建议您在删除网络策略解决方案后，替换所有应用过该网络策略解决方案的节点。这是因为如果解决方案的某个容器组（pod）突然退出，其流量规则可能会遗留下来。

安装。

- 网络策略功能创建并需要一个名为 `policyendpoints.networking.k8s.aws` 的 `PolicyEndpoint` 自定义资源定义（CRD）。自定义资源的 `PolicyEndpoint` 对象由 Amazon EKS 管理。您不应该修改或删除这些资源。
- 如果您运行的容器组（pod）使用实例角色 IAM 凭证或连接到 EC2 IMDS，请注意检查是否存在会阻止访问 EC2 IMDS 的网络策略。您可能需要添加网络策略以允许访问 EC2 IMDS。有关更多信息，请参阅 Amazon EC2 用户指南中的实例元数据和用户数据。

  对服务账户使用 IAM 角色或使用 EKS 容器组身份的容器组（pod）无法访问 EC2 IMDS。
- 适用于 Kubernetes 的 Amazon VPC CNI 插件不会将网络策略应用于每个容器组（pod）的其他网络接口，只会应用于每个容器组（pod）的主接口 (`eth0`)。这会影响以下架构：

- ENABLE_V4_EGRESS 变量设置为 true 的 IPv6 容器组。此变量使 IPv4 出口功能能够将 IPv6 容器组连接到 IPv4 端点，例如集群外部的端点。IPv4 出口功能的工作原理是使用本地环回 IPv4 地址创建一个额外的网络接口。
- 使用诸如 Multus 之类的链式网络插件时。由于这些插件为每个容器组（pod）添加了网络接口，因此网络策略不适用于链式网络插件。

使用 Kubernetes 网络策略限制容器组（pod）网络流量

您可以使用 Kubernetes 网络策略来限制进出容器组（pod）的网络流量。有关更多信息，请参阅 Kubernetes 文档中的网络策略。

要使用此功能，您必须配置以下内容：

1. 在容器组（pod）启动时设置策略实施。您可以在 VPC CNI DaemonSet 的 aws-node 容器中执行此操作。
2. 为附加组件启用网络策略参数。
3. 将集群配置为使用 Kubernetes 网络策略

开始之前，请查看注意事项。有关更多信息，请参阅 the section called “注意事项”。

先决条件

以下是使用此功能的先决条件：

最低集群版本

现有 Amazon EKS 集群。要部署一个角色，请参阅开始使用。该集群当前运行的必须是下表中列出的 Kubernetes 版本和平台版本之一。请注意，也支持所有比所列版本更高的 Kubernetes 和平台版本。您可以通过将以下命令中的 *my-cluster* 替换为集群名称，然后运行修改后的命令来检查当前的 Kubernetes 版本：

```
aws eks describe-cluster --name my-cluster --query cluster.version --output text
```

Kubernetes 版本	平台版本
1.27.4	eks.5

Kubernetes 版本	平台版本
1.26.7	eks.6

最低 Amazon CNI 版本

要同时创建标准 Kubernetes 网络策略和管理员网络策略，您需要运行 1.21 版的 VPC CNI 插件。您可以使用以下命令查看当前使用的版本。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

如果您的版本低于 1.21，请查看 the section called “更新（EKS 附加组件）” 将版本升级到 1.21 版本或更高版本。

最低 Amazon 内核版本

您的节点必须具有 Linux 内核版本 5.10 或更高版本。您可以使用 uname -r 检查您的内核版本。如果您使用的是最新版本的 Amazon EKS 优化 Amazon Linux、Amazon EKS 优化版加速型 Amazon Linux AMI 和 Bottlerocket AMI，则已拥有所需的内核版本。

Amazon EKS 优化加速的 Amazon Linux AMI 版本 v20231116，或更高版本拥有内核版本 5.10。

第 1 步：在容器组（pod）启动时设置策略实施

适用于 Kubernetes 的 Amazon VPC CNI 插件为容器组（pod）配置网络策略，同时进行容器组（pod）预置。在为新的容器组（pod）配置所有策略之前，新的容器组（pod）中的容器将从默认允许策略开始。这称为标准模式。默认允许策略意味着允许所有进出新容器组（pod）的入口和出口流量。例如，在使用生效的策略更新新的容器组之前，容器组不会强制执行任何防火墙规则（允许所有流量）。

将 NETWORK_POLICY_ENFORCING_MODE 变量设置为 strict 后，使用 VPC CNI 的容器组（pod）从默认拒绝策略开始，然后将配置策略。这称为严格模式。在严格模式下，您必须为集群中容器组（pod）需要访问的每个端点制定网络策略。请注意，此要求适用于 CoreDNS 容器组（pod）。未为带有主机网络的容器组（pod）配置默认拒绝策略。

您可以通过在 VPC CNI DaemonSet 的 aws-node 容器中将环境变量 NETWORK_POLICY_ENFORCING_MODE 设置为 strict 来更改此默认网络策略。

```
env:
  - name: NETWORK_POLICY_ENFORCING_MODE
    value: "strict"
```

第 2 步：为附加组件启用网络策略参数

默认情况下，网络策略功能使用节点上的端口 8162 获取指标。此外，该功能还使用端口 8163 进行运行状况探测。如果您在需要使用这些端口的节点或 Pod 内运行其他应用程序，则该应用程序将无法运行。在 VPC CNI v1.14.1 版或更高版本中，您可以更改这些端口。

要为附加组件启用网络策略参数，请使用以下程序。

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群，然后选择要为其配置 Amazon VPC CNI 附加组件的集群名称。
3. 选择附加组件选项卡。
4. 选择附加组件框右上角的框，然后选择 Edit（编辑）。
5. 在配置 **Amazon VPC CNI** 页面上：
   a. 在版本列表中选择 v1.14.0-eksbuild.3 或更高版本。
   b. 展开可选配置设置。
   c. 在配置值中输入 JSON 键 "enableNetworkPolicy": 和值 "true"。生成的文本必须是有效的 JSON 对象。如果此键和值是文本框中的唯一数据，请用大括号 { } 将键和值括起来。

      以下示例启用了网络策略功能，并将指标和运行状况探测设置为默认端口号：

      ```
      {
          "enableNetworkPolicy": "true",
          "nodeAgent": {
              "healthProbeBindAddr": "8163",
              "metricsBindAddr": "8162"
          }
      }
      ```

Helm

如果已通过 helm 安装适用于 Kubernetes 的 Amazon VPC CNI 插件，则可以更新配置以更改端口。

1. 运行以下命令以更改端口。分别在键 nodeAgent.metricsBindAddr 或键 nodeAgent.healthProbeBindAddr 的值中设置端口号。

```
helm upgrade --set nodeAgent.metricsBindAddr=8162 --set
 nodeAgent.healthProbeBindAddr=8163 aws-vpc-cni --namespace kube-system eks/aws-vpc-
cni
```

kubectl

1. 在编辑器中打开 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

2. 在 VPC CNI aws-node 守护程序集清单中 aws-network-policy-agent 容器的 args: 中，替换以下命令参数中的端口号。

```
    - args:
            - --metrics-bind-addr=:8162
            - --health-probe-bind-addr=:8163
```

步骤 3：将集群配置为使用 Kubernetes 网络策略

您可以为 Amazon EKS 附加组件或自行管理的附加组件设置此项。

Amazon EKS 附加组件

使用 Amazon CLI，您可以通过运行以下命令将集群配置为使用 Kubernetes 网络策略。将 my-cluster 替换为集群的名称，并将 IAM 角色 ARN 替换为您正在使用的角色。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws-cn:iam::123456789012:role/AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"enableNetworkPolicy":
 "true"}'
```

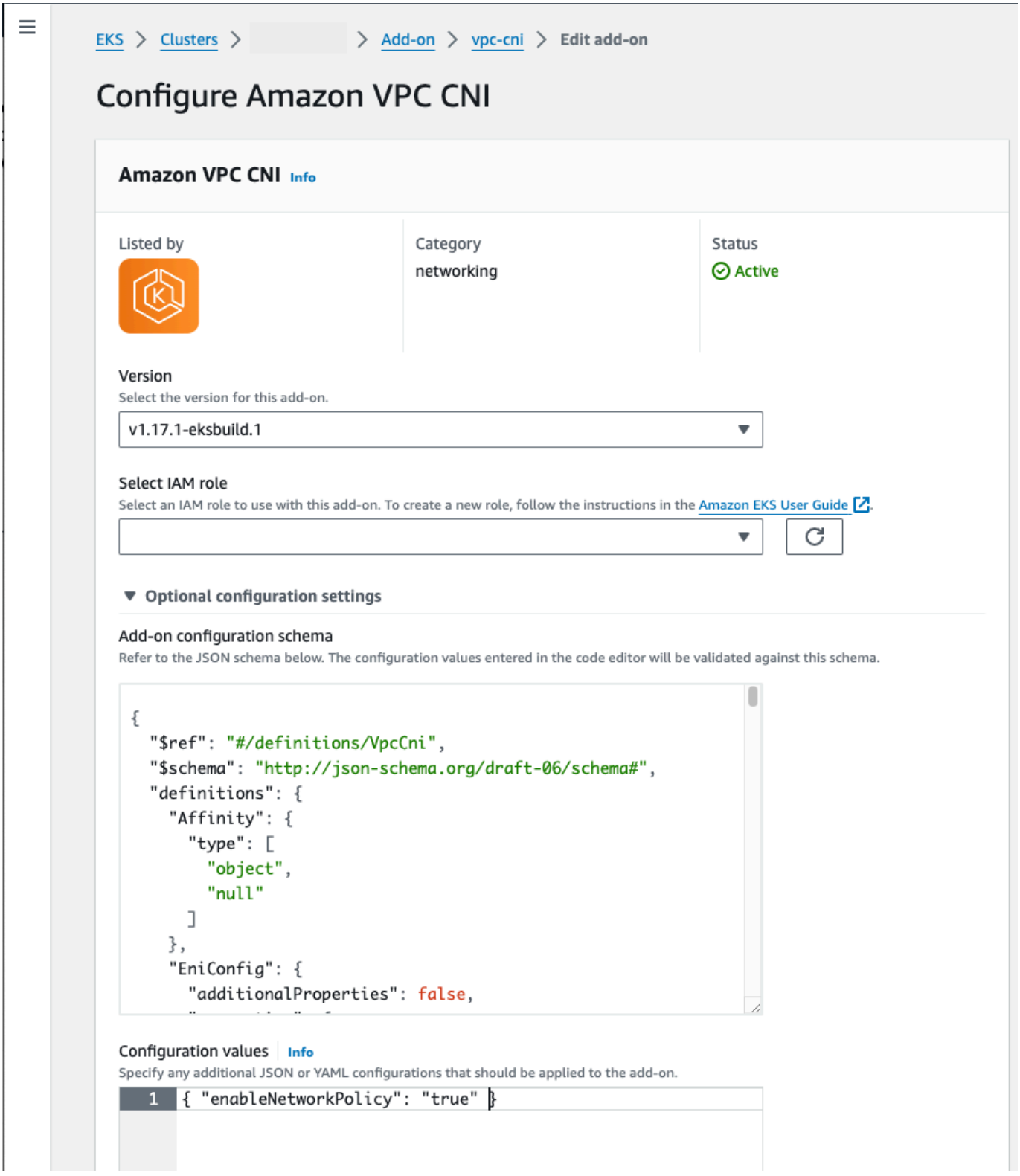
要使用 Amazon 管理控制台配置此项，请按以下步骤操作：

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群，然后选择要为其配置 Amazon VPC CNI 附加组件的集群名称。

3. 选择附加组件选项卡。
4. 选择附加组件框右上角的框，然后选择 Edit（编辑）。
5. 在配置 **Amazon VPC CNI** 页面上：
   a. 在版本列表中选择 v1.14.0-eksbuild.3 或更高版本。
   b. 展开可选配置设置。
   c. 在配置值中输入 JSON 键 "enableNetworkPolicy": 和值 "true"。生成的文本必须是有效的 JSON 对象。如果此键和值是文本框中的唯一数据，请用大括号 { } 将键和值括起来。以下示例显示网络策略已启用：

```
{ "enableNetworkPolicy": "true" }
```

   下面的屏幕截图为此场景的一个示例。


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 { "enableNetworkPolicy": "true" }

自主管理的附加组件

Helm

如果已通过 helm 安装适用于 Kubernetes 的 Amazon VPC CNI 插件，则可以更新配置以启用网络策略。

1. 运行以下命令以启用网络策略。

```
helm upgrade --set enableNetworkPolicy=true aws-vpc-cni --namespace kube-system eks/
aws-vpc-cni
```

kubectl

1. 在编辑器中打开 amazon-vpc-cni ConfigMap。

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

2. 将以下行添加到 ConfigMap 中的 data。

```
enable-network-policy-controller: "true"
```

   添加该行后，您的 ConfigMap 应该看起来像下面的示例。

```
apiVersion: v1
 kind: ConfigMap
 metadata:
  name: amazon-vpc-cni
  namespace: kube-system
 data:
  enable-network-policy-controller: "true"
```

3. 在编辑器中打开 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

   a. 在 VPC CNI aws-node 守护程序集清单中 aws-network-policy-agent 容器的 args: 中，将命令参数 --enable-network-policy=false 中的 false 替换为 true。

```
    - args:
       - --enable-network-policy=true
```

步骤 4：后续步骤

完成配置后，确认 aws-node 容器组（pod）正在您的集群上运行。

```
kubectl get pods -n kube-system | grep 'aws-node\|amazon'
```

示例输出如下。

```
aws-node-gmqp7                                          2/2     Running   1 (24h ago)
 24h
aws-node-prnsh                                          2/2     Running   1 (24h ago)
 24h
```

版本 1.14 及更高版本的 aws-node 容器组中有 2 个容器。在以前的版本中，如果禁用网络策略，则 aws-node 容器组（pod）中只有 1 个容器。

您现在可以将 Kubernetes 网络策略部署到集群。

要实施 Kubernetes 网络策略，您可以创建 Kubernetes NetworkPolicy 或 ClusterNetworkPolicy 对象并将其部署到集群。NetworkPolicy 对象的范围限定为命名空间，而 ClusterNetworkPolicy 对象的范围可限定为整个集群或多个命名空间。您可以实施策略，根据标签选择器、命名空间和 IP 地址范围来允许或拒绝容器组（pod）之间的流量。有关创建 NetworkPolicy 对象的更多信息，请参阅 Kubernetes 文档中的网络策略。

Kubernetes NetworkPolicy 对象的执行是使用 Extended Berkeley Packet Filter（eBPF）实施的。与基于 iptables 的实施相比，它具有更低的延迟和性能，包括降低 CPU 利用率和避免顺序查找。此外，eBPF 探测器还可以访问上下文丰富的数据，帮助调试复杂的内核级问题并提高可观测性。Amazon EKS 支持基于 eBPF 的导出程序，该导出程序利用探测器记录每个节点上的策略结果，并将数据导出到外部日志收集器以帮助调试。有关更多信息，请参阅 eBPF 文档。

禁用 Amazon EKS 容器组（pod）网络流量的 Kubernetes 网络策略

禁用 Kubernetes 网络策略，停止限制 Amazon EKS 容器组（pod）网络流量

1. 列出所有 Kubernetes 网络策略。

```
kubectl get netpol -A
```

2. 删除所有 Kubernetes 网络策略。必须首先删除所有网络策略，然后才能禁用网络策略。

```
kubectl delete netpol <policy-name>
```

3. 在编辑器中打开 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

4. 在 VPC CNI `aws-node` 守护程序集清单中 `aws-network-policy-agent` 容器的 `args:` 中，将命令参数 `--enable-network-policy=true` 中的 `true` 替换为 `false`。

```
    - args:
       - --enable-network-policy=true
```

排查 Amazon EKS 的 Kubernetes 网络策略问题

这是 Amazon VPC CNI 网络策略功能的故障排除指南。

本指南涵盖以下内容：

- 安装信息、CRD 和 RBAC 权限（the section called “新的 `policyendpoints` CRD 和权限”）
- 诊断网络策略问题时要检查的日志（the section called “网络策略日志”）
- 运行 eBPF SDK 工具集进行故障排除
- 已知问题和解决方案 the section called “已知问题和解决方案”

> **Note**
>
> 请注意，网络策略仅适用于通过 Kubernetes 部署创建的容器组。有关 VPC CNI 中网络策略的更多限制，请参阅the section called “注意事项”。

您可以通过查看网络策略日志和运行 eBPF SDK 中的工具，对使用网络策略的网络连接进行问题排查和调查。

新的 **policyendpoints** CRD 和权限

- CRD：policyendpoints.networking.k8s.aws
- Kubernetes API：名为 v1.networking.k8s.io 的 apiservice
- Kubernetes 资源：Kind: NetworkPolicy
- RBAC：名为 aws-node 的 ClusterRole（VPC CNI），名为 eks:network-policy-controller 的 ClusterRole（EKS 集群控制面板中的网络策略控制器）

对于网络策略，VPC CNI 会创建一个名为 policyendpoints.networking.k8s.aws 的新 CustomResourceDefinition（CRD）。VPC CNI 必须有相关权限，以创建 CRD 以及创建由 VPC CNI（eniconfigs.crd.k8s.amazonaws.com）安装的此 CRD 和另一个 CRD 的 CustomResources（CR）。这两个 CRD 都在 GitHub 上的 crds.yaml 文件中提供。具体而言，VPC CNI 必须具有对 policyendpoints 执行“获取”、“列出”和“监控”操作的权限。

Kubernetes 网络策略是名为 v1.networking.k8s.io 的 apiservice 的一部分，这是您的策略 YAML 文件中的 apiversion: networking.k8s.io/v1。VPC CNI DaemonSet 必须有使用 Kubernetes API 这一部分的权限。

VPC CNI 权限位于名为 aws-node 的 ClusterRole 中。请注意，ClusterRole 对象未按命名空间分组。以下显示了集群的 aws-node：

```
kubectl get clusterrole aws-node -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/instance: aws-vpc-cni
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: aws-node
    app.kubernetes.io/version: v1.19.4
    helm.sh/chart: aws-vpc-cni-1.19.4
    k8s-app: aws-node
  name: aws-node
rules:
- apiGroups:
  - crd.k8s.amazonaws.com
  resources:
  - eniconfigs
```

```
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  - events.k8s.io
  resources:
  - events
  verbs:
  - create
  - patch
  - list
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints
  verbs:
  - get
  - list
```

```
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints/status
  verbs:
  - get
- apiGroups:
  - vpcresources.k8s.aws
  resources:
  - cninodes
  verbs:
  - get
  - list
  - watch
  - patch
```

此外，每个 EKS 集群的控制面板中都会运行一个新的控制器。控制器使用名为 eks:network-policy-controller 的 ClusterRole 的权限。以下显示了集群的 eks:network-policy-controller：

```
kubectl get clusterrole eks:network-policy-controller -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/name: amazon-network-policy-controller-k8s
  name: eks:network-policy-controller
rules:
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - pods
```

```
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints/finalizers
  verbs:
  - update
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints/status
  verbs:
  - get
  - patch
  - update
- apiGroups:
  - networking.k8s.io
  resources:
  - networkpolicies
  verbs:
  - get
```

```
  - list
  - patch
  - update
  - watch
```

网络策略日志

VPC CNI 根据网络策略允许还是拒绝连接的每个决定都将记录在流日志中。每个节点上的网络策略日志包括每个具有网络策略的容器组（pod）流日志。网络策略日志存储在 /var/log/aws-routed-eni/network-policy-agent.log。以下示例来自 network-policy-agent.log 文件：

```
{"level":"info","timestamp":"2023-05-30T16:05:32.573Z","logger":"ebpf-
client","msg":"Flow Info: ","Src
IP":"192.168.87.155","Src Port":38971,"Dest IP":"64.6.160","Dest
Port":53,"Proto":"UDP","Verdict":"ACCEPT"}
```

默认情况下，网络策略日志被禁用。要启用网络策略日志，请按照下列步骤操作：

> **Note**
>
> 网络策略日志需要为 VPC CNI aws-node DaemonSet 清单中的 aws-network-policy-agent 容器额外提供 1 个 vCPU。

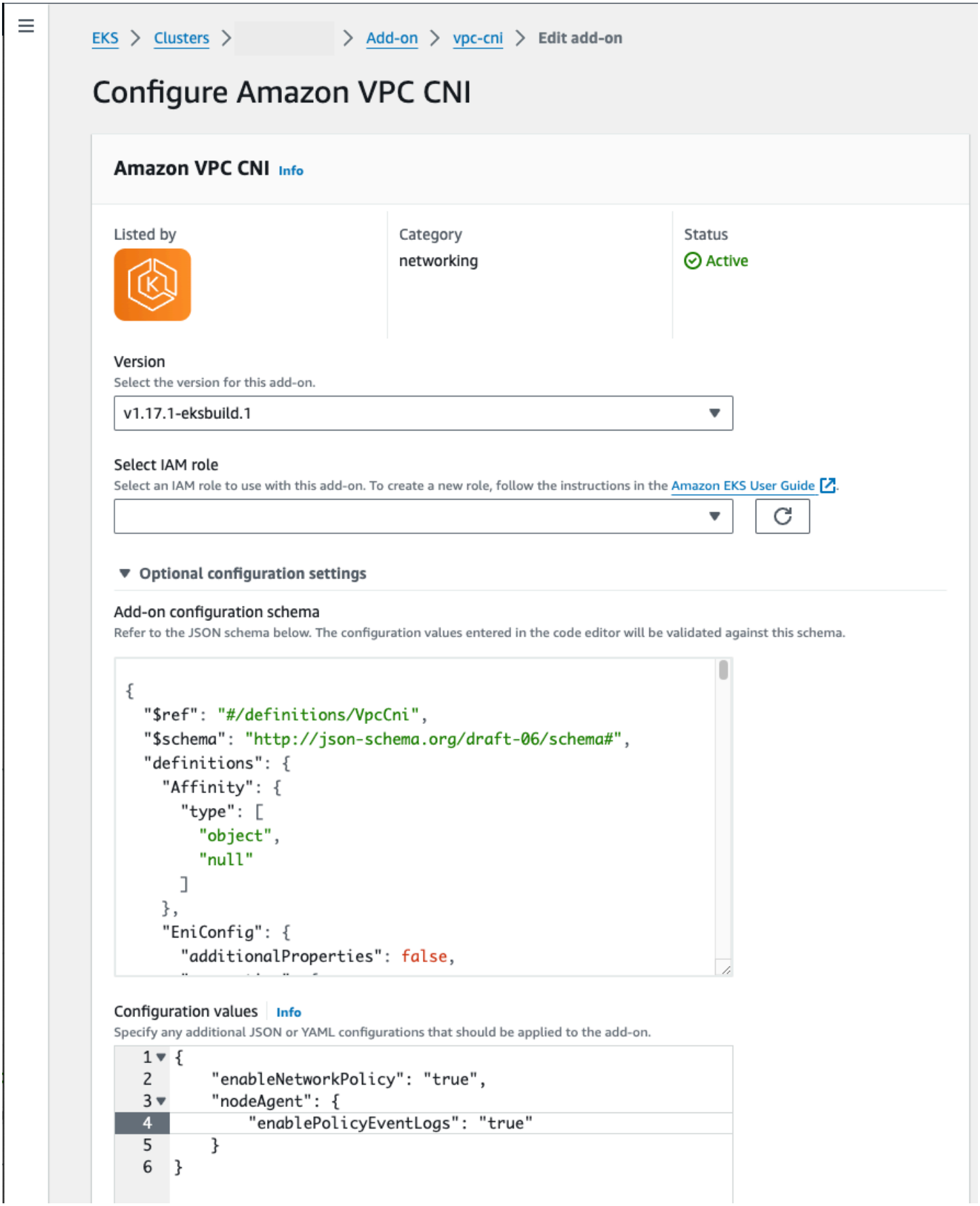
Amazon EKS 附加组件

Amazon Web Services 管理控制台

a. 打开 Amazon EKS 控制台。
b. 在左侧导航窗格中，选择集群，然后选择要为其配置 Amazon VPC CNI 附加组件的集群名称。
c. 选择附加组件选项卡。
d. 选择附加组件框右上角的框，然后选择 Edit（编辑）。
e. 在配置 ***Amazon VPC CNI*** 页面上：
   i. 在版本下拉列表中选择 v1.14.0-eksbuild.3 或更高版本。
   ii. 展开可选配置设置。
   iii. 输入顶级 JSON 键 "nodeAgent":，值是一个在配置值中键为 "enablePolicyEventLogs": 且值为 "true" 的对象。生成的文本必须是有效的 JSON 对象。以下示例显示网络策略和网络策略日志已启用，并将网络策略日志发送到 CloudWatch Logs：

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "enablePolicyEventLogs": "true"
    }
}
```

下面的屏幕截图为此场景的一个示例。


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 {
2 "enableNetworkPolicy": "true",
3 "nodeAgent": {
4 "enablePolicyEventLogs": "true"
5 }
6 }

Amazon CLI

a. 运行以下 Amazon CLI 命令。将 `my-cluster` 替换为集群的名称，并将 IAM 角色 ARN 替换为您正在使用的角色。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-
version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws-cn:iam::123456789012:role/
AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent":
 {"enablePolicyEventLogs": "true"}}'
```

自主管理的附加组件

Helm

如果已通过 `helm` 安装适用于 Kubernetes 的 Amazon VPC CNI 插件，则可以更新配置来编写网络策略日志。

a. 运行以下命令以启用网络策略。

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true aws-vpc-cni --namespace
 kube-system eks/aws-vpc-cni
```

kubectl

如果已通过 `kubectl` 安装适用于 Kubernetes 的 Amazon VPC CNI 插件，则可以更新配置来编写网络策略日志。

a. 在编辑器中打开 `aws-node DaemonSet`。

```
kubectl edit daemonset -n kube-system aws-node
```

b. 在 VPC CNI `aws-node DaemonSet` 清单中 `aws-network-policy-agent` 容器的 `args:` 中，将命令参数 `--enable-policy-event-logs=false` 中的 `false` 替换为 `true`。

```
    - args:
       - --enable-policy-event-logs=true
```

将网络策略日志发送到 Amazon CloudWatch Logs

您可以使用 Amazon CloudWatch Logs 等服务监控网络策略日志。您可以使用以下方法将网络策略日志发送到 CloudWatch Logs。

对于 EKS 集群，策略日志将放在 /aws/eks/*cluster-name*/cluster/ 下；对于自主管理型 K8S 集群，日志将放在 /aws/k8s-cluster/cluster/ 下。

使用适用于 Kubernetes 的 Amazon VPC CNI 插件发送网络策略日志

如果启用网络策略，则会将第二个容器添加到节点代理的 aws-node 容器组（pod）。此节点代理可将网络策略日志发送到 CloudWatch Logs。

> **Note**
>
> 节点代理仅发送网络策略日志。不包括 VPC CNI 生成的其它日志。

先决条件

- 将以下权限作为节或单独的策略添加到您用于 VPC CNI 的 IAM 角色中。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon EKS 附加组件

Amazon Web Services 管理控制台

a. 打开 Amazon EKS 控制台。

b. 在左侧导航窗格中，选择集群，然后选择要为其配置 Amazon VPC CNI 附加组件的集群名称。

c. 选择附加组件选项卡。

d. 选择附加组件框右上角的框，然后选择 Edit（编辑）。

e. 在配置 ***Amazon VPC CNI*** 页面上：

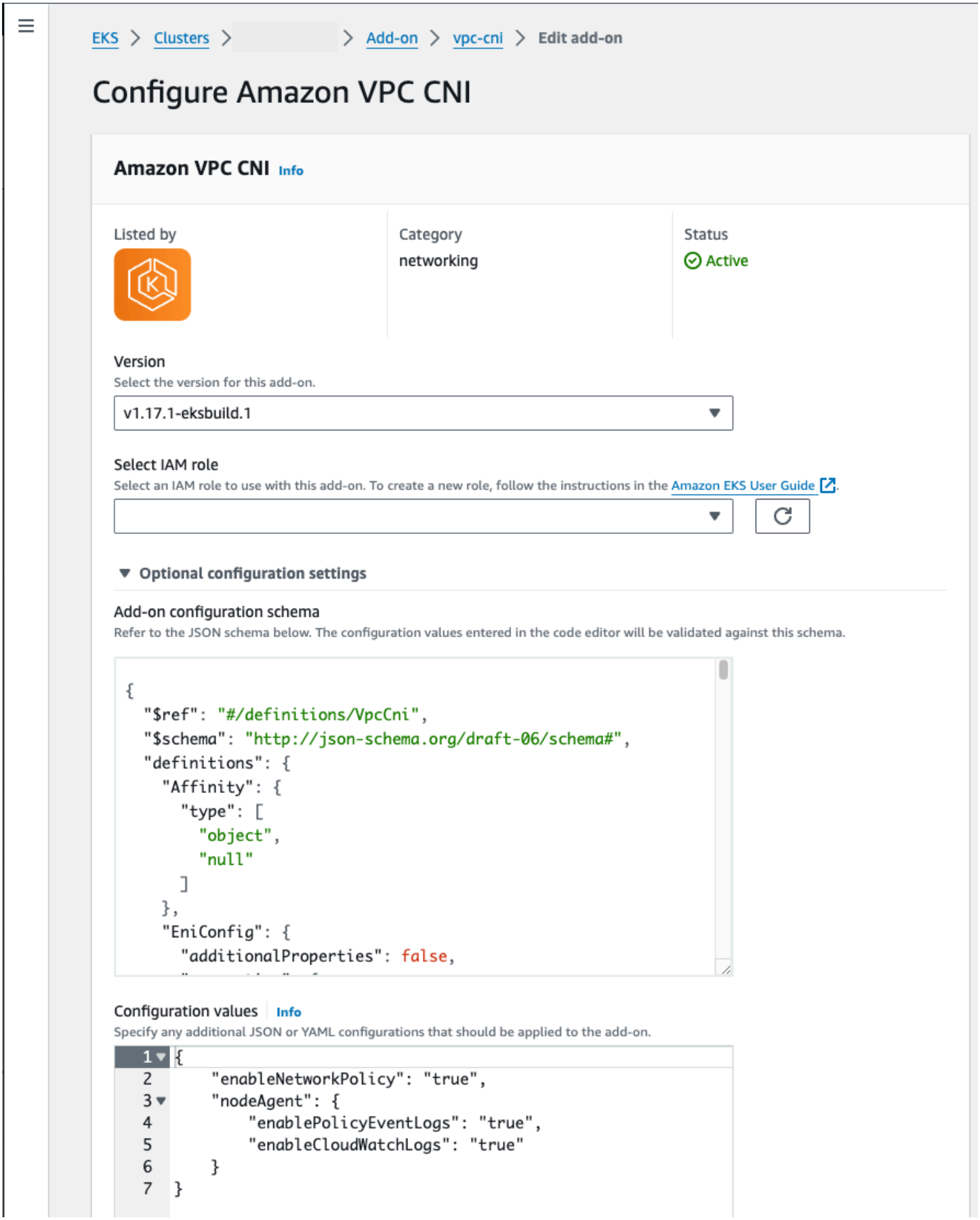
i. 在版本下拉列表中选择 v1.14.0-eksbuild.3 或更高版本。

ii. 展开可选配置设置。

iii. 输入顶级 JSON 键 "nodeAgent":，值是一个在配置值中键为 "enableCloudWatchLogs": 且值为 "true" 的对象。生成的文本必须是有效的 JSON 对象。以下示例显示网络策略和网络策略日志已启用，并将日志发送到 CloudWatch Logs：

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "enablePolicyEventLogs": "true",
        "enableCloudWatchLogs": "true",
    }
}
```

下面的屏幕截图为此场景的一个示例。


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 {
2 "enableNetworkPolicy": "true",
3 "nodeAgent": {
4 "enablePolicyEventLogs": "true",
5 "enableCloudWatchLogs": "true"
6 }
7 }

AmazonCLI

a. 运行以下 Amazon CLI 命令。将 my-cluster 替换为集群的名称，并将 IAM 角色 ARN 替换为您正在使用的角色。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-
version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws-cn:iam::123456789012:role/
AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent":
 {"enablePolicyEventLogs": "true", "enableCloudWatchLogs": "true"}}'
```

自主管理的附加组件

Helm

如果已通过 helm 安装适用于 Kubernetes 的 Amazon VPC CNI 插件，则可以更新配置，将网络策略日志发送至 CloudWatch 日志。

a. 运行以下命令以启用网络策略日志并将它们发送到 CloudWatch Logs。

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true --set
 nodeAgent.enableCloudWatchLogs=true aws-vpc-cni --namespace kube-system eks/aws-
vpc-cni
```

kubectl

a. 在编辑器中打开 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

b. 在 VPC CNI aws-node DaemonSet 清单中 aws-network-policy-agent 容器的 args: 中，将两个命令参数 --enable-policy-event-logs=false 和 --enable-cloudwatch-logs=false 中的 false 替换为 true。

```
    - args:
       - --enable-policy-event-logs=true
       - --enable-cloudwatch-logs=true
```

使用 Fluent Bit **DaemonSet** 发送网络策略日志

如果在 DaemonSet 中使用 Fluent Bit 从节点发送日志，则可以添加配置，纳入来自网络策略的网络策略日志。您可以使用以下示例配置：

```
    [INPUT]
        Name              tail
        Tag               eksnp.*
        Path              /var/log/aws-routed-eni/network-policy-agent*.log
        Parser            json
        DB                /var/log/aws-routed-eni/flb_npagent.db
        Mem_Buf_Limit     5MB
        Skip_Long_Lines   On
        Refresh_Interval  10
```

包含的 eBPF SDK

适用于 Kubernetes 的 Amazon VPC CNI 插件在节点上安装 eBPF SDK 工具集。您可以使用 eBPF SDK 工具来识别网络策略问题。例如，以下命令列出了节点上正在运行的程序。

```
sudo /opt/cni/bin/aws-eks-na-cli ebpf progs
```

要运行此命令，您可以使用任何方法连接到节点。

已知问题和解决方案

以下各节介绍了 Amazon VPC CNI 网络策略功能的已知问题及其解决方案。

尽管 enable-policy-event-logs 设置为 false，但仍会生成网络策略日志

问题：即使将 `enable-policy-event-logs` 设置为 `false`，EKS VPC CNI 仍会生成网络策略日志。

解决方案：`enable-policy-event-logs` 设置仅禁用策略“决策”日志，但不会禁用所有网络策略代理日志记录。此行为已在 GitHub 上的 aws-network-policy-agent README 中记录。要完全禁用日志记录，您可能需要调整其他日志记录配置。

网络策略映射清理问题

问题：删除容器组后，网络 `policyendpoint` 仍存在且未被清理。

解决方案：此问题是由于 VPC CNI 附加组件版本 1.19.3-eksbuild.1 的问题引起的。更新为较新版本的 VPC CNI 附加组件即可解决此问题。

网络策略未应用

问题：网络策略功能在 Amazon VPC CNI 附加组件中已启用，但未正确应用网络策略。

如果您制定了网络策略 `kind: NetworkPolicy`，但其对容器组没有影响，则请检查 policyendpoint 对象是否是在与容器组相同的命名空间中创建的。如果命名空间中没有 `policyendpoint` 对象，则网络策略控制器（EKS 集群的一部分）无法为网络策略代理（VPC CNI 的一部分）创建要应用的网络策略规则。

解决方案：解决方案是修复 VPC CNI（`ClusterRole`：`aws-node`）和网络策略控制器（`ClusterRole`：`eks:network-policy-controller`）的权限，并允许在任何策略执行工具（例如 Kyverno）中执行这些操作。确保 Kyverno 策略不会阻止创建 `policyendpoint` 对象。有关the section called “新的 policyendpoints CRD 和权限”中所需的权限，请参阅上一节。

在严格模式下删除策略后，容器组不会恢复为默认拒绝状态

问题：在严格模式下启用网络策略时，容器组以默认拒绝策略开头。应用策略后，允许流量到达指定的端点。但是，删除策略后，容器组不会恢复为默认拒绝状态，而是进入默认允许状态。

解决方案：此问题在 VPC CNI 版本 1.19.3（包含网络策略代理 1.2.0 版本）中已修复。修复后，启用严格模式时，移除策略后，容器组将按预期恢复为默认拒绝状态。

容器组的安全组启动延迟

问题：在 EKS 中使用容器组的安全组功能时，容器组启动延迟会增加。

解决方案：延迟是由于资源控制器对 `CreateNetworkInterface` API 的速率限制（API 节流）造成的，VPC 资源控制器使用其为容器组创建分支 ENI。请查看您的账户针对此操作的 API 限制，并在需要时考虑申请提高限制。

由于 vpc.amazonaws.com/pod-eni 不足导致 FailedScheduling

问题：容器组无法调度，错误信息：`FailedScheduling 2m53s (x28 over 137m) default-scheduler 0/5 nodes are available: 5 Insufficient vpc.amazonaws.com/pod-eni. preemption: 0/5 nodes are available: 5 No preemption victims found for incoming pod.`

解决方案：与上一个问题类似，为容器组分配安全组会增加容器组调度延迟，并且添加每个 ENI 的时间可能会超出 CNI 阈值，从而导致容器组启动失败。这是使用容器组的安全组时的预期行为。在设计工作负载架构时，请考虑调度的影响。

IPAM 连接问题和分段故障

问题：发生多个错误，包括 IPAM 连接问题、限制请求和分段故障：

- `Checking for IPAM connectivity …`
- `Throttling request took 1.047064274s`
- `Retrying waiting for IPAM-D`
- `panic: runtime error: invalid memory address or nil pointer dereference`

解决方案：如果您在 AL2023 上安装 `systemd-udev`，则会出现此问题，因为文件使用违反策略重写。更新为包含更新软件包的其他 `releasever` 或手动更新软件包本身时，可能会发生这种情况。避免在 AL2023 节点上安装或更新 `systemd-udev`。

无法按名称查找设备错误

问题：错误消息：`{"level":"error","ts":"2025-02-05T20:27:18.669Z","caller":"ebpf/bpf_client.go:578","msg":"failed to find device by name eni9ea69618bf0: %!w(netlink.LinkNotFoundError={0xc000115310})"}`

解决方案：此问题已发现并在最新版本的 Amazon VPC CNI 网络策略代理（v1.2.0）中修复。更新到最新版本的 VPC CNI 即可解决此问题。

Multus CNI 映像中的 CVE 漏洞

问题：增强版 EKS ImageScan CVE 报告发现 Multus CNI 映像版本 v4.1.4-eksbuild.2_thick 中存在漏洞。

解决方案：更新为无漏洞的新版本 Multus CNI 映像和新的网络策略控制器映像。可以更新扫描器以修复先前版本中发现的漏洞。

日志中的 Flow Info DENY 裁决

问题：网络策略日志显示 DENY 裁决：`{"level":"info","ts":"2024-11-25T13:34:24.808Z","logger":"ebpf-client","caller":"events/events.go:193","msg":"Flow Info: ","Src IP":"","Src Port":9096,"Dest IP":"","Dest Port":56830,"Proto":"TCP","Verdict":"DENY"}`

解决方案：此问题已在新版本的网络策略控制器中得到解决。更新为最新的 EKS 平台版本即可解决日志记录问题。

从 Calico 迁移后容器组到容器组的通信问题

问题：将 EKS 集群升级到版本 1.30 并将网络策略从 Calico 切换到 Amazon VPC CNI 后，应用网络策略后容器组到容器组的通信失败。删除网络策略后，通信即恢复。

解决方案：VPC CNI 中的网络策略代理不能像 Calico 那样指定那么多的端口。相反，在网络策略中使用端口范围。网络策略中每个 `ingress:` 或 `egress:` 选择器中每个协议的唯一端口组合数最多为 24 个。使用端口范围可减少唯一端口的数量和避免此限制。

网络策略代理不支持独立容器组

问题：应用于独立容器组的网络策略可能存在不一致的行为。

解决方案：网络策略代理目前仅支持作为部署/副本集一部分部署的容器组。如果将网络策略应用至独立容器组，则行为可能存在一些不一致。这将记录在本页顶部、the section called “注意事项” 中和 GitHub 上的 aws-network-policy-agent GitHub issue #327 中。将容器组作为部署或副本集的一部分进行部署，以实现一致的网络策略行为。

Amazon EKS 的网络策略的 Stars 演示

该演示在您的 Amazon EKS 集群上创建前端、后端和客户端服务。该演示还创建管理图形用户界面，用于显示各服务之间可用的传入和传出路径。我们建议您在不运行生产工作负载的集群上完成演示。

在您创建任何网络策略之前，所有服务可以双向通信。在应用网络策略后，您可以看到客户端只能与前端服务进行通信，而后端只能接受来自前端的流量。

1. 应用前端、后端、客户端和管理用户界面服务：

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/client.yaml
```

2. 查看集群上的所有容器组（pod）。

```
kubectl get pods -A
```

示例输出如下。

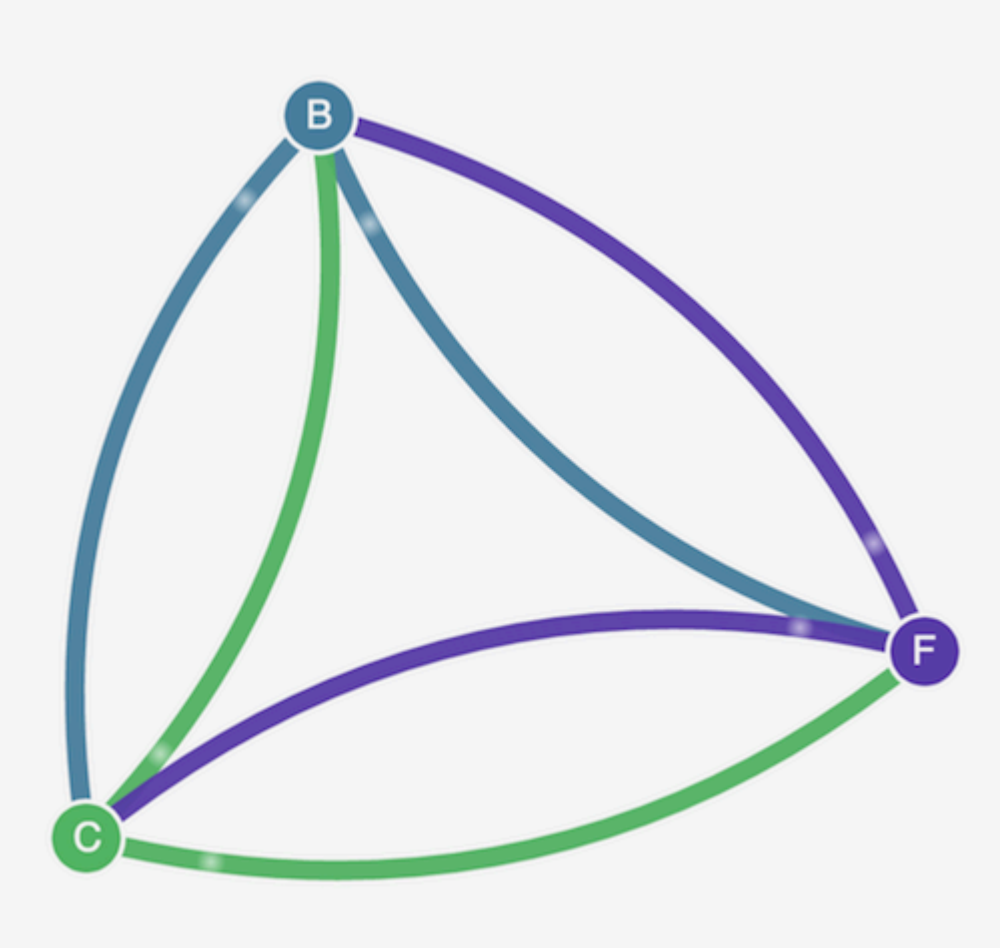
在输出中，您应该会在以下输出中显示的命名空间中看到容器。容器组（pod）的 *NAMES* 和 READY 列中的容器组（pod）数量与以下输出中的不同。在看到具有相似名称的容器并且它们在 STATUS 列中都具有 Running 之前，请不要继续。

```
NAMESPACE          NAME                                       READY   STATUS
 RESTARTS   AGE
[...]
client             client-xlffc                               1/1     Running   0
     5m19s
[...]
management-ui      management-ui-qrb2g                        1/1     Running   0
     5m24s
stars              backend-sz87q                              1/1     Running   0
     5m23s
stars              frontend-cscnf                             1/1     Running   0
     5m21s
[...]
```

3. 要连接到管理用户界面，请连接到集群上运行的 EXTERNAL-IP 服务：

```
kubectl get service/management-ui -n management-ui
```

4. 打开浏览器，进入上一步中的位置。您应该会看到管理用户界面。C 节点是客户端服务，F 节点是前端服务，B 节点是后端服务。每个节点都有到所有其他节点的完整通信访问权限（如粗体、彩色行所示）。



5. 在 stars 和 client 命名空间中应用以下网络策略，以将服务彼此隔离：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels: {}
```

您可以使用以下命令将策略应用于两个命名空间：

```
kubectl apply -n stars -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
kubectl apply -n client -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
```

6. 刷新您的浏览器。您可以看到管理用户界面不再能访问任何节点，因此它们不会显示在用户界面中。
7. 应用以下不同的网络策略以允许管理用户界面访问这些服务。应用此策略以允许 UI：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

应用此策略以允许客户端：

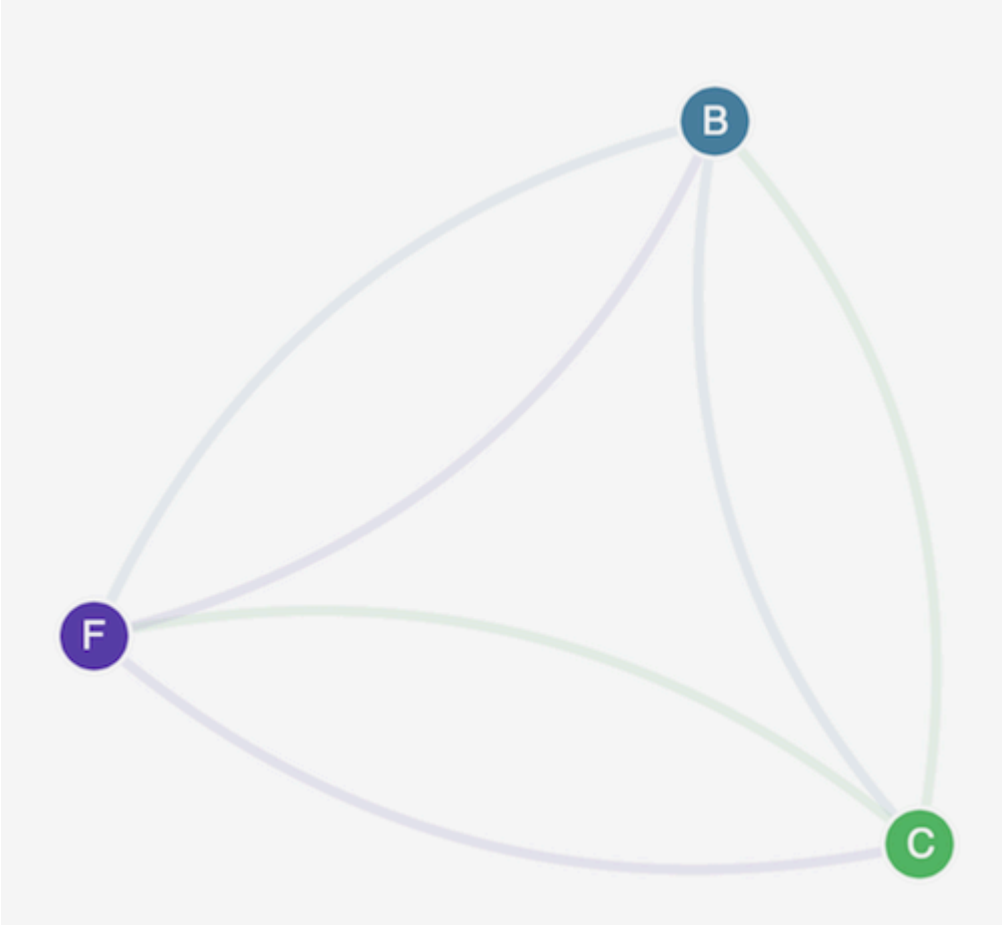
```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: client
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

您可以使用以下命令应用这两个策略：

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui-client.yaml
```

8. 刷新您的浏览器。您可以看到管理用户界面可以再次访问节点，但各节点无法相互通信。

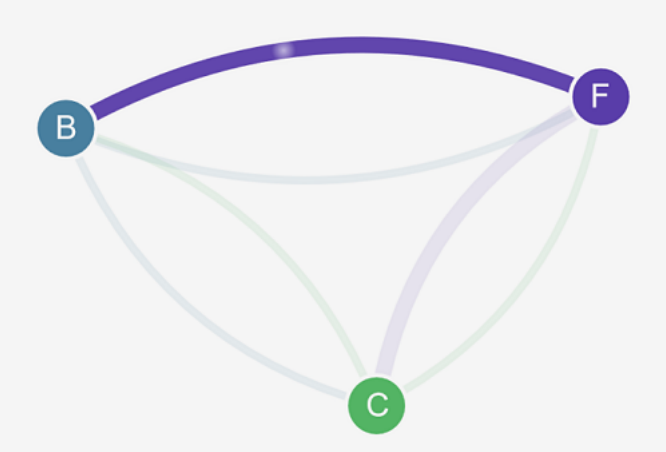


9. 应用以下网络策略以允许流量从前端服务传到后端服务：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: backend-policy
spec:
  podSelector:
    matchLabels:
      role: backend
  ingress:
    - from:
        - podSelector:
```

```
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 6379
```

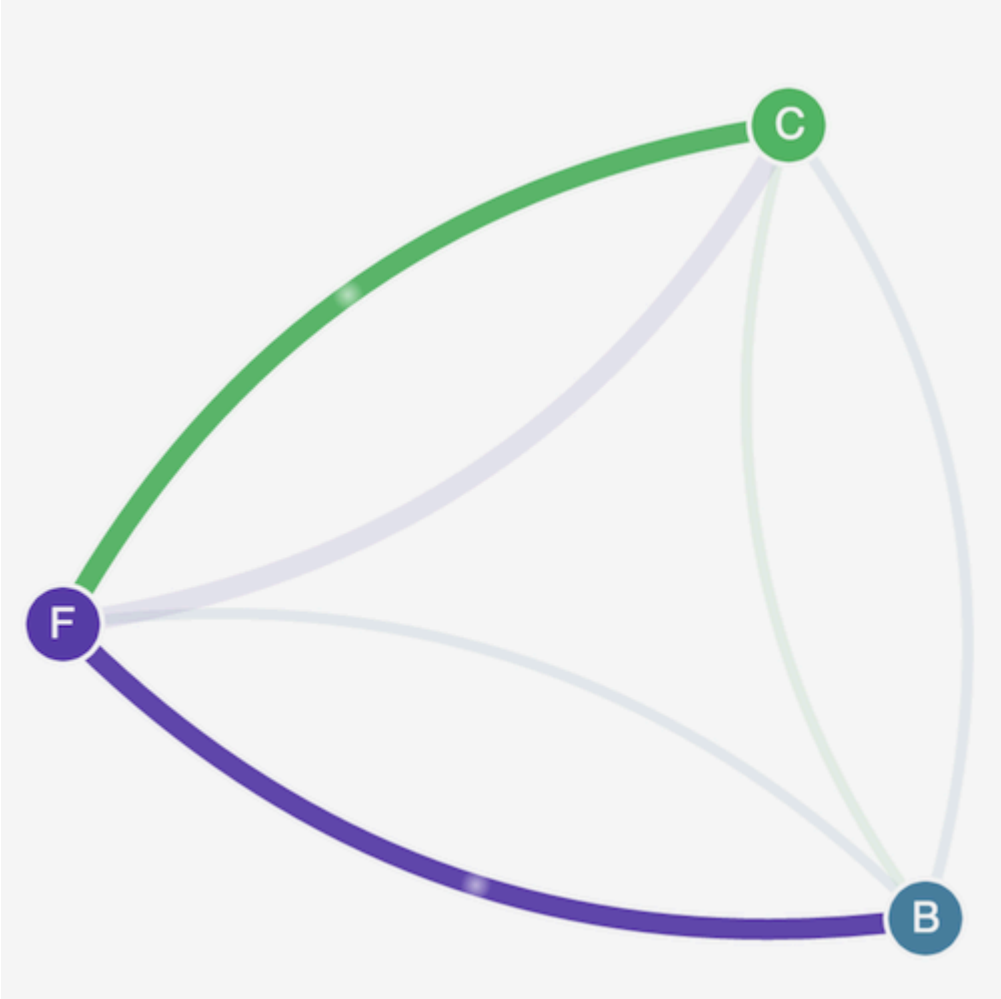
10刷新您的浏览器。您将看到前端可以与后端进行通信。



11应用以下网络策略以允许流量从客户端传到前端服务：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: frontend-policy
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: client
      ports:
        - protocol: TCP
          port: 80
```

12刷新您的浏览器。您将看到客户端可以与前端服务进行通信。前端服务仍可以与后端服务进行通信。



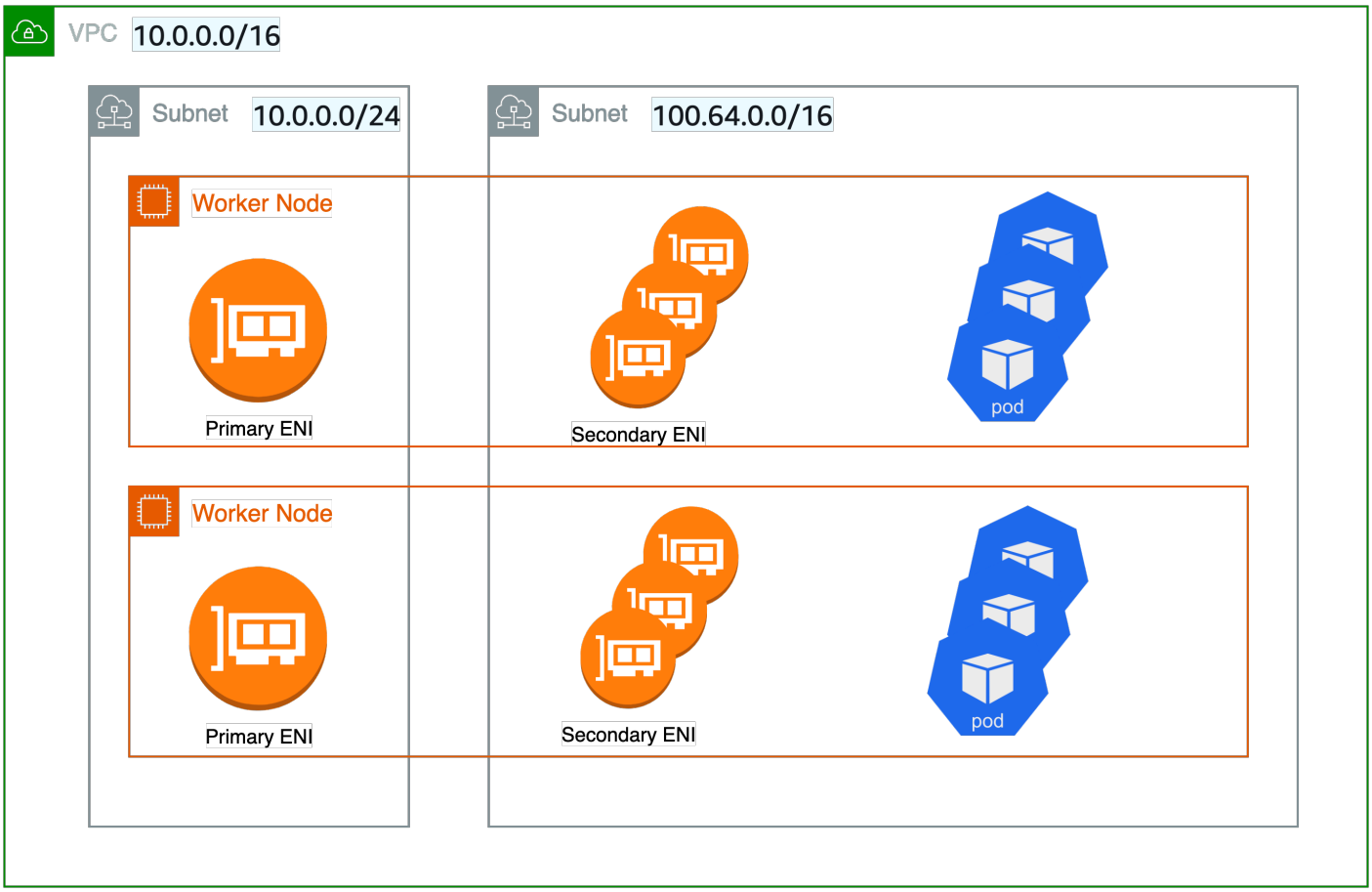
13. (可选) 完成该演示后，您可以删除其资源。

```
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/client.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
```

甚至在删除资源后，节点上仍可能存在网络策略端点，这些端点可能会以意想不到的方式干扰集群中的网络。删除这些规则的唯一可靠方法是重新启动节点或终止所有节点并将它们回收。要终止所有节点，请将自动扩缩组所需计数设置为 0，然后创建所需数量的备份，或者仅终止节点。

使用自定义网络在备用子网中部署容器组（pod）

适用对象：Linux IPv4 Fargate 节点以及带有 Amazon EC2 实例的 Linux 节点



默认情况下，当适用于 Kubernetes 的 Amazon VPC CNI 插件为 Amazon EC2 节点创建辅助弹性网络接口（网络接口）时，会在与该节点的主网络接口相同的子网中创建它们。它还将相同的安全组关联到与主网络接口关联的辅助网络接口。出于以下一个或多个原因，您可能希望该插件在不同子网中创建辅助网络接口，或者希望将不同的安全组与辅助网络接口关联，或者希望同时实现此两者：

- 主网络接口所在子网中可用的 IPv4 地址数量有限。这可能会限制您可以在子网中创建的容器组（pod）数量。通过对辅助网络接口使用不同的子网，您可以增加可用于容器组（pod）的 IPv4 地址的数量。

- 出于安全原因，容器组（pod）可能需要使用与节点的主网络接口不同的子网或安全组。
- 在公有子网中配置节点，在私有子网中放置容器组（pod）。与公有子网关联的路由表包含指向互联网网关的路由。与私有子网关联的路由表不包含指向互联网网关的路由。

> **Tip**
>
> 您也可以直接向 Amazon EKS 集群添加新子网或现有子网，而不使用自定义联网。有关更多信息，请参阅 the section called “通过管理控制台将现有 VPC 子网添加到 Amazon EKS 集群”。

注意事项

以下是使用该功能的注意事项。

- 启用自定义联网后，不会将分配给主网络接口的 IP 地址分配给容器组（pod）。仅会将辅助网络接口的 IP 地址分配给容器组（pod）。
- 如果您的集群使用 IPv6 系列，您无法使用自定义联网。
- 如果您计划仅为帮助缓解 IPv4 地址耗尽而使用自定义联网，您可以改为使用 IPv6 系列创建集群。有关更多信息，请参阅 the section called “IPv6”。
- 尽管部署到为辅助网络接口指定的子网的容器组（pod）可以使用与节点的主网络接口不同的子网和安全组，子网和安全组也必须与节点位于同一 VPC 中。
- 对于 Fargate，子网通过 Fargate 配置文件控制。有关更多信息，请参阅 the section called “定义配置文件”。

自定义 Amazon EKS 节点中的辅助网络接口

开始教程前，请完成以下任务：

- 查看注意事项
- 了解适用于 Kubernetes 的 Amazon VPC CNI 插件如何创建辅助网络接口，并将 IP 地址分配给容器组（pod）。有关更多信息，请参阅 GitHub 上的 ENI 分配。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 2.12.3 或更高版本，或版本 1.27.160 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon

CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。

- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 我们建议您在 Bash Shell 中完成本主题中的步骤。如果您没有使用 Bash Shell，则某些脚本命令（例如行延续字符以及变量的设置和使用方式）需要调整 Shell。此外，您的 Shell 的引用和转义规则可能有所不同。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的在 Amazon CLI 中将引号和字符串结合使用。

在本教程中，我们建议使用 example values，除非有说明要替换它们。您可以在完成生成集群的步骤时替换任何示例值。我们建议在同一终端完成所有步骤。这是因为变量是在整个步骤中设置和使用的，并且不会存在于不同的终端中。

本主题中的命令使用使用 Amazon CLI 示例中列出的惯例进行格式化。如果从命令行运行命令，且该命令行针对的资源位于与您正在使用的 Amazon CLI 配置文件中定义的默认 Amazon 区域不同的 Amazon 区域，则需要向命令添加 `--region us-west-2`，并将 `us-west-2` 替换为 Amazon 区域。

当您想将自定义联网部署到生产集群时，请跳至 the section called “步骤 2：配置 VPC”。

步骤 1：创建测试 VPC 和集群

以下步骤帮助您创建测试 VPC 和集群并为该集群配置自定义联网。我们不建议将测试集群用于生产工作负载，因为本主题没有涵盖在生产集群上可能使用的几种不相关的功能。有关更多信息，请参阅 the section called “创建集群”。

1. 运行以下命令以定义 `account_id` 变量。

```
account_id=$(aws sts get-caller-identity --query Account --output text)
```

2. 创建 VPC。

   a. 如果您要部署到测试系统，则请使用 Amazon EKS Amazon CloudFormation 模板创建 VPC。

```
aws cloudformation create-stack --stack-name my-eks-custom-networking-vpc \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml \
  --parameters ParameterKey=VpcBlock,ParameterValue=192.168.0.0/24 \
  ParameterKey=PrivateSubnet01Block,ParameterValue=192.168.0.64/27 \
  ParameterKey=PrivateSubnet02Block,ParameterValue=192.168.0.96/27 \
```

```
  ParameterKey=PublicSubnet01Block,ParameterValue=192.168.0.0/27 \
  ParameterKey=PublicSubnet02Block,ParameterValue=192.168.0.32/27
```

b. Amazon CloudFormation 堆栈需要几分钟的时间来创建。要检查堆栈的部署状态，请运行以下命令。

```
aws cloudformation describe-stacks --stack-name my-eks-custom-networking-vpc --
query Stacks\[\].StackStatus  --output text
```

在命令的输出变成 CREATE_COMPLETE 之前，请勿继续执行下一步。

c. 使用模板创建的私有子网 ID 的值定义变量。

```
subnet_id_1=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
    --query "StackResources[?
LogicalResourceId=='PrivateSubnet01'].PhysicalResourceId" --output text)
subnet_id_2=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
    --query "StackResources[?
LogicalResourceId=='PrivateSubnet02'].PhysicalResourceId" --output text)
```

d. 使用上一步中检索到的子网的可用区定义变量。

```
az_1=$(aws ec2 describe-subnets --subnet-ids $subnet_id_1 --query
 'Subnets[*].AvailabilityZone' --output text)
az_2=$(aws ec2 describe-subnets --subnet-ids $subnet_id_2 --query
 'Subnets[*].AvailabilityZone' --output text)
```

3. 创建集群 IAM 角色。

a. 运行以下命令以创建 IAM 信任策略 JSON 文件。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
```

```
  ]
}
```

b. 创建 Amazon EKS 集群 IAM 角色。如有必要，使用您在上一步中将文件写入到的计算机上的路径为 `eks-cluster-role-trust-policy.json` 添加前缀。该命令将您在上一步中创建的信任策略与角色关联。要创建 IAM 角色，必须为正在创建角色的 IAM 主体分配 `iam:CreateRole` 操作（权限）。

```
aws iam create-role --role-name myCustomNetworkingAmazonEKSClusterRole --assume-
role-policy-document file://"eks-cluster-role-trust-policy.json"
```

c. 将名为 AmazonEKSClusterPolicy 的 Amazon EKS 托管 IAM 策略附加到角色。要将 IAM policy 附加到某个 IAM 主体，必须为附加该策略的主体分配以下 IAM 操作（权限）之一：`iam:AttachUserPolicy` 或 `iam:AttachRolePolicy`。

```
aws iam attach-role-policy --policy-arn arn:aws-cn:iam::aws:policy/
AmazonEKSClusterPolicy --role-name myCustomNetworkingAmazonEKSClusterRole
```

4. 创建 Amazon EKS 集群并配置您的设备以与其进行通信。

a. 创建集群。

```
aws eks create-cluster --name my-custom-networking-cluster \
   --role-arn arn:aws-cn:iam::$account_id:role/
myCustomNetworkingAmazonEKSClusterRole \
   --resources-vpc-config subnetIds="$subnet_id_1","$subnet_id_2"
```

> **Note**
>
> 您可能会收到一个错误，指示请求中的可用区之一没有足够容量来创建 Amazon EKS 集群。如果发生这种情况，错误输出将包含可支持新集群的可用区。再次尝试使用至少两个位于您账户中支持的可用区的子网创建集群。有关更多信息，请参阅 the section called “容量不足”。

b. 创建集群需要几分钟时间。要检查集群的部署状态，请运行以下命令。

```
aws eks describe-cluster --name my-custom-networking-cluster --query
 cluster.status
```

在命令的输出变成 `"ACTIVE"` 之前，请勿继续执行下一步。

c. 配置 kubectl 以与集群通信。

```
aws eks update-kubeconfig --name my-custom-networking-cluster
```

步骤 2：配置 VPC

本教程需要在 the section called “步骤 1：创建测试 VPC 和集群” 中创建的 VPC。对于生产集群，通过将所有示例值替换为您自己的值来相应调整 VPC 的步骤。

1. 确认当前安装的适用于 Kubernetes 的 Amazon VPC CNI 插件版本为最新版本。要确定 Amazon EKS 附加组件类型的最新版本并将您的版本更新至此版本，请参阅 the section called “更新附加组件”。要确定自行管理的附加组件类型的最新版本并将您的版本更新至此版本，请参阅 the section called “Amazon VPC CNI”。
2. 检索您的集群 VPC 的 ID，并将其存储在变量中，以便在后续步骤中使用。

```
vpc_id=$(aws eks describe-cluster --name my-custom-networking-cluster --query
 "cluster.resourcesVpcConfig.vpcId" --output text)
```

3. 将另一个无类别域间路由（CIDR）块与集群的 VPC 关联。CIDR 块不能与任何现有关联的 CIDR 块重叠。

   a. 查看与您的 VPC 关联的当前 CIDR 块。

```
aws ec2 describe-vpcs --vpc-ids $vpc_id \
    --query 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

   示例输出如下。

```
---------------------------------
|          DescribeVpcs         |
+-----------------+-------------+
|    CIDRBlock    |    State    |
+-----------------+-------------+
|  192.168.0.0/24 |  associated |
+-----------------+-------------+
```

   b. 将额外 CIDR 块与 VPC 关联。在以下命令中，替换 CIDR 块的值。有关更多信息，请参阅《Amazon VPC 用户指南》中的将额外的 IPv4 CIDR 块与 VPC 关联。

```
aws ec2 associate-vpc-cidr-block --vpc-id $vpc_id --cidr-block 192.168.1.0/24
```

c. 确认新区块已关联。

```
aws ec2 describe-vpcs --vpc-ids $vpc_id --query
 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

示例输出如下。

```
---------------------------------
|          DescribeVpcs          |
+-----------------+--------------+
|    CIDRBlock    |    State     |
+-----------------+--------------+
|  192.168.0.0/24 |  associated  |
|  192.168.1.0/24 |  associated  |
+-----------------+--------------+
```

在您的新 CIDR 区块的 `State` 为 `associated` 之前，请勿继续执行下一步。

4. 在现有子网所在的每个可用区中创建要使用的任意数量的子网。指定一个在之前的步骤中与 VPC 关联的 CIDR 块内的 CIDR 块。

a. 创建新子网。在以下命令中，替换 CIDR 块的值。子网必须在不同于现有子网所在的 VPC CIDR 块中创建子网，但与现有子网位于同一可用区中。在此示例中，在当前私有子网所在的每个可用区的新 CIDR 块中创建一个子网。创建的子网的 ID 存储在变量中以供后续步骤使用。`Name` 值与分配给上一步中使用 Amazon EKS VPC 模板创建的子网的值相匹配。名称不是必填项。您可以使用不同的名称。

```
new_subnet_id_1=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_1
 --cidr-block 192.168.1.0/27 \
    --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet01},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
    --query Subnet.SubnetId --output text)
new_subnet_id_2=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_2
 --cidr-block 192.168.1.32/27 \
    --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet02},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
    --query Subnet.SubnetId --output text)
```

> **Important**
>
> 默认情况下，您的新子网与您的 VPC 的主路由表隐式关联。此路由表允许在 VPC 中部署的所有资源之间进行通信。但是，它不允许与 IP 地址位于与您的 VPC 关联的 CIDR 块之外的资源进行通信。您可以将自己的路由表关联到子网以改变此行为。有关更多信息，请参阅《Amazon VPC 用户指南》中的子网路由表。

b. 查看 VPC 中的当前子网。

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" \
    --query 'Subnets[*].{SubnetId: SubnetId,AvailabilityZone:
 AvailabilityZone,CidrBlock: CidrBlock}' \
    --output table
```

示例输出如下。

```
-------------------------------------------------------------------------
|                           DescribeSubnets                             |
+------------------+--------------------+-----------------------------+
| AvailabilityZone |     CidrBlock      |         SubnetId            |
+------------------+--------------------+-----------------------------+
|  us-west-2d      |  192.168.0.0/27    |     subnet-example1         |
|  us-west-2a      |  192.168.0.32/27   |     subnet-example2         |
|  us-west-2a      |  192.168.0.64/27   |     subnet-example3         |
|  us-west-2d      |  192.168.0.96/27   |     subnet-example4         |
|  us-west-2a      |  192.168.1.0/27    |     subnet-example5         |
|  us-west-2d      |  192.168.1.32/27   |     subnet-example6         |
+------------------+--------------------+-----------------------------+
```

您可以看到您创建的 `192.168.1.0` CIDR 块中的子网与 `192.168.0.0` CIDR 块中的子网在同一个可用区中。

步骤 3：配置 Kubernetes 资源

1. 将 `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` 环境变量设置为 `aws-node` DaemonSet 中的 `true`。

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

2. 检索集群安全组的 ID，并将其存储在变量中，以便在后续步骤中使用。当您创建集群时，Amazon EKS 会自动创建此安全组。

```
cluster_security_group_id=$(aws eks describe-cluster --name my-custom-networking-
cluster --query cluster.resourcesVpcConfig.clusterSecurityGroupId --output text)
```

3. 为希望在其中部署容器组（pod）的每个子网创建 ENIConfig 自定义资源。

   a. 为每个网络接口配置创建一个唯一的文件。

   以下命令为您在上一步中创建的两个子网创建单独的 ENIConfig 文件。name 的值必须是唯一的。该名称与子网所在的可用区相同。将集群安全组分配给 ENIConfig。

```
cat >$az_1.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_1
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_1
EOF
```

```
cat >$az_2.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_2
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_2
EOF
```

   对于生产集群，您可以对以前的命令进行以下更改：

   - 将 $cluster_security_group_id 替换为希望用于每个 ENIConfig 的现有安全组的 ID。

- 我们建议尽可能将 ENIConfigs 命名为与您将使用 ENIConfig 的可用区相同的名称。您可能需要出于各种原因为您的 ENIConfigs 使用与可用区名称不同的名称。例如，如果您在同一可用区中有两个以上的子网，并且想将它们同时用于自定义联网，则需要将多个 ENIConfigs 用于同一个可用区。由于每个 ENIConfig 都需要一个唯一名称，您不能使用可用区名称命名多个 ENIConfigs。

  如果 ENIConfig 名称与可用区名称不同，请将 $az_1 和 $az_2 替换为上一个命令中您自己的名称，并且稍后在本教程中使用 ENIConfig 注释节点。

  > **Note**
  >
  > 如果未指定与生产集群结合使用的有效安全组并且您使用的是：

- 版本 1.8.0 或更高版本的适用于 Kubernetes 的 Amazon VPC CNI 插件，则会使用与节点的主弹性网络接口关联的安全组。
- 1.8.0 之前的适用于 Kubernetes 的 Amazon VPC CNI 插件版本，则 VPC 的默认安全组将分配给辅助网络接口。

  > **Important**
  >
  > - AWS_VPC_K8S_CNI_EXTERNALSNAT=false 是适用于 Kubernetes 的 Amazon VPC CNI 插件配置中的默认设置。如果您使用的是默认设置，则发往不在与 VPC 关联的 CIDR 块之一内的 IP 地址的流量将使用节点主网络接口的安全组和子网。在用于创建辅助网络接口的 ENIConfigs 中定义的子网和安全组不用于此流量。有关该设置的更多信息，请参阅 the section called “出站流量”。
> - 如果您还将安全组用于容器组（pod），则会使用 SecurityGroupPolicy 中指定的安全组而不是 ENIConfigs 中指定的安全组。有关更多信息，请参阅 the section called “适用于容器组（pod）的安全组”。

b. 使用以下命令将您创建的每个自定义资源文件应用于您的集群。

```
kubectl apply -f $az_1.yaml
kubectl apply -f $az_2.yaml
```

4. 确认您的 ENIConfigs 已创建。

```
kubectl get ENIConfigs
```

示例输出如下。

```
NAME         AGE
us-west-2a   117s
us-west-2d   105s
```

5. 如果您在生产集群上启用自定义联网并将您的 ENIConfigs 命名为您将它们用于可用区之外的其他名称，然后跳到下一步以部署 Amazon EC2 节点。

   启用 Kubernetes 以将可用区的 ENIConfig 自动应用于在您的集群中创建的任何新 Amazon EC2 节点。

   a. 对于本教程中的测试集群，请跳至下一步。

      对于生产集群，请检查带有 ENI_CONFIG_ANNOTATION_DEF 环境变量的键 k8s.amazonaws.com/eniConfig 的注释是否存在于 aws-node DaemonSet 的容器规范中。

      ```
      kubectl describe daemonset aws-node -n kube-system | grep
       ENI_CONFIG_ANNOTATION_DEF
      ```

      如果返回输出，则注释存在。如果没有返回输出，则未设置此变量。对于生产集群，您可以使用此设置或以下步骤中的设置。如果使用此设置，它将覆盖以下步骤中的设置。在本教程中，将使用下一步中的设置。

   b. 更新 aws-node DaemonSet 以将可用区的 ENIConfig 自动应用于在集群中创建的任何新 Amazon EC2 节点。

      ```
      kubectl set env daemonset aws-node -n kube-system
       ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone
      ```

步骤 4：部署 Amazon EC2 节点

1. 创建节点 IAM 角色。

   a. 运行以下命令以创建 IAM 信任策略 JSON 文件。

      ```
      {
        "Version":"2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
      ```

```
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 创建 IAM 角色并将返回的 Amazon 资源名称（ARN）存储在变量中以供后续步骤使用。

```
node_role_arn=$(aws iam create-role --role-name myCustomNetworkingNodeRole --
assume-role-policy-document file://"node-role-trust-relationship.json" \
    --query Role.Arn --output text)
```

c. 将三个所需的 IAM 托管策略附加到 IAM 角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy \
    --role-name myCustomNetworkingNodeRole
```

> **Important**
>
> 为简单起见，在本教程中，将 AmazonEKS_CNI_Policy 策略附加到节点 IAM 角色。但是，在生产集群中，我们建议将该策略附加到仅用于适用于 Kubernetes 的 Amazon VPC CNI 插件的单独 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。

2. 创建以下一种类型的节点组。要确定您想要部署的实例类型，请参阅 the section called “Amazon EC2 实例类型”。在本教程中，请完成 Managed（托管）、Without a launch template or with a launch template without an AMI ID specified（没有启动模板或带有未指定 AMI ID 的启动模板）选项。如果要将节点组用于生产工作负载，我们建议在部署节点组之前熟悉所有的托管式节点组和自主管理型节点组选项。

   - 托管：使用以下选项之一部署您的节点组：

- 没有启动模板或者带有未指定 AMI ID 的启动模板 – 运行以下命令。在本教程中，使用 example values。对于生产节点组，将所有的 example values 替换为您自己的值。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。

```
aws eks create-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup \
    --subnets $subnet_id_1 $subnet_id_2 --instance-types t3.medium --node-role
 $node_role_arn
```

- 使用具有指定 AMI ID 的启动模板：

  A. 确定 Amazon EKS 为节点推荐的最大容器组（pod）数量。按照 Amazon EKS 针对每种 Amazon EC2 实例类型建议的最大容器组（pod）数量中的说明进行操作，将 `--cni-custom-networking-enabled` 添加至该主题中的步骤 3。请记下输出的内容，以便在下一个步骤中使用。

  B. 在启动模板中，指定 Amazon EKS 优化的 AMI ID，或者指定基于 Amazon EKS 优化 AMI 构建的自定义 AMI，然后使用启动模板部署节点组并在启动模板中提供以下用户数据。此用户数据会将实际参数传递到 `NodeConfig` 规范中。有关 NodeConfig 的更多信息，请参阅 NodeConfig API reference。您可以将 `20` 替换为上一步的值（建议）或您自己的值。

```
---
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"
--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    ...
    kubelet:
      config:
        maxPods: 20
```

  如果您创建的自定义 AMI 不是基于 Amazon EKS 优化版 AMI 构建的，则需要自行自定义创建配置。

- 自行管理：

i. 确定 Amazon EKS 为节点推荐的最大容器组（pod）数量。按照 the section called “Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量” 中的说明进行操作，将 `--cni-custom-networking-enabled` 添加到该主题的步骤 3。请记下输出的内容，以便在下一个步骤中使用。

ii. 按照 the section called “Amazon Linux” 中的说明部署节点组。

> **Note**
>
> 如果您希望生产集群中的节点支持更高数量的容器组（pod），请再次运行 the section called “Amazon EKS 建议每种 Amazon EC2 实例类型的最大容器组数量” 中的脚本。同时，将 `--cni-prefix-delegation-enabled` 选项添加到命令中。例如，将为 `m5.large` 实例类型返回 `110`。有关如何启用此功能的说明，请参阅 the section called “增加 IP 地址”。您可以将此功能与自定义联网一起使用。

3. 节点组创建需要几分钟时间。您可以使用以下命令检查托管节点组创建的状态。

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup --query nodegroup.status --output text
```

在返回的输出为 `ACTIVE` 之前，请勿继续执行下一步。

4. 在本教程中，您可以跳过此步骤。

对于生产集群，如果您没有将您的 `ENIConfigs` 命名为您将其用于的可用区相同，则必须使用应与节点结合使用的 `ENIConfig` 名称对节点进行注释。如果您在每个可用区中只有一个子网并且将 `ENIConfigs` 命名为与可用区相同的名称，则此步骤不是必需的。这是因为当您在上一步中启用适用于 Kubernetes 的 Amazon VPC CNI 插件以执行此操作后，它会自动为您将正确的 `ENIConfig` 与节点关联。

a. 获取集群中的节点列表。

```
kubectl get nodes
```

示例输出如下。

```
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-0-126.us-west-2.compute.internal    Ready    <none>   8m49s   v1.22.9-eks-810597c
```

```
ip-192-168-0-92.us-west-2.compute.internal    Ready    <none>   8m34s   v1.22.9-
eks-810597c
```

b. 确定每个节点所在的可用区。运行上一步中返回的每个节点的以下命令，根据之前的输出替换 IP 地址。

```
aws ec2 describe-instances --filters Name=network-interface.private-dns-
name,Values=ip-192-168-0-126.us-west-2.compute.internal \
--query 'Reservations[].Instances[].{AvailabilityZone: Placement.AvailabilityZone,
 SubnetId: SubnetId}'
```

示例输出如下。

```
[
    {
        "AvailabilityZone": "us-west-2d",
        "SubnetId": "subnet-Example5"
    }
]
```

c. 使用您为子网 ID 和可用区创建的 ENIConfig 注释每一个节点。您只能使用一个 ENIConfig 注释节点，尽管使用同一个 ENIConfig 可以注释多个节点。将 example values 替换为您自己的值。

```
kubectl annotate node ip-192-168-0-126.us-west-2.compute.internal
 k8s.amazonaws.com/eniConfig=EniConfigName1
kubectl annotate node ip-192-168-0-92.us-west-2.compute.internal
 k8s.amazonaws.com/eniConfig=EniConfigName2
```

5. 如果在切换到使用自定义联网功能之前，生产集群中有任何正在运行容器组（pod）的节点，请完成以下任务：

a. 确保您有可用的节点正在使用自定义联网功能。

b. 封锁并耗尽节点以正常关闭容器组（pod）。有关更多信息，请参阅 Kubernetes 文档中的安全耗尽节点。

c. 终止节点。如果节点位于现有的托管节点组中，则可以删除该节点组。运行如下命令。

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup
```

只有注册有 k8s.amazonaws.com/eniConfig 标注的新节点将使用新的自定义联网功能。

6. 确认已从 CIDR 块中为容器组（pod）分配一个 IP 地址，该地址与您在上一步中创建的其中一个子网相关联。

```
kubectl get pods -A -o wide
```

示例输出如下。

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE     IP
           NODE                                            NOMINATED NODE   READINESS
 GATES
kube-system   aws-node-2rkn4                1/1     Running   0          7m19s
 192.168.0.92    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   aws-node-k96wp                1/1     Running   0          7m15s
 192.168.0.126   ip-192-168-0-126.us-west-2.compute.internal   <none>
 <none>
kube-system   coredns-657694c6f4-smcgr      1/1     Running   0          56m
 192.168.1.23    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   coredns-657694c6f4-stwv9      1/1     Running   0          56m
 192.168.1.28    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-jgshq              1/1     Running   0          7m19s
 192.168.0.92    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-wx9vk              1/1     Running   0          7m15s
 192.168.0.126   ip-192-168-0-126.us-west-2.compute.internal   <none>
 <none>
```

您可以看到 coredns 容器组（pod）从您添加到 VPC 中的 `192.168.1.0` CIDR 块中分配到了 IP 地址。如果没有自定义联网，他们就会从 `192.168.0.0` CIDR 块分配地址，因为它是最初与 VPC 关联的唯一 CIDR 块。

如果容器组（pod）的 `spec` 包含 `hostNetwork=true`，则会向其分配节点的主 IP 地址。不会从您添加的子网中向其分配地址。默认情况下，该值设置为 `false`。对于在集群上运行的 `kube-proxy` 和适用于 Kubernetes 的 Amazon VPC CNI 插件 (`aws-node`) 容器组（pod），此值设置为 `true`。这就是在之前的输出中没有向 `kube-proxy` 和插件的 `aws-node` 容器组（pod）分配 192.168.1.x 地址的原因。有关容器组（pod）`hostNetwork` 设置的更多信息，请参阅 Kubernetes API 参考中的 PodSpec v1 core。

步骤 5：删除教程资源

完成本教程后，我们建议您删除创建的资源。然后，您可以调整步骤以为生产集群启用自定义联网。

1. 如果您创建的节点组只是为了测试，那么请将其删除。

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name
 my-nodegroup
```

2. 即使在 Amazon CLI 输出显示集群已删除，删除过程实际上可能尚未完成。删除过程需要几分钟时间。通过运行以下命令确认已完成。

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup --query nodegroup.status --output text
```

   在返回的输出类似于以下输出之前，请勿继续执行。

```
An error occurred (ResourceNotFoundException) when calling the DescribeNodegroup
 operation: No node group found for name: my-nodegroup.
```

3. 如果您创建的节点组只是为了测试，则请删除节点 IAM 角色。

   a. 将策略与角色分离。

```
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy
```

   b. 删除角色。

```
aws iam delete-role --role-name myCustomNetworkingNodeRole
```

4. 请删除集群。

```
aws eks delete-cluster --name my-custom-networking-cluster
```

   通过以下命令确认集群已删除。

```
aws eks describe-cluster --name my-custom-networking-cluster --query cluster.status
 --output text
```

返回类似以下内容的输出时，将成功删除集群。

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
 operation: No cluster found for name: my-custom-networking-cluster.
```

5. 删除集群 IAM 角色。

   a. 将策略与角色分离。

   ```
   aws iam detach-role-policy --role-name myCustomNetworkingAmazonEKSClusterRole --
   policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSClusterPolicy
   ```

   b. 删除角色。

   ```
   aws iam delete-role --role-name myCustomNetworkingAmazonEKSClusterRole
   ```

6. 删除您在上一步中创建的子网。

```
aws ec2 delete-subnet --subnet-id $new_subnet_id_1
aws ec2 delete-subnet --subnet-id $new_subnet_id_2
```

7. 删除您创建的 VPC。

```
aws cloudformation delete-stack --stack-name my-eks-custom-networking-vpc
```

为带前缀的 Amazon EKS 节点分配更多 IP 地址

适用于：带有 Amazon EC2 实例的 Linux 和 Windows 节点

适用于：公有子网和私有子网

每个 Amazon EC2 实例支持最大数量的弹性网络接口以及可分配给每个网络接口的最大数量的 IP 地址。每个节点的每个网络接口都需要一个 IP 地址。可以将所有其他可用 IP 地址分配给 Pods。每个都 Pod 需要自己的 IP 地址。因此，您的节点可能有可用的计算和内存资源，但无法容纳其它 Pods，因为该节点已用完分配给 Pods 的 IP 地址。

您可以通过分配 IP 前缀而不是为节点分配单个辅助 IP 地址来显著增加节点可以分配给 Pods 的 IP 地址数量。每个前缀都包含多个 IP 地址。如果没有为集群配置 IP 前缀分配，则集群必须进行更多 Amazon EC2 应用程序编程接口（API）调用才能配置容器组（pod）连接所需的网络接口和 IP 地址。随着集群规模的扩大，这些 API 调用的频率可能会导致容器组（pod）和实例启动时间延长。这会导致扩展延迟以满足大型和尖峰工作负载的需求，并增加成本和管理开销，因为您需要配置额外的集群和 VPC 来满足扩展需求。有关更多信息，请参阅 GitHub 上的Kubernetes 可扩展性阈值。

与适用于 Kubernetes 的 Amazon VPC CNI 插件功能的兼容性

可以将 IP 前缀用于以下功能：

- IPv4 源网络地址转换 - 有关更多信息，请参阅 the section called “出站流量”。
- 集群、容器组（pod）和服务的 IPv6 地址 - 有关更多信息，请参阅 the section called “IPv6”。
- 使用 Kubernetes 网络策略限制流量 – 有关更多信息，请参阅the section called “Kubernetes 策略”。

以下列表提供了适用的 Amazon VPC CNI 插件设置的相关信息。有关各项设置的更多信息，请参阅 GitHub 上的 amazon-vpc-cni-k8s。

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_PREFIX_TARGET`

注意事项

在使用此功能时，请考虑以下几点：

- 每种 Amazon EC2 实例类型支持最大数量的容器组（pod）。如果托管节点组由多种实例类型组成，则集群中某个实例的最大容器组（pod）数量的最小值将应用于集群中的所有节点。
- 默认情况下，可以在节点上运行的最大 Pods 数量为 110，但您可以更改该数字。如果您更改该数量并且有现有的托管节点组，则节点组的下一次 AMI 或启动模板更新会导致新节点出现时使用更改后的值。
- 从分配 IP 地址过渡到分配 IP 前缀时，建议您创建新的节点组以增加可用 IP 地址的数量，而不是滚动替换现有节点。在同时分配 IP 地址和前缀的节点上运行容器组（pod）可能会导致通告的 IP 地址容量不一致，从而影响节点上未来的工作负载。有关执行过渡的建议方法，请参阅《Amazon EKS Best Practices Guide》中的 Prefix Delegation mode for Linux。
- 安全组范围在节点级别 - 有关更多信息，请参阅安全组。

- 分配给网络接口的 IP 前缀支持在每个节点上运行更高密度的容器组（pod），并且实现最短的启动时间。
- IP 前缀和 IP 地址与标准 Amazon EC2 弹性网络接口相关联。需要特定安全组的容器组被分配给分支网络接口的主 IP 地址。您可以将获取 IP 地址或从 IP 前缀获取 IP 地址的容器组（pod）与在同一节点上获取分支网络接口的容器组（pod）混合使用。
- 仅适用于具有 Linux 节点的集群。
  - 将附加组件配置为向网络接口分配前缀后，将无法在不删除集群所有节点组中的所有节点的情况下将适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件降级到 `1.9.0`（或 `1.10.1`）以下版本。
  - 如果容器组（pod）在与 VPC 外部的端点通信的同时使用了容器组（pod）的安全组（`POD_SECURITY_GROUP_ENFORCING_MODE=standard` 和 `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`），则使用节点的安全组，而不是分配给容器组（pod）的任何安全组。

    如果您的 Pods 在与 VPC 外部的端点通信的同时使用了[容器组（pod）的安全组](#)（`POD_SECURITY_GROUP_ENFORCING_MODE=strict`），则使用 Pod's 安全组。

增加 Amazon EKS 节点的可用 IP 地址

您可以通过分配 IP 前缀而不是为节点分配单个辅助 IP 地址，来增加节点可以分配给容器组（pod）的 IP 地址数量。

先决条件

- 您需要现有集群。要部署一个角色，请参阅the section called “创建集群”。
- 您的 Amazon EKS 节点所在的子网必须有足够的连续 `/28`（适用于 `IPv4` 集群）或 `/80`（适用于 `IPv6` 集群）无类别域间路由 (CIDR) 块。`IPv6` 集群中只能有 Linux 节点。如果 IP 地址分散在整个子网 CIDR 中，则使用 IP 前缀可能会失败。我们建议执行下列操作：
  - 使用子网 CIDR 预留，这样即使保留范围内的任何 IP 地址仍在使用，在其释放后，这些 IP 地址也不会重新分配。这样可以确保前缀在不分段的情况下进行分配。
  - 使用专门用于运行分配 IP 前缀的工作负载的新子网。分配 IP 前缀时，Windows 和 Linux工作负载可以在同一个子网中运行。
- 要为节点分配 IP 前缀，您的节点必须基于 Amazon Nitro。不基于 Nitro 的实例会继续分配单个辅助 IP 地址，但分配给容器组（pod）的 IP 地址数量比基于 Nitro 的实例少得多。
- 仅适用于具有 Linux 节点的集群 – 如果集群针对 `IPv4` 系列配置，则必须安装适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件的版本 `1.9.0` 或更高版本。您可以使用以下命令检查当前版本。

```
kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/"
 -f 2
```

如果您的集群是针对 IPv6 系列配置的，则必须安装附加组件的 1.10.1 版。如果您的插件版本低于所需版本，则必须进行更新。要了解更多信息，请参阅向具有 Amazon VPC CNI 的容器组（pod）分配 IP 的更新部分。

- 仅适用于具有 Windows 节点的集群
  - 必须为集群启用 Windows 支持。有关更多信息，请参阅 the section called “启用 Windows 支持”。

将 IP 地址前缀分配给节点

配置集群以将 IP 地址前缀分配给节点。完成与节点的操作系统相匹配的步骤。

Linux

1. 启用参数，以便为 Amazon VPC CNI DaemonSet 的网络接口分配前缀。部署集群时，版本 1.10.1 或更高版本的适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件随之部署。如果您使用 IPv6 系列创建集群，这个设置将被默认设置为 true。如果您使用 IPv4 系列创建集群，这个设置将被默认设置为 false。

```
kubectl set env daemonset aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

> **Important**
>
> 即使子网有可用的 IP 地址，如果子网没有任何连续 /28 数据块可用，您还是会在适用于 Kubernetes 的 Amazon VPC CNI 插件日志中看到以下错误。
>
> ```
> InsufficientCidrBlocks: The specified subnet does not have enough free cidr
>  blocks to satisfy the request
> ```
>
> 发生这种情况的原因可能是分散在子网中的现有辅助 IP 地址的碎片。要解决此错误，请创建一个新子网并在其中启动容器组（pod），或者使用 Amazon EC2 子网 CIDR 预留，在子网中预留空间以便与前缀分配结合使用。有关更多信息，请参阅《Amazon VPC 用户指南》中的子网 CIDR 预留。

2. 如果计划在没有启动模板的情况下部署托管节点组，或者采用尚未在其中指定 AMI ID 的启动模板，并且使用的是在先决条件中列出的适用于 Kubernetes 的 Amazon VPC CNI 插件版本或更高版本，则跳至下一步。托管节点组会自动为您计算最大容器组（pod）数量。

   如果正在部署自主管理型节点组或带有启动模板的托管节点组，且其启动模板已指定 AMI ID，则必须确定 Amazon EKS 为节点推荐的最大容器组（pod）数量。按照 Amazon EKS 针对每种 Amazon EC2 实例类型建议的最大容器组（pod）数量中的说明进行操作，将 `--cni-prefix-delegation-enabled` 添加到步骤 3。请记下输出的内容，以便在下一个步骤中使用。

   > **Important**
   >
   > 托管节点组强制执行 `maxPods` 的值的最大数量。对于 vCPUs 少于 30 个的实例，最大数量为 110，对于所有其他实例，最大数量为 250。无论是否启用前缀委派，均应用此最大数量。

3. 如果使用为 IPv6 配置的集群，则跳至下一步。

   在以下选项中指定参数。要确定哪个选项适合您以及为其提供哪些值，请参阅 GitHub 上的 WARM_PREFIX_TARGET, WARM_IP_TARGET, and MINIMUM_IP_TARGET。

   您可以将示例值替换为大于零的值。

   - `WARM_PREFIX_TARGET`

     ```
     kubectl set env ds aws-node -n kube-system WARM_PREFIX_TARGET=1
     ```

   - `WARM_IP_TARGET` 或 `MINIMUM_IP_TARGET`：如果设置了两种值之一，则其会覆盖所设置的 `WARM_PREFIX_TARGET` 的值。

     ```
     kubectl set env ds aws-node -n kube-system WARM_IP_TARGET=5
     ```

     ```
     kubectl set env ds aws-node -n kube-system MINIMUM_IP_TARGET=2
     ```

4. 使用至少一种 Amazon EC2 Nitro Amazon Linux 2023 实例类型创建以下类型之一的节点组。有关 Nitro 实例类型的列表，请参阅《Amazon EC2 用户指南》中的基于 Nitro 系统构建的实例。Windows 中不支持此功能。对于包含 *110* 的选项，将其替换为第 3 步中的值（建议）或您自己的值。

- 自行管理 – 按照创建自行管理的 Amazon Linux 节点中的说明部署节点组。在创建 CloudFormation 堆栈之前，请打开模板文件并将 NodeLaunchTemplate 中的 UserData 调整为如下所示

```
...
            apiVersion: node.eks.aws/v1alpha1
            kind: NodeConfig
            spec:
              cluster:
                name: ${ClusterName}
                apiServerEndpoint: ${ApiServerEndpoint}
                certificateAuthority: ${CertificateAuthorityData}
                cidr: ${ServiceCidr}
              kubelet:
                config:
                  maxPods: 110
...
```

如果使用 eksctl 创建节点组，可以使用下面的命令。

```
eksctl create nodegroup --cluster my-cluster --managed=false --max-pods-per-node
 110
```

- 托管：使用以下选项之一部署您的节点组：
  - 没有启动模板或者没有指定 AMI ID 的启动模板：完成为集群创建托管式节点组中的过程。托管节点组会自动为您计算 Amazon EKS 建议的 max-pods 值。
  - 使用具有指定 AMI ID 的启动模板：在启动模板中，指定 Amazon EKS 优化的 AMI ID，或者指定基于 Amazon EKS 优化 AMI 构建的自定义 AMI，然后使用启动模板部署节点组并在启动模板中提供以下用户数据。此用户数据传递一个 NodeConfig 对象，供 nodeadm 工具在节点上读取。有关 nodeadm 的详细信息，请参阅 nodeadm 文档。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="//"

--//
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
```

```
spec:
 cluster:
   apiServerEndpoint: [.replaceable]`my-cluster`
   certificateAuthority: [.replaceable]`LS0t...`
   cidr: [.replaceable]`10.100.0.0/16`
   name: [.replaceable]`my-cluster
 kubelet:
   config:
     maxPods: [.replaceable]`110`
--//--
```

如果使用 eksctl 创建节点组，可以使用下面的命令。

```
eksctl create nodegroup --cluster my-cluster --max-pods-per-node 110
```

如果您创建的自定义 AMI 不是基于 Amazon EKS 优化版 AMI 构建的，则需要自行自定义创建配置。

> **Note**
>
> 如果还想将 IP 地址分配给不同于实例子网的容器组（pod），则需要在此步骤中启用该功能。有关更多信息，请参阅 the section called “自定义联网”。

Windows

1. 启用 IP 前缀的分配。
   a. 打开 amazon-vpc-cni ConfigMap 进行编辑。

   ```
   kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
   ```

   b. 将以下行添加到 data 部分。

   ```
     enable-windows-prefix-delegation: "true"
   ```

   c. 保存文件，然后关闭编辑器。
   d. 确认该行已添加到 ConfigMap。

```
kubectl get configmap -n kube-system amazon-vpc-cni -o "jsonpath={.data.enable-
windows-prefix-delegation}"
```

如果返回的输出并非 `true`，则可能存在错误。尝试再次完成该步骤。

> **Important**
>
> 即使子网有可用的 IP 地址，如果子网没有任何连续 /28 数据块可用，您还是会在适用于 Kubernetes 的 Amazon VPC CNI 插件日志中看到以下错误。
>
> ```
> InsufficientCidrBlocks: The specified subnet does not have enough free cidr
>  blocks to satisfy the request
> ```
>
> 发生这种情况的原因可能是分散在子网中的现有辅助 IP 地址的碎片。要解决此错误，请创建一个新子网并在其中启动容器组（pod），或者使用 Amazon EC2 子网 CIDR 预留，在子网中预留空间以便与前缀分配结合使用。有关更多信息，请参阅《Amazon VPC 用户指南》中的子网 CIDR 预留。

2. （可选）指定其他配置以控制集群的预扩展和动态扩展行为。有关更多信息，请参阅 GitHub 上的在 Windows 上使用前缀委派模式的配置选项。

   a. 打开 `amazon-vpc-cni` ConfigMap 进行编辑。

   ```
   kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
   ```

   b. 用大于零的值替换示例值，然后将所需的条目添加到 `ConfigMap` 的 `data` 部分。如果您为 `warm-ip-target` 或 `minimum-ip-target` 设置了值，则该值将覆盖为 `warm-prefix-target` 设置的任何值。

   ```
   warm-prefix-target: "1"
   warm-ip-target: "5"
   minimum-ip-target: "2"
   ```

   c. 保存文件，然后关闭编辑器。

3. 创建至少包含一种 Amazon EC2 Nitro 实例类型的 Windows 节点组。有关 Nitro 实例类型的列表，请参阅《Amazon EC2 用户指南》中的基于 Nitro 系统构建的实例。默认情况下，可以部署到节点的最大容器组（pod）数量为 110。如果要增加或减少该数量，则请在引导配置的用户数据中指定以下内容。将 *`max-pods-quantity`* 替换为最大容器组（pod）值。

```
-KubeletExtraArgs '--max-pods=max-pods-quantity'
```

如果您要部署托管节点组，则需要在启动模板中添加此配置。有关更多信息，请参阅 the section called “启动模板”。有关 Windows 引导脚本配置参数的更多信息，请参阅the section called “引导脚本配置参数”。

确定最大容器组（pod）数量和可用的 IP 地址

1. 节点部署完成后，请查看集群中的节点。

```
kubectl get nodes
```

示例输出如下。

```
NAME                                              STATUS     ROLES    AGE   VERSION
ip-192-168-22-103.region-code.compute.internal    Ready      <none>   19m   v1.XX.X-
eks-6b7464
ip-192-168-97-94.region-code.compute.internal     Ready      <none>   19m   v1.XX.X-
eks-6b7464
```

2. 描述其中一个节点以确定该节点的 max-pods 值和可用 IP 地址数量。将 *192.168.30.193* 替换为之前输出中返回的其中一个节点名称中的 IPv4 地址。

```
kubectl describe node ip-192-168-30-193.region-code.compute.internal | grep 'pods\|
PrivateIPv4Address'
```

示例输出如下。

```
pods:                                  110
vpc.amazonaws.com/PrivateIPv4Address:  144
```

在之前的输出中，尽管有 *144* 个 IP 地址可用，但 110 是 Kubernetes 将部署到节点的最大容器组（pod）数量。

将安全组分配给各个容器组（pod）

适用于：带有 Amazon EC2 实例的 Linux 节点

适用于：私有子网

容器组（pod）的安全组将 Amazon EC2 安全组与 Kubernetes Pod 集成在一起。您可以使用 Amazon EC2 安全组定义允许流向和来自于容器组（pod）的入站和出站网络流量的规则，而这些容器组（pod）将部署到在许多 Amazon EC2 实例类型和 Fargate 上运行的节点。有关此功能的详细说明，请参阅容器组（pod）的安全组介绍博客文章。

与适用于 Kubernetes 的 Amazon VPC CNI 插件功能的兼容性

您可以将容器组（pod）的安全组与以下功能结合使用：

- IPv4 源网络地址转换 - 有关更多信息，请参阅 the section called “出站流量”。
- 集群、容器组（pod）和服务的 IPv6 地址 - 有关更多信息，请参阅 the section called “IPv6”。
- 使用 Kubernetes 网络策略限制流量 – 有关更多信息，请参阅the section called “Kubernetes 策略”。

注意事项

在部署容器组（pod）的安全组之前，请考虑以下限制和条件：

- 容器组（pod）的安全组不能与 Windows 节点或 EKS 自动模式一起使用。
- 通过使用版本 1.16.0 或更高版本的 Amazon VPC CNI 插件，容器组（pod）的安全组可以与为包含 Amazon EC2 节点的 IPv6 系列配置的集群一起使用。通过使用版本 1.7.7 或更高版本的 Amazon VPC CNI 插件，容器组（pod）的安全组可以与为仅包含 Fargate 节点的 IPv6 系列配置的集群一起使用。有关更多信息，请参阅 the section called “IPv6”。
- 大多数基于 Nitro 的 Amazon EC2 实例系列都支持容器组（pod）的安全组，但并非所有实例代的系列都支持安全组。例如，支持 m5、c5、r5、m6g、c6g 与 r6g 实例系列和实例代。但不支持 t 系列中的实例类型。有关支持的实例类型的完整列表，请参阅 GitHub 上的 limits.go 文件。您的节点必须是在该文件中拥有 IsTrunkingCompatible: true 的所列出的实例类型之一。
- 如果要将自定义联网和容器组（pod）的安全组一起使用，则请使用容器组（pod）的安全组所指定的安全组，不要使用 ENIConfig 中指定的安全组。
- 如果您使用的是版本 1.10.2 或更低版本的 Amazon VPC CNI 插件，并且将 terminationGracePeriodSeconds 设置包括在您的容器组（pod）规范中，该设置的值不能为零。
- 如果您使用的是版本 1.10 或更低版本的 Amazon VPC CNI 插件或 POD_SECURITY_GROUP_ENFORCING_MODE=strict 的版本 1.11（默认设置），则您向其分配安全组的容器组（pod）将不支持使用实例目标（externalTrafficPolicy 设置为 Local）的

NodePort 和 LoadBalancer 型 Kubernetes 服务。有关将负载均衡器与实例目标一起使用的更多信息，请参阅 the section called “网络负载均衡”。

- 如果您使用的是版本 1.10 或更低版本的 Amazon VPC CNI 插件或 POD_SECURITY_GROUP_ENFORCING_MODE=strict 的版本 1.11（默认设置），对于来自分配有安全组的容器组（pod）的出站流量，禁用源 NAT，以便应用出站安全组规则。要访问 Internet，必须在配置了 NAT 网关或实例的私有子网中部署的节点上启动具有已分配安全组的容器组（pod）。部署到公有子网的已分配安全组的 Pod 无法访问 Internet。

  如果您使用的是 POD_SECURITY_GROUP_ENFORCING_MODE=standard 的版本 1.11 或者更高版本的插件，则发往 VPC 之外的容器组（pod）流量将转换为实例主网络接口的 IP 地址。对于此流量，将使用主网络接口的安全组中的规则，而不是容器组（pod）安全组中的规则。
- 要将 Calico 网络策略用于具有关联安全组的容器组（pod），您必须使用版本 POD_SECURITY_GROUP_ENFORCING_MODE 或更高版本的 Amazon VPC CNI 插件并设置 1.11.0=standard。否则，流向和来自具有关联安全组的容器组（pod）的流量不受 Calico 网络策略执行的限制，并且仅受限于 Amazon EC2 安全组执行。要更新 Amazon VPC CNI 版本，请参阅 the section called “Amazon VPC CNI”
- 在使用集群中的安全组的 Amazon EC2 节点上运行且使用 NodeLocal DNSCache 的容器组（pod）仅支持版本 1.11.0 或更高版本的 Amazon VPC CNI 插件和 POD_SECURITY_GROUP_ENFORCING_MODE=standard。要更新 Amazon VPC CNI 插件版本，请参阅 the section called “Amazon VPC CNI”
- 对于具有高流失率的容器组（pod）而言，容器组（pod）的安全组可能会导致更高的容器组（pod）启动延迟。这是由于资源控制器中的速率限制造成的。
- EC2 安全组范围处于容器组（pod）级别。有关更多信息，请参阅安全组。

  如果您设置了 POD_SECURITY_GROUP_ENFORCING_MODE=standard 和 AWS_VPC_K8S_CNI_EXTERNALSNAT=false，则发往 VPC 外部端点的流量使用节点的安全组，而不是容器组（pod）的安全组。

为 Amazon EKS 容器组（pod）的安全组配置适用于 Kubernetes 的 Amazon VPC CNI 插件

如果将容器组（pod）用于 Amazon EC2 实例，则需要为安全组配置适用于 Kubernetes 的 Amazon VPC CNI 插件

如果仅使用 Fargate 容器组（pod），但集群中没有任何 Amazon EC2 节点，则请参阅the section called “SecurityGroupPolicy”。

1. 使用以下命令查看当前适用于 Kubernetes 的 Amazon VPC CNI 插件版本：

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

示例输出如下。

```
v1.7.6
```

如果适用于 Kubernetes 的 Amazon VPC CNI 插件版本低于 1.7.7，请将该插件更新到版本 1.7.7 或更高版本。有关更多信息，请参阅 the section called “Amazon VPC CNI”。

2. 将 AmazonEKSVPCResourceController 托管 IAM 策略添加到与您的 Amazon EKS 集群关联的集群角色。策略允许角色管理网络接口、网络接口的私有 IP 地址以及与网络实例之间的连接和分离。

   a. 检索您的集群 IAM 角色的名称，然后将其存储在一个变量中。将 *my-cluster* 替换为您的集群的名称。

   ```
   cluster_role=$(aws eks describe-cluster --name my-cluster --query cluster.roleArn
    --output text | cut -d / -f 2)
   ```

   b. 将 策略附加到该角色。

   ```
   aws iam attach-role-policy --policy-arn arn:aws-cn:iam::aws:policy/
   AmazonEKSVPCResourceController --role-name $cluster_role
   ```

3. 通过在 aws-node DaemonSet 中将 ENABLE_POD_ENI 变量设置为 true，启用 Amazon VPC CNI 附加组件来管理容器组（pod）的网络接口。此设置一旦设置为 true 后，插件会为集群中的每个节点创建一个 cninode 自定义资源。VPC 资源控制器会创建并附加一个带有 aws-k8s-trunk-eni 描述且名为中继网络接口的特殊网络接口。

   ```
   kubectl set env daemonset aws-node -n kube-system ENABLE_POD_ENI=true
   ```

   > **Note**
   >
   > 中继网络接口包含在实例类型支持的最大网络接口数中。有关每种实例类型支持的最大网络接口数的列表，请参阅 Amazon EC2 用户指南中的每种实例类型的每个网络接口的 IP 地址数。如果您的节点已经附加了最大数量的标准网络接口，则 VPC 资源控制器将预订一个空间。您必须缩减正在运行的容器组（pod），以便控制器足以分离和删除标准网络接口、创建中继网络接口并将其附加到实例。

4. 您可以使用以下命令查看哪些节点具有 CNINode 自定义资源。如果返回 No resources found，则等几秒钟后重试。上一步需要重新启动适用于 Kubernetes 容器组（pod）的 Amazon VPC CNI 插件，这需要几秒钟的时间。

```
kubectl get cninode -A
     NAME FEATURES
     ip-192-168-64-141.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
     ip-192-168-7-203.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
```

如果您使用的是早于 1.15 的 VPC CNI 版本，则使用节点标签代替 CNINode 自定义资源。您可以使用以下命令查看哪些节点的节点标签 aws-k8s-trunk-eni 设置为 true。如果返回 No resources found，则等几秒钟后重试。上一步需要重新启动适用于 Kubernetes 容器组（pod）的 Amazon VPC CNI 插件，这需要几秒钟的时间。

```
kubectl get nodes -o wide -l vpc.amazonaws.com/has-trunk-attached=true
```

创建中继网络接口后，可以从中继或标准网络接口为容器组（pod）分配辅助 IP 地址。如果节点被删除，中继接口将自动删除。

当您在后面的步骤中部署容器组（pod）的安全组时，VPC 资源控制器会创建一个具有 aws-k8s-branch-eni 描述的名为分支网络接口的特殊网络接口，并将安全组与其关联。除了附加到节点的标准网络接口和中继网络接口之外，还会创建分支网络接口。

如果您使用的是存活探测器或就绪探测器，则您还需要禁用 TCP 早期解复用器，以便 kubelet 可以使用 TCP 连接到分支网络接口上的容器组（pod）。要禁用 TCP 早期解复用器，请运行以下命令：

```
kubectl patch daemonset aws-node -n kube-system \
  -p '{"spec": {"template": {"spec": {"initContainers": [{"env":
[{"name":"DISABLE_TCP_EARLY_DEMUX","value":"true"}],"name":"aws-vpc-cni-init"}]}}}}'
```

> **Note**
>
> 如果您使用的是 1.11.0 或更高版本的适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件并设置 POD_SECURITY_GROUP_ENFORCING_MODE=standard，如下一步所述，则您就不需要运行上一个命令。

5. 如果您的集群使用 NodeLocal DNSCache，或者您想将 Calico 网络策略与拥有自己的安全组的容器组（pod）结合使用，或者您有 NodePort 和 LoadBalancer 类型的 Kubernetes 服务，这些服务将实例目标（externalTrafficPolicy 设置为 Local）用于您向其分配安全组的容器组（pod），则您必须使用版本 1.11.0 或更高版本的适用于 Kubernetes 附加组件的 Amazon VPC CNI 插件，并且必须启用以下设置：

```
kubectl set env daemonset aws-node -n kube-system
 POD_SECURITY_GROUP_ENFORCING_MODE=standard
```

重要提示：容器组（pod）安全组规则不适用于位于相同节点上的容器组（pod）之间或介于容器组（pod）和服务之间的流量，例如 **kubelet** 或 **nodeLocalDNS**。在同一节点上使用不同安全组的容器组（pod）无法通信，因为它们配置在不同的子网中，且这些子网之间的路由被禁用。从容器组（pod）到 VPC 以外的地址的出站流量是转换为实例的主网络接口的 IP 地址的网络地址（除非您还设置了 AWS_VPC_K8S_CNI_EXTERNALSNAT=true）。对于此流量，将使用主网络接口的安全组中的规则，而不是容器组（pod）安全组中的规则。**要将此设置应用于现有容器组（pod），必须重新启动容器组（pod）或其上运行着容器组（pod）的节点。

6. 要了解如何为容器组（pod）使用安全组策略，请参阅the section called “SecurityGroupPolicy”。

对 Amazon EKS 容器组（pod）使用安全组策略

要使用容器组（pod）的安全组，您必须拥有现有的安全组。以下步骤展示了如何将安全组策略用于容器组（pod）。除非另有说明，请从同一个终端完成所有步骤，因为以下步骤中使用的变量不会在终端之间持续存在。

如果您具有带 Amazon EC2 实例的容器组（pod），则必须在使用此过程之前配置插件。有关更多信息，请参阅 the section called “配置”。

1. 创建要将资源部署到的 Kubernetes 命名空间。您可以将 *my-namespace* 替换为您要使用的命名空间名称。

```
kubectl create namespace my-namespace
```

2. 将 Amazon EKS SecurityGroupPolicy 部署到集群。

   a. 将以下内容复制到您的设备。如果您宁愿根据服务账户标签选择容器组（pod），则可以将 *podSelector* 替换为 serviceAccountSelector。您必须指定一个或其他选择器。空的 podSelector（示例：podSelector: {}）会选择命名空间中的所有容器组（pod）。您可以将 *my-role* 更改为您的角色名称。空的 serviceAccountSelector 会选择命名空间中的所

有服务账户。您可以将 *my-security-group-policy* 替换为 SecurityGroupPolicy 的名称，并将 *my-namespace* 替换为要在其中创建 SecurityGroupPolicy 的命名空间。

您必须将 *my_pod_security_group_id* 替换为现有安全组的 ID。如果您没有现有安全组，则必须创建一个安全组。有关更多信息，请参阅《Amazon EC2 用户指南》https://docs.amazonaws.cn/AWSEC2/latest/UserGuide/中的适用于 Linux 实例的 Amazon EC2 安全组。您可以指定 1-5 个安全组 ID。如果指定了多个 ID，则所有安全组中的所有规则的组合都会对选定的容器组（pod）生效。

```
cat >my-security-group-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-security-group-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: my-role
  securityGroups:
    groupIds:
      - my_pod_security_group_id
EOF
```

Important

您为容器组（pod）指定的一个或多个安全组必须符合以下标准：

- 它们必须存在。如果它们不存在，当您部署与选择器匹配的容器组（pod）时，该容器组（pod）会在创建过程中处于卡住状态。如果您描述容器组（pod），则会看到类似于以下内容的错误消息：`An error occurred (InvalidSecurityGroupID.NotFound) when calling the CreateNetworkInterface operation: The securityGroup ID 'sg-05b1d815d1EXAMPLE' does not exist`。
- 这些安全组必须允许通过您为其配置探测器的任何端口，从应用于您节点的安全组（对于 kubelet）进行入站通信。
- 这些安全组必须允许通过 TCP 和 UDP 端口 53 与分配到运行 CoreDNS 的容器组（pod）（或其上会运行容器组的节点）的安全组进行出站通信。适用于您的

CoreDNS 容器组（pod）的安全组必须允许来自您指定的安全组的入站 TCP 和 UDP 端口 53 流量。

- 它们必须具备必要的入站和出站规则，才能与它们需要与其通信的其他容器组（pod）进行通信。
- 如果您将安全组与 Fargate 一起使用，它们必须具有允许容器组（pod）与 Kubernetes 控制面板通信的规则。执行此操作的最简单方法是将集群安全组指定为安全组之一。

安全组策略仅适用于新调度的容器组（pod），不会影响正在运行的容器组（pod）。

b. 部署策略。

```
kubectl apply -f my-security-group-policy.yaml
```

3. 部署其标签与上一步中指定的 *podSelector* 的 *my-role* 值相匹配的示例应用程序。

a. 将以下内容复制到您的设备。将示例值替换为您自己的值，然后运行修改后的命令。如果您替换 *my-role*，请确保它与上一步中为选择器指定的值相同。

```
cat >sample-application.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  namespace: my-namespace
  labels:
    app: my-app
spec:
  replicas: 4
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        role: my-role
    spec:
      terminationGracePeriodSeconds: 120
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
        - containerPort: 80
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
EOF
```

b. 使用以下命令部署应用程序。当您部署应用程序时，适用于 Kubernetes 的 Amazon VPC CNI 插件将匹配 `role` 标签，并且您在上一步中指定的安全组将应用到容器组（pod）。

```
kubectl apply -f sample-application.yaml
```

4. 查看使用示例应用程序部署的容器组（pod）。对于本主题的其余部分，此终端称为 `TerminalA`。

```
kubectl get pods -n my-namespace -o wide
```

示例输出如下。

```
NAME                             READY   STATUS    RESTARTS   AGE     IP
  NODE                                                NOMINATED NODE   READINESS GATES
my-deployment-5df6f7687b-4fbjm   1/1     Running   0          7m51s   192.168.53.48
  ip-192-168-33-28.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-j9fl4   1/1     Running   0          7m51s   192.168.70.145
  ip-192-168-92-33.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-rjxcz   1/1     Running   0          7m51s   192.168.73.207
  ip-192-168-92-33.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-zmb42   1/1     Running   0          7m51s   192.168.63.27
  ip-192-168-33-28.region-code.compute.internal   <none>           <none>
```

> **Note**
>
> 如果容器组（pod）被卡住，请遵循以下提示。

- 如果任何容器组（pod）卡在 `Waiting` 状态，则运行 `kubectl describe pod *my-deployment-xxxxxxxxxx-xxxxx* -n *my-namespace*`。如果您看到了 `Insufficient permissions: Unable to create Elastic Network Interface.`，请确认您已在上一步中将 IAM 策略添加到 IAM 集群角色。
- 如果任何容器组（pod）卡在 `Pending` 状态，请确认您的节点实例类型已在 limits.go 中列出，并且尚未达到实例类型支持的最大分支网络接口数与您的节点组中节点数的乘积。例如，`m5.large` 实例支持 9 个分支网络接口。如果节点组有 5 个节点，则最多可以为节点组创建 45 个分支网络接口。在删除另一个具有关联安全组的容器组（pod）前，您尝试部署的第 46 个容器组（pod）将会处于 `Pending` 状态。

如果您运行 `kubectl describe pod *my-deployment-xxxxxxxxxx-xxxxx* -n *my-namespace*` 并看到类似于以下消息内容的消息，则可以安全地忽略该消息。当适用于 Kubernetes 的 Amazon VPC CNI 插件尝试在创建网络接口时设置主机联网并失败时，可能会出现此消息。该插件会记录此事件，直到创建了网络接口为止。

```
Failed to create Pod sandbox: rpc error: code = Unknown desc = failed to set up
 sandbox container "e24268322e55c8185721f52df6493684f6c2c3bf4fd59c9c121fd4cdc894579f"
 network for Pod "my-deployment-5df6f7687b-4fbjm": networkPlugin
cni failed to set up Pod "my-deployment-5df6f7687b-4fbjm-c89wx_my-namespace" network:
 add cmd: failed to assign an IP address to container
```

不能超过可在实例类型上运行的容器组（pod）的最大数量。有关可在每种实例类型上运行的容器组（pod）的最大数量的列表，请参阅 GitHub 上的 eni-max-pods.txt。当您删除具有关联安全组的容器组（pod）或删除运行该容器组的节点时，VPC 资源控制器会删除分支网络接口。如果您使用适用于安全组的容器组（pod）删除带有容器组的集群，则该控制器不会删除分支网络接口，因此您需要自行删除这些接口。有关如何删除网络接口的信息，请参阅《Amazon EC2 用户指南》中的删除网络接口。

5. 在单独的终端中，Shell 进入其中一个容器组（pod）。对于本主题的其余部分，此终端称为 `TerminalB`。请将 *`5df6f7687b-4fbjm`* 替换为您在上一步输出中返回的其中一个容器组（pod）的 ID。

```
kubectl exec -it -n my-namespace my-deployment-5df6f7687b-4fbjm -- /bin/bash
```

6. 从 `TerminalB` 的 Shell 中，确认示例应用程序是否有效。

```
curl my-app
```

示例输出如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

您收到了输出，因为运行该应用程序的所有容器组（pod）与您创建的安全组关联。该组包含一条规则，允许安全组所关联的所有容器组（pod）之间的所有流量。允许 DNS 流量从该安全组出站到与您的节点关联的集群安全组。节点正在运行 CoreDNS 容器组（pod），您的容器组对其进行了名称查找。

7. 从 TerminalA 中，删除允许从安全组与集群安全组进行 DNS 通信的安全组规则。如果您在上一步中没有将 DNS 规则添加到集群安全组中，请将 *$my_cluster_security_group_id* 替换为您在其中创建规则的安全组的 ID。

```
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_tcp_rule_id
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_udp_rule_id
```

8. 从 TerminalB 中，尝试再次访问该应用程序。

```
curl my-app
```

示例输出如下。

```
curl: (6) Could not resolve host: my-app
```

由于容器组（pod）不再能够访问具有与其关联的集群安全组的 CoreDNS 容器组，尝试失败。集群安全组不再具有安全组规则，该规则允许从与您的容器组（pod）关联的安全组进行 DNS 通信。

如果您尝试使用上一步中的其中一个容器组（pod）返回的 IP 地址访问应用程序，您仍然会收到响应，因为在与安全组相关联的容器组之间允许所有端口，并且不需要名称查找。

9. 完成实验后，您可以移除您创建的示例安全组策略、应用程序和安全组。从 TerminalA 运行以下命令。

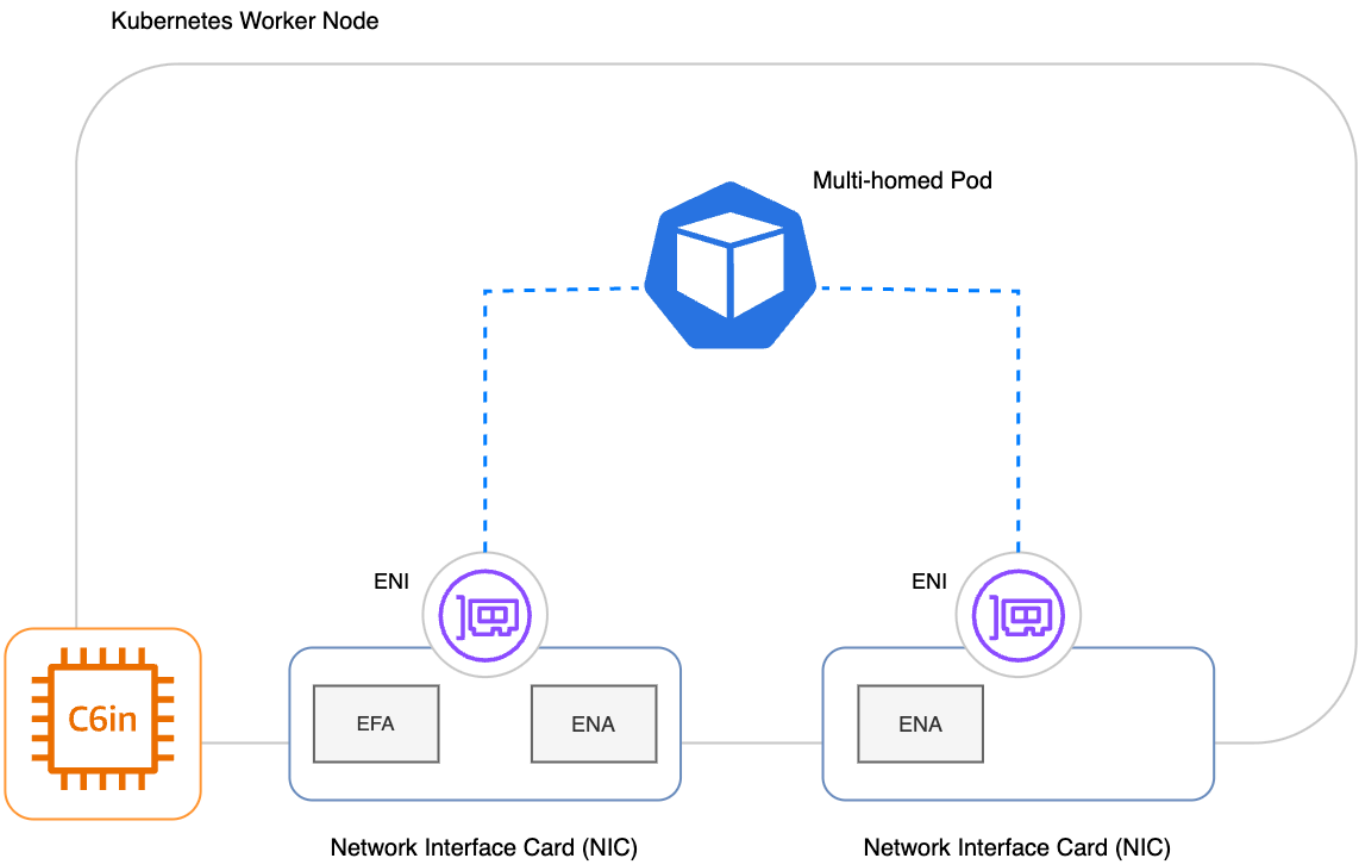
```
kubectl delete namespace my-namespace
```

```
aws ec2 revoke-security-group-ingress --group-id $my_pod_security_group_id --
security-group-rule-ids $my_inbound_self_rule_id
wait
sleep 45s
aws ec2 delete-security-group --group-id $my_pod_security_group_id
```

将多个网络接口连接到容器组（pod）

默认情况下，Amazon VPC CNI 插件为每个容器组（pod）分配一个 IP 地址。此 IP 地址连接到一个弹性网络接口，该接口负责处理容器组（pod）的所有传入和传出流量。要提高带宽和每秒数据包速率性能，您可以使用 VPC CNI 的多 NIC 功能配置多宿主容器组（pod）。多宿主容器组（pod）是使用多个网络接口（和多个 IP 地址）的单个 Kubernetes 容器组（pod）。通过运行多宿主容器组（pod），您可以使用并发连接将其应用程序流量分散到多个网络接口。这对于人工智能（AI）、机器学习（ML）和高性能计算（HPC）使用案例特别有用。

下图显示一个在使用多个网络接口卡（NIC）的 Worker 节点上运行的多宿主容器组（pod）。



背景

在 Amazon EC2 上，弹性网络接口是 VPC 中的逻辑联网组件，表示虚拟网卡。对于许多 EC2 实例类型，网络接口在硬件中共享一个网络接口卡（NIC）。此单个 NIC 具有最大带宽和每秒数据包速率。

如果启用了多 NIC 功能，则 VPC CNI 不会像默认那样批量分配 IP 地址。相反，新容器组（pod）启动后，VPC CNI 会按需为每个网卡上的网络接口分配一个 IP 地址。这种行为可降低因使用多宿主容器组（pod）而增加的 IP 地址耗尽率。由于 VPC CNI 按需分配 IP 地址，因此在启用多 NIC 功能的实例上，容器组（pod）可能需要更长的时间才能启动。

注意事项

- 确保您的 Kubernetes 集群运行的是 VPC CNI 版本 `1.20.0` 及更高版本。多 NIC 功能仅在 VPC CNI `1.20.0` 或更高版本中可用。
- 在 VPC CNI 插件中启用 `ENABLE_MULTI_NIC` 环境变量。您可以运行以下命令来设置该变量并开始部署 DaemonSet。
  - `kubectl set env daemonset aws-node -n kube-system ENABLE_MULTI_NIC=true`
- 确保创建具有多个网络接口卡（NIC）的 Worker 节点。有关具有多个网络接口卡的 EC2 实例列表，请参阅《Amazon EC2 用户指南》中的网卡。
- 如果启用了多 NIC 功能，则 VPC CNI 不会像默认那样批量分配 IP 地址。由于 VPC CNI 按需分配 IP 地址，因此在启用多 NIC 功能的实例上，容器组（pod）可能需要更长的时间才能启动。有关更多信息，请参阅前一章节the section called “背景”。
- 启用多 NIC 功能后，容器组（pod）默认没有多个网络接口。必须将每个工作负载配置为使用多 NIC。为应具有多个网络接口的工作负载添加 `k8s.amazonaws.com/nicConfig: multi-nic-attachment` 注解。

IPv6 注意事项

- 自定义 IAM 策略：对于 IPv6 集群，请为 VPC CNI 创建并使用以下自定义 IAM 策略。此策略专用于多 NIC。有关将 VPC CNI 与 IPv6 集群结合使用的更多一般信息，请参阅 the section called “IPv6”。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonEKSCNIPolicyIPv6MultiNIC",
            "Effect": "Allow",
```

```
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeInstances",
                "ec2:AssignIpv6Addresses",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeTags",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeInstanceTypes",
                "ec2:UnassignIpv6Addresses",
                "ec2:AttachNetworkInterface",
                "ec2:DescribeSubnets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmazonEKSCNIPolicyENITagIPv6MultiNIC",
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:*:*:network-interface/*"
        }
    ]
}
```

- IPv6 转换机制不可用：如果您使用多 NIC 功能，则 VPC CNI 不会为 IPv6 集群上的容器组（pod）分配 IPv4 地址。否则，VPC CNI 会为每个容器组（pod）分配一个主机本地 IPv4 地址，这样容器组（pod）便可与其他 Amazon VPC 或互联网中的外部 IPv4 资源进行通信。

用法

在 VPC CNI 中启用多 NIC 功能并且 aws-node 容器组（pod）重新启动后，您可以将每个工作负载配置为多宿主。以下是带有所需注解的 YAML 配置示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders-deployment
  namespace: ecommerce
  labels:
    app: orders
spec:
  replicas: 3
```

```
  selector:
    matchLabels:
      app: orders
  template:
    metadata:
      annotations:
         k8s.amazonaws.com/nicConfig: multi-nic-attachment
      labels:
        app: orders
    spec:
...
```

常见问题

1. 什么是网络接口卡（NIC）？

网络接口卡（NIC），也简称网卡，是一种支持底层云计算硬件网络连接的物理设备。在现代 EC2 服务器中，这指的是 Nitro 网卡。弹性网络接口（ENI）是该底层网卡的虚拟表示。

某些 EC2 实例类型有多个 NIC，以提升带宽和数据包速率性能。对于此类实例，您可以将辅助 ENI 分配给其他网卡。例如，ENI #1 可以作为连接到网卡索引 0 的 NIC 的接口，而 ENI #2 可以作为连接到单独网卡索引的 NIC 的接口。

2. 什么是多宿主容器组（pod）？

多宿主容器组（pod）是具有多个网络接口（也意味着多个 IP 地址）的单个 Kubernetes 容器组（pod）。每个容器组（pod）网络接口都关联一个弹性网络接口（ENI），而这些 ENI 是底层 Worker 节点上独立 NIC 的逻辑表示。通过多个网络接口，多宿主容器组（pod）具有额外的数据传输容量，从而也提升了其数据传输速率。

> **Important**
>
> VPC CNI 只能在具有多个 NIC 的实例类型上配置多宿主容器组（pod）。

3. 为什么应该使用此功能？

如果您需要在基于 Kubernetes 的工作负载中扩展网络性能，可以使用多 NIC 功能运行多宿主容器组（pod），这些容器组（pod）与连接 ENA 设备的所有底层 NIC 相连。通过将应用程序流量分布到多个并发连接，利用额外的网卡可以提高应用程序的带宽容量和数据包速率性能。这对于人工智能（AI）、机器学习（ML）和高性能计算（HPC）使用案例特别有用。

4. 如何使用此功能？

1. 首先，必须确保您的 Kubernetes 集群使用的是 VPC CNI 版本 1.20 或更高版本。有关将 VPC CNI 更新为 EKS 附加组件的步骤，请参阅the section called “更新（EKS 附加组件）”。
2. 然后，您必须使用 ENABLE_MULTI_NIC 环境变量在 VPC CNI 中启用多 NIC 支持。
3. 接着，您必须确保创建并加入具有多个网卡的节点。有关具有多个网卡的 EC2 实例类型列表，请参阅《Amazon EC2 用户指南》中的网卡。
4. 最后，您可以将每个工作负载配置为使用多个网络接口（多宿主容器组（pod））或使用单个网络接口。

5. 如何将我的工作负载配置为在支持的 Worker 节点上使用多个 NIC？

要使用多宿主容器组（pod），您需要添加以下注解：k8s.amazonaws.com/nicConfig: multi-nic-attachment。这会将底层实例中每个 NIC 的 ENI 连接到容器组（pod）（容器组（pod）与 NIC 之间是一对多映射）。

如果缺少此注解，VPC CNI 会假设您的容器组（pod）只需要 1 个网络接口，并从任何可用 NIC 上的 ENI 为其分配一个 IP。

6. 此功能支持哪些网络接口适配器？

如果至少有一个 ENA 连接到底层网卡用于传输 IP 流量，则可以使用任何网络接口适配器。有关 ENA 的更多信息，请参阅《Amazon EC2 用户指南》中的弹性网络适配器（ENA）。

支持的网络设备配置：

- ENA 接口提供支持 VPC 的 IP 联网所需的所有传统 IP 联网和路由功能。有关更多信息，请参阅在 EC2 实例上使用 ENA 启用增强联网功能。
- EFA（带 ENA 的 EFA）接口既提供用于 IP 联网的 ENA 设备，又提供用于低延迟、高吞吐量通信的 EFA 设备。

> **Important**
>
> 如果网卡仅连接了仅限 EFA 适配器，则在为多宿主容器组（pod）预置网络连接时，VPC CNI 将跳过。但是，如果您在网卡上将仅限 EFA 适配器与 ENA 适配器结合使用，则 VPC CNI 也将管理此设备上的 ENI。要将仅限 EFA 接口与 EKS 集群结合使用，请参阅the section called “使用 EFA 设置训练集群”。

7. 我可以查看集群中的某个节点是否支持 ENA 吗？

是的，您可以使用 Amazon CLI 或 EC2 API 来检索有关集群中 EC2 实例的网络信息。这将提供有关该实例是否支持 ENA 的详细信息。在以下示例中，将 <your-instance-id> 替换为节点的 EC2 实例 ID。

Amazon CLI 示例：

```
aws ec2 describe-instances --instance-ids <your-instance-id> --query
 "Reservations[].Instances[].EnaSupport"
```

输出示例：

```
[ true ]
```

8. 我可以查看与容器组（pod）关联的不同 IP 地址吗？

这不太容易。但您可以从节点使用 nsenter 运行常用网络工具（例如 ip route show）并查看其他 IP 地址和接口。

9. 我可以控制控制容器组（pod）的网络接口数量吗？

不可以。当您的工作负载配置为在支持的实例上使用多个 NIC 时，单个容器组（pod）会自动从该实例上的每个网卡获得一个 IP 地址。或者，单宿主容器组（pod）将有一个网络接口连接到实例上的一个 NIC。

> **Important**
>
> VPC CNI 会跳过仅连接仅限 EFA 设备的网卡。

10. 我可以将容器组（pod）配置为使用特定的 NIC 吗？

不可以，不支持此功能。如果某个容器组（pod）有相关注解，则 VPC CNI 会自动将其配置为使用 Worker 节点上带有 ENA 适配器的每个 NIC。

11. 此功能可以与其他 VPC CNI 联网功能结合使用吗？

可以，VPC CNI 中的多 NIC 功能可以与自定义联网和增强型子网发现功能结合使用。但是，多宿主容器组（pod）不会使用自定义子网或安全组。相反，VPC CNI 会将 IP 地址和网络接口分配给与节

点具有相同子网和安全组配置的多宿主容器组（pod）。有关自定义联网功能的更多信息，请参阅 the section called “自定义联网”。

VPC CNI 中的多 NIC 功能不适用且无法与容器组（pod）的安全组结合使用。

12. 我可以将网络策略与此功能结合使用吗？

可以，您可以将 Kubernetes 网络策略与多 NIC 结合使用。Kubernetes 网络策略可限制进出容器组（pod）的网络流量。有关使用 VPC CNI 应用网络策略的更多信息，请参阅the section called “Kubernetes 策略”。

13. EKS 自动模式下是否启用了多 NIC 支持？

EKS 自动模式集群不支持多 NIC。

适用于 Amazon EKS 集群的备用 CNI 插件

适用于 Kubernetes 的 Amazon VPC CNI 插件是使用 Amazon EC2 节点的 Amazon EKS 唯一支持的 CNI 插件。对于 Amazon EKS 混合节点功能，Amazon EKS 支持 Cilium 和 Calico 的核心功能。Amazon EKS 运行上游 Kubernetes，因此您可以将备选的兼容 CNI 插件安装到集群中的 Amazon EC2 节点。如果集群中包含 Fargate 节点，则适用于 Kubernetes 的 Amazon VPC CNI 插件已在 Fargate 节点上。这是唯一可与 Fargate 节点结合使用的 CNI 插件。尝试在 Fargate 节点上安装备用 CNI 插件失败。

如果您计划在 Amazon EC2 节点上使用备用 CNI 插件，我们建议您获得插件的商业支持，或者利用内部专业知识对 CNI 插件项目进行故障排除并提供修复。

Amazon EKS 与为备选的兼容 CNI 插件提供支持的合作伙伴网络保持着合作关系。有关版本、资格和所执行测试的详细信息，请参阅以下合作伙伴的文档。

合作伙伴	产品	文档
Tigera	Calico	安装说明
Isovalent	Cilium	安装说明
Juniper	云原生 Contrail 联网（CN2）	安装说明
VMware	Antrea	安装说明

Amazon EKS 旨在为您提供广泛的选项来涵盖所有使用案例。

可选的兼容网络策略插件

Calico 是一种广泛采用的容器网络和安全解决方案。在 EKS 上使用 Calico 可为 EKS 集群提供完全合规的网络策略实施。此外，也可选择使用 Calico 的网络，便于保护底层 VPC 的 IP 地址。Calico Cloud 增强了 Calico Open Source 的功能，因此可提供更高的安全性与可观测性。

流向和来自具有关联安全组的 Pod 的流量不受 Calico 网络策略执行的限制，并且仅受限于 Amazon VPC 安全组执行。

如果使用 Calico 网络策略实施，则建议您将环境变量 `ANNOTATE_POD_IP` 设置为 `true`，以避免出现 Kubernetes 的已知问题。要使用此功能，必须将容器组（pod）的 `patch` 权限添加到 `aws-node` ClusterRole。请注意，向 `aws-node` DaemonSet 添加 patch 权限会提高插件的安全范围。有关更多信息，请参阅 GitHub 上的 VPC CNI 存储库中的 ANNOTATE_POD_IP。

Amazon EKS 自动模式的注意事项

Amazon EKS 自动模式不支持备用 CNI 插件或网络策略插件。有关更多信息，请参阅 EKS 自动模式。

使用 Multus 将多个网络接口附加到容器组（pod）

Multus CNI 是一个适用于 Amazon EKS 的容器网络接口（CNI）插件，可将多个网络接口附加到容器组（pod）。有关更多信息，请参阅 GitHub 上的 Multus-CNI 文档。

在 Amazon EKS 中，每个容器组（pod）都有一个由 Amazon VPC CNI 插件分配的网络接口。使用 Multus，您可以创建一个具有多个接口的多宿主容器组（pod）。这是由作为“元插件”的 Multus 完成的；一个 CNI 插件，可以调用多个其他 CNI 插件。Amazon 对 Multus 的支持将 Amazon VPC CNI 插件配置为默认委托插件。

- Amazon EKS 将不会构建和发布单个根输入/输出虚拟化（SR-IOV）和数据面板开发套件（DPDK）CNI 插件。但是，您可以通过 Multus 托管主机设备和 `ipvlan` 插件直接连接到 Amazon EC2 弹性网络适配器 (ENA) 来实现数据包加速。
- Amazon EKS 支持 Multus，它提供了一个通用的流程，可以简单地连接其他 CNI 插件。支持 Multus 和连接过程，但 Amazon 不会支持所有可以连接的兼容 CNI 插件，或者那些与连接配置无关的 CNI 插件中可能出现的问题。
- Amazon EKS 正在为 Multus 插件提供支持和生命周期管理，但不对与其它网络接口相关的任何 IP 地址或其它管理负责。使用 Amazon VPC CNI 插件的默认网络接口的 IP 地址和管理保持不变。

- 官方仅支持 Amazon VPC CNI 插件作为默认委派插件。如果您选择不对主联网使用 Amazon VPC CNI 插件，则需要修改已发布的 Multus 安装清单，以将默认委派插件重新配置为备用 CNI。
- 仅当使用 Amazon VPC CNI 作为主 CNI 时，才支持 Multus。当用于更高阶接口时，我们不支持 Amazon VPC CNI，无论是次要接口还是其他。
- 为防止 Amazon VPC CNI 插件尝试管理分配给容器组（pod）的其他网络接口，请将以下标签添加到网络接口：

  密钥

  ```
  : node.k8s.amazonaws.com/no_manage
  ```

  值

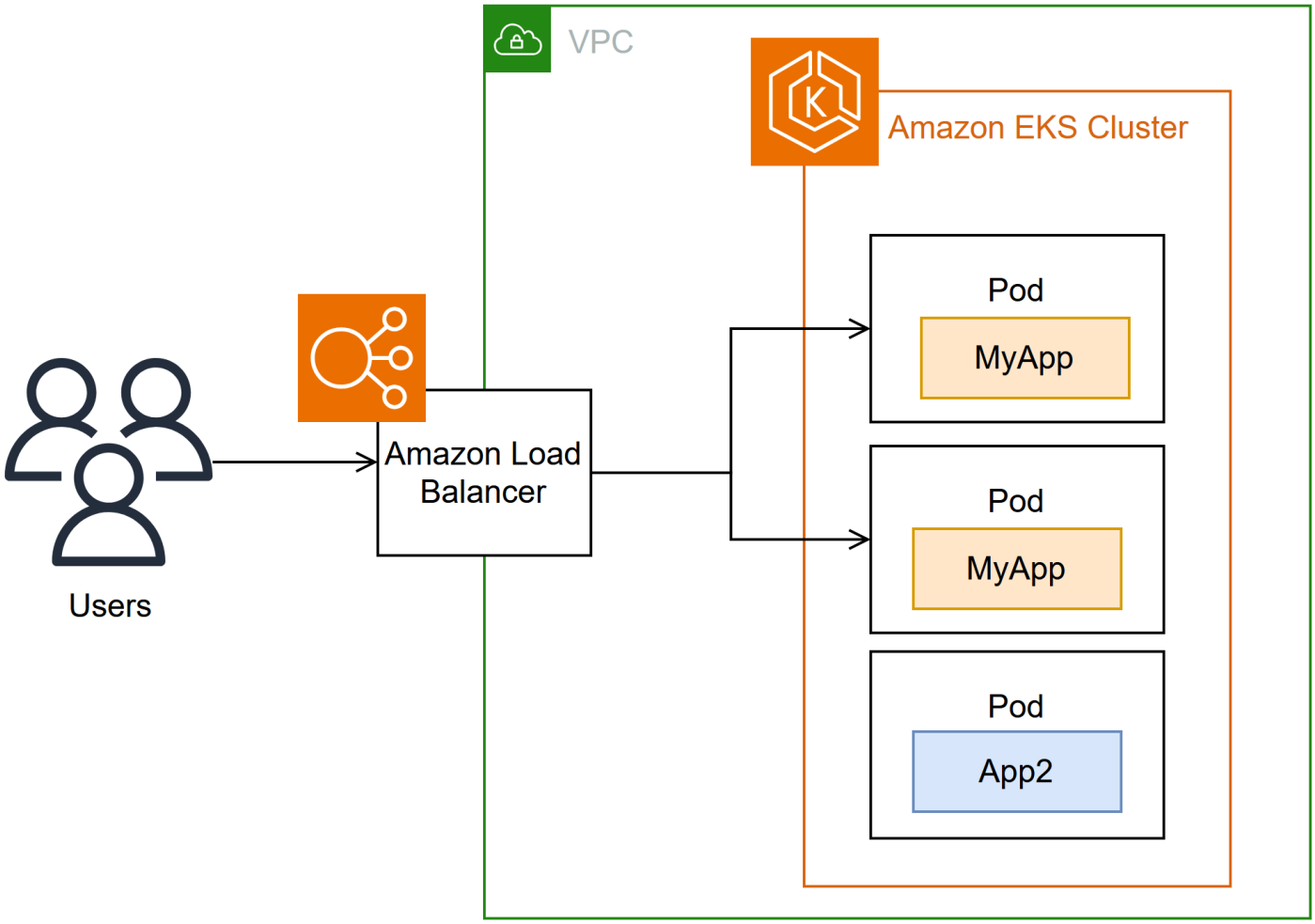
  ```
  : true
  ```

- Multus 与网络策略兼容，但必须对策略进行充实，以包括某些端口和 IP 地址，而其可能是附加到容器组（pod）的其他网络接口的一部分。

有关实施演练，请参阅 GitHub 上的 Mulus 设置指南。

使用 Amazon 负载均衡器控制器路由互联网流量

Amazon负载均衡器控制器管理适用于 Kubernetes 集群的 Amazon Elastic Load Balancer。您可以使用控制器将您的集群应用程序公开到互联网。控制器预调配指向集群服务或入口资源的 Amazon 负载均衡器。换句话说，控制器创建一个指向集群中多个容器组（pod）的 IP 地址或 DNS 名称。



控制器会监控 Kubernetes 入口或服务资源。作为响应，它会创建相应的 Amazon 弹性负载均衡资源。您可以通过对 Kubernetes 资源应用注释来配置负载均衡器的特定行为。例如，您可以使用注释将 Amazon 安全组附加到负载均衡器。

此控制器预置以下资源：

Kubernetes **`Ingress`**

LBC 会在您创建 Kubernetes `Ingress` 时创建 Amazon 应用程序负载均衡器（ALB）。查看可以应用于入口资源的注释。

`LoadBalancer` 类型的 Kubernetes 服务

LBC 将在您创建 `LoadBalancer` 类型的 Kubernetes 服务时创建 Amazon 网络负载均衡器（NLB）。查看可以应用于服务资源的注释。

在过去，实例目标使用 Kubernetes 网络负载均衡器，而 IP 目标使用 LBC。现在，有了 Amazon 负载均衡器控制器版本 `2.3.0` 或更高版本，您可以使用任一目标类型创建 NLB。有关 NLB 目标类型的更多信息，请参阅 Network Load Balancer 用户指南中的目标类型。

控制器是托管在 GitHub 上的开源项目。

在部署控制器之前，我们建议您查看使用应用程序负载均衡器路由应用程序和 HTTP 流量和the section called “网络负载均衡”中的先决条件和注意事项。在这些主题中，您将部署包含 Amazon 负载均衡器的示例应用程序。

Kubernetes **`Gateway`** API

对于 Amazon 负载均衡器控制器 `2.14.0` 版本或更高版本，LBC 会在您创建 Kubernetes `Gateway` 时创建 Amazon 应用程序负载均衡器（ALB）。与 Ingress 相比，Kubernetes 网关标准化的配置更多，前者需要为许多常用选项自定义注释。查看可以应用于网关资源的配置。有关 `Gateway` API 的更多信息，请参阅 Kubernetes 文档中的 Gateway API。

安装控制器

可以按照以下过程之一安装 Amazon 负载均衡器控制器：

- 如果您不熟悉 Amazon EKS，我们建议您使用 Helm 进行安装，因为它可以简化 Amazon 负载均衡器控制器安装。有关更多信息，请参阅 the section called “使用 Helm 安装”。
- 对于高级配置，例如对公有容器注册表具有有限网络访问权限的集群，请使用 Kubernetes 清单。有关更多信息，请参阅 the section called “使用清单安装 ”。

从已弃用的控制器版本迁移

- 如果您安装了已弃用的 Amazon 负载均衡器控制器版本，请参阅the section called “从已弃用的控制器迁移应用程序”。
- 已弃用的版本无法升级。必须将其移除并安装最新版本的 Amazon 负载均衡器控制器。
- 已弃用的版本包括：
  - 适用于 Kubernetes 的 Amazon ALB 入口控制器（简称“入口控制器”），是 Amazon 负载均衡器控制器的前身。
  - 任何 `0.1.`*`x`* 版本的 Amazon 负载均衡器控制器

旧版云提供商

Kubernetes 包括 Amazon 的旧版云提供商。旧版云提供商能够预置 Amazon 负载均衡器，这与 Amazon 负载均衡器控制器类似。传统云提供商创建经典负载均衡器。如果不安装 Amazon 负载均衡器控制器，则 Kubernetes 将默认设置为使用旧版云提供商。您应该安装 Amazon 负载均衡器控制器，避免使用旧版云提供商。

> **Important**
>
> 在版本 2.5 及更高版本中，Amazon 负载均衡器控制器成为 Kubernetes 服务资源的默认控制器（包含 `type: LoadBalancer`），并为每个服务创建了 Amazon 网络负载均衡器（NLB）。为此它将为服务创建一个变异的 Webhook，对于新的 `type: LoadBalancer` 服务，后者会将 `spec.loadBalancerClass` 字段设置为 `service.k8s.aws/nlb`。您可以关闭此功能，然后将 Helm 图表值 `enableServiceMutatorWebhook` 设置为 `false`，从而恢复将 传统云提供商 作为默认控制器。除非您关闭此功能，否则集群不会为您的服务预置新的经典负载均衡器。现有的经典负载均衡器将继续运行。

使用 Helm 安装 Amazon 负载均衡器控制器

> **Tip**
>
> 使用 Amazon EKS 自动模式时，您无需安装或升级联网附加组件。自动模式包含容器组联网和负载均衡功能。
> 有关更多信息，请参阅 EKS 自动模式。

本主题介绍如何使用 Helm（Kubernetes 的程序包管理器）和 `eksctl` 安装 Amazon 负载均衡器控制器。控制器安装了默认的选项。有关控制器的更多信息，包括使用注释对其进行配置的详细信息，请参阅 GitHub 上的 Amazon Load Balancer Controller 文档。

在以下步骤中，将 example values 替换为您自己的值。

先决条件

在开始本教程之前，请完成以下步骤：

- 创建一个 Amazon EKS 集群。要创建该文件，请参阅 开始使用。

- 在本地计算机上安装 Helm。
- 确保适用于 Kubernetes 的 Amazon VPC CNI 插件、kube-proxy 和 CoreDNS 附加组件为服务账户令牌中列出的最低版本。
- 了解 Amazon Elastic Load Balancing 概念。有关更多信息，请参阅 Elastic Load Balancing 用户指南。
- 了解 Kubernetes 服务和入口资源。

注意事项

在继续执行本页面的配置步骤之前，请注意以下事项：

- IAM 策略和角色 (AmazonEKSLoadBalancerControllerRole) 可以在同一 Amazon 账户中的多个 EKS 集群中重复使用。
- 如果要在最初创建角色 (AmazonEKSLoadBalancerControllerRole) 的同一集群上安装控制器，请先确认角色存在，然后转到步骤 2：安装负载均衡器控制器。
- 如果使用的是服务账户的 IAM 角色（IRSA），则必须为每个集群设置 IRSA，并且角色信任策略中的 OpenID Connect（OIDC）提供者 ARN 应特定于每个 EKS 集群。此外，如果要在具有现有 AmazonEKSLoadBalancerControllerRole 的新集群上安装控制器，请更新该角色的信任策略以包含新集群的 OIDC 提供者，并通过适当的角色注释创建一个新的服务账户。要确定是否已经拥有 OIDC 提供者或者要创建一个 OIDC 提供者，请参阅the section called “IAM OIDC 提供者”。

步骤 1：使用 **eksctl** 创建 IAM 角色

以下步骤参考 Amazon 负载均衡器控制器 v2.14.1 发行版。有关所有发行版的更多信息，请参阅 GitHub 上的 Amazon Load Balancer Controller Release 页面。

1. 下载Amazon负载均衡器控制器的 IAM 策略，该策略允许负载均衡器代表您调用 Amazon API。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.14.1/docs/install/iam_policy.json
```

   - 对于政府区域或中国区域等非标准 Amazon 分区，请查看 GitHub 上的政策并下载适用于您所在区域的相应政策。

2. 使用上一步中下载的策略创建一个 IAM 策略。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
```

```
    --policy-document file://iam_policy.json
```

> **Note**
>
> 如果您在Amazon Web Services 管理控制台中查看策略，则控制台会显示有关 ELB 服务的警告，但不会显示有关 ELB v2 服务的警告。之所以发生这种情况，是因为策略中的某些操作适用于 ELB v2，但不适用于 ELB。您可以忽略有关 ELB 的警告。

3. 替换集群名称、区域代码和账户 ID 的值。

```
eksctl create iamserviceaccount \
    --cluster=<cluster-name> \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
    --attach-policy-arn=arn:aws-cn:iam::<AWS_ACCOUNT_ID>:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region <aws-region-code> \
    --approve
```

步骤 2：安装 Amazon 负载均衡器控制器

1. 添加 `eks-charts` Helm 图表存储库。Amazon 在 GitHub 上维护此存储库。

```
helm repo add eks https://aws.github.io/eks-charts
```

2. 更新您的本地存储库，以确保您拥有最新的图表。

```
helm repo update eks
```

3. 安装Amazon负载均衡器控制器。

   如果要将控制器部署到限制访问 Amazon EC2 实例元数据服务（IMDS）的 Amazon EC2 节点，或者部署到 Fargate 节点或 Amazon EKS 混合节点功能，则需要在以下 `helm` 命令中添加以下标志：

   - `--set region=`*`region-code`*
   - `--set vpcId=`*`vpc-xxxxxxxx`*

   将 *`my-cluster`* 替换为您的集群的名称。在以下命令中，`aws-load-balancer-controller` 是您在上一步中创建的 Kubernetes 服务账户。

有关配置 Helm 图表的更多信息，请参阅 GitHub 上的 values.yaml。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=my-cluster \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
  --version 1.14.0
```

> **Important**
>
> 已部署的图表不会自动接收安全更新。当新图表可用时，您需要手动升级到新图表。升级时，在上一个命令中将 *install* 更改为 upgrade。

helm install 命令会自动安装控制器的自定义资源定义（CRD）。helm upgrade 命令不会。如果您使用 helm upgrade，则必须手动安装 CRD。运行以下命令来安装 CRD：

```
wget https://raw.githubusercontent.com/aws/eks-charts/master/stable/aws-load-balancer-
controller/crds/crds.yaml
kubectl apply -f crds.yaml
```

步骤 3：验证控制器是否已安装

1. 验证控制器是否已安装。

   ```
   kubectl get deployment -n kube-system aws-load-balancer-controller
   ```

   示例输出如下。

   ```
   NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
   aws-load-balancer-controller   2/2     2            2           84s
   ```

   如果使用 Helm 进行部署，则会收到之前的输出结果。如果您使用 Kubernetes 清单进行部署，则只有一个副本。

2. 在使用控制器预置Amazon资源前，您的集群必须满足特定要求。有关更多信息，请参阅the section called “应用程序负载均衡”和the section called “网络负载均衡”。

使用清单安装 Amazon 负载均衡器控制器

> **Tip**
>
> 使用 Amazon EKS 自动模式时，您无需安装或升级联网附加组件。自动模式包含容器组联网和负载均衡功能。
> 有关更多信息，请参阅 EKS 自动模式。

本主题介绍如何通过下载和应用 Kubernetes 清单来安装控制器。您可以在 GitHub 上查看关于该控制器的完整文档。

在以下步骤中，将 example values 替换为您自己的值。

先决条件

在开始本教程之前，请完成以下步骤：

- 创建一个 Amazon EKS 集群。要创建该文件，请参阅 开始使用。
- 在本地计算机上安装 Helm。
- 确保适用于 Kubernetes 的 Amazon VPC CNI 插件、`kube-proxy` 和 CoreDNS 附加组件为服务账户令牌中列出的最低版本。
- 了解 Amazon Elastic Load Balancing 概念。有关更多信息，请参阅 Elastic Load Balancing 用户指南。
- 了解 Kubernetes 服务和入口资源。

注意事项

在继续执行本页面的配置步骤之前，请注意以下事项：

- IAM 策略和角色 (`AmazonEKSLoadBalancerControllerRole`) 可以在同一 Amazon 账户中的多个 EKS 集群中重复使用。
- 如果要在最初创建角色 (`AmazonEKSLoadBalancerControllerRole`) 的同一集群上安装控制器，请先确认角色存在，然后转到步骤 2：安装 cert-manager。
- 如果使用的是服务账户的 IAM 角色（IRSA），则必须为每个集群设置 IRSA，并且角色信任策略中的 OpenID Connect（OIDC）提供者 ARN 应特定于每个 EKS 集群。此外，如果要在具有现有 `AmazonEKSLoadBalancerControllerRole` 的新集群上安装控制器，请更新该角色的信任策略

以包含新集群的 OIDC 提供者，并通过适当的角色注释创建一个新的服务账户。要确定是否已经拥有 OIDC 提供者或者要创建一个 OIDC 提供者，请参阅the section called “IAM OIDC 提供者”。

步骤 1：配置 IAM

以下步骤参考 Amazon 负载均衡器控制器 v2.14.1 发行版。有关所有发行版的更多信息，请参阅 GitHub 上的 Amazon Load Balancer Controller Release 页面。

1. 下载Amazon负载均衡器控制器的 IAM 策略，该策略允许负载均衡器代表您调用 Amazon API。

   Example

   Amazon

   ```
   curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/install/iam_policy.json
   ```

   Amazon GovCloud (US)

   ```
   curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/install/iam_policy_us-gov.json
   ```

   ```
   mv iam_policy_us-gov.json iam_policy.json
   ```

2. 使用上一步中下载的策略创建一个 IAM 策略。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

> **Note**
>
> 如果您在Amazon Web Services 管理控制台中查看策略，则控制台会显示有关 ELB 服务的警告，但不会显示有关 ELB v2 服务的警告。之所以发生这种情况，是因为策略中的某些操作适用于 ELB v2，但不适用于 ELB。您可以忽略有关 ELB 的警告。

Example

eksctl

a. 请将 *my-cluster* 替换为您的集群的名称，将 *111122223333* 替换为您的账户 ID，然后运行命令。

```
eksctl create iamserviceaccount \
  --cluster=my-cluster \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws-cn:iam::111122223333:policy/
AWSLoadBalancerControllerIAMPolicy \
  --approve
```

Amazon CLI and kubectl

a. 检索集群的 OIDC 提供者 ID 并将其存储在变量中。

```
oidc_id=$(aws eks describe-cluster --name my-cluster --query
 "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

b. 确定账户中是否已存在具有集群 ID 的 IAM OIDC 提供者。您需要为集群和 IAM 配置 OIDC。

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

如果返回输出，则表示集群已有 IAM OIDC 提供者。如果没有返回输出，则您必须为集群创建 IAM OIDC 提供者。有关更多信息，请参阅 the section called “IAM OIDC 提供者”。

c. 将以下内容复制到您的设备。请将 *111122223333* 替换为您的账户 ID。将 *region-code* 替换为您的集群所在的 Amazon 区域。将 *EXAMPLED539D4633E53DE1B71EXAMPLE* 替换为上一步中返回的输出。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
```

```
                "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
load-balancer-controller"
                }
            }
        }
    ]
}
```

d. 创建 IAM 角色。

```
aws iam create-role \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --assume-role-policy-document file://"load-balancer-role-trust-policy.json"
```

e. 将所需的 Amazon EKS 托管 IAM policy 附加到 IAM 角色。请将 *111122223333* 替换为您的账户 ID。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::111122223333:policy/
AWSLoadBalancerControllerIAMPolicy \
  --role-name AmazonEKSLoadBalancerControllerRole
```

f. 将以下内容复制到您的设备。请将 *111122223333* 替换为您的账户 ID。替换文本后，运行修改后的命令可创建 `aws-load-balancer-controller-service-account.yaml` 文件。

```
cat >aws-load-balancer-controller-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
```

```
  annotations:
    eks.amazonaws.com/role-arn: arn:aws-cn:iam::111122223333:role/
AmazonEKSLoadBalancerControllerRole
EOF
```

g. 在集群上创建 Kubernetes 服务账户。使用您创建的名为 *AmazonEKSLoadBalancerControllerRole* 的 IAM 角色注释名为 `aws-load-balancer-controller` 的 Kubernetes 服务账户。

```
kubectl apply -f aws-load-balancer-controller-service-account.yaml
```

步骤 2：安装 **`cert-manager`**

使用以下方法之一安装 `cert-manager` 以将证书配置注入 Webhook。有关更多信息，请参阅 cert-manager 文档中的 Getting Started。

我们建议使用 `quay.io` 容器注册表安装 `cert-manager`。如果您的节点无权访问 `quay.io` 容器注册表，则请使用 Amazon ECR 安装 `cert-manager`（见下文）。

Example

Quay.io

a. 如果您的节点有权访问 `quay.io` 容器注册表，请安装 `cert-manager` 以将证书配置注入 Webhook。

```
kubectl apply \
    --validate=false \
    -f https://github.com/jetstack/cert-manager/releases/download/v1.13.5/cert-
manager.yaml
```

Amazon ECR

a. 使用以下方法之一安装 `cert-manager` 以将证书配置注入 Webhook。有关更多信息，请参阅 cert-manager 文档中的 Getting Started。
b. 下载清单。

```
curl -Lo cert-manager.yaml https://github.com/jetstack/cert-manager/releases/
download/v1.13.5/cert-manager.yaml
```

c. 请提取以下镜像并将其推送到节点有权访问的存储库。有关如何提取、标记和推送镜像到您自己的存储库的更多信息，请参阅 the section called “将镜像复制到存储库”。

```
quay.io/jetstack/cert-manager-cainjector:v1.13.5
quay.io/jetstack/cert-manager-controller:v1.13.5
quay.io/jetstack/cert-manager-webhook:v1.13.5
```

d. 请将三个镜像的清单中的 quay.io 替换为您自己的注册表名称。以下命令假定您的私有存储库名称与源存储库的名称相同。请将 *111122223333.dkr.ecr.region-code.amazonaws.com* 替换为您的私有注册表。

```
sed -i.bak -e 's|quay.io|111122223333.dkr.ecr.region-code.amazonaws.com|' ./cert-manager.yaml
```

e. 运用该清单。

```
kubectl apply \
    --validate=false \
    -f ./cert-manager.yaml
```

步骤 3：安装 Amazon 负载均衡器控制器

1. 下载控制器规范。有关控制器的更多信息，请参阅 GitHub 上的文档。

```
curl -Lo v2_14_1_full.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.14.1/v2_14_1_full.yaml
```

2. 对文件进行以下编辑。

a. 如果您已下载 v2_14_1_full.yaml 文件，请运行以下命令以删除清单中的 ServiceAccount 部分。如果您不删除此部分，系统将覆盖您在上一步中对服务账户所做的必需注释。如果删除此部分，若您删除了控制器，系统还会保留您在上一步中创建的服务账户。

```
sed -i.bak -e '764,772d' ./v2_14_1_full.yaml
```

如果您已下载其他文件版本，请在编辑器中打开此文件，然后删除以下行。

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
---
```

b. 请将 *my-cluster* 替换为您的集群名称，将文件的 Deployment spec 部分中的 your-cluster-name 替换为您的集群名称。

```
sed -i.bak -e 's|your-cluster-name|my-cluster|' ./v2_14_1_full.yaml
```

c. 如果您的节点无权访问 Amazon EKS Amazon ECR 镜像存储库，则需要提取以下镜像并将其推送到节点有权访问的存储库。有关如何提取、标记和推送镜像到您自己的存储库的更多信息，请参阅 the section called “将镜像复制到存储库”。

```
public.ecr.aws/eks/aws-load-balancer-controller:v2.14.1
```

请为清单添加注册表的名称。以下命令假定您的私有存储库名称与源存储库的名称相同，并将您的私有注册表名称添加到文件中。请将 *111122223333.dkr.ecr.region-code.amazonaws.com* 替换为您的注册表。此行假定您的私有存储库名称与源存储库的名称相同。如果不相同，请将私有注册表名称后面的 eks/aws-load-balancer-controller 文本给更改为您的存储库名称。

```
sed -i.bak -e 's|public.ecr.aws/eks/aws-load-balancer-controller|
111122223333.dkr.ecr.region-code.amazonaws.com/eks/aws-load-balancer-
controller|' ./v2_14_1_full.yaml
```

d. （只有 Fargate 或受限的 IMDS 才需要此选项）

如果要将控制器部署到限制访问 Amazon EC2 实例元数据服务（IMDS）的 Amazon EC2 节点，或者部署到 Fargate 节点或 Amazon EKS 混合节点功能，则需要在 - args: 下添加 following parameters。

```
[...]
spec:
      containers:
        - args:
            - --cluster-name=your-cluster-name
            - --ingress-class=alb
            - --aws-vpc-id=vpc-xxxxxxxx
```

```
            - --aws-region=region-code


[...]
```

3. 应用文件。

```
kubectl apply -f v2_14_1_full.yaml
```

4. 将 IngressClass 和 IngressClassParams 清单下载到您的集群。

```
curl -Lo v2.14.1_ingclass.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.14.1/v2_14_1_ingclass.yaml
```

5. 将清单应用于集群。

```
kubectl apply -f v2_14_1_ingclass.yaml
```

步骤 4：验证控制器是否已安装

1. 验证控制器是否已安装。

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

   示例输出如下。

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

   如果使用 Helm 进行部署，则会收到之前的输出结果。如果您使用 Kubernetes 清单进行部署，则只有一个副本。

2. 在使用控制器预置Amazon资源前，您的集群必须满足特定要求。有关更多信息，请参阅the section called “应用程序负载均衡”和the section called “网络负载均衡”。

从已弃用的 ALB 入口控制器迁移应用程序

本主题介绍如何从已弃用的控制器版本迁移。更具体地说，是介绍如何移除 Amazon 负载均衡器控制器的已弃用版本。

- 已弃用的版本无法升级。必须先将其移除，然后再安装当前版本。
- 已弃用的版本包括：
  - 适用于 Kubernetes 的 Amazon ALB 入口控制器（简称“入口控制器”），是 Amazon 负载均衡器控制器的前身。
  - 任何 `0.1.x` 版本的 Amazon 负载均衡器控制器

移除已弃用的控制器版本

> **Note**
>
> 您可能已经使用 Helm 安装了已弃用的版本，或者使用 Kubernetes 清单手动安装了该版本。使用原来安装它的工具完成该过程。

1. 如果您安装了 `incubator/aws-alb-ingress-controller` Helm 图表，请将其卸载。

```
helm delete aws-alb-ingress-controller -n kube-system
```

2. 如果安装了 `eks-charts/aws-load-balancer-controller` 图表的 `0.1.x` 版本，请将其卸载。由于与 Webhook API 版本不兼容，从 `0.1.x` 升级到版本 `1.0.0` 不起作用。

```
helm delete aws-load-balancer-controller -n kube-system
```

3. 检查当前是否安装了该控制器。

```
kubectl get deployment -n kube-system alb-ingress-controller
```

这是未安装控制器情况下的输出。

```
Error from server (NotFound): deployments.apps "alb-ingress-controller" not found
```

这是安装了控制器情况下的输出。

```
NAME                      READY UP-TO-DATE AVAILABLE AGE
alb-ingress-controller 1/1   1          1         122d
```

4. 输入以下命令以删除控制器。

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/alb-ingress-controller.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/rbac-role.yaml
```

迁移到 Amazon 负载均衡器控制器

要从适用于 Kubernetes 的 ALB 入口控制器迁移到 Amazon 负载均衡器控制器，您需要：

1. 移除 ALB 入口控制器（请参阅上述内容）。
2. 安装Amazon负载均衡器控制器。
3. 向 Amazon 负载均衡器控制器使用的 IAM 角色添加其他策略。此策略允许 LBC 管理由适用于 Kubernetes 的 ALB 入口控制器创建的资源。
4. 下载该 IAM 策略。此策略允许 Amazon 负载均衡器控制器管理由适用于 Kubernetes 的 ALB 入口控制器创建的资源。您还可以查看策略。

   ```
   curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/install/iam_policy_v1_to_v2_additional.json
   ```
5. 如果您的集群位于 Amazon GovCloud（美国东部）或 Amazon GovCloud（美国西部）Amazon 区域，则将 `arn:aws-cn:` 替换为 `arn:aws-us-gov:`。

   ```
   sed -i.bak -e 's|arn:aws-cn:|arn:aws-us-gov:|' iam_policy_v1_to_v2_additional.json
   ```
6. 创建 IAM 策略并记下返回的 ARN。

   ```
   aws iam create-policy \
     --policy-name AWSLoadBalancerControllerAdditionalIAMPolicy \
     --policy-document file://iam_policy_v1_to_v2_additional.json
   ```
7. 将 IAM 策略附加到 Amazon 负载均衡器控制器使用的 IAM 角色。将 *`your-role-name`* 替换为角色的名称，例如 `AmazonEKSLoadBalancerControllerRole`。

   如果您使用 `eksctl` 创建该角色，请找到已创建的角色名称，打开 Amazon CloudFormation 控制台并选择 eksctl-***`my-cluster`***-addon-iamserviceaccount-kube-system-aws-load-balancer-controller 堆栈。选择资源选项卡。角色名称位于 Physical ID（物理 ID）列。

   ```
   aws iam attach-role-policy \
   ```

```
  --role-name your-role-name \
  --policy-arn arn:aws-cn:iam::111122223333:policy/
AWSLoadBalancerControllerAdditionalIAMPolicy
```

在 Amazon EKS 集群中管理 DNS 的 CoreDNS

> **Tip**
>
> 使用 Amazon EKS 自动模式时，您无需安装或升级联网附加组件。自动模式包含容器组联网和负载均衡功能。
> 有关更多信息，请参阅 EKS 自动模式。

CoreDNS 是一个灵活、可扩展的 DNS 服务器，可用作 Kubernetes 集群 DNS。当您启动具有至少一个节点的 Amazon EKS 集群时，无论集群中部署的节点数量如何，预设情况下都会部署 CoreDNS 镜像的两个副本。CoreDNS Pod 为集群中的所有 Pod 提供名称解析。如果您的集群包含命名空间与 CoreDNS `deployment` 的命名空间相匹配的 Fargate 配置文件，则可以将 CoreDNS 容器组部署到 Fargate 节点。有关 CoreDNS 的更多信息，请参阅 Kubernetes 文档中的使用 CoreDNS 进行服务发现。

CoreDNS 版本

下表列出了每个 Kubernetes 版本的 Amazon EKS 附加组件类型的最新版本。

Kubernetes 版本	CoreDNS 版本
1.34	v1.12.4-eksbuild.1
1.33	v1.12.4-eksbuild.1
1.32	v1.11.4-eksbuild.24
1.31	v1.11.4-eksbuild.24
1.30	v1.11.4-eksbuild.24
1.29	v1.11.4-eksbuild.24
1.28	v1.10.1-eksbuild.38

> **Important**
>
> 如果您自行管理此附加组件，则表中的版本可能与可用的自行管理版本不同。有关更新此附加组件的自行管理类型的更多信息，请参阅the section called “更新（自主管理型）”。

重要的 CoreDNS 升级注意事项

- CoreDNS 更新利用 PodDisruptionBudget 来帮助在更新过程中维持 DNS 服务的可用性。
- 为了提高 CoreDNS 部署的稳定性和可用性，`v1.9.3-eksbuild.6` 版本及更高版本和 `v1.10.1-eksbuild.3` 使用 `PodDisruptionBudget` 进行部署。如果您已部署现有 `PodDisruptionBudget`，则升级到这些版本可能会失败。如果升级失败，则完成以下任务之一应可解决问题：
  - 升级 Amazon EKS 附加组件时，选择覆盖现有设置作为冲突解决选项。如果您对部署进行了其他自定义设置，请务必在升级之前备份您的设置，以便在升级后可以重新应用其他自定义设置。
  - 删除现有的 `PodDisruptionBudget`，然后再次尝试升级。
- 在 EKS 附加组件 `v1.9.3-eksbuild.3` 及更高版本和 `v1.10.1-eksbuild.6` 及更高版本中，CoreDNS 部署将 `readinessProbe` 设置为使用 `/ready` 端点。此端点已在 CoreDNS 的 `Corefile` 配置文件中启用。

  如果您使用自定义 `Corefile`，则必须将 `ready` 插件添加到该配置，以便 `/ready` 端点在 CoreDNS 中处于活动状态，供探测器使用。
- 在 EKS 附加组件版本 `v1.9.3-eksbuild.7` 及更高版本 和 `v1.10.1-eksbuild.4` 及更高版本中，您可以更改 `PodDisruptionBudget`。您可以使用以下示例中的字段编辑附加组件并在可选配置设置中更改这些设置。此示例显示默认 `PodDisruptionBudget`。

```
{
    "podDisruptionBudget": {
        "enabled": true,
        "maxUnavailable": 1
        }
}
```

  您可以设置 `maxUnavailable` 或 `minAvailable`，但不能在一个 `PodDisruptionBudget` 中同时设置两者。有关 `PodDisruptionBudgets` 的更多信息，请参阅《Kubernetes 文档》中的指定 PodDisruptionBudget。

请注意，如果将 enabled 设置为 false，则不会删除 PodDisruptionBudget。将此字段设置为 false 后，必须删除 PodDisruptionBudget 对象。同样，如果您在升级到带有 PodDisruptionBudget 的版本后编辑附加组件，以使用该附加组件的旧版本（降级附加组件），则不会删除 PodDisruptionBudget。要删除 PodDisruptionBudget，您可以运行以下命令：

```
kubectl delete poddisruptionbudget coredns -n kube-system
```

- 在 EKS 插件版本 v1.10.1-eksbuild.5 及更高版本中，将默认容差从 node-role.kubernetes.io/master:NoSchedule 更改为 node-role.kubernetes.io/control-plane:NoSchedule 以符合 KEP 2067。有关 KEP 2067 的更多信息，请参阅 GitHub 上的 Kubernetes 增强建议（KEP）中的 KEP-2067: Rename the kubeadm "master" label and taint。

  在 EKS 附加组件 v1.8.7-eksbuild.8 及更高版本和 v1.9.3-eksbuild.9 及更高版本中，两个公差都设置为与每个 Kubernetes 版本兼容。

- 在 EKS 附加组件 v1.9.3-eksbuild.11 和 v1.10.1-eksbuild.7 及更高版本中，CoreDNS 部署为 topologySpreadConstraints 设置一个默认值。如果多个可用性区域内都有节点，则该默认值可确保 CoreDNS 容器组（pod）分布在可用性区域内。您可以设置一个用于代替默认值的自定义值。默认值如下所示：

```
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

CoreDNS **v1.11** 升级注意事项

- 在 EKS 附加组件版本 v1.11.1-eksbuild.4 以及更高版本中，容器映像基于 Amazon EKS Distro 维护的最小基本映像，其中包含最少的软件包且没有外壳。有关更多信息，请参阅 Amazon EKS Distro。CoreDNS 映像的使用和问题排查保持不变。

创建 CoreDNS Amazon EKS 附加组件

创建 CoreDNS Amazon EKS 附加组件。在创建附加组件之前，您必须拥有一个集群。有关更多信息，请参阅 the section called “创建集群”。

1. 查看集群上当前安装的附加组件版本。

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
d : -f 3
```

示例输出如下。

```
v1.10.1-eksbuild.13
```

2. 查看集群上当前安装的附加组件类型。根据您创建集群时使用的工具，您的集群上目前可能没有安装 Amazon EKS 附加组件类型。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

如果返回来的是版本号，则表明您的集群上安装有 Amazon EKS 类型的附加组件，而且此流程中其余的步骤，您也不需要走完。如果返回来的是一个错误，则表明您的集群上没有安装 Amazon EKS 类型的附加组件。要安装，就需完成此流程中其余的步骤。

3. 保存您当前安装的附加组件的配置。

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

4. 使用 Amazon CLI 创建附加组件。如果要使用 Amazon Web Services 管理控制台 或 `eksctl` 来创建附加组件，请参阅 the section called “创建附加组件” 并指定 `coredns` 为附加组件名称。将以下命令复制到您的设备。根据需要对该命令进行以下修改，然后运行修改后的命令。

   - 将 *my-cluster* 替换为您的集群的名称。
   - 将 *v1.11.3-eksbuild.1* 替换为集群版本的最新版本（在最新版本表中列出）。

```
aws eks create-addon --cluster-name my-cluster --addon-name coredns --addon-version
 v1.11.3-eksbuild.1
```

如果您对当前附加组件应用的自定义设置与 Amazon EKS 附加组件的默认设置相冲突，则创建可能会失败。如果创建失败，您会收到一条可以帮助您解决问题的错误信息。或者，您可以将 `--`

`resolve-conflicts OVERWRITE` 添加到前面的命令中。这样一来，附加组件会覆盖任何现有的自定义设置。创建附加组件后，您可以使用自定义设置对其进行更新。

5. 确认集群的 Kubernetes 版本的附加组件最新版本已添加到集群。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

附加组件创建可能需要几秒钟才能完成。

示例输出如下。

```
v1.11.3-eksbuild.1
```

6. 如果您在创建 Amazon EKS 附加组件之前对原始附加组件进行了自定义设置，则请使用您在上一步中保存的配置，以使用您的自定义设置更新 Amazon EKS 附加组件。有关如何更新附加组件的说明，请参阅 the section called “更新（EKS 附加组件）”。

更新 CoreDNS Amazon EKS 附加组件

更新 附加组件的 Amazon EKS 类型。如果您尚未将 Amazon EKS 附件组件添加到集群，请添加它或参阅the section called “更新（自主管理型）”。

在开始之前，请查看升级的注意事项。有关更多信息，请参阅 the section called “重要的 CoreDNS 升级注意事项”。

1. 查看集群上当前安装的附加组件版本。将 *my-cluster* 替换为您的集群名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 "addon.addonVersion" --output text
```

示例输出如下。

```
v1.10.1-eksbuild.13
```

如果返回的版本与最新版本表中集群的 Kubernetes 版本相同，则表示集群上已经安装最新版本，您无需完成此过程的其余部分。如果您在输出中收到错误信息而不是版本号，则您的集群上没有安装 Amazon EKS 类型的附加组件。您需要先创建附加组件，然后才能使用此过程对其进行更新。

2. 保存您当前安装的附加组件的配置。

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

3. 使用 Amazon CLI 更新您的附加组件。如果您想要使用 Amazon Web Services 管理控制台 或 eksctl 更新附加组件，则请参阅 the section called “更新附加组件”。将以下命令复制到您的设备。根据需要对该命令进行以下修改，然后运行修改后的命令。
   - 将 *my-cluster* 替换为您的集群的名称。
   - 将 *v1.11.3-eksbuild.1* 替换为集群版本的最新版本（在最新版本表中列出）。
   - --resolve-conflicts*PRESERVE* 选项保留附加组件的现有配置值。如果您为附加组件设置设定了自定义值，但未使用此选项，则 Amazon EKS 会使用其默认值覆盖您的值。如果您使用此选项，那么我们建议您在更新生产集群上的附加组件之前，先测试非生产集群上所有更改的字段和值。如果您将该值改为 OVERWRITE，则所有设置都将更改为 Amazon EKS 的默认值。如果您为任何设置设定了自定义值，这些值可能会被 Amazon EKS 的默认值覆盖。如果您将该值改为 none，Amazon EKS 不会更改任何设置的值，但更新可能会失败。如果更新失败，您会收到一条帮助您解决冲突的错误消息。
   - 如果您没有更新配置设置，则请从命令中移除 --configuration-values '{*"replicaCount":3*}'。如果您要更新配置设置，请将 *"replicaCount":3* 替换为要设置的设置。在此示例中，CoreDNS 的副本数设置为 3。您指定的值必须对配置架构有效。如果您不知道配置架构，请运行 aws eks describe-addon-configuration --addon-name coredns --addon-version *v1.11.3-eksbuild.1* ，并将 *v1.11.3-eksbuild.1* 替换为您要查看配置的附加组件的版本号。将在输出中返回架构。如果您有任何现有自定义配置，想要将其全部删除，并将所有设置的值设置回 Amazon EKS 默认值，请从命令中删除 *"replicaCount":3* ，这样就可以有空的 {}。有关 CoreDNS 设置的更多信息，请参阅 Kubernetes 文档中的自定义 DNS 服务。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns --addon-version
 v1.11.3-eksbuild.1 \
    --resolve-conflicts PRESERVE --configuration-values '{"replicaCount":3}'
```

   可能需要几秒钟才能完成更新。

4. 确认附加组件版本已更新。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns
```

   可能需要几秒钟才能完成更新。

示例输出如下。

```
{
    "addon": {
        "addonName": "coredns",
        "clusterName": "my-cluster",
        "status": "ACTIVE",
        "addonVersion": "v1.11.3-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws-cn:eks:region:111122223333:addon/my-cluster/coredns/
d2c34f06-1111-2222-1eb0-24f64ce37fa4",
        "createdAt": "2023-03-01T16:41:32.442000+00:00",
        "modifiedAt": "2023-03-01T18:16:54.332000+00:00",
        "tags": {},
        "configurationValues": "{\"replicaCount\":3}"
    }
}
```

更新 CoreDNS Amazon EKS 自主管理型附加组件

> **Important**
>
> 建议您向集群添加 Amazon EKS 类型的附加组件，而不是自行管理类型的附加组件。如果不熟悉这些类型之间的区别，请参阅 the section called “Amazon EKS 附加组件”。有关向集群中添加 Amazon EKS 附加组件的更多信息，请参阅 the section called “创建附加组件”。如果您无法使用 Amazon EKS 附加组件，我们鼓励您向容器路线图 GitHub 存储库提交有关您为什么无法使用的问题。

在开始之前，请查看升级的注意事项。有关更多信息，请参阅 the section called “重要的 CoreDNS 升级注意事项”。

1. 确认已在集群上安装自行管理类型的附加组件。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

如果返回错误消息，则表明集群上安装有自行管理类型的附加组件。完成此过程的其余步骤。如果返回版本号，则表明集群上安装有 Amazon EKS 类型的附加组件。要更新附加组件的 Amazon EKS 类型，请使用更新 CoreDNS Amazon EKS 附加组件中的过程，而不是此过程。如果不熟悉这些附加组件类型之间的区别，请参阅 the section called “Amazon EKS 附加组件”。

2. 查看集群上当前安装的容器映像版本。

```
kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
```

示例输出如下。

```
v1.8.7-eksbuild.2
```

3. 如果当前 CoreDNS 版本是 v1.5.0 或更高版本，但低于 CoreDNS 版本表中列出的版本，请跳过此步骤。如果当前版本低于 1.5.0，则需修改 ConfigMap，以使 CoreDNS 使用转发附加组件，而不是代理附加组件。

   a. 使用以下命令打开 ConfigMap。

   ```
   kubectl edit configmap coredns -n kube-system
   ```

   b. 将以下行中的 proxy 替换为 forward。保存文件，然后退出编辑器。

   ```
   proxy . /etc/resolv.conf
   ```

4. 如果您最初在 Kubernetes 1.17 或更早版本上部署了集群，则可能需要从 CoreDNS 清单中删除已停用的行。

> **Important**
>
> 在更新到 CoreDNS 版本 1.7.0 之前，您必须完成此步骤，但是即使要更新到早期版本，我们也建议您完成此步骤。

   a. 检查您的 CoreDNS 清单是否包含该行。

   ```
   kubectl get configmap coredns -n kube-system -o jsonpath='{$.data.Corefile}' |
    grep upstream
   ```

如果未返回任何输出，则说明清单没有此类行，您可以跳到下一步来更新 CoreDNS。如果返回了输出，则需要删除该行。

b. 使用以下命令编辑 ConfigMap，删除文件中包含词语 upstream 的该行。不要更改文件中的任何其他内容。删除该行后，保存更改。

```
kubectl edit configmap coredns -n kube-system -o yaml
```

5. 检索当前的 CoreDNS 映像版本：

```
kubectl describe deployment coredns -n kube-system | grep Image
```

示例输出如下。

```
602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.8.7-eksbuild.2
```

6. 如果要更新到 CoreDNS 1.8.3 或更高版本，则需要将 endpointslices 权限添加到 system:coredns Kubernetes clusterrole。

```
kubectl edit clusterrole system:coredns -n kube-system
```

在文件中的 rules 部分的现有权限行下添加以下行。

```
[...]
- apiGroups:
  - discovery.k8s.io
  resources:
  - endpointslices
  verbs:
  - list
  - watch
[...]
```

7. 通过将 *602401143452* 和 *region-code* 替换为上一步中返回的输出的值，更新 CoreDNS 附加组件。将 *v1.11.3-eksbuild.1* 替换为 Kubernetes 版本的最新版本表中列出的 CoreDNS 版本。

```
kubectl set image deployment.apps/coredns -n kube-system
 coredns=602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.11.3-
eksbuild.1
```

示例输出如下。

```
deployment.apps/coredns image updated
```

8. 再次检查容器映像版本，确认它已更新到您在上一步中指定的版本。

```
kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
```

示例输出如下。

```
v1.11.3-eksbuild.1
```

扩展 CoreDNS 容器以获得高 DNS 流量

启动具有至少一个节点的 Amazon EKS 集群时，无论集群中部署的节点数是多少，默认情况下都会部署 CoreDNS 映像的两个副本。CoreDNS Pod 为集群中的所有 Pod 提供名称解析。应用程序会通过名称解析连接到集群中的容器组和服务，并连接到集群外部的服务。随着容器组（pod）的名称解析（查询）请求数量增加，CoreDNS 容器组（pod）可能会不堪重负，继而减速，并拒绝容器组（pod）无法处理的请求。

要处理 CoreDNS 容器组（pod）上增加的负载，可以考虑使用 CoreDNS 的自动扩缩系统。Amazon EKS 可以在 CoreDNS 的 EKS 附加组件版本中管理 CoreDNS 部署的自动扩缩。此 CoreDNS 自动扩缩器会持续监控集群状态，包括节点数量和 CPU 核心数量。控制器会根据这些信息，动态调整 EKS 集群中 CoreDNS 部署的副本数量。此功能适用于 CoreDNS v1.9 及更高版本。有关与 CoreDNS 自动扩缩兼容的版本的更多信息，请参阅以下章节。

系统使用基于集群中节点数和 CPU 内核数的动态公式自动管理 CoreDNS 副本，计算方法为（numberOfNodes 除以 16）与（numberOfCPUCores 除以 256）的最大值。系统会评估 10 分钟高峰期内的需求，在需要时立即扩展以处理增加的 DNS 查询负载，同时通过每 3 分钟减少 33% 的副本数来逐步缩减，以保持系统稳定性并避免中断。

我们建议将此功能与其他 EKS 集群自动扩缩最佳实践结合使用，以提高应用程序的整体可用性和集群可扩展性。

先决条件

要让 Amazon EKS 扩展 CoreDNS 部署，需要满足三个先决条件：

- 您必须使用 CoreDNS 的 EKS 附加组件版本。

- 集群必须至少运行最低集群版本和平台版本。
- 集群运行的 CoreDNS EKS 附加组件版本必须至少是最低版本。

最低集群版本

由集群控制面板中的新组件完成 CoreDNS 自动扩缩，该组件由 Amazon EKS 管理。因此，您必须将集群升级到支持的最低平台版本（且其中包含新组件）的 EKS 版本。

新的 Amazon EKS 集群。要部署一个角色，请参阅开始使用。集群必须运行下表中列出的 Kubernetes 版本和平台版本之一或更高版本。请注意，也支持所有比所列版本更高的 Kubernetes 和平台版本。您可以通过将以下命令中的 *my-cluster* 替换为集群名称，然后运行修改后的命令来检查当前的 Kubernetes 版本：

```
aws eks describe-cluster --name my-cluster --query cluster.version --output text
```

Kubernetes 版本	平台版本
未列出	所有平台版本
1.29.3	eks.7
1.28.8	eks.13
1.27.12	eks.17
1.26.15	eks.18

> **Note**
>
> 还支持更高 Kubernetes 版本的各平台版本，例如来自 eks.1 的 Kubernetes 版本 1.30 及更高版本。

EKS 附加组件最低版本

Kubernetes 版本	1.29	1.28
	v1.11.1-eksbuild.9	v1.10.1-eksbuild.11

在 Amazon Web Services 管理控制台 中配置 CoreDNS 自动扩缩

1. 确保集群版本为最低版本或更高版本。

   Amazon EKS 会自动在同一 Kubernetes 版本的平台版本之间升级集群，您无法自行启动此过程。不过，您可以将集群升级到下一个 Kubernetes 版本，而后，集群将升级到该 K8s 版本和最新平台版本。

   新的 Kubernetes 版有时会引入重大更改。因此，我们建议您先使用新 Kubernetes 版本的单个集群来测试应用程序的行为，然后再更新生产集群。

   要将集群升级到新的 Kubernetes 版本，请按照将现有集群更新到新的 Kubernetes 版本中的过程操作。
2. 确保具有 CoreDNS 的 EKS 附加组件，而不是自主管理型 CoreDNS 部署。

   根据您创建集群时使用的工具，您的集群上目前可能没有安装 Amazon EKS 附加组件类型。要查看集群上安装了哪种类型的附加组件，可以运行以下命令。将 `my-cluster` 替换为您的集群名称。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   如果返回版本号，则表明集群上安装有 Amazon EKS 类型的附加组件，您可以继续执行下一步。如果返回来的是一个错误，则表明您的集群上没有安装 Amazon EKS 类型的附加组件。完成创建 CoreDNS Amazon EKS 附加组件过程的剩余步骤，将自行管理的版本替换为 Amazon EKS 附加组件。
3. 确保 CoreDNS 的 EKS 附加组件的版本与 EKS 附加组件版本相同或高于其最低版本。

   查看集群上当前安装的附加组件版本。您可以在 Amazon Web Services 管理控制台 中查看或运行以下命令：

   ```
   kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
   d : -f 3
   ```

示例输出如下。

```
v1.10.1-eksbuild.13
```

将此版本与上一节中的 EKS 附加组件最低版本进行比较。如果需要，请按照以下更新 CoreDNS Amazon EKS 附加组件过程将 EKS 附加组件升级为更高版本。

4. 将自动扩缩配置添加到 EKS 附加组件的可选配置设置中。
   a. 打开 Amazon EKS 控制台。
   b. 在左侧导航窗格中，选择 Clusters（集群），然后选择要为其配置附加组件的集群的名称。
   c. 选择附加组件选项卡。
   d. 选择 CoreDNS 附加组件框右上角的框，然后选择编辑。
   e. 在配置 CoreDNS 页面上：
      i. 选择您想使用的版本。我们建议您保留与上一步相同的版本，并通过单独的操作更新版本和配置。
      ii. 展开可选配置设置。
      iii. 在配置值中输入 JSON 键 `"autoscaling":`，以及键为 `"enabled":` 且值为 `true` 的嵌套 JSON 对象的值。生成的文本必须是有效的 JSON 对象。如果此键和值是文本框中的唯一数据，请用大括号 `{ }` 将键和值括起来。以下示例显示了已启用的自动扩缩：

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

      iv. （可选）可以提供自动扩缩所能扩展的 CoreDNS 容器组（pod）数量的最小值和最大值。

      以下示例显示了已启用的自动扩缩以及所有带值的可选键。我们建议 CoreDNS 容器组（pod）的最小数量始终大于 2，以便为集群中的 DNS 服务提供弹性。

```
{
  "autoScaling": {
    "enabled": true,
    "minReplicas": 2,
    "maxReplicas": 10
  }
```

```
}
```

f. 要通过替换 CoreDNS 容器组（pod）来应用新配置，请选择保存更改。

Amazon EKS 通过推出适用于 CoreDNS 的 Kubernetes 部署，将更改应用到 EKS 附加组件。可在 Amazon Web Services 管理控制台 中的附加组件更新历史记录中跟踪推出状态和 `kubectl rollout status deployment/coredns --namespace kube-system`。

`kubectl rollout` 具有以下命令：

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

如果推出时间过长，Amazon EKS 将撤销推出，并且会将类型为附加组件更新且状态为失败的消息添加到附加组件的更新历史记录中。要调查任何问题，请先查看推出历史记录，然后在 CoreDNS 容器组（pod）上运行 `kubectl logs`，进而查看 CoreDNS 的日志。

5. 如果更新历史记录中的新条目为成功状态，则表示推出已完成，并且附加组件正在所有 CoreDNS 容器组（pod）中使用新配置。更改集群中的节点数量和 CPU 核心数量时，Amazon EKS 会扩展 CoreDNS 部署的副本数量。

在 Amazon 命令行界面中配置 CoreDNS 自动扩缩

1. 确保集群版本为最低版本或更高版本。

Amazon EKS 会自动在同一 Kubernetes 版本的平台版本之间升级集群，您无法自行启动此过程。不过，您可以将集群升级到下一个 Kubernetes 版本，而后，集群将升级到该 K8s 版本和最新平台版本。

新的 Kubernetes 版有时会引入重大更改。因此，我们建议您先使用新 Kubernetes 版本的单个集群来测试应用程序的行为，然后再更新生产集群。

要将集群升级到新的 Kubernetes 版本，请按照将现有集群更新到新的 Kubernetes 版本中的过程操作。

2. 确保具有 CoreDNS 的 EKS 附加组件，而不是自主管理型 CoreDNS 部署。

   根据您创建集群时使用的工具，您的集群上目前可能没有安装 Amazon EKS 附加组件类型。要查看集群上安装了哪种类型的附加组件，可以运行以下命令。将 `my-cluster` 替换为您的集群名称。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   如果返回版本号，则表明集群上安装有 Amazon EKS 类型的附加组件。如果返回来的是一个错误，则表明您的集群上没有安装 Amazon EKS 类型的附加组件。完成创建 CoreDNS Amazon EKS 附加组件过程的剩余步骤，将自行管理的版本替换为 Amazon EKS 附加组件。

3. 确保 CoreDNS 的 EKS 附加组件的版本与 EKS 附加组件版本相同或高于其最低版本。

   查看集群上当前安装的附加组件版本。您可以在 Amazon Web Services 管理控制台 中查看或运行以下命令：

   ```
   kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
   d : -f 3
   ```

   示例输出如下。

   ```
   v1.10.1-eksbuild.13
   ```

   将此版本与上一节中的 EKS 附加组件最低版本进行比较。如果需要，请按照以下更新 CoreDNS Amazon EKS 附加组件过程将 EKS 附加组件升级为更高版本。

4. 将自动扩缩配置添加到 EKS 附加组件的可选配置设置中。

   运行以下 Amazon CLI 命令。将 `my-cluster` 替换为集群的名称，并将 IAM 角色 ARN 替换为您正在使用的角色。

   ```
   aws eks update-addon --cluster-name my-cluster --addon-name coredns \
       --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
   {"enabled":true}}'
   ```

   Amazon EKS 通过推出适用于 CoreDNS 的 Kubernetes 部署，将更改应用到 EKS 附加组件。可在 Amazon Web Services 管理控制台 中的附加组件更新历史记录中跟踪推出状态和 `kubectl rollout status deployment/coredns --namespace kube-system`。

kubectl rollout 具有以下命令：

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

如果推出时间过长，Amazon EKS 将撤销推出，并且会将类型为附加组件更新且状态为失败的消息添加到附加组件的更新历史记录中。要调查任何问题，请先查看推出历史记录，然后在 CoreDNS 容器组（pod）上运行 kubectl logs，进而查看 CoreDNS 的日志。

5. （可选）可以提供自动扩缩所能扩展的 CoreDNS 容器组（pod）数量的最小值和最大值。

   以下示例显示了已启用的自动扩缩以及所有带值的可选键。我们建议 CoreDNS 容器组（pod）的最小数量始终大于 2，以便为集群中的 DNS 服务提供弹性。

   ```
   aws eks update-addon --cluster-name my-cluster --addon-name coredns \
       --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
   {"enabled":true,"minReplicas":2,"maxReplicas":10}}'
   ```

6. 运行以下命令，检查附加组件的更新状态：

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns
   ```

   如果您看到此行："status": "ACTIVE"，则表示推出已完成，并且附加组件正在所有 CoreDNS 容器组（pod）中使用新配置。更改集群中的节点数量和 CPU 核心数量时，Amazon EKS 会扩展 CoreDNS 部署的副本数量。

使用 CoreDNS 指标监控 Kubernetes DNS 解析

CoreDNS 作为 EKS 附加组件，会在 kube-dns 服务中以 Prometheus 格式公开端口 9153 上 CoreDNS 的指标。您可以使用 Prometheus、Amazon CloudWatch 代理，或任何其他兼容系统来抓取（收集）这些指标。

有关同时兼容 Prometheus 和 CloudWatch 代理的抓取配置示例，请参阅《Amazon CloudWatch 用户指南》中的适用于 Prometheus 的 CloudWatch 代理配置。

在 Amazon EKS 集群中管理 kube-proxy

> **Tip**
>
> 使用 Amazon EKS 自动模式时，您无需安装或升级联网附加组件。自动模式包含容器组联网和负载均衡功能。
> 有关更多信息，请参阅 EKS 自动模式。

建议您向集群添加 Amazon EKS 类型的附加组件，而不是自行管理类型的附加组件。如果不熟悉这些类型之间的区别，请参阅 the section called “Amazon EKS 附加组件”。有关向集群中添加 Amazon EKS 附加组件的更多信息，请参阅 the section called “创建附加组件”。如果您无法使用 Amazon EKS 附加组件，我们鼓励您向容器路线图 GitHub 存储库提交有关您为什么无法使用的问题。

kube-proxy 附加组件部署在 Amazon EKS 集群中的每个 Amazon EC2 节点上。该附加组件会在节点上维护网络规则，并实现与容器组（pod）的网络通信。该附加组件不会部署到集群中的 Fargate 节点。有关更多信息，请参阅 Kubernetes 文档中的 kube-proxy。

作为 Amazon EKS 附加组件安装

kube-proxy 版本

下表列出了每个 Kubernetes 版本的 Amazon EKS 附加组件类型的最新版本。

Kubernetes 版本	kube-proxy 版本
1.34	v1.34.1-eksbuild.2
1.33	v1.33.5-eksbuild.2
1.32	v1.32.9-eksbuild.2
1.31	v1.31.13-eksbuild.2
1.30	v1.30.14-eksbuild.8
1.29	v1.29.15-eksbuild.16

Kubernetes 版本	**kube-proxy** 版本
1.28	v1.28.15-eksbuild.31

> **Note**
>
> 该文档的早期版本不正确。kube-proxy 版本 v1.28.5、v1.27.9 和 v1.26.12 不可用。如果您自行管理此附加组件，则表中的版本可能与可用的自行管理版本不同。

kube-proxy 容器映像

kube-proxy 容器映像基于 Amazon EKS Distro 维护的最小基本映像，其中包含最低要求的软件包且没有 Shell。有关更多信息，请参阅 Amazon EKS Distro。

下表列出了每个 Amazon EKS 集群版本可用的自行管理 kube-proxy 容器映像的最新版本。

版本	kube-proxy
1.34	v1.34.1-eksbuild.2
1.33	v1.33.5-minimal-eksbuild.2
1.32	v1.32.9-minimal-eksbuild.2
1.31	v1.31.13-minimal-eksbuild.2
1.30	v1.30.14-minimal-eksbuild.8
1.29	v1.29.15-minimal-eksbuild.16
1.28	v1.28.15-minimal-eksbuild.31

更新 Amazon EKS 附加组件类型时，可以指定有效的 Amazon EKS 附加组件版本，该版本可能不是此表中列出的版本。这是因为 Amazon EKS 附加组件版本并不总是与更新此附加组件的自行管理类型时指定的容器映像版本相匹配。更新此附加组件的自行管理类型时，可指定此表中列出的有效容器映像版本。

更新 Kubernetes **kube-proxy** 自行管理的附加组件

> **Important**
>
> 建议您向集群添加 Amazon EKS 类型的附加组件，而不是自行管理类型的附加组件。如果不熟悉这些类型之间的区别，请参阅 the section called “Amazon EKS 附加组件”。有关向集群中添加 Amazon EKS 附加组件的更多信息，请参阅 the section called “创建附加组件”。如果您无法使用 Amazon EKS 附加组件，我们鼓励您向容器路线图 GitHub 存储库提交有关您为什么无法使用的问题。

先决条件

- 现有 Amazon EKS 集群。要部署一个角色，请参阅开始使用。

注意事项

- Amazon EKS 集群上的 Kube-proxy 具有与 Kubernetes 相同的兼容性和偏斜策略。了解如何验证 Amazon EKS 附加组件版本与集群的兼容性。
  1. 确认已在集群上安装自行管理类型的附加组件。将 *my-cluster* 替换为您的集群的名称。

     ```
     aws eks describe-addon --cluster-name my-cluster --addon-name kube-proxy --query
      addon.addonVersion --output text
     ```

     如果返回错误消息，则表明集群上安装有自行管理类型的附加组件。本主题中的其余步骤用于更新自行管理类型的附加组件。如果返回版本号，则表明集群上安装有 Amazon EKS 类型的附加组件。要更新附加组件的 Amazon EKS 类型，请使用更新 Amazon EKS 附件组件中的程序，而不是此主题中的程序。如果不熟悉这些附加组件类型之间的区别，请参阅 the section called “Amazon EKS 附加组件”。
  2. 查看集群上当前安装的容器映像版本。

     ```
     kubectl describe daemonset kube-proxy -n kube-system | grep Image
     ```

     示例输出如下。

     ```
     Image:    602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.29.1-
     eksbuild.2
     ```

在示例输出中，集群上安装的版本是 *v1.29.1-eksbuild.2*。

3. 通过将 *602401143452* 和 *region-code* 替换为上一步中输出的值，更新 kube-proxy 附加组件。将 *v1.30.6-eksbuild.3* 替换为最新可用的自行管理 kube-proxy 容器映像版本中针对各 Amazon EKS 集群版本表列出的 kube-proxy 版本。

   > **Important**
   >
   > 每种图像类型的清单都不同，且在默认或最小图像类型之间不兼容。您必须使用与前一个图像相同的图像类型，以便入口点与参数匹配。

   ```
   kubectl set image daemonset.apps/kube-proxy -n kube-system kube-
   proxy=602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.30.6-
   eksbuild.3
   ```

   示例输出如下。

   ```
   daemonset.apps/kube-proxy image updated
   ```

4. 确认新版本现已安装在集群上。

   ```
   kubectl describe daemonset kube-proxy -n kube-system | grep Image | cut -d ":" -f 3
   ```

   示例输出如下。

   ```
   v1.30.0-eksbuild.3
   ```

5. 如果您在同一集群中使用 x86 和 Arm 节点，而且您的集群是在 2020 年 8 月 17 日之前部署的。那么请使用以下命令编辑您的 kube-proxy 清单以将多个硬件架构的节点选择器纳入其中。您只需执行此操作一次。将选择器添加到清单中之后，您无需在每次更新附加组件时都进行添加。如果您的集群是在 2020 年 8 月 17 日或之后部署的，则 kube-proxy 已经具有多架构功能。

   ```
   kubectl edit -n kube-system daemonset/kube-proxy
   ```

   将以下节点选择器添加到编辑器中的文件中，然后保存文件。有关在编辑器的何处纳入此文本的示例，请参阅 GitHub 上的 CNI 清单文件。这让 Kubernetes 能够根据节点的硬件架构提取正确的硬件映像。

```
- key: "kubernetes.io/arch"
  operator: In
  values:
  - amd64
  - arm64
```

6. 如果集群最初是使用 Kubernetes 1.14 版本或更高版本创建的，则可以跳过此步骤，因为 kube-proxy 已经包含了此 Affinity Rule。如果 Amazon EKS 集群最初是使用 Kubernetes 1.13 版本或更低版本创建的，且打算在集群中使用 Fargate 节点，则编辑 kube-proxy 清单纳入 NodeAffinity 规则，防止从 Fargate 节点上调度 kube-proxy 容器组（pod）。您只需执行此编辑一次。将 Affinity Rule 添加到清单中之后，您无需在每次更新附加组件时都进行添加。编辑 kube-proxy Daemonset。

   ```
   kubectl edit -n kube-system daemonset/kube-proxy
   ```

   将以下 Affinity Rule 添加到编辑器中文件中的 DaemonSet spec 部分，然后保存文件。有关在编辑器的何处纳入此文本的示例，请参阅 GitHub 上的 CNI 清单文件。

   ```
   - key: eks.amazonaws.com/compute-type
     operator: NotIn
     values:
     - fargate
   ```

了解如何将工作负载和附加组件部署到 Amazon EKS

工作负载部署在容器中，而容器部署在 Kubernetes 的容器组（pod）中。一个容器组（pod）包含一个或多个容器。通常会有一个或多个提供相同服务的容器组（pod）部署在 Kubernetes 服务中。在部署多个提供相同服务的容器组（pod）以后，您可以：

- 使用 Amazon Web Services 管理控制台 查看在您的每个集群上运行的工作负载的相关信息。
- 使用 Kubernetes Vertical Pod Autoscaler 垂直扩缩容器组（pod）。
- 使用 Kubernetes Horizontal Pod Autoscaler 水平扩缩满足需求所需的容器组（pod）数量。
- 创建外部 [面向可通过互联网访问的容器组（pod）] 或内部 [面向私有容器组（pod）] 网络负载均衡器，平衡容器组（pod）之间的网络流量。负载均衡器在 OSI 模型的第 4 层路由流量。
- 创建应用程序负载均衡器，平衡容器组（pod）之间的应用程序流量。Application Load Balancer 在 OSI 模型的第 7 层路由流量。
- 如果不熟悉 Kubernetes，本主题可帮助您部署示例应用程序。
- 您可以使用 `externalIPs` 限制可分配给服务的 IP 地址。

在 Linux 上部署示例应用程序

在本主题中，您在 Linux 节点上的集群中部署一个示例应用程序。

先决条件

- 现有的 Kubernetes 集群至少有一个节点。如果没有现有 Amazon EKS 集群，可以根据 开始使用 中的指南之一部署一个集群。
- 安装在计算机上的 `Kubectl`。有关更多信息，请参阅 the section called “设置 kubectl 和 eksctl”。
- `Kubectl` 配置为与集群通信。有关更多信息，请参阅 the section called “使用 kubectl 访问集群”。
- 如果计划将示例工作负载部署到 Fargate，现有 Fargate 配置文件必须包含本教程创建的相同命名空间，即 `eks-sample-app`，除非更改名称。如果使用开始使用中的指南之一创建集群，则必须创建新配置文件或将命名空间添加到现有配置文件，因为在入门指南中创建的配置文件没有指定本教程使用的命名空间。您的 VPC 还必须具有至少一个私有子网。

尽管下列步骤中有许多变量都可以更改，但我们建议仅在指定的情况下更改变量值。更深入地了解 Kubernetes 容器组（pod）、部署和服务后，即可尝试更改其他值。

创建命名空间

命名空间允许您在 Kubernetes 中对资源进行分组。有关更多信息，请参阅 Kubernetes 文档中的命名空间。如果您计划部署示例应用程序以使用 Amazon Fargate 简化计算管理，请确保在定义哪些容器组（pod）在启动时使用 namespace Fargate 中 Amazon 的值为 `eks-sample-app`。

```
kubectl create namespace eks-sample-app
```

创建 Kubernetes 部署

创建 Kubernetes 部署。此示例部署从公共存储库中提取容器映像，并将各个容器组（pod）的三个副本部署到您的集群中。有关更多信息，请参阅 Kubernetes 文档中的部署。

1. 将以下内容保存到名为 `eks-sample-deployment.yaml` 的文件中。示例应用程序中的容器不使用网络存储，但可能有需要的应用程序。有关更多信息，请参阅 应用程序数据存储。
   - `kubernetes.io/arch` 键下的 `amd64` 或 `arm64` `values` 意味着应用程序可以部署到任一硬件架构（如果集群中两者都有）。可能出现这种情况，因为此映像是一个多架构映像，但并非全部都是。可以通过查看从中提取的存储库的映像详细信息，确定支持映像的硬件结构。部署不支持硬件架构类型的映像或不希望将映像部署到的映像时，请从清单中删除该类型。有关更多信息，请参阅 Kubernetes 文档中的众所周知的标签、注释和污点。
   - `kubernetes.io/os: linux` `nodeSelector` 意味着集群中有 Linux 和 Windows 节点（例如），则该映像将只部署到 Linux 节点。有关更多信息，请参阅 Kubernetes 文档中的众所周知的标签、注释和污点。

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: eks-sample-linux-deployment
     namespace: eks-sample-app
     labels:
       app: eks-sample-linux-app
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: eks-sample-linux-app
     template:
       metadata:
         labels:
           app: eks-sample-linux-app
   ```

```
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
                - arm64
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
      nodeSelector:
        kubernetes.io/os: linux
```

2. 将部署清单应用于集群。

```
kubectl apply -f eks-sample-deployment.yaml
```

创建服务

服务允许您通过单个 IP 地址或名称访问所有副本。有关更多信息，请参阅 Kubernetes 文档中的服务。虽然没有在示例应用程序中实施，但如果应用程序需要与其他 Amazon 服务交互，我们建议为容器组（pod）创建 Kubernetes 服务帐户，然后将其关联到 Amazon IAM 账户。指定服务账户可使您的 Pod （一组容器）拥有为其指定的与其他服务交互的最低权限。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。

1. 将以下内容保存到名为 `eks-sample-service.yaml` 的文件中。Kubernetes 为服务分配自己的 IP 地址，该 IP 地址只能从集群内访问。要从集群外部访问服务，请部署 Amazon Load Balancer Controller 以负载均衡服务的应用程序或网络流量。

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-linux-service
```

```
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  selector:
    app: eks-sample-linux-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. 将服务清单应用于集群。

```
kubectl apply -f eks-sample-service.yaml
```

审核已创建的资源

1. 查看 eks-sample-app 命名空间中存在的所有资源。

```
kubectl get all -n eks-sample-app
```

示例输出如下。

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/eks-sample-linux-deployment-65b7669776-m6qxz   1/1     Running   0          27m
pod/eks-sample-linux-deployment-65b7669776-mmxvd   1/1     Running   0          27m
pod/eks-sample-linux-deployment-65b7669776-qzn22   1/1     Running   0          27m

NAME                                 TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
service/eks-sample-linux-service   ClusterIP    10.100.74.8     <none>        80/TCP
   32m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-linux-deployment 3/3     3            3           27m

NAME                                                      DESIRED   CURRENT   READY
 AGE
replicaset.apps/eks-sample-linux-deployment-776d8f8fd8    3         3         3
 27m
```

在输出中，可以看到之前步骤部署的示例清单中指定的服务和部署。您还可以看到三组容器。这是因为在示例清单 指定了 3 `replicas`。有关更多信息，请参阅 Kubernetes 文档中的 Pod。即使没有在示例清单中指定，Kubernetes 也会自动创建 `replicaset` 资源。有关 `ReplicaSets` 的更多信息，请参阅 Kubernetes 文档中的 ReplicaSet。

> **Note**
>
> Kubernetes 保持清单中指定的副本数目。如果这是生产部署，并且您希望 Kubernetes 为容器组（pod）横向扩展副本数量或纵向扩展计算资源，则按照使用 Horizontal Pod Autoscaler 调整容器组（pod）部署和使用 Vertical Pod Autoscaler 调整容器组（pod）资源进行此操作。

2. 查看已部署服务的详细信息。

```
kubectl -n eks-sample-app describe service eks-sample-linux-service
```

示例输出如下。

```
Name:              eks-sample-linux-service
Namespace:         eks-sample-app
Labels:            app=eks-sample-linux-app
Annotations:       <none>
Selector:          app=eks-sample-linux-app
Type:              ClusterIP
IP Families:       <none>
IP:                10.100.74.8
IPs:               10.100.74.8
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:  None
Events:            <none>
```

在前面的输出中，`IP:` 值是一个唯一 IP 地址，可以从集群内的任何节点或容器组（pod）访问，但无法从集群外部访问。`Endpoints` 值是从 VPC 内分配给属于服务一部分的容器组（pod）的 IP 地址。

3. 在上一步查看命名空间时，查看输出列出的一个 Pod（一组容器）的详细信息。将 *`776d8f8fd8-78w66`* 替换为针对其中一个容器组（pod）返回的值。

```
kubectl -n eks-sample-app describe pod eks-sample-linux-deployment-65b7669776-m6qxz
```

缩减的输出示例

```
Name:         eks-sample-linux-deployment-65b7669776-m6qxz
Namespace:    eks-sample-app
Priority:     0
Node:         ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:           192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By:  ReplicaSet/eks-sample-linux-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-linux-deployment-65b7669776-m6qxz to
 ip-192-168-45-132.us-west-2.compute.internal
[...]
```

在上一个输出中，IP: 的值是唯一 IP，该 IP 从分配给节点所在子网的 CIDR 块中分配给容器组（pod）。如果您希望从其他 CIDR 块中为 Pod 分配 IP 地址，则可以更改默认行为。有关更多信息，请参阅 the section called “自定义联网”。您还可以看到 Kubernetes 调度器在 IP 地址为 *192.168.45.132* 的 Node 上调度容器组（pod）。

> **Tip**
>
> 无需使用命令行，即可在 Amazon Web Services 管理控制台中查看有关容器组（pod）、服务、部署和其他 Kubernetes 资源的许多详细信息。有关更多信息，请参阅 the section called “访问集群资源”。

在容器组（pod）上运行 Shell

1. 在上一步中描述的 Pod（一组容器）上运行 Shell，将 *65b7669776-m6qxz* 替换为其中一个 Pod 的 ID。

```
kubectl exec -it eks-sample-linux-deployment-65b7669776-m6qxz -n eks-sample-app -- /
bin/bash
```

2. 在容器组（pod）Shell 中，查看上一步中随部署一起安装的 Web 服务器的输出。您只需指定服务名称。默认情况下，CoreDNS 将其解析为服务的 IP 地址，该地址与 Amazon EKS 集群一起部署。

```
curl eks-sample-linux-service
```

示例输出如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

3. 在 Pod（一组容器）Shell 中，查看 Pod（一组容器）的 DNS 服务器。

```
cat /etc/resolv.conf
```

示例输出如下。

```
nameserver 10.100.0.10
search eks-sample-app.svc.cluster.local svc.cluster.local cluster.local us-
west-2.compute.internal
options ndots:5
```

在之前的输出中，`10.100.0.10` 自动分配为部署到集群的所有 Pod（一组容器）的 `nameserver`。

4. 键入 `exit`，与 Pod（一组容器）断开连接。
5. 使用完示例应用程序后，您可以使用以下命令删除示例命名空间、服务和部署。

```
kubectl delete namespace eks-sample-app
```

后续步骤

部署示例应用程序后，您可能想尝试以下其中一些练习：

- 使用应用程序负载均衡器路由应用程序和 HTTP 流量
- 使用网络负载均衡器路由 TCP 和 UDP 流量

在 Windows 上部署示例应用程序

在本主题中，您在 Windows 节点上的集群中部署一个示例应用程序。

先决条件

- 现有的 Kubernetes 集群至少有一个节点。如果没有现有 Amazon EKS 集群，可以根据 开始使用 中的指南之一部署一个集群。必须为集群和至少一个 Amazon EC2 Windows 节点启用 Windows 支持。
- 安装在计算机上的 `Kubectl`。有关更多信息，请参阅 the section called “设置 kubectl 和 eksctl”。
- `Kubectl` 配置为与集群通信。有关更多信息，请参阅 the section called “使用 kubectl 访问集群”。
- 如果计划将示例工作负载部署到 Fargate，现有 Fargate 配置文件必须包含本教程创建的相同命名空间，即 `eks-sample-app`，除非更改名称。如果使用开始使用中的指南之一创建集群，则必须创建新配置文件或将命名空间添加到现有配置文件，因为在入门指南中创建的配置文件没有指定本教程使用的命名空间。您的 VPC 还必须具有至少一个私有子网。

尽管下列步骤中有许多变量都可以更改，但我们建议仅在指定的情况下更改变量值。更深入地了解 Kubernetes 容器组（pod）、部署和服务后，即可尝试更改其他值。

创建命名空间

命名空间允许您在 Kubernetes 中对资源进行分组。有关更多信息，请参阅 Kubernetes 文档中的命名空间。如果您计划部署示例应用程序以使用 Amazon Fargate 简化计算管理，请确保在定义哪些容器组（pod）在启动时使用 namespace Fargate 中 Amazon 的值为 `eks-sample-app`。

```
kubectl create namespace eks-sample-app
```

创建 Kubernetes 部署

此示例部署从公共存储库中提取容器映像，并将各个容器组（pod）的三个副本部署到您的集群中。有关更多信息，请参阅 Kubernetes 文档中的部署。

1. 将以下内容保存到名为 `eks-sample-deployment.yaml` 的文件中。示例应用程序中的容器不使用网络存储，但可能有需要的应用程序。有关更多信息，请参阅 应用程序数据存储。
   - `kubernetes.io/os: windows` `nodeSelector` 意味着如果集群中有 Windows 和 Linux 节点（例如），则该映像将只部署到 Windows 节点。有关更多信息，请参阅 Kubernetes 文档中的众所周知的标签、注释和污点。

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: eks-sample-windows-deployment
     namespace: eks-sample-app
     labels:
       app: eks-sample-windows-app
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: eks-sample-windows-app
     template:
       metadata:
         labels:
           app: eks-sample-windows-app
       spec:
         affinity:
           nodeAffinity:
             requiredDuringSchedulingIgnoredDuringExecution:
               nodeSelectorTerms:
               - matchExpressions:
   ```

```
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
      containers:
      - name: windows-server-iis
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        command:
        - powershell.exe
        - -command
        - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\
\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
      nodeSelector:
        kubernetes.io/os: windows
```

2. 将部署清单应用于集群。

```
kubectl apply -f eks-sample-deployment.yaml
```

创建服务

服务允许您通过单个 IP 地址或名称访问所有副本。有关更多信息，请参阅 Kubernetes 文档中的服务。虽然没有在示例应用程序中实施，但如果应用程序需要与其他 Amazon 服务交互，我们建议为容器组（pod）创建 Kubernetes 服务帐户，然后将其关联到 Amazon IAM 账户。指定服务账户可使您的 Pod （一组容器）拥有为其指定的与其他服务交互的最低权限。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。

1. 将以下内容保存到名为 `eks-sample-service.yaml` 的文件中。Kubernetes 为服务分配自己的 IP 地址，该 IP 地址只能从集群内访问。要从集群外部访问服务，请部署 Amazon Load Balancer Controller 以负载均衡服务的应用程序或网络流量。

```
apiVersion: v1
kind: Service
metadata:
```

```
  name: eks-sample-windows-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-windows-app
spec:
  selector:
    app: eks-sample-windows-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. 将服务清单应用于集群。

```
kubectl apply -f eks-sample-service.yaml
```

审核已创建的资源

1. 查看 eks-sample-app 命名空间中存在的所有资源。

```
kubectl get all -n eks-sample-app
```

示例输出如下。

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/eks-sample-windows-deployment-65b7669776-m6qxz   1/1     Running   0          27m
pod/eks-sample-windows-deployment-65b7669776-mmxvd   1/1     Running   0          27m
pod/eks-sample-windows-deployment-65b7669776-qzn22   1/1     Running   0          27m

NAME                                 TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
service/eks-sample-windows-service   ClusterIP    10.100.74.8     <none>        80/
TCP    32m

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-windows-deployment 3/3     3            3           27m

NAME                                                       DESIRED   CURRENT   READY
 AGE
```

```
replicaset.apps/eks-sample-windows-deployment-776d8f8fd8    3         3         3
   27m
```

在输出中，可以看到之前步骤部署的示例清单中指定的服务和部署。您还可以看到三组容器。这是因为在示例清单 指定了 3 `replicas`。有关更多信息，请参阅 Kubernetes 文档中的 Pod。即使没有在示例清单中指定，Kubernetes 也会自动创建 `replicaset` 资源。有关 `ReplicaSets` 的更多信息，请参阅 Kubernetes 文档中的 ReplicaSet。

> **Note**
>
> Kubernetes 保持清单中指定的副本数目。如果这是生产部署，并且您希望 Kubernetes 为容器组（pod）横向扩展副本数量或纵向扩展计算资源，则按照使用 Horizontal Pod Autoscaler 调整容器组（pod）部署和使用 Vertical Pod Autoscaler 调整容器组（pod）资源进行此操作。

2. 查看已部署服务的详细信息。

```
kubectl -n eks-sample-app describe service eks-sample-windows-service
```

示例输出如下。

```
Name:              eks-sample-windows-service
Namespace:         eks-sample-app
Labels:            app=eks-sample-windows-app
Annotations:       <none>
Selector:          app=eks-sample-windows-app
Type:              ClusterIP
IP Families:       <none>
IP:                10.100.74.8
IPs:               10.100.74.8
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:  None
Events:            <none>
```

在前面的输出中，`IP:` 值是一个唯一 IP 地址，可以从集群内的任何节点或容器组（pod）访问，但无法从集群外部访问。`Endpoints` 值是从 VPC 内分配给属于服务一部分的容器组（pod）的 IP 地址。

3. 在上一步查看命名空间时，查看输出列出的一个 Pod（一组容器）的详细信息。将 *776d8f8fd8-78w66* 替换为针对其中一个容器组（pod）返回的值。

```
kubectl -n eks-sample-app describe pod eks-sample-windows-deployment-65b7669776-m6qxz
```

缩减的输出示例

```
Name:         eks-sample-windows-deployment-65b7669776-m6qxz
Namespace:    eks-sample-app
Priority:     0
Node:         ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:           192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By:  ReplicaSet/eks-sample-windows-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-windows-deployment-65b7669776-m6qxz
 to ip-192-168-45-132.us-west-2.compute.internal
[...]
```

在上一个输出中，IP: 的值是唯一 IP，该 IP 从分配给节点所在子网的 CIDR 块中分配给容器组（pod）。如果您希望从其他 CIDR 块中为 Pod 分配 IP 地址，则可以更改默认行为。有关更多信息，请参阅 the section called “自定义联网”。您还可以看到 Kubernetes 调度器在 IP 地址为 *192.168.45.132* 的 Node 上调度容器组（pod）。

> **Tip**
>
> 无需使用命令行，即可在 Amazon Web Services 管理控制台中查看有关容器组（pod）、服务、部署和其他 Kubernetes 资源的许多详细信息。有关更多信息，请参阅 the section called “访问集群资源”。

在容器组（pod）上运行 Shell

1. 在上一步中描述的 Pod（一组容器）上运行 Shell，将 *65b7669776-m6qxz* 替换为其中一个 Pod 的 ID。

```
kubectl exec -it eks-sample-windows-deployment-65b7669776-m6qxz -n eks-sample-app --
 powershell.exe
```

2. 在容器组（pod）Shell 中，查看上一步中随部署一起安装的 Web 服务器的输出。您只需指定服务名称。默认情况下，CoreDNS 将其解析为服务的 IP 地址，该地址与 Amazon EKS 集群一起部署。

```
Invoke-WebRequest -uri eks-sample-windows-service/default.html -UseBasicParsing
```

示例输出如下。

```
StatusCode        : 200
StatusDescription : OK
Content           : < h t m l > < b o d y > < b r / > < b r / > < m a r q u e e > < H
 1 > H e l l o
                      E K S ! ! ! < H 1 > < m a r q u e e > < / b o d y > < h t m l >
```

3. 在 Pod（一组容器）Shell 中，查看 Pod（一组容器）的 DNS 服务器。

```
Get-NetIPConfiguration
```

缩减的输出

```
InterfaceAlias       : vEthernet
[...]
IPv4Address          : 192.168.63.14
[...]
```

```
DNSServer            : 10.100.0.10
```

在之前的输出中，10.100.0.10 自动分配为部署到集群的所有 Pod（一组容器）的 DNS 服务器。

4. 键入 exit，与 Pod（一组容器）断开连接。
5. 使用完示例应用程序后，您可以使用以下命令删除示例命名空间、服务和部署。

```
kubectl delete namespace eks-sample-app
```

后续步骤

部署示例应用程序后，您可能想尝试以下其中一些练习：

- 使用应用程序负载均衡器路由应用程序和 HTTP 流量
- 使用网络负载均衡器路由 TCP 和 UDP 流量

使用 Vertical Pod Autoscaler 调整容器组（pod）资源

Kubernetes Vertical Pod Autoscaler 为容器组（pod）自动调整 CPU 和内存预留，帮助实现“合适规模”的应用程序。此调整可以帮助提高集群资源利用率，释放 CPU 和内存供其他容器组（pod）使用。本主题可帮助您将 Vertical Pod Autoscaler 部署到集群并验证它在正常工作。

- 您拥有现有 Amazon EKS 集群。如果没有，请参阅开始使用。
- 您已安装 Kubernetes Metrics Server。有关更多信息，请参阅 the section called “Metrics Server”。
- 您使用的是配置为与 Amazon EKS 集群通信的 kubectl 客户端。
- 您的设备已安装 OpenSSL 1.1.1 或更高版本。

部署 Vertical Pod Autoscaler

在此部分中，您将部署 Vertical Pod Autoscaler 到集群。

1. 打开终端窗口，导航到您要下载 Vertical Pod Autoscaler 源代码的目录。
2. 克隆 kubernetes/autoscaler GitHub 存储库。

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. 切换到 `vertical-pod-autoscaler` 目录。

```
cd autoscaler/vertical-pod-autoscaler/
```

4. （可选）如果您已经部署另一个版本的 Vertical Pod Autoscaler，请使用以下命令将其删除。

```
./hack/vpa-down.sh
```

5. 如果您的节点无权访问 `registry.k8s.io` 容器注册表，则您需要提取下面的映像并将它们推送到自己的私有存储库。有关如何提取镜像并将它们推送到您自己的私有存储库的更多信息，请参阅 the section called “将镜像复制到存储库”。

```
registry.k8s.io/autoscaling/vpa-admission-controller:0.10.0
registry.k8s.io/autoscaling/vpa-recommender:0.10.0
registry.k8s.io/autoscaling/vpa-updater:0.10.0
```

   如果要将映像推送到私有 Amazon ECR 存储库，请将清单中的 `registry.k8s.io` 替换为您的注册表。请将 *111122223333* 替换为您的账户 ID。将 *region-code* 替换为您的集群所在的 Amazon 区域。以下命令假定您将存储库命名为与清单中的存储库名称相同。如果已将存储库命名为其它名称，则也需要进行更改。

```
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/admission-controller-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/recommender-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/updater-deployment.yaml
```

6. 使用以下命令将 Vertical Pod Autoscaler 部署到您的集群。

```
./hack/vpa-up.sh
```

7. 验证已成功创建 Vertical Pod Autoscaler 容器组（pod）。

```
kubectl get pods -n kube-system
```

   示例输出如下。

```
NAME                                        READY   STATUS    RESTARTS   AGE
[...]
```

```
metrics-server-8459fc497-kfj8w              1/1     Running   0          83m
vpa-admission-controller-68c748777d-ppspd   1/1     Running   0          7s
vpa-recommender-6fc8c67d85-gljpl            1/1     Running   0          8s
vpa-updater-786b96955c-bgp9d                1/1     Running   0          8s
```

测试 Vertical Pod Autoscaler 安装

在此部分中，您部署示例应用程序以验证 Vertical Pod Autoscaler 在正常运行。

1. 使用以下命令部署 `hamster.yaml` Vertical Pod Autoscaler 示例。

```
kubectl apply -f examples/hamster.yaml
```

2. 从 `hamster` 示例应用程序获取容器组（pod）。

```
kubectl get pods -l app=hamster
```

示例输出如下。

```
hamster-c7d89d6db-rglf5   1/1     Running   0          48s
hamster-c7d89d6db-znvz5   1/1     Running   0          48s
```

3. 描述其中一个容器组（pod），查看其 `cpu` 和 `memory` 预留。请将 *`c7d89d6db-rglf5`* 替换为上一步输出中返回的 ID 之一。

```
kubectl describe pod hamster-c7d89d6db-rglf5
```

示例输出如下。

```
[...]
Containers:
  hamster:
    Container ID:  docker://
e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
    Image:         registry.k8s.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
```

```
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:35:16 -0700
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        100m
      memory:     50Mi
[...]
```

您可以看到原始容器组（pod）预留了 100 millicpu 的 CPU 和 50 MiB 内存。对于本示例应用程序，100 millicpu 小于容器组（pod）运行所需的数量，因此 CPU 受限。它预留的内存也远小于所需的数量。Vertical Pod Autoscaler `vpa-recommender` 部署会分析 hamster 容器组（pod），查看 CPU 和内存需求是否合适。如果需要调整，`vpa-updater` 使用更新后的值重新启动容器组（pod）。

4. 等待 `vpa-updater` 启动新 hamster 容器组（pod）。这大概需要一两分钟。您可以使用以下命令监控容器组（pod）。

> **Note**
>
> 如果不确定是否启动了新容器组（pod），请将容器组名称与您之前的列表比较。新容器组（pod）启动时，您会看到新容器组（pod）名称。

```
kubectl get --watch Pods -l app=hamster
```

5. 当新 hamster 容器组（pod）启动时，对其进行描述，查看更新后的 CPU 和内存预留。

```
kubectl describe pod hamster-c7d89d6db-jxgfv
```

示例输出如下。

```
[...]
Containers:
  hamster:
```

```
    Container ID:
 docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
    Image:         registry.k8s.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:37:08 -0700
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        587m
      memory:     262144k
[...]
```

在之前的输出中，您可以看到 cpu 预留提升到了 587 个 millicpu，这是原始值的五倍多。memory 提高到了 262144 KB，即大约 250 MB，也就是原始值的五倍。此容器组（pod）资源不足，Vertical Pod Autoscaler 使用更为合适的值纠正了估计值。

6. 描述 hamster-vpa 资源以查看新的建议。

```
kubectl describe vpa/hamster-vpa
```

示例输出如下。

```
Name:         hamster-vpa
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"autoscaling.k8s.io/
v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-
vpa","namespace":"d...
API Version:  autoscaling.k8s.io/v1beta2
Kind:         VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2019-09-27T18:22:51Z
```

```
  Generation:          23
  Resource Version:    14411
  Self Link:           /apis/autoscaling.k8s.io/v1beta2/namespaces/default/
verticalpodautoscalers/hamster-vpa
  UID:                 d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time:  2019-09-27T18:23:28Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  hamster
      Lower Bound:
        Cpu:     550m
        Memory:  262144k
      Target:
        Cpu:     587m
        Memory:  262144k
      Uncapped Target:
        Cpu:     587m
        Memory:  262144k
      Upper Bound:
        Cpu:     21147m
        Memory:  387863636
Events:          <none>
```

7. 在完成对示例应用程序的试验后，使用以下命令可将其删除。

```
kubectl delete -f examples/hamster.yaml
```

使用 Horizontal Pod Autoscaler 扩展容器组（pod）部署

Kubernetes Horizontal Pod Autoscaler 根据资源的 CPU 利用率，自动扩展部署、复制控制器或副本集中的容器组（pod）数量。这有助于您的应用程序进行扩展以满足增长的需求，或在不需要资源时

进行缩减，从而释放出节点用于其他应用程序。当您设置目标 CPU 利用率百分比时，Horizontal Pod Autoscaler 扩展或缩减应用程序来尝试满足该目标。

Horizontal Pod Autoscaler 是 Kubernetes 中的标准 API 资源，只需在 Amazon EKS 集群上安装一个指标源（如 Kubernetes Metrics Server）即可正常运行。您不需要在集群上部署或安装 Horizontal Pod Autoscaler 以开始扩展您的应用程序。有关更多信息，请参阅 Kubernetes 文档中的 Horizontal Pod Autoscaler。

使用本主题为您的 Amazon EKS 集群准备 Horizontal Pod Autoscaler 并验证它可用于示例应用程序。

> **Note**
>
> 本主题基于 Kubernetes 文档中的 Horizontal Pod Autoscaler 演练。

- 您拥有现有 Amazon EKS 集群。如果没有，请参阅开始使用。
- 您已安装 Kubernetes Metrics Server。有关更多信息，请参阅 the section called “Metrics Server”。
- 您使用的是配置为与 Amazon EKS 集群通信的 `kubectl` 客户端。

运行 Horizontal Pod Autoscaler 测试应用程序

在此部分中，您部署示例应用程序以验证 Horizontal Pod Autoscaler 在正常运行。

> **Note**
>
> 本示例基于 Kubernetes 文档中的 Horizontal Pod Autoscaler 演练。

1. 使用以下命令部署一个简单的 Apache Web 服务器应用程序。

   ```
   kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
   ```

   向此 Apache Web 服务器容器组（pod）提供 500 millicpu 的 CPU 限制，并在端口 80 上提供服务。
2. 为 `php-apache` 部署创建 Horizontal Pod Autoscaler 资源。

   ```
   kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
   ```

此命令创建的 Autoscaler 将部署的 CPU 利用率定为 50%，最少有一个容器组（pod），最多有 10 个容器组（pod）。当平均 CPU 负载低于 50% 时，Autoscaler 尝试减少部署中的容器组（pod）数量，最少要保留一个。当负载大于 50% 时，Autoscaler 尝试增加部署中的容器组（pod）数量，最多增加到十个。有关更多信息，请参阅 Kubernetes 文档中的 HorizontalPodAutoscaler 是如何工作的？。

3. 使用以下命令描述 Autoscaler 以查看其详细信息。

```
kubectl get hpa
```

示例输出如下。

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    0%/50%    1         10        1          51s
```

如您所见，当前 CPU 负载是 0%，因为服务器上尚没有负载。容器组（pod）数量已处于最低界限（1 个），因此无法横向缩减。

4. 通过运行容器为 Web 服务器创建负载。

```
kubectl run -i \
    --tty load-generator \
    --rm --image=busybox \
    --restart=Never \
    -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

5. 要监视部署的向外扩展情况，请在与执行上一步骤的终端不同的终端上定期运行以下命令。

```
kubectl get hpa php-apache
```

示例输出如下。

```
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    250%/50%   1         10        5          4m44s
```

可能需要一分钟以上的时间才能增加副本计数。只要实际 CPU 百分比高于目标百分比，副本计数就会增加（最大值为 10）。在此情况下，此百分比为 250%，因此 REPLICAS 数会继续增加。

> **Note**
>
> 可能需要在几分钟后，您才能看到副本计数达到其最大值。例如，如果只需 6 个副本即可让 CPU 负载小于或等于 50%，则负载将不会超过 6 个副本。

6. 停止负载。在正在生成负载的终端窗口中，按住 Ctrl+C 键停止负载。在观察扩缩的终端中再次运行以下命令，您会看到副本数缩减回 1。

```
kubectl get hpa
```

示例输出如下。

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    0%/50%    1         10        1          25m
```

> **Note**
>
> 默认缩减时间范围为 5 分钟，因此，在您看到副本计数再次达到 1 之前也需要一段时间，甚至在当前 CPU 百分比为 0% 时也是如此。时间范围可以修改。有关更多信息，请参阅 Kubernetes 文档中的 Horizontal Pod Autoscaler。

7. 完成示例应用程序的试验之后，删除 php-apache 资源。

```
kubectl delete deployment.apps/php-apache service/php-apache
 horizontalpodautoscaler.autoscaling/php-apache
```

使用网络负载均衡器路由 TCP 和 UDP 流量

> **Note**
>
> 新增：Amazon EKS 自动模式可自动执行负载均衡例行任务。有关更多信息，请参阅：
>
> - the section called “部署负载均衡器”
> - the section called “创建服务”

网络流量在 OSI 模型的 L4 上实现负载均衡。若要对 L7 上的应用程序流量进行负载均衡，请部署 Kubernetes `ingress`，它会预置 Amazon 应用程序负载均衡器。有关更多信息，请参阅 the section called “应用程序负载均衡”。要了解更多有关两种负载均衡类型之间差异的信息，请参阅 Amazon 网站上的 Elastic Load Balancing 功能。

创建类型为 `LoadBalancer` 的 Kubernetes `Service` 时，Amazon 云提供程序负载均衡器控制器默认创建 Amazon 经典负载均衡器 ，但也可以创建 Amazon Network Load Balancer。此控制器将来仅接收关键错误修复。有关如何使用 Amazon 云提供程序负载均衡器的更多信息，请参阅 Kubernetes 文档中的 Amazon 云提供程序负载均衡器控制器。本主题中未涵盖其使用。

我们建议您使用版本 `2.7.2` 或更高版本的 Amazon Load Balancer Controller，而不是Amazon 云提供程序负载均衡器控制器。Amazon 负载均衡器控制器创建 Amazon 网络负载均衡器，但不创建 Amazon 经典负载均衡器。本主题的其余部分介绍如何使用 Amazon 负载均衡器控制器。

Amazon 网络负载均衡器可以对部署到 Amazon EC2 IP 和实例目标、Amazon Fargate IP 目标或 Amazon EKS 混合节点功能即 IP 目标的容器组（pod）的网络流量进行负载均衡。有关更多信息，请参阅 GitHub 上的 Amazon Load Balancer Controller。

先决条件

必须满足以下要求，才能使用 Amazon 负载均衡器控制器对网络流量进行负载均衡。

- 拥有现有集群。如果您没有现有集群，请参阅开始使用。如果您需要更新现有集群的版本，请参阅 the section called “更新 Kubernetes 版本”。
- 在集群上部署 Amazon Load Balancer Controller。有关更多信息，请参阅 the section called “ Amazon 负载均衡器控制器”。建议升级到版本 `2.7.2` 或更高版本。
- 至少有一个子网。如果在一个可用区中发现多个标记子网，则该控制器会按子网 ID 的字典顺序选择第一个子网。子网必须具有至少 8 个可用 IP 地址。
- 如果使用 `2.1.1` 或更低版本的 Amazon 负载均衡器控制器，则必须按如下方式标记子网。如果使用版本 `2.1.2` 或更高版本，则此标签是可选的。如果您在同一 VPC 中运行多个集群，或者 VPC 中有多个 Amazon 服务共享子网，并且希望对每个集群中预置负载均衡器的位置时效性更多控制，则可能需要标记子网。如果您明确指定子网 ID 作为服务对象上的注释，则 Kubernetes 和 Amazon 负载均衡器控制器会直接使用这些子网来创建负载均衡器。如果您选择使用此方法来预置负载均衡器，则不需要子网标记，您可以跳过以下私有和公有子网标记要求。将 *`my-cluster`* 替换为您的集群名称。
  - 密钥 – `kubernetes.io/cluster/<my-cluster>`
  - 值 – `shared` 或 `owned`

- 除非您明确指定子网 ID 作为服务或入口对象上的注释，否则您的公有子网和私有子网必须满足以下要求。如果通过明确指定子网 ID 作为服务或入口对象上的注释来预置负载均衡器，则 Kubernetes 和 Amazon 负载均衡器控制器会直接使用这些子网来创建负载均衡器，并且不需要以下标记。
  - 私有子网：必须采用以下格式标记。这样 Kubernetes 和 Amazon 负载均衡器控制器就会知道子网可用于内部负载均衡器。如果您在 2020 年 3 月 26 日之后使用 `eksctl` 或 Amazon EKS Amazon Amazon CloudFormation 模板创建 VPC，则在创建子网时会对子网进行适当标记。有关 Amazon EKS Amazon Amazon CloudFormation VPC 模板的更多信息，请参阅the section called “创建 VPC”。
    - 密钥 – `kubernetes.io/role/internal-elb`
    - 值 – `1`
  - 公有子网：必须采用以下格式标记。这样， Kubernetes 就会知道仅将这些子网用于外部负载均衡器，而不是在每个可用区中选择公有子网（按子网 ID 的字典表顺序选择）。如果您在 2020 年 3 月 26 日之后使用 `eksctl` 或 Amazon EKS Amazon CloudFormation 模板创建 VPC，则在创建子网时会对子网进行适当标记。有关 Amazon EKS Amazon CloudFormation VPC 模板的更多信息，请参阅the section called “创建 VPC”。
    - 密钥 – `kubernetes.io/role/elb`
    - 值 – `1`

  如果未显式添加子网角色标记，则 Kubernetes 服务控制器将检查集群 VPC 子网的路由表，确定子网是私有子网还是公有子网。我们建议您不要依赖此行为，而是明确地添加私有或公有角色标记。Amazon 负载均衡器控制器不会检查路由表，并且需要存在私有和公有标记才能成功地自动发现。

注意事项

- 负载均衡器的配置由添加到服务清单中的注释控制。使用 Amazon 负载均衡器控制器和使用 Amazon 云提供程序负载均衡器控制器的服务注释不同。部署服务前，请务必检查 Amazon 负载均衡器控制器的注释。
- 使用适用于 Kubernetes 的 Amazon VPC CNI 插件时，Amazon 负载均衡器控制器可以对 Amazon EC2 IP 或实例目标和 Fargate IP 目标进行负载均衡。使用备选的兼容 CNI 插件时，控制器只能对实例目标进行负载均衡，除非您要对 Amazon EKS 混合节点功能执行负载均衡。对于混合节点，控制器可以对 IP 目标进行负载均衡。有关 Network Load Balancer 目标类型的更多信息，请参阅 Network Load Balancer 用户指南中的目标类型
- 如果要在创建负载均衡器时或之后将标记添加到负载均衡器，请在服务规范中添加以下注释。有关更多信息，请参阅 Amazon Load Balancer Controller 文档中的 Amazon 资源标签。

```
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags
```

- 您可以通过添加以下注释，从而将弹性 IP 地址分配到 Network Load Balancer。将示例值替换为弹性 IP 地址的 `Allocation IDs`。`Allocation IDs` 的数量必须与用于负载均衡器的子网数量相匹配。有关更多信息，请参阅 Amazon Load Balancer Controller 文档。

```
service.beta.kubernetes.io/aws-load-balancer-eip-allocations: eipalloc-
xxxxxxxxxxxxxxxx,eipalloc-yyyyyyyyyyyyyyyy
```

- 对于您创建的每个网络负载均衡器，Amazon EKS 会向节点的安全组添加一个用于客户端流量的入站规则，并为 VPC 中的每个负载均衡器子网添加一个用于运行状况检查的规则。如果 Amazon EKS 尝试创建的规则超过安全组允许的最大规则数配额，则部署 `LoadBalancer` 类型的服务可能会失败。有关更多信息，请参阅 Amazon VPC 用户指南的 Amazon VPC 配额中的安全组。请考虑以下选项，以最大限度地减少超过安全组最大规则数的可能性：
  - 请求增加每个安全组配额的规则数。有关更多信息，请参阅 Service Quotas 用户指南中的请求增加配额。
  - 使用 IP 目标，而不是实例目标。对于 IP 目标，可以为相同目标端口共享规则。可以使用注释手动指定负载均衡器子网。有关更多信息，请参阅 GitHub 上的注释。
  - 使用入口（而不是 `LoadBalancer` 类型的服务）将流量发送到您的服务。Amazon Application Load Balancer 需要的规则数比 Network Load Balancer 少。您可以跨多个入口共享 ALB。有关更多信息，请参阅 the section called “应用程序负载均衡”。您无法跨多个服务共享网络负载均衡器。
  - 将您的集群部署到多个账户。
- 如果您的容器组（pod）在 Windows 上运行，则在 Amazon EKS 集群中，采用负载均衡器的单个服务最多可支持 1024 个后端容器组（pod）。每个 Pod 都有自己的唯一 IP 地址。
- 我们建议您仅使用 Amazon 负载均衡器控制器创建新的 Network Load Balancer。尝试替换使用 Amazon 云提供程序负载均衡器控制器创建的现有 Network Load Balancer，将产生多个可能导致应用程序停机的 Network Load Balancer。

创建网络负载均衡器

您可以使用 IP 或实例目标创建网络负载均衡器。

创建网络负载均衡器 – IP 目标

- 您可以将 IP 目标与部署到 Amazon EC2 节点、Fargate 或 Amazon EKS 混合节点功能的容器组（pod）结合使用。您的 Kubernetes 服务必须创建为类型 `LoadBalancer`。有关更多信息，请参阅 Kubernetes 文档中的类型 LoadBalancer。

  要创建使用 IP 目标的负载均衡器，请将以下注释添加到服务清单中并部署您的服务。`aws-load-balancer-type` 的 `external` 值是导致 Amazon 负载均衡器控制器而不是 Amazon 云提供程序负载均衡器控制器创建 Network Load Balancer 的原因。您可以查看带有注释的示例服务清单。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

> **Note**
>
> 如果要对 `IPv6` 容器组（pod）进行负载均衡，请添加以下注释。您只能通过 `IPv6` 对 IP 目标进行负载均衡，无法对实例目标进行负载均衡。如果没有此注释，则通过 `IPv4` 进行负载均衡。

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: dualstack
```

默认情况下，Network Load Balancer 使用 `internal` `aws-load-balancer-scheme` 创建。您可以在集群 VPC 的任意子网中启动网络负载均衡器，包括在创建集群时未指定的子网。

Kubernetes 会检查子网的路由表来确定它们是公有还是私有。公有子网有使用互联网网关直接连接到 Internet 的路由，但私有子网没有。

如果要在公有子网中创建 Network Load Balancer 以将负载均衡到 Amazon EC2 节点（Fargate 只能是私有的），请指定具有以下注释的 `internet-facing`：

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

> **Note**
>
> 仍支持 `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` 注释的向后兼容性。但是，我们建议您将以前的注释用于新负载均衡器，而不是 `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"`。

> **Important**
>
> 创建服务后，请勿编辑注释。如果需要对其进行修改，请删除相应服务对象，然后使用此注释的所需值重新创建它。

创建网络负载均衡器 – 实例目标

- Amazon 云提供程序负载均衡器控制器仅使用实例目标创建 Network Load Balancer。`2.2.0` 版和更高版本的 Amazon 负载均衡器控制器也使用实例目标创建网络负载均衡器。我们建议使用该控制器而不是 Amazon 云提供程序负载均衡器控制器来创建新的 Network Load Balancer。您可以将网络负载均衡器实例目标与部署到 Amazon EC2 节点（而不是部署到 Fargate）的容器组（pod）结合使用。要在部署到 Fargate 的不同 Pod 之间对网络流量进行负载均衡，必须使用 IP 目标。

  要将 Network Load Balancer 部署到私有子网，您的服务规范须具有以下注释。您可以查看带有注释的示例服务清单。`aws-load-balancer-type` 的 `external` 值是导致 Amazon 负载均衡器控制器而不是 Amazon 云提供程序负载均衡器控制器创建 Network Load Balancer 的原因。

  ```
  service.beta.kubernetes.io/aws-load-balancer-type: "external"
  service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
  ```

  默认情况下，Network Load Balancer 使用 `internal` `aws-load-balancer-scheme` 创建。对于内部 Network Load Balancer，Amazon EKS 集群必须配置为使用 VPC 中的至少一个私有子网。Kubernetes 会检查子网的路由表来确定它们是公有还是私有。公有子网有使用互联网网关直接连接到 Internet 的路由，但私有子网没有。

  如果要在公有子网中创建 Network Load Balancer 以将负载均衡到 Amazon EC2 节点，请指定具有以下注释的 `internet-facing`：

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

> **Important**
>
> 创建服务后，请勿编辑注释。如果需要对其进行修改，请删除相应服务对象，然后使用此注释的所需值重新创建它。

（可选）部署示例应用程序

- 集群 VPC 中至少有一个公有或私有子网。
- 在集群上部署 Amazon Load Balancer Controller。有关更多信息，请参阅 the section called “Amazon 负载均衡器控制器”。建议升级到版本 2.7.2 或更高版本。

1. 如果部署到 Fargate，请确保您的 VPC 中有一个可用的私有子网，然后创建 Fargate 配置文件。如果您不部署到 Fargate，请跳过此步骤。您可以通过运行以下命令来创建配置文件，也可以在 Amazon Web Services 管理控制台 中，使用该命令中 name 和 namespace 的相同值创建配置文件。将 example values 替换为您自己的值。

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --region region-code \
    --name nlb-sample-app \
    --namespace nlb-sample-app
```

2. 部署示例应用程序。

   a. 为应用程序创建命名空间。

```
kubectl create namespace nlb-sample-app
```

   b. 将以下内容保存到计算机上名为 sample-deployment.yaml 文件的文件中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nlb-sample-app
  namespace: nlb-sample-app
spec:
  replicas: 3
```

```
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: tcp
              containerPort: 80
```

c. 将清单应用于集群。

```
kubectl apply -f sample-deployment.yaml
```

3. 创建具有面向互联网的网络负载均衡器的服务，以便将负载均衡到 IP 目标。

a. 将以下内容保存到计算机上名为 `sample-service.yaml` 文件的文件中。如果部署到 Fargate 节点，请删除 `service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing` 行。

```
apiVersion: v1
kind: Service
metadata:
  name: nlb-sample-service
  namespace: nlb-sample-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

b. 将清单应用于集群。

```
kubectl apply -f sample-service.yaml
```

4. 确认是否已部署相应服务。

```
kubectl get svc nlb-sample-service -n nlb-sample-app
```

示例输出如下。

```
NAME             TYPE           CLUSTER-IP         EXTERNAL-IP
                                                 PORT(S)        AGE
sample-service   LoadBalancer   10.100.240.137   k8s-nlbsampl-nlbsampl-xxxxxxxxxx-
xxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com  80:32400/TCP   16h
```

> **Note**
>
> *10.100.240.137* 和 *xxxxxxxxxx-xxxxxxxxxxxxxxxx* 的值与示例输出不同（它们将是您的负载均衡器独有的），并且 *us-west-2* 根据集群所在的 Amazon 区域而不同。

5. 打开 Amazon EC2 Amazon Web Services 管理控制台。在左侧导航窗格中，选择 Target Groups（目标组）（位于 Load Balancing（负载均衡）下）。在名称列中，选择目标组的名称，其中负载均衡器列的值与上一步输出的 EXTERNAL-IP 列中的一部分名称相匹配。例如，如果您的输出与之前的输出相同，则应选择名为 k8s-default-samplese-*xxxxxxxxxx* 的目标组。Target type（目标类型）为 IP，因为在示例服务清单中指定。
6. 选择 Target group（目标组），然后选择 Targets（目标）选项卡。在 Registered targets（已注册目标）中，您应该可看到上一步中部署的三个副本的三个 IP 地址。在所有目标的状态均为 healthy（正常）之前，请耐心等待，然后再继续。可能需要几分钟时间所有目标的状态才能达到 healthy。在更改为 healthy 状态之前，目标可能是 unhealthy 状态。
7. 将流量发送到服务，将 *xxxxxxxxxx-xxxxxxxxxxxxxxxx* 和 *us-west-2* 替换为在上一步输出中为 EXTERNAL-IP 返回的值。如果您已部署到私有子网，则需要从 VPC 中的设备（例如堡垒主机）查看页面。有关更多信息，请参阅 Amazon 上的 Linux 堡垒主机。

```
curl k8s-default-samplese-xxxxxxxxxx-xxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com
```

示例输出如下。

```
<!DOCTYPE html>
<html>
```

```
<head>
<title>Welcome to nginx!</title>
[...]
```

8. 完成示例部署、服务和命名空间后，将其移除。

```
kubectl delete namespace nlb-sample-app
```

使用应用程序负载均衡器路由应用程序和 HTTP 流量

> **Note**
>
> 新增：Amazon EKS 自动模式可自动执行负载均衡例行任务。有关更多信息，请参阅：
>
> - the section called “部署负载均衡器”
> - the section called “创建 IngressClass”

当您创建 Kubernetes `ingress` 时，会预置一个 Amazon Application Load Balancer ALB)，以实现应用程序流量负载均衡。要了解详情，请参阅 Application Load Balancer 用户指南中的“什么是 Application Load Balancer？”，以及 Kubernetes 文档中的入口。ALB 可以与部署到节点或 Amazon Fargate 的 Pod 一起使用。您可以将 ALB 部署到公有子网或私有子网。

应用程序流量在 OSI 模型的 L7 上实现均衡。要在 L4 上对网络流量进行负载均衡，您可以部署 `LoadBalancer` 类型的 Kubernetes `service`。这种类型将预置 Amazon Network Load Balancer。有关更多信息，请参阅 the section called “网络负载均衡”。要了解更多有关两种负载均衡类型之间差异的信息，请参阅 Amazon 网站上的 Elastic Load Balancing 功能。

先决条件

在对应用程序的应用程序流量进行负载均衡之前，您必须符合以下要求。

- 拥有现有集群。如果您没有现有集群，请参阅开始使用。如果您需要更新现有集群的版本，请参阅 the section called “更新 Kubernetes 版本”。
- 在集群上部署 Amazon Load Balancer Controller。有关更多信息，请参阅 the section called “Amazon 负载均衡器控制器”。建议升级到版本 `2.7.2` 或更高版本。

- 至少两个子网位于不同的可用区。Amazon 负载均衡器控制器会从每个可用区中选择一个子网。如果在一个可用区中发现多个标记子网，则该控制器会按子网 ID 的字典顺序选择第子网。每个子网必须具有至少 8 个可用 IP 地址。

  如果您使用的是挂载到 Worker 节点的多个安全组，则必须按如下方式标记一个安全组。将 *my-cluster* 替换为您的集群名称。

  - 密钥 – `kubernetes.io/cluster/<my-cluster>`
  - 值 – `shared` 或 `owned`
- 如果您使用的是 Amazon 负载均衡器控制器版本 `2.1.1` 或更早版本，则必须按如下格式标记子网。如果使用版本 `2.1.2` 或更高版本，则此标记是可选的。但是，如果出现以下任一情况，我们建议您标记子网。您有多个集群在同一 VPC 中运行，或者有多个 Amazon 服务在 VPC 中共享子网。或者，您希望更好地控制为每个集群配置的负载均衡器的位置。将 *my-cluster* 替换为您的集群名称。
  - 密钥 – `kubernetes.io/cluster/<my-cluster>`
  - 值 – `shared` 或 `owned`
- 您的公有子网和私有子网必须满足以下要求。除非您明确指定子网 ID 作为服务或入口对象的注释。假设您通过明确指定子网 ID 作为服务或入口对象的注释来预置负载均衡器。在这种情况下，Kubernetes 和 Amazon 负载均衡器控制器会直接使用这些子网创建负载均衡器，并且不需要以下标签。
  - 私有子网：必须采用以下格式标记。这样 Kubernetes 和 Amazon 负载均衡器控制器就会知道子网可用于内部负载均衡器。如果您在 2020 年 3 月 26 日之后使用 `eksctl` 或 Amazon EKS Amazon CloudFormation 模板创建 VPC，则在创建子网时会对子网进行适当标记。有关 Amazon EKS Amazon CloudFormation VPC 模板的更多信息，请参阅the section called “创建 VPC”。
    - 密钥 – `kubernetes.io/role/internal-elb`
    - 值 – `1`
  - 公有子网：必须采用以下格式标记。如此一来，Kubernetes 将知道仅使用为外部负载均衡器指定的子网。这样，Kubernetes 就不会在每个可用区中选择一个公有子网（根据其子网 ID 按字典顺序排列）。如果您在 2020 年 3 月 26 日之后使用 `eksctl` 或 Amazon EKS Amazon CloudFormation 模板创建 VPC，则在创建子网时会对子网进行适当标记。有关 Amazon EKS Amazon CloudFormation VPC 模板的更多信息，请参阅the section called “创建 VPC”。
    - 密钥 – `kubernetes.io/role/elb`
    - 值 – `1`

  如果未显式添加子网角色标签，则 Kubernetes 服务控制器将检查集群 VPC 子网的路由表。这是为了确定子网是私有还是公有。我们建议您不要依赖此行为。相反，明确添加私有或公有角色标

签。Amazon 负载均衡器控制器不会检查路由表。它还需要私有标签和公有标签才能成功实现自动发现。

- 每当使用 `kubernetes.io/ingress.class: alb` 注释在集群上创建 Kubernetes 入口资源时，Amazon 负载均衡器控制器就会创建 ALB 和必要的支持 Amazon 资源。入口资源会配置 ALB 以便将 HTTP 或 HTTPS 流量路由到集群中的不同容器组（pod）。要确保您的入口对象使用 Amazon 负载均衡器控制器，请将以下注释添加到您的 Kubernetes 入口规范。有关更多信息，请参阅 GitHub 上的入口规范。

```
annotations:
    kubernetes.io/ingress.class: alb
```

> **Note**
>
> 如果要对 IPv6 容器组（pod）进行负载均衡，请将以下注释添加到入口规范。您只能通过 IPv6 对 IP 目标进行负载均衡，无法对实例目标进行负载均衡。如果没有此注释，则通过 IPv4 进行负载均衡。

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

- Amazon负载均衡器控制器支持以下流量模式：
  - 实例 – 将您的集群中的节点注册为 ALB 的目标。传输到 ALB 的流量将路由到服务的 `NodePort`，然后转发到容器组（pod）。这是默认流量模式。您也可以使用 `alb.ingress.kubernetes.io/target-type: instance` 注释明确地指定该模式。

    > **Note**
    >
    > Kubernetes 服务必须指定 `NodePort` 或 `LoadBalancer` 类型，才能使用此流量模式。

  - IP：将 Pod 注册为 ALB 的目标。传输到 ALB 的流量将直接路由到服务的容器组（pod）。您必须指定 `alb.ingress.kubernetes.io/target-type: ip` 注释，才能使用此流量模式。当目标容器组（pod）在 Fargate 或 Amazon EKS 混合节点功能上运行时，必须使用 IP 目标类型。
- 要标记由控制器创建的 ALB，请向控制器添加以下注释：`alb.ingress.kubernetes.io/tags`。有关Amazon负载均衡器控制器支持的所有可用注释的列表，请参阅 GitHub 上的入口注释。
- 升级或降级 ALB 控制器版本可能会对依赖它的功能带来重大变化。有关在每个版本中引入的突发性更改的更多信息，请参阅 GitHub 上的 ALB 控制器发布说明。

在入口组中重复使用 ALB

您可以使用 IngressGroups 跨多个服务资源共享应用程序负载均衡器

要将入口加入到组，请将以下注释添加到 Kubernetes 入口资源规范中。

```
alb.ingress.kubernetes.io/group.name: my-group
```

组名称必须为：

- 长度不超过 63 个字符。
- 它们包含小写字母、数字、- 和 .。
- 以字母或数字开头和结尾。

控制器将自动合并同一入口组中所有入口的入口规则。它用单个 ALB 提供支持。在入口上定义的大多数注释仅适用于由该入口定义的路径。默认情况下，入口资源不属于任何入口组。

> **Warning**
>
> 潜在的安全风险
> 仅当所有具有创建或修改入口资源的 RBAC 权限的 Kubernetes 用户均在同一信任边界内时，为入口指定一个入口组。如果添加具有组名称的注释，则其他 Kubernetes 用户可创建或修改其入口，以使其属于同一入口组。这样做可能会导致不良行为，例如使用优先级更高的规则覆盖现有规则。

您可以添加入口资源的编号。

```
alb.ingress.kubernetes.io/group.order: '10'
```

该号码可以为 1-1000。首先计算同一入口组中所有入口的最小数目。没有此注释的所有入口将使用零值进行评估。数字越大的重复规则可以覆盖具有较小数字的规则。默认情况下，同一入口组中不同入口之间的规则顺序由入口的命名空间和名称的字典顺序决定。

> **Important**
>
> 确保同一入口组中的每个入口都具有唯一的优先级编号。不同入口中不能有重复的编号。

（可选）部署示例应用程序

- 集群 VPC 中至少有一个公有或私有子网。
- 在集群上部署 Amazon Load Balancer Controller。有关更多信息，请参阅 the section called “Amazon 负载均衡器控制器”。建议升级到版本 2.7.2 或更高版本。

您可以在具有 Amazon EC2 节点、Fargate Pod 或这两者的集群上运行示例应用程序。

1. 如果您不部署到 Fargate，请跳过此步骤。如果您要部署到 Fargate，请创建一个 Fargate 配置文件。您可以通过运行以下命令来创建配置文件，也可以在 Amazon Web Services 管理控制台 中，使用该命令中 name 和 namespace 的相同值创建配置文件。将 example values 替换为您自己的值。

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --region region-code \
    --name alb-sample-app \
    --namespace game-2048
```

2. 将游戏 2048 部署为示例应用程序，以确认作为入口对象的结果，Amazon 负载均衡器控制器是否会创建 Amazon ALB。完成您要部署到的子网类型的步骤。

   a. 如果要部署到使用 IPv6 系列创建的集群中的容器组（pod），请跳至下一步。

   - 公有：

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
   ```

   - 私有：

     A. 下载清单。

     ```
     curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
     ```

     B. 编辑文件并找到显示 alb.ingress.kubernetes.io/scheme: internet-facing 的行。

     C. 将 *internet-facing* 更改为 internal 并保存该文件。

     D. 将清单应用于集群。

```
kubectl apply -f 2048_full.yaml
```

b. 如果要部署到使用 IPv6 系列创建的集群中的容器组（pod），请完成以下步骤。

i. 下载清单。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
```

ii. 在编辑器中打开文件，并将以下行添加到入口规范的注释中。

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

iii. 如果要对内部容器组（pod）[而不是面向互联网的容器组（pod）] 进行负载均衡，请将显示 `alb.ingress.kubernetes.io/scheme: internet-facing` 的行更改为 `alb.ingress.kubernetes.io/scheme: internal`

iv. 保存该文件。

v. 将清单应用于集群。

```
kubectl apply -f 2048_full.yaml
```

3. 几分钟后，验证是否已使用以下命令创建入口资源。

```
kubectl get ingress/ingress-2048 -n game-2048
```

示例输出如下。

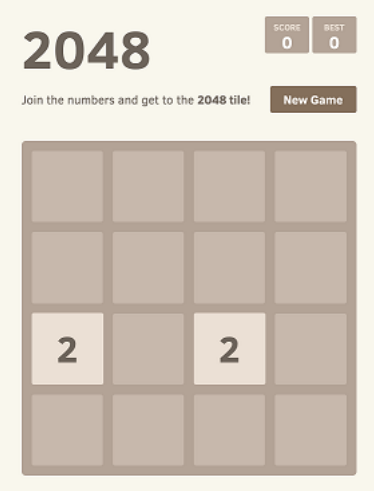
```
NAME           CLASS    HOSTS   ADDRESS
                      PORTS   AGE
ingress-2048   <none>   *       k8s-game2048-ingress2-xxxxxxxxxx-yyyyyyyyyy.region-
code.elb.amazonaws.com   80      2m32s
```

> **Note**
>
> 如果您在私有子网中创建了负载均衡器，则之前输出中 ADDRESS 下的值前面加上 `internal-`。

如果几分钟后仍未成功创建入口，请运行以下命令以查看 Amazon 负载均衡器控制器日志。这些日志包含可让您诊断部署中问题的错误消息。

```
kubectl logs -f -n kube-system -l app.kubernetes.io/instance=aws-load-balancer-
controller
```

1. 如果您部署到了公有子网，请打开浏览器并从上一命令输出导航到 ADDRESS URL 以查看示例应用程序。如果您没有看到任何内容，请刷新浏览器并重试。如果您已部署到私有子网，则需要从 VPC 中的设备（例如堡垒主机）查看页面。有关更多信息，请参阅 Amazon 上的 Linux 堡垒主机。



2. 在完成对示例应用程序的试验后，通过运行以下命令之一将其删除。
   - 如果您应用了清单，而不是应用下载的副本，请使用以下命令。

     ```
     kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-
     balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
     ```

   - 如果您下载并编辑了清单，请使用以下命令。

     ```
     kubectl delete -f 2048_full.yaml
     ```

限制可分配给服务的外部 IP 地址

可以通过以下方式从集群内部访问 Kubernetes 服务：

- Kubernetes 自动分配的集群 IP 地址

- 您在服务规格中为 `externalIPs` 属性指定的任何 IP 地址。外部 IP 地址不由 Kubernetes 管理，而是由集群管理员负责。使用 `externalIPs` 指定的外部 IP 地址不同于由云提供商分配给 `LoadBalancer` 类型服务的外部 IP 地址。

要了解有关 Kubernetes 服务的更多信息，请参阅 Kubernetes 文档中的服务。您能够限制可在服务规格中为 `externalIPs` 指定的 IP 地址。

1. 部署 `cert-manager` 来管理 Webhook 证书。有关更多信息，请参阅 cert-manager 文档。

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/
cert-manager.yaml
```

2. 验证 `cert-manager` 容器组（pod）是否正在运行。

```
kubectl get pods -n cert-manager
```

   示例输出如下。

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-58c8844bb8-nlx7q              1/1     Running   0          15s
cert-manager-cainjector-745768f6ff-696h5   1/1     Running   0          15s
cert-manager-webhook-67cc76975b-4v4nk      1/1     Running   0          14s
```

3. 检查您的现有服务，以确保为它们分配的外部 IP 地址都包含在您要将地址限制到的 CIDR 块之中。

```
kubectl get services -A
```

   示例输出如下。

```
NAMESPACE                   NAME                                    TYPE
 CLUSTER-IP       EXTERNAL-IP     PORT(S)         AGE
cert-manager                cert-manager                            ClusterIP
 10.100.102.137   <none>          9402/TCP        20m
cert-manager                cert-manager-webhook                    ClusterIP
 10.100.6.136     <none>          443/TCP         20m
default                     kubernetes                              ClusterIP
 10.100.0.1       <none>          443/TCP         2d1h
externalip-validation-system   externalip-validation-webhook-service   ClusterIP
 10.100.234.179   <none>          443/TCP         16s
```

```
kube-system                    kube-dns                                    ClusterIP
 10.100.0.10      <none>           53/UDP,53/TCP    2d1h
my-namespace                   my-service                                  ClusterIP
 10.100.128.10    192.168.1.1      80/TCP           149m
```

如果其中有任何一个 IP 地址值不在要将访问限制到的块之内，则需要将这些地址更改为块的范围内，然后重新部署服务。例如，上面输出中的 `my-service` 服务获得分配的一个外部 IP 地址不在步骤 5 中 CIDR 块示例范围内。

4. 下载外部 IP Webhook 清单。您还可以在 GitHub 上查看 Webhook 的源代码。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/externalip-webhook.yaml
```

5. 指定 CIDR 块。在编辑器中打开下载的文件并删除下面代码行开头的 `\#`。

```
#args:
#- --allowed-external-ip-cidrs=10.0.0.0/8
```

将 `10.0.0.0/8` 替换为您自己的 CIDR 块。您可以根据需要指定任意数量的块。如果指定多个块，请在块之间添加逗号。

6. 如果您的集群不在 Amazon `us-west-2` 区域中，则将文件中的 `us-west-2`、`602401143452` 和 `amazonaws.com` 替换为以下命令。在运行命令之前，请将 *`region-code`* 和 *`111122223333`* 替换为查看 Amazon EKS 附加组件的 Amazon 容器镜像注册表中的列表中您所在 Amazon 区域的值。

```
sed -i.bak -e 's|602401143452|111122223333|' externalip-webhook.yaml
sed -i.bak -e 's|us-west-2|region-code|' externalip-webhook.yaml
sed -i.bak -e 's|amazonaws.com||' externalip-webhook.yaml
```

7. 将清单应用于集群。

```
kubectl apply -f externalip-webhook.yaml
```

若尝试将服务部署到为 `externalIPs` 指定的 IP 地址不在指定 CIDR 块步骤中指定的地址块范围内的集群，则部署将会失败。

将容器镜像从一个存储库复制到另一个存储库

本主题介绍如何从节点无权访问的存储库中提取容器镜像，然后将该镜像推送到节点有权访问的存储库。您可以将镜像推送到 Amazon ECR 或节点有权访问的备用存储库。

- 已在计算机上安装并配置 Docker 引擎。有关说明，请参阅 Docker 文档中的安装 Docker 引擎。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 如果您希望节点通过 Amazon 的网络从私有 Amazon ECR 存储库中提取容器镜像或将容器镜像推送到私有 Amazon ECR 存储库，请为 Amazon ECR 创建接口 VPC 端点。有关更多信息，请参阅 Amazon Elastic Container Registry 用户指南中的为 Amazon ECR 创建 VPC 端点。

请完成以下步骤，以从存储库中提取容器镜像并将其推送到自己的存储库。在本主题提供的以下示例中，将提取 Kubernetes 指标帮助程序的 Amazon VPC CNI 插件的映像。按照这些步骤操作时，请确保将示例值替换为您自己的值。

1. 如果您还没有 Amazon ECR 存储库或其它存储库，请创建一个节点可以访问的存储库。以下命令将创建一个 Amazon ECR 私有存储库。Amazon ECR 私有存储库的名称必须以字母开头。它只能包含小写字母、数字、连字符 (-)、下划线 (_) 和正斜杠 (/)。有关更多信息，请参阅 Amazon Elastic Container Registry 用户指南中的创建私有存储库。

   您可以将 *`cni-metrics-helper`* 替换为所选的任何内容。作为最佳实践，请为每个镜像创建单独的存储库。我们建议您这样做，因为镜像标签在存储库中必须唯一。请将 *`region-code`* 替换为 Amazon ECR 支持的 Amazon 区域。

   ```
   aws ecr create-repository --region region-code --repository-name cni-metrics-helper
   ```

2. 请确定节点需要提取的镜像的注册表、存储库和标签（可选）。此信息采用 `registry/repository[:tag]` 格式。

许多关于安装镜像的 Amazon EKS 主题都要求您应用清单文件或使用 Helm 图表安装镜像。但是，在应用清单文件或安装 Helm 图表之前，请先查看清单或图表 `values.yaml` 文件的内容。这样您就可以确定要提取的注册表、存储库和标签。

例如，您可以在 Kubernetes 指标帮助程序的 Amazon VPC CNI 插件的清单文件中找到以下行。`602401143452.dkr.ecr.us-west-2.amazonaws.com` 注册表是 Amazon ECR 私有注册表。存储库是 `cni-metrics-helper`。

```
image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/cni-metrics-helper:v1.12.6"
```

您可能会看到镜像位置的以下变体：

- 仅限 `repository-name:tag`。在本例中，`docker.io` 通常是未指定的注册表，因为如果没有指定注册表，默认情况下 Kubernetes 会将它置于存储库名称前面。
- `repository-name/repository-namespace/repository:tag`。存储库命名空间为可选，但有时由存储库所有者指定以用于对镜像进行分类。例如，Amazon ECR 公开映像浏览馆中的所有 Amazon EC2 镜像都使用 `aws-ec2` 命名空间。

  在使用 Helm 安装镜像之前，请查看 Helm `values.yaml` 文件以确定镜像位置。例如，Kubernetes 指标帮助程序的 Amazon VPC CNI 插件的 values.yaml 文件包括以下行。

  ```
  image:
    region: us-west-2
    tag: v1.12.6
    account: "602401143452"
    domain: "amazonaws.com"
  ```

3. 请提取清单文件中指定的容器镜像。

   a. 如果从公有注册表中提取，例如 Amazon ECR 公开映像浏览馆，则可以跳到下一个子步骤，因为不需要身份验证。在此示例中，您可以对包含 CNI 指标帮助程序映像存储库的 Amazon ECR 私有注册表进行身份验证。Amazon EKS 在查看 Amazon EKS 附加组件的 Amazon 容器映像注册表中所列的每个注册表中维护映像。通过将 *602401143452* 和 *region-code* 替换为其它注册表的信息，您可以向任何注册表进行身份验证。支持 Amazon EKS 的每个 Amazon 区域都存在一个单独的注册表。

   ```
   aws ecr get-login-password --region region-code | docker login --username Amazon
    --password-stdin 602401143452.dkr.ecr.region-code.amazonaws.com
   ```

b. 提取镜像。在此示例中，您从上一个子步骤中进行身份验证的注册表中提取。将 *602401143452* 和 *region-code* 替换为您在上一个子步骤中提供的信息。

```
docker pull 602401143452.dkr.ecr.region-code.amazonaws.com/cni-metrics-
helper:v1.12.6
```

4. 请使用您的注册表、存储库和标签来标记提取的镜像。以下示例假设您从清单文件中提取镜像，然后将其推送到您在第一步中创建的 Amazon ECR 私有存储库。请将 *111122223333* 替换为您的账户 ID。将 *region-code* 替换为您在其中创建 Amazon ECR 私有存储库的 Amazon 区域。

```
docker tag cni-metrics-helper:v1.12.6 111122223333.dkr.ecr.region-code.amazonaws.com/
cni-metrics-helper:v1.12.6
```

5. 对注册表进行身份验证。在此示例中，您可以对在第一步中创建的 Amazon ECR 私有注册表进行身份验证。有关更多信息，请参阅 Amazon Elastic Container Registry 用户指南中的注册表身份验证。

```
aws ecr get-login-password --region region-code | docker login --username Amazon --
password-stdin 111122223333.dkr.ecr.region-code.amazonaws.com
```

6. 将镜像推送到存储库。在此示例中，您将镜像推送到在第一步中创建的 Amazon ECR 私有存储库。有关更多信息，请参阅 Amazon Elastic Container Registry 用户指南中的推送 Docker 镜像。

```
docker push 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

7. 请使用您推送的镜像 `registry/repository:tag` 来更新在上一步中用于确定镜像的清单文件。如果您使用 Helm 图表进行安装，通常可以选择指定 `registry/repository:tag`。安装图表时，请为您推送到存储库的镜像指定 `registry/repository:tag`。

查看 Amazon EKS 附加组件的 Amazon 容器映像注册表

当您将 Amazon Amazon EKS 附加组件部署到集群时，您的节点会从附加组件（例如安装清单或 Helm `values.yaml` 文件）安装机制中指定的注册表中提取所需的容器映像。映像从 Amazon EKS Amazon ECR 私有存储库中提取。Amazon EKS 会将映像复制到 Amazon 区域支持的每个 Amazon EKS 中的存储库。您的节点可以通过互联网从以下任何注册表中提取容器镜像。或者，如果您在 VPC 中为 Amazon ECR（Amazon PrivateLink）创建接口 VPC 端点，则该节点可以通过 Amazon 的网络提取映像。注册表需要使用 Amazon IAM 账户进行身份验证。您的节点使用 Amazon EKS 节点 IAM

角色进行身份验证，该角色在与之关联的 AmazonEC2ContainerRegistryReadOnly 托管 IAM policy 中具有相关权限。

Amazon 区域	注册表
af-south-1	877085696533.dkr.ecr.af-south-1.amazonaws.com
ap-east-1	800184023465.dkr.ecr.ap-east-1.amazonaws.com
ap-east-2	533267051163.dkr.ecr.ap-east-1.amazonaws.com
ap-southeast-3	296578399912.dkr.ecr.ap-southeast-3.amazonaws.com
ap-south-2	900889452093.dkr.ecr.ap-south-2.amazonaws.com
ap-southeast-4	491585149902.dkr.ecr.ap-southeast-4.amazonaws.com
ap-southeast-6	333609536671.dkr.ecr.ap-southeast-6.amazonaws.com
ap-south-1	602401143452.dkr.ecr.ap-south-1.amazonaws.com
ap-northeast-3	602401143452.dkr.ecr.ap-northeast-3.amazonaws.com
ap-northeast-2	602401143452.dkr.ecr.ap-northeast-2.amazonaws.com
ap-southeast-1	602401143452.dkr.ecr.ap-southeast-1.amazonaws.com
ap-southeast-2	602401143452.dkr.ecr.ap-southeast-2.amazonaws.com

Amazon 区域	注册表
ap-southeast-7	121268973566.dkr.ecr.ap-southeast-7.amazonaws.com
ap-northeast-1	602401143452.dkr.ecr.ap-northeast-1.amazonaws.com
cn-north-1	918309763551.dkr.ecr.cn-north-1.amazonaws.com.cn
cn-northwest-1	961992271922.dkr.ecr.cn-northwest-1.amazonaws.com.cn
eu-central-1	602401143452.dkr.ecr.eu-central-1.amazonaws.com
eu-west-1	602401143452.dkr.ecr.eu-west-1.amazonaws.com
eu-west-2	602401143452.dkr.ecr.eu-west-2.amazonaws.com
eu-south-1	590381155156.dkr.ecr.eu-south-1.amazonaws.com
eu-west-3	602401143452.dkr.ecr.eu-west-3.amazonaws.com
eu-south-2	455263428931.dkr.ecr.eu-south-2.amazonaws.com
eu-north-1	602401143452.dkr.ecr.eu-north-1.amazonaws.com
eu-central-2	900612956339.dkr.ecr.eu-central-2.amazonaws.com
il-central-1	066635153087.dkr.ecr.il-central-1.amazonaws.com

Amazon 区域	注册表
mx-central-1	730335286997.dkr.ecr.mx-central-1.amazonaws.com
me-south-1	558608220178.dkr.ecr.me-south-1.amazonaws.com
me-central-1	759879836304.dkr.ecr.me-central-1.amazonaws.com
us-east-1	602401143452.dkr.ecr.us-east-1.amazonaws.com
us-east-2	602401143452.dkr.ecr.us-east-2.amazonaws.com
us-west-1	602401143452.dkr.ecr.us-west-1.amazonaws.com
us-west-2	602401143452.dkr.ecr.us-west-2.amazonaws.com
ca-central-1	602401143452.dkr.ecr.ca-central-1.amazonaws.com
ca-west-1	761377655185.dkr.ecr.ca-west-1.amazonaws.com
sa-east-1	602401143452.dkr.ecr.sa-east-1.amazonaws.com
us-gov-east-1	151742754352.dkr.ecr.us-gov-east-1.amazonaws.com
us-gov-west-1	013241004608.dkr.ecr.us-gov-west-1.amazonaws.com

Amazon EKS 附加组件

附加组件是为 Kubernetes 应用程序提供辅助操作功能的软件，但并不特定于应用程序。这包括可观测性代理或 Kubernetes 驱动程序等软件，这些软件允许集群与用于联网、计算和存储的基础 Amazon 资源进行交互。附加组件软件通常由 Kubernetes 社群、Amazon 等云提供商或第三方供应商构建和维护。Amazon EKS 会自动为每个集群安装自主管理型附加组件，例如适用于 Kubernetes 的 Amazon VPC CNI 插件、`kube-proxy` 以及 CoreDNS。请注意，VPC CNI 附加组件与 Amazon EKS 混合节点功能不兼容，也不会部署到混合节点。您可以更改附加组件的默认配置并在需要时加以更新。

Amazon EKS 附加组件为 Amazon EKS 集群的一组经策管附加组件提供安装和管理。Amazon EKS 附加组件包含最新的安全补丁、错误修复，并经 Amazon 验证能够与 Amazon EKS 一起使用。Amazon EKS 附加组件允许您始终如一地确保您的 Amazon EKS 集群安全稳定，并减少您在安装、配置和更新上所需执行的工作量。如果是类似于 `kube-proxy` 的自我管理附加组件，若已在您的集群上运行并且可作为 Amazon EKS 附加组件使用，那么您便可以安装 `kube-proxy` Amazon EKS 加载项开始享受 Amazon EKS 附加组件的功能所带来的益处。

您可以通过 Amazon EKS API 为 Amazon EKS 附加组件更新特定的 Amazon EKS 托管配置字段。在附加组件启动后，您还可以直接在 Kubernetes 集群中修改并非由 Amazon EKS 管理的配置字段，包括在适用时为附加组件定义特定的配置字段。Amazon EKS 不会覆盖您所做的这些更改。使用 Kubernetes 服务器端应用功能可以实现这一点。有关更多信息，请参阅 the section called “可以自定义的字段”。

您可以将 Amazon EKS 附加组件与任何 Amazon EKS 节点类型配合使用。有关更多信息，请参阅 管理计算。

您可以使用 Amazon EKS API、Amazon Web Services 管理控制台、Amazon CLI 和 `eksctl` 添加、更新或删除 Amazon EKS 附加组件。您也可以使用 Amazon CloudFormation 创建 Amazon EKS 附加组件。

注意事项

使用 Amazon EKS 附加组件时应注意以下几点：

- 要为集群配置附加组件，您的 IAM 主体必须具有 IAM 权限，以使用附加组件。如需了解更多信息，请参阅 Amazon Elastic Kubernetes Service 定义的操作 中的名称中带有 Addon 的操作。
- Amazon EKS 附加组件能在您为集群预置或配置的节点上运行。节点类型包括 Amazon EC2 实例 、Fargate 和混合节点。
- 您可以修改并非由 Amazon EKS 管理的字段以自定义 Amazon EKS 附加组件的安装。有关更多信息，请参阅 the section called “可以自定义的字段”。

- 如果使用 Amazon Web Services 管理控制台 创建集群，则 Amazon EKS kube-proxy、适用于 Kubernetes 的 Amazon VPC CNI 插件和 CoreDNS Amazon EKS 附加组件会自动添加到集群中。如果您借助 eksctl 使用 config 文件创建集群，则 eksctl 还可以使用 Amazon EKS 附加组件创建集群。如果使用不带 config 文件的 eksctl 或使用任何其他工具创建集群，则会安装自主管理型 kube-proxy、适用于 Kubernetes 的 Amazon VPC CNI 插件和 CoreDNS 附加组件，不会安装 Amazon EKS 附加组件。您可以自行管理，也可以在集群创建后手动添加 Amazon EKS 附加组件。无论您使用哪种方法创建集群，VPC CNI 附加组件都不会安装到混合节点上。
- eks:addon-cluster-admin ClusterRoleBinding 将 cluster-admin ClusterRole 绑定到 eks:addon-manager Kubernetes 身份。该角色拥有 eks:addon-manager 身份所需的权限，可以创建 Kubernetes 命名空间并将附加组件安装到命名空间中。如果删除 eks:addon-cluster-admin ClusterRoleBinding，Amazon EKS 集群将继续运行，但 Amazon EKS 将无法再管理任何附加组件。以下平台版本开头的所有集群都使用新的 ClusterRoleBinding。
- Amazon 提供的部分 EKS 附加组件经过验证，可以与 Amazon EKS 混合节点功能兼容。有关更多信息，请参阅 the section called “ Amazon 附加组件”中的兼容性表。

附加组件的自定义命名空间

对于社区和 Amazon 附加组件，您可以选择在创建附加组件时指定自定义命名空间。在特定命名空间中安装附加组件后，必须移除并重新创建该附加组件才能更改其命名空间。

如果您未指定命名空间，它将使用附加组件的预定义命名空间。

使用自定义命名空间可以更好地组织和隔离 EKS 集群中的附加组件对象。这种灵活性有助于您将附加组件与您的运营需求和现有命名空间策略保持一致。

您可以在创建附加组件时设置自定义命名空间。有关更多信息，请参阅 the section called “创建附加组件”。

获取附加组件的预定义命名空间

如果您不指定，附加组件的预定义命名空间就是它将安装到的命名空间。

要获取附加组件的预定义命名空间，请使用以下命令：

```
aws eks describe-addon-versions --addon-name <addon-name> --query
 "addons[].defaultNamespace"
```

输出示例：

```
[
    "kube-system"
]
```

Amazon EKS 自动模式的注意事项

Amazon EKS 自动模式包含了若干提供基本集群功能的功能，包括：

- 容器组联网
- 服务联网
- 集群 DNS
- 自动扩缩
- 数据块存储
- 负载均衡器控制器
- 容器组身份代理
- 节点监控代理

使用自动模式计算时，许多常用的 EKS 附加组件都变得多余，例如：

- Amazon VPC CNI
- kube-proxy
- CoreDNS
- Amazon EBS CSI 驱动程序
- EKS 容器组身份代理

但是，如果集群将自动模式与其他计算选项（例如自主管理型 EC2 实例、托管式节点组或 Amazon Fargate）结合使用，则这些附加组件仍然是必需的。Amazon 使用反亲和性规则增强了 EKS 附加组件，可自动确保附加组件容器组仅在支持的计算类型上进行调度。此外，用户现在可以利用 EKS 附加组件 `DescribeAddonVersions` API，来验证每个附加组件及其特定版本支持的计算类型。此外，使用 EKS 自动模式时，上面列出的控制器是在 Amazon 拥有的基础设施上运行的。因此，您可能不会在账户中看到这些控制器，除非您在其他类型的计算中使用 EKS 自动模式，对于后一种情况，您会看到在集群上安装的控制器。

如果您计划在现有集群上启用 EKS 自动模式，则可能需要升级某些附加组件的版本。有关更多信息，请参阅 EKS 自动模式的the section called “必需的附加组件版本”。

支持

Amazon 会发布具有不同支持级别的多种附加组件。

- Amazon 附加组件：此类附加组件由 Amazon 构建并提供全面支持。
  - 将 Amazon 附加组件与其他 Amazon 服务配合使用，例如 Amazon EFS。
  - 有关更多信息，请参阅 the section called “ Amazon 附加组件”。
- Amazon Marketplace 附加组件：此类附加组件经 Amazon 扫描并由独立的 Amazon 合作伙伴提供支持。
  - 使用 Marketplace 附加组件可为集群添加重要的复杂功能，例如使用 Splunk 进行监控。
  - 有关更多信息，请参阅 the section called “Marketplace 附加组件”。
- 社区附加组件：此类附加组件由 Amazon 扫描，但由开源社区提供支持。
  - 使用社区附加组件可降低安装常用开源软件（例如 Kubernetes Metrics Server）的复杂性。
  - 社区附加组件由 Amazon 基于源代码进行打包。Amazon 仅针对社区附加组件的版本兼容性进行验证。
  - 有关更多信息，请参阅 the section called “社区附加组件”。

下表详细说明了每种附加组件类型的支持范围：

类别	功能	Amazon 附加组件	Amazon Marketplace 附加组件	社区附加组件
开发	由 Amazon 构建	是	否	是
开发	经 Amazon 验证	是	否	是*
开发	经 Amazon 合作伙伴验证	否	是	否
维护	经 Amazon 扫描	支持	是	是
维护	由 Amazon 修补	是	否	是
维护	由 Amazon 合作伙伴修补	否	是	否

类别	功能	Amazon 附加组件	Amazon Marketplace 附加组件	社区附加组件
分配	由 Amazon 发布	是	否	是
分配	由 Amazon 合作伙伴发布	否	是	否
支持	由 Amazon 提供基本安装支持	支持	是	是
支持	Amazon 提供全面支持	是	否	否
支持	Amazon 合作伙伴提供全面支持	否	是	否

*：社区附加组件的验证仅包括 Kubernetes 版本兼容性。例如，如果您在集群上安装了社区附加组件，则 Amazon 会检查其是否与集群的 Kubernetes 版本兼容。

Amazon Marketplace 附加组件可能会从 Amazon 之外的来源下载其他软件依赖项。这些外部依赖项未经 Amazon 扫描或验证。在部署会获取外部依赖项的 Amazon Marketplace 附加组件时，请注意您的安全要求。

Amazon 附加组件

可在您的集群上创建以下 Amazon EKS 附加组件。您可以使用 `eksctl`、Amazon Web Services 管理控制台或 Amazon CLI 查看可用附加组件的最新列表。要查看所有可用附加组件或安装附加组件，请参阅 the section called “创建附加组件”。如果附加组件需要 IAM 权限，则集群必须具有 IAM OpenID Connect（OIDC）提供者。要确定是否具有一个提供商，还是创建一个提供商，请参阅the section called “IAM OIDC 提供者”。安装附加组件后，您可以对其进行创建或将其删除。有关更多信息，请参阅the section called “更新附加组件”或the section called “移除附加组件”。要详细了解有关使用 Amazon EKS 混合节点功能运行 EKS 附加组件的特定注意事项，请参阅the section called “配置附加组件”。

您可以使用以下任何 Amazon EKS 附加组件。

说明	了解更多	兼容的计算类型
为集群提供原生 VPC 联网	the section called “适用于 Kubernetes 的 Amazon VPC CNI 插件”	EC2
一个灵活、可扩展的 DNS 服务器，可用作 Kubernetes 集群 DNS	the section called “CoreDNS”	EC2、Fargate、EKS 自动模式、EKS 混合节点
在每个 Amazon EC2 节点上维护网络规则	the section called “Kube-proxy ”	EC2、EKS 混合节点
为集群提供 Amazon EBS 存储	the section called “Amazon EBS CSI 驱动程序”	EC2
为集群提供 Amazon EFS 存储	the section called “Amazon EFS CSI 驱动程序”	EC2、EKS 自动模式
为集群提供适用于 Lustre 的 Amazon FSx 存储	the section called “Amazon FSx CSI 驱动程序”	EC2、EKS 自动模式
为集群提供 Amazon S3 存储	the section called “适用于 Amazon S3 的 Mountpoint CSI 驱动程序”	EC2、EKS 自动模式
检测其他节点运行状况问题	the section called “节点监控代理”	EC2、EKS 混合节点
实现在兼容的 CSI 驱动程序（例如 Amazon EBS CSI 驱动程序）中使用快照功能	the section called “CSI 快照控制器”	EC2、Fargate、EKS 自动模式、EKS 混合节点
SageMaker HyperPod 任务监管可优化 Amazon EKS 集群中各团队的计算资源分配和使用情况，从而解决任务优先级划分和资源共享效率低下的问题。	the section called “Amazon SageMaker HyperPod 任务监管”	EC2、EKS 自动模式、

说明	了解更多	兼容的计算类型
Amazon SageMaker HyperPod 可观测性附加组件为 HyperPod 集群提供全面的监控和可观测性功能。	the section called “Amazon SageMaker HyperPod 可观测性附加组件”	EC2、EKS 自动模式、
Amazon SageMaker HyperPod 训练操作程序通过高级计划和资源管理功能，在 Amazon EKS 集群上实现高效的分布式训练。	the section called “Amazon SageMaker HyperPod 训练操作程序”	EC2、EKS 自动模式
一种 Kubernetes 代理程序，用于收集网络流量数据并将其报告给 Amazon CloudWatch，从而全面监控集群节点之间的 TCP 连接。	the section called “Amazon Network Flow Monitor Agent”	EC2、EKS 自动模式
OpenTelemetry 项目的安全、生产就绪型发行版，由 Amazon 提供支持	the section called “适用于 OpenTelemetry 的 Amazon Distro”	EC2、Fargate、EKS 自动模式、EKS 混合节点
用于分析和处理基础数据源的安全监控服务，包括 Amazon CloudTrail 管理事件和 Amazon VPC 流日志等。Amazon GuardDuty 还处理 Kubernetes 审计日志和运行时监控等功能	the section called “Amazon GuardDuty 代理”	EC2、EKS 自动模式
Amazon 提供的监控和可观测性服务。此附加组件将会安装 CloudWatch 代理，并启用 CloudWatch Application Signals 和 CloudWatch Container Insights，从而增强 Amazon EKS 的可观测性	the section called “Amazon CloudWatch 可观测性代理”	EC2、EKS 自动模式、EKS 混合节点

说明	了解更多	兼容的计算类型
能够用于管理应用程序凭证，类似于 EC2 实例配置文件为 EC2 实例提供凭证的方式	the section called “EKS 容器组身份代理”	EC2、EKS 混合节点
启用证书管理器以从 Amazon Private CA 颁发 X.509 证书。需要证书管理器。	the section called “适用于 Kubernetes 的 Amazon Private CA Connector”	EC2、Fargate、EKS 自动模式、EKS 混合节点
生成有关 SR-IOV 网络设备性能的 Prometheus 指标	the section called “SR-IOV 网络指标导出器”	EC2
从 Amazon Secrets Manager 中检索密码，从 Amazon Systems Manager Parameter Store 中检索参数，然后将它们作为文件挂载到 Kubernetes 容器组（pod）中。	the section called “Amazon Secrets Store CSI Driver 提供程序”	EC2、EKS 自动模式、EKS 混合节点
借助 Spaces，您可以创建和管理 JupyterLab 和代码编辑器应用程序，以运行交互式机器学习工作负载。	the section called “Amazon SageMaker Spaces”	HyperPod

适用于 Kubernetes 的 Amazon VPC CNI 插件

适用于 Kubernetes Amazon EKS 附加组件的 Amazon VPC CNI 插件是一种 Kubernetes 容器网络接口（CNI）插件，可为集群提供原生 VPC 联网功能。默认情况下，在每个 Amazon EC2 节点上安装自行管理的或托管类型的附加组件。有关更多信息，请参阅 Kubernetes 容器网络接口（CNI）插件。

> **Note**
>
> 您无需在 Amazon EKS 自动模式集群上安装此附加组件。有关更多信息，请参阅 the section called “Amazon EKS 自动模式的注意事项”。

Amazon EKS 附加组件名称为 `vpc-cni`。

所需的 IAM 权限

此附加组件使用 Amazon EKS 的服务账户 IAM 角色功能。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。

如果集群使用 IPv4 系列，则需要 AmazonEKS_CNI_Policy 中的权限。如果集群使用 IPv6 系列，则必须使用 IPv6 模式中的权限创建 IAM 策略。您可以创建 IAM 角色，将其中一个策略附加到该角色，并使用以下命令为附加组件使用的 Kubernetes 服务账户添加注释。

将 *my-cluster* 替换为集群名称，并将 *AmazonEKSVPCCNIRole* 替换为角色名称。如果集群使用 IPv6 系列，则将 *AmazonEKS_CNI_Policy* 替换为您创建的策略名称。此命令要求您为您的设备安装 eksctl。如果您需要使用其他工具来创建角色、将策略附加到该角色并为 Kubernetes 服务账户添加注释，请参阅the section called “分配 IAM 角色”。

```
eksctl create iamserviceaccount --name aws-node --namespace kube-system --cluster my-
cluster --role-name AmazonEKSVPCCNIRole \
    --role-only --attach-policy-arn arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy --
approve
```

更新信息

您一次只能更新一个次要版本。例如，如果当前版本为 `1.28.x-eksbuild.y` ，并且您想要更新到 `1.30.x-eksbuild.y` ，则必须首先更新到 `1.29.x-eksbuild.y` ，再更新到 `1.30.x-eksbuild.y` 。有关更新附加组件的更多信息，请参阅 the section called “更新（EKS 附加组件）”。

CoreDNS

CoreDNS Amazon EKS 附加组件是一个灵活、可扩展的 DNS 服务器，可用作 Kubernetes 集群 DNS。默认情况下，创建集群时会安装自行管理或托管类型的附加组件。当您启动具有至少一个节点的 Amazon EKS 集群时，无论集群中部署的节点数量如何，预设情况下都会部署 CoreDNS 镜像的两个副本。CoreDNS Pod 为集群中的所有 Pod 提供名称解析。如果集群包含命名空间与 CoreDNS 部署的命名空间相匹配的 Fargate 配置文件，则可以将 CoreDNS 容器组（pod）部署到 Fargate 节点。有关更多信息，请参阅 the section called “定义配置文件”。

> **Note**
>
> 您无需在 Amazon EKS 自动模式集群上安装此附加组件。有关更多信息，请参阅 the section called “Amazon EKS 自动模式的注意事项”。

Amazon EKS 附加组件名称为 coredns。

所需的 IAM 权限

此附加组件不需要任何权限。

附加信息

要了解有关 CoreDNS 的更多信息，请参阅 Kubernetes 文档中的使用 CoreDNS 进行服务发现和自定义 DNS 服务。

Kube-proxy

Kube-proxy Amazon EKS 附加组件在每个 Amazon EC2 节点上维护网络规则。其可实现与容器组（pod）的网络通信。默认情况下，在集群中的每个 Amazon EC2 节点上安装自行管理或托管类型的附加组件。

> **Note**
>
> 您无需在 Amazon EKS 自动模式集群上安装此附加组件。有关更多信息，请参阅 the section called “Amazon EKS 自动模式的注意事项”。

Amazon EKS 附加组件名称为 kube-proxy。

所需的 IAM 权限

此附加组件不需要任何权限。

更新信息

在更新当前版本之前，请注意以下要求：

- Amazon EKS 集群上的 Kube-proxy 具有与 Kubernetes 相同的兼容性和偏斜策略。

附加信息

要了解有关 kube-proxy 的更多信息，请参阅 Kubernetes 文档中的 kube-proxy。

Amazon EBS CSI 驱动程序

Amazon EBS CSI 驱动程序 Amazon EKS 附加组件是一种 Kubernetes 容器存储接口（CSI）插件，可为集群提供 Amazon EBS 存储。

> **Note**
>
> 您无需在 Amazon EKS 自动模式集群上安装此附加组件。自动模式包含块存储功能。有关更多信息，请参阅 the section called “部署有状态工作负载”。

Amazon EKS 附加组件名称为 `aws-ebs-csi-driver`。

所需的 IAM 权限

此附加组件使用 Amazon EKS 的服务账户 IAM 角色功能。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。AmazonEBSCSIDriverPolicy Amazon 托管策略中的权限是必需的。您可以使用以下命令创建 IAM 角色并将托管策略附加到其上。将 *my-cluster* 替换为您的集群的名称，并将 *AmazonEKS_EBS_CSI_DriverRole* 替换为您的角色的名称。此命令要求您为您的设备安装 eksctl。如果您需要使用其他工具或需要使用自定义 KMS 密钥进行加密，请参阅 the section called “第 1 步：创建 IAM 角色”。

```
eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws-cn:iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicy \
    --approve
```

附加信息

要了解有关此附加组件的更多信息，请参阅 the section called “Amazon EBS”。

Amazon EFS CSI 驱动程序

Amazon EFS CSI 驱动程序 Amazon EKS 附加组件是一种 Kubernetes 容器存储接口（CSI）插件，可为集群提供 Amazon EFS 存储。

Amazon EKS 附加组件名称为 `aws-efs-csi-driver`。

所需的 IAM 权限

必需的 IAM 权限 – 此附加组件使用 Amazon EKS 的服务账户的 IAM 角色功能。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。AmazonEFSCSIDriverPolicy Amazon 托管策略中的权限是

必需的。您可以使用以下命令创建 IAM 角色并为其附加托管式策略。将 *my-cluster* 替换为您的集群的名称，并将 *AmazonEKS_EFS_CSI_DriverRole* 替换为您的角色的名称。这些命令要求您的设备事先安装 eksctl。如果您需要使用其他工具，请参阅 the section called “第 1 步：创建 IAM 角色”。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --role-only \
    --attach-policy-arn arn:aws-cn:iam::aws:policy/service-role/
AmazonEFSCSIDriverPolicy \
    --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
    sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
 "$TRUST_POLICY"
```

附加信息

要了解有关此附加组件的更多信息，请参阅 the section called “Amazon EFS”。

Amazon FSx CSI 驱动程序

Amazon FSx CSI 驱动程序 Amazon EKS 附加组件是一种 Kubernetes 容器存储接口（CSI）插件，可为集群提供适用于 Lustre 的 Amazon FSx 存储。

Amazon EKS 附加组件名称为 `aws-fsx-csi-driver`。

Note

- 集群中预先安装的 Amazon FSx CSI 驱动程序可能会导致附加组件安装失败。当您在存在非 EKS FSx CSI 驱动程序的情况下尝试安装 Amazon EKS 附加组件版本时，由于资源冲突，安装将失败。在安装过程中使用 `OVERWRITE` 标志解决此问题：

  ```
  aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
   --resolve-conflicts OVERWRITE
  ```

- Amazon FSx CSI 驱动程序 EKS 附加组件需要 EKS 容器组身份代理进行身份验证。如果没有此组件，附加组件将因错误 `Amazon EKS Pod Identity agent is not installed in the cluster` 而失败，从而无法进行卷操作。在部署 FSx CSI 驱动程序附加组件之前或之后安装容器组身份代理。有关更多信息，请参阅 the section called “设置代理”。

所需的 IAM 权限

此附加组件使用 Amazon EKS 的服务账户 IAM 角色功能。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。AmazonFSxFullAccess Amazon 托管策略中的权限是必需的。您可以使用以下命令创建 IAM 角色并将托管策略附加到其上。将 *my-cluster* 替换为集群名称，并将 *AmazonEKS_FSx_CSI_DriverRole* 替换为角色名称。此命令要求您为您的设备安装 eksctl。

```
eksctl create iamserviceaccount \
    --name fsx-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKS_FSx_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws-cn:iam::aws:policy/AmazonFSxFullAccess \
    --approve
```

附加信息

要了解有关此附加组件的更多信息，请参阅 the section called “Amazon FSx for Lustre”。

适用于 Amazon S3 的 Mountpoint CSI 驱动程序

适用于 Amazon S3 的 Mountpoint CSI 驱动程序 Amazon EKS 附加组件是一种 Kubernetes 容器存储接口（CSI）插件，可为集群提供 Amazon S3 存储。

Amazon EKS 附加组件名称为 `aws-mountpoint-s3-csi-driver`。

所需的 IAM 权限

此附加组件使用 Amazon EKS 的服务账户 IAM 角色功能。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。

创建的 IAM 角色需要一个允许访问 S3 的策略。创建策略时，请遵循 Mountpoint IAM 权限建议。您也可以使用 Amazon 托管式策略 AmazonS3FullAccess，但此托管式策略授予的权限超出了 Mountpoint 所需的权限。

您可以使用以下命令创建 IAM 角色，并为其附加策略。将 *my-cluster* 替换为您的集群的名称，将 *region-code* 替换为正确的 Amazon 区域代码，将 *AmazonEKS_S3_CSI_DriverRole* 替换为您的角色的名称，并将 *AmazonEKS_S3_CSI_DriverRole_ARN* 替换为角色 ARN。这些命令要求您的设备事先安装 eksctl。有关如何使用 IAM 控制台或 Amazon CLI 的说明，请参阅the section called “步骤 2：创建 IAM 角色”。

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
    --name s3-csi-driver-sa \
    --namespace kube-system \
    --cluster $CLUSTER_NAME \
    --attach-policy-arn $POLICY_ARN \
    --approve \
    --role-name $ROLE_NAME \
    --region $REGION \
    --role-only
```

附加信息

要了解有关此附加组件的更多信息，请参阅 the section called “适用于 Amazon S3 的 Mountpoint”。

CSI 快照控制器

借助容器存储接口（CSI）快照控制器，可以在 Amazon EBS CSI 驱动程序等兼容的 CSI 驱动程序中使用快照功能。

Amazon EKS 附加组件名称为 `snapshot-controller`。

所需的 IAM 权限

此附加组件不需要任何权限。

附加信息

要了解有关此附加组件的更多信息，请参阅 the section called “CSI 快照控制器”。

Amazon SageMaker HyperPod 任务监管

SageMaker HyperPod 任务监管是一个强大的管理系统，旨在简化资源分配，并确保团队和项目高效利用 Amazon EKS 集群计算资源。这为管理员提供了设置以下方面的功能：

- 各种任务的优先级别
- 各个团队的计算分配
- 各个团队如何借出和借用闲置计算
- 某个团队是否抢占了自己的任务

HyperPod 任务监管还提供 Amazon EKS 集群可观测性功能，从而实现对集群容量的实时可见性。这包括计算可用性和使用情况、团队分配和利用率以及任务运行和等待时间信息，使您能够做出明智的决策以及主动进行资源管理。

Amazon EKS 附加组件名称为 `amazon-sagemaker-hyperpod-taskgovernance`。

所需的 IAM 权限

此附加组件不需要任何权限。

附加信息

要了解有关该附加组件的更多信息，请参阅 SageMaker HyperPod 任务监管

Amazon SageMaker HyperPod 可观测性附加组件

Amazon SageMaker HyperPod 可观测性附加组件为 HyperPod 集群提供全面的监控和可观测性功能。此附加组件自动部署和管理基本的监控组件，包括节点导出器、DCGM 导出器、kube-state-metrics 和 EFA 导出器。它收集指标并将其转发给客户指定的 Amazon Managed Prometheus（AMP）实例，并公开一个 OTLP 端点，用于从客户训练作业中获取自定义指标和事件。

该附加组件通过从各种组件（包括 HyperPod 任务治理附加组件、HyperPod Training Operator、Kubeflow 和 KEDA）中抓取指标，与更广泛的 HyperPod 生态系统集成。所有收集的指标都集中在 Amazon Managed Prometheus 中，使客户能够通过 Amazon Managed Grafana 控制面板获得统一的可观测性视图。这为整个 HyperPod 环境中的集群运行状况、资源利用率和训练作业性能提供了端到端的可见性。

Amazon EKS 附加组件名称为 `amazon-sagemaker-hyperpod-observability`。

所需的 IAM 权限

此附加组件使用 Amazon EKS 的服务账户 IAM 角色功能。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。需要以下托管策略：

- `AmazonPrometheusRemoteWriteAccess`：用于将指标从集群远程写入到 AMP
- `CloudWatchAgentServerPolicy`：用于将集群中的日志远程写入到 CloudWatch

附加信息

要了解有关该附加组件及其功能的更多信息，请参阅 SageMaker HyperPod Observability。

Amazon SageMaker HyperPod 训练操作程序

Amazon SageMaker HyperPod 训练操作程序通过高效管理大型 GPU 集群上的分布式训练，帮助您加快生成式人工智能模型的开发。它引入了智能故障恢复、挂起作业检测和进程级管理功能，可最大限度地减少训练中断并降低成本。与发生故障时需要完全重启作业的传统训练基础设施不同，此操作程序会精确执行流程恢复，以保持训练作业顺利运行。

该操作程序还可与 HyperPod 的运行状况监控和可观测性功能结合使用，支持实时查看训练执行情况，并自动监控损失激增和吞吐量下降等关键指标。无需更改代码，即可通过简单的 YAML 配置定义恢复策略，从而快速响应无法恢复的训练状态并协助完成恢复。这些监控和恢复功能协同工作，可以保持最佳训练性能，同时最大限度地减少运营开销。

Amazon EKS 附加组件名称为 `amazon-sagemaker-hyperpod-training-operator`。

有关更多信息，请参阅《Amazon SageMaker 开发人员指南》中的使用 HyperPod 训练操作符。

所需的 IAM 权限

此附加组件需要 IAM 权限，并使用 EKS 容器组身份。

Amazon 建议使用 `AmazonSageMakerHyperPodTrainingOperatorAccess` 托管式策略。

有关更多信息，请参阅《Amazon SageMaker Developer Guide》中的 Installing the training operator。

附加信息

要了解有关该附加组件的更多信息，请参阅 SageMaker HyperPod training operator。

Amazon Network Flow Monitor Agent

Amazon CloudWatch Network Flow Monitor Agent 是一款 Kubernetes 应用程序，可从集群中的所有节点收集 TCP 连接统计数据，并将网络流量报告发布到 Amazon CloudWatch Network Flow Monitor Ingestion API。

Amazon EKS 附加组件名称为 `aws-network-flow-monitoring-agent`。

所需的 IAM 权限

此附加组件需要 IAM 权限。

您需要向此附加组件附加 `CloudWatchNetworkFlowMonitorAgentPublishPolicy` 托管式策略 。

有关所需 IAM 设置的更多信息，请参阅 Amazon CloudWatch Network Flow Monitor Agent GitHub 存储库中的 IAM Policy。

有关此托管式策略的更多信息，请参阅《Amazon CloudWatch 用户指南》中的 CloudWatchNetworkFlowMonitorAgentPublishPolicy。

附加信息

要了解有关此附加组件的更多信息，请参阅 Amazon CloudWatch Network Flow Monitor Agent GitHub 存储库。

节点监控代理

节点监控代理 Amazon EKS 附加组件可以检测其他的节点运行状况问题。可选的节点自动修复功能也可以利用这些额外的运行状况信号，根据需要自动替换节点。

> **Note**
>
> 您无需在 Amazon EKS 自动模式集群上安装此附加组件。有关更多信息，请参阅 the section called “Amazon EKS 自动模式的注意事项”。

Amazon EKS 附加组件名称为 `eks-node-monitoring-agent`。

所需的 IAM 权限

此附加组件不需要额外的权限。

附加信息

有关更多信息，请参阅 the section called “节点运行状况”。

适用于 OpenTelemetry 的 Amazon Distro

适用于 OpenTelemetry 的 Amazon Distro Amazon EKS 是 OpenTelemetry 项目的安全、受 Amazon 支持的生产就绪型发行版。有关更多信息，请参阅 GitHub 上的 Amazon Distro for OpenTelemetry。

Amazon EKS 附加组件名称为 `adot`。

所需的 IAM 权限

仅在使用可通过高级配置选择加入的预配置自定义资源时，此附加组件才需要 IAM 权限。

附加信息

有关更多信息，请参阅适用于 OpenTelemetry 的 Amazon Distro 文档中的 Getting Started with Amazon Distro for OpenTelemetry using EKS Add-Ons。

ADOT 要求将 `cert-manager` 附加组件作为先决条件部署在集群上，否则如果使用 https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest `cluster_addons` 属性直接部署，此附加组件将无法运行。有关更多要求，请参阅适用于 OpenTelemetry 的 Amazon Distro 文档中的 Requirements for Getting Started with Amazon Distro for OpenTelemetry using EKS Add-Ons。

Amazon GuardDuty 代理

Amazon GuardDuty 代理 Amazon EKS 附加组件将从 EKS 集群节点收集运行时事件（文件访问、进程执行、网络连接），以供 GuardDuty 运行时监控进行分析。GuardDuty 本身（并非代理）是一项安全监控服务，用于分析和处理基础数据源，包括 Amazon CloudTrail 管理事件和 Amazon VPC 流日志，以及 Kubernetes 审计日志和运行时监控等功能。

Amazon EKS 附加组件名称为 `aws-guardduty-agent`。

所需的 IAM 权限

此附加组件不需要任何权限。

附加信息

有关更多信息，请参阅 Runtime Monitoring for Amazon EKS clusters in Amazon GuardDuty。

- 要检测 Amazon EKS 集群中的潜在安全威胁，请启用 Amazon GuardDuty 运行时监控，然后将 GuardDuty 安全代理部署到您的 Amazon EKS 集群。

Amazon CloudWatch 可观测性代理

Amazon CloudWatch 可观测性代理 Amazon EKS 附加组件是由 Amazon 提供的监控和可观测性服务。此插件将安装 CloudWatch 代理，并启用 CloudWatch Application Signals 和 CloudWatch Container Insights，从而增强 Amazon EKS 的可观测性。有关更多信息，请参阅 Amazon CloudWatch 代理。

Amazon EKS 附加组件名称为 `amazon-cloudwatch-observability`。

所需的 IAM 权限

此附加组件使用 Amazon EKS 的服务账户 IAM 角色功能。有关更多信息，请参阅 the section called “包含 IRSA 的凭证”。AWSXrayWriteOnlyAccess 和 CloudWatchAgentServerPolicy Amazon 托管策略中的权限是必需的。您可以创建 IAM 角色，将托管式策略附加到该角色，并使用以下命令为附加组件使用的 Kubernetes 服务账户添加注释。将 *`my-cluster`* 替换为您的集群的名称，并将 *`AmazonEKS_Observability_role`* 替换为您的角色的名称。此命令要求您为您的设备安装 eksctl。如果您需要使用其他工具来创建角色、将策略附加到该角色并为 Kubernetes 服务账户添加注释，请参阅the section called “分配 IAM 角色”。

```
eksctl create iamserviceaccount \
    --name cloudwatch-agent \
    --namespace amazon-cloudwatch \
    --cluster my-cluster \
    --role-name AmazonEKS_Observability_Role \
    --role-only \
    --attach-policy-arn arn:aws-cn:iam::aws:policy/AWSXrayWriteOnlyAccess \
    --attach-policy-arn arn:aws-cn:iam::aws:policy/CloudWatchAgentServerPolicy \
    --approve
```

附加信息

有关更多信息，请参阅安装 CloudWatch 代理。

适用于 Kubernetes 的 Amazon Private CA Connector

适用于 Kubernetes 的 Amazon Private CA Connector 是证书管理器的附加组件，允许用户从 Amazon Private Certificate Authority（Amazon Private CA）获取证书。

- Amazon EKS 附加组件名称为 `aws-privateca-connector-for-kubernetes`。
- 附加组件命名空间是 `aws-privateca-issuer`。

此附加组件需要 cert-manager，cert-manager 可作为社区附加组件在 Amazon EKS 上使用。有关附加组件的更多信息，请参阅the section called “Cert Manager”。有关安装附加组件的更多信息，请参阅the section called “创建附加组件”。

所需的 IAM 权限

此附加组件需要 IAM 权限。

使用 EKS 容器组身份将 AWSPrivateCAConnectorForKubernetesPolicy IAM 策略附加到 aws-privateca-issuer Kubernetes 服务账户。有关更多信息，请参阅 the section called “使用容器组身份”。

要查看此策略的权限，请参阅《Amazon Managed Policy Reference》中的 AWSPrivateCAConnectorForKubernetesPolicy。

附加信息

有关更多信息，请参阅 Amazon Private CA Issuer for Kubernetes GitHub repository。

有关配置附加组件的更多信息，请参阅 aws-privateca-issuer GitHub 存储库中的 values.yaml。确认 values.yaml 版本与集群上所安装的附加组件版本相匹配。

在 EKS 自动模式下，此附加组件可容忍 system 节点池所使用的 CriticalAddonsOnly 污点。有关更多信息，请参阅 the section called “运行关键附加组件”。

EKS 容器组身份代理

Amazon EKS 容器组身份代理 Amazon EKS 附加组件提供了管理应用程序凭证的功能，类似于 EC2 实例配置文件为 EC2 实例提供凭证的方式。

> **Note**
>
> 您无需在 Amazon EKS 自动模式集群上安装此附加组件。Amazon EKS 自动模式与 EKS 容器组身份集成。有关更多信息，请参阅 the section called “Amazon EKS 自动模式的注意事项”。

Amazon EKS 附加组件名称为 eks-pod-identity-agent。

所需的 IAM 权限

容器组身份代理附加组件本身不需要 IAM 角色。它通过 Amazon EKS 节点 IAM 角色的权限来运行，但无需为该附加组件配置专用 IAM 角色。

更新信息

您一次只能更新一个次要版本。例如，如果当前版本为 1.28.x-eksbuild.y，并且您想要更新到 1.30.x-eksbuild.y，则必须首先更新到 1.29.x-eksbuild.y，再更新到 1.30.x-eksbuild.y。有关更新附加组件的更多信息，请参阅 the section called “更新附加组件”。

SR-IOV 网络指标导出器

SR-IOV 网络指标导出器 Amazon EKS 附加组件收集并以 Prometheus 格式公开 SR-IOV 网络设备的指标。它可以监控 EKS 裸机节点上的 SR-IOV 网络性能。导出器作为 DaemonSet 在具有支持 SR-IOV 的网络接口的节点上运行，并导出可供 Prometheus 抓取的指标。

> **Note**
>
> 此附加组件需要具有支持 SR-IOV 的网络接口的节点。

属性	值
附加组件名称	sriov-network-metrics-exporter
命名空间	monitoring
文档	SR-IOV 网络指标导出器 GitHub 存储库
服务账户名称	无
托管 IAM policy	无
自定义 IAM 权限	无

Amazon Secrets Store CSI Driver 提供程序

Amazon Secrets Store CSI Driver 提供程序是一个附加组件，支持从 Amazon Secrets Manager 中检索密码，从 Amazon Systems Manager Parameter Store 中检索参数，并将它们作为文件挂载到 Kubernetes 容器组（pod）中。

所需的 IAM 权限

该附加组件不需要 IAM 权限。但是，应用程序容器组（pod）需要 IAM 权限才能从 Amazon Secrets Manager 获取密码和从 Amazon Systems Manager Parameter Store 获取参数。安装该附加组件后，必须通过服务账户的 IAM 角色（IRSA）或 EKS 容器组身份配置访问权限。要使用 IRSA，请参阅 Secrets Manager IRSA 设置文档。要使用 EKS 容器组身份，请参阅 Secrets Manager 容器组身份设置文档。

Amazon 建议使用 `AWSSecretsManagerClientReadOnlyAccess` 托管式策略。

有关所需权限的更多信息，请参阅《Amazon 托管式策略参考》中的`AWSSecretsManagerClientReadOnlyAccess`。

附加信息

有关更多信息，请参阅 secrets-store-csi-driver-provider-aws GitHub 存储库。

要了解有关该附加组件的更多信息，请参阅该附加组件的 Amazon Secrets Manager 文档。

Amazon SageMaker Spaces

Amazon SageMaker Spaces 附加组件提供了在 EKS 或 HyperPod-EKS 集群上运行 IDE 和笔记本的功能。管理员可以使用 EKS 控制台在其集群上安装该附加组件，并定义默认空间配置，例如图像、计算资源、笔记本设置的本地存储（要附加到其空间的额外存储）、文件系统和初始化脚本。

AI 开发人员可以使用 kubectl 创建、更新和删除空间。他们可以灵活地使用管理员提供的默认配置，也可以自定义设置。AI 开发人员可以使用本地 VS Code IDE 和/或在管理员配置的自定义 DNS 域上托管 JupyterLab 或 CodeEditor IDE 的 Web 浏览器，访问他们在 EKS 或 Hypod-EKS 上的空间。他们还可以使用 kubernetes 的端口转发功能来访问 Web 浏览器中的空间。

Amazon EKS 附加组件名称为 `amazon-sagemaker-spaces`。

所需的 IAM 权限

此附加组件需要 IAM 权限。有关所需的 IAM 设置的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的 IAM 权限设置。

附加信息

要了解有关该附加组件及其功能的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的 HyperPod 上的 SageMaker AI Notebook。

社区附加组件

您可以使用 Amazon API 来安装社区附加组件，例如 Kubernetes Metrics Server。您可以选择将社区附加组件作为 Amazon EKS 附加组件安装，以降低在多个集群上维护软件的复杂性。

例如，您可以使用 Amazon API、CLI 或管理控制台来安装社区附加组件。您可以在创建集群的过程中安装社区附加组件。

您可以像管理现有 Amazon EKS 附加组件一样管理社区附加组件。社区附加组件与现有附加组件的不同之处在于，社区附加组件具有独特的支持范围。

> **Note**
>
> 是否使用社区附加组件由您自行决定。根据您与 Amazon 之间的责任共担模式，对于这些第三方插件，您需要自行了解将在您的集群中具体安装什么。此外，您还负责确保社区附加组件满足您的集群安全需求。有关更多信息，请参阅 the section called “支持部署到 EKS 的软件”。

社区附加组件并非由 Amazon 构建。Amazon 仅验证社区附加组件的版本兼容性。例如，如果您在集群上安装了社区附加组件，则 Amazon 会检查其是否与集群的 Kubernetes 版本兼容。

但应注意，Amazon 并不为社区附加组件提供全面支持。Amazon 仅支持使用 Amazon API 完成的生命周期操作，例如安装附加组件或删除附加组件。

如果您需要对某个社区附加组件的支持，请利用现有的项目资源。例如，您可以在项目的存储库上创建 GitHub 问题。

确定附加组件类型

您可以使用 Amazon CLI 来确定 Amazon EKS 附加组件的类型。

使用以下 CLI 命令检索附加组件的详细信息。您可以将 `metrics-server` 替换为任何附加组件的名称。

```
aws eks describe-addon-versions --addon-name metrics-server
```

检查 CLI 输出的 `owner` 字段。

```
{
    "addons": [
```

```
        {
            "addonName": "metrics-server",
            "type": "observability",
            "owner": "community",
            "addonVersions": [
```

如果 owner 的值为 community，则该附加组件就是社区附加组件。Amazon 仅支持安装、更新和移除附加组件。如果您对附加组件本身的功能和操作有疑问，请使用 GitHub 问题等社区资源。

安装或更新社区附加组件

您可以像安装或更新其他 Amazon EKS 附加组件一样安装或更新社区附加组件。

- the section called “创建附加组件”
- the section called “更新附加组件”
- the section called “移除附加组件”

可用的社区附加组件

Amazon EKS 提供了以下社区附加组件。

- the section called “Kubernetes Metrics Server”
- the section called “kube-state-metrics”
- the section called “Prometheus Node Exporter”
- the section called “Cert Manager”
- the section called “External DNS”
- the section called “Fluent Bit”

Kubernetes Metrics Server

Kubernetes Metrics Server 是 Kubernetes 内置自动扩缩管道的高效可扩展的容器资源指标来源。该附加组件会从 Kubelets 收集资源指标，并通过指标 API 在 Kubernetes apiserver 中公开这些指标，供 Horizontal Pod Autoscaler 和 Vertical Pod Autoscaler 使用。

属性	值
附加组件名称	metrics-server

属性	值
命名空间	kube-system
文档	GitHub Readme
服务账户名称	无
托管 IAM policy	无
自定义 IAM 权限	无

kube-state-metrics

用于生成和公开集群级别指标的附加组件代理。

Kubernetes API 中 Kubernetes 对象的状态可以作为指标公开。名为 kube-state-metrics 的附加组件代理可以连接到 Kubernetes API 服务器，并公开一个 HTTP 端点，其中包含根据集群中单个对象的状态生成的指标。它会公开有关对象状态的各种信息，例如标签和注释、启动和终止时间、状态或对象当前所处的阶段。

属性	值
附加组件名称	kube-state-metrics
命名空间	kube-state-metrics
文档	Kubernetes 文档中的 Kubernetes 对象状态的指标
服务账户名称	无
托管 IAM policy	无
自定义 IAM 权限	无

Prometheus Node Exporter

Prometheus 导出器，用于由 *NIX 内核公开的硬件和操作系统指标，用 Go 编写，带有可插拔的指标收集器。Prometheus Node Exporter 会公开各种与硬件和内核相关的指标。

属性	值
附加组件名称	`prometheus-node-exporter`
命名空间	`prometheus-node-exporter`
文档	Prometheus 文档中的 Monitoring Linux host metrics with the Node Exporter
服务账户名称	无
托管 IAM policy	无
自定义 IAM 权限	无

Cert Manager

Cert Manager 可用于管理证书的创建和续订。

属性	值
附加组件名称	`cert-manager`
命名空间	`cert-manager`
文档	Cert Manager 文档
服务账户名称	无
托管 IAM policy	无
自定义 IAM 权限	无

External DNS

External DNS EKS 附加组件可用于通过 Kubernetes 资源管理 Route53 DNS 记录。

External DNS 权限可以限定为您要管理的托管区 上的 `route53:ChangeResourceRecordSets`、`route53:ListHostedZones` 和 `route53:ListResourceRecordSets`。

属性	值
附加组件名称	`external-dns`
命名空间	`external-dns`
文档	GitHub Readme
服务账户名称	`external-dns`
托管 IAM policy	`arn:aws-cn:iam::aws:policy/AmazonRoute53FullAccess`
自定义 IAM 权限	无

Fluent Bit

Fluent Bit 是一款轻量级且高性能的日志处理器和转发器。它允许您从不同的来源收集数据/日志，对其进行统一，然后将其发送到多个目标，包括 Amazon CloudWatch Logs、Amazon S3 和 Amazon Kinesis Data Firehose。Fluent Bit 在设计时考虑了性能和资源效率，使其非常适合 Kubernetes 环境。

此附加组件在默认配置中无需 IAM 权限。但是，如果您配置 Amazon 输出位置，则可能需要授予此附加组件 IAM 权限。有关更多信息，请参阅 the section called “使用容器组身份”。

属性	值
附加组件名称	`fluent-bit`
命名空间	`fluent-bit`
文档	Fluent Bit 文档

属性	值
服务账户名称	`fluent-bit`
托管 IAM policy	无
自定义 IAM 权限	无

查看归因

您可以下载社区附加组件的开源署名和许可证信息。

1. 确定要下载归因的附加组件名称和版本。
2. 使用名称和版本更新以下命令：

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/<add-on-name>/<add-on-version>/attributions.zip
```

例如：

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/kube-state-metrics/v2.14.0-eksbuild.1/attributions.zip
```

3. 使用命令下载文件。

使用此 zip 文件查看有关许可证归因的信息。

Amazon Marketplace 附加组件

除了之前的 Amazon EKS 附加组件列表外，您还可以添加来自独立软件供应商的各种操作软件 Amazon EKS 附加组件。选择一个附加组件以了解有关该附加组件及其安装要求的更多信息。

Accuknox

附加组件名称为 `accuknox_kubearmor`，命名空间为 `kubearmor`。Accuknox 发布了该附加组件。

有关此附加组件的信息，请参阅 KubeArmor 文档中的 Getting Started with KubeArmor。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Akuity

附加组件名称为 `akuity_agent`，命名空间为 `akuity`。Akuity 发布了该附加组件。

有关此附加组件的信息，请参阅 Akuity 平台文档中的 Installing the Akuity Agent on Amazon EKS with the Akuity EKS add-on。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Calyptia

附加组件名称为 `calyptia_fluent-bit`，命名空间为 `calytia-fluentbit`。Calyptia 发布了该附加组件。

有关此附加组件的信息，请参阅 Calyptia 文档网站上的 Getting Started with Calyptia Core Agent。

服务账户名称

服务账户名称为 `clyptia-fluentbit`。

Amazon 托管 IAM policy

此附加组件使用 `AWSMarketplaceMeteringRegisterUsage` 托管式策略。有关更多信息，请参阅《Amazon 托管式策略参考指南》中的 AWSMarketplaceMeteringRegisterUsage。

用于创建所需 IAM 角色的命令

以下命令要求集群具有 IAM OpenID Connect（OIDC）提供者。要确定是否具有一个提供商，还是创建一个提供商，请参阅 the section called “IAM OIDC 提供者”。将 *my-cluster* 替换为您的集群的名称，并将 *my-calyptia-role* 替换为您的角色的名称。此命令要求您为您的设备安装 eksctl。如果您需要使用其他工具来创建角色并为 Kubernetes 服务账户添加注释，请参阅the section called “分配 IAM 角色”。

```
eksctl create iamserviceaccount --name service-account-name  --namespace calyptia-
fluentbit --cluster my-cluster --role-name my-calyptia-role \
    --role-only --attach-policy-arn arn:aws-cn:iam::aws:policy/
AWSMarketplaceMeteringRegisterUsage --approve
```

Cisco Observability Collector

附加组件名称为 `cisco_cisco-cloud-observability-collectors`，命名空间为 `appdynamics`。Cisco 发布了该附加组件。

有关此附加组件的信息，请参阅 Cisco AppDynamics 文档中的 Use the Cisco Cloud Observability Amazon Marketplace Add-Ons。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Cisco Observability Operator

附加组件名称为 `cisco_cisco-cloud-observability-operators`，命名空间为 `appdynamics`。Cisco 发布了该附加组件。

有关此附加组件的信息，请参阅 Cisco AppDynamics 文档中的 Use the Cisco Cloud Observability Amazon Marketplace Add-Ons。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

CLOUDSOFT

附加组件名称为 `cloudsoft_cloudsoft-amp`，命名空间为 `cloudsoft-amp`。CLOUDSOFT 发布了该附加组件。

有关此附加组件的信息，请参阅 CLOUDSOFT 文档中的 Amazon EKS ADDON。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Cribl

附加组件名称为 `cribl_cribledge`，命名空间为 `cribledge`。Cribl 发布了该附加组件。

有关此附加组件的信息，请参阅 Cribl 文档中的 Installing the Cribl Amazon EKS Add-on for Edge

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Dynatrace

附加组件名称为 `dynatrace_dynatrace-operator`，命名空间为 `dynatrace`。Dynatrace 发布了该附加组件。

有关此附加组件的信息，请参阅 Dynatrace 文档中的 Kubernetes 监控。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Datree

附加组件名称为 `datree_engine-pro`，命名空间为 `datree`。Datree 发布了该附加组件。

有关此附加组件的信息，请参阅 Datree 文档中的 Amazon EKS-intergration。

服务账户名称

服务账户名称为 datree-webhook-server-awsmp。

Amazon 托管 IAM policy

托管式策略为 AWSLicenseManagerConsumptionPolicy。有关更多信息，请参阅《Amazon 托管式策略参考指南》中的 AWSLicenseManagerConsumptionPolicy。

用于创建所需 IAM 角色的命令

以下命令要求集群具有 IAM OpenID Connect（OIDC）提供者。要确定是否具有一个提供商，还是创建一个提供商，请参阅 the section called “IAM OIDC 提供者”。将 *`my-cluster`* 替换为您的集群的名称，并将 *`my-datree-role`* 替换为您的角色的名称。此命令要求您为您的设备安装 eksctl。如果您需要使用其他工具来创建角色并为 Kubernetes 服务账户添加注释，请参阅the section called “分配 IAM 角色”。

```
eksctl create iamserviceaccount --name datree-webhook-server-awsmp --namespace datree
 --cluster my-cluster --role-name my-datree-role \
    --role-only --attach-policy-arn arn:aws-cn:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

自定义权限

此附加组件不使用自定义权限。

Datadog

附加组件名称为 `datadog_operator`，命名空间为 `datadog-agent`。Datadog 发布了该附加组件。

有关此附加组件的信息，请参阅 Datadog 文档中的 Installing the Datadog Agent on Amazon EKS with the Datadog Operator Add-on。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Groundcover

附加组件名称为 `groundcover_agent`，命名空间为 `groundcover`。Groundcover 发布了该附加组件。

有关此附加组件的信息，请参阅 groundcover 文档中的 Installing the groundcover Amazon EKS Add-on。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

IBM Instana

附加组件名称为 `instana-agent`，命名空间为 `instana-agent`。IBM 发布了该附加组件。

有关该附加组件的信息，请参阅 Amazon 博客中的 Implement observability for Amazon EKS workloads using the Instana Amazon EKS add-on 和 Monitor and optimize Amazon EKS costs with IBM Instana and Kubecost。

Instana Observability（Instana）提供了一个 Amazon EKS 附加组件，可将 Instana 代理部署到 Amazon EKS 集群。客户可以使用此附加组件收集和分析实时性能数据，从而深入了解其容器化应用程序。Instana Amazon EKS 附加组件可让您全面了解 Kubernetes 环境。部署完成后，Instana 代理便会自动发现 Amazon EKS 集群中的组件，包括节点、命名空间、部署、服务和容器组（pod）。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Grafana Labs

附加组件名称为 `grafana-labs_kubernetes-monitoring`，命名空间为 `monitoring`。Grafana Labs 发布了该附加组件。

有关此附加组件的信息，请参阅 Grafana Labs 文档中的 Configure Kubernetes Monitoring as an Add-on with Amazon EKS。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Guance

- 发布者 – GUANCE
- 名称 – `guance_datakit`
- 命名空间：`datakit`
- 服务账户名称 – 此附加组件不使用服务账户。
- Amazon 托管 IAM 策略 – 此附加组件不使用托管策略。
- 自定义 IAM 权限 – 此附加组件不使用自定义权限。
- 安装和使用说明 – 请参阅 Guance 文档中的 Using Amazon EKS add-on。

HA Proxy

名称为 `haproxy-technologies_kubernetes-ingress-ee`，命名空间为 `haproxy-controller`。HA Proxy 发布了该附加组件。

有关此附加组件的信息，请参阅 Datree 文档中的 Amazon EKS-intergration。

服务账户名称

服务账户名称为 `customer defined`。

Amazon 托管 IAM policy

托管式策略为 AWSLicenseManagerConsumptionPolicy。有关更多信息，请参阅《Amazon 托管式策略参考指南》中的 AWSLicenseManagerConsumptionPolicy。

用于创建所需 IAM 角色的命令

以下命令要求集群具有 IAM OpenID Connect（OIDC）提供者。要确定是否具有一个提供商，还是创建一个提供商，请参阅 the section called “IAM OIDC 提供者”。将 *`my-cluster`* 替换为您的集群的名称，并将 *`my-haproxy-role`* 替换为您的角色名称。此命令要求您为您的设备安装 eksctl。如果您需要使用其他工具来创建角色并为 Kubernetes 服务账户添加注释，请参阅the section called “分配 IAM 角色”。

```
eksctl create iamserviceaccount --name service-account-name  --namespace haproxy-
controller --cluster my-cluster --role-name my-haproxy-role \
```

```
    --role-only --attach-policy-arn arn:aws-cn:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

自定义权限

此附加组件不使用自定义权限。

Kpow

附加组件名称为 `factorhouse_kpow`，命名空间为 `factorhouse`。Factorhouse 发布了该附加组件。

有关该附加组件的信息，请参阅 Kpow 文档中的 Amazon Marketplace LM。

服务账户名称

服务账户名称为 kpow。

Amazon 托管 IAM policy

托管式策略为 AWSLicenseManagerConsumptionPolicy。有关更多信息，请参阅《Amazon 托管式策略参考指南》中的 AWSLicenseManagerConsumptionPolicy。

用于创建所需 IAM 角色的命令

以下命令要求集群具有 IAM OpenID Connect（OIDC）提供者。要确定是否具有一个提供商，还是创建一个提供商，请参阅the section called “IAM OIDC 提供者”。将 *my-cluster* 替换为您的集群的名称，并将 *my-kpow-role* 替换为您的角色名称。此命令要求您为您的设备安装 eksctl。如果您需要使用其他工具来创建角色并为 Kubernetes 服务账户添加注释，请参阅the section called “分配 IAM 角色”。

```
eksctl create iamserviceaccount --name kpow --namespace factorhouse --cluster my-
cluster --role-name my-kpow-role \
    --role-only --attach-policy-arn arn:aws-cn:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

自定义权限

此附加组件不使用自定义权限。

Kubecost

附加组件名称为 `kubecost_kubecost`，命名空间为 `kubecost`。Kubecost 发布了该附加组件。

有关该附加组件的信息，请参阅 Kubecost 文档中的 Amazon Cloud Billing Integration。

您必须在集群上安装使用 Amazon EBS 存储 Kubernetes 卷，否则您将收到错误消息。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Kasten

附加组件名称为 `kasten_k10`，命名空间为 `kasten-io`。Kasten by Veeam 发布了该附加组件。

有关此附加组件的信息，请参阅 Kasten 文档中的 Installing K10 on Amazon using Amazon EKS Add-on。

您必须在集群上安装 Amazon EBS CSI 驱动程序，并配置默认 `StorageClass`。

服务账户名称

服务账户名称为 `k10-k10`。

Amazon 托管 IAM policy

托管式策略为 AWSLicenseManagerConsumptionPolicy。有关更多信息，请参阅《Amazon 托管式策略参考指南》中的 AWSLicenseManagerConsumptionPolicy。

用于创建所需 IAM 角色的命令

以下命令要求集群具有 IAM OpenID Connect（OIDC）提供者。要确定是否具有一个提供商，还是创建一个提供商，请参阅 the section called “IAM OIDC 提供者”。将 *my-cluster* 替换为您的集群的名称，并将 *my-kasten-role* 替换为您的角色的名称。此命令要求您为您的设备安装 eksctl。如果您需要使用其他工具来创建角色并为 Kubernetes 服务账户添加注释，请参阅the section called “分配 IAM 角色”。

```
eksctl create iamserviceaccount --name k10-k10 --namespace kasten-io --cluster my-
cluster --role-name my-kasten-role \
    --role-only --attach-policy-arn arn:aws-cn:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

自定义权限

此附加组件不使用自定义权限。

Kong

附加组件名称为 `kong_konnect-ri`，命名空间为 `kong`。Kong 发布了该附加组件。

有关此附加组件的信息，请参阅 Kong 文档中的 Installing the Kong Gateway EKS Add-on。

您必须在集群上安装使用 Amazon EBS 存储 Kubernetes 卷，否则您将收到错误消息。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

LeakSignal

附加组件名称为 `leaksignal_leakagent`，命名空间为 `leakagent`。LeakSignal 发布了该附加组件。

有关此附加组件的信息，请参阅 LeakSignal 文档中的 https://www.leaksignal.com/docs/LeakAgent/Deployment/ Amazon%20EKS%20Addon/[Install the LeakAgent add-on]

您必须在集群上安装使用 Amazon EBS 存储 Kubernetes 卷，否则您将收到错误消息。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

NetApp

附加组件名称为 `netapp_trident-operator`，命名空间为 `trident`。NetApp 发布了该附加组件。

有关此附加组件的信息，请参阅 NetApp 文档中的 Configure the Trident EKS add-on。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

New Relic

附加组件名称为 `new-relic_kubernetes-operator`，命名空间为 `newrelic`。New Relic 发布了该附加组件。

有关此附加组件的信息，请参阅 New Relic 文档中的 Installing the New Relic Add-on for EKS。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Rafay

附加组件名称为 `rafay-systems_rafay-operator`，命名空间为 `rafay-system`。Rafay 发布了该附加组件。

有关此附加组件的信息，请参阅 Rafay 文档中的 Installing the Rafay Amazon EKS Add-on。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

容器组（pod）安全

- 发布者 – RAD SECURITY
- 名称 – `rad-security_rad-security`
- 命名空间：`ksoc`
- 服务账户名称 – 此附加组件不使用服务账户。
- Amazon 托管 IAM 策略 – 此附加组件不使用托管策略。
- 自定义 IAM 权限 – 此附加组件不使用自定义权限。
- 安装和使用说明 – 请参阅 Rad Security 文档中的 Installing Rad Through The Amazon Marketplace。

SolarWinds

- 发布者 – SOLARWINDS
- 名称 – `solarwinds_swo-k8s-collector-addon`
- 命名空间：`solarwinds`
- 服务账户名称 – 此附加组件不使用服务账户。
- Amazon 托管 IAM 策略 – 此附加组件不使用托管策略。

- 自定义 IAM 权限 – 此附加组件不使用自定义权限。
- 安装和使用说明 – 请参阅 SolarWinds 文档中的 Monitor an Amazon EKS cluster。

Solo

附加组件名称为 `solo-io_istio-distro`，命名空间为 `istio-system`。Solo 发布了该附加组件。

有关此附加组件的信息，请参阅 Solo.io 文档中的 Installing Istio。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Snyk

- 发布者 – SNYK
- 名称 – `snyk_runtime-sensor`
- 命名空间：`snyk_runtime-sensor`
- 服务账户名称 – 此附加组件不使用服务账户。
- Amazon 托管 IAM 策略 – 此附加组件不使用托管策略。
- 自定义 IAM 权限 – 此附加组件不使用自定义权限。
- 安装和使用说明 – 请参阅 Snyk 用户文档中的 Snyk runtime sensor。

Stormforge

附加组件名称为 `stormforge_optimize-Live`，命名空间为 `stormforge-system`。Stormforge 发布了该附加组件。

有关此附加组件的信息，请参阅 StormForge 文档中的 Installing the StormForge Agent。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

SUSE

- 发布者：SUSE
- 名称 – `suse_observability-agent`
- 命名空间：`suse-observability`
- 服务账户名称 – 此附加组件不使用服务账户。
- Amazon 托管 IAM 策略 – 此附加组件不使用托管策略。
- 自定义 IAM 权限 – 此附加组件不使用自定义权限。
- 设置和使用说明：请参阅 SUSE 文档中的 Quick Start。

Splunk

附加组件名称为 `splunk_splunk-otel-collector-chart`，命名空间为 `splunk-monitoring`。Splunk 发布了该附加组件。

有关此附加组件的信息，请参阅 Splunk 文档中的 Install the Splunk add-on for Amazon EKS。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Teleport

附加组件名称为 `teleport_teleport`，命名空间为 `teleport`。Teleport 发布了该附加组件。

有关该附加组件的信息，请参阅 Teleport 文档中的 How Teleport Works。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Tetrate

附加组件名称为 `tetrate-io_istio-distro`，命名空间为 `istio-system`。Tetrate Io 发布了该附加组件。

有关此附加组件的信息，请参阅 Tetrate Istio Distro 网站。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Upbound Universal Crossplane

附加组件名称为 `upbound_universal-crossplane`，命名空间为 `upbound-system`。Upbound 发布了该附加组件。

有关此附加组件的信息，请参阅 Upbound 文档中的 Upbound Universal Crossplane (UXP)。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

Upwind

附加组件名称为 upwind，命名空间为 upwind。Upwind 发布了该附加组件。

有关此附加组件的信息，请参阅 Upwind 文档。

服务账户名称

此附加组件不使用服务账户。

Amazon 托管 IAM policy

此附加组件不使用托管策略。

自定义 IAM 权限

此附加组件不使用自定义权限。

创建 Amazon EKS 附加组件

Amazon EKS 附加组件是用于 Amazon EKS 集群的附加软件。所有 Amazon EKS 附加组件都具有以下特性：

- 包含最新安全补丁和错误修复。
- 经过 Amazon 验证，可与 Amazon EKS 一起使用。
- 可减少管理附加软件所需的工作量。

您可以使用 eksctl、Amazon Web Services 管理控制台或 Amazon CLI 创建 Amazon EKS 附加组件。如果附加组件需要某个 IAM 角色，请参阅 Amazon EKS 附加组件中有关特定附加组件的详细信息，以了解有关创建该角色的详细信息。

先决条件

在创建附加组件之前，请完成以下步骤：

- 集群必须已经存在，然后才能为其创建附加组件。有关更多信息，请参阅 the section called “创建集群”。
- 检查附加组件是否需要 IAM 角色。有关更多信息，请参阅 the section called “验证兼容性”。
- 验证 Amazon EKS 附加组件版本是否与集群兼容。有关更多信息，请参阅 the section called “验证兼容性”。
- 验证已在计算机或 Amazon CloudShell 上安装 0.190.0 版或更高版本的 `eksctl` 命令行工具。有关更多信息，请参阅 `eksctl` 网站上的安装。

过程

您可以使用 `eksctl`、Amazon Web Services 管理控制台或 Amazon CLI 创建 Amazon EKS 附加组件。如果附加组件需要 IAM 角色，请参阅来自 Amazon 的 Amazon EKS 附加组件中有关特定附加组件的详细信息，以了解有关创建该角色的详细信息。

创建附加组件（eksctl）

1. 查看某个集群版本的可用附加组件名称。将 *1.33* 替换为集群的版本。

```
eksctl utils describe-addon-versions --kubernetes-version 1.33 | grep AddonName
```

示例输出如下。

```
"AddonName": "aws-ebs-csi-driver",
                        "AddonName": "coredns",
                        "AddonName": "kube-proxy",
                        "AddonName": "vpc-cni",
                        "AddonName": "adot",
                        "AddonName": "dynatrace_dynatrace-operator",
                        "AddonName": "upbound_universal-crossplane",
                        "AddonName": "teleport_teleport",
                        "AddonName": "factorhouse_kpow",
                        [...]
```

2. 查看要创建的附加组件的可用版本。将 *1.33* 替换为集群的版本。将 *name-of-addon* 替换为您要查看其版本的附加组件的名称。该名称必须是上一步返回的名称之一。

```
eksctl utils describe-addon-versions --kubernetes-version 1.33 --name name-of-addon |
 grep AddonVersion
```

以下输出是一个示例，显示了为名为 vpc-cni 的附加组件返回的内容。您可以看到该附加组件有多个可用版本。

```
"AddonVersions": [
    "AddonVersion": "v1.12.0-eksbuild.1",
    "AddonVersion": "v1.11.4-eksbuild.1",
    "AddonVersion": "v1.10.4-eksbuild.1",
    "AddonVersion": "v1.9.3-eksbuild.1",
```

a. 确定您要创建的附加组件是 Amazon EKS 还是 Amazon Marketplace 附加组件。Amazon Marketplace 具有第三方附加组件，需要您完成额外的步骤才能创建附加组件。

```
eksctl utils describe-addon-versions --kubernetes-version 1.33 --name name-of-
addon | grep ProductUrl
```

如果未返回输出，则该附加组件是 Amazon EKS。如果返回输出，则该附加组件是 Amazon Marketplace 附加组件。以下输出适用于名为 teleport_teleport 的附加组件。

```
"ProductUrl": "https://aws.amazon.com/marketplace/pp?sku=3bda70bb-566f-4976-806c-
f96faef18b26"
```

您可以通过返回的 URL 在 Amazon Marketplace 中了解有关该附加组件的更多信息。
如果该附加组件需要订阅，您可以通过 Amazon Marketplace 对其进行订阅。如果您要从 Amazon Marketplace 创建附加组件，则用于创建附加组件的 IAM 主体必须具有创建 AWSServiceRoleForAWSLicenseManagerRole 服务相关角色的权限。有关向 IAM 实体分配权限的更多信息，请参阅《IAM 用户指南》中的添加和移除 IAM 身份权限。

3. 创建 Amazon EKS 附加组件。复制该命令并替换 *user-data*，如下所示：

- 将 *my-cluster* 替换为您的集群的名称。
- 将 *name-of-addon* 替换为您要创建的附加组件的名称。
- 如果您需要早于最新版本的附加组件版本，请将 *latest* 替换为前面步骤的输出中返回的适用版本号。
- 如果附加组件使用了服务账户角色，请将 *111122223333* 替换为您的账户 ID，并将 *role-name* 替换为该角色的名称。有关为服务账户创建角色的说明，请参阅您正在创建的附加组件的文档。

有关可用附加组件的列表，请参阅the section called “ Amazon 附加组件”。指定服务账户角色需要您的集群具有 IAM OpenID Connect（OIDC）提供者。要确定您的集群是否具有此提供程序，或者要创建此提供程序，请参阅 the section called “IAM OIDC 提供者”。

如果附加组件不使用服务账户角色，请删除 `--service-account-role-arnarn:aws-cn:iam::111122223333:role/role-name`。

- 此示例命令将覆盖附加组件的任何现有自行管理版本（如果有的话）的配置。如果您不想覆盖现有的自行管理附加组件的配置，请删除 *`--force`* 选项。如果您删除此选项，并且 Amazon EKS 附加组件需要覆盖现有的自行管理附加组件的配置，那么创建 Amazon EKS 附加组件将会失败，并显示一条帮助您解决冲突的错误消息。在指定此选项之前，请确保 Amazon EKS 附加组件不会管理您需要管理的设置，因为这些设置会被此选项覆盖。

```
eksctl create addon --cluster my-cluster --name name-of-addon --version latest \
    --service-account-role-arn arn:aws-cn:iam::111122223333:role/role-name --force
```

您可以看到该命令所有可用选项的列表。

```
eksctl create addon --help
```

有关可用选项的更多信息，请参阅 `eksctl` 文档中的 Addons（附加组件）。

创建附加组件（Amazon 控制台）

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 选择要为其创建此附加组件的集群名称。
4. 选择附加组件选项卡。
5. 选择获取更多附加组件。
6. 在 Select add-ons（选择附加组件）页面上，选择要添加到集群的附加组件。您可以根据需要添加任意数量的 Amazon EKS 附加组件和 Amazon Marketplace 附加组件。

   对于 Amazon Marketplace 附加组件，用于创建附加组件的 IAM 主体必须有权从 Amazon LicenseManager 读取附加组件的权限。AmazonLicenseManager 需要 AWSServiceRoleForAWSLicenseManagerRole 服务相关角色（SLR），该角色允许 Amazon 资源代表您管理许可证。SLR 是每个账户的一次性要求，您无需为每个附加组件或每个集群创建单独的

SLR。有关向 IAM 主体分配权限的更多信息，请参阅《IAM 用户指南》中的添加和移除 IAM 身份权限。

如果未列出要安装的 Amazon Marketplace 附加组件，则您可以单击页码编号查看更多页面结果或在搜索框中来搜索。在筛选选项中，您也可以按类别、供应商或定价模式进行搜索，然后从搜索结果中选择附加组件。选择要安装的附加组件后，选择下一步。

7. 在配置选定的附加组件设置页面上，执行以下操作：

   a. 选择查看订阅选项，打开订阅选项表单。查看定价详细信息和法律部分，然后选择订阅按钮继续。

   b. 对于版本，请选择要安装的版本。我们建议选择标记为最新的版本，除非您正在创建的个别附加组件推荐了不同的版本。要确定附加组件是否有推荐版本，请参阅您正在创建的附加组件的文档。有关可用附加组件的列表，请参阅the section called “ Amazon 附加组件”。

   c. 您可以使用两个选项为附加组件配置角色：EKS 容器组身份 IAM 角色和服务账户 IAM 角色（IRSA）。按照以下相应步骤选择首选选项。如果您选择的所有附加组件在状态下都有需要订阅字样，请选择下一步。创建集群后，您在订阅到此类附加组件之前无法进一步配置此类附加组件。如果附件组件的状态下没有需要订阅字样：

      i. 对于服务账户的容器组身份 IAM 角色，您可以使用现有 EKS 容器组身份 IAM 角色，也可以使用创建推荐角色按钮创建。此字段将仅提供具有适当信任策略的选项。如果没有可选择的角色，则表示没有具有匹配信任策略的现有角色。要为所选附加组件配置服务账户的 EKS 容器组身份 IAM 角色，请选择创建推荐角色。角色创建向导将在单独的窗口中打开。向导将自动填充角色信息，如下所示。对于要在其中创建 EKS 容器组身份 IAM 角色的各个附加组件，请如下方式完成 IAM 向导中的步骤。

         - 在选择可信实体步骤中，预先选择 EKS 的 Amazon 服务选项和 EKS – 容器组身份的用例，并将自动为附加组件填充相应的信任策略。例如，该角色将使用相应的信任策略创建，该策略包含 pods.eks.amazonaws.com IAM 主体，详见the section called “EKS 容器组身份的优势”。选择下一步。
         - 在添加权限步骤中，系统将为附加组件预先选择角色策略的相应托管策略。例如，对于 Amazon VPC CNI 附加组件，系统将使用托管式策略 `AmazonEKS_CNI_Policy` 创建角色，详见the section called “适用于 Kubernetes 的 Amazon VPC CNI 插件”。选择下一步。
         - 在命名、查看和创建步骤的角色名称中，系统将自动填充附加组件的默认角色名称。例如，对于 Amazon VPC CNI 附加组件，系统将使用 AmazonEKSPodIdentityAmazonVPCCNIRole 名称创建角色。在描述中，默认描述会自动填充附加组件的相应描述。例如，对于 Amazon VPC CNI 附加组件，该角色的创建描述为允许在 Amazon EKS 集群中运行容器组（pod）以访问 Amazon 资源。在信任策略中，查看为附加组件填充的信任策略。选择创建角色。

注意：通过保留默认角色名称，EKS 能够在新集群中或在向现有集群添加附加组件时为附加组件预先选择角色。您仍然可以覆盖此名称，并且该角色可在您的集群中用于该附加组件，但需要从下拉列表中手动选择该角色。

ii. 对于在状态下没有需要订阅且您想使用 IRSA 配置角色的附加组件，请参阅您正在创建的附加组件的文档，以创建 IAM 策略并将其附加至角色。有关可用附加组件的列表，请参阅the section called “ Amazon 附加组件”。选择 IAM 角色需要您的集群具有 IAM OpenID Connect（OIDC）提供者。要确定您的集群是否具有此提供程序，或者要创建此提供程序，请参阅the section called “IAM OIDC 提供者”。

iii. 选择 Optional configuration settings（可选配置设置）。

iv. 如果附加组件需要配置，请在 Configuration values（配置值）框中输入相应的值。要确定附加组件是否需要配置信息，请参阅您正在创建的附加组件的文档。有关可用附加组件的列表，请参阅the section called “ Amazon 附加组件”。

v. 对于冲突解决方法，请选择一个可用的选项。如果对于冲突解决方法，您选择了覆盖，则可能会用 Amazon EKS 附加组件设置覆盖现有附加组件的一个或多个设置。如果不启用此选项，并且与现有设置存在冲突，则操作将失败。您可以使用生成的错误消息对冲突进行故障排除。在选择此选项之前，请确保 Amazon EKS 附加组件不会管理您需要自行管理的设置。

vi. 如果要将附加组件安装到特定的命名空间，请在命名空间字段中输入该命名空间。对于 Amazon 和社区附加组件，可以定义自定义的 Kubernetes 命名空间来安装附加组件。有关更多信息，请参阅 the section called “附加组件的自定义命名空间”。

vii.选择下一步。

8. 在查看和添加页面上，选择创建。附加组件安装完成后，您会看到您安装的附加组件。

9. 如果您安装的任何附加组件需要订阅，请完成以下步骤：

a. 选择该附加组件右下角的 Subscribe（订阅）按钮。您将转到该附加组件在 Amazon Marketplace 中的页面。阅读有关该附加组件的信息，例如其 Product Overview（产品概述）和 Pricing Information（定价信息）。

b. 选择附加组件页面右上角的 Continue to Subscribe（继续订阅）按钮。

c. 通读 Terms and Conditions（条款和条件）。如果您同意这些条款和条件，请选择 Accept Terms（接受条款）。处理订阅可能需要几分钟的时间。在处理订阅时，Return to Amazon EKS Console（返回 Amazon EKS 控制台）按钮将显示为灰色。

d. 订阅处理完成后，Return to Amazon EKS Console（返回 Amazon EKS 控制台）按钮将不再显示为灰色。选择此按钮返回集群的 Amazon EKS 控制台 Add-ons（附加组件）选项卡。

e. 对于您已订阅的附加组件，请选择 Remove and reinstall（删除并重新安装），然后选择 Reinstall add-on（重新安装附加组件）。安装该附加组件可能需要几分钟的时间。安装完成后，您可以配置该附加组件。

创建附加组件（Amazon CLI）

1. 您需要在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
2. 确定哪些附加组件可用。您可以看到所有可用的附加组件及其类型和发布者。您还可以看到通过 Amazon Marketplace 提供的附加组件的 URL。将 *1.33* 替换为集群的版本。

```
aws eks describe-addon-versions --kubernetes-version 1.33 \
    --query 'addons[].{MarketplaceProductUrl: marketplaceInformation.productUrl,
 Name: addonName, Owner: owner Publisher: publisher, Type: type}' --output table
```

示例输出如下。

```
-------------------------------------------------------------------------------------------------
|
 DescribeAddonVersions
  |
+-----------------------------------------------------------
+------------------------------+------------------+-------------
+---------------------+
|                     MarketplaceProductUrl                     |             Name
          |      Owner       |  Publisher   |        Type         |
+-----------------------------------------------------------
+------------------------------+------------------+-------------
+---------------------+
|  None                                                         |  aws-ebs-csi-driver
          |  aws             |  eks         |  storage            |
|  None                                                         |  coredns
          |  aws             |  eks         |  networking         |
```

```
|  None                                                          |  kube-proxy
           |  aws             |  eks         |  networking         |
|  None                                                          |  vpc-cni
           |  aws             |  eks         |  networking         |
|  None                                                          |  adot
           |  aws             |  eks         |  observability      |
| https://aws.amazon.com/marketplace/pp/prodview-brb73nceicv7u |
 dynatrace_dynatrace-operator |  aws-marketplace |  dynatrace   |  monitoring
 |
| https://aws.amazon.com/marketplace/pp/prodview-uhc2iwi5xysoc |  upbound_universal-
crossplane |  aws-marketplace |  upbound     |  infra-management   |
| https://aws.amazon.com/marketplace/pp/prodview-hd2ydsrgqy4li |  teleport_teleport
          |  aws-marketplace |  teleport    |  policy-management  |
| https://aws.amazon.com/marketplace/pp/prodview-vgghgqdsplhvc |  factorhouse_kpow
          |  aws-marketplace |  factorhouse |  monitoring         |
|  [...]                                                         |  [...]
           |  [...]           |  [...]       |  [...]              |
+-----------------------------------------------------------------
+-------------------------------+------------------+--------------
+---------------------+
```

您的输出可能会有所不同。在此输出示例中，有三个不同的 `networking` 类型的附加组件和五个发布者类型为 `eks` 的附加组件。在 `Owner` 列中值为 `aws-marketplace` 的附加组件可能需要订阅才能安装。您可以访问相应的 URL 以了解有关该附加组件的更多信息并进行订阅。

3. 您可以看到每个附加组件的可用版本。将 *1.33* 替换为集群的版本，将 *vpc-cni* 替换为上一步返回的附加组件的名称。

```
aws eks describe-addon-versions --kubernetes-version 1.33 --addon-name vpc-cni \
    --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
 compatibilities[0].defaultVersion}' --output table
```

示例输出如下。

```
-----------------------------------------
|          DescribeAddonVersions         |
+-----------------+----------------------+
| Defaultversion  |       Version        |
+-----------------+----------------------+
|  False          |  v1.12.0-eksbuild.1  |
|  True           |  v1.11.4-eksbuild.1  |
|  False          |  v1.10.4-eksbuild.1  |
```

```
|  False          |  v1.9.3-eksbuild.1   |
+-----------------+----------------------+
```

默认情况下，Defaultversion 列中值为 True 的版本是创建附加组件时使用的版本。

4. （可选）通过运行以下命令查找所选附加组件的配置选项：

```
aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.12.0-
eksbuild.1
```

```
{
    "addonName": "vpc-cni",
    "addonVersion": "v1.12.0-eksbuild.1",
    "configurationSchema": "{\"$ref\":\"#/definitions/VpcCni\",\"$schema\":\"http://
json-schema.org/draft-06/schema#\",\"definitions\":{\"Cri\":{\"additionalProperties
\":false,\"properties\":{\"hostPath\":{\"$ref\":\"#/definitions/HostPath\"}},\"title
\":\"Cri\",\"type\":\"object\"},\"Env\":{\"additionalProperties\":false,\"properties
\":{\"ADDITIONAL_ENI_TAGS\":{\"type\":\"string\"},\"AWS_VPC_CNI_NODE_PORT_SUPPORT
\":{\"format\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_ENI_MTU\":{\"format\":
\"integer\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_CONFIGURE_RPFILTER\":{\"format
\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG\":{\"format
\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_EXTERNALSNAT\":{\"format\":
\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_LOGLEVEL\":{\"type\":\"string\"},
\"AWS_VPC_K8S_CNI_LOG_FILE\":{\"type\":\"string\"},\"AWS_VPC_K8S_CNI_RANDOMIZESNAT
\":{\"type\":\"string\"},\"AWS_VPC_K8S_CNI_VETHPREFIX\":{\"type\":\"string\"},
\"AWS_VPC_K8S_PLUGIN_LOG_FILE\":{\"type\":\"string\"},\"AWS_VPC_K8S_PLUGIN_LOG_LEVEL
\":{\"type\":\"string\"},\"DISABLE_INTROSPECTION\":{\"format\":\"boolean\",\"type
\":\"string\"},\"DISABLE_METRICS\":{\"format\":\"boolean\",\"type\":\"string
\"},\"DISABLE_NETWORK_RESOURCE_PROVISIONING\":{\"format\":\"boolean\",\"type
\":\"string\"},\"ENABLE_POD_ENI\":{\"format\":\"boolean\",\"type\":\"string
\"},\"ENABLE_PREFIX_DELEGATION\":{\"format\":\"boolean\",\"type\":\"string\"},
\"WARM_ENI_TARGET\":{\"format\":\"integer\",\"type\":\"string\"},\"WARM_PREFIX_TARGET
\":{\"format\":\"integer\",\"type\":\"string\"}},\"title\":\"Env\",\"type\":
\"object\"},\"HostPath\":{\"additionalProperties\":false,\"properties\":{\"path
\":{\"type\":\"string\"}},\"title\":\"HostPath\",\"type\":\"object\"},\"Limits
\":{\"additionalProperties\":false,\"properties\":{\"cpu\":{\"type\":\"string
\"},\"memory\":{\"type\":\"string\"}},\"title\":\"Limits\",\"type\":\"object\"},
\"Resources\":{\"additionalProperties\":false,\"properties\":{\"limits\":{\"$ref\":
\"#/definitions/Limits\"},\"requests\":{\"$ref\":\"#/definitions/Limits\"}},\"title
\":\"Resources\",\"type\":\"object\"},\"VpcCni\":{\"additionalProperties\":false,
\"properties\":{\"cri\":{\"$ref\":\"#/definitions/Cri\"},\"env\":{\"$ref\":\"#/
definitions/Env\"},\"resources\":{\"$ref\":\"#/definitions/Resources\"}},\"title\":
\"VpcCni\",\"type\":\"object\"}}}"
```

```
}
```

输出是标准的 JSON 架构。

以下是适用于上述架构的 JSON 格式的有效配置值示例。

```
{
  "resources": {
    "limits": {
      "cpu": "100m"
    }
  }
}
```

以下是适用于上述架构的 YAML 格式的有效配置值示例。

```
  resources:
    limits:
      cpu: 100m
```

5. 确定该附加组件是否需要 IAM 权限。如果是，则需要（1）确定是否要使用 EKS 容器组身份或服务账户的 IAM 角色（IRSA），（2）确定要与附加组件一起使用的 IAM 角色的 ARN，以及（3）确定附加组件使用的 Kubernetes 服务账户的名称。有关更多信息，请参阅 the section called “检索 IAM 信息”。
   - 如果附加组件支持，Amazon EKS 建议使用 EKS 容器组身份。这需要在您的集群上安装容器组身份代理。有关将容器组身份与附加组件配合使用的更多信息，请参阅 the section called “IAM 角色”。
   - 如果附加组件或您的集群未针对 EKS 容器组身份进行设置，请使用 IRSA。确认您的集群上已设置 IRSA。
   - 审查 Amazon EKS 附加组件文档，确定该附加组件是否需要 IAM 权限以及关联 Kubernetes 服务账户的名称。
     a. 创建 Amazon EKS 附加组件。将以下命令复制到您的设备。根据需要对该命令进行以下修改，然后运行修改后的命令：
   - 将 *my-cluster* 替换为您的集群的名称。
   - 将 *vpc-cni* 替换为前面步骤的输出中返回的要创建的附加组件名称。
   - 将 *version-number* 替换为前面步骤的输出中返回的要使用的版本。

- 如果要将附加组件安装到自定义的 Kubernetes 命名空间，请添加 `--namespace-config 'namespace=<my-namespace>` 选项。此选项仅适用于 Amazon 和社区附加组件。有关更多信息，请参阅 the section called “附加组件的自定义命名空间”。
- 如果附加组件不需要 IAM 权限，则删除 *<service-account-configuration>*。
- 请执行以下操作之一：
  - 如果附加组件（1）需要 IAM 权限，并且（2）集群使用 EKS 容器组身份，请将 *<service-account-configuration>* 替换为以下容器组身份关联。将 *<service-account-name>* 替换为附加组件使用的服务账户名称。将 *<role-arn>* 替换为 IAM 角色的 ARN。该角色必须拥有 EKS 容器组身份所需的信任策略。

    ```
    --pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
    ```

  - 如果附加组件（1）需要 IAM 权限，并且（2）您的集群使用 IRSA，请将 *<service-account-configuration>* 替换为以下 IRSA 配置。将 *111122223333* 替换为您的账户 ID，并将 *role-name* 替换为您创建的现有 IAM 角色的名称。有关创建角色的说明，请参阅您正在创建的附加组件的文档。有关可用附加组件的列表，请参阅the section called “ Amazon 附加组件”。指定服务账户角色需要您的集群具有 IAM OpenID Connect（OIDC）提供者。要确定您的集群是否具有此提供程序，或者要创建此提供程序，请参阅 the section called “IAM OIDC 提供者”。

    ```
    --service-account-role-arn arn:aws-cn::iam::111122223333:role/role-name
    ```

- 这些示例命令将覆盖附加组件的任何现有自行管理版本（如果有的话）的 `--configuration-values` 选项。将其替换为所需配置值，例如字符串或文件输入。如果不想提供配置值，请删除 `--configuration-values` 选项。如果您不希望 Amazon CLI 覆盖现有的自行管理附加组件的配置，请删除 *`--resolve-conflicts OVERWRITE`* 选项。如果您删除此选项，并且 Amazon EKS 附加组件需要覆盖现有的自行管理附加组件的配置，那么创建 Amazon EKS 附加组件将会失败，并显示一条帮助您解决冲突的错误消息。在指定此选项之前，请确保 Amazon EKS 附加组件不会管理您需要管理的设置，因为这些设置会被此选项覆盖。

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    <service-account-configuration> --configuration-values '{"resources":
{"limits":{"cpu":"100m"}}}' --resolve-conflicts OVERWRITE
```

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    <service-account-configuration> --configuration-values 'file://example.yaml' --
resolve-conflicts OVERWRITE
```

有关可用选项的完整列表，请参阅 Amazon EKS Command Line Reference（《Amazon EKS 命令行参考》）中的 `create-addon` 。如果您创建的附加组件在前面步骤的 `Owner` 列中的值为 `aws-marketplace`，那么创建可能会失败，您可能会收到与以下错误类似的错误消息。

```
{
    "addon": {
        "addonName": "addon-name",
        "clusterName": "my-cluster",
        "status": "CREATE_FAILED",
        "addonVersion": "version",
        "health": {
            "issues": [
                {
                    "code": "AddonSubscriptionNeeded",
                    "message": "You are currently not subscribed to this add-on. To
 subscribe, visit the Amazon Marketplace console, agree to the seller EULA, select
 the pricing type if required, then re-install the add-on"
                }
            ]
        }
    }
}
```

如果您收到的错误与之前输出中的错误类似，请访问前面步骤的输出中的 URL，以订阅附加组件。订阅后，再次运行 `create-addon` 命令。

更新 Amazon EKS 附加组件

在新版本发布时或您将集群更新到新的 Kubernetes 次要版本后，Amazon EKS 不会自动更新附加组件。要更新现有集群的附加组件，必须启动更新。启动更新后，Amazon EKS 会为您更新附加组件。在更新某个附加组件之前，请查看该附加组件的最新文档。有关可用附加组件的列表，请参阅 the section called “ Amazon 附加组件”。如果附加组件需要 IAM 角色，请参阅来自 Amazon 的 Amazon EKS 附加组件中有关特定附加组件的详细信息，以了解有关创建该角色的详细信息。

先决条件

在创建附加组件之前，请完成以下步骤：

- 检查附加组件是否需要 IAM 角色。有关更多信息，请参阅 the section called “Amazon EKS 附加组件”。
- 验证 Amazon EKS 附加组件版本是否与集群兼容。有关更多信息，请参阅 the section called “验证兼容性”。

过程

您可以使用 `eksctl`、Amazon Web Services 管理控制台 或 Amazon CLI 更新 Amazon EKS 附加组件。

更新附加组件（eksctl）

1. 确定集群上目前安装的附加组件和附加组件版本。将 *my-cluster* 替换为您的集群的名称。

```
eksctl get addon --cluster my-cluster
```

   示例输出如下。

```
NAME        VERSION              STATUS  ISSUES  IAMROLE  UPDATE AVAILABLE
coredns     v1.8.7-eksbuild.2    ACTIVE  0
kube-proxy  v1.23.7-eksbuild.1   ACTIVE  0                v1.23.8-eksbuild.2
vpc-cni     v1.10.4-eksbuild.1   ACTIVE  0                v1.12.0-eksbuild.1,v1.11.4-
eksbuild.1,v1.11.3-eksbuild.1,v1.11.2-eksbuild.1,v1.11.0-eksbuild.1
```

   您的输出可能会有所不同，这取决于您的集群上安装的附加组件和版本。您可以看到，在前面的输出示例中，集群上的两个现有附加组件在 UPDATE AVAILABLE 列中有较新的版本可用。

2. 更新附加组件。

   a. 将以下命令复制到您的设备。根据需要对该命令进行以下修改：

   - 将 *my-cluster* 替换为您的集群的名称。
   - 将 *region-code* 替换为您的集群所在的 Amazon 区域。
   - 将 *vpc-cni* 替换为上一步的输出中返回的要更新的附加组件名称。
   - 如果要更新到的版本早于最新版本，请将 *latest* 替换为上一步的输出中返回的适用版本号。一些附加组件有推荐版本。有关更多信息，请参阅您正在更新的附加组件的文档。有关附加组

件的列表，请参阅 the section called “ Amazon 附加组件”。*如果附加组件使用 Kubernetes 服务账户和 IAM 角色，请将 *111122223333* 替换为您的账户 ID，并将 *role-name* 替换为您创建的现有 IAM 角色的名称。有关创建角色的说明，请参阅您正在创建的附加组件的文档。有关可用附加组件的列表，请参阅the section called “ Amazon 附加组件”。指定服务账户角色需要您的集群具有 IAM OpenID Connect（OIDC）提供者。要确定您的集群是否具有此提供程序，或者要创建此提供程序，请参阅 the section called “IAM OIDC 提供者”。

如果附加组件不使用 Kubernetes 服务账户和 IAM 角色，请删除 `serviceAccountRoleARN: arn:aws-cn:iam::111122223333:role/role-name` 行。

- *preserve* 选项保留附加组件的现有值。如果您为附加组件设置设定了自定义值，但未使用此选项，则 Amazon EKS 会使用其默认值覆盖您的值。如果您使用此选项，那么我们建议您在更新生产集群上的附加组件之前，先测试非生产集群上所有更改的字段和值。如果您将该值改为 `overwrite`，则所有设置都将更改为 Amazon EKS 的默认值。如果您为任何设置设定了自定义值，这些值可能会被 Amazon EKS 的默认值覆盖。如果您将该值改为 none，Amazon EKS 不会更改任何设置的值，但更新可能会失败。如果更新失败，您会收到一条帮助您解决冲突的错误消息。

```
cat >update-addon.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code

addons:
- name: vpc-cni
  version: latest
  serviceAccountRoleARN: arn:aws-cn:iam::111122223333:role/role-name
  resolveConflicts: preserve
EOF
```

b. 运行修改后的命令以创建 `update-addon.yaml` 文件。

c. 将配置文件应用到您的集群。

```
eksctl update addon -f update-addon.yaml
```

有关更新附加组件的更多信息，请参阅 `eksctl` 文档中的 Updating addons。

更新附加组件（Amazon 控制台）

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 选择要为其更新附加组件的集群名称。
4. 选择附加组件选项卡。
5. 选择要更新的附加组件
6. 选择编辑。
7. 在配置 ***name of addon*** 页面上，执行以下操作：
   a. 选择要使用的版本。该附加组件可能有推荐版本。有关更多信息，请参阅您正在更新的附加组件的文档。有关可用附加组件的列表，请参阅the section called “ Amazon 附加组件”。
   b. 您可以使用两个选项为附加组件配置角色：EKS 容器组身份 IAM 角色和服务账户 IAM 角色（IRSA）。按照以下相应步骤选择首选选项。如果您选择的所有附加组件在状态下都有需要订阅字样，请选择下一步。如果附件组件的状态下没有需要订阅字样：
      i. 对于服务账户的容器组身份 IAM 角色，您可以使用现有 EKS 容器组身份 IAM 角色，也可以使用创建推荐角色按钮创建。此字段将仅提供具有适当信任策略的选项。如果没有可选择的角色，则表示没有具有匹配信任策略的现有角色。要为所选附加组件配置服务账户的 EKS 容器组身份 IAM 角色，请选择创建推荐角色。角色创建向导将在单独的窗口中打开。向导将自动填充角色信息，如下所示。对于要在其中创建 EKS 容器组身份 IAM 角色的各个附加组件，请如下方式完成 IAM 向导中的步骤。
         - 在选择可信实体步骤中，预先选择 EKS 的 Amazon 服务选项和 EKS – 容器组身份的使用案例，并将自动为附加组件填充相应的信任策略。例如，该角色将使用相应的信任策略创建，该策略包含 pods.eks.amazonaws.com IAM 主体，详见the section called “EKS 容器组身份的优势”。选择下一步。
         - 在添加权限步骤中，系统将为附加组件预先选择角色策略的相应托管策略。例如，对于 Amazon VPC CNI 附加组件，系统将使用托管式策略 `AmazonEKS_CNI_Policy` 创建角色，详见the section called “适用于 Kubernetes 的 Amazon VPC CNI 插件”。选择下一步。
         - 在命名、查看和创建步骤的角色名称中，系统将自动填充附加组件的默认角色名称。例如，对于 Amazon VPC CNI 附加组件，系统将使用 AmazonEKSPodIdentityAmazonVPCCNIRole 名称创建角色。在描述中，默认描述会自动填充附加组件的相应描述。例如，对于 Amazon VPC CNI 附加组件，该角色的创建描述为允许在 Amazon EKS 集群中运行容器组（pod）以访问 Amazon 资源。在信任策略中，查看为附加组件填充的信任策略。选择创建角色。

> **Note**
>
> 保留默认角色名称，使 EKS 能够在新集群中或在向现有集群添加附加组件时为附加组件预先选择角色。您仍然可以覆盖此名称，并且该角色可在您的集群中用于该附加组件，但需要从下拉列表中手动选择该角色。

ii. 对于在状态下没有需要订阅且您想使用 IRSA 配置角色的附加组件，请参阅您正在创建的附加组件的文档，以创建 IAM 策略并将其附加至角色。有关可用附加组件的列表，请参阅the section called “ Amazon 附加组件”。选择 IAM 角色需要您的集群具有 IAM OpenID Connect（OIDC）提供者。要确定您的集群是否具有此提供程序，或者要创建此提供程序，请参阅 the section called “IAM OIDC 提供者”。

c. 展开可选配置设置。

d. 在配置值中，请输入附加组件特有的任何配置信息。有关更多信息，请参阅您正在更新的附加组件的文档。有关附加组件列表，请参阅the section called “ Amazon 附加组件”... 对于冲突解决方法，选择其中一个选项。如果您为附加组件设置设定了自定义值，我们建议选择 Preserve（保留）选项。如果您不选择此选项，Amazon EKS 会使用其默认值覆盖您的值。如果您使用此选项，那么我们建议您在更新生产集群上的附加组件之前，先测试非生产集群上所有更改的字段和值。如果您将该值改为 overwrite，则所有设置都将更改为 Amazon EKS 的默认值。如果您为任何设置设定了自定义值，这些值可能会被 Amazon EKS 的默认值覆盖。如果您将该值改为 none，Amazon EKS 不会更改任何设置的值，但更新可能会失败。如果更新失败，您会收到一条帮助您解决冲突的错误消息。

8. 选择保存更改。

更新附加组件（Amazon CLI）

1. 您需要在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
2. 查看已安装的附加组件列表。将 *`my-cluster`* 替换为您的集群的名称。

```
aws eks list-addons --cluster-name my-cluster
```

示例输出如下。

```
{
    "addons": [
        "coredns",
        "kube-proxy",
        "vpc-cni"
    ]
}
```

3. 查看要更新的附加组件的当前版本。将 *my-cluster* 替换为您的集群名称，将 *vpc-cni* 替换为要更新的附加组件的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 "addon.addonVersion" --output text
```

示例输出如下。

```
v1.10.4-eksbuild.1
```

4. 确定哪些版本的附加组件可用于您的集群版本。将 *1.33* 替换为集群的版本，将 *vpc-cni* 替换为要更新的附加组件的名称。

```
aws eks describe-addon-versions --kubernetes-version 1.33 --addon-name vpc-cni \
    --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
 compatibilities[0].defaultVersion}' --output table
```

示例输出如下。

```
-----------------------------------------
|          DescribeAddonVersions         |
+-----------------+----------------------+
| Defaultversion  |       Version        |
+-----------------+----------------------+
|  False          |  v1.12.0-eksbuild.1  |
|  True           |  v1.11.4-eksbuild.1  |
|  False          |  v1.10.4-eksbuild.1  |
|  False          |  v1.9.3-eksbuild.1   |
+-----------------+----------------------+
```

默认情况下，Defaultversion 列中值为 True 的版本是创建附加组件时使用的版本。

5. 更新您的附加组件。将以下命令复制到您的设备。根据需要对该命令进行以下修改，然后运行修改后的命令。有关此命令的更多信息，请参阅《Amazon EKS 命令行参考》中的 update-addon。
   - 将 *my-cluster* 替换为您的集群的名称。
   - 将 *vpc-cni* 替换为前面步骤的输出中返回的要更新的附加组件名称。
   - 将 *version-number* 替换为上一步的输出中返回的要更新到的版本。一些附加组件有推荐版本。有关更多信息，请参阅您正在更新的附加组件的文档。有关附加组件的列表，请参阅 the section called " Amazon 附加组件"。*如果附加组件使用 Kubernetes 服务账户和 IAM 角色，请将 *111122223333* 替换为您的账户 ID，并将 *role-name* 替换为您创建的现有 IAM 角色的名称。有关创建角色的说明，请参阅您正在创建的附加组件的文档。有关可用附加组件的列表，请参阅the section called " Amazon 附加组件"。指定服务账户角色需要您的集群具有 IAM OpenID Connect（OIDC）提供者。要确定您的集群是否具有此提供程序，或者要创建此提供程序，请参阅 the section called "IAM OIDC 提供者"。

     如果附加组件不使用 Kubernetes 服务账户和 IAM 角色，请删除 `serviceAccountRoleARN: arn:aws-cn:iam::`*111122223333*`:role/`*role-name* 行。
   - `--resolve-conflicts PRESERVE` 选项保留附加组件的现有值。如果您为附加组件设置设定了自定义值，但未使用此选项，则 Amazon EKS 会使用其默认值覆盖您的值。如果您使用此选项，那么我们建议您在更新生产集群上的附加组件之前，先测试非生产集群上所有更改的字段和值。如果您将该值改为 OVERWRITE，则所有设置都将更改为 Amazon EKS 的默认值。如果您为任何设置设定了自定义值，这些值可能会被 Amazon EKS 的默认值覆盖。如果您将该值改为 NONE，Amazon EKS 不会更改任何设置的值，但更新可能会失败。如果更新失败，您会收到一条帮助您解决冲突的错误消息。
   - 如果要删除所有自定义配置，请使用 *--configuration-values '{}'* 选项执行更新。这会将所有自定义配置设置回默认值。如果不想更改自定义配置，请勿提供 *--configuration-values* 标志。如果想要调整自定义配置，请将 *{}* 替换为新参数。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    --service-account-role-arn arn:aws-cn:iam::111122223333:role/role-name --
configuration-values '{}' --resolve-conflicts PRESERVE
```

6. 检查更新的状态。将 *my-cluster* 替换为您的集群名称，将*vpc-cni* 替换为您正在更新的附加组件的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

示例输出如下。

```
{
    "addon": {
        "addonName": "vpc-cni",
        "clusterName": "my-cluster",
        "status": "UPDATING",
    }
}
```

当状态为 ACTIVE 时更新完成。

验证 Amazon EKS 附加组件版本与集群的兼容性

在创建 Amazon EKS 附加组件之前，需要验证 Amazon EKS 附加组件版本与集群的兼容性。

使用 describe-addon-versions API 列出 EKS 附加组件的可用版本，以及每个附加组件版本支持的具体 Kubernetes 版本。

1. 验证 Amazon CLI 是否已安装并在使用 `aws sts get-caller-identity`。如果此命令不起作用，请了解如何开始使用 Amazon CLI。
2. 确定要检索其版本兼容性信息的附加组件的名称，例如 `amazon-cloudwatch-observability`。
3. 确定集群的 Kubernetes 版本，例如 `1.33`。
4. 使用 Amazon CLI 检索与集群的 Kubernetes 版本兼容的插件版本。

```
aws eks describe-addon-versions --addon-name amazon-cloudwatch-observability --
kubernetes-version 1.33
```

示例输出如下。

```
{
    "addons": [
        {
            "addonName": "amazon-cloudwatch-observability",
            "type": "observability",
            "addonVersions": [
                {
```

```
                    "addonVersion": "vX.X.X-eksbuild.X",
                    "architecture": [
                        "amd64",
                        "arm64"
                    ],
                    "computeTypes": [
                        "ec2",
                        "auto",
                        "hybrid"
                    ],
                    "compatibilities": [
                        {
                            "clusterVersion": "1.33",
                            "platformVersions": [
                                "*"
                            ],
                            "defaultVersion": true
                        }
                    ],
                }
            ]
        }
    ]
}
```

此输出显示插件版本 `vX.X.X-eksbuild.X` 与 Kubernetes 集群版本 `1.33` 兼容。

附加组件与计算类型的兼容性

`describe-addon-versions` 输出中的 `computeTypes` 字段指示附加组件与 EKS 自动模式托管式节点或混合节点的兼容性。标记 `auto` 的附加组件支持 EKS 自动模式基于云的 Amazon 托管式基础设施，而标记 `hybrid` 的附加组件可以在连接到 EKS 云控制面板的本地节点上运行。

有关更多信息，请参阅 the section called “Amazon EKS 自动模式的注意事项”。

从集群中移除 Amazon EKS 附加组件

您可以使用 `eksctl`、Amazon Web Services 管理控制台或 Amazon CLI 从集群中移除 Amazon EKS 附加组件。

从集群中移除 Amazon EKS 附加组件时：

- 附加组件提供的功能不停机。
- 如果您使用的是服务账户的 IAM 角色（IRSA），并且附加组件有与其关联的 IAM 角色，则不会移除该 IAM 角色。
- 如果您使用的是容器组身份，则该附加组件拥有的所有容器组身份关联都将被移除。如果将 `--preserve` 选项指定为 Amazon CLI，则将会保留关联。
- Amazon EKS 停止管理附加组件的设置。
- 控制台停止通知您新版本何时可用。
- 您无法使用任何 Amazon 工具或 API 更新附加组件。
- 您可以选择将附加软件留在集群上，以便您自行管理，您也可以从集群中删除附加软件。只有当集群上没有资源依赖于附加组件提供的功能时，您才能从集群中删除附加软件。

先决条件

在创建附加组件之前，请完成以下步骤：

- 现有 Amazon EKS 集群。要部署一个角色，请参阅开始使用。
- 检查附加组件是否需要 IAM 角色。有关更多信息，请参阅
- 您的设备或 Amazon CloudShell 上安装 `0.215.0` 版或更高版本的 `eksctl` 命令行工具。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的 Installation。

过程

您可以通过两种方式来移除 Amazon EKS 附加组件：

- 将附加组件保留在您的集群上 - 此方法将移除 Amazon EKS 对任何设置的管理。还移除 Amazon EKS 通知您更新以及在您启动更新后自动更新 Amazon EKS 附加组件的功能。但是，此方法会保留集群上的附加组件软件。此选项会将附加组件作为自行管理安装使用，而不是作为 Amazon EKS 附加组件使用。使用此方法，附加组件不停机。
- 从集群中完全删除附加组件 - 我们建议只有当集群中没有资源依赖于附加组件时，才从集群移除 Amazon EKS 附加组件。

您可以使用 `eksctl`、Amazon Web Services 管理控制台或 Amazon CLI 移除 Amazon EKS 附加组件。

移除附加组件（eksctl）

1. 确定集群上目前安装的附加组件。将 *my-cluster* 替换为您的集群的名称。

   ```
   eksctl get addon --cluster my-cluster
   ```

   示例输出如下。

   ```
   NAME        VERSION              STATUS  ISSUES  IAMROLE  UPDATE AVAILABLE
   coredns     v1.8.7-eksbuild.2    ACTIVE  0
   kube-proxy  v1.23.7-eksbuild.1   ACTIVE  0
   vpc-cni     v1.10.4-eksbuild.1   ACTIVE  0
   [...]
   ```

   您的输出可能会有所不同，这取决于您的集群上安装的附加组件和版本。
2. 移除附加组件。将 *my-cluster* 替换为您的集群名称，将 *name-of-add-on* 替换为上一步的输出中返回的要删除的附加组件名称。如果您移除了 *--preserve* 选项，则除 Amazon EKS 不再管理该附加组件外，该附加软件也会从集群中删除。

   ```
   eksctl delete addon --cluster my-cluster --name name-of-addon --preserve
   ```

   有关移除附加组件的更多信息，请参阅 eksctl 文档中的 Deleting addons。

移除附加组件（Amazon 控制台）

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 选择要从中移除 Amazon EKS 附加组件的集群名称。
4. 选择附加组件选项卡。
5. 选择要移除的附加组件。
6. 选择移除。
7. 在移除：***name of addon*** 确认对话框中，执行以下操作：
   a. 如果希望 Amazon EKS 停止管理附加组件设置，请选择在集群上保留。如果要在集群上保留附加组件软件，请执行此操作。这样您可以自行管理附加组件的所有设置。
   b. 输入附加组件的名称。
   c. 选择移除。

移除附加组件（Amazon CLI）

1. 您需要在设备或 Amazon CloudShell 上安装 `0.215.0` 版或更高版本的 `eksctl` 命令行工具。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的 Installation。
2. 查看已安装的附加组件列表。将 *`my-cluster`* 替换为您的集群的名称。

```
aws eks list-addons --cluster-name my-cluster
```

示例输出如下。

```
{
    "addons": [
        "coredns",
        "kube-proxy",
        "vpc-cni",
        "name-of-addon"
    ]
}
```

3. 移除已安装的附加组件。将 *`my-cluster`* 替换为您的集群名称，将 *`name-of-add-on`* 替换为要删除的附加组件的名称。移除 *`--preserve`* 会从集群中移除附加组件软件。

```
aws eks delete-addon --cluster-name my-cluster --addon-name name-of-addon --preserve
```

简短的示例输出如下。

```
{
    "addon": {
        "addonName": "name-of-add-on",
        "clusterName": "my-cluster",
        "status": "DELETING",
    }
}
```

4. 检查移除操作的状态。将 *`my-cluster`* 替换为您的集群名称，将 *`name-of-addon`* 替换为您正在删除的附加组件的名称。

```
aws eks describe-addon --cluster-name my-cluster --addon-name name-of-addon
```

移除附加组件后，示例输出将如下所示。

```
An error occurred (ResourceNotFoundException) when calling the DescribeAddon
 operation: No addon: name-of-addon found in cluster: my-cluster
```

Amazon EKS 附加组件的 IAM 角色

某些 Amazon EKS 附加组件需要 IAM 角色和权限才能调用 Amazon API。例如，Amazon VPC CNI 附加组件会调用某些 Amazon API 来配置您账户中的网络资源。需要使用 IAM 对这些附加组件授予权限。更具体地说，运行附加组件的容器组（Pod）的服务账户需要关联到具有具体 IAM 策略的 IAM 角色。

向集群工作负载授予 Amazon 权限的建议方法是使用 Amazon EKS 功能容器组身份。您可以使用容器组身份关联将附加组件的服务账户映射到 IAM 角色。如果某个容器组使用具有关联的服务账户，Amazon EKS 会在容器组的容器中设置环境变量。环境变量配置 Amazon SDK（包括 Amazon CLI）使用 EKS 容器组身份凭证。有关更多信息，请参阅 the section called “容器组身份”

Amazon EKS 附加组件可以帮助管理与该附加组件对应的容器组身份关联的生命周期。例如，您可以在单个 API 调用中创建或更新 Amazon EKS 附加组件和必要的容器组身份关联。Amazon EKS 还提供用于检索建议 IAM 策略的 API。

1. 确认已在您的集群上设置 Amazon EKS 容器组身份代理。
2. 使用 `describe-addon-versions` Amazon CLI 操作确定要安装的附加组件是否需要 IAM 权限。如果 `requiresIamPermissions` 标志是 `true`，则应使用 `describe-addon-configurations` 操作来确定附加组件所需的权限。响应中将包含建议的托管 IAM 策略列表。
3. 使用 CLI `describe-addon-configuration` 操作检索 Kubernetes 服务账户的名称和 IAM 策略。根据您的安全要求评估建议策略的范围。
4. 使用建议的权限策略和容器组身份所需的信任策略创建 IAM 角色。有关更多信息，请参阅 the section called “创建容器组身份关联（Amazon 控制台）”。
5. 使用该 CLI 创建或更新 Amazon EKS 附加组件。指定至少一个容器组身份关联。容器组身份关联是 Kubernetes 服务账户名称与 IAM 角色 ARN。
   - 使用附加组件 API 创建的容器组身份关联归相应的附加组件所有。如果您删除该附加组件，则容器组身份关联也会被删除。在使用 Amazon CLI 或 API 删除附加组件时，您可以通过使用 `preserve` 选项来防止这种级联删除。如有必要，您也可以直接更新或删除容器组身份关联。附加组件不能代入现有容器组身份关联的所有权。您必须删除现有关联，然后使用附加组件创建或更新操作重新创建该关联。

- Amazon EKS 建议使用容器组身份关联来管理附加组件的 IAM 权限。仍然支持之前的方法（服务账户的 IAM 角色（IRSA））。您可以为附加组件同时指定 IRSA `serviceAccountRoleArn` 和容器组身份关联。如果在集群上安装了 EKS 容器组身份代理，则 `serviceAccountRoleArn` 将被忽略，并且 EKS 将使用提供的容器组身份关联。如果未启用容器组身份，则将使用 `serviceAccountRoleArn`。
- 如果您更新现有附加组件的容器组身份关联，Amazon EKS 会启动附加组件容器组（Pod）的滚动重新启动。

检索 Amazon EKS 附加组件的 IAM 信息

在创建附加组件之前，请使用 Amazon CLI 确定如下因素：

- 该附加组件是否需要 IAM 权限
- 建议使用的 IAM 策略

过程

1. 确定要安装的附加组件的名称以及集群的 Kubernetes 版本。有关附加组件的更多信息，请参阅 the section called “Amazon EKS 附加组件”。
2. 使用 Amazon CLI 确定该附加组件是否需要 IAM 权限。

```
aws eks describe-addon-versions \
--addon-name <addon-name> \
--kubernetes-version <kubernetes-version>
```

例如：

```
aws eks describe-addon-versions \
--addon-name aws-ebs-csi-driver \
--kubernetes-version 1.30
```

审查以下示例输出。请注意，`requiresIamPermissions` 的值为 `true`，即默认的附加组件版本。在检索建议的 IAM 策略时，您需要指定附加组件版本。

```
{
    "addons": [
        {
```

```
            "addonName": "aws-ebs-csi-driver",
            "type": "storage",
            "addonVersions": [
                {
                    "addonVersion": "v1.31.0-eksbuild.1",
                    "architecture": [
                        "amd64",
                        "arm64"
                    ],
                    "compatibilities": [
                        {
                            "clusterVersion": "1.30",
                            "platformVersions": [
                                "*"
                            ],
                            "defaultVersion": true
                        }
                    ],
                    "requiresConfiguration": false,
                    "requiresIamPermissions": true
                },
[...]
```

3. 如果附加组件需要 IAM 权限，请使用 Amazon CLI 检索建议的 IAM 策略。

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name <addon-name> \
--addon-version <addon-version>
```

例如：

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name aws-ebs-csi-driver \
--addon-version v1.31.0-eksbuild.1
```

审查以下输出。记下 `recommendedManagedPolicies`。

```
[
    {
        "serviceAccount": "ebs-csi-controller-sa",
```

```
        "recommendedManagedPolicies": [
            "arn:aws-cn:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy"
        ]
    }
]
```

4. 创建 IAM 角色并附加托管策略。或者，审查托管策略并根据需要缩小权限范围。有关更多信息，请参阅 the section called “创建容器组身份关联（Amazon 控制台）”。

容器组身份支持参考

下表显示了某些 Amazon EKS 附加组件是否支持 EKS 容器组身份。

附加组件名称	容器组身份支持	所需最低版本
Amazon EBS CSI 驱动程序	是	v1.26.0-eksbuild.1
Amazon VPC CNI	是	v1.15.5-eksbuild.1
Amazon EFS CSI 驱动程序	是	v2.0.5-eksbuild.1
适用于 OpenTelemetry 的 Amazon Distro	是	v0.94.1-eksbuild.1
适用于 Amazon S3 的 Mountpoint CSI 驱动程序	否	不适用
Amazon CloudWatch 可观测性代理	是	v3.1.0-eksbuild.1

此表最近于 2024 年 10 月 28 日更新。

使用容器组身份将 IAM 角色分配给 Amazon EKS 附加组件

某些 Amazon EKS 附加组件需要 IAM 角色和权限。在添加或更新 Amazon EKS 附加组件以使用容器组身份关联之前，请验证要使用的角色和策略。有关更多信息，请参阅 the section called “检索 IAM 信息”。

1. 确定：

   - `cluster-name` – 要在其上安装附加组件的集群名称。

- addon-name – 要安装的附加组件名称。
- service-account-name – 该附加组件使用的 Kubernetes 服务账户的名称。
- iam-role-arn – 对附加组件拥有足够权限的 IAM 角色的 ARN。该角色必须具有 EKS 容器组身份所需的信任策略。有关更多信息，请参阅the section called “创建容器组身份关联（Amazon 控制台）”。

2. 使用 Amazon CLI 更新附加组件。您还可以在创建附加组件时使用相同的 --pod-identity-assocations 语法指定容器组身份关联。请注意，如果您在更新附加组件时指定容器组身份关联，则所有之前的容器组身份关联都将被覆盖。

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
```

例如：

```
aws eks update-addon --cluster-name mycluster \
--addon-name aws-ebs-csi-driver \
--pod-identity-associations 'serviceAccount=ebs-csi-controller-sa,roleArn=arn:aws-cn:iam::123456789012:role/StorageDriver'
```

3. 验证是否已创建容器组身份关联：

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

如果成功，您应看到与以下内容类似的输出。记下 EKS 附加组件的 OwnerARN。

```
{
    "associations": [
        {
            "clusterName": "mycluster",
            "namespace": "kube-system",
            "serviceAccount": "ebs-csi-controller-sa",
            "associationArn": "arn:aws-cn:eks:us-west-2:123456789012:podidentityassociation/mycluster/a-4wvljrezsukshq1bv",
            "associationId": "a-4wvljrezsukshq1bv",
            "ownerArn": "arn:aws-cn:eks:us-west-2:123456789012:addon/mycluster/aws-ebs-csi-driver/9cc7ce8c-2e15-b0a7-f311-426691cd8546"
        }
```

```
    ]
}
```

从 Amazon EKS 附加组件中移除容器组身份关联

从 Amazon EKS 附加组件中移除容器组身份关联。

1. 确定：
   - `cluster-name` – 要在其上安装附加组件的 EKS 集群的名称。
   - `addon-name` – 要安装的 Amazon EKS 附加组件的名称。
2. 更新附加组件以指定容器组身份关联的空数组。

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations "[]"
```

排查 EKS 附加组件的容器组身份问题

如果您的附加组件在尝试 Amazon API、SDK 或 CLI 操作时遇到错误，请确认以下内容：

- 容器组身份代理已安装在您的集群中。
  - 有关如何安装容器组身份代理的信息，请参阅the section called “设置代理”。
- 此附加组件具有有效的容器组身份关联。
  - 使用 Amazon CLI 检索此附加组件所用服务账户名称的关联。

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

- 此 IAM 角色具有容器组身份所需的信任策略。
  - 使用 Amazon CLI 检索附加组件的信任策略。

```
aws iam get-role --role-name <role-name> --query Role.AssumeRolePolicyDocument
```

- 此 IAM 角色具有此附加组件所需的权限。
  - 使用 Amazon CloudTrail 查看 `AccessDenied` 或 `UnauthorizedOperation` 事件。
- 容器组身份关联中的服务账户名称与附加组件使用的服务账户名称相匹配。
  - 有关可用附加组件的信息，请参阅the section called “ Amazon 附加组件”。

- 检查名为 pod-identity-webhook 的 MutatingWebhookConfiguration 配置
  - Webhook 的 admissionReviewVersions 必须为 v1beta1，如为 v1 则不能正常工作。

确定可以为 Amazon EKS 附加组件自定义的字段

Amazon EKS 使用标准的最佳实践配置，为您的集群安装附加组件。有关向集群中添加 Amazon EKS 附加组件的更多信息，请参阅 the section called “Amazon EKS 附加组件”。

您若想要启用高级功能，则建议自定义 Amazon EKS 附加组件的配置。Amazon EKS 使用 Kubernetes 服务器端应用功能来启用 Amazon EKS 对附加组件的管理，而不会覆盖您对并非由 Amazon EKS 管理的设置的配置。有关更多信息，请参阅 Kubernetes 文档中的服务器端应用。为此，Amazon EKS 为其安装的每个附加组件管理最小的字段组。您可以修改并非由 Amazon EKS 或其他 Kubernetes 控制面板进程（例如 kube-controller-manager）管理的所有字段，这样做不会导致任何问题。

> **Important**
>
> 修改由 Amazon EKS 管理的字段会阻止 Amazon EKS 管理附加组件，并可能导致在更新附加组件时覆盖您所做的更改。

字段管理语法

当您查看某个 Kubernetes 对象的详细信息时，输出中会同时返回托管和非托管字段。托管字段可以是以下任一类型：

- 完全托管：该字段的所有密钥都由 Amazon EKS 管理。修改任何值都会导致冲突。
- 部分托管：该字段的部分密钥将由 Amazon EKS 管理。只有对 Amazon EKS 明确管理的密钥进行修改才会导致冲突。

这两种类型的字段都标有 manager: eks。

每个键要么是表示字段本身的 .（始终映射到空集），要么是表示子字段或项目的字符串。字段管理的输出由以下类型的声明组成：

- f:*name* ，其中 *name* 是列表中字段的名称。
- k:*keys* ，其中 *keys* 是列表项字段的映射。

- `v:`*value* ，其中 *value* 是列表项的确切 JSON 格式化值。
- `i:`*index* ，其中 *index* 是列表中某个项的位置。

CoreDNS 附加组件的以下输出部分说明了上面的声明：

- 完全托管字段：如果托管字段指定了 `f:`（字段），但没有 `k:`（键），则整个字段都是托管的。修改此字段中的任何值都会导致冲突。

  在以下输出中，您可以看到名为 `coredns` 的容器由 `eks` 管理。`args`、`image` 和 `imagePullPolicy` 子字段也由 `eks` 管理。修改此字段中的任何值都会导致冲突。

```
[...]
f:containers:
  k:{"name":"coredns"}:
  .: {}
  f:args: {}
  f:image: {}
  f:imagePullPolicy: {}
[...]
manager: eks
[...]
```

- 部分托管字段：如果托管键具有指定的值，则为该字段管理所声明的键。修改指定的键会导致冲突。

  在以下输出中，您可以看到 `eks` 管理着包含 `name` 键的 `config-volume` 和 `tmp` 卷集。

```
[...]
f:volumes:
  k:{"name":"config-volume"}:
    .: {}
    f:configMap:
      f:items: {}
      f:name: {}
    f:name: {}
  k:{"name":"tmp"}:
    .: {}
    f:name: {}
[...]
manager: eks
[...]
```

- 向部分托管的字段添加键：如果只管理一个特定的键值，则可以安全地向字段添加其他键（如实际参数），而不会导致冲突。如果您添加了其它键，请先确保该字段不是托管字段。添加或修改任何托管值都会导致冲突。

  在以下输出中，您可以看到 name 键和 name 字段都是托管的。添加或修改任何容器名称都会导致与此托管键发生冲突。

```
[...]
f:containers:
  k:{"name":"coredns"}:
[...]
    f:name: {}
[...]
manager: eks
[...]
```

过程

您可以使用 kubectl 查看对于任何 Amazon EKS 附加组件，有哪些字段由 Amazon EKS 管理。

您可以修改并非由 Amazon EKS 或其他 Kubernetes 控制面板进程（例如 kube-controller-manager）管理的所有字段，这样做不会导致任何问题。

1. 确定要检查的附加组件。要查看部署到集群的所有 deployments 和 DaemonSets，请参阅the section called “访问集群资源”。
2. 通过运行以下命令查看附加组件的托管字段：

```
kubectl get type/add-on-name -n add-on-namespace -o yaml
```

   例如，您可以通过以下命令查看 CoreDNS 附加组件的托管字段。

```
kubectl get deployment/coredns -n kube-system -o yaml
```

   字段管理列在所返回的输出中的以下部分中。

```
[...]
managedFields:
  - apiVersion: apps/v1
    fieldsType: FieldsV1
```

```
    fieldsV1:
[...]
```

> **Note**
>
> 如果在输出中没有看到 `managedFields`，将 `--show-managed-fields` 添加到命令中，然后再次运行。您使用的 `kubectl` 版本决定默认情况下是否返回托管字段。

后续步骤

为附加组件自定义非 Amazon 拥有的字段。

在部署期间验证容器映像签名

如果您在部署时使用 Amazon Signer 并且想要验证已签名的容器映像，则可以使用以下解决方案之一：

- Gatekeeper 和 Ratify – 将 Gatekeeeper 用作准入控制器，并将配置了 Amazon Signer 插件的 Ratify 作为用于验证签名的 Webhook。
- Kyverno – 配置有用于验证签名的 Amazon Signer 插件的 Kubernetes 策略引擎。

> **Note**
>
> 在验证容器镜像签名之前，请根据所选准入控制器的要求配置 Notation 信任存储和信任策略。

EKS 功能

> **Tip**
>
> 入门：创建 ACK 功能 | 创建 Argo CD 功能 | 创建 kro 功能

Amazon EKS 功能是一组分层的完全托管式集群功能，可帮助加快开发人员的速度并减轻使用 Kubernetes 构建和扩展的复杂性。EKS 功能是 Kubernetes 原生功能，用于声明式持续部署、Amazon 资源管理以及 Kubernetes 资源编写和编排，所有这些功能都由 Amazon 完全托管。借助 EKS 功能，您可以更加专注于构建和扩展工作负载，从而将这些基础平台服务的运营负担转移至 Amazon。这些功能在 EKS 中运行，而不是在您的集群中运行，因此无需在 Worker 节点上安装、维护和扩展关键平台组件。

首先，您可以在新的或现有的 EKS 集群上创建一个或多个 EKS 功能。为此，您可以使用 Amazon CLI、Amazon Web Services 管理控制台、EKS API、eksctl 或您偏好的基础设施即代码工具。虽然 EKS 功能旨在协同工作，但它们是独立的云资源，您可以根据自己的使用案例和要求进行选择。

EKS 支持的所有 Kubernetes 版本都支持 EKS 功能。

> **Note**
>
> 所有提供 Amazon EKS 的 Amazon 商业区域均提供 EKS 功能。有关支持的区域列表，请参阅《Amazon 一般参考》中的 Amazon EKS 端点和配额。

可用的功能

Amazon Controllers for Kubernetes（ACK）

ACK 支持使用 Kubernetes API 管理 Amazon 资源，允许您使用 Kubernetes 自定义资源创建和管理 S3 存储桶、RDS 数据库、IAM 角色和其他 Amazon 资源。ACK 会持续调整您的所需状态与 Amazon 中的实际状态，纠正随时间推移而出现的任何漂移，以保持系统运行状况良好并按照指定配置资源。您能够使用统一的工具与工作流程，将 Amazon 资源和 Kubernetes 工作负载纳入同一管理体系，该工具支持包括 S3、RDS、DynamoDB 及 Lambda 在内的 50 余项 Amazon 服务。ACK 支持跨账户和跨区域资源管理，支持复杂的多账户、多集群系统管理架构。ACK 支持只读资源和只读采用，便于从其他基础设施即代码工具迁移到基于 Kubernetes 的系统。

了解有关 ACK 的更多信息 →

Argo CD

Argo CD 将 Git 存储库作为工作负载和系统状态的事实来源，为您的应用程序实施基于 GitOps 的持续部署。Argo CD 会自动将应用程序资源从 Git 存储库同步到您的集群，从而检测和修复漂移，确保您部署的应用程序与预期状态保持一致。您可以通过单个 Argo CD 实例跨多个集群部署和管理应用程序，当 Git 存储库中提交变更时，该工具会自动触发部署流程。同时使用 Argo CD 和 ACK 可以提供基础的 GitOps 系统，简化工作负载依赖关系管理，并支持包括大规模管理集群和基础设施在内的整个系统设计。Argo CD 与 Amazon Identity Center 集成，可进行身份验证和授权，并提供托管 Argo 用户界面，用于可视化应用程序运行状况和部署状态。

了解有关 Argo CD 的更多信息 →

kro（Kube Resource Orchestrator）

kro 支持您创建自定义 Kubernetes API，可将多种资源组合为更上层的抽象资源，方便平台团队针对常见的资源组合云基础设施组件，定义可复用的模式。使用 kro，您可以将 Kubernetes 和 Amazon 资源组合成统一的抽象资源，使用简单的语法实现动态配置和条件逻辑。kro 使平台团队能够提供带有适当护栏的自助服务功能，允许开发人员在维护组织标准和最佳实践的同时，使用简单、专门构建的 API 预置复杂的基础设施。kro 资源只是 Kubernetes 资源，在可以存储在 Git 中的 Kubernetes 清单中指定，或者推送到 Amazon ECR 等兼容 OCI 的注册表，用于在组织范围广泛分布。

了解有关 kro 的更多信息 →

EKS 功能的优势

EKS 功能完全由 Amazon 管理，无需安装、维护和扩展基础集群服务。Amazon 可处理安全修补、更新和运营管理，让您的团队腾出时间专注于使用 Amazon 进行构建，而无需顾虑集群操作。与消耗集群资源的传统 Kubernetes 附加组件不同，这些功能在 EKS 中运行，而不是在 Worker 节点上运行。这可以为工作负载释放集群容量和资源，同时最大限度地减少管理集群内控制器和其他平台组件的运营负担。

借助 EKS 功能，您可以使用原生 Kubernetes API 和诸如 `kubectl` 等工具管理部署、Amazon 资源、自定义 Kubernetes 资源和组合。所有功能都在集群环境中运行，可自动检测和纠正应用程序和云基础设施资源中的配置漂移。您可以从一个控制点跨多个集群、Amazon 账户和区域部署和管理资源，从而简化复杂的分布式环境中的操作。

EKS 功能专为 GitOps 工作流程而设计，提供声明式、版本控制的基础设施和应用程序管理。这些变更通过 Git 在系统中传递，从而提供审计跟踪记录、回滚功能以及与您现有开发流程相整合的协作工作流

程。这种 Kubernetes 原生方法意味着您无需使用多种工具或管理集群外部的基础设施即代码系统，并且只有一个事实来源供参考。在版本控制的 Kubernetes 声明式配置中定义的预期状态将在环境中持续强制执行。

定价

使用 EKS 功能无需预付费用承诺，也没有最低费用。您需要为每项功能资源在 Amazon EKS 集群上处于活动状态的每小时支付费用。由 EKS 功能管理的特定 Kubernetes 资源也按小时费率计费。

有关当前定价信息，请参阅 Amazon EKS 定价页面。

> **Tip**
>
> 您可以使用 Amazon Cost Explorer 成本管理服务以及成本和使用情况报告，分别追踪容量成本与其他 EKS 费用。您可以使用集群名称、功能类型和其他详细信息来标记您的功能，用于成本分配。

EKS 功能的工作原理

每项功能都是您在 EKS 集群上创建的 Amazon 资源。创建后，该功能将在 EKS 中运行并完全由 Amazon 管理。

> **Note**
>
> 您可以针对给定集群创建每种类型（Argo CD、ACK 和 kro）的功能资源。您不能在同一个集群上创建相同类型的多个功能资源。

您可以使用标准 Kubernetes API 和工具与集群中的功能进行交互：

- 使用 `kubectl` 应用 Kubernetes 自定义资源
- 使用 Git 存储库作为 GitOps 工作流程的事实来源

某些功能还支持其他工具。例如：

- 使用 Argo CD CLI 在 Argo CD 功能中配置和管理存储库和集群

- 使用 Argo CD 用户界面，对由 Argo CD 功能管理的应用程序进行可视化和管理

各功能旨在协同工作，但它们是独立的，且完全可由您自行选择加入。您可以根据需要启用一项、两项或全部三项功能，并随着需求的变化更新配置。

所有 EKS 计算类型均支持与 EKS 功能配合使用。有关更多信息，请参阅 管理计算。

有关安全配置和 IAM 角色的详细信息，请参阅 the section called “EKS 功能的注意事项”。有关多集群架构模式的信息，请参阅 the section called “注意事项”。

常见使用案例

适用于应用程序和基础设施的 GitOps

使用 Argo CD 部署应用程序和操作组件，使用 ACK 管理集群配置和预置基础设施，两者均从 Git 存储库中获得。整个堆栈（应用程序、数据库、存储和网络）都会被定义为代码并自动部署。

示例：开发团队将更改推送到 Git。Argo CD 会部署更新的应用程序，ACK 会使用正确的配置预置新的 RDS 数据库。所有更改均可审计、可逆且可在不同环境中保持一致。

具有自助服务能力的平台工程

使用 kro 创建用于构建 ACK 和 Kubernetes 资源的自定义 API。平台团队使用护栏定义经批准的模式。应用程序团队使用简单的高级别 API 预置完整的堆栈。

示例：平台团队创建了“WebApplication”API，用于配置部署、服务、入口和 S3 存储桶。开发人员无需了解底层复杂性或 Amazon 权限即可使用此 API。

多集群应用程序管理

使用 Argo CD，在不同地区或账户的多个 EKS 集群中部署应用程序。使用一致的策略和工作流程，使用单个 Argo CD 实例管理所有部署。

示例：将同一个应用程序部署到多个区域中的开发、暂存和生产集群。Argo CD 可确保每个环境与其相应的 Git 分支保持同步。

多集群管理

使用 ACK 定义和预置 EKS 集群，使用 kro 根据组织标准自定义集群配置，使用 Argo CD 管理集群生命周期和配置。这能实现从创建到持续运营的端到端集群管理。

示例：使用 ACK 和 kro 定义 EKS 集群，以预置和管理集群基础设施，同时定义网络、安全策略、附加组件和其他配置的组织标准。使用 Argo CD，利用一致标准和自动化生命周期管理，跨实例集创建并持续管理集群、配置和 Kubernetes 版本更新。

迁移与现代化

借助原生云资源预置和 GitOps 工作流程，简化向 EKS 的迁移。在不重新创建现有 Amazon 资源的情况下通过 ACK 采用这些资源，并使用 Argo CD 从 Git 实施工作负载部署。

示例：从 EC2 迁移到 EKS 的团队使用 ACK 采用其现有的 RDS 数据库和 S3 存储桶，然后使用 Argo CD 从 Git 部署容器化应用程序。迁移路径很明确，操作从第一天起就实现标准化。

账户和区域引导

同时使用 Argo CD 和 ACK，实现跨账户和地区自动化基础设施部署。在 Git 中将基础设施定义为代码，让各类功能处理部署和管理事务。

示例：平台团队维护负责定义标准账户配置（VPC、IAM 角色、RDS 实例和监控堆栈）的 Git 存储库。Argo CD 会自动将这些配置部署到新账户和区域，从而确保一致性，并将手动设置时间从几天缩短到几分钟。

使用功能资源

本主题介绍管理所有功能类型的功能资源的常见操作。

EKS 功能资源

EKS 功能是在 Amazon EKS 集群上启用托管功能的 Amazon 资源。各项功能在 EKS 中运行，无需在 Worker 节点上安装和维护控制器和其他操作组件。这些功能是为特定的 EKS 集群创建，并且在整个生命周期内都与该集群保持关联。

每种能力资源都有：

- 集群中的唯一名称
- 功能类型（ACK、ARGOCD 或 KRO）
- Amazon 资源名称（ARN），同时指定名称和类型
- 功能 IAM 角色
- 表示当前状态的状态

- 配置，包括通用配置和特定于功能类型的配置

了解功能状态

功能资源的状态表示其当前状态。您可以在 EKS 控制台或使用 Amazon CLI 查看功能状态和运行状况。

控制台：

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称。
3. 选择功能选项卡，查看所有功能的状态。
4. 有关详细的运行状况信息，请选择可观测性选项卡，然后依次选择监控集群、功能选项卡。

AmazonCLI：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name
```

功能状态

CREATING：正在设置功能。您可以离开控制台，功能将继续在后台创建。

ACTIVE：功能正在运行中并可供使用。如果资源未按预期运行，请检查资源状态和 IAM 权限。请参阅 the section called “问题排查功能” 查看相关指南。

UPDATING：正在应用配置更改。等待状态恢复为 ACTIVE。

删除：正在从集群中删除功能。

CREATE_FAILED：设置时遇到了错误。常见原因包括：

- IAM 角色信任策略不正确或缺失
- IAM 角色不存在或无法访问
- 集群访问权限问题
- 无效的配置参数

有关具体错误的详细信息，请查看功能运行状况部分。

UPDATE_FAILED：配置更新失败。有关详细信息，请查看功能运行状况部分并验证 IAM 权限。

> **Tip**
>
> 有关详细的问题排查指导，请参阅：
>
> - the section called “问题排查功能”：一般功能问题排查
> - the section called “问题排查”：ACK 特有的问题
> - the section called “问题排查”：Argo CD 特有的问题
> - the section called “问题排查”：kro 特有的问题

创建功能

要在集群上创建功能，请参阅以下主题：

- the section called “创建 ACK 功能”：创建 ACK 功能以使用 Kubernetes API 管理 Amazon 资源
- the section called “创建 Argo CD 功能”：创建用于 GitOps 持续交付的 Argo CD 功能
- the section called “创建 kro 功能”：创建用于资源组合与编排的 kro 功能

列出功能

您可以列出集群上的所有功能资源。

控制台

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称，打开集群详细信息页面。
3. 选择功能选项卡。
4. 在托管功能下查看功能资源。

Amazon CLI

使用 `list-capabilities` 命令查看集群上的所有功能。将 *`region-code`* 替换为您的集群所在的 Amazon 区域，并将 *`my-cluster`* 替换为您的集群的名称。

```
aws eks list-capabilities \
  --region region-code \
  --cluster-name my-cluster
```

```
{
    "capabilities": [
        {
            "capabilityName": "my-ack",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/ack/my-
ack/abc123",
            "type": "ACK",
            "status": "ACTIVE",
            "createdAt": "2025-11-02T10:30:00.000000-07:00",
            "modifiedAt": "2025-11-02T10:32:15.000000-07:00",
        },
        {
            "capabilityName": "my-kro",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/kro/my-
kro/abc123",
            "type": "KRO",
            "status": "ACTIVE",
            "version": "v0.6.3",
            "createdAt": "2025-11-02T10:30:00.000000-07:00",
            "modifiedAt": "2025-11-02T10:32:15.000000-07:00",
        },
        {
            "capabilityName": "my-argocd",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/argocd/my-
argocd/abc123",
            "type": "ARGOCD",
            "status": "ACTIVE",
            "version": "3.1.8-eks-1",
            "createdAt": "2025-11-21T08:22:28.486000-05:00",
            "modifiedAt": "2025-11-21T08:22:28.486000-05:00"
        }
    ]
}
```

描述功能

获取有关特定功能的详细信息，包括其配置和状态。

控制台

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称，打开集群详细信息页面。
3. 选择功能选项卡。
4. 从托管功能中选择要查看的功能。
5. 查看功能详细信息，包括状态、配置和创建时间。

Amazon CLI

使用 `describe-capability` 命令查看详细信息。将 *region-code* 替换为集群所在的 Amazon 区域，将 *my-cluster* 替换为集群的名称，将 *capability-name* 替换为功能名称（ack、argocd 或 kro）。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name
```

输出示例：

```
{
  "capability": {
    "capabilityName": "my-ack",
    "capabilityArn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/ack/my-
ack/abc123",
    "clusterName": "my-cluster",
    "type": "ACK",
    "roleArn": "arn:aws:iam::111122223333:role/AmazonEKSCapabilityACKRole",
    "status": "ACTIVE",
    "configuration": {},
    "tags": {},
    "health": {
      "issues": []
    },
    "createdAt": "2025-11-19T17:11:30.242000-05:00",
    "modifiedAt": "2025-11-19T17:11:30.242000-05:00",
    "deletePropagationPolicy": "RETAIN"
  }
```

```
}
```

更新功能的配置

创建后，您可以更新功能配置的某些方面。具体的配置选项因功能类型而异。

> **Note**
>
> EKS 功能资源为完全托管，包括修补和版本更新。更新某项功能将更新资源配置，并且不会导致托管功能组件的版本更新。

Amazon CLI

使用 `update-capability` 命令修改功能：

```
aws eks update-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name \
  --role-arn arn:aws:iam::[.replaceable]111122223333:role/NewCapabilityRole
```

> **Note**
>
> 并非所有功能属性在创建后都可以更新。有关可以修改的内容的详细信息，请参阅特定于功能的文档。

删除功能

如果不再需要集群上的某项功能，可以删除该功能资源。

> **Important**
>
> 在删除功能之前，请先删除集群资源。
> 删除功能资源不会自动删除通过该功能创建的资源：
>
> - 所有 Kubernetes 自定义资源定义（CRD）仍安装在您的集群中。

- ACK 资源保留在您的集群中，相应的 Amazon 资源则保留在您的账户中
- Argo CD 应用程序及其 Kubernetes 资源保留在您的集群中
- kro ResourceGraphDefinitions 和实例保留在您的集群中

在删除功能之前，应删除这些资源，以避免存在孤立的资源。
您可以选择保留与 ACK Kubernetes 资源关联的 Amazon 资源。请参阅 ACK 注意事项

控制台

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称，打开集群详细信息页面。
3. 选择功能选项卡。
4. 从托管功能列表中选择要删除的功能。
5. 选择删除功能。
6. 在确认对话框中，输入功能名称以确认删除。
7. 选择删除。

Amazon CLI

使用 `delete-capability` 命令删除功能资源：

将 *region-code* 替换为集群所在的 Amazon 区域，将 *my-cluster* 替换为集群的名称，并将 *capability-name* 替换为要删除的功能名称。

```
aws eks delete-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name
```

后续步骤

- the section called “Kubernetes 资源”：了解每种功能类型提供的 Kubernetes 资源
- the section called “ACK 概念”：了解 ACK 概念和资源生命周期

- the section called “使用 Argo CD”：在 GitOps 工作流程中使用 Argo CD 功能
- the section called “kro 概念”：了解 kro 的基本概念及资源组合编排

功能 Kubernetes 资源

在集群上启用某项功能后，通常需要通过在集群中创建和管理 Kubernetes 自定义资源来与该功能进行交互。每种功能都提供自己的一组自定义资源定义（CRD），这些定义通过功能特定的功能扩展 Kubernetes API。

Argo CD 资源

启用 Argo CD 功能后，您可以创建和管理以下 Kubernetes 资源：

应用程序 ()

定义从 Git 存储库到目标集群的部署。Application 资源指定源存储库、目标命名空间和同步策略。您可以针对每个 Argo CD 功能实例创建 1000 个 Application 资源。

ApplicationSet

通过模板生成多个 Application 资源，从而实现多集群和多环境部署。ApplicationSet 资源使用生成器根据集群列表、Git 目录或其他来源动态创建 Application 资源。

AppProject

为 Application 资源提供逻辑分组和访问控制。AppProject 资源定义了 Application 资源可以使用的存储库、集群和命名空间，从而实现了多租户和安全边界。

示例 Application 资源：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/org/repo
    targetRevision: main
```

```
    path: manifests
  destination:
    server: https://kubernetes.default.svc
    namespace: production
```

有关 Argo CD 资源和概念的更多信息，请参阅 the section called “Argo CD 概念”。

kro 资源

启用 kro 功能后，您可以创建和管理以下 Kubernetes 资源：

ResourceGraphDefinition（RGD）

定义一个自定义 API，该 API 将多个 Kubernetes 和 Amazon 资源组合成更高级别的抽象。平台团队可创建 `ResourceGraphDefinition` 资源，以提供带有护栏的可重用模式。

自定义资源实例

创建 `ResourceGraphDefinition` 资源后，您可以创建由 `ResourceGraphDefinition` 定义的自定义 API 的实例。kro 会自动创建和管理 `ResourceGraphDefinition` 中指定的资源。

示例 `ResourceGraphDefinition` 资源：

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: web-application
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    spec:
      name: string
      replicas: integer
  resources:
    - id: deployment
      template:
        apiVersion: apps/v1
        kind: Deployment
        # ... deployment spec
    - id: service
      template:
```

```
        apiVersion: v1
        kind: Service
        # ... service spec
```

示例 `WebApplication` 实例：

```
apiVersion: v1alpha1
kind: WebApplication
metadata:
  name: my-web-app
  namespace: default
spec:
  name: my-web-app
  replicas: 3
```

当您应用此实例时，kro 会自动创建 `ResourceGraphDefinition` 中定义的 `Deployment` 和 `Service` 资源。

有关 kro 资源和概念的更多信息，请参阅 the section called “kro 概念”。

ACK 资源

启用 ACK 功能后，您可以使用 Kubernetes 自定义资源创建和管理 Amazon 资源。ACK 为 50 多项 Amazon 服务提供超过 200 个 CRD，允许您在使用 Kubernetes 工作负载的同时定义 Amazon 资源，并使用 Kubernetes 管理专用的 Amazon 基础设施资源。

ACK 资源示例：

S3 Bucket。

`Bucket` 资源使用版本控制、加密和生命周期策略创建和管理 Amazon S3 存储桶。

RDS DBInstance

`DBInstance` 资源通过自动备份和维护时段预置和管理 Amazon RDS 数据库实例。

DynamoDB 表

`Table` 资源使用已预置或按需容量创建和管理 DynamoDB 表。

IAM 角色

`Role` 资源通过信任策略和权限策略来定义 IAM 角色，以实现 Amazon 服务访问。

Lambda 函数

Function 资源使用代码、运行时和执行角色配置创建和管理 Lambda 函数。

Bucket 资源规范示例：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-app-bucket
spec:
  name: my-unique-bucket-name-12345
  versioning:
    status: Enabled
  encryption:
    rules:
      - applyServerSideEncryptionByDefault:
          sseAlgorithm: AES256
```

有关 ACK 资源和概念的更多信息，请参阅 the section called “ACK 概念”。

资源限制

EKS 功能具有以下资源限制：

Argo CD 使用量限制：

- 每个 Argo CD 功能实例最多 1000 个 Application 资源
- 每个 Argo CD 功能实例最多配置 100 个远程集群

资源配置限制：

- Argo CD 中每个 Application 资源最多 64 个 Kubernetes 资源
- kro 中每个 ResourceGraphDefinition 最多 64 个 Kubernetes 资源

> **Note**
>
> 这些上限适用于每个功能实例管理的资源数量。如果您需要更高上限，可以跨多个集群部署功能。

后续步骤

有关功能特定的任务和高级配置，请参阅以下主题：

- the section called “ACK 概念”：了解 ACK 概念和资源生命周期
- the section called “使用 Argo CD”：在 GitOps 工作流程中使用 Argo CD 功能
- the section called “kro 概念”：了解 kro 的基本概念及资源组合编排

EKS 功能和注意事项

本主题涵盖了使用 EKS 功能的重要注意事项，包括访问控制设计、在 EKS 功能和自主管理型解决方案之间进行选择、多集群部署的架构模式以及最佳操作实践。

功能 IAM 角色和 Kubernetes RBAC

每个 EKS 功能资源都有一个已配置的功能 IAM 角色。功能角色用于授予 EKS 功能代表您执行操作的 Amazon 服务权限。例如，要使用适用于 ACK 的 EKS 功能来管理 Amazon S3 存储桶，您需要向该功能授予 S3 存储桶管理权限，使其能够创建和管理存储桶。

配置该功能后，即可使用集群中的 Kubernetes 自定义资源创建和管理 AWS 中的 S3 资源。Kubernetes RBAC 是一种集群内访问控制机制，用于确定哪些用户和组可以创建和管理这些自定义资源。例如，向特定的 Kubernetes RBAC 用户和组授予在您选择的命名空间中创建和管理 `Bucket` 资源的权限。

因此，IAM 和 Kubernetes RBAC 共同构成了管理与 EKS 功能和资源相关的权限的端到端访问控制系统。为您的使用案例设计一个 IAM 权限与 RBAC 访问策略的正确组合非常重要。

有关功能 IAM 角色和 Kubernetes 权限的更多信息，请参阅 the section called “EKS 功能的注意事项”。

多集群架构模式

在跨多个集群部署功能时，请考虑以下常见的架构模式：

采用集中管理的中心和分支

在集中管理的集群中运行所有三项功能，以编排工作负载并管理多个工作负载集群中的云基础设施。

- 管理集群上的 Argo CD 可将应用程序部署到不同区域或账户中的工作负载集群
- 管理集群上的 ACK 会为所有集群预置 Amazon 资源（RDS、S3、IAM）
- 管理集群上的 kro 创建了适用于所有集群的可移植平台抽象

这种模式可以集中管理工作负载和云基础设施，并且可以简化管理许多集群的组织的操作。

分散式 GitOps

工作负载和云基础设施由运行工作负载的同一集群上的功能进行管理。

- Argo CD 管理本地集群上的应用程序资源。
- ACK 资源用于满足集群和工作负载需求。
- 已安装 kro 平台抽象并编排本地资源。

这种模式可以分散操作，团队在一个或多个集群中管理自己的专用平台服务。

采用混合 ACK 部署的中心和分支

将集中式和分散式模型与基于范围和所有权的集中式应用程序部署和资源管理相结合。

- 中心集群：
  - Argo CD 管理本地集群和所有远程工作负载集群的 GitOps 部署
  - ACK 在管理集群上用于管理员范围资源（生产数据库、IAM 角色、VPC）
  - kro 在管理集群上用于可重用的平台抽象
- 分支集群：
  - 工作负载通过集中式中心集群上的 Argo CD 进行管理
  - ACK 在本地用于工作负载范围资源（S3 存储桶、ElastiCache 实例、SQS 队列）
  - kro 在本地用于资源组合和构建块模式

这种模式将关注点进行分离：平台团队在管理集群上集中管理关键基础设施，可以选择包括工作负载集群，而应用程序团队则指定和管理云资源以及工作负载。

选择模式

选择架构时要考虑以下因素：

- 组织结构：集中式平台团队偏向于选择中心模式；去中心化团队可能更偏好每个集群的功能
- 资源范围：管理员范围的资源（数据库、IAM）通常受益于中心管理；而工作负载资源（存储桶、队列）可以在本地管理
- 自助服务：集中式平台团队可以编写和分发规范性的自定义资源，以便实现常见工作负载需求的云资源安全自助服务
- 集群实例集管理：集中式管理集群为 EKS 集群实例集管理提供了客户拥有的控制面板以及其他管理员范围资源
- 合规性要求：一些组织需要对审计和治理采用集中式的控制
- 操作复杂性：较少的功能实例可以简化操作，但可能会造成瓶颈

> **Note**
>
> 您可以从一种模式开始，然后随着平台的成熟发展到另一种模式。功能是独立的，您可以根据需要跨集群以不同的方式进行部署。

将 EKS 功能与自主管理型解决方案进行比较

EKS 功能为在 EKS 中运行的常用 Kubernetes 工具和控制器提供完全托管的体验。这与自主管理型解决方案不同，后者是在集群中安装和运行的。

主要区别

部署和管理

Amazon 完全管理 EKS 功能，无需安装、配置或维护组件软件。Amazon 在集群中自动安装和管理所有必需的 Kubernetes 自定义资源定义（CRD）。

使用自主管理型解决方案，您可以使用 Helm 图表、kubectl 或其他运算符来安装和配置集群软件。您可以完全控制自主管理型解决方案的软件生命周期和运行时配置，并在解决方案的任何层面提供自定义设置。

操作与维护

Amazon 通过自动更新和安全补丁管理 EKS 功能的补丁和其他软件生命周期操作。EKS 功能与 Amazon 功能集成，实现配置简化，提供内置的高可用性和容错能力，并且无需对控制器工作负载进行集群内问题排查。

自主管理型的解决方案要求您监控组件运行状况和日志，应用安全补丁和版本更新，配置包含多个副本和容器组（pod）中断预算的高可用性，对控制器工作负载问题进行问题排查和修复，以及管理发行版和版本。您可以完全控制自己的部署，但这通常需要为私有集群访问权限和其他集成提供定制解决方案，这些解决方案必须符合组织标准和安全合规性要求。

资源消耗

EKS 功能在 EKS 和集群之外运行，从而释放节点资源和集群资源。EKS 功能不使用集群工作负载资源，不消耗 Worker 节点上的 CPU 或内存，可自动扩展，也可尽量减少对集群容量规划的影响。

自主管理型的解决方案在您的 Worker 节点上运行控制器和其他组件，直接消耗 Worker 节点资源、集群 IP 和其他集群资源。管理集群服务需要对其工作负载进行容量规划，并且需要规划和配置资源请求和限制，以管理扩展和高可用性需求。

功能支持

作为完全托管的服务功能，与自主管理型的解决方案相比，EKS 功能本质上是较为固定的。虽然 EKS 功能支持大多数功能和使用案例，但与自主管理型的解决方案相比，覆盖范围会有所不同。

使用自主管理型的解决方案，您可以完全控制软件的配置、可选功能和其他方面的功能。您可以选择运行自己的自定义映像，自定义配置的各个方面，并完全控制自主管理型的解决方案功能。

与成本相关的注意事项

每个 EKS 功能资源都有相关的每小时成本，该费用因功能类型而异。由该功能管理的集群资源还会产生关联的每小时成本，此部分成本具有独立定价。有关更多信息，请参阅 Amazon EKS 定价。

自主管理型的解决方案没有与 Amazon 费用相关的直接成本，但您需要为控制器和相关工作负载使用的集群计算资源付费。除了节点和集群的资源消耗外，自主管理型解决方案的全部拥有成本还包括运营开销以及维护、问题排查和支持费用。

在 EKS 功能和自主管理型解决方案之间进行选择

EKS 功能当您想要减少运营开销，将注意力集中在软件和系统的差异化价值上，而不是为了基本要求而进行集群平台运营，请考虑这种选择。如果您希望最大限度地减少安全补丁和软件生命周期管理的运营负担，为应用程序工作负载腾出节点和集群资源，简化配置和安全管理，并从 Amazon 支持范围中受益，请使用 EKS 功能。EKS 功能非常适合大多数生产使用案例，也是新部署的推荐方法。

自主管理型解决方案当您需要特定的 Kubernetes 资源 API 版本、自定义控制器构建，拥有围绕自主管理型部署构建的现有自动化和工具，或者需要对控制器运行时配置进行深度自定义时，可以考虑这种选

择。自主管理型的解决方案为特殊使用案例提供了灵活性，并且您可以完全控制自己的部署和运行时配置。

> **Note**
>
> EKS 功能可以通过自主管理型解决方案在您的集群中共存，并且可以实现分步迁移。

特定功能的比较

有关详细比较，包括功能特定的功能、上游差异和迁移路径，请参阅：

- the section called “与自主管理型的比较”
- the section called “与自主管理型的比较”
- the section called “与自主管理型的比较”

通过 Amazon Controllers for Kubernetes（ACK）部署来自 Kubernetes 的 Amazon 资源

借助 Amazon Controllers for Kubernetes（ACK），您可以直接在 Kubernetes 定义和管理 Amazon 服务资源。借助 Amazon Controllers for Kubernetes（ACK），您还可以利用熟悉的 Kubernetes API 和工具，将 Kubernetes 自定义资源与应用程序工作负载配合使用，来管理工作负载资源和云基础设施。

得益于 EKS 功能，ACK 完全由 Amazon 托管，无需您在集群上安装、维护和扩展 ACK 控制器。

ACK 的工作原理

ACK 会将 Kubernetes 自定义资源规范转换为 Amazon API 调用。创建、更新或删除代表 Amazon 服务资源的 Kubernetes 自定义资源时，ACK 会进行必要的 Amazon API 调用来创建、更新或删除相应的 Amazon 资源。

ACK 支持的每个 Amazon 资源都有自己的自定义资源定义（CRD），其作用是定义用以指定其配置的 Kubernetes API 架构。例如，ACK 为 S3 提供 CRD，包括存储桶、存储桶策略及其他 S3 资源。

ACK 会持续协调 Amazon 资源的实际状态，使其与 Kubernetes 自定义资源中定义的预期状态保持一致。如果资源实际状态与预期状态存在偏差，ACK 会检测到这一情况，并采取纠正措施使其恢复一致。对 Kubernetes 资源进行的更改会立即反映在 Amazon 资源状态中，而对上游 Amazon 资源更改的被动偏差检测和修复可能需要长达 10 个小时（即重新同步周期），但实际速度通常要快得多。

示例 S3 存储桶资源清单

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-ack-bucket
spec:
  name: my-unique-bucket-name
```

将此自定义资源应用于集群时，如果 Amazon S3 存储桶尚不存在，ACK 会在账户中创建一个存储桶。对该资源的后续更改（例如指定非默认存储层或添加策略）将应用于 Amazon 中的 S3 资源。从集群中删除此资源时，默认情况下也将删除 Amazon 中的 S3 存储桶。

ACK 的优势

ACK 有助于您在 Kubernetes 本地管理 Amazon 资源，从而让您更方便地使用与管理应用程序相同的 Kubernetes API 和工具来管理 Amazon 资源。这种统一的方法简化了基础设施管理工作流程，无需在不同工具之间切换或学习单独的基础设施即代码系统。您可以在 Kubernetes 清单中以声明方式定义 Amazon 资源，从而实现 GitOps 工作流程和基础设施即代码实践，并与现有的开发流程无缝集成。

ACK 会根据 Amazon 资源的预期状态来持续协调实际状态，从而纠正偏差并确保整个基础设施的一致性。这种持续协调意味着，对 Amazon 资源进行的命令式带外更改都将自动恢复，以匹配所声明的配置，从而维护基础设施即代码的完整性。您可以配置 ACK 以管理跨多个 Amazon 账户和区域的资源，无需其他工具即可部署复杂的多账户架构。

对于从其他基础设施管理工具迁移的组织，ACK 支持资源接管，从而允许您将现有的 Amazon 资源纳入 ACK 管理，无需重新创建资源。此外，ACK 不仅可在不修改访问权限的前提下提供用于观察 Amazon 资源的只读资源，还可提供注释，即使从集群中删除 Kubernetes 资源也可选择保留 Amazon 资源。

要了解更多信息并开始使用适用于 ACK 的 EKS 功能，请参阅 the section called “ACK 概念”和the section called “注意事项”。

支持的 Amazon 服务

ACK 支持众多 Amazon 服务，包括但不限于：

- Amazon EC2

- Amazon S3
- Amazon RDS
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon EKS
- Amazon SQS
- Amazon SNS
- Amazon Lambda
- Amazon IAM

适用于 ACK 的 EKS 功能支持所有在上游列为“已正式发布”的 Amazon 服务。有关详细信息，请参阅支持的 Amazon 服务完整列表。

与其他 EKS 托管功能集成

ACK 可与其他 EKS 托管功能集成。

- Argo CD：使用 Argo CD 跨多个集群管理 ACK 资源的部署，为 Amazon 基础设施启用 GitOps 工作流。
  - 当与 ArgoCD 配对时，ACK 能扩大 GitOps 的优势，但 ACK 本身不要求与 git 集成。
- kro（Kube Resource Orchestrator）：使用 kro 从 ACK 资源组合复杂的资源，创建更高级别的抽象以简化资源管理。
  - 您可以使用 kro 创建复合自定义资源，这些资源同时定义了 Kubernetes 资源和 Amazon 资源。团队成员可以使用这些自定义资源，快速部署复杂的应用程序。

开始使用 ACK

要开始使用适用于 ACK 的 EKS 功能，请执行以下操作：

1. 创建并配置一个 IAM 功能角色，授予 ACK 代表您管理 Amazon 资源所需的必要权限。
2. 通过 Amazon 管理控制台、Amazon CLI 或偏好的基础设施即代码工具，在 EKS 集群上创建 ACK 功能资源。
3. 将 Kubernetes 自定义资源应用于集群，从而开始在 Kubernetes 中管理 Amazon 资源。

创建 ACK 功能

本章节旨在介绍如何在 Amazon EKS 集群上创建 ACK 功能。

先决条件

创建 ACK 功能之前，确保满足以下条件：

- Amazon EKS 集群
- 具有允许 ACK 管理 Amazon 资源的权限的 IAM 功能角色
- 可在 EKS 集群上创建功能资源的充足 IAM 权限
- 已安装和配置相应的 CLI 工具，或可访问 EKS 控制台

有关如何创建 IAM 功能角色的说明，请参阅 the section called “功能 IAM 角色”。

> **Important**
>
> ACK 是一种基础设施管理功能，可创建、修改和删除 Amazon 资源。这是一项管理员范围的功能，应谨慎控制。任何具有在您的集群中创建 Kubernetes 资源权限的用户，都可以通过 ACK 有效地创建 Amazon 资源，但需具有 IAM 功能角色权限。您提供的 IAM 功能角色决定 ACK 可以创建和管理哪些 Amazon 资源。有关创建具备最低权限的合适角色的操作指引，请参阅 the section called “功能 IAM 角色”和 the section called “EKS 功能的注意事项”。

选择工具

您可以使用Amazon Web Services 管理控制台、Amazon CLI 或 eksctl 创建 ACK 功能：

- the section called “控制台”：使用控制台获得引导式体验
- the section called “ Amazon CLI”：使用 Amazon CLI 进行脚本编写和自动化
- the section called “eksctl”：使用 eksctl 获得 Kubernetes 原生体验

在创建 ACK 功能时会执行的操作

创建 ACK 功能时：

1. EKS 会创建 ACK 功能服务，并配置该服务来监控和管理集群内的资源

2. 自定义资源定义（CRD）将被安装到集群中
3. 该功能会代入您提供的 IAM 功能角色
4. ACK 开始监视集群中的自定义资源
5. 功能状态从 `CREATING` 更改为 `ACTIVE`

激活后，您可以在集群中创建 ACK 自定义资源，来管理 Amazon 资源。

后续步骤

创建 ACK 功能后：

- the section called “ACK 概念”：了解 ACK 概念并开始使用 Amazon 资源
- the section called “ACK 概念”：了解协调、字段导出和资源采用模式
- the section called “配置权限”：配置 IAM 权限和多账户模式

使用控制台创建 ACK 功能

本主题将介绍如何使用 Amazon Web Services 管理控制台创建 Amazon Controllers for Kubernetes（ACK）功能。

创建 ACK 功能

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称，打开集群详细信息页面。
3. 选择功能选项卡。
4. 在左侧导航栏中，选择 Amazon Controllers for Kubernetes（ACK）。
5. 选择创建 Amazon Controllers for Kubernetes 功能。
6. 对于 IAM 功能角色：
   - 如果已有 IAM 功能角色，请从下拉列表中选择该角色
   - 如果您需要创建一个新角色，请选择创建管理员角色

     这将在新选项卡中打开 IAM 控制台，其中包含预先填充的信任策略和 `AdministratorAccess` 托管式策略。您可以根据需要取消选择此策略并添加其他权限。

     创建角色后，返回 EKS 控制台，系统将自动选择该角色。

> **Important**
>
> 建议的 `AdministratorAccess` 策略授予了广泛权限，旨在简化入门流程。对于生产用途，请将其替换为自定义策略，该策略仅授予您计划通过 ACK 管理的特定 Amazon 服务所需的权限。有关创建最低权限策略的指导，请参阅the section called “配置权限”和 the section called “EKS 功能的注意事项”。

7. 选择创建。

功能创建过程随即开始。

验证功能是否处于活动状态

1. 在功能选项卡上，查看 ACK 功能状态。
2. 等待状态从 `CREATING` 更改为 `ACTIVE`。
3. 变为活动状态后，该功能即可使用。

有关功能状态和问题排查的信息，请参阅the section called “使用功能”。

验证自定义资源是否可用

功能激活后，验证 ACK 自定义资源是否已在集群中可用。

使用控制台

1. 在 Amazon EKS 控制台中导航至集群
2. 选择资源选项卡
3. 选择扩展
4. 选择 CustomResourceDefinitions

您应该会看到列出的多个用于 Amazon 资源的 CRD。

使用 kubectl

```
kubectl api-resources | grep services.k8s.aws
```

您应该会看到列出的多个用于 Amazon 资源的 API。

> **Note**
>
> Amazon Controllers for Kubernetes 功能将为各种 Amazon 资源安装大量 CRD。

后续步骤

- the section called “ACK 概念”：了解 ACK 概念并开始使用
- the section called “配置权限”：为其他 Amazon 服务配置 IAM 权限
- the section called “使用功能”：管理 ACK 功能资源

使用 Amazon CLI 创建 ACK 功能

本主题将介绍如何使用 Amazon CLI 创建 Amazon Controllers for Kubernetes（ACK）功能。

先决条件

- Amazon CLI：版本 `2.12.3` 或更高版本。要检查版本，请运行 `aws --version`。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的安装。
- **`kubectl`**：用于与 Kubernetes 集群结合使用的命令行工具。有关更多信息，请参阅 the section called “设置 kubectl 和 eksctl”。

步骤 1：创建 IAM 功能角色

创建信任策略文件：

```
cat > ack-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
```

```
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

创建 IAM 角色：

```
aws iam create-role \
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://ack-trust-policy.json
```

将 AdministratorAccess 托管式策略附加到角色：

```
aws iam attach-role-policy \
  --role-name ACKCapabilityRole \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

> **Important**
>
> 建议的 AdministratorAccess 策略授予了广泛权限，旨在简化入门流程。对于生产用途，请将其替换为自定义策略，该策略仅授予您计划通过 ACK 管理的特定 Amazon 服务所需的权限。有关创建最低权限策略的指导，请参阅the section called “配置权限”和 the section called “EKS 功能的注意事项”。

步骤 2：创建 ACK 功能

在集群上创建 ACK 功能资源。将 *region-code* 替换为您的集群所在的 Amazon 区域，并将 *my-cluster* 替换为您的集群的名称。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack \
  --type ACK \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ACKCapabilityRole \
  --delete-propagation-policy RETAIN
```

命令会立即返回，但是由于 EKS 正在创建所需的功能基础设施和组件，该功能需要一些时间才能变为活动状态。在创建集群时，EKS 将在集群中安装与此功能相关的 Kubernetes 自定义资源定义。

> **Note**
>
> 如果收到集群不存在或您没有权限的错误消息，请验证：
>
> - 集群名称是否正确
> - Amazon CLI 是否针对正确的区域进行配置
> - 您是否拥有所需的 IAM 权限

步骤 3：验证功能是否处于活动状态

等待功能变为活动状态。将 *region-code* 替换为您的集群所在的 Amazon 区域，并将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack \
  --query 'capability.status' \
  --output text
```

当状态显示为 ACTIVE 时，表示功能已准备就绪。在该状态变为 ACTIVE 之前，请勿继续执行下一步。

您也可以查看完整功能详细信息：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack
```

步骤 4：验证自定义资源是否可用

功能激活后，验证 ACK 自定义资源是否已在集群中可用：

```
kubectl api-resources | grep services.k8s.aws
```

您应该会看到列出的多个用于 Amazon 资源的 API。

> **Note**
>
> Amazon Controllers for Kubernetes 功能将为各种 Amazon 资源安装大量 CRD。

后续步骤

- the section called “ACK 概念”：了解 ACK 概念并开始使用
- the section called “配置权限”：为其他 Amazon 服务配置 IAM 权限
- the section called “使用功能”：管理 ACK 功能资源

使用 eksctl 创建 ACK 功能

本主题将介绍如何使用 eksctl 创建 Amazon Controllers for Kubernetes（ACK）功能。

> **Note**
>
> 以下步骤需要 eksctl 版本 `0.220.0` 或更高版本。要检查版本，请运行 `eksctl version`。

步骤 1：创建 IAM 功能角色

创建信任策略文件：

```
cat > ack-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
```

```
  ]
}
EOF
```

创建 IAM 角色：

```
aws iam create-role \
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://ack-trust-policy.json
```

将 AdministratorAccess 托管式策略附加到角色：

```
aws iam attach-role-policy \
  --role-name ACKCapabilityRole \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

Important

建议的 AdministratorAccess 策略授予了广泛权限，旨在简化入门流程。对于生产用途，请将其替换为自定义策略，该策略仅授予您计划通过 ACK 管理的特定 Amazon 服务所需的权限。有关创建最低权限策略的指导，请参阅the section called “配置权限”和 the section called “EKS 功能的注意事项”。

Important

此策略通过 "Resource": "*" 授予 S3 存储桶管理权限，允许对所有 S3 存储桶进行操作。用于生产用途：* 将 Resource 字段限制为特定的存储桶 ARN 或名称模式 * 使用 IAM 条件键通过资源标签限制访问权限 * 仅授予使用案例所需的最低权限
有关其他 Amazon 服务，请参阅the section called “配置权限”。

将该策略附加到角色：

```
aws iam attach-role-policy \
  --role-name ACKCapabilityRole \
  --policy-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):policy/ACKS3Policy
```

步骤 2：创建 ACK 功能

使用 eksctl 创建 ACK 功能。将 *region-code* 替换为您的集群所在的 Amazon 区域，并将 *my-cluster* 替换为您的集群的名称。

```
eksctl create capability \
  --cluster [.replaceable]`my-cluster` \
  --region [.replaceable]`region-code` \
  --name ack \
  --type ACK \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ACKCapabilityRole \
  --ack-service-controllers s3
```

> **Note**
>
> `--ack-service-controllers` 标记是可选的。如果省略，ACK 将启用所有可用的控制器。为了获得更好的性能和安全性，请考虑仅启用所需的控制器。您可以指定多个控制器，例如：`--ack-service-controllers s3,rds,dynamodb`

命令会立即返回，但该功能需要一些时间才能变为活动状态。

步骤 3：验证功能是否处于活动状态

检查功能状态：

```
eksctl get capability \
  --cluster [.replaceable]`my-cluster` \
  --region [.replaceable]`region-code` \
  --name ack
```

当状态显示为 ACTIVE 时，表示功能已准备就绪。

步骤 4：验证自定义资源是否可用

功能激活后，验证 ACK 自定义资源是否已在集群中可用：

```
kubectl api-resources | grep services.k8s.aws
```

您应该会看到列出的多个用于 Amazon 资源的 API。

> **Note**
>
> Amazon Controllers for Kubernetes 功能将为各种 Amazon 资源安装大量 CRD。

后续步骤

- the section called “ACK 概念”：了解 ACK 概念并开始使用
- the section called “配置权限”：为其他 Amazon 服务配置 IAM 权限
- the section called “使用功能”：管理 ACK 功能资源

ACK 概念

ACK 通过 Kubernetes API 管理 Amazon 资源，方法是根据清单中定义的预期状态，持续协调 Amazon 中的实际状态。创建或更新 Kubernetes 自定义资源时，ACK 会进行必要的 Amazon API 调用来创建或修改相应的 Amazon 资源，随后持续监控其状态是否存在偏差，并更新 Kubernetes 中的资源状态以反映当前实际状态。借助此方法，您可以使用熟悉的 Kubernetes 工具和工作流程来管理基础设施，同时保持集群与 Amazon 之间的一致性。

本主题将阐释有关 ACK 如何通过 Kubernetes API 管理 Amazon 资源的基本概念。

开始使用 ACK

创建 ACK 功能后（请参阅the section called “创建 ACK 功能”），即可开始在集群中使用 Kubernetes 清单管理 Amazon 资源。

例如，将以下 S3 存储桶清单创建为 `bucket.yaml`，并为自己的存储桶选择唯一名称。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-test-bucket
  namespace: default
spec:
  name: my-unique-bucket-name-12345
```

应用清单：

```
kubectl apply -f bucket.yaml
```

检查状态：

```
kubectl get bucket my-test-bucket
kubectl describe bucket my-test-bucket
```

验证存储桶是否已在 Amazon 中创建：

```
aws s3 ls | grep my-unique-bucket-name-12345
```

删除 Kubernetes 资源：

```
kubectl delete bucket my-test-bucket
```

验证存储桶是否已从 Amazon 中删除：

```
aws s3 ls | grep my-unique-bucket-name-12345
```

该存储桶不应再出现在列表中，这表明 ACK 正在管理 Amazon 资源的完整生命周期。

有关 ACK 入门的更多信息，请参阅 ACK 入门。

资源生命周期与协调

ACK 使用持续的协调循环，以确保 Amazon 资源与 Kubernetes 清单中定义的预期状态保持一致。

协调的工作原理：

1. 您创建或更新 Kubernetes 自定义资源（例如 S3 存储桶）
2. ACK 检测到更改，并将预期状态与 Amazon 中的实际状态进行比较
3. 如果两者不一致，ACK 会进行 Amazon API 调用来协调差异
4. ACK 会更新 Kubernetes 中的资源状态以反映当前状态
5. 该循环持续重复，通常每隔几个小时运行一次

在以下情况下会触发协调：创建新的 Kubernetes 资源、更新现有资源的 `spec`，或 ACK 检测到在 ACK 外部进行的手动更改导致 Amazon 中的状态存在偏差。此外，ACK 还会定期进行协调，其重新同

步周期为 10 个小时。对 Kubernetes 资源进行更改会立即触发协调，而对上游 Amazon 资源更改的被动偏差检测则会在重新同步周期内发生。

当您完成上述开始使用示例后，ACK 会执行以下步骤：

1. 检查 Amazon 中是否存在存储桶
2. 如果不存在，则调用 `s3:CreateBucket`
3. 使用存储桶 ARN 和状态更新 Kubernetes 状态
4. 持续监控状态偏差

要了解有关 ACK 工作原理的更多信息，请参阅 ACK 协调。

状态条件

ACK 资源使用状态条件来传达其状态。理解这些条件有助于您排查问题并了解资源运行状况。

- Ready：表示资源已准备就绪，可供使用（标准化的 Kubernetes 条件）。
- ACK.ResourceSynced：表示资源规范与 Amazon 资源状态相匹配。
- ACK.Terminal：表示发生了不可恢复的错误。
- ACK.Adopted：表示资源是从现有 Amazon 资源接管而来，而非新建的。
- ACK.Recoverable：表示发生了可恢复的错误，可能无需更新规范即可解决。
- ACK.Advisory：提供有关资源的参考信息。
- ACK.LateInitialized：表示是否已完成字段的延迟初始化。
- ACK.ReferencesResolved：表示是否已解析所有 `AWSResourceReference` 字段。
- ACK.IAMRoleSelected：表示是否已选择 IAMRoleSelector 来管理此资源。

检查资源状态：

```
# Check if resource is ready
kubectl get bucket my-bucket -o jsonpath='{.status.conditions[?
(@.type=="Ready")].status}'

# Check for terminal errors
kubectl get bucket my-bucket -o jsonpath='{.status.conditions[?
(@.type=="ACK.Terminal")]}'
```

示例状态：

```
status:
  conditions:
  - type: Ready
    status: "True"
    lastTransitionTime: "2024-01-15T10:30:00Z"
  - type: ACK.ResourceSynced
    status: "True"
    lastTransitionTime: "2024-01-15T10:30:00Z"
  - type: ACK.Terminal
    status: "True"
  ackResourceMetadata:
    arn: arn:aws:s3:::my-unique-bucket-name
    ownerAccountID: "111122223333"
    region: us-west-2
```

要了解有关 ACK 状态和条件的更多信息，请参阅 ACK 条件。

删除策略

ACK 的删除策略决定了当您删除 Kubernetes 资源时，相应 Amazon 资源的处置方式。

删除（默认）

删除 Kubernetes 资源时，相应的 Amazon 资源也将删除，这是默认行为。

```
# No annotation needed - this is the default
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: temp-bucket
spec:
  name: temporary-bucket
```

删除此资源将同时删除 Amazon 中的 S3 存储桶。

Retain：

删除 Kubernetes 资源时，相应的 Amazon 资源将保留：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: important-bucket
```

```
  annotations:
    services.k8s.aws/deletion-policy: "retain"
spec:
  name: production-data-bucket
```

删除此资源会将其从 Kubernetes 中移除，但会保留 Amazon 中的 S3 存储桶。

`retain` 策略适用于以下场景：生产数据库的生命周期超过 Kubernetes 资源，多个应用程序使用共享资源，资源包含不应意外删除的重要数据，或者临时管理 ACK（即先接管一个资源，对其进行配置，然后将其释放回手动管理）。

要了解有关 ACK 删除策略的更多信息，请参阅 ACK 删除策略。

资源接管

通过接管，您可以将现有的 Amazon 资源纳入 ACK 管理，而无需重新创建它们。

何时使用接管：

- 将现有基础设施迁移到 ACK 管理
- 在 Kubernetes 中意外删除资源的情况下，恢复孤立的 Amazon 资源
- 导入由其他工具（CloudFormation、Terraform）创建的资源

接管的工作原理：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: existing-bucket
  annotations:
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
  name: my-existing-bucket-name
```

当您创建此资源时：

1. ACK 会检查 Amazon 中是否存在具有该名称的存储桶
2. 如果找到存储桶，ACK 就会接管它（无需调用 API 创建）
3. ACK 从 Amazon 读取当前配置
4. ACK 会更新 Kubernetes 状态以反映实际状态

5. 未来的更新会正常协调该资源

接管后，资源将像任何其他 ACK 资源一样进行管理，并且删除 Kubernetes 资源时将删除相应的 Amazon 资源，除非您使用了 `retain` 删除策略。

接管资源时，Amazon 资源必须已存在，并且 ACK 需要读取权限才能发现它。如果资源存在，`adopt-or-create` 策略会接管该资源，如果资源不存在，则会创建。当您需要一个无论资源是否存在都能正常工作的声明式工作流程时，这将非常有用。

要了解有关 ACK 资源接管的更多信息，请参阅 ACK 资源接管。

跨账户和跨区域资源

ACK 可以从单个集群管理位于不同 Amazon 账户和区域的资源。

跨区域资源注释

您可以使用注释指定 Amazon 资源的区域：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: eu-bucket
  annotations:
    services.k8s.aws/region: eu-west-1
spec:
  name: my-eu-bucket
```

您还可以为特定命名空间中创建的所有 Amazon 资源指定默认区域：

命名空间注释

为命名空间中的所有资源设置默认区域：

```
apiVersion: v1
kind: Namespace
metadata:
  name: production
  annotations:
    services.k8s.aws/default-region: us-west-2
```

除非被资源级注释覆盖，否则在此命名空间中创建的资源将使用此区域。

跨账户

使用 IAM 角色选择器将特定的 IAM 角色映射到命名空间：

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
metadata:
  name: target-account-config
spec:
  arn: arn:aws:iam::444455556666:role/ACKTargetAccountRole
  namespaceSelector:
    names:
      - production
```

在映射的命名空间中创建的资源会自动使用指定的角色。

要了解有关 IAM 角色选择器的更多信息，请参阅 ACK 跨账户资源管理。有关跨账户配置的详细信息，请参阅the section called “配置权限”。

错误处理和重试行为

ACK 会自动处理瞬态错误并重试失败的操作。

重试策略：

- 瞬态错误（速率限制、临时服务问题、权限不足）会触发自动重试
- 指数回退可防止 API Amazon 负载过重
- 最大重试次数因错误类型而异
- 永久错误（参数无效、资源名称冲突）不会重试

使用 `kubectl describe` 检查资源状态以了解错误详情：

```
kubectl describe bucket my-bucket
```

查找包含错误消息的状态条件、显示最近协调尝试的事件，以及状态条件中说明失败原因的 `message` 字段。常见错误包括 IAM 权限不足、Amazon 中的资源名称冲突、`spec` 中的配置值无效、超出 Amazon 服务配额等。

有关如何排查常见错误，请参阅the section called “问题排查”。

使用 kro 进行资源组合

如需将多个 ACK 资源组合并连接在一起，请使用适用于 kro 的 EKS 功能（Kube Resource Orchestrator）。kro 提供了一种声明方式来定义资源组，通过在资源之间传递配置，来简化复杂基础设施模式的管理。

有关使用 ACK 资源创建自定义资源组合的详细示例，请参阅 the section called “kro 概念”

后续步骤

- the section called “注意事项”：EKS 特定模式与集成策略

配置 ACK 权限

ACK 需要 IAM 权限才能代表您创建和管理 Amazon 资源。本主题将介绍 IAM 如何与 ACK 结合使用，并提供有关为不同使用案例配置权限的指导。

IAM 如何与 ACK 结合使用

ACK 使用 IAM 角色向 Amazon 进行身份验证并对资源执行操作。向 ACK 提供权限有两种方式：

功能角色：您在创建 ACK 功能时提供的 IAM 角色。默认情况下，所有 ACK 操作都使用此角色。

IAM 角色选择器：可映射到特定命名空间或资源的其他 IAM 角色。这些角色会覆盖其作用域内资源的功能角色。

当 ACK 需要创建或管理资源时，它会决定使用哪个 IAM 角色：

1. 检查 IAMRoleSelector 是否与资源所在的命名空间相匹配
2. 如果找到匹配项，则代入该 IAM 角色
3. 否则，请使用功能角色

这种方式可以实现灵活的权限管理，从简单的单角色设置到复杂的多账户、多团队配置均适用。

入门：简单权限设置

适用于开发、测试或简单使用案例，您可以将所有必要的服务权限直接添加到功能角色。

此方式适用于以下情况：

- 您刚开始使用 ACK

- 所有资源都在同一 Amazon 账户中
- 由单个团队管理所有 ACK 资源
- 您信任所有 ACK 用户拥有相同的权限

生产最佳实践：IAM 角色选择器

对于生产环境，请使用 IAM 角色选择器来实现最低权限访问和命名空间级隔离。

使用 IAM 角色选择器时，功能角色只需要 `sts:AssumeRole` 权限即可代入服务特定角色。您无需向功能角色本身添加任何 Amazon 服务权限（如 S3 或 RDS），系统会向功能角色所代入的各个 IAM 角色授予这些权限。

选择权限模式：

在以下情况下，使用直接权限（将服务权限添加到功能角色）：

- 您刚开始使用并希望最简单的设置
- 所有资源与集群位于同一账户
- 您有集群范围的管理权限要求
- 所有团队都可以共享相同的权限

在以下情况下，使用 IAM 角色选择器：

- 跨多个 Amazon 账户管理资源
- 不同的团队或命名空间需要不同的权限
- 您需要按命名空间进行精细访问控制
- 您希望遵循最低权限安全实践

您可以从直接权限开始，随着需求的增长，再迁移到 IAM 角色选择器。

为什么要在生产环境中使用 IAM 角色选择器：

- 最低权限：每个命名空间仅获得其所需的权限
- 团队隔离：团队 A 不会意外使用团队 B 的权限
- 更易审计：清晰映射哪个命名空间使用哪个角色
- 跨账户支持：在多个账户中管理资源所必需的功能

- 关注点分离：不同的服务或环境使用不同的角色

基本 IAM 角色选择器设置

步骤 1：创建服务特定的 IAM 角色

创建具有特定 Amazon 服务权限的 IAM 角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": "*"
    }
  ]
}
```

配置信任策略，以允许功能角色代入该策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ACKCapabilityRole"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

步骤 2：向功能角色授予 AssumeRole 权限

向功能角色添加代入该服务特定角色的权限：

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::111122223333:role/ACK-S3-Role"
    }
  ]
}
```

步骤 3：创建 IAMRoleSelector

将 IAM 角色映射到命名空间：

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
metadata:
  name: s3-namespace-config
spec:
  arn: arn:aws:iam::111122223333:role/ACK-S3-Role
  namespaceSelector:
    names:
      - s3-resources
```

步骤 4：在映射的命名空间中创建资源

`s3-resources` 命名空间中的资源会自动使用指定的角色：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-bucket
  namespace: s3-resources
spec:
  name: my-production-bucket
```

多账户管理

使用 IAM 角色选择器跨多个 Amazon 账户管理资源。

步骤 1：创建跨账户 IAM 角色

在目标账户（444455556666）中，创建一个信任源账户功能角色的角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ACKCapabilityRole"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

将服务特定权限附加到此角色。

步骤 2：授予 AssumeRole 权限

在源账户（111122223333）中，允许功能角色代入目标账户角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::444455556666:role/ACKTargetAccountRole"
    }
  ]
}
```

步骤 3：创建 IAMRoleSelector

将跨账户角色映射到命名空间：

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
metadata:
  name: production-account-config
spec:
  arn: arn:aws:iam::444455556666:role/ACKTargetAccountRole
  namespaceSelector:
    names:
```

```
      - production
```

步骤 4：创建资源

`production` 命名空间中的资源将在目标账户中创建：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-bucket
  namespace: production
spec:
  name: my-cross-account-bucket
```

高级 IAM 角色选择器模式

有关高级配置，例如标签选择器、资源特定角色映射及其他示例，请参阅 ACK IRSA 文档。

后续步骤

- the section called “ACK 概念”：了解 ACK 概念和资源生命周期
- the section called “ACK 概念”：了解资源接管和删除策略
- the section called “EKS 功能的注意事项”：了解针对功能的安全最佳实践

针对 EKS 的 ACK 注意事项

本主题涵盖使用适用于 ACK 的 EKS 功能的重要注意事项，包括 IAM 配置、多账户模式以及与其他 EKS 功能集成。

IAM 配置模式

ACK 功能使用 IAM 功能角色向 Amazon 进行身份验证。根据需求选择合适的 IAM 模式。

简易模式：单一功能角色

适用于开发、测试或简单使用案例，直接向功能角色授予所有必要权限。

何时使用

- 开始使用 ACK
- 单账户部署

- 所有资源均由一个团队托管
- 开发和测试环境

示例：向功能角色添加带有资源标记条件的 S3 和 RDS 权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:*"],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-west-2", "us-east-1"]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["rds:*"],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-west-2", "us-east-1"]
        },
      }
    }
  ]
}
```

此示例将 S3 和 RDS 操作限制在特定区域，并要求 RDS 资源必须带有 ManagedBy： ACK 标签。

生产模式：IAM 角色选择器

对于生产环境，请使用 IAM 角色选择器来实现最低权限访问和命名空间级隔离。

何时使用

- 生产环境
- 多团队集群

- 多账户资源管理
- 满足最低权限安全要求
- 不同的服务需要不同的权限

优点：

- 每个命名空间仅获得其所需的权限
- 团队隔离：团队 A 无法使用团队 B 的权限
- 更易于审计和合规
- 需要进行跨账户资源管理

有关 IAM 角色选择器配置的详细信息，请参阅the section called “配置权限”。

与其他 EKS 功能集成

将 Argo CD 与 GitOps 搭配使用

使用适用于 Argo CD 的 EKS 功能，从 Git 存储库部署 ACK 资源，从而为基础设施管理启用 GitOps 工作流程。

注意事项：

- 将 ACK 资源与应用程序清单存储在一起，实现端到端 GitOps
- 根据团队结构，按环境、服务或资源类型整理
- 使用 Argo CD 的自动同步功能实现持续协调
- 启用修剪来自动移除已删除的资源
- 考虑采用轴辐模式进行多集群基础设施管理

GitOps 提供审计跟踪记录、回滚功能和声明式基础设施管理。有关 Argo CD 的更多信息，请参阅the section called “使用 Argo CD”。

使用 kro 进行资源组合

使用适用于 kro 的 EKS 功能（Kube Resource Orchestrator），将多个 ACK 资源组合成更高级别的抽象和自定义 API。

何时将 kro 与 ACK 结合使用：

- 为常见的基础设施堆栈创建可重复使用的模式（数据库 + 备份 + 监控）
- 为应用团队构建带有简化 API 的自助式平台
- 管理资源依赖项并在资源之间传递值（例如将 S3 存储桶 ARN 传递给 Lambda 函数）
- 跨多个团队标准化基础设施配置
- 通过将实现细节隐藏在自定义资源背后来降低复杂性

示例模式：

- 应用程序堆栈：S3 存储桶 + SQS 队列 + 通知配置
- 数据库设置：RDS 实例 + 参数组 + 安全组 + 密钥
- 联网：VPC + 子网 + 路由表 + 安全组

kro 负责处理组合资源的依赖项排序、状态传播和生命周期管理。有关 kro 的更多信息，请参阅 the section called “kro 概念”。

整理资源

使用 Kubernetes 命名空间和 Amazon 资源标签来整理 ACK 资源，以便更轻松地进行管理、访问控制和成本跟踪。

整理命名空间

使用 Kubernetes 命名空间按环境（生产、暂存、开发）、团队（平台、数据、ml）或应用程序在逻辑上分隔 ACK 资源。

优点：

- 通过作用域为命名空间的 RBAC 实现访问控制
- 使用注释为每个命名空间设置默认区域
- 更轻松地管理和清理资源
- 逻辑分隔与组织结构一致

资源标签

EKS 会自动将标签应用于由 ACK 托管的 Amazon 资源，包括功能资源 ARN。为成本分配、所有权跟踪和整理目的添加其他标签。

推荐标签：

- 环境（生产、暂存、开发）
- 团队或部门所有权
- 用于账单分配的成本中心
- 应用程序或服务名称
- ManagedBy：ACK（用于识别由 ACK 托管的资源）

从其他基础设施即代码工具迁移

许多组织发现，将 Kubernetes 标准化扩展到工作负载编排之外具有重要价值。将基础设施和 Amazon 资源管理迁移到 ACK，可以让您使用 Kubernetes API 来标准化基础设施管理，并与应用程序工作负载一起管理。

在 Kubernetes 中标准化基础设施管理的优势：

- 单一事实来源：在 Kubernetes 中同时管理应用程序和基础设施，实现端到端 GitOps 实践
- 统一工具：团队使用 Kubernetes 资源和工具，无需学习多种工具和框架
- 一致协调机制：ACK 会像 Kubernetes 协调工作负载一样持续协调 Amazon 资源，相比命令式工具，ACK 能检测和纠正状态偏差
- 原生组合：通过将 kro 与 ACK 结合，直接在应用程序和资源清单中引用 Amazon 资源，在资源之间传递连接字符串和 ARN
- 简化操作：为整个系统提供统一的控制面板，用于实现部署、回滚和可观测性

ACK 支持在不重新创建的情况下接管现有 Amazon 资源，从而实现从 CloudFormation、Terraform 或集群外部资源进行零停机时间迁移。

接管现有资源：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: existing-bucket
  annotations:
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
```

```
  name: my-existing-bucket-name
```

接管后，资源将由 ACK 管理，并且可以通过 Kubernetes 清单进行更新。您可以增量迁移，根据需要接管资源，同时为其他资源保留现有 IaC 工具。

ACK 也支持只读资源。对于由其他团队或工具管理、您希望引用但不希望修改的资源，可以将接管与 `retain` 删除策略结合使用，并授予 IAM 只读权限。这使得应用程序可以通过 Kubernetes API 发现共享基础设施（VPC、IAM 角色、KMS 密钥），而无需承担修改风险。

有关资源接管的更多信息，请参阅 the section called “ACK 概念”。

删除策略

删除策略决定了当您删除 Kubernetes 资源时，相应 Amazon 资源的处置方式。根据资源生命周期和运营需求选择合适的策略。

删除（默认）

当您删除 Kubernetes 资源时，相应的 Amazon 也将删除。这样可以保持集群与 Amazon 之间的一致性，确保资源不会累积。

何时使用删除：

- 开发和测试环境中的清理操作至关重要时
- 临时资源与应用程序生命周期（测试数据库、临时存储桶）相关联时
- 涉及不应超过应用程序生命周期的资源时（SQS 队列、ElastiCache 集群）
- 成本优化：自动清理未使用的资源
- 涉及通过 GitOps 管理的环境时，其中从 Git 中移除资源即应删除基础设施

默认的删除策略与 Kubernetes 的声明式模型一致：集群中存在的内容与 Amazon 中存在的内容相匹配。

保留

删除 Kubernetes 资源时，相应的 Amazon 资源将保留。这样可以保护关键数据，并使资源的存在时间超过其 Kubernetes 表示形式。

何时使用保留：

- 生产数据库包含关键数据且此类数据必须在集群更改后保留时

- 长期存储桶需符合合规性或审计要求时
- 多个应用程序或团队使用共享资源时
- 资源正在迁移到不同管理工具时
- 希望保留基础设施的灾难恢复场景中
- 涉及具有复杂依赖项、需要谨慎停用的资源时

```
apiVersion: rds.services.k8s.aws/v1alpha1
kind: DBInstance
metadata:
  name: production-db
  annotations:
    services.k8s.aws/deletion-policy: "retain"
spec:
  dbInstanceIdentifier: prod-db
  # ... configuration
```

> **Important**
>
> 被保留的资源将继续产生 Amazon 费用，如果您不再需要这些资源，则必须手动将其从 Amazon 中删除。使用资源标记来跟踪保留的资源，以便后续清理。

有关删除策略的更多信息，请参阅 the section called “ACK 概念”。

上游文档

有关使用 ACK 的详细信息，请参阅下列文档：

- ACK 使用指南：创建和管理资源
- ACK API 参考：所有服务的完整 API 文档
- ACK 文档：全面的用户文档

后续步骤

- the section called “配置权限”：配置 IAM 权限和多账户模式
- the section called “ACK 概念”：了解 ACK 概念和资源生命周期

- the section called “问题排查”：排查 ACK 问题
- the section called “使用 Argo CD”：通过 GitOps 部署 ACK 资源
- the section called “kro 概念”：将 ACK 资源组合成更高级别的抽象

排查 ACK 功能问题

本主题将提供针对适用于 ACK 的 EKS 功能的问题排查指南，包括功能运行状况检查、资源状态验证和 IAM 权限问题。

> **Note**
>
> EKS 功能完全托管，可在您的集群之外运行。您无权访问控制器日志或控制器命名空间。问题排查侧重于功能运行状况、资源状态和 IAM 配置。

功能处于活动状态，但无法创建资源

如果 ACK 功能显示为 `ACTIVE` 状态但无法在 Amazon 中创建资源，请检查功能运行状况、资源状态和 IAM 权限。

检查功能运行状况：

您可以在 EKS 控制台或使用 Amazon CLI 查看功能运行状况和状态问题。

控制台：

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称。
3. 选择可观测性选项卡。
4. 选择监控集群。
5. 选择功能选项卡，查看所有功能的运行状况和状态。

Amazon CLI：

```
# View capability status and health
aws eks describe-capability \
```

```
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack

# Look for issues in the health section
```

常见原因：

- 缺少 IAM 权限：功能角色缺少 Amazon 服务权限
- 命名空间错误：在未配置正确 IAMRoleSelector 的命名空间中创建了资源
- 资源规范无效：检查资源状态条件是否存在验证错误
- API 节流：达到了 Amazon API 速率限制
- 准入 Webhook：准入 Webhook 阻止了控制器修补资源状态

检查资源状态：

```
# Describe the resource to see conditions and events
kubectl describe bucket my-bucket -n default

# Look for status conditions
kubectl get bucket my-bucket -n default -o jsonpath='{.status.conditions}'

# View resource events
kubectl get events --field-selector involvedObject.name=my-bucket -n default
```

验证 IAM 权限：

```
# View the Capability Role's policies
aws iam list-attached-role-policies --role-name my-ack-capability-role
aws iam list-role-policies --role-name my-ack-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-ack-capability-role --policy-name policy-name
```

资源已在 Amazon 中创建，但未在 Kubernetes 中显示

ACK 仅跟踪通过 Kubernetes 清单创建的资源。要使用 ACK 管理现有 Amazon 资源，请使用接管功能。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: existing-bucket
  annotations:
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
  name: my-existing-bucket-name
```

有关资源接管的更多信息，请参阅 the section called “ACK 概念”。

无法创建跨账户资源

如果使用 IAM 角色选择器时，无法在目标 Amazon 账户中创建资源，请验证信任关系和 IAMRoleSelector 配置。

验证信任关系：

```
# Check the trust policy in the target account role
aws iam get-role --role-name cross-account-ack-role --query
 'Role.AssumeRolePolicyDocument'
```

信任策略必须允许源账户的功能角色代入它。

确认 IAMRoleSelector 配置：

```
# List IAMRoleSelectors (cluster-scoped)
kubectl get iamroleselector

# Describe specific selector
kubectl describe iamroleselector my-selector
```

验证命名空间对应关系：

IAMRoleSelectors 是集群范围内的资源，但针对特定的命名空间。请确保 ACK 资源位于与 IAMRoleSelector 的命名空间选择器匹配的命名空间：

```
# Check resource namespace
kubectl get bucket my-cross-account-bucket -n production
```

```
# List all IAMRoleSelectors (cluster-scoped)
kubectl get iamroleselector

# Check which namespace the selector targets
kubectl get iamroleselector my-selector -o jsonpath='{.spec.namespaceSelector}'
```

检查 IAMRoleSelected 条件：

通过检查 ACK.IAMRoleSelected 条件，验证 IAMRoleSelector 是否已与资源成功匹配：

```
# Check if IAMRoleSelector was matched
kubectl get bucket my-cross-account-bucket -n production -o
 jsonpath='{.status.conditions[?(@.type=="ACK.IAMRoleSelected")]}'
```

如果条件为 False 或缺失，则说明 IAMRoleSelector 的命名空间选择器与资源所在的命名空间不匹配。验证选择器的 namespaceSelector 是否与资源的命名空间标签匹配。

检查功能角色权限：

功能角色需要目标账户角色的 sts:AssumeRole 权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::[.replaceable]`444455556666`:role/[.replaceable]`cross-
account-ack-role`"
    }
  ]
}
```

有关跨账户配置的详细信息，请参阅the section called “配置权限”。

后续步骤

- the section called “注意事项”：ACK 注意事项和最佳实践
- the section called “配置权限”：配置 IAM 权限和多账户模式
- the section called “ACK 概念”：了解 ACK 概念和资源生命周期

- the section called “问题排查功能”：一般功能问题排查指导

适用于 ACK 的 EKS 功能与自主管理型 ACK 功能的比较

适用于 ACK 的 EKS 功能提供了与自主管理型 ACK 控制器相同的功能，且具备显著的运营优势。有关 EKS 功能与自主管理型解决方案的总体比较，请参阅 the section called “注意事项”。本主题将重点介绍 ACK 特有的差异。

与上游 ACK 的区别

适用于 ACK 的 EKS 功能基于上游 ACK 控制器，但在 IAM 集成方面有所不同。

IAM 功能角色：该功能将使用专用 IAM 角色，其信任策略允许 `capabilities.eks.amazonaws.com` 服务主体，而不允许 IRSA（用于服务账户的 IAM 角色）。您可以将 IAM 策略直接附加到功能角色，而无需创建 Kubernetes 服务账户或为其添加注释，也无需配置 OIDC 提供者。对于生产使用案例，最佳实践是使用 `IAMRoleSelector` 来配置服务权限。有关更多信息，请参阅the section called “配置权限”。

资源兼容性：ACK 自定义资源的工作方式与上游 ACK 完全相同，您无需更改 ACK 资源 YAML 文件。该功能使用同一 Kubernetes API 和 CRD，因此 `kubectl` 等工具的使用方式将保持不变。支持上游 ACK 中所有已正式发布的控制器和资源。

有关完整的 ACK 文档及针对具体服务的指南，请参阅 ACK 文档。

迁移路径

您可以从自主管理型 ACK 迁移到托管功能，且停机时间为零：

1. 更新自主管理型 ACK 控制器，使其使用 `kube-system` 实现主节点选举租约，例如：

```
helm upgrade --install ack-s3-controller \
  oci://public.ecr.aws/aws-controllers-k8s/s3-chart \
  --namespace ack-system \
  --set leaderElection.namespace=kube-system
```

   这会将控制器的租约移至 `kube-system`，从而允许托管功能与其协调。
2. 在集群上创建 ACK 功能（请参阅the section called “创建 ACK 功能”）
3. 托管功能可识别出由 ACK 托管的 Amazon 现有资源，并接管协调工作
4. 逐步缩减或移除自主管理型控制器部署：

```
helm uninstall ack-s3-controller --namespace ack-system
```

此方法允许两个控制器在迁移期间安全共存。托管功能会自动接管先前由自主管理型控制器托管的资源，确保持续协调且不发生冲突。

后续步骤

- the section called “创建 ACK 功能”：创建 ACK 功能资源
- the section called “ACK 概念”：了解 ACK 概念和资源生命周期
- the section called “配置权限”：配置 IAM 和权限

使用 Argo CD 持续部署

Argo CD 是一款适用于 Kubernetes 的声明式 GitOps 持续交付工具。借助 Argo CD，您可以跨多个集群和环境自动部署应用程序并管理其生命周期。Argo CD 支持多种来源类型，包括 Git 存储库、Helm 注册表（HTTP 和 OCI）和 OCI 映像，为具有不同安全和合规要求的组织提供了灵活性。

得益于 EKS 功能，Argo CD 完全由 Amazon 托管，无需您在集群上安装、维护和扩展 Argo CD 控制器及其依赖项。

Argo CD 的工作原理

Argo CD 遵循 GitOps 模式，其中您的应用程序源（Git 存储库、Helm 注册表或 OCI 映像）是用于定义所需应用程序状态的事实来源。在创建 Argo CD `Application` 资源时，需要指定包含应用程序清单的源以及目标 Kubernetes 集群和命名空间。Argo CD 持续监控集群中的源状态和实时状态，并且自动同步所有更改以确保集群状态与所需状态保持一致。

> **Note**
>
> 借助适用于 Argo CD 的 EKS 功能，Argo CD 软件可在 Amazon 控制面板上运行，而不是在 Worker 节点上运行。这意味着您的 Worker 节点不需要直接访问 Git 存储库或 Helm 注册表：此功能可处理来自 Amazon 账户的源访问。

Argo CD 提供三种主要资源类型：

- Application：定义从 Git 存储库到目标集群的部署

- ApplicationSet：根据用于多集群部署的模板生成多个应用程序
- AppProject：为应用程序提供逻辑分组和访问权限控制

示例：创建 Argo CD 应用程序

以下示例说明了如何创建 Argo CD `Application` 资源：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook
  destination:
    name: my-cluster
    namespace: guestbook
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Argo CD 的优势

Argo CD 实施 GitOps 工作流程，在该工作流程中，您可以在 Git 存储库中定义应用程序配置，然后 Argo CD 会自动同步您的应用程序，以便与预期状态保持一致。这种以 Git 为中心的方法可提供对所有更改的完整审计跟踪记录，实现轻松回滚，并且可以自然地与您现有的代码审查和批准流程集成。Argo CD 会自动检测和协调 Git 中所需状态与集群中实际状态之间的偏差，确保您的部署与声明的配置保持一致。

借助 Argo CD，您可以从单个 Argo CD 实例跨多个集群部署和管理应用程序，从而简化多集群和多区域环境中的操作。Argo CD 用户界面提供可视化和监控功能，允许您查看应用程序的部署状态、运行状况和历史记录。用户界面与 Amazon Identity Center（原 Amazon SSO）集成，可实现无缝身份验证和授权，使您能够使用现有的身份管理基础设施控制访问权限。

作为 EKS 托管功能的一部分，Argo CD 完全由 Amazon 管理，无需安装、配置和维护 Argo CD 基础设施。Amazon 负责扩展、修补和操作管理，让您的团队能够专注于应用程序交付而不是工具维护。

与 Amazon Identity Center 集成

EKS 托管功能提供 Argo CD 与 Amazon Identity Center 之间的直接集成，为您的用户提供无缝身份验证和授权。启用 Argo CD 功能后，您可以将 Amazon Identity Center 集成配置为将 Identity Center 组和用户映射到 Argo CD RBAC 角色，从而控制谁可以访问和管理 Argo CD 中的应用程序。

与其他 EKS 托管功能集成

Argo CD 可与其他 EKS 托管功能集成。

- Amazon Controllers for Kubernetes（ACK）：使用 Argo CD 跨多个集群管理 ACK 资源的部署，为 Amazon 基础设施启用 GitOps 工作流程。
- kro（Kube Resource Orchestrator）：使用 Argo CD 在多个集群中部署 kro 组合，从而在整个 Kubernetes 资产中实现一致的资源组合。

开始使用 Argo CD

要开始使用适用于 Argo CD 的 EKS 功能，请执行以下操作：

1. 创建和配置 IAM 功能角色，该角色具有必要权限，能让 Argo CD 访问您的 Git 存储库和管理应用程序。
2. 通过 Amazon 控制台、Amazon CLI 或偏好的基础设施即代码工具，在 EKS 集群上创建 Argo CD 功能资源。
3. 配置存储库访问权限并注册集群，进行应用程序部署。
4. 创建 Application 资源，从 Git 存储库部署应用程序。

创建 Argo CD 功能

本主题旨在介绍如何在 Amazon EKS 集群上创建 Argo CD 功能。

先决条件

创建 Argo CD 功能之前，确保满足以下条件：

- 运行受支持 Kubernetes 版本的现有 Amazon EKS 集群（标准支持与扩展支持范围内的所有版本均适用）
- Amazon Identity Center 已配置：此为 Argo CD 身份验证所需（不支持本地用户）

- 具有 Argo CD 操作权限的 IAM 功能角色
- 可在 EKS 集群上创建功能资源的充足 IAM 权限
- 已配置 `kubectl` 以与集群通信
- （可选）已安装 Argo CD CLI，以便更便捷地进行集群与存储库管理
- （适用于 CLI/eksctl）已安装并配置相应 CLI 工具

有关如何创建 IAM 功能角色的说明，请参阅 the section called “功能 IAM 角色”。有关设置 Identity Center 的信息，请参阅 Amazon Identity Center 入门。

> **Important**
>
> 您提供的 IAM 功能角色，将决定 Argo CD 可以访问的 Amazon 资源范围。这包括通过 CodeConnections 访问 Git 存储库，以及访问 Secrets Manager 中的密钥。有关创建具备最低权限的合适角色的操作指引，请参阅 the section called “功能 IAM 角色”和 the section called “EKS 功能的注意事项”。

选择工具

您可以使用Amazon Web Services 管理控制台、Amazon CLI 或 eksctl 创建 Argo CD 功能：

- the section called “控制台”：使用控制台获得引导式体验
- the section called “ Amazon CLI”：使用 Amazon CLI 进行脚本编写和自动化
- the section called “eksctl”：使用 eksctl 获得 Kubernetes 原生体验

创建 Argo CD 功能时会执行的操作

创建 Argo CD 功能时：

1. EKS 会创建 Argo CD 功能服务，并配置该服务来监控和管理集群内的资源
2. 自定义资源定义（CRD）将被安装到集群中
3. 该功能会代入您提供的 IAM 功能角色
4. Argo CD 开始监听其自定义资源
5. 功能状态从 `CREATING` 更改为 `ACTIVE`

6. 可通过对应的端点访问 Argo CD UI

功能进入运行状态后，即可在集群中创建 Argo CD 应用程序，基于 Git 存储库完成部署操作。

后续步骤

创建 Argo CD 功能后：

- the section called “使用 Argo CD”：配置存储库访问权限、注册目标集群及创建应用程序
- the section called “Argo CD 概念”：了解 GitOps 原则、同步策略及多集群模式
- the section called “Argo CD 注意事项”：探索多集群架构模式及高级配置

使用控制台创建 Argo CD 功能

本主题将介绍如何使用 Amazon Web Services 管理控制台创建 Argo CD 功能。

先决条件

- 已配置 Amazon Identity Center：Argo CD 需要使用 Amazon Identity Center 进行身份验证。不支持本地用户。如果您尚未设置 Amazon Identity Center，请参阅开始使用 Amazon Identity Center 以创建 Identity Center 实例，并参阅添加用户和添加组以创建用于访问 Argo CD 的用户和组。

创建 Argo CD 功能

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称，打开集群详细信息页面。
3. 选择功能选项卡。
4. 在左侧导航栏中，选择 Argo CD。
5. 选择创建 Argo CD 功能。
6. 对于 IAM 功能角色：
   - 如果已有 IAM 功能角色，请从下拉列表中选择该角色
   - 如需创建角色，请选择创建 Argo CD 角色

     此操作会在新选项卡中打开 IAM 控制台，其中包含预填充的信任策略和 Secrets Manager 的完全读取权限。默认情况下不会添加其他权限，但您可以根据需要自行添加。如果计划使用 CodeCommit 存储库或其他 Amazon 服务，请在创建角色之前添加相应的权限。

创建角色后，返回 EKS 控制台，系统将自动选择该角色。

> **Note**
>
> 如果计划使用与 Amazon Secrets Manager 或 Amazon CodeConnections 的可选集成，则需要向该角色添加权限。有关 IAM 策略示例和配置指南，请参阅the section called “ Amazon Secrets Manager”和the section called “ Amazon CodeConnections”。

7. 配置 Amazon Identity Center 集成：
   a. 选择启用 Amazon Identity Center 集成。
   b. 从下拉列表中选择 Identity Center 实例。
   c. 通过将用户或组分配给 Argo CD 角色（ADMIN、EDITOR 或 VIEWER）来配置 RBAC 的角色映射
8. 选择创建。

功能创建过程随即开始。

验证功能是否处于活动状态

1. 在功能选项卡上，查看 Argo CD 功能状态。
2. 等待状态从 `CREATING` 更改为 `ACTIVE`。
3. 变为活动状态后，该功能即可使用。

有关功能状态和问题排查的信息，请参阅the section called “使用功能”。

访问 Argo CD 用户界面

待该功能处于活动状态后，可以访问 Argo CD 用户界面：

1. 在 Argo CD 功能页面上，选择打开 Argo CD 用户界面。
2. Argo CD 用户界面将在新的浏览器选项卡中打开。
3. 现在，您可以通过用户界面创建应用程序和管理部署。

后续步骤

- the section called “使用 Argo CD”：配置存储库、注册集群和创建应用程序

- the section called “Argo CD 注意事项”：多集群架构和高级配置
- the section called “使用功能”：管理 Argo CD 功能资源

使用 Amazon CLI 创建 Argo CD 功能

本主题将介绍如何使用 Amazon CLI 创建 Argo CD 功能。

先决条件

- Amazon CLI：版本 `2.12.3` 或更高版本。要检查版本，请运行 `aws --version`。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的安装。
- **`kubectl`**：用于与 Kubernetes 集群结合使用的命令行工具。有关更多信息，请参阅 the section called “设置 kubectl 和 eksctl”。
- 已配置 Amazon Identity Center：Argo CD 需要使用 Amazon Identity Center 进行身份验证。不支持本地用户。如果您尚未设置 Amazon Identity Center，请参阅开始使用 Amazon Identity Center 以创建 Identity Center 实例，并参阅添加用户和添加组以创建用于访问 Argo CD 的用户和组。

步骤 1：创建 IAM 功能角色

创建信任策略文件：

```
cat > argocd-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

创建 IAM 角色：

```
aws iam create-role \
  --role-name ArgoCDCapabilityRole \
  --assume-role-policy-document file://argocd-trust-policy.json
```

> **Note**
>
> 如果计划使用与 Amazon Secrets Manager 或 Amazon CodeConnections 的可选集成，则需要向该角色添加权限。有关 IAM 策略示例和配置指南，请参阅the section called “ Amazon Secrets Manager”和the section called “ Amazon CodeConnections”。

步骤 2：创建 Argo CD 功能

在集群上创建 Argo CD 功能资源。

首先，为 Identity Center 配置设置环境变量：

```
# Get your Identity Center instance ARN (replace region if your IDC instance is in a
 different region)
export IDC_INSTANCE_ARN=$(aws sso-admin list-instances --region [.replaceable]`region`
 --query 'Instances[0].InstanceArn' --output text)

# Get a user ID for RBAC mapping (replace with your username and region if needed)
export IDC_USER_ID=$(aws identitystore list-users \
  --region [.replaceable]`region` \
  --identity-store-id $(aws sso-admin list-instances --region [.replaceable]`region` --
query 'Instances[0].IdentityStoreId' --output text) \
  --query 'Users[?UserName==`your-username`].UserId' --output text)

echo "IDC_INSTANCE_ARN=$IDC_INSTANCE_ARN"
echo "IDC_USER_ID=$IDC_USER_ID"
```

创建集成了 Identity Center 的功能。将 *region-code* 替换为集群所在的 Amazon 区域，并将 *my-cluster* 替换为集群名称：

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --type ARGOCD \
```

```
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ArgoCDCapabilityRole \
  --delete-propagation-policy RETAIN \
  --configuration '{
    "argoCd": {
      "awsIdc": {
        "idcInstanceArn": "'$IDC_INSTANCE_ARN'",
        "idcRegion": "'[.replaceable]`region-code`'"
      },
      "rbacRoleMappings": [{
        "role": "ADMIN",
        "identities": [{
          "id": "'$IDC_USER_ID'",
          "type": "SSO_USER"
        }]
      }]
    }
  }'
```

命令会立即返回，但是由于 EKS 正在创建所需的功能基础设施和组件，该功能需要一些时间才能变为活动状态。在创建集群时，EKS 将在集群中安装与此功能相关的 Kubernetes 自定义资源定义。

> **Note**
>
> 如果收到集群不存在或您没有权限的错误消息，请验证：
>
> - 集群名称是否正确
> - Amazon CLI 是否针对正确的区域进行配置
> - 您是否拥有所需的 IAM 权限

步骤 3：验证功能是否处于活动状态

等待功能变为活动状态。将 *region-code* 替换为集群所在的 Amazon 区域，并将 *my-cluster* 替换为集群名称。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.status' \
```

```
  --output text
```

当状态显示为 ACTIVE 时，表示功能已准备就绪。在该状态变为 ACTIVE 之前，请勿继续执行下一步。

您也可以查看完整功能详细信息：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd
```

步骤 4：验证自定义资源是否可用

待该功能处于活动状态后，验证 Argo CD 自定义资源是否已在集群中正常可用：

```
kubectl api-resources | grep argoproj.io
```

您应该会看到列出的 Application 和 ApplicationSet 资源类型。

后续步骤

- the section called “使用 Argo CD”：配置存储库、注册集群和创建应用程序
- the section called “Argo CD 注意事项”：多集群架构和高级配置
- the section called “使用功能”：管理 Argo CD 功能资源

使用 eksctl 创建 Argo CD 功能

本主题将介绍如何使用 eksctl 创建 Argo CD 功能。

> **Note**
>
> 以下步骤需要 eksctl 版本 0.220.0 或更高版本。要检查版本，请运行 eksctl version。

步骤 1：创建 IAM 功能角色

创建信任策略文件：

```
cat > argocd-trust-policy.json << 'EOF'
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

创建 IAM 角色：

```
aws iam create-role \
  --role-name ArgoCDCapabilityRole \
  --assume-role-policy-document file://argocd-trust-policy.json
```

> **Note**
>
> 对于此基本设置，无需其他 IAM 策略。如果计划使用 Secrets Manager 管理存储库凭证或使用 CodeConnections，则需为此角色添加相应权限。有关 IAM 策略示例和配置指南，请参阅the section called “ Amazon Secrets Manager”和the section called “ Amazon CodeConnections”。

步骤 2：获取 Amazon Identity Center 配置

获取 Identity Center 实例 ARN 和用户 ID，用于 RBAC 配置：

```
# Get your Identity Center instance ARN
aws sso-admin list-instances --query 'Instances[0].InstanceArn' --output text

# Get your Identity Center region
aws sso-admin list-instances --query 'Instances[0].IdentityStoreId' --output text | cut
 -d'/' -f1
```

```
# Get a user ID for admin access (replace 'your-username' with your Identity Center
 username)
aws identitystore list-users \
  --identity-store-id $(aws sso-admin list-instances --query
 'Instances[0].IdentityStoreId' --output text) \
  --query 'Users[?UserName==`your-username`].UserId' --output text
```

记下这些值，下一步将会用到。

步骤 3：创建 eksctl 配置文件

使用以下内容创建名为 `argocd-capability.yaml` 的文件。将占位符值替换为集群名称、区域、IAM 角色 ARN、Identity Center 实例 ARN、Identity Center 区域和用户 ID：

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code

capabilities:
  - name: my-argocd
    type: ARGOCD
    roleArn: arn:aws:iam::[.replaceable]111122223333:role/ArgoCDCapabilityRole
    configuration:
      argocd:
        awsIdc:
          idcInstanceArn: arn:aws:sso:::instance/ssoins-123abc
          idcRegion: idc-region-code
        rbacRoleMappings:
          - role: ADMIN
            identities:
              - id: 38414300-1041-708a-01af-5422d6091e34
                type: SSO_USER
```

> **Note**
>
> 您可以向 RBAC 映射添加多个用户或组。对于组，请使用 `type: SSO_GROUP` 并提供组 ID。可用角色包括 `ADMIN`、`EDITOR` 和 `VIEWER`。

步骤 4：创建 Argo CD 功能

应用配置文件：

```
eksctl create capability -f argocd-capability.yaml
```

命令会立即返回，但该功能需要一些时间才能变为活动状态。

步骤 5：验证功能是否处于活动状态

检查功能状态。将 *region-code* 替换为您的集群所在的 Amazon 区域，并将 *my-cluster* 替换为您的集群的名称。

```
eksctl get capability \
  --region region-code \
  --cluster my-cluster \
  --name my-argocd
```

当状态显示为 ACTIVE 时，表示功能已准备就绪。

步骤 6：验证自定义资源是否可用

待该功能处于活动状态后，验证 Argo CD 自定义资源是否已在集群中正常可用：

```
kubectl api-resources | grep argoproj.io
```

您应该会看到列出的 Application 和 ApplicationSet 资源类型。

后续步骤

- the section called “使用 Argo CD”：了解如何创建和管理 Argo CD 应用程序
- the section called “Argo CD 注意事项”：配置 SSO 和多集群访问权限
- the section called “使用功能”：管理 Argo CD 功能资源

Argo CD 概念

Argo CD 通过将 Git 视为应用程序部署的单一事实来源，实现了 GitOps。本主题将介绍一个实际示例，然后解释使用适用于 Argo CD 的 EKS 功能时需要了解的核心概念。

开始使用 Argo CD

创建 Argo CD 功能（请参阅the section called “创建 Argo CD 功能”）后，即可开始部署应用程序。本示例将演示如何注册集群和创建应用程序。

步骤 1：设置

注册集群（必需）

注册要在其中部署应用程序的集群。在本示例中，我们将注册运行 Argo CD 所在的同一集群（可以使用 `in-cluster` 作为名称，以便与大多数 Argo CD 示例兼容）：

```
# Get your cluster ARN
CLUSTER_ARN=$(aws eks describe-cluster \
  --name my-cluster \
  --query 'cluster.arn' \
  --output text)

# Register the cluster using Argo CD CLI
argocd cluster add $CLUSTER_ARN \
  --aws-cluster-name $CLUSTER_ARN \
  --name in-cluster \
  --project default
```

> **Note**
>
> 有关配置 Argo CD CLI 以使其与 EKS 中的 Argo CD 功能结合使用的信息，请参阅the section called “将 Argo CD CLI 与托管功能结合使用”。

或者，您也可以使用 Kubernetes Secret 注册集群（有关详细信息，请参阅the section called “注册集群”）。

配置存储库访问权限（可选）

此示例使用公有 GitHub 存储库，因此无需配置存储库。对于私有存储库，可使用 Amazon Secrets Manager、CodeConnection 或 Kubernetes Secret 配置访问权限（有关详细信息，请参阅the section called “配置存储库”）。

对于 Amazon 服务（用于 Helm 图表的 ECR、CodeConnections 和 CodeCommit），您可以直接在 Application 资源中引用它们，而无需创建存储库。功能角色必须具有必要的 IAM 权限。有关详细信息，请参阅 the section called “配置存储库”。

步骤 2：创建应用程序

在 my-app.yaml 中创建以下应用程序清单：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook
  destination:
    name: in-cluster
    namespace: guestbook
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

应用应用程序：

```
kubectl apply -f my-app.yaml
```

应用此应用程序后，Argo CD 会执行以下操作：1. 将应用程序从 Git 同步到集群（初始部署）2. 监控 Git 存储库中的更改 3. 自动将后续更改同步到集群 4. 检测并纠正任何与预期状态的偏差 5. 在用户界面中提供运行状况状态和同步历史记录

查看应用程序状态：

```
kubectl get application guestbook -n argocd
```

您也可以使用 Argo CD CLI (`argocd app get guestbook`) 或 Argo CD 用户界面（可通过 EKS 控制台中集群的“功能”选项卡访问）查看应用程序。

> **Note**
>
> 在 `destination.name` 中使用集群名称（注册集群时使用的名称）。托管功能不支持本地的集群内默认值 (`kubernetes.default.svc`)。

核心概念

GitOps 原则和源类型

Argo CD 实现了 GitOps，应用程序源是部署的单一事实来源：

- 声明式：使用 YAML 清单、Helm 图表或 Kustomize 叠加层声明预期状态
- 版本控制：每次更改都有完整的审计跟踪记录
- 自动化：Argo CD 持续监控源并自动同步更改
- 自我修复：检测并纠正预期状态与集群实际状态之间的偏差

支持的源类型：

- Git 存储库：GitHub、GitLab、Bitbucket、CodeCommit（HTTPS、SSH 或 CodeConnections）
- Helm 注册表：HTTP 注册表（如 `https://aws.github.io/eks-charts`）和 OCI 注册表（如 `public.ecr.aws`）
- OCI 映像：包含清单或 Helm 图表的容器映像（如 `oci://registry-1.docker.io/user/my-app`）

得益于这种灵活性，组织能够选择符合其安全性和合规性要求的源。例如，限制集群访问 Git 的组织可以使用 ECR 存放 Helm 图表或 OCI 映像。

有关更多信息，请参阅 Argo CD 文档中的 Application Sources。

同步与协调

Argo CD 会持续监控源和集群，以检测并纠正偏差：

1. 轮询源以查找变更（默认：每 6 分钟一次）
2. 将预期状态与集群实际状态进行比较

3. 将应用程序标记为 Synced 或 OutOfSync
4. 自动同步更改（如果已配置）或等待手动审批
5. 同步后监控资源运行状况

同步波次使用注释控制资源创建顺序：

```
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "0"  # Default if not specified
```

按波次顺序应用资源（数字小的先执行，例如 -1 等负数）。这样您就可以先创建命名空间（波次 -1）等依赖项，再进行部署（波次 0）。

自我修复会自动还原手动更改：

```
spec:
  syncPolicy:
    automated:
      selfHeal: true
```

> **Note**
>
> 托管功能使用基于注释的资源跟踪（而不是基于标签），以便更好地兼容 Kubernetes 规范和其他工具。

有关同步阶段、钩子和高级模式的详细信息，请参阅 Argo CD 同步文档。

应用程序运行状况

Argo CD 会监控应用程序中所有资源的运行状况：

运行状况：* 良好：所有资源均按预期运行 * 正在处理：正在创建或更新资源 * 已降级：部分资源运行状况不佳 [容器组（pod）崩溃、任务失败] * 已暂停：应用程序故意暂停 * 缺失：集群中缺少 Git 中定义的资源

Argo CD 内置有针对常见 Kubernetes 资源（Deployment、StatefulSets、Jobs 等）的运行状况检查，并且支持对 CRD 进行自定义运行状况检查。

应用程序运行状况由所有资源决定，如果任何资源为 Degraded，则应用程序为 Degraded。

有关更多信息，请参阅 Argo CD 文档中的 Resource Health。

多集群模式

Argo CD 支持两种主要的部署模式：

轴辐式：在专用的管理集群（部署到多个工作负载集群）上运行 Argo CD：* 集中控制和可见性 * 在所有集群之间保持策略一致 * 只需一个 Argo CD 实例即可进行管理 * 实行控制面板与工作负载之间的清晰分离

单集群：在每个集群上运行 Argo CD，仅管理该集群的应用程序：* 集群分离（一个故障不会影响其他集群）* 联网更简单（无需跨集群通信）* 初始设置更轻松（无需注册集群）

对于管理许多集群的平台团队，选择轴辐式；对于独立的团队或必须完全隔离的集群，选择单集群。

有关多集群配置的详细信息，请参阅 the section called “Argo CD 注意事项”。

Projects

项目为应用程序提供逻辑分组和访问权限控制：

- 源限制：限制可以使用哪些 Git 存储库
- 目标限制：限制可以指向哪些集群和命名空间
- 资源限制：限制可以部署哪些 Kubernetes 资源类型
- RBAC 集成：将项目映射到 Amazon Identity Center 的用户和组 ID

所有应用程序同属一个项目。如果未指定，将使用 default 项目（没有任何限制）。对于生产用途，请创建具有适当限制的项目。

有关项目配置和 RBAC 模式，请参阅the section called “配置权限”。

存储库组织方式

多数团队采用基于目录的组织方式，配合 Kustomize 叠加层或 Helm 值文件来管理不同环境：

```
my-app/
### base/
```

```
#   ### deployment.yaml
#   ### service.yaml
### overlays/
    ### dev/
    #   ### kustomization.yaml
    ### staging/
    #   ### kustomization.yaml
    ### prod/
        ### kustomization.yaml
```

这种方法将所有环境配置集中于单个存储库，同时保持了灵活性与清晰度。

有关详细的存储库结构模式和最佳实践，请参阅 Argo CD 最佳实践文档。

同步选项

使用常用选项微调同步行为：

- `CreateNamespace=true`：自动创建目标命名空间
- `ServerSideApply=true`：使用服务器端应用程序以更好地解决冲突
- `SkipDryRunOnMissingResource=true`：当 CRD 尚不存在时跳过试运行（对 kro 实例非常有用）

```
spec:
  syncPolicy:
    syncOptions:
    - CreateNamespace=true
    - ServerSideApply=true
```

有关完整的同步选项列表，请参阅 Argo CD 同步选项文档。

后续步骤

- the section called “配置存储库”：配置 Git 存储库访问权限
- the section called “注册集群”：注册用于部署的目标集群
- the section called “创建应用程序”：创建第一个应用程序
- the section called “Argo CD 注意事项”：EKS 特定模式、Identity Center 集成和多集群配置
- Argo CD 文档：全面的 Argo CD 文档，包括同步钩子、运行状况检查和高级模式

配置 Argo CD 权限

Argo CD 托管功能与 Amazon Identity Center 集成以进行身份验证，并使用内置的 RBAC 角色进行授权。本主题将介绍如何为用户和团队配置权限。

Argo CD 权限的工作原理

Argo CD 功能通过 Amazon 进行身份验证，并提供三个内置的 RBAC 角色进行授权。

当用户访问 Argo CD 时：

1. 他们使用 Amazon Identity Center（可以与企业身份提供者联合）进行身份验证
2. Amazon Identity Center 向 Argo CD 提供用户和组信息
3. Argo CD 会根据配置将用户和组映射到 RBAC 角色
4. 用户只能看到他们有权访问的应用程序和资源

内置 RBAC 角色

Argo CD 功能提供三个内置角色，您需要将其映射到 Amazon Identity Center 的用户和组。

ADMIN

可以完全访问所有应用程序和设置：

- 创建、更新和删除应用程序和 ApplicationSets
- 管理 Argo CD 配置
- 注册和管理部署目标集群
- 配置存储库访问权限
- 管理项目
- 查看所有应用程序状态和历史记录

EDITOR

可以创建和修改应用程序，但无法更改 Argo CD 设置：

- 创建和更新应用程序和 ApplicationSets
- 同步和刷新应用程序

- 查看应用程序状态和历史记录
- 无法删除应用程序
- 无法更改 Argo CD 配置
- 无法管理集群或存储库

VIEWER

可以对应用程序进行只读访问：

- 查看应用程序状态和历史记录
- 查看应用程序清单和资源
- 无法进行任何更改
- 无法同步或刷新应用程序

配置角色映射

创建或更新功能时，将 Amazon Identity Center 用户和组映射到 Argo CD 角色。

示例角色映射：

```
{
  "rbacRoleMapping": {
    "ADMIN": ["AdminGroup", "alice@example.com"],
    "EDITOR": ["DeveloperGroup", "DevOpsTeam"],
    "VIEWER": ["ReadOnlyGroup", "bob@example.com"]
  }
}
```

> **Note**
>
> 角色名称区分大小写，必须为大写（ADMIN、EDITOR、VIEWER）。

> **Important**
>
> EKS 功能与 Amazon Identity Center 集成后，每个 Argo CD 功能可支持最多 1000 个身份。身份可以是用户或组。

更新角色映射：

```
aws eksfe update-capability \
  --region us-east-1 \
  --cluster-name cluster \
  --capability-name capname \
  --endpoint "https://eks.ap-northeast-2.amazonaws.com" \
  --role-arn "arn:aws:iam::[.replaceable]111122223333:role/
[.replaceable]`EKSCapabilityRole`" \
  --configuration '{
    "argoCd": {
      "rbacRoleMappings": {
        "addOrUpdateRoleMappings": [
          {
            "role": "ADMIN",
            "identities": [
              { "id": "686103e0-f051-7068-b225-e6392b959d9e", "type": "SSO_USER" }
            ]
          }
        ]
      }
    }
  }'
```

使用管理员账户

管理员账户专为初始设置和管理任务（例如注册集群和配置存储库）而设计。

何时适合使用管理员账户：

- 初始功能设置和配置
- 单人开发或快速演示
- 管理任务（注册集群、配置存储库、创建项目）

管理员账户最佳实践：

- 请勿将账户令牌提交给版本控制
- 如果令牌暴露，则立即轮换
- 将账户令牌的使用限制在设置和管理任务
- 设置较短的过期时间（最长 12 小时）

- 在任何给定时间只能创建 5 个账户令牌

何时改用基于项目的访问：

- 与多个用户共享的开发环境
- 任何与生产环境相似的环境
- 需要有关操作执行者的审计跟踪记录时
- 需要强制实施资源限制或访问边界时

对于生产环境和多用户场景，请使用基于项目的访问控制，并将专用 RBAC 角色映射到 Amazon Identity Center 组。

基于项目的访问控制

使用 Argo CD 项目（AppProject）为团队提供精细的访问控制和资源隔离。

项目提供以下功能：

- 源限制：限制可以使用哪些 Git 存储库
- 目标限制：限制可以指向哪些集群和命名空间
- 资源限制：限制可以部署哪些 Kubernetes 资源类型
- RBAC 集成：将项目映射到 Amazon Identity Center 组或 Argo CD 角色

团队隔离示例项目：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  description: Team A applications

  # Source restrictions
  sourceRepos:
  - https://github.com/myorg/team-a-apps
```

```
  # Destination restrictions
  destinations:
  - namespace: team-a-*
    server: arn:aws:eks:us-west-2:111122223333:cluster/production

  # Resource restrictions
  clusterResourceWhitelist:
  - group: ''
    kind: Namespace
  namespaceResourceWhitelist:
  - group: 'apps'
    kind: Deployment
  - group: ''
    kind: Service
  - group: ''
    kind: ConfigMap
```

为用户分配项目：

拥有 EDITOR 或 VIEWER 角色的用户可以限制为访问特定项目。ADMIN 用户则可以访问所有项目。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
spec:
  # ... project configuration ...

  # Map Identity Center groups to project roles
  roles:
  - name: developer
    description: Team A developers
    policies:
    - p, proj:team-a:developer, applications, *, team-a/*, allow
    groups:
    - TeamADevelopers

  - name: viewer
    description: Team A viewers
    policies:
    - p, proj:team-a:viewer, applications, get, team-a/*, allow
    groups:
    - TeamAViewers
```

常见权限模式

模式 1：拥有完整访问权限的管理员团队

```
{
  "rbacRoleMapping": {
    "ADMIN": ["PlatformTeam", "SRETeam"]
  }
}
```

模式 2：开发人员可以部署，其他人可以查看

```
{
  "rbacRoleMapping": {
    "ADMIN": ["PlatformTeam"],
    "EDITOR": ["DevelopmentTeam", "DevOpsTeam"],
    "VIEWER": ["AllEmployees"]
  }
}
```

模式 3：通过项目实现团队隔离

1. 将所有开发者映射到 EDITOR 角色
2. 为每个团队创建单独的 AppProjects
3. 使用项目角色，限制访问团队特定应用程序

```
{
  "rbacRoleMapping": {
    "ADMIN": ["PlatformTeam"],
    "EDITOR": ["AllDevelopers"]
  }
}
```

然后创建具有团队特定限制和角色映射的项目。

最佳实践

优先使用用户组而非独立用户：将 Amazon Identity Center 用户组映射到 Argo CD 角色，而不是单个用户，以便于管理。

从最低权限开始：从 VIEWER 访问权限开始，然后根据需要授予 EDITOR 或 ADMIN 权限。

使用项目实现团队隔离：为不同的团队或环境创建单独的 AppProjects 以强制实施边界。

利用 Identity Center 联合身份验证：配置 Amazon Identity Center，以与企业身份提供者联合，从而实现集中式用户管理。

定期审查访问权限：定期审查角色映射和项目分配，以确保适当的访问级别。

限制集群访问权限：请记住，Argo CD RBAC 会控制 Argo CD 资源和操作的访问权限，这与 Kubernetes RBAC 无关。拥有 Argo CD 访问权限的用户可以将应用程序部署到 Argo CD 有权访问的集群。限制 Argo CD 可以访问的集群，并使用项目目标限制来控制应用程序的部署位置。

Amazon 服务权限

要在 Application 资源中直接使用 Amazon 服务（而无需创建存储库资源），请将所需的 IAM 权限附加到功能角色。

用于 Helm 图表的 ECR：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

CodeCommit 存储库：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": "arn:aws:codecommit:region:account-id:repository-name"
    }
  ]
}
```

CodeConnections（GitHub、GitLab、Bitbucket）：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection"
      ],
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-id"
    }
  ]
}
```

有关使用这些集成的详细信息，请参阅the section called “配置存储库”。

后续步骤

- the section called “使用 Argo CD”：了解如何创建应用程序和管理部署
- the section called “Argo CD 概念”：了解 Argo CD 概念，包括项目
- the section called “EKS 功能的注意事项”：查看针对功能的安全最佳实践

使用 Argo CD

通过 Argo CD 功能，您可以在 Git 存储库中定义应用程序，然后 Argo CD 会自动将它们同步到 Kubernetes 集群。此功能支持通过自动偏差检测实现声明性、版本控制的应用程序部署。

先决条件

使用 Argo CD 之前，您需要：

- 已创建具有 Argo CD 功能的 EKS 集群（请参阅the section called “创建 Argo CD 功能”）
- 包含 Kubernetes 清单的 Git 存储库
- 已配置 `kubectl` 以与集群通信

常见任务

以下主题将指导您完成常见的 Argo CD 任务：

the section called “配置存储库”：配置 Argo CD 以使用 Amazon Secrets Manager、Amazon CodeConnections 或 Kubernetes Secrets 访问您的 Git 存储库。

the section called “注册集群”：注册 Argo CD 将在其中部署应用程序的目标集群。

the section called “Projects”：使用适用于多租户环境的项目来组织应用程序并强制实施安全边界。

the section called “创建应用程序”：通过自动或手动同步策略，创建从 Git 存储库部署的应用程序。

the section called “ApplicationSets”：使用 ApplicationSets，通过模板和生成器跨多个环境或集群部署应用程序。

访问 Argo CD 用户界面

通过 EKS 控制台访问 Argo CD 用户界面：

1. 打开 Amazon EKS 控制台
2. 选择您的集群
3. 选择功能选项卡
4. 选择 Argo CD
5. 选择 Open Argo CD UI

用户界面提供可视化应用程序拓扑、同步状态和历史记录、资源运行状况和事件、手动同步控制以及应用程序管理。

上游文档

有关 Argo CD 功能的详细信息：

- Argo CD 文档：完整用户指南

- Application Spec：完整的应用程序 API 参考
- ApplicationSet Guide：ApplicationSet 模式和示例
- Argo CD GitHub：源代码和示例

配置存储库访问权限

在部署应用程序之前，请配置 Argo CD 以访问 Git 存储库和 Helm 图表注册表。Argo CD 支持针对 GitHub、GitLab、Bitbucket、Amazon CodeCommit 和 Amazon ECR 的多种身份验证方法。

> **Note**
>
> 若要与 Amazon 服务直接集成（ECR Helm 图表、CodeCommit 存储库和 CodeConnections），您可以直接在 Application 资源中引用它们，而无需创建存储库配置。功能角色必须具有必要的 IAM 权限。有关详细信息，请参阅 the section called “配置权限”。

先决条件

- 已创建具有 Argo CD 功能的 EKS 集群
- Git 存储库包含 Kubernetes 清单
- 已配置 `kubectl` 以与集群通信

> **Note**
>
> 要在多个存储库中重复使用凭证，可以使用存储库凭证模板（repocreds）。有关更多信息，请参阅 Argo CD 文档中的 Private Repositories。

身份验证方法

方法	用例	所需的 IAM 权限
与 Amazon 服务直接集成		
CodeCommit	直接与 Amazon CodeCommit Git 存储库集成。无需配置存储库。	`codecommit:GitPull`

方法	用例	所需的 IAM 权限
CodeConnections	通过托管身份验证连接到 GitHub、GitLab 或 Bitbucket。需要进行连接设置。	`codeconnections:UseConnection`
ECR Helm 图表	直接与 Amazon ECR 集成以获取 OCI Helm 图表。无需配置存储库。	`ecr:GetAuthorizationToken` ，`ecr:BatchGetImage` ，`ecr:GetDownloadUrlForLayer`
使用凭证配置存储库		
Amazon Secrets Manager（用户名/令牌）	存储个人访问令牌或密码	`secretsmanager:GetSecretValue`
Amazon Secrets Manager（SSH 密钥）	使用 SSH 密钥身份验证	`secretsmanager:GetSecretValue`
Amazon Secrets Manager（GitHub 应用程序）	使用私有密钥对 GitHub 应用程序进行身份验证	`secretsmanager:GetSecretValue`
Kubernetes Secret	使用集群内密钥的标准 Argo CD 方法	无（仅限信任策略）

直接访问 Amazon 服务

对于 Amazon 服务，您可以直接在 Application 资源中引用它们，而无需创建存储库配置。功能角色必须具有必要的 IAM 权限。

CodeCommit 存储库

直接在应用程序中引用 CodeCommit 存储库：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
```

```
metadata:
  name: my-app
  namespace: argocd
spec:
  source:
    repoURL: https://git-codecommit.region.amazonaws.com/v1/repos/repository-name
    targetRevision: main
    path: kubernetes/manifests
```

所需的功能角色权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codecommit:GitPull",
      "Resource": "arn:aws:codecommit:region:account-id:repository-name"
    }
  ]
}
```

CodeConnections

通过 CodeConnections 引用 GitHub、GitLab 或 Bitbucket 存储库。存储库 URL 格式派生自 CodeConnections 连接 ARN。

存储库 URL 格式为：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  source:
    repoURL: https://codeconnections.region.amazonaws.com/git-http/account-
id/region/connection-id/owner/repository.git
    targetRevision: main
    path: kubernetes/manifests
```

所需的功能角色权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codeconnections:UseConnection",
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-id"
    }
  ]
}
```

ECR Helm 图表

ECR 将 Helm 图表存储为 OCI 构件。Argo CD 支持两种引用方式：

Helm 格式（建议用于 Helm 图表）：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-helm
  namespace: argocd
spec:
  source:
    repoURL: account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: chart-version
    chart: chart-name
    helm:
      valueFiles:
        - values.yaml
```

注意：使用 Helm 格式时，请勿包含 `oci://` 前缀。使用 `chart` 字段指定图表名称。

OCI 格式（适用于包含 Kubernetes 清单的 OCI 构件）：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-oci
  namespace: argocd
spec:
  source:
```

```
    repoURL: oci://account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: artifact-version
    path: path-to-manifests
```

注意：使用 OCI 格式时，请包含 `oci://` 前缀。请使用 `path` 字段而非 `chart`。

所需的功能角色权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": "*"
    }
  ]
}
```

使用 Amazon Secrets Manager

将存储库凭证存储在 Secrets Manager 中，然后在 Argo CD 存储库配置中引用它们。

用户名和令牌身份验证

对于使用个人访问令牌或密码的 HTTPS 存储库：

在 Secrets Manager 中创建 Secret：

```
aws secretsmanager create-secret \
  --name argocd/my-repo \
  --description "GitHub credentials for Argo CD" \
  --secret-string '{"username":"your-username","token":"your-personal-access-token"}'
```

可选 TLS 客户端证书字段（适用于私有 Git 服务器）：

```
aws secretsmanager create-secret \
  --name argocd/my-private-repo \
```

```
  --secret-string '{
    "username":"your-username",
    "token":"your-token",

 "tlsClientCertData":"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCi4uLgotLS0tLUVORCBDRVJUSUZJQ0FURS0tL

 "tlsClientCertKey":"LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCi4uLgotLS0tLUVORCBQUklWQVRFIEtFWS0tLS
  }'
```

> **Note**
>
> `tlsClientCertData` 和 `tlsClientCertKey` 值必须采用 base64 编码。

创建引用 Secrets Manager 的 Repository Secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/my-repo-AbCdEf
  project: default
```

SSH 密钥身份验证

对于基于 SSH 的 Git 访问，请将私有密钥存储为纯文本（而非 JSON）：

创建包含 SSH 私有密钥的 Secret：

```
aws secretsmanager create-secret \
  --name argocd/my-repo-ssh \
  --description "SSH key for Argo CD" \
  --secret-string "-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
...
-----END OPENSSH PRIVATE KEY-----"
```

为 SSH 创建 Repository Secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo-ssh
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: git@github.com:your-org/your-repo.git
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/my-repo-ssh-
AbCdEf
  project: default
```

GitHub 应用程序身份验证

对于使用私有密钥的 GitHub 应用程序身份验证：

创建包含 GitHub 应用程序凭证的 Secret：

```
aws secretsmanager create-secret \
  --name argocd/github-app \
  --description "GitHub App credentials for Argo CD" \
  --secret-string '{

 "githubAppPrivateKeySecret":"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQouLi4KLS0tLS1FTkQgUlNBI
    "githubAppID":"123456",
    "githubAppInstallationID":"12345678"
  }'
```

> **Note**
>
> `githubAppPrivateKeySecret` 值必须采用 base64 编码。

用于 GitHub Enterprise 的可选字段：

```
aws secretsmanager create-secret \
  --name argocd/github-enterprise-app \
  --secret-string '{
```

```
 "githubAppPrivateKeySecret":"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQouLi4KLS0tLS1FTkQgUlNBI
    "githubAppID":"123456",
    "githubAppInstallationID":"12345678",
    "githubAppEnterpriseBaseUrl":"https://github.example.com/api/v3"
  }'
```

为 GitHub 应用程序创建 Repository Secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo-github-app
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/github-app-
AbCdEf
  project: default
```

> **Important**
>
> 确保 IAM 功能角色拥有所创建密钥的 `secretsmanager:GetSecretValue` 权限。有关 IAM 策略配置，请参阅 the section called “Argo CD 注意事项”。

使用 Amazon CodeConnections

有关 CodeConnections 集成，请参阅the section called “ Amazon CodeConnections”。

CodeConnections 为 GitHub、GitLab 和 Bitbucket 提供托管身份验证，无需存储凭证。

使用 Kubernetes Secrets

使用标准的 Argo CD 方法，将凭证直接存储在 Kubernetes 中。

对于使用个人访问令牌的 HTTPS：

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: my-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
  username: your-username
  password: your-personal-access-token
```

对于 SSH：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo-ssh
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: git@github.com:your-org/your-repo.git
  sshPrivateKey: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    ... your private key ...
    -----END OPENSSH PRIVATE KEY-----
```

公有存储库

公有存储库无需其他配置：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: public-app
  namespace: argocd
spec:
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps
    targetRevision: HEAD
    path: guestbook
```

```
  # ... rest of configuration
```

CodeCommit 存储库

对于 Amazon CodeCommit，授予 IAM 功能角色 CodeCommit 权限 (`codecommit:GitPull`)。

配置存储库：

```
apiVersion: v1
kind: Secret
metadata:
  name: codecommit-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://git-codecommit.us-west-2.amazonaws.com/v1/repos/my-repo
  project: default
```

有关详细的 IAM 策略配置，请参阅 the section called “Argo CD 注意事项”。

验证存储库连接

通过 Argo CD 用户界面，在“设置 → 存储库”下检查连接状态。用户界面会显示连接状态和任何身份验证错误。

Repository Secret 不包含状态信息。

其他资源

- the section called “注册集群”：注册用于部署的目标集群
- the section called “创建应用程序”：创建第一个应用程序
- the section called “Argo CD 注意事项”：IAM 权限和安全配置
- Private Repositories：上游存储库配置参考

注册目标集群

注册集群以允许 Argo CD 向其部署应用程序。您可以注册 Argo CD 运行所在的同一集群（本地集群），也可以注册位于不同账户或区域的远程集群。

先决条件

- 已创建具有 Argo CD 功能的 EKS 集群
- 已配置 `kubectl` 以与集群通信
- 对于远程集群：需要适当的 IAM 权限和访问条目

注册本地集群

要将应用程序部署到 Argo CD 运行所在的同一集群，请将其注册为部署目标。

> **Note**
>
> Argo CD 功能不会自动注册本地集群。必须进行明确注册才能将应用程序部署到同一集群。

使用 Argo CD CLI：

```
argocd cluster add <cluster-context-name> \
  --aws-cluster-name arn:aws:eks:us-west-2:111122223333:cluster/my-cluster \
  --name local-cluster
```

使用 Kubernetes Secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: local-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
stringData:
  name: local-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster
  project: default
```

应用配置：

```
kubectl apply -f local-cluster.yaml
```

> **Note**
>
> 在 `server` 字段中使用 EKS 集群 ARN，而不是 Kubernetes API 服务器 URL。托管功能需要使用 ARN 来识别集群。不支持默认的 `kubernetes.default.svc`。

注册远程集群

要部署到远程集群，您必须：

1. 在远程集群上为 Argo CD IAM 功能角色创建访问条目
2. 将访问策略与适当权限关联
3. 在 Argo CD 中注册集群

步骤 1：在远程集群上创建访问条目

将 *region-code* 替换为远程集群所在的 Amazon 区域，将 *remote-cluster* 替换为远程集群名称，然后将 ARN 替换为 Argo CD 功能角色 ARN。

```
aws eks create-access-entry \
  --region region-code \
  --cluster-name remote-cluster \
  --principal-arn arn:aws:iam::[.replaceable]111122223333:role/ArgoCDCapabilityRole \
  --type STANDARD
```

步骤 2：关联访问策略

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name remote-cluster \
  --principal-arn arn:aws:iam::[.replaceable]111122223333:role/ArgoCDCapabilityRole \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Note**
>
> 对于生产环境，可以考虑使用限制性更强的访问策略。有关最低权限配置，请参阅the section called “EKS 功能的注意事项”。

步骤 3：在 Argo CD 中注册集群

使用 Argo CD CLI：

```
argocd cluster add <cluster-context-name> \
  --aws-cluster-name arn:aws:eks:us-west-2:111122223333:cluster/remote-cluster \
  --name remote-cluster
```

使用 Kubernetes Secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: remote-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
stringData:
  name: remote-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/remote-cluster
  project: default
```

应用配置：

```
kubectl apply -f remote-cluster.yaml
```

跨账户和跨区域集群

要部署到不同 Amazon 账户或地区中的集群，请执行以下操作：

1. 在远程集群上将 Argo CD 功能角色添加为访问条目
2. 关联适当的访问策略（通常为 `AmazonEKSClusterAdminPolicy`）
3. 使用其完整 ARN（包括区域）注册集群

集群 ARN 格式包括区域，因此跨账户注册与跨区域注册并无差异，两者所使用的流程相同。

有关详细的跨账户配置（包括信任策略和 IAM 权限），请参阅 the section called “Argo CD 注意事项”。

验证集群注册

查看已注册集群：

```
kubectl get secrets -n argocd -l argocd.argoproj.io/secret-type=cluster
```

或在 Argo CD 用户界面的“设置 → 集群”下查看集群状态。

私有集群

Argo CD 功能支持对完全私有的 EKS 集群进行透明访问，无需进行 VPC 对等连接或专门的网络配置。

Amazon 会自动管理 Argo CD 功能与私有远程集群之间的连接。

只需使用其 ARN 注册私有集群，无需进行额外的联网设置。

使用项目限制集群访问

使用项目来控制应用程序可以部署到哪些集群：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: production
  namespace: argocd
spec:
  destinations:
  - server: arn:aws:eks:us-west-2:111122223333:cluster/prod-cluster
    namespace: '*'
  - server: arn:aws:eks:eu-west-1:111122223333:cluster/prod-eu-cluster
    namespace: '*'
  sourceRepos:
  - 'https://github.com/example/production-apps'
```

有关更多信息，请参阅 the section called “Projects”。

其他资源

- the section called “Projects”：组织应用程序并强制实施安全边界
- the section called “创建应用程序”：部署第一个应用程序
- the section called “ApplicationSets”：使用 ApplicationSets 部署到多个集群

- the section called “Argo CD 注意事项”：多集群模式和跨账户设置
- Declarative Cluster Setup：上游集群配置参考

使用 Argo CD 项目

Argo CD 项目（AppProject）为应用程序提供逻辑分组和访问控制。项目定义了应用程序可以使用哪些 Git 存储库、目标集群和命名空间，从而在 Argo CD 共享实例中实现多租户和安全边界。

何时使用项目

使用项目以：

- 按团队、环境或业务部门分离应用程序
- 限制团队可以从中部署哪些存储库
- 限制团队可以部署到哪些集群和命名空间
- 强制执行资源配额和允许的资源类型
- 提供带护栏的应用程序自助部署

默认项目

每个 Argo CD 功能都包含 `default` 项目，该项目允许访问所有存储库、集群和命名空间。这对于初始测试很有用，但对于生产用途，请创建具有明确限制的专用项目。

有关默认项目配置以及如何对其进行限制的详细信息，请参阅 Argo CD 文档中的 The Default Project。

创建项目

通过将 `AppProject` 资源应用到集群来创建项目。

示例：团队特定项目

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  description: Applications for Team A
```

```
  # Source repositories this project can deploy from
  sourceRepos:
    - 'https://github.com/my-org/team-a-*'
    - 'https://github.com/my-org/shared-libs'

  # Destination clusters and namespaces
  destinations:
    - name: dev-cluster
      namespace: team-a-dev
    - name: prod-cluster
      namespace: team-a-prod

  # Allowed resource types
  clusterResourceWhitelist:
    - group: ''
      kind: Namespace

  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
    - group: ''
      kind: Service
    - group: ''
      kind: ConfigMap
```

应用项目：

```
kubectl apply -f team-a-project.yaml
```

项目配置

源存储库

控制此项目中的应用程序可以使用哪些 Git 存储库：

```
spec:
  sourceRepos:
    - 'https://github.com/my-org/app-*'  # Wildcard pattern
    - 'https://github.com/my-org/infra'  # Specific repo
```

您可以使用通配符和否定模式（！前缀）来允许或拒绝特定的存储库。有关详细信息，请参阅 Argo CD 文档中的 Managing Projects。

目标限制

限制可以部署应用程序的位置：

```
spec:
  destinations:
    - name: prod-cluster  # Specific cluster by name
      namespace: production
    - name: '*'  # Any cluster
      namespace: team-a-*  # Namespace pattern
```

> **Important**
>
> 对于生产项目，请使用特定的集群名称和命名空间模式，而不是通配符。这样可以防止意外部署到未经授权的集群或命名空间。

您可以使用通配符和否定模式来控制目标。有关详细信息，请参阅 Argo CD 文档中的 Managing Projects。

资源限制

控制可以部署的 Kubernetes 资源类型：

集群范围资源：

```
spec:
  clusterResourceWhitelist:
    - group: ''
      kind: Namespace
    - group: 'rbac.authorization.k8s.io'
      kind: Role
```

命名空间范围资源：

```
spec:
  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
    - group: ''
      kind: Service
```

```
    - group: ''
      kind: ConfigMap
    - group: 's3.services.k8s.aws'
      kind: Bucket
```

使用黑名单拒绝特定资源：

```
spec:
  namespaceResourceBlacklist:
    - group: ''
      kind: Secret  # Prevent direct Secret creation
```

将应用程序分配给项目

创建应用程序时，在 spec.project 字段中指定项目：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  project: team-a  # Assign to team-a project
  source:
    repoURL: https://github.com/my-org/my-app
    path: manifests
  destination:
    name: prod-cluster
    namespace: team-a-prod
```

未指定项目的应用程序将使用 default 项目。

项目角色和 RBAC

项目可以定义自定义角色以实现精细访问控制。在功能配置中将项目角色映射到 Amazon Identity Center 用户和用户组，以控制哪些人可以同步、更新或删除应用程序。

示例：具有开发人员和管理员角色的项目

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
```

```
  name: team-a
  namespace: argocd
spec:
  sourceRepos:
    - '*'
  destinations:
    - name: '*'
      namespace: 'team-a-*'

  roles:
    - name: developer
      description: Developers can sync applications
      policies:
        - p, proj:team-a:developer, applications, sync, team-a/*, allow
        - p, proj:team-a:developer, applications, get, team-a/*, allow
      groups:
        - team-a-developers

    - name: admin
      description: Admins have full access
      policies:
        - p, proj:team-a:admin, applications, *, team-a/*, allow
      groups:
        - team-a-admins
```

有关项目角色、用于 CI/CD 管道的 JWT 令牌以及 RBAC 配置的详细信息，请参阅 Argo CD 文档中的 Project Roles。

常见模式

基于环境的项目

为每个环境创建单独的项目：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: production
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/*'
  destinations:
```

```
    - name: prod-cluster
      namespace: '*'
  # Strict resource controls for production
  clusterResourceWhitelist: []
  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
    - group: ''
      kind: Service
```

基于团队的项目

使用专用项目隔离团队：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: platform-team
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/platform-*'
  destinations:
    - name: '*'
      namespace: 'platform-*'
  # Platform team can manage cluster resources
  clusterResourceWhitelist:
    - group: '*'
      kind: '*'
```

多集群项目

使用一致的策略部署到多个集群：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: global-app
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/global-app'
  destinations:
```

```
    - name: us-west-cluster
      namespace: app
    - name: eu-west-cluster
      namespace: app
    - name: ap-south-cluster
      namespace: app
```

最佳实践

从限制性项目开始：从范围较小的权限开始，然后根据需要进行扩展，而不是一开始就授予广泛的访问权限。

使用命名空间模式：在命名空间限制中使用通配符（例如 `team-a-*`），在保持边界的同时提供灵活性。

隔离生产项目：为生产环境使用具有更严格控制和手动同步策略的专用项目。

记录项目用途：使用 `description` 字段说明每个项目的用途以及适用人员。

定期审查项目权限：定期审核项目，确保限制始终符合团队需求和安全性要求。

其他资源

- the section called “配置权限”：配置 RBAC 和 Identity Center 集成
- the section called “创建应用程序”：在项目内创建应用程序
- the section called “ApplicationSets”：将 ApplicationSets 与项目结合使用，以进行多集群部署
- Argo CD Projects Documentation：完整的上游参考

创建应用程序

应用程序代表目标集群中的部署。每个应用程序都定义了源（Git 存储库）和目标（集群和命名空间）。应用后，Argo CD 会将 Git 存储库中清单指定的资源创建到集群中的命名空间。应用程序通常用于指定工作负载部署，但它们可以管理目标集群中任何可用的 Kubernetes 资源。

先决条件

- 已创建具有 Argo CD 功能的 EKS 集群
- 已配置存储库访问权限（请参阅the section called “配置存储库”）
- 已注册目标集群（请参阅the section called “注册集群”）
- 已配置 `kubectl` 以与集群通信

创建基础应用程序

定义从 Git 存储库部署的应用程序：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps
    targetRevision: HEAD
    path: guestbook
  destination:
    server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster
    namespace: default
```

> **Important**
>
> 在 `destination.server` 字段中使用 EKS 集群 ARN，而不是 Kubernetes API 服务器 URL。托管功能需要使用 ARN 来识别集群。

应用应用程序：

```
kubectl apply -f application.yaml
```

查看应用程序状态：

```
kubectl get application guestbook -n argocd
```

源配置

Git 存储库：

```
spec:
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: main
```

```
    path: kubernetes/manifests
```

特定的 Git 标签或提交：

```
spec:
  source:
    targetRevision: v1.2.0  # or commit SHA
```

Helm 图表 –

```
spec:
  source:
    repoURL: https://github.com/example/helm-charts
    targetRevision: main
    path: charts/my-app
    helm:
      valueFiles:
      - values.yaml
      parameters:
      - name: image.tag
        value: v1.2.0
```

来自 ECR 的 Helm 图表：

```
spec:
  source:
    repoURL: oci://account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: chart-version
    chart: chart-name
```

如果功能角色具有所需的 ECR 权限，则可以直接使用该存储库，无需配置。有关详细信息，请参阅 the section called “配置存储库”。

来自 CodeCommit 的 Git 存储库：

```
spec:
  source:
    repoURL: https://git-codecommit.region.amazonaws.com/v1/repos/repository-name
    targetRevision: main
    path: kubernetes/manifests
```

如果功能角色具有所需的 CodeCommit 权限，则可以直接使用该存储库，无需配置。有关详细信息，请参阅 the section called “配置存储库”。

来自 CodeConnections 的 Git 存储库：

```
spec:
  source:
    repoURL: https://codeconnections.region.amazonaws.com/git-http/account-
id/region/connection-id/owner/repository.git
    targetRevision: main
    path: kubernetes/manifests
```

存储库 URL 格式派生自 CodeConnections 连接 ARN。如果功能角色具有所需的 CodeConnections 权限且已配置连接，则可以直接使用该存储库，无需配置。有关详细信息，请参阅 the section called “配置存储库”。

Kustomize：

```
spec:
  source:
    repoURL: https://github.com/example/kustomize-app
    targetRevision: main
    path: overlays/production
    kustomize:
      namePrefix: prod-
```

同步策略

控制 Argo CD 如何同步应用程序。

手动同步（默认）：

应用程序需要手动批准才能同步：

```
spec:
  syncPolicy: {}  # No automated sync
```

手动触发同步：

```
kubectl patch application guestbook -n argocd \
  --type merge \
```

```
  --patch '{"operation": {"initiatedBy": {"username": "admin"}, "sync": {}}}'
```

自动同步：

检测到 Git 更改时，应用程序会自动同步：

```
spec:
  syncPolicy:
    automated: {}
```

自我修复：

自动还原对集群所作的手动更改：

```
spec:
  syncPolicy:
    automated:
      selfHeal: true
```

启用后，Argo CD 会还原直接对集群所作的任何手动更改，确保 Git 始终是事实来源。

修剪：

自动删除从 Git 中移除的资源：

```
spec:
  syncPolicy:
    automated:
      prune: true
```

> **Warning**
>
> 修剪将从集群中删除资源。在生产环境中请谨慎使用。

组合式自动同步：

```
spec:
  syncPolicy:
    automated:
```

```
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

同步选项

其他同步配置：

如果命名空间不存在，请创建一个：

```
spec:
  syncPolicy:
    syncOptions:
    - CreateNamespace=true
```

应用前验证资源：

```
spec:
  syncPolicy:
    syncOptions:
    - Validate=true
```

仅应用不同步项：

```
spec:
  syncPolicy:
    syncOptions:
    - ApplyOutOfSyncOnly=true
```

高级同步功能

Argo CD 支持用于复杂部署的高级同步功能：

- 同步波次：使用 `argocd.argoproj.io/sync-wave` 注释控制资源创建顺序
- 同步钩子：使用 `argocd.argoproj.io/hook` 注释（PreSync、PostSync、SyncFail）在同步前后运行任务
- 资源运行状况评测：针对特定于应用程序的资源进行自定义运行状况检查

有关详细信息，请参阅 Argo CD 文档中的 Sync Waves 和 Resource Hooks。

忽略差异

防止 Argo CD 同步由其他控制器管理的特定字段（例如由 HPA 管理的副本）：

```
spec:
  ignoreDifferences:
  - group: apps
    kind: Deployment
    jsonPointers:
    - /spec/replicas
```

有关忽略模式和字段排除项的详细信息，请参阅 Argo CD 文档中的 Diffing Customization。

多环境部署

将同一应用程序部署到多个环境：

开发

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-dev
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: develop
    path: overlays/development
  destination:
    server: arn:aws:eks:us-west-2:111122223333:cluster/dev-cluster
    namespace: my-app
```

生产环境：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-prod
  namespace: argocd
spec:
```

```
  project: default
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: main
    path: overlays/production
  destination:
    server: arn:aws:eks:us-west-2:111122223333:cluster/prod-cluster
    namespace: my-app
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

监控和管理应用程序

查看应用程序状态：

```
kubectl get application my-app -n argocd
```

访问 Argo CD 用户界面：

通过 EKS 控制台打开 Argo CD 用户界面，查看应用程序拓扑、同步状态、资源运行状况和部署历史记录。有关用户界面访问说明，请参阅the section called “使用 Argo CD”。

回滚应用程序：

可以通过 Argo CD 用户界面，或将应用程序规范中的 `targetRevision` 更新为之前的 Git 提交或标签，回滚到先前版本。

其他资源

- the section called “Projects”：为多租户环境使用项目组织应用程序
- the section called “ApplicationSets”：通过模板部署到多个集群
- Application Specification：完整的应用程序 API 参考
- Sync Options：高级同步配置

使用 ApplicationSets

ApplicationSet 通过模板生成多个应用程序，使您能够通过单个资源定义，跨多个集群、环境或命名空间部署相同的应用程序。

先决条件

- 已创建具有 Argo CD 功能的 EKS 集群
- 已注册多个目标集群（请参阅the section called “注册集群”）
- 已配置存储库访问权限（请参阅the section called “配置存储库”）
- 已配置 `kubectl` 以与集群通信

ApplicationSets 的工作原理

ApplicationSets 使用生成器来生成参数，然后将这些参数应用于应用程序模板。每组生成的参数都会创建一个应用程序。

适用于 EKS 部署的常用生成器：

- 列表生成器：为每个环境明确定义集群和参数
- 集群生成器：自动部署到所有已注册的集群
- Git 生成器：基于存储库结构生成应用程序
- 矩阵生成器：组合多个生成器以实现多维部署

有关完整的生成器参考，请参阅 ApplicationSet 文档。

列表生成器

通过明确配置部署到多个集群：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: guestbook-all-clusters
  namespace: argocd
spec:
  generators:
  - list:
      elements:
      - cluster: arn:aws:eks:us-west-2:111122223333:cluster/dev-cluster
        environment: dev
        replicas: "2"
      - cluster: arn:aws:eks:us-west-2:111122223333:cluster/staging-cluster
        environment: staging
        replicas: "3"
```

```
      - cluster: arn:aws:eks:us-west-2:111122223333:cluster/prod-cluster
        environment: prod
        replicas: "5"
  template:
    metadata:
      name: 'guestbook-{{environment}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/guestbook
        targetRevision: HEAD
        path: 'overlays/{{environment}}'
      destination:
        server: '{{cluster}}'
        namespace: guestbook
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

> **Note**
>
> 将已注册的 EKS 集群设为目标时，请在 `server` 字段中使用 EKS 集群 ARN。您也可以使用带有 `destination.name` 的集群名称，而非 `destination.server`。

这将创建三个应用程序：`guestbook-dev`、`guestbook-staging` 和 `guestbook-prod`。

集群生成器

自动部署到所有已注册的集群：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: cluster-addons
  namespace: argocd
spec:
  generators:
  - clusters: {}
  template:
    metadata:
```

```
      name: '{{name}}-addons'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/cluster-addons
        targetRevision: HEAD
        path: addons
      destination:
        server: '{{server}}'
        namespace: kube-system
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

这将自动为每个已注册的集群创建应用程序。

筛选集群：

```
spec:
  generators:
  - clusters:
      selector:
        matchLabels:
          environment: production
```

Git 生成器

Git 生成器基于存储库结构创建应用程序：

- 目录生成器：将每个目录作为单独的应用程序部署（适用于微服务）
- 文件生成器：根据参数文件生成应用程序（适用于多租户部署）

示例：微服务部署

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: microservices
  namespace: argocd
spec:
  generators:
```

```
  - git:
      repoURL: https://github.com/example/microservices
      revision: HEAD
      directories:
      - path: services/*
  template:
    metadata:
      name: '{{path.basename}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/microservices
        targetRevision: HEAD
        path: '{{path}}'
      destination:
        server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster
        namespace: '{{path.basename}}'
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
        syncOptions:
        - CreateNamespace=true
```

有关 Git 生成器和基于文件的配置的详细信息，请参阅 Argo CD 文档中的 Git Generator。

矩阵生成器

组合多个生成器，以实现跨多个维度（环境 × 集群）的部署：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: multi-env-multi-cluster
  namespace: argocd
spec:
  generators:
  - matrix:
      generators:
      - list:
          elements:
          - environment: dev
          - environment: staging
          - environment: prod
```

```
      - clusters:
          selector:
            matchLabels:
              region: us-west-2
  template:
    metadata:
      name: 'app-{{environment}}-{{name}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/app
        targetRevision: HEAD
        path: 'overlays/{{environment}}'
      destination:
        server: '{{server}}'
        namespace: 'app-{{environment}}'
```

有关组合生成器的详细信息，请参阅 Argo CD 文档中的 Matrix Generator。

逐步部署

使用不同的同步策略，按顺序部署到环境：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: progressive-rollout
  namespace: argocd
spec:
  generators:
  - list:
      elements:
      - environment: dev
        autoSync: "true"
      - environment: staging
        autoSync: "true"
      - environment: prod
        autoSync: "false"
  template:
    metadata:
      name: 'app-{{environment}}'
    spec:
      project: default
      source:
```

```
        repoURL: https://github.com/example/app
        targetRevision: HEAD
        path: 'overlays/{{environment}}'
      destination:
        server: arn:aws:eks:us-west-2:111122223333:cluster/{{environment}}-cluster
        namespace: app
      syncPolicy:
        automated:
          prune: true
          selfHeal: '{{autoSync}}'
```

开发和暂存采用自动同步，而生产则需手动批准。

多区域部署

跨多个区域部署到集群：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: global-app
  namespace: argocd
spec:
  generators:
  - list:
      elements:
      - cluster: arn:aws:eks:us-west-2:111122223333:cluster/prod-us-west
        region: us-west-2
      - cluster: arn:aws:eks:us-east-1:111122223333:cluster/prod-us-east
        region: us-east-1
      - cluster: arn:aws:eks:eu-west-1:111122223333:cluster/prod-eu-west
        region: eu-west-1
  template:
    metadata:
      name: 'app-{{region}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/app
        targetRevision: HEAD
        path: kubernetes
        helm:
          parameters:
          - name: region
```

```
            value: '{{region}}'
      destination:
        server: '{{cluster}}'
        namespace: app
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

管理 ApplicationSets

查看 ApplicationSets 及生成的应用程序：

```
kubectl get applicationsets -n argocd
kubectl get applications -n argocd -l argocd.argoproj.io/application-set-
name=<applicationset-name>
```

更新 ApplicationSet：

修改 ApplicationSet 规范并重新应用。Argo CD 会自动更新所有生成的应用程序：

```
kubectl apply -f applicationset.yaml
```

删除 ApplicationSet：

```
kubectl delete applicationset <name> -n argocd
```

> **Warning**
>
> 删除 ApplicationSet 会同时删除所有生成的应用程序。如果这些应用程序设置了 `prune: true`，其资源也将从目标集群中删除。

其他资源

- the section called “Projects”：通过项目整理 ApplicationSets
- the section called “创建应用程序”：了解应用程序配置
- ApplicationSet Documentation：完整的生成器参考和模式
- Generator Reference：详细的生成器规范

Argo CD 注意事项

本主题将介绍使用适用于 Argo CD 的 EKS 功能的重要注意事项，包括规划、权限、身份验证和多集群部署模式。

规划

在部署 Argo CD 之前，请考虑以下事项：

存储库策略：确定应用程序清单的存储位置（CodeCommit、GitHub、GitLab、Bitbucket）。为不同的环境规划存储库结构和分支策略。

身份验证方法：决定是使用 Amazon Identity Center 进行 SSO 还是直接管理 Argo CD 用户。建议将 SSO 用于生产环境。

RBAC 策略：规划哪些团队或用户应具有管理员、编辑者或查看者访问权限。将其映射到 Amazon Identity Center 组或 Argo CD 角色。

多集群架构：确定是否要通过单个 Argo CD 实例管理多个集群。考虑为 Argo CD 使用专用的管理集群。

应用程序组织：规划如何构建 Applications 和 ApplicationSets。考虑使用项目按团队或环境组织应用程序。

同步策略：决定应用程序应自动同步还是需要手动批准。开发环境通常采用自动同步，生产环境则采用手动同步。

权限

有关 IAM 功能角色、信任策略和安全最佳实践的详细信息，请参阅 the section called “功能 IAM 角色”和 the section called “EKS 功能的注意事项”。

IAM 功能角色概述

创建 Argo CD 功能资源时，您需要提供 IAM 功能角色。与 ACK 不同，Argo CD 主要管理 Kubernetes 资源，而不是直接管理 Amazon 资源。但是，IAM 角色在以下场景是必需的：

- 访问 CodeCommit 中的私有 Git 存储库
- 与 Amazon Identity Center 集成以进行身份验证
- 访问 Amazon Secrets Manager 中的密钥（如果已配置）
- 跨集群部署到其他 EKS 集群

CodeCommit 集成

如果使用的是 CodeCommit 存储库，请附加具有读取权限的策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": "*"
    }
  ]
}
```

Important

对于生产用途，请将 `Resource` 字段限制为特定的存储库 ARN，而不是使用 `"*"`。
示例：

```
"Resource": "arn:aws:codecommit:us-west-2:111122223333:my-app-repo"
```

这会将 Argo CD 限制为仅管理其所需的存储库。

Secrets Manager 集成

如果要将存储库凭证存储在 Secrets Manager 中，请附加具有读取权限的策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:argocd/*"
```

```
    }
  ]
}
```

基本设置

对于使用公有 Git 存储库的基本 Argo CD 功能，除了信任策略外，不需要其他的 IAM 策略。

身份验证

Amazon Identity Center 集成

Argo CD 托管功能直接与 Amazon Identity Center（原 Amazon SSO）集成，您能够使用现有的身份提供者进行身份验证。

配置 Amazon Identity Center 集成时：

1. 用户通过 EKS 控制台访问 Argo CD 用户界面
2. 他们使用 Amazon Identity Center（可以与企业身份提供者联合）进行身份验证
3. Amazon Identity Center 向 Argo CD 提供用户和组信息
4. Argo CD 会根据配置将用户和组映射到 RBAC 角色
5. 用户只能看到他们有权访问的应用程序和资源

使用 IAM Center 权限集简化访问管理

使用 Argo CD 时，Amazon Identity Center 提供两种不同的身份验证路径：

Argo CD API 身份验证：Identity Center 向 Argo CD 用户界面和 API 提供 SSO 身份验证。这通过 Argo CD 功能的 RBAC 角色映射进行配置。

EKS 集群访问权限：Argo CD 功能使用客户提供的 IAM 角色，通过访问条目向 EKS 集群进行身份验证。可以手动配置这些访问条目以添加或移除权限。

您可以使用 Identity Center 权限集来简化身份管理，允许单个身份同时访问 Argo CD 和 EKS 集群。这样一来，您只需在两个系统中管理一个身份，而无需为 Argo CD 访问和集群访问保留单独的凭证，从而减少了开销。

RBAC 角色映射

Argo CD 具有内置角色，您可以将其映射到 Amazon Identity Center 用户和组：

ADMIN：可以完全访问所有应用程序和设置。可以创建、更新和删除应用程序。可以管理 Argo CD 配置。

EDITOR：可以创建和修改应用程序。无法更改 Argo CD 设置或删除应用程序。

VIEWER：可以对应用程序进行只读访问。可以查看应用程序状态和历史记录。无法进行更改。

> **Note**
>
> 角色名称区分大小写，必须为大写（ADMIN、EDITOR、VIEWER）。

> **Important**
>
> EKS 功能与 Amazon Identity Center 集成后，每个 Argo CD 功能可支持最多 1000 个身份。身份可以是用户或组。

多集群部署

Argo CD 托管功能支持多集群部署，便于您通过单个 Argo CD 实例在开发、暂存和生产集群之间管理应用程序。

多集群工作原理

当您向 Argo CD 注册其他集群时：

1. 您需要创建集群密钥，并通过 ARN 来引用目标 EKS 集群
2. 您需要创建针对不同集群的 Applications 或 ApplicationSets
3. Argo CD 会连接到每个集群以部署应用程序和监视资源
4. 您可以通过单个 Argo CD 用户界面查看和管理所有集群

多集群的先决条件

在注册其他集群之前：

- 在目标集群上，为 Argo CD 功能角色创建访问条目
- 确保 Argo CD 功能与目标集群之间建立网络连接

- 验证访问目标集群所需的 IAM 权限

注册集群

使用 `argocd` 命名空间中的 Kubernetes Secret 注册集群。

获取目标集群 ARN。将 *region-code* 替换为目标集群所在的 Amazon 区域，并将 *target-cluster* 替换为目标集群的名称。

```
aws eks describe-cluster \
  --region region-code \
  --name target-cluster \
  --query 'cluster.arn' \
  --output text
```

使用集群 ARN 创建集群密钥：

```
apiVersion: v1
kind: Secret
metadata:
  name: target-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
type: Opaque
stringData:
  name: target-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/target-cluster
  project: default
```

> **Important**
>
> 在 `server` 字段中使用 EKS 集群 ARN，而不是 Kubernetes API 服务器 URL。托管功能需要使用 ARN 来识别目标集群。

应用该密钥：

```
kubectl apply -f cluster-secret.yaml
```

在目标集群上配置访问条目

目标集群必须具有访问条目，以授予 Argo CD 功能角色部署应用程序的权限。将 *region-code* 替换为目标集群所在的 Amazon 区域，将 *target-cluster* 替换为目标集群的名称，然后将 ARN 替换为 Argo CD 功能角色 ARN。

```
aws eks create-access-entry \
  --region region-code \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::[.replaceable]111122223333:role/ArgoCDCapabilityRole \
  --type STANDARD \
  --kubernetes-groups system:masters
```

> **Note**
>
> 对于生产用途，可以考虑使用限制性更强的 Kubernetes 组，而不使用 `system:masters`。

私有集群访问

Argo CD 托管功能可以部署到完全私有的 EKS 集群，无需进行 VPC 对等连接或专门的网络配置。Amazon 会自动管理 Argo CD 功能与私有远程集群之间的连接。

跨账户部署

对于跨账户部署，请将源账户中的 Argo CD IAM 功能角色添加到目标集群的 EKS 访问条目中：

1. 在目标账户中的目标 EKS 集群上创建访问条目
2. 将源账户中的 Argo CD IAM 功能角色 ARN 用作主体
3. 为访问条目配置相应的 Kubernetes RBAC 权限
4. 使用其 EKS 集群 ARN 在 Argo CD 中注册目标集群

无需创建额外的 IAM 角色或配置信任策略，EKS 访问条目会处理跨账户访问。

最佳实践

使用声明式源作为事实来源：将所有应用程序清单存储在 Git 存储库、Helm 注册表或 OCI 映像中，从而实现版本控制、审计跟踪记录和协作。

实施适当的 RBAC：使用 Amazon Identity Center 集成来控制哪些人可以访问和管理 Argo CD 中的应用程序。Argo CD 支持对应用程序内的资源（Deployment、Pod、ConfigMap、Secret）进行精细访问控制。

使用 ApplicationSets 进行多环境部署：使用 ApplicationSets 在不同集群或命名空间中以不同配置部署应用程序。

生命周期管理

应用程序同步策略

控制 Argo CD 如何同步应用程序：

手动同步：应用程序需要手动批准才能同步更改。建议将其用于生产环境。

自动同步：检测到 Git 更改时，应用程序会自动同步。通常用于开发和暂存环境。

自我修复：自动还原对集群所作的手动更改。确保集群状态与 Git 一致。

修剪：自动删除从 Git 中移除的资源。请谨慎使用，因为此操作可能会删除资源。

应用程序运行状况

Argo CD 会持续监控应用程序运行状况：

- 良好：所有资源均按预期运行
- 正在处理：正在创建或更新资源
- 已降级：部分资源运行状况不佳
- 已暂停：应用程序暂停
- 缺失：集群中缺少资源

同步时段

配置同步时段以控制何时可以同步应用程序：

- 仅允许在维护时段进行同步
- 在工作时间阻止同步
- 为特定时间安排自动同步

- 在需要暂时停止所有同步时，采用同步时段应对紧急情况

配置 Webhook 以实现更快同步

默认情况下，Argo CD 每 6 分钟轮询一次 Git 存储库以检测更改。要实现响应速度更快的部署，请配置 Git Webhook 以在推送更改时立即触发同步。

Webhook 具有以下优势：

- 推送代码时立即同步响应（秒级而非分钟级）
- 减少轮询开销并提高系统性能
- 更有效地使用 API 速率限制
- 用户体验更好，反馈更快

Webhook 端点

Argo CD 功能提供了用于接收 Git 通知的 Webhook 端点。您可以在 EKS 控制台中的集群的“功能”选项卡下查找 Webhook URL，或通过 Amazon CLI 进行检索：

```
aws eks describe-capability \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.configuration.argoCd.webhookUrl' \
  --output text \
  --region region-code
```

按 Git 提供者配置 Webhook

GitHub：在存储库设置中，添加指向 Argo CD Webhook URL 的 Webhook。将内容类型设置为 `application/json`，然后选择“仅推送事件”。

GitLab：在项目设置中，添加指向 Argo CD Webhook URL 的 Webhook。启用“推送事件”和“标签推送事件”（可选）。

Bitbucket：在存储库设置中，添加指向 Argo CD Webhook URL 的 Webhook。选择“存储库推送”作为触发器。

CodeCommit：创建 Amazon EventBridge 规则，该规则会在 CodeCommit 存储库状态更改时触发，并将通知发送到 Argo CD Webhook 端点。

有关详细的 Webhook 配置说明，请参阅 Argo CD Webhook Configuration。

> **Note**
>
> Webhooks 是对轮询的补充，并不能取而代之。Argo CD 会继续轮询存储库，作为错过 Webhook 通知时的回退机制。

后续步骤

- the section called “使用 Argo CD”：了解如何创建和管理 Argo CD 应用程序
- the section called “问题排查”：排查 Argo CD 问题
- the section called “使用功能”：管理 Argo CD 功能资源

排查 Argo CD 功能的问题

本主题提供了适用于 Argo CD 的 EKS 功能的问题排查指南，包括功能运行状况检查、应用程序同步问题、存储库身份验证和多集群部署。

> **Note**
>
> EKS 功能完全托管，可在您的集群之外运行。您无权访问 Argo CD 服务器日志或 `argocd` 命名空间。问题排查侧重于功能运行状况、应用程序状态和配置。

功能已激活，但应用程序未同步

如果 Argo CD 功能显示 `ACTIVE` 状态但应用程序未同步，请检查功能运行状况和应用程序状态。

检查功能运行状况：

您可以在 EKS 控制台或使用 Amazon CLI 查看功能运行状况和状态问题。

控制台：

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称。
3. 选择可观测性选项卡。

4. 选择监控集群。

5. 选择功能选项卡，查看所有功能的运行状况和状态。

Amazon CLI：

```
# View capability status and health
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd

# Look for issues in the health section
```

常见原因：

- 未配置存储库：Git 存储库未添加到 Argo CD
- 身份验证失败：SSH 密钥、令牌或 CodeCommit 凭证无效
- 未创建应用程序：集群中不存在 Application 资源
- 同步策略：需要手动同步（未启用自动同步）
- IAM 权限：缺少 CodeCommit 或 Secrets Manager 的权限

检查应用程序状态：

```
# List applications
kubectl get application -n argocd

# View sync status
kubectl get application my-app -n argocd -o jsonpath='{.status.sync.status}'

# View application health
kubectl get application my-app -n argocd -o jsonpath='{.status.health}'
```

查看应用程序条件：

```
# Describe application to see detailed status
kubectl describe application my-app -n argocd

# View application health
```

```
kubectl get application my-app -n argocd -o jsonpath='{.status.health}'
```

应用程序始终处于“Progressing”状态

如果应用程序显示 Progressing 但从未到达 Healthy 状态，请检查该应用程序的资源状态和事件。

检查资源运行状况：

```
# View application resources
kubectl get application my-app -n argocd -o jsonpath='{.status.resources}'

# Check for unhealthy resources
kubectl describe application my-app -n argocd | grep -A 10 "Health Status"
```

常见原因：

- 部署未就绪：容器组（pod）无法启动或就绪探测器故障
- 资源依赖关系：等待其他资源准备就绪的资源
- 图片拉取错误：无法访问容器映像
- 资源不足：集群缺乏用于容器组（pod）的 CPU 或内存

验证目标集群配置（适用于多集群设置）：

```
# List registered clusters
kubectl get secret -n argocd -l argocd.argoproj.io/secret-type=cluster

# View cluster secret details
kubectl get secret cluster-secret-name -n argocd -o yaml
```

存储库身份验证失败

如果 Argo CD 无法访问您的 Git 存储库，请确认身份验证配置是否正确。

对于 CodeCommit 存储库：

验证 IAM 功能角色是否具有 CodeCommit 权限：

```
# View IAM policies
aws iam list-attached-role-policies --role-name my-argocd-capability-role
```

```
aws iam list-role-policies --role-name my-argocd-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-argocd-capability-role --policy-name policy-name
```

该角色需要存储库的 codecommit:GitPull 权限。

对于私有 Git 存储库：

验证存储库凭证配置是否正确：

```
# Check repository secret exists
kubectl get secret -n argocd repo-secret-name -o yaml
```

确保密钥包含正确的身份验证凭证（SSH 密钥、令牌或用户名/密码）。

对于使用 Secrets Manager 的存储库：

```
# Verify IAM Capability Role has Secrets Manager permissions
aws iam list-attached-role-policies --role-name my-argocd-capability-role

# Test secret retrieval
aws secretsmanager get-secret-value --secret-id arn:aws:secretsmanager:region-
code:111122223333:secret:my-secret
```

多集群部署问题

如果应用程序未部署到远程集群，请确认集群注册和访问配置是否正确。

检查集群注册：

```
# List registered clusters
kubectl get secret -n argocd -l argocd.argoproj.io/secret-type=cluster

# Verify cluster secret format
kubectl get secret CLUSTER_SECRET_NAME -n argocd -o yaml
```

确保 server 字段包含 EKS 集群 ARN，而不是 Kubernetes API URL。

验证目标集群访问入口：

在目标集群上，检查 Argo CD 功能角色是否具有访问条目：

```
# List access entries (run on target cluster or use AWS CLI)
aws eks list-access-entries --cluster-name target-cluster

# Describe specific access entry
aws eks describe-access-entry \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::[.replaceable]111122223333:role/my-argocd-capability-
role
```

检查跨账户的 IAM 权限：

对于跨账户部署，请验证 Argo CD 功能角色在目标集群上是否有访问条目。托管功能使用 EKS 访问条目进行跨账户访问，而不是 IAM 角色代入。

有关多集群配置的更多信息，请参阅the section called “注册集群”。

后续步骤

- the section called “Argo CD 注意事项”：Argo CD 注意事项和最佳实践
- the section called “使用 Argo CD”：创建和管理 Argo CD 应用程序
- the section called “注册集群”：配置多集群部署
- the section called “问题排查功能”：一般功能问题排查指导

适用于 Argo CD 的 EKS 功能与自主管理型 Argo CD 的比较

适用于 Argo CD 的 EKS 功能提供了在 EKS 中运行的完全托管的 Argo CD。有关 EKS 功能与自主管理型解决方案的总体比较，请参阅 the section called “注意事项”。本主题将重点介绍 Argo CD 特有的差异，包括身份验证、多集群管理及上游功能支持。

与上游 Argo CD 的区别

适用于 Argo CD 的 EKS 功能基于上游 Argo CD，但在访问方式、配置以及与 Amazon 服务的集成方面有所不同。

RBAC 和身份验证：该功能附带三个 RBAC 角色（管理员、编辑者和查看者），并使用 Amazon Identity Center 进行身份验证，而不是 Argo CD 的内置身份验证。通过该功能的 `rbacRoleMapping` 参数配置角色映射，以将 Identity Center 组映射到 Argo CD 角色，而不是通过 Argo CD 的 `argocd-rbac-cm` ConfigMap 进行配置。Argo CD 用户界面通过其自身的直接 URL 托管（可在 EKS 控制台中集群的“功能”选项卡下找到），API 访问则通过 IAM 使用 Amazon 身份验证和授权。

集群配置：该功能不会自动配置本地集群或轴辐式拓扑。您需要自行配置部署目标集群和 EKS 访问条目。该功能仅支持将 Amazon EKS 集群作为部署目标，且必须使用 EKS 集群 ARN（而非 Kubernetes API 服务器 URL）。该功能不会自动将本地集群 (`kubernetes.default.svc`) 添加为部署目标，若要部署到创建该功能的同一集群，请使用其 ARN 明确注册该集群。

简化的远程集群访问：该功能使用 EKS 访问条目来授予 Argo CD 远程集群访问权限，从而简化了多集群部署，无需配置服务账户的 IAM 角色（IRSA）或设置跨账户 IAM 角色代入。该功能还支持对完全私有的 EKS 集群进行透明访问，无需进行 VPC 对等连接或专门的网络配置，Amazon 会自动管理 Argo CD 功能与私有远程集群之间的连接。

直接的 Amazon 服务集成：该功能通过功能角色的 IAM 权限，提供与 Amazon 服务的直接集成。您可以直接在 Application 资源中引用 CodeCommit 存储库、ECR Helm 图表和 CodeConnections，而无需创建存储库配置。这简化了身份验证，无需为 Amazon 服务管理单独的凭证。有关详细信息，请参阅 the section called “配置存储库”。

命名空间支持：该功能最初仅支持将应用程序部署到单个命名空间，在创建此功能时需指定该命名空间。未来版本有望支持将应用程序部署到多个命名空间。

不支持的功能：托管功能中不提供以下功能：

- 用于生成自定义清单的 Config 管理插件（CMP）
- 用于资源运行状况评测的自定义 Lua 脚本（支持标准资源的内置运行状况检查）
- 通知控制器
- Argo CD 图像更新程序
- 自定义 SSO 提供者（仅支持 Amazon Identity Center，包括通过 Amazon Identity Center 提供的第三方联合身份）
- 用户界面扩展和自定义横幅
- 直接访问 `argocd-cm`、`argocd-params` 和其他配置 ConfigMaps

兼容性：Applications 和 ApplicationSets 的工作方式与上游 Argo CD 完全相同，清单无需更改。该功能使用同一 Kubernetes API 和 CRD，因此 `kubectl` 等工具的使用方式将保持不变。该功能完全支持

Applications 和 ApplicationSets、带自动同步的 GitOps 工作流程、多集群部署、同步策略（自动化、修剪、自我修复）、同步波次和钩子、标准 Kubernetes 资源的运行状况评估、回滚功能、Git 存储库源（HTTPS 和 SSH）、Helm、Kustomize 和原生 YAML 清单、GitHub 应用程序凭证、用于多租户的项目，以及资源排除项和包含项。

将 Argo CD CLI 与托管功能结合使用

对于大多数操作，Argo CD CLI 的工作方式与上游 Argo CD 相同，但身份验证和集群注册有所不同。

先决条件

按照上游安装说明，安装 Argo CD CLI。

配置

使用环境变量配置 CLI：

1. 从 EKS 控制台（在集群的功能选项卡下）或使用 Amazon CLI 获取 Argo CD 服务器 URL：

```
export ARGOCD_SERVER=$(aws eks describe-capability \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.configuration.argoCd.serverUrl' \
  --output text \
  --region region-code)
```

2. 从 Argo CD 用户界面（设置 → 账户 → 管理员 → 生成新令牌）生成账户令牌，然后将其设置为环境变量：

```
export ARGOCD_AUTH_TOKEN="your-token-here"
```

> **Important**
>
> 此配置使用管理员账户令牌进行初始设置和开发工作流程。对于生产使用案例，为遵循最低权限原则，请使用项目作用域内的角色和令牌。有关配置项目角色和 RBAC 的更多信息，请参阅the section called “配置权限”。

1. 设置必需的 gRPC 选项：

```
export ARGOCD_OPTS="--grpc-web"
```

设置这些环境变量后，无需使用 `argocd login` 命令即可使用 Argo CD CLI。

主要区别

托管功能具有以下 CLI 限制：

- 不支持 `argocd admin` 命令 [这些命令需要直接访问容器组（pod）]
- 不支持 `argocd login`（改用账户或项目令牌）
- `argocd cluster add` 需要 `--aws-cluster-name` 标志并指定 EKS 集群 ARN

示例：注册集群

注册 EKS 集群以进行应用程序部署：

```
# Get the cluster ARN
CLUSTER_ARN=$(aws eks describe-cluster \
  --name my-cluster \
  --query 'cluster.arn' \
  --output text)

# Register the cluster
argocd cluster add $CLUSTER_ARN \
  --aws-cluster-name $CLUSTER_ARN \
  --name in-cluster \
  --project default
```

有关完整的 Argo CD CLI 文档，请参阅 Argo CD CLI 参考。

迁移路径

您可以从自主管理型 Argo CD 迁移到托管功能：

1. 检查当前的 Argo CD 配置中是否存在不支持的功能（通知控制器、CMP、自定义运行状况检查）
2. 将自主管理型 Argo CD 控制器缩减到零副本
3. 在集群上创建 Argo CD 功能资源
4. 导出现有的 Applications、ApplicationSets 和 AppProjects

5. 迁移存储库凭证、集群密钥和存储库凭证模板（repocreds）
6. 如果使用 GPG 密钥、TLS 证书或 SSH 已知主机，请同时迁移这些配置
7. 更新 `destination.server` 字段以使用集群名称或 EKS 集群 ARN
8. 将其应用于 Argo CD 托管实例
9. 验证应用程序是否正确同步
10.停用自行管理型 Argo CD 安装

托管功能使用相同的 Argo CD API 和资源定义，因此只需最少的修改即可使用现有清单。

后续步骤

- the section called “创建 Argo CD 功能”：创建 Argo CD 功能资源
- the section called “使用 Argo CD”：部署第一个应用程序
- the section called “Argo CD 注意事项”：配置 Amazon Identity Center 集成

使用 kro（Kube Resource Orchestrator）组合编排资源

kro（Kube Resource Orchestrator）是一个 Kubernetes 原生开源项目，支持通过简洁直观的配置来定义自定义 Kubernetes API。借助 kro，您可以轻松配置全新的自定义 API，这些 API 可用于创建一组 Kubernetes 对象，并定义对象之间的逻辑操作。

基于 EKS 功能，kro 由 Amazon 提供全托管服务，无需您在集群中安装、维护和扩缩 kro 控制器。

kro 工作原理

kro 引入了一种名为 `ResourceGraphDefinition`（RGD）的自定义资源定义（CRD），支持以简洁高效的方式创建自定义 Kubernetes API。在您创建 `ResourceGraphDefinition` 后，kro 会借助 Kubernetes 原生扩展能力，在集群中创建并管理这些新的 API。基于这份单一的资源规范，kro 会自动根据您的配置创建并注册一个全新的 CRD，同时自适应地管理您所定义的这些新自定义资源。

RGD 可包含多种资源，kro 会自动判断资源间的依赖关系与创建顺序，无需人工干预。您能够通过简单的语法，将配置信息从一个资源注入另一个资源，极大简化了资源组合编排流程，不再需要在集群中部署用于资源衔接的“粘合型”控制器。借助 kro，您所定义的自定义资源既可以包含 Kubernetes 原生资源，也能够兼容集群中已安装的任意自定义资源定义（CRD）。

kro 仅支持一种核心资源类型：

- ResourceGraphDefinition (RGD)：定义一种 Kubernetes 自定义资源，该资源可封装一个或多个底层的 Kubernetes 原生资源或自定义资源

除该核心资源外，kro 还会负责创建并管理由其生成的自定义资源的生命周期，同时管理这些自定义资源所包含的全部组成资源。

kro 可与 Amazon Controllers for Kubernetes（ACK）无缝集成，支持您将工作负载资源与 Amazon 资源进行组合，从而创建更高级别的抽象资源。此能力可帮助您构建专属的云基础设施组件，简化资源管理流程，并能基于企业标准配置默认值与不可变配置项，形成可复用的模式。

kro 优势

kro 支持平台团队创建自定义 Kubernetes API，将多个资源组合为更高级别的抽象资源。这一能力可以简化资源管理工作，开发者只需通过简洁、标准化且带版本控制的自定义资源，就能完成复杂应用的部署。您可以为常用的资源组合定义可复用模式，在企业内部实现一致化的资源创建流程。

kro 在 Kubernetes 环境中采用公共表达式语言（CEL）实现资源间的值传递与条件逻辑配置，让资源组合编排具备更强的灵活性。您既可以组合 Kubernetes 原生资源，也能将 ACK 管理的 Amazon 资源纳入其中，构建出一体化的自定义 API，从而实现应用与基础设施的完整定义。

kro 支持通过 Kubernetes 清单完成声明式配置，可无缝对接 GitOps 工作流程与基础设施即代码实践，与您现有的开发流程深度融合。作为 EKS 托管功能的一部分，kro 由 Amazon 提供全托管服务，无需您在集群中安装、配置和维护 kro 控制器。

示例：创建 ResourceGraphDefinition

以下示例展示了一个简单的 `ResourceGraphDefinition` 配置，用于创建一个包含 Deployment 和 Service 的 Web 应用：

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: web-application
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    spec:
      name: string
      replicas: integer | default=3
  resources:
```

```
    - id: deployment
      template:
        apiVersion: apps/v1
        kind: Deployment
        metadata:
          name: ${schema.spec.name}
        spec:
          replicas: ${schema.spec.replicas}
    - id: service
      template:
        apiVersion: v1
        kind: Service
        metadata:
          name: ${schema.spec.name}
```

当用户创建 WebApplication 自定义资源实例时，kro 会自动生成对应的 Deployment 与 Service 资源，并将其生命周期与自定义资源进行统一管理。

与其他 EKS 托管功能集成

kro 与其他 EKS 托管功能集成。

- Amazon Controllers for Kubernetes（ACK）：借助 kro 将 ACK 资源组合为更高级别的抽象资源，简化 Amazon 资源的管理流程。
- Argo CD：通过 Argo CD 管理 kro 自定义资源在多集群的部署，为平台基础设施组件与应用栈启用 GitOps 工作流程。

kro 入门

若要开始使用 EKS 托管型 kro，可按照以下步骤操作：

1. 通过 Amazon 控制台、Amazon CLI 或您偏好的基础设施即代码工具，在 EKS 集群上创建 kro 功能资源。
2. 创建 ResourceGraphDefinition（RGD），定义自己的自定义 API 与资源组合编排规则。
3. 应用自定义资源实例，完成底层 Kubernetes 资源与 Amazon 资源的配置和管理。

创建 kro 功能

本主题旨在介绍如何在 Amazon EKS 集群上创建 kro 功能。

先决条件

创建 kro 功能之前，确保满足以下条件：

- 运行受支持 Kubernetes 版本的现有 Amazon EKS 集群（标准支持与扩展支持范围内的所有版本均适用）
- 可在 EKS 集群上创建功能资源的充足 IAM 权限
- （适用于 CLI/eksctl）已安装并配置相应 CLI 工具

> **Note**
>
> 与 ACK 和 Argo CD 不同，kro 除信任策略外，无需额外的 IAM 权限。kro 完全在集群中运行，不会发起 Amazon API 调用。不过，您仍需提供一个配置了相应信任策略的 IAM 功能角色。有关配置 Kubernetes RBAC 权限的信息，请参阅the section called “配置权限”。

选择工具

您可以使用Amazon Web Services 管理控制台、Amazon CLI 或 eksctl 创建 kro 功能：

- the section called “控制台”：使用控制台获得引导式体验
- the section called “ Amazon CLI”：使用 Amazon CLI 进行脚本编写和自动化
- the section called “eksctl”：使用 eksctl 获得 Kubernetes 原生体验

在创建 kro 功能时会执行的操作

在创建 kro 功能时：

1. EKS 会创建 kro 功能服务，并配置该服务来监控和管理集群内的资源
2. 自定义资源定义（CRD）将被安装到集群中
3. 该功能会代入您提供的 IAM 功能角色（此角色仅用于信任关系配置）
4. kro 开始监听 `ResourceGraphDefinition` 资源及其对应的实例
5. 功能状态从 `CREATING` 更改为 `ACTIVE`

功能进入运行状态后，即可创建 ResourceGraphDefinition，以此定义自定义 API，并创建这些 API 对应的实例。

后续步骤

创建 kro 功能后：

- the section called “kro 概念”：了解 kro 的基本概念及资源组合编排
- the section called “kro 概念”：了解 SimpleSchema、CEL 表达式及资源组合编排模式

使用控制台创建 kro 功能

本主题旨在介绍如何使用 Amazon Web Services 管理控制台 创建 kro（Kube Resource Orchestrator）功能。

创建 kro 功能

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称，打开集群详细信息页面。
3. 选择功能选项卡。
4. 在左侧导航栏中，选择 kro（Kube Resource Orchestrator）。
5. 选择创建 kro 功能。
6. 对于 IAM 功能角色：
   - 如果已有 IAM 功能角色，请从下拉列表中选择该角色
   - 如需创建角色，请选择创建 kro 角色

     此操作会在新选项卡中打开 IAM 控制台，其中包含预填充的信任策略。该角色无需配置额外的 IAM 权限，因为 kro 的所有操作均在集群内部完成。

     创建角色后，返回 EKS 控制台，系统将自动选择该角色。

     > **Note**
     >
     > 与 ACK 和 Argo CD 不同，kro 除信任策略外，无需额外的 IAM 权限。kro 完全在集群中运行，不会发起 Amazon API 调用。
7. 选择创建。

功能创建过程随即开始。

验证功能是否处于活动状态

1. 在功能选项卡上，查看 kro 功能状态。
2. 等待状态从 CREATING 更改为 ACTIVE。
3. 变为活动状态后，该功能即可使用。

有关功能状态和问题排查的信息，请参阅the section called “使用功能”。

授予 Kubernetes 资源管理权限

默认情况下，kro 仅能创建和管理 ResourceGraphDefinition 及其对应的实例。若要允许 kro 创建和管理 ResourceGraphDefinition 中定义的底层 Kubernetes 资源，需将 AmazonEKSClusterAdminPolicy 访问策略关联到该功能的访问条目。

1. 在 EKS 控制台中，导航到 EKS 集群的访问选项卡。
2. 在访问条目下，找到对应 kro 功能角色的条目（将包含之前创建的角色 ARN）。
3. 选择访问条目，打开对应的详细信息页面。
4. 在访问策略部分，选择关联访问策略。
5. 从策略列表中选择 AmazonEKSClusterAdminPolicy。
6. 在访问范围中，选择集群。
7. 选择关联。

> **Important**
>
> AmazonEKSClusterAdminPolicy 会授予宽泛的权限，支持创建和管理所有 Kubernetes 资源，该策略的设计目的是简化功能的入门流程。在生产环境使用时，请创建权限范围更严格的 RBAC 策略，仅授予 ResourceGraphDefinition 需管理的特定资源对应的权限。有关配置最低权限的指引，请参阅the section called “配置权限”和 the section called “EKS 功能的注意事项”。

验证自定义资源是否可用

待该功能处于激活状态后，验证 kro 自定义资源是否已在集群中正常可用。

使用控制台

1. 在 Amazon EKS 控制台中导航至集群
2. 选择资源选项卡
3. 选择扩展
4. 选择 CustomResourceDefinitions

您应能看到 `ResourceGraphDefinition` 这一资源类型已被列出。

使用 kubectl

```
kubectl api-resources | grep kro.run
```

您应能看到 `ResourceGraphDefinition` 这一资源类型已被列出。

后续步骤

- the section called “kro 概念”：了解 kro 的基本概念及资源组合编排
- the section called “kro 概念”：了解 SimpleSchema、CEL 表达式和组合编排模式
- the section called “使用功能”：管理 kro 功能资源

使用 Amazon CLI 创建 kro 功能

本主题旨在介绍如何使用 Amazon CLI 创建 kro（Kube Resource Orchestrator）功能。

先决条件

- Amazon CLI：版本 `2.12.3` 或更高版本。要检查版本，请运行 `aws --version`。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的安装。
- **`kubectl`**：用于与 Kubernetes 集群结合使用的命令行工具。有关更多信息，请参阅 the section called “设置 kubectl 和 eksctl”。

步骤 1：创建 IAM 功能角色

创建信任策略文件：

```
cat > kro-trust-policy.json << 'EOF'
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

创建 IAM 角色：

```
aws iam create-role \
  --role-name KROCapabilityRole \
  --assume-role-policy-document file://kro-trust-policy.json
```

> **Note**
>
> 与 ACK 和 Argo CD 不同，kro 无需额外的 IAM 权限。kro 完全在集群中运行，不会发起 Amazon API 调用。只有在与 EKS 功能服务建立信任关系时，才需要该角色。

步骤 2：创建 kro 功能

在集群上创建 kro 功能资源。将 *region-code* 替换为集群所在的 Amazon 区域（例如 us-west-2），并将 *my-cluster* 替换为集群的实际名称。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --type KRO \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/KROCapabilityRole \
  --delete-propagation-policy RETAIN
```

命令会立即返回，但是由于 EKS 正在创建所需的功能基础设施和组件，该功能需要一些时间才能变为活动状态。在创建集群时，EKS 将在集群中安装与此功能相关的 Kubernetes 自定义资源定义。

> **Note**
>
> 如果收到集群不存在或您没有权限的错误消息，请验证：
>
> - 集群名称是否正确
> - Amazon CLI 是否针对正确的区域进行配置
> - 您是否拥有所需的 IAM 权限

步骤 3：验证功能是否处于活动状态

等待功能变为活动状态。将 *region-code* 替换为您的集群所在的 Amazon 区域，并将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --query 'capability.status' \
  --output text
```

当状态显示为 ACTIVE 时，表示功能已准备就绪。

您也可以查看完整功能详细信息：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro
```

步骤 4：授予 Kubernetes 资源管理权限

默认情况下，kro 仅能创建和管理 ResourceGraphDefinition 及其对应的实例。若要允许 kro 创建和管理 ResourceGraphDefinition 中定义的底层 Kubernetes 资源，需将 AmazonEKSClusterAdminPolicy 访问策略关联到该功能的访问条目。

获取功能角色的 ARN：

```
CAPABILITY_ROLE_ARN=$(aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --query 'capability.roleArn' \
  --output text)
```

关联集群管理员策略：

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn $CAPABILITY_ROLE_ARN \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Important**
>
> AmazonEKSClusterAdminPolicy 会授予宽泛的权限，支持创建和管理所有 Kubernetes 资源，该策略的设计目的是简化功能的入门流程。在生产环境使用时，请创建权限范围更严格的 RBAC 策略，仅授予 ResourceGraphDefinition 需管理的特定资源对应的权限。有关配置最低权限的指引，请参阅the section called “配置权限”和 the section called “EKS 功能的注意事项”。

步骤 5：验证自定义资源是否可用

待该功能处于激活状态后，验证 kro 自定义资源是否已在集群中正常可用：

```
kubectl api-resources | grep kro.run
```

您应能看到 ResourceGraphDefinition 这一资源类型已被列出。

后续步骤

- the section called “kro 概念”：了解 kro 的基本概念及资源组合编排
- the section called “kro 概念”：了解 SimpleSchema、CEL 表达式和组合编排模式
- the section called “使用功能”：管理 kro 功能资源

使用 eksctl 创建 kro 功能

本主题旨在介绍如何使用 eksctl 创建 kro（Kube Resource Orchestrator）功能。

> **Note**
>
> 以下步骤需要 eksctl 版本 `0.220.0` 或更高版本。要检查版本，请运行 `eksctl version`。

步骤 1：创建 IAM 功能角色

创建信任策略文件：

```
cat > kro-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

创建 IAM 角色：

```
aws iam create-role \
  --role-name KROCapabilityRole \
  --assume-role-policy-document file://kro-trust-policy.json
```

> **Note**
>
> 与 ACK 和 Argo CD 不同，kro 除信任策略外，无需额外的 IAM 权限。kro 完全在集群中运行，不会发起 Amazon API 调用。

步骤 2：创建 kro 功能

使用 eksctl 创建 kro 功能。将 *region-code* 替换为您的集群所在的 Amazon 区域，并将 *my-cluster* 替换为您的集群的名称。

```
eksctl create capability \
  --region region-code \
  --cluster my-cluster \
  --name my-kro \
  --type KRO \
  --role-arn arn:aws:iam::[.replaceable]111122223333:role/KROCapabilityRole
```

命令会立即返回，但该功能需要一些时间才能变为活动状态。

步骤 3：验证功能是否处于活动状态

检查功能状态。将 *region-code* 替换为您的集群所在的 Amazon 区域，并将 *my-cluster* 替换为您的集群的名称。

```
eksctl get capability \
  --region region-code \
  --cluster my-cluster \
  --name my-kro
```

当状态显示为 ACTIVE 时，表示功能已准备就绪。

步骤 4：授予 Kubernetes 资源管理权限

默认情况下，kro 仅能创建和管理 ResourceGraphDefinition 及其对应的实例。若要允许 kro 创建和管理 ResourceGraphDefinition 中定义的底层 Kubernetes 资源，需将 AmazonEKSClusterAdminPolicy 访问策略关联到该功能的访问条目。

获取功能角色的 ARN：

```
CAPABILITY_ROLE_ARN=$(aws eks describe-capability \
  --region region-code \
  --cluster my-cluster \
  --name my-kro \
  --query 'capability.roleArn' \
  --output text)
```

关联集群管理员策略：

```
aws eks associate-access-policy \
  --region region-code \
  --cluster my-cluster \
  --principal-arn $CAPABILITY_ROLE_ARN \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

Important

AmazonEKSClusterAdminPolicy 会授予宽泛的权限，支持创建和管理所有 Kubernetes 资源，该策略的设计目的是简化功能的入门流程。在生产环境使用时，请创建权限范围更严格的 RBAC 策略，仅授予 ResourceGraphDefinition 需管理的特定资源对应的权限。有关配置最低权限的指引，请参阅the section called “配置权限”和 the section called “EKS 功能的注意事项”。

步骤 5：验证自定义资源是否可用

待该功能处于激活状态后，验证 kro 自定义资源是否已在集群中正常可用：

```
kubectl api-resources | grep kro.run
```

您应能看到 ResourceGraphDefinition 这一资源类型已被列出。

后续步骤

- the section called “kro 概念”：了解 kro 的基本概念及资源组合编排
- the section called “kro 概念”：了解 SimpleSchema、CEL 表达式和组合编排模式
- the section called “使用功能”：管理 kro 功能资源

kro 概念

kro 支持平台团队创建自定义 Kubernetes API，将多个资源组合为更高级别的抽象资源。本主题会先引导您完成一个实操案例的演练，再详细阐释使用 EKS 托管型 kro 所需掌握的核心概念。

kro 入门

创建 kro 托管功能后（详见the section called “创建 kro 功能”文档），即可开始在集群中通过 ResourceGraphDefinition 来创建自定义 API。

以下完整示例用于创建简单的 Web 应用程序抽象资源：

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: webapplication
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    group: kro.run
    spec:
      name: string | required=true
      image: string | default="nginx:latest"
      replicas: integer | default=3
  resources:
  - id: deployment
    template:
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: ${schema.spec.name}
      spec:
        replicas: ${schema.spec.replicas}
        selector:
          matchLabels:
            app: ${schema.spec.name}
        template:
          metadata:
            labels:
              app: ${schema.spec.name}
          spec:
            containers:
            - name: app
              image: ${schema.spec.image}
              ports:
              - containerPort: 80
  - id: service
    template:
      apiVersion: v1
      kind: Service
      metadata:
        name: ${schema.spec.name}
      spec:
```

```
        selector:
          app: ${schema.spec.name}
        ports:
        - protocol: TCP
          port: 80
          targetPort: 80
```

应用该 ResourceGraphDefinition 后，应用团队即可通过您定义的简化版 API 来创建 Web 应用程序：

```
apiVersion: kro.run/v1alpha1
kind: WebApplication
metadata:
  name: my-app
spec:
  name: my-app
  replicas: 5
```

kro 会自动创建带有相应配置的 Deployment 和 Service 资源。由于配置中未指定 `image` 参数，系统会采用模式定义中预设的默认值 `nginx:latest`。

核心概念

> **Important**
>
> kro 会在创建阶段对 ResourceGraphDefinition 进行验证，而非在运行时阶段。当您创建一个 RGD 时，kro 会执行以下验证操作：校验 CEL 语法的正确性、基于实际的 Kubernetes 模式对表达式进行类型检查、验证字段是否存在、检测是否存在循环依赖。这意味着相关错误会在创建 RGD 时被立即发现，无需等到部署任何实例之后。

ResourceGraphDefinition

ResourceGraphDefinition（RGD）通过配置以下内容来定义一种 Kubernetes 自定义 API：

- Schema：采用 SimpleSchema 格式定义的 API 结构（字段名称、字段类型、默认值及验证规则）
- Resources：待创建的底层 Kubernetes 资源或 Amazon 资源的模板
- Dependencies：资源之间的关联方式（会通过字段引用自动识别）

当您应用一份 RGD 配置后，kro 会在集群中注册一个全新的自定义资源定义（CRD）。之后应用团队即可创建该自定义 API 的实例，由 kro 负责创建并管理所有对应的底层资源。

有关更多信息，请参阅 kro 文档中的 ResourceGraphDefinition Overview。

SimpleSchema 格式

SimpleSchema 提供了一种简化的 API 模式定义方式，无需使用者掌握 OpenAPI 相关知识：

```
schema:
  apiVersion: v1alpha1
  kind: Database
  spec:
    name: string | required=true description="Database name"
    size: string | default="small" enum=small,medium,large
    replicas: integer | default=1 minimum=1 maximum=5
```

SimpleSchema 支持 `string`、`integer`、`boolean` 和 `number` 等类型，并可配置 `required`、`default`、`minimum/maximum`、`enum` 和 `pattern` 等约束规则。

有关更多信息，请参阅 kro 文档中的 SimpleSchema。

CEL 表达式

kro 采用公共表达式语言（CEL）来实现值的动态引用和条件逻辑的添加。CEL 表达式需要包裹在 `${` 与 `}` 之内，主要有两种使用方式：

独立表达式：字段的完整值由单个表达式构成：

```
spec:
  replicas: ${schema.spec.replicaCount}  # Expression returns integer
  labels: ${schema.spec.labelMap}        # Expression returns object
```

表达式的计算结果会直接替换整个字段的值，且结果类型必须与字段预期的类型相匹配。

字符串模板：在字符串中嵌入一个或多个表达式：

```
metadata:
  name: "${schema.spec.prefix}-${schema.spec.name}"  # Multiple expressions
  annotation: "Created by ${schema.spec.owner}"      # Single expression in string
```

字符串模板中的所有表达式必须返回字符串类型。若要转换其他类型，可使用 `string()` 函数，例如：`"replicas-${string(schema.spec.count)}"`。

字段引用：通过 `schema.spec` 访问实例的规格参数：

```
template:
  metadata:
    name: ${schema.spec.name}-deployment
    namespace: ${schema.metadata.namespace}  # Can also reference metadata
  spec:
    replicas: ${schema.spec.replicas}
```

可选字段访问：对于可能不存在的字段，可使用？符号，例如：

```
# For ConfigMaps or Secrets with unknown structure
value: ${configmap.data.?DATABASE_URL}

# For optional status fields
ready: ${deployment.status.?readyReplicas > 0}
```

若目标字段不存在，表达式会返回 null 而非直接执行失败。

条件式资源：仅在满足指定条件时才创建资源：

```
resources:
- id: ingress
  includeWhen:
    - ${schema.spec.enableIngress == true}
  template:
    # ... ingress configuration
```

includeWhen 字段会接收一份布尔表达式列表。只有所有表达式结果均为 true，对应的资源才会被创建。目前 includeWhen 仅支持引用 schema.spec 中的字段。

数据转换：通过三元运算符与函数实现值的转换处理：

```
template:
  spec:
    resources:
      requests:
        memory: ${schema.spec.size == "small" ? "512Mi" : "2Gi"}

    # String concatenation
    image: ${schema.spec.registry + "/" + schema.spec.imageName}

    # Type conversion
```

```
    port: ${string(schema.spec.portNumber)}
```

跨资源引用：引用其他资源的值：

```
resources:
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-data

- id: configmap
  template:
    apiVersion: v1
    kind: ConfigMap
    data:
      BUCKET_NAME: ${bucket.spec.name}
      BUCKET_ARN: ${bucket.status.ackResourceMetadata.arn}
```

当您在 CEL 表达式中引用另一资源时，kro 会自动创建资源间的依赖关系，并确保被引用的资源优先完成创建。

有关更多信息，请参阅 kro 文档中的 CEL Expressions。

资源依赖关系

kro 会通过 CEL 表达式自动推导资源间的依赖关系，无需您手动指定资源的创建顺序，只需描述资源之间的关联逻辑即可。当某一资源通过 CEL 表达式引用另一资源时，kro 会自动创建两者的依赖关系，并确定正确的资源创建顺序。

```
resources:
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-data

- id: notification
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: BucketNotification
```

```
    spec:
      bucket: ${bucket.spec.name}  # Creates dependency: notification depends on bucket
```

表达式 `${bucket.spec.name}` 会触发依赖关系的生成。kro 会为所有资源及其依赖关系构建一个有向无环图（DAG），再据此计算出资源的拓扑创建顺序。

创建顺序：资源会按照拓扑顺序进行创建（被依赖的资源优先创建）。

并行创建：无依赖关系的资源会被同时创建。

删除顺序：资源会按照拓扑顺序的逆序进行删除（依赖方优先被删除）。

循环依赖：不被允许。kro 会在 ResourceGraphDefinition 验证阶段，直接拒绝存在循环依赖的配置。

您可以查看系统计算出的资源创建顺序：

```
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.topologicalOrder}'
```

有关更多信息，请参阅 kro 文档中的 Graph inference。

与 ACK 的组合编排

kro 可与 EKS 托管型 ACK 无缝协同，实现 Amazon 资源与 Kubernetes 资源的组合编排：

```
resources:
# Create {aws} S3 bucket with ACK
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-files

# Inject bucket details into Kubernetes ConfigMap
- id: config
  template:
    apiVersion: v1
    kind: ConfigMap
    data:
      BUCKET_NAME: ${bucket.spec.name}
      BUCKET_ARN: ${bucket.status.ackResourceMetadata.arn}

# Use ConfigMap in application deployment
```

```
- id: deployment
  template:
    apiVersion: apps/v1
    kind: Deployment
    spec:
      template:
        spec:
          containers:
          - name: app
            envFrom:
            - configMapRef:
                name: ${config.metadata.name}
```

这种模式支持您创建 Amazon 资源、提取其详细信息（如 ARN、URL、端点）并注入应用配置，且所有操作均作为单一单元进行管理。

有关更多组合编排模式及进阶示例，请参阅 the section called “注意事项”。

后续步骤

- the section called “注意事项”：了解 EKS 专属模式、RBAC 以及与 ACK、Argo CD 的集成方法
- kro 文档：包含进阶 CEL 表达式、验证模式及问题排查方案的完整 kro 文档

配置 kro 权限

与 ACK 和 Argo CD 不同，kro 无需额外的 IAM 权限。kro 完全在 Kubernetes 集群中运行，不会发起 Amazon API 调用。可通过标准的 Kubernetes RBAC 机制管控对 kro 资源的访问权限。

kro 权限的工作原理

kro 涉及两种作用域不同的 Kubernetes 资源：

ResourceGraphDefinition：属于集群级资源，用于定义自定义 API。这类资源通常由平台团队负责管理，平台团队会基于组织内的标准规范完成资源的设计与维护工作。

实例：属于命名空间级资源，基于 ResourceGraphDefinition 创建而来。拥有对应 RBAC 权限的应用团队，即可创建这类实例。

默认情况下，kro 功能会通过 `AmazonEKSKROPolicy` 访问条目策略，获得管理 ResourceGraphDefinition 及其实例的权限。但如果需要让 kro 创建和管理 ResourceGraphDefinition 中定义的底层 Kubernetes 资源（例如 Deployment、Service 或 ACK 资源），则需要为其配置额外权

限。您可通过访问条目策略或 Kubernetes RBAC 两种方式授予这些权限。有关授予这些权限的详细信息，请参阅 kro 自定义资源权限。

平台团队权限

平台团队需要具备创建和管理 ResourceGraphDefinition 的权限。

平台团队的 ClusterRole 示例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-platform-admin
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
  verbs: ["*"]
```

绑定到平台团队成员：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: platform-team-kro-admin
subjects:
- kind: Group
  name: platform-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: kro-platform-admin
  apiGroup: rbac.authorization.k8s.io
```

应用团队权限

应用团队需要具备在其所属命名空间内创建自定义资源实例的权限。

应用团队的角色示例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
```

```
  name: kro-app-developer
  namespace: my-app
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps", "databases"]
  verbs: ["create", "get", "list", "update", "delete", "patch"]
```

绑定到应用团队成员：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-team-kro-developer
  namespace: my-app
subjects:
- kind: Group
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: kro-app-developer
  apiGroup: rbac.authorization.k8s.io
```

> **Note**
>
> Role 配置中的 API 组（本例中为 `kro.run`）必须与 ResourceGraphDefinition 模式文件里定义的 `apiVersion` 保持一致。

只读访问权限

授予只读访问权限，允许相关人员查看 ResourceGraphDefinition 及实例，且不赋予修改权限。

只读 ClusterRole：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-viewer
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
```

```
  verbs: ["get", "list", "watch"]
```

实例的只读 Role：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kro-instance-viewer
  namespace: my-app
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps", "databases"]
  verbs: ["get", "list", "watch"]
```

多命名空间访问权限

借助 ClusterRole 搭配 RoleBinding，为应用团队授予多命名空间的访问权限。

多命名空间访问的 ClusterRole：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-multi-namespace-developer
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps"]
  verbs: ["create", "get", "list", "update", "delete"]
```

绑定到指定命名空间：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-team-dev-access
  namespace: development
subjects:
- kind: Group
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
```

```
  name: kro-multi-namespace-developer
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-team-staging-access
  namespace: staging
subjects:
- kind: Group
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: kro-multi-namespace-developer
  apiGroup: rbac.authorization.k8s.io
```

最佳实践

最低权限原则：仅为每个团队授予完成其职责所需的最小权限。

优先使用用户组而非独立用户：将角色绑定到用户组而非单个用户，便于权限的统一管理。

分离平台团队与应用团队的职责边界：平台团队负责管理 ResourceGraphDefinition，应用团队负责管理实例。

命名空间隔离机制：利用命名空间实现不同团队或环境的隔离，并通过 RBAC 管控各命名空间的访问权限。

审计场景的只读访问权限：为安全与合规团队配置只读访问权限，满足审计工作的需求。

后续步骤

- the section called “kro 概念”：了解 kro 的基本概念及资源组合编排
- the section called “kro 概念”：了解 SimpleSchema、CEL 表达式和组合编排模式
- the section called “EKS 功能的注意事项”：查看针对功能的安全最佳实践

EKS 托管型 kro 注意事项

本主题旨在介绍使用 EKS 托管型 kro 的重要注意事项，包括资源组合编排的适用场景、RBAC 配置模式以及与其他 EKS 托管功能的集成方法。

kro 的适用场景

kro 旨在创建可复用的基础设施模式与自定义 API，以简化复杂的资源管理工作。

需要使用 kro 的场景：

- 为应用团队构建带有简化 API 的自助式平台
- 在团队间统一基础设施模式（如数据库 + 备份 + 监控的组合）
- 管理资源间的依赖关系，并实现资源间的值传递
- 构建自定义抽象资源，隐藏底层的实现复杂度
- 将多个 ACK 资源组合为更高级别的基础设施组件
- 为复杂的基础设施栈启用 GitOps 工作流程

无需使用 kro 的场景：

- 管理简单的独立资源（直接使用 ACK 或 Kubernetes 原生资源）
- 需要动态运行时逻辑（kro 为声明式框架，不支持命令式操作）
- 资源之间不存在依赖关系或共享配置

kro 的优势在于构建标准化部署流程，即提供预设规则、可复用模式，帮助各团队轻松、合规地部署复杂的基础设施。

RBAC 模式

kro 可实现职责分离，将创建 ResourceGraphDefinition 的平台团队与创建实例的应用团队的工作边界清晰划分。

平台团队职责

平台团队负责创建并维护用于定义自定义 API 的 ResourceGraphDefinition（RGD）。

所需权限：

- 创建、更新、删除 ResourceGraphDefinition
- 管理各类底层资源（Deployment、Service、ACK 资源）
- 拥有 RGD 待使用的所有命名空间的访问权限

平台团队的 ClusterRole 示例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: platform-kro-admin
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
  verbs: ["*"]
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
```

有关 RBAC 配置的详细信息，请参阅the section called “配置权限”。

应用团队职责

应用团队负责创建由 RGD 定义的自定义资源实例，无需理解底层实现的复杂逻辑。

所需权限：

- 创建、更新、删除自定义资源实例
- 拥有其所在命名空间的读取权限
- 不授予底层资源或 RGD 的访问权限

优点：

- 团队可直接使用简洁的高层级 API
- 平台团队把控底层实现细节
- 降低配置错误的风险加快
- 新团队成员的上手速度

与其他 EKS 功能集成

ACK 资源组合编排

kro 与 ACK 结合使用时，在创建基础设施模式方面的能力尤为突出。

常见模式：

- 带存储功能的应用程序：S3 存储桶 + SQS 队列 + 通知配置
- 数据库资源栈：RDS 实例 + 参数组 + 安全组 + Secrets Manager 密钥
- 联网资源：VPC + 子网 + 路由表 + 安全组 + NAT 网关
- 带存储功能的计算资源：EC2 实例 + EBS 卷 + IAM 实例配置文件

kro 会自动处理资源的依赖顺序、实现资源间的值传递（如 ARN、连接字符串），并将所有资源作为单一单元进行全生命周期管理。

有关 ACK 资源组合编排的示例，请参阅 the section called “ACK 概念”。

将 Argo CD 与 GitOps 搭配使用

借助适用于 Argo CD 的 EKS 功能，可从 Git 存储库中部署 RGD 及实例。

存储库组织方式：

- 平台存储库：存放由平台团队管理的 ResourceGraphDefinition
- 应用存储库：存放由应用团队管理的自定义资源容器实例
- 共享存储库：小型组织可将 RGD 与实例存放于同一存储库

注意事项：

- 需先部署 RGD，再部署实例（可借助 Argo CD 的同步波功能实现）
- 为平台团队与应用团队分别配置独立的 Argo CD 项目
- 由平台团队管控 RGD 存储库的访问权限
- 为应用团队分配 RGD 定义文件的只读访问权限

有关 Argo CD 的更多信息，请参阅the section called “使用 Argo CD”。

ResourceGraphDefinition 的组织方式

可按照用途、复杂度和归属方对 RGD 进行分类管理。

按用途分类：

- 基础设施：数据库资源栈、联网资源、存储资源

- 应用程序：Web 应用、API 服务、批处理任务
- 平台：共享服务、监控组件、日志组件

按复杂度分类：

- 简单：包含 2-3 个资源，依赖关系极少
- 中等：包含 5-10 个资源，存在部分依赖关系
- 复杂：包含 10 个以上资源，依赖关系复杂

命名规范：

- 采用描述性名称，例如：`webapp-with-database`、`s3-notification-queue`
- 若存在破坏性变更，可在名称中加入版本号，例如：`webapp-v2`
- 为同类型 RGD 添加统一前缀，例如：`platform-` 、`app-`

命名空间策略：

- RGD 为集群级资源（非命名空间级资源）
- 实例为命名空间级资源
- 可在 RGD 中配置命名空间选择器，管控实例允许被创建的命名空间范围

版本控制与更新

需提前规划 RGD 的迭代优化与实例的迁移方案。

RGD 更新：

- 非破坏性变更：直接原地更新 RGD 即可（操作包括新增可选字段、添加带 includeWhen 条件的新资源）
- 破坏性变更：需创建名称不同的新 RGD（例如 webapp-v2）
- 弃用策略：通过注解标记旧版 RGD，并同步告知用户迁移时间规划

实例迁移：

- 基于更新版 RGD 创建新的实例

- 验证新实例运行状态正常
- 删除旧版实例
- kro 会自动处理底层资源的更新操作

最佳实践

- RGD 变更需先在非生产环境完成测试
- 有重大变更时，在 RGD 名称中采用语义版本控制命名规则
- 记录破坏性变更内容及对应的迁移方案
- 为应用团队提供可直接参考的迁移示例

验证与测试

将 RGD 部署到生产环境前，需对其进行验证。

验证策略：

- 模式验证：kro 会自动对 RGD 模式进行验证
- 实例试运行：在开发命名空间中创建测试用实例
- 集成测试：验证组合后的各类资源是否能协同正常工作
- 策略强制执行：通过准入控制器落实组织内部的标准规范

需测试的常见问题：

- 资源依赖关系与创建顺序
- 资源间的值传递逻辑（CEL 表达式）
- 条件式资源的包含规则（includeWhen）
- 底层资源的状态反馈机制
- 实例创建操作对应的 RBAC 权限配置

上游文档

有关使用 kro 的详细信息，请参阅以下内容：

- kro 入门：创建 ResourceGraphDefinition

- CEL 表达式：编写 CEL 表达式
- kro 指南：高级组合编排模式
- 问题排查：问题排查与调试

后续步骤

- the section called “配置权限”：为平台团队与应用团队配置 RBAC
- the section called “kro 概念”：了解 kro 的基本概念及资源生命周期
- the section called “问题排查”：诊断并解决 kro 相关问题
- the section called “ACK 概念”：了解用于资源组合编排的 ACK 资源
- the section called “使用 Argo CD”：通过 GitOps 部署 RGD 及实例

排查 kro 功能问题

本主题旨在介绍 EKS 托管型 kro 问题排查指引，内容涵盖功能运行状况检查、RBAC 权限、CEL 表达式错误以及资源组合编排问题。

> **Note**
>
> EKS 功能完全托管，可在您的集群之外运行。您无法访问控制器日志或 `kro-system` 命名空间。问题排查侧重于功能运行状况、RBAC 配置和资源状态。

功能状态显示为 ACTIVE，但 ResourceGraphDefinition 无法正常工作

若 kro 功能状态显示为 `ACTIVE`，但 ResourceGraphDefinition 无法创建底层资源，请检查功能运行状况、RBAC 权限以及资源状态。

检查功能运行状况：

您可以在 EKS 控制台或使用 Amazon CLI 查看功能运行状况和状态问题。

控制台：

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称。

3. 选择可观测性选项卡。
4. 选择监控集群。
5. 选择功能选项卡，查看所有功能的运行状况和状态。

AmazonCLI：

```
# View capability status and health
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro

# Look for issues in the health section
```

常见原因：

- RBAC 权限缺失：kro 缺少创建底层 Kubernetes 资源的权限
- CEL 表达式无效：ResourceGraphDefinition 中存在语法错误
- 资源依赖异常：被依赖的资源未处于就绪状态
- 模式验证失败：实例不符合 RGD 模式要求

验证 RBAC 权限：

```
# Check if capability has cluster admin policy
kubectl get accessentry -A | grep kro
```

若该功能不具备所需权限，可将 AmazonEKSClusterAdminPolicy 关联到 kro 功能的访问条目；对于生产环境，也可创建权限范围更严格的 RBAC 策略。有关详细信息，请参阅 the section called “配置权限”。

检查 ResourceGraphDefinition 状态：

```
# List all RGDs
kubectl get resourcegraphdefinition

# Describe specific RGD
kubectl describe resourcegraphdefinition my-rgd
```

```
# Check for validation errors
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.conditions}'
```

ResourceGraphDefinition 包含三项关键状态条件：

- `ResourceGraphAccepted`：RGD 是否通过验证（CEL 语法、类型校验、字段存在性验证）
- `KindReady`：自定义 API 对应的 CRD 是否已生成并完成注册
- `ControllerReady`：kro 是否正在主动监听自定义 API 的实例

若 `ResourceGraphAccepted` 状态为 `False`，请查看状态条件信息，排查是否存在未知字段、类型不匹配或循环依赖等验证类错误。

已创建实例但未生成底层资源

若自定义资源实例已存在，但对应的底层 Kubernetes 资源（如 Deployment、Service、ConfigMap）未被创建，请验证 kro 是否具备相应权限，同时排查资源组合编排错误。

检查实例状态：

```
# Describe the instance (replace with your custom resource kind and name)
kubectl describe custom-kind
         my-instance

# View instance events
kubectl get events --field-selector involvedObject.name=my-instance

# Check instance status conditions
kubectl get custom-kind
         my-instance -o jsonpath='{.status.conditions}'

# Check instance state
kubectl get custom-kind
         my-instance -o jsonpath='{.status.state}'
```

实例包含一个 `state` 字段，用于展示其宏观状态：

- `ACTIVE`：实例已成功运行
- `IN_PROGRESS`：实例正被处理或协调
- `FAILED`：实例协调失败

- DELETING：实例正被删除
- ERROR：处理过程中发生错误

实例同时包含四项状态条件：

- InstanceManaged：终结器与标签已正确配置
- GraphResolved：运行时依赖图已创建，且相关资源已完成解析
- ResourcesReady：所有资源均已创建并处于就绪状态
- Ready：实例的整体运行状况（仅当所有子条件均为 True 时，该状态才会变为 True）

可重点关注 Ready 条件来判断实例的运行状况。若 Ready 状态为 False，请检查各子条件，明确是哪个阶段执行失败。

验证 RBAC 权限：

kro 功能需要具备创建 ResourceGraphDefinition 中所定义的底层 Kubernetes 资源的权限。

```
# Check if the capability has the AmazonEKSClusterAdminPolicy
kubectl get accessentry -A | grep kro
```

若存在权限缺失，可将 AmazonEKSClusterAdminPolicy 关联到 kro 功能的访问条目；对于生产环境，则建议创建权限范围更严格的 RBAC 策略。有关详细信息，请参阅 the section called “配置权限”。

CEL 表达式错误

CEL 表达式错误会在 ResourceGraphDefinition 创建阶段被捕获，而非在实例创建阶段。当您创建 RGD 时，kro 会对所有 CEL 语法进行验证、基于 Kubernetes 模式对表达式进行类型检查，同时验证字段是否存在。

常见 CEL 验证错误：

- 未定义字段引用：引用了模式或资源中不存在的字段
- 类型不匹配：表达式返回的类型不符合预期（例如：预期为整数类型，实际返回字符串类型）
- 语法无效：CEL 表达式中存在缺失括号、引号或运算符的情况
- 未知资源类型：引用了集群中未存在的 CRD

检查 RGD 验证状态：

```
# Check if RGD was accepted
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.conditions[?
(@.type=="ResourceGraphAccepted")]}'

# View detailed validation errors
kubectl describe resourcegraphdefinition my-rgd
```

若 ResourceGraphAccepted 状态为 False，对应的条件信息中会包含具体的验证错误信息。

有效的 CEL 表达式示例：

```
# Reference schema field
${schema.spec.appName}

# Conditional expression
${schema.spec.replicas > 1}

# String template (expressions must return strings)
name: "${schema.spec.appName}-service"

# Standalone expression (can be any type)
replicas: ${schema.spec.replicaCount}

# Resource reference
${deployment.status.availableReplicas}

# Optional field access (returns null if field doesn't exist)
${configmap.data.?DATABASE_URL}
```

资源依赖关系无法解析

kro 会从 CEL 表达式中自动推导资源依赖关系，并按正确顺序创建资源。若资源未按预期生成，请检查依赖顺序以及资源的就绪状态。

查看计算出的创建顺序：

```
# See the order kro will create resources
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.topologicalOrder}'
```

该操作会展示基于资源间 CEL 表达式引用关系计算得出的资源创建顺序。

检查资源就绪状态：

```
# View instance status to see which resources are ready
kubectl get custom-kind
         my-instance -o jsonpath='{.status}'

# Check specific resource status
kubectl get deployment my-deployment -o jsonpath='{.status.conditions}'
```

验证 readyWhen 条件（若已配置）：

`readyWhen` 字段为可选项。若未指定该字段，资源在创建完成后会被立即判定为就绪状态。若已定义 `readyWhen` 条件，请验证该条件是否能正确校验资源的就绪状态：

```
resources:
  - id: deployment
    readyWhen:
      - ${deployment.status.availableReplicas == deployment.spec.replicas}
```

检查资源事件：

```
# View events for the underlying resources
kubectl get events -n namespace --sort-by='.lastTimestamp'
```

模式验证失败

若实例因模式验证错误而无法创建，请确认该实例是否符合 RGD 模式要求。

检查验证错误：

```
# Attempt to create instance and view error
kubectl apply -f instance.yaml

# View existing instance validation status
kubectl describe custom-kind
         my-instance | grep -A 5 "Validation"
```

常见验证问题：

- 必填字段缺失：实例未提供模式要求的全部必填字段
- 类型不匹配：在需要整数类型的位置传入了字符串类型
- 枚举值无效：使用了不在允许取值列表内的数值
- 格式不匹配：字符串未匹配正则表达式格式

查看 RGD 模式：

```
# View the schema definition
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.spec.schema}'
```

确保实例提供所有必填字段且类型正确。

后续步骤

- the section called “注意事项”：kro 注意事项和最佳实践
- the section called “配置权限”：为平台团队与应用团队配置 RBAC
- the section called “kro 概念”：了解 kro 的基本概念及资源生命周期
- the section called “问题排查功能”：一般功能问题排查指导

EKS 托管型 kro 与自主管理型 kro 的对比

EKS 托管型 kro 具备与自主管理型 kro 完全相同的功能，同时还拥有显著的运维优势。有关 EKS 功能与自主管理型解决方案的总体比较，请参阅 the section called “注意事项”。

EKS 托管型 kro 基于上游 kro 控制器构建，且与上游 kro 完全兼容。ResourceGraphDefinition、CEL 表达式以及资源组合编排的工作方式均保持一致。如需查看 kro 的完整文档与示例，请参阅 kro 文档。

迁移路径

您可以从自主管理型 kro 无缝迁移到托管型 kro，实现零停机时间。

> **Important**
>
> 迁移前，请确保自主管理型 kro 控制器与 EKS 托管型 kro 运行的是同一版本。通过 EKS 控制台或 `aws eks describe-capability` 查看托管型 kro 的版本，随后将自主管理型 kro 安装版本升级到与之匹配。此举可避免迁移过程中出现兼容性问题。

1. 更新自主管理型 kro 控制器，将其主节点选举租约配置在 kube-system 命名空间中：

```
helm upgrade --install kro \
  oci://ghcr.io/awslabs/kro/kro-chart \
  --namespace kro \
  --set leaderElection.namespace=kube-system
```

   这会将控制器的租约移至 kube-system，从而允许托管功能与其协调。
2. 在集群中创建 kro 托管功能（详见the section called “创建 kro 功能”文档）
3. 托管功能会自动识别现有的 ResourceGraphDefinition 及其实例，并接管资源调和工作
4. 逐步缩减或移除自主管理型 kro 的部署资源：

```
helm uninstall kro --namespace kro
```

此方法允许两个控制器在迁移期间安全共存。托管功能会自动接管此前由自主管理型 kro 管理的 ResourceGraphDefinition 及其实例，保障资源调和流程持续进行且无冲突。

后续步骤

- the section called “创建 kro 功能”：创建 kro 托管功能资源
- the section called “kro 概念”：了解 kro 的基本概念及资源组合编排

EKS 功能问题排查

本主题提供了 EKS 功能的一般问题排查指南，包括功能运行状况检查、常见问题以及功能特定的问题排查链接。

> **Note**
>
> EKS 功能完全托管，可在您的集群之外运行。您无权访问控制器日志或控制器命名空间。问题排查侧重于功能运行状况、资源状态和配置。

一般问题排查方法

对 EKS 功能进行问题排查时，请遵循以下一般方法：

1. 检查功能运行状况：使用 `aws eks describe-capability` 查看功能状态和运行状况问题
2. 验证资源状态：检查您为状态条件和事件创建的 Kubernetes 资源（CRD）
3. 查看 IAM 权限：确保功能角色具有必要的权限
4. 检查配置：验证功能特定的配置是否正确

检查功能运行状况

所有 EKS 功能都通过 EKS 控制台和 `describe-capability` API 提供运行状况信息。

控制台：

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择集群名称。
3. 选择可观测性选项卡。
4. 选择监控集群。
5. 选择功能选项卡，查看所有功能的运行状况和状态。

“功能”选项卡显示：

- 功能名称和类型
- 当前状态
- 运行状况问题，附说明

Amazon CLI：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name
```

响应包括以下内容：

- 状态：当前功能状态
  （`CREATING`、`ACTIVE`、`UPDATING`、`DELETING`、`CREATE_FAILED`、`UPDATE_FAILED`）
- 运行状况：运行状况信息，包括该功能检测到的任何问题

常见功能状态

CREATING：正在设置功能。

ACTIVE：功能正在运行中并可供使用。如果资源未按预期运行，请检查资源状态和 IAM 权限。

UPDATING：正在应用配置更改。等待状态恢复为 ACTIVE。

CREATE_FAILED 或 UPDATE_FAILED：安装或更新遇到错误。有关详细信息，请查看“运行状况”部分。常见原因：

- IAM 角色信任策略不正确或缺失
- IAM 角色不存在或无法访问
- 集群访问权限问题
- 无效的配置参数

验证 Kubernetes 资源状态

EKS 功能在您的集群中创建并管理 Kubernetes 自定义资源定义（CRD）。进行问题排查时，请检查您创建的资源的状态：

```
# List resources of a specific type
kubectl get resource-kind -A

# Describe a specific resource to see conditions and events
kubectl describe resource-kind
         resource-name -n namespace

# View resource status conditions
kubectl get resource-kind
         resource-name -n namespace -o jsonpath='{.status.conditions}'

# View events related to the resource
kubectl get events --field-selector involvedObject.name=resource-name -n namespace
```

资源状态条件提供有关以下内容的信息：

- 资源是否准备就绪
- 遇到的任何错误

- 当前的协调状态

查看 IAM 权限和集群访问权限

许多功能问题源于 IAM 权限问题或缺少集群访问配置。验证功能角色权限和集群访问条目。

检查 IAM 角色权限

验证功能角色是否具有必要的权限：

```
# List attached managed policies
aws iam list-attached-role-policies --role-name my-capability-role

# List inline policies
aws iam list-role-policies --role-name my-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-capability-role --policy-name policy-name

# View the role's trust policy
aws iam get-role --role-name my-capability-role --query 'Role.AssumeRolePolicyDocument'
```

此信任策略必须允许 capabilities.eks.amazonaws.com 服务主体执行以下操作：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

检查 EKS 访问条目和访问策略

所有功能都需要在其运行的集群上使用适当的 EKS 访问条目和访问策略。

验证访问条目是否存在：

```
aws eks list-access-entries \
  --cluster-name my-cluster \
  --region region-code
```

在列表中查找功能角色 ARN。如果缺失，则该功能将无法访问集群。

查看条目所附加的访问策略：

```
aws eks list-associated-access-policies \
  --cluster-name my-cluster \
  --principal-arn arn:aws:iam::111122223333:role/my-capability-role \
  --region region-code
```

所有功能都需要适当的访问策略：

- ACK：需要权限才能创建和管理 Kubernetes 资源
- kro：需要权限才能创建和管理 Kubernetes 资源
- Argo CD：需要权限才能创建和管理应用程序，并且需要远程目标集群上的访问条目才能进行多集群部署

对于 Argo CD 多集群部署：

如果部署到远程集群，请验证功能角色是否在每个目标集群上都有访问条目：

```
# Check Access Entry on target cluster
aws eks describe-access-entry \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::111122223333:role/argocd-capability-role \
  --region region-code
```

如果目标集群上缺少访问条目，Argo CD 将无法向其部署应用程序。有关配置详细信息，请参阅the section called “注册集群”。

功能特定问题排查

有关每种功能类型特定的详细问题排查指南：

- the section called “问题排查”：对 ACK 资源创建、IAM 权限和跨账户访问进行问题排查
- the section called “问题排查”：对应用程序同步、存储库身份验证和多集群部署进行问题排查
- the section called “问题排查”：对 ResourceGraphDefinitions、CEL 表达式和 RBAC 权限进行问题排查

所有功能的常见问题

功能停留在 CREATING 状态

如果某项功能保持在 CREATING 状态的时间超过预期：

1. 可在控制台（可观测性 > 监控集群 > 功能选项卡）中或使用 Amazon CLI 检查功能运行状况，了解特定问题：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name \
  --query 'capability.health'
```

2. 验证 IAM 角色是否存在并且是否具有正确的信任策略
3. 确保您的集群可访问且运行状况良好
4. 检查是否存在任何可能阻碍功能设置的集群级别问题

没有创建或更新资源

如果功能状态为 ACTIVE，但没有创建或更新资源：

1. 检查资源状态是否存在错误条件
2. 验证特定 Amazon 服务（ACK）或存储库（Argo CD）的 IAM 权限
3. 检查创建底层资源的 RBAC 权限（kro）
4. 查看资源规范中是否存在验证错误

功能运行状况显示问题

如果 describe-capability 显示运行状况问题：

1. 仔细阅读问题描述：它们通常会指出具体问题
2. 解决根本原因（IAM 权限、配置错误等）
3. 问题解决后，该功能将自动恢复

后续步骤

- the section called “使用功能”：管理功能资源
- the section called “问题排查”：ACK 特定问题排查
- the section called “问题排查”：Argo CD 特定问题排查
- the section called “问题排查”：kro 特定问题排查
- the section called “EKS 功能的注意事项”：针对功能的安全最佳实践

组织和监控集群资源

本章包含可以帮助您管理集群的以下主题。您还可以使用 Amazon Web Services 管理控制台 查看有关 Kubernetes 资源的信息。

- Kubernetes 控制面板是适用于 Kubernetes 集群的基于 Web 的通用用户界面。用户能通过它管理集群中运行的应用程序并对其进行故障排除，以及管理集群本身。有关更多信息，请参阅 GitHub 存储库中的 Kubernetes 控制面板。
- the section called “Metrics Server” – Kubernetes Metrics Server 是集群中资源使用数据的聚合器。该聚合器并不会默认部署到集群中，而是由 Kubernetes 附加组件（例如 Kubernetes 控制面板和 the section called “Horizontal Pod Autoscaler”）使用。在本主题中，您将了解如何安装 Metrics Server。
- the section called “使用 Helm 部署应用程序” – Kubernetes 的 Helm 包管理器可帮助您在 Kubernetes 集群上安装和管理应用程序。本主题可帮助您安装并运行 Helm 二进制文件，以便您可以在本地计算机中使用 Helm CLI 安装和管理图表。
- the section called “标记您的资源” – 为了帮助您管理 Amazon EKS 资源，您可以以标签的形式为每个资源分配您自己的元数据。本主题介绍标签并演示如何创建标签。
- the section called “服务配额” – 您的Amazon账户对于每项Amazon服务都有默认配额（以前称为限制）。了解 Amazon EKS 的配额以及如何增加这些配额。

监控和优化 Amazon EKS 集群成本

成本监控是管理 Amazon EKS 上 Kubernetes 集群的一个重要方面。通过了解集群成本，您可以优化资源利用率，设定预算，并对部署做出数据驱动型决策。Amazon EKS 提供两种成本监控解决方案，每种解决方案都有其独特的优势，可帮助您有效跟踪和分配成本：

Amazon EKS 的 Amazon 账单拆分成本分配数据 – 此原生功能与 Amazon 账单控制台无缝集成，使您能够使用与其它 Amazon 服务相同的熟悉界面和工作流来分析和分配成本。通过拆分成本分配，您可以直接了解自己的 Kubernetes 成本以及其他 Amazon 支出，从而更轻松地在整个 Amazon 环境中全面优化成本。您还可以利用 Cost Categories 和成本异常检测等现有 Amazon 账单功能，进一步增强您的成本管理能力。有关更多信息，请参阅《Amazon 账单用户指南》中的了解拆分成本分配数据。

Kubecost ：Amazon EKS 支持 Kubecost，后者是一种 Kubernetes 成本监控工具。Kubecost 提供一种功能丰富的 Kubernetes 原生成本监控方法，提供按 Kubernetes 资源划分的精细成本明细、成本优化建议以及开箱即用的控制面板和报告。Kubecost 还通过与 Amazon 成本和使用情况报告集成来检索

准确的定价数据，从而确保您精确地了解自己的 Amazon EKS 成本。了解如何安装 Kubecost。有关获取免费 Kubecost 订阅的信息，请参阅 Kubecost Amazon Marketplace 页面。

在 Amazon Billing 中使用拆分成本分配功能按容器组（pod）查看成本

使用 Amazon EKS 的 Amazon 拆分成本分配数据进行成本监控

您可以使用 Amazon EKS 的 Amazon 拆分成本分配数据来获得对您的 Amazon EKS 集群的精细的了解。这样可以分析、优化并退还 Kubernetes 应用程序的成本和使用量。您可以根据 Kubernetes 应用程序消耗的 Amazon EC2 CPU 和内存资源，将应用程序成本分配给各个业务部门和团队。Amazon EKS 的拆分成本分配数据便于您了解每个容器组（pod）的成本，并帮助您使用命名空间、集群和其他 Kubernetes 基元汇总每个容器组（pod）的成本数据。以下是可用于分析 Amazon EKS 成本分配数据的 Kubernetes 基元示例。

- 集群名称
- 部署
- 命名空间
- 节点
- 工作负载名称
- 工作负载类型

此外还支持用户定义的成本分配标签。有关使用拆分成本分配数据的更多信息，请参阅《Amazon 账单用户指南》中的了解拆分成本分配数据。

设置成本和使用情况报告

您可以在成本管理控制台、Amazon 命令行界面或 Amazon SDK 中启用 ECS 的拆分成本分配数据。

将以下内容用于拆分成本分配数据：

1. 选择拆分成本分配数据。有关更多信息，请参阅《Amazon 成本和使用情况报告用户指南》中的启用拆分成本分配数据。
2. 将数据纳入新报告或现有报告。
3. 查看报告。您可以使用账单和成本管理控制台或在 Amazon Simple Storage Service 中查看报告文件。

安装 Kubecost

Amazon EKS 支持 Kubecost，您可以使用该功能监控按 Kubernetes 资源 [包括容器组（pod）、节点、命名空间和标签] 细分的成本。本主题将介绍如何安装 Kubecost 和访问 Kubecost 控制面板。

Amazon EKS 提供 Amazon 优化版 Kubecost 捆绑包，以便了解集群成本。您可以使用现有的 Amazon 支持协议获取支持。有关 Kubecost 可用版本的更多信息，请参阅the section called “了解有关 Kubecost 的更多信息”。

> **Note**
>
> Kubecost v2 引入了几项重要的新功能。了解有关 Kubecost v2 的更多信息。

有关 Kubecost 的更多信息，请参阅 Kubecost 文档和常见问题。

安装 Amazon EKS 优化版 Kubecost 捆绑包

您可以使用以下过程之一来安装 Amazon EKS 优化版 Kubecost 捆绑包：

- 在开始之前，建议先查看 Kubecost – 架构概述，以了解 Kubecost 在 Amazon EKS 上的工作原理。
- 如果您不熟悉 Amazon EKS，我们建议您使用 Amazon EKS 附加组件进行安装，因为它可以简化 Amazon EKS 优化版 Kubecost 捆绑包安装。有关更多信息，请参阅 Deploying Kubecost on an Amazon EKS cluster using Amazon EKS add-on。
- 要自定义安装，您可以使用 Helm 配置 Amazon EKS 优化版 Kubecost 捆绑包。有关更多信息，请参阅 Kubecost 文档中的 Deploying Kubecost on an Amazon EKS cluster using Helm。

访问 Kubecost 控制面板

Amazon EKS 优化版 Kubecost 捆绑包设置完成后，您应该可以访问 Kubecost 控制面板。有关更多信息，请参阅 the section called “访问 Kubecost 控制面板”。

访问 Kubecost 控制面板

先决条件

1. 确保与 kubecost 相关的容器组（pod）的状态为“正在运行”。

```
kubectl get pods --namespace kubecost
```

访问 Kubecost 控制面板

1. 在设备上启用端口转发以公开 Kubecost 控制面板。
   - 如果 kubecost 是使用 helm 安装的：

     ```
     kubectl port-forward deployment/kubecost-cost-analyzer 9090 --namespace kubecost
     ```

   - 如果 kubecost 是使用 Amazon EKS 附加组件安装的：

     ```
     kubectl port-forward deployment/cost-analyzer 9090 --namespace kubecost
     ```

     您也可以使用 Amazon 负载均衡器控制器公开 Kubecost，并使用 Amazon Cognito 进行身份验证、授权和用户管理。有关更多信息，请参阅如何使用应用程序负载均衡器和 Amazon Cognito 对您的 Kubernetes Web 应用程序的用户进行身份验证。
2. 在完成上一步的同一台设备上，打开 Web 浏览器并输入以下地址。

   ```
   http://localhost:9090
   ```

   浏览器将显示“Kubecost 概述”页面。Kubecost 可能需要 5-10 分钟（或更长时间）来收集指标，具体取决于您的集群大小。您可以查看 Amazon EKS 支出，包括累计的集群成本、关联的 Kubernetes 资产成本和每月汇总支出。
3. 要跟踪集群级别的成本，请标记您的 Amazon EKS 资源以进行计费。有关更多信息，请参阅 the section called “标记资源以便于计费”。
   - 成本分配 – 查看过去七天内每个命名空间和其他维度的 Amazon EKS 月度成本和累计成本。这有助于了解应用程序的哪些部分产生 Amazon EKS 支出。
   - 资产 – 查看与您的 Amazon EKS 资源关联的 Amazon 基础设施资产的成本。

了解有关 Kubecost 的更多信息

Amazon EKS 提供 Amazon 优化版 Kubecost 捆绑包，以便了解集群成本。Amazon EKS 支持 Kubecost，您可以使用该功能监控按 Kubernetes 资源 [包括容器组（pod）、节点、命名空间和标签] 细分的成本。

本主题将介绍 Kubecost 的可用版本以及可用套餐之间的区别。EKS 支持 Kubecost 版本 1 和版本 2。每个版本都有不同的套餐。您可以为 Amazon EKS 集群使用 Amazon EKS 优化版 Kubecost 捆绑包，无需支付额外费用。您可能需要为使用相关 Amazon 服务（例如 Amazon Managed Service for Prometheus）付费。同时，您还可以使用现有的 Amazon 支持协议获取支持。

作为 Kubernetes 平台管理员和财务主管，您可以使用 Kubecost 可视化 Amazon EKS 费用明细、分配成本以及向应用程序团队等组织部门退款。您可以根据内部团队和业务部门的实际 Amazon 账单为其提供透明、准确的成本数据。此外，您还可以根据他们的基础设施环境及其集群内的使用模式获得定制的成本优化建议。有关 Kubecost 的更多信息，请参阅 Kubecost 文档。

Kubecost 的自定义捆绑包与免费版本的 Kubecost（也称为 OpenCost）之间有何区别？

Amazon 与 Kubecost 合作提供了一个自定义版本的 Kubecost。此版本包括一部分商业功能，无需额外付费。有关 Kubecost 的自定义捆绑包中包含的功能，请参阅下表。

Kubecost v2

Kubecost v1 与 v2 有何区别？

Kubecost 2.0 是对先前版本的重大升级，包含了重大新功能，如全新的 API 后端。请注意，分配和资产 API 完全向后兼容。请查看 Kubecost 文档以确保平稳转换。有关增强功能的完整列表，请参阅 Kubecost v2.0 announcement 和 the full release notes。

> **Important**
>
> 升级前请查看 Kubecost 文档。升级可能会影响报告的可用性。

核心功能对比：

功能	Kubecost 免费套餐 2.0	Amazon EKS 优化版 Kubecost 捆绑包 2.0	Kubecost 企业版 2.0
集群成本可见性	最多 250 个核心的无限集群	与 Amazon Managed Service for Prometheus 集成时的统一多集群，无核心限制	在无限数量的环境（即多云）中统一且不限数量的集群

功能	Kubecost 免费套餐 2.0	Amazon EKS 优化版 Kubecost 捆绑包 2.0	Kubecost 企业版 2.0
部署	用户托管	用户托管	用户托管、Kubecost 托管（专用租户）、SaaS
支持的数据库	本地 Prometheus	Amazon Managed Service for Prometheus 或本地 Prometheus	任何 Prometheus 风格和自定义数据库
数据库保留支持（原始指标）	15 天	无限历史数据	无限历史数据
Kubecost API 和 UI 保留率（ETL）	15 天	15 天	无限制
混合云可见性	-	Amazon EKS 和 Amazon EKS Anywhere 集群	多云和混合云
提醒和定期报告	仅在主集群上支持，限制为 250 个核心	所有集群上的效率提醒、预算提醒、支出变化提醒及更多支持	所有集群上的效率提醒、预算提醒、支出变化提醒及更多支持
已保存的报告	-	使用 15 天指标的报告	使用无限历史数据的报告和指标
云账单集成	仅在主集群上支持，限制为 250 个核心	Amazon 自定义定价支持（包括多个集群和多个账户）	任何云的自定义定价支持
节省建议	仅在主集群上支持，限制为 250 个核心	主集群见解，但没有 250 个核心的限制	多集群洞察
治理：审计	-	-	审计历史成本事件

功能	Kubecost 免费套餐 2.0	Amazon EKS 优化版 Kubecost 捆绑包 2.0	Kubecost 企业版 2.0
单点登录（SSO）支持	-	支持 Amazon Cognito	Okta、Auth 0、PingID、keyCloak 和其他任何自定义服务
使用 SAML 2.0 的基于角色的访问控制（RBAC）	-	-	Okta、Auth 0、PingID、keyCloak 和其他任何自定义服务
企业培训和引导	-	-	全方位服务培训和 FinOps 引导
团队	-	-	是

新功能：

以下功能有指标限制：

- Kubecost 聚合器
- 网络监控
- Kubecost 操作
- 集合
- 异常检测
- 合理调整容器请求大小
- Kubecost 预测
- 用于筛选和聚合的自动完成功能

指标限制：

指标	Kubecost 免费套餐 2.0	Amazon EKS 优化版 Kubecost 捆绑包 2.0	Kubecost 企业版 2.0
集群大小	最多 250 个核心的无限集群	无限制	无限制
指标保留	15 天	15 天	无限制
多集群支持	不可用	可用	Available
核心限制	每个集群 250 个核心	没有核心限制	没有核心限制

Kubecost v1

功能	Kubecost 免费套餐	Amazon EKS 优化版 Kubecost 捆绑包	Kubecost 企业版
部署	用户托管	用户托管	用户托管或 Kubecost 托管（SaaS）
支持的集群数量	无限制	无限制	无限制
支持的数据库	本地 Prometheus	本地 Prometheus 或 Amazon Managed Service for Prometheus	Prometheus、Amazon Managed Service for Prometheus、Cortex 或 Thanos
数据库保留支持	15 天	无限历史数据	无限历史数据
Kubecost API 保留（ETL）	15 天	15 天	无限历史数据
集群成本可见性	单个集群	统一多集群	统一多集群
混合云可见性	-	Amazon EKS 和 Amazon EKS Anywhere 集群	多云和混合云支持

功能	Kubecost 免费套餐	Amazon EKS 优化版 Kubecost 捆绑包	Kubecost 企业版
提醒和定期报告	-	支持效率提醒、预算提醒、支出变化提醒等	支持效率提醒、预算提醒、支出变化提醒等
保存的报告	-	基于 15 天数据的报告	基于无限历史数据的报告
云账单集成	每个单独的集群都需要	Amazon 自定义定价支持（包括多个集群和多个账户）	Amazon 自定义定价支持（包括多个集群和多个账户）
节省建议	单集群洞察	单集群洞察	多集群洞察
治理：审计	-	-	审计历史成本事件
单点登录（SSO）支持	-	支持 Amazon Cognito	Okta、Auth 0、PingID、keyCloak
使用 SAML **2.0** 的基于角色的访问控制（RBAC）	-	-	Okta、Auth 0、PingID、keyCloak
企业培训和引导	-	-	全方位服务培训和 FinOps 引导

常见问题

请参阅以下有关将 Kubecost 与 Amazon EKS 结合使用的常见问题和答案。

什么是 Kubecost API 保留（ETL）功能？

Kubecost ETL 功能可汇总和组织各种指标，以显示不同粒度级别的成本可见性（例如 `namespace-level`、`pod-level` 和 `deployment-level`）。对于 Amazon EKS 优化版 Kubecost 捆绑包，客户可以获得最近 15 天的指标数据和见解。

什么是提醒和定期报告功能？ 它包括哪些提醒和报告？

借助 Kubecost 提醒，团队可以实时获得有关 Kubecost 支出和云支出的动态。借助定期报告，团队能够获得有关历史 Kubecost 和云支出的自定义视图。这两者都可以使用 Kubecost UI 或 Helm 值来配置。它们支持电子邮件、Slack 和 Microsoft Teams。

保存的报告包含哪些内容？

Kubecost 保存的报告是有关成本和效率指标的预定义视图。其中包括按集群、命名空间、标签等划分的成本。

什么是云账单集成？

通过与 Amazon 账单 API 集成，可让 Kubecost 显示集群之外的成本（例如 Amazon S3 成本）。此外，通过这种集成，还可让 Kubecost 根据实际账单核对 Kubecost 的集群内预测，从而可以考虑竞价型实例使用情况、节省计划和企业折扣。

节省建议包含哪些内容？

Kubecost 提供了相关的洞察和自动功能，可帮助用户优化其 Kubernetes 基础设施和支出。

此功能是否收费？

不收费。无需支付额外费用，您就能使用 Amazon EKS 优化版 Kubecost 捆绑包。如果您需要未包括在此捆绑包中的其他 Kubecost 功能，则可以通过 Amazon Marketplace 购买 Kubecost 企业许可证，也可以直接从 Kubecost 购买。

是否支持 Amazon EKS 优化版 Kubecost 捆绑包？

支持，但前提是您使用的是 Amazon EKS 优化版 Kubecost 捆绑包。

如何支持 Amazon EKS 优化版 Kubecost 捆绑包？

您可以通过联系 Amazon 借助 Amazon 支持团队打开支持案例。

我是否需要许可证才能使用 Amazon EKS 集成提供的 Kubecost 功能？

否。

我是否可以将 Kubecost 与 Amazon 成本和使用情况报告集成，以获得更准确的报告？

是。您可以将 Kubecost 配置为从 Amazon 成本和使用情况报告中摄取数据，以获取准确的成本可见性，包括折扣、竞价型定价、预留实例定价，以及其他内容。有关更多信息，请参阅 Kubecost 文档中的 Amazon 云账单集成。

此版本是否支持 Amazon EC2 上自行管理的 Kubernetes 集群的成本管理？

不支持。Amazon EKS 优化版 Kubecost 捆绑包仅与 Amazon EKS 集群兼容。

Kubecost 是否能够追踪 Amazon Fargate 上的 Amazon EKS 的成本？

Kubecost 已尽最大努力显示 Fargate 上的 Amazon EKS 的集群成本可见性，但准确性低于 Amazon EC2 上的 Amazon EKS。这主要是由于您的使用量的计费方式不同。使用 Fargate 上的 Amazon EKS，将针对消耗的资源为您计费。使用 Amazon EC2 节点上的 Amazon EKS，将根据预配置的资源为您计费。Kubecost 将根据节点规格（包括 CPU、RAM 和临时存储）来计算 Amazon EC2 节点的成本。使用 Fargate，将根据为 Fargate 容器组（pod）请求的资源计算成本。

我如何获得 Kubecost 的更新和新版本？

您可以使用标准的 Helm 升级程序升级您的 Kubecost 版本。最新版本位于 Amazon ECR Public Gallery 中。

是否支持 **`kubectl-cost`** CLI？ 我如何安装它？

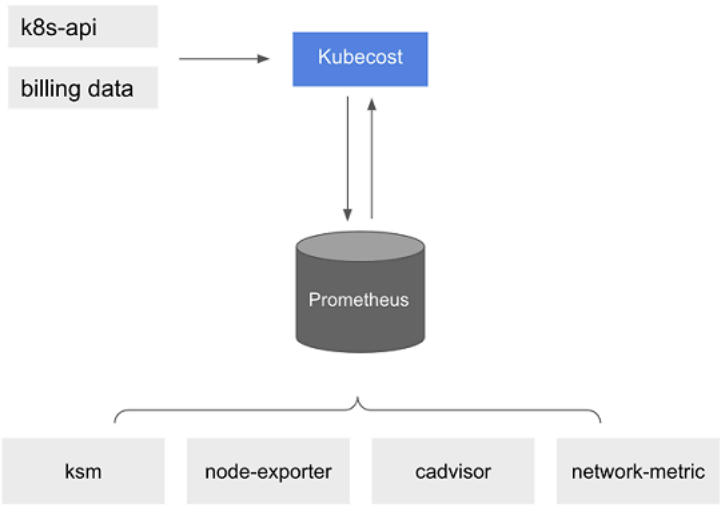
支持。`Kubectl-cost` 是一个通过 Kubecost（Apache 2.0 许可证）使用的开源工具，提供针对 Kubernetes 成本分配指标的 CLI 访问权限。要安装 `kubectl-cost`，请参阅 GitHub 上的安装。

是否支持 Kubernetes 用户界面？ 我如何访问它？

Kubecost 提供了一个 Web 控制面板，您可以通过 `kubectl` 端口转发、入口或负载均衡器访问该控制面板。您也可以使用 Amazon 负载均衡器控制器公开 Kubecost，并使用 Amazon Cognito 进行身份验证、授权和用户管理。有关更多信息，请参阅 Amazon 博客上的如何使用应用程序负载均衡器和 Amazon Cognito 对您的 Kubernetes Web 应用程序的用户进行身份验证。

其他 Kubecost 功能

- Kubecost v1 和 v2 均提供以下功能。
  - 导出成本指标 – Amazon EKS 优化版成本监控通过 Kubecost 和 Prometheus 进行部署，Prometheus 是一个开源监控系统和时间序列数据库。Kubecost 从 Prometheus 读取指标，然后执行成本分配计算并将指标写回 Prometheus。Kubecost 前端从 Prometheus 读取指标并在 Kubecost 用户界面上显示这些指标。此架构如下图所示。



预装 Prometheus 后，可以编写查询，以将 Kubecost 数据提取到当前的商业智能系统，进行进一步分析。您也可以将其用作当前 Grafana 控制面板的数据来源，以显示您的内部团队熟悉的 Amazon EKS 集群成本。要了解有关如何编写 Prometheus 查询的更多信息，请参阅 GitHub 上的 https://opencost.io/docs/installation/prometheus/ `readme` 文件，或者参考 Kubecost Github 存储库中的示例 Grafana JSON 模型。

- Amazon 成本和使用情况报告集成 – 为了对 Amazon EKS 集群执行成本分配计算，Kubecost 会从 Amazon 价目表 API 中检索 Amazon 服务和 Amazon 资源的公开定价信息。您还可以将 Kubecost 和 Amazon 成本和使用情况报告集成，以提高特定于 Amazon 账户的定价信息的准确性。这些信息包括企业折扣计划、预留实例使用情况、Savings Plans 以及 Spot 使用情况。要了解有关 Amazon 成本和使用情况报告集成工作原理的更多信息，请参阅 Kubecost 文档中的 Amazon 云账单集成。

使用 Kubernetes Metrics Server 查看资源使用情况

Kubernetes Metrics Server 是集群中资源使用数据的聚合器，默认不会部署在 Amazon EKS 集群中。有关更多信息，请参阅 GitHub 上的 Kubernetes Metrics Server。Metrics Server 通常由其他 Kubernetes 附加组件使用，例如使用 Horizontal Pod Autoscaler 扩展容器组（pod）部署或 Kubernetes 控制面板。有关详细信息，请参阅 Kubernetes 文档中的资源指标管道。本主题介绍了如何在您的 Amazon EKS 集群上部署 Kubernetes Metrics Server。

> **Important**
>
> 这些指标是用于时间点分析，不是历史分析的准确来源。它们不能用作监控解决方案或用于其它非自动扩缩目的。有关监控工具的信息，请参阅 监控集群。

注意事项

- 如果使用清单将 Kubernetes Metrics Server 手动部署到 Fargate 节点，请将 `metrics-server` 部署配置为使用除默认端口 `10250` 以外的端口。此端口为 Fargate 保留。Metrics Server 的 Amazon EKS 附加组件版本已预配置为使用端口 `10251`。
- 确保安全组和网络 ACL 允许 `metrics-server` 容器组（pod）与所有其他节点和容器组（pod）之间的端口 `10250`。Kubernetes Metrics Server 仍使用端口 `10250` 从集群中的其他端点收集指标。如果在 Fargate 节点上部署，请同时允许已配置的 Metrics Server 备用端口和端口 `10250`。

将 Amazon EKS 附加组件作为社区附加组件部署

新增：您现在可以使用 Amazon 控制台或 Amazon EKS API 将 Metrics Server 部署为社区附加组件。

使用 Amazon 控制台部署

1. 在 Amazon 控制台中打开 EKS 集群
2. 在“附加组件”选项卡中，选择获取更多附加组件。
3. 从“社区附加组件”部分中选择 Metrics Server，然后选择下一步
4. EKS 可确定适合集群的附加组件。您可以使用版本下拉菜单更改版本。
5. 选择下一步，然后选择创建以安装此附加组件。

其他资源

了解有关 the section called “社区附加组件” 的更多信息。

您可以像安装或更新其他 Amazon EKS 附加组件一样安装或更新社区附加组件。

- the section called “创建附加组件”
- the section called “更新附加组件”

- the section called “移除附加组件”

使用清单部署

新增：您现在可以使用 Amazon 控制台或 Amazon EKS API 将 Metrics Server 部署为社区附加组件。这些清单安装说明将被存档。

1. 使用以下命令部署 Metrics Server：

   ```
   kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
   ```

   如果您使用的是 Fargate，则需要更改此文件。在默认配置中，指标服务器使用端口 10250。此端口在 Fargate 上保留。将 components.yaml 中对端口 10250 的引用替换为其他端口，例如 10251。

2. 使用以下命令验证 `metrics-server` 部署是否运行所需数量的容器组（pod）。

   ```
   kubectl get deployment metrics-server -n kube-system
   ```

   示例输出如下。

   ```
   NAME             READY   UP-TO-DATE   AVAILABLE   AGE
   metrics-server   1/1     1            1           6m
   ```

3. 通过显示节点的资源（CPU/内存）使用情况来测试 Metrics Server 是否正常工作。

   ```
   kubectl top nodes
   ```

4. 如果您收到错误消息 `Error from server (Forbidden)`，则需要更新 Kubernetes RBAC 配置。您的 Kubernetes RBAC 身份需要具有足够的权限才能读取集群指标。在 GitHub 上查看读取指标所需的最低 Kubernetes API 权限。了解如何向 Amazon IAM 身份（例如角色）授予对 Kubernetes API 的访问权限。

使用 Helm 在 Amazon EKS 上部署应用程序

Kubernetes 的 Helm 包管理器可帮助您在 Kubernetes 集群上安装和管理应用程序。有关更多信息，请参阅 Helm 文档。本主题可帮助您安装并运行 Helm 二进制文件，以便您可以在本地系统中使用 Helm CLI 安装和管理图表。

> **Important**
>
> 您必须先将 `kubectl` 配置为用于 Amazon EKS，然后才能在 Amazon EKS 集群上安装 Helm Chart。如果您尚未执行此操作，请参阅the section called “使用 kubectl 访问集群”后再继续。如果集群的以下命令成功，说明您已正确配置。
>
> ```
> kubectl get svc
> ```

1. 运行适用于您的客户端操作系统的命令。
   - 如果您将 macOS 与 Homebrew 结合使用，请使用以下命令安装二进制文件。

     ```
     brew install helm
     ```

   - 有关更多安装选项，请参阅 Helm 文档中的安装 Helm。

     > **Note**
     >
     > 如果您收到一条消息，显示必须首先安装 `openssl`，则可以使用以下命令进行安装。

     ```
     sudo yum install openssl
     ```

1. 要在 PATH 中选择新的二进制文件，请关闭当前的终端窗口，然后打开一个新窗口。
2. 查看您已安装的 Helm 版本。

   ```
   helm version --template='{{ .Version }}{{ "\n" }}'
   ```

   示例输出如下。

   ```
   v3.17.2
   ```

3. 确保安装的版本与您的集群版本兼容。查看支持的版本偏差以了解更多信息。例如，如果您使用 `3.17.x` 运行，则支持的 Kubernetes 版本不应超出 `1.29.x ~ 1.32.x` 的范围。
4. 此时，您可以运行任何 Helm 命令（例如 `helm install` *`chart-name`*），以便安装、修改、删除或查询您的集群中的 Helm Chart。如果您刚接触 Helm，并且没有要安装的特定图表，则可以：

- 通过安装示例图表来进行试验。请参阅 Helm 快速入门指南中的安装示例图表。
- 创建示例图表并将其推送至 Amazon ECR。有关更多信息，请参阅 Amazon Elastic Container Registry 用户指南中的推送 Helm Chart。
- 从 eks-charts GitHub 存储库或从 ArtifactHub 中安装 Amazon EKS 图表。

使用标签整理 Amazon EKS 资源

您可以使用标签帮助您管理 Amazon EKS 资源。本主题提供了标签功能的概述，并说明如何创建标签。

主题

- 有关标签的基本知识
- 标记您的资源
- 标签限制
- 标记资源以便于计费
- 通过控制台使用标签
- 通过 CLI、API 或 eksctl 使用标签

> **Note**
>
> 标签是一种与 Kubernetes 标签和注释分开的元数据。有关这些其他元数据类型的更多信息，请参阅 Kubernetes 文档中的以下各节：
>
> - 标签和选择器
> - Annotations

有关标签的基本知识

标签是为Amazon资源分配的标记。每个标签都包含一个键和一个可选值。

借助标签，您可以对 Amazon 资源进行分类。例如，您可以按用途、所有者或环境对资源进行分类。在您具有相同类型的许多资源时，可以使用分配给特定资源的标签来快速识别该资源。例如，您可以为 Amazon EKS 集群定义一组标签，以帮助您跟踪每个集群的拥有者和堆栈级别。我们建议为每个资源类型设计一组一致的标签键。然后，您可以根据添加的标签搜索和筛选资源。

添加标签后，可以编辑标签键和值，还可以随时删除资源的标签。如果删除资源，资源的所有标签也会被删除。

标签对 Amazon EKS 没有任何语义意义，应严格按字符串进行解析。您可以将标签值设置为空字符串。但是，您不能将标签值设置为 null。如果您添加的标签的键与该资源上现有标签的键相同，则新值会覆盖旧值。

如果您使用的是 Amazon Identity and Access Management（IAM），则可以控制您的Amazon账户中的哪些用户拥有管理标签的权限。

标记您的资源

以下 Amazon EKS 资源支持标签：

- 集群
- 托管节点组
- Fargate 配置文件

您可以使用以下内容标记这些资源：

- 如果您使用的是 Amazon EKS 控制台，可以随时对新的或现有的资源应用标签。您可以使用相关资源页面上的标签选项卡执行此操作。有关更多信息，请参阅 the section called “通过控制台使用标签”。
- 如果您使用的是 `eksctl`，可以在使用 `--tags` 选项创建资源时为资源应用标签。
- 如果您使用的是 Amazon CLI、Amazon EKS API 或 Amazon SDK，则可以使用相关 API 操作上的 `tags` 参数对新资源应用标签。您也可以通过使用 `TagResource` API 操作将标签应用于现有资源。有关更多信息，请参阅 TagResource。

在您使用一些资源创建操作时，您还可以在创建资源的同时为资源指定标签。如果在创建资源期间无法应用标签，则创建资源会失败。此机制可确保对于您希望标记的资源，要么使用您指定的标签创建，要么完全不创建。如果您在创建资源时标记这些资源，则无需在创建资源后运行自定义标记脚本。

标签不会传播到与您创建的资源关联的其它资源。例如，Fargate 配置文件标签不会传播到与 Fargate 配置文件关联的其他资源，比如使用配置文件调度的容器组（pod）。

标签限制

以下限制适用于标签：

- 一个资源最多可以关联 50 个标签。
- 不能对一个资源重复使用标签键。每个标签键必须具有唯一性，而且只能有一个值。
- 键最长可达 128 个字符（采用 UTF-8 格式）。
- 值最长可达 256 个字符（采用 UTF-8 格式）。
- 如果有多个 Amazon 服务和资源使用您的标记方案，请限制您使用的字符类型。某些服务可能对允许使用的字符有限制。通常允许使用的字符包括字母、数字、空格以及以下字符：`+ - = . _ : / @`。
- 标签键和值区分大小写。
- 请不要使用 `aws:`、 `Amazon:` 或任何大写或小写组合（例如，键或值的前缀）。这些字符串保留供Amazon使用。无法编辑或删除带此前缀的标签键或值。具有此前缀的标签不计入每个资源的标签数限制。

标记资源以便于计费

将标签应用于 Amazon EKS 集群时，您可以使用它们在成本和使用情况报告中进行成本分配。成本和使用情况报告中的计量数据显示了所有 Amazon ECS 集群的使用情况。有关更多信息，请参阅 Amazon Billing 用户指南中的Amazon 成本和使用情况报告。

通过 Amazon 生成的成本分配标签（特别是 `aws:eks:cluster-name`），您可以在 Cost Explorer 中按单个 Amazon EKS 集群细分 Amazon EC2 实例成本。但此标签不会捕获控制面板开支。该标签会自动添加到参与 Amazon EKS 集群的 Amazon EC2 实例中。无论实例是使用 Amazon EKS 托管节点组或 Karpenter 预置，还是直接通过 Amazon EC2 预置，都会发生此行为。此特定标签不会计入 50 个标签的限制。要使用该标签，账户所有者必须在 Amazon Billing 控制台中或者通过使用 API 来激活它。当 Amazon Organizations 管理账户所有者激活该标签时，还将同时为所有组织成员账户激活该标签。

您还可以根据具有相同标签键值的资源组织您的账单信息。例如，您可以将特定的应用程序名称用作几个资源的标签，然后组织您的账单信息。这样，您可以查看多个服务中使用该应用程序的总成本。有关设置带有标签的成本分配报告的更多信息，请参阅 Amazon Billing 用户指南中的月度成本分配报告。

> **Note**
>
> 如果您刚刚启用报告，则可以在 24 小时后查看当月的数据。

Cost Explorer 是一个报告工具，作为 Amazon 免费套餐的一部分提供。您可以使用 Cost Explorer 查看过去 13 个月的 Amazon EKS 资源图表。您还可以预测您在接下来三个月内可能产生的费用。您可

以查看您在 Amazon 资源上的花费随时间变化的模式。如，您可以使用它来确定需要进一步查询的方面，并查看可用于了解成本的趋势。您还可以指定数据的时间范围，并按天或按月查看时间数据。

通过控制台使用标签

通过使用 Amazon EKS 控制台，您可以管理与新的或现有的集群和托管节点组关联的标签。

当您在 Amazon EKS 控制台中选择特定资源页面时，该页面会显示这些资源的列表。例如，如果您从左侧导航窗格中选择 Clusters（集群），则控制台会显示 Amazon EKS 集群列表。当您从其中一个列表中选择一种支持标签的资源（例如，特定集群）时，您可以在标签选项卡上查看和管理其标签。

您还可以在 Amazon Web Services 管理控制台 中使用标签编辑器，它为管理标签提供了统一的方法。有关更多信息，请参阅 Amazon 标签编辑器用户指南中的使用标签编辑器为 Amazon 资源添加标签。

在创建时为资源添加标签

您可以在创建 Amazon EKS 集群、托管节点组和 Fargate 配置文件时，为它们添加标签。有关更多信息，请参阅 the section called “创建集群”。

为资源添加和删除标签

您可以直接从资源的页面中添加或删除与集群关联的标签。

1. 打开 Amazon EKS 控制台。
2. 在导航栏中，选择要使用的 Amazon 区域。
3. 在左侧导航窗格中，选择集群。
4. 选择特定集群。
5. 选择标签选项卡，然后选择管理标签。
6. 在 Manage tags（管理标签）页面上，根据需要添加或删除标签。
   - 要添加标签，请选择 Add tag（添加标签）。然后，指定每个标签的键和值。
   - 要删除标签，请选择 Remove tag（删除标签）。
7. 对要添加或删除的每个标签重复此过程。
8. 选择 Update (更新) 完成操作。

通过 CLI、API 或 `eksctl` 使用标签

使用以下 Amazon CLI 命令或 Amazon EKS API 操作来添加、更新、列出和删除资源的标签。您只能使用 `eksctl` 在使用一个命令同时创建新资源的同时添加标签。

Task	Amazon CLI	Amazon Tools for Windows PowerShell	API 操作
添加或覆盖一个或多个标签。	tag-resource	Add-EKSResourceTag	TagResource
删除一个或多个标签。	untag-resource	Remove-EKSResourceTag	UntagResource

以下示例说明如何使用 Amazon CLI 给资源加标签或取消标签。

示例 1：标记现有集群

以下命令标记现有集群。

```
aws eks tag-resource --resource-arn resource_ARN --tags team=devs
```

示例 2：取消标记现有集群

以下命令从现有集群删除标签。

```
aws eks untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

示例 3：列出资源的标签

以下命令列出与现有资源关联的标签。

```
aws eks list-tags-for-resource --resource-arn resource_ARN
```

在您使用一些资源创建操作时，您可以在创建资源的同时指定标签。以下操作支持在创建资源时指定标签。

Task	Amazon CLI	Amazon Tools for Windows PowerShell	API 操作	eksctl
创建集群	create-cluster	New-EKSCluster	CreateCluster	`create cluster`

Task	Amazon CLI	Amazon Tools for Windows PowerShell	API 操作	eksctl
创建托管节点组*	create-nodegroup	New-EKSNodegroup	CreateNodegroup	`create nodegroup`
创建 Fargate 配置文件	create-fargate-profile	New-EKSFargateProfile	CreateFargateProfile.html	`create fargatepr ofile`

- 如果您还想在创建托管节点组时为 Amazon EC2 实例添加标签，请使用启动模板创建托管节点组。有关更多信息，请参阅 the section called “为 Amazon EC2 实例添加标签”。如果您的实例已经存在，您可以手动为实例添加标签。有关更多信息，请参阅《Amazon EC2 用户指南》中的标记您的资源。

查看和管理 Amazon EKS 和 Fargate 服务配额

Amazon EKS 已与服务配额集成，后者是一项 Amazon 服务，您可以使用该服务从中心位置查看和管理您的配额。有关更多信息，请参阅《服务配额用户指南》中的 What Is Service Quotas?。借助服务配额集成，您可以使用 Amazon Web Services 管理控制台 和 Amazon 快速查找 Amazon EKS 和 Amazon Fargate 服务配额的值。

在 Amazon Web Services 管理控制台中查看 EKS 服务配额

1. 打开服务配额控制台。
2. 在左侧导航窗格中，选择 Amazon 服务。
3. 从 Amazon 服务列表中，搜索并选择 Amazon Elastic Kubernetes Service（Amazon EKS）或 Amazon Fargate。

   在服务配额列表中，您可以查看服务配额名称、应用的值（如果该值可用）、Amazon 默认配额以及配额值是否可调整。
4. 要查看有关服务配额的其他信息（如描述），请选择配额名称。
5. （可选）要请求增加配额，请选择要增加的配额，选择 Request quota increase（请求增加配额），输入或选择所需信息，然后选择 Request（请求）。

要使用 Amazon Web Services 管理控制台 进一步处理服务配额，请参阅服务配额用户指南。要请求提高配额，请参阅《Service Quotas 用户指南》中的请求提高配额。

使用 Amazon CLI 查看 EKS 服务配额

运行以下命令查看您的 Amazon EKS 配额。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code eks \
    --output table
```

运行以下命令查看您的 Fargate 配额。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code fargate \
    --output table
```

> **Note**
>
> 返回的配额是当前 Amazon 区域中此账户下可在 Fargate 上并发运行的 Amazon ECS 任务或 Amazon EKS 容器组（pod）的数量。

要使用 Amazon CLI 进一步处理服务配额，请参阅《Amazon CLI 命令参考》中的服务配额。要请求提高配额，请参阅《Amazon CLI 命令参考》中的 request-service-quota-increase 命令。

Amazon EKS 服务配额

Amazon 建议使用 Amazon Web Services 管理控制台查看当前配额。有关更多信息，请参阅 the section called “在 Amazon Web Services 管理控制台中查看 EKS 服务配额”。

要查看默认 EKS 服务配额，请参阅《Amazon General Reference》中的 Amazon Elastic Kubernetes Service endpoints and quotas。

这些服务配额列于服务配额控制台中 Amazon Elastic Kubernetes Service（Amazon EKS）下。对于显示为可调整的值，要请求提高配额，请参阅《服务配额用户指南》中的 Requesting a quota increase。

> **Note**
>
> 服务配额不支持调整以下组件：* 每个集群的容器组身份关联。有关限制，请参阅the section called “容器组身份”。* 用于远程节点网络的 CIDR 或用于混合节点的远程容器组网络。有关限制，请参阅the section called “混合节点”。

Amazon Fargate 服务配额

服务配额控制台中的 Amazon Fargate 服务列出了多个服务配额。您可以配置警报，以在用量接近服务配额时向您发出警报。有关更多信息，请参阅 the section called “创建 CloudWatch 警报以监控 Fargate 资源使用情况指标”。

新 Amazon 账户的初始配额可能较低，但会随着时间的推移而增加。Fargate 会持续监控每个 Amazon 区域内的账户使用情况，然后根据使用情况自动增加配额。对于显示为可调整的值，您还可以请求提高限额。有关更多信息，请参阅《服务配额用户指南》中的 Requesting a quota increase。

Amazon 建议使用 Amazon Web Services 管理控制台查看当前配额。有关更多信息，请参阅 the section called “在 Amazon Web Services 管理控制台中查看 EKS 服务配额”。

要查看 EKS 服务配额相关默认 Amazon Fargate，请参阅《Amazon General Reference》中的 Fargate service quotas。

> **Note**
>
> 此外，Fargate 还强制执行 Amazon ECS 任务和 Amazon EKS 容器组（pod）启动率限额。有关更多信息，请参阅《Amazon ECS 指南》中的 Amazon Fargate 节流配额。

Amazon EKS 中的安全性

Amazon 十分重视云安全性。作为 Amazon 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。

安全性是 Amazon 和您的共同责任。责任共担模型将其描述为云的安全性和云中的安全性：

- 云的安全性 – Amazon 负责保护在 Amazon 云中运行 Amazon 服务的基础设施。对于 Amazon EKS，Amazon 负责 Kubernetes 控制层面，其中包括控制层面节点和 etcd 数据库。作为 Amazon 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于 Amazon EKS 的合规性计划，请参阅合规性计划范围内的Amazon服务。
- 云中的安全性 – 您的责任包括以下各个方面。
  - 数据层面的安全配置，包括配置安全组以允许流量从 Amazon EKS 控制层面传入客户 VPC
  - 节点和容器本身的配置
  - 节点操作系统（包括更新和安全补丁）
  - 其他关联的应用程序软件：
    - 设置和管理网络控制功能，例如防火墙规则
    - 使用 IAM 或其他服务管理平台级身份和访问管理
  - 您的数据的敏感性、您公司的要求以及适用的法律法规

Amazon EKS 已通过有关受监管和敏感应用程序的多个合规计划认证。Amazon EKS 符合 SOC、PCI、ISO、FedRAMP-Moderate、IRAP、C5、K-ISMS、ENS High、OSPAR、HITRUST CSF 标准，并且是一项符合 HIPAA 要求的服务。有关更多信息，请参阅 管理访问权限。

此文档将帮助您了解如何在使用 Amazon EKS 时应用责任共担模式。以下主题说明如何配置 Amazon EKS 以实现您的安全性和合规性目标。您还会了解如何使用其他Amazon服务以帮助您监控和保护 Amazon EKS 资源。

> **Note**
>
> Linux 容器由控制组 (cgroup) 和命名空间组成，这些控制组和命名空间有助于限制容器可以访问的内容，但所有容器都与主机 Amazon EC2 实例共享相同的 Linux 内核。非常不建议以根用户 (UID 0) 身份运行容器或授予容器对主机资源或命名空间（如主机网络或主机 PID 命名空间）的访问权限，因为这样做会降低容器提供的隔离的有效性。

主题

- 使用最佳实践保护 Amazon EKS 集群
- 分析 Amazon EKS 中的漏洞
- Amazon EKS 集群的合规性验证
- Amazon Elastic Kubernetes Service 的安全注意事项
- Kubernetes 的安全注意事项
- Amazon EKS 自动模式的安全注意事项
- EKS 功能的安全注意事项
- 适用于 Amazon EKS 的身份和访问管理

使用最佳实践保护 Amazon EKS 集群

Amazon EKS 安全最佳实践在《Amazon EKS 最佳实践指南》中的安全最佳实践中。

分析 Amazon EKS 中的漏洞

安全性是配置和维护 Kubernetes 集群和应用程序的重要注意事项。下方列出了供您分析 EKS 集群安全配置的资源、用于检查漏洞的资源，以及可为您执行该分析的 Amazon 服务的集成。

适用于 Amazon EKS 的互联网安全中心（CIS）基准

Center for Internet Security（CIS）Kubernetes 基准提供了关于 Amazon EKS 安全配置的指导。该基准：

- 适用于 Amazon EC2 节点（托管和自我管理），您在其中负责 Kubernetes 组件的安全配置。
- 提供了一种社区批准的标准方法，以确保您在使用 Amazon EKS 时已安全地配置了 Kubernetes 集群和节点。
- 由四个部分组成：控制层面日志记录配置、节点安全配置、策略和托管服务。
- 支持 Amazon EKS 中当前提供的所有 Kubernetes 版本，并且可以使用 kube-bench（一个用于在 Kubernetes 集群上使用 CIS 基准检查配置的标准开源工具）运行。

要了解更多信息，请参阅 CIS Amazon EKS 基准简介。

有关使用 CIS 基准 AMI 更新节点组的自动 `aws-sample` 管道，请参阅 EKS-Optimized AMI Hardening Pipeline。

Amazon EKS 平台版本

Amazon EKS 平台版本表示集群控制面板的功能，包括启用哪些 Kubernetes API 服务器标志和当前的 Kubernetes 补丁版本。使用最新平台版本部署新集群。有关详细信息，请参阅 EKS platform-versions。

您可以将 Amazon EKS 集群更新到更新的 Kubernetes 版本。随着新 Kubernetes 版本在 Amazon EKS 中提供，我们建议您主动将集群更新为使用最新的可用版本。要了解有关 EKS 中 Kubernetes 版本的更多信息，请参阅 Amazon EKS 支持的版本。

操作系统漏洞列表

AL2023 漏洞列表

在 Amazon Linux 安全中心跟踪 Amazon Linux 2023 的安全和隐私事件，或订阅关联的 RSS 源。安全和隐私事件包括受影响问题的概述、程序包以及更新实例以解决问题的说明。

Amazon Linux 2 漏洞列表

在 Amazon Linux 安全中心跟踪 Amazon Linux 2 的安全和隐私事件，或订阅关联的 RSS 源。安全和隐私事件包括受影响问题的概述、程序包以及更新实例以解决问题的说明。

使用 Amazon Inspector 进行节点检测

您可以使用 Amazon Inspector 来检查节点的意外网络访问和这些 Amazon EC2 实例上的漏洞。

使用 Amazon GuardDuty 进行集群和节点检测

Amazon GuardDuty 是一项威胁检测服务，有助于保护您的账户、容器、工作负载和 Amazon 环境中的数据。除其他功能外，GuardDuty 还提供以下两项功能，用于检测 EKS 集群面临的潜在威胁：EKS 保护和运行时监控。

有关更多信息，请参阅 the section called “Amazon GuardDuty”。

Amazon EKS 集群的合规性验证

要了解某个 Amazon 服务是否在特定合规性计划范围内，请参阅合规性计划范围内的 Amazon 服务，然后选择您感兴趣的合规性计划。有关一般信息，请参阅 Amazon合规性计划。

您可以使用 Amazon Artifact 下载第三方审计报告。有关更多信息，请参阅下载 Amazon Artifact 中的报告。

您在使用 Amazon 服务时的合规性责任由您数据的敏感性、贵公司的合规性目标以及适用的法律法规决定。Amazon 提供以下资源来帮助满足合规性：

- Security Compliance & Governance：这些解决方案实施指南讨论了架构考虑因素，并提供了部署安全性和合规性功能的步骤。
- 符合 HIPAA 要求的服务参考：列出符合 HIPAA 要求的服务。并非所有 Amazon 服务都符合 HIPAA 条件。
- Amazon 合规性资源 – 此业务手册和指南集合可能适用于您的行业和位置。
- Amazon 客户合规指南：从合规角度了解责任共担模式。这些指南总结了保护 Amazon 服务的最佳实践，并将指南映射到跨多个框架的安全控制，包括美国国家标准与技术研究院（NIST）、支付卡行业安全标准委员会（PCI）和国际标准化组织（ISO）。
- 《Amazon Config 开发人员指南》中的使用规则评估资源 – Amazon Config 服务评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- Amazon Security Hub – 该 Amazon 服务向您提供 Amazon 中安全状态的全面视图。Security Hub 通过安全控制措施评估您的 Amazon 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控制措施的列表，请参阅 Security Hub 控制措施参考。
- Amazon GuardDuty – 该 Amazon 服务通过监控您的环境中是否存在可疑和恶意活动，来检测您的 Amazon 账户、工作负载、容器和数据面临的潜在威胁。GuardDuty 可以通过满足某些合规性框架规定的入侵检测要求，来协助您满足各种合规性要求，如 PCI DSS。
- Amazon Audit Manager – 此 Amazon服务帮助您持续审计您的Amazon使用情况，以简化管理风险以及与相关法规与行业标准的合规性的方式。

Amazon Elastic Kubernetes Service 的安全注意事项

以下是关于云安全的注意事项，这些注意事项会对 Amazon EKS 造成影响。

主题

- Amazon EKS 中的基础设施安全性
- 了解 Amazon EKS 集群中的弹性
- 防止在 Amazon EKS 中出现跨服务混淆代理

Amazon EKS 中的基础设施安全性

作为一项托管式服务，Amazon Elastic Kubernetes Service 受 Amazon 全球网络安全保护。有关 Amazon 安全服务以及 Amazon 如何保护基础设施的信息，请参阅 Amazon 云安全性。要按照

基础设施安全最佳实践设计您的 Amazon 环境，请参阅《安全性支柱 Amazon Well-Architected Framework》中的基础设施保护。

您可以使用Amazon发布的 API 调用通过网络访问 Amazon EKS。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密（PFS）的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 Amazon Security Token Service（Amazon STS）生成临时安全凭证，对请求进行签名。

在创建 Amazon EKS 集群时，为要使用的集群指定 VPC 子网。Amazon EKS 需要至少位于两个可用区中的子网。我们建议使用同时带有公有子网和私有子网的 VPC，以便 Kubernetes 可以在公有子网中创建公有负载均衡器，进而将流量负载均衡到在私有子网中的节点上运行的负载均衡器。

有关 VPC 注意事项的更多信息，请参阅the section called “VPC 和子网要求”。

如果使用开始使用 Amazon EKS 演练中提供的 Amazon CloudFormation 模板创建 VPC 和节点组，则将使用推荐设置来配置控制面板和节点安全组。

有关安全组注意事项的更多信息，请参阅the section called “安全组要求”。

在创建新集群时，Amazon EKS 将为您用于与集群进行通信的托管 Kubernetes API 服务器（使用 Kubernetes 管理工具，如 `kubectl`）创建端点。默认情况下，此 API 服务器端点对于互联网是公有的，对 API 服务器的访问将使用 Amazon Identity and Access Management（IAM）与本机 Kubernetes 基于角色的访问控制（RBAC）相结合的方式加以保护。

您可以启用对 Kubernetes API 服务器的私有访问，以便节点与 API 服务器之间的所有通信都在 VPC 内。您可以限制从 Internet 访问 API 服务器的 IP 地址，或完全禁用对 API 服务器的 Internet 访问。

有关修改集群终端节点访问的更多信息，请参阅the section called “修改集群终端节点访问”。

您可以使用 Amazon VPC CNI 或第三方工具（如 Project Calico）实施 Kubernetes 网络策略。有关将 Amazon VPC CNI 用于网络策略的更多信息，请参阅 the section called “Kubernetes 策略”。Project Calico 是一个第三方开源项目。有关更多信息，请参阅 Project Calico 文档。

使用 Amazon PrivateLink 访问 Amazon EKS

您可以使用 Amazon PrivateLink 在您的 VPC 和 Amazon Elastic Kubernetes Service 之间创建私有连接。您可以像在 VPC 中一样访问 Amazon EKS，而无需使用互联网网关、NAT 设备、VPN 连接或 Amazon Direct Connect 连接。VPC 中的实例不需要公有 IP 地址即可访问 Amazon EKS。

您可以通过创建由 Amazon PrivateLink 提供支持的接口端点来建立此私有连接。我们将在您为接口端点启用的每个子网中创建一个端点网络接口。这些是请求者托管式网络接口，用作发往 Amazon EKS 的流量的入口点。

有关更多信息，请参阅《Amazon PrivateLink 指南》中的 Access Amazon services through Amazon PrivateLink。

开始前的准备工作

开始之前，请确保您已执行以下任务：

- 查看《Amazon PrivateLink Guide》中的 Access an Amazon service using an interface VPC endpoint

注意事项

- 支持和限制：Amazon EKS 接口端点允许从您的 VPC 安全访问所有 Amazon EKS API 操作，但有特定的限制：它们不支持访问 Kubernetes API，因为其有单独的私有端点，因此您无法将 Amazon EKS 配置为仅通过接口端点访问。
- 定价：使用 Amazon EKS 的接口端点会产生标准 Amazon PrivateLink 费用：每个可用区中预置的每个端点的小时费用，以及通过端点的流量的数据处理费用。要了解更多信息，请参阅 Amazon PrivateLink 定价。
- 安全和访问控制：我们建议通过以下附加配置增强安全性和控制访问 – 使用 VPC 端点策略控制通过接口端点访问 Amazon EKS，将安全组与端点网络接口关联以管理流量，使用 VPC 流日志捕获和监控进出接口端点的 IP 流量，日志可发布到 Amazon CloudWatch 或 Amazon S3。要了解更多信息，请参阅 Control access to VPC endpoints using endpoint policies 和使用 VPC 流日志记录 IP 流量。
- 连接选项：接口端点提供灵活的连接选项，支持通过本地访问（使用 Amazon Direct Connect 或 Amazon Site-to-Site VPN 将本地数据中心连接到包含接口端点的 VPC）或通过 VPC 间连接（使用 Amazon Transit Gateway 或 VPC 对等连接将其他 VPC 连接到包含接口端点的 VPC，从而将流量保持在 Amazon 网络内）。

- IP 版本支持：在 2024 年 8 月之前创建的端点仅支持使用 eks.region.amazonaws.com 的 IPv4。2024 年 8 月之后创建的新端点支持双堆栈 IPv4 和 IPv6（例如，eks.region.amazonaws.com、eks.region.api.aws）。
- 区域可用性：适用于 EKS API 的 Amazon PrivateLink 在以下区域不可用：亚太地区（马来西亚）（ap-southeast-5）、亚太地区（泰国）（ap-southeast-7）、墨西哥（中部）（mx-central-1）和亚太地区（台北）（ap-east-2）区域。AmazonPrivateLink 对 eks-auth（EKS 容器组身份）的支持已在亚太地区（马来西亚）(ap-southeast-5) 区域开放。

为 Amazon EKS 创建接口端点

您可以使用 Amazon VPC 控制台或 Amazon 命令行界面（Amazon CLI）为 Amazon EKS 创建接口端点。有关更多信息，请参阅《Amazon PrivateLink 指南》中的创建 VPC 端点。

使用以下服务名称为 Amazon EKS 创建接口端点：

EKS API

- com.amazonaws.region-code.eks
- com.amazonaws.region-code.eks-fips（适用于符合 FIPS 的端点）

EKS Auth API（EKS 容器组身份）

- com.amazonaws.region-code.eks-auth

Amazon EKS 接口端点的私有 DNS 功能

Amazon EKS 和其他 Amazon 服务的接口端点默认启用私有 DNS 功能，便于使用默认区域 DNS 名称进行安全和私有 API 请求。此功能可确保 API 调用通过接口端点经由私有 Amazon 网络路由，从而增强安全性和性能。

当您为 Amazon EKS 或其他 Amazon 服务创建接口端点时，将自动激活 DNS 功能。要启用该功能，您需要通过设置特定属性来正确配置您的 VPC：

- enableDnsHostnames：允许 VPC 内的实例拥有 DNS 主机名。
- enableDnsSupport：在整个 VPC 中启用 DNS 解析。

有关检查或修改这些设置的分步说明，请参阅查看和更新 VPC 的 DNS 属性。

DNS 名称和 IP 地址类型

启用私有 DNS 功能后，您可以使用特定的 DNS 名称连接到 Amazon EKS，这些选项会随着时间的推移而不断演进：

- eks.region.amazonaws.com：传统的 DNS 名称，在 2024 年 8 月之前仅解析为 IPv4 地址。对于更新为双堆栈的现有端点，此名称将同时解析为 IPv4 和 IPv6 地址。
- eks.region.api.aws：适用于 2024 年 8 月之后创建的新端点，此双堆栈 DNS 名称可同时解析为 IPv4 和 IPv6 地址。

2024 年 8 月之后，新的接口端点将包含两个 DNS 名称，您可以选择双堆栈 IP 地址类型。对于现有端点，更新为双堆栈将修改 eks.region.amazonaws.com 以同时支持 IPv4 和 IPv6。

使用私有 DNS 功能

配置完成后，可以将私有 DNS 功能集成到您的工作流程中，提供以下功能：

- API 请求：根据端点的设置，使用默认区域 DNS 名称（`eks.region.amazonaws.com` 或 `eks.region.api.aws`）向 Amazon EKS 发出 API 请求。
- 应用程序兼容性：调用 EKS API 的现有应用程序无需进行任何更改即可利用此功能。
- 具有双堆栈的 Amazon CLI：要将双堆栈端点与 Amazon CLI 结合使用，请参阅《Amazon SDKs and Tools Reference Guide》中的 Dual-stack and FIPS endpoints 配置。
- 自动路由：对 Amazon EKS 默认服务端点的任何调用都会通过接口端点自动定向，从而确保连接私有和安全。

了解 Amazon EKS 集群中的弹性

Amazon全球基础架构围绕Amazon区域和可用区构建。Amazon区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

Amazon EKS 跨多个Amazon可用区运行和扩展 Kubernetes 控制层面以确保高可用性。Amazon EKS 可以根据负载自动缩放控制层面实例，检测并替换运行状况不佳的控制层面实例，并自动修补控制层面。启动版本更新后，Amazon EKS 会为您更新控制面板，从而在更新期间保持控制面板的高可用性。

此控制面板包含至少两个 API 服务器实例和三个 etcd 实例，这些实例在一个 Amazon 区域内的三个可用区之间运行。Amazon EKS：

- 主动监控控制面板实例上的负载，并自动扩展以确保高性能。
- 自动检测和替换运行状况不佳的控制面板实例，并根据需要跨 Amazon 区域内的可用区重新启动它们。
- 利用Amazon区域的架构以保持高可用性。因此，Amazon EKS 能够提供确保 API 服务器端点可用性的 SLA。

有关 Amazon 区域和可用区的更多信息，请参阅 Amazon 全球基础设施。

防止在 Amazon EKS 中出现跨服务混淆代理

混淆代理问题是一个安全问题，即没有执行操作权限的实体可能会迫使更具权限的实体执行该操作。在 Amazon 中，跨服务模拟可能会导致混淆代理问题。一个服务（呼叫服务）调用另一项服务（所谓的服务）时，可能会发生跨服务模拟。可以操纵调用服务以使用其权限对另一个客户的资源进行操作，否则该服务不应有访问权限。为防止这种情况，Amazon 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务主体有权限访问账户中的资源。

我们建议在资源策略中使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键，以限制 Amazon Elastic Kubernetes Service（Amazon EKS）为其他服务提供的资源访问权限。

aws:SourceArn

使用 aws:SourceArn 来仅将一个资源与跨服务访问相关联。

aws:SourceAccount

使用 aws:SourceAccount 来让该账户中的任何资源与跨服务使用相关联。

防范混淆代理问题最有效的方法是使用 aws:SourceArn 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符（*）的 aws:SourceArn 全局上下文条件键。例如 arn:aws-cn:<servicename>:*:<123456789012>:*。

如果 aws:SourceArn 值不包含账户 ID，例如 Amazon S3 桶 ARN，您必须使用 aws:SourceAccount 和 aws:SourceArn 来限制权限。

防止 Amazon EKS 集群角色出现跨服务混淆代理

每个集群都需要一个 Amazon EKS 集群 IAM 角色。由 Amazon EKS 管理的 Kubernetes 集群会使用此角色来管理节点，而传统云提供商会使用此角色为服务创建带有 Elastic Load Balancing 的负载均衡器。这些集群操作只能影响同一账户，因此建议您将每个集群角色限制为该集群和账户。这是 Amazon 建议的具体应用，即在您的账户中遵循最低权限原则。

资源 ARN 格式

`aws:SourceArn` 的值必须是 EKS 集群的 ARN，格式为 `arn:aws-cn:eks:region:account:cluster/cluster-name` 。例如， `arn:aws-cn:eks:us-west-2:123456789012:cluster/my-cluster`。

EKS 集群角色的信任策略格式

以下示例演示如何使用 Amazon EKS 中的 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键来防止混淆代理问题。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:eks:us-west-2:123456789012:cluster/my-cluster"
          },
        "StringEquals": {
            "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Kubernetes 的安全注意事项

以下是云安全的注意事项，这些注意事项会对 Amazon EKS 集群中的 Kubernetes 造成影响。要深入了解 Kubernetes 中的安全控制措施和实践，请参阅 Kubernetes 文档中的云原生安全和 Kubernetes。

主题

- 使用 Kubernetes 证书保护工作负载
- 了解 Amazon EKS 如何创建 RBAC 角色和用户
- 在现有集群上使用 KMS 加密 Kubernetes 密钥
- 将 Amazon Secrets Manager 密钥与 Amazon EKS 容器组（pod）结合使用
- 所有 Kubernetes API 数据默认启用信封加密

使用 Kubernetes 证书保护工作负载

Kubernetes 证书 API 可自动执行 X.509 凭证预置。该 API 具有一个命令行界面，供 Kubernetes API 客户端从证书颁发机构（CA）请求和获取 X.509 证书。您可以使用 `CertificateSigningRequest`（CSR）资源请求指示签署人对证书进行签名。您的请求在签署前获得批准或被拒绝。Kubernetes 支持内置签署人和具有明确定义行为的自定义签署人。这样，客户端就可以预测 CSR 会发生什么。要了解有关证书签名的更多信息，请参阅签名请求。

内置签署人之一是 `kubernetes.io/legacy-unknown`。CSR 资源的 `v1beta1` API 遵循这个旧版未知的签署人。但是，CSR 的稳定 `v1` API 不允许将 `signerName` 设置为 `kubernetes.io/legacy-unknown`。

如果想使用 Amazon EKS CA 在集群上生成证书，则您必须使用自定义签署人。要使用 CSR `v1` API 版本并生成新证书，必须迁移任何现有清单和 API 客户端。使用现有 `v1beta1` API 创建的现有证书在证书到期之前有效且正常运行。这包括以下这些：

- 信任分配：无。在 Kubernetes 集群中，此签署人没有标准的信任或分配。
- 允许的主题：任何
- 允许的 x509 扩展：遵循 subjectAltName 和密钥使用扩展，并丢弃其他扩展
- 允许的密钥用法：不得包括 [“密钥加密”、“数字签名”、“服务器身份验证”] 以外的用法

> **Note**
>
> 不支持客户端证书签名。

- 到期/证书使用寿命：1 年（默认值和最大值）
- 允许/不允许 CA 位：不允许

使用 signerName 生成 CSR 示例

这些步骤介绍如何使用 signerName: beta.eks.amazonaws.com/app-serving 为 DNS 名称 myserver.default.svc 生成服务证书。将此用作您自己环境的指南。

1. 运行 openssl genrsa -out myserver.key 2048 命令以生成 RSA 私有密钥。

```
openssl genrsa -out myserver.key 2048
```

2. 运行以下命令以生成证书请求。

```
openssl req -new -key myserver.key -out myserver.csr -subj "/CN=myserver.default.svc"
```

3. 为 CSR 请求生成 base64 值，并将其存储在变量中，以便在后续步骤中使用。

```
base_64=$(cat myserver.csr | base64 -w 0 | tr -d "
")
```

4. 要创建名为 mycsr.yaml 的文件，请运行以下命令。在以下示例中，beta.eks.amazonaws.com/app-serving 是 signerName。

```
cat >mycsr.yaml <<EOF
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: myserver
spec:
  request: $base_64
  signerName: beta.eks.amazonaws.com/app-serving
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

5. 提交 CSR。

```
kubectl apply -f mycsr.yaml
```

6. 批准服务证书。

```
kubectl certificate approve myserver
```

7. 验证证书是否已颁发。

```
kubectl get csr myserver
```

示例输出如下。

```
NAME       AGE     SIGNERNAME                              REQUESTOR          CONDITION
myserver   3m20s   beta.eks.amazonaws.com/app-serving   kubernetes-admin
 Approved,Issued
```

8. 导出已颁发的证书。

```
kubectl get csr myserver -o jsonpath='{.status.certificate}'| base64 -d >
 myserver.crt
```

了解 Amazon EKS 如何创建 RBAC 角色和用户

创建 Kubernetes 集群时，会在该集群上创建多个默认 Kubernetes 身份，以便 Kubernetes 正常运行。Amazon EKS 会为其每个默认组件创建 Kubernetes 身份。这些身份为集群组件提供 Kubernetes 基于角色的授权控制（RBAC）。有关更多信息，请参阅 Kubernetes 文档中的使用 RBAC 授权。

在向集群安装可选附加组件时，可能会向集群添加其他 Kubernetes 身份。有关本主题未涉及身份的更多信息，请参阅附加组件文档。

您可以使用 Amazon Web Services 管理控制台或 `kubectl` 命令行工具，查看 Amazon EKS 在集群上创建的 Kubernetes 身份列表。所有用户身份都会出现在 kube 审计日志中，可通过 Amazon CloudWatch 向你提供。

Amazon Web Services 管理控制台

先决条件

您使用的 IAM 主体必须拥有所需权限中描述的权限。

要使用Amazon Web Services 管理控制台查看 Amazon EKS 创建的身份

1. 打开 Amazon EKS 控制台。
2. 在 Clusters（集群）列表中，选择包含要查看的身份的集群。
3. 选择资源选项卡。
4. 在 Resource types（资源类型）下，选择 Authorization（授权）。
5. 选择 ClusterRoles、ClusterRoleBindings、Roles 或 RoleBindings。所有以 eks 为前缀的资源均由 Amazon EKS 创建。Amazon EKS 创建的其他身份资源包括：
   - ClusterRole 和名为 aws-node 的 ClusterRoleBinding。aws-node 资源支持适用于 Kubernetes 的 Amazon VPC CNI 插件，Amazon EKS 会将其安装在所有集群上。
   - 名为 vpc-resource-controller-role 的 ClusterRole 和名为 vpc-resource-controller-rolebinding 的 ClusterRoleBinding。这些资源支持 Amazon VPC 资源控制器，Amazon EKS 会将其安装在所有集群上。

   除了控制台中的资源外，即使以下特殊用户身份在集群配置中不可见，但它们仍存在于您的集群上：
   - **`eks:cluster-bootstrap`** – 在集群引导期间用于 `kubectl` 操作。
   - **`eks:support-engineer`** – 用于集群管理操作。
6. 选择特定资源以查看有关该资源的详细信息。默认情况下，信息在结构化视图中显示。在详细信息页面的右上角，您可以选择 Raw view（原始视图）以查看该资源的所有信息。

Kubectl

先决条件

用来列出集群上的 Kubernetes 资源的实体 [Amazon Identity and Access Management（IAM）或 OpenID Connect（OIDC）]，必须由 IAM 或您的 OIDC 身份提供者进行身份验证。必须向实体授予权限，才能为您希望该实体使用的集群上的 `Role`、`ClusterRole`、`RoleBinding` 和 `ClusterRoleBinding` 资源使用 Kubernetes `get` 和 `list` 动词。有关向 IAM 实体授予集群访问权限的更多信息，请参阅the section called “Kubernetes API 访问权限”。有关向经过您自己的 OIDC 提供者身份验证的实体授予集群访问权限的更多信息，请参阅the section called “关联 OIDC 提供者”。

要使用 **`kubectl`** 查看 Amazon EKS 创建的身份

为要查看的资源类型运行命令。所有以 eks 为前缀的返回资源均由 Amazon EKS 创建。除命令输出中返回的资源，即使以下特殊用户身份在集群配置中不可见，但它们仍存在于您的集群上：

- **eks:cluster-bootstrap** – 在集群引导期间用于 kubectl 操作。
- **eks:support-engineer** – 用于集群管理操作。

ClusterRoles – ClusterRoles 范围限定为您的集群，因此授予角色的任何权限都适用于集群上任何 Kubernetes 命名空间中的资源。

以下命令返回 Amazon EKS 在您的集群上创建的所有 Kubernetes ClusterRoles。

```
kubectl get clusterroles | grep eks
```

除了输出中返回的 ClusterRoles（具有前缀）外，还存在以下 ClusterRoles。

- **aws-node** – 此 ClusterRole 支持适用于 Kubernetes 的 Amazon VPC CNI 插件，Amazon EKS 会将其安装在所有集群上。
- **vpc-resource-controller-role** – 该 ClusterRole 支持 Amazon VPC 资源控制器，Amazon EKS 会将其安装在所有集群上。

要查看 ClusterRole 的规范，请将以下命令中的 *eks:k8s-metrics* 替换为上一条命令输出中返回的 ClusterRole。以下示例返回 *eks:k8s-metrics* ClusterRole 的规范。

```
kubectl describe clusterrole eks:k8s-metrics
```

示例输出如下。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
                    [/metrics]         []              [get]
  endpoints         []                 []              [list]
  nodes             []                 []              [list]
  pods              []                 []              [list]
  deployments.apps  []                 []              [list]
```

ClusterRoleBindings：ClusterRoleBindings 范围限定为您的集群。

以下命令返回 Amazon EKS 在您的集群上创建的所有 Kubernetes ClusterRoleBindings。

```
kubectl get clusterrolebindings | grep eks
```

除了输出中返回的 ClusterRoleBindings 外，还存在以下 ClusterRoleBindings。

- **aws-node** – 此 ClusterRoleBinding 支持适用于 Kubernetes 的 Amazon VPC CNI 插件，Amazon EKS 会将其安装在所有集群上。
- **vpc-resource-controller-rolebinding** – 该 ClusterRoleBinding 支持 Amazon VPC 资源控制器，Amazon EKS 会将其安装在所有集群上。

要查看 ClusterRoleBinding 的规范，请将以下命令中的 *eks:k8s-metrics* 替换为上一条命令输出中返回的 ClusterRoleBinding。以下示例返回 *eks:k8s-metrics* ClusterRoleBinding 的规范。

```
kubectl describe clusterrolebinding eks:k8s-metrics
```

示例输出如下。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name             Namespace
  ----  ----             ---------
  User  eks:k8s-metrics
```

Roles – Roles 范围限定为 Kubernetes 命名空间。Amazon EKS 创建的所有 Roles 范围限定为 kube-system 命名空间。

以下命令返回 Amazon EKS 在您的集群上创建的所有 Kubernetes Roles。

```
kubectl get roles -n kube-system | grep eks
```

要查看 Role 的规范，请将以下命令中的 *eks:k8s-metrics* 替换为上一条命令输出中返回的 Role 名称。以下示例返回 *eks:k8s-metrics* Role 的规范。

```
kubectl describe role eks:k8s-metrics -n kube-system
```

示例输出如下。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names              Verbs
  ---------         -----------------  --------------              -----
  daemonsets.apps   []                 [aws-node]                  [get]
  deployments.apps  []                 [vpc-resource-controller]  [get]
```

RoleBindings – RoleBindings 范围限定为 Kubernetes 命名空间。Amazon EKS 创建的所有 RoleBindings 范围限定为 kube-system 命名空间。

以下命令返回 Amazon EKS 在您的集群上创建的所有 Kubernetes RoleBindings。

```
kubectl get rolebindings -n kube-system | grep eks
```

要查看 RoleBinding 的规范，请将以下命令中的 *eks:k8s-metrics* 替换为上一条命令输出中返回的 RoleBinding。以下示例返回 *eks:k8s-metrics* RoleBinding 的规范。

```
kubectl describe rolebinding eks:k8s-metrics -n kube-system
```

示例输出如下。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name             Namespace
  ----  ----             ---------
  User  eks:k8s-metrics
```

在现有集群上使用 KMS 加密 Kubernetes 密钥

> **Important**
>
> 此过程仅适用于运行 Kubernetes 版本 1.27 或更低版本的 EKS 集群。如果运行的是 Kubernetes 1.28 或更高版本，则 Kubernetes 密钥将默认启用信封加密保护。有关更多信息，请参阅 the section called “所有 Kubernetes API 数据默认启用信封加密”。

如果启用密钥加密，则会使用您选择的 Amazon KMS 密钥对 Kubernetes 密钥加密。KMS 密钥必须符合以下条件：

- 对称
- 可以加密和解密数据
- 在与集群相同的 Amazon 区域中创建
- 如果 KMS 密钥是在其他账户中创建的，则 IAM 主体必须拥有对 KMS 密钥的访问权限。

有关更多信息，请参阅 Amazon Key Management Service 者指南 中的允许其它账户中的 IAM 主体使用 KMS 密钥。

> **Warning**
>
> 密钥加密启用后将无法禁用。此操作不可逆。

eksctl

此过程仅适用于运行 Kubernetes 版本 1.27 或更低版本的 EKS 集群。有关更多信息，请参阅 the section called “所有 Kubernetes API 数据默认启用信封加密”。

可以通过下列两种方式启用加密：

- 使用单个命令将加密添加到您的集群。

  要自动重新加密密钥，请运行以下命令。

```
eksctl utils enable-secrets-encryption \
```

```
    --cluster my-cluster \
    --key-arn arn:aws-cn:kms:region-code:account:key/key
```

要选择退出自动重新加密密钥，请运行以下命令。

```
eksctl utils enable-secrets-encryption
    --cluster my-cluster \
    --key-arn arn:aws-cn:kms:region-code:account:key/key \
    --encrypt-existing-secrets=false
```

- 使用 `kms-cluster.yaml` 文件向集群添加加密。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code

secretsEncryption:
  keyARN: arn:aws-cn:kms:region-code:account:key/key
```

要自动重新加密密钥，请运行以下命令。

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml
```

要选择退出自动重新加密密钥，请运行以下命令。

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml --encrypt-existing-
secrets=false
```

Amazon Web Services 管理控制台

a. 此过程仅适用于运行 Kubernetes 版本 1.27 或更低版本的 EKS 集群。有关更多信息，请参阅 the section called “所有 Kubernetes API 数据默认启用信封加密”。

b. 打开 Amazon EKS 控制台。

c. 选择要向其添加 KMS 加密的集群。

d. 选择 Overview（概述）选项卡（默认处于选中状态）。

e. 向下滚动到 Secrets encryption（密钥加密）部分，然后选择 Enable（启用）按钮。

f. 从下拉列表中选择一个密钥，然后选择 Enable（启用）按钮。如果未列出任何密钥，则必须先创建一个密钥。有关更多信息，请参阅创建密钥。

g. 选择 Confirm（确认）按钮以使用选定的密钥。

Amazon CLI

a. 此过程仅适用于运行 Kubernetes 版本 1.27 或更低版本的 EKS 集群。有关更多信息，请参阅 the section called “所有 Kubernetes API 数据默认启用信封加密”。

b. 使用以下 Amazon CLI 命令将密钥加密配置与您的集群相关联。将 example values 替换为您自己的值。

```
aws eks associate-encryption-config \
    --cluster-name my-cluster \
    --encryption-config '[{"resources":["secrets"],"provider":
{"keyArn":"arn:aws-cn:kms:region-code:account:key/key"}}]'
```

示例输出如下。

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "InProgress",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\"arn:aws-cn:kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734,
    "errors": []
  }
}
```

c. 您可以使用以下命令监控加密更新的状态。使用上一个输出中返回的特定 `cluster name` 和 `update ID`。当 `Successful` 状态显示时，更新完成。

```
aws eks describe-update \
    --region region-code \
    --name my-cluster \
```

```
    --update-id 3141b835-8103-423a-8e68-12c2521ffa4d
```

示例输出如下。

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "Successful",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\"arn:aws-cn:kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734>,
    "errors": []
  }
}
```

d. 要验证集群已启用加密，请运行 `describe-cluster` 命令。响应包含 `EncryptionConfig` 字符串。

```
aws eks describe-cluster --region region-code --name my-cluster
```

在集群上启用加密后，您必须使用新密钥加密所有现有密钥：

> **Note**
>
> 如果您使用 `eksctl`，只有在选择不自动重新加密密钥时才需要运行以下命令。

```
kubectl get secrets --all-namespaces -o json | kubectl annotate --overwrite -f - kms-
encryption-timestamp="time value"
```

> **Warning**
>
> 如果您为现有集群启用密钥加密，并且您使用的 KMS 密钥已被删除，那么将无法恢复集群。如果您删除 KMS 密钥，会将集群永久性置于降级状态。有关更多信息，请参阅删除 Amazon KMS 密钥..

> **Note**
>
> 默认情况下，`create-key` 命令会创建一个具有密钥政策的对称加密 KMS 密钥，该密钥政策向账户的根管理员授予对 Amazon KMS 操作和资源的访问权限。如果要缩小权限的范围，请确保允许对将调用 `create-cluster` API 的主体的策略执行 `kms:DescribeKey` 和 `kms:CreateGrant` 操作。
> 对于使用 KMS 信封加密的集群，需要具有 `kms:CreateGrant` 权限。CreateCluster 操作不支持条件 `kms:GrantIsForAWSResource`，也不应在 KMS 策略中用于控制执行 CreateCluster 的用户的 `kms:CreateGrant` 权限。

将 Amazon Secrets Manager 密钥与 Amazon EKS 容器组（pod）结合使用

要将 Secrets Manager 中的密钥和 Parameter Store 中的参数显示为挂载在 Amazon EKS 容器组（pod）中的文件，您可以使用 Kubernetes Secrets Store CSI Driver 的 Amazon Secrets and Configuration Provider（ASCP）插件。

使用 ASCP，您可以在 Secrets Manager 中存储并管理密钥，然后通过 Amazon EKS 上运行的工作负载检索。您可以使用 IAM 角色和策略限制集群中特定 Kubernetes 容器组（pod）的密钥访问权限。ASCP 检索容器组身份并交换 IAM 角色的身份。ASCP 担任容器组（pod）的 IAM 角色，然后可以从授权该角色的 Secrets Manager 中检索密码。

如果对密钥使用 Secrets Manager 自动轮换，还可以使用 Secrets Store CSI Driver 轮换协调程序功能，确保从 Secrets Manager 中检索最新的密码。

> **Note**
>
> 不支持 Amazon Fargate（Fargate）节点组。

有关更多信息，请参阅 Amazon Secrets Manager 用户指南中的在 Amazon EKS 中使用 Secrets Manager 密钥。

所有 Kubernetes API 数据默认启用信封加密

对于运行 Kubernetes 1.28 或更高版本的 EKS 集群，Amazon Elastic Kubernetes Service（Amazon EKS）为所有 Kubernetes API 数据提供了默认信封加密。

信封加密可以保护在 Kubernetes API 服务器上存储的数据。例如，信封加密会保护诸如 `ConfigMaps` 之类的 Kubernetes 集群配置。信封加密不适用于节点或 EBS 卷上的数据。EKS 以前支持加密 Kubernetes 密钥，这种信封加密现已扩展到所有 Kubernetes API 数据。

这提供了一种托管式的默认体验，可以为 Kubernetes 应用程序实现深度防御，并且不需要您执行任何操作。

Amazon EKS 将 Amazon Key Management Service（KMS）和 Kubernetes KMS provider v2 结合使用，借助由亚马逊云科技拥有的密钥来提供这一额外的安全防护层，此外还提供了让您从 Amazon KMS 自带客户自主管理型密钥（CMK）的选项。

了解信封加密

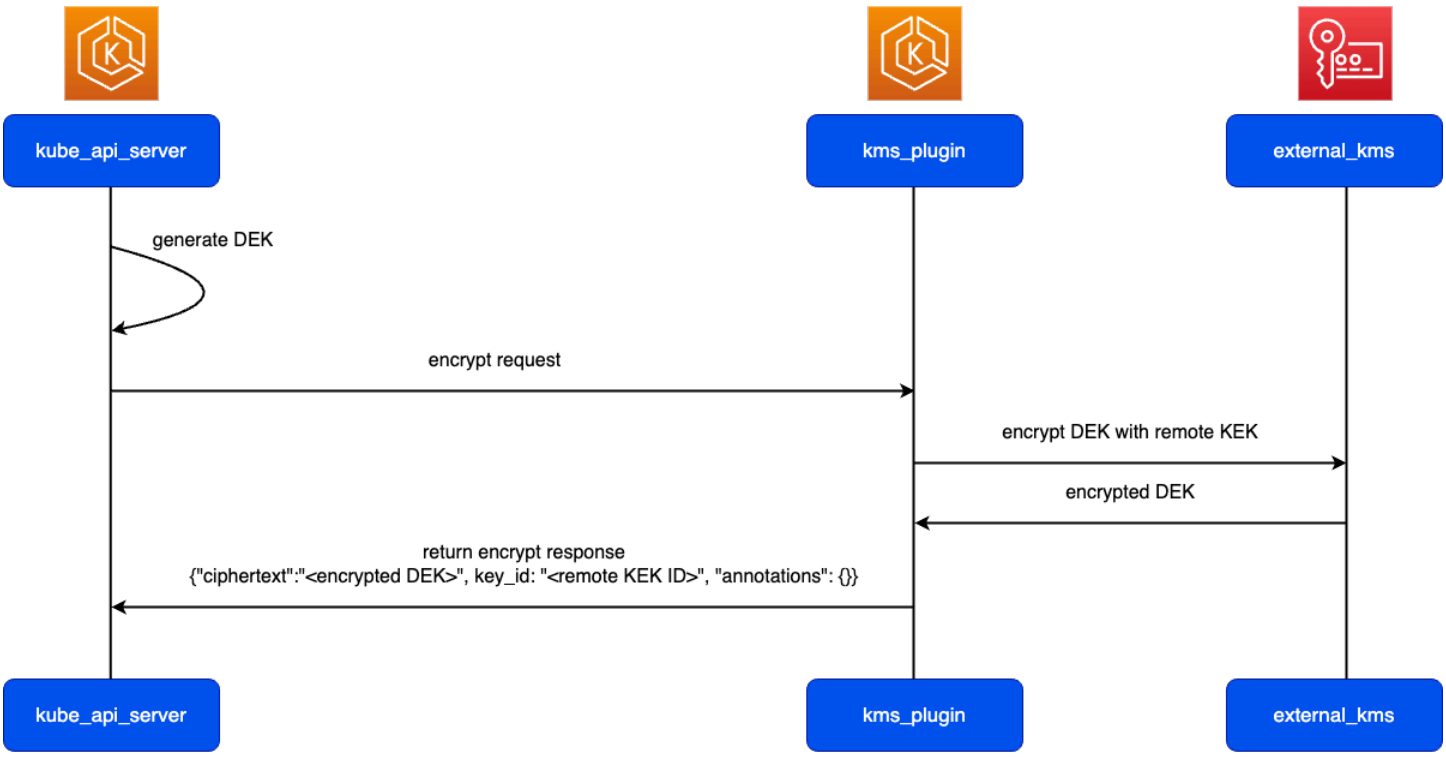
信封加密是一种加密明文数据的方法，在将数据发送到数据存储（etcd）之前会首先使用数据加密密钥（DEK）将数据加密，然后再使用根 KMS 密钥对 DEK 进行加密，而后者存储在一个远程集中管理的 KMS 系统（Amazon KMS）中。这是一种深度防御策略，首先使用加密密钥（DEK）来保护数据，然后使用安全存储的单独加密密钥 [称为密钥加密密钥（KEK）] 来保护 DEK，从而增加另一层安全防护。

Amazon EKS 如何使用 KMS v2 和 Amazon KMS 实现默认信封加密

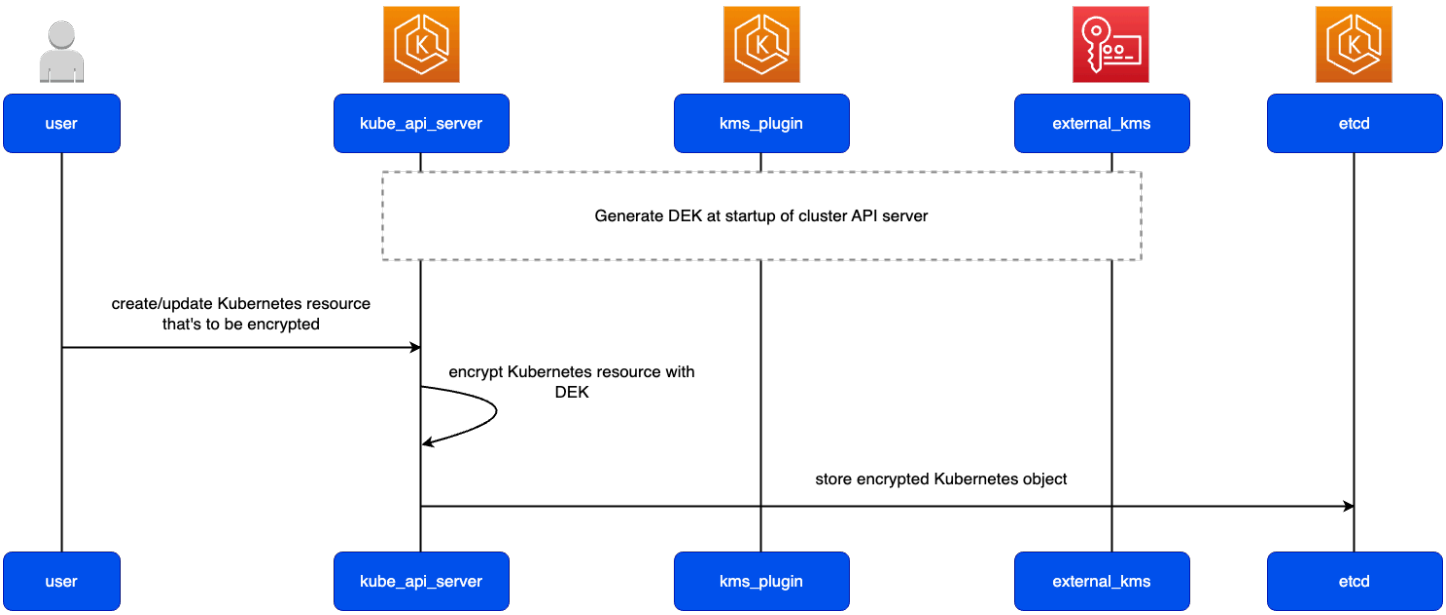
Amazon EKS 使用 KMS v2 为托管式 KBernetes 控制面板中的所有 API 数据实施默认信封加密，然后才会将其持久保存在 etcd 数据库中。集群 API 服务器在启动时会根据密钥种子和随机生成的数据组合来生成数据加密密钥（DEK）。此外，API 服务器在启动时还会调用 KMS 插件，使用来自 Amazon KMS 的远程密钥加密密钥（KEK）来加密 DEK 种子。这是在 API 服务器启动和 KEK 轮换时执行的一次性调用。然后，API 服务器会缓存加密的 DEK 种子。在此之后，API 服务器将根据密钥派生函数（KDF），使用缓存的 DEK 种子生成其他一次性的 DEK。在这些生成的 DEK 中，每个都只能使用一次来加密单个 Kubernetes 资源，然后才会将资源存储在 etcd 中。通过使用 KMS v2 中的加密缓存 DEK 种子，在 API 服务器中加密 Kubernetes 资源的过程不仅性能更高，而且成本效益更好。

默认情况下，此 KEK 由 Amazon 拥有，但您可以从 Amazon KMS 自带密钥。

下图展示了 API 服务器启动时生成和加密 DEK 的过程。



下面的简要示意图展示了在将 Kubernetes 资源存储到 etcd 之前对其进行加密的过程。



常见问题

默认信封加密是如何提升 EKS 集群安全状况的？

此功能可减少未加密元数据和客户内容的暴露面并缩短暴露时间。启用默认信封加密时，元数据和客户内容在存储到 etcd 之前，只会在 kube-apiserver 的内存中处于短暂未加密状态。kube-apiserver 的内

存是通过 Nitro 系统保护的。Amazon EKS 的托管式 Kubernetes 控制面板仅使用基于 Nitro 的 EC2 实例。此类实例采用安全控制设计，可防止任何系统或人员访问其内存。

我需要运行哪个版本的 Kubernetes 才能启用此功能？

要启用默认信封加密，Amazon EKS 集群必须运行 Kubernetes 版本 1.28 或更高版本。

如果我运行的 Kubernetes 集群版本不支持此功能，我的数据还安全吗？

可以。在 Amazon，安全性是我们的第一要务。我们所有的数字化转型和创新都以最高的安全运营实践为基石，并矢志不断提升改进。

无论运行的是哪个 Kubernetes 版本，存储在 etcd 中的所有数据都会在每个 EKS 集群进行磁盘级别加密。EKS 使用根密钥生成卷加密密钥，这些密钥由 EKS 服务托管。此外，每个 Amazon EKS 集群都使用集群专用的虚拟机在隔离的 VPC 中运行。得益于这种架构以及我们在运营安全方面的实践，Amazon EKS 通过了多项合规评级和标准认证，包括 SOC 1,2,3、PCI-DSS、ISO 和 HIPAA 资格等。无论是否使用默认信封加密功能，所有 EKS 集群都将始终遵循这些合规评级和标准。

Amazon EKS 中信封加密的工作原理是什么？

集群 API 服务器在启动时会根据密钥种子和随机生成的数据组合来生成数据加密密钥（DEK）。此外，API 服务器在启动时还会调用 KMS 插件，使用来自 Amazon KMS 的远程密钥加密密钥（KEK）来加密 DEK。这是在 API 服务器启动和 KEK 轮换时执行的一次性调用。然后，API 服务器会缓存加密的 DEK 种子。在此之后，API 服务器将根据密钥派生函数（KDF），使用缓存的 DEK 种子生成其他一次性的 DEK。在这些生成的 DEK 中，每个都只能使用一次来加密单个 Kubernetes 资源，然后才会将资源存储在 etcd 中。

但请注意，API 服务器还会进行其他调用来验证 Amazon KMS 集成的运行状况和正常功能。这些额外的运行状况检查在 Amazon CloudTrail 中可见。

我是否需要执行任何操作或更改任何权限才能在 EKS 集群中使用此功能？

不需要，您不需要执行任何操作。信封加密现已是 Amazon EKS 中的一个默认配置，已在所有运行 Kubernetes 1.28 或更高版本的集群中启用。Amazon KMS 集成是由 Amazon 管理的 Kubernetes API 服务器建立的。这意味着您无需配置任何权限即可开始为集群使用 KMS 加密。

如何才能知道集群是否启用了默认信封加密？

如果迁移到使用自己的 CMK，则会看到与集群关联的 KMS 密钥的 ARN。此外，您还可以查看与集群使用 CMK 相关的 Amazon CloudTrail 事件日志。

如果集群使用的密钥由 Amazon 拥有，则会在 EKS 控制台中提供详细信息（不包括密钥的 ARN）。

如果在 Amazon EKS 中用于默认信封加密的密钥由 Amazon 拥有，Amazon 是否可以访问该密钥？

不能。Amazon 在 Amazon EKS 中执行严格的安全控制，可防止任何人访问用于保护 etcd 数据库中数据的任何明文加密密钥。这些安全措施也适用于由 Amazon 拥有的 KMS 密钥。

我现有的 EKS 集群是否启用了默认信封加密？

如果您运行的是 Kubernetes 版本 1.28 或更高版本的 Amazon EKS 集群，则会为所有 Kubernetes API 数据启用信封加密。对于现有集群，Amazon EKS 使用 `eks:kms-storage-migrator` RBAC 集群角色将以前未在 etcd 中进行信封加密的数据迁移到这一新加密状态。

如果我已在 EKS 集群中为密钥启用了信封加密，这意味着什么？

如果您使用 KMS 中现有的客户自主管理型密钥（CMK）来进行 Kubernetes 密钥的信封加密，则该密钥将用作 KEK，用于对集群中的所有 Kubernetes API 数据类型进行信封加密。

运行启用了默认信封加密的 EKS 集群会是否会产生额外的成本？

如果您使用亚马逊云科技拥有的密钥进行默认信封加密，则托管式的 Kubernetes 控制面板不会产生任何额外的成本。默认情况下，每个运行 Kubernetes 1.28 或更高版本的 EKS 集群都使用由亚马逊云科技拥有的密钥。但如果您使用自己的 Amazon KMS 密钥，则将正常的 KMS 定价收费。

使用我自己的 Amazon KMS 密钥对集群中的 Kubernetes API 数据进行加密，将如何收费？

您每月需要支付 1 美元来存储您创建或导入到 KMS 的任何自定义密钥。KMS 会对加密和解密请求收费。每个账户的免费套餐为每月 2 万个请求，超过免费套餐后，每月每 1 万个请求需支付 0.03 美元。这适用于账户的所有 KMS 使用量，因此在集群上使用自己的 Amazon KMS 密钥时，成本将取决于其他集群或账户中 Amazon 资源使用该密钥的情况。

现在我使用客户自主管理型密钥（CMK）来进行所有 Kubernetes API 数据的信封加密，而不仅限于密钥，我的 KMS 成本是否会增加？

不会。我们使用 KMS v2 来实现加密，显著减少了对 Amazon KMS 的调用量，从而降低了与 CMK 相关的成本，无论 EKS 集群中是否有额外的 Kubernetes 数据需要加密或解密。

如上所述，生成的用于加密 Kubernetes 资源的 DEK 种子在使用远程 KEK 加密后，会存储在 Kubernetes API 服务器的本地缓存中。如果加密的 DEK 种子不在 API 服务器的缓存中，API 服务器将调用 Amazon KMS 来加密 DEK 种子。然后 API 服务器会缓存加密的 DEK 种子，以备将来在集

群中使用，从而不再需要调用 KMS。同样，对于解密请求，API 服务器将针对第一个解密请求调用 Amazon KMS，然后缓存解密后的 DEK 种子并用于未来的解密操作。

有关更多信息，请参阅 GitHub 上 Kubernetes 增强功能中的 KEP-3299: KMS v2 Improvements。

我能否将同一 CMK 密钥用于多个 Amazon EKS 集群？

可以。要重复使用某个密钥，可以在创建集群时将该 ARN 关联到集群，从而将其关联到同一区域内的集群。但如果要将同一 CMK 用于多个 EKS 集群，则应采取必要的措施来防止对该 CMK 的随意禁用。否则，关联到多个 EKS 集群的 CMK 被禁用将对这些集群产生更广泛的影响，具体取决于该密钥。

如果启用默认信封加密后我的 CMK 不再可用，我的 EKS 集群会发生什么？

如果禁用了某个 KMS 密钥，则将不能用于任何加密操作。如果无法访问现有的 CMK，API 服务器将无法加密并持久保存任何新创建的 Kubernetes 对象，也无法解密 etcd 中存储的任何先前加密的 Kubernetes 对象。如果 CMK 被禁用，集群将立即进入运行不正常/性能降级状态，并且在您重新启用关联的 CMK 之前，我们将无法履行我们的服务承诺。

禁用 CMK 后，您将收到通知，提示 EKS 集群运行性能降级，并且需要在禁用 CMK 后 30 天内重新启用，才能确保成功恢复 Kubernetes 控制面板资源。

如何确保我的 EKS 集群不受已禁用/删除 CMK 的影响？

为确保 EKS 集群不受此类情况的影响，密钥管理员应使用遵循最低权限原则的 IAM 策略来管理 KMS 密钥操作权限，减少随意禁用或删除与 EKS 集群关联的密钥的风险。此外，您可以设置 CloudWatch 警报来接收有关 CMK 状态的通知。

如果重新启用 CMK，我的 EKS 集群会恢复吗？

为确保成功恢复 EKS 集群，我们强烈建议您在禁用 CMK 后 30 天内重新启用。但是，是否能够成功恢复 EKS 集群，还取决于是否因集群处于运行不正常/性能降级状态时的自动 Kubernetes 升级导致了任何 API 更改中断。

禁用 CMK 后，为什么我的 EKS 集群处于运行不正常/性能降级状态？

EKS 控制面板的 API 服务器将一个 DEK 密钥加密后缓存在 API 服务器内存中，并在对象创建/更新操作期间使用该密钥对所有对象进行加密，然后才会将对象存储在 etcd 中。从 etcd 检索现有对象时，API 服务器会使用同一缓存的 DEK 密钥来解密 Kubernetes 资源对象。如果您禁用了 CMK，由于 API 服务器的内存中缓存了该 DEK 密钥，API 服务器将不会立即看到任何影响。但在 API 服务器实例

重启时，将不会有缓存的 DEK，需要调用 Amazon KMS 进行加密和解密操作。由于没有 CMK，此过程将会失败，并显示 KMS_KEY_DISABLED 错误代码，导致 API 服务器不能成功启动。

如果我删除了 CMK，我的 EKS 集群会发生什么情况？

删除与 EKS 集群关联的 CMK 密钥会使其运行状况降级，无法恢复。如果没有现有的 CMK，API 服务器将不再能够加密并持久保存任何新 Kubernetes 对象，也无法解密 etcd 数据库中存储的任何先前加密的 Kubernetes 对象。只有当您确信不再需要使用该 EKS 集群时，才应删除 EKS 集群的 CMK 密钥。

请注意，如果找不到 CMK（KMS_KEY_NOT_FOUND）或与集群关联的 CMK 授权被撤销（KMS_GRANT_REVOKED），则集群将无法恢复。有关集群运行状况和错误代码的更多信息，请参阅集群运行状况常见问题解答和错误代码以及解析路径。

对于因禁用或删除了 CMK 而性能降级/运行不正常的 EKS 集群，我是否仍需要付费？

可以。尽管 EKS 控制面板在 CMK 被禁用时将无法使用，但在客户将 EKS 集群删除之前，Amazon 仍将运行分配给该集群的专用基础设施资源。此外，发生这种情况时，我们的服务承诺将不适用，因为这是客户的自愿行为或不作为阻碍了 EKS 集群的正常运行和操作。

当我的 EKS 集群因禁用 CMK 而处于运行不正常/性能降级状态时，能否自动升级？

可以。但是，如果集群禁用了 CMK，您可在 30 天内将其重新启用。在此 30 天内，Kubernetes 集群不会自动升级。如果您未在此期限内重新启用 CMK，则集群将按照 EKS 中的 Kubernetes 版本生命周期，在此期限届满时自动升级到提供标准支持的下一版本（n+1）。

我们强烈建议您在意识到集群受影响时尽快重新启用被禁用的 CMK。请注意，尽管 EKS 会自动升级这些受影响的集群，但不能保证能成功恢复集群，尤其是在集群经过多次自动升级的情况下，因为这可能包括 Kubernetes API 更改以及 API 服务器引导过程中的意外行为。

是否可以使用 KMS 密钥别名？

可以。Amazon EKS 支持使用 KMS 密钥别名。别名是一个易记的亚马逊云科技 KMS 密钥名称。例如，您可以用别名将 KMS 密钥称为 my-key，而不是 **1234abcd-12ab-34cd-56ef-1234567890ab** 。

是否仍可使用自己的 Kubernetes 备份解决方案备份和恢复集群资源？

可以。您可以使用 Kubernetes 备份解决方案（例如 Velero）来进行 Kubernetes 集群的灾难恢复、数据迁移和数据保护。如果您运行的 Kubernetes 备份解决方案会通过 API 服务器访问集群资源，则应用

程序检索到的任何数据都将在到达客户端之前进行解密，以便您将集群资源恢复到其他 Kubernetes 集群中。

Amazon EKS 自动模式的安全注意事项

本主题介绍了 Amazon EKS 自动模式的安全架构、控制和最佳实践。随着组织大规模部署容器化应用程序，保持良好的安全态势变得越来越复杂。EKS 自动模式实施自动化的安全控制并与 Amazon 安全服务集成，以帮助您保护集群基础设施、工作负载和数据。通过强制节点生命周期管理和自动补丁部署等内置安全功能，EKS 自动模式可帮助您在减少运营开销的同时保持遵循安全最佳实践。

在继续阅读本主题之前，请确保您熟悉 EKS 自动模式的基本概念，并已检查了在集群上启用 EKS 自动模式的先决条件。有关 Amazon EKS 安全性的一般信息，请参阅 安全性。

Amazon EKS 自动模式以 Amazon EKS 现有的安全性为基础，同时为 EC2 托管式实例引入了额外的自动安全控制措施。

API 安全性和身份验证

Amazon EKS 自动模式使用 Amazon 平台安全机制来保护并对 Amazon EKS API 调用进行身份验证。

- 对 Kubernetes API 的访问通过与 Amazon IAM 身份集成的 EKS 访问条目进行保护。
  - 有关更多信息，请参阅 the section called “访问条目”。
- 客户可以通过配置 EKS 访问条目来实施对 Kubernetes API 端点的精细访问控制。

网络安全

Amazon EKS 自动模式支持多层网络安全性：

- VPC 集成
  - 在 Amazon Virtual Private Cloud（VPC）中运行
  - 支持自定义 VPC 配置和子网布局
  - 支持集群组件之间的私有联网
  - 有关更多信息，请参阅管理 Amazon Virtual Private Cloud 的安全责任
- 网络策略
  - 对 Kubernetes 网络策略的原生支持
  - 定义精细网络流量规则的功能

- 有关更多信息，请参阅 the section called “Kubernetes 策略”

EC2 托管式实例安全性

Amazon EKS 自动模式使用以下安全控制措施运行 EC2 托管式实例：

EC2 安全性

- EC2 托管式实例保持了 Amazon EC2 的安全功能。
- 有关 EC2 托管式实例的更多信息，请参阅 Amazon EC2 中的安全性。

实例生命周期管理

对于由 EKS 自动模式运行的 EC2 托管式实例，最长生命周期为 21 天。Amazon EKS 自动模式会自动终止超过此生命周期的实例。此生命周期限制有助于防止配置偏差并保持安全态势。

数据保护

- Amazon EC2 实例存储直接挂载到实例并进行了加密。有关更多信息，请参阅 Amazon EC2 中的数据保护。
- EKS 自动模式负责管理在创建时挂载到 EC2 实例的卷，包括根卷和数据卷。EKS 自动模式并不完全管理使用 Kubernetes 持久性存储功能创建的 EBS 卷。

补丁管理

- Amazon EKS 自动模式会自动将补丁应用于托管式实例。
- 这些补丁包括：
  - 操作系统更新
  - 安全补丁
  - Amazon EKS 自动模式组件

> **Note**
>
> 客户继续负责保护和更新在这些实例上运行的工作负载。

访问控制

- 直接实例访问权限受到限制：
  - 不支持 SSH 访问。
  - 不支持 Amazon Systems Manager 会话管理器（SSM）。
- 管理操作通过 Amazon EKS API 和 Kubernetes API 执行。

自动化的资源管理

Amazon EKS 自动模式并不完全管理使用 Kubernetes 持久性存储功能创建的 Amazon Elastic Block Store（Amazon EBS）卷。EKS 自动模式也不会管理弹性负载均衡（ELB）。Amazon EKS 自动模式可自动执行这些资源的例行任务。

存储安全性

- Amazon 建议您为 Kubernetes 持久性存储功能预置的 EBS 卷启用加密。有关更多信息，请参阅 the section called “创建 StorageClass”。
- 使用 Amazon KMS 的静态加密
- 您可以配置 Amazon 账户对您创建的新 EBS 卷和快照副本进行加密。有关更多信息，请参阅《Amazon EBS 用户指南》中的 Enable Amazon EBS encryption by default。
- 有关更多信息，请参阅 Security in Amazon EBS。

负载均衡器安全性

- 自动配置负载均衡器
- 通过 Amazon Certifice Manager 集成来管理 SSL/TLS 证书
- 通过安全组自动化实施负载均衡器访问控制
- 有关更多信息，请参阅 Security in Elastic Load Balancing。

安全最佳实践

下一部分介绍 Amazon EKS 自动模式的安全最佳实践。

- 定期检查 Amazon IAM 策略和 EKS 访问条目。

- 为工作负载实施最低权限访问模式。
- 通过 Amazon CloudTrail 和 Amazon CloudWatch 监控集群活动。有关更多信息，请参阅the section called “ Amazon CloudTrail” 和the section called “Amazon CloudWatch”。
- 使用 Amazon Security Hub 进行安全态势评估。
- 实施适合工作负载的容器组安全标准。

EKS 功能的安全注意事项

本主题涵盖了 EKS 功能的重要安全注意事项，包括 IAM 角色配置、Kubernetes 权限、多集群部署的架构模式和跨账户 Amazon 资源管理。

EKS 功能使用 IAM 角色、EKS 访问条目和 Kubernetes RBAC 的组合来提供对 Amazon 服务、集群内 Kubernetes 资源以及与 Amazon CodeConnections、Amazon Secrets Manager 和其他 Amazon 服务的集成的安全访问。

功能 IAM 角色

创建功能时，您将提供一个 IAM 功能角色，EKS 使用该角色代表您执行操作。此角色必须满足以下要求：

- 与集群和功能资源位于同一 Amazon 账户中
- 拥有信任策略，允许 `capabilities.eks.amazonaws.com` 服务主体代入该角色
- 拥有适合功能类型和使用案例的 IAM 权限，具体取决于您的需求。有关所需 IAM 权限的详细信息，请参阅the section called “ Amazon CodeConnections”、the section called “ Amazon Secrets Manager”和the section called “配置权限”

最佳做法是考虑您的特定使用案例所需的权限范围，并仅授予满足您的要求所必需的权限。例如，在使用 Kube Resource Orchestrator 的 EKS 功能时，可能不需要 IAM 权限，而在使用 Amazon Controllers for Kubernetes 的 EKS 功能时，您可以授予对一项或多项 Amazon 服务的完全访问权限。

> **Important**
>
> 虽然某些使用案例可能需要使用广泛的管理权限，但要遵循最低权限原则，即仅授予特定使用案例所需的最低 IAM 权限，使用 ARN 和条件键而不是使用通配符权限来限制对特定资源的访问。

有关创建和配置功能 IAM 角色的详细信息，请参阅 the section called “功能 IAM 角色”。

EKS 访问条目

当您使用 IAM 角色创建功能时，Amazon EKS 会在您的集群上自动为该角色创建访问条目。此访问条目授予功能基准 Kubernetes 权限，使其能够运行。

> **Note**
>
> 将为创建该功能所在的集群创建访问条目。要将 Argo CD 部署到远程集群，您必须在这些集群上创建访问条目，使其具有部署和管理应用程序的 Argo CD 功能的相应权限。

访问条目包括：

- 作为主体的 IAM 角色 ARN
- 授予基准 Kubernetes 权限的功能特定访问条目策略
- 基于功能类型的适当范围（集群范围或命名空间范围）

> **Note**
>
> 对于 Argo CD，将向功能配置中指定的命名空间授予命名空间范围的权限（默认为 `argocd`）。

按功能划分的默认访问条目策略

每种功能类型都向功能角色授予所需的权限，设置如下所示的不同默认访问条目策略：

kro

- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSKROPolicy`（集群范围）

  授予监视和管理 ResourceGraphDefinitions 以及创建由 RGD 定义的自定义资源实例的权限。

确认

- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSACKPolicy`（集群范围）

  授予在所有命名空间中创建、读取、更新和删除 ACK 自定义资源的权限。

Argo CD

- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSArgoCDClusterPolicy`（集群范围）

  授予 Argo CD 的集群级别权限，以便发现资源，并管理集群范围内的对象。

- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSArgoCDPolicy`（命名空间范围）

  授予 Argo CD 部署和管理应用程序的命名空间级权限。范围限定为功能配置中指定的命名空间（默认为 `argocd`）。

有关更多详细信息，请参阅the section called “检查访问策略”。

其他 Kubernetes 权限

某些功能可能需要除默认访问条目策略之外的其他 Kubernetes 权限。您可以使用以下任一方法授予这些权限：

- 访问条目策略：将其他托管策略与访问条目相关联
- Kubernetes RBAC：为该功能的 Kubernetes 用户创建 `Role` 或 `ClusterRole` 绑定

ACK 密钥读取器权限

某些 ACK 控制器需要读取 Kubernetes 密钥才能检索数据库密码等敏感数据。以下 ACK 控制器需要密钥读取权限：

- `acm`, `acmpca`, `documentdb`, `memorydb`, `mq`, `rds`, `secretsmanager`

要授予密钥读取权限，请执行以下操作：

1. 将 `arn:aws:eks::aws:cluster-access-policy/AmazonEKSSecretReaderPolicy` 访问条目策略与功能的访问条目相关联
2. 将策略范围限定到特定的命名空间，ACK 资源将在其中引用密钥，或者授予整个集群的访问权限

> **Important**
>
> 密钥读取权限的范围限定为您在关联访问条目策略时指定的命名空间。这允许您限制该功能可以访问的密钥。

kro 任意资源权限

kro 需要权限才能创建和管理 ResourceGraphDefinitions 中定义的资源。默认情况下，kro 只能自行监视和管理 RGD。

要授予 kro 创建资源的权限，请执行以下操作：

选项 1：访问条目策略

将预定义的访问条目策略（如 `AmazonEKSAdminPolicy` 或 `AmazonEKSEditPolicy`）与功能的访问条目相关联。

选项 2：Kubernetes RBAC

创建一个向该功能的 Kubernetes 用户授予必要权限的 `ClusterRoleBinding`：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kro-cluster-admin
subjects:
- kind: User
  name: arn:aws:sts::111122223333:assumed-role/my-kro-role/kro
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

> **Note**
>
> kro 的 Kubernetes 用户名遵循以下模式：`arn:aws:sts::ACCOUNT_ID:assumed-role/ROLE_NAME/kro`
> 会话名称 `/kro` 由 EKS kro 功能自动设置。

功能所需的 IAM 权限

kro（Kube Resource Orchestrator）

不需要 IAM 权限。您可以创建没有附加策略的功能角色。kro 只需要 Kubernetes RBAC 权限。

ACK（Amazon Controllers for Kubernetes）

需要权限才能管理 ACK 将创建和管理的 Amazon 资源。您应根据自己的要求将权限范围限定为特定服务、操作和资源。有关配置 ACK 权限的详细信息，包括使用 IAM 角色选择器的生产最佳实践，请参阅the section called “配置权限”。

Argo CD

默认情况下，不需要 IAM 权限。以下功能可能需要可选权限：

- Amazon Secrets Manager：如果在 Secrets Manager 中存储 Git 存储库凭证
- Amazon CodeConnections：如果使用 CodeConnections 进行 Git 存储库身份验证
- Amazon ECR：如果使用在 Amazon ECR 中以 OCI 格式存储的 Helm 图表

安全最佳实践

IAM 最低权限

仅授予您的功能资源您的使用案例所需的权限。这并不意味着在需要时您无法向自己的功能授予广泛的管理权限。在这种情况下，您应适当地管理对这些资源的访问权限。

功能角色：

- ACK：在可能的情况下，根据使用案例和要求，将 IAM 权限限制为仅能访问团队所需的特定 Amazon 服务和资源
- Argo CD：限制对特定的 Git 存储库和 Kubernetes 命名空间的访问权限
- kro：需要信任策略的功能角色，但不需要 IAM 权限（仅使用集群 RBAC）

示例：为特定资源或资源组指定模式，而非 `"Resource": "*"`。

```
"Resource": [
  "arn:aws:s3:::my-app-*",
  "arn:aws:rds:us-west-2:111122223333:db:prod-*"
```

```
]
```

使用 IAM 条件键进一步限制访问权限：

```
"Condition": {
  "StringEquals": {
    "aws:ResourceTag/Environment": "production"
  }
}
```

有关其他 IAM 配置信息，请参阅每项功能的注意事项部分。

Argo CD 密钥的命名空间隔离

托管 Argo CD 功能可以访问其配置的命名空间中的所有 Kubernetes 密钥（默认：`argocd`）。要保持最佳安全状况，请遵循以下命名空间隔离实践：

- 在 Argo CD 命名空间中只保留 Argo CD 相关的密钥
- 避免将不相关的应用程序密钥存储在与 Argo CD 相同的命名空间中
- 使用单独的命名空间来存放 Argo CD 操作不需要的应用程序密钥

这种隔离可确保 Argo CD 的密钥访问权限仅限于 Git 存储库身份验证和其他 Argo CD 特定操作所需的凭证。

Kubernetes RBAC

控制哪些用户和服务账户可以创建和管理功能资源。使用适当的 RBAC 策略在专用命名空间中部署功能资源是最佳实践。

示例：与 ACK 配合使用 RBAC 角色，允许在 `app-team` 命名空间中管理 S3 存储桶资源：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ack-s3-manager
  namespace: app-team
rules:
- apiGroups: ["s3.services.k8s.aws"]
  resources: ["buckets"]
```

```
  verbs: ["get", "list", "create", "update", "delete"]
```

审计日志记录

CloudTrail：所有 EKS 功能 API 操作（创建、更新、删除）都将记录到 Amazon CloudTrail 中。

启用 CloudTrail 日志记录来跟踪：

- 哪些用户创建或修改了功能
- 功能配置发生更改时
- 哪些功能角色在使用中

网络访问和 VPC 端点

私有 Argo CD API 访问权限

您可以通过将一个或多个 VPC 端点与托管的 Argo CD 端点关联，来限制对 Argo CD API 服务器的访问。这样就可以在您的 VPC 内私有连接到 Argo CD 用户界面和 API，而无需通过公共互联网。

> **Note**
>
> 连接到托管的 Argo CD API 端点的 VPC 端点（使用 eks-capabilities.*region*.amazonaws.com）不支持 VPC 端点策略。

部署到私有集群

Argo CD 功能可以将应用程序部署到完全私有的 EKS 集群，无需进行 VPC 对等互连或复杂的网络配置，从而提供显著的运营优势。但是，在设计此架构时，请考虑 Argo CD 将从 Git 存储库（可能是公有存储库）中拉取配置并将其应用于您的私有集群。

请确保：

- 使用私有 Git 存储库处理敏感工作负载
- 实施适当的 Git 仓库访问控制和身份验证
- 先通过拉取请求查看和批准更改，再进行合并
- 考虑使用 Argo CD 的同步窗口来控制部署的时间

- 监控 Argo CD 审计日志中是否有未经授权的配置更改

合规性

EKS 功能是完全托管的，并已获得 Amazon EKS 的合规性认证。

有关更多信息，请参阅按合规性计划划分的范围内的 Amazon 服务。

后续步骤

- the section called “配置权限”：为 ACK 配置 IAM 权限
- the section called “配置权限”：为 kro 配置 Kubernetes RBAC
- the section called “配置权限”：为 Argo CD 配置 Identity Center 集成
- the section called “问题排查功能”：对安全和权限问题进行问题排查

适用于 Amazon EKS 的身份和访问管理

Amazon Identity and Access Management（IAM）是一项 Amazon 服务，可帮助管理员安全地控制对 Amazon 资源的访问。IAM 管理员控制谁可以通过身份验证（登录）和授权（具有权限）使用 Amazon EKS 资源。IAM 是一项可以免费使用的 Amazon 服务。

受众

您使用 Amazon Identity and Access Management（IAM）的方式有所不同，具体取决于您在 Amazon EKS 中完成的工作。

服务用户 – 如果您使用 Amazon EKS 服务来执行任务，则您的管理员会为您提供所需的凭证和权限。随着您使用更多 Amazon EKS 功能来完成工作，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon EKS 中的功能，请参阅 the section called “故障排除”。

服务管理员 – 如果您在公司负责管理 Amazon EKS 资源，您可能对 Amazon EKS 具有完全访问权限。您有责任确定您的服务用户应访问哪些 Amazon EKS 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 Amazon EKS 搭配使用的更多信息，请参阅 the section called “Amazon EKS 和 IAM”。

IAM 管理员 – 如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 Amazon EKS 的访问的详细信息。要查看您可在 IAM 中使用的 Amazon EKS 基于身份的策略示例，请参阅 the section called “基于身份的策略”。

使用身份进行身份验证

身份验证是您使用身份凭证登录 Amazon 的方法。您必须作为 Amazon 账户根用户、IAM 用户或通过代入 IAM 角色进行身份验证（登录到 Amazon）。

您可以使用通过身份源提供的凭证以联合身份登录到 Amazon。AmazonIAM Identity Center（IAM Identity Center）用户、您公司的单点登录身份验证以及您的 Google 或 Facebook 凭证都是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当您使用联合身份验证访问 Amazon 时，您就是在间接代入角色。

根据您的用户类型，您可以登录 Amazon Web Services 管理控制台 或 Amazon 访问门户。有关登录到 Amazon 的更多信息，请参阅《Amazon Sign-In 用户指南》中的如何登录到您的 Amazon 账户。

如果您以编程方式访问 Amazon，则 Amazon 将提供软件开发工具包（SDK）和命令行界面（CLI），以便使用您的凭证以加密方式签署您的请求。如果您不使用 Amazon 工具，则必须自行对请求签名。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的签署 Amazon API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如，Amazon 建议您使用多重身份验证（MFA）来提高账户的安全性。要了解更多信息，请参阅《Amazon IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的在 Amazon 中使用多重身份验证（MFA）。

Amazon 账户根用户

在创建 Amazon 账户时，您首先需要使用一个对账户中所有 Amazon 服务和资源拥有完全访问权限的登录身份。此身份称为 Amazon 账户根用户，使用您创建账户时所用的电子邮件地址和密码登录，即可获得该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

IAM 用户和群组

IAM 用户是 Amazon 账户内对某个人员或应用程序具有特定权限的一个身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用案例需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用案例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

IAM 角色是 Amazon 账户中具有特定权限的实体。它类似于 IAM 用户，但与特定人员不关联。您可以通过切换角色，在 Amazon Web Services 管理控制台 中暂时代入 IAM 角色。可以通过调用 Amazon CLI 或 Amazon API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供者创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关更多信息，请参阅《Amazon IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限：IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户访问：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些 Amazon 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南中的 IAM 中的跨账户资源访问。
- 跨服务访问：某些 Amazon 服务使用其它 Amazon 服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service（Amazon S3）中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话（FAS）：当您使用 IAM 用户或角色在 Amazon 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 Amazon 服务的权限，结合请求的 Amazon 服务，向下游服务发出请求。只有在服务收到需要与其它 Amazon 服务或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

- 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅 IAM 用户指南中的创建向 Amazon 服务委派权限的角色。
- 服务相关角色 – 服务相关角色是与 Amazon 服务关联的一种服务角色。服务可以代入代表您执行操作的角色。服务相关角色显示在您的 Amazon 账户中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务关联角色的权限。

- 在 Amazon EC2 上运行的应用程序：您可以使用 IAM 角色管理在 EC2 实例上运行并发出 Amazon CLI 或 Amazon API 请求的应用程序的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 Amazon 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您将创建策略并将其附加到 Amazon 身份或资源，以控制 Amazon 中的访问。策略是 Amazon 中的对象；在与身份或资源相关联时，策略定义它们的权限。在主体（用户、根用户或角色会话）发出请求时，Amazon 将评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在 Amazon 中存储为 JSON 文档。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 Amazon JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。具有该策略的用户可以从 Amazon Web Services 管理控制台、Amazon CLI 或 Amazon API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管式策略是可以附加到 Amazon 账户中的多个用户、组和角色的独立策略。托管式策略包括 Amazon 托管式策略和客户托管式策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅 IAM 用户指南中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 Amazon 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用来自 IAM 的 Amazon 托管式策略。

访问控制列表（ACL）

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，但它们不使用 JSON 策略文档格式。

Amazon S3、Amazon WAF 和 Amazon VPC 是支持 ACL 的服务示例。要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

Amazon 支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅IAM 用户指南中的 IAM 实体的权限边界。

- 服务控制策略 (SCP) – SCP 是指定 Amazon Organizations 中的组织或组织单元 (OU) 的最大权限的 JSON 策略。AmazonOrganizations 是一个服务，用于对您的企业拥有的多个 Amazon 账户进行分组和集中管理。如果在组织内启用了所有特征，则可对任意或全部账户应用服务控制策略（SCP）。SCP 限制成员账户中实体的权限，包括每个 Amazon 账户根用户。有关组织和 SCP 的更多信息，请参阅《Amazon Organizations 用户指南》中的 Service control policies。
- 会话策略：会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解 Amazon 如何确定在涉及多种策略类型时是否允许请求，请参阅 IAM 用户指南中的策略评测逻辑。

Amazon EKS 如何与 IAM 配合使用

在使用 IAM 管理对 Amazon EKS 的访问权限之前，您应该了解哪些 IAM 功能可用于 Amazon EKS。要大致了解 Amazon EKS 和其他Amazon服务如何与 IAM 一起使用，请参阅 IAM 用户指南中的与 IAM 一起使用的Amazon服务。

主题

- Amazon EKS 基于身份的策略
- Amazon EKS 基于资源的策略
- 基于 Amazon EKS 标签的授权
- Amazon EKS IAM 角色

Amazon EKS 基于身份的策略

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。Amazon EKS 支持特定的操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素参考。

操作

管理员可以使用 Amazon JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 Action 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 Amazon API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

Amazon EKS 中的策略操作在操作前面使用以下前缀：eks:。例如，要授予某人获取关于 Amazon EKS 集群的描述性信息的权限，您应将 DescribeCluster 操作纳入其策略中。策略语句必须包含 Action 或 NotAction 元素。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": ["eks:action1", "eks:action2"]
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 Describe 开头的所有操作，包括以下操作：

```
"Action": "eks:Describe*"
```

要查看 Amazon EKS 操作的列表，请参阅《服务授权参考》中的 Amazon Elastic Kubernetes Service 定义的操作。

资源

管理员可以使用 Amazon JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

Amazon EKS 集群资源具有以下 ARN。

```
arn:aws-cn:eks:region-code:account-id:cluster/cluster-name
```

有关 ARN 格式的更多信息，请参阅 Amazon 资源名称 (ARN) 和 Amazon 服务命名空间。

例如，要在语句中指定名为 *my-cluster* 的集群，请使用以下 ARN：

```
"Resource": "arn:aws-cn:eks:region-code:111122223333:cluster/my-cluster"
```

要指定属于特定账户和 Amazon 区域的所有集群，请使用通配符（*）：

```
"Resource": "arn:aws-cn:eks:region-code:111122223333:cluster/*"
```

无法对特定资源执行某些 Amazon EKS 操作，例如用于创建资源的操作。在这些情况下，您必须使用通配符（*)。

```
"Resource": "*"
```

要查看 Amazon EKS 资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 Amazon Elastic Kubernetes Service 定义的资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 Amazon Elastic Kubernetes Service 定义的操作。

条件键

Amazon EKS 定义了自己的一组条件键，还支持使用一些全局条件键。要查看所有 Amazon 全局条件键，请参阅《IAM 用户指南》中的 Amazon 全局条件上下文键。

在将 OpenID Connect 提供商与集群关联时，您可以设置条件键。有关更多信息，请参阅 the section called “示例 IAM policy”。

所有 Amazon EC2 操作都支持 `aws:RequestedRegion` 和 `ec2:Region` 条件键。有关更多信息，请参阅示例：限制对特定 Amazon 区域的访问。

有关 Amazon EKS 条件键的列表，请参阅《Service Authorization Reference》中的 Conditions defined by Amazon Elastic Kubernetes Service。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon Elastic Kubernetes Service 定义的操作。

示例

要查看 Amazon EKS 基于身份的策略的示例，请参阅 the section called “基于身份的策略”。

创建 Amazon EKS 集群时，将为创建集群的 IAM 主体自动授予 Amazon EKS 控制面板中基于集群角色的访问控制（RBAC）配置中的 `system:masters` 权限。该主体不会显示在任何可见配置中，因此请确保跟踪最初创建集群的主体。要授予其他 IAM 主体与集群进行交互的能力，请编辑 Kubernetes 中的 `aws-auth ConfigMap`，创建 Kubernetes `rolebinding` 或 `clusterrolebinding`，名为 `aws-auth ConfigMap` 中指定的 `group`。

有关使用 ConfigMap 的更多信息，请参阅 the section called “Kubernetes API 访问权限”。

Amazon EKS 基于资源的策略

Amazon EKS 不支持基于资源的策略。

基于 Amazon EKS 标签的授权

您可以将标签附加到 Amazon EKS 资源，或者在请求中将标签传递给 Amazon EKS。要基于标签控制访问，您需要使用 `aws:ResourceTag/key-name` 、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关标记 Amazon EKS 资源的更多信息，请参阅 the section called “标记您的资源”。想要了解您可以将条件键中的标签用于哪些操作，请参阅服务授权参考中的 Amazon EKS 定义的操作。

Amazon EKS IAM 角色

IAM 角色是 Amazon 账户中具有特定权限的实体。

将临时凭证用于 Amazon EKS

可以使用临时凭证进行联合身份验证登录，分派 IAM 角色或分派跨账户角色。您可以通过调用Amazon STS API 操作（AssumeRole 或 GetFederationToken）获得临时安全凭证。

Amazon EKS 支持使用临时凭证。

服务相关角色

服务相关角色允许 Amazon 服务访问其他服务中的资源以代表您完成操作。服务相关角色显示在 IAM 账户中，并归该服务所有。管理员可以查看但不能编辑服务相关角色的权限。

Amazon EKS 支持服务相关角色。有关创建或管理 Amazon EKS 服务相关角色的详细信息，请参阅 the section called “服务相关角色”。

服务角色

此功能允许服务代表您担任服务角色。此角色允许服务访问其他服务中的资源以代表您完成操作。服务角色显示在 IAM 账户中，并归该账户所有。这意味着，IAM 管理员可以更改该角色的权限。但是，这样做可能会中断服务的功能。

Amazon EKS 支持服务角色。有关更多信息，请参阅the section called “集群 IAM 角色”和the section called “节点 IAM 角色”。

在 Amazon EKS 中选择 IAM 角色

当您在 Amazon EKS 中创建集群资源时，您必须选择一个角色以允许 Amazon EKS 代表您访问一些其他Amazon资源。如果您之前创建了一个服务角色，Amazon EKS 会为您提供一个角色列表供您选择。请务必选择一个附带 Amazon EKS 托管策略的角色。有关更多信息，请参阅the section called “检查现有集群角色”和the section called “检查现有节点角色”。

Amazon EKS 基于身份的策略示例

默认情况下，IAM 用户和角色没有创建或修改 Amazon EKS 资源的权限。它们也无法使用Amazon Web Services 管理控制台、Amazon CLI 或者 Amazon API 执行任务。IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的在 JSON 选项卡上创建策略。

创建 Amazon EKS 集群时，将为创建集群的 IAM 主体自动授予 Amazon EKS 控制面板中基于集群角色的访问控制（RBAC）配置中的 `system:masters` 权限。该主体不会显示在任何可见配置中，因此请确保跟踪最初创建集群的主体。要授予其他 IAM 主体与集群进行交互的能力，请编辑 Kubernetes 中的 `aws-auth ConfigMap`，创建 Kubernetes `rolebinding` 或 `clusterrolebinding`，名为 `aws-auth ConfigMap` 中指定的 `group`。

有关使用 ConfigMap 的更多信息，请参阅 the section called “Kubernetes API 访问权限”。

主题

- 策略最佳实践
- 使用 Amazon EKS 控制台
- 允许 IAM 用户查看他们自己的权限
- 在 Amazon 云上创建 Kubernetes 集群
- 在 Outpost 上创建本地 Kubernetes 集群
- 更新 Kubernetes 集群
- 列出或描述所有集群

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon EKS 资源。这些操作可能会使 Amazon 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- Amazon 托管策略及转向最低权限许可入门 – 要开始向用户和工作负载授予权限，请使用 Amazon 托管式策略来为许多常见使用案例授予权限。您可以在 Amazon 账户中找到这些策略。我们建议通过定义特定于您的使用案例的 Amazon 客户管理型策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 Amazon 托管策略或工作职能的 Amazon 托管策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果通过特定 Amazon 服务（例如 Amazon CloudFormation）使用服务操作，您还可以使用条件来授予对服务操作的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性：IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证（MFA）– 如果您所处的场景要求您的 Amazon 账户中有 IAM 用户或根用户，请启用 MFA 来提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

使用 Amazon EKS 控制台

要访问 Amazon EKS 控制台，IAM 主体必须具有一组最低的权限。这些权限允许主体列出和查看有关您的 Amazon 账户中的 Amazon EKS 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的主体，控制台将无法按预期正常运行。

为确保 IAM 主体仍可以使用 Amazon EKS 控制台，请用唯一名称创建一个策略，如 `AmazonEKSAdminPolicy`。将策略附加到主体。有关更多信息，请参阅《IAM 用户指南》中的添加和移除 IAM 身份权限。

> **Important**
>
> 以下示例策略将允许主体查看控制台中配置选项卡上的信息。要在 Amazon Web Services 管理控制台中的概述和资源选项卡上查看信息，该主体还需要 Kubernetes 权限。有关更多信息，请参阅 the section called “所需的权限”。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

对于仅调用 Amazon CLI 或 Amazon API 的主体，您不需要允许最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

允许 IAM 用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包含通过控制台或者使用 Amazon CLI 或 Amazon API 以编程方式完成此操作所需的权限。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
```

```
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

在 Amazon 云上创建 Kubernetes 集群

此示例策略包含在 *us-west-2* Amazon 区域中创建名为 *my-cluster* 的 Amazon EKS 集群所需的最低权限。您可以将 Amazon 区域替换为要在其中创建集群的 Amazon 区域。如果 Amazon Web Services 管理控制台 中显示警告，提示您的策略中的操作不支持资源级权限，因而要求您选择 **All resources** ，您可以放心地忽略该警告。如果账户已经有 *AWSServiceRoleForAmazonEKS* 角色，您可以从策略中删除 `iam:CreateServiceLinkedRole` 操作。如果在账户中创建过 Amazon EKS 集群，则此角色已经存在，除非您已将其删除。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:CreateCluster",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
            "Resource": "arn:aws:iam::111122223333:role/aws-service-role/
eks.amazonaws.com/AWSServiceRoleForAmazonEKS",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iam:AWSServiceName": "eks"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
        }
    ]
}
```

在 Outpost 上创建本地 Kubernetes 集群

此示例策略包括在 *us-west-2* Amazon 区域中创建名为 *my-cluster* 的 Amazon EKS 本地集群所需的最低权限。您可以将 Amazon 区域替换为要在其中创建集群的 Amazon 区域。如果 Amazon Web Services 管理控制台 中显示警告，提示您的策略中的操作不支持资源级权限，因而要求您选择 **All resources** ，您可以放心地忽略该警告。如果您的账户已经有 AWSServiceRoleForAmazonEKSLocalOutpost 角色，您可以从策略中删除 iam:CreateServiceLinkedRole 操作。如果您曾经在您的账户中的 Outpost 上创建过 Amazon EKS 本地集群，则此角色已存在，除非您将其删除。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:CreateCluster",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        },
        {
            "Action": [
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "iam:GetRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
```

```
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::111122223333:role/aws-service-role/outposts.eks-
local.amazonaws.com/AWSServiceRoleForAmazonEKSLocalOutpost"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
        },
        {
            "Action": [
                "iam:CreateInstanceProfile",
                "iam:TagInstanceProfile",
                "iam:AddRoleToInstanceProfile",
                "iam:GetInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile"
            ],
            "Resource": "arn:aws:iam::*:instance-profile/eks-local-*",
            "Effect": "Allow"
        }
    ]
}
```

更新 Kubernetes 集群

此示例策略包含在 us-west-2 Amazon 区域中更新名为 *my-cluster* 的集群所需的最低权限。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:UpdateClusterVersion",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        }
    ]
```

```
}
```

列出或描述所有集群

此示例策略包含列出和描述账户中所有集群所需的最低权限。IAM 主体必须能够列出和描述集群，才能使用 `update-kubeconfig` Amazon CLI 命令。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

对 Amazon EKS 使用服务相关角色

Amazon Elastic Kubernetes Service 使用 Amazon Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。服务相关角色是由 Amazon EKS 预定义的，并包含服务代表您调用其它 Amazon 服务所需的所有权限。

主题

- 使用 Amazon EKS 集群的角色
- 使用 Amazon EKS 节点组的角色
- 使用 Amazon EKS Fargate 配置文件的角色
- 使用角色将 Kubernetes 集群连接到 Amazon EKS
- 使用 Outpost 上的 Amazon EKS 本地集群的角色
- 使用 Amazon EKS 控制面板的角色

使用 Amazon EKS 集群的角色

Amazon Elastic Kubernetes Service 使用 Amazon Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。服务相关角色是由 Amazon EKS 预定义的，并包含服务代表您调用其它 Amazon 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon EKS，因为您不必手动添加必要的权限。Amazon EKS 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon EKS 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 Amazon EKS 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅与 IAM 配合使用的 Amazon 服务，并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

Amazon EKS 的服务相关角色权限

Amazon EKS 使用名为 `AWSServiceRoleForAmazonEKS` 的服务相关角色。该角色允许 Amazon EKS 管理您账户中的集群。附加的策略允许角色管理以下资源：网络接口、安全组、日志和 VPC。

> **Note**
>
> `AWSServiceRoleForAmazonEKS` 服务相关角色不同于创建集群所需的角色。有关更多信息，请参阅 the section called “集群 IAM 角色”。

`AWSServiceRoleForAmazonEKS` 服务相关角色信任以下服务代入该角色：

- `eks.amazonaws.com`

角色权限策略允许 Amazon EKS 对指定资源完成以下操作：

- AmazonEKSServiceRolePolicy

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon EKS 创建服务相关角色

您无需手动创建服务相关角色。当您在 Amazon Web Services 管理控制台、Amazon CLI 或 Amazon API 中创建集群时，Amazon EKS 会为您创建服务相关角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建集群时，Amazon EKS 将再次为您创建服务相关角色。

为 Amazon EKS 编辑服务相关角色

Amazon EKS 不允许您编辑 `AWSServiceRoleForAmazonEKS` 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除适用于 Amazon EKS 的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能手动删除它。

清除服务相关角色

必须先删除服务相关角色使用的所有资源，然后才能使用 IAM 删除该角色。

> **Note**
>
> 如果当您试图删除资源时 Amazon EKS 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 如果您的集群具有任何节点组或 Fargate 配置文件，则必须先将其删除，然后才能删除该集群。有关更多信息，请参阅 the section called “删除” 和 the section called “删除配置文件”。
4. 在 Clusters（集群）页面上，选择要删除的集群，然后选择 Delete（删除）。
5. 在删除确认窗口中键入集群的名称，然后选择 Delete（删除）。
6. 对您账户中的任何其他集群重复此过程。等待所有删除操作完成。

手动删除服务相关角色

使用 IAM 控制台、Amazon CLI 或 Amazon API 删除 AWSServiceRoleForAmazonEKS 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Amazon EKS 服务相关角色支持的区域

Amazon EKS 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon EKS 端点和配额。

使用 Amazon EKS 节点组的角色

Amazon API Gateway 使用 Amazon Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。服务相关角色是由 Amazon EKS 预定义的，并包含服务代表您调用其它 Amazon 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon EKS，因为您不必手动添加必要的权限。Amazon EKS 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon EKS 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 Amazon EKS 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅与 IAM 配合使用的 Amazon 服务，并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

Amazon EKS 的服务相关角色权限

Amazon EKS 使用名为 AWSServiceRoleForAmazonEKSNodegroup 的服务相关角色。该角色允许 Amazon EKS 管理您账户中的节点组。附加的 AWSServiceRoleForAmazonEKSNodegroup 策略允许角色管理以下资源：自动扩缩组、安全组、启动模板和 IAM 实例配置文件。有关更多信息，请参阅 the section called “Amazon托管策略：AWSServiceRoleForAmazonEKSNodegroup”。

AWSServiceRoleForAmazonEKSNodegroup 服务相关角色信任以下服务代入该角色：

- eks-nodegroup.amazonaws.com

角色权限策略允许 Amazon EKS 对指定资源完成以下操作：

- AWSServiceRoleForAmazonEKSNodegroup

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon EKS 创建服务相关角色

您无需手动创建服务相关角色。当您在 Amazon Web Services 管理控制台、Amazon CLI 或 Amazon API 中创建节点组时，Amazon EKS 会为您创建服务相关角色。

> **Important**
>
> 如果您在其他使用此角色支持的功能的服务中完成某个操作，此服务相关角色可以出现在您的账户中。如果在 2017 年 1 月 1 日（从此时开始支持服务相关角色）之前已使用 Amazon EKS 服务，则 Amazon EKS 已在您的账户中创建了 AWSServiceRoleForAmazonEKSNodegroup 角色。要了解更多信息，请参阅我的 IAM 账户中出现新角色。

在 Amazon EKS（Amazon API）中创建服务相关角色

您无需手动创建服务相关角色。当您在 Amazon Web Services 管理控制台、Amazon CLI 或 Amazon API 中创建托管节点组时，Amazon EKS 会为您创建服务相关角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建另一个托管节点组时，Amazon EKS 再次为您创建服务相关角色。

为 Amazon EKS 编辑服务相关角色

Amazon EKS 不允许您编辑 `AWSServiceRoleForAmazonEKSNodegroup` 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除适用于 Amazon EKS 的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能手动删除它。

清除服务相关角色

必须先删除服务相关角色使用的所有资源，然后才能使用 IAM 删除该角色。

> **Note**
>
> 如果当您试图删除资源时 Amazon EKS 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 选择 Compute（计算）选项卡。
4. 在 Node groups（节点组）部分中，选择要删除的节点组。
5. 在删除确认窗口中键入节点组的名称，然后选择 Delete（删除）。
6. 对集群中的任何其他节点组重复此过程。等待所有删除操作完成。

手动删除服务相关角色

使用 IAM 控制台、Amazon CLI 或 Amazon API 删除 `AWSServiceRoleForAmazonEKSNodegroup` 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Amazon EKS 服务相关角色支持的区域

Amazon EKS 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon EKS 端点和配额。

使用 Amazon EKS Fargate 配置文件的角色

Amazon API Gateway 使用 Amazon Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。服务相关角色是由 Amazon EKS 预定义的，并包含服务代表您调用其它 Amazon 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon EKS，因为您不必手动添加必要的权限。Amazon EKS 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon EKS 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 Amazon EKS 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅与 IAM 配合使用的 Amazon 服务，并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

Amazon EKS 的服务相关角色权限

Amazon EKS 使用名为 AWSServiceRoleForAmazonEKSForFargate 的服务相关角色。该角色允许 Amazon EKS Fargate 配置 Fargate 容器组（pod）所需的 VPC 联网。附加的策略允许角色创建和删除弹性网络接口，并描述弹性网络接口和资源。

AWSServiceRoleForAmazonEKSForFargate 服务相关角色信任以下服务代入该角色：

- eks-fargate.amazonaws.com

角色权限策略允许 Amazon EKS 对指定资源完成以下操作：

- AmazonEKSForFargateServiceRolePolicy

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon EKS 创建服务相关角色

您无需手动创建服务相关角色。当您在 Amazon Web Services 管理控制台、Amazon CLI 或 Amazon API 中创建 Fargate 配置文件时，Amazon EKS 将为您创建服务相关角色。

> **Important**
>
> 如果您在其他使用此角色支持的功能的服务中完成某个操作，此服务相关角色可以出现在您的账户中。如果在 2019 年 12 月 13 日（从此时开始支持服务相关角色）之前已使用 Amazon EKS 服务，则 Amazon EKS 已在您的账户中创建了 AWSServiceRoleForAmazonEKSForFargate 角色。要了解更多信息，请参阅我的 IAM 账户中出现新角色。

在 Amazon EKS（Amazon API）中创建服务相关角色

您无需手动创建服务相关角色。当您在 Amazon Web Services 管理控制台、Amazon CLI 或 Amazon API 中创建 Fargate 配置文件时，Amazon EKS 将为您创建服务相关角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建另一个托管节点组时，Amazon EKS 再次为您创建服务相关角色。

为 Amazon EKS 编辑服务相关角色

Amazon EKS 不允许您编辑 AWSServiceRoleForAmazonEKSForFargate 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除适用于 Amazon EKS 的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能手动删除它。

清除服务相关角色

必须先删除服务相关角色使用的所有资源，然后才能使用 IAM 删除该角色。

> **Note**
>
> 如果当您试图删除资源时 Amazon EKS 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 在 Clusters (集群) 页面上，选择您的集群。
4. 选择 Compute（计算）选项卡。
5. 如果 Fargate profiles（Fargate 配置文件）部分中有任何 Fargate 配置文件，则分别选择每个配置文件，然后选择 Delete（删除）。
6. 在删除确认窗口中键入配置文件的名称，然后选择 Delete（删除）。
7. 对集群中以及您账户中任何其他集群的任何其它 Fargate 配置文件重复此过程。

手动删除服务相关角色

使用 IAM 控制台、Amazon CLI 或 Amazon API 删除 AWSServiceRoleForAmazonEKSForFargate 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Amazon EKS 服务相关角色支持的区域

Amazon EKS 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon EKS 端点和配额。

使用角色将 Kubernetes 集群连接到 Amazon EKS

Amazon API Gateway 使用 Amazon Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。服务相关角色是由 Amazon EKS 预定义的，并包含服务代表您调用其它 Amazon 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon EKS，因为您不必手动添加必要的权限。Amazon EKS 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon EKS 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务关联角色。这将保护您的 Amazon EKS 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅与 IAM 配合使用的 Amazon 服务，并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务关联角色文档。

Amazon EKS 的服务相关角色权限

Amazon EKS 使用名为 `AWSServiceRoleForAmazonEKSConnector` 的服务相关角色。该角色允许 Amazon EKS 连接 Kubernetes 集群。附加的策略允许角色管理必要的资源，以便连接到注册的 Kubernetes 集群。

`AWSServiceRoleForAmazonEKSConnector` 服务相关角色信任以下服务代入该角色：

- `eks-connector.amazonaws.com`

角色权限策略允许 Amazon EKS 对指定资源完成以下操作：

- AmazonEKSConnectorServiceRolePolicy

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

此角色使用 SSM（Systems Manager）权限来建立安全连接和管理连接的 Kubernetes 集群。

为 Amazon EKS 创建服务相关角色

无需手动创建服务相关角色来连接集群。当您在 Amazon Web Services 管理控制台、Amazon CLI、`eksctl` 或 Amazon API 中连接集群时，Amazon EKS 会为您创建服务相关角色。

如果您删除该服务关联角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。连接集群时，Amazon EKS 将再次为您创建服务相关角色。

为 Amazon EKS 编辑服务相关角色

Amazon EKS 不允许您编辑 AWSServiceRoleForAmazonEKSConnector 服务相关角色。创建服务关联角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务关联角色。

删除适用于 Amazon EKS 的服务相关角色

如果不再需要使用某个需要服务关联角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能手动删除它。

清除服务相关角色

必须先删除服务相关角色使用的所有资源，然后才能使用 IAM 删除该角色。

> **Note**
>
> 如果当您试图删除资源时 Amazon EKS 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

1. 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 在 Clusters (集群) 页面上，选择您的集群。
4. 选择 Deregister（取消注册）选项卡，然后选择 Ok（确定）选项卡。

手动删除服务相关角色

使用 IAM 控制台、Amazon CLI 或 Amazon API 删除 AWSServiceRoleForAmazonEKSConnector 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

使用 Outpost 上的 Amazon EKS 本地集群的角色

Amazon Elastic Kubernetes Service 使用 Amazon Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。服务相关角色是由 Amazon EKS 预定义的，并包含服务代表您调用其它 Amazon 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon EKS，因为您不必手动添加必要的权限。Amazon EKS 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon EKS 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 Amazon EKS 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅与 IAM 配合使用的 Amazon 服务，并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

Amazon EKS 的服务相关角色权限

Amazon EKS 使用名为 `AWSServiceRoleForAmazonEKSLocalOutpost` 的服务相关角色。该角色允许 Amazon EKS 管理您账户中的本地集群。附加的策略允许角色管理以下资源：网络接口、安全组、日志和 Amazon EC2 实例。

> **Note**
>
> `AWSServiceRoleForAmazonEKSLocalOutpost` 服务相关角色不同于创建集群所需的角色。有关更多信息，请参阅 the section called “集群 IAM 角色”。

`AWSServiceRoleForAmazonEKSLocalOutpost` 服务相关角色信任以下服务代入该角色：

- `outposts.eks-local.amazonaws.com`

角色权限策略允许 Amazon EKS 对指定资源完成以下操作：

- AmazonEKSServiceRolePolicy

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon EKS 创建服务相关角色

您无需手动创建服务相关角色。当您在 Amazon Web Services 管理控制台、Amazon CLI 或 Amazon API 中创建集群时，Amazon EKS 会为您创建服务相关角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建集群时，Amazon EKS 将再次为您创建服务相关角色。

为 Amazon EKS 编辑服务相关角色

Amazon EKS 不允许您编辑 AWSServiceRoleForAmazonEKSLocalOutpost 服务相关角色。在创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。不过，您可以使用 IAM 编辑角色的说明。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除适用于 Amazon EKS 的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能手动删除它。

清除服务相关角色

必须先删除服务相关角色使用的所有资源，然后才能使用 IAM 删除该角色。

> **Note**
>
> 如果当您试图删除资源时 Amazon EKS 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

1. 打开 Amazon EKS 控制台。
2. 请在左侧导航窗格中，选择 Amazon EKS Clusters（集群）。
3. 如果您的集群具有任何节点组或 Fargate 配置文件，则必须先将其删除，然后才能删除该集群。有关更多信息，请参阅 the section called “删除” 和 the section called “删除配置文件”。
4. 在 Clusters（集群）页面上，选择要删除的集群，然后选择 Delete（删除）。
5. 在删除确认窗口中键入集群的名称，然后选择 Delete（删除）。
6. 对您账户中的任何其他集群重复此过程。等待所有删除操作完成。

手动删除服务相关角色

使用 IAM 控制台、Amazon CLI 或 Amazon API 删除 AWSServiceRoleForAmazonEKSLocalOutpost 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Amazon EKS 服务相关角色支持的区域

Amazon EKS 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon EKS 端点和配额。

使用 Amazon EKS 控制面板的角色

Amazon EKS 控制面板通过此服务相关角色，来聚合有关来自多个区域和账户的集群资源的信息。控制面板与 Amazon Organizations 集成，可安全读取有关组织中多个账户的信息。

要了解有关 Amazon EKS 控制面板的更多信息，请参阅 the section called “Amazon EKS 控制面板”。

背景

Amazon API Gateway 使用 Amazon Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EKS 直接相关。服务相关角色是由 Amazon EKS 预定义的，并包含服务代表您调用其它 Amazon 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon EKS，因为您不必手动添加必要的权限。Amazon EKS 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon EKS 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 Amazon EKS 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅与 IAM 配合使用的 Amazon 服务，并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

Amazon EKS 的服务相关角色权限

Amazon EKS 使用名为 `AWSServiceRoleForAmazonEKSDashboard` 的服务相关角色。该角色允许 Amazon EKS 生成和显示 EKS 控制面板，包括有关集群资源的聚合信息。附加的 `AmazonEKSDashboardServiceRolePolicy` 策略允许角色管理以下资源：自动扩缩组、安全组、启动模板和 IAM 实例配置文件。有关更多信息，请参阅 the section called “Amazon 托管策略：AmazonEKSDashboardServiceRolePolicy”。

`AWSServiceRoleForAmazonEKSDashboard` 服务相关角色信任以下服务代入该角色：

- `dashboard.eks.amazonaws.com`

要查看 JSON 策略文档的最新版本，请参阅《Amazon Managed Policy Reference Guide》中的 AmazonEKSDashboardServiceRolePolicy。

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon EKS 创建服务相关角色

您无需手动创建服务相关角色。当您在 Amazon 控制台中启用控制面板时，Amazon EKS 会为您创建服务相关角色。

有关启用 EKS 控制面板的更多信息，请参阅 the section called “配置组织”。

> **Important**
>
> 如果您在其他使用此角色支持的功能的服务中完成某个操作，此服务相关角色可以出现在您的账户中。

为 Amazon EKS 编辑服务相关角色

Amazon EKS 不允许您编辑 `AWSServiceRoleForAmazonEKSDashboard` 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除适用于 Amazon EKS 的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能手动删除它。

清除服务相关角色

必须先删除服务相关角色使用的所有资源，然后才能使用 IAM 删除该角色。

> **Note**
>
> 如果当您试图删除资源时 Amazon EKS 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

有关禁用 EKS 控制面板的更多信息，请参阅 the section called “配置组织”。

手动删除服务相关角色

使用 IAM 控制台、Amazon CLI 或 Amazon API 删除 `AWSServiceRoleForAmazonEKSDashboard` 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Amazon EKS 服务相关角色支持的区域

Amazon EKS 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon EKS 端点和配额。

Amazon EKS 容器组（pod）执行 IAM 角色

要在 Amazon Fargate 基础设施上运行容器组（pod），需要有 Amazon EKS 容器组（pod）执行角色。

当集群在 Amazon Fargate 基础设施上创建容器组（pod）时，在 Fargate 基础设施上运行的组件必须代表您调用 Amazon API。这是为了他们可以执行诸如从 Amazon ECR 中拉取容器镜像或将日志路由到其他 Amazon 服务的操作。Amazon EKS 容器组（pod）执行角色提供执行此操作的 IAM 权限。

创建 Fargate 配置文件时，必须使用配置文件为在 Fargate 基础设施上运行的 Amazon EKS 组件指定容器组（pod）执行角色。此角色将被添加到集群的 Kubernetes 基于角色的访问控制（RBAC）以进行授权。这允许在 Fargate 基础设施上运行的 `kubelet` 注册到您的 Amazon EKS 集群，以便它可以作为节点显示在您的集群中。

> **Note**
>
> Fargate 配置文件的 IAM 角色必须与 Amazon EC2 节点组的 IAM 角色不同。

> **Important**
>
> 在 Fargate 容器组（pod）中运行的容器不能承担与容器组（pod）执行角色相关联的 IAM 权限。要授予 Fargate 容器组（pod）中的容器访问其他 Amazon 服务的权限，您必须使用服务账户的 IAM 角色。

在创建 Fargate 配置文件之前，必须使用 AmazonEKSFargatePodExecutionRolePolicy创建 IAM 角色。

检查是否有配置正确的现有容器组（pod）执行角色

您可以按照以下程序检查并查看账户是否已有正确配置的 Amazon EKS 容器组（pod）执行角色。为了避免混淆代理安全问题，该角色必须根据 `SourceArn` 限制访问权限。您可以根据需要修改执行角色，以包括对其他集群上的 Fargate 配置文件的支持。

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 角色。
3. 在 Roles（角色）页面上，搜索 AmazonEKSFargatePodExecutionRole 的角色列表。如果该角色不存在，请参阅 the section called “创建 Amazon EKS 容器组（pod）执行角色” 以创建该角色。如果该角色存在，请选择该角色。
4. 在 AmazonEKSFargatePodExecutionRole 页面上，请执行以下操作：
   a. 选择权限。
   b. 确保将 AmazonEKSFargatePodExecutionRolePolicy Amazon 托管策略附加到该角色。
   c. 选择信任关系。
   d. 选择 Edit trust policy（编辑信任策略）。
5. 在 Edit trust policy（编辑信任策略）页面中，验证信任关系是否包含以下策略，并且在集群上是否有 Fargate 配置文件的行。如果是这样，请选择 Cancel（取消）。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:us-east-1:111122223333:fargateprofile/my-
cluster/*"
         }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

如果策略匹配但没有指定集群上 Fargate 配置文件的行，则可以在 `ArnLike` 对象顶部添加以下行。将 *`region-code`* 替换为您在其中创建该集群的 Amazon 区域，将 *`111122223333`* 替换为您的账户 ID，并将 *`my-cluster`* 替换为您的集群的名称。

```
"aws:SourceArn": "arn:aws-cn:eks:region-code:111122223333:fargateprofile/my-cluster/
*",
```

如果策略不匹配，请将之前的完整策略复制到表单中，然后选择 Update policy（更新策略）。将 *region-code* 替换为您的集群所在的 Amazon 区域。如果您希望在账户中的所有 Amazon 区域使用相同的角色，请将 *region-code* 替换为 *。将 *111122223333* 替换为您的账户 ID，并将 *my-cluster* 替换为您的集群名称。如果您希望在账户中对所有集群使用相同角色，请将 *my-cluster* 替换为 *。

创建 Amazon EKS 容器组（pod）执行角色

如果尚无用于集群的 Amazon EKS 容器组（pod）执行角色，您可以使用 Amazon Web Services 管理控制台 或 Amazon CLI 来创建该角色。

Amazon Web Services 管理控制台

a. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
b. 在左侧导航窗格中，选择 Roles（角色）。
c. 在 Roles（角色）页面上，选择 Create role（创建角色）。
d. 在 Select trusted entity（选择受信任的实体）页面上，请执行以下操作：
   i. 在可信实体类型部分中，选择 Amazon 服务。
   ii. 从其它 Amazon 服务的使用案例下拉列表中，选择 EKS。
   iii. 选择 EKS – Fargate 容器组（pod）。
   iv. 选择下一步。
e. 在 Add permissions（添加权限）页面上，选择 Next（下一步）。
f. 在 Name, review, and create（命名、查看和创建）页面中，请执行以下操作：
   i. 对于 Role name (角色名称)，为角色输入唯一名称，例如 `AmazonEKSFargatePodExecutionRole`。
   ii. 在添加标签（可选）下，将标签作为键值对附加，以将元数据添加到角色。有关在 IAM 中使用标签的更多信息，请参阅《IAM 用户指南》 中的标记 IAM 资源。
   iii. 选择创建角色。
g. 在角色页面上，搜索 AmazonEKSFargatePodExecutionRole 的角色列表。选择角色。
h. 在 AmazonEKSFargatePodExecutionRole 页面上，请执行以下操作：
   i. 选择信任关系。
   ii. 选择 Edit trust policy（编辑信任策略）。
i. 在 Edit trust policy（编辑信任策略）页面上，执行以下操作：

i. 将以下内容复制并粘贴到 Edit trust policy（编辑信任策略）表单中。将 *region-code* 替换为您的集群所在的 Amazon 区域。如果您希望在账户中的所有 Amazon 区域使用相同的角色，请将 *region-code* 替换为 *。将 *111122223333* 替换为您的账户 ID，并将 *my-cluster* 替换为您的集群名称。如果您希望在账户中对所有集群使用相同角色，请将 *my-cluster* 替换为 *。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:us-
east-1:111122223333:fargateprofile/my-cluster/*"
         }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ii. 选择更新策略。

Amazon CLI

a. 复制并将以下内容粘贴到名为 pod-execution-role-trust-policy.json 的文件中。将 *region-code* 替换为您的集群所在的 Amazon 区域。如果您希望在账户中的所有 Amazon 区域使用相同的角色，请将 *region-code* 替换为 *。将 *111122223333* 替换为您的账户 ID，并将 *my-cluster* 替换为您的集群名称。如果您希望在账户中对所有集群使用相同角色，请将 *my-cluster* 替换为 *。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
```

```
            "aws:SourceArn": "arn:aws:eks:us-east-1:111122223333:fargateprofile/
my-cluster/*"
          }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 创建容器组（pod）执行 IAM 角色。

```
aws iam create-role \
  --role-name AmazonEKSFargatePodExecutionRole \
  --assume-role-policy-document file://"pod-execution-role-trust-policy.json"
```

c. 将所需的 Amazon EKS 托管 IAM policy 附加到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy \
  --role-name AmazonEKSFargatePodExecutionRole
```

Amazon EKS Connector IAM 角色

您可以连接 Kubernetes 集群以便在 Amazon Web Services 管理控制台 中查看。要连接 Kubernetes 集群，创建 IAM 角色。

检查现有 EKS connector 角色

您可以使用以下过程检查并查看您的账户是否已有 Amazon EKS connector 角色。

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 角色。
3. 在角色列表中搜索 `AmazonEKSConnectorAgentRole`。如果不存在包含 `AmazonEKSConnectorAgentRole` 的角色，请参阅 the section called “创建 Amazon EKS Connector 代理角色” 来创建角色。如果包含 `AmazonEKSConnectorAgentRole` 的角色确实存在，则选择角色以查看附加的策略。
4. 选择权限。

5. 确保将 AmazonEKSConnectorAgentPolicy 托管策略附加到此角色。如果附加该策略，则将正确配置 Amazon EKS connector 角色。
6. 选择 Trust relationships（信任关系），然后选择 Edit trust policy（编辑信任策略）。
7. 验证信任关系是否包含以下策略。如果信任关系符合以下策略，请选择 Cancel（取消）。如果信任关系不匹配，请将策略复制到编辑信任策略窗口并选择更新策略。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ssm.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

创建 Amazon EKS Connector 代理角色

您可以使用 Amazon Web Services 管理控制台 或 Amazon CloudFormation 创建 connector 代理角色。

Amazon CLI

a. 创建一个名为 `eks-connector-agent-trust-policy.json` 的文件，其中包含要用于 IAM 角色的以下 JSON。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ssm.amazonaws.com"
                ]
            },
```

```
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 创建一个名为 eks-connector-agent-policy.json 的文件，其中包含要用于 IAM 角色的以下 JSON。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SsmControlChannel",
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateControlChannel"
            ],
            "Resource": "arn:aws:eks:*:*:cluster/*"
        },
        {
            "Sid": "ssmDataplaneOperations",
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateDataChannel",
                "ssmmessages:OpenDataChannel",
                "ssmmessages:OpenControlChannel"
            ],
            "Resource": "*"
        }
    ]
}
```

c. 使用您在之前列表项中创建的信任策略和策略创建 Amazon EKS Connector 代理角色。

```
aws iam create-role \
     --role-name AmazonEKSConnectorAgentRole \
     --assume-role-policy-document file://eks-connector-agent-trust-policy.json
```

d. 将该策略附加到 Amazon EKS Connector 代理角色。

```
aws iam put-role-policy \
     --role-name AmazonEKSConnectorAgentRole \
     --policy-name AmazonEKSConnectorAgentPolicy \
```

```
    --policy-document file://eks-connector-agent-policy.json
```

Amazon CloudFormation

a. 将以下 Amazon CloudFormation 模板保存到本地系统中的文本文件。

> **Note**
>
> 此模板还创建服务相关角色，否则调用 `registerCluster` API 时将创建。有关详细信息，请参阅 the section called “集群连接器角色”。

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Provisions necessary resources needed to register clusters in EKS'
Parameters: {}
Resources:
  EKSConnectorSLR:
    Type: Amazon::IAM::ServiceLinkedRole
    Properties:
      AWSServiceName: eks-connector.amazonaws.com

  EKSConnectorAgentRole:
    Type: Amazon::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action: [ 'sts:AssumeRole' ]
            Principal:
              Service: 'ssm.amazonaws.com'

  EKSConnectorAgentPolicy:
    Type: Amazon::IAM::Policy
    Properties:
      PolicyName: EKSConnectorAgentPolicy
      Roles:
        - {Ref: 'EKSConnectorAgentRole'}
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
```

```
            Action: [ 'ssmmessages:CreateControlChannel' ]
            Resource:
            - Fn::Sub: 'arn:${Amazon::Partition}:eks:*:*:cluster/*'
          - Effect: 'Allow'
            Action: [ 'ssmmessages:CreateDataChannel',
 'ssmmessages:OpenDataChannel', 'ssmmessages:OpenControlChannel' ]
            Resource: "*"
Outputs:
  EKSConnectorAgentRoleArn:
    Description: The agent role that EKS connector uses to communicate with Amazon
 services.
    Value: !GetAtt EKSConnectorAgentRole.Arn
```

b. 打开 Amazon CloudFormation 控制台。

c. 选择使用新资源创建堆栈（标准）。

d. 对于指定模板，选择上传模板文件，然后选择选择文件。

e. 选择您之前创建的文件，然后选择下一步。

f. 对于 Stack name (堆栈名称)，输入角色的名称，如 `eksConnectorAgentRole`，然后选择 Next (下一步)。

g. 在配置堆栈选项页面上，请选择下一步。

h. 在 Review（审核）页面上审核您的信息，确认堆栈可创建 IAM 资源，然后选择 Create stack（创建堆栈）。

AmazonAmazon Elastic Kubernetes Service 的托管策略

Amazon 托管式策略是由 Amazon 创建和管理的独立策略。Amazon 托管式策略旨在为许多常见使用场景提供权限，以便您可以开始为用户、组和角色分配权限。

请记住，Amazon 托管式策略可能不会为您的特定使用案例授予最低权限，因为其可供所有 Amazon 客户使用。我们建议通过定义特定于使用案例的客户管理型策略来进一步减少权限。

您无法更改 Amazon 托管式策略中定义的权限。如果 Amazon 更新在 Amazon 托管策略中定义的权限，则更新会影响该策略所附加到的所有主体身份（用户、组和角色）。当新的 Amazon 服务启动或新的 API 操作可用于现有服务时，Amazon 最有可能更新 Amazon 托管策略。

有关更多信息，请参阅Amazon《IAM 用户指南》中的托管策略。

Amazon托管策略：AmazonEKS_CNI_Policy

您可以将 AmazonEKS_CNI_Policy 附加到 IAM 实体。在创建 Amazon EC2 节点组之前，此策略必须附加到节点 IAM 角色，或适用于 Kubernetes 的 Amazon VPC CNI 插件专用的 IAM 角色。这样它可以代表您执行操作。我们建议您将策略附加到仅由插件使用的角色。有关更多信息，请参阅the section called “Amazon VPC CNI”和the section called “配置 IRSA”。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- **ec2:*NetworkInterface** 和 **ec2:*PrivateIpAddresses**：允许 Amazon VPC CNI 插件执行操作，例如为容器组（pod）预置弹性网络接口和 IP 地址，以便为 Amazon EKS 中运行的应用程序提供联网。
- **ec2** 读取操作 – 允许 Amazon VPC CNI 插件执行操作，例如描述实例和子网，以查看您的 Amazon VPC 子网中的免费 IP 地址数量。VPC CNI 可以使用每个子网中的免费 IP 地址来选择空闲 IP 地址最多的子网，以便在创建弹性网络接口时使用。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKS_CNI_Policy。

Amazon托管策略：AmazonEKSClusterPolicy

您可以将 AmazonEKSClusterPolicy 附加到您的 IAM 实体。在创建集群之前，您必须拥有附加了此策略的集群 IAM 角色。由 Amazon EKS 托管的 Kubernetes 集群会代表您调用其他 Amazon 服务。它们这样做的目的是为了管理您与服务一起使用的资源。

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- **autoscaling** – 读取和更新自动扩缩组的配置。Amazon EKS 不使用这些权限，但它们保留在策略中，以确保向后兼容。
- **ec2** – 使用与 Amazon EC2 节点关联的卷和网络资源执行工作。此为必需操作，以便 Kubernetes 控制面板可以将实例加入到集群，并动态预置和管理 Kubernetes 持久卷请求的 Amazon EBS 卷。
- **ec2**：删除由 VPC CNI 创建的弹性网络接口。这是必需项，以便 EKS 可以清理 VPC CNI 意外退出时留下的弹性网络接口。
- **elasticloadbalancing** ：使用弹性负载均衡器并将节点添加到其中作为目标。此为必需操作，以便 Kubernetes 控制层面能够动态预置 Kubernetes 服务请求的 Elastic Load Balancer。

- **iam** ：创建服务相关角色。此为必需操作，以便 Kubernetes 控制面板能够动态预置 Kubernetes 服务请求的 Elastic Load Balancer。
- **kms** – 从 Amazon KMS 读取密钥。这对于 Kubernetes 控制面板是必需的，以支持 etcd 中存储的 Kubernetes 密钥的密钥加密。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSClusterPolicy。

Amazon 托管策略：AmazonEKSDashboardConsoleReadOnly

您可以将 AmazonEKSDashboardConsoleReadOnly 附加到 IAM 实体。

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- **eks**：对 EKS 控制面板数据、资源和集群版本信息的只读访问权限。这允许查看与 EKS 相关的指标和集群配置详细信息。
- **organizations**：对 Amazon Organizations 信息的只读访问权限，包括：
  - 查看组织详细信息和服务访问权限
  - 列出组织根、账户和组织单位
  - 查看组织结构

要查看 JSON 策略文档的最新版本，请参阅《Amazon Managed Policy Reference Guide》中的 AmazonEKSDashboardConsoleReadOnly。

Amazon托管策略：AmazonEKSFargatePodExecutionRolePolicy

您可以将 AmazonEKSFargatePodExecutionRolePolicy 附加到您的 IAM 实体。在创建 Fargate 配置文件之前，您必须创建 Fargate 容器组（pod）执行角色并将此策略附加到其上。有关更多信息，请参阅 the section called “第 2 步：创建 Fargate 容器组（pod）执行角色” 和 the section called “定义配置文件”。

此策略向该角色授予权限，可提供对必须在 Fargate 上运行 Amazon EKS 容器组（pod）的其他 Amazon 服务资源的访问权限。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- **ecr** ：允许在 Fargate 上运行的容器组（pod）提取存储在 Amazon ECR 中的容器镜像。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSFargatePodExecutionRolePolicy。

Amazon 托管式策略：AmazonEKSConnectorServiceRolePolicy

您不能将 AmazonEKSConnectorServiceRolePolicy 附加到自己的 IAM 实体。将此策略附加到允许 Amazon EKS 代表您执行操作的服务相关角色。有关更多信息，请参阅 the section called “集群连接器角色”。

该角色允许 Amazon EKS 连接 Kubernetes 集群。附加的策略允许角色管理必要的资源，以便连接到注册的 Kubernetes 集群。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限。

- **SSM Management** ：创建、描述和删除 SSM 激活，并取消注册托管实例。此功能允许基本的 Systems Manager 操作。
- **Session Management** ：启动专门针对 EKS 集群的 SSM 会话，并使用 AmazonEKS 文档执行非交互式命令。
- **IAM Role Passing** ：将 IAM 角色专门传递给 SSM 服务，该服务由将传递角色限制为 ssm.amazonaws.com 的条件控制。
- **EventBridge Rules** ：创建 EventBridge 规则和目标，但仅当由 eks-connector.amazonaws.com 管理时。作为事件源，规则特别限于 Amazon SSM。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSConnectorServiceRolePolicy。

Amazon托管策略：AmazonEKSForFargateServiceRolePolicy

您不能将 AmazonEKSForFargateServiceRolePolicy 附加到自己的 IAM 实体。将此策略附加到允许 Amazon EKS 代表您执行操作的服务相关角色。有关更多信息，请参阅 AWSServiceRoleforAmazonEKSForFargate。

此策略授予 Amazon EKS 运行 Fargate 任务所必要的权限。仅当您具有 Fargate 节点时，才会使用此策略。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限。

- **ec2** ：创建和删除弹性网络接口，并描述弹性网络接口和资源。此为必需操作，以便 Amazon EKS Fargate 服务可以配置 Fargate 容器组（pod）所需的 VPC 联网。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSForFargateServiceRolePolicy。

Amazon 托管策略：AmazonEKSComputePolicy

您可以将 AmazonEKSComputePolicy 附加到 IAM 实体。您可以将此策略附加到您的集群 IAM 角色，以扩展 EKS 可以在您的账户中管理的资源。

该策略授予 Amazon EKS 为 EKS 集群创建和管理 EC2 实例所需的权限，以及配置 EC2 所需的 IAM 权限。此外，此策略还授予 Amazon EKS 代表您创建 EC2 竞价型实例服务相关角色的权限。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- **ec2** 权限：
  - ec2:CreateFleet 和 ec2:RunInstances – 允许创建 EC2 实例，并将特定的 EC2 资源（映像、安全组、子网）用于 EKS 集群节点。
  - ec2:CreateLaunchTemplate – 允许为 EKS 集群节点创建 EC2 启动模板。
  - 该策略还包括一些条件，用于将这些 EC2 权限的使用限制在标有 EKS 集群名称和其他相关标签的资源。
  - ec2:CreateTags – 允许向 CreateFleet、RunInstances 和 CreateLaunchTemplate 操作创建的 EC2 资源添加标签。
- **iam** 权限：
  - iam:AddRoleToInstanceProfile – 允许向 EKS 计算实例配置文件添加 IAM 角色。
  - iam:PassRole – 允许将必要的 IAM 角色传递给 EC2 服务。

要查看 JSON 策略文档的最新版本，请参阅《Amazon Managed Policy Reference Guide》中的 AmazonEKSComputePolicy。

Amazon 托管策略：AmazonEKSNetworkingPolicy

您可以将 AmazonEKSNetworkingPolicy 附加到 IAM 实体。您可以将此策略附加到您的集群 IAM 角色，以扩展 EKS 可以在您的账户中管理的资源。

该策略旨在授予 Amazon EKS 创建和管理 EKS 集群网络接口的必要权限，从而允许控制面板和 Worker 节点正常通信和运行。

权限详细信息

此策略授予以下权限以允许 Amazon EKS 管理集群的网络接口：

- **ec2**网络接口权限：
  - `ec2:CreateNetworkInterface` – 允许创建 EC2 网络接口。
  - 该策略包括一些条件，用于将此权限的使用限制在标有 EKS 集群名称和 Kubernetes CNI 节点名称的网络接口。
  - `ec2:CreateTags` – 允许向 `CreateNetworkInterface` 操作创建的网络接口添加标签。
- **ec2**网络接口管理权限：
  - `ec2:AttachNetworkInterface`、`ec2:DetachNetworkInterface` – 允许将网络接口连接和分离到 EC2 实例。
  - `ec2:UnassignPrivateIpAddresses`、`ec2:UnassignIpv6Addresses`、`ec2:AssignPrivateIp` – 允许管理网络接口的 IP 地址分配。
  - 这些权限仅限于标有 EKS 集群名称的网络接口。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSNetworkingPolicy。

Amazon 托管策略：AmazonEKSBlockStoragePolicy

您可以将 `AmazonEKSBlockStoragePolicy` 附加到 IAM 实体。您可以将此策略附加到您的集群 IAM 角色，以扩展 EKS 可以在您的账户中管理的资源。

该策略授予 Amazon EKS 为 EKS 集群创建、管理和维护 EC2 卷和快照的必要权限，从而允许控制面板和 Worker 节点按照 Kubernetes 工作负载的要求预置和使用永久存储。

权限详细信息

此 IAM 策略授予以下允许 Amazon EKS 管理 EC2 卷和快照的权限：

- **ec2** 卷管理权限：
  - `ec2:AttachVolume`、`ec2:DetachVolume`、`ec2:ModifyVolume`、`ec2:EnableFastSnapshotRe` – 允许附加、分离、修改和启用 EC2 卷的快速快照还原。

  - 这些权限仅限于标有 EKS 集群名称的卷。
  - `ec2:CreateTags` – 允许向 `CreateVolume` 和 `CreateSnapshot` 操作创建的 EC2 卷和快照添加标签。
- **ec2** 卷创建权限：
  - `ec2:CreateVolume` – 允许创建新的 EC2 卷。
  - 该策略包括一些条件，用于将此项权限的使用限制在标有 EKS 集群名称和其他相关标签的卷。
  - `ec2:CreateSnapshot` – 允许创建新的 EC2 卷快照。
  - 该策略包括一些条件，用于将此项权限的使用限制在标有 EKS 集群名称和其他相关标签的快照。

要查看 JSON 策略文档的最新版本，请参阅《Amazon Managed Policy Reference Guide》中的 AmazonEKSBlockStoragePolicy。

Amazon 托管策略：AmazonEKSLoadBalancingPolicy

您可以将 `AmazonEKSLoadBalancingPolicy` 附加到 IAM 实体。您可以将此策略附加到您的集群 IAM 角色，以扩展 EKS 可以在您的账户中管理的资源。

此 IAM 策略授予 Amazon EKS 使用各种 Amazon 服务管理弹性负载均衡器（ELB）和相关资源的必要权限。

权限详细信息

此策略授予的密钥权限是：

- **elasticloadbalancing** ：允许创建、修改和管理弹性负载均衡器和目标组。这包括创建、更新和删除负载均衡器、目标组、侦听器和规则的权限。
- **ec2** ：允许创建和管理 Kubernetes 控制面板将实例加入到集群并管理 Amazon EBS 卷所需的安全组。还允许描述和列出 EC2 资源，例如实例、VPC、子网、安全组和其他联网资源。
- **iam** ：允许为弹性负载均衡创建服务相关角色，这是 Kubernetes 控制面板动态预置 ELB 所必需的。
- **kms** ：允许通过 Amazon KMS 读取密钥，这对于 Kubernetes 控制面板是必需的，以支持 etcd 中存储的 Kubernetes 密钥的加密。
- **wafv2** 和 **shield** ：允许关联和取消关联 Web ACL 以及为弹性负载均衡器创建/删除 Amazon Shield 保护。
- **cognito-idp** 、 **acm** 和 **elasticloadbalancing** ：授予描述用户池客户端、列出和描述证书以及描述目标群体的权限，这些权限是 Kubernetes 控制面板管理弹性负载均衡器所必需的。

该策略还包括多项条件检查，确保使用 `eks:eks-cluster-name` 标签将权限范围限于所托管的特定 EKS 集群。

要查看 JSON 策略文档的最新版本，请参阅《Amazon Managed Policy Reference Guide》中的 AmazonEKSLoadBalancingPolicy。

Amazon 托管式策略：AmazonEKSMCPReadOnlyAccess

您可以将 `AmazonEKSMCPReadOnlyAccess` 附加到 IAM 实体。此策略提供对 Amazon EKS 资源和相关 Amazon 服务的只读访问权限，使 Amazon EKS 模型上下文协议（MCP）服务器无需对您的基础设施进行任何修改，即可执行可观测性和问题排查操作。

权限详细信息

此策略包含允许主体完成下列任务的以下权限：

- **`eks`** 允许主体描述和列出 EKS 集群、节点组、附加组件、访问条目、见解以及访问 Kubernetes API 以执行只读操作。
- **`iam`** 允许主体检索有关 IAM 角色、策略及其附件的信息，以了解与 EKS 资源关联的权限。
- **`ec2`** 允许主体描述 VPC、子网和路由表，以了解 EKS 集群的网络配置。
- **`sts`** 允许主体出于身份验证和授权目的检索呼叫方身份信息。
- **`logs`** 允许主体启动查询并从 CloudWatch Logs 中检索查询结果以进行问题排查和监控。
- **`cloudwatch`** 允许主体检索用于监控集群和工作负载性能的指标数据。
- **`eks-mcp`** 允许主体在 Amazon EKS MCP 服务器中调用 MCP 操作和调用只读工具。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSMCPReadOnlyAccess。

Amazon托管策略：AmazonEKSServicePolicy

您可以将 `AmazonEKSServicePolicy` 附加到 IAM 实体。在 2020 年 4 月 16 日之前创建的集群需要您创建 IAM 角色并向其附加此策略。在 2020 年 4 月 16 日或之后创建的集群无需您创建角色，也不需要您分配此策略。当您使用具有 `iam:CreateServiceLinkedRole` 权限的 IAM 主体创建集群时，将会自动为您创建 AWSServiceRoleforAmazonEKS 服务相关角色。服务相关角色附加了托管策略：AmazonEKSServiceRolePolicy。

此策略允许 Amazon EKS 创建和管理运行 Amazon EKS 集群所需的资源。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限。

- **eks** ：启动更新后，更新您集群的 Kubernetes 版本。Amazon EKS 不使用此权限，但其仍保留在策略中，以确保向后兼容。
- **ec2** ：使用弹性网络接口和其它网络资源和标签。这是 Amazon EKS 配置联网以便于节点和 Kubernetes 控制层面之间进行通信所必需的。阅读有关安全组的信息。更新安全组上的标签。
- **route53** ：将 VPC 与托管区域关联。这是 Amazon EKS 实现 Kubernetes 集群 API 服务器私有端点联网所必需的。
- **logs** ：录入事件。此为必需操作，以便 Amazon EKS 可以将 Kubernetes 控制层面日志发送到 CloudWatch。
- **iam** ：创建服务相关角色。此为必需操作，以便 Amazon EKS 可以代表您创建 the section called “Amazon EKS 的服务相关角色权限” 服务相关角色。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSServicePolicy。

Amazon托管策略：AmazonEKSServiceRolePolicy

您不能将 AmazonEKSServiceRolePolicy 附加到自己的 IAM 实体。将此策略附加到允许 Amazon EKS 代表您执行操作的服务相关角色。有关更多信息，请参阅 the section called “Amazon EKS 的服务相关角色权限”。当您使用具有 iam:CreateServiceLinkedRole 权限的 IAM 主体创建集群时，将会自动为您创建 AWSServiceRoleforAmazonEKS 服务相关角色，而且此策略会附加到该角色。

此策略允许服务相关角色代表您调用Amazon服务。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限。

- **ec2** ：创建和描述弹性网络接口和 Amazon EC2 实例、集群安全组及创建集群所需的 VPC。有关更多信息，请参阅 the section called “安全组要求”。阅读有关安全组的信息。更新安全组上的标签。阅读关于按需容量预留的信息。阅读包括路由表和网络 ACL 在内的 VPC 配置，以检测配置问题，这是集群见解的一部分。
- **ec2** 自动模式 – 终止由 EKS 自动模式创建的 EC2 实例。有关更多信息，请参阅 EKS 自动模式。
- **iam** ：列出附加到 IAM 角色的所有托管式策略。此为必需操作，以便 Amazon EKS 能够列出和验证创建集群所需的所有托管策略和权限。

- 将 VPC 与托管区关联 – 这是 Amazon EKS 实现 Kubernetes 集群 API 服务器私有端点联网所必需的。
- 日志事件 – 此为必需操作，以便 Amazon EKS 可以将 Kubernetes 控制面板日志发送到 CloudWatch。
- 放入指标 – 此为必需操作，这样 Amazon EKS 就可以将 Kubernetes 控制面板日志发送到 CloudWatch。
- **`eks`** ：管理集群访问条目和策略，允许精细控制可以访问 EKS 资源的主体以及其可执行的操作。这包括为计算、联网、负载均衡和存储操作关联标准访问策略。
- **`elasticloadbalancing`** ：创建、管理和删除与 EKS 集群关联的负载均衡器及其组件（侦听器、目标组、证书）。查看负载均衡器属性和运行状态。
- **`events`** – 创建并管理 EventBridge 规则，以便监控与 EKS 集群相关的 EC2 和 Amazon Health 事件，实现对基础设施更改和运行状况警报的自动响应。
- **`iam`** ：管理含“eks”前缀的 EC2 实例配置文件，包括创建、删除和角色关联，这是 EKS 节点管理所必需的。
- **`pricing shield`** ：访问 Amazon 定价信息和 Shield 保护状态，为 EKS 资源启用成本管理和高级安全功能。
- 资源清理 – 在集群清理操作期间，安全删除带有 EKS 标签的资源，包括卷、快照、启动模板和网络接口。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSServiceRolePolicy。

Amazon托管策略：AmazonEKSVPCResourceController

您可以将 `AmazonEKSVPCResourceController` 策略附加到 IAM 身份。如果您使用适用于容器组（pod）的安全组，您必须将此策略附加到您的 Amazon EKS 集群 IAM 角色，以便其代表您执行操作。

此策略授予集群角色管理弹性网络接口和节点 IP 地址的权限。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- **`ec2`** ：管理弹性网络接口和 IP 地址以支持容器组（pod）安全组和 Windows 节点。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSVPCResourceController。

Amazon托管策略：AmazonEKSWorkerNodePolicy

您可以将 AmazonEKSWorkerNodePolicy 附加到 IAM 实体。您必须将此策略附加到您在创建允许 Amazon EKS 代表您执行操作的 Amazon EC2 节点时指定的节点 IAM 角色。如果您使用 eksctl 创建节点组，则其会创建节点 IAM 角色并自动将此策略附加到角色。

此策略授予 Amazon EKS Amazon EC2 节点连接到 Amazon EKS 集群的权限。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- **ec2** ：读取实例卷和网络信息。此为必需操作，以便 Kubernetes 节点能够描述有关节点加入 Amazon EKS 集群所需的 Amazon EC2 资源的信息。
- **eks** ：可选地将集群描述为节点引导启动的一部分。
- **eks-auth:AssumeRoleForPodIdentity** ：允许检索节点上 EKS 工作负载的凭证。需要执行此操作 EKS 容器组身份才能正常运行。

要查看 JSON 策略文档的最新版本，请参阅《Amazon Managed Policy Reference Guide》中的 AmazonEKSWorkerNodePolicy。

Amazon 托管策略：AmazonEKSWorkerNodeMinimalPolicy

您可以将 AmazonEKSWorkerNodeMinimalPolicy 附加到您的 IAM 实体。您可以将此策略附加到您在创建允许 Amazon EKS 代表您执行操作的 Amazon EC2 节点时指定的节点 IAM 角色。

此策略授予 Amazon EKS Amazon EC2 节点连接到 Amazon EKS 集群的权限。与 AmazonEKSWorkerNodePolicy 相比，此策略的权限较少。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- eks-auth:AssumeRoleForPodIdentity – 允许检索节点上 EKS 工作负载的凭证。需要执行此操作 EKS 容器组身份才能正常运行。

要查看 JSON 策略文档的最新版本，请参阅《Amazon Managed Policy Reference Guide》中的 AmazonEKSWorkerNodePolicy。

Amazon托管策略：AWSServiceRoleForAmazonEKSNodegroup

您不能将 AWSServiceRoleForAmazonEKSNodegroup 附加到自己的 IAM 实体。将此策略附加到允许 Amazon EKS 代表您执行操作的服务相关角色。有关更多信息，请参阅 the section called “Amazon EKS 的服务相关角色权限”。

此策略授予 AWSServiceRoleForAmazonEKSNodegroup 角色权限，允许其在您的账户中创建和管理 Amazon EC2 节点组。

权限详细信息

此策略包含允许 Amazon EKS 完成以下任务的以下权限：

- **ec2** ：与安全组、标签、容量预留和启动模板结合使用。Amazon EKS 托管式节点组需要此功能才能启用远程访问配置和描述可在托管式节点组中使用的容量预留。此外，Amazon EKS 托管节点组会代表您创建启动模板。这样做的目的是配置为每个托管节点组提供支持的 Amazon EC2 Auto Scaling 组。
- **iam** ：创建服务相关角色并传递角色。这是 Amazon EKS 托管节点组管理创建托管节点组时传递的角色的实例配置文件所必需的。此实例配置文件由作为托管节点组的一部分启动的 Amazon EC2 实例使用。Amazon EKS 需要为其他服务（如 Amazon EC2 Auto Scaling 组）创建服务相关角色。这些权限用于创建托管节点组。
- **autoscaling** ：使用安全自动扩缩组。这是 Amazon EKS 托管节点组管理支持每个托管节点组的 Amazon EC2 Auto Scaling 组所必需的。它还用于支持一些功能，例如，在节点组更新期间终止或回收节点时移出容器组（pod）。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AWSServiceRoleForAmazonEKSNodegroup。

Amazon 托管策略：AmazonEKSDashboardServiceRolePolicy

您不能将 AmazonEKSDashboardServiceRolePolicy 附加到自己的 IAM 实体。将此策略附加到允许 Amazon EKS 代表您执行操作的服务相关角色。有关更多信息，请参阅 the section called “Amazon EKS 的服务相关角色权限”。

此策略授予 AWSServiceRoleForAmazonEKSDashboard 角色权限，允许其在您的账户中创建和管理 Amazon EC2 节点组。

权限详细信息

此策略包含允许访问以完成这些任务的以下权限：

- **organizations**：查看有关 Amazon Organizations 结构和账户的信息。这包括列出组织中的账户、查看组织单元和根、列出委派管理员、查看有权访问组织的服务以及检索有关组织和账户详细信息的权限。

要查看 JSON 策略文档的最新版本，请参阅《Amazon Managed Policy Reference Guide》中的 AmazonEKSDashboardServiceRolePolicy。

Amazon 托管策略：AmazonEBSCSIDriverPolicy

AmazonEBSCSIDriverPolicy 策略允许 Amazon EBS Container Storage Interface（CSI）驱动程序代表您创建、修改、复制、附加、分离和删除卷。这包括修改现有卷上的标签以及在 EBS 卷上启用快速快照还原（FSR）。此外还将向 EBS CSI 驱动程序授予创建、还原和删除快照，以及列出实例、卷和快照的权限。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEBSCSIDriverServiceRolePolicy。

Amazon 托管式策略：AmazonEFSCSIDriverPolicy

AmazonEFSCSIDriverPolicy 策略允许 Amazon EFS 容器存储接口（CSI）代表您创建和删除接入点。该策略还将向 Amazon EFS CSI 驱动程序授予列出您的接入点文件系统、挂载目标和 Amazon EC2 可用区的权限。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEFSCSIDriverServiceRolePolicy。

Amazon 托管策略：AmazonEKSLocalOutpostClusterPolicy

您可以将此策略附加到 IAM 实体。在创建本地集群之前，您必须将此策略附加到集群角色。由 Amazon EKS 托管的 Kubernetes 集群会代表您调用其他 Amazon 服务。它们这样做的目的是为了管理您与服务一起使用的资源。

AmazonEKSLocalOutpostClusterPolicy 包含以下权限：

- **ec2** 读取操作：允许控制面板实例描述可用区、路由表、实例和网络接口属性。Amazon EC2 实例作为控制面板实例成功加入集群所必需的权限。

- **ssm** ：允许通向控制面板实例的 Amazon EC2 Systems Manager 连接，Amazon EKS 使用该连接与您账户中的本地集群通信并对其进行管理。
- **logs** ：允许实例将日志推送到 Amazon CloudWatch。
- **secretsmanager** – 允许实例安全地从 Amazon Secrets Manager 中获取和删除控制面板实例的引导数据。
- **ecr** ：允许容器组（pod）和在控制面板实例上运行的容器拉取存储在 Amazon Elastic Container Registry 中的容器映像。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSLocalOutpostClusterPolicy。

Amazon 托管策略：AmazonEKSLocalOutpostServiceRolePolicy

您不能将此策略附加到您的 IAM 实体。当您使用具有 `iam:CreateServiceLinkedRole` 权限的 IAM 主体创建集群时，Amazon EKS 将自动为您创建 AWSServiceRoleforAmazonEKSLocalOutpost 服务相关角色，并将此策略附加到该角色。此策略允许该服务相关角色代表您为本地集群调用 Amazon 服务。

`AmazonEKSLocalOutpostServiceRolePolicy` 包含以下权限：

- **ec2** ：允许 Amazon EKS 使用安全、网络和其他资源，成功启动和管理您的账户中的控制面板实例。
- **ssm**、**ssmmessages**：允许通向控制面板实例的 Amazon EC2 Systems Manager 连接，Amazon EKS 使用该连接与您账户中的本地集群通信并对其进行管理。
- **iam** ：允许 Amazon EKS 管理与控制面板实例关联的实例配置文件。
- **secretsmanager** – 允许 Amazon EKS 将控制面板实例的引导数据放入 Amazon Secrets Manager，以便能在实例引导期间安全地引用它。
- **outposts** ：允许 Amazon EKS 从您的账户中获取 Outpost 信息，以便在 Outpost 中成功启动本地集群。

要查看 JSON 策略文档的最新版本，请参阅《Amazon 托管式策略参考指南》中的 AmazonEKSLocalOutpostServiceRolePolicy。

Amazon托管策略的 Amazon EKS 更新

查看有关 Amazon EKS（自从其开始跟踪更新更改以来）的Amazon托管策略的更新的详细信息。

要接收此特定文档页面的所有源文件更改通知，您可以通过 RSS 阅读器订阅以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/security/iam-reference/security-iam-awsmanpol.adoc.atom
```

更改	描述	日期
引入 the section called “Amazon 托管式策略：AmazonEKSMCPReadOnlyAccess”。	Amazon EKS 已发布一项全新的托管式策略 AmazonEKSMCPReadOnlyAccess ，用于在 Amazon EKS MCP 服务器中启用只读工具，实现可观测性和问题排查功能。	2025 年 11 月 21 日
为 AmazonEBSCSIDriverPolicy 增加了权限。	增加了 ec2:CopyVolumes 权限，支持 EBS CSI 驱动程序直接复制 EBS 卷。	2025 年 11 月 17 日
已将权限添加至 the section called “Amazon托管策略：AmazonEKSServiceRolePolicy”。	已为 AmazonEKSServiceRolePolicy 新增 ec2:DescribeRouteTables 和 ec2:DescribeNetworkAcls 权限。作为集群洞察功能的一部分，此项能力支持 Amazon EKS 检测混合节点的 VPC 路由表与网络 ACL 的配置问题。	2025 年 10 月 22 日
为 AWSServiceRoleForAmazonEKSConnector 添加了权限	已为 AmazonEKSConnectorServiceRolePolicy 新增 ssmmessages:OpenDataChannel 权限	2025 年 10 月 15 日
已将权限添加至 the section called “Amazon托管策略：AmazonEKSServiceRolePolicy”	此角色可以附加新的访问策略 AmazonEKSEventPolicy 。ec2:DeleteLaunchTemplate 和	2025 年 8 月 26 日

更改	描述	日期
	ec2:TerminateInstances 的权限受限。	
已将权限添加至 the section called “Amazon 托管策略：AmazonEKSLocalOutpostServiceRolePolicy”	已将 ssmmessages:OpenDataChannel 权限添加至 AmazonEKSLocalOutpostServiceRolePolicy 。	2025 年 6 月 26 日
已将权限添加至 the section called “Amazon 托管策略：AmazonEKSComputePolicy”。	更新了 ec2:RunInstances 和 ec2:CreateFleet 操作的资源权限以包括容量预留 arn:aws-cn:ec2:*:*:capacity-reservation/* 。这允许 Amazon EKS 自动模式使用您账户中的 EC2 按需容量预留来启动实例。添加了 iam:CreateServiceLinkedRole 以允许 Amazon EKS 自动模式代表您创建 EC2 竞价型实例服务相关角色 AWSServiceRoleForEC2Spot 。	2025 年 6 月 20 日
已将权限添加至 AmazonEKSServiceRolePolicy。	添加了 ec2:DescribeCapacityReservations 权限以允许 Amazon EKS 自动模式使用您账户中的 EC2 按需容量预留来启动实例。	2025 年 6 月 20 日
引入 the section called “Amazon 托管策略：AmazonEKSDashboardConsoleReadOnly”。	引入了新的 AmazonEKSDashboardConsoleReadOnly 策略。	2025 年 6 月 19 日

更改	描述	日期
引入 the section called “Amazon 托管策略：AmazonEKSDashboardServiceRolePolicy”。	引入了新的 AmazonEKSDashboardServiceRolePolicy 策略。	2025 年 5 月 21 日
已将权限添加至 AmazonEKSClusterPolicy。	添加了 ec2:DeleteNetworkInterfaces 权限，允许 Amazon EKS 删除 VPC CNI 意外退出时留下的弹性网络接口。	2025 年 4 月 16 日
已将权限添加至 AmazonEKSServiceRolePolicy。	作为 EKS 1.33 版本发行的一部分，添加了 ec2:RevokeSecurityGroupEgress 和 ec2:AuthorizeSecurityGroupEgress 权限，允许 EKS AI/ML 客户向与 EFA 兼容的默认 EKS 集群安全组添加安全组出口规则。	2025 年 4 月 14 日
已将权限添加至 AmazonEKSServiceRolePolicy。	增加了终止由 EKS 自动模式创建的 EC2 实例的权限。	2025 年 2 月 28 日

更改	描述	日期
为 AmazonEBSCSIDriver Policy 增加了权限。	增加了授权 EBS CSI 驱动程序还原所有快照的新语句。此前现有策略允许这一操作，但由于 IAM 处理 `CreateVolume` 的方式发生了变化，因此需要新的显式语句。 增加了 EBS CSI 驱动程序修改现有卷上标签的功能。EBS CSI 驱动程序可以通过 Kubernetes VolumeAttributesClasses 中的参数修改现有卷的标签。 增加了 EBS CSI 驱动程序在 EBS 卷上启用快速快照还原（FSR）的功能。EBS CSI 驱动程序可以通过 Kubernetes 存储类中的参数在新卷上启用 FSR。	2025 年 1 月 13 日
已将权限添加至 the section called “Amazon 托管策略：AmazonEKSLoadBalancingPolicy”。	更新了 `AmazonEKSLoadBalancingPolicy` ，以允许列出和描述联网和 IP 地址资源。	2024 年 12 月 26 日
已将权限添加至 the section called “Amazon托管策略：AWSServiceRoleForAmazonEKSNodegroup”。	为兼容中国区域，更新了 `AWSServiceRoleForAmazonEKSNodegroup` 。	2024 年 11 月 22 日

更改	描述	日期
已将权限添加至 the section called “Amazon 托管策略：AmazonEKSLocalOutpostClusterPolicy”	为 AmazonEKSLocalOutpostClusterPolicy 添加了 ec2:DescribeAvailabilityZones 权限，以便集群控制面板上的 Amazon Cloud Controller Manager 可以识别各个节点所在的可用区。	2024 年 11 月 21 日
已将权限添加至 the section called “Amazon托管策略：AWSServiceRoleForAmazonEKSNodegroup”。	更新了 AWSServiceRoleForAmazonEKSNodegroup 策略，允许 ec2:RebootInstances ，以实现由 Amazon EKS 托管节点组创建的实例。限制了 Amazon EC2 资源的 ec2:CreateTags 权限。	2024 年 11 月 20 日
已将权限添加至 the section called “Amazon托管策略：AmazonEKSServiceRolePolicy”。	EKS 更新了 Amazon 托管策略 AmazonEKSServiceRolePolicy 。添加了 EKS 访问策略、负载均衡器管理和自动集群资源清理的权限。	2024 年 11 月 16 日
引入 the section called “Amazon 托管策略：AmazonEKSComputePolicy”。	EKS 更新了 Amazon 托管策略 AmazonEKSComputePolicy 。更新了 iam:AddRoleToInstanceProfile 操作的资源权限。	2024 年 11 月 7 日
引入 the section called “Amazon 托管策略：AmazonEKSComputePolicy”。	Amazon 引入了 AmazonEKSComputePolicy 。	2024 年 11 月 1 日

更改	描述	日期
已将权限添加至 AmazonEKS ClusterPolicy	添加了 ec2:Describe InstanceTopology 权限，允许 Amazon EKS 将拓扑信息作为标签附加到节点。	2024 年 11 月 1 日
引入 the section called “Amazon 托管策略：AmazonEKSBlockStoragePolicy”。	Amazon 引入了 AmazonEKS BlockStoragePolicy 。	2024 年 10 月 30 日
引入 the section called “Amazon 托管策略：AmazonEKSLoadBalancing Policy”。	Amazon 引入了 AmazonEKS LoadBalancingPolicy 。	2024 年 10 月 30 日
已将权限添加至 AmazonEKS ServiceRolePolicy。	添加了 cloudwatch:PutMetricData 权限，允许 Amazon EKS 向 Amazon CloudWatch 发布指标。	2024 年 10 月 29 日
引入 the section called “Amazon 托管策略：AmazonEKSNetworkingPolicy”。	Amazon 引入了 AmazonEKS NetworkingPolicy 。	2024 年 10 月 28 日
增加了对 AmazonEKS ServicePolicy 和 AmazonEKSServiceRolePolicy 的权限。	添加了 ec2:GetSecurityGroupsForVpc 和相关的标签权限，以允许 EKS 读取安全组信息和更新相关标签。	2024 年 10 月 10 日
引入了 AmazonEKS WorkerNodeMinimalPolicy。	Amazon 引入了 AmazonEKS WorkerNodeMinimalPolicy 。	2024 年 10 月 3 日

更改	描述	日期
已将权限添加至 AWSServiceRoleForAmazonEKSNodegroup。	添加了 autoscaling:ResumeProcesses 和 autoscaling:SuspendProcesses 权限，以允许 Amazon EKS 在 Amazon EKS 托管的自动扩缩组中暂停和恢复 AZRebalance 。	2024 年 8 月 21 日
已将权限添加至 AWSServiceRoleForAmazonEKSNodegroup。	增加了 ec2:DescribeCapacityReservations 权限，以允许 Amazon EKS 描述用户账户的容量预留。增加了 autoscaling:PutScheduledUpdateGroupAction 权限，以在 CAPACITY_BLOCK 节点组上启用设置计划扩缩。	2024 年 6 月 27 日
AmazonEKS_CNI_Policy – 更新到现有策略	Amazon EKS 添加了新的 ec2:DescribeSubnets 权限，允许适用于 Kubernetes 的 Amazon VPC CNI 插件查看 Amazon VPC 子网中的免费 IP 地址数量。VPC CNI 可以使用每个子网中的免费 IP 地址来选择空闲 IP 地址最多的子网，以便在创建弹性网络接口时使用。	2024 年 3 月 4 日
AmazonEKSWorkerNodePolicy：更新现有策略	Amazon EKS 添加了新的权限以允许 EKS 容器组身份。Amazon EKS 容器组身份代理使用节点角色。	2023 年 11 月 26 日

更改	描述	日期
推出了 AmazonEFSCSIDriverPolicy。	Amazon 引入了 AmazonEFSCSIDriverPolicy 。	2023 年 7 月 26 日
已将权限添加至 AmazonEKSClusterPolicy。	增加了 ec2:DescribeAvailabilityZones 权限以允许 Amazon EKS 在创建负载均衡器时在子网自动发现期间获取可用区详细信息。	2023 年 2 月 7 日
更新了 AmazonEBSCSIDriverPolicy 中的策略条件。	删除了 StringLike 键字段中带有通配符的无效策略条件。还将一个新条件 ec2:ResourceTag/kubernetes.io/created-for/pvc/name: "*" 添加到 ec2:DeleteVolume ，以允许 EBS CSI 驱动程序删除由树内插件创建的卷。	2022 年 11 月 17 日

更改	描述	日期
添加了对 AmazonEKS LocalOutpostServiceRolePolicy 的权限。	添加了 `ec2:DescribeVPCAttribute` 、`ec2:GetConsoleOutput` 和 `ec2:DescribeSecret` 以实现更好的先决条件验证和托管式生命周期控制。还添加了 `ec2:DescribePlacementGroups` 、`"arn:aws-cn:ec2:*:*:placement-group/*"` 和 `ec2:RunInstances` 以支持控制面板 Amazon EC2 实例在 Outposts 上的置放控制。	2022 年 10 月 24 日
更新了 AmazonEKSLocalOutpostClusterPolicy 中的 Amazon Elastic Container Registry 权限。	将操作 `ecr:GetDownloadUrlForLayer` 从所有资源部分移至限定范围部分。添加了资源 `arn:aws-cn:ecr:*:*:repository/eks/` 。删除了资源 `arn:aws-cn:ecr:` 。此资源涵盖在增加的 `arn:aws-cn:ecr:*:*:repository/eks/*` 资源中。	2022 年 10 月 20 日
将权限添加到了 AmazonEKS LocalOutpostClusterPolicy。	添加了 `arn:aws-cn:ecr:*:*:repository/kubelet-config-updater` Amazon Elastic Container Registry 存储库，以便集群控制面板实例能够更新某些 `kubelet` 参数。	2022 年 8 月 31 日

更改	描述	日期
引入了 AmazonEKSLocalOutpostClusterPolicy。	Amazon 引入了 AmazonEKSLocalOutpostClusterPolicy 。	2022 年 8 月 24 日
引入了 AmazonEKSLocalOutpostServiceRolePolicy。	Amazon 引入了 AmazonEKSLocalOutpostServiceRolePolicy 。	2022 年 8 月 23 日
引入的 AmazonEBSCSIDriverPolicy。	Amazon 引入了 AmazonEBSCSIDriverPolicy 。	2022 年 4 月 22 日
已添加权限到 AmazonEKSWorkerNodePolicy。	增加了 ec2:DescribeInstanceTypes 以启用能够自动发现实例级别属性的 Amazon EKS 优化版 AMI。	2022 年 3 月 21 日
已将权限添加至 AWSServiceRoleForAmazonEKSNodegroup。	添加了 autoscaling:EnableMetricsCollection 权限以允许 Amazon EKS 启用指标收集。	2021 年 12 月 13 日
已将权限添加至 AmazonEKSClusterPolicy。	添加了 ec2:DescribeAccountAttributes 、ec2:DescribeAddresses 和 ec2:DescribeInternetGateways 权限，以允许 Amazon EKS 为 Network Load Balancer 创建服务相关角色。	2021 年 6 月 17 日
Amazon EKS 已开始跟踪更改。	Amazon EKS 开始跟踪其Amazon托管策略的更改。	2021 年 6 月 17 日

IAM 故障排除

本主题介绍将 Amazon EKS 与 IAM 结合使用时可能遇到的一些常见错误以及相应的错误处理方式。

AccessDeniedException

如果您在调用 Amazon API 操作时收到 `AccessDeniedException`，则表明您使用的 IAM 主体凭证不具有发起该调用所需的权限。

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws-cn:iam::111122223333:user/user_name is not authorized to perform:
eks:DescribeCluster on resource: arn:aws-cn:eks:region:111122223333:cluster/my-cluster
```

在前述示例消息中，用户没有权限调用 Amazon EKS `DescribeCluster` API 操作。要为 IAM 主体提供 Amazon EKS 管理员权限，请参阅the section called “基于身份的策略”。

有关 IAM 的一般信息，请参阅 IAM 用户指南中的使用策略控制访问。

无法在计算选项卡上看到节点，或在资源选项卡上看到任何内容，您将在 Amazon Web Services 管理控制台 中收到错误

您可能会看到一条内容为“`Your current user or role does not have access to Kubernetes objects on this EKS cluster`”的控制台错误消息。确保您将 Amazon Web Services 管理控制台 与其结合使用的 IAM 主体用户具有必要的权限。有关更多信息，请参阅 the section called “所需的权限”。

aws-auth **`ConfigMap`** 不授予对集群的访问权限

Amazon IAM 身份验证器不允许在 `ConfigMap` 中使用角色 ARN 中的路径。因此，在指定 `rolearn` 之前，请删除路径。例如，将 `arn:aws-cn:iam::111122223333:role/team/developers/eks-admin` 更改为 `arn:aws-cn:iam::111122223333:role/eks-admin` 。

我无权执行 iam:PassRole

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Amazon EKS。

有些 Amazon 服务允许您将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 Amazon EKS 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: {arn-aws}iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系 Amazon 管理员。您的管理员是提供登录凭证的人。

我希望允许我的Amazon账户以外的人访问我的 Amazon EKS 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon EKS 是否支持这些功能，请参阅 the section called “Amazon EKS 和 IAM”。
- 要了解如何为您拥有的 Amazon 账户中的资源提供访问权限，请参阅 IAM 用户指南中的为您拥有的另一个 Amazon 账户中的 IAM 用户提供访问权限。
- 要了解如何为第三方 Amazon 账户提供您的资源的访问权限，请参阅IAM 用户指南中的 为第三方拥有的 Amazon 账户提供访问权限。
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

容器组（pod）容器收到以下错误：**An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region**

如果应用程序明确向 Amazon STS 全局端点（ `https://sts.amazonaws` ）提出要求并且 Kubernetes 服务账户已配置为使用区域端点，您的容器会收到此错误。您可以使用以下选项之一解决问题：

- 更新应用程序代码以删除对 Amazon STS 全局端点的显式调用。
- 更新应用程序代码以明确调用区域端点，例如 `https://sts.us-west-2.amazonaws.com` 。您的应用程序应内置冗余功能，以便在该 Amazon 区域的服务出现故障时选择其它 Amazon 区域。有关更多信息，请参阅《IAM 用户指南》中的在 Amazon 区域中管理 Amazon STS。

- 将服务账户配置为使用全局端点。默认情况下，集群使用区域端点。有关更多信息，请参阅 the section called “STS 端点”。

Amazon EKS 集群 IAM 角色

每个集群都需要一个 Amazon EKS 集群 IAM 角色。由 Amazon EKS 管理的 Kubernetes 集群会使用此角色来管理节点，而旧版云提供商会使用此角色为服务创建带有 Elastic Load Balancing 的负载均衡器。

必须先使用以下任一 IAM policy 创建 IAM 角色，然后才能创建 Amazon EKS 集群：

- AmazonEKSClusterPolicy
- 自定义 IAM policy。随后的最低权限允许 Kubernetes 集群管理节点，但不允许旧版云提供商创建带有 Elastic Load Balancing 的负载均衡器。您的自定义 IAM policy必须至少具有以下权限：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
      ],
      "Resource": "arn:aws:ec2:*:*:instance/*",
      "Condition": {
        "ForAnyValue:StringLike": {
          "aws:TagKeys": "kubernetes.io/cluster/*"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstanceTopology",
        "kms:DescribeKey"
      ],
```

```
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 在 2023 年 10 月 3 日之前，每个集群的 IAM 角色都必须有 AmazonEKSClusterPolicy。在 2020 年 4 月 16 日之前，AmazonEKSServicePolicy 和 AmazonEKSClusterPolicy 是必需的，建议的角色名称是 eksServiceRole。有了 AWSServiceRoleForAmazonEKS 服务相关角色后，在 2020 年 4 月 16 日或之后创建的集群将不再需要 AmazonEKSServicePolicy 策略。

检查现有集群角色

可使用以下程序检查并确定您的账户是否已有 Amazon EKS 集群角色。

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 角色。
3. 在角色列表中搜索 eksClusterRole。如果不存在包含 eksClusterRole 的角色，请参阅 the section called “创建 Amazon EKS 集群角色” 来创建角色。如果包含 eksClusterRole 的角色确实存在，则选择角色以查看附加的策略。
4. 选择权限。
5. 确保将 AmazonEKSClusterPolicy 托管策略附加到此角色。如果附加该策略，则将正确配置 Amazon EKS 集群角色。
6. 选择 Trust relationships（信任关系），然后选择 Edit trust policy（编辑信任策略）。
7. 验证信任关系是否包含以下策略。如果信任关系符合以下策略，请选择 Cancel（取消）。如果信任关系不匹配，请将策略复制到编辑信任策略窗口并选择更新策略。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

创建 Amazon EKS 集群角色

要创建集群角色，您可以使用 Amazon Web Services 管理控制台 或 Amazon CLI。

Amazon Web Services 管理控制台

a. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
b. 选择 Roles，然后选择 Create role。
c. 在受信任的实体类型下，选择 Amazon 服务。
d. 从其它 Amazon 服务的使用案例下拉列表中，选择 EKS。
e. 为您的使用案例选择 EKS - Cluster（EKS - 集群），然后选择 Next（下一步）。
f. 在 Add permissions（添加权限）选项卡上，选择 Next（下一步）。
g. 对于 Role name (角色名称)，为角色输入唯一名称，例如 `eksClusterRole`。
h. 对于 Description（说明），请输入描述性文本，例如 `Amazon EKS - Cluster role`。
i. 选择创建角色。

Amazon CLI

a. 将以下内容复制到名为 *`cluster-trust-policy.json`* 的文件中。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 创建角色。您可以将 *`eksClusterRole`* 替换为您选择的任何名称。

```
aws iam create-role \
  --role-name eksClusterRole \
  --assume-role-policy-document file://"cluster-trust-policy.json"
```

c. 将所需的 IAM policy 附加到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name eksClusterRole
```

Amazon EKS 节点 IAM 角色

Amazon EKS 节点 `kubelet` 守护进程代表您调用 Amazon API。节点通过 IAM 实例配置文件和关联的策略获得这些 API 调用的权限。您必须先为节点创建 IAM 角色以在启动它们时使用，然后才能启动这些节点并在集群中注册它们。此要求适用于通过由 Amazon 提供的 Amazon EKS 优化版 AMI 启动的节点，也适用于您打算使用的任何其他节点 AMI。此外，此要求适用于托管节点组和自行管理的节点。

> **Note**
>
> 您不能使用创建任何集群时使用的相同角色。

必须先使用以下权限创建 IAM 角色，然后才能创建节点：

- `kubelet` 描述 VPC 中 Amazon EC2 资源的权限，例如 AmazonEKSWorkerNodePolicy 策略提供的权限。该策略还为 Amazon EKS 容器组身份代理提供权限。
- `kubelet` 使用来自 Amazon Elastic Container Registry（Amazon ECR）的容器映像的权限，例如 AmazonEC2ContainerRegistryPullOnly 策略提供的权限。使用来自 Amazon Elastic Container Registry（Amazon ECR）的容器映像的权限是必要条件，因为用于联网的内置附加组件运行的 Pod 使用来自 Amazon ECR 的容器映像。
- （可选）Amazon EKS 容器组身份代理使用 `eks-auth:AssumeRoleForPodIdentity` 操作检索容器组凭证的权限。如果不使用 AmazonEKSWorkerNodePolicy，则除了使用 EKS 容器组身份的 EC2 权限外，您还必须提供此权限。
- （可选）如果不使用 IRSA 或 EKS 容器组身份向 VPC CNI 容器组授予权限，则必须在实例角色中提供 VPC CNI 的权限。您可以使用 `AmazonEKS_CNI_Policy` 托管策略（如果使用 IPv4 系列创

建集群）或您创建的 IPv6 策略（如果使用 IPv6 系列创建集群）。但是，我们建议您将策略附加到专门用于 Amazon VPC CNI 附加组件的单独角色，而不是附加到此角色。有关为 Amazon VPC CNI 附加组件创建单独角色的详细信息，请参阅 the section called “配置 IRSA”。

> **Note**
>
> Amazon EC2 节点组必须具有与 Fargate 配置文件不同的 IAM 角色。有关更多信息，请参阅 the section called “容器执行 IAM 角色”。

检查现有节点角色

可以使用以下过程检查并查看您的账户是否已有 Amazon EKS 节点角色。

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 角色。
3. 在角色列表中搜索 eksNodeRole、AmazonEKSNodeRole 或 NodeInstanceRole。如果其中一个名称的角色不存在，请参阅the section called “创建 Amazon EKS 节点 IAM 角色”以创建该角色。如果包含 eksNodeRole、AmazonEKSNodeRole 或 NodeInstanceRole 的角色确实存在，请选择该角色以查看附加的策略。
4. 选择权限。
5. 确保将 AmazonEKSWorkerNodePolicy 和 AmazonEC2ContainerRegistryPullOnly 托管策略附加到此角色，或者使用最低权限附加自定义策略。

   > **Note**
   >
   > 如果 AmazonEKS_CNI_Policy 策略已附加到角色，我们建议删除此策略并改为将其附加到映射到 aws-node Kubernetes 服务账户的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。

6. 选择 Trust relationships（信任关系），然后选择 Edit trust policy（编辑信任策略）。
7. 验证信任关系是否包含以下策略。如果信任关系符合以下策略，请选择 Cancel（取消）。如果信任关系不匹配，请将策略复制到编辑信任策略窗口并选择更新策略。

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

创建 Amazon EKS 节点 IAM 角色

您可以使用 Amazon Web Services 管理控制台 或 Amazon 创建节点 IAM 角色。

Amazon Web Services 管理控制台

a. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
b. 在左侧导航窗格中，选择 Roles（角色）。
c. 在 Roles（角色）页面上，选择 Create role（创建角色）。
d. 在 Select trusted entity（选择受信任的实体）页面上，请执行以下操作：
   i. 在可信实体类型部分中，选择 Amazon 服务。
   ii. 在 Use case（使用案例）下，选择 EC2。
   iii. 选择下一步。
e. 在添加权限页面上，附加自定义策略或执行以下操作：
   i. 在 Filter policies (筛选器策略) 框中，输入 `AmazonEKSWorkerNodePolicy`。
   ii. 选中搜索结果中的 AmazonEKSWorkerNodePolicy 左侧的复选框。
   iii. 请选择 Clear filters（清除筛选条件）。
   iv. 在 Filter policies (筛选器策略) 框中，输入 `AmazonEC2ContainerRegistryPullOnly`。
   v. 选中搜索结果中的 AmazonEC2ContainerRegistryPullOnly 左侧的复选框。

      AmazonEKS_CNI_Policy 托管式策略或您创建的 IPv6 策略还必须附加到此角色或映射到 `aws-node` Kubernetes 服务账户的其他角色。我们建议将策略分配给与 Kubernetes 服务账

户关联的角色，而不是将其分配给此角色。有关更多信息，请参阅 the section called “配置 IRSA”。

vi. 选择下一步。

f. 在 Name, review, and create（命名、查看和创建）页面中，请执行以下操作：

i. 对于 Role name (角色名称)，为角色输入唯一名称，例如 `AmazonEKSNodeRole`。

ii. 对于 Description（说明），请将当前文本替换为描述性文本，例如 `Amazon EKS - Node role`。

iii. 在添加标签（可选）下，将标签作为键值对附加，以将元数据添加到角色。有关在 IAM 中使用标签的更多信息，请参阅《IAM 用户指南》 中的标记 IAM 资源。

iv. 选择创建角色。

Amazon CLI

a. 运行以下命令以创建 `node-role-trust-relationship.json` 文件。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

b. 创建 IAM 角色。

```
aws iam create-role \
  --role-name AmazonEKSNodeRole \
  --assume-role-policy-document file://"node-role-trust-relationship.json"
```

c. 将两个所需的 IAM 托管策略附加到 IAM 角色。

```
aws iam attach-role-policy \
```

```
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name AmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly \
  --role-name AmazonEKSNodeRole
```

d. 根据创建集群时使用的 IP 系列，将以下 IAM policy 之一附加到 IAM 角色。该策略必须附加到此角色，或与用于 Kubernetes 的 Amazon VPC CNI 插件的 Kubernetes `aws-node` 服务账户关联的角色。我们建议将策略分配给与 Kubernetes 服务账户关联的角色。要将策略分配给与 Kubernetes 服务账户关联的角色，请参阅 the section called “配置 IRSA”。

- IPv4

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name AmazonEKSNodeRole
```

- IPv6

A. 复制以下文本并将其保存到名为 `vpc-cni-ipv6-policy.json` 的文件。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
```

```
    ]
}
```

B. 创建 IAM policy。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-
document file://vpc-cni-ipv6-policy.json
```

C. 将 IAM policy 附加到 IAM 角色。将 *111122223333* 替换为您的账户 ID。

```
aws iam attach-role-policy \
  --policy-arn arn:aws-cn:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
 \
  --role-name AmazonEKSNodeRole
```

Amazon EKS 自动模式集群 IAM 角色

每个集群都需要一个 Amazon EKS 集群 IAM 角色。由 Amazon EKS 管理的 Kubernetes 集群使用此角色来自动执行存储、联网和计算自动扩缩等方面的例行任务。

在创建 Amazon EKS 集群之前，必须首先使用 EKS 自动模式必需的策略创建一个 IAM 角色。您可以附加建议的 Amazon IAM 托管式策略，也可以创建具有等效权限的自定义策略。

- AmazonEKSComputePolicy
- AmazonEKSBlockStoragePolicy
- AmazonEKSLoadBalancingPolicy
- AmazonEKSNetworkingPolicy
- AmazonEKSClusterPolicy：

检查现有集群角色

可使用以下程序检查并确定您的账户是否已有 Amazon EKS 集群角色。

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 角色。
3. 在角色列表中搜索 AmazonEKSAutoClusterRole。如果不存在包含 AmazonEKSAutoClusterRole 的角色，请参阅下一部分的说明以创建该角色。如果包含 AmazonEKSAutoClusterRole 的角色确实存在，则选择角色以查看附加的策略。

4. 选择权限。
5. 确保将 AmazonEKSClusterPolicy 托管策略附加到此角色。如果附加该策略，则将正确配置 Amazon EKS 集群角色。
6. 选择 Trust relationships（信任关系），然后选择 Edit trust policy（编辑信任策略）。
7. 验证信任关系是否包含以下策略。如果信任关系符合以下策略，请选择 Cancel（取消）。如果信任关系不匹配，请将策略复制到编辑信任策略窗口并选择更新策略。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

> **Note**
>
> Amazon 不要求此角色使用名称 `AmazonEKSAutoClusterRole`。

创建 Amazon EKS 集群角色

要创建集群角色，您可以使用 Amazon Web Services 管理控制台 或 Amazon CLI。

Amazon Web Services 管理控制台

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 选择 Roles，然后选择 Create role。
3. 在受信任的实体类型下，选择 Amazon 服务。
4. 从其它 Amazon 服务的使用案例下拉列表中，选择 EKS。

5. 为您的使用案例选择 EKS - Cluster（EKS - 集群），然后选择 Next（下一步）。
6. 在添加权限选项卡上，选择相关策略，然后选择下一步。
   - AmazonEKSComputePolicy
   - AmazonEKSBlockStoragePolicy
   - AmazonEKSLoadBalancingPolicy
   - AmazonEKSNetworkingPolicy
   - AmazonEKSClusterPolicy：
7. 对于 Role name (角色名称)，为角色输入唯一名称，例如 `AmazonEKSAutoClusterRole`。
8. 对于 Description（说明），请输入描述性文本，例如 `Amazon EKS - Cluster role`。
9. 选择创建角色。

Amazon CLI

1. 将以下内容复制到名为 *`cluster-trust-policy.json`* 的文件中。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

2. 创建角色。您可以将 *`AmazonEKSAutoClusterRole`* 替换为您选择的任何名称。

```
aws iam create-role \
  --role-name AmazonEKSAutoClusterRole \
  --assume-role-policy-document file://"cluster-trust-policy.json"
```

3. 将所需的 IAM 策略附加到角色：

AmazonEKSClusterPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSComputePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSNetworkingPolicy
```

Amazon EKS 自动模式节点 IAM 角色

> **Note**
>
> 您不能使用创建任何集群时使用的相同角色。

在创建节点之前，必须首先创建具有以下策略或同等权限的 IAM 角色：

- AmazonEKSWorkerNodeMinimalPolicy

- AmazonEC2ContainerRegistryPullOnly

检查现有节点角色

可以使用以下过程检查并查看您的账户是否已有 Amazon EKS 节点角色。

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 角色。
3. 在角色列表中搜索 AmazonEKSAutoNodeRole。如果具有其中一个名称的角色不存在，请参阅下一部分的说明以创建该角色。如果包含 AmazonEKSAutoNodeRole 的角色确实存在，请选择该角色以查看附加的策略。
4. 选择权限。
5. 确保附加了上述必需的策略或等效自定义策略。
6. 选择 Trust relationships（信任关系），然后选择 Edit trust policy（编辑信任策略）。
7. 验证信任关系是否包含以下策略。如果信任关系符合以下策略，请选择 Cancel（取消）。如果信任关系不匹配，请将策略复制到编辑信任策略窗口并选择更新策略。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

创建 Amazon EKS 节点 IAM 角色

您可以使用 Amazon Web Services 管理控制台 或 Amazon 创建节点 IAM 角色。

Amazon Web Services 管理控制台

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 Roles（角色）。

3. 在 Roles（角色）页面上，选择 Create role（创建角色）。
4. 在 Select trusted entity（选择受信任的实体）页面上，请执行以下操作：
   a. 在可信实体类型部分中，选择 Amazon 服务。
   b. 在 Use case（使用案例）下，选择 EC2。
   c. 选择下一步。
5. 在添加权限页面上，附加以下策略：
   - AmazonEKSWorkerNodeMinimalPolicy
   - AmazonEC2ContainerRegistryPullOnly
6. 在 Name, review, and create（命名、查看和创建）页面中，请执行以下操作：
   a. 对于 Role name (角色名称)，为角色输入唯一名称，例如 `AmazonEKSAutoNodeRole`。
   b. 对于 Description（说明），请将当前文本替换为描述性文本，例如 `Amazon EKS - Node role`。
   c. 在添加标签（可选）下，将标签作为键值对附加，以将元数据添加到角色。有关在 IAM 中使用标签的更多信息，请参阅《IAM 用户指南》 中的标记 IAM 资源。
   d. 选择创建角色。

Amazon CLI

创建节点 IAM 角色

使用上一步中的 node-trust-policy.json 文件来定义哪些实体可以代入该角色。运行以下命令以创建节点 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

记下角色 ARN

创建角色后，检索并保存节点 IAM 角色的 ARN。在后续步骤中，您将需要此 ARN。使用以下命令来获取 ARN：

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

附加必需的策略

将以下 Amazon 托管式策略附加到节点 IAM 角色，以提供必要的权限：

附加 AmazonEKSWorkerNodeMinimalPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

附加 AmazonEC2ContainerRegistryPullOnly：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

Amazon EKS 功能 IAM 角色

EKS 功能需要配置功能 IAM 角色（或功能角色）。EKS 功能使用此角色对 Amazon 服务执行操作，并通过自动创建的访问条目访问集群中的 Kubernetes 资源。

在功能创建期间指定功能角色之前，您必须创建具有相应信任策略和相应功能类型的权限的 IAM 角色。创建此 IAM 角色后，可以将其重用于任意数量的功能资源。

功能角色要求

功能角色必须满足以下要求：

- 该角色必须与集群和功能资源位于同一 Amazon 账户中
- 该角色必须拥有信任策略，以允许 EKS 功能服务代入该角色
- 该角色必须拥有与功能类型和使用案例要求相适应的权限（请参阅the section called “按功能类型划分的权限”）

功能角色的信任策略

所有功能角色必须包含以下信任策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

此信任策略允许 EKS 执行以下操作：

- 担任该角色，以执行 Amazon API 操作
- 标记会话以进行审计和跟踪

按功能类型划分的权限

所需的 IAM 权限取决于您使用的功能和部署模式。

> **Note**
>
> 对于搭配使用 IAM 角色选择器与 ACK 的生产部署，或者在没有 Amazon 服务集成的情况下使用 kro 或 Argo CD 时，功能角色可能不需要信任策略之外的任何 IAM 权限。

kro（Kube Resource Orchestrator）

不需要 IAM 权限。您可以创建没有附加策略的功能角色。kro 只需要 Kubernetes RBAC 权限即可创建和管理 Kubernetes 资源。

Amazon Controllers for Kubernetes（ACK）

ACK 支持两种权限模型：

- 简单设置（开发/测试）：直接向功能角色添加 Amazon 服务权限。这非常适合入门级的单账户部署或所有用户都需要相同权限的情况。
- 生产最佳实践：使用 IAM 角色选择器，实施最低权限访问。使用这种方法，功能角色只需要获得 `sts:AssumeRole` 权限即可代入服务特定角色。您无需向功能角色本身添加 Amazon 服务权限（如 S3 或 RDS），系统会将这些权限授予映射到特定命名空间的各个 IAM 角色。

IAM 角色选择器可启用以下功能：

- 命名空间级别权限隔离
- 跨账户资源管理
- 团队特定 IAM 角色
- 最低权限安全模型

IAM 角色选择器方法的功能角色策略示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::111122223333:role/ACK-S3-Role",
        "arn:aws:iam::111122223333:role/ACK-RDS-Role",
        "arn:aws:iam::444455556666:role/ACKCrossAccountRole"
      ]
    }
  ]
}
```

有关包括 IAM 角色选择器在内的 ACK 权限配置的详细信息，请参阅the section called “配置权限”。

Argo CD

默认情况下，不需要 IAM 权限。以下功能可能需要可选权限：

- Amazon Secrets Manager：如果使用 Secrets Manager 来存储 Git 存储库凭证
- Amazon CodeConnections：如果使用 CodeConnections 进行 Git 存储库身份验证

Secrets Manager 和 CodeConnections 策略示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:region:account-id:secret:argocd/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection",
        "codeconnections:GetConnection"
      ],
      "Resource": "arn:aws:codeconnections:region:account-id:connection/*"
    }
  ]
}
```

有关 Argo CD 权限要求的详细信息，请参阅 the section called “Argo CD 注意事项”。

检查现有功能角色

您可使用以下过程来检查账户是否已具有适合您使用案例的功能 IAM 角色。

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 角色。
3. 在角色列表中搜索功能角色名称（例如，`ACKCapabilityRole` 或 `ArgoCDCapabilityRole`）。
4. 如果存在角色，请选择该角色，查看附加策略和信任关系。
5. 选择 Trust relationships（信任关系），然后选择 Edit trust policy（编辑信任策略）。
6. 验证信任关系是否与功能信任策略相匹配。如果不匹配，请更新信任策略。
7. 选择权限并验证该角色是否具有适用于您的功能类型和使用案例的权限。

创建功能 IAM 角色

要创建功能角色，您可以使用 Amazon Web Services 管理控制台 或 Amazon CLI。

Amazon Web Services 管理控制台

a. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
b. 选择 Roles，然后选择 Create role。
c. 对于可信实体类型，选择自定义信任策略。

d. 将功能信任策略复制并粘贴到信任策略编辑器中。

e. 选择下一步。

f. 在添加权限选项卡上，选择或创建适合您的功能类型的策略（请参阅the section called “按功能类型划分的权限”）。对于 kro，您可以跳过此步骤。

g. 选择下一步。

h. 对于角色名称，请为角色输入唯一名称，例如 `ACKCapabilityRole`、`ArgoCDCapabilityRole` 或 `kroCapabilityRole`。

i. 对于 Description（说明），请输入描述性文本，例如 `Amazon EKS - ACK capability role`。

j. 选择创建角色。

AmazonCLI

a. 将功能信任策略复制到名为 `capability-trust-policy.json` 的文件中。

b. 创建角色。将 `ACKCapabilityRole` 替换为您所需的角色名称。

```
aws iam create-role \
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://capability-trust-policy.json
```

c. 将所需的 IAM 策略附加到角色。对于 ACK，请为要管理的 Amazon 服务附加策略。对于 Argo CD，请根据需要为 Secrets Manager 或 CodeConnections 附加策略。对于 kro，您可以跳过此步骤。

具有 S3 权限的 ACK 示例：

```
aws iam put-role-policy \
  --role-name ACKCapabilityRole \
  --policy-name S3Management \
  --policy-document file://s3-policy.json
```

对功能角色问题进行问题排查

功能创建失败，并显示“IAM 角色无效”

验证：

- 该角色与集群位于同一账户中
- 信任策略与功能信任策略相匹配

- 您拥有该角色的 `iam:PassRole` 权限

功能显示权限错误

验证：

- 该角色具有此功能类型所需的 IAM 权限
- 该角色的访问条目存在于集群上
- 如有需要，还可配置其他 Kubernetes 权限（请参阅the section called “其他 Kubernetes 权限”）

ACK 资源失败并出现“权限被拒绝”错误

验证：

- 此角色具有您的使用案例所需的 IAM 权限
- 对于引用密钥的 ACK 控制器，请确保已将范围限定为相应命名空间的 `AmazonEKSSecretReaderPolicy` 访问条目策略进行了关联。

有关更多问题排查指导，请参阅the section called “EKS 功能的注意事项”。

监控集群性能并查看日志

您可以使用许多可用的监控或日志记录工具在 Amazon EKS 中观察数据。您的 Amazon EKS 日志数据可以流式传输到 Amazon 服务或合作伙伴工具来进行数据分析。Amazon Web Services 管理控制台 中提供有很多服务，它们提供数据来对您的 Amazon EKS 问题进行排查。您还可以使用 Amazon 支持的开源解决方案来监控 Amazon EKS 基础设施。

在 Amazon EKS 控制台的左侧导航窗格中选择集群后，您可以通过选择集群名称和可观测性选项卡，来查看集群运行状况和详细信息。要查看有关部署到集群的任何现有 Kubernetes 资源的详细信息，请参阅 the section called “访问集群资源”。

监控是保持 Amazon EKS 和您的 Amazon 解决方案的可靠性、可用性和性能的重要方面。我们建议您从您的 Amazon 解决方案的所有部分收集监控数据。这样，您可以更轻松地调试出现的多点故障。在开始监控 Amazon EKS 之前，请确保您的监控计划可以解决以下问题。

- 您的目标是什么？ 如果集群大幅扩展，是否需要实时通知？
- 需要观察哪些资源？
- 您需要多久观察一次这些资源？ 贵公司是否想快速应对风险？
- 您想使用什么工具？ 如果您已经在启动过程中运行 Amazon Fargate，则您可以使用内置的日志路由器。
- 您想由谁负责执行监控任务？
- 当出现问题时，您希望向谁发送通知？

Amazon EKS 上的监控和日志记录

Amazon EKS 提供了用于监控和日志记录的内置工具。对于支持的版本，通过可观测性仪表板，您可以了解集群的性能。它可帮助您进行快速检测、排除故障和修复问题。除了监控功能外，它还包括基于控制面板审计日志的列表。Kubernetes 控制面板公开了许多指标，这些指标也可以在控制台之外抓取。

控制面板日志记录记录对集群的所有 API 调用、审计信息，以捕获哪些用户对集群执行了哪些操作，以及基于角色的信息。有关更多信息，请参阅 Amazon 规范性指导中的 Amazon EKS 的日志记录和监控。

Amazon EKS 控制面板日志记录将审计和诊断日志直接从 Amazon EKS 控制面板提供到您账户中的 CloudWatch Logs。这些日志可让您轻松地保护和运行您的集群。您可以选择您需要的确切的日志类

型，日志将作为日志流发送到 CloudWatch 中每个 Amazon EKS 集群的组。有关更多信息，请参阅 the section called “控制面板日志”。

> **Note**
>
> 当您在 Amazon CloudWatch 中检查 Amazon EKS 身份验证器日志时，系统将显示一些文本类似于以下示例文本的条目。
>
> ```
> level=info msg="mapping IAM role" groups="[]" role="arn:aws-
> cn:iam::111122223333:role/XXXXXXXXXXXXXXXXXX-NodeManagerRole-XXXXXXXX"
>  username="eks:node-manager"
> ```
>
> 预期为包含此文本的条目。`username` 是一个 Amazon EKS 内部服务角色，用于为托管节点组和 Fargate 执行特定操作。
> 对于低级、可自定义的日志记录，Kubernetes 日志记录可用。

Amazon EKS 与 Amazon CloudTrail 集成，后者是在 Amazon EKS 中提供用户、角色或 Amazon 服务所采取操作的记录的服务。CloudTrail 将 Amazon EKS 的所有 API 调用作为事件捕获。这些捕获包括来自 Amazon EKS 控制台的调用和对 Amazon EKS API 操作的代码调用。有关更多信息，请参阅 the section called “ Amazon CloudTrail”。

Kubernetes API 服务器公开了大量可用于监控和分析的指标。有关更多信息，请参阅 the section called “Prometheus 指标”。

要配置自定义 Amazon CloudWatch 日志的 Fluent Bit，请参阅《Amazon CloudWatch 用户指南》中的设置 Fluent Bit。

Amazon EKS 监控和日志记录工具

Amazon Web Services 为您提供了各种可用于监控 Amazon EKS 的工具。您可以配置一些工具来设置自动监控，但有些工具需要手动调用。我们建议您在环境和现有工具集允许的范围内尽量实现监控任务自动化。

下表介绍了各种监控工具选项。

领域	工具	说明	设置
控制面板	可观测性仪表板	对于支持的版本，通过可观测性仪表板，您可以了解集群的性能。它可帮助您进行快速检测、排除故障和修复问题。	设置过程
应用程序/控制面板	Prometheus	Prometheus 可用于监控应用程序和控制面板的指标和警报。	设置过程
应用程序	CloudWatch Container Insights	CloudWatch Container Insights 从容器化应用程序和微服务中收集、聚合和汇总指标与日志。	设置过程
应用程序	Amazon Distro for OpenTelemetry（ADOT）	ADOT 收集和发送相关的指标，跟踪数据和元数据并将其发送到 Amazon 监控服务或合作伙伴。它可以通过 CloudWatch Container Insights 设置。	设置过程
应用程序	Amazon DevOps Guru	Amazon DevOps Guru 可以检测节点级别的运行性能和可用性。	设置过程
应用程序	Amazon X-Ray	Amazon X-Ray 接收有关应用程序的跟踪数据。此跟踪数据包括传入请求和传出请求以及有关请	设置过程

领域	工具	说明	设置
		求的元数据。对于 Amazon EKS，实施需要 OpenTelemetry 附加组件。	
应用程序	Amazon CloudWatch	CloudWatch 在支持的版本上免费提供部分基本的 Amazon EKS 指标。您可以使用 CloudWatch Observability Operator 扩展此功能，以处理收集指标、日志和跟踪数据。	设置过程

下表介绍了各种日志记录工具选项。

领域	工具	说明	设置
控制面板	可观测性仪表板	对于支持的版本，可观测性仪表板会根据控制面板审计日志显示列表。它还包括指向 Amazon CloudWatch 中控制面板日志的链接。	设置过程
应用程序	Amazon CloudWatch Container Insights	Amazon CloudWatch Container Insights 从容器化应用程序和微服务中收集、聚合及汇总指标与日志。	设置过程

领域	工具	说明	设置
控制面板	Amazon CloudWatch Logs	您可以直接从 Amazon EKS 控制面板将审计和诊断日志提供到您账户中的 CloudWatch Logs。	设置过程
控制面板	Amazon CloudTrail	它记录用户、角色或服务进行的 API 调用。	设置过程
Amazon Fargate 实例的多个区域	Amazon Fargate 日志路由器	对于 Amazon Fargate 实例，日志路由器将日志流式传输到 Amazon 服务或合作伙伴工具。它将 Amazon 用于 Fluent Bit。日志可以流式传输到其他 Amazon 服务或合作伙伴工具。	设置过程

使用可观测性仪表板监控您的集群

Amazon EKS 控制台包含一个可观测性仪表板，可供您了解集群的性能。其提供的信息有助于快速检测、排除故障和修复问题。在运行状况和性能摘要中选择一项即可打开可观测性控制面板的相应部分。此摘要包含在多个地方，包括可观测性选项卡。

可观测性控制面板分为几个选项卡。

摘要

运行状况和性能摘要列出了不同类别项目的数量。每个数字都充当指向可观测性控制面板某个位置的超链接，其中包含该类别的列表。

集群运行状况

集群运行状况提供了需要注意的重要通知，您可能需要对其中部分通知尽快采取措施。通过此列表，您可以查看描述和受影响的资源。集群运行状况包括两个表：运行状况问题和配置见解。要刷新运行状况问题的状态，请选择刷新按钮（↻）。配置见解每 24 小时自动更新一次，无法手动刷新。

有关运行状况问题的更多信息，请参阅the section called “集群运行状况常见问题解答和错误代码以及解析路径”。有关配置见解的更多信息，请参阅the section called “集群见解”。

控制面板监控

控制面板监控选项卡分为三个部分，每个部分均可帮助您监控集群的控制面板并对其进行故障排除。

指标

对于 Kubernetes 1.28 及更高版本的集群，指标部分会显示为各种控制面板组件收集的多个指标的图表。

您可以通过在该部分的顶部进行选择来设置每个图表 X 轴所用的时间段。您可以使用刷新按钮（↻）刷新数据。对于每个单独的图表，垂直省略号按钮（⋮）会打开一个菜单，其中包含来自 CloudWatch 的选项。

这些指标以及更多指标会自动作为 AWS/EKS 命名空间下的 CloudWatch 中的基本监控指标提供。有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的基本监控和详细监控。要获取更多详细的指标、可视化和洞察，请参阅《Amazon CloudWatch 用户指南》中的 Container Insights。如果更喜欢基于 Prometheus 的监控，请参阅 the section called “Prometheus 指标”。

下表列出了可用指标。

指标	说明
APIServer 请求	每分钟向 API 服务器发出的请求。
APIServer 请求总数 4XX	发生 HTTP 4XX 响应代码（客户端错误）的 API 服务器请求数。
APIServer 请求总数 5XX	发生 HTTP 5XX 响应代码（服务器端错误）的 API 服务器请求数。
APIServer 请求总数 429	发生 HTTP 429 响应代码（请求过多）的 API 服务器请求数。

指标	说明
存储大小	存储数据库 (etcd) 的大小。
调度器尝试	尝试按结果“不可调度”、“错误”和“已调度”调度容器组（pod）的次数。
待处理容器组（pod）	按队列类型“活跃”、“退避”、“不可调度”和“受限”分列的待处理容器组（pod）数量。
API 服务器请求延迟	API 服务器请求的延迟。
API 服务器当前的传输中请求	当前对 API 服务器的传输中请求。
Webhook 请求	每分钟 Webhook 请求数。
Webhook 请求拒绝	被拒绝的 Webhook 请求数。
Webhook 请求延迟 P99	外部第三方 Webhook 请求的第 99 个百分位延迟。

CloudWatch Log Insights

CloudWatch Log Insights 部分根据控制面板审计日志显示各种列表。需要打开 Amazon EKS 控制面板日志才能使用此功能，您可以在查看 CloudWatch 中的控制面板日志部分执行此操作。

经过足够的时间来收集数据后，您可以运行所有查询，也可以选择运行查询一次获取单个列表。每当您运行查询时，CloudWatch 均会产生额外费用。选择要在该部分顶部查看结果的时间段。如果您想对任何查询进行更高级的控制，可以选择在 CloudWatch 中查看。这将允许您根据需求更新 CloudWatch 中的查询。

有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的使用 CloudWatch Logs Insights 分析日志数据。

在 CloudWatch 中查看控制面板日志

选择管理日志记录即可更新可用的日志类型。启用日志记录后，日志需要几分钟才能显示在 CloudWatch Logs 中。经过足够的时间，选择本节中的任何查看链接即可导航到适用的日志。

有关更多信息，请参阅 the section called “控制面板日志”。

集群见解

升级洞察表既显示了问题又建议了纠正措施，从而加快了升级到新版 Kubernetes 的验证过程。Amazon EKS 会根据可能影响 Kubernetes 版本升级的问题列表自动扫描集群。升级洞察表列出了 Amazon EKS 针对该集群执行的洞察检查及其相关状态。

根据对 Kubernetes 项目变更以及与新版本相关的 Amazon EKS 服务变更的评估，Amazon EKS 维护并定期刷新待执行的洞察检查列表。Amazon EKS 控制台自动刷新每个洞察的状态，这可以在上次刷新时间列看到。

有关更多信息，请参阅 the section called “集群见解”。

节点运行状况问题

Amazon EKS 节点监控代理会自动读取节点日志以检测运行状况问题。无论使用哪种自动修复设置，节点监控代理都会报告所有节点运行状况问题，以便您可以根据需要进行调查。如果列出的问题类型没有描述，则可以在其弹出框元素中阅读描述。

刷新页面后，所有已解决的问题都将从列表中消失。如果启用了自动修复，则可能会暂时看到某些无需您执行任何操作即可解决的运行状况问题。不支持自动修复的问题可能需要您手动操作，具体取决于其类型。

要报告节点运行状况问题，集群必须使用 Amazon EKS 自动模式或安装了节点监控代理附加组件。有关更多信息，请参阅 the section called “节点运行状况”。

EKS 功能

功能部分显示集群中 EKS 功能资源的状态和运行状况。可以在此处监控集群中这些功能及其托管式 Kubernetes 资源的运行状况和状态通知。刷新页面后，所有已解决的问题都将从列表中消失。

有关更多信息，请参阅 the section called “使用功能”。

利用容器网络可观测性监控 Kubernetes 工作负载流量

Amazon EKS 提供了增强型网络可观测性功能，可让您更深入地了解容器网络环境。这些功能可帮助您更好地了解、监控 Amazon 中的 Kubernetes 网络格局并对其进行问题排查。借助增强型容器网络可观测性，您可以利用与网络相关的精细指标，更好地主动检测集群流量、跨可用区流量和 Amazon 服务的异常。通过这些指标，您可以使用首选的可观测性堆栈来衡量系统性能并可视化基础指标。

此外，Amazon EKS 现在在 Amazon 控制台中提供网络监控可视化功能，可加快和增强准确的问题排查，从而更快地分析根本原因。您还可以利用这些视觉功能来精确定位导致重新传输和重新传输超时的主要贡献者和网络流量，从而消除事件期间的盲点。

这些功能由 Amazon CloudWatch 网络流量监测仪启用。

使用案例

衡量网络性能以检测异常

一些团队对可观测性堆栈进行了标准化，该堆栈允许他们衡量系统的性能，可视化系统指标，并在超出特定阈值时收到警报。EKS 中的容器网络可观测性与此保持一致，公开了关键的系统指标，您可以抓取这些指标来扩大对系统在容器组（pod）和 Worker 节点级别的网络性能可观测性。

利用控制台可视化进行更准确的问题排查

如果监控系统发出警报，您可能需要仔细研究问题来源所在的集群和工作负载。为了支持此操作，您可以利用 EKS 控制台中的可视化来缩小集群级别的调查范围，并加速披露导致重新传输次数最多的网络流量、重新传输超时和已传输数据量。

在 Amazon EKS 环境中跟踪主要贡献者

许多团队都将 EKS 作为其平台的基础来运行，使其成为应用程序环境网络活动的焦点。使用此功能中的网络监控功能，可以跟踪哪些工作负载负责集群内、跨可用区的大多数流量（按数据量衡量），以及流向 Amazon 内（DynamoDB 和 S3）和 Amazon 云外（互联网或本地）外部目标的流量。此外，您可以根据重新传输次数、重新传输超时和已传输数据量监控这些流量的性能。

功能

1. 性能指标：此功能允许您直接从 EKS 集群中运行的网络流量监测仪（NFM）代理中获取容器组（pod）和 Worker 节点的网络相关系统指标。
2. 服务地图：此功能可动态可视化集群中工作负载之间的互通，允许您快速披露与通信容器组（pod）之间的网络流量相关的关键指标（重新传输次数 – RT、重新传输超时 – RTO 和已传输数据量 – DT）。
3. 流表：使用此表，可以从三个不同角度监控集群中 Kubernetes 工作负载中的主要贡献者：Amazon 服务视图、集群视图和外部视图。对于每个视图，可以查看重新传输次数、重新传输超时以及源容器组（pod）与其目标之间传输的数据量。
   - Amazon 服务视图：显示 Amazon 服务（DynamoDB 和 S3）的主要贡献者
   - 集群视图：显示集群内的主要贡献者（东 ← 到 → 西）

- 外部视图：向 Amazon 外部的外部集群目标显示主要贡献者

开始使用

要开始使用，请在 EKS 控制台中为新集群或现有集群启用容器网络可观测性功能。这将自动创建网络流量监测仪（NFM）依赖项（范围和监视器资源）。此外，还必须安装网络流量监测仪代理附加组件。或者，可以使用 Amazon CLI、EKS API（用于附加组件）、NFM API 或基础设施即代码（如 Terraform）来安装这些依赖项。这些依赖项安装到位后，便可配置首选的监控工具，以从 NFM 代理中获取容器组（pod）和 Worker 节点的网络性能指标。要可视化工作负载的网络活动和性能，可以导航到集群可观测性控制面板的“网络”选项卡下的 EKS 控制台。

在 EKS 中使用网络流量监测仪时，可以维护现有的可观测性工作流程和技术堆栈，同时利用一系列其他功能，进一步了解和优化 EKS 环境的网络层。您可以在此处了解有关网络流量监测仪定价的更多信息。

先决条件和重要提示

1. 如上所述，如果您从 EKS 控制台启用容器网络可观测性，则系统将代表您自动创建基础 NFM 资源依赖项（范围和监视器），并指导您完成适用于 NFM 的 EKS 附加组件的安装过程。
2. 如果要使用 Terraform 之类的基础设施即代码（IaC）启用此功能，则必须在 IaC 中定义以下依赖项：NFM 范围、NFM 监视器、适用于 NFM 的 EKS 附加组件。此外，必须使用容器组身份或 服务账户的 IAM 角色（IRSA）向 EKS 附加组件授予相关权限。
3. NFM 代理的 EKS 附加组件运行的最低版本必须为 1.1.0。
4. 您必须使用 Terraform Amazon Provider 的 v6.21.0 或更高版本才能支持网络流量监测仪资源。

所需的 IAM 权限

适用于 NFM 代理的 EKS 附加组件

可将 `CloudWatchNetworkFlowMonitorAgentPublishPolicy` Amazon 托管式策略与容器组身份结合使用。此策略包含 NFM 代理向网络流量监测仪端点发送遥测报告（指标）所需的权限。

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
```

```
        "networkflowmonitor:Publish"
      ],
      "Resource" : "*"
    }
  ]
}
```

EKS 控制台中的容器网络可观测性

需要以下权限才能启用该功能并在控制台中可视化服务地图和流表。

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect": "Allow",
      "Action": [
        "networkflowmonitor:ListScopes",
        "networkflowmonitor:ListMonitors",
        "networkflowmonitor:GetScope",
        "networkflowmonitor:GetMonitor",
        "networkflowmonitor:CreateScope",
        "networkflowmonitor:CreateMonitor",
        "networkflowmonitor:TagResource",
        "networkflowmonitor:StartQueryMonitorTopContributors",
        "networkflowmonitor:StopQueryMonitorTopContributors",
        "networkflowmonitor:GetQueryStatusMonitorTopContributors",
        "networkflowmonitor:GetQueryResultsMonitorTopContributors"
      ],
      "Resource": "*"
    }
  ]
}
```

使用 Amazon CLI、EKS API 和 NFM API

```
#!/bin/bash

# Script to create required Network Flow Monitor resources
set -e

CLUSTER_NAME="my-eks-cluster"
CLUSTER_ARN="arn:aws:eks:{Region}:{Account}:cluster/{ClusterName}"
```

```
REGION="us-west-2"
AGENT_NAMESPACE="amazon-network-flow-monitor"

echo "Creating Network Flow Monitor resources..."

# Check if Network Flow Monitor agent is running in the cluster
echo "Checking for Network Flow Monitor agent in cluster..."
if kubectl get pods -n "$AGENT_NAMESPACE" --no-headers 2>/dev/null | grep -q "Running";
 then
    echo "Network Flow Monitor agent exists and is running in the cluster"
else
    echo "Network Flow Monitor agent not found. Installing as EKS addon..."
    aws eks create-addon \
        --cluster-name "$CLUSTER_NAME" \
        --addon-name "$AGENT_NAMESPACE" \
        --region "$REGION"
    echo "Network Flow Monitor addon installation initiated"
fi

# Get Account ID
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

echo "Cluster ARN: $CLUSTER_ARN"
echo "Account ID: $ACCOUNT_ID"

# Check for existing scope
echo "Checking for existing Network Flow Monitor Scope..."
EXISTING_SCOPE=$(aws networkflowmonitor list-scopes --region $REGION --query
 'scopes[0].scopeArn' --output text 2>/dev/null || echo "None")

if [ "$EXISTING_SCOPE" != "None" ] && [ "$EXISTING_SCOPE" != "null" ]; then
    echo "Using existing scope: $EXISTING_SCOPE"
    SCOPE_ARN=$EXISTING_SCOPE
else
    echo "Creating new Network Flow Monitor Scope..."
    SCOPE_RESPONSE=$(aws networkflowmonitor create-scope \
        --targets "[{\"targetIdentifier\":{\"targetId\":{\"accountId\":
\"${ACCOUNT_ID}\"},\"targetType\":\"ACCOUNT\"},\"region\":\"${REGION}\"}]" \
        --region $REGION \
        --output json)

    SCOPE_ARN=$(echo $SCOPE_RESPONSE | jq -r '.scopeArn')
    echo "Scope created: $SCOPE_ARN"
fi
```

```
# Create Network Flow Monitor with EKS Cluster as local resource
echo "Creating Network Flow Monitor..."
MONITOR_RESPONSE=$(aws networkflowmonitor create-monitor \
    --monitor-name "${CLUSTER_NAME}-monitor" \
    --local-resources "type=AWS::EKS::Cluster,identifier=${CLUSTER_ARN}" \
    --scope-arn "$SCOPE_ARN" \
    --region $REGION \
    --output json)

MONITOR_ARN=$(echo $MONITOR_RESPONSE | jq -r '.monitorArn')

echo "Monitor created: $MONITOR_ARN"

echo "Network Flow Monitor setup complete!"
echo "Monitor ARN: $MONITOR_ARN"
echo "Scope ARN: $SCOPE_ARN"
echo "Local Resource: AWS::EKS::Cluster (${CLUSTER_ARN})"
```

使用基础设施即代码（IaC）

Terraform

如果您使用 Terraform 来管理 Amazon 云基础设施，则可以包括以下资源配置，以便为集群启用容器网络可观测性功能。

NFM 范围

```
data "aws_caller_identity" "current" {}

resource "aws_networkflowmonitor_scope" "example" {
  target {
    region = "us-east-1"
    target_identifier {
      target_type = "ACCOUNT"
      target_id {
        account_id = data.aws_caller_identity.current.account_id
      }
    }
  }

  tags = {
    Name = "example"
```

```
  }
}
```

NFM 监视器

```
resource "aws_networkflowmonitor_monitor" "example" {
  monitor_name = "eks-cluster-name-monitor"
  scope_arn    = aws_networkflowmonitor_scope.example.scope_arn

  local_resource {
    type       = "AWS::EKS::Cluster"
    identifier = aws_eks_cluster.example.arn
  }

  remote_resource {
    type       = "AWS::Region"
    identifier = "us-east-1" # this must be the same region that the cluster is in
  }

  tags = {
    Name = "example"
  }
}
```

适用于 NFM 的 EKS 附加组件

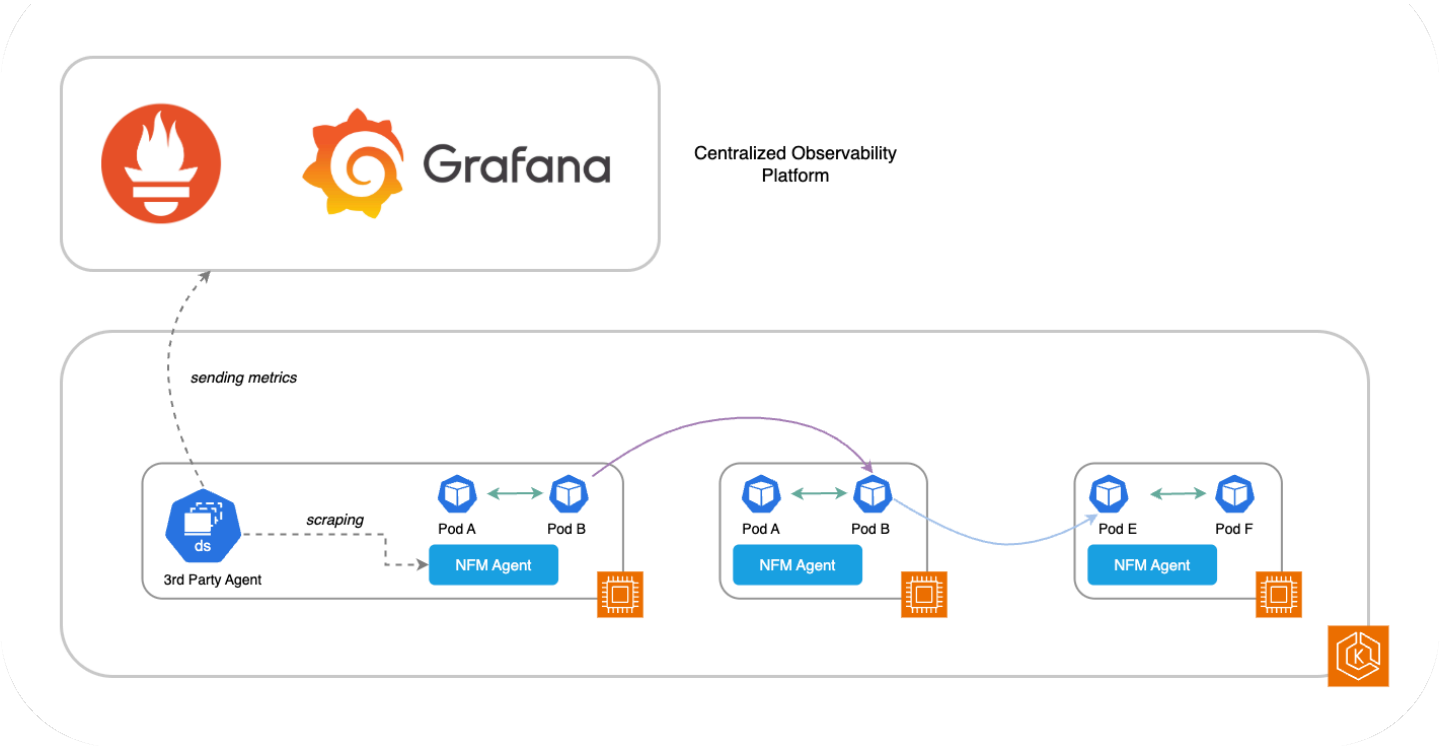
```
resource "aws_eks_addon" "example" {
  cluster_name                = aws_eks_cluster.example.name
  addon_name                  = "aws-network-flow-monitoring-agent"
}
```

如何工作？

性能指标

系统指标

如果您正在运行第三方（3P）工具来监控 EKS 环境（例如 Prometheus 和 Grafana），则可以直接从网络流量监测仪代理抓取支持的系统指标。这些指标可以发送到监控堆栈，以扩展对系统在容器组（pod）和 Worker 节点级别的网络性能的衡量范围。可用指标列于表格的“支持的系统指标”下。



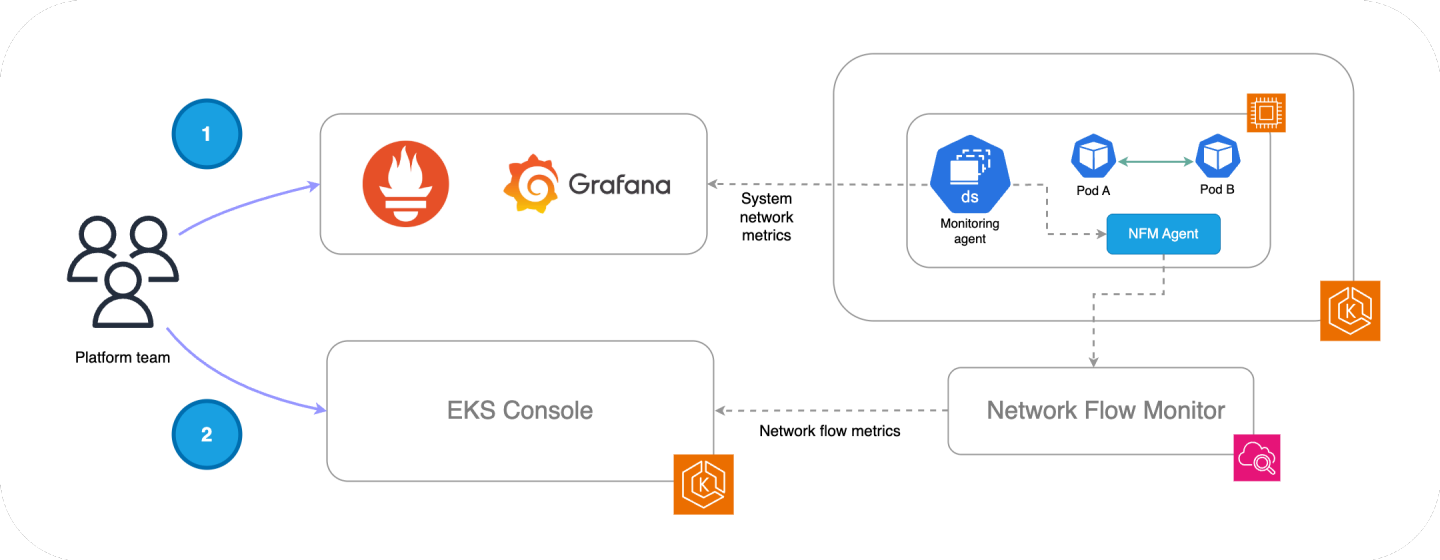
要启用这些指标，请在安装过程中使用配置变量覆盖以下环境变量（请参阅：https://aws.amazon.com/blogs/containers/amazon-eks-add-ons-advanced-configuration/）：

```
OPEN_METRICS:
    Enable or disable open metrics. Disabled if not supplied
    Type: String
    Values: [“on”, “off”]
OPEN_METRICS_ADDRESS:
    Listening IP address for open metrics endpoint. Defaults to 127.0.0.1 if not
 supplied
    Type: String
OPEN_METRICS_PORT:
    Listening port for open metrics endpoint. Defaults to 80 if not supplied
    Type: Integer
    Range: [0..65535]
```

流量级别指标

此外，网络流量监测仪会捕获网络流量数据以及流量级别指标：重新传输次数、重新传输超时和已传输数据量。这些数据由网络流量监测仪处理，并在 EKS 控制台中进行可视化，以显示集群环境中的流量，以及基于这些流量级别指标的表现。

下图描绘了一个工作流程，在该工作流程中，可以利用两种类型的指标（系统和流量级别）来获得更多的运营情报。



1. 平台团队可以在其监控堆栈中收集和可视化系统指标。有了警报，他们就可以使用 NFM 代理中的系统指标来检测网络异常或影响容器组（pod）或 Worker 节点的问题。
2. 下一步，平台团队可以利用 EKS 控制台中的原生可视化来进一步缩小调查范围，并根据流表示及其相关指标加快问题排查。

重要说明：从 NFM 代理抓取系统指标和 NFM 代理将流量级别指标推送到 NFM 后端的过程是独立的过程。

支持的系统指标

重要说明：系统指标以 OpenMetrics 格式导出。

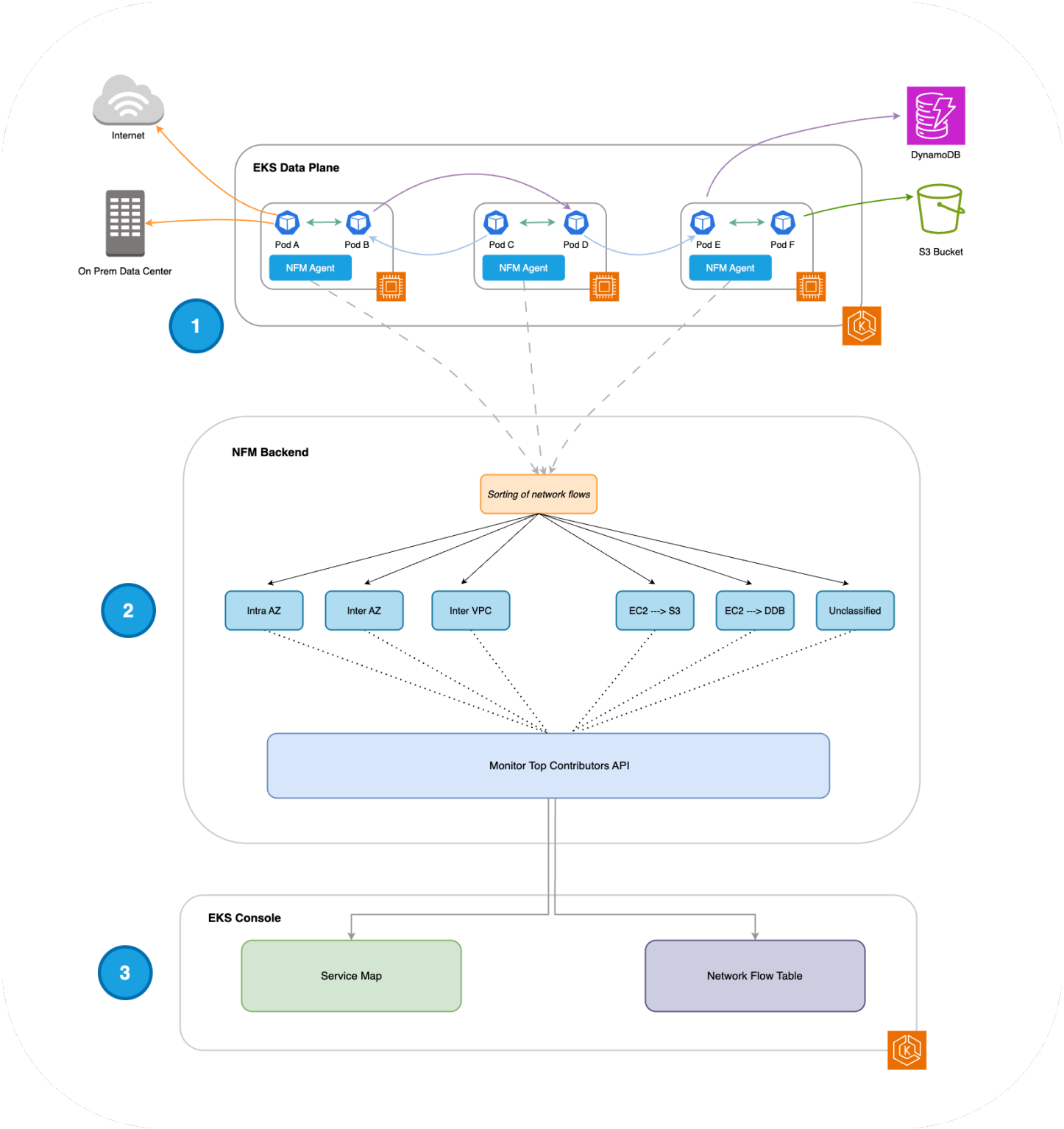
指标名称	类型	Dimensions	说明
ingress_flow	计量表	instance_id、iface、pod、namespace、node	入口 TCP 流计数（tcpAssiveOpens）
egress_flow	计量表	instance_id、iface、pod、namespace、node	出口 TCP 流计数（tcpActiveOpens）

指标名称	类型	Dimensions	说明
ingress_packets	计量表	instance_id、iface、pod、namespace、node	入口数据包计数（增量）
egress_packets	计量表	instance_id、iface、pod、namespace、node	出口数据包计数（增量）
ingress_bytes	计量表	instance_id、iface、pod、namespace、node	入口字节数（增量）
egress_bytes	计量表	instance_id、iface、pod、namespace、node	出口字节计数（增量）
bw_in_lowance_exceede	计量表	instance_id、eni、node	由于入站带宽限制，数据包已排队/已丢弃
bw_out_allowance_exceeded	计量表	instance_id、eni、node	由于出站带宽限制，数据包已排队/已丢弃
pps_allowance_exceeded	计量表	instance_id、eni、node	由于双向 PPS 限制，数据包已排队/已丢弃
conntrack_allowance_exceeded	计量表	instance_id、eni、node	由于连接跟踪限制，数据包已丢弃
linklocal_allowance_exceeded	计量表	instance_id、eni、node	由于本地代理服务 PPS 限制，数据包已丢弃

支持的流量级别指标

指标名称	类型	说明
TCP 重新传输次数	计数器	发送端重新发送在传输过程中丢失或损坏的数据包的次数。
TCP 重新传输超时	计数器	发送端启动等待期以确定数据包是否在传输过程中丢失的次数。
已传输数据量（字节数）	计数器	给定流量在源和目标之间传输的数据量。

服务地图和流表



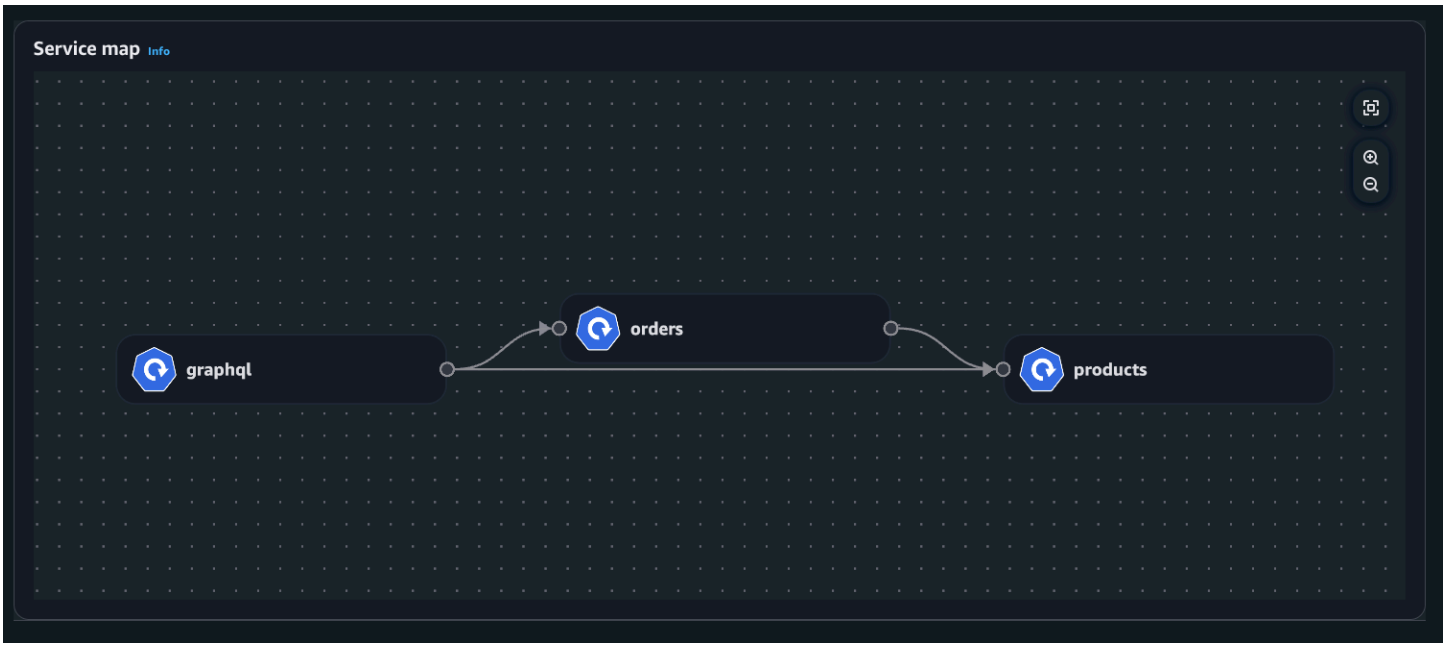
1. 安装后，网络流量监测仪代理将作为 DaemonSet 在每个 Worker 节点上运行，并每 30 秒收集排名前 500 的网络流（基于已传输数据量）。

2. 这些网络流分为以下类别：可用区内、可用区间、EC2 → S3、EC2 → DynamoDB（DDB）和未分类。每个流都有 3 个与之关联的指标：重新传输次数、重新传输超时和已传输数据量（以字节为单位）。
   - 可用区内部：同一可用区中容器组（pod）之间的网络流
   - 可用区间：不同可用区中容器组（pod）之间的网络流
   - EC2 → S3：从容器组（pod）流向 S3 的网络流
   - EC2 → DDB：从容器组（pod）流向 DDB 的网络流
   - 未分类：从容器组（pod）流向互联网或本地的网络流
3. 来自网络流量监测仪 Top Contributors API 的网络流用于在 EKS 控制台中支持以下体验：
   - 服务地图：集群内网络流的可视化（可用区内和可用区间）。
   - 流表：集群内（可用区内和可用区间）、从容器组（pod）到 Amazon 服务（EC2 → S3 和 EC2 → DDB）以及从容器组（pod）到外部目标（未分类）的网络流的表格呈现。

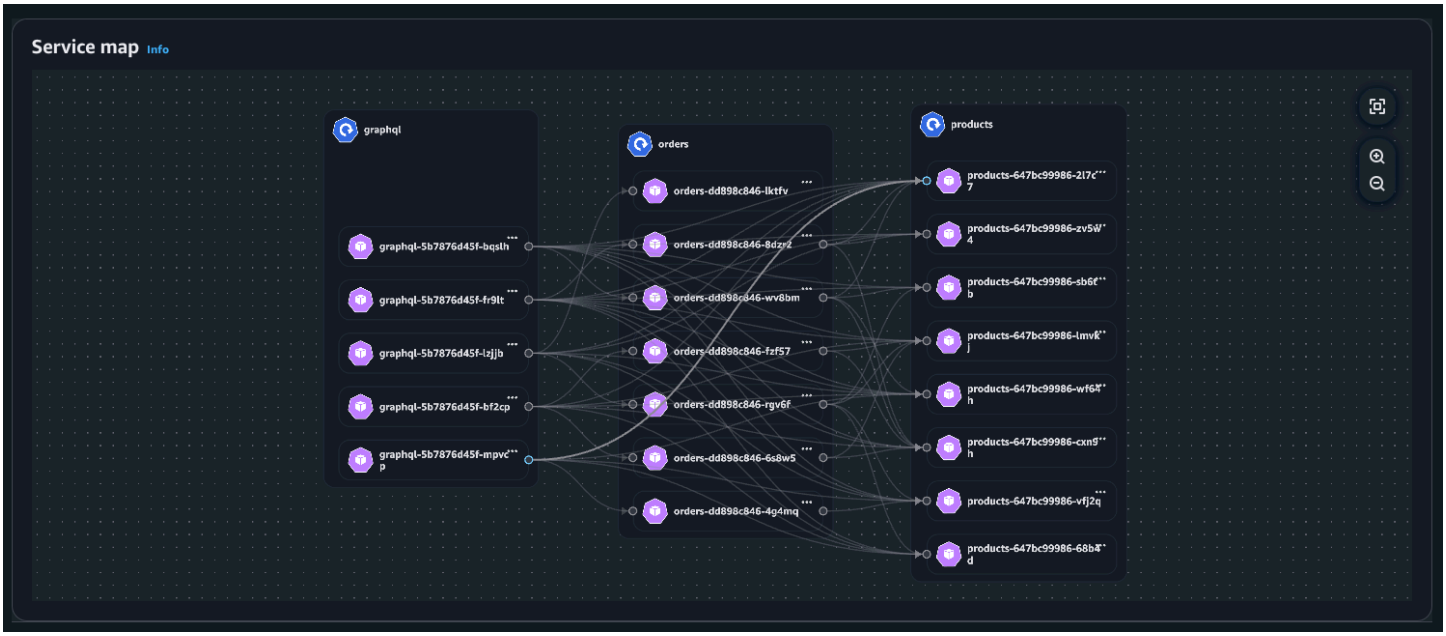
从 Top Contributors API 提取的网络流限定为 1 小时的时间范围，并且每个类别最多可以包含 500 个流。对于服务地图，这意味着在 1 小时的时间范围内可以从可用区内和可用区间流类别中获取和呈现多达 1000 个流。对于流表，这意味着在 2 小时的时间范围内可以从所有 6 个网络流类别中获取和呈现多达 3000 个网络流。

示例：服务地图

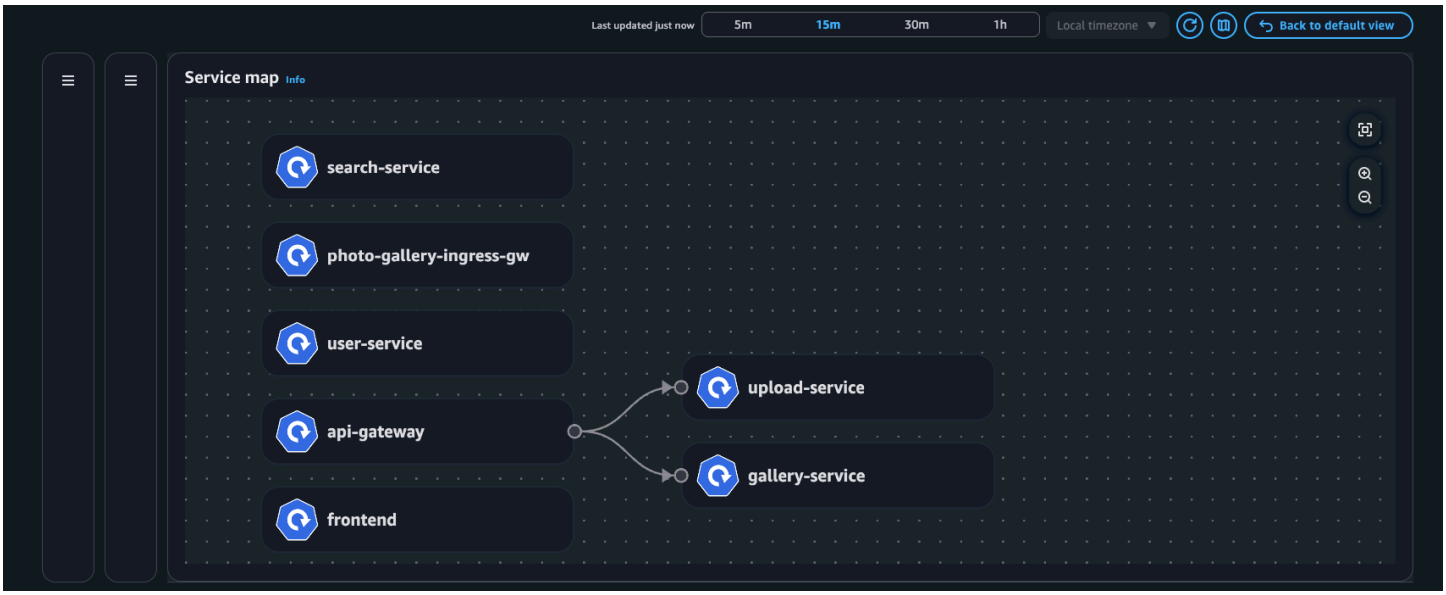
部署视图



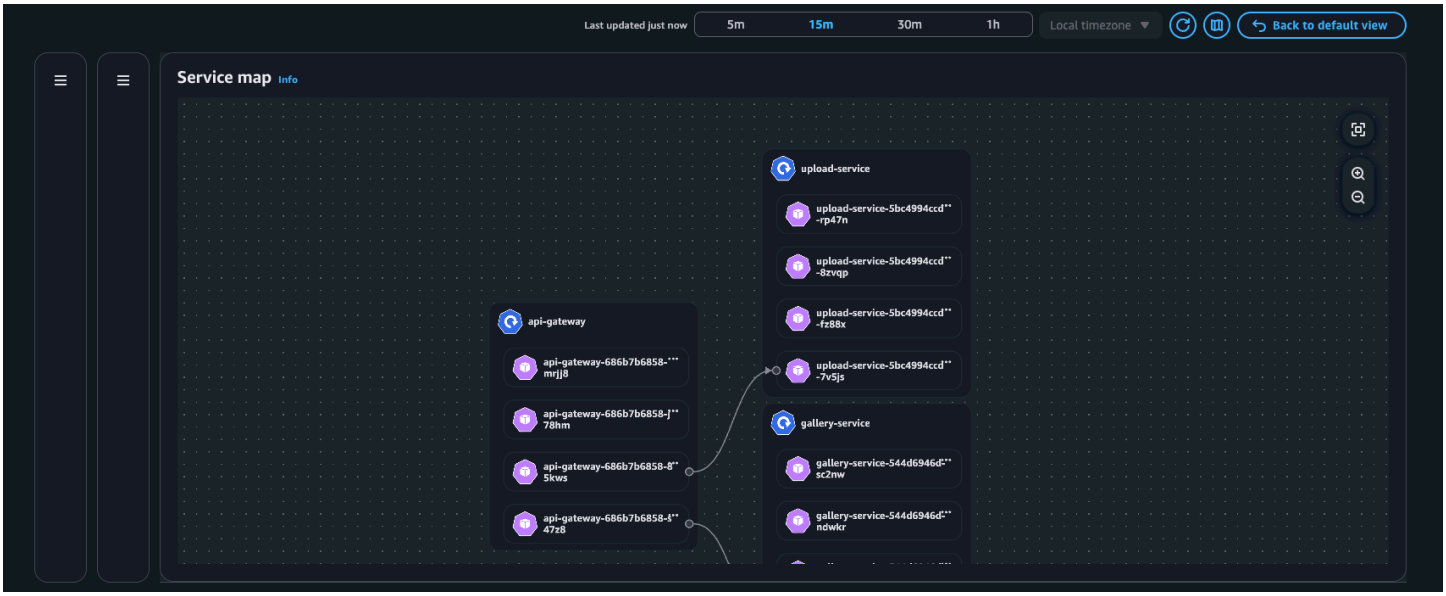
容器组（pod）视图



部署视图

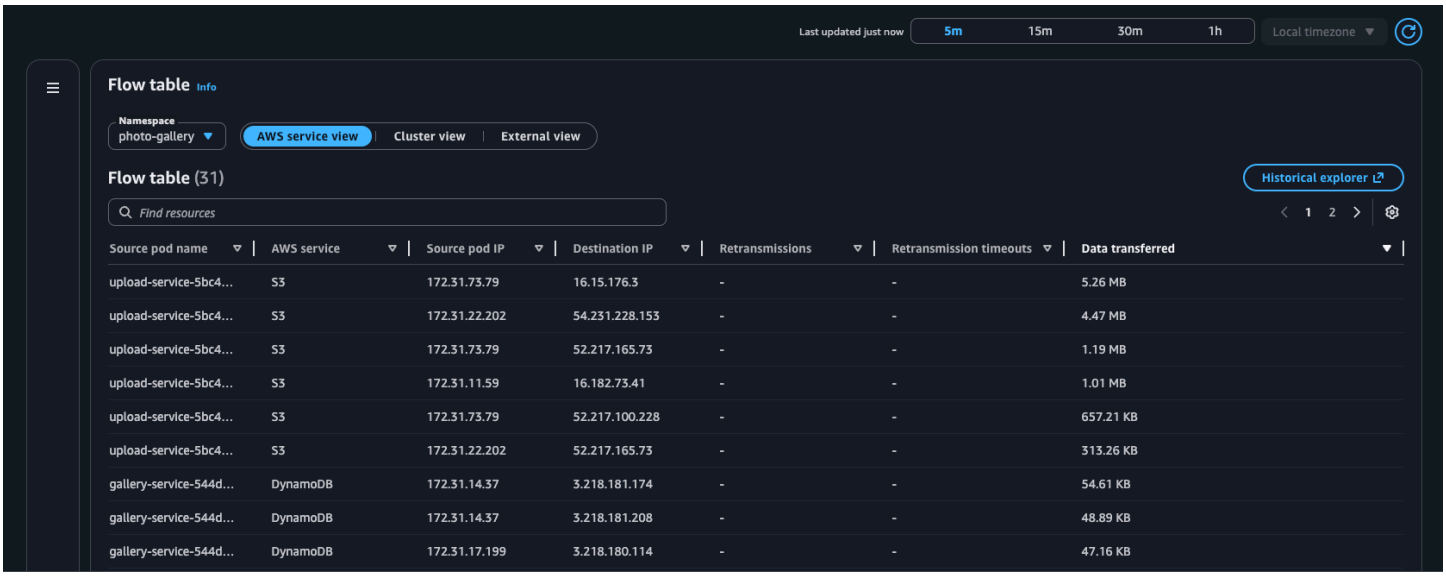


容器组（pod）视图

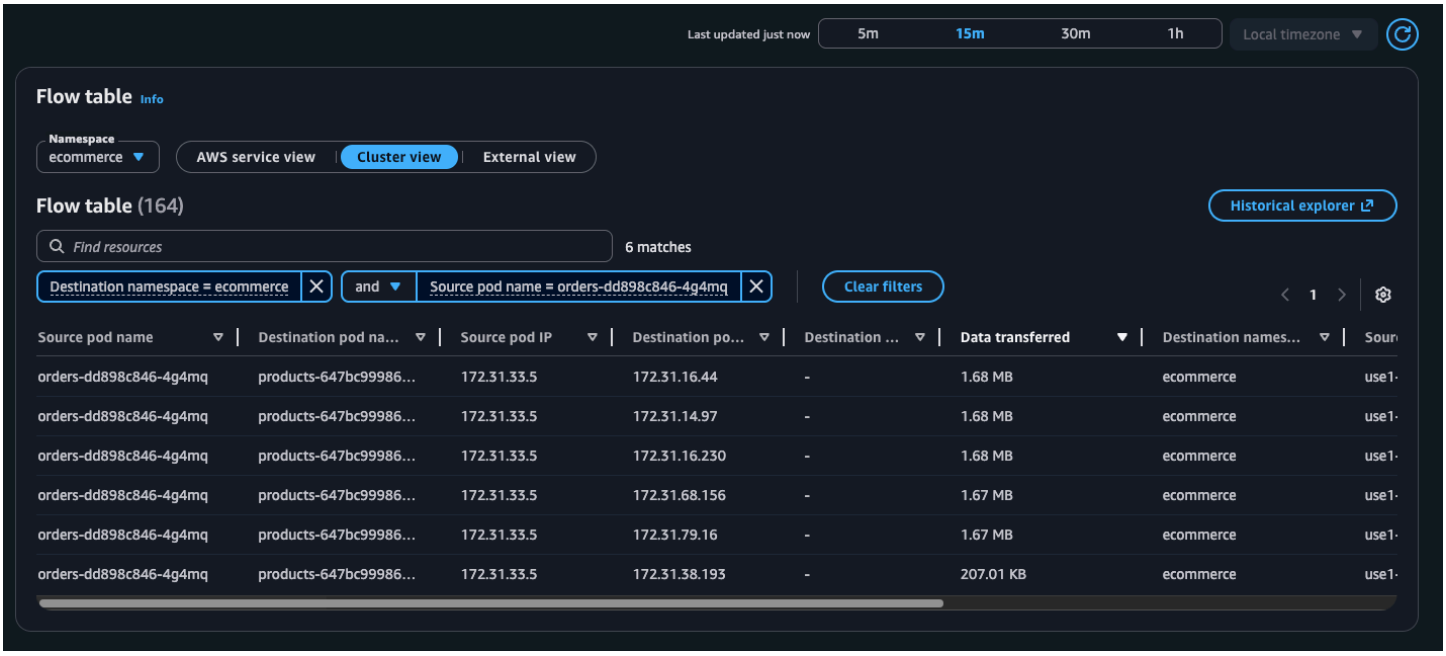


示例：流表

Amazon 服务视图



集群视图



注意事项和限制

- EKS 中的容器网络可观测性功能仅在支持网络流量监测仪的区域中提供。
- 支持的系统指标采用 OpenMetrics 格式，可以直接从网络流量监测仪（NFM）代理中抓取。
- 要使用 Terraform 之类的基础设施即代码（IaC）在 EKS 中启用容器网络可观测性功能，您需要在配置中定义和创建这些依赖项：NFM 范围、NFM 监视器和 NFM 代理。
- 网络流量监测仪每分钟最多支持约 500 万个流。这是安装了网络流量监测仪代理的大约 5000 个 EC2 实例（EKS Worker 节点）。在有更多容量可用之前，在超过 5000 个实例上安装代理可能会影响监控性能。
- NFM 代理的 EKS 附加组件运行的最低版本必须为 1.1.0。
- 您必须使用 Terraform Amazon Provider 的 v6.21.0 或更高版本才能支持网络流量监测仪资源。
- 要使用容器组（pod）元数据丰富网络流，您的容器组（pod）应在自己的隔离网络命名空间中运行，而不是在主机网络命名空间中运行。

使用 Prometheus 监控集群指标

Prometheus 是一个用于抓取端点的监控和时间序列数据库。提供查询、聚合和存储收集的数据的功能。您还可以将其用于警报和警报聚合。本主题介绍如何将 Prometheus 设置为托管或开源选项。一个常见用例为监控 Amazon EKS 控制面板指标。

Amazon Managed Service for Prometheus 是一项与 Prometheus 兼容的监控和警报服务，可以轻松实现对容器化应用程序和基础设施的大规模监控。这是一项完全托管的服务，可自动扩展指标的提取、存储、查询和警报。这项服务还集成了 Amazon 安全服务，可以快速安全地访问您的数据。您可以使用开源 PromQL 查询语言来查询指标并发出警报。此外，您还可以使用 Amazon Managed Service for Prometheus 中的警报管理器来设置关键警报的提醒规则。然后，您可以将这些关键警报作为通知发送到某个 Amazon SNS 主题。

可以通过多种方式将 Prometheus 与 Amazon EKS 结合使用：

- 您可以在首次创建 Amazon EKS 集群时开启 Prometheus 指标，也可以为现有集群创建自己的 Prometheus 抓取程序。本主题涵盖了这两个选项。
- 您可以使用 Helm 部署 Prometheus。有关更多信息，请参阅 the section called “使用 Helm 部署”。
- 您可以查看 Prometheus 格式的控制面板原始指标。有关更多信息，请参阅 the section called “控制面板”。

第 1 步：开启 Prometheus 指标

> **Important**
>
> 适用于 Prometheus 的 Amazon 托管服务资源不在集群生命周期内，需要独立于集群进行维护。删除集群时，请务必同时删除所有适用的抓取器以停止适用的费用。有关更多信息，请参阅《Amazon Managed Service for Prometheus 用户指南》中的查找和删除抓取程序。

Prometheus 通过一个名为抓取的拉取模型从集群中发现和收集指标。设置抓取程序以从您的集群基础设施和容器化应用程序中收集数据。当您开启发送 Prometheus 指标的选项时，Amazon Managed Service for Prometheus 会提供一个完全托管的无代理抓取程序。

如果尚未创建集群，您可以在首次创建集群时开启向 Prometheus 发送指标的选项。在 Amazon EKS 控制台中，此选项位于创建新集群的配置可观测性步骤中。有关更多信息，请参阅 the section called “创建集群”。

如果您已经有现有的集群，则可以创建自己的 Prometheus 抓取程序。要在 Amazon EKS 控制台中执行此操作，请导航到集群的可观测性选项卡，然后选择添加抓取器按钮。如果您更愿意使用 Amazon API 或 Amazon CLI 执行此操作，请参阅《Amazon Managed Service for Prometheus User Guide》中的 Create a scraper。

使用 Amazon EKS 控制台创建抓取程序时，可以使用以下选项。

抓取程序别名

（可选）输入抓取程序的唯一别名。

目标位置

选择 Amazon Managed Service for Prometheus 工作区。工作区是专用于存储和查询 Prometheus 指标的逻辑空间。借助此工作区，您将能够查看有权访问该工作区账户的 Prometheus 指标。创建新工作区选项指示 Amazon EKS 使用您提供的工作区别名，来代表您创建工作区。您可以使用选择现有工作区选项，从下拉列表中选择一个现有工作区。有关工作区的更多信息，请参阅《Amazon Managed Service for Prometheus 用户指南》中的管理工作区。

服务访问

本节总结了您在发送 Prometheus 指标时授予的权限：

- 允许 Amazon Managed Service for Prometheus 描述抓取的 Amazon EKS 集群
- 允许远程写入 Amazon Managed Prometheus 工作区

如果 `AmazonManagedScraperRole` 已存在，抓取程序会进行使用。选择 `AmazonManagedScraperRole` 链接以查看权限详细信息。如果 `AmazonManagedScraperRole` 尚不存在，请选择查看权限详细信息链接，查看您通过发送 Prometheus 指标授予的特定权限。

子网

根据需要修改抓取程序将继承的子网。如果需要添加灰显的子网选项，请返回创建集群指定网络步骤。

抓取程序配置

根据需要修改 YAML 格式的抓取程序进行配置。为此，请使用表单或上传替换的 YAML 文件。有关更多信息，请参阅《Amazon Managed Service for Prometheus 用户指南》中的抓取程序配置。

Amazon Managed Service for Prometheus 指的是与集群一起创建的无代理抓取程序，作为 Amazon 托管的收集器。有关 Amazon 托管的收集器的更多信息，请参阅《Amazon Managed Service for Prometheus 用户指南》中的 使用 Amazon 托管收集器摄取指标。

Important

- 如果您使用 Amazon CLI 或 Amazon API 创建 Prometheus 抓取程序，则需要调整其配置，向抓取程序授予集群内的权限。有关更多信息，请参阅《Amazon Managed Service for Prometheus 用户指南》中的配置您的 Amazon EKS 集群。

- 如果您在 2024 年 11 月 11 日之前创建了使用 `aws-auth ConfigMap` 而非访问条目的 Prometheus 抓取程序，则需要对其进行更新才能从 Amazon EKS 集群控制面板访问其他指标。有关更新的配置，请参阅《Amazon Managed Service for Prometheus 用户指南》中的手动配置 Amazon EKS 以获取抓取工具访问权限。

第 2 步：使用 Prometheus 指标

有关为集群开启 Prometheus 指标后如何使用这些指标的详细信息，请参阅《Amazon Managed Service for Prometheus User Guide》。

第 3 步：管理 Prometheus 抓取程序

要管理抓取程序，请在 Amazon EKS 控制台中选择可观测性选项卡。一个显示集群抓取程序列表的表格，包括抓取程序 ID、别名、状态和创建日期等信息。您可以添加更多抓取程序、编辑抓取程序、删除抓取程序或查看有关当前抓取程序的更多信息。

要查看有关抓取程序的更多详细信息，请选择抓取程序 ID 链接。例如，您可以查看 ARN、环境、工作空间 ID、IAM 角色、配置和网络信息。您可以使用抓取程序 ID 作为 Amazon Managed Service for Prometheus API 操作的输入，例如 `DescribeScraper`、`UpdateScraper` 和 `DeleteScraper`。有关使用 Prometheus API 的更多信息，请参阅 Amazon Managed Service for Prometheus API Reference。

使用 Helm 部署 Prometheus

您可以使用 Helm 将 Prometheus 部署到集群中，而不使用 Amazon Managed Service for Prometheus。如果您已安装有 Helm，则可使用 `helm version` 命令来检查您的版本。Helm 是 Kubernetes 集群的包管理器。有关 Helm 及其安装方法的更多信息，请参阅 the section called “使用 Helm 部署应用程序”。

在为您的 Amazon EKS 集群配置 Helm 后，您可以使用它来部署 Prometheus，步骤如下。

1. 创建 Prometheus 命名空间。

```
kubectl create namespace prometheus
```

2. 添加 `prometheus-community` 图表存储库。

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

3. 部署 Prometheus。

```
helm upgrade -i prometheus prometheus-community/prometheus \
    --namespace prometheus \
    --set alertmanager.persistence.storageClass="gp2" \
    --set server.persistentVolume.storageClass="gp2"
```

> **Note**
>
> 如果您在执行此命令时收到错误 `Error: failed to download "stable/prometheus" (hint: running helm repo update may help)`，请运行 `helm repo update prometheus-community`，然后尝试再次运行第 2 步命令。

如果您收到错误 `Error: rendered manifests contain a resource that already exists`，请运行 `helm uninstall` *`your-release-name`* `-n` *`namespace`* ，然后尝试再次运行第 3 步命令。

4. 确认 `prometheus` 命名空间中的所有容器组（pod）均处于 `READY` 状态。

```
kubectl get pods -n prometheus
```

示例输出如下。

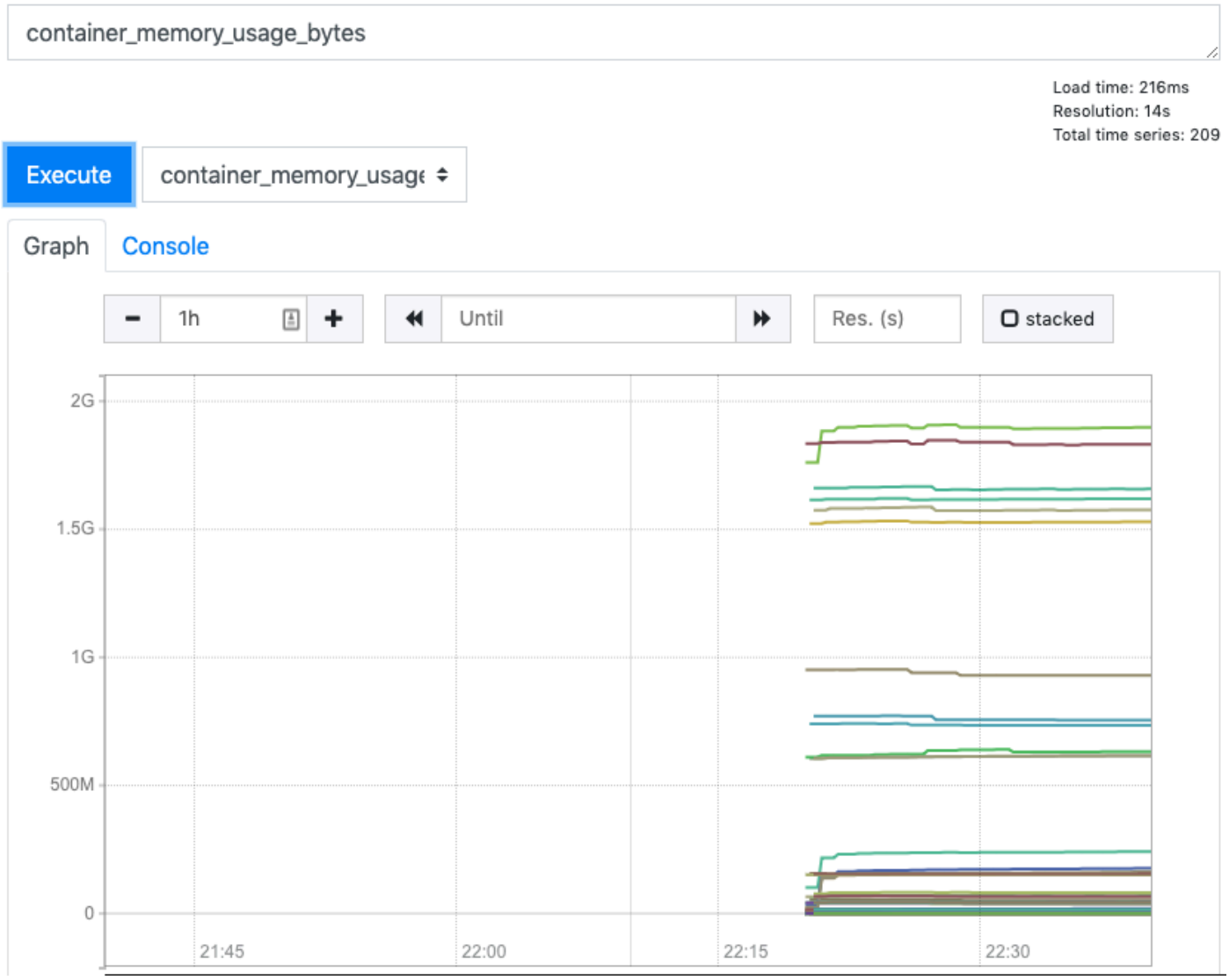
```
NAME                                             READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-59b4c8c744-r7bgp         1/2     Running   0          48s
prometheus-kube-state-metrics-7cfd87cf99-jkz2f   1/1     Running   0          48s
prometheus-node-exporter-jcjqz                   1/1     Running   0          48s
prometheus-node-exporter-jxv2h                   1/1     Running   0          48s
prometheus-node-exporter-vbdks                   1/1     Running   0          48s
prometheus-pushgateway-76c444b68c-82tnw          1/1     Running   0          48s
prometheus-server-775957f748-mmht9               1/2     Running   0          48s
```

5. 使用 `kubectl` 将 Prometheus 控制台端口转发到本地计算机。

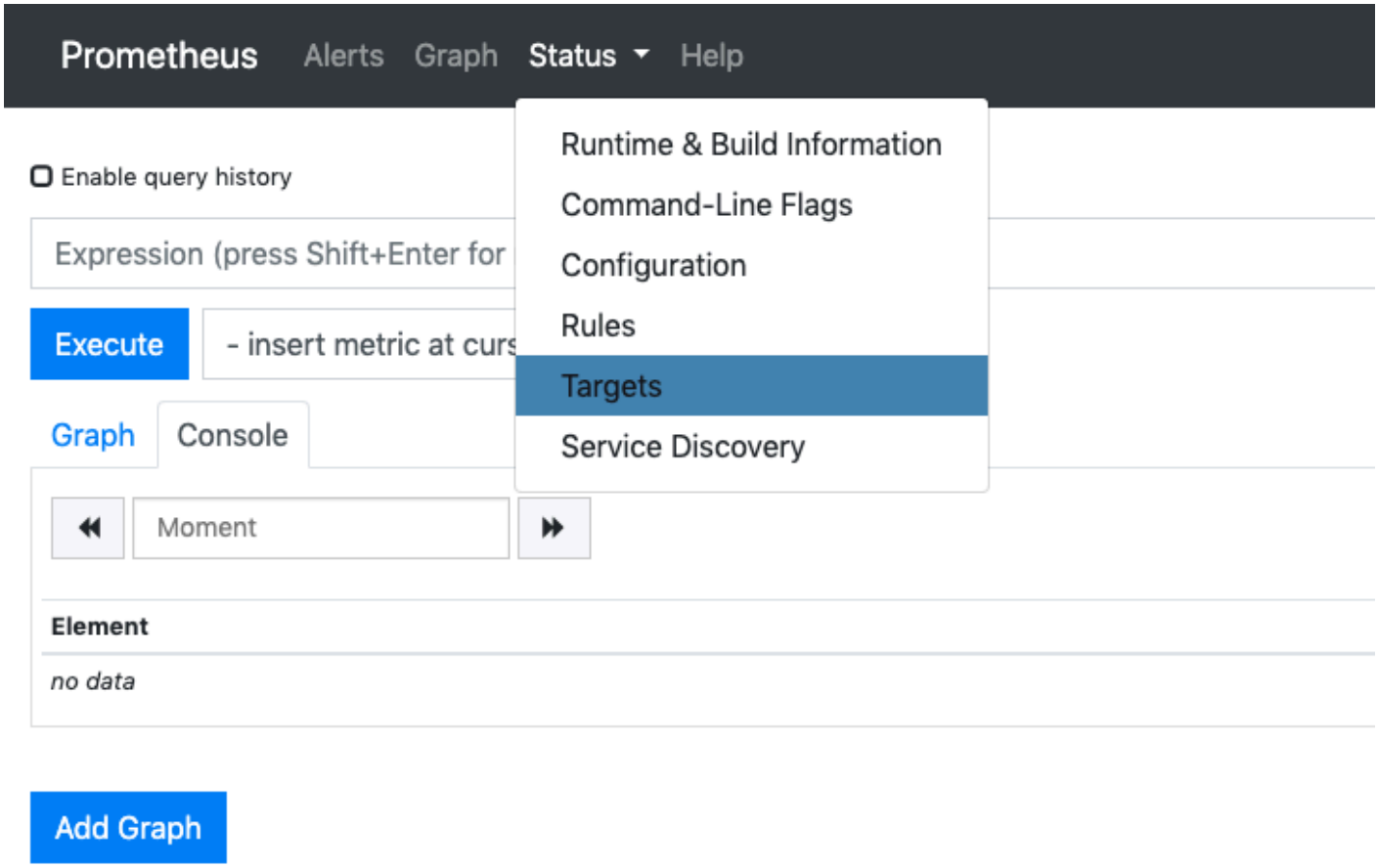
```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

6. 在 Web 浏览器中打开 http://localhost:9090 来查看 Prometheus 控制台。

7. 从 - insert metric at cursor (- 在光标位置插入指标) 菜单选择一个指标，然后选择 Execute (执行)。选择 Graph (图表) 选项卡显示一段时间内的指标。下图显示了一段时间内的 `container_memory_usage_bytes`。



8. 在顶部导航栏中，选择 Status (状态)，然后选择 Targets (目标)。



使用服务发现连接到 Prometheus 的所有 Kubernetes 终端节点将显示。

获取 Prometheus 格式的控制面板原始指标

Kubernetes 控制面板公开了许多以 Prometheus 格式表示的指标。这些指标对于监控和分析非常有用。这些指标通过指标端点在内部公开，无需完全部署 Prometheus 即可访问。但是，随着时间的推移，部署 Prometheus 更加容易，即可实现指标分析。

要查看原始指标输出，请替代 `endpoint` 并运行以下命令。

```
kubectl get --raw endpoint
```

此命令允许您传递任何端点路径，并返回原始响应。输出结果逐行列出了不同指标，每行都包含指标名称、标签和值。

```
metric_name{tag="value"[,...]} value
```

从 API 服务器获取指标

常规 API 服务器端点在 Amazon EKS 控制面板上公开。此端点主要用于查看特定指标。

```
kubectl get --raw /metrics
```

示例输出如下。

```
[...]
# HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
 method, and host.
# TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
# HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
# TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
# HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
# TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

此原始输出逐字返回 API 服务器公开的内容。

使用 `metrics.eks.amazonaws.com` 获取控制面板指标

对于 Kubernetes 版本 `1.28` 及以上版本的集群，Amazon EKS 还会在 API 组 `metrics.eks.amazonaws.com` 下公开指标。这些指标包括控制面板组件，例如 `kube-scheduler` 和 `kube-controller-manager`。

> **Note**
>
> 如果有可能会阻止在集群上新建 `APIService` 资源 `v1.metrics.eks.amazonaws.com` 的 Webhook 配置，则指标端点功能可能不可用。您可以通过在 `kube-apiserver` 审计日志中搜索 `v1.metrics.eks.amazonaws.com` 关键词进行验证。

获取 **kube-scheduler** 指标

使用以下命令来检索 kube-scheduler 指标。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics"
```

示例输出如下。

```
# TYPE scheduler_pending_pods gauge
scheduler_pending_pods{queue="active"} 0
scheduler_pending_pods{queue="backoff"} 0
scheduler_pending_pods{queue="gated"} 0
scheduler_pending_pods{queue="unschedulable"} 18
# HELP scheduler_pod_scheduling_attempts [STABLE] Number of attempts to successfully
 schedule a pod.
# TYPE scheduler_pod_scheduling_attempts histogram
scheduler_pod_scheduling_attempts_bucket{le="1"} 79
scheduler_pod_scheduling_attempts_bucket{le="2"} 79
scheduler_pod_scheduling_attempts_bucket{le="4"} 79
scheduler_pod_scheduling_attempts_bucket{le="8"} 79
scheduler_pod_scheduling_attempts_bucket{le="16"} 79
scheduler_pod_scheduling_attempts_bucket{le="+Inf"} 81
[...]
```

获取 **kube-controller-manager** 指标

使用以下命令来检索 kube-controller-manager 指标。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics"
```

示例输出如下。

```
[...]
workqueue_work_duration_seconds_sum{name="pvprotection"} 0
workqueue_work_duration_seconds_count{name="pvprotection"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-08"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-07"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-05"} 19
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.001"} 109
```

```
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.01"} 139
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.1"} 181
workqueue_work_duration_seconds_bucket{name="replicaset",le="1"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="10"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="+Inf"} 191
workqueue_work_duration_seconds_sum{name="replicaset"} 4.265655885000002
[...]
```

了解调度器和控制器管理器指标

下表描述了可用于 Prometheus 样式获取的调度器和控制器管理器指标。有关这些指标的更多信息，请参阅 Kubernetes 文档中的 Kubernetes Metrics Reference。

指标	控制面板组件	描述
scheduler_pending_pods	调度器	等待调度到节点上执行的容器组（pod）数量。
scheduler_schedule_attempts_total	调度器	尝试调度容器组（pod）的次数。
scheduler_preemption_tapts_total	调度器	调度器尝试通过驱逐优先级较低的容器组（pod）来调度优先级较高的容器组（pod）的次数。
scheduler_preemption_victims	调度器	为容纳优先级更高的容器组（pod）腾出空间而被选中驱逐的容器组（pod）的数量。
scheduler_pod_scheduling_attempts	调度器	成功尝试调度容器组（pod）的次数。
scheduler_scheduling_attempt_duration_seconds	调度器	表示调度器能够根据可用资源和调度规则等各种因素找到容器组（pod）的合适运行位置的速度。
scheduler_pod_scheduling_sli_duration_seconds	调度器	从容器组（pod）进入调度队列之时起调度容器组（pod）的端

指标	控制面板组件	描述
		到端延迟。这可能涉及多次尝试调度。
cronjob_controller_job_creation_skew_duration_skeond	控制器管理器	从计划运行 cronjob 到创建相应作业的时间。
workqueue_depth	控制器管理器	当前工作队列的深度。
workqueue_adds_total	控制器管理器	工作队列处理的添加总数。
workqueue_queue_duration_seconds	控制器管理器	显示项目在被请求之前在工作队列中停留的时长（以秒为单位）。
workqueue_work_duration_seconds	控制器管理器	工作队列中项目的处理时长（以秒为单位）。

部署 Prometheus 抓取工具持续抓取指标

要部署 Prometheus 抓取工具持续抓取指标，请使用以下配置：

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-conf
data:
  prometheus.yml: |-
    global:
      scrape_interval: 30s
    scrape_configs:
    # apiserver metrics
    - job_name: apiserver-metrics
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
```

```
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
            __meta_kubernetes_namespace,
            __meta_kubernetes_service_name,
            __meta_kubernetes_endpoint_port_name,
          ]
        action: keep
        regex: default;kubernetes;https
    # Scheduler metrics
    - job_name: 'ksh-metrics'
      kubernetes_sd_configs:
      - role: endpoints
      metrics_path: /apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
            __meta_kubernetes_namespace,
            __meta_kubernetes_service_name,
            __meta_kubernetes_endpoint_port_name,
          ]
        action: keep
        regex: default;kubernetes;https
    # Controller Manager metrics
    - job_name: 'kcm-metrics'
      kubernetes_sd_configs:
      - role: endpoints
      metrics_path: /apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
            __meta_kubernetes_namespace,
            __meta_kubernetes_service_name,
```

```
            __meta_kubernetes_endpoint_port_name,
          ]
        action: keep
        regex: default;kubernetes;https
---
apiVersion: v1
kind: Pod
metadata:
  name: prom-pod
spec:
  containers:
  - name: prom-container
    image: prom/prometheus
    ports:
    - containerPort: 9090
    volumeMounts:
    - name: config-volume
      mountPath: /etc/prometheus/
  volumes:
  - name: config-volume
    configMap:
      name: prometheus-conf
```

容器组（pod）需要以下权限才能访问新的指标端点。

```
{
  "effect": "allow",
  "apiGroups": [
    "metrics.eks.amazonaws.com"
  ],
  "resources": [
    "kcm/metrics",
    "ksh/metrics"
  ],
  "verbs": [
    "get"
  ] },
```

要修补正在使用的角色，请使用以下命令。

```
kubectl patch clusterrole <role-name> --type=json -p='[
  {
    "op": "add",
```

```
    "path": "/rules/-",
    "value": {
      "verbs": ["get"],
      "apiGroups": ["metrics.eks.amazonaws.com"],
      "resources": ["kcm/metrics", "ksh/metrics"]
    }
  }
]'
```

然后，您可以通过将 Prometheus 抓取程序的端口代理到本地端口来查看 Prometheus 控制面板。

```
kubectl port-forward pods/prom-pod 9090:9090
```

对于您的 Amazon EKS 集群，Kubernetes 控制面板核心指标也将摄取到 AWS/EKS 命名空间下的 Amazon CloudWatch 指标中。要查看这些指标，请打开 CloudWatch 控制台，然后从左侧导航窗格中选择所有指标。在指标选择页面上，为您的集群选择 AWS/EKS 命名空间和指标维度。

使用 Amazon CloudWatch 监控集群数据

Amazon CloudWatch 是一项监控服务，可从您的云资源中收集指标和日志。当使用 1.28 版本及更高版本的新集群时，CloudWatch 会免费提供部分基本的 Amazon EKS 指标。但是，当使用 CloudWatch Observability Operator 作为 Amazon EKS 附加组件时，您可以获得增强的可观测性功能。

Amazon CloudWatch 中的基本指标

如果集群为 Kubernetes 版本 1.28 及以上版本，您可以在 AWS/EKS 命名空间中免费获得 CloudWatch 提供的指标。下表列出了支持版本可用的基本指标。列出的每个指标的频率为一分钟。

指标名称	说明
apiserver_flowcontrol_current_executing_seats	当前用于执行 API 请求的席位数量。席位分配由 Kubernetes API 优先级和公平性功能中的 priority_level 和 flow_schema 配置决定。 单位：计数 有效统计数据：Max
scheduler_schedule_attempts_total	在给定时间段内，调度器尝试调度集群中容器组（pod）的总次数。该指标有助于监控调度器的

指标名称	说明
	工作负载，并且可以指示调度压力或容器组（pod）放置方面的潜在问题。 单位：计数 有效统计数据：Sum
scheduler_schedule_attempts_SCHEDULED	在给定时间段内，调度器尝试将容器组（pod）调度到集群中节点的成功次数。 单位：计数 有效统计数据：Sum
scheduler_schedule_attempts_UNSCHEDULABLE	由于有效限制（例如节点上的 CPU 或内存不足）而在给定时间段内尝试调度不可调度的容器组（pod）的次数。 单位：计数 有效统计数据：Sum
scheduler_schedule_attempts_ERROR	由于调度器本身存在内部问题（例如 API 服务器连接问题）而在给定时间段内尝试调度容器组（pod）的失败次数。 单位：计数 有效统计数据：Sum
scheduler_pending_pods	在给定时间段内，集群中调度器要调度的待处理容器组（pod）总数。 单位：计数 有效统计数据：Sum

指标名称	说明
scheduler_pending_pods_ACTIVEQ	在给定时间段内，activeQ 中在集群中等待调度的待处理容器组（pod）数量。 单位：计数 有效统计数据：Sum
scheduler_pending_pods_UNSCHEDULABLE	调度器尝试调度但失败且处于不可调度状态以便重试的待处理容器组（pod）数量。 单位：计数 有效统计数据：Sum
scheduler_pending_pods_BACKOFF	backoffQ 中处于回退状态且正在等待回退期到期的待处理容器组（pod）数量。 单位：计数 有效统计数据：Sum
scheduler_pending_pods_GATED	当前处于受限状态（必须满足所需条件才能对其进行调度）的待处理容器组（pod）数量。 单位：计数 有效统计数据：Sum
apiserver_request_total	跨集群中所有 API 服务器发出的 HTTP 请求数。 单位：计数 有效统计数据：Sum

指标名称	说明
apiserver_request_total_4XX	向集群中所有 API 服务器发出但导致 4XX（客户端错误）状态代码的 HTTP 请求数。 单位：计数 有效统计数据：Sum
apiserver_request_total_429	向集群中所有 API 服务器发出但导致 429 状态代码的 HTTP 请求数，当客户端超过速率限制阈值时，就会出现这种情况。 单位：计数 有效统计数据：Sum
apiserver_request_total_5XX	向集群中所有 API 服务器发出但导致 5XX（服务器错误）状态代码的 HTTP 请求数。 单位：计数 有效统计数据：Sum
apiserver_request_total_LIST_PODS	向集群中所有 API 服务器发出的 LIST 容器组（pod）请求数。 单位：计数 有效统计数据：Sum
apiserver_request_duration_seconds_PUT_P99	PUT 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的 PUT 请求均在该时间内完成。 单位：秒 有效统计数据：平均值

指标名称	说明
apiserver_request_duration_seconds_PATCH_P99	PATCH 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的 PATCH 请求均在该时间内完成。 单位：秒 有效统计数据：平均值
apiserver_request_duration_seconds_POST_P99	POST 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的 POST 请求均在该时间内完成。 单位：秒 有效统计数据：平均值
apiserver_request_duration_seconds_GET_P99	GET 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的 GET 请求均在该时间内完成。 单位：秒 有效统计数据：平均值
apiserver_request_duration_seconds_LIST_P99	LIST 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的 LIST 请求均在该时间内完成。 单位：秒 有效统计数据：平均值

指标名称	说明
apiserver_request_duration_seconds_DELETE_P99	DELETE 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的 DELETE 请求均在该时间内完成。 单位：秒 有效统计数据：平均值
apiserver_current_inflight_requests_MUTATING	集群中所有 API 服务器当前正在处理的变更请求（POST、PUT、DELETE、PATCH）数。该指标表示正在进行但尚未完成处理的请求。 单位：计数 有效统计数据：Sum
apiserver_current_inflight_requests_READONLY	集群中所有 API 服务器当前正在处理的只读请求（GET、LIST）数。该指标表示正在进行但尚未完成处理的请求。 单位：计数 有效统计数据：Sum
apiserver_admission_webhook_request_total	跨集群中所有 API 服务器发出的准入 Webhook 请求数。 单位：计数 有效统计数据：Sum
apiserver_admission_webhook_request_total_ADMIT	跨集群中所有 API 服务器发出的变更性准入 Webhook 请求数。 单位：计数 有效统计数据：Sum

指标名称	说明
apiserver_admission_webhook_request_total_VALIDATING	跨集群中所有 API 服务器发出的验证准入 Webhook 请求数。 单位：计数 有效统计数据：Sum
apiserver_admission_webhook_rejection_count	跨集群中所有 API 服务器发出但被拒绝的准入 Webhook 请求数。 单位：计数 有效统计数据：Sum
apiserver_admission_webhook_rejection_count_ADMIT	跨集群中所有 API 服务器发出但被拒绝的变更性准入 Webhook 请求数。 单位：计数 有效统计数据：Sum
apiserver_admission_webhook_rejection_count_VALIDATING	跨集群中所有 API 服务器发出但被拒绝的验证准入 Webhook 请求数。 单位：计数 有效统计数据：Sum
apiserver_admission_webhook_admission_duration_seconds	第三方准入 Webhook 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的第三方准入 Webhook 请求均在该时间内完成。 单位：秒 有效统计数据：平均值

指标名称	说明
apiserver_admission_webhook_admission_duration_seconds_ADMIT_P99	第三方变更性准入 Webhook 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的第三方变更性准入 Webhook 请求均在该时间内完成。 单位：秒 有效统计数据：平均值
apiserver_admission_webhook_admission_duration_seconds_VALIDATING_P99	第三方验证准入 Webhook 请求的第 99 个百分位延迟，根据集群中所有 API 服务器的所有请求计算得出。表示响应时间，即 99% 的第三方验证准入 Webhook 请求均在该时间内完成。 单位：秒 有效统计数据：平均值
apiserver_storage_size_bytes	集群中 API 服务器使用的 etcd 存储数据库文件的物理大小（以字节为单位）。该指标表示为存储分配的实际磁盘空间。 单位：字节 有效统计数据：最大值

Amazon CloudWatch Observability Operator

Amazon CloudWatch Observability 收集实时日志、指标并跟踪数据。然后将其发送到 Amazon CloudWatch 和 Amazon X-Ray。您可以安装此插件，以启用 CloudWatch Application Signals 和 CloudWatch Container Insights，从而增强 Amazon EKS 的可观测性。这有助于您监控基础设施和容器化应用程序的运行状况和性能。Amazon CloudWatch Observability Operator 旨在安装和配置必要的组件。

Amazon EKS 作为 Amazon EKS 附加组件对 CloudWatch Observability Operator 提供支持。附加组件可在集群中的 Linux 和 Windows Worker 节点上启用 Container Insights。要在 Windows 上启

用 Container Insights，Amazon EKS 附加组件版本必须为 `1.5.0` 或更高版本。目前，Amazon EKS Windows 不支持 CloudWatch Application Signals。

以下主题介绍了如何开始使用 CloudWatch Observability Operator 用于 Amazon EKS 集群。

- 有关安装此附加组件的说明，请参阅《Amazon CloudWatch 用户指南》中的使用 Amazon CloudWatch Observability EKS 附加组件或 Helm 图表安装 CloudWatch 代理。
- 有关 CloudWatch Application Signals 的更多信息，请参阅《Amazon CloudWatch 用户指南》中的应用程序信号。
- 有关 Container Insights 的更多信息，请参阅《Amazon CloudWatch 用户指南》中的使用 Container Insights。

将控制面板日志发送到 CloudWatch Logs

Amazon EKS 控制面板日志记录将审计和诊断日志直接从 Amazon EKS 控制面板提供到您账户中的 CloudWatch Logs。这些日志可让您轻松地保护和运行您的集群。您可以选择您需要的确切的日志类型，日志将作为日志流发送到 CloudWatch 中每个 Amazon EKS 集群的组。您可以使用 CloudWatch 订阅筛选条件来实时分析日志或将日志转发到其他服务（日志将采用 Base64 编码并使用 gzip 格式压缩）。有关更多信息，请参阅 Amazon CloudWatch 日志记录。

您可以通过选择要为每个新的或现有 Amazon EKS 集群启用的日志类型来开始使用 Amazon EKS 控制面板日志记录。您可以使用 Amazon Web Services 管理控制台、Amazon CLI（版本 `1.16.139` 或更高版本）或通过 Amazon EKS API 按集群启用或禁用每种日志类型。启用后，日志将从 Amazon EKS 集群自动发送到同一账户中的 CloudWatch Logs。

使用 Amazon EKS 控制面板日志记录时，您运行的每个集群将按标准 Amazon EKS 定价收费。对于从集群发送到 CloudWatch Logs 的任何日志，将向您收取标准 CloudWatch Logs 数据提取和存储费用。您还需要为预置为集群一部分的任何Amazon资源付费，如 Amazon EC2 实例或 Amazon EBS 卷。

以下集群控制面板日志类型可用。每个日志类型对应一个 Kubernetes 控制层面组件。要了解有关这些组件的更多信息，请参阅 Kubernetes 文档中的 Kubernetes 组件。

API 服务器**`api`** ()

> 集群的 API 服务器是公开 Kubernetes API 的控制面板组件。如果您在启动集群时或在启动之后不久启用 API 服务器日志，则这些日志包括用于启动 API 服务器的 API 服务器标志。有关更多信息，请参阅 Kubernetes 文档中的 kube-apiserver 和审核策略。

审计**audit** ()

Kubernetes 审计日志提供单个用户、管理员或影响集群的系统组件的记录。有关更多信息，请参阅 Kubernetes 文档中的审计。

身份验证器**authenticator** ()

身份验证器日志对于 Amazon EKS 是唯一的。这些日志代表 Amazon EKS 用于 Kubernetes 基于角色的访问控制(RBAC) 身份验证（使用 IAM 凭证）的控制层面组件。有关更多信息，请参阅 集群管理。

控制器管理器**controllerManager** ()

控制器管理器用于管理 Kubernetes 附带的核心控制环路。有关更多信息，请参阅 Kubernetes 文档中的 kube-controller-manager。

计划程序**scheduler** ()

调度器组件用于管理在集群中运行容器组（pod）的时间和位置。有关更多信息，请参阅 Kubernetes 文档中的 kube-scheduler。

启用或禁用控制面板日志

默认情况下，集群控制面板面日志不会发送到 CloudWatch Logs 中。您必须单独启用每个日志类型来为集群发送日志。CloudWatch Logs 摄取、归档存储和数据扫描速率适用于已启用的控制面板日志。有关更多信息，请参阅 CloudWatch 定价。

为了更新控制面板日志记录配置，Amazon EKS 要求每个子网中最多有五个可用的 IP 地址。启用日志类型后，发送的日志具有详细程度级别 2。

您可以使用 Amazon Web Services 管理控制台 或 Amazon CLI 启用或禁用控制面板日志。

Amazon Web Services 管理控制台

1. 打开 Amazon EKS console（Amazon EKS 控制台）。
2. 选择集群的名称可以显示集群信息。
3. 选择可观测性选项卡。
4. 在控制面板日志部分，选择管理日志记录。
5. 对于每个日志类型，选择开启还是关闭日志类型。默认情况下，每种日志类型都处于关闭状态。
6. 选择 Save changes（保存更改）以完成操作。

Amazon CLI

1. 使用以下命令查看您的 Amazon CLI 版本。

```
aws --version
```

   如果您的 Amazon CLI 版本低于 `1.16.139`，则必须先更新到最新版本。要安装或升级 Amazon CLI，请参阅《Amazon 命令行界面用户指南》中的安装 Amazon 命令行界面。

2. 使用下面的 Amazon CLI 命令更新集群的控制面板日志导出配置。将 *`my-cluster`* 替换为您的集群名称并指定所需的端点访问值。

> **Note**
>
> 以下命令将所有可用日志类型发送到 CloudWatch Logs。

```
aws eks update-cluster-config \
    --region region-code \
    --name my-cluster \
    --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}'
```

   示例输出如下。

```
{
    "update": {
        "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
        "status": "InProgress",
        "type": "LoggingUpdate",
        "params": [
            {
                "type": "ClusterLogging",
                "value": "{\"clusterLogging\":[{\"types\":[\"api\",\"audit\",
\"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}]}"
            }
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
```

```
}
```

3. 使用以下命令通过上一命令返回的集群名称和更新 ID 监控您的日志配置更新的状态。当状态显示为 Successful 时，您的更新将完成。

```
aws eks describe-update \
    --region region-code\
    --name my-cluster \
    --update-id 883405c8-65c6-4758-8cee-2a7c1340a6d9
```

示例输出如下。

```
{
    "update": {
        "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
        "status": "Successful",
        "type": "LoggingUpdate",
        "params": [
            {
                "type": "ClusterLogging",
                "value": "{\"clusterLogging\":[{\"types\":[\"api\",\"audit\",
\"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}]}"
            }
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
}
```

查看集群控制面板日志

为 Amazon EKS 集群启用任何控制面板日志类型后，您可以在 CloudWatch 控制台上进行查看。

要了解有关在 CloudWatch 中查看、分析和管理日志的详情，请参阅 Amazon CloudWatch Logs 用户指南。

1. 打开 CloudWatch console（CloudWatch 控制台）。此链接将打开控制台并显示您当前可用的日志组，并使用 /aws/eks 前缀筛选它们。
2. 选择您要查看日志的集群。日志组名称格式为 /aws/eks/*my-cluster*/cluster。
3. 选择要查看的日志流。以下列表介绍了每个日志类型的日志流名称格式。

> **Note**
>
> 随着日志流数据增加，日志流名称将轮换。当某个日志类型存在多个日志流时，您可以通过查找包含最新的上次事件时间的日志流名称来查看最新的日志流。

- Kubernetes API 服务器组件日志 (**api**) – kube-apiserver-*1234567890abcdef01234567890abcde*
- 审计 (**audit**) – kube-apiserver-audit-*1234567890abcdef01234567890abcde*
- 身份验证器 (**authenticator**) – authenticator-*1234567890abcdef01234567890abcde*
- 控制器管理器 (**controllerManager**) – kube-controller-manager-*1234567890abcdef01234567890abcde*
- 调度器 (**scheduler**) – kube-scheduler-*1234567890abcdef01234567890abcde*

4. 查看日志流的事件。

   例如，在查看 kube-apiserver-*1234567890abcdef01234567890abcde* 顶部时，您应看到集群的初始 API 服务器标志。

> **Note**
>
> 如果您在日志流的开头未看到 API 服务器日志，有可能是因为在您在服务器上启用 API 服务器日志记录之前，API 服务器日志文件已在服务器上轮换。在启用 API 服务器日志记录之前轮换的任何日志文件都无法导出到 CloudWatch。

不过，您可以使用相同的 Kubernetes 版本创建新集群，并在创建集群时启用 API 服务器日志记录。具有相同平台版本的集群会启用相同的标记，因此您的标记应该会与新集群的标记匹配。在 CloudWatch 中查看新集群的标记后，您可以删除新集群。

日志 API 调用作为 Amazon CloudTrail 事件

Amazon EKS 与 Amazon CloudTrail 集成。CloudTrail 服务在 Amazon EKS 中提供用户、角色或 Amazon 服务的操作记录。CloudTrail 将 Amazon EKS 的所有 API 调用作为事件捕获。这包含来自 Amazon EKS 控制台的调用和对 Amazon EKS API 操作的代码调用。

如果您创建了跟踪，则可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶。这包括 Amazon EKS 的事件。如果您不配置跟踪，则仍可在 CloudTrail 控制台中的事件历史记录中查看最新事件。使用 CloudTrail 收集的信息，您可以确定有关请求的几个详细信息。例如，您可以确定何时向 Amazon EKS 发出了请求，发出请求的 IP 地址以及何人发出的请求。

要了解有关 CloudTrail 的更多信息，请参阅 Amazon CloudTrail 用户指南。

主题

- 查看有关 Amazon CloudTrail 的实用参考信息
- 分析 Amazon CloudTrail 日志文件条目
- 查看 Amazon EC2 Auto Scaling 组的指标

查看有关 Amazon CloudTrail 的实用参考信息

在您创建 Amazon 账户时，还将在您的 Amazon 账户上启用 CloudTrail。当 Amazon EKS 中发生任何活动时，该活动将记录在 CloudTrail 事件中，并与其他 Amazon 服务事件一同保存在 Event history（事件历史记录）中。您可以在 Amazon 账户中查看、搜索和下载最新事件。有关更多信息，请参阅 使用 CloudTrail 事件历史记录查看事件 。

要持续记录Amazon账户中的事件（包括 Amazon EKS 的事件），请创建跟踪记录。通过跟踪记录，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 Amazon 区域。此跟踪记录来自 Amazon 分区中的所有 Amazon 区域的事件，并将日志文件传送至您指定的 Amazon S3 存储桶。此外，您可以配置其它 Amazon 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关详细信息，请参阅以下资源：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

所有 Amazon EKS 操作都由 CloudTrail 记录，并记录在 Amazon EKS API 参考中。例如，对 CreateCluster、ListClusters 和 DeleteCluster 部分的调用将在 CloudTrail 日志文件中生成条目。

每个事件或日志条目包含有关发出请求的 IAM 身份之类型的详细信息，以及使用了哪些凭证。如果使用的是临时凭证，则该条目显示凭证是如何获取的。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

分析 Amazon CloudTrail 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日记账条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作的信息。其中包括操作的日期和时间以及所使用的请求参数等信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目说明了 `CreateCluster` 操作。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws-cn:iam::111122223333:user/username",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "username"
  },
  "eventTime": "2018-05-28T19:16:43Z",
  "eventSource": "eks.amazonaws.com",
  "eventName": "CreateCluster",
  "awsRegion": "region-code",
  "sourceIPAddress": "205.251.233.178",
  "userAgent": "PostmanRuntime/6.4.0",
  "requestParameters": {
    "resourcesVpcConfig": {
      "subnetIds": [
        "subnet-a670c2df",
        "subnet-4f8c5004"
      ]
    },
    "roleArn": "arn:aws-cn:iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
    "clusterName": "test"
  },
  "responseElements": {
    "cluster": {
      "clusterName": "test",
      "status": "CREATING",
      "createdAt": 1527535003.208,
      "certificateAuthority": {},
      "arn": "arn:aws-cn:eks:region-code:111122223333:cluster/test",
```

```
      "roleArn": "arn:aws-cn:iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
      "version": "1.10",
      "resourcesVpcConfig": {
        "securityGroupIds": [],
        "vpcId": "vpc-21277358",
        "subnetIds": [
          "subnet-a670c2df",
          "subnet-4f8c5004"
        ]
      }
    }
  },
  "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
  "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Amazon EKS 服务相关角色的日志条目

Amazon EKS 服务相关角色对Amazon资源进行 API 调用。将显示由 Amazon EKS 服务相关角色进行的调用的 CloudTrail 日志条目以及 `username: AWSServiceRoleForAmazonEKS` 和 `username: AWSServiceRoleForAmazonEKSNodegroup`。有关 Amazon EKS 和服务相关角色的更多信息，请参阅 the section called “服务相关角色”。

以下示例显示了一个 CloudTrail 日志条目，该条目演示 `AWSServiceRoleForAmazonEKSNodegroup` 服务相关角色执行的 `DeleteInstanceProfile` 操作（在 `sessionContext` 中注明）。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3WHGPEZ7SJ2CW55C5:EKS",
        "arn": "arn:aws-cn:sts::111122223333:assumed-role/
AWSServiceRoleForAmazonEKSNodegroup/EKS",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
```

```
                "type": "Role",
                "principalId": "AROA3WHGPEZ7SJ2CW55C5",
                "arn": "arn:aws-cn:iam::111122223333:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "accountId": "111122223333",
                "userName": "AWSServiceRoleForAmazonEKSNodegroup"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-02-26T00:56:33Z"
            }
        },
        "invokedBy": "eks-nodegroup.amazonaws.com"
    },
    "eventTime": "2020-02-26T00:56:34Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "DeleteInstanceProfile",
    "awsRegion": "region-code",
    "sourceIPAddress": "eks-nodegroup.amazonaws.com",
    "userAgent": "eks-nodegroup.amazonaws.com",
    "requestParameters": {
        "instanceProfileName": "eks-11111111-2222-3333-4444-abcdef123456"
    },
    "responseElements": null,
    "requestID": "11111111-2222-3333-4444-abcdef123456",
    "eventID": "11111111-2222-3333-4444-abcdef123456",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


查看 Amazon EC2 Auto Scaling 组的指标

Amazon EKS 托管式节点组会默认启用 Amazon EC2 Auto Scaling 组指标，无需额外付费。自动扩缩组每分钟向 Amazon CloudWatch 发送采样数据。这些指标可以按照自动扩缩组名称进行细化。这些指标让您能够始终了解支持托管式节点组的自动扩缩组的历史记录，例如组大小随时间推移的变化。自动扩缩组指标可在 Amazon CloudWatch 或 Auto Scaling 控制台中获取。有关更多信息，请参阅 Monitor CloudWatch metrics for your Auto Scaling groups and instances。

借助自动扩缩组指标收集，您将能够监控托管式节点组的扩缩情况。自动扩缩组指标可报告自动扩缩组的最小、最大和所需大小。如果节点组中的节点数量小于最小大小（这表示节点组运行状况不佳），您可以创建警报。跟踪节点组大小在调整最大数量时也很有用，以便您的数据面板不会耗尽容量。

如果不想收集这些指标，则可以选择全部禁用或仅禁用其中的某些指标。例如，您可以通过禁用指标来避免 CloudWatch 控制面板出现噪音。有关更多信息，请参阅 Amazon CloudWatch metrics for Amazon EC2 Auto Scaling。

使用 ADOT Operator 发送指标和跟踪数据

Amazon EKS 支持使用 Amazon Web Services 管理控制台、Amazon CLI 和 Amazon EKS API 来安装和管理 Amazon Distro for OpenTelemetry（ADOT）Operator。这样，在 Amazon EKS 上运行的应用程序就能轻松地向多个监控服务选项发送指标和跟踪数据，比如 Amazon CloudWatch、Prometheus 和 X-Ray。

有关更多信息，请参阅适用于 OpenTelemetry 的 Amazon Distro 文档中的 Getting Started with Amazon Distro for OpenTelemetry using EKS Add-Ons。

使用 EKS 控制面板查看有关集群资源的聚合数据

image::images/eks-dashboard.png[screenshot of account level cluster metrics]

什么是 Amazon EKS 控制面板？

Amazon EKS 控制面板可以让您全面了解多个 Amazon 区域和 Amazon 账户的 Kubernetes 集群情况。您可以通过控制面板执行以下操作：

- 对计划在未来 90 天内终止支持自动升级的集群进行跟踪。
- 对扩展支持中的集群的 EKS 控制面板成本进行预测。
- 对升级之前需关注其见解的集群进行审查。
- 确定正在运行特定 AMI 版本的托管节点组。
- 监控集群支持类型分配（标准支持与扩展支持对比）。

EKS 控制面板与 EKS 集群见解集成，以发现集群存在的问题，例如使用已弃用的 Kubernetes API。有关更多信息，请参阅 the section called “集群见解”。

> **Note**
>
> EKS 控制面板不是实时的，每 12 小时更新一次。有关实时集群监控的信息，请参阅 监控集群

控制面板如何使用 Amazon Organizations？

Amazon EKS 控制面板需要与 Amazon Organiations 集成才能发挥作用。它可利用 Amazon Organizations 安全地跨账户收集集群信息。随着 Amazon 基础设施的扩展，这种集成可提供集中的管理和治理。

如果您的基础设施未启用 Amazon Organizations，请参阅 Amazon Organizations User Guide 获取设置说明。

跨区域和跨账户访问

EKS 控制面板可以查看属于 Amazon Organization 成员的任何账户中的集群资源。要生成组织中的 Amazon 账户列表，请参阅 Export details for all accounts in an organization。

us-east-1 Amazon 区域可生成该控制面板。您必须登录此区域才能查看控制面板。该控制面板聚合了各个 Amazon 区域的数据，但不包括 GovCloud 或中国区域。

关键术语

- Amazon Organization：多个 Amazon 账户的统一管理结构。
- 管理账户：控制 Amazon Organization 的主账户。
- 成员账户：组织内的任何账户，管理账户除外。
- 委派管理员：获得特定跨账户管理权限的成员账户。在管理账户中，您可以为每项 Amazon 服务选择一个委派管理员账户。
- 可信访问：授权 EKS 控制面板跨组织账户访问集群信息。
- 服务相关角色（SLR）：一种独特类型的 IAM 角色，与 Amazon 服务直接相关。EKS 控制面板使用 SLR 来读取有关您的账户和组织的信息。
- 有关更多信息，请参阅《Amazon Organizations User Guide》中的 Terminology and concepts。

一般概述

1. 访问 Amazon Organization 的管理账户。
   - 访问管理账户的步骤取决于您的 Amazon Organization 配置方式。例如，您可以通过 Amazon Identity Center 或 Okta 访问管理账户。
2. 通过 EKS 控制台启用可信访问。
3. 使用委派管理员的 Amazon 账户 ID 分配委派管理员。

4. 切换到委派管理员账户。
5. 访问具有组织范围可见性的增强型 EKS 控制台。

使用 Amazon 控制台启用 EKS 控制面板

> **Important**
>
> 您必须登录 Amazon Organization 的管理账户才能启用 EKS 控制面板。

访问 EKS 控制面板设置

1. 请确认以下内容：
   a. 您已经启用并配置 Amazon Organizations。
   b. 您已登录组织的管理账户。
   c. 您正在查看 us-east-1 区域的 Amazon 管理控制台。
2. 导航到 EKS 控制台。
3. 在左侧边栏中，打开“控制面板设置”。

设置对 Amazon EKS 控制面板的访问权限

1. 找到您想要允许查看 EKS 控制面板的 Amazon 账户的 Amazon 账户 ID。
   a. 此步骤为可选项，但建议使用。如果不这样做，您将只能从管理账户访问该控制面板。作为最佳实践，您应该限制对管理账户的访问。
2. 单击启用可信访问。
   a. 现在，您可以通过管理账户查看控制面板。
3. 单击注册委派管理员，然后输入您将用于查看控制面板的 Amazon 账户的账户 ID。
   a. 现在，您可以通过委派管理员账户或管理账户查看控制面板。

有关启用控制面板所需的权限的信息，请参阅所需的最低 IAM 策略。

查看 EKS 控制面板

1. 登录委派管理员账户（建议）或管理账户。

2. 登录 us-east-1 区域。
3. 前往 EKS 服务，然后从左侧边栏中选择“控制面板”。

> **Note**
>
> 审查查看 EKS 控制面板所需的 IAM 权限。

配置控制面板

您可以配置控制面板的视图并筛选资源。

可用资源

- 集群：查看有关 EKS 集群状态和位置的聚合信息。
  - 存在运行状况问题的集群。
  - EKS 扩展支持上的集群。
  - 按 Kubernetes 版本对集群进行细分。
- 托管节点组：查看托管节点组和 EC2 实例。
  - 按 AMI 类型划分的节点组，例如 Amazon Linux 或 Bottlerocket。
  - 节点组运行状况问题。
  - 实例类型分配。
- 附加组件：了解您已安装的 Amazon EKS 附加组件及其状态。
  - 每个附加组件的安装次数。
  - 存在运行状况问题的附加组件。
  - 每个附加组件的版本分发。

可用的视图

- 图表视图
  - 可自定义的小部件视图，显示所选资源的图表和可视化效果。
  - EKS 控制面板的所有用户均可查看对图表视图的更改，例如移除小部件。
- 资源视图
  - 所选资源的列表视图，支持筛选器。

- 地图视图
  - 查看所选资源的地理分布。

筛选 EKS 控制面板

您可以通过以下方式筛选 EKS 控制面板：

- Amazon 账户
- 由 Amazon Organizations 定义的组织单位
- Amazon 区域

使用 Amazon 控制台禁用 EKS 控制面板

1. 请确认以下内容：
   a. 您已经启用并配置 Amazon Organizations。
   b. 您已登录组织的管理账户。
   c. 您正在查看 us-east-1 区域的 Amazon 管理控制台。
2. 导航到 EKS 控制台。
3. 在左侧边栏中，打开“控制面板设置”。
4. 单击禁用可信访问。

EKS 控制面板问题排查

启用 EKS 控制面板时出现问题

- 您必须登录 Amazon Organization 的管理账户。
  - 如果您还没有 Amazon Organization，请创建一个。了解如何创建和配置组织。
  - 如果您的 Amazon 账户已经是 Amazon Organization 的成员，请确定该组织的管理员。
- 您必须登录具有足够 IAM 权限来创建和更新 Amazon Organizations 资源的 Amazon 账户。

查看 EKS 控制面板时出现问题

- 您必须登录以下其中一个 Amazon 账户：
  - Amazon Organization 的管理账户

  - 在管理账户的 EKS 控制面板设置中确定的委派管理员账户。
- 如果您最近启用了 EKS 控制面板，请注意，初始数据填充最多可能需要 12 个小时。
- 尝试使用 CLI 重新启用控制面板，包括创建服务相关角色。

控制面板小部件意外移动

- EKS 控制面板在 Amazon 账户级别保存可配置的小部件视图。如果您更改了小部件视图，则使用相同 Amazon 账户的其他人将看到更改。

配置 EKS 控制面板与 Amazon Organizations 的集成

本节将逐步说明如何配置 EKS 控制面板与 Amazon Organizations 的集成。您将了解如何启用和禁用服务之间的可信访问，以及如何注册和取消注册委派管理员账户。每项配置任务均可使用 Amazon 控制台或 Amazon CLI 来执行。

启用信任访问权限

可信访问会授权 EKS 控制面板安全地访问组织中所有账户的集群信息。

使用 Amazon 控制台

1. 登录 Amazon Organization 的管理账户。
2. 导航到 us-east-1 区域中的 EKS 控制台。
3. 在左侧边栏中，选择“控制面板设置”。
4. 单击启用可信访问。

> **Note**
>
> 您通过 EKS 控制台启用可信访问时，系统会自动创建 `AWSServiceRoleForAmazonEKSDashboard` 服务相关角色。如果您使用 Amazon CLI 或 Amazon Organizations 控制台启用可信访问，系统则不会自动创建。

使用 Amazon CLI

1. 登录 Amazon Organization 的管理账户。

2. 运行以下命令：

```
aws iam create-service-linked-role --aws-service-name dashboard.eks.amazonaws.com
aws organizations enable-aws-service-access --service-principal eks.amazonaws.com
```

禁用信任访问权限

禁用可信访问会撤销 EKS 控制面板访问您组织账户中集群信息的权限。

使用 Amazon 控制台

1. 登录 Amazon Organization 的管理账户。
2. 导航到 us-east-1 区域中的 EKS 控制台。
3. 在左侧边栏中，选择“控制面板设置”。
4. 单击禁用可信访问。

使用 Amazon CLI

1. 登录 Amazon Organization 的管理账户。
2. 运行如下命令：

```
aws organizations disable-aws-service-access --service-principal eks.amazonaws.com
```

启用委派管理员账户

委派管理员是获得 EKS 控制面板访问权限的成员账户。

使用 Amazon 控制台

1. 登录 Amazon Organization 的管理账户。
2. 导航到 us-east-1 区域中的 EKS 控制台。
3. 在左侧边栏中，选择“控制面板设置”。
4. 单击注册委派管理员。
5. 输入您要选择为委派管理员的 Amazon 账户的 ID。
6. 确认注册。

使用 Amazon CLI

1. 登录 Amazon Organization 的管理账户。
2. 运行以下命令，将 123456789012 替换为您的账户 ID：

```
aws organizations register-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

禁用委派管理员账户

禁用委派管理员会移除该账户访问 EKS 控制面板的权限。

使用 Amazon 控制台

1. 登录 Amazon Organization 的管理账户。
2. 导航到 us-east-1 区域中的 EKS 控制台。
3. 在左侧边栏中，选择“控制面板设置”。
4. 在列表中找到委派管理员。
5. 在您要移除的作为委派管理员的账户旁边，单击取消注册。

使用 Amazon CLI

1. 登录 Amazon Organization 的管理账户。
2. 运行以下命令，将 123456789012 替换为委派管理员的账户 ID：

```
aws organizations deregister-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

所需的最低 IAM 策略

本节概述了针对 EKS 控制面板与 Amazon Organizations 的集成启用可信访问和委派管理员所需的最低 IAM 策略。

启用可信访问的策略

要在 EKS 控制面板和 Amazon Organizations 之间启用可信访问，您需要以下权限：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "organizations:EnableAWSServiceAccess",
                "organizations:DescribeOrganization",
                "organizations:ListAWSServiceAccessForOrganization"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
dashboard.eks.amazonaws.com/AWSServiceRoleForAmazonEKSDashboard"
        }
    ]
}
```

委派管理员的策略

要注册或取消注册 EKS 控制面板的委派管理员，您需要以下权限：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "organizations:RegisterDelegatedAdministrator",
                "organizations:DeregisterDelegatedAdministrator",
                "organizations:ListDelegatedAdministrators"
            ],
            "Resource": "*"
        }
    ]
}
```

查看 EKS 控制面板的策略

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonEKSDashboardReadOnly",
            "Effect": "Allow",
            "Action": [
                "eks:ListDashboardData",
                "eks:ListDashboardResources",
                "eks:DescribeClusterVersions"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmazonOrganizationsReadOnly",
            "Effect": "Allow",
            "Action": [
                "organizations:DescribeOrganization",
                "organizations:ListAWSServiceAccessForOrganization",
                "organizations:ListRoots",
                "organizations:ListAccountsForParent",
                "organizations:ListOrganizationalUnitsForParent"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmazonOrganizationsDelegatedAdmin",
            "Effect": "Allow",
            "Action": [
                "organizations:ListDelegatedAdministrators"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "organizations:ServicePrincipal": "eks.amazonaws.com"
                }
            }
        }
    ]
```

```
}
```

> **Note**
>
> 这些策略必须附加到 Amazon Organizations 管理账户中的 IAM 主体（用户或角色）。成员账户无法启用可信访问或委派管理员。

通过集成的 Amazon 服务增强 EKS

除了其他部分介绍的服务外，Amazon EKS 还与其他 Amazon 服务配合使用，以提供其他解决方案。本主题介绍了使用 Amazon EKS 添加功能的一些其他服务，或 Amazon EKS 用来执行任务的服务。

主题

- 使用 Amazon Backup 备份 EKS 集群
- 使用 Amazon CloudFormation 创建 Amazon EKS 资源
- 使用 Amazon CodeConnections 连接到 Git 存储库
- 使用 Amazon Detective 在 EKS 上分析安全事件
- 使用 Amazon GuardDuty 检测威胁
- 使用 Amazon Local Zones 启动低延迟 EKS 集群
- 使用 Amazon 弹性中心评测 EKS 集群弹性
- 使用 Amazon Secrets Manager 管理应用程序密钥
- 使用 Security Lake 集中和分析 EKS 安全数据
- 使用 Amazon VPC Lattice 实现安全的跨集群连接

使用 Amazon Backup 备份 EKS 集群

Amazon Backup 支持对 Amazon EKS 集群进行备份，范围涵盖 Kubernetes 集群状态，以及通过持久卷声明挂载到 EKS 集群的持久存储（包括 EBS 卷、EFS 文件系统和 S3 存储桶）。针对 Amazon EKS 集群的备份操作会生成一个复合恢复点，其中每一项被备份的资源都会对应一个子恢复点。

- 资源备份方法：Amazon Backup 入门
- 按资源类型创建备份：Amazon EKS 备份
- Amazon EKS 集群恢复方法：Amazon EKS 备份恢复

若要通过 EKS 控制台、Amazon Backup 控制台或 CLI 开始操作，请确保 IAM 角色具备以下权限：

- 这些权限可在 Amazon Backup 的托管策略 AWSBackupServiceRolePolicyForBackup 中找到。该策略包含备份 Amazon EKS 集群以及 EBS、EFS 持久存储所需的全部权限
- 若 EKS 集群中包含 S3 存储桶，则需按照文档要求，为该 S3 存储桶添加并启用以下策略及先决条件：

- AWSBackupServiceRolePolicyForS3Backup
- S3 备份先决条件

确保 EKS 集群已配置以下设置：

- 将 EKS 集群授权模式设置为 API 或 API_AND_CONFIG_MAP，以便 Amazon Backup 创建访问条目来访问该 EKS 集群。

使用 Amazon CloudFormation 创建 Amazon EKS 资源

Amazon EKS 与 Amazon CloudFormation 集成，后者是一项服务，可帮助您对 Amazon 资源进行建模和设置，这样您只需花较少的时间来创建和管理资源与基础设施。您创建一个描述您所需的所有 Amazon 资源（如 Amazon EKS 集群）的模板，而 Amazon CloudFormation 将负责为您设置和配置这些资源。

在您使用 Amazon CloudFormation 时，可重复使用您的模板来不断地重复设置您的 Amazon EKS 资源。仅描述您的资源一次，然后在多个 Amazon 账户和区域中反复配置相同的资源。

Amazon EKS 和 Amazon CloudFormation 模板

要为 Amazon EKS 和相关服务预置和配置资源，您必须了解 Amazon CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板描述要在 Amazon CloudFormation 堆栈中预调配的资源。如果您不熟悉 JSON 或 YAML，则可以使用 Amazon CloudFormation Designer，以帮助您开始使用 Amazon CloudFormation 模板。有关更多信息，请参阅 Amazon CloudFormation 用户指南中的什么是 Amazon CloudFormation Designer？。

Amazon EKS 支持在 Amazon CloudFormation 中创建集群和节点组。有关更多信息（包括您的 Amazon EKS 资源的 JSON 和 YAML 模板示例），请参阅 Amazon CloudFormation 用户指南中的 Amazon EKS 资源类型参考。

了解有关 Amazon CloudFormation 的更多信息

要了解有关 Amazon CloudFormation 的更多信息，请参阅以下资源：

- Amazon CloudFormation
- Amazon CloudFormation 用户指南
- Amazon CloudFormation 命令行界面用户指南

使用 Amazon CodeConnections 连接到 Git 存储库

Amazon CodeConnections 提供了一种安全方式，用于将 Amazon 服务与第三方源代码存储库建立连接。AmazonCodeConnections 支持 GitHub、GitLab、Bitbucket 以及其他服务商。如需了解更多信息并快速上手，，请参阅使用连接。

在 Argo CD 中使用 Amazon CodeConnections

当您使用适用于 Argo CD 的 EKS 功能时，可选择借助 Amazon CodeConnections 实现与 Git 存储库的安全身份认证，无需手动管理长期凭证或个人访问令牌。AmazonCodeConnections 会处理 OAuth 身份认证流程，并负责维护与 Git 服务商的连接。这种方式能提供一套安全且易于管控的方案，用于访问存储在第三方 Git 服务商中的 GitOps 存储库及应用程序清单。

先决条件

- 已创建带有 Argo CD 功能的 Amazon EKS 集群
- 已在 Amazon CodeConnections 中创建与 Git 服务商的连接
- 已为 Argo CD 配置好使用该连接所需的 IAM 权限

配置 CodeConnections 实现 Argo CD 存储库访问

1. 在 CodeConnections 控制台中创建连接：
   a. 打开 CodeConnections 控制台。
   b. 选择创建连接。
   c. 选择所用的服务商（GitHub、GitLab 或 Bitbucket），再按照指引完成身份认证流程。
   d. 记录该连接的 ARN，以便在 Argo CD 配置中使用。
2. 确保 Argo CD 功能角色已通过基于资源的策略，获得使用该连接的权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection",
        "codeconnections:GetConnection"
      ],
```

```
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-
id"
    }
  ]
}
```

3. 在 Argo CD 中添加存储库时，需将其配置为引用 CodeConnections 资源，并将 CodeConnections 资源端点作为存储库 URL。Argo CD 功能会通过该连接完成与 Git 服务商的身份认证，无需使用长期凭证。

搭配 Argo CD 使用 CodeConnections 的注意事项

在适用于 Argo CD 的 EKS 功能中搭配使用 Amazon CodeConnections 时，请留意以下几点：

- CodeConnections 连接必须与 EKS 集群处于同一 Amazon 区域
- Argo CD 功能角色必须具备 `codeconnections:UseConnection` 和 `codeconnections:GetConnection` 权限
- CodeConnections 会自动管理 OAuth 流程与凭证的生命周期

有关使用 Argo CD 配置存储库访问权限的更多信息，请参阅the section called “配置存储库”。

使用 Amazon Detective 在 EKS 上分析安全事件

Amazon Detective 可帮助您分析、调查和快速识别安全结果或可疑活动的根本原因。Detective 会自动从Amazon资源收集日志数据。然后，它使用机器学习、统计分析和图形理论生成可视化效果，帮助更快、更高效地进行安全调查。Detective 的预构建数据聚合、摘要和上下文可有助于分析和确定潜在安全问题的性质和程度。有关更多信息，请参阅 Amazon Detective 用户指南。

Detective 将 Kubernetes 和 Amazon 数据组织到发现结果中，例如：

- Amazon EKS 集群详细信息，包括创建集群的 IAM 身份和集群的服务角色。可以使用 Detective 调查这些 IAM 身份的 Amazon 和 Kubernetes API 活动。
- 容器详细信息，例如映像和安全上下文。此外，您还可以查看已终止容器组（pod）的详细信息。
- Kubernetes API 活动，包括 API 活动的总体趋势和特定 API 调用的详细信息。例如，您可以显示在选定时间范围内发出的成功和失败 Kubernetes API 调用的数量。此外，有关新观察到的 API 调用的部分可能有助于识别可疑活动。

Amazon EKS 审核日志是一个可选的数据来源软件包，可以添加到您的 Detective 行为图中。您可以查看自己的账户中可用的可选来源软件包及其状态。有关更多信息，请参阅 Amazon Detective 用户指南中的适用于 Detective 的 Amazon EKS 审核日志。

将 Amazon Detective 与 Amazon EKS 结合使用

在查看发现结果之前，必须于集群所在的同一 Amazon 区域内启用 Detective 至少 48 小时。有关更多信息，请参阅 Amazon Detective 用户指南中的设置 Amazon Detective。

1. 打开 Detective 控制台，网址为 https://console.aws.amazon.com/detective/。
2. 从左侧导航窗格中选择搜索。
3. 选择选择类型，然后选择 EKS 集群。
4. 输入集群名称或 ARN，然后选择搜索。
5. 在搜索结果中，选择要查看其活动的集群的名称。有关您可以查看的内容的更多信息，请参《Amazon Detective 用户指南》中的涉及 Amazon EKS 集群的整体 Kubernetes API 活动。

使用 Amazon GuardDuty 检测威胁

Amazon GuardDuty 是一项威胁检测服务，有助于保护您的账户、容器、工作负载和 Amazon 环境中的数据。GuardDuty 使用机器学习（ML）模型以及异常和威胁检测功能，持续监控不同的日志源和运行时活动，以识别环境中的潜在安全风险和恶意活动并确定其优先级。

除其他功能外，GuardDuty 还提供以下两项功能，用于检测 EKS 集群面临的潜在威胁：EKS 保护和运行时监控。

> **Note**
>
> 新增功能：Amazon EKS 自动模式与 GuardDuty 集成。

EKS 保护

此功能提供威胁检测覆盖，通过监控关联的 Kubernetes 审计日志来帮助保护 Amazon EKS 集群。Kubernetes 审计日志可捕获集群内的连续操作，包括来自用户、使用 Kubernetes API 的应用程序和控制面板的活动。例如，GuardDuty 可以识别出未经身份验证的用户执行的 API 调用，这些调用可能用于篡改 Kubernetes 集群中的资源。

启用 EKS 保护后，GuardDuty 将只能访问 Amazon EKS 审计日志，从而持续进行威胁检测。如果 GuardDuty 发现集群存在潜在威胁，就会生成具有特定类型的关联的 Kubernetes 审计日志调查发现。有关 Kubernetes 审计日志中可用的调查发现类型的更多信息，请参阅《Amazon GuardDuty User Guide》中的 Kubernetes audit logs finding types。

有关更多信息，请参阅《Amazon GuardDuty User Guide》中的 EKS Protection。

运行时监控

此功能可监控和分析操作系统级事件、网络事件和文件事件，帮助您检测环境中特定 Amazon 工作负载中的潜在威胁。

启用运行时监控并在 Amazon EKS 集群中安装 GuardDuty 代理后，GuardDuty 就会开始监控与此集群关联的运行时事件。请注意，Amazon EKS 混合节点功能不支持 GuardDuty 代理和运行时监控，因此运行时监控不适用于混合节点上发生的运行时事件。如果 GuardDuty 发现集群存在潜在威胁，就会生成关联的运行时监控调查发现。例如，威胁可能会从破坏单个容器开始，而这种容器通常在运行易受攻击的 Web 应用程序。此 Web 应用程序可能拥有对底层容器和工作负载的访问权限。在这种情况下，错误配置的凭证可能会导致对账户及其所存储数据的访问权限扩大。

要配置运行时监控，可将 GuardDuty 代理作为 Amazon EKS 附加组件安装到集群中。有关附加组件的更多信息，请参阅 the section called “ Amazon 附加组件”。

有关更多信息，请参阅《Amazon GuardDuty User Guide》中的 Runtime Monitoring。

使用 Amazon Local Zones 启动低延迟 EKS 集群

Amazon Local Zone 是在地理上靠近您的用户的 Amazon 区域的扩展。Local Zones 有自己的互联网连接并支持 Amazon Direct Connect。在 Local Zones 中创建的资源可以通过低延迟通信服务于本地用户。有关更多信息，请参阅《Amazon Local Zones User Guide》和《Amazon EC2 用户指南》中的 Local Zones。

Amazon EKS 支持本地区域中的某些资源。这包括自行管理的节点组、自行管理的 Amazon EC2 节点、Amazon EBS 卷以及应用程序负载均衡器（ALB）。将本地区域用作 Amazon EKS 集群的一部分时，我们建议您考虑以下几个因素。

- 您无法在 Local Zones 中使用 Amazon EKS 创建 Fargate 节点。
- Amazon EKS 托管的 Kubernetes 控制层面始终在Amazon区域中运行。Amazon EKS 托管的 Kubernetes 控制面板不能在 Local Zones 中运行。由于 Local Zones 在 VPC 中显示为子网，因此 Kubernetes 会将您的 Local Zones 资源视为该子网的一部分。

- Amazon EKS Kubernetes 集群使用 Amazon EKS 托管的弹性网络接口，与您在Amazon区域或 Local Zones 中运行的 Amazon EC2 实例进行通信。要了解有关 Amazon EKS 联网架构的更多信息，请参阅 配置联网。
- 与区域子网不同，Amazon EKS 不能将网络接口放置到您的本地区域子网中。这意味着您在创建集群时一定不要指定 Local Zones 子网。但是，您可以将位于多个不同本地区域中的 Worker 节点连接到同一个集群。

使用 Amazon 弹性中心评测 EKS 集群弹性

Amazon Resilience Hub 通过分析 Amazon EKS 集群的基础设施来评估 Amazon EKS 集群的弹性。AmazonResilience Hub 使用 Kubernetes 基于角色的访问控制（RBAC）配置来评测部署到集群的 Kubernetes 工作负载。有关更多信息，请参阅《Amazon Resilience Hub 用户指南》中的为 Amazon EKS 集群启用 Amazon Resilience Hub 访问。

使用 Amazon Secrets Manager 管理应用程序密钥

Amazon Secrets Manager 可帮助您在凭证、API 密钥及其他机密信息的全生命周期内，完成对其的管理、访问与轮换操作。借助 Secrets Manager，您能够安全管理用于访问 Amazon 环境、第三方服务以及本地环境中各类资源的机密信息。有关更多信息，请参阅 Amazon Secrets Manager 用户指南。

在适用于 Argo CD 的 EKS 功能中使用 Secrets Manager 时，该服务可提供一种安全方式来存储和获取 Git 存储库凭证，无需在 Argo CD 的配置项和资源列表中硬编码敏感数据。这种集成方式对于管理私有存储库访问令牌，以及 Argo CD 用于从 Git 存储库同步应用的 SSH 密钥，都尤为实用。

在 Argo CD 中使用 Amazon Secrets Manager

在适用于 Argo CD 的 EKS 功能中进行操作时，可将 Git 存储库凭证存储在 Secrets Manager 中，并配置 Argo CD 来获取这些凭证。相较于直接在 Argo CD 配置中存储凭证，或是使用长期有效的个人访问令牌，这种方式安全性要更高。

先决条件

- 已启用 Argo CD 功能的 Amazon EKS 集群
- 已在 Amazon Secrets Manager 中存储 Git 存储库凭证
- 已为 Argo CD 配置访问 Secrets Manager 所需的 IAM 权限

配置 Argo CD 使用 Secrets Manager 管理存储库凭证

1. 将 Git 凭证存储到 Secrets Manager 中。例如，存储一个 GitHub 个人访问令牌：

```
aws secretsmanager create-secret \
  --name argocd/github-token \
  --secret-string '{"username":"git","password":"ghp_xxxxxxxxxxxx"}'
```

2. 确保 Argo CD 功能角色具备获取机密信息的权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:region:account-id:secret:argocd/github-
token*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:region:account-id:key/*",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:SecretARN": "arn:aws:secretsmanager:region:account-
id:secret:argocd/*",
          "kms:ViaService": "secretsmanager.*.amazonaws.com"
        }
      }
    }
  ]
}
```

> **Note**
>
> 由于 Secrets Manager 会使用 Amazon KMS 对所有机密信息进行加密，因此 KMS 解密权限是必需项。配置中的条件会将解密操作限制为仅针对带有 `argocd/` 前缀的机密信息。若

使用 Secrets Manager 的默认 Amazon 托管式密钥，则该权限已足够。若使用客户托管式 KMS 密钥，则在 `Resource` 字段中更新为对应的密钥 ARN。

3. 配置 Argo CD，使其从 Secrets Manager 中获取凭证。有关将 Secrets Manager 中的机密信息同步为 Argo CD 可引用的 Kubernetes 机密的方法，请参阅 Argo CD 文档中的机密管理章节。
4. 创建一个引用该机密 ARN 的 Argo CD 存储库配置：

```
apiVersion: v1
kind: Secret
metadata:
  name: private-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/org/repo
  secretArn: arn:aws:secretsmanager:region:account-id:secret:argocd/github-token
```

有关使用 Argo CD 配置存储库访问权限的更多信息，请参阅the section called “配置存储库”。

使用 Security Lake 集中和分析 EKS 安全数据

Amazon Security Lake 是一项完全托管的安全数据湖服务，通过它，您可以集中来自各种来源（包括 Amazon EKS）的安全数据。将 Amazon EKS 与 Security Lake 集成，您可以更深入地了解在自己 Kubernetes 资源上执行的活动，增强 Amazon EKS 集群的安全状况。

> **Note**
>
> 有关将 Security Lake 与 Amazon EKS 结合使用和设置数据来源的更多信息，请参阅 Amazon Security Lake 文档。

将 Security Lake 与 Amazon EKS 结合使用的益处

集中安全数据 – Security Lake 自动收集和集中来自 Amazon EKS 集群的安全数据，以及来自其它 Amazon 服务、SaaS 提供商、本地来源和第三方来源的数据。这将提供整个组织的安全状况的全面视图。

标准化数据格式 – Security Lake 将收集的数据转换为标准开源架构 – 开放网络安全架构框架（OCSF）格式。这种标准化使分析和集成其他安全工具和服务变得更加容易。

改进了威胁检测 – 通过分析集中式安全数据，包括 Amazon EKS 控制面板日志，您可以更有效地检测 Amazon EKS 集群中可能存在的可疑活动。这有助于及时识别和响应安全事件。

简化数据管理 – Security Lake 通过可自定义的保留设置和复制设置来管理安全数据的生命周期。这简化了数据管理任务，并确保您可以保留用于合规和审计目的所需的数据。

为 Amazon EKS 启用 Security Lake

1. 为 EKS 集群启用 Amazon EKS 控制面板日志记录。有关详细说明，请参阅启用和禁用控制面板日志。
2. 将 Amazon EKS 审核日志作为来源添加到 Security Lake 中。然后，Security Lake 将开始收集在您的 EKS 集群中运行的 Kubernetes 资源上执行的活动的相关深入信息。
3. 根据您的要求在 Security Lake 中为您的安全数据配置保留和复制设置。
4. 使用存储在 Security Lake 中的标准化 OCSF 数据进行事件响应、安全分析以及与其他 Amazon 服务或第三方工具的集成。例如，您可以使用 Amazon OpenSearch Ingestion 从 Amazon Security Lake 数据中生成安全见解。

在 Security Lake 中分析 EKS 日志

Security Lake 将 EKS 日志事件标准化为 OCSF 格式，从而使分析数据并将其与其他安全事件关联起来更加容易。您可以使用各种工具和服务，例如 Amazon Athena、Amazon QuickSight 或第三方安全分析工具，来查询和可视化标准化数据。

有关 EKS 日志事件的 OCSF 映射的详细信息，请参阅 OCSF GitHub 存储库中的 https://github.com/ocsf/examples/tree/main/mappings/markdown/ Amazon/v1.1.0/EKS Audit Logs[mapping reference]。

使用 Amazon VPC Lattice 实现安全的跨集群连接

Amazon VPC Lattice 是一项完全托管的应用程序网络服务，它直接内置在 Amazon 网络基础设施中，您可以将其用于连接、保护和监控您在多个账户和虚拟私有云（VPC）中的服务。使用 Amazon EKS，您可以通过使用 Amazon Gateway API 控制器（这是 Kubernetes Gateway API 的一种实施）来利用 Amazon VPC Lattice。您可以使用 Amazon VPC Lattice，以简单一致的方式设置具有标准 Kubernetes 语义的跨集群连接。要开始将 Amazon VPC Lattice 与 Amazon EKS 结合使用，请参阅 Amazon Gateway API 控制器用户指南。

使用 Amazon EKS Connector 将 Kubernetes 集群连接到 Amazon EKS 管理控制台

您可以使用 Amazon EKS Connector 注册并将任何符合要求的 Kubernetes 集群连接至 Amazon，并在 Amazon EKS 控制台中进行显示。连接集群后，您可以在 Amazon EKS 控制台中查看集群的状态、配置和工作负载。您可以使用此功能在 Amazon EKS 控制台中查看已连接的集群，但您无法对其进行管理。Amazon EKS Connector 需要一个代理，该代理是 Github 上的开源项目。有关其它技术内容，包括常见问题和故障排除，请参阅the section called “EKS Connector 问题排查”

Amazon EKS Connector 能够将以下类型的 Kubernetes 集群连接到 Amazon EKS。

- 本地部署 Kubernetes 集群
- 在 Amazon EC2 上运行的自托管式集群
- 来自其他云提供商的管理集群

Amazon EKS Connector 注意事项

在使用 Amazon EKS Connector 之前，请先了解以下内容：

- 您必须拥有 Kubernetes 集群的管理权限才能将集群连接到 Amazon EKS。
- 连接前，Kubernetes 集群必须有 Linux 64 位 (x86) Worker 节点。不支持 ARM Worker 节点。
- 您的 Kubernetes 集群中必须包含拥有对 `ssm.` 和 `ssmmessages.` Systems Manager 端点出站访问的 Worker 节点。有关更多信息，请参阅 Amazon 一般参考中的 Systems Manager 端点。
- 每个区域原定设置最多可连接 10 个集群。您可以通过服务配额控制台请求增加配额。请参阅请求增加配额了解更多信息。
- 对于外部 Kubernetes 集群，仅支持 Amazon EKS `RegisterCluster`、`ListClusters`、`DescribeCluster` 和 `DeregisterCluster` API。
- 您必须拥有以下权限才能注册集群：
  - eks:RegisterCluster
  - ssm:CreateActivation
  - ssm:DeleteActivation
  - iam:PassRole
- 您必须拥有以下权限才能注销集群：

- eks:DeregisterCluster
- ssm:DeleteActivation
- ssm:DeregisterManagedInstance

Amazon EKS Connector 所需的 IAM 角色

使用 Amazon EKS Connector 需要以下两个 IAM 角色：

- 首次注册集群时，您需要创建 Amazon EKS Connector 服务相关角色。
- 您必须创建 Amazon EKS Connector 代理 IAM 角色。有关详细信息，请参阅the section called “Connector IAM 角色”。

要为 IAM 主体启用集群和工作负载视图权限，请将 `eks-connector` 和 Amazon EKS Connector 集群角色应用到集群。按授予在 Amazon EKS 控制台上查看 Kubernetes 集群资源的权限中的步骤操作。

将外部 Kubernetes 集群连接到 Amazon EKS 管理控制台

可在以下过程中使用多种方法将外部 Kubernetes 集群连接到 Amazon EKS。此过程包括两个步骤：向 Amazon EKS 注册集群，在集群中安装 `eks-connector` 代理。

> **Important**
>
> 您必须在完成第一步后 3 天内（注册到期之前）完成第二步。

注意事项

安装代理时，您可以使用 YAML 清单。或者，如果您向 Amazon Web Services 管理控制台 或 Amazon 命令行界面注册集群，则可以使用 Helm。但是，如果您向 `eksctl` 注册集群，则无法使用 Helm 安装代理。

先决条件

- 确保已创建 Amazon EKS Connector 代理角色。按照创建 Amazon EKS Connector 代理角色中的步骤操作。

- 您必须拥有以下权限才能注册集群：
  - `eks:RegisterCluster`
  - `ssm:CreateActivation`
  - `ssm:DeleteActivation`
  - `iam:PassRole`

步骤 1：注册集群

要向 Amazon EKS 连接器注册集群，您可以使用以下工具之一：

- the section called “Amazon CLI”
- the section called “Amazon Web Services 管理控制台”
- the section called “`eksctl`”

Amazon CLI

1. 必须安装 Amazon CLI。要进行安装或升级，请参阅安装 Amazon CLI。
2. 对于 Connector 配置，请指定您的 Amazon EKS Connector 代理 IAM 角色。有关更多信息，请参阅 the section called “Amazon EKS Connector 所需的 IAM 角色”。

```
aws eks register-cluster \
     --name my-first-registered-cluster \
     --connector-config roleArn=arn:aws-cn:iam::111122223333:role/
AmazonEKSConnectorAgentRole,provider="OTHER" \
     --region aws-region
```

示例输出如下。

```
{
    "cluster": {
        "name": "my-first-registered-cluster",
        "arn": "arn:aws-cn:eks:region:111122223333:cluster/my-first-registered-
cluster",
        "createdAt": 1627669203.531,
        "ConnectorConfig": {
            "activationId": "xxxxxxxxACTIVATION_IDxxxxxxxx",
            "activationCode": "xxxxxxxxACTIVATION_CODExxxxxxxx",
```

```
            "activationExpiry": 1627672543.0,
            "provider": "OTHER",
            "roleArn": "arn:aws-cn:iam::111122223333:role/
AmazonEKSConnectorAgentRole"
        },
        "status": "CREATING"
    }
}
```

在下一步中使用 aws-region、activationId 和 activationCode 值。

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择 Add cluster（添加集群），然后选择 Register（注册）以打开配置页面。
3. 在 Configure cluster (配置集群) 部分，填写以下字段：
   - Name（名称）– 集群的唯一名称。
   - Provider（提供程序）– 选择以显示 Kubernetes 集群提供程序的下拉列表。如果您不了解具体的提供程序，请选择其它。
   - EKS Connector 角色：选择用于连接集群的角色。
4. 选择 Register cluster（注册集群）。
5. 此时将显示集群概览页面。如果您想使用 Helm 图表，请复制 helm install 命令并继续下一步。如果要使用 YAML 清单，请选择下载 YAML 文件，将清单文件下载到本地驱动程序。

> **Important**
>
> 这是您复制 helm install 命令或下载此文件的唯一机会。不要离开此页面，否则将无法访问此链接，届时您必须注销集群并从头开始此步骤。

对于注册的集群，命令或清单文件只能使用一次。如果从 Kubernetes 集群中删除资源，则必须重新注册集群并获取新的清单文件。

继续执行下一步，以将清单文件应用于 Kubernetes 集群。

eksctl

1. 必须安装 eksctl 版本 0.68 或更高版本。要安装或对其升级，请参阅 the section called “创建集群（eksctl）”。
2. 提供名称、提供程序和区域来注册集群。

```
eksctl register cluster --name my-cluster --provider my-provider --region region-code
```

输出示例：

```
2021-08-19 13:47:26 [#]  creating IAM role "eksctl-20210819194112186040"
2021-08-19 13:47:26 [#]  registered cluster "<name>" successfully
2021-08-19 13:47:26 [#]  wrote file eks-connector.yaml to <current directory>
2021-08-19 13:47:26 [#]  wrote file eks-connector-clusterrole.yaml to <current
 directory>
2021-08-19 13:47:26 [#]  wrote file eks-connector-console-dashboard-full-access-
group.yaml to <current directory>
2021-08-19 13:47:26 [!]  note: "eks-connector-clusterrole.yaml" and "eks-connector-
console-dashboard-full-access-group.yaml" give full EKS Console access to IAM
 identity "<aws-arn>", edit if required; read https://eksctl.io/usage/eks-connector
 for more info
2021-08-19 13:47:26 [#]  run `kubectl apply -f eks-connector.yaml,eks-connector-
clusterrole.yaml,eks-connector-console-dashboard-full-access-group.yaml` before
 expiry> to connect the cluster
```

这将在本地计算机上创建文件。必须在 3 天内将这些文件应用到外部集群，否则注册将会过期。

3. 在可以访问集群的终端中，应用 eks-connector-binding.yaml 文件：

```
kubectl apply -f eks-connector-binding.yaml
```

步骤 2：安装 eks-connector 代理

要安装 eks-connector 代理，请使用以下工具之一：

- the section called “Helm”
- the section called “yaml”

Helm

> **Note**
>
> 如果您向 eksctl 注册了集群，则请使用 YAML 清单方法而不是 Helm 图表方法。

1. 如果在上一步中使用了 Amazon CLI，请将以下命令中的 ACTIVATION_CODE 和 ACTIVATION_ID 分别替换为 activationId 和 activationCode 值。将 aws-region 替换为在上一步中使用的 Amazon 区域。然后运行命令在注册集群上安装 eks-connector 代理：

   ```
   $ helm install eks-connector \
     --namespace eks-connector \
     oci://public.ecr.aws/eks-connector/eks-connector-chart \
     --set eks.activationCode=ACTIVATION_CODE \
     --set eks.activationId=ACTIVATION_ID \
     --set eks.agentRegion=aws-region
   ```

   如果在上一步中使用了 Amazon Web Services 管理控制台，请使用从上一步复制的已填充这些值的命令。
2. 检查已安装 eks-connector 部署的运行状况，并等待 Amazon EKS 中已注册集群的状态变为 ACTIVE。

yaml

通过将 Amazon EKS Connector 清单文件应用于 Kubernetes 集群来完成连接。为此，必须使用前面描述的方法。如果没有在 3 天内应用清单，Amazon EKS Connector 注册将过期。如果集群连接过期，则必须在再次连接集群之前注销集群。

1. 下载 Amazon EKS Connector YAML 文件。

   ```
   curl -O https://amazon-eks.s3.us-west-2.amazonaws.com/eks-connector/manifests/eks-connector/latest/eks-connector.yaml
   ```
2. 编辑 Amazon EKS Connector YAML 文件，将 %AWS_REGION%、%EKS_ACTIVATION_ID%、%EKS_ACTIVATION_CODE% 的所有引用替换为上一步输出的 aws-region、activationId 和 activationCode。

   以下示例命令可以替换这些值。

```
sed -i "s~%AWS_REGION%~$aws-region~g; s~%EKS_ACTIVATION_ID%~$EKS_ACTIVATION_ID~g; s~
%EKS_ACTIVATION_CODE%~$(echo -n $EKS_ACTIVATION_CODE | base64)~g" eks-connector.yaml
```

> **Important**
>
> 确保激活码采用 base64 格式。

3. 在可以访问集群的终端中，可以运行以下命令应用更新的清单文件：

```
kubectl apply -f eks-connector.yaml
```

4. 将绑定 Amazon EKS Connector 清单和角色的 YAML 文件应用于 Kubernetes 集群后，确认该集群现在已连接。

```
aws eks describe-cluster \
     --name "my-first-registered-cluster" \
     --region AWS_REGION
```

   该输出应包含 `status=ACTIVE`。

5. （可选）向集群添加标签。有关更多信息，请参阅 the section called “标记您的资源”。

后续步骤

如果您对这些步骤有任何疑问，请参阅 the section called “EKS Connector 问题排查”。

要向其他 IAM 主体授予对 Amazon EKS 控制台的访问权限以查看已连接的集群中的 Kubernetes 资源，请参阅the section called “授予对集群的访问权限”。

授予在 Amazon EKS 控制台上查看 Kubernetes 集群资源的权限

向 IAM 主体授予对 Amazon EKS 控制台的访问权限，查看有关在已连接集群上运行的 Kubernetes 资源的信息。

先决条件

您访问 Amazon Web Services 管理控制台 所用的 IAM 主体必须满足以下要求：

- 它必须具有 eks:AccessKubernetesApi IAM 权限。
- Amazon EKS Connector 服务账户应能够模拟集群中的 IAM 主体。这将允许 Amazon EKS Connector 将 IAM 主体映射到 Kubernetes 用户。

创建并应用 Amazon EKS Connector 集群角色

1. 下载 eks-connector 集群角色模板。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml
```

2. 编辑集群角色模板 YAML 文件。将 %IAM_ARN% 参考替换为您的 IAM 主体的 Amazon 资源名称（ARN）。
3. 将 Amazon EKS Connector 集群角色 YAML 应用于您的 Kubernetes 集群。

```
kubectl apply -f eks-connector-clusterrole.yaml
```

要让某个 IAM 主体能够在 Amazon EKS 控制台上查看 Kubernetes 资源，该主体必须与 Kubernetes role 或 clusterrole 关联，并拥有读取这些资源的必要权限。有关更多信息，请参阅 Kubernetes 文档中的使用 RBAC 授权。

要配置 IAM 主体以访问连接的集群

1. 您可以下载以下任一示例清单文件，分别创建 clusterrole 和 clusterrolebinding 或 role 和 rolebinding：

   查看所有命名空间中的 Kubernetes 资源

   - eks-connector-console-dashboard-full-access-clusterrole 集群角色允许访问控制台中可视化的所有命名空间和资源。您可以更改 role、clusterrole 及其对应绑定的名称，然后再将其应用于集群。使用以下命令下载示文件。

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-full-access-group.yaml
     ```

   查看特定命名空间中的 Kubernetes 资源

   - 此文件中的命名空间是 default，因此如果要指定不同命名空间，请编辑该文件，然后应用到集群。使用以下命令下载示文件。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/
eks-connector-console-roles/eks-connector-console-dashboard-restricted-access-
group.yaml
```

2. 编辑完全访问权限或受限访问权限 YAML 文件以将 %IAM_ARN% 参考替换为您的 IAM 主体的 Amazon 资源名称（ARN）。
3. 将完全访问权限或受限访问权限 YAML 文件应用于 Kubernetes 集群。将 YAML 文件的值替换为您自己的值。

```
kubectl apply -f eks-connector-console-dashboard-full-access-group.yaml
```

要查看已连接集群中的 Kubernetes 资源，请参阅 the section called “访问集群资源”。资源选项卡上某些资源类型的数据不适用于已连接的集群。

从 Amazon EKS 控制台注销 Kubernetes 集群

使用完已连接集群后，可以将其注销。取消注册后，该集群将不再在 Amazon EKS 控制台中显示。

您必须拥有以下权限才能调用 deregisterCluster API：

- `eks:DeregisterCluster`
- `ssm:DeleteActivation`
- `ssm:DeregisterManagedInstance`

此过程包括两个步骤：向 Amazon EKS 取消注册集群，在集群中卸载 eks-connector 代理。

注销 Kubernetes 集群

要从 Amazon EKS 连接器取消注册集群，您可以使用以下工具之一：

- the section called “Amazon CLI”
- the section called “Amazon Web Services 管理控制台”
- the section called “`eksctl`”

Amazon CLI

1. 必须安装 Amazon CLI。要进行安装或升级，请参阅安装 Amazon CLI。
2. 确保已创建 Amazon EKS Connector 代理角色。
3. 注销已连接的集群。

```
aws eks deregister-cluster \
    --name my-cluster \
    --region region-code
```

Amazon Web Services 管理控制台

1. 打开 Amazon EKS 控制台。
2. 选择 Clusters (集群)。
3. 请在 Clusters（集群）页面上，选择连接的集群，然后选择 Deregister（注销）。
4. 确认您要注销该层。

`eksctl`

1. 安装 `eksctl` 版本 `0.68` 或更高版本。要安装或对其升级，请参阅 the section called “创建集群（eksctl）”。
2. 确保已创建 Amazon EKS Connector 代理角色。
3. 取消注册已连接的集群：

```
eksctl deregister cluster --name my-cluster
```

清除 Kubernetes 集群中的资源

要卸载 `eks-connector` 代理，请使用以下工具之一：

- the section called “helm”
- the section called “yaml”

helm

运行以下命令来卸载代理。

```
helm -n eks-connector uninstall eks-connector
```

yaml

1. 从 Kubernetes 集群中删除 Amazon EKS Connector YAML 文件。

   ```
   kubectl delete -f eks-connector.yaml
   ```

2. 如果为其他 IAM 主体创建了 `clusterrole` 或 `clusterrolebindings` 来访问集群，请从 Kubernetes 集群中删除它们。

排查 Amazon EKS Connector 问题

本主题介绍您在使用 Amazon EKS Connector 时可能遇到的一些常见错误，包括有关如何解决这些错误的说明和变通方法。

基本问题排查

本节介绍诊断 Amazon EKS Connector 问题的步骤。

检查 Amazon EKS Connector 状态

要检查 Amazon EKS Connector 的状态，请输入：

```
kubectl get pods -n eks-connector
```

检查 Amazon EKS Connector 的日志

Amazon EKS Connector Pod 包含三个容器。要检索所有这些容器的完整日志以便进行查看，请运行以下命令：

- `connector-init`

```
kubectl logs eks-connector-0 --container connector-init -n eks-connector
kubectl logs eks-connector-1 --container connector-init -n eks-connector
```

- connector-proxy

```
kubectl logs eks-connector-0 --container connector-proxy -n eks-connector
kubectl logs eks-connector-1 --container connector-proxy -n eks-connector
```

- connector-agent

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
kubectl exec eks-connector-1 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
```

获取有效的集群名称

Amazon EKS 集群通过一个 Amazon 账户和 Amazon 区域内的 clusterName 唯一标识。如果您在 Amazon EKS 中有多个连接的集群，则可以确认当前 Kubernetes 集群注册到了哪个 Amazon EKS 集群。为此，请输入以下内容以找出当前集群的 clusterName。

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
 | sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9]+.*$/\1/"
kubectl exec eks-connector-1 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
 | sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9]+.*$/\1/"
```

其他命令

以下命令可用于检索排查问题所需的信息。

- 使用以下命令可收集 Amazon EKS Connector 中容器组（pod）使用的映像。

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.containers[*].image}"
 | tr -s '[[:space:]]' '\n'
```

- 使用以下命令可确定运行 Amazon EKS Connector 的节点名称。

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.nodeName}" | tr -s
 '[[:space:]]' '\n'
```

- 运行以下命令可获取您的 Kubernetes 客户端和服务器版本。

```
kubectl version
```

- 运行以下命令可获取有关节点的信息。

```
kubectl get nodes -o wide --show-labels
```

Helm 问题：403 Forbidden

如果在运行 helm install 命令时收到以下错误：

```
Error: INSTALLATION FAILED: unexpected status from HEAD request to https://
public.ecr.aws/v2/eks-connector/eks-connector-chart/manifests/0.0.6: 403 Forbidden
```

您可以运行以下行来修复它：

```
docker logout public.ecr.aws
```

控制台错误：集群卡在待处理状态

如果注册集群之后，集群在 Amazon EKS 控制台中卡在 Pending 状态，可能是因为 Amazon EKS Connector 尚未成功将集群连接到 Amazon。对于已注册的集群，Pending 状态表示未成功建立连接。要解决此问题，请确保您已将清单应用到目标 Kubernetes 集群。如果将其应用于集群，但集群仍处于 Pending 状态，则 `eks-connector` statefulset 可能不正常。要排查此问题，请参阅本主题中的the section called “Amazon EKS Connector 容器组（pod）处于崩溃循环”。

控制台错误：用户“system:serviceaccount:eks-connector:eks-connector”无法在集群范围内模拟 API 组中的资源用户

Amazon EKS Connector 使用 Kubernetes 用户模拟 代表 Amazon Web Services 管理控制台 中的 IAM 主体进行操作。对于从 Amazon `eks-connector` 服务账户访问 Kubernetes API 的每个主体，必须授予权限以使用 IAM ARN 作为其 Kubernetes 用户名来模拟相应 Kubernetes 用户。在以下示例中，IAM ARN 映射到 Kubernetes 用户。

- 来自 Amazon 账户 *111122223333* 的 IAM 用户 *john* 映射到 Kubernetes 用户。IAM 最佳实践建议您向角色而不是用户授予权限。

```
                arn:aws-cn:iam::111122223333:user/john
```

- 来自 Amazon 账户 *111122223333* 的 IAM 角色 *admin* 映射到 Kubernetes 用户：

```
                arn:aws-cn:iam::111122223333:role/admin
```

  结果是 IAM 角色 ARN，而不是 Amazon STS 会话 ARN。

有关如何配置 `ClusterRole` 和 `ClusterRoleBinding` 以授予 `eks-connector` 服务账户模拟映射用户权限的说明，请参阅the section called “授予对集群的访问权限”。确保在模板中将 `%IAM_ARN%` 替换为 Amazon Web Services 管理控制台 IAM 主体的 IAM ARN。

控制台错误：[...] 已禁止：用户 [...] 无法在集群范围内列出 API 组中的列表资源 [...]

考虑以下问题。Amazon EKS Connector 已成功模拟目标 Kubernetes 集群中请求的 Amazon Web Services 管理控制台 IAM 主体。但模拟主体没有 Kubernetes API 操作的 RBAC 权限。

要解决此问题，有两种方法可以向其他用户授予权限。如果您之前通过 Helm 图表安装了 eks-connector，则可以通过运行以下命令轻松授予用户访问权限。将 `userARN1` 和 `userARN2` 替换为 IAM 角色的 ARN 列表，以授予查看 Kubernetes 资源的访问权限：

```
helm upgrade eks-connector oci://public.ecr.aws/eks-connector/eks-connector-chart \
    --reuse-values \
    --set 'authentication.allowedUserARNs={userARN1,userARN2}'
```

或者，作为集群管理员，向各个 Kubernetes 用户授予适当级别的 RBAC 权限。有关更多信息以及示例，请参阅 the section called “授予对集群的访问权限”。

控制台错误：Amazon EKS 无法与您的 Kubernetes 集群 API 服务器进行通信。集群必须处于 ACTIVE（活动）状态才能成功连接。过几分钟再试。

如果 Amazon EKS 服务无法与目标集群中的 Amazon EKS Connector 进行通信，可能是由于以下原因之一导致：

- 目标集群中的 Amazon EKS Connector 运行状况不佳。
- 目标集群与 Amazon 区域之间的连接不佳或连接中断。

要解决此问题，请查看 Amazon EKS Connector 日志。如果您没有看到 Amazon EKS Connector 的错误，请在几分钟后重试连接。如果经常遇到目标集群的高延迟或间歇性连接，请考虑将集群重新注册到离您更近的 Amazon 区域。

Amazon EKS Connector 容器组（pod）处于崩溃循环

可导致 Amazon EKS Connector 容器组（pod）进入 `CrashLoopBackOff` 状态的原因有很多。此问题可能涉及 `connector-init` 容器。检查 Amazon EKS Connector 容器组（pod）的状态。

```
kubectl get pods -n eks-connector
```

示例输出如下。

```
NAME              READY   STATUS                  RESTARTS   AGE
eks-connector-0   0/2     Init:CrashLoopBackOff   1          7s
```

如果您的输出与之前的输出类似，请参阅 the section called “检查 Amazon EKS Connector 的日志” 排查问题。

无法启动 eks-connector：InvalidActivation

初次启动 Amazon EKS Connector 时，它会向 Amazon Web Services 注册 `activationId` 和 `activationCode`。注册可能会失败，进而可能导致 `connector-init` 容器崩溃，并显示类似以下错误。

```
F1116 20:30:47.261469       1 init.go:43] failed to initiate eks-connector:
 InvalidActivation:
```

要排查此问题，请考虑以下原因和建议的修复方法：

- 注册失败可能是因为 `activationId` 和 `activationCode` 不在清单文件中。如果是这种情况，请确保它们是从 `RegisterCluster` API 操作返回的正确值，并且 `activationCode` 位于清单文件中。`activationCode` 已添加到 Kubernetes 密钥中，因此必须为 `base64` 编码。有关更多信息，请参阅 the section called “步骤 1：注册集群”。
- 注册失败可能是因为激活已过期。这是因为，出于安全原因，您必须在注册集群后的三天内激活 Amazon EKS Connector。要解决此问题，请确保在到期日期和时间之前将 Amazon EKS Connector 清单应用到目标 Kubernetes 集群。要确认激活到期日期，请调用 `DescribeCluster` API 操作。

```
aws eks describe-cluster --name my-cluster
```

在以下示例响应中，到期日期和时间记录为 2021-11-12T22:28:51.101000-08:00。

```
{
    "cluster": {
        "name": "my-cluster",
        "arn": "arn:aws-cn:eks:region:111122223333:cluster/my-cluster",
        "createdAt": "2021-11-09T22:28:51.449000-08:00",
        "status": "FAILED",
        "tags": {
        },
        "connectorConfig": {
            "activationId": "00000000-0000-0000-0000-000000000000",
            "activationExpiry": "2021-11-12T22:28:51.101000-08:00",
            "provider": "OTHER",
            "roleArn": "arn:aws-cn:iam::111122223333:role/my-connector-role"
        }
    }
}
```

如果 activationExpiry 已过，则注销集群然后重新注册。执行此操作会生成新的激活信息。

集群节点缺少出站连接

为正常工作，Amazon EKS Connector 需要到多个 Amazon 端点的出站连接。如果没有到目标 Amazon 区域的出站连接，则无法连接私有集群。要解决此问题，您必须添加必要的出站连接。有关连接器要求的信息，请参阅 the section called “Amazon EKS Connector 注意事项”。

Amazon EKS Connector 容器组（pod）处于 **ImagePullBackOff** 状态

如果您运行 get pods 命令且容器组（pod）处于 ImagePullBackOff 状态，则其无法正常工作。如果 Amazon EKS Connector 容器组（pod）处于 ImagePullBackOff 状态，则其无法正常工作。检查 Amazon EKS Connector 容器组（pod）的状态。

```
kubectl get pods -n eks-connector
```

示例输出如下。

```
NAME              READY   STATUS                  RESTARTS   AGE
eks-connector-0   0/2     Init:ImagePullBackOff   0          4s
```

默认 Amazon EKS Connector 清单文件引用来自 Amazon ECR Public Gallery 的映像。目标 Kubernetes 集群可能无法从 Amazon ECR 公开映像浏览馆提取映像。解决 Amazon ECR Public Gallery 映像提取问题，或考虑映像您选择的私有容器注册表中的映像。

Amazon Connector 常见问题

问：Amazon EKS Connector 背后的底层技术是如何工作的？

答：Amazon EKS Connector 基于 Amazon Systems Manager（Systems Manager）代理。Amazon EKS Connector 作为 `StatefulSet` 在您的 Kubernetes 集群上运行。它建立连接并代理集群的 API 服务器与 Amazon Web Services 之间的通信。这样做是为了在 Amazon EKS 控制台中显示集群数据，直到您将集群与 Amazon 断开连接。Systems Manager 代理是一个开源项目。有关此项目的更多信息，请参阅 GitHub 项目页面。

问：我想要连接一个本地部署的 Kubernetes 集群。我需要打开防火墙端口来连接它吗？

答：不，您不需要打开任何防火墙端口。Kubernetes 集群只需要到 Amazon 区域的出站连接。Amazon 服务永远不会访问本地部署网络中的资源。Amazon EKS Connector 在您的集群上运行并启动与 Amazon 的连接。集群注册完成后，Amazon 仅在您从 Amazon EKS 控制台启动操作（需要集群中 Kubernetes API 服务器的信息）后向 Amazon EKS Connector 发出命令。

问：Amazon EKS Connector 将哪些数据从我的集群发送到 Amazon？

答：Amazon EKS Connector 会发送在 Amazon 上注册集群所必需的技术信息。它还发送客户请求的 Amazon EKS 控制台功能的集群和工作负载元数据。Amazon EKS Connector 仅当您从 Amazon EKS 控制台或 Amazon EKS API 启动的操作需要将数据发送到 Amazon 时才会收集或发送此数据。默认情况下，Amazon 不会存储除 Kubernetes 版本号以外的任何数据。它仅在您授权的情况下才会存储数据。

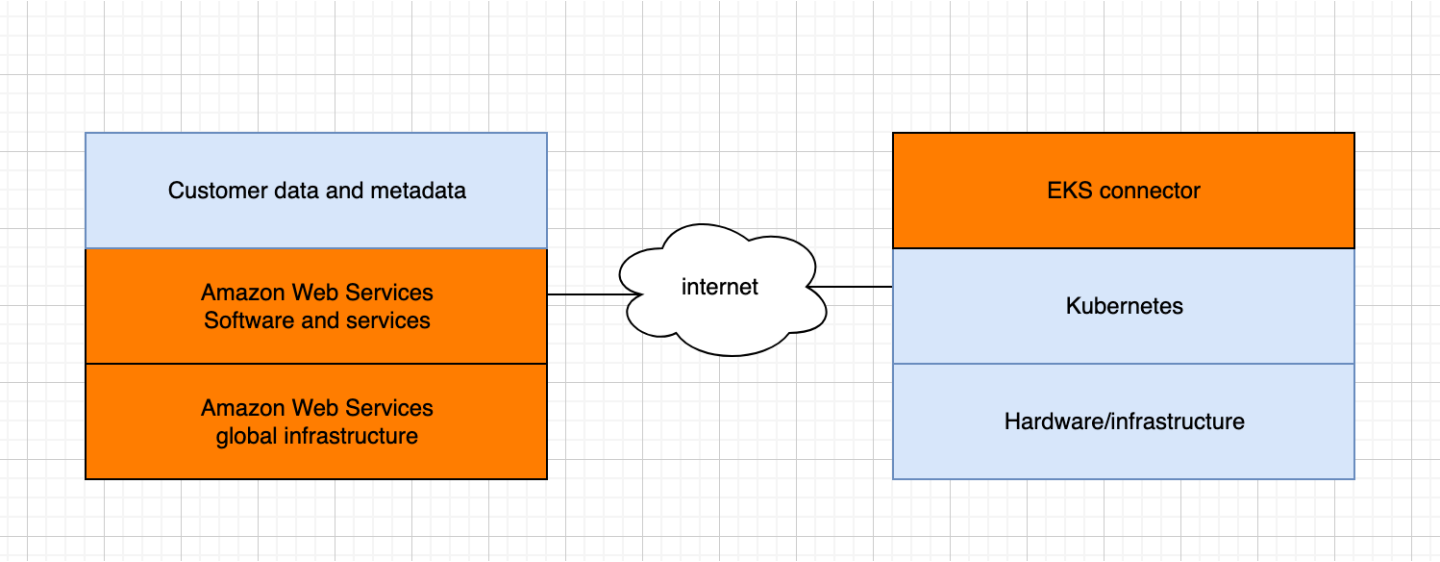
问：我可以连接 Amazon 区域以外的集群吗？

答：可以，您可以将任何位置的集群连接到 Amazon EKS。此外，您的 Amazon EKS 服务可以位于任何 Amazon 公共商业 Amazon 区域。这适用于从集群到目标 Amazon 区域的有效网络连接。我们建议您选择距集群位置最近的 Amazon 区域以优化 UI 性能。例如，如果您的集群在东京运行，请将集群连接到东京的 Amazon 区域（即 `ap-northeast-1` Amazon 区域）以实现低延迟。您可以将任何

位置的集群连接到任何公共商业 Amazon 区域（中国或 GovCloud Amazon 区域除外）中的 Amazon EKS。

了解 Amazon EKS Connector 中的安全性

Amazon EKS Connector 是在您的 Kubernetes 集群上运行的开源组件。此集群可以位于 Amazon 环境之外。这为安全责任创造了额外的注意事项。下面的图表对此配置进行说明。橙色代表 Amazon 责任，蓝色代表客户责任：



本主题介绍了连接的集群在 Amazon 外部时责任模型的差异。

Amazon 责任

- 维护、构建和交付 Amazon EKS Connector，后者是在客户的 Kubernetes 集群上运行并与 Amazon 通信的开源组件。
- 维护连接的 Kubernetes 集群和 Amazon 服务之间的传输层和应用程序层通信安全。

客户责任

- Kubernetes 集群特定的安全性，具体来说如下：
  - Kubernetes 密钥必须得到妥当的加密和保护。
  - 锁定对 `eks-connector` 命名空间的访问权限。
- 配置基于角色的访问控制（RBAC）权限以管理 Amazon 的 IAM 主体访问权限。有关说明，请参阅 the section called “授予对集群的访问权限”。

- 安装和升级 Amazon EKS Connector。
- 维护支持已连接的 Kubernetes 集群的硬件、软件和基础设施。
- 保护他们的 Amazon 账户安全（例如，通过保护您的根用户凭证安全）。

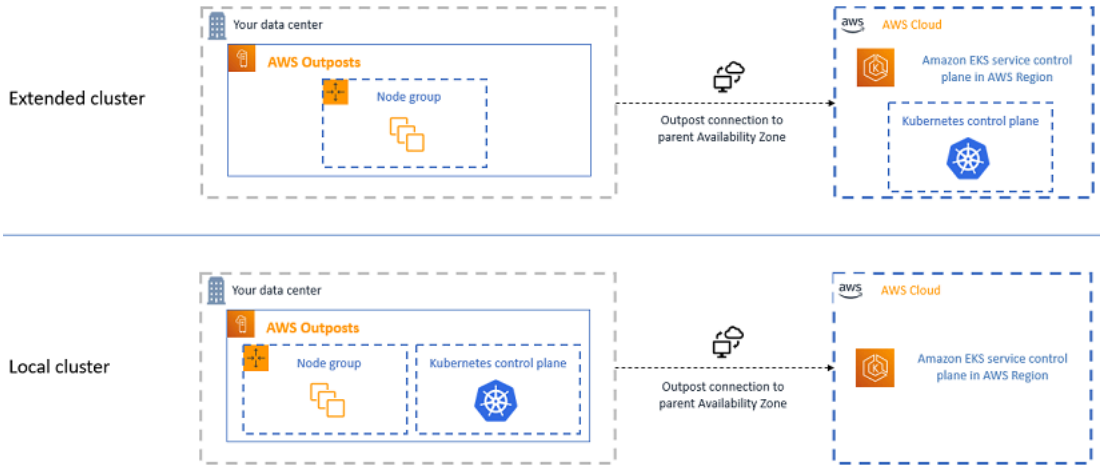
使用 Amazon Outposts 在本地部署 Amazon EKS

您可以使用 Amazon EKS 在 Amazon Outposts 上运行本地 Kubernetes 应用程序。您可以通过以下方式在 Outpost 上部署 Amazon EKS：

- 扩展集群 – 在 Amazon 区域中运行 Kubernetes 控制面板，在您的 Outpost 上运行节点。
- 本地集群 – 在您的 Outpost 上运行 Kubernetes 控制面板和节点。

对于这两个部署选项，Kubernetes 控制面板完全由 Amazon 托管。您可以使用在云中使用的相同 Amazon EKS API、工具和控制台，在 Outpost 上创建和运行 Amazon EKS。

下图显示了这些部署选项。



何时使用每个部署选项

本地和扩展集群都是通用部署选项，可以用于一系列应用程序。

本地集群使您能在 Outpost 上本地运行整个 Amazon EKS 集群。此选项可以降低因临时断开与云的网络连接而造成的应用程序停机风险。光纤中断或天气事件可能会导致网络断开连接。由于整个 Amazon EKS 集群在 Outpost 上本地运行，应用程序仍然可用。您可以在断开与云的网络连接期间执行集群操作。有关更多信息，请参阅 the section called “做好断网准备”。如果您担心从 Outpost 到父 Amazon 区域的网络连接质量，并且需要在整个网络断开连接期间保持高可用性，请使用本地集群部署选项。

扩展集群让您能够节省 Outpost 上的容量，因为 Kubernetes 控制面板在父 Amazon 区域中运行。如果您能投资建立从 Outpost 到 Amazon 区域的可靠、冗余网络连接，则此选项很合适。网络连接的质量对于此选项至关重要。Kubernetes 处理 Kubernetes 控制面板和节点之间网络断开连接的方式可能会

导致应用程序停机。有关 Kubernetes 的行为的更多信息，请参阅 Kubernetes 文档中的调度、抢占和驱逐。

部署选项比较

下表比较了两个选项之间的差异。

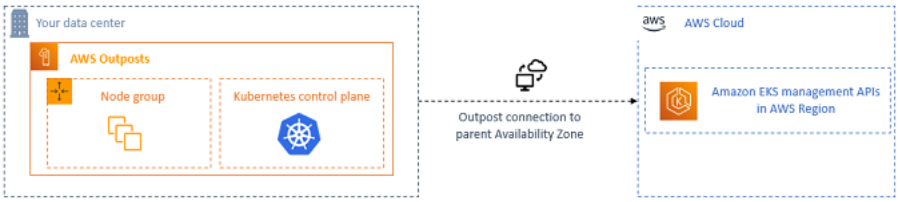
功能	扩展集群	本地集群
Kubernetes 控制面板位置	Amazon 区域	Outpost
Kubernetes 控制面板账户	Amazon 账户	您的账户
区域可用性	请参阅 Service endpoints	美国东部（俄亥俄州）、美国东部（弗吉尼亚州北部）、美国西部（北加利福尼亚）、美国西部（俄勒冈州）、亚太地区（首尔）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲地区（法兰克福）、欧洲地区（爱尔兰）、欧洲地区（伦敦）、中东（巴林）和南美洲（圣保罗）。
Kubernetes 次要版本	eks/latest/userguide/kubernetes-versions.html[Supported Amazon EKS versions, type="documentation"]。	eks/latest/userguide/kubernetes-versions.html[Supported Amazon EKS versions, type="documentation"]。
平台版本	请参阅 EKS platform-versions	请参阅 the section called “EKS 平台版本”
Outpost 外形规格	Outpost 机架	Outpost 机架
用户界面	Amazon Web Services 管理控制台、Amazon CLI、Amazon	Amazon Web Services 管理控制台、Amazon CLI、Amazon

功能	扩展集群	本地集群
	EKS API、`eksctl`、Amazon CloudFormation 和 Terraform	EKS API、`eksctl`、Amazon CloudFormation 和 Terraform
托管策略	AmazonEKSClusterPolicy 和 the section called “Amazon托管策略：AmazonEKSServiceRolePolicy”	AmazonEKSLocalOutpostClusterPolicy 和 the section called “Amazon 托管策略：AmazonEKSLocalOutpostServiceRolePolicy”
集群 VPC 和子网	请参阅 the section called “VPC 和子网要求”	请参阅 the section called “创建 VPC 和子网”
集群终端节点访问	公共或私有，或者两者兼而有之	仅限私有
Kubernetes API 服务器身份验证	Amazon Identity and Access Management（IAM）和 OIDC	IAM 和 `x.509` 证书
节点类型	仅自行管理	仅自行管理
节点计算类型	Amazon EC2 按需	Amazon EC2 按需
节点存储类型	Amazon EBS gp2 和本地 NVMe SSD	Amazon EBS gp2 和本地 NVMe SSD
Amazon EKS 优化版 AMI	Amazon Linux、Windows 和 Bottlerocket	仅限 Amazon Linux
IP 版本	仅限 `IPv4`	仅限 `IPv4`
附加组件	Amazon EKS 附加组件或自行管理的附加组件	自行管理的附加组件
默认容器网络接口	适用于 Kubernetes 的 Amazon VPC CNI 插件	适用于 Kubernetes 的 Amazon VPC CNI 插件
Kubernetes 控制面板日志	Amazon CloudWatch Logs	Amazon CloudWatch Logs

功能	扩展集群	本地集群
负载均衡	使用 Amazon Load Balancer Controller 仅预调配应用程序负载均衡器（无网络负载均衡器）	使用 Amazon Load Balancer Controller 仅预调配应用程序负载均衡器（无网络负载均衡器）
秘密信封加密	请参阅 the section called “启用密钥加密”	不支持
服务账户的 IAM 角色	请参阅 the section called “包含 IRSA 的凭证”	不支持
故障排除	请参阅 故障排除	请参阅 the section called “集群问题排查”

在 Amazon Outposts 上创建本地 Amazon EKS 集群以实现高可用性

您可以使用本地集群在 Amazon Outposts 上本地运行您的整个 Amazon EKS 集群。这有助于降低因临时断开与云的网络连接而造成的应用程序停机风险。光纤中断或天气事件可能会导致此类连接断开的情况。由于整个 Kubernetes 集群在 Outpost 上本地运行，应用程序仍然可用。您可以在断开与云的网络连接期间执行集群操作。有关更多信息，请参阅 the section called “做好断网准备”。下图显示了本地集群部署。



本地集群通常可用于与 Outpost 机架配合使用。

支持的 Amazon 区域

您可以在以下 Amazon 区域创建本地集群：美国东部（俄亥俄州）、美国东部（弗吉尼亚州北部）、美国西部（北加利福尼亚）、美国西部（俄勒冈州）、亚太地区（首尔）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、加拿大（中部）、欧洲地区（法兰克福）、欧洲地区（爱尔兰）、欧洲地区（伦敦）、中东（巴林）和南美洲（圣保罗）。有关受支持的功能的详细信息，请参阅 the section called “部署选项比较”。

在 Amazon Outposts 上部署 Amazon EKS 集群

本主题概述了在 Outpost 上运行本地集群时需要考虑的事项。本主题还提供了有关如何在 Outpost 上部署本地集群的说明。

> **Important**
>
> - 相关的 Amazon EKS 文档中没有重复这些注意事项。若其他 Amazon EKS 文档主题与本文的考虑事项冲突，则遵从本文的考虑事项。
> - 这些注意事项可能会经常更改。因此，我们建议您定期查看本主题。
> - 很多注意事项与在 Amazon 云上创建集群的注意事项不同。

- 本地集群仅支持 Outpost 机架。单个本地集群可跨构成单个逻辑 Outpost 的多个物理 Outpost 机架运行。单个本地集群不能跨多个逻辑 Outpost 运行。每个逻辑 Outpost 都有一个 Outpost ARN。
- 本地集群运行和管理 Outpost 上您账户中的 Kubernetes 控制面板。您不能在 Kubernetes 控制面板实例上运行工作负载或对 Kubernetes 控制面板组件进行修改。这些节点由 Amazon EKS 服务管理。对 Kubernetes 控制面板的更改不会在自动 Amazon EKS 管理操作（如修补）期间一直持续存在。
- 本地集群支持自行管理的附加组件和自行管理的 Amazon Linux 节点组。适用于 Kubernetes 的 Amazon VPC CNI 插件、kube-proxy 和 CoreDNS 附加组件会自动安装在本地集群上。
- 本地集群需要使用 Outpost 上的 Amazon EBS。您的 Outpost 必须拥有可用于 Kubernetes 控制面板存储的 Amazon EBS。Outpost 仅支持 Amazon EBS gp2 卷。
- 使用 Amazon EBS CSI 驱动程序支持 Amazon EBS 支持的 Kubernetes `PersistentVolumes`。
- 本地集群的控制面板实例在堆叠式高可用性拓扑中设置。在三个控制面板实例中，必须有两个示例始终处于正常运行状态才能进行仲裁。如果失去了仲裁，请联系 Amazon 支持人员，因为启用新托管实例需要进行部分服务端操作。

先决条件

- 熟悉 Outposts 部署选项、根据容量因素为 Amazon Outposts 上的 Amazon EKS 集群选择实例类型和置放群组以及 VPC 要求和注意事项。
- 一个现有的 Outpost。有关更多信息，请参阅什么是 Amazon Outposts。
- 计算机或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以

将 kubectl 的 1.28、1.29 或 1.30 版本与之配合使用。要安装或升级 kubectl，请参阅 the section called “设置 kubectl 和 eksctl”。

- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 2.12.3 或更高版本，或版本 1.27.160 或更高版本。要查看当前版本，请使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。yum、apt-get 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。
- 具有 create 和 describe Amazon EKS 集群权限的 IAM 主体（用户或角色）。有关更多信息，请参阅the section called “在 Outpost 上创建本地 Kubernetes 集群”和the section called “列出或描述所有集群”。

创建本地 Amazon EKS 集群后，创建集群的 IAM 主体将永久添加。主体作为管理员专门添加到 Kubernetes RBAC 授权表。该实体具有 system:masters 权限。此实体的身份在您的集群配置中不可见。因此，重要的是要注意创建集群的实体并确保永远不会删除它。最初，仅创建服务器的主体可以使用 kubectl 调用 Kubernetes API 服务器。如果使用控制台创建集群，请确保在集群上运行 kubectl 命令时，相同的 IAM 凭证位于 Amazon SDK 凭证链中。创建集群后，您可以向其他 IAM 主体授予对集群的访问权限。

创建 Amazon EKS 本地集群

您可以使用本页中描述的下列工具创建本地集群：

- the section called “eksctl”
- the section called “Amazon Web Services 管理控制台”

您也可以使用 Amazon CLI、Amazon EKS API、Amazon SDK、Amazon CloudFormation 或 Terraform 在 Outposts 上创建集群。

eksctl

使用 **eksctl** 创建本地集群

1. 在您的设备或 Amazon CloudShell 上安装 0.215.0 版或更高版本的 eksctl 命令行工具。要安装或更新 eksctl，请参阅 eksctl 文档中的 Installation。

2. 将后续内容复制到您的设备。替换以下值，然后运行修改后的命令以创建 `outpost-control-plane.yaml` 文件：

   - 将 *region-code* 替换为您要在其中创建集群的受支持的 Amazon 区域。
   - 将 *my-cluster* 替换为您的集群名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   - 将 *vpc-ExampleID1* 和 *subnet-ExampleID1* 替换为您的现有 VPC 和子网的 ID。VPC 和子网必须符合在 Amazon Outposts 上为上的 Amazon EKS 集群创建 VPC 和子网中的要求。
   - 将 *uniqueid* 替换为您的 Outpost 的 ID。
   - 将 *m5.large* 替换为您的 Outpost 上可用的实例类型。在选择实例类型之前，请参阅 the section called “容量注意事项”。将部署三个控制面板实例。您无法更改此数字。

   ```
   cat >outpost-control-plane.yaml <<EOF
   apiVersion: eksctl.io/v1alpha5
   kind: ClusterConfig

   metadata:
     name: my-cluster
     region: region-code
     version: "1.33"

   vpc:
     clusterEndpoints:
       privateAccess: true
     id: "vpc-vpc-ExampleID1"
     subnets:
       private:
         outpost-subnet-1:
           id: "subnet-subnet-ExampleID1"

   outpost:
     controlPlaneOutpostARN: arn:aws-cn:outposts:region-code:111122223333:outpost/op-
   uniqueid
     controlPlaneInstanceType: m5.large
   EOF
   ```

   有关所有可用选项和默认设置的完整列表，请参阅 Amazon Outposts 支持和 `eksctl` 文档中的配置文件架构。

3. 使用您在上一步中创建的配置文件创建集群。eksctl 将在您的 Outpost 上创建一个 VPC 和一个子网，以在其中部署集群。

```
eksctl create cluster -f outpost-control-plane.yaml
```

集群预配置需要几分钟时间。在创建集群时，将显示几行输出。输出的最后一行类似于以下示例行。

```
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

> **Tip**
>
> 要查看在使用 eksctl 创建集群时可指定的大多数选项，请使用 eksctl create cluster --help 命令。要查看所有可用的选项，请使用 config 文件。有关更多信息，请参阅 eksctl 文档中的使用配置文件和配置文件架构。您可以在 GitHub 上查找配置文件示例。

eksctl 命令自动为创建集群的 IAM 主体（用户或角色）创建了访问条目，并授予 IAM 主体对集群上 Kubernetes 对象的管理员权限。如果不希望集群创建者拥有集群上 Kubernetes 对象的管理员访问权限，请在之前的配置文件中添加以下文本：bootstrapClusterCreatorAdminPermissions: false（与 metadata、vpc 和 outpost 的级别相同）。如果添加了该选项，则在创建集群后，您需要为至少一个 IAM 主体创建访问条目，否则任何 IAM 主体都无法访问集群上的 Kubernetes 对象。

Amazon Web Services 管理控制台

使用 Amazon Web Services 管理控制台创建集群

1. 您需要有满足 Amazon EKS 要求的现有 VPC 和子网。有关更多信息，请参阅 the section called “创建 VPC 和子网”。
2. 如果您已经拥有本地集群 IAM 角色，或者您将使用 eksctl 创建集群，则可以跳过此步骤。默认情况下，eksctl 会为您创建角色。

   a. 运行以下命令以创建 IAM 信任策略 JSON 文件。

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 创建 Amazon EKS 集群 IAM 角色。要创建 IAM 角色，必须为正在创建角色的 IAM 主体分配 `iam:CreateRole` 操作（权限）。

```
aws iam create-role --role-name myAmazonEKSLocalClusterRole --assume-role-policy-
document file://"eks-local-cluster-role-trust-policy.json"
```

c. 将名为 AmazonEKSLocalOutpostClusterPolicy 的 Amazon EKS 托管策略附加到该角色。要将 IAM 策略附加到某个 IAM 主体，必须为附加该策略的主体分配以下 IAM 操作（权限）之一：`iam:AttachUserPolicy` 或 `iam:AttachRolePolicy`。

```
aws iam attach-role-policy --policy-arn arn:aws-cn:iam::aws:policy/
AmazonEKSLocalOutpostClusterPolicy --role-name myAmazonEKSLocalClusterRole
```

3. 打开 Amazon EKS 控制台。
4. 在控制台屏幕的顶部，确保您已选择受支持的 Amazon 区域。
5. 请选择 Add cluster（添加集群），然后选择 Create（创建）。
6. 在 Configure cluster（配置集群）页面上，为以下字段输入或选择值：
   - Kubernetes 控制面板位置 – 选择 Amazon Outposts。
   - Outpost ID - 选择您要在其上创建控制面板的 Outpost 的 ID。
   - Instance type（实例类型）– 选择一个实例类型。仅显示您的 Outpost 中可用的实例类型。在下拉列表中，每种实例类型都将描述为实例类型推荐多少个节点。在选择实例类型之前，请参阅 the section called “容量注意事项”。所有副本都将使用相同实例类型进行部署。在创建您的集群后，您将无法更改实例类型。将部署三个控制面板实例。您无法更改此数字。
   - Name（名称）– 集群的名称。它在您的 Amazon 账户中必须是唯一的。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。

- Kubernetes 版本 – 选择要用于集群的 Kubernetes 版本。建议选择最新版本，除非您需要使用早期版本。
- 集群服务角色 – 选择您在上一步创建的 Amazon EKS 集群 IAM 角色，允许 Kubernetes 控制面板管理 Amazon 资源。
- Kubernetes 集群管理员访问权限 – 如果您希望创建集群的 IAM 主体（角色或用户）拥有集群上 Kubernetes 对象的管理员访问权限，请接受默认设置（允许）。Amazon EKS 为 IAM 主体创建访问条目，并向集群管理员授予访问条目的权限。有关访问条目的更多信息，请参阅 the section called “访问条目”。

  如果您希望与创建集群的主体不同的 IAM 主体拥有 Kubernetes 集群对象的管理员访问权限，请选择“不允许”选项。创建集群后，任何拥有创建访问条目的 IAM 权限的 IAM 主体都可以为需要访问 Kubernetes 集群对象的任何 IAM 主体添加访问条目。如需了解所需的 IAM 权限，请参阅《服务授权参考》中的 Amazon Elastic Kubernetes Service 定义的操作。如果您选择“不允许”选项并且不创建任何访问条目，则没有 IAM 主体有权访问集群上的 Kubernetes 对象。
- Tags（标签）–（可选）向集群添加任何标签。有关更多信息，请参阅 the section called “标记您的资源”。完成此页面后，请选择下一步。

7. 在 Specify networking (指定联网) 页面上，为以下字段选择值：
   - VPC – 选择现有 VPC。VPC 必须有足够数量的 IP 地址来供集群、任何节点和您想要创建的其他 Kubernetes 资源使用。您的 VPC 必须符合 VPC 要求和注意事项中的要求。
   - Subnets（子网）– 预设情况下，已预先选中在之前字段中指定的 VPC 中的所有可用子网。您选择的子网必须符合子网要求和注意事项中的要求。
   - Security groups（安全组）–（可选）指定您希望 Amazon EKS 将之与其创建的网络接口关联的一个或多个安全组。Amazon EKS 会自动创建安全组，以实现集群与 VPC 之间的通信。Amazon EKS 将此安全组以及您选择的任何安全组与它创建的网络接口关联起来。有关 Amazon EKS 创建的集群安全组的更多信息，请参阅 the section called “安全组要求”。您可以修改 Amazon EKS 创建的集群安全组中的规则。如果您选择添加自己的安全组，则无法更改集群创建后选择的安全组。要使本地主机与集群端点进行通信，您必须允许来自集群安全组的入站流量。对于没有传入和传出互联网连接（也被称为私有集群）的集群，您必须执行以下操作之一：
     - 添加与所需 VPC 端点关联的安全组。有关所需端点的更多信息，请参阅子网对 Amazon 服务的访问权限中的the section called “使用接口 VPC 端点”。
     - 修改 Amazon EKS 创建的安全组，以允许来自与 VPC 端点关联的安全组的流量。完成此页面后，请选择下一步。
8. 在配置可观测性页面上，您可以选择要开启的指标和控制面板日志记录选项。默认情况下，每种日志类型都处于关闭状态。

- 有关 Prometheus 指标选项的更多信息，请参阅the section called “第 1 步：开启 Prometheus 指标”。
- 有关控制面板日志记录选项的更多信息，请参阅 the section called “控制面板日志”。完成此页面后，请选择下一步。

9. 在 Review and create (审核和创建) 页面上，审核您在之前页面输入或选择的信息。如果需要进行更改，请选择 Edit（编辑）。在您感到满意后，选择创建。Status（状态）字段在预置集群时显示 CREATING（正在创建）。

   集群预配置需要几分钟时间。

查看 Amazon EKS 本地集群

1. 在创建您的集群后，您可以查看已创建的 Amazon EC2 控制面板实例。

   ```
   aws ec2 describe-instances --query 'Reservations[*].Instances[*].{Name:Tags[?
   Key==`Name`]|[0].Value}' | grep my-cluster-control-plane
   ```

   示例输出如下。

   ```
   "Name": "my-cluster-control-plane-id1"
   "Name": "my-cluster-control-plane-id2"
   "Name": "my-cluster-control-plane-id3"
   ```

   每个实例都将受到 `node-role.eks-local.amazonaws.com/control-plane` 的污染，这样就不会在控制面板实例上安排任何工作负载。有关污点的更多信息，请参阅 Kubernetes 文档中的污点和容忍度。Amazon EKS 将持续监控本地集群的状态。我们将执行自动管理操作，如安全补丁和修复运行状况不佳的实例。当本地集群与云断开连接时，我们会完成操作，以确保在重新连接后将集群修复到运行状况正常的状态。

2. 如果您使用 `eksctl` 创建集群，则可以跳过此步骤。`eksctl` 会为您完成此步骤。通过向 `kubectl config` 文件添加新上下文来启用 `kubectl` 与您的集群通信。有关如何创建或更新文件的说明，请参阅 the section called “使用 kubectl 访问集群”。

   ```
   aws eks update-kubeconfig --region region-code --name my-cluster
   ```

   示例输出如下。

```
Added new context arn:aws-cn:eks:region-code:111122223333:cluster/my-cluster to /
home/username/.kube/config
```

3. 要连接到本地集群的 Kubernetes API 服务器，您必须有权访问子网的本地网关，或从 VPC 内部进行连接。有关将 Outpost 机架连接到您的本地网络的更多信息，请参阅《Amazon Outposts 用户指南》中的机架的本地网关的工作原理。如果使用直接 VPC 路由且 Outpost 子网具有通向本地网关的路由，则 Kubernetes 控制面板实例的私有 IP 地址将在您的本地网络内自动广播。本地集群的 Kubernetes API 服务器端点托管在 Amazon Route 53（Route 53）中。API 访问端点可由公有 DNS 服务器解析至 Kubernetes API 服务器的私有 IP 地址。

   本地集群的 Kubernetes 控制面板实例将配置具有固定私有 IP 地址的静态弹性网络接口，这些固定私有 IP 地址在整个集群生命周期中都不会更改。在网络连接断开期间，与 Kubernetes API 服务器互动的设备可能无法连接到 Route 53。如果是这种情况，我们建议使用静态私有 IP 地址配置 `/etc/hosts` 以实现持续运行。我们还建议设置本地 DNS 服务器，并将其连接到 Outpost。有关更多信息，请参阅 Amazon Outposts 文档。运行以下命令以确认已建立与集群的通信。

   ```
   kubectl get svc
   ```

   示例输出如下。

   ```
   NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
   kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
   ```

4. （可选）当本地集群与 Amazon 云断开连接时，对其进行身份验证测试。有关说明，请参阅the section called “做好断网准备”。

内部资源

Amazon EKS 将在您的集群上创建以下资源。这些资源将供 Amazon EKS 内部使用。为使您的集群正常运行，请不要编辑或修改这些资源。

- 以下镜像容器组（pod）：
  - `aws-iam-authenticator-`*`node-hostname`*
  - `eks-certificates-controller-`*`node-hostname`*
  - `etcd-`*`node-hostname`*
  - `kube-apiserver-`*`node-hostname`*
  - `kube-controller-manager-`*`node-hostname`*

  - kube-scheduler-*node-hostname*
- 以下自行管理的附加组件：
  - kube-system/coredns
  - kube-system/ kube-proxy（在您添加第一个节点之前不会创建）
  - kube-system/aws-node（在您添加第一个节点之前不会创建）。本地集群将使用适用于 Kubernetes 的 Amazon VPC CNI 插件进行集群联网。不要更改控制面板实例（名为 aws-node-controlplane-* 的容器组 (pod)）的配置。您可以使用一些配置变量在插件创建新网络接口时更改默认值。有关更多信息，请参阅 GitHub 上的文档。
- 以下服务：
  - default/kubernetes
  - kube-system/kube-dns
- 名为 eks.system 的 PodSecurityPolicy
- 名为 eks:system:podsecuritypolicy 的 ClusterRole
- 名为 eks:system 的 ClusterRoleBinding
- 除集群安全组以外，Amazon EKS 还会在您名为 eks-local-internal-do-not-use-or-edit-*cluster-name*-*uniqueid* 的 Amazon 账户中创建一个安全组。此安全组允许流量在控制面板实例上运行的 Kubernetes 组件之间自由流动。

向您建议的后续步骤：

- 向创建集群的 IAM 主体授予所需的权限，以便在 Amazon Web Services 管理控制台中查看 Kubernetes 资源
- 授予 IAM 实体访问您的集群的权限。如果想让实体在 Amazon EKS 控制台中查看 Kubernetes 资源，请向实体授予所需的权限。
- 为集群配置日志记录。
- 熟悉网络断开连接期间发生的情况。
- 将节点添加到集群。
- 考虑为您的 etcd 制定备份计划。Amazon EKS 不支持为本地集群自动备份和恢复 etcd。有关更多信息，请参阅 Kubernetes 文档中的备份 etcd 集群。两个主要选项使用 etcdctl 自动拍摄快照或使用 Amazon EBS 存储卷备份。

了解适用于 Amazon Outposts 的 Kubernetes 和 Amazon EKS 平台版本

本地集群平台版本表示 Amazon Outposts 上的 Amazon EKS 集群的功能。这些版本包括在 Kubernetes 控制面板上运行的组件，启用了哪些 Kubernetes API 服务器标志，以及当前的 Kubernetes 补丁版本。每个 Kubernetes 次要版本都有一个或多个关联的 平台版本。不同 Kubernetes 次要版本对应的平台版本相互独立。本地集群和云中的 Amazon EKS 集群的平台版本是独立的。

当新的 Kubernetes 次要版本可用于本地集群（如 `1.31`）时，该 Kubernetes 次要版本的初始平台版本将从 `eks-local-outposts.1` 开始。不过，Amazon EKS 会定期发布新平台版本来支持新的 Kubernetes 控制层面设置并提供安全修复。

当新的本地集群平台版本可用于次要版本时：

- 平台版本号将递增 (`eks-local-outposts.n+1`)。
- Amazon EKS 会自动将所有现有本地集群更新到其对应的 Kubernetes 次要版本的最新平台版本。现有平台版本的自动更新将以增量方式推出。推出过程包括逐一替换在 Outpost 上运行的托管型 Kubernetes 控制面板实例，直到全部 3 个实例都被新实例取代。
- 如果存在服务中断风险，Kubernetes 控制面板实例替换过程会停止进行。只有在 2 个 Kubernetes 控制面板实例运行状况良好且作为集群节点通过了所有就绪条件时，Amazon EKS 才会尝试替换另外一个实例。
- 平台版本推出过程通常不到 30 分钟就可完成。如果集群在较长一段时间内保持 `UPDATING` 状态，请参阅the section called “集群问题排查”，向 Amazon Support 支持部门寻求帮助。若未得到 Amazon Support 支持部门的指示，切勿手动终止 Kubernetes 控制面板实例。
- Amazon EKS 可能会发布具有对应的修补程序版本的新工作线程 AMI。对于单个 Kubernetes 次要版本，在 Kubernetes 控制面板与节点 AMI 之间的所有补丁版本都兼容。

新平台版本不会引发破坏性的更改或导致服务中断。

本地集群始终使用指定 Kubernetes 版本的最新可用平台版本 (`eks-local-outposts.n`) 来创建。

下列各表列出了当前和最新的平台版本。

要接收此特定文档页面的所有源文件更改通知，您可以通过 RSS 阅读器订阅以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/outposts/
eks-outposts-platform-versions.adoc.atom
```

Kubernetes 版**1.31**

所有 1.31 平台版本都启用了以下准入控制器：CertificateApproval、CertificateSigning、CertificateSubjectRestriction、Cluste
和 ValidatingAdmissionWebhook。

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.31.12	eks-local-outposts.5	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.31.10。AmazonIAM 身份验证器已更新为 v0.7.4。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.20.2。已将 Bottlerocket 版本更新为 v1.47.0。	2025 年 10 月 3 日
1.31.9	eks-local-outposts.4	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.31.9。AmazonIAM 身份验证器已更新为 v0.7.2。适用于 Kubernetes 的 Amazon VPC CNI 插件已更新到 v1.20.0；Bottlerocket 版本也已更新到 v1.43.0。	2025 年 8 月 9 日
1.31.7	eks-local-outposts.3	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为	2025 年 6 月 19 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
		v1.31.9。AmazonIAM 身份验证器已更新为 v0.7.1。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.5。已将 Bottlerocket 版本更新为 v1.40.0。	
1.31.6	eks-local-outposts.2	新的平台版本（包括安全修复和增强功能）。已将 Bottlerocket 版本更新为 v1.36.0。	2025 年 4 月 24 日
1.31.6	eks-local-outposts.1	Outpost 上适用于本地 Amazon EKS 集群的 Kubernetes 版本 v1.31 的初始版本。	2025 年 4 月 9 日

Kubernetes 版**1.30**

所有 1.30 平台版本都启用了以下准入控制器：CertificateApproval、CertificateSigning、CertificateSubjectRestriction、Cluste
和 ValidatingAdmissionWebhook。

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.30.14	eks-local-outposts.7	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.30.14。AmazonIAM 身份验证器已更	2025 年 10 月 3 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
		新为 v0.7.4。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.20.2。已将 Bottlerocket 版本更新为 v1.47.0。	
1.30.13	eks-local-outposts.6	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.30.13。AmazonIAM 身份验证器已更新为 v0.7.2。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.20.0。已将 Bottlerocket 版本更新为 v1.43.0。	2025 年 8 月 9 日
1.30.11	eks-local-outposts.5	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.30.11。AmazonIAM 身份验证器已更新为 v0.7.1。适用于 Kubernetes 的 Amazon VPC CNI 插件已更新到 v1.19.5；Bottlerocket 版本也已更新到 v1.40.0。	2025 年 6 月 19 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.30.10	eks-local-outposts.4	新的平台版本（包括安全修复和增强功能）。已将 Bottlerocket 版本更新为 v1.36.0。	2025 年 4 月 24 日
1.30.10	eks-local-outposts.3	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.30.10。AmazonIAM 身份验证器已更新为 v0.6.29。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.2。CoreDNS 已更新至 v1.11.4。Amazon 云控制器管理器已更新至 v1.30.8。已将 Bottlerocket 版本更新为 v1.34.0。	2025 年 3 月 27 日
1.30.7	eks-local-outposts.2	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.30.7。AmazonIAM 身份验证器已更新为 v0.6.28。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.0。已将 Bottlerocket 版本更新为 v1.29.0。	2025 年 1 月 10 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.30.5	eks-local-outposts.1	Outpost 上适用于本地 Amazon EKS 集群的 Kubernetes 版本 v1.30 的初始版本。	2024 年 11 月 13 日

Kubernetes 版1.29

所有 1.29 平台版本都启用了以下准入控制器：CertificateApproval、CertificateSigning、CertificateSubjectRestriction、Cluste
和 ValidatingAdmissionWebhook。

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
v1.29.15	eks-local-outposts.10	搭载安全补丁与功能增强的全新平台版本。AmazonIAM 身份验证器已更新为 v0.7.4。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.20.2。已将 Bottlerocket 版本更新为 v1.47.0。	2025 年 10 月 3 日
v1.29.15	eks-local-outposts.9	搭载安全补丁与功能增强的全新平台版本。AmazonIAM 身份验证器已更新为 v0.7.2。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.20.0。	2025 年 8 月 9 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
		已将 Bottlerocket 版本更新为 v1.43.0。	
v1.29.15	eks-local-outposts.8	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.29.15。AmazonIAM 身份验证器已更新为 v0.7.1。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.5。已将 Bottlerocket 版本更新为 v1.40.0。	2025 年 6 月 19 日
v1.29.14	eks-local-outposts.7	新的平台版本（包括安全修复和增强功能）。已将 Bottlerocket 版本更新为 v1.36.0。	2025 年 3 月 24 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
v1.29.14	eks-local-outposts.6	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.29.14。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.2。CoreDNS 已更新至 v1.11.4。Amazon 云控制器管理器已更新至 v1.29.8。已将 Bottlerocket 版本更新为 v1.34.0。	2025 年 3 月 27 日
v1.29.11	eks-local-outposts.5	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.29.11。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.0。已将 CoreDNS 映像更新为 v1.11.3。已将 Bottlerocket 版本更新为 v1.29.0。	2025 年 1 月 10 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.29.9	eks-local-outposts.4	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.29.9。AmazonIAM 身份验证器已更新为 v0.6.26。已将 Bottlerocket 版本更新为 v1.26.0。	2024 年 11 月 8 日
1.29.6	eks-local-outposts.3	新的平台版本（包括安全修复和增强功能）。已将 Bottlerocket 版本更新为 v1.22.0。	2024 年 10 月 22 日
1.29.6	eks-local-outposts.2	新的平台版本（包括安全修复和增强功能）。已将 Bottlerocket 版本更新为 v1.21.0。	2024 年 8 月 27 日
1.29.6	eks-local-outposts.1	Outpost 上适用于本地 Amazon EKS 集群的 Kubernetes 版本 v1.29 的初始版本。	2024 年 8 月 20 日

Kubernetes 版**1.28**

所有 1.28 平台版本都启用了以下准入控制器：CertificateApproval、CertificateSigning、CertificateSubjectRestriction、Cluste
和 ValidatingAdmissionWebhook。

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.28.15	eks-local-outposts.17	搭载安全补丁与功能增强的全新平台版本。AmazonIAM 身份验证器已更新为 v0.7.4。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.20.2。已将 Bottlerocket 版本更新为 v1.47.0。	2025 年 10 月 3 日
1.28.15	eks-local-outposts.16	搭载安全补丁与功能增强的全新平台版本。AmazonIAM 身份验证器已更新为 v0.7.2。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.20.0。已将 Bottlerocket 版本更新为 v1.43.0。	2025 年 8 月 9 日
1.28.15	eks-local-outposts.15	搭载安全补丁与功能增强的全新平台版本。AmazonIAM 身份验证器已更新为 v0.7.1。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.5。已将 CoreDNS 更新为 v1.11.4。已将	2025 年 6 月 19 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
		Bottlerocket 版本更新为 v1.40.0。	
1.28.15	eks-local-outposts.14	新的平台版本（包括安全修复和增强功能）。已将 Bottlerocket 版本更新为 v1.36.0。	2025 年 4 月 24 日
1.28.15	eks-local-outposts.13	已更新具有安全修复和增强功能 kube-proxy v1.28.15 版的新平台版本。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.0。Amazon云控制器管理器已更新至 v1.28.11。	2025 年 3 月 27 日
1.28.15	eks-local-outposts.12	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.28.15。适用于 Kubernetes 的 Amazon VPC CNI 插件升级到 v1.19.0。已将 Bottlerocket 版本更新为 v1.29.0。	2025 年 1 月 10 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.28.14	eks-local-outposts.11	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.28.14。AmazonIAM 身份验证器已更新为 v0.6.26。已将 CoreDNS 映像更新为 v1.11.1。已将 Bottlerocket 版本更新为 v1.26.0。	2024 年 11 月 8 日
1.28.10	eks-local-outposts.10	新的平台版本（包括安全修复和增强功能）。已将 Bottlerocket 版本更新为 v1.22.0。	2024 年 10 月 22 日
1.28.10	eks-local-outposts.9	新的平台版本（包括安全修复和增强功能）。已将 Bottlerocket 版本更新为 v1.21.0。	2024 年 8 月 27 日
1.28.10	eks-local-outposts.8	新的平台版本（包括安全修复和增强功能）。	2024 年 7 月 30 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.28.10	eks-local-outposts.6	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.28.10。AmazonIAM 身份验证器已更新为 v0.6.20。已将 Bottlerocket 版本更新为 v1.20.2。	2024 年 6 月 19 日
1.28.6	eks-local-outposts.5	已将 Bottlerocket 版本更新为 v1.19.3（其中包含最新的错误修复），以支持 Outposts 中的本地启动。	2024 年 4 月 18 日
1.28.6	eks-local-outposts.4	新的平台版本（包括安全修复和增强功能）。已恢复 Outposts 中的支持或本地启动。出于兼容性考虑，已将 Bottlerocket 版本降级为 v1.15.1。	2024 年 4 月 2 日
1.28.6	eks-local-outposts.3	新的平台版本（包括安全修复和增强功能）。	2024 年 3 月 22 日

Kubernetes 版本	Amazon EKS 平台版本	发行说明	发行日期
1.28.6	eks-local-outposts.2	已将具有安全修复和增强功能 kube-proxy 的新平台版本更新为 v1.28.6。AmazonIAM 身份验证器已更新为 v0.6.17。出于兼容性考虑，已将适用于 Kubernetes 的 Amazon VPC CNI 插件降级为 v1.13.2。已将 Bottlerocket 版本更新为 v1.19.2。	2024 年 3 月 8 日
1.28.1	eks-local-outposts.1	Outpost 上适用于本地 Amazon EKS 集群的 Kubernetes 版本 v1.28 的初始版本。	2023 年 10 月 4 日

为 Amazon Outposts 上的 Amazon EKS 集群创建 VPC 和子网

在您创建本地集群时，您将指定 VPC 以及在 Outpost 上运行的至少一个私有子网。本主题概述适用于本地集群的 VPC 和子网的要求和注意事项。

VPC 要求和注意事项

在您创建本地集群时，您指定的 VPC 必须满足以下要求和注意事项：

- 请确保 VPC 具有足够的 IP 地址用于本地集群、任何节点以及您要创建的其他 Kubernetes 资源。如果要使用的 VPC 没有足够的 IP 地址，请增加可用的 IP 地址数量。您可以通过关联其他无类别域间路由（CIDR）块与 VPC 来执行此操作。您可以在创建集群之前或之后将私有 (RFC 1918) 和公有（非 RFC 1918）CIDR 块关联到您的 VPC。集群最多可能需要 5 个小时的时间来识别与 VPC 关联的 CIDR 块。
- VPC 无法分配 IP 前缀或 IPv6 CIDR 块。由于这些限制，为带前缀的 Amazon EKS 节点分配更多 IP 地址和the section called “IPv6”中包含的信息不适用于您的 VPC。

- VPC 已启用 DNS 主机名和 DNS 解析。如果没有这些功能，本地集群将无法创建，因此您需要启用这些功能，然后重新创建集群。有关更多信息，请参阅《Amazon VPC 用户指南》中的 VPC 的 DNS 属性。
- 要通过本地网络访问本地集群，VPC 必须与 Outpost 的本地网关路由表关联。有关更多信息，请参阅《Amazon Outposts 用户指南》中的 VPC 关联。

子网要求和注意事项

创建集群时，请指定至少一个私有子网。如果您指定多个子网，Kubernetes 控制面板实例将均匀分布在这些子网中。如果指定多个子网，则这些子网必须存在于同一 Outpost 上。此外，子网还必须具有适当的路由和安全组权限，以便相互通信。创建本地集群时，您指定的子网必须满足以下要求：

- 子网均位于同一逻辑 Outpost 上。
- 这些子网总共具有至少三个可用 IP 地址用于 Kubernetes 控制面板实例。如果指定三个子网，则每个子网必须具有至少一个可用 IP 地址。如果指定两个子网，则每个子网必须具有至少两个可用 IP 地址。如果指定一个子网，则该子网必须具有至少三个可用 IP 地址。
- 子网具有通向 Outpost 机架的本地网关的路由，用于通过您的本地网络访问 Kubernetes API 服务器。如果子网没有通向 Outpost 机架的本地网关的路由，则您必须从 VPC 中与 Kubernetes API 服务器通信。
- 子网必须使用基于 IP 地址的命名。Amazon EKS 不支持 Amazon EC2 基于资源的命名。

子网对 Amazon 服务的访问权限

Outposts 上本地集群的私有子网必须能够与区域性 Amazon 服务通信。您可以通过使用 NAT 网关进行出站互联网访问来实现此目的，或者，如果您想在 VPC 内保持所有流量的私密性，则使用接口 VPC 端点。

使用 NAT 网关

Outposts 上本地集群的私有子网必须具有关联的路由表，其中包含通向 NAT 网关（在 Outposts 的父可用区内的公有子网中）的路由。该公有子网必须具有一条通向互联网网关的路由。NAT 网关可以实现出站互联网访问，并防止从互联网到 Outpost 上实例进行未经请求的入站连接。

使用接口 VPC 端点

如果 Outposts 上本地集群的私有子网没有出站互联网连接，或者如果您想在 VPC 内保持所有流量的私密性，则在创建集群之前，您必须在区域性子网中创建以下接口 VPC 端点和网关端点。

终端节点	端点类型
com.amazonaws. *region-code* .ssm	接口
com.amazonaws. *region-code* .ssmmessages	接口
com.amazonaws. *region-code* .ec2messages	接口
com.amazonaws. *region-code* .ec2	接口
com.amazonaws. *region-code* .secretsmanager	接口
com.amazonaws. *region-code* .logs	接口
com.amazonaws. *region-code* .sts	接口
com.amazonaws. *region-code* .ecr.api	接口
com.amazonaws. *region-code* .ecr.dkr	接口
com.amazonaws. *region-code* .s3	Gateway

端点必须满足以下要求：

- 在位于 Outposts 父可用区的私有子网中创建
- 启用私有 DNS 名称
- 具有附加的安全组，该组允许来自私有 outpost 子网的 CIDR 范围的入站 HTTPS 流量。

创建端点会产生费用。有关更多信息，请参阅 Amazon PrivateLink 定价。如果容器组（pod）需要访问其他 Amazon 服务，您需要创建额外的端点。有关端点的完整列表，请参阅与 Amazon PrivateLink 集成的 Amazon 服务。

创建 VPC

您可以使用以下 Amazon CloudFormation 模板之一，创建满足上述要求的 VPC：

- 模板 1 – 此模板会创建一个 VPC，该 VPC 具有一个位于 Outpost 的私有子网，以及一个位于 Amazon 区域的公有子网。该私有子网具有通过位于 Amazon 区域中公有子网中的 NAT 网关通向互联网的路由。此模板可用于在具有传出互联网访问权限的子网中创建本地集群。
- 模板 2 – 此模板会创建一个 VPC，该 VPC 具有一个位于 Outpost 的私有子网，以及在没有传入或传出互联网访问权限的子网（也称为私有子网）中创建本地集群所需的最小 VPC 端点集。

在 Amazon Outposts 上准备本地 Amazon EKS 集群以防断网

如果本地网络断开与 Amazon 云的连接，您可以继续在 Outpost 上使用本地 Amazon EKS 集群。本主题介绍如何针对网络断开连接的情况准备本地集群以及相关注意事项。

- 本地集群可在临时、计划外网络断开连接期间实现稳定性和持续操作。AmazonOutposts 仍是一个完全连接的产品，充当您的数据中心内 Amazon 云的扩展。当 Outpost 与 Amazon 云之间的网络断开连接，我们建议您尝试恢复连接。如需查看相关说明，请参阅 Amazon Outposts 用户指南中的 Amazon Outposts 机架网络故障排除核对清单。有关如何对本地集群的问题进行故障排除的更多信息，请参阅 the section called “集群问题排查”。
- Outpost 将发出一个 `ConnectedStatus` 指标，您可用于监控 Outpost 的连接状态。有关更多信息，请参阅 Amazon Outposts 用户指南中的 Outposts 指标。
- 本地集群使用 IAM 作为默认身份验证机制，该身份验证机制将 Amazon Identity and Access Management 身份验证器用于 Kubernetes。IAM 在网络连接断开期间不可用。因此，本地集群支持使用 `x.509` 证书的替代身份验证机制，您可以使用该机制在网络连接断开期间连接到集群。有关如何获取 `x.509` 证书并将其用于您的集群的信息，请参阅 the section called “在网络断开连接期间对您的本地集群进行身份验证”。
- 如果您在网络连接断开期间无法访问 Route 53，请考虑在您的本地环境中使用本地 DNS 服务器。Kubernetes 控制面板实例使用静态 IP 地址。您可以使用端点主机名和 IP 地址配置用于连接到您的集群的主机，作为使用本地 DNS 服务器的替代方法。有关更多信息，请参阅 Amazon Outposts 用户指南中的 DNS。
- 如果您预计在网络连接断开期间应用程序流量会增加，则可以在连接到云时在您的集群中预调配备用计算容量。Amazon EC2 实例包含在 Amazon Outposts 的价格中。因此，运行备用实例不会影响您的 Amazon 用量成本。
- 在网络连接断开期间，要启用工作负载的创建、更新和扩缩操作，必须可以通过本地网络访问您的应用程序的容器映像，并且您的集群必须具有足够的容量。本地集群不会为您托管容器注册表。如果容器组（pod）之前已在这些节点上运行过，则容器映像将缓存在这些节点上。如果您通常从云中的 Amazon ECR 拉取应用程序的容器映像，建议运行本地缓存或注册表。如果您需要在网络连接断开期间创建、更新和扩展工作负载资源的操作，那么本地缓存或注册表会很有帮助。

- 本地集群使用 Amazon EBS 作为持久卷的默认存储类，并使用 Amazon EBS CSI 驱动程序来管理 Amazon EBS 持久卷的生命周期。网络连接断开期间，由 Amazon EBS 备份的容器组（pod）将无法创建、更新或扩展。这是因为这些操作需要调用云端的 Amazon EBS API。如果您在本地集群上部署有状态工作负载，并且需要在网络断开连接期间执行创建、更新或扩缩操作，请考虑使用替代存储机制。
- 如果 Amazon Outposts 无法访问相关 Amazon 区域内 API（例如 Amazon EBS 或 Amazon S3 的 API），则无法创建或删除 Amazon EBS 快照。
- 将 ALB（Ingress）与 Amazon Certificate Manager（ACM）集成时，证书会被推送并存储在 Amazon Outposts ALB 计算实例的内存中。如果与 Amazon 区域断开连接，当前的 TLS 终止将继续运行。在这种情况下，更改操作将失败（例如新的传入定义、新的基于 ACM 的证书 API 操作、ALB 计算扩展或证书轮换）。有关更多信息，请参阅 Amazon Certificate Manager 用户指南中的排查托管证书续订的问题。
- 在网络断开连接期间，Amazon EKS 控制面板日志将本地缓存在 Kubernetes 控制面板实例上。重新连接后，会将这些日志发送到父级 Amazon 区域中的 CloudWatch 日志。您可以使用 Prometheus、Grafana 或 Amazon EKS 合作伙伴解决方案，利用 Kubernetes API 服务器的指标端点或将 Kubernetes 用于日志，在本地监控集群。
- 如果您将 Outpost 上的 Amazon 负载均衡器控制器用于应用程序流量，则在网络连接断开期间，前面带有 Amazon 负载均衡器控制器的现有容器组（pod）将继续接收流量。在网络断开连接期间创建的新容器组（pod）不会接收流量，直到 Outpost 重新连接到 Amazon 云为止。考虑在连接到 Amazon 云时为您的应用程序设置副本计数，以满足您在网络断开连接期间的扩缩需求。
- 适用于 Kubernetes 的 Amazon VPC CNI 插件默认设置为辅助 IP 模式。它配置为 `WARM_ENI_TARGET=1`，这将允许插件保持可用 IP 地址的“完全弹性网络接口”。考虑在断开连接状态下根据您的扩缩需求更改 `WARM_ENI_TARGET`、`WARM_IP_TARGET` 和 `MINIMUM_IP_TARGET` 值。有关更多信息，请参阅 GitHub 上插件的自述文件。有关每种实例类型支持的容器组（pod）最大数量列表，请参阅 GitHub 上的 eni-max-pods.txt 文件。

在网络断开连接期间对您的本地集群进行身份验证

网络断开连接期间，Amazon Identity and Access Management（IAM）不可用。在断开连接时，您将无法使用 IAM 凭证对您的本地集群进行身份验证。但是，在断开连接时，您可以使用 `x509` 证书通过本地网络连接到集群。您需要下载并存储客户端 `X509` 证书，以便在断开连接期间使用。在本主题中，您将学习在集群处于断开连接状态时，如何创建和使用证书对其进行身份验证。

1. 创建证书签名请求。

   a. 生成证书签名请求。

```
openssl req -new -newkey rsa:4096 -nodes -days 365 \
    -keyout admin.key -out admin.csr -subj "/CN=admin"
```

b. 在 Kubernetes 中创建证书签名请求。

```
BASE64_CSR=$(cat admin.csr | base64 -w 0)
cat << EOF > admin-csr.yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: admin-csr
spec:
  signerName: kubernetes.io/kube-apiserver-client
  request: ${BASE64_CSR}
  usages:
  - client auth
EOF
```

2. 使用 kubectl 创建证书签名请求。

```
kubectl create -f admin-csr.yaml
```

3. 检查证书签名请求的状态。

```
kubectl get csr admin-csr
```

示例输出如下。

```
NAME       AGE   REQUESTOR                      CONDITION
admin-csr  11m   kubernetes-admin               Pending
```

Kubernetes 已创建证书签名请求。

4. 批准证书签名请求。

```
kubectl certificate approve admin-csr
```

5. 重新检查证书签名请求状态是否为已批准。

```
kubectl get csr admin-csr
```

示例输出如下。

```
NAME       AGE   REQUESTOR                     CONDITION
admin-csr  11m   kubernetes-admin              Approved
```

6. 检索并验证证书。

   a. 检索证书。

   ```
   kubectl get csr admin-csr -o jsonpath='{.status.certificate}' | base64 --decode >
    admin.crt
   ```

   b. 验证证书。

   ```
   cat admin.crt
   ```

7. 为 admin 用户创建集群角色绑定。

```
kubectl create clusterrolebinding admin --clusterrole=cluster-admin \
    --user=admin --group=system:masters
```

8. 为断开连接状态生成用户范围的 kubeconfig。

   您可以使用已下载的 admin 证书生成 kubeconfig 文件。替换以下命令中的 *my-cluster* 和 *apiserver-endpoint*。

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.certificateAuthority" \
    --output text | base64 --decode > ca.crt
```

```
kubectl config --kubeconfig admin.kubeconfig set-cluster my-cluster \
    --certificate-authority=ca.crt --server apiserver-endpoint --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-credentials admin \
    --client-certificate=admin.crt --client-key=admin.key --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-context admin@my-cluster \
    --cluster my-cluster --user admin
```

```
kubectl config --kubeconfig admin.kubeconfig use-context admin@my-cluster
```

9. 查看您的 kubeconfig 文件。

```
kubectl get nodes --kubeconfig admin.kubeconfig
```

10如果您的 Outpost 上已有投入生产中的服务，请跳过此步骤。如果 Amazon EKS 是在您的 Outpost 上运行的唯一服务，并且 Outpost 当前未投入生产，则您可以模拟网络断开连接。在使用本地集群投入生产之前，请模拟断开连接，以确保在集群处于断开连接状态时可以访问该集群。

a. 在将您的 Outpost 连接到 Amazon 区域的联网设备上应用防火墙规则。这将断开 Outpost 的服务链接。您无法创建任何新实例。当前正在运行的实例将断开与 Amazon 区域和互联网的连接。

b. 您可以使用 x509 证书在断开连接时测试与您的本地集群的连接。确保将 kubeconfig 更改为您在上一步创建的 admin.kubeconfig。将 *my-cluster* 替换为您的本地集群的名称。

```
kubectl config use-context admin@my-cluster --kubeconfig admin.kubeconfig
```

如果您在本地集群处于断开连接状态时发现其存在任何问题，则我们建议开具支持票证。

根据容量因素为 Amazon Outposts 上的 Amazon EKS 集群选择实例类型和置放群组

本主题提供以下指南：为 Outpost 上的本地 Amazon EKS 集群选择 Kubernetes 控制面板实例类型以及（可选）使用置放群组满足高可用性要求。

在 Outpost 上选择要用于本地集群的 Kubernetes 控制面板的实例类型（如 m5、c5 或 r5）前，请确认可用于 Outpost 配置的实例类型。确定可用实例类型后，请根据您的工作负载所需的节点数量选择实例大小（如 large、xlarge 或 2xlarge）。下表提供了有关选择实例大小的建议。

> **Note**
>
> 必须在 Outpost 上插入实例大小。请确保本地集群生命周期内，有足够的容量可供该大小的三个实例在 Outpost 上运行。有关可用 Amazon EC2 实例类型的列表，请参阅 Amazon Outposts 机架功能中的“计算和存储”章节。

节点数量	Kubernetes 控制面板实例实例大小
1–20	large
21–100	xlarge
101–250	2xlarge
251–500	4xlarge

Kubernetes 控制面板的存储需要每个本地集群拥有 246 GB 的 Amazon EBS 存储才能满足 etcd 所需的 IOPS。创建本地集群后，系统将为您自动预置 Amazon EBS 卷。

控制面板置放

如果不指定具有 OutpostConfig.ControlPlanePlacement.GroupName 属性的置放群组，则在您的 Outpost 上可用的基础容量下，为 Kubernetes 控制面板预置的 Amazon EC2 实例不会接受任何特定的硬件置放强制执行。

您可以使用置放群组满足 Outpost 上的本地 Amazon EKS 集群的高可用性要求。通过在集群创建期间指定置放群组，您可以影响 Kubernetes 控制面板实例的置放。这些实例分布在独立的基础硬件（机架或主机）上，从而最大限度地减少相关实例对硬件故障事件的影响。

您可以配置的分布类型取决于部署中的 Outpost 机架数量。

- 在单个逻辑 Outpost 中使用一个或两个物理机架的部署：您必须至少具有三台主机，这些主机通过为 Kubernetes 控制面板实例选择的实例类型进行配置。使用主机级分布的分布置放群组可确保所有 Kubernetes 控制面板实例在 Outpost 部署中可用的基础机架内的不同主机上运行。
- 在单个逻辑 Outpost 中使用三个或更多物理机架的部署：您必须至少具有三台主机，这些主机通过为 Kubernetes 控制面板实例选择的实例类型进行配置。使用机架级分布的分布置放群组可确保所有 Kubernetes 控制面板实例在 Outpost 部署中的不同机架上运行。或者，您可以使用上一个选项中描述的主机级分布置放群组。

您负责创建所需的置放群组。您在调用 CreateCluster API 时指定置放群组。有关置放群组及如何创建置放群组的更多信息，请参阅《Amazon EC2 用户指南》中的置放群组。

- 指定置放群组后，Outpost 上必须有可用的插槽容量才能成功创建本地 Amazon EKS 集群。容量因您使用的主机或机架分布类型而异。如果没有足够的容量，则集群仍然处于 Creating 状态。您可

以检查 DescribeCluster API 响应运行状况字段上的 `Insufficient Capacity Error`。您必须释放容量才能使创建过程继续进行。

- 在 Amazon EKS 本地集群平台和版本更新期间，使用滚动更新策略将您集群中的 Kubernetes 控制面板实例替换为新实例。在此替换过程中，每个控制面板实例都将终止，从而释放其相应插槽。新的更新实例将预置到相应插槽位置。更新的实例可能会放置在已发布的插槽中。如果该插槽被另一个不相关的实例占用，并且没有符合所需分布拓扑要求的剩余容量，则集群将保持 `Updating` 状态。您可以查看 DescribeCluster API 响应运行状况字段上的相应 `Insufficient Capacity Error`。您必须释放容量，以便更新过程继续进行并重新建立之前的高可用性级别。
- 在每个 Amazon 区域中，最多可以为每个账户创建 500 个置放群组。有关更多信息，请参阅《Amazon EC2 用户指南》中的一般规则和限制。

排查 Amazon Outposts 上的本地 Amazon EKS 集群问题

本主题将介绍您在使用本地集群时可能会看到的一些常见错误，以及如何排查这些错误。虽然本地集群与云端 Amazon EKS 集群相似，但在 Amazon EKS 的托管方式上存在一些差异。

> **Important**
>
> 若未得到 Amazon Support 支持部门明确指示，切勿终止在 Outpost 上运行的任何托管 EKS 本地集群 `Kubernetes` 控制面板实例。终止这些实例会给本地集群服务可用性带来风险，包括：如果同时终止多个实例，会丢失本地集群。EKS 本地集群 `Kubernetes` 控制面板实例由 EC2 实例控制台上的 `eks-local:controlplane-name` 标签标识。

API 行为

本地集群是通过 Amazon EKS API 创建的，但以异步方式运行。这意味着对于本地集群，对 Amazon EKS API 的请求会立即返回。但是，这些请求可能会成功，也可能由于输入验证错误而快速失效，或者失败并出现描述性验证错误。此行为类似于 Kubernetes API。

本地集群不会过渡到 `FAILED` 状态。Amazon EKS 会尝试以连续方式协调集群状态与用户请求的所需状态。因此，本地集群可能会长时间保持在 `CREATING` 状态，直到根本问题得到解决。

描述集群运行状况字段

可以使用 describe-cluster Amazon EKS Amazon CLI 命令发现本地集群问题。本地集群问题将由 `describe-cluster` 命令响应的 `cluster.health` 字段显示。此字段中包含的消息包括错误代码、

描述性消息和相关的资源 ID。此信息将仅通过 Amazon EKS API 和 Amazon CLI 提供。在以下示例中，将 *my-cluster* 替换为您的本地集群的名称。

```
aws eks describe-cluster --name my-cluster --query 'cluster.health'
```

示例输出如下。

```
{
    "issues": [
        {
            "code": "ConfigurationConflict",
            "message": "The instance type 'm5.large' is not supported in Outpost 'my-
outpost-arn'.",
            "resourceIds": [
                "my-cluster-arn"
            ]
        }
    ]
}
```

如果问题无法修复，您可能需要删除该本地集群，然后创建一个新的本地集群。例如，尝试使用 Outpost 上不可用的实例类型预调配集群。下表包括常见的运行状况相关错误。

错误情形	代码	消息	ResourceIds
无法找到提供的子网。	ResourceNotFound	The subnet ID *subnet-id* does not exist	所有提供的子网 ID
提供的子网不属于同一 VPC。	ConfigurationConflict	Subnets specified must belong to the same VPC	所有提供的子网 ID
提供的某些子网不属于指定的 Outpost。	ConfigurationConflict	Subnet *subnet-id* expected to be in *outpost-arn*, but is in *other-outpost-arn*	子网 ID 有问题

错误情形	代码	消息	ResourceIds
提供的某些子网不属于任何 Outpost。	`ConfigurationConflict`	`Subnet subnet-id is not part of any Outpost`	子网 ID 有问题
提供的某些子网没有足够的空闲地址来为控制面板实例创建弹性网络接口。	`ResourceLimitExceeded`	`The specified subnet does not have enough free addresses to satisfy the request.`	子网 ID 有问题
Outpost 上不支持指定的控制面板实例类型。	`ConfigurationConflict`	`The instance type type is not supported in Outpost outpost-arn`	集群 ARN
您已终止控制面板 Amazon EC2 实例，或 `run-instance` 已成功但观察到的状态更改为 `Terminated` 。在您的 Outpost 重新连接并且 Amazon EBS 内部错误导致 Amazon EC2 内部工作流失败后，这种情况可能会发生一段时间。	`InternalFailure`	`EC2 instance state "Terminated" is unexpected`	集群 ARN

错误情形	代码	消息	ResourceIds
Outpost 容量不足。在集群创建期间，如果 Outpost 与 Amazon 区域断开连接，也会发生这种情况。	`ResourceLimitExceeded`	`There is not enough capacity on the Outpost to launch or start the instance.`	集群 ARN
您的账户已超出安全组配额。	`ResourceLimitExceeded`	Amazon EC2 API 返回的错误消息	目标 VPC ID
您的账户已超出弹性网络接口配额。	`ResourceLimitExceeded`	Amazon EC2 API 返回的错误消息	目标子网 ID
无法通过 Amazon Systems Manager 访问控制面板实例。有关分辨率的信息，请参阅 the section called “无法通过 Amazon Systems Manager 访问控制面板实例”。	`ClusterUnreachable`	无法通过 SSM 访问 Amazon EKS 控制面板实例。请验证您的 SSM 和网络配置，并参考 Outpost 上的 EKS 故障排除文档。	Amazon EC2 实例 ID
获取托管安全组或弹性网络接口的详细信息时出错。	基于 Amazon EC2 客户端错误代码。	Amazon EC2 API 返回的错误消息	所有托管安全组 ID
授权或撤销安全组传入规则时出错。这既适用于集群安全组，也适用于控制面板安全组。	基于 Amazon EC2 客户端错误代码。	Amazon EC2 API 返回的错误消息	安全组 ID 有问题
删除控制面板实例的弹性网络接口时出错。	基于 Amazon EC2 客户端错误代码。	Amazon EC2 API 返回的错误消息	弹性网络接口 ID 有问题

下表列出了来自 `describe-cluster` 响应的运行状况字段中呈现的其它 Amazon 服务的错误。

Amazon EC2 错误代码	集群运行状况问题代码	说明
AuthFailure	AccessDenied	有多种原因可能导致此错误。最常见的原因是，您从控制面板中意外删除了服务用于缩小服务相关角色策略范围的标签。如果出现这种情况，Amazon EKS 将无法继续管理和监控这些 Amazon 资源。
UnauthorizedOperation	AccessDenied	有多种原因可能导致此错误。最常见的原因是，您从控制面板中意外删除了服务用于缩小服务相关角色策略范围的标签。如果出现这种情况，Amazon EKS 将无法继续管理和监控这些 Amazon 资源。
InvalidSubnetID.NotFound	ResourceNotFound	如果无法找到安全组传入规则的子网 ID，将发生此错误。
InvalidPermission.NotFound	ResourceNotFound	如果安全组传入规则的权限不正确，将发生此错误。
InvalidGroup.NotFound	ResourceNotFound	如果无法找到安全组的传入规则组，将发生此错误。
InvalidNetworkInterfaceID.NotFound	ResourceNotFound	如果无法找到安全组传入规则的网络接口 ID，将发生此错误。
InsufficientFreeAddressesInSubnet	ResourceLimitExceeded	如果超出子网资源配额，将发生此错误。
InsufficientCapacityOnOutpost	ResourceLimitExceeded	如果超出 Outpost 容量配额，将发生此错误。

Amazon EC2 错误代码	集群运行状况问题代码	说明
NetworkInterfaceLimitExceeded	ResourceLimitExceeded	如果超出弹性网络接口配额，将发生此错误。
SecurityGroupLimitExceeded	ResourceLimitExceeded	如果超出安全组配额，将发生此错误。
VcpuLimitExceeded	ResourceLimitExceeded	在新账户中创建 Amazon EC2 实例时，将观察到此情况。错误可能类似于以下内容：“You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit http://aws.amazon.com/contact-us/ec2-request to request an adjustment to this limit."
InvalidParameterValue	ConfigurationConflict	如果 Outpost 上不支持指定的实例类型，Amazon EC2 将返回此错误代码。
所有其他故障	InternalFailure	无

无法创建或修改集群

本地集群所需的权限和策略与云端托管的 Amazon EKS 集群需要的权限和策略不同。当集群创建失败并显示 InvalidPermissions 错误时，请仔细检查您使用的集群角色是否附加了 AmazonEKSLocalOutpostClusterPolicy 托管策略。所有其他 API 调用都需要与云中的 Amazon EKS 集群相同的权限集。

集群卡在 **CREATING** 状态

创建本地集群所需的时间取决于多个因素。这些因素包括您的网络配置、Outpost 配置和集群配置。一般情况下，可在 15–20 分钟内创建本地集群并更改为 ACTIVE 状态。如果本地集群仍处于 CREATING 状态，您可以在 cluster.health 输出字段中调用 describe-cluster 以了解关于原因的信息。

最常见的问题如下：

- 您的集群无法从 Systems Manager 所在的 Amazon 区域连接到控制面板实例。您可以通过从区域内的堡垒主机调用 aws ssm start-session --target *instance-id* 来对此进行验证。如果该命令不起作用，请检查 Systems Manager 是否在控制面板实例上运行。或者采用另一种解决方法，即删除集群，然后重新创建集群。
- 由于 EBS 卷的 KMS 密钥权限问题，无法创建控制面板实例。若使用用户管理型 KMS 密钥加密了 EBS 卷，如果密钥不可访问，控制面板实例会终止。如果实例终止，要么切换到 Amazon 托管式 KMS 密钥，要么确保用户管理型密钥策略会向集群角色授予必要权限。
- Systems Manager 控制面板实例可能无法访问互联网。检查您在创建集群时提供的子网是否具有 NAT 网关和带有互联网网关的 VPC。使用 VPC Reachability Analyzer 验证控制面板实例是否可以访问互联网网关。有关更多信息，请参阅 VPC Reachability Analyzer 入门。
- 您提供的角色 ARN 缺少策略。检查是否已从角色中移除了 Amazon 托管策略：AmazonEKSLocalOutpostClusterPolicy。如果 Amazon CloudFormation 堆栈配置错误，也可能会出现这种情况。
- 提供的所有子网都必须与同一 Outpost 关联，并且能够相互访问。如果在创建集群期间指定了多个子网，Amazon EKS 会尝试将控制面板实例分布到多个子网中。
- 在弹性网络接口上应用 Amazon EKS 托管安全组。但是，NACL 防火墙规则等其他配置元素也可能会与弹性网络接口的规则冲突。

VPC 和子网 DNS 配置配置错误或缺失

查阅为 Amazon Outposts 上的 Amazon EKS 集群创建 VPC 和子网

集群卡在 **UPDATING** 状态

Amazon EKS 会自动将所有现有本地集群更新到其对应的 Kubernetes 次要版本的最新平台版本。有关平台版本的更多信息，请参阅the section called “EKS 平台版本”。

在平台版本自动推出期间，集群状态更改为 UPDATING。更新过程包括将所有 Kubernetes 控制面板实例，替换成包含为相应 Kubernetes 次要版本发布的最新安全补丁和错误修复的新实例。本地集群平

台更新过程通常会在 30 分钟内完成，集群将变回 ACTIVE 状态。如果本地集群在较长时间内仍处于 UPDATING 状态，您可以调用 describe-cluster，在 cluster.health 输出字段中了解关于原因的信息。

为保持本地集群可用性和防止服务中断，Amazon EKS 确保至少三分之二的 Kubernetes 控制面板实例是能够正常运行的集群节点。如果本地集群停滞在 UPDATING 状态，通常是因为存在某些基础设施或配置问题，无法在过程继续进行时保证两个实例的最低可用性。因此，为了避免本地集群服务中断，更新过程会停止进行。

必须对卡在 UPDATING 状态的本地集群进行问题排查，从根本原因上解决问题，这样更新过程才能顺利完成，让本地集群恢复到 ACTIVE 状态，实现 3 个 Kubernetes 控制面板实例都可用的高可用性。

若未得到 Amazon Support 支持部门明确指示，切勿终止在 Outpost 上的任何托管 EKS 本地集群 Kubernetes 实例。这对于卡在 UPDATING 状态的本地集群尤其重要，因为另一个控制面板节点很有可能无法完全正常运行，而终止错误的实例可能会导致服务中断，造成本地集群数据丢失的风险。

最常见的问题如下：

- 自首次创建本地集群以来，由于网络配置发生了变化，一个或多个控制面板实例无法连接到 System Manager。您可以通过从区域内的堡垒主机调用 `aws ssm start-session --target instance-id` 来对此进行验证。如果该命令不起作用，请检查 Systems Manager 是否在控制面板实例上运行。
- 由于 EBS 卷的 KMS 密钥权限问题，无法创建新的控制面板实例。若使用用户管理型 KMS 密钥加密了 EBS 卷，如果密钥不可访问，控制面板实例会终止。如果实例终止，要么切换到 Amazon 托管式 KMS 密钥，要么确保用户管理型密钥策略会向集群角色授予必要权限。
- Systems Manager 控制面板实例可能已失去互联网访问权限。检查您在创建集群时提供的子网是否具有 NAT 网关和带有互联网网关的 VPC。使用 VPC Reachability Analyzer 验证控制面板实例是否可以访问互联网网关。有关更多信息，请参阅 VPC Reachability Analyzer 入门。如果私有网络没有出站互联网连接，则确保所有必要的 VPC 端点和网关端点仍存在于集群的区域子网中（请参阅the section called “子网对 Amazon 服务的访问权限”）。
- 您提供的角色 ARN 缺少策略。检查是否从角色中移除了 Amazon 托管式策略：AmazonEKSLocalOutpostClusterPolicy。
- 其中一个新的 Kubernetes 控制面板实例可能遇到了意料之外的引导失败问题。如果遇到这种特殊情况，请向 Amazon Support Center 提交工单，获取有关问题排查和日志收集的进一步指导。

无法将节点加入集群

- AMI 问题：

  - 当前使用的 AMI 不兼容。仅支持 Amazon EKS 优化版 Amazon Linux 2 AMI（`amazon-linux-2`、`amazon-linux-2-gpu`、`amazon-linux-2-arm64`）。若尝试将 AL2023 节点加入部署于 Amazon Outposts 上的 EKS 本地集群，这些节点将无法成功加入集群。有关更多信息，请参阅在 Amazon Outposts 上创建 Amazon Linux 节点。
  - 如果您使用 Amazon CloudFormation 模板创建您的节点，请确保它没有使用不受支持的 AMI。
- 缺失 Amazon IAM 身份验证器 `ConfigMap` – 如果缺失，则必须进行创建。有关更多信息，请参阅 the section called “将 `aws-authConfigMap` 应用到集群”。
- 使用了错误的安全组 – 请确保将 `eks-cluster-sg-`*`cluster-name`*`-`*`uniqueid`* 用于 Worker 节点的安全组。将通过 Amazon CloudFormation 更改所选的安全组，以便在每次使用堆栈时允许新的安全组。
- 遵循意外的私有链接 VPC 步骤 - 传递错误的 CA 数据 (`--b64-cluster-ca`) 或 API 端点 (`--apiserver-endpoint`)。

收集日志

如果 Outpost 与其关联的 Amazon 区域断开连接，Kubernetes 集群可能会继续正常运行。但是，如果集群无法正常运行，请执行在 Amazon Outposts 上准备本地 Amazon EKS 集群以防断网中的问题排查步骤。如果遇到任何问题，请与 Amazon Support 联系。AmazonSupport 会指导您如何下载和运行日志收集工具。这样一来，您就可以从 Kubernetes 集群控制面板实例中收集日志，并将其发送到 Amazon Support 支持部门，以便进一步调查。

无法通过 Amazon Systems Manager 访问控制面板实例

当无法通过 Amazon Systems Manager 访问 Amazon EKS 控制面板实例时，Amazon EKS 将为您的集群显示以下错误。

```
Amazon EKS control plane instances are not reachable through SSM. Please verify your
 SSM and network configuration, and reference the EKS on Outposts troubleshooting
 documentation.
```

要解决此问题，请确保您的 VPC 和子网满足为 Amazon Outposts 上的 Amazon EKS 集群创建 VPC 和子网中的要求，并且您已完成《Amazon Systems Manager 用户指南》内设置 Session Manager 中的步骤。

在 Amazon Outpost 上创建 Amazon Linux 节点

Important

部署于 Outposts 上的 Amazon EKS 本地集群，仅支持通过以下 Amazon EKS 优化版 Amazon Linux 2023 AMI 创建的节点：

- 标准 Amazon Linux 2023 (`amazon-linux-2023/x86_64/standard`)
- 加速型 Nvidia Amazon Linux 2023 (`amazon-linux-2023/x86_64/nvidia`)
- 加速型 Neuron Amazon Linux 2023 (`amazon-linux-2023/x86_64/neuron`)

Amazon 将于 2025 年 11 月 26 日终止对 EKS AL2 优化版和 AL2 加速型 AMI 的支持。尽管您可以在终止支持（EOS）日期（2025 年 11 月 26 日）之后继续使用 EKS AL2 AMI，但在该日期之后，EKS 将不再为 AL2 AMI 发布任何新的 Kubernetes 版本或更新，包括次要版本、补丁和错误修复。有关 AL2 弃用的更多信息，请参阅此处。

本主题介绍如何在向您的 Amazon EKS 集群的 Outpost 上启动 Amazon Linux 节点的自动扩缩组。集群可以位于 Amazon 云上，也可以位于 Outpost 上。

- 一个现有的 Outpost。有关更多信息，请参阅什么是 Amazon Outposts。
- 现有 Amazon EKS 集群。要在 Amazon 云上部署集群，请参阅the section called “创建集群”。要在 Outpost 上部署集群，请参阅 the section called “运行本地集群”。
- 假设您正在 Amazon 云上的集群中创建节点，并在已启用 Amazon Outposts、Amazon Wavelength 或 Amazon Local Zones 的 Amazon 区域中拥有子网。接下来，当您创建集群时，不得传入这些子网。如果您要在 Outpost 上的集群中创建节点，则在创建集群时必须已传入 Outpost 子网。
- （建议用于 Amazon 云上的集群）适用于 Kubernetes 附加组件 的 Amazon VPC CNI 插件已配置了自己的 IAM 角色，并附加了必要的 IAM 策略。有关更多信息，请参阅 the section called “配置 IRSA”。本地集群不支持服务账户的 IAM 角色。

您可以使用 `eksctl` 或 Amazon Web Services 管理控制台（带有 Amazon CloudFormation 模板）创建自行管理的 Amazon Linux 节点组。您还可以使用 Terraform。

您可以使用本页中描述的下列工具为本地集群创建自主管理型节点组：

- the section called “`eksctl`”

- the section called “Amazon Web Services 管理控制台”

> **Important**
>
> - 自主管理型节点组包括您的账户中的 Amazon EC2 实例。当您或 Amazon EKS 代表您更新控制面板版本时，这些实例不会自动升级。对于自行管理的节点组，控制台中没有任何其需要更新的标示。您可以在集群的 Overview（概览）选项卡上的 Nodes（节点）列表中选择节点查看其上安装的 `kubelet` 版本，确定哪些节点需要更新。您必须手动更新节点。有关更多信息，请参阅 the section called “更新方法”。
> - kubelet 在自主管理型节点上所用证书的有效期为一年。默认不启用证书轮换（请参阅：https://kubernetes.io/docs/reference/config-api/kubelet-config.v1beta1/#kubelet-config-k8s-io-v1beta1-KubeletConfiguration），这表示：如果自主管理型节点运行时间超过一年，则无法再向 Kubernetes API 进行身份验证。
> - 建议客户最好定期更新自己的自主管理型节点组，以便接收来自最新 Amazon EKS 优化版 AMI 的 CVE 和安全补丁。更新自主管理型节点组中使用的 AMI 还会触发节点的重新创建，确保不会因为 kubelet 证书过期而出现问题。
> - 您也可以在创建自主管理型节点组时启用客户端证书轮换（请参阅：https://kubernetes.io/docs/tasks/tls/certificate-rotation/），确保在当前证书即将到期时续签 kubelet 证书。

`eksctl`

使用 **`eksctl`** 启动自主管理型 Linux 节点

1. 在您的设备或 Amazon CloudShell 上安装 `0.215.0` 版或更高版本的 `eksctl` 命令行工具。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的 Installation。
2. 如果您的集群位于 Amazon 云上，并且 AmazonEKS_CNI_Policy 托管 IAM 策略附加到您的 Amazon EKS 节点 IAM 角色，我们建议将其分配给您与 Kubernetes `aws-node` 服务账户关联的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。如果您的集群位于 Outpost 上，则必须将该策略附加到您的节点角色。
3. 以下命令在现有集群中创建节点组。必须已经使用 `eksctl` 创建了集群。将 *`al-nodes`* 替换为您的节点组名称。节点组名称的长度不能超过 63 个字符。它必须以字母或数字开头，但也可以包括其余字符的连字符和下划线。将 *`my-cluster`* 替换为您的集群的名称。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。如果您的集群存在于

Outpost 上，则将 *id* 替换为 Outpost 子网的 ID。如果您的集群存在于 Amazon 云上，则将 *id* 替换为您在创建集群时未指定的子网的 ID。将其余的 example values 替换为您自己的值。预设情况下，将使用与控制层面相同的 Kubernetes 版本创建节点。

将 *instance-type* 替换为您的 Outpost 上可用的实例类型。

将 *my-key* 替换为您的 Amazon EC2 密钥对或公有密钥的名称。此密钥用于在节点启动后通过 SSH 进入节点。如果还没有 Amazon EC2 密钥对，可以在 Amazon Web Services 管理控制台 中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。

使用以下命令创建您的节点组。

```
eksctl create nodegroup --cluster my-cluster --name al-nodes --node-type instance-
type \
    --nodes 3 --nodes-min 1 --nodes-max 4 --managed=false \
    --node-volume-type gp2 --subnet-ids subnet-id \
    --node-ami-family AmazonLinux2023
```

如果您的集群部署在 Amazon 云上：

- 您部署的节点组可以将 IPv4 地址分配给来自不同 CIDR 块的容器组（pod），而非实例的容器组。有关更多信息，请参阅 the section called “自定义联网”。
- 您部署的节点组不需要出站互联网访问权限。有关更多信息，请参阅 the section called “私有集群”。

有关所有可用选项和默认设置的完整列表，请参阅 `eksctl` 文档中的 Amazon Outposts 支持。

- 如果节点无法加入集群，请参阅排查 Amazon EKS 集群和节点问题中的the section called “节点未能加入集群”和排查 Amazon Outposts 上的本地 Amazon EKS 集群问题中的the section called “无法将节点加入集群”。
- 示例输出如下。创建节点时会输出几行。输出的最后几行类似于以下示例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

4. （可选）部署示例应用程序以测试集群和 Linux 节点。

Amazon Web Services 管理控制台

第 1 步：使用 Amazon Web Services 管理控制台 启动自主管理型 Linux 节点

1. 下载最新版本的 Amazon CloudFormation 模板。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2025-11-24/
amazon-eks-outpost-nodegroup.yaml
```

2. 打开 Amazon CloudFormation 控制台。
3. 选择 Create stack（创建堆栈），然后选择 With new resources (standard)（使用新资源（标准））。
4. 对于 Specify template（指定模板），选择 Upload a template file（上传模板文件），然后选择Choose file（选择文件）。选择您在上一步中下载的 `amazon-eks-nodegroup.yaml` 文件，然后选择 Next（下一步）。
5. 在 Specify stack details（指定堆栈详细信息）页面上，相应填写以下参数，然后选择 Next（下一步）：
   - 堆栈名称：为 Amazon CloudFormation 堆栈选择堆栈名称。例如，您可以将其命名为 *al-nodes*。名称只能包含字母数字字符（区分大小写）和连字符。该名称必须以字母数字字符开头，且不得超过 100 个字符。对于您在其中创建集群的 Amazon 区域和 Amazon 账户，该名称必须在其内具有唯一性。
   - ApiServerEndpoint：请输入 Kubernetes API Server 端点，该信息可在 EKS 控制台查看，也可通过 DescribeCluster API 获取。
   - ClusterName：输入您的集群的名称。如果此名称与您的集群名称不匹配，则您的节点将无法加入该集群。
   - ClusterId：请输入由 EKS 服务分配给集群的 ID。该信息可通过 DescribeCluster API 获取。若该 ID 与实际集群 ID 不匹配，节点将无法加入集群。
   - CertificateAuthority：请输入 Kubernetes 证书颁发机构的 Base64 编码字符串。该信息可在 EKS 控制台查看，也可通过 DescribeCluster API 获取。
   - ServiceCidr：请输入 Kubernetes 服务 CIDR。该信息可在 EKS 控制台查看，也可通过 DescribeCluster API 获取。
   - ClusterControlPlaneSecurityGroup：从您在创建 VPC 时生成的 Amazon CloudFormation 输出中，选择 SecurityGroups 值。

     以下步骤显示了检索适用组的一种操作。

     i. 打开 Amazon EKS 控制台。

     ii. 选择集群的名称。

     iii. 选择 Networking（联网）选项卡。

iv. 从 ClusterControlPlaneSecurityGroup 下拉列表中选择时使用 Additional security groups（其他安全组）值作为参考。

- NodeGroupName：输入节点组的名称。稍后您可以使用此名称来标识为您的节点创建的自动扩缩节点组。
- NodeAutoScalingGroupMinSize：输入您的节点自动扩缩组可横向缩减到的最小节点数。
- NodeAutoScalingGroupDesiredCapacity：输入创建堆栈时要扩展到的所需节点数目。
- NodeAutoScalingGroupMaxSize：输入您的节点自动扩缩组可横向扩展到的最大节点数。
- NodeInstanceType：选择节点的实例类型。如果您的集群在 Amazon 云上运行，则有关更多信息，请参阅the section called “Amazon EC2 实例类型”。如果您的集群在 Outpost 上运行，则您只能选择您的 Outpost 上可用的实例类型。
- NodeImageIdSSMParam：使用用于某个变量 Kubernetes 版本最近的 Amazon EKS 优化版 AMI 的 Amazon EC2 Systems Manager 参数进行预填充。要使用 Amazon EKS 支持的其他 Kubernetes 次要版本，可以将 *1.XX* 替换为不同的支持版本。我们建议您指定与您的集群相同的 Kubernetes 版本。

  若要使用 Amazon EKS 优化版加速型 AMI，请将 *NodeImageIdSSMParam* 参数值更新为目标 SSM 参数。有关如何从 SSM 中获取 EKS AMI ID 的方法，请参阅此处。

  > **Note**
  >
  > Amazon EKS 节点 AMI 基于 Amazon Linux。您可以在 Amazon Linux 安全中心通过选择所需版本的选项卡来跟踪 Amazon Linux 的安全和隐私事件。您也可以订阅相应的 RSS 源。安全和隐私事件包括问题的概述、受影响的程序包以及如何更新实例以解决问题。

- NodeImageId：（可选）如果使用自定义 AMI（而不是 Amazon EKS 优化型 AMI），则输入您 Amazon 区域的节点 AMI ID。如果您在此处指定值，它会覆盖 NodeImageIdSSMParam 字段中的任意值。
- NodeVolumeSize：指定您的节点的根卷大小（以 GiB 为单位）。
- NodeVolumeType：指定您的节点的根卷类型。
- KeyName：输入 Amazon EC2 SSH 密钥对的名称，您可使用该密钥对来在节点启动后使用 SSH 连接到这些节点。如果还没有 Amazon EC2 密钥对，可以在 Amazon Web Services 管理控制台中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。

> **Note**
>
> 如果此处不提供密钥对，Amazon CloudFormation 堆栈创建将失败。

- DisableIMDSv1：预设情况下，每个节点支持实例元数据服务版本 1 (IMDSv1) 和 IMDSv2。您可以禁用 IMDSv1。要防止节点组中的未来节点和容器组（pod）使用 IMDSv1，请将 DisableIMDSv1 设置为 true。有关 IMDS 的更多信息，请参阅配置实例元数据服务。有关限制在节点上访问的更多信息，请参阅限制对分配给 Worker 节点的实例配置文件的访问。
- VpcId：输入您创建的 VPC 的 ID。选择 VPC 之前，请查看 VPC 要求和注意事项。
- 子网：如果您的集群位于 Outpost 上，则请在您的 VPC 中选择至少一个私有子网。选择子网之前，请查看子网要求和注意事项。您可以通过从集群的 Networking（联网）选项卡中打开每个子网链接来查看哪些子网是私有的。

6. 在 Configure stack options（配置堆栈选项）页面上选择所需选项，然后选择 Next（下一步）。
7. 选择我了解 Amazon CloudFormation 可能会创建 IAM 资源。左侧的复选框，然后选择创建堆栈。
8. 完成创建堆栈后，在控制台中选中它，然后选择 Outputs（输出）。
9. 记录已创建的节点组的 NodeInstanceRole。您在配置 Amazon EKS 节点时需要此值。

步骤 2：使节点能够加入集群

1. 检查您是否已经应用拥有 `aws-auth` `ConfigMap`。

```
kubectl describe configmap -n kube-system aws-auth
```

2. 如果您看到的是 `aws-auth` `ConfigMap`，则请根据需要对其进行更新。

   a. 打开 `ConfigMap` 文件进行编辑。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 根据需要添加新的 `mapRoles` 条目。将 `rolearn` 值设置为您在上一个步骤中记录的 NodeInstanceRole 值。

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
```

```
      groups:
        - system:bootstrappers
        - system:nodes
[...]
```

c. 保存文件并退出文本编辑器。

3. 如果您收到错误提示 "Error from server (NotFound): configmaps "aws-auth" not found，则请使用库存 ConfigMap。

a. 下载配置映射。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/aws-auth-cm.yaml
```

b. 在 aws-auth-cm.yaml 文件中，将 rolearn 设置为您在上一个步骤中记录的 NodeInstanceRole 值。您可以使用文本编辑器或者通过替换 *my-node-instance-role* 和运行以下命令来执行此操作：

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-role|' aws-auth-cm.yaml
```

c. 应用配置。此命令可能需要几分钟才能完成。

```
kubectl apply -f aws-auth-cm.yaml
```

4. 查看节点的状态并等待它们达到 Ready 状态。

```
kubectl get nodes --watch
```

输入 Ctrl+C 以返回到 Shell 提示符。

> **Note**
>
> 如果您收到任何授权或资源类型错误，请参阅故障排除主题中的 the section called “未经授权或访问被拒绝 (kubectl)”。

如果节点无法加入集群，请参阅排查 Amazon EKS 集群和节点问题中的the section called “节点未能加入集群”和排查 Amazon Outposts 上的本地 Amazon EKS 集群问题中的the section called “无法将节点加入集群”。

5. 安装 Amazon EBS CSI 驱动程序。有关更多信息，请参阅 GitHub 上的安装。在设置驱动程序权限部分中，确保按照 Using IAM instance profile（使用 IAM 实例配置文件）选项的说明进行操作。您必须使用 gp2 存储类。不支持 gp3 存储类。

   要在您的集群上创建 gp2 存储类，请完成以下步骤。

   a. 运行以下命令以创建 gp2-storage-class.yaml 文件。

```
cat >gp2-storage-class.yaml <<EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp2
  encrypted: "true"
allowVolumeExpansion: true
EOF
```

   b. 将清单应用于集群。

```
kubectl apply -f gp2-storage-class.yaml
```

6. （仅限 GPU 节点）如果选择 GPU 实例类型和 Amazon EKS 优化版加速型 AMI，则必须在集群上将适用于 Kubernetes 的 NVIDIA 设备插件应用为 DaemonSet。将 *vX.X.X* 替换为您需要的 NVIDIA/k8s-device-plugin 版本，然后运行以下命令。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

第 3 步：其它操作

1. （可选）部署示例应用程序以测试集群和 Linux 节点。
2. 如果您的集群部署在 Outpost 上，则请跳过此步骤。如果您的集群部署在 Amazon 云上，以下信息是可选的。如果 AmazonEKS_CNI_Policy 托管 IAM 策略附加到您的 Amazon EKS 节点 IAM 角色，

我们建议将其分配给您与 Kubernetes aws-node 服务账户关联的 IAM 角色。有关更多信息，请参阅 the section called “配置 IRSA”。

Amazon EKS 上的人工智能（AI）和机器学习（ML）概述

Amazon Elastic Kubernetes Service（EKS）是托管型 Kubernetes 平台，能够让组织以无与伦比的灵活性与控制力部署、管理和扩展人工智能和机器学习（ML）工作负载。EKS 在开源 Kubernetes 生态系统上构建，不仅便于您利用现有的 Kubernetes 专业知识，还可帮助您与开源工具和 Amazon 服务无缝集成。

无论您是想训练大型模型、运行实时在线推理，还是部署生成式人工智能应用程序，EKS 都能提供人工智能/机器学习项目所需的优异性能、可扩展性以及成本效益。

为何选择 EKS 部署人工智能/机器学习？

EKS 是一个托管型 Kubernetes 平台，可帮助您部署和管理复杂的人工智能/机器学习工作负载。它在开源 Kubernetes 生态系统上构建，可与 Amazon 服务集成，提供高级项目所需的控制力和可扩展性。对于刚接触人工智能/机器学习部署的团队，可以直接转移现有的 Kubernetes 技能，实现多个工作负载的高效编排。

EKS 支持从操作系统自定义到计算扩缩的所有内容，其开源基础提高了技术灵活性，为未来的基础设施决策保留了选择余地。该平台提供人工智能/机器学习工作负载所需的性能和调整选项，支持以下功能：

- 集群完全控制，无需隐藏抽象即可微调成本和配置
- 生产环境中实时推理工作负载的亚秒级延迟保障
- 高级自定义设置，例如多实例 GPU、多云策略和操作系统级调整
- 能够使用 EKS 作为跨人工智能/机器学习管道的统一编排工具，集中管理工作负载

关键使用案例

Amazon EKS 提供了一个适用于人工智能/机器学习工作负载的强大平台，支持各种技术和部署模式：

- 实时（在线）推理：EKS 借助 Amazon EC2 Inf1 和 Inf2 实例上的 TorchServe、Triton Inference Server 和 KServe 等工具，为传入数据（例如欺诈检测）提供亚秒级延迟的即时预测能力。这些工作负载不仅受益于 Karpenter 和 KEDA 的动态扩缩，还能利用 Amazon EFS 在容器组（pod）之间进行模型分片。Amazon ECR Pull Through Cache（PTC） 可加快模型更新速度，而具有 Amazon EBS 优化型卷的 Bottlerocket 数据卷可确保快速访问数据。

- 常规模型训练：组织可利用 EKS，基于 Amazon EC2 P4d 和 Amazon EC 2 Trn1 实例，使用 Kubeflow Training Operator、Ray Serve 和 Torch Distributed Elastic，在大型数据集上长时间训练复杂模型。这些工作负载由 Volcano、Yunikorn 和 Kueue 等工具提供批量计划支持。Amazon EFS 支持共享模型检查点，并且 Amazon S3 使用生命周期策略处理模型的导入/导出，以进行版本管理。
- 检索增强生成（RAG）管道：EKS 通过集成检索和生成流程来管理客户支持聊天机器人和类似的应用程序。这些工作负载通常使用 Argo Workflows 和 Kubeflow 等工具进行编排，使用 Pinecone、Weaviate 或 Amazon OpenSearch 等向量数据库，并通过应用程序负载均衡器控制器（LBC）向用户公开应用程序。NVIDIA NIM 可优化 GPU 利用率，而 Prometheus 和 Grafana 则监控资源使用情况。
- 生成式人工智能模型部署：各公司使用 Amazon EC2 G5 和 Inferentia 加速器上的 Ray Serve、vLLM、和 Triton Inference Server，在 EKS 上部署实时内容创建服务，例如文本或图像生成。这些部署优化了大型模型的性能和内存利用率。JupyterHub 支持迭代开发，Gradio 提供简单的 Web 界面，并且 S3 Mountpoint CSI 驱动程序允许将 S3 存储桶安装为文件系统，以访问大型模型文件。
- 批量（离线）推理：组织通过使用 Amazon Batch 或 Volcano 的计划作业高效处理大型数据集。这些工作负载通常将 Inf1 和 Inf2 EC2 实例用于 Amazon Inferentia 芯片，将 Amazon EC2 G4dn 实例用于 NVIDIA T4 GPU，或者将 c5 和 c6i CPU 实例用于最大限度地提高非高峰时段分析任务的资源利用率。Amazon Neuron SDK 和 NVIDIA GPU 驱动程序优化了性能，MIG/TS 则实现了 GPU 共享。存储解决方案包括 Amazon S3、Amazon EFS 和 FSx for Lustre，以及适用于各种存储类的 CSI 驱动程序。模型管理利用 Kubeflow Pipelines、Argo Workflows 和 Ray Cluster 等工具，监控管理则利用 Prometheus、Grafana 以及自定义模型监控工具。

案例研究

客户选择 Amazon EKS 的原因多种多样，例如优化 GPU 使用率或以亚秒级延迟运行实时推理工作负载，如以下案例研究所示。有关 Amazon EKS 的所有案例研究列表，请参阅 Amazon 客户成功案例。

- Unitary 将人工智能内容审核范围扩大到每天 2600 万个视频，不仅实现了高吞吐量、低延迟的推理，还将容器启动时间缩短了 80%，确保在流量波动时快速响应扩缩事件。
- Miro，一个为全球 7000 万用户提供服务的可视化协作平台，相较于之前的自主管理型 Kubernetes 集群，计算成本降低了 80%。
- Synthesia 提供生成式人工智能视频创建服务，供客户根据文本提示创建逼真的视频，其机器学习模型训练吞吐量提升了 30 倍。
- Harri 为酒店业提供 HR 技术解决方案，迁移至 Amazon Graviton 处理器后，横向缩减速度提高了 90%，可应对激增的需求，同时计算成本降低了 30%。

- Ada Support，一家人工智能驱动的客户服务自动化公司，其计算成本降低了 15%，并且计算效率提升了 30%。
- Snorkel AI，旨在帮助企业构建和调整基础模型和大型语言模型，通过为其 GPU 资源实施智能扩缩机制，节省了 40% 以上的成本。

开始在 EKS 上部署机器学习

要开始在 Amazon 云端的 EKS 上规划和使用机器学习平台和工作负载，请继续阅读 the section called “资源” 部分。

在 Amazon EKS 上运行实时在线推理工作负载

本节旨在帮助您在 Amazon Elastic Kubernetes Service（EKS）上部署和运行实时在线推理工作负载。您将找到有关构建优化的集群（含有 GPU 加速节点）、集成 Amazon 服务（用于存储和自动扩展）、部署示例模型进行验证，以及关键架构注意事项（例如解耦 CPU 和 GPU 任务、选择适当的 AMI 和实例类型，以及确保推理端点的低延迟暴露）的指导。

主题

- Amazon EKS 上实时推理的最佳实践集群设置指南
- 快速入门：在 Amazon EKS 上使用 vLLM 进行高吞吐量 LLM 推理

Amazon EKS 上实时推理的最佳实践集群设置指南

简介

本指南提供一个实作演练，用于设置针对实时在线推理工作负载进行优化的 Amazon Elastic Kubernetes Service（EKS）集群，其中包含了由 Amazon 专家整理的最佳实践。它使用带有预设偏好的 EKS 快速启动架构：一组精心策划的驱动程序、实例类型和配置，与模型、加速器和扩展的 Amazon 最佳实践保持一致。这种方法可以帮助您绕过选择集群设置的任务，让您能够快速启动并运行功能齐全的预配置集群。在此过程中，我们将部署示例工作负载来验证您的设置，解释关键架构概念（例如将 CPU 密集型任务与 GPU 密集型计算分离），解决常见问题（例如，为什么选择 Bottlerocket AMI 而不是 AL2023？），并概述扩展集群功能的后续步骤。

本指南专为不熟悉 Amazon 和 EKS 生态系统的机器学习（ML）和人工智能（AI）工程师、平台管理员、操作员以及数据/人工智能专家而设计，其假设您熟悉 Kubernetes，但之前没有 EKS 使用经验。本指南旨在帮助您了解启动和运行实时在线推理工作负载所需的步骤和流程。本指南向您展示创建单节

点推理集群的基本要素，包括预置 GPU 资源、为模型构件集成存储、启用安全 Amazon 服务访问以及公开推理端点。本指南自始至终强调面向用户的应用程序（例如欺诈检测、实时聊天机器人和客户反馈系统中的情感分析）的低延迟、弹性设计。

在本指南中，我们专门介绍如何使用 G5 EC2 实例设置基础的规范性起点。如果您正在寻找 Amazon Inferentia 特定的集群配置或端到端工作流程，请参阅the section called “使用 Inferentia 设置推理集群”或the section called “资源”中我们的讲习会。

开始前的准备工作

开始之前，请确保您已执行以下任务：

- 为 Amazon EKS 设置您的环境
- 安装 eksctl 的最新版本
- 安装 Helm
- （可选）安装 Docker
- （可选）安装 NGC CLI

架构

实时在线推理是指使用经过训练的机器学习模型，以最小的延迟对传入的数据流生成预测或输出的过程。例如，它支持实时欺诈检测、图像分类或根据用户输入生成知识图谱。实时在线推理系统的架构通过将 CPU 密集型 Web 流量处理与 GPU 密集型人工智能计算分离，在面向用户的应用程序中提供低延迟的机器学习预测。此过程通常存在于更大的应用程序生态系统中，通常包括后端、前端、向量和模型服务，侧重于专门的组件，以实现独立扩展、并行开发和故障恢复能力。在专用 GPU 硬件上隔离推理任务并利用 API 和 WebSocket 等接口可确保高并发性、快速处理转换器等模型以及通过前端的用户交互。请注意，尽管向量数据库和检索增强生成（RAG）管道通常在实时推理系统中发挥重要作用，但本指南中并未涵盖这些组件。典型的架构通常至少包括以下部分：

- 前端服务：作为面向用户的界面，处理客户端逻辑，呈现动态内容并促进实时交互。该服务与后端服务通信以启动推理请求并显示结果，通常向使用 WebSocket 进行流式更新或使用 API 进行结构化数据交换的后端服务发起请求。该服务通常不需要专用的负载均衡器，因为它可以托管在 Amazon CloudFront 等内容分发网络（CDN）上（用于静态资产），也可以直接从 Web 服务器提供服务，如果需要动态内容，则可通过自动扩缩组进行扩展。
- 后端服务：充当应用程序的编排工具，管理业务逻辑，如用户身份验证、数据验证和服务协调（例如，通过 RESTful 端点的 API 或用于持久性连接的 WebSocket）。该服务与推理服务通信，在多核 CPU 和 RAM 上独立扩展，无需依赖 GPU 即可处理高 Web 流量，并且通常需要负载均衡器（例如

Amazon 应用程序负载均衡器或网络负载均衡器）将传入的请求分发到多个实例，尤其是在高并发场景中。入口控制器可以进一步管理外部访问和路由规则，以增强安全性和流量整形。

- 推理服务：作为人工智能计算的核心，在具有充足 VRAM（例如需要 8-12 GB 的 DistilBERT 等模型）的 GPU 上运行，以使用自定义或开源模型执行向量嵌入、知识提取和模型推理（例如，通过 API 发布批处理请求或通过 WebSocket 发布实时流）。这种隔离可以防止依赖项冲突，允许在不停机的情况下更新模型，并支持水平扩缩，实现多个并发请求的负载均衡。为了有效地公开模型服务，该服务通常位于负载均衡器之后，以在复制的实例之间分配 GPU 密集型工作负载，而入口资源或控制器（例如 Amazon 中的 ALB Ingress Controller）处理外部路由、SSL 终止和基于路径的转发，以确保安全高效的访问，不会让单个 GPU 不堪重负。

解决方案概述

实时在线推理系统需要高性能、弹性的架构，能够在处理不可预测的大流量突发时提供超低延迟。本解决方案概述说明以下 Amazon 组件如何在我们将创建的 Amazon EKS 集群中协同工作，以确保我们的集群能够托管和管理机器学习模型，以最小的延迟为最终用户提供对实时数据的即时预测。

- Amazon G5 EC2 实例：对于 GPU 密集型推理任务，我们使用 g5.xlarge 和 g5.2xlarge G5 EC2 实例类型，它们具有一个 (1) 个 NVIDIA A10G GPU，内存为 24 GB（例如 FP16 上有 80 亿个参数）。这些 GPU 基于 NVIDIA Ampere 架构，由 NVIDIA A10G Tensor Core GPU 和第二代 AMD EPYC 处理器提供支持，支持 4-8 个 vCPU、高达 10 Gbps 的网络带宽和 250-450 GB 的本地 NVMe SSD 存储，可确保复杂模型的快速数据移动和计算能力，非常适合低延迟、高吞吐量的推理任务。选择 EC2 实例类型因应用程序而异，具体取决于您的模型（例如图像、视频、文本模型）以及您的延迟和吞吐量要求。例如，如果使用图像和/或视频模型，则可能需要使用 P5 EC2 实例才能获得最佳实时延迟。我们建议从 G5 EC2 实例开始（因为它为快速启动和运行提供了一个很好的起点），然后通过性能基准测试来评估其是否适合您的工作负载。对于更高级的案例，可以考虑 G6 EC2 实例。
- Amazon EC2 M7g 实例：对于数据预处理、API 请求处理、托管 Karpenter 控制器、附加组件和其他系统组件等 CPU 密集型任务，我们使用 m5.xlarge M7g EC2 实例类型。M7g 实例是基于 ARM 的实例，具有 4 个 vCPU、16 GB 的内存、高达 12.5 Gbps 的网络带宽，由 Amazon Graviton3 处理器提供支持。选择 EC2 实例类型因应用程序而异，具体取决于工作负载的计算、内存和可扩展性要求。对于计算优化型工作负载，您可以考虑 C7g EC2 实例，其同样使用 Graviton3 处理器，但在某些使用案例中经过优化，计算性能高于 M7g 实例。或者，较新的 C8g EC2 实例（如果可用）可提供最多比 C7g 实例高 30% 的计算性能。我们建议从 M7g EC2 实例开始，因为其具有成本效益以及与各种工作负载（例如应用程序服务器、微服务、游戏服务器、中型数据存储）的兼容性，然后通过性能基准测试来评估其是否适合您的工作负载。

- Amazon S3 Mountpoint CSI 驱动程序：对于多个容器组（pod）共享一个 GPU 的单 GPU 实例上的工作负载（例如，在同一个节点上安排多个容器组（pod）以利用其 GPU 资源），我们使用 Mountpoint S3 CSI 驱动程序来优化内存使用情况，这对于成本敏感、低复杂度设置中的大型模型推断等任务至关重要。它将 Amazon S3 存储桶作为类似 POSIX 的文件系统向 Kubernetes 集群公开，允许推理容器组（pod）无需先下载即可将模型构件（例如模型权重）直接读入内存，并使用标准文件操作输入数据集。此外，S3 具有几乎无限的存储容量，可加速数据密集型推理工作负载。选择存储 CSI 驱动程序因应用程序而异，具体取决于工作负载的吞吐量和延迟要求。尽管 FSx for OpenZFS CSI 启动程序可为随机 I/O 或完全符合 POSIX 标准的跨节点共享持久性卷提供亚毫秒级延迟，但我们建议从 Mountpoint S3 CSI 驱动程序开始，因为它具有可扩展性、对于大型数据集较低的成本以及与 S3 托管对象存储的内置集成，可实现读取密集型推理模式（例如流式模型输入），然后通过性能基准测试来评估其是否适合您的工作负载。
- EKS 容器组身份代理：为了允许访问 Amazon 服务，我们使用了 EKS 容器组身份代理，后者使用单一服务主体，促进了 Amazon EKS 集群内容器组（pod）级的 IAM 角色关联。EKS 容器组身份为传统的服务账户 IAM 角色（IRSA）方法提供了一种简化的替代方案，其使用单一服务主体（pods.eks.amazonaws.com），而不是依赖每个集群单独的 OIDC 提供商，这样可以更轻松地分配权限。此外，它还能够跨多个集群重复使用角色，并支持 IAM 角色会话标签和目标 IAM 角色等高级功能。
- EKS 节点监控代理：为了确保推理服务的持续可用性和可靠性，我们使用带自动修复功能的 EKS 节点监控代理，其可以自动检测和替换运行状况不佳的节点，从而最大限度地减少停机时间。该代理使用增强的运行状况检查（例如 KernelReady、NetworkingReady）持续监控节点的硬件、内核、联网和存储问题。对于 GPU 节点，它会检测特定于加速器的故障，然后通过以下方式启动优雅补救：封锁运行状况不佳的节点、等待 10 分钟以解决临时 GPU 问题，以及在 30 分钟后更换节点来应对持续故障。
- Bottlerocket AMI：为向 EKS 集群提供加强安全的基础，我们使用了 Bottlerocket AMI，它仅包含运行容器所需的基本组件，并为快速扩展提供最短的启动时间。选择节点 AMI 因应用程序而异，具体取决于工作负载的自定义、安全性和可扩展性要求。尽管 AL2023 AMI 为主机级安装和自定义提供了更大的灵活性（例如，在 PV/PVC 中指定专用缓存目录而无需任何其他节点配置），但我们建议从 Bottlerocket AMI 开始，因为其占用空间更小，并对容器化工作负载（例如微服务、推理服务器、可扩展 API）进行了内置优化，然后通过性能基准测试来评估其是否适合您的工作负载。
- Amazon 负载均衡器控制器（LBC）：为公开实时推理端点，我们使用了 Amazon 负载均衡器控制器，它会根据 Kubernetes Ingress 和服务资源自动预置和管理针对 HTTP/HTTPS 流量的应用程序负载均衡器（ALB）和针对 TCP/UDP 流量的网络负载均衡器（NLB），从而实现推理模型与外部客户端的集成。此外，它还支持基于路径的路由等功能，可在多个容器组（pod）或节点之间分发推理请求，确保流量高峰期间的可扩展性，并通过连接多路复用和运行状况检查等 Amazon 原生优化来最大限度地减少延迟。

1. 创建您的 EKS 集群

在此步骤中，我们将使用由 Amazon CloudFormation 提供支持的 eksctl ClusterConfig 模板创建一个包含 CPU 节点和托管节点组的集群。仅使用 CPU 节点初始化集群，可以让我们专门使用 Karpenter 来管理 CPU 密集型和 GPU 节点，从而使用我们在后面的步骤中创建的 Karpenter 节点池来优化资源分配。为了支持我们的实时推理工作负载，我们在集群中预置了 EKS Bottlerocket AMI、EKS 节点监控代理、EKS 容器组身份代理、Mountpoint S3 CSI 驱动程序、Amazon 负载均衡器控制器（LBC）以及 kube-proxy、vpc-cni 和 coredns 驱动程序。m7g.xlarge 实例将用于 CPU 系统任务，包括托管 Karpenter 控制器、附加组件和其他系统组件。

默认情况下，`eksctl` 将为集群创建一个专用 VPC，其 CIDR 块为 `192.168.0.0/16`。VPC 包括三个公有子网和三个私有子网，各自分布在三个不同的可用区（或在 `us-east-1` 区域中为两个可用区），这是部署 Kubernetes 工作负载的理想方法。该模板还部署了一个互联网网关，通过其路由表中的默认路由提供对公有子网的互联网访问，并在其中一个公有子网中部署单个 NAT 网关，私有子网路由表中的默认路由使出站流量通过该 NAT 网关进行互联网访问。要了解有关此设置的更多信息，请参阅将节点部署到私有子网。

检查您的凭证

检查您的 Amazon CLI 凭证是否有效以及是否能够使用 Amazon 服务进行身份验证：

```
aws sts get-caller-identity
```

如果成功，该 CLI 将返回有关您的 Amazon 身份的详细信息（用户 ID、账户和 Arn）。

检查实例可用性

G5 实例类型并非在所有区域中都可用。请检查离您最近的区域。例如：

```
aws ec2 describe-instance-types --instance-types g5.xlarge g5.2xlarge --region us-
east-1
```

如果成功，将在您指定的区域中提供 G5 实例类型。

Bottlerocket AMI 并未在所有区域提供。通过检索离您最近的区域的 Bottlerocket AMI ID 进行检查。例如：

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-1.33/arm64/latest/
image_id \
```

```
    --region us-east-1 --query "Parameter.Value" --output text
```

如果成功，将在您指定的区域提供 Bottlerocket AMI。

准备环境

首先，在新的终端窗口中设置以下环境变量。注意：请务必将示例占位符替换为您的唯一值，包括集群名称、所需区域、Karpenter 发布版本和 Kubernetes 版本。

> **Tip**
>
> 某些变量（例如 ${AWS_REGION} 和 ${K8S_VERSION}）会在代码块的前面定义，然后在后续命令中引用，以保持一致性并避免重复。确保按顺序运行这些命令，以便正确导出这些值并可用于后续定义。

```
export TEMPOUT="$(mktemp)"
export K8S_VERSION=1.33
export KARPENTER_VERSION="1.5.0"
export AWS_REGION="us-east-1"
export EKS_CLUSTER_NAME="eks-rt-inference-${AWS_REGION}"
export S3_BUCKET_NAME="eks-rt-inference-models-${AWS_REGION}-$(date +%s)"
export NVIDIA_BOTTLEROCKET_AMI="$(aws ssm get-parameter --name /aws/service/
bottlerocket/aws-k8s-${K8S_VERSION}-nvidia/x86_64/latest/image_id --query
 Parameter.Value --output text)"
export STANDARD_BOTTLEROCKET_AMI="$(aws ssm get-parameter --name /aws/service/
bottlerocket/aws-k8s-${K8S_VERSION}/arm64/latest/image_id --query Parameter.Value --
output text)"
export AWS_ACCOUNT_ID="$(aws sts get-caller-identity --query Account --output text)"
export ALIAS_VERSION="$(aws ssm get-parameter --name "/aws/service/eks/optimized-
ami/${K8S_VERSION}/amazon-linux-2023/x86_64/standard/recommended/image_id" --query
 Parameter.Value | xargs aws ec2 describe-images --query 'Images[0].Name' --image-ids |
 sed -r 's/^.*(v[[:digit:]]+).*$/\1/')"
```

创建所需的角色和策略

Karpenter 需要特定的 IAM 角色和策略（例如 Karpenter 控制器 IAM 角色、实例配置文件和策略）来管理作为 Kubernetes Worker 节点的 EC2 实例。它使用这些角色来执行如启动和终止 EC2 实例、标记资源以及与其他 Amazon 服务交互之类的操作。使用 Karpenter 的 cloudformation.yaml 创建 Karpenter 角色和策略：

```
curl -fsSL https://raw.githubusercontent.com/aws/karpenter-provider-aws/v
${KARPENTER_VERSION}/website/content/en/preview/getting-started/getting-started-with-
karpenter/cloudformation.yaml > "${TEMPOUT}" \
&& aws cloudformation deploy \
  --stack-name "Karpenter-${EKS_CLUSTER_NAME}" \
  --template-file "${TEMPOUT}" \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides "ClusterName=${EKS_CLUSTER_NAME}"
```

Amazon LBC 需要权限才能预置和管理 Amazon 负载均衡器，例如为 Ingress 资源创建 ALB 或为类型为 `LoadBalancer` 的服务创建 NLB。我们将在创建集群的过程中指定此权限策略。在创建集群的过程中，我们将在 ClusterConfig 中使用 eksctl 创建服务账户。创建 LBC IAM 策略：

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy \
  --policy-document "$(curl -fsSL https://raw.githubusercontent.com/kubernetes-sigs/
aws-load-balancer-controller/v2.14.1/docs/install/iam_policy.json)"
```

安装 Mountpoint S3 CSI 驱动程序后，其 DaemonSet 容器组（pod）将配置为使用服务账户执行。Mountpoint S3 CSI 驱动程序的挂载点需要权限才能与您在本指南后面部分创建的 Amazon S3 存储桶进行交互。我们将在创建集群的过程中指定此权限策略。在创建集群的过程中，我们将在 ClusterConfig 中使用 eksctl 创建服务账户。创建 S3 IAM 策略：

```
aws iam create-policy \
    --policy-name S3CSIDriverPolicy \
    --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Effect\":
 \"Allow\", \"Action\": [\"s3:GetObject\", \"s3:PutObject\", \"s3:AbortMultipartUpload
\", \"s3:DeleteObject\", \"s3:ListBucket\"], \"Resource\": [\"arn:aws:s3:::
${S3_BUCKET_NAME}\", \"arn:aws:s3:::${S3_BUCKET_NAME}/*\"]}]}"
```

注意：如果已存在使用此名称的角色，请为该角色指定不同的名称。我们在此步骤中创建的角色专用于您的集群和 S3 存储桶。

创建集群

在此模板中，eksctl 会自动为 EKS 容器组身份、节点监控代理、CoreDNS、Kubeproxy、VPC CNI 插件创建 Kubernetes 服务账户。到目前为止，Mountpoint S3 CSI 驱动程序尚未支持 EKS 容器组身份，因此我们创建了一个服务账户 IAM 角色（IRSA）和一个 OIDC 端点。此外，我们还为 Amazon 负载均衡器控制器（LBC）创建了一个服务账户。要访问 Bottlerocket 节点，eksctl 会为 Bottlerocket 自动附加 AmazonSSMManagedInstanceCore，以允许通过 SSM 进行安全的 Shell 会话。

在设置环境变量的同一终端，运行以下命令块创建集群：

```
eksctl create cluster -f - <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: ${EKS_CLUSTER_NAME}
  region: ${AWS_REGION}
  version: "${K8S_VERSION}"
  tags:
    karpenter.sh/discovery: ${EKS_CLUSTER_NAME}
    # Add more tags if needed for billing
iam:
  # Creates an OIDC endpoint and IRSA service account for the Mountpoint S3 CSI Driver
  # Uses the S3 CSI Driver policy for permissions
  withOIDC: true
  podIdentityAssociations:
  # Creates the pod identity association and service account
  # Uses the Karpenter controller IAM policy for permissions
  - namespace: "kube-system"
    serviceAccountName: karpenter
    roleName: ${EKS_CLUSTER_NAME}-karpenter
    permissionPolicyARNs:
    - arn:aws:iam::${AWS_ACCOUNT_ID}:policy/KarpenterControllerPolicy-
${EKS_CLUSTER_NAME}
  # Creates the pod identity association and service account
  # Uses the {aws} LBC policy for permissions
  - namespace: kube-system
    serviceAccountName: aws-load-balancer-controller
    createServiceAccount: true
    roleName: AmazonEKSLoadBalancerControllerRole
    permissionPolicyARNs:
    - arn:aws:iam::${AWS_ACCOUNT_ID}:policy/AWSLoadBalancerControllerIAMPolicy
iamIdentityMappings:
- arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/KarpenterNodeRole-${EKS_CLUSTER_NAME}"
  username: system:node:{{EC2PrivateDNSName}}
  groups:
  - system:bootstrappers
  - system:nodes
managedNodeGroups:
  # Creates 2 CPU nodes for lightweight system tasks
  - name: ${EKS_CLUSTER_NAME}-m7-cpu
    instanceType: m7g.xlarge
```

```
    amiFamily: Bottlerocket
    desiredCapacity: 2
    minSize: 1
    maxSize: 10
    labels:
      role: cpu-worker
# Enable automatic Pod Identity associations for VPC CNI Driver, coreDNS, kube-proxy
addonsConfig:
  autoApplyPodIdentityAssociations: true
addons:
  # Installs the S3 CSI Driver addon and creates IAM role
  # Uses the S3 CSI Driver policy for IRSA permissions
  - name: aws-mountpoint-s3-csi-driver
    attachPolicyARNs:
      - "arn:aws:iam::${AWS_ACCOUNT_ID}:policy/S3CSIDriverPolicy"
  - name: eks-pod-identity-agent
  - name: eks-node-monitoring-agent
  - name: coredns
  - name: kube-proxy
  - name: vpc-cni
EOF
```

此过程需要几分钟时间才能完成。如果要监控状态，请访问 Amazon CloudFormation 控制台。

2. 验证集群节点和容器组（pod）运行状况

让我们执行一些运行状况检查来确保集群已准备就绪。当上一个命令完成时，使用以下命令查看实例类型并验证您的 CPU 系统节点是否已达到 Ready 状态：

```
kubectl get nodes -L node.kubernetes.io/instance-type
```

预期输出应如下所示：

```
NAME                              STATUS   ROLES    AGE     VERSION
 INSTANCE-TYPE
ip-192-168-35-103.ec2.internal   Ready    <none>   12m     v1.33.0-eks-802817d
 m7g.xlarge
ip-192-168-7-15.ec2.internal     Ready    <none>   12m     v1.33.0-eks-802817d
 m7g.xlarge
```

使用以下命令验证所有容器组身份关联及其如何将角色映射到集群命名空间中的服务账户：

```
eksctl get podidentityassociation --cluster ${EKS_CLUSTER_NAME} --region ${AWS_REGION}
```

输出应显示 Karpenter 的 IAM 角色（“karpenter”）和 Amazon LBC（“aws-load-balancer-controller”）。

验证 DaemonSet 是否可用：

```
kubectl get daemonsets -n kube-system
```

预期输出应如下所示：

```
NAME                           DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR
          AGE
aws-node                       3       3       3     3          3         <none>
          12m
dcgm-server                    0       0       0     0          0
 kubernetes.io/os=linux 12m
eks-node-monitoring-agent      3       3       3     3          3
 kubernetes.io/os=linux 12m
eks-pod-identity-agent         3       3       3     3          3         <none>
          12m
kube-proxy                     3       3       3     3          3         <none>
          12m
s3-csi-node                    2       2       2     2          2
 kubernetes.io/os=linux 12m
```

验证集群上是否已安装所有附加组件：

```
eksctl get addons --cluster ${EKS_CLUSTER_NAME} --region ${AWS_REGION}
```

预期输出应如下所示：

```
NAME                           VERSION              STATUS    ISSUES    IAMROLE
                                      UPDATE AVAILABLE    CONFIGURATION VALUES    POD
 IDENTITY ASSOCIATION ROLES
aws-mountpoint-s3-csi-driver   v1.15.0-eksbuild.1   ACTIVE    0
 arn:aws:iam::143095308808:role/eksctl-eks-rt-inference-us-east-1-addon-aws-m-Role1-
RAUjk4sJnc0L
coredns                        v1.12.1-eksbuild.2   ACTIVE    0
eks-node-monitoring-agent      v1.3.0-eksbuild.2    ACTIVE    0
```

```
eks-pod-identity-agent         v1.3.7-eksbuild.2    ACTIVE    0
kube-proxy                     v1.33.0-eksbuild.2   ACTIVE    0
metrics-server                 v0.7.2-eksbuild.3    ACTIVE    0
vpc-cni                        v1.19.5-eksbuild.1   ACTIVE    0
```

3. 安装 Karpenter

在 CPU Worker 节点（`cpu-worker`）上安装 Karpenter 控制器，以优化成本并节省 GPU 资源。我们将其安装在“kube-system”命名空间中，并指定在创建集群过程中定义的“karpenter”服务账户。此外，此命令还会配置集群名称和 CPU 节点的竞价型实例中断队列。Karpenter 将使用 IRSA 来代入此 IAM 角色。

```
# Logout of helm registry before pulling from public ECR
helm registry logout public.ecr.aws

# Install Karpenter
helm upgrade --install karpenter oci://public.ecr.aws/karpenter/karpenter --version
 "${KARPENTER_VERSION}" --namespace "kube-system" --create-namespace  \
  --set "settings.clusterName=${EKS_CLUSTER_NAME}" \
  --set "settings.interruptionQueue=${EKS_CLUSTER_NAME}" \
  --set controller.resources.requests.cpu=1 \
  --set controller.resources.requests.memory=1Gi \
  --set controller.resources.limits.cpu=1 \
  --set controller.resources.limits.memory=1Gi \
  --set serviceAccount.annotations."eks\.amazonaws\.com/role-arn"="arn:aws:iam::
${AWS_ACCOUNT_ID}:role/${EKS_CLUSTER_NAME}-karpenter" \
  --wait
```

预期输出应如下所示：

```
Release "karpenter" does not exist. Installing it now.
Pulled: public.ecr.aws/karpenter/karpenter:1.5.0
Digest: sha256:9a155c7831fbff070669e58500f68d7ccdcf3f7c808dcb4c21d3885aa20c0a1c
NAME: karpenter
LAST DEPLOYED: Thu Jun 19 09:57:06 2025
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

验证 Karpenter 是否正在运行：

```
kubectl get pods -n kube-system -l app.kubernetes.io/name=karpenter
```

预期输出应如下所示：

```
NAME                         READY   STATUS    RESTARTS   AGE
karpenter-555895dc-865bc   1/1     Running   0          5m58s
karpenter-555895dc-j7tk9   1/1     Running   0          5m58s
```

4. 设置 Karpenter 节点池

在此步骤中，我们将配置互斥的 CPU 和 GPU Karpenter 节点池。节点池规格中的 `limits` 字段限制了每个节点池在所有已预置节点上可以消耗的最大总资源（例如 CPU、内存、GPU），从而防止在超过这些限制时额外预置节点。虽然节点池支持广泛的实例类别（例如，c、g），但指定具体的实例类型、容量类型和资源限制可帮助您更轻松地估算按需工作负载的成本。在这些节点池中，我们使用 G5 实例系列中的多种实例类型。这允许 Karpenter 根据容器组（pod）资源请求自动选择最合适的实例类型，从而在遵守节点池总限制的同时优化资源利用率。要了解更多信息，请参阅 Creating NodePools。

设置 GPU 节点池

在此节点池 中，我们将设置资源限制以管理具有 GPU 功能的节点预置。这些限制旨在限制池中所有节点的总资源，总共允许最多 10 个实例。每个实例可以是 g5.xlarge（4 个 vCPU、16 GiB 内存、1 个 GPU），也可以是 g5.2xlarge（8 个 vCPU、32 GiB 内存、1 个 GPU），只要 vCPU 总数不超过 80 个，总内存不超过 320 GiB，GPU 总数不超过 10 个。例如，该池可以预置 10 个 g5.2xlarge 实例（80 个 vCPU、320 GiB、10 个 GPU）、10 个 g5.xlarge 实例（40 个 vCPU、160 GiB、10 个 GPU），或者混合预置 5 个 g5.xlarge 和 5 个 g5.2xlarge（60 个 vCPU、240 GiB、10 个 GPU），确保基于工作负载需求的灵活性，同时遵循资源限制。

此外，我们还指定了 Bottlerocket AMI 的 Nvidia 变体 ID。最后，我们将中断策略设置为在 30 分钟后移除空节点（`consolidateAfter: 30m`），并将节点的最大生命周期设置为 30 天（`expireAfter: 720h`），以优化成本并维护 GPU 密集型任务的节点运行状况。要了解更多信息，请参阅 Disable Karpenter Consolidation for interruption sensitive workloads 和 Use ttlSecondsAfterFinished to Auto Clean-Up Kubernetes Jobs。

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-a10g-inference-g5
spec:
```

```
  template:
    metadata:
      labels:
        role: gpu-worker
        gpu-type: nvidia-a10g
    spec:
      requirements:
        - key: node.kubernetes.io/instance-type
          operator: In
          values: ["g5.xlarge", "g5.2xlarge"]
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
      nodeClassRef:
        name: gpu-a10g-inference-ec2
        group: karpenter.k8s.aws
        kind: EC2NodeClass
      expireAfter: 720h
  limits:
    cpu: "80"
    memory: "320Gi"
    nvidia.com/gpu: "10"
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: 30m
---
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: gpu-a10g-inference-ec2
spec:
  amiFamily: Bottlerocket
  amiSelectorTerms:
    - id: ${NVIDIA_BOTTLEROCKET_AMI}
  role: "KarpenterNodeRole-${EKS_CLUSTER_NAME}"
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${EKS_CLUSTER_NAME}"
  securityGroupSelectorTerms:
    - tags:
```

```
        karpenter.sh/discovery: "${EKS_CLUSTER_NAME}"
  tags:
    nvidia.com/gpu: "true"
EOF
```

预期输出应如下所示：

```
nodepool.karpenter.sh/gpu-a10g-inference-g5 created
ec2nodeclass.karpenter.k8s.aws/gpu-a10g-inference-ec2 created
```

验证节点池是否已创建且正常运行：

```
kubectl get nodepool gpu-a10g-inference-g5 -o yaml
```

查找类似 `ValidationSucceeded: True`、`NodeClassReady: True` 和 `Ready: True` 的 `status.conditions`，以确认节点池正常运行。

设置 CPU 节点池

在此节点池中，我们将限制设置为支持约 50 个实例，这与中等 CPU 工作负载（例如 100-200 个容器组（pod））和典型 Amazon vCPU 配额（例如 128-1152）保持一致。计算限制时假设节点池应扩展到 50 个 m7.xlarge 实例：CPU（每个实例 4 个 vCPU × 50 个实例 = 200 个 vCPU）和内存（每个实例 16 GiB × 50 个实例 = 800 GiB）。这些限制旨在限制池中所有节点的总资源，允许最多 50 个 m7g.xlarge 实例（每个实例有 4 个 vCPU 和 16 GiB 内存），前提是 vCPU 总数不超过 200 个且总内存不超过 800 GiB。

此外，我们还指定了 Bottlerocket AMI 的标准变体 ID。最后，我们将中断策略设置为在 60 分钟后移除空节点（`consolidateAfter: 60m`），并将节点的最大生命周期设置为 30 天（`expireAfter: 720h`），以优化成本并维护 GPU 密集型任务的节点运行状况。要了解更多信息，请参阅 Disable Karpenter Consolidation for interruption sensitive workloads 和 Use ttlSecondsAfterFinished to Auto Clean-Up Kubernetes Jobs。

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: cpu-inference-m7gxlarge
spec:
  template:
    metadata:
      labels:
```

```
        role: cpu-worker
    spec:
      requirements:
        - key: node.kubernetes.io/instance-type
          operator: In
          values: ["m7g.xlarge"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
      taints:
        - key: role
          value: cpu-intensive
          effect: NoSchedule
      nodeClassRef:
        name: cpu-inference-m7gxlarge-ec2
        group: karpenter.k8s.aws
        kind: EC2NodeClass
      expireAfter: 720h
  limits:
    cpu: "200"
    memory: "800Gi"
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: 60m
---
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: cpu-inference-m7gxlarge-ec2
spec:
  amiFamily: Bottlerocket
  amiSelectorTerms:
    - id: ${STANDARD_BOTTLEROCKET_AMI}
  role: "KarpenterNodeRole-${EKS_CLUSTER_NAME}"
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${EKS_CLUSTER_NAME}"
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${EKS_CLUSTER_NAME}"
EOF
```

预期输出应如下所示：

```
nodepool.karpenter.sh/cpu-inference-m7gxlarge created
ec2nodeclass.karpenter.k8s.aws/cpu-inference-m7gxlarge-ec2 created
```

验证节点池是否已创建且正常运行：

```
kubectl get nodepool cpu-inference-m7gxlarge -o yaml
```

查找类似 `ValidationSucceeded: True`、`NodeClassReady: True` 和 `Ready: True` 的 `status.conditions`，以确认节点池正常运行。

5. 部署 GPU 容器组（pod）以公开 GPU

您需要 Nvidia 设备插件才能让 Kubernetes 向 Kubernetes 集群公开 GPU 设备。通常，您需要将插件部署为 DaemonSet；但是，Bottlerocket AMI 会将该插件作为 AMI 的一部分进行预安装。这意味着在使用 Bottlerocket AMI 时，无需部署 Nvidia 设备插件 DaemonSet。要了解更多信息，请参阅 Kubernetes Device Plugin to expose GPUs。

部署示例容器组（pod）

Karpenter 动态运行：当工作负载（容器组（pod））请求 GPU 资源时，它会预置 GPU 节点。要验证容器组（pod）是否能够请求和使用 GPU，请部署一个在其限制内请求 `nvidia.com/gpu` 资源的容器组（pod）（例如 `nvidia.com/gpu: 1`）。要了解有关这些标签的更多信息，请参阅 Schedule workloads with GPU requirements using Well-Known labels。

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: gpu-nvidia-smi
spec:
  restartPolicy: OnFailure
  tolerations:
  - key: "nvidia.com/gpu"
    operator: "Exists"
    effect: "NoSchedule"
  nodeSelector:
    role: gpu-worker  # Matches GPU NodePool's label
  containers:
  - name: cuda-container
    image: nvidia/cuda:12.9.1-base-ubuntu20.04
    command: ["nvidia-smi"]
```

```
    resources:
      limits:
        nvidia.com/gpu: 1
      requests:
        nvidia.com/gpu: 1
EOF
```

预期输出应如下所示：

```
pod/gpu-ndivia-smi created
```

稍等片刻，然后检查容器组（pod）状态是否为“Pending”、“ContainerCreating”、“Running”，然后变为“Completed”：

```
kubectl get pod gpu-nvidia-smi -w
```

验证容器组（pod）的节点是否属于 GPU 节点池：

```
kubectl get node $(kubectl get pod gpu-nvidia-smi -o jsonpath='{.spec.nodeName}') -o
 custom-columns="Name:.metadata.name,Nodepool:.metadata.labels.karpenter\.sh/nodepool"
```

预期输出应如下所示：

```
Name                              Nodepool
ip-192-168-83-245.ec2.internal    gpu-a10g-inference-g5
```

检查容器组（pod）的日志：

```
kubectl logs gpu-nvidia-smi
```

预期输出应如下所示：

```
Thu Jul 17 04:31:33 2025
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 570.148.08                 Driver Version: 570.148.08         CUDA
 Version: 12.9 |
|-----------------------------------------+----------------------
+----------------------+
```

```
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC
 |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M.
 |
|                                         |                      |               MIG M.
 |
|=========================================+======================
+======================|
|   0  NVIDIA A10G                    On  | 00000000:00:1E.0 Off |                    0
 |
|  0%   30C    P8               9W / 300W |      0MiB / 23028MiB |      0%      Default
 |
|                                         |                      |                  N/A
 |
+-----------------------------------------------------------------------------------------
+

+-----------------------------------------------------------------------------------------
+
| Processes:
 |
|  GPU        GI     CI        PID   Type   Process name                 GPU Memory
 |
|                     ID        ID                                        Usage
 |
|
=========================================================================================|
|  No running processes found
 |
+-----------------------------------------------------------------------------------------
+
```

6.（可选）准备并上传模型构件以进行部署

在此步骤中，您将部署用于实时图像分类的模型服务，首先将模型权重上传到 Amazon S3 存储桶。为了进行演示，我们使用 NVIDIA 的 GPUNet 中的开源 GPUNet-0 视觉模型部分，其支持使用 NVIDIA GPU 和 TensorRT 对图像进行低延迟推理。此模型在 ImageNet 上进行了预训练，可以即时分类照片或视频流中的对象，被视为一个小型模型（包含 1190 万个参数）。

设置环境

要下载 GPUNet-0 模型权重：在此步骤中，您需要访问 NVIDIA 的 NGC 目录并在本地计算机上安装 Docker。按照以下步骤操作，设置免费账户并配置 NGC CLI：

- 注册一个免费的 NGC 账户并从 NGC 控制面板生成 API 密钥（用户图标 > 设置 > 生成 API 密钥 > 生成个人密钥 > NGC 目录）。
- 下载并安装 NGC CLI（Linux/macOS/Windows），然后使用以下命令配置 CLI：`ngc config set`。出现提示时输入您的 API 密钥；将 org 设置为 `nvidia`，然后按 Enter 键接受其他默认设置。如果成功，您应该会看到类似以下内容：`Successfully saved NGC configuration to /Users/your-username/.ngc/config`。

验证服务账户权限

开始之前，请检查 Kubernetes 服务账户的权限：

```
kubectl get serviceaccount s3-csi-driver-sa -n kube-system -o yaml
```

在集群创建过程中，我们将 S3CSIDriverPolicy 附加到 IAM 角色并注释了服务账户（“s3-csi-driver-sa”）。Mountpoint S3 CSI 驱动程序容器组（pod）在与 S3 交互时会继承 IAM 角色的权限。预期输出应如下所示：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::143095308808:role/eksctl-eks-rt-inference-
us-east-1-addon-aws-m-Role1-fpXXjRYdKN8r
  creationTimestamp: "2025-07-17T03:55:29Z"
  labels:
    app.kubernetes.io/component: csi-driver
    app.kubernetes.io/instance: aws-mountpoint-s3-csi-driver
    app.kubernetes.io/managed-by: EKS
    app.kubernetes.io/name: aws-mountpoint-s3-csi-driver
  name: s3-csi-driver-sa
  namespace: kube-system
  resourceVersion: "2278"
  uid: 50b36272-6716-4c68-bdc3-c4054df1177c
```

添加容忍

S3 CSI 驱动程序作为 DaemonSet 在所有节点上运行。容器组（pod）使用这些节点上的 CSI 驱动程序来挂载 S3 卷。要允许其在有污点的 GPU 节点上进行调度，请向 DaemonSet 添加容忍：

```
kubectl patch daemonset s3-csi-node -n kube-system --type='json' -p='[{"op": "add",
 "path": "/spec/template/spec/tolerations/-", "value": {"key": "nvidia.com/gpu",
 "operator": "Exists", "effect": "NoSchedule"}}]'
```

预期输出应如下所示：

```
daemonset.apps/s3-csi-node patched
```

将模型权重上传到 S3

在此步骤中，您将创建一个 Amazon S3 存储桶，从 NVIDIA GPU Cloud（NGC）下载 GPUNet-0 模型权重，然后将其上传到存储桶。我们的应用程序将在运行时访问这些权重以进行推理。

创建您的 Amazon S3 存储桶：

```
aws s3 mb s3://${S3_BUCKET_NAME} --region ${AWS_REGION}
```

为存储桶启用 S3 版本控制，以防止意外删除和覆盖导致即使和永久的数据丢失：

```
aws s3api put-bucket-versioning --bucket ${S3_BUCKET_NAME} --versioning-configuration
 Status=Enabled
```

对存储桶应用生命周期规则，以便在覆盖或删除的对象版本变为非最新版本 14 天后将其移除，移除过期的删除标记，并在 7 天后移除未完成的分段上传。要了解更多信息，请参阅 S3 生命周期配置的示例。

```
aws s3api put-bucket-lifecycle-configuration --bucket $S3_BUCKET_NAME --lifecycle-
configuration '{"Rules":[{"ID":"LifecycleRule","Status":"Enabled","Filter":
{},"Expiration":{"ExpiredObjectDeleteMarker":true},"NoncurrentVersionExpiration":
{"NoncurrentDays":14},"AbortIncompleteMultipartUpload":{"DaysAfterInitiation":7}}]}'
```

从 NGC 下载 GPUNet-0 模型权重。例如，在 macOS 上，请执行以下操作：

```
ngc registry model download-version nvidia/dle/gpunet_0_pyt_ckpt:21.12.0_amp --dest ~/
downloads
```

> **Note**
>
> 您可能需要根据操作系统调整此下载命令。要使此命令在 Linux 系统上运行，您可能需要在命令中创建目录（例如 `mkdir ~/downloads`）。

预期输出应如下所示：

```
{
  "download_end": "2025-07-18 08:22:39",
  "download_start": "2025-07-18 08:22:33",
  "download_time": "6s",
  "files_downloaded": 1,
  "local_path": "/Users/your-username/downloads/gpunet_0_pyt_ckpt_v21.12.0_amp",
  "size_downloaded": "181.85 MB",
  "status": "Completed",
  "transfer_id": "gpunet_0_pyt_ckpt[version=21.12.0_amp]"
}
```

在后续步骤中，重命名检查点文件以匹配应用程序代码中预期的命名（无需提取，因为它是包含模型状态字典的标准 PyTorch *.pth.tar 检查点）：

```
mv ~/downloads/gpunet_0_pyt_ckpt_v21.12.0_amp/0.65ms.pth.tar gpunet-0.pth
```

在 Amazon CLI 中启用 Amazon 通用运行时以优化 S3 吞吐量：

```
aws configure set s3.preferred_transfer_client crt
```

将模型权重上传到 S3 存储桶：

```
aws s3 cp gpunet-0.pth s3://${S3_BUCKET_NAME}/gpunet-0.pth
```

预期输出应如下所示：

```
upload: ./gpunet-0.pth to s3://eks-rt-inference-models-us-east-1-1752722786/
gpunet-0.pth
```

创建模型服务

在此步骤中，您将使用 GPUNet-0 视觉模型设置一个 FastAPI Web 应用程序，用于 GPU 加速的图像分类。该应用程序在运行时从 Amazon S3 下载模型权重，从 NVIDIA 的存储库中获取模型架构进行缓存，并通过 HTTP 下载 ImageNet 类标签。该应用程序包括图像预处理转换并公开两个端点：一个用于状态检查的根 GET 端点和一个接受图像 URL 的 POST `/predict` 端点。

我们使用带有 PyTorch 的 FastAPI 来提供模型，在运行时以容器化设置从 Amazon S3 加载权重，以便快速进行原型设计和 Kubernetes 部署。有关其他方法，例如优化的批处理或高吞吐量引擎，请参阅 Serving ML Models。

创建应用程序

为应用程序文件创建一个目录（例如 `model-testing`），然后切换到该目录，并将以下代码添加到名为 `app.py` 的新文件中：

```
import os
import torch
import json
import requests
from fastapi import FastAPI, HTTPException
from PIL import Image
from io import BytesIO, StringIO
import torchvision.transforms as transforms
from torch.nn.functional import softmax
import warnings
from contextlib import redirect_stdout, redirect_stderr
import argparse
import boto3
app = FastAPI()

# Suppress specific warnings from the model code (quantization is optional and unused
 here)
warnings.simplefilter("ignore", UserWarning)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load model code from cache (if present)
# Use backed cache directory
torch.hub.set_dir('/cache/torch/hub')

# Allowlist for secure deserialization (handles potential issues in older checkpoints)
torch.serialization.add_safe_globals([argparse.Namespace])
```

```
# Load the model architecture only on container startup (changed to pretrained=False)
# Precision (FP32 for full accuracy, could be 'fp16' for speed on Ampere+ GPUs)
with redirect_stdout(StringIO()), redirect_stderr(StringIO()):
    gpunet = torch.hub.load('NVIDIA/DeepLearningExamples:torchhub', 'nvidia_gpunet',
 pretrained=False, model_type='GPUNet-0', model_math='fp32')

# Download weights from S3 if not present, then load them
model_path = os.getenv('MODEL_PATH', '/cache/torch/hub/checkpoints/gpunet-0.pth')
os.makedirs(os.path.dirname(model_path), exist_ok=True)  # Ensure checkpoints dir
 exists
if not os.path.exists(model_path):
    s3 = boto3.client('s3')
    s3.download_file(os.getenv('S3_BUCKET_NAME'), 'gpunet-0.pth', model_path)
checkpoint = torch.load(model_path, map_location=device, weights_only=True)
gpunet.load_state_dict(checkpoint['state_dict'])
# Move to GPU/CPU
gpunet.to(device)
gpunet.eval()

# Preprocessing
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Load ImageNet labels
labels_url = "https://s3.amazonaws.com/deep-learning-models/image-models/
imagenet_class_index.json"
response = requests.get(labels_url)
json_data = json.loads(response.text)
labels = [json_data[str(i)][1].replace('_', ' ') for i in range(1000)]

# Required, FastAPI root
@app.get("/")
async def hello():
    return {"status": "hello"}

# Serve model requests
@app.post("/predict")
async def predict(image_url: str):
    try:
        response = requests.get(image_url)
```

```
        response.raise_for_status()
        img = Image.open(BytesIO(response.content)).convert("RGB")
        input_tensor = preprocess(img).unsqueeze(0).to(device)

        with torch.no_grad():
            output = gpunet(input_tensor)

        probs = softmax(output, dim=1)[0]
        top5_idx = probs.topk(5).indices.cpu().numpy()
        top5_probs = probs.topk(5).values.cpu().numpy()

        results = [{ "label": labels[idx], "probability": float(prob) } for idx, prob
 in zip(top5_idx, top5_probs)]

        return {"predictions": results}
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))
```

创建 Dockerfile

以下 Dockerfile 使用 NVIDIA Deep Learning Examples for Tensor Cores GitHub 存储库的 GPUNet 模型为我们的应用程序创建容器映像。

我们通过以下方式减小容器映像大小：使用仅运行时的 PyTorch 基础映像、只安装必要的软件包并清理缓存、预缓存模型代码，以及避免在容器映像中“烘焙”权重，从而实现更快的拉取和更新。要了解更多信息，请参阅 Reducing Container Image Sizes。

在与 `app.py` 相同的目录中，创建 `Dockerfile`：

```
FROM pytorch/pytorch:2.4.0-cuda12.4-cudnn9-runtime

# Install required system packages required for git cloning
RUN apt-get update && apt-get install -y git && rm -rf /var/lib/apt/lists/*

# Install application dependencies
RUN pip install --no-cache-dir fastapi uvicorn requests pillow boto3 timm==0.5.4

# Pre-cache the GPUNet code from Torch Hub (without weights)
# Clone the repository containing the GPUNet code
RUN mkdir -p /cache/torch/hub && \
    cd /cache/torch/hub && \
    git clone --branch torchhub --depth 1 https://github.com/NVIDIA/
DeepLearningExamples NVIDIA_DeepLearningExamples_torchhub
```

```
COPY app.py /app/app.py

WORKDIR /app

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "80"]
```

测试应用程序

从与 app.py 和 Dockerfile 相同的目录中，构建推理应用程序的容器映像，目标架构为 AMD64：

```
docker build --platform linux/amd64 -t gpunet-inference-app .
```

设置 Amazon 凭证的环境变量，并可选设置 Amazon 会话令牌。例如：

```
export AWS_REGION="us-east-1"
export AWS_ACCESS_KEY_ID=ABCEXAMPLESCUJFEIELSMUHHAZ
export AWS_SECRET_ACCESS_KEY=123EXAMPLEMZREoQXr8XkiicsOgWDQ5TpUsq0/Z
```

在本地运行容器，将 Amazon 凭证作为环境变量注入以访问 S3。例如：

```
docker run --platform linux/amd64 -p 8080:80 \
  -e S3_BUCKET_NAME=${S3_BUCKET_NAME} \
  -e AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID} \
  -e AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY} \
  -e AWS_DEFAULT_REGION=${AWS_REGION} \
  gpunet-inference-app
```

预期输出应如下所示：

```
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
```

在新的终端窗口中，通过发送以公共图像 URL 作为查询参数的示例 POST 请求来测试推理端点：

```
curl -X POST "http://localhost:8080/predict?image_url=http://images.cocodataset.org/
test-stuff2017/000000024309.jpg"
```

预期输出应为包含前 5 个预测的 JSON 响应，类似如下（实际标签和概率可能会因图像和模型精度而略有不同）：

```
{"predictions":[{"label":"desk","probability":0.28885871171951294},
{"label":"laptop","probability":0.24679335951805115},
{"label":"notebook","probability":0.08539070934057236},
{"label":"library","probability":0.030645888298749924},
{"label":"monitor","probability":0.02989606373012066}]}
```

使用“Ctrl + C”退出应用程序。

将容器推送至 Amazon ECR

在此步骤中，我们将 GPUNet-0 模型服务的容器映像上传到 Amazon Elastic Container Registry（ECR），使其可在 Amazon EKS 上部署。此过程包括创建一个新的 ECR 存储库来存储映像，使用 ECR 进行身份验证，然后标记容器映像并将其推送到我们的注册表。

首先，导航回到本指南开头设置环境变量的目录。例如：

```
cd ..
```

在 Amazon ECR 中创建存储库：

```
aws ecr create-repository --repository-name gpunet-inference-app --region ${AWS_REGION}
```

登录 Amazon ECR：

```
aws ecr get-login-password --region ${AWS_REGION} | docker login --username AWS --
password-stdin ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com
```

预期输出应如下所示：

```
Login Succeeded
```

标记映像：

```
docker tag gpunet-inference-app:latest ${AWS_ACCOUNT_ID}.dkr.ecr.
${AWS_REGION}.amazonaws.com/gpunet-inference-app:latest
```

将映像推送到您的 Amazon ECR 存储库：

```
docker push ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/gpunet-inference-
app:latest
```

最后一步需要几分钟的时间才能完成。

7.（可选）公开模型服务

在此步骤中，您将使用 Amazon 负载均衡器控制器（LBC）在 Amazon EKS 外部公开实时推理模型服务。这包括设置 LBC、使用 Mountpoint S3 CSI 驱动程序将 Amazon S3 的模型权重挂载为持久性卷、部署 GPU 加速的应用程序容器组（pod）、创建用于预置应用程序负载均衡器（ALB）的服务和入口以及测试端点。

首先，验证 Amazon LBC 的容器组身份关联，确认服务账户已正确关联到所需的 IAM 角色：

```
eksctl get podidentityassociation --cluster ${EKS_CLUSTER_NAME} --namespace kube-system
 --service-account-name aws-load-balancer-controller
```

预期输出应如下所示：

```
ASSOCIATION ARN                                                    NAMESPACE    SERVICE
 ACCOUNT NAME        IAM ROLE ARN    OWNER ARN
arn:aws:eks:us-east-1:143095308808:podidentityassociation/eks-rt-inference-us-
east-1/a-buavluu2wp1jropya    kube-system     aws-load-balancer-controller
 arn:aws:iam::143095308808:role/AmazonEKSLoadBalancerControllerRole
```

为集群安全组添加标签

Amazon 负载均衡器控制器仅支持一个安全组，其标签键为 `karpenter.sh/discovery: "${EKS_CLUSTER_NAME}"`，用于选择 Karpenter 的安全组。使用 eksctl 创建集群时，默认集群安全组（带有 `"kubernetes.io/cluster/<cluster-name>: owned"` 标签）不会使用 `karpenter.sh/discovery` 标签自动标记。此标签对于 Karpenter 发现此安全组并将其附加到预置的节点至关重要。连接此安全组可确保与 Amazon 负载均衡器控制器（LBC）兼容，使其能够自动管理通过 Ingress 公开的服务（例如这些步骤中的模型服务）的入站流量规则。

导出集群的 VPC ID：

```
CLUSTER_VPC_ID="$(aws eks describe-cluster --name ${EKS_CLUSTER_NAME} --query
 cluster.resourcesVpcConfig.vpcId --output text)"
```

导出集群的默认安全组：

```
CLUSTER_SG_ID="$(aws ec2 describe-security-groups --filters Name=vpc-id,Values=
$CLUSTER_VPC_ID Name=tag-key,Values=kubernetes.io/cluster/${EKS_CLUSTER_NAME} --query
 'SecurityGroups[].[GroupId]' --output text)"
```

将 karpenter.sh/discovery 标签添加到默认集群安全组。这将允许我们的 CPU 和 GPU EC2NodeClass 选择器使用它：

```
aws ec2 create-tags --resources ${CLUSTER_SG_ID} --tags Key=karpenter.sh/
discovery,Value=${EKS_CLUSTER_NAME}
```

验证该标签是否已添加：

```
aws ec2 describe-security-groups --group-ids ${CLUSTER_SG_ID} --query
 "SecurityGroups[].Tags"
```

在结果中，您应该看到以下内容，其中包含标签和集群名称。例如：

```
{
  "Key": "karpenter.sh/discovery",
  "Value": "eks-rt-inference-us-east-1"
}
```

设置 Amazon 负载均衡器控制器（LBC）

Amazon LBC 对于管理 Amazon EKS 上人工智能/机器学习工作负载的入口流量至关重要，可确保对推理端点或数据处理管道的访问。通过与 Amazon 应用程序负载均衡器（ALB）和网络负载均衡器（NLB）集成，LBC 可以动态地将流量路由到容器化应用程序，例如运行大型语言模型、计算机视觉模型或实时推理服务的应用程序。由于我们在创建集群期间已经创建了服务账户和容器组身份关联，因此将 serviceAccount.name 设置为匹配集群配置（aws-load-balancer-controller）中定义的内容。

添加 Amazon 拥有的 eks-charts Helm 图表存储库：

```
helm repo add eks https://aws.github.io/eks-charts
```

使用最新的图表刷新本地 Helm 存储库：

```
helm repo update eks
```

使用 Helm 部署 Amazon LBC，指定 EKS 集群名称并引用预先创建的服务账户：

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=${EKS_CLUSTER_NAME} \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

预期输出应如下所示：

```
NAME: aws-load-balancer-controller
LAST DEPLOYED: Wed Jul 9 15:03:31 2025
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Amazon Load Balancer controller installed!
```

将模型挂载到持久性卷中

在此步骤中，您将使用由适用于 Amazon S3 的 Mountpoint CSI 驱动程序支持的 PersistentVolume（PV）从 Amazon S3 存储桶挂载模型权重。这使 Kubernetes 容器组（pod）可以将 S3 对象作为本地文件访问，从而无需到临时容器组（pod）存储或初始化容器的资源密集型下载，非常适合大型、多 GB 的模型权重。

PV 挂载整个存储桶根目录（`volumeAttributes` 中未指定路径），支持多个容器组（pod）的并行只读访问，并在容器内公开模型权重等文件（`/models/gpunet-0.pth`）以供推断。这可确保应用程序（`app.py`）中的回退“下载”不会触发，因为文件是通过挂载存在的。通过将模型与容器映像分离，无需重新构建映像即可实现共享访问和独立模型版本更新。

创建 PersistentVolume（PV）

创建 PersistentVolume（PV）资源以挂载包含模型权重的 S3 存储桶，使多个容器组（pod）在运行时可以只读访问而无需下载文件：

```
cat <<EOF | envsubst | kubectl apply -f -
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: s3-model-pv
spec:
  capacity:
    storage: 5Gi  # Ignored by the driver; can be any value
  accessModes:
    - ReadOnlyMany  # Read only
  persistentVolumeReclaimPolicy: Retain
  storageClassName: ""  # Required for static provisioning
  claimRef:
    namespace: default  # Adjust if you prefer a different namespace
    name: s3-model-pvc
  mountOptions:
    - allow-other  # Enables multi-user access (useful for non-root pods)
    - region ${AWS_REGION} # Optional, include if your bucket is in a different region
 than the cluster
  csi:
    driver: s3.csi.aws.com
    volumeHandle: gpunet-model-volume  # Must be unique across all PVs
    volumeAttributes:
      bucketName: ${S3_BUCKET_NAME}
EOF
```

创建 PersistentVolumeClaim（PVC）

创建 PersistentVolumeClaim（PVC）以绑定到 PV，请求对已挂载 S3 模型数据的只读访问：

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: s3-model-pvc
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: ""  # Required for static provisioning
  resources:
    requests:
      storage: 5Gi  # Ignored, match PV capacity
  volumeName: s3-model-pv  # Bind to the PV created above
EOF
```

部署应用程序

将推理应用程序部署为 Kubernetes 部署，挂载 S3 支持的持久性卷以进行模型访问，应用 GPU 节点选择器和容忍，并设置模型路径的环境变量。此部署设置模型路径（"/models/gpunet-0.pth" 的环境变量），因此默认情况下，我们的应用程序（app.py 中）将使用此路径。部署的卷装载到 /models（只读）时，如果文件已经通过 PVC 存在，则不会触发模型下载。

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpunet-inference-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gpunet-inference-app
  template:
    metadata:
      labels:
        app: gpunet-inference-app
    spec:
      tolerations:
      - key: "nvidia.com/gpu"
        operator: "Exists"
        effect: "NoSchedule"
      nodeSelector:
        role: gpu-worker
      containers:
      - name: inference
        image: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/gpunet-inference-
app:latest
        ports:
        - containerPort: 80
        env:
        - name: MODEL_PATH
          value: "/models/gpunet-0.pth"
        resources:
          limits:
            nvidia.com/gpu: 1
          requests:
            nvidia.com/gpu: 1
        volumeMounts:
```

```
        - name: model-volume
          mountPath: /models
          readOnly: true
      volumes:
      - name: model-volume
        persistentVolumeClaim:
          claimName: s3-model-pvc
EOF
```

如果 GPU 节点不可用，Karpenter 需要几分钟才能完成预置。验证推理容器组（pod）是否处于“Running”状态：

```
kubectl get pods -l app=gpunet-inference-app
```

预期输出应如下所示：

```
NAME                                READY   STATUS    RESTARTS   AGE
gpunet-inference-app-5d4b6c7f8-abcde        1/1     Running   0          2m
```

使用 Ingress 和负载均衡器公开服务

创建 ClusterIP 服务，在 EKS 集群内部公开推理部署，目标为应用程序的端口：

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: gpunet-model-service
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: gpunet-inference-app
EOF
```

创建 Ingress 资源，通过 Amazon LBC 预置面向互联网的应用程序负载均衡器（ALB），将外部流量路由到推理服务：

```
cat <<EOF | envsubst | kubectl apply -f -
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gpunet-model-ingress
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: gpunet-model-service
            port:
              number: 80
EOF
```

等待几分钟，以便应用程序负载均衡器（ALB）完成预置。监控 Ingress 资源状态，确认 ALB 已完成预置：

```
kubectl get ingress gpunet-model-ingress
```

预期的输出应如下所示（已填充 ADDRESS 字段）：

```
NAME                   CLASS   HOSTS   ADDRESS
 PORTS   AGE
gpunet-model-ingress   alb     *       k8s-default-gpunetmo-183de3f819-516310036.us-
east-1.elb.amazonaws.com   80      6m58s
```

从 Ingress 状态中提取并导出 ALB 主机名，以供后续测试使用：

```
export ALB_HOSTNAME=$(kubectl get ingress gpunet-model-ingress -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')
```

测试模型服务

通过发送带有示例图像 URL（例如，来自 COCO 数据集）的 POST 请求来验证已公开的推理端点，模拟实时预测：

```
curl -X POST "http://${ALB_HOSTNAME}/predict?image_url=http://images.cocodataset.org/
test-stuff2017/000000024309.jpg"
```

预期输出应为包含前 5 个预测的 JSON 响应，类似如下（实际标签和概率可能会因图像和模型精度而略有不同）：

```
{"predictions":[{"label":"desk","probability":0.2888975441455841},
{"label":"laptop","probability":0.2464350312948227},
{"label":"notebook","probability":0.08554483205080032},
{"label":"library","probability":0.030612602829933167},
{"label":"monitor","probability":0.029896672815084457}]}
```

您可以选择在新的 POST 请求中继续测试其他图像。例如：

```
http://images.cocodataset.org/test-stuff2017/000000024309.jpg
http://images.cocodataset.org/test-stuff2017/000000028117.jpg
http://images.cocodataset.org/test-stuff2017/000000006149.jpg
http://images.cocodataset.org/test-stuff2017/000000004954.jpg
```

结论

在本指南中，您设置了一个针对 GPU 加速实时推理工作负载进行优化的 Amazon EKS 集群。您为集群预置了 G5 EC2 实例，安装了 Mountpoint S3 CSI 驱动程序、EKS 容器组身份代理、EKS 节点监控代理、Bottlerocket AMI、Amazon 负载均衡器控制器（LBC）和 Karpenter 来管理 CPU 和 GPU 节点池。您使用 NVIDIA 设备插件启用 GPU 调度，并为 S3 配置了 PersistentVolume 和 PersistentVolumeClaim 用于访问模型。您通过部署示例 GPU 容器组（pod）、在 Amazon S3 上设置 NVIDIA GPUNet-0 模型的模型访问权限、启用容器组（pod）初始化以及通过应用程序负载均衡器公开推理服务来验证设置。要充分利用您的集群，请将 EKS 节点监控代理配置为自动修复。请务必进行基准测试，包括 GPU 性能、延迟和吞吐量评测，以优化响应时间。要了解更多信息，请参阅 Using Monitoring and Observability Tools for your AI/ML Workloads。

清理

为避免将来产生费用，您需要手动删除关联的 CloudFormation 堆栈，以删除在本指南中创建的所有资源，包括 VPC 网络。

使用 eksctl 及 `--wait` 标志删除 CloudFormation 堆栈：

```
eksctl delete cluster --region ${AWS_REGION} --name ${EKS_CLUSTER_NAME} --wait
```

完成后，您应会看到以下响应输出：

```
2025-07-29 13:03:55 [#]  all cluster resources were deleted
```

使用 Amazon S3 控制台删除在本指南中创建的 Amazon S3 存储桶。

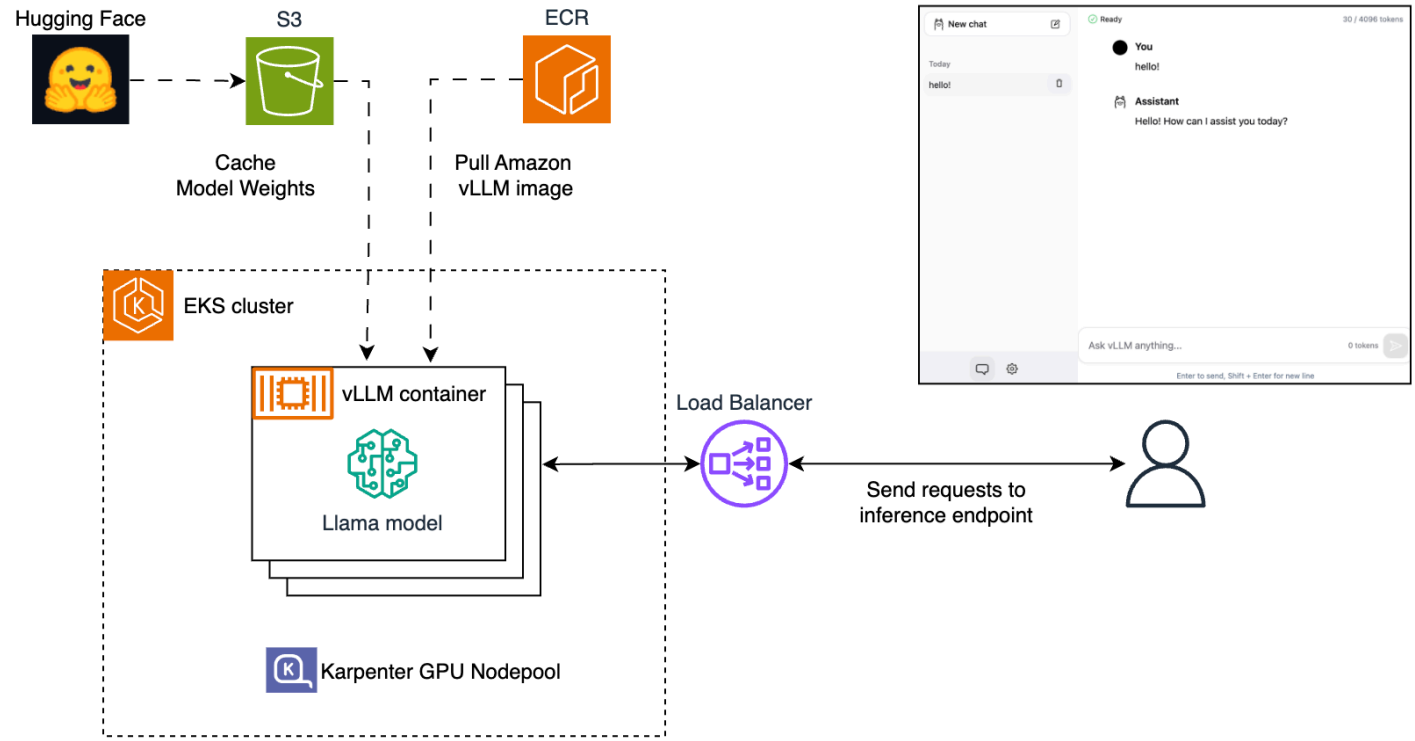
快速入门：在 Amazon EKS 上使用 vLLM 进行高吞吐量 LLM 推理

简介

本快速入门指南提供了使用 vLLM 和 GPU 在 Amazon EKS 上部署大语言模型（LLM）的演练，用于基于文本的实时推理应用程序。

该解决方案利用 Amazon EKS 进行容器编排，利用 vLLM 实现高效的模型服务，让您能够通过 GPU 加速和高吞吐量推理服务构建可扩展的 AI 应用程序。Llama 3.1 8B Instruct 模型用于演示，但您可以部署 vLLM 支持的任何其他 LLM（有关支持的模型列表，请查看 vLLM 文档）。为了测试 LLM 推理，我们使用了一个基于 nextjs-vllm-ui 项目的示例聊天机器人应用程序。最后，我们使用 GuideLLM 对 vLLM 配置参数进行基准测试和调整，以优化推理性能。

EKS 上的 vLLM 架构



完成此过程后，您将拥有针对吞吐量和低延迟进行了优化的 vLLM 推理端点，并且将能够通过聊天前端应用程序与 Llama 模型进行交互，从而演示聊天机器人助手和其他基于 LLM 的应用程序的典型使用案例。

如需其他指导和高级部署资源，请查看EKS AI/ML 工作负载最佳实践指南以及生产就绪型 EKS 推理图表上的 AI。

开始前的准备工作

在开始之前，请确保您满足以下条件：

- 具有包含以下主要组件的 Amazon EKS 集群：带有 G5 或 G6 EC2 实例系列的 Karpenter 节点池、在支持 GPU 的 Worker 节点上安装的 NVIDIA 设备插件，以及安装的 S3 Mountpoint CSI 驱动程序。要创建此基线设置，请按照 the section called “创建集群”中的步骤操作，直至完成步骤 4。
- 一个 Hugging Face 账户。要注册账户，请参阅 https://huggingface.co/login。

使用 Amazon S3 设置模型存储

在 Amazon S3 中高效存储大型 LLM 文件，以便将存储与计算资源分开。此方法可简化模型更新和生产设置中的管理，并降低成本。S3 可以可靠地处理海量文件，而通过 Mountpoint CSI 驱动程序与 Kubernetes 集成，让容器组（pod）可以访问本地存储等模型，无需在启动期间进行耗时的下载。按照以下步骤创建 S3 存储桶、上传 LLM 并将其作为卷挂载到推理服务容器中。

EKS 上还提供用于模型缓存的其他存储解决方案，例如适用于 Lustre 的 EFS 和 FSx。有关更多信息，请查看 EKS 最佳实践。

设置环境变量

为新 Amazon S3 存储桶创建唯一名称，我们将在本指南稍后部分创建该名称。创建后，在所有步骤中使用这一相同的存储桶名称。例如：

```
MY_BUCKET_NAME=model-store-$(date +%s)
```

定义环境变量并将其存储在文件中：

```
cat << EOF > .env-quickstart-vllm
export BUCKET_NAME=${MY_BUCKET_NAME}
export AWS_REGION=us-east-1
export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
EOF
```

在 Shell 环境中加载环境变量。如果关闭当前的 Shell 环境并打开新的 Shell 环境，请确保使用相同的命令重新获取环境变量：

```
source .env-quickstart-vllm
```

创建 S3 存储桶以存储模型文件

创建 S3 存储桶以存储模型文件：

```
aws s3 mb s3://${BUCKET_NAME} --region ${AWS_REGION}
```

从 Hugging Face 下载模型

Hugging Face 是访问 LLM 模型的主要模型中心之一。要下载 Llama 模型，您需要接受模型许可并设置令牌身份验证：

1. 接受 Llama 3.1 8B Instruct 模型许可，网址为 https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct。
2. 生成访问令牌（转到“个人资料”>“设置”>“访问令牌”，然后使用“读取令牌”类型创建新令牌）。

使用 Hugging Face 令牌设置环境变量：

```
export HF_TOKEN=your_token_here
```

如果环境中尚未安装 pip3 软件包，请安装。Amazon Linux 2023 中的示例命令：

```
sudo dnf install -y python3-pip
```

安装 Hugging Face CLI：

```
pip install huggingface-hub
```

使用 `--exclude` 标志从 Hugging Face 下载 Llama-3.1-8B-Instruct 模型（约 15 GB），跳过旧版 PyTorch 格式，仅下载经过优化的 Safetensors 格式文件，这样可以减少下载大小，同时保持与常用推理引擎完全兼容：

```
huggingface-cli download meta-llama/Meta-Llama-3.1-8B-Instruct \
  --exclude "original/*" \
  --local-dir ./llama-3.1-8b-instruct \
```

```
  --token $HF_TOKEN
```

验证下载文件：

```
$ ls llama-3.1-8b-instruct
```

预期输出应如下所示：

```
LICENSE        config.json                        model-00002-of-00004.safetensors
 model.safetensors.index.json  tokenizer_config.json
README.md      generation_config.json             model-00003-of-00004.safetensors
 special_tokens_map.json
USE_POLICY.md  model-00001-of-00004.safetensors  model-00004-of-00004.safetensors
 tokenizer.json
```

上传模型文件

启用 Amazon 通用运行时（CRT）以提高 S3 传输性能。基于 CRT 的传输客户端为大型文件操作提供了更高的吞吐量和可靠性：

```
aws configure set s3.preferred_transfer_client crt
```

上传模型：

```
aws s3 cp ./llama-3.1-8b-instruct s3://$BUCKET_NAME/llama-3.1-8b-instruct \
  --recursive
```

预期输出应如下所示：

```
...
upload: llama-3.1-8b-instruct/tokenizer.json to s3://model-store-1753EXAMPLE/
llama-3.1-8b-instruct/tokenizer.json
upload: llama-3.1-8b-instruct/model-00004-of-00004.safetensors to s3://model-
store-1753890326/llama-3.1-8b-instruct/model-00004-of-00004.safetensors
upload: llama-3.1-8b-instruct/model-00002-of-00004.safetensors to s3://model-
store-1753890326/llama-3.1-8b-instruct/model-00002-of-00004.safetensors
upload: llama-3.1-8b-instruct/model-00003-of-00004.safetensors to s3://model-
store-1753890326/llama-3.1-8b-instruct/model-00003-of-00004.safetensors
upload: llama-3.1-8b-instruct/model-00001-of-00004.safetensors to s3://model-
store-1753890326/llama-3.1-8b-instruct/model-00001-of-00004.safetensors
```

设置 S3 Mountpoint CSI 权限

S3 Mountpoint CSI 驱动程序支持 Kubernetes 与 S3 之间的原生集成，让容器组（pod）可以像本地存储一样直接访问模型文件，从而无需在容器启动期间使用本地副本。

创建 IAM 策略以允许 S3 挂载点从 S3 存储桶读取：

```
aws iam create-policy \
  --policy-name S3BucketAccess-${BUCKET_NAME} \
  --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Effect\":
 \"Allow\", \"Action\": [\"s3:GetObject\", \"s3:GetObjectVersion\", \"s3:ListBucket
\", \"s3:GetBucketLocation\"], \"Resource\": [\"arn:aws:s3:::${BUCKET_NAME}\",
 \"arn:aws:s3:::${BUCKET_NAME}/*\"]}]}"
```

查看 S3 CSI 驱动程序服务账户注释，找到 S3 Mountpoint CSI 驱动程序使用的 IAM 角色名称：

```
ROLE_NAME=$(kubectl get serviceaccount s3-csi-driver-sa -n kube-system -o
 jsonpath='{.metadata.annotations.eks\.amazonaws\.com/role-arn}' | cut -d'/' -f2)
```

将您的 IAM 策略附加至 S3 Mountpoint CSI 角色：

```
aws iam attach-role-policy \
 --role-name ${ROLE_NAME} \
 --policy-arn arn:aws:iam::${AWS_ACCOUNT_ID}:policy/S3BucketAccess-${BUCKET_NAME}
```

如果集群中未安装 S3 Mountpoint CSI，请按照 the section called “创建集群”中的部署步骤进行操作。

将 S3 存储桶挂载为 Kubernetes 卷

创建持久卷（PV）和持久卷声明（PVC），以提供跨多个推理容器组（pod）对 S3 存储桶的只读访问权限。ReadOnlyMany 访问模式可确保并行访问模型文件，而 CSI 驱动程序可处理 S3 存储桶的挂载：

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: v1
kind: PersistentVolume
metadata:
  name: model-store
spec:
  storageClassName: ""
  capacity:
    storage: 100Gi
```

```
  accessModes:
    - ReadOnlyMany
  persistentVolumeReclaimPolicy: Retain
  mountOptions:
    - region ${AWS_REGION}
  csi:
    driver: s3.csi.aws.com
    volumeHandle: model-store
    volumeAttributes:
      bucketName: ${BUCKET_NAME}
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: model-store
spec:
  storageClassName: ""
  volumeName: model-store
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 100Gi
EOF
```

GPU 基础设施设置

集群节点

我们正在使用 the section called “创建集群”中创建的 EKS 集群。此集群包括 Karpenter 节点池，这些节点池可以为支持 GPU 的节点预调配足够的节点存储空间来下载 vLLM 容器映像。如果使用自定义 EKS 集群，请确保它可以启动支持 GPU 的节点。

实例选择

正确选择用于 LLM 推理的实例，需要确保可用的 GPU 内存足以加载模型权重。Llama 3.1 8B Instruct 的模型权重约为 16 GB（模型文件 .safetensor 的大小），因此我们需要向 vllm 进程提供至少此数量的内存来加载模型。

搭载 A10G GPU 的 Amazon G5 EC2 实例和搭载 L4 GPU 的 G6 EC2 实例每个 GPU 都提供 24 GB 的 VRAM，足以加载 Llama 3.1 8B Instruct 权重。如果您要部署权重更大的模型，请考虑使用多 GPU 实例类型或多节点设置。

NVIDIA 设备驱动程序

NVIDIA 驱动程序为容器提供了高效访问 GPU 资源所需的运行时环境。该驱动程序支持在 Kubernetes 内分配和管理 GPU 资源，使 GPU 可以作为可调度资源使用。

我们的集群使用 EKS Bottlerocket AMI，其中包括所有支持 GPU 的节点上所有必要的 NVIDIA 设备驱动程序和插件，确保容器化工作负载无需额外设置即可立即访问 GPU。如果您使用的是其他类型的 EKS 节点，则需要确保已安装所有必要的驱动程序和插件。

测试 GPU 基础设施

通过执行以下步骤来测试集群的 GPU 功能，确保容器组（pod）可以访问 NVIDIA GPU 资源并在支持 GPU 的节点上正确调度。

部署 Nvidia SMI 测试容器组（pod）：

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: gpu-nvidia-smi-test
spec:
  restartPolicy: OnFailure
  tolerations:
  - key: "nvidia.com/gpu"
    operator: "Exists"
    effect: "NoSchedule"
  nodeSelector:
    role: gpu-worker  # Matches GPU NodePool's label
  containers:
  - name: cuda-container
    image: nvidia/cuda:12.9.1-base-ubuntu20.04
    command: ["nvidia-smi"]
    resources:
      requests:
        memory: "24Gi"
      limits:
        nvidia.com/gpu: 1
EOF
```

查看容器组（pod）日志以检查是否列出了 GPU 详细信息，类似于以下输出（不一定是相同的 GPU 模型）：

```
$ kubectl wait --for=jsonpath='{.status.phase}'=Succeeded pod/gpu-nvidia-smi-test
$ kubectl logs gpu-nvidia-smi-test
```

```
Wed Jul 30 15:39:58 2025
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 570.172.08             Driver Version: 570.172.08     CUDA Version: 12.9
   |
|-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA A10G                    On  |   00000000:00:1E.0 Off |
 0 |
|  0%   30C    P8              9W /  300W |       0MiB /  23028MiB |      0%
 Default |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| Processes:
   |
|  GPU   GI   CI              PID   Type   Process name                        GPU
 Memory |
|        ID   ID                                                               Usage
   |
|
=========================================================================================|
|  No running processes found
   |
+-----------------------------------------------------------------------------------------
+
```

此输出显示容器组（pod）可以成功访问 GPU 资源。

重要提示：此容器组（pod）使用与 the section called “创建集群”中的 Karpenter 节点池一致的 nodeSelector 配置。如果您使用不同的节点池，请确保容器组（pod）与 nodeSelector 和 Tolerations 相匹配。

部署推理容器

服务堆栈决定了推理基础设施的性能和可扩展性。vLLM 已成为生产部署的领先解决方案。vLLM 的架构可为动态请求处理提供持续批处理、实现内核优化以加快推理，以及通过 PageDattention 进行高效的 GPU 内存管理。这些功能，再加上生产就绪的 REST API 和对常用模型格式的支持，使其成为高性能推理部署的最佳选择。

选择 Amazon 深度学习容器映像

Amazon 深度学习容器（DLC）为预先优化的环境提供安全更新、Amazon 基础设施兼容性和优化的驱动程序配置。这降低了部署复杂性和维护开销，同时确保生产就绪性。

在此部署中，我们将使用适用于 vLLM 0.9 的 Amazon DLC，其中包括 Nvidia 库和经优化的 GPU 性能配置，这些配置专门针对 Amazon GPU 实例上的转换器模型推理进行了调整。

```
image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/vllm:0.9-gpu-py312-ec2
```

应用 vLLM Kubernetes 清单

在 EKS 中部署 vLMM 有多种方法。本指南演示了使用 Kubernetes 部署进行的 vLLM 部署，这是一种 Kubernetes 原生且易于入门的方法。有关高级部署选项，请参阅 vLLM 文档和 EKS 蓝图上的人工智能。

通过 Kubernetes 清单定义部署参数，以控制资源分配、节点放置、运行状况探测、服务公开等。使用适用于 vLLM 的 Amazon 深度学习容器映像，将您的部署配置为运行支持 GPU 的容器组（pod）。为 LLM 推理设置优化参数，并通过 Amazon 负载均衡器服务公开兼容 vLLM OpenAPI 的端点：

```
cat <<EOF | envsubst | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vllm-inference-app
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
      app: vllm-inference-app
  template:
    metadata:
      labels:
        app: vllm-inference-app
    spec:
      tolerations:
      - key: "nvidia.com/gpu"
        operator: "Exists"
        effect: "NoSchedule"
      nodeSelector:
        role: gpu-worker
      containers:
      - name: vllm-inference
        image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/vllm:0.9-gpu-py312-ec2
        ports:
        - containerPort: 8000
        env:
        - name: MODEL_PATH
          value: "/mnt/models/llama-3.1-8b-instruct"
        args:
        - "--model=/mnt/models/llama-3.1-8b-instruct"
        - "--host=0.0.0.0"
        - "--port=8000"
        - "--tensor-parallel-size=1"
        - "--gpu-memory-utilization=0.9"
        - "--max-model-len=8192"
        - "--max-num-seqs=1"
        readinessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 5
          timeoutSeconds: 10
        resources:
          limits:
            nvidia.com/gpu: 1
          requests:
            memory: "24Gi"
            cpu: "4"
            ephemeral-storage: "25Gi"  # Ensure enough node storage for vLLM container
image
        volumeMounts:
```

```
        - name: models
          mountPath: /mnt/models
          readOnly: true
      volumes:
      - name: models
        persistentVolumeClaim:
          claimName: model-store
---
apiVersion: v1
kind: Service
metadata:
  name: vllm-inference-svc
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8000
      protocol: TCP
  selector:
    app: vllm-inference-app
EOF
```

检查 vlLM 容器组（pod）是否处于 Ready 1/1 状态：

```
kubectl get pod -l app=vllm-inference-app -w
```

预期输出：

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
vllm-inference-app-65df5fddc8-5kmjm   1/1     1            1           5m
```

提取容器映像并且 vLLM 将模型文件加载到 GPU 内存中可能需要几分钟时间。仅当容器组（pod）处于“就绪且可用”时才继续。

公开服务

通过 Kubernetes 端口转发在本地公开推理端点，用于本地开发和测试。让此命令在单独的终端窗口中运行：

```
export POD_NAME=$(kubectl get pod -l app=vllm-inference-app -o
 jsonpath='{.items[0].metadata.name}')
kubectl port-forward pod/$POD_NAME 8000:8000
```

Amazon 负载均衡器控制器会自动创建向外部公开 vLM 服务端点的网络负载均衡器。通过运行以下命令获取 NLB 端点：

```
NLB=$(kubectl get service vllm-inference-svc -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')
```

是否需要安装 Amazon 负载均衡器控制器？ 按照the section called “ Amazon 负载均衡器控制器”中的部署步骤进行操作。

运行推理

验证推理容器组（pod）

通过转发的端口在本地验证推理容器功能。发送连接请求并确保响应中包含 HTTP 代码 200：

```
$ curl -IX GET "http://localhost:8000/v1/models"
```

```
HTTP/1.1 200 OK
date: Mon, 13 Oct 2025 23:24:57 GMT
server: uvicorn
content-length: 516
content-type: application/json
```

通过 NLB 端点向 LLM 发送完成请求，测试推理功能并验证外部连接：

```
curl -X POST "http://$NLB:80/v1/completions" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "/mnt/models/llama-3.1-8b-instruct",
    "prompt": "Explain artificial intelligence:",
    "max_tokens": 512,
    "temperature": 0.7
  }'
```

此端点遵循 OpenAI API 格式，使其与现有应用程序兼容，同时提供可配置的生成参数（例如响应长度和温度），以控制输出多样性。

运行聊天机器人应用程序

为了进行演示，本指南使用 nextjs-vllm-ui 项目运行示例聊天机器人应用程序，以展示用户与模型的互动。

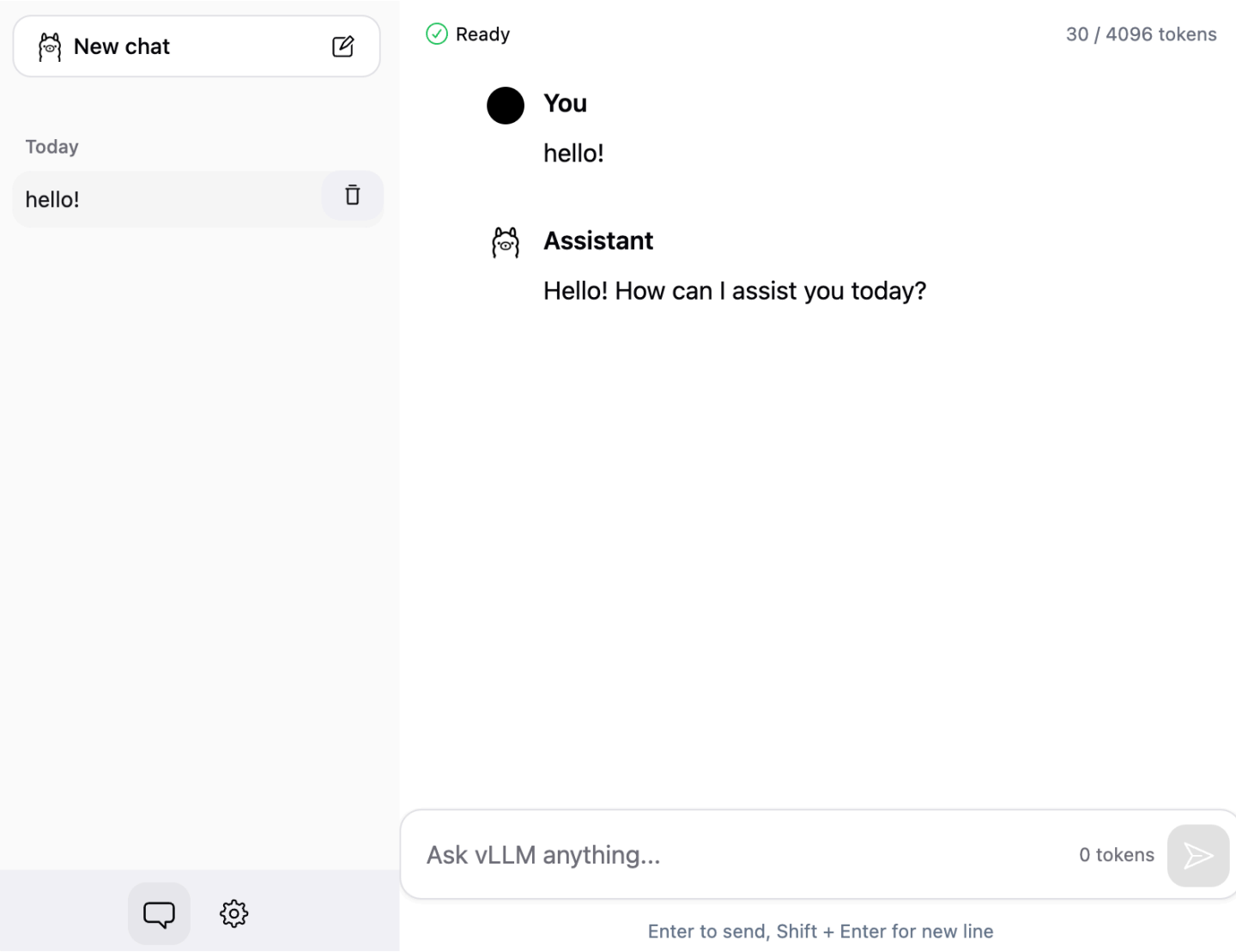
将聊天机器人用户界面作为 Docker 容器运行，该容器可将端口 3000 映射到本地主机并连接至 vLLM NLB 端点：

```
docker run --rm \
  -p 3000:3000 \
  -e VLLM_URL="http://${NLB}:80" \
  --name nextjs-vllm-ui-demo \
  ghcr.io/yoziru/nextjs-vllm-ui:latest
```

打开 Web 浏览器并导航至：http://localhost:3000/

您应该会看到聊天界面，您可以在其中与 Llama 模型互动。

聊天用户界面



优化推理性能

像 vLLM 这样的专业推理引擎提供了可显著提高推理性能的高级功能，包括连续批处理、高效的 KV 缓存和经优化的记忆注意力机制。您可以调整 vLLM 配置参数以提高推理性能，同时满足特定使用案例要求和工作负载模式。正确的配置对于实现 GPU 饱和度至关重要，从而确保您从昂贵的 GPU 资源中获取最大价值，同时提供高吞吐量、低延迟和具有成本效益的操作。以下优化有助于您最大限度地提高 vllM 部署在 EKS 上的性能。

对 vLLM 配置进行基准测试

要针对使用案例调整 vLLM 配置参数，请使用 GuideLLM 等全面推理基准测试工具，对不同的设置进行基准测试。这将收集关键指标，例如每秒请求吞吐量（RPS）、端到端延迟（E2E）、首令牌输出时延（TTFT）和尾部延迟（TPOT），以比较不同的配置。

基线 vLLM 配置

这是用于运行 vlLM 的基线配置：

vlLM 参数	说明
tensor_parallel_size：1	在 1 个 GPU 上分配模型
gpu_memory_utilization：0.90	预留 10% 的 GPU 内存以备系统开销
max_sequence_length：8192	最大序列总长度（输入 + 输出）
max_num_seqs：1	每个 GPU 的最大并发请求数（批处理）

使用此基线设置运行 GuideLLM 以建立性能基线。在此测试中，GuideLLM 配置为每秒生成 1 个请求，包含 256 个令牌请求和 128 个令牌响应。

```
guidellm benchmark \
--target "http://${NLB}:80" \
--processor meta-llama/Llama-3.1-8B-Instruct \
--rate-type constant \
--rate 1 \
--max-seconds 30 \
--data "prompt_tokens=256,output_tokens=128"
```

预期输出：

基线基准测试结果

```
Benchmarks Info:
==========================================================================================================================================
Metadata                                   |||| Requests Made  ||| Prompt Tok/Req ||| Output Tok/Req ||| Prompt Tok Total||| Output Tok Total||
    Benchmark| Start Time| End Time| Duration (s)|  Comp|  Inc|  Err|  Comp|   Inc| Err|  Comp|   Inc| Err|  Comp|   Inc|  Err|  Comp|   Inc|  Err
-------------|-----------|---------|-------------|------|-----|-----|------|------|----|------|------|----|------|------|-----|------|------|-----
constant@1.00|   03:09:34| 03:10:04|         30.0|     6|   24|    0| 256.0| 256.0| 0.0| 128.0| 126.8| 0.0|  1536|  6144|    0|   768|  3043|    0
==========================================================================================================================================


Benchmarks Stats:
==============================================================================================================================================
Metadata     | Request Stats         || Out Tok/sec| Tot Tok/sec| Req Latency (sec) ||| TTFT (ms)            ||| ITL (ms)        ||| TPOT (ms)       ||
    Benchmark| Per Second| Concurrency|        mean|        mean|  mean| median|   p99|   mean| median|     p99| mean| median|  p99| mean| median|  p99
-------------|-----------|------------|------------|------------|------|-------|------|-------|-------|--------|-----|-------|-----|-----|-------|-----
constant@1.00|       0.23|        2.94|        28.9|        86.8| 12.99|  11.27| 21.54| 8637.2| 6923.1| 17185.8| 34.2|   34.2| 34.3| 34.0|   34.0| 34.0
==============================================================================================================================================
```

经调整的 vLLM 配置

调整 vlLM 参数以更好地利用 GPU 资源和并行化：

vLLM 参数	说明
tensor_parallel_size：1	保持在 1 个 GPU。张量并行化必须与 vLLM 要使用的 GPU 数量相匹配
gpu_memory_utilization：0.92	尽可能减少 GPU 内存开销，同时确保 vLLM 继续运行而不会出现错误
max_sequence_length：4096	根据使用案例要求调整最大序列；较低的最大序列可以腾出可用于提高并行化的资源
max_num_seqs：8	增加最大序列会增加吞吐量，但也会增加延迟。增加此值以最大限度地提高吞吐量，同时确保延迟保持在使用案例要求范围内

使用 kubectl patch 命令将这些更改应用于正在运行的部署：

```
kubectl patch deployment vllm-inference-app --type='json' -p='[
  {"op": "replace", "path": "/spec/template/spec/containers/0/args/4", "value": "--gpu-
memory-utilization=0.92"},
  {"op": "replace", "path": "/spec/template/spec/containers/0/args/5", "value": "--max-
model-len=4096"},
  {"op": "replace", "path": "/spec/template/spec/containers/0/args/6", "value": "--max-
num-seqs=8"}
]'
```

检查 vLLM 容器组（pod）是否处于 Ready 1/1 状态：

```
kubectl get pod -l app=vllm-inference-app -w
```

预期输出：

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
vllm-inference-app-65df5fddc8-5kmjm   1/1     1            1           5m
```

然后使用与以前相同的基准测试值再次运行 GuideLLM：

```
guidellm benchmark \
--target "http://${NLB}:80" \
--processor meta-llama/Llama-3.1-8B-Instruct \
```

```
--rate-type constant \
--rate 1 \
--max-seconds 30 \
--data "prompt_tokens=256,output_tokens=128"
```

预期输出：

经优化的基准测试结果

```
Benchmarks Info:
═══════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════
Metadata                                  |||| Requests Made  ||| Prompt Tok/Req ||| Output Tok/Req||| Prompt Tok Total||| Output Tok Total  ||
    Benchmark| Start Time| End Time| Duration (s)|  Comp|  Inc|  Err|  Comp|   Inc| Err|  Comp|  Inc| Err|  Comp|   Inc|  Err|   Comp|   Inc|   Err
-------------|-----------|---------|-------------|------|-----|-----|------|------|----|------|-----|----|------|------|-----|-------|------|------
constant@1.00|   03:20:01| 03:20:31|         30.0|    25|    5|    0| 256.1| 256.0| 0.0| 128.0| 73.6| 0.0|  6403|  1280|    0|   3200|   368|     0
═══════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════


Benchmarks Stats:
═══════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════
Metadata     | Request Stats         || Out Tok/sec| Tot Tok/sec| Req Latency (sec)  ||| TTFT (ms)         ||| ITL (ms)        ||| TPOT (ms)       ||
    Benchmark| Per Second| Concurrency|       mean|        mean|  mean|  median|   p99|  mean| median|   p99| mean| median|  p99| mean| median|  p99
-------------|-----------|------------|-----------|------------|------|--------|------|------|-------|------|-----|-------|-----|-----|-------|-----
constant@1.00|       0.86|        4.45|      109.6|       328.9|  5.19|    5.20|  5.23| 147.9|  148.4| 169.4| 39.7|   39.8| 39.8| 39.4|   39.5| 39.5
═══════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════════
```

基准测试结果

针对基线和经优化的 vLLM 配置，在表格中计算基准测试结果：

平均值	基线配置	经优化的配置
RPS	0.23 个请求/秒	0.86 个请求/秒
E2E	12.99 秒	5.19 秒
TTFT	8637.2 毫秒	147.9 毫秒
TPOT	34.0 毫秒	39.5 毫秒

经优化的 vLLM 配置显著提高了推理吞吐量（RPS）并降低了延迟（E2E、TTFT），尾部延迟（TPOT）仅略增加毫秒。这些结果表明 vLLM 如何显著提高推理性能，让每个容器都可以在更短的时间内处理更多请求，从而实现更具成本效益的操作。

适用于人工智能/机器学习工作负载的 Amazon EKS 集群配置

本节旨在帮助您配置针对人工智能/机器学习工作负载进行优化的 Amazon EKS 集群。您将找到有关使用 Linux 和 Windows 优化的 AMI 运行 GPU 加速容器、使用 Elastic Fabric Adapter（EFA）设置训练

集群以实现高性能联网，以及使用 Amazon Inferentia 实例创建推理集群的指导，包括先决条件、分步过程和部署注意事项。

主题

- 为 GPU 实例使用 EKS 优化版加速 AMI
- 安装适用于 GPU 的 Kubernetes 设备插件
- 运行 GPU 加速容器（EC2 G 系列上的 Windows）
- 使用 Elastic Fabric Adapter 在 Amazon EKS 上运行机器学习训练
- 将 Amazon Inferentia 实例与 Amazon EKS 用于机器学习

为 GPU 实例使用 EKS 优化版加速 AMI

Amazon EKS 支持适用于 GPU 实例的 EKS 优化版 Amazon Linux 和 Bottlerocket AMI。EKS 优化版加速 AMI 通过为加速的 Kubernetes 堆栈提供预先构建、经过验证的操作系统映像，简化了 EKS 集群中人工智能和机器学习工作负载的运行。除了标准 EKS 优化版 AMI 中包含的 Kubernetes 核心组件外，EKS 优化版加速 AMI 还包括运行 NVIDIA GPU G 和 P EC2 实例以及 EKS 集群中的 Amazon GPU Inferentia 和 Trainium EC2 实例所需的内核模块和驱动程序。

下表显示了每个 EKS 优化版加速 AMI 变体支持的 GPU 实例类型。有关 AMI 变体的最新更新，请在 GitHub 上查看 EKS 优化版 AL2023 版本和 Bottlerocket 版本。

EKS AMI 变体	EC2 实例类型
AL2023 x86_64 NVIDIA	p6-b300、p6-b200、p5、p5e、p5en、p4d、p4de、p3、p3dn、gr6、g6、g6e、g6f、gr6f、g5、g4dn
AL2023 ARM NVIDIA	p6e-gb200、g5g
AL2023 x86_64 Neuron	inf1、inf2、trn1、trn2
Bottlerocket x86_64 aws-k8s-nvidia	p6-b300、p6-b200、p5、p5e、p5en、p4d、p4de、p3、p3dn、gr6、g6、g6e、g6f、gr6f、g5、g4dn
Bottlerocket aarch64/arm64 aws-k8s-nvidia	G5g
Bottlerocket x86_64 aws-k8s	inf1、inf2、trn1、trn2

EKS 优化版 NVIDIA AMI

使用 Amazon EKS 优化版 NVIDIA AMI，即表明您同意 NVIDIA 云最终用户许可协议（EULA）。

要查找最新的 EKS 优化版 NVIDIA AMI，请参阅the section called “获取最新 ID”和the section called “获取最新 ID”。

将 Amazon Elastic Fabric Adaptor（EFA）与 EKS 优化版 AL2023 或 Bottlerocket NVIDIA AMI 搭配使用时，必须单独安装 EFA 设备插件。有关更多信息，请参阅 the section called “使用 EFA 设置训练集群”。

EKS AL2023 NVIDIA AMI

将 NVIDIA GPU 操作器与 EKS 优化版 AL2023 NVIDIA AMI 搭配使用时，必须禁用驱动程序和工具包的操作器安装功能，因为这些驱动程序和工具包已包含在 EKS AMI 中。EKS 优化版 AL2023 NVIDIA AMI 不包括 NVIDIA Kubernetes 设备插件或 NVIDIA DRA 驱动程序，这些插件和驱动程序必须单独安装。有关更多信息，请参阅 the section called “安装 NVIDIA Kubernetes 设备插件”。

除标准 EKS AMI 组件外，EKS 优化版 AL2023 NVIDIA 还包含下列组件。

- NVIDIA 驱动程序
- NVIDIA CUDA 用户模式驱动程序
- Nvidia 容器工具包
- NVIDIA Fabric Manager
- NVIDIA 持久化守护进程
- NVIDIA IMEX 驱动程序
- NVIDIA NVLink Subnet Manager
- 最小 EFA（内核模块和 rdma-core）

有关 NVIDIA CUDA 用户模式驱动程序和应用程序容器中使用的 CUDA 运行时/库的详细信息，请参阅 NVIDIA 文档。`nvidia-smi` 显示的 CUDA 版本是安装在主机上的 NVIDIA CUDA 用户模式驱动程序的版本，该驱动程序必须与应用程序容器中使用的 CUDA 运行时/库兼容。

EKS 优化版 AL2023 NVIDIA AMI 支持适用于 Kubernetes 版本 1.33 及更高版本的内核 6.12，以及适用于所有 Kubernetes 版本的 NVIDIA 驱动程序 580 版。需要 NVIDIA 580 驱动程序才能使用 CUDA 13+。

有关 AMI 中包含的组件版本的详细信息，请参阅 GitHub 上的 EKS 优化版 AL2023 版本。有关 EKS AMI 如何配置 NVIDIA 依赖项的详细信息，请参阅 EKS AL2023 NVIDIA AMI 安装脚本和内核加载脚本。您可以使用 `dnf list installed` 命令在正在运行的 EC2 实例上找到已安装的软件包及其版本的列表。

以 EKS 优化版 AMI 为基础构建自定义 AMI 时，不建议且不提供技术支持运行操作系统升级（即 `dnf upgrade`），亦不建议升级 EKS 优化版 AMI 中预装的任何 Kubernetes 或 GPU 相关软件包，因为此类操作可能破坏组件间的兼容性。若仍选择对 EKS 优化版 AMI 中的操作系统或软件包执行升级操作，建议先在开发或预发布环境中开展全面测试，再部署至生产环境。

针对 GPU 实例构建自定义 AMI 时，建议为计划运行的各代实例类型及实例系列分别构建独立的自定义 AMI。EKS 优化版加速型 AMI 会在运行时阶段，根据底层实例的代次与系列选择性安装驱动程序及软件包。有关安装及运行时的详细信息，请参阅 EKS AMI 脚本。

EKS Bottlerocket NVIDIA AMI

将 NVIDIA GPU 操作器与 EKS 优化版 Bottlerocket NVIDIA AMI 搭配使用时，必须禁用驱动程序、工具包和设备插件的操作器安装功能，因为这些驱动程序、工具包和设备插件已包含在 EKS AMI 中。

除标准 EKS AMI 组件外，EKS 优化型 Bottlerocket NVIDIA AMI 还包含下列组件。所有 Bottlerocket 变体中都安装了 EFA（内核模块和 rdma-core）的最少依赖项。

- NVIDIA Kubernetes 设备插件
- NVIDIA 驱动程序
- NVIDIA CUDA 用户模式驱动程序
- Nvidia 容器工具包
- NVIDIA Fabric Manager
- NVIDIA 持久化守护进程
- NVIDIA IMEX 驱动程序
- NVIDIA NVLink Subnet Manager
- NVIDIA MIG Manager

有关 NVIDIA CUDA 用户模式驱动程序和应用程序容器中使用的 CUDA 运行时/库的详细信息，请参阅 NVIDIA 文档。显示的 CUDA 版本`nvidia-smi`是安装在主机上的 NVIDIA CUDA 用户模式驱动程序的版本，该驱动程序必须与应用程序容器中使用的 CUDA 运行时/库兼容。

有关已安装软件包及其版本的详细信息，请参阅 Bottlerocket 文档中的 Bottlerocket 版本信息。EKS 优化版 Bottlerocket NVIDIA AMI 支持适用于 Kubernetes 版本 1.33 及更高版本的内核 6.12，以及适用于 Kubernetes 版本 1.34 及更高版本的 NVIDIA 驱动程序 580 版。需要 NVIDIA 580 驱动程序才能使用 CUDA 13+。

EKS 优化版 Neuron AMI

有关如何将 Neuron 与 Amazon EKS 搭配使用来运行训练和推理工作负载的详细信息，请参阅以下参考资料：

- Amazon Neuron 文档中的 容器 – Kubernetes – 入门
- GitHub 上 Amazon Neuron EKS 示例中的训练示例
- 使用 Inferentia 在 Amazon EKS 上部署机器学习推理工作负载

要查找最新的 EKS 优化版 Neuron AMI，请参阅the section called “获取最新 ID”和the section called “获取最新 ID”。

将 Amazon Elastic Fabric Adaptor（EFA）与 EKS 优化版 AL2023 或 Bottlerocket Neuron AMI 搭配使用时，必须单独安装 EFA 设备插件。有关更多信息，请参阅 the section called “使用 EFA 设置训练集群”。

EKS AL2023 Neuron AMI

EKS 优化版 AL2023 Neuron AMI 不包括 Neuron Kubernetes 设备插件或 Neuron Kubernetes 调度器扩展程序，它们必须单独安装。有关更多信息，请参阅 the section called “安装 Neuron Kubernetes 设备插件”。

除标准 EKS AMI 组件外，EKS 优化版 AL2023 Neuron AMI 还包含下列组件。

- Neuron 驱动程序 (aws-neuronx-dkms)
- Neuronx 工具 (aws-neuronx-tools)
- 最小 EFA（内核模块和 rdma-core）

有关 EKS AMI 如何配置 Neuron 依赖项的详细信息，请参阅 EKS AL2023 Neuron AMI 安装脚本。在 GitHub 上查看 EKS 优化版 AL2023 版本 以及 AMI 中包含的组件版本。您可以使用 `dnf list installed` 命令在正在运行的 EC2 实例上找到已安装的软件包及其版本的列表。

EKS Bottlerocket Neuron AMI

标准 Bottlerocket 变体 (aws-k8s) 包括在 Amazon Inferentia 或 Trainium EC2 实例上运行时会自动检测和加载的 Neuron 依赖项。

EKS 优化版 Bottlerocket AMI 不包括 Neuron Kubernetes 设备插件或 Neuron Kubernetes 调度器扩展程序，它们必须单独安装。有关更多信息，请参阅 the section called “安装 Neuron Kubernetes 设备插件”。

除标准 EKS AMI 组件外，EKS 优化版 Bottlerocket Neuron AMI 还包含下列组件。

- Neuron 驱动程序 (aws-neuronx-dkms)
- 最小 EFA（内核模块和 rdma-core）

在 Neuron 实例中使用 EKS 优化版 Bottlerocket AMI 时，必须在 Bottlerocket 用户数据中配置以下内容。此设置允许容器根据工作负载规范中提供的 `runAsUser` 和 `runAsGroup` 值，获得已挂载 Neuron 设备的所有权。有关 Bottlerocket 中 Neuron 支持的更多信息，请参阅 GitHub 上的 EKS 快速入门自述文件。

```
[settings]
[settings.kubernetes]
device-ownership-from-security-context = true
```

有关 EKS 优化版 Bottlerocket AMI 中包含的 Neuron 驱动程序版本的信息，请参阅 Bottlerocket 内核套件更改日志。

安装适用于 GPU 的 Kubernetes 设备插件

Kubernetes 设备插件一直是将专用基础设施（例如 GPU、网络接口和网络适配器）作为 Kubernetes 工作负载消耗性资源进行公开发布的主要机制。虽然动态资源分配（DRA）定位为 Kubernetes 设备管理的未来，但大多数专业基础设施提供商很早就支持 DRA 驱动程序。如今，Kubernetes 设备插件仍然是在 Kubernetes 集群中使用 GPU 的广泛可用的方法。

注意事项

- 使用搭载 NVIDIA GPU 的 EKS 优化版 AL2023 AMI 时，必须安装 NVIDIA Kubernetes 设备插件。可以通过 Helm、所选的 Kubernetes 工具或 NVIDIA GPU 操作器来安装和管理 NVIDIA Kubernetes 设备插件。

- 使用搭载 NVIDIA GPU 的 EKS 优化版 Bottlerocket AMI 时，无需安装 NVIDIA Kubernetes 设备插件，因为它已经包含在 EKS 优化版 Bottlerocket AMI 中。这包括在 EKS 自动模式下使用 GPU 实例的情况。
- 使用搭载 Amazon Inferentia 或 Trainium GPU 的 EKS 优化版 AL2023 或 Bottlerocket AMI 时，必须安装 Neuron Kubernetes 设备插件，并可以选择安装 Neuron Kubernetes 调度器扩展程序。有关更多信息，请参阅适用于在 EKS 上运行的 Neuron 文档。

安装 NVIDIA Kubernetes 设备插件

以下步骤介绍如何安装 NVIDIA Kubernetes 设备插件以及如何在 NVIDIA GPU 实例上运行示例测试。

先决条件

- EKS 集群已创建
- 使用 EKS 优化版 AL2023 NVIDIA AMI 在集群中运行的 NVIDIA GPU 节点
- 在命令行环境中安装 Helm，请参阅安装 Helm 说明。

过程

1. 添加 nvdp Helm 图表存储库。

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
```

2. 更新本地 Helm 存储库，确保拥有最新的图表。

```
helm repo update
```

3. 获取最新版本的 NVIDIA Kubernetes 设备插件

```
helm search repo nvdp --devel
```

```
NAME                        CHART VERSION APP VERSION DESCRIPTION
nvdp/gpu-feature-discovery  0.17.4        0.17.4      ...
nvdp/nvidia-device-plugin   0.17.4        0.17.4      ...
```

4. 在集群上安装 NVIDIA Kubernetes 设备插件，并将 `0.17.4` 替换为以上命令中的最新版本。

```
helm install nvdp nvdp/nvidia-device-plugin \
```

```
  --namespace nvidia \
  --create-namespace \
  --version 0.17.4 \
  --set gfd.enabled=true
```

5. 验证 NVIDIA Kubernetes 设备插件是否在您的集群中运行。下面的输出显示了集群中有两个节点的输出。

```
kubectl get ds -n nvidia nvdp-nvidia-device-plugin
```

```
NAME                          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
nvdp-nvidia-device-plugin   2         2         2       2            2
 <none>          11m
```

6. 使用以下命令验证节点是否具有可分配的 GPU。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

```
NAME                                           GPU
ip-192-168-11-225.us-west-2.compute.internal   1
ip-192-168-24-96.us-west-2.compute.internal    1
```

7. 使用以下内容创建名为 nvidia-smi.yaml 的文件。此清单会启动最小 AL2023 容器映像，该映像可在节点上运行 nvidia-smi。

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  restartPolicy: OnFailure
  containers:
    - name: gpu-demo
      image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
      command: ['/bin/sh', '-c']
      args: ['nvidia-smi && tail -f /dev/null']
      resources:
        limits:
          nvidia.com/gpu: 1
```

```
  tolerations:
    - key: 'nvidia.com/gpu'
      operator: 'Equal'
      value: 'true'
      effect: 'NoSchedule'
```

8. 使用下面的命令应用清单。

```
kubectl apply -f nvidia-smi.yaml
```

9. 容器组（pod）运行完成后，使用下面的命令查看其日志。

```
kubectl logs nvidia-smi
```

示例输出如下。

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI XXX.XXX.XX            Driver Version: XXX.XXX.XX     CUDA Version: XX.X
     |
|-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
 Compute M. |
|                                         |                        |
 MIG M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:31:00.0 Off |
   0 |
| N/A   27C    P8             11W /   72W |       0MiB /  23034MiB |      0%
 Default |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| Processes:
     |
```

```
|  GPU   GI   CI              PID   Type   Process name                            GPU
 Memory |
|        ID   ID                                                                   Usage
     |
|
=============================================================================================|
|  No running processes found
     |
+---------------------------------------------------------------------------------------------
+
```

安装 Neuron Kubernetes 设备插件

以下步骤介绍如何安装 Neuron Kubernetes 设备插件以及如何在 Inferentia 实例上运行示例测试。

先决条件

- EKS 集群已创建
- 使用 EKS 优化版 AL2023 Neuron AMI 或 Bottlerocket AMI 在集群中运行的 Neuron GPU 节点
- 在命令行环境中安装 Helm，请参阅安装 Helm 说明。

过程

1. 在集群上安装 Neuron Kubernetes 设备插件。

```
helm upgrade --install neuron-helm-chart oci://public.ecr.aws/neuron/neuron-helm-
chart \
    --set "npd.enabled=false"
```

2. 验证 Neuron Kubernetes 设备插件是否在您的集群中运行。下面的输出显示了集群中有一个 Neuron 节点的输出。

```
kubectl get ds -n kube-system neuron-device-plugin
```

```
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
neuron-device-plugin   1         1         1       1            1           <none>
       72s
```

3. 使用以下命令验证节点是否具有可分配的 NueronCores。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,NeuronCore:.status.allocatable.aws\.amazon\.com/
neuroncore"
```

```
NAME                                           NeuronCore
ip-192-168-47-173.us-west-2.compute.internal   2
```

4. 使用以下命令验证节点是否具有可分配的 NueronDevices。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,NeuronDevice:.status.allocatable.aws\.amazon\.com/neuron"
```

```
NAME                                           NeuronDevice
ip-192-168-47-173.us-west-2.compute.internal   1
```

5. 使用以下内容创建名为 neuron-ls.yaml 的文件。此清单会启动安装了 neuron-ls 工具的 Neuron Monitor 容器。

```
apiVersion: v1
kind: Pod
metadata:
  name: neuron-ls
spec:
  restartPolicy: Never
  containers:
  - name: neuron-container
    image: public.ecr.aws/g4h4h0b5/neuron-monitor:1.0.0
    command: ["/bin/sh"]
    args: ["-c", "neuron-ls"]
    resources:
      limits:
        aws.amazon.com/neuron: 1
  tolerations:
  - key: "aws.amazon.com/neuron"
    operator: "Exists"
    effect: "NoSchedule"
```

6. 使用下面的命令应用清单。

```
kubectl apply -f neuron-ls.yaml
```

7. 容器组（pod）运行完成后，使用下面的命令查看其日志。

```
kubectl logs neuron-ls
```

输出示例如下。

```
instance-type: inf2.xlarge
instance-id: ...
+--------+--------+--------+---------+
| NEURON | NEURON | NEURON |   PCI   |
| DEVICE | CORES  | MEMORY |   BDF   |
+--------+--------+--------+---------+
| 0      | 2      | 32 GB  | 00:1f.0 |
+--------+--------+--------+---------+
```

运行 GPU 加速容器（EC2 G 系列上的 Windows）

> **Important**
>
> TensorWorks 推出的适用于 DirectX 的 Kubernetes 设备插件是一个第三方工具，不由 Amazon 认可、支持或维护。Amazon 对此插件的安全性、可靠性或性能不承担任何责任。

了解如何将 NVIDIA GPU 与 TensorWorks 推出的适用于 DirectX 的 Kubernetes 设备插件结合使用，在 Amazon EKS（Elastic Kubernetes Service）上运行 GPU 加速型 Windows 容器工作负载。有关更多信息，请参阅 Kubernetes Device Plugin for DirectX。

为 Windows 容器设置 GPU 加速的方法主要有两种：

- 方法 1：构建自定义 EKS Windows 优化型 AMI并预装必需的 GPU 驱动程序。
  - 当您需要一致的预配置环境来运行 GPU 加速型 Windows 容器，并且可以投入额外的精力来构建和维护自定义 AMI 时，可使用这种方法。
- 方法 2：启动实例后在 EKS Worker 节点上安装必要的 GPU 驱动程序。
  - 如果您想要更简单的设置过程并且不介意在每个新的 Worker 节点上安装 GPU 驱动程序，则可使用此方法。更适合对 GPU 加速型工作负载进行评估或原型设计的开发环境。

这两种方法都可以按照本指南中详述的步骤来利用。

注意事项

本指南提供了使用 NVIDIA GPU、NVIDIA GRID 驱动程序和 TensorWorks 推出的适用于 DirectX 的 Kubernetes 设备插件，为 Windows 容器安装和设置 GPU 加速的步骤。这些步骤已经过测试和验证，能够为您在 Amazon EKS 上的 Windows 容器工作负载提供 GPU 加速。有关兼容驱动程序和设备插件的更多信息，请参阅the section called “已知限制条件”。在开始操作之前，请注意以下事项：

- 只有带 NVIDIA GRID 驱动程序的 G 系列实例类型经过测试和验证，可与本指南配合使用。虽然其他实例类型和驱动程序组合或许也能运行 GPU 加速型 Windows 容器，但可能需要本指南中未涉及的其他配置步骤。
- 只有基于 DirectX 的工作负载经过测试和验证，可与本指南配合使用。虽然其他 GPU API（例如 OpenGL、Vulkan 和 OpenCL）或许可与运行 GPU 加速型 Windows 容器兼容，但可能需要本指南中未涉及的其他配置步骤。
- 在运行 GPU 加速型 Windows 容器之前，需要注意一些已知的限制。请参阅the section called “已知限制条件”部分以了解更多信息。

先决条件

要为 Amazon EKS 上的 Windows 容器启用 GPU 加速，在开始操作之前需要满足以下要求：

- 启动一个使用 Kubernetes 版本 1.27 或更高版本的 Amazon EKS 集群。
- 预置使用 Windows Server 2022 或更高版本的 Windows 节点。
- 在 G 系列实例类型（例如 G4 或 G5）中预置 Windows 节点。
- 预置具有 containerd `1.7.x` 或 `2.x.x` 容器运行时的 Windows 节点。（参阅the section called “获取版本信息”以了解如何验证 Amazon EKS 优化型 AMI 中的 containerd 版本。）

在每个 Windows 节点上安装 GPU 驱动程序

要在 EKS Worker 节点上安装 NVIDIA GRID 驱动程序，请按照 Amazon EC2 实例的 NVIDIA 驱动程序中概述的步骤执行操作。导航到安装选项 – 选项 3：GRID 驱动程序，然后按照安装步骤进行操作。

适用于 Windows Server Core 的安装

对于不提供桌面体验的 Windows Server Core，请使用以下命令静默安装 NVIDIA GRID 驱动程序：

```
$nvidiaInstallerFilePath = nvidia-driver-installer.exe # Replace with path to installer
$installerArguments = "-s -clean -noreboot -noeula"
```

```
Start-Process -FilePath $nvidiaInstallerFilePath -ArgumentList $installerArguments -
Wait -NoNewWindow -PassThru
```

验证安装

运行以下 PowerShell 命令以显示有关实例上 GPU 的诊断信息：

```
nvidia-smi
```

此命令会显示 NVIDIA 驱动程序的版本以及有关 GPU 硬件的信息。确保此命令的输出与您预期安装的 NVIDIA GRID 驱动程序版本相符。

在每个节点上部署 GPU 设备插件

为确保能够发现 GPU 资源并将向 Windows 节点上的容器公开，您需要一个设备插件。将 Tensorworks 的 DirectX 设备插件作为 EKS 集群中的进程守护程序集运行，从而在每个 Worker 节点上部署此插件。按照 README.md 中指定的安装指南进行操作，其中涉及以下步骤。操作建议：

- 将此设备插件部署到 `kube-system` 命名空间中。
- 为进程守护程序集设置适当的资源限制，以确保不会过多消耗节点上的资源。

> **Note**
>
> 设备插件进程守护程序集将作为具有升级权限的主机进程容器在每个节点上运行。建议实施 RBAC 控制措施来限制对此进程守护程序集的访问，从而确保只有获得授权的用户能够执行特权命令。

运行 GPU 加速型容器时，设备插件支持两种模式：

- 单租户模式：此模式将所有 GPU 资源专用于实例上的单个容器。使用以下命令安装支持单租户模式的设备插件。有关更多信息，请参阅 README.md。

  ```
  kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-
  plugins/main/deployments/default-daemonsets.yml"
  ```

- 多租户模式：此模式允许在实例上的多个容器之间共享 GPU 资源。使用以下命令安装支持多租户模式的设备插件。有关更多信息，请参阅 README.md。

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/multitenancy-inline.yml"
```

也使用 ConfigMap 来指定多租户。

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/multitenancy-configmap.yml"
```

验证设备插件部署

部署设备插件后，替换 <namespace> 并运行以下命令来验证 DirectX 设备插件是否在所有 Windows 节点上正常运行。

```
kubectl get ds device-plugin-wddm -n <namespace>
```

验证容器是否已准备好部署

设备插件进程守护程序集在 GPU 驱动的 Windows Worker 节点上开始运行后，使用以下命令验证每个节点是否都有可分配的 GPU。相应的数字应与每个节点上的 DirectX 设备数量相匹配。

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,DirectX:.status.allocatable.directx\.microsoft\.com/display"
```

运行带 GPU 加速的 Windows 容器

在启动容器组之前，请在 `.spec.containers[].resources` 中指定资源名称 `directx.microsoft.com/display`。这将指示您的容器要求启用 GPU 的功能，并且 kube-scheduler 会尝试将容器组部署到具有可用 GPU 资源的预配置 Windows 节点上。

例如，下面的示例命令会启动一个 Job 来运行蒙特卡洛模拟，从而估计 π 的值。此示例来自 Kubernetes Device Plugins for DirectX GitHub 存储库，其中有多个示例可供选择，您可以运行这些示例来测试 Windows 节点 GPU 功能。

```
cat <<EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: example-cuda-montecarlo-wddm
```

```
spec:
  template:
    spec:
      containers:
      - name: example-cuda-montecarlo-wddm
        image: "index.docker.io/tensorworks/example-cuda-montecarlo:0.0.1"
        resources:
          limits:
            directx.microsoft.com/display: 1
      nodeSelector:
        "kubernetes.io/os": windows
      restartPolicy: Never
  backoffLimit: 0
EOF
```

已知限制条件

所有 GPU 都可使用

即使您为给定容器请求了特定数量的 GPU，主机上正在运行的每个容器都可以使用实例上的所有 GPU。此外，默认行为是即使节点上有多个 GPU 可用，主机上运行的所有容器都将使用索引为 0 的 GPU。因此，为确保多 GPU 任务正常运行，必须在应用程序代码中明确指定要使用的具体 GPU 设备。

将某个设备分配给应用程序的确切实现，将取决于您使用的编程语言或框架。例如，假设您使用 CUDA 编程，要选择某个特定的 GPU，则可以使用函数 cudaSetDevice() 显式指定要在应用程序代码中使用的设备。

之所以需要显式指定设备，是因为存在一个影响 Windows 容器的已知问题。您可以通过 microsoft/Windows-Containers issue #333 跟踪此问题的解决进展。下表是这种 GPU 分配行为的直观再现和实际示例。

假设使用 EC2 实例类型 `g4dn.12xlarge` 的单个 Windows 节点有四个 GPU，并假设在此实例上启动三个容器组的场景。该表说明，无论每个容器请求多少个 GPU，所有三个容器组都可以访问实例上的所有四个 GPU，并且默认情况下将使用设备索引为 0 的 GPU。

Pod	请求的 GPU 数	实际的 GPU 访问权限	GPU 默认使用方法	可用的 GPU 索引数	实例 GPU 总数
容器组 1	1 个 GPU	所有 4 个 GPU	索引为 0 的 GPU	0、1、2、3	4

Pod	请求的 GPU 数	实际的 GPU 访问权限	GPU 默认使用方法	可用的 GPU 索引数	实例 GPU 总数
容器组 2	2 个 GPU	所有 4 个 GPU	索引为 0 的 GPU	0、1、2、3	4
容器组 3	1 个 GPU	所有 4 个 GPU	索引为 0 的 GPU	0、1、2、3	4

Kubernetes 设备插件支持

Kubernetes 设备插件的 NVIDIA 官方实现不支持 Windows。您可以在通过 NVIDIA/k8s-device-plugin issue #419 跟踪有关增加官方 Windows 支持的进展。

GPU 计算实例限制

您可以启动的 Amazon EC2 GPU 计算实例数量和类型可能存在服务限制，具体取决于您的 Amazon 账户配置。如果您需要更多容量，可以申请增加配额。

必须构建 Windows GPU 优化型 AMI

Amazon EKS 没有提供 EKS Windows GPU 优化型 AMI 或 EC2 Image Builder 托管式组件。您需要按照本指南中的步骤构建预装了必需 GPU 驱动程序的自定义 EKS Windows 优化型 AMI，或者在启动实例后在 EKS Worker 节点上安装必要的 GPU 驱动程序。

不支持 Inferentia 和 Trainium

Windows 不支持基于 Amazon Inferentia 和 Amazon Trainium 的工作负载。

使用 Elastic Fabric Adapter 在 Amazon EKS 上运行机器学习训练

本主题介绍如何将 Elastic Fabric Adapter（EFA）与部署在 Amazon EKS 集群中的容器组（pod）集成。Elastic Fabric Adapter (EFA) 是 Amazon EC2 实例的网络接口，使您能够在Amazon上运行要求大规模高级别节点间通信的应用程序。其量身定制的操作系统旁路硬件接口增强了实例间通信的性能，这对扩缩这些应用程序至关重要。借助 EFA，采用消息传递接口 (MPI) 的高性能计算 (HPC) 应用程序和采用 NVIDIA Collective Communications Library (NCCL) 的 Machine Learning (ML) 应用程序可以扩展到数千个 CPU 或 GPU。因此，您同时获得了本地 HPC 集群的应用程序性能，以及Amazon云的按需弹性和灵活度。将 EFA 与 Amazon EKS 集群上运行的应用程序集成，可以缩短完成大规模分布式培训工作负载的时间，而无需向集群添加更多实例。有关 EFA 的更多信息，请参阅 Elastic Fabric Adapter。

具有 EFA 的实例类型

Amazon EFA Kubernetes 设备插件 支持所有具有 EFA 的 Amazon EC2 实例类型。要查看具有 EFA 的所有实例类型的列表，请参阅《Amazon EC2 用户指南》中的支持的实例类型。但是，要快速运行机器学习应用程序，我们建议除了 EFA 之外，实例还应具备 nVidia GPU、Amazon Inferentia 芯片或 Amazon Trainium 芯片等硬件加速芯片。要查看具有硬件加速芯片和 EFA 的实例类型列表，请参阅《Amazon EC2 用户指南》中的加速型计算。

在比较实例类型以便进行选择时，应注意实例类型可用的 EFA 网卡数量以及加速卡数量、CPU 大小和内存大小。您最多可以为每张网卡分配一个 EFA。一个 EFA 计为一个网络接口。要查看每种具有 EFA 的实例类型有多少 EFA 可用，请参阅《Amazon EC2 用户指南》中的网卡列表。

仅限 EFA 和 EFA 的接口

Elastic Fabric Adapter（EFA） 是一种网络接口，它结合了弹性网络适配器（ENA）和操作系统旁路接口的功能，由 Amazon 可扩展的可靠数据报（SRD）协议提供支持。EFA 功能允许应用程序直接与硬件通信，实现低延迟传输。您可以选择使用仅限 EFA 的接口仅访问 EFA 功能，从而将通信限制在同一可用区内的接口。

要创建可以具有仅限 EFA 的接口的节点，您必须使用自定义 EC2 启动模板并将 `InterfaceType` 设置为 `efa-only`。在您的自定义启动模板中，您不能将网卡 `0` 设置为仅限 EFA 的接口，因为该接口是 EC2 实例的主网卡和网络接口。您必须拥有 VPC CNI `1.18.5` 或更高版本才能使用仅限 EFA 的接口。如果您使用 Amazon Linux 2，则必须是 `v20240928` 或更高 ami 版本才能使用仅限 EFA 的接口。

以下步骤指导您使用 `eksctl` 创建具有 nVidia GPU 和 EFA 接口的节点的 EKS 集群。您不能使用 `eksctl` 创建使用仅限 EFA 的接口的节点和节点组。

先决条件

- 现有 Amazon EKS 集群。如果您没有现有集群，请参考开始使用 创建一个集群。您的集群必须部署在至少具有一个私有子网的 VPC 中，该子网应当具有足够多的可用 IP 地址以便在其中部署节点。私有子网必须具有由外部设备（如 NAT 网关）提供的出站 Internet 访问。

  如果您计划使用 `eksctl` 创建您的节点组，`eksctl` 还可以为您创建集群。
- 在您的设备或 Amazon CloudShell 上安装和配置 Amazon 命令行界面（Amazon CLI）的版本 `2.12.3` 或更高版本，或版本 `1.27.160` 或更高版本。要查看当前版本，请使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。`yum`、`apt-get` 或适用于 macOS 的 Homebrew 等软件包管理器通常比 Amazon CLI 的最新版本落后几个版本。要安装最新版本，请参阅《Amazon 命令

行界面用户指南》中的安装和使用 aws configure 快速配置。Amazon CloudShell 中安装的 Amazon CLI 版本也可能比最新版本落后几个版本。要对其进行更新，请参阅《Amazon CloudShell 用户指南》中的将 Amazon CLI 安装到您的主目录。

- 您的设备或 Amazon CloudShell 上安装了 `kubectl` 命令行工具。该版本可以与集群的 Kubernetes 版本相同，或者最多早于或晚于该版本一个次要版本。例如，如果您的集群版本为 `1.29`，则可以将 `kubectl` 的 `1.28`、`1.29` 或 `1.30` 版本与之配合使用。要安装或升级 `kubectl`，请参阅 the section called “设置 kubectl 和 eksctl”。
- 必须先安装适用于 Kubernetes 的 Amazon VPC CNI 插件的 `1.7.10` 或更高版本，才能启动支持多个 Elastic Fabric Adapter 的 Worker 节点（例如 p4d 或 p5）。有关更新适用于 Kubernetes 的 Amazon VPC CNI 插件版本的更多信息，请参阅the section called “Amazon VPC CNI”。
- 对于 p6-b200 实例，您必须使用 EFA 设备插件版本 0.5.6 或更高版本。

> **Important**
>
> 在 Kubernetes 中采用 EFA 时需要考虑的一个重要因素，是将 Huge Pages 作为集群中的资源进行配置和管理。有关更多信息，请参阅 Kubernetes 文档中的管理 Huge Pages。安装了 EFA 驱动程序的 Amazon EC2 实例会预先分配 5128 个 2MiB 的 Huge Pages，您可以请求将其作为资源在作业规范中使用。

创建节点组

下面的程序通过一个包含 EFA 接口和 GPUDirect RDMA 且受 `p4d.24xlarge` 支持的节点组，帮助您创建节点组，并使用 EFA 针对多节点 NCCL 性能运行一个 NVIDIA Collective Communications Library (NCCL) 示例测试。该示例可用作使用 EFA 在 Amazon EKS 上进行分布式深度学习培训的模板。

1. 确定您想在其中部署节点的 Amazon 区域中有哪些支持 EFA 的 Amazon EC2 实例类型可用。将 *`region-code`* 替换为您想要在其中部署节点组的 Amazon 区域。

```
aws ec2 describe-instance-types --region region-code \
    --filters Name=network-info.efa-supported,Values=true \
    --query "InstanceTypes[*].[InstanceType]" --output text
```

   部署节点时，您想要部署的实例类型必须在集群所在的 Amazon 区域中可用。

2. 确定您想要部署的实例类型在哪些可用区中可用。本教程使用了 p5.48xlarge 实例类型，必须在上一步所指定的 Amazon 区域的输出中返回该实例类型。在生产集群中部署节点时，请将 *p5.48xlarge* 替换为上一步中返回的任何实例类型。

```
aws ec2 describe-instance-type-offerings --region region-code \
    --location-type availability-zone --filters Name=instance-
type,Values=p4d.24xlarge,p5.48xlarge \
    --query 'InstanceTypeOfferings[*].Location' --output text
```

示例输出如下。

```
us-west-2a    us-west-2c    us-west-2b
```

记下返回的可用区，以供后续步骤使用。将节点部署到集群时，VPC 的子网必须在输出返回的其中一个可用区中具有可用的 IP 地址。

3. 使用 eksctl 创建节点组。您需要在设备或 Amazon CloudShell 上安装 0.215.0 版或更高版本的 eksctl 命令行工具。要安装或更新 eksctl，请参阅 eksctl 文档中的 Installation。

   a. 将以下内容复制到名为 *efa-cluster.yaml* 的文件中。将 example values 替换为您自己的值。您可以将 p5.48xlarge 替换为不同的实例，但是如果您进行替换，请确保 availabilityZones 的值为第 1 步中针对实例类型返回的可用区。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-efa-cluster
  region: region-code
  version: "1.XX"

iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
  - name: my-efa-ng
    instanceType: p5.48xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
```

```
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
```

b. 在现有集群中创建托管节点组。

```
eksctl create nodegroup -f efa-cluster.yaml
```

如果您没有现有集群，可以运行以下命令来创建集群和节点组。

```
eksctl create cluster -f efa-cluster.yaml
```

> **Note**
>
> 此示例中使用的实例类型具有 GPU，因此在使用 Amazon Linux 2 时，`eksctl` 会在每个实例上为您自动安装 NVIDIA Kubernetes 设备插件。这对于 Bottlerocket 来说不是必需的，因为 NVIDIA 设备插件内置于 Bottlerocket 的 EKS NVIDIA 变体中。如果在节点组配置中，`efaEnabled` 设置为 `true`，`eksctl` 还将在节点上自动部署 EFA 设备插件。

将 Bottlerocket 与 EFA 结合使用

Bottlerocket AMI 版本 1.28.0 及更高版本提供对 EFA 的官方支持。要将 Bottlerocket 用于启用 EFA 的节点，请在配置中指定 `amiFamily: Bottlerocket`。如果需要使用自定义 AMI ID，则必须使用标准 `nodeGroups` 而不是 `managedNodeGroups`。

以下为配置示例：

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-efa-bottlerocket-cluster
  region: region-code
  version: "1.XX"

iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]
```

```
managedNodeGroups:
  - name: my-efa-bottlerocket-ng
    instanceType: p5.48xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
    amiFamily: Bottlerocket
    bottlerocket:
      enableAdminContainer: true
      settings:
        kernel:
          sysctl:
            "vm.nr_hugepages": "3000"  # Configures 3000 * 2Mi = 6000Mi hugepages
```

上方的 vm.nr_hugepages sysctl 设置用于配置 2Mi 大页的数量。在此示例中，3000 表示 3000 * 2Mi = 6000Mi 的大页。

验证 EFA 设备插件是否安装

创建带有 efaEnabled: true 的节点组时，eksctl 会自动为您部署 EFA Kubernetes 设备插件。您可以通过以下方式验证该设备插件是否已安装且运行正常：

1. 检查 DaemonSet 的状态：

```
kubectl get daemonsets -n kube-system
```

示例输出：

```
NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
aws-efa-k8s-device-plugin-daemonset   2         2         2       2            2
      <none>          6m16s
...
```

在此示例中，EFA 设备插件 DaemonSet 在两个节点上运行。两者均处于就绪状态且可用。

2. 接下来，验证由 DaemonSet 创建的容器组（pod）：

```
kubectl get pods -n kube-system -l name=aws-efa-k8s-device-plugin
```

示例输出：

```
NAME                                          READY   STATUS    RESTARTS   AGE
aws-efa-k8s-device-plugin-daemonset-d68bs     1/1     Running   0          6m16s
aws-efa-k8s-device-plugin-daemonset-w4l8t     1/1     Running   0          6m16s
```

EFA 设备插件容器组（pod）处于“正在运行”状态，这表明该插件已成功部署且可进行操作。

3. 验证资源注册：

   您可以通过描述节点来确认 `vpc.amazonaws.com/efa` 资源是否已注册到 kubelet：

```
kubectl describe nodes
```

   如果 EFA 资源已正确注册，则该资源将在节点的“容量”和“可分配资源”下列出。例如：

```
Capacity:
  ...
  vpc.amazonaws.com/efa:  4
Allocatable:
  ...
  vpc.amazonaws.com/efa:  4
```

   此输出确认了节点可以识别 EFA 资源，从而使其可供请求该资源的容器组（pod）使用。

（可选）测试 EFA 的性能

建议您测试 EFA 安装。您可以使用 GitHub 上 `aws-samples/awsome-distributed-training` 存储库中的 NCCL Tests。NCCL Tests 使用 Nvidia 集合通信库来评估网络性能。以下步骤会在 Amazon EKS 上提交 NCCL 测试。

1. 部署 Kubeflow MPI Operator：

   对于 NCCL 测试，您可以应用 Kubeflow MPI Operator。MPI Operator 可以轻松地在 Kubernetes 上运行 Allreduce 模式的分布式训练。有关更多信息，请参阅 GitHub 上的 MPI Operator。

2. 运行多节点 NCCL 性能测试以验证 GPUDirectRDMA/EFA：

要通过 EFA 使用 GPUDirectRDMA 验证 NCCL 性能，请运行标准 NCCL 性能测试。有关更多信息，请参阅 GitHub 上的官方 NCCL 测试存储库。

完成以下步骤以运行双节点 NCCL 性能测试。在示例 NCCL 测试作业中，每个 Worker 节点会请求八个 GPU、5210Mi 的 hugepages-2Mi、四个 EFA 和 8000Mi 的内存，这意味着每个 Worker 节点都会使用一个 p5.48xlarge 实例的所有资源。

a. 创建 MPIJob 清单：

将以下内容复制到名为 nccl-tests.yaml 的文件中：

```
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nccl-tests
spec:
  runPolicy:
    cleanPodPolicy: Running
    backoffLimit: 20
  slotsPerWorker: 8
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
         spec:
          restartPolicy: OnFailure
          containers:
          - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
            imagePullPolicy: IfNotPresent
            name: test-nccl-launcher
            env:
             - name: PATH
               value: $PATH:/opt/amazon/efa/bin:/usr/bin
             - name: LD_LIBRARY_PATH
               value: /opt/amazon/openmpi/lib:/opt/nccl/build/lib:/opt/amazon/efa/
lib:/opt/aws-ofi-nccl/install/lib:/usr/local/nvidia/lib:$LD_LIBRARY_PATH
             - name: NCCL_DEBUG
               value: INFO
             - name: NCCL_BUFFSIZE
               value: '8388608'
             - name: NCCL_P2P_NET_CHUNKSIZE
               value: '524288'
             - name: NCCL_TUNER_PLUGIN
```

```
            value: /opt/aws-ofi-nccl/install/lib/libnccl-ofi-tuner.so
        command:
        - /opt/amazon/openmpi/bin/mpirun
        - --allow-run-as-root
        - --tag-output
        - -np
        - "16"
        - -N
        - "8"
        - --bind-to
        - none
        - -x
        - PATH
        - -x
        - LD_LIBRARY_PATH
        - -x
        - NCCL_DEBUG=INFO
        - -x
        - NCCL_BUFFSIZE
        - -x
        - NCCL_P2P_NET_CHUNKSIZE
        - -x
        - NCCL_TUNER_PLUGIN
        - --mca
        - pml
        - ^cm,ucx
        - --mca
        - btl
        - tcp,self
        - --mca
        - btl_tcp_if_exclude
        - lo,docker0,veth_def_agent
        - /opt/nccl-tests/build/all_reduce_perf
        - -b
        - "8"
        - -e
        - "16G"
        - -f
        - "2"
        - -g
        - "1"
        - -c
        - "1"
        - -n
```

```
            - "100"
    Worker:
      replicas: 2
      template:
        spec:
          nodeSelector:
            node.kubernetes.io/instance-type: "p5.48xlarge"
          containers:
          - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
            imagePullPolicy: IfNotPresent
            name: nccl-tests-worker
            volumeMounts:
            - name: shmem
              mountPath: /dev/shm
            resources:
              limits:
                nvidia.com/gpu: 8
                hugepages-2Mi: 5120Mi
                vpc.amazonaws.com/efa: 32
                memory: 32000Mi
              requests:
                nvidia.com/gpu: 8
                hugepages-2Mi: 5120Mi
                vpc.amazonaws.com/efa: 32
                memory: 32000Mi
          volumes:
          - name: shmem
            hostPath:
              path: /dev/shm
```

b. 应用 NCCL-tests MPIJob：

通过应用清单提交 MPIJob。这将创建两个 p5.48xlarge Amazon EC2 实例。

```
kubectl apply -f nccl-tests.yaml
```

示例输出如下。

```
mpijob.kubeflow.org/nccl-tests created
```

c. 验证作业是否启动了容器组（pod）：

查看正在运行的容器组（pod）。

```
kubectl get pods
```

示例输出如下。

```
NAME                                 READY    STATUS     RESTARTS    AGE
nccl-tests-launcher-nbql9    0/1      Init:0/1    0           2m49s
nccl-tests-worker-0          1/1      Running     0           2m49s
nccl-tests-worker-1          1/1      Running     0           2m49s
```

MPI Operator 创建一个启动程序容器组（pod）和 2 个工作线程容器组（pod）（每个节点上一个）。

d. 使用日志来验证作业是否成功运行：

查看 `nccl-tests-launcher` 容器组（pod）的日志。将 *`nbql9`* 替换为输出中的值。

```
kubectl logs -f nccl-tests-launcher-nbql9
```

如果测试成功完成，则可以部署使用 Nvidia 集合通信库的应用程序。

将 Amazon Inferentia 实例与 Amazon EKS 用于机器学习

本主题介绍如何创建一个包含运行 Amazon EC2 Inf1 实例的节点的 Amazon EKS 集群，并（可选地）部署示例应用程序。Amazon EC2 Inf1 实例由 Amazon Inferentia 芯片提供支持，这些芯片由Amazon 定制构建，可在云中提供高性能和最低成本的推理。Machine Learning 模型通过 Amazon Neuron 部署到容器，而 Amazon Neuron 是一个专用的软件开发工具包 (SDK)，由编译器、运行时间和分析工具组成，可用于优化 Inferentia 芯片的 Machine Learning 推理性能。AmazonNeuron 支持常用的 Machine Learning 框架，例如 TensorFlow、PyTorch 和 MXNet。

> **Note**
>
> Neuron 设备逻辑 ID 必须是连续的。如果在 `inf1.6xlarge` 或 `inf1.24xlarge` 实例类型（具有多个 Neuron 设备）上调度请求多个 Neuron 设备的容器组（pod），则当 Kubernetes 调度程序选择不连续的设备 ID 时，该容器组（pod）将无法启动。有关更多信息，请参阅 GitHub 上的设备逻辑 ID 必须是连续的。

先决条件

- 在计算机上安装 eksctl。如果未安装，请参阅 eksctl 文档中的安装。
- 在计算机上安装 kubectl。有关更多信息，请参阅 the section called “设置 kubectl 和 eksctl”。
- （可选）在计算机上安装 python3。如果未安装，请参阅 Python 下载以查看安装说明。

创建集群

1. 创建包含 Inf1 Amazon EC2 实例节点的集群。您可以将 *inf1.2xlarge* 替换为任何 Inf1 实例类型。eksctl 实用程序检测到您在使用 Inf1 实例类型启动节点组，并将使用一个 Amazon EKS 优化版加速型 Amazon Linux AMI 来启动您的节点。

   > **Note**
   >
   > 您不能将服务账户的 IAM 角色与 TensorFlow Serving 结合使用。

   ```
   eksctl create cluster \
       --name inferentia \
       --region region-code \
       --nodegroup-name ng-inf1 \
       --node-type inf1.2xlarge \
       --nodes 2 \
       --nodes-min 1 \
       --nodes-max 4 \
       --ssh-access \
       --ssh-public-key your-key \
       --with-oidc
   ```

   > **Note**
   >
   > 请记住下一个输出行的值。在后面的（可选）步骤中将使用该值。

   ```
   [9]  adding identity "arn:aws-cn:iam::111122223333:role/eksctl-inferentia-nodegroup-
   ng-in-NodeInstanceRole-FI7HIYS3BS09" to auth ConfigMap
   ```

在启动包含 Inf1 实例的节点组时，eksctl 将自动安装 Amazon Neuron Kubernetes 设备插件。此插件将 Neuron 设备作为系统资源传播到 Kubernetes 调度程序，以供容器请求。除了默认的 Amazon EKS 节点 IAM policy 之外，还添加了 Amazon S3 只读访问策略，以便下一个步骤中所述的示例应用程序能够从 Amazon S3 加载经过训练的模型。

2. 确保所有容器组（pod）已正常启动。

```
kubectl get pods -n kube-system
```

缩减的输出：

```
NAME                                   READY   STATUS    RESTARTS   AGE
[...]
neuron-device-plugin-daemonset-6djhp   1/1     Running   0          5m
neuron-device-plugin-daemonset-hwjsj   1/1     Running   0          5m
```

（可选）部署 TensorFlow Serving 应用程序映像

经过训练的模型必须先编译为 Inferentia 目标，才能部署在 Inferentia 实例上。要继续，您将需要一个在 Amazon S3 中保存的经 Neuron 优化的 TensorFlow 模型。如果您还没有 SavedModel，请按照创建 Neuron 兼容的 Resnet50 模型教程操作，并将生成的 SavedModel 上载到 S3。ResNet-50 是一种常用的 Machine Learning 模型，用于图像识别任务。有关如何编译 Neuron 模型的更多信息，请参阅《Amazon Deep Learning AMI 开发人员指南》中的带有 DLAMI 的 Amazon Inferentia 芯片。

示例部署清单管理一个预构建的推理服务容器，用于由 Amazon Deep Learning Containers 提供的 TensorFlow。容器内部是 Amazon Neuron 运行时间和 TensorFlow 服务应用程序。在 GitHub 上的可用镜像下，维护着一个针对 Neuron 优化的预构建 Deep Learning Containers 的完整列表。在启动时，DLC 将从 Amazon S3 中提取您的模型，使用保存的模型启动 Neuron TensorFlow Serving，然后等待预测请求。

可以通过更改部署 yaml 中的 aws.amazon.com/neuron 资源，来调整分配给您的服务应用程序的 Neuron 设备的数量。请注意，TensorFlow Serving 和 Neuron 运行时间之间通过 GRPC 进行通信，这需要将 IPC_LOCK 功能传递给容器。

1. 将 AmazonS3ReadOnlyAccess IAM 策略添加到已在创建集群的第 1 步中创建的节点实例角色。必须执行此操作，示例应用程序才能从 Amazon S3 加载经过训练的模型。

```
aws iam attach-role-policy \
```

```
    --policy-arn arn:aws-cn:iam::aws:policy/AmazonS3ReadOnlyAccess \
    --role-name eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09
```

2. 使用以下内容创建名为 rn50_deployment.yaml 的文件。更新区域代码和模型路径以匹配您需要的设置。当客户端向 TensorFlow 服务器发出请求时，模型名称用于标识目的。此示例使用模型名称匹配一个示例 ResNet50 客户端脚本，后面的步骤中将使用该脚本发送预测请求。

```
aws ecr list-images --repository-name neuron-rtd --registry-id 790709498068 --region
 us-west-2
```

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
    role: master
spec:
  replicas: 2
  selector:
    matchLabels:
      app: eks-neuron-test
      role: master
  template:
    metadata:
      labels:
        app: eks-neuron-test
        role: master
    spec:
      containers:
        - name: eks-neuron-test
          image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-
neuron:1.15.4-neuron-py37-ubuntu18.04
          command:
            - /usr/local/bin/entrypoint.sh
          args:
            - --port=8500
            - --rest_api_port=9000
            - --model_name=resnet50_neuron
            - --model_base_path=s3://${your-bucket-of-models}/resnet50_neuron/
          ports:
            - containerPort: 8500
```

```
            - containerPort: 9000
          imagePullPolicy: IfNotPresent
          env:
            - name: AWS_REGION
              value: "us-east-1"
            - name: S3_USE_HTTPS
              value: "1"
            - name: S3_VERIFY_SSL
              value: "0"
            - name: S3_ENDPOINT
              value: s3.us-east-1.amazonaws.com
            - name: AWS_LOG_LEVEL
              value: "3"
          resources:
            limits:
              cpu: 4
              memory: 4Gi
              aws.amazon.com/neuron: 1
            requests:
              cpu: "1"
              memory: 1Gi
          securityContext:
            capabilities:
              add:
                - IPC_LOCK
```

3. 部署模型。

```
kubectl apply -f rn50_deployment.yaml
```

4. 使用以下内容创建名为 rn50_service.yaml 的文件。这将打开 HTTP 和 gRPC 端口以接受预测请求。

```
kind: Service
apiVersion: v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
spec:
  type: ClusterIP
  ports:
    - name: http-tf-serving
```

```
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
      port: 9000
      targetPort: 9000
  selector:
    app: eks-neuron-test
    role: master
```

5. 为 TensorFlow 模型 Serving 应用程序创建 Kubernetes 服务。

```
kubectl apply -f rn50_service.yaml
```

（可选）根据 TensorFlow Serving 服务进行预测

1. 要在本地进行测试，请将 gRPC 端口转发到 eks-neuron-test 服务。

```
kubectl port-forward service/eks-neuron-test 8500:8500 &
```

2. 创建一个名为 tensorflow-model-server-infer.py 的 Python 脚本，其中包含以下内容。该脚本通过 GRPC（这是一个服务框架）运行推理过程。

```
import numpy as np
   import grpc
   import tensorflow as tf
   from tensorflow.keras.preprocessing import image
   from tensorflow.keras.applications.resnet50 import preprocess_input
   from tensorflow_serving.apis import predict_pb2
   from tensorflow_serving.apis import prediction_service_pb2_grpc
   from tensorflow.keras.applications.resnet50 import decode_predictions

   if __name__ == '__main__':
       channel = grpc.insecure_channel('localhost:8500')
       stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
       img_file = tf.keras.utils.get_file(
           "./kitten_small.jpg",
           "https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/kitten_small.jpg")
       img = image.load_img(img_file, target_size=(224, 224))
       img_array = preprocess_input(image.img_to_array(img)[None, ...])
       request = predict_pb2.PredictRequest()
```

```
        request.model_spec.name = 'resnet50_inf1'
        request.inputs['input'].CopyFrom(
            tf.make_tensor_proto(img_array, shape=img_array.shape))
        result = stub.Predict(request)
        prediction = tf.make_ndarray(result.outputs['output'])
        print(decode_predictions(prediction))
```

3. 运行脚本以将预测数据提交给服务。

```
python3 tensorflow-model-server-infer.py
```

示例输出如下。

```
[[(u'n02123045', u'tabby', 0.68817204), (u'n02127052', u'lynx', 0.12701613),
 (u'n02123159', u'tiger_cat', 0.08736559), (u'n02124075', u'Egyptian_cat',
 0.063844085), (u'n02128757', u'snow_leopard', 0.009240591)]]
```

管理 Amazon EKS 上人工智能/机器学习工作负载的计算资源

本节旨在帮助您管理 Amazon Elastic Kubernetes Service（EKS）中机器学习工作负载的计算资源。您将找到有关使用容量块为托管节点组和自行管理的节点预留 GPU 的详细信息，包括先决条件、启动模板设置、扩展配置、工作负载准备以及处理预留生命周期和正常节点终止的关键注意事项。

主题

- 创建带适用于机器学习的容量块的托管式节点组
- 使用适用于机器学习的容量块创建自行管理的节点
- 将 P6e-GB200 UltraServer 与 Amazon EKS 搭配使用

创建带适用于机器学习的容量块的托管式节点组

机器学习（ML）容量块允许您在未来某个日期预留 GPU 实例，从而支持您的短期 ML 工作负载。有关更多信息，请参阅《Amazon EC2 用户指南（适用于 Linux 实例）》中的适用于 ML 的容量块。

注意事项

> **Important**
>
> - 容量块仅适用于某些 Amazon EC2 实例类型和 Amazon 区域。有关兼容性信息，请参阅《Amazon EC2 用户指南（适用于 Linux 实例）》中的使用容量块。
> - 有关更多信息，请参阅《Amazon EC2 Auto Scaling 用户指南》中的将容量块用于机器学习工作负载。
> - 带有容量块的托管式节点组只能使用自定义启动模板创建。
> - 升级带有容量块的托管式节点组时，请务必将所需的节点组大小设置为 0。

创建带有 Amazon EC2 容量块的托管式节点组

您可以将容量块与 Amazon EKS 托管式节点组配合使用来预置和扩展由 GPU 加速的 Worker 节点。下面的 Amazon CloudFormation 模板示例并未涵盖生产集群所需的各个方面。通常，您还需要一个引导脚本来将节点加入集群，并指定由 Amazon EKS 加速的 AMI。有关更多信息，请参阅 the section called “创建”。

1. 创建适合工作负载并且可与 Amazon EKS 托管节点组结合使用的启动模板。有关更多信息，请参阅 the section called “启动模板”。

   除了上述过程中的要求外，还要确保 `LaunchTemplateData` 包括以下内容：

   - 对于 `InstanceMarketOptions`，将 `MarketType` 设置为 `"capacity-block"`
   - `CapacityReservationSpecification: CapacityReservationTarget`，将 `CapacityReservationId` 设置为该容量块（例如：`cr-02168da1478b509e0`）
   - 将 `InstanceType` 设置为支持容量块的实例类型（例如：`p5.48xlarge`）

   以下 CloudFormation 模板节选将创建一个以容量块为目标的启动模板。要创建自定义 AMI 托管式节点组，您还可以添加 `ImageId` 和 `UserData` 参数。

```
NodeLaunchTemplate:
  Type: "Amazon::EC2::LaunchTemplate"
  Properties:
    LaunchTemplateData:
      InstanceMarketOptions:
        MarketType: "capacity-block"
      CapacityReservationSpecification:
```

```
        CapacityReservationTarget:
          CapacityReservationId: "cr-02168da1478b509e0"
      InstanceType: p5.48xlarge
```

2. 使用启动模板创建托管式节点组。

   以下是容量块的创建节点组命令示例。将 *example-values* 替换为适用于集群的相应值。

   创建容量块托管式节点组时，请执行以下操作：

   - 将 capacity-type 设置为 "CAPACITY_BLOCK"。如果容量类型未设置为"CAPACITY_BLOCK" 或者缺失上述任何其他必需的启动模板值，则创建请求将被拒绝。
   - 在创建请求中指定 subnets 时，请确保仅在与容量预留相同的可用区中指定子网。
   - 如果在创建请求中指定某个非零 desiredSize，Amazon EKS 将在创建自动扩缩组（ASG）时遵守该值。但是，如果创建请求是在容量预留激活之前发出的，则在容量预留处于活动状态之前，ASG 将无法启动 Amazon EC2 实例。这样，ASG 扩缩活动将出现启动错误。每当预留变为活动状态时，实例将会成功启动，并且 ASG 将纵向扩展至创建时提到的 desiredSize。

   ```
   aws eks create-nodegroup \
       --cluster-name my-cluster \
       --nodegroup-name my-mng \
       --node-role node-role-arn \
       --region region-code \
       --subnets subnet-id \
       --scaling-config minSize=node-group-min-size,maxSize=node-group-max-
   size,desiredSize=node-group-desired-size \
       --ami-type "AL2023_x86_64_NVIDIA" \
       --capacity-type "CAPACITY_BLOCK" \
       --launch-template id="lt-id",version=1
   ```

3. 请确保节点在纵向扩展后加入。使用带容量块的托管节点组时，Amazon EKS 集群不会验证启动的实例确实已经加入并注册到集群。
4. 如果您在创建时将 desiredSize 设置为 0，则在容量预留变为活动状态时，您可通过不同的选项来纵向扩展节点组：
   - 为 ASG 创建与容量块预留时间保持一致的计划扩缩策略。有关更多信息，请参阅 Amazon EC2 Auto Scaling 用户指南中的 Amazon EC2 Auto Scaling 的计划扩展。
   - 使用 Amazon EKS 控制台或 eks update-nodegroup-config 更新扩缩配置并设置所需的节点组大小。
   - 使用 Kubernetes Cluster Autoscaler。有关更多信息，请参阅 Amazon 上的 Cluster Autoscaler。

5. 节点组现已准备就绪，可供计划工作负载和容器组（pod）。
6. 为确保能够在预留结束之前正常耗尽容器组（pod），Amazon EKS 使用计划扩缩策略将节点组大小缩减到 0。此计划扩缩策略的名称将设置为 `Amazon EKS Node Group Capacity Scaledown Before Reservation End`。我们建议不要编辑或删除此操作。

   Amazon EC2 会在预留结束前 30 分钟开始关闭实例。因此，Amazon EKS 将在预留结束前 40 分钟设置节点组计划缩减，以便安全、正常地退出容器组（pod）。

使用适用于机器学习的容量块创建自行管理的节点

机器学习（ML）容量块允许您在未来某个日期预留 GPU 实例，从而支持您的短期 ML 工作负载。有关更多信息，请参阅《Amazon EC2 用户指南（适用于 Linux 实例）》中的适用于 ML 的容量块。

注意事项

Important

- 容量块仅适用于某些 Amazon EC2 实例类型和 Amazon 区域。有关兼容性信息，请参阅《Amazon EC2 用户指南（适用于 Linux 实例）》中的使用容量块。
- 如果您在容量预留变为活动状态之前创建自行管理的节点组，请将所需容量设置为 0。
- 为了留出足够的时间来正常地耗尽节点，我们建议您在容量块预留结束时间前 30 分钟计划扩展至零。
- 为确保能正常耗尽容器组（pod），我们建议按照示例步骤中的说明设置 Amazon 节点终止处理程序。

将容量块与自行管理的节点配合使用

您可以将容量块与 Amazon EKS 配合使用来预置和扩展自己的自行管理节点。以下步骤给出了一般的示例概述。Amazon CloudFormation 模板示例并未涵盖生产工作负载所需的各个方面。通常，您还需要一个引导脚本才能将节点加入集群，其中应指定一个由 Amazon EKS 加速的 AMI，以及用于加入集群的适当实例配置文件。有关更多信息，请参阅 the section called “Amazon Linux”。

1. 创建适合工作负载的启动模板。有关更多信息，请参阅《Amazon EC2 Auto Scaling 用户指南》中的将容量块用于机器学习工作负载。

   确保 `LaunchTemplateData` 包含以下内容：

- 对于 InstanceMarketOptions，将 MarketType 设置为 "capacity-block"
- CapacityReservationSpecification: CapacityReservationTarget，将 CapacityReservationId 设置为该容量块（例如：cr-*02168da1478b509e0* ）
- IamInstanceProfile，将 Arn 设置为适用的 *iam-instance-profile-arn*
- 将 ImageId 设置为适用的 *image-id*
- 将 InstanceType 设置为支持容量块的实例类型（例如：*p5.48xlarge* ）
- 将 SecurityGroupIds 设置为适用的 ID（例如：*sg-05b1d815d1EXAMPLE* ）
- 将 UserData 设置您自行管理的节点组的适用 *user-data*

  以下 CloudFormation 模板节选将创建一个以容量块为目标的启动模板。

```
NodeLaunchTemplate:
  Type: "aws::EC2::LaunchTemplate"
  Properties:
    LaunchTemplateData:
      InstanceMarketOptions:
        MarketType: "capacity-block"
      CapacityReservationSpecification:
        CapacityReservationTarget:
          CapacityReservationId: "cr-02168da1478b509e0"
      IamInstanceProfile:
        Arn: iam-instance-profile-arn
      ImageId: image-id
      InstanceType: p5.48xlarge
      KeyName: key-name
      SecurityGroupIds:
      - sg-05b1d815d1EXAMPLE
      UserData: user-data
```

  由于容量块是区域性的，因此必须将子网传递到进行预留的可用区。

2. 使用启动模板创建自行管理的节点组。如果您在容量预留变为活动状态之前执行此操作，请将所需容量设置为 0。创建节点组时，请确保仅为预留容量的可用区指定相应的子网。

   您在创建适合工作负载的模板时可以参考以下 CloudFormation 模板示例。此示例会获取上一步骤中所示 Amazon::Amazon EC2::LaunchTemplate 资源的 LaunchTemplateId 和 Version。它还会获取 DesiredCapacity、MaxSize、MinSize 和 VPCZoneIdentifier 的值，这些内容在同一模板的其他位置声明。

```
NodeGroup:
  Type: "Amazon::AutoScaling::AutoScalingGroup"
  Properties:
    DesiredCapacity: !Ref NodeAutoScalingGroupDesiredCapacity
    LaunchTemplate:
      LaunchTemplateId: !Ref NodeLaunchTemplate
      Version: !GetAtt NodeLaunchTemplate.LatestVersionNumber
    MaxSize: !Ref NodeAutoScalingGroupMaxSize
    MinSize: !Ref NodeAutoScalingGroupMinSize
    VPCZoneIdentifier: !Ref Subnets
    Tags:
      - Key: Name
        PropagateAtLaunch: true
        Value: !Sub ${ClusterName}-${NodeGroupName}-Node
      - Key: !Sub kubernetes.io/cluster/${ClusterName}
        PropagateAtLaunch: true
        Value: owned
```

3. 成功创建节点组后，请务必记录已创建节点组的 NodeInstanceRole。您需要此信息以确保在扩展节点组时，新节点会加入集群，并且 Kubernetes 能够识别节点。有关更多信息，请参阅创建自行管理的 Amazon Linux 节点中的 Amazon Web Services 管理控制台说明。
4. 我们建议您为自动扩缩组创建与容量块预留时间保持一致的计划扩缩策略。有关更多信息，请参阅 Amazon EC2 Auto Scaling 用户指南中的 Amazon EC2 Auto Scaling 的计划扩展。

   您可以在容量块结束时间前 30 分钟使用预留的所有实例。那时仍在运行的实例将开始终止。为了留出足够的时间来正常地耗尽节点，我们建议您在容量块预留结束时间前 30 分钟计划扩展至零。

   如果您想改为在容量预留变为 Active 时手动纵向扩展，则需要在容量块预留开始时更新自动扩缩组的所需容量。然后，您还需要在容量块预留结束前 30 多分钟手动缩减。
5. 节点组现已准备就绪，可供计划工作负载和容器组（pod）。
6. 为确保能正常耗尽容器组（pod），我们建议设置 Amazon 节点终止处理程序。该处理程序将能够使用 EventBridge 监控来自 Amazon EC2 Auto Scaling 的“ASG 横向缩减”生命周期事件，并允许 Kubernetes 控制面板在实例不可用之前执行所需的操作。否则，容器组（pod）和 Kubernetes 对象将卡在待处理状态。有关更多信息，请参阅 GitHub 上的 Amazon 节点终止处理程序。

   如果您没有设置节点终止处理程序，我们建议您在到达 30 分钟时段之前开始手动耗尽容器组（pod），这样就有充足的时间正常耗尽这些容器组。

将 P6e-GB200 UltraServer 与 Amazon EKS 搭配使用

本主题介绍如何配置和使用 Amazon EKS 与 P6e-GB200 UltraServer。配备 4 个 NVIDIA Blackwell GPU 的 p6e-gb200.36xlarge 实例类型仅作为 p6e-GB200 UltraServer 提供。P6e-GB200 UltraServer 有两种类型。u-p6e-gb200x36 UltraServer 有 9 个 p6e-gb200.36xlarge 实例，u-p6e-gb200x72 UltraServer 有 18 个 p6e-gb200.36xlarge 实例。

要了解更多信息，请参阅 Amazon EC2 P6e-GB200 UltraServer 网页。

注意事项

- Amazon EKS 支持适用于 Kubernetes 1.33 及更高版本的 P6e-GB200 UltraServer。此 Kubernetes 版本支持动态资源分配（DRA），在 EKS 和 AL2023 EKS 优化版加速 AMI 中默认已启用。将 P6e-GB200 UltraServer 与 EKS 搭配使用才能实现 DRA。Karpenter 或 EKS 自动模式不支持 DRA，建议在将 p6e-GB200 UltraServer 与 EKS 搭配使用时，使用 EKS 自主管理型节点组或 EKS 托管式节点组。
- p6e-GB200 UltraServer 可通过 EC2 机器学习容量块提供。有关如何使用容量块启动 EKS 节点的信息，请参阅the section called “容量管理”。
- 将 EKS 托管式节点组与容量块搭配使用时，必须使用自定义启动模板。升级采用 P6e-GB200 UltraServer 的 EKS 托管式节点组时，在升级前必须将所需的节点组大小设置为 0。
- 建议使用 EKS 优化版加速 AMI 的 AL2023 ARM NVIDIA 变体。此 AMI 包括使用 p6e-GB200 UltraServer 所需的节点组件和配置。如果您决定构建自己的 AMI，则应负责安装和验证节点与系统软件（包括驱动程序）的兼容性。有关更多信息，请参阅 the section called “使用 EKS Linux GPU AMI”。
- 建议使用 EKS 优化版 AMI 版本 v20251103 或更高版本，其中包括 NVIDIA 驱动程序版本 580。此 NVIDIA 驱动程序版本启用了基于驱动程序的一致性内存管理（CDMM），可解决潜在的内存过度报告问题。启用 CDMM 后，不支持以下功能：NVIDIA 多实例 GPU（MIG）和 vGPU。有关 CDMM 的更多信息，请参阅 NVIDIA Coherent Driver-based Memory Management (CDMM)。
- 在将 NVIDIA GPU 操作器与 EKS 优化版 AL2023 NVIDIA AMI 搭配使用时，必须禁用驱动程序和工具包的操作器安装功能，因为这些驱动程序和工具包已包含在 AMI 中。EKS 优化版 AL2023 NVIDIA AMI 不包括 NVIDIA Kubernetes 设备插件或 NVIDIA DRA 驱动程序，这些插件和驱动程序必须单独安装。
- 每个 p6e-gb200.36xlarge 实例最多可以配置 17 张网卡，并且可以利用 EFA 在 UltraServer 之间进行通信。工作负载网络流量可以跨 UltraServer 进行传输，但为了获得最高性能，建议在同一个 UltraServer 中计划工作负载，并利用 IMEX 进行 UltraServer 内的 GPU 通信。有关更多信息，请参阅 P6e-GB200 实例的 EFA 配置。

- 每个 p6e-gb200.36xlarge 实例有 3 个 7.5 TB 的实例存储空间。默认情况下，EKS 优化版 AMI 不会格式化和挂载实例存储。节点的临时存储空间可以在请求临时存储的容器组（pod）和下载到该节点的容器映像之间共享。如果使用 AL2023 EKS 优化版 AMI，则可以通过将 NodeConfig 中的实例本地存储策略设置为 RAID0，将该 AMI 配置为用户数据中节点引导的一部分。如果设置为 RAID0 条带，则实例会存储并配置容器运行时和 kubelet 以使用此临时存储。

组件

建议使用以下组件在 EKS 上使用 p6e-GB200 UltraServer 运行工作负载。可以选择使用 NVIDIA GPU 操作器来安装 NVIDIA 节点组件。在将 NVIDIA GPU 操作器与 EKS 优化版 AL2023 NVIDIA AMI 搭配使用时，必须禁用驱动程序和工具包的操作器安装功能，因为这些驱动程序和工具包已包含在 AMI 中。

堆栈	组件
EKS 优化版加速 AMI	内核 6.12
	NVIDIA GPU 驱动程序
	NVIDIA CUDA 用户模式驱动程序
	Nvidia 容器工具包
	NVIDIA Fabric Manager
	NVIDIA IMEX 驱动程序
	NVIDIA NVLink Subnet Manager
	EFA 驱动程序
在节点上运行的组件	VPC CNI
	EFA 设备插件
	NVIDIA K8s 设备插件
	NVIDIA DRA 驱动程序
	NVIDIA 节点功能发现（NFD）

堆栈	组件
	NVIDIA GPU 功能发现（GFD）

上表中的节点组件可执行以下功能：

- VPC CNI：将 VPC IP 分配为在 EKS 上运行的容器组（pod）的主网络接口
- EFA 设备插件：将 EFA 设备分配为在 EKS 上运行的容器组（pod）的辅助网络。负责跨 P6e-GB200 UltraServer 传输的网络流量。对于多节点工作负载，UltraServer 内的 GPU 到 GPU 流量可以流经多节点 NVLink。
- NVIDIA Kubernetes 设备插件：将 GPU 分配为在 EKS 上运行的容器组（pod）的设备。建议使用 NVIDIA Kubernetes 设备插件，直到 NVIDIA DRA 驱动程序 GPU 分配功能结束实验阶段。有关更新后的信息，请参阅 NVIDIA DRA 驱动程序版本。
- NVIDIA DRA 驱动程序：启用 ComputeDomain 自定义资源，便于创建 IMEX 域，这些域可跟踪在 p6e-GB200 UltraServer 上运行的工作负载。
  - ComputeDomain 资源描述了节点间内存交换（IMEX）域。将具有 ComputeDomain 的 ResourceClaim 的工作负载部署到集群时，NVIDIA DRA 驱动程序会自动创建在匹配节点上运行的 IMEX DaemonSet，并在工作负载启动之前在节点之间建立 IMEX 通道。要了解有关 IMEX 的更多信息，请参阅适用于多节点 NVLink 系统的 NVIDIA IMEX 概述。
  - NVIDIA DRA 驱动程序使用 NVIDIA GFD 应用的 clique ID 标签 (`nvidia.com/gpu.clique`)，该标签传递了网络拓扑和 NVLink 域的知识。
  - 最佳做法是为每个工作负载作业创建 ComputeDomain。
- NVIDIA 节点功能发现（NFD）：GFD 需要依赖项才能根据发现的节点级属性应用节点标签。
- NVIDIA GPU 功能发现（GFD）：将称为 `nvidia.com/gpu.clique` 的 NVIDIA 标准拓扑标签应用于节点。同一个 `nvidia.com/gpu.clique` 内的节点具有多节点 NVLink 可访问性，您可以在应用程序中使用容器组（pod）亲和性，将容器组（pod）调度到同一 NVLink 域。

过程

以下部分假设您有一个运行 Kubernetes 1.33 或更高版本的 EKS 集群，其中一个或多个节点组的 p6e-GB200 UltraServer 运行 AL2023 ARM NVIDIA EKS 优化版加速 AMI。有关 EKS 自主管理型节点和托管式节点组的先决条件步骤，请参阅the section called “容量管理”中的链接。

以下步骤使用下面的组件。

名称	版本	说明
NVIDIA GPU 操作器	25.3.4+	用于管理所需插件（例如 NVIDIA Kubernetes 设备插件和 NFD/GFD）的生命周期。
NVIDIA DRA 驱动程序	25.8.0+	用于 ComputeDomain CRD 和 IMEX 域管理。
EFA 设备插件	0.5.14+	用于跨 UltraServer 通信。

安装 NVIDIA GPU 操作器

NVIDIA GPU 操作器可简化在 Kubernetes 集群中使用 GPU 所需的组件的管理。由于 NVIDIA GPU 驱动程序和容器工具包是作为 EKS 优化版加速 AMI 的一部分安装，因此必须在 Helm 值配置中将这些对象设置为 `false`。

1. 使用以下配置创建名为 `gpu-operator-values.yaml` 的 Helm 值文件。

```
devicePlugin:
  enabled: true
nfd:
  enabled: true
gfd:
  enabled: true
driver:
  enabled: false
toolkit:
  enabled: false
migManager:
  enabled: false
```

2. 使用在上一步中创建 `gpu-operator-values.yaml` 文件为集群安装 NVIDIA GPU 操作器。

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

```
helm install gpu-operator nvidia/gpu-operator \
 --namespace gpu-operator \
 --create-namespace \
 --version v25.3.4 \
```

```
  --values gpu-operator-values.yaml
```

安装 NVIDIA DRA 驱动程序

自 NVIDIA GPU 操作器版本 v25.3.4 起，必须单独安装 NVIDIA DRA 驱动程序。建议持续关注 NVIDIA GPU 操作器发行说明，因为此内容在未来的版本中可能会发生变更。

1. 使用以下配置创建名为 dra-values.yaml 的 Helm 值文件。请注意 nodeAffinity 和 tolerations 会将 DRA 驱动程序配置为仅在搭载 NVIDIA GPU 的节点上部署。

```
resources:
  gpus:
    enabled: false # set to false to disable experimental gpu support
  computeDomains:
    enabled: true

controller:
  nodeSelector: null
  affinity: null
  tolerations: []

kubeletPlugin:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "nvidia.com/gpu.present"
            operator: In
            values:
            - "true"
  tolerations:
    - key: "nvidia.com/gpu"
      operator: Exists
      effect: NoSchedule
```

2. 使用在上一步中创建的 dra-values.yaml 文件为集群安装 NVIDIA DRA 驱动程序。

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

```
helm install nvidia-dra-driver-gpu nvidia/nvidia-dra-driver-gpu \
  --version="25.8.0" \
  --namespace nvidia-dra-driver-gpu \
  --create-namespace \
  -f dra-values.yaml
```

3. 安装后，DRA 驱动程序会创建 `DeviceClass` 资源，使 Kubernetes 能够了解和分配 `ComputeDomain` 资源，并使 P6e-GB200 UltraServer 上的分布式 GPU 工作负载可以进行 IMEX 管理。

   使用以下命令确认 DRA 资源是否可用。

```
kubectl api-resources | grep resource.k8s.io
```

```
deviceclasses           resource.k8s.io/v1  false        DeviceClass
resourceclaims          resource.k8s.io/v1  true         ResourceClaim
resourceclaimtemplates  resource.k8s.io/v1  true         ResourceClaimTemplate
resourceslices          resource.k8s.io/v1  false        ResourceSlice
```

```
kubectl get deviceclasses
```

```
NAME
compute-domain-daemon.nvidia.com
compute-domain-default-channel.nvidia.com
```

安装 EFA 设备插件

要在 UltraServer 之间使用 EFA 通信，必须安装适用于 EFA 的 Kubernetes 设备插件。P6e-GB200 实例最多可以配置 17 张网卡，并且主 NCI（索引 0）必须是类型 `interface` 且支持高达 100 Gbps 的 ENA 带宽。在节点预调配期间，根据您的要求配置 EFA 和 ENA 接口。查看 P6e-GB200 实例的 EFA 配置 Amazon 文档，了解有关 EFA 配置的更多详细信息。

1. 使用以下配置创建名为 `efa-values.yaml` 的 Helm 值文件。

```
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
```

```
    effect: NoSchedule
```

2. 使用在上一步中创建的 `dra-values.yaml` 文件为集群安装 NVIDIA DRA 操作器。

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update
```

```
helm install efa eks/aws-efa-k8s-device-plugin -n kube-system \
  --version="0.5.14" \
  -f efa-values.yaml
```

例如，如果您在每个 NCI 组中为实例配置了 1 个仅限 EFA 的接口，则在描述节点时，预计每个节点会有 4 个可分配的 EFA 设备。

```
kubectl describe node/<gb200-node-name>
```

```
Capacity:
  ...
  vpc.amazonaws.com/efa:  4
Allocatable:
  ...
  vpc.amazonaws.com/efa:  4
```

通过多节点 NVLink 验证 IMEX

要进行多节点 NVLINK NCCL 测试和其他微基准测试，请查看 awesome-distributed-training GitHub 存储库。以下步骤说明如何使用 nvbandwidth 运行多节点 NVLink 测试。

1. 要在 NVL72 域中的两个节点上运行多节点带宽测试，请先安装 MPI Operator：

```
kubectl create -f https://github.com/kubeflow/mpi-operator/releases/download/v0.7.0/
mpi-operator.yaml
```

2. 创建一个名为`nvbandwidth-test-job.yaml` 的 Helm 值文件，用于定义测试清单。请注意在具有多节点 NVLink 可访问性的同一 NVLink 域中调度 Worker 的 `nvidia.com/gpu.clique` 容器组（pod）亲和性。

   自 NVIDIA DRA 驱动程序版本 `v25.8.0` 起，ComputeDomains 具有弹性，且 `.spec.numNodes` 可在 ComputeDomain 定义中设置为 `0`。查看最新的 NVIDIA DRA 驱动程序发行说明以获取更新。

```
---
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomain
metadata:
  name: nvbandwidth-test-compute-domain
spec:
  numNodes: 0 # This can be set to 0 from NVIDIA DRA Driver version v25.8.0+
  channel:
    resourceClaimTemplate:
      name: nvbandwidth-test-compute-domain-channel

---
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nvbandwidth-test
spec:
  slotsPerWorker: 4 # 4 GPUs per worker node
  launcherCreationPolicy: WaitForWorkersReady
  runPolicy:
    cleanPodPolicy: Running
  sshAuthMountPath: /home/mpiuser/.ssh
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        metadata:
          labels:
            nvbandwidth-test-replica: mpi-launcher
        spec:
          containers:
          - image: ghcr.io/nvidia/k8s-samples:nvbandwidth-v0.7-8d103163
            name: mpi-launcher
            securityContext:
              runAsUser: 1000
            command:
            - mpirun
            args:
            - --bind-to
            - core
            - --map-by
            - ppr:4:node
            - -np
```

```
            - "8"
            - --report-bindings
            - -q
            - nvbandwidth
            - -t
            - multinode_device_to_device_memcpy_read_ce
    Worker:
      replicas: 2 # 2 worker nodes
      template:
        metadata:
          labels:
            nvbandwidth-test-replica: mpi-worker
        spec:
          affinity:
            podAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                  - key: nvbandwidth-test-replica
                    operator: In
                    values:
                    - mpi-worker
                topologyKey: nvidia.com/gpu.clique
          containers:
          - image: ghcr.io/nvidia/k8s-samples:nvbandwidth-v0.7-8d103163
            name: mpi-worker
            securityContext:
              runAsUser: 1000
            env:
            command:
            - /usr/sbin/sshd
            args:
            - -De
            - -f
            - /home/mpiuser/.sshd_config
            resources:
              limits:
                nvidia.com/gpu: 4  # Request 4 GPUs per worker
              claims:
              - name: compute-domain-channel # Link to IMEX channel
          resourceClaims:
          - name: compute-domain-channel
            resourceClaimTemplateName: nvbandwidth-test-compute-domain-channel
```

3. 创建 ComputeDomain，然后使用以下命令启动作业。

```
kubectl apply -f nvbandwidth-test-job.yaml
```

4. 创建 ComputeDomain 时，可以看到工作负载的 ComputeDomain 有两个节点：

```
kubectl get computedomains.resource.nvidia.com -o yaml
```

```
status:
  nodes:
  - cliqueID: <ClusterUUID>.<Clique ID>
    ipAddress: <node-ip>
    name: <node-hostname>
  - cliqueID: <ClusterUUID>.<Clique ID>
    ipAddress: <node-ip>
    name: <node-hostname>
  status: Ready
```

5. 使用以下命令查看作业结果。

```
kubectl logs --tail=-1 -l job-name=nvbandwidth-test-launcher
```

6. 测试完成后，使用以下命令将其删除。

```
kubectl delete -f nvbandwidth-test-job.yaml
```

优化 Amazon EKS 集群以支持人工智能/机器学习工作负载的方案

本节旨在提供优化 Amazon EKS 集群的简明方案，尤其适用于涉及专用硬件的人工智能/机器学习工作负载。您将找到有关如何通过向托管节点组添加污点来防止在特定节点上调度容器组（pod）的指导，包括先决条件、分步过程和部署注意事项。

主题

- 方案：防止在特定节点上调度容器组（pod）

方案：防止在特定节点上调度容器组（pod）

概览

具有专用处理器（例如 GPU）的节点运行成本可能比在标准计算机上的节点更昂贵。为防止这些节点被不需要专用硬件的工作负载占用，您可以使用 Kubernetes 污点。污点会将节点标记为排斥容差不匹配的容器组（pod），从而确保仅调度兼容的工作负载。有关更多信息，请参阅 Kubernetes 文档中的 Taints and Tolerations。

可以使用 Amazon Web Services 管理控制台 或通过 Amazon EKS API，将 Kubernetes 节点污点应用于新的和现有的托管节点组。此方案展示了如何使用 Amazon CLI 对 Amazon EKS 托管节点组应用污点。有关使用 Amazon Web Services 管理控制台 创建带有污点的节点组的信息，请参见 the section called “创建”。

先决条件

- 现有 Amazon EKS 集群。
- Amazon CLI 已安装并配置适当的权限。

步数

步骤 1：创建带污点的节点组

使用 `aws eks create-nodegroup` 命令创建带污点的新托管节点组。此示例应用了一个键为 `dedicated`、值为 `gpuGroup`、效果为 `NO_SCHEDULE` 的污点。

```
aws eks create-nodegroup \
 --cli-input-json '
{
  "clusterName": "my-cluster",
  "nodegroupName": "node-taints-example",
  "subnets": [
     "subnet-1234567890abcdef0",
     "subnet-abcdef01234567890",
     "subnet-021345abcdef67890"
   ],
  "nodeRole": "arn:aws-cn:iam::111122223333:role/AmazonEKSNodeRole",
  "taints": [
     {
         "key": "dedicated",
```

```
        "value": "gpuGroup",
        "effect": "NO_SCHEDULE"
    }
  ]
}'
```

有关更多信息和示例，请参阅 Kubernetes 参考文档中的 taint。

步骤 2：更新现有节点组上的污点

使用 aws eks update-nodegroup-config Amazon CLI 命令为托管节点组添加、删除或替换污点。

```
aws eks update-nodegroup-config
  --cluster-name my-cluster
  --nodegroup-name node-taints-example
  --taints 'removeTaints=[{key=dedicated,value=gpuGroup,effect=NO_SCHEDULE}]'
```

备注

- 创建节点组后，可以使用 UpdateNodegroupConfig API 更新污点。
- 污点键必须以字母或数字开头。可以包含字母、数字、连字符（-）、句点（.）和下划线（_）。最长可为 63 个字符
- 污点键也可以以 DNS 子域前缀和单个 / 开头。如果它以 DNS 子域前缀开头，则长度可以为 253 个字符。
- 值是可选的，必须以字母或数字开头。可以包含字母、数字、连字符（-）、句点（.）和下划线（_）。最长可为 63 个字符
- 直接使用 Kubernetes 或 Amazon Web Services 管理控制台 时，污点效果必须为 NoSchedule、PreferNoSchedule 或 NoExecute。但是，使用 Amazon CLI 或 API 时，污点效果必须是 NO_SCHEDULE、PREFER_NO_SCHEDULE 或 NO_EXECUTE。
- 每个节点组最多允许 50 个污点。
- 如果从节点中手动删除使用托管节点组创建的污点，则 Amazon EKS 不会将污点重新添加回该节点。即使在托管节点组配置中指定了污点，也是如此。

在 Amazon EKS 上开始使用人工智能/机器学习的资源

要在 EKS 上部署机器学习，首先要从这些规范模式中进行选择，让 EKS 集群和机器学习软件和硬件快速准备就绪，以开始运行机器学习工作负载。

研讨会

Amazon EKS 上的生成式人工智能讲习会

了解如何在 Amazon EKS 上开始使用大语言模型（LLM）应用程序和推理。探索如何部署和管理生产级 LLM 工作负载。通过动手实验，您将探索如何利用 Amazon EKS 以及 Amazon 服务和开源工具来创建强大的 LLM 解决方案。讲习会环境提供了所有必要的基础设施和工具，使您可以专注于学习和实施。

使用神经元在 Amazon EKS 上实现生成式人工智能

了解如何在 Amazon EKS 上开始使用大语言模型（LLM）应用程序和推理。探索如何部署和管理生产级 LLM 工作负载，使用向量数据库实现高级 RAG 模式，以及如何使用开源框架构建数据支持的 LLM 应用程序。通过动手实验，您将探索如何利用 Amazon EKS 以及 Amazon 服务和开源工具来创建强大的 LLM 解决方案。讲习会环境提供了所有必要的基础设施和工具，使您可以专注于学习和实施。

最佳实践

Amazon EKS 最佳实践指南中重点介绍人工智能/机器学习的主题提供了以下领域的详细建议，以优化您在 Amazon EKS 上的人工智能/机器学习工作负载。

人工智能/机器学习计算和自动扩展

本节概述了在 Amazon EKS 中优化人工智能/机器学习计算和自动扩展的最佳实践，重点介绍 GPU 资源管理、节点恢复能力和应用程序扩展。内容包括各种策略，例如调度带有知名标签和节点亲和性的工作负载、使用机器学习容量块或按需容量预留，以及使用 EKS 节点监控代理等工具实施节点运行状况检查。

人工智能/机器学习联网

本节概述了优化 Amazon EKS 中的人工智能/机器学习联网以提高性能和可扩展性的最佳实践，包括选择网络带宽更高或支持 Elastic Fabric Adapter（EFA）的实例进行分布式训练、安装 MPI 和 NCCL 等工具，以及启用前缀委派以增加 IP 地址和缩短容器组（pod）启动时间等策略。

人工智能/机器学习安全性

本节重点介绍如何保护数据存储和确保 Amazon EKS 上人工智能/机器学习工作负载的合规性，包括以下实践：使用带有 Amazon Key Management Service（KMS）的 Amazon S3 进行服务器端加

密（SSE-KMS），使用区域 KMS 密钥和 S3 存储桶密钥配置存储桶以降低成本，授予 EKS 容器组（pod）解密等 KMS 操作的 IAM 权限，以及使用 Amazon CloudTrail 日志进行审计。

人工智能/机器学习存储

本节提供在 Amazon EKS 上的人工智能/机器学习工作负载中优化存储的最佳实践，包括以下实践：使用 CSI 驱动程序部署模型以将 S3、FSx for Lustre 或 EFS 等服务挂载为持久性卷，根据工作负载需求选择存储（例如，使用 FSx for Lustre 进行分布式训练，并提供 Scratch-SSD 或 Persistent-SSD 等选项），以及启用数据压缩和条带化等功能。

人工智能/机器学习可观测性

本节重点介绍如何监控和优化 Amazon EKS 上人工智能/机器学习工作负载的 GPU 利用率以提高效率和降低成本，包括针对高 GPU 使用率使用 CloudWatch Container Insights 以及与 Prometheus 和 Grafana 集成的 NVIDIA DCGM-Exporter 等工具的策略，以及我们建议您为人工智能/机器学习工作负载分析的指标。

人工智能/机器学习性能

本节重点介绍如何通过容器映像管理和启动优化来增强 Amazon EKS 上人工智能/机器学习工作负载的应用程序扩展和性能，包括以下实践：使用小型轻量级基础映像或具有多阶段构建的 Amazon Deep Learning Containers，通过 EBS 快照预加载映像或使用 DaemonSet 或部署将映像预拉入运行时缓存。

参考架构

探索这些 GitHub 存储库，获取参考架构、示例代码和实用工具，以便在 Amazon EKS 和其他 Amazon 服务上对人工智能/机器学习工作负载实现分布式训练和推理。

AWSome 分布式训练

该存储库提供了一系列最佳实践、参考架构、模型训练示例和实用工具，用于在 Amazon 上训练大型模型。它支持使用 Amazon EKS 进行分布式训练，包括 EKS 集群的 CloudFormation 模板、自定义 AMI 和容器构建、PyTorch（DDP/FSDP、MegaTronLM、NeMo）和 JAX 等框架的测试用例，以及用于验证、可观测性和性能监控的工具，例如 EFA Prometheus 导出器和 Nvidia Nsight Systems。

AWSome 推理

该存储库提供在 Amazon 上用于优化推理解决方案的参考架构和测试用例，重点介绍 Amazon EKS 和加速 EC2 实例。它包括 VPC 和 EKS 集群的基础设施设置；NVIDIA NIM、TensorRT-LLM、Triton

Inference Server 和 RayService 等框架的项目，以及 Llama3-8B 和 Llama 3.1 405B 等模型的示例。具有使用 K8s LeaderWorkerSet、EKS 自动扩展、多实例 GPU（MIG）的多节点部署功能，以及用于 ASR、推理和 TTS 的音频机器人等实际使用案例。

教程

如果您有兴趣在 EKS 中设置机器学习平台和框架，请浏览本节中所述的教程。此类教程涵盖了所有内容，包括充分利用 GPU 处理器的模式、选择建模工具以及为专业行业构建框架。

在 EKS 上构建生成式人工智能平台

- 在 Amazon EKS 上部署生成式人工智能模型
- 在 Amazon EKS 上构建多租户 JupyterHub 平台

在 EKS 上运行专用的生成式人工智能框架

- 使用 Amazon EKS 上的 NVIDIA NeMo Framework 加速生成式人工智能分布式训练工作负载
- 在 Amazon Elastic Kubernetes Service 上运行 TorchServe

最大限度提高 NVIDIA GPU 在 EKS 上部署机器学习的性能

- 实现 GPU 共享，以便为 EKS 集群高效使用 NVIDIA GPU：

  使用 NVIDIA 时间切片和加速 EC2 实例，在 Amazon EKS 上实现 GPU 共享
- 使用多实例 GPU（MIG）和 NIM 微服务，让每个 EKS 集群上的 GPU 运行更多容器组（pod）：

  在 Amazon EKS 上使用 NVIDIA 的多实例 GPU（MIG），最大限度地提高 GPU 利用率：让每个 GPU 运行更多容器组（pod）以增强性能
- 在 Kubernetes 上使用 Amazon 上的 Kubeflow 构建和部署可扩展机器学习系统

在 EKS 上运行视频编码工作负载

- 在 Amazon EKS 上使用容器中的少量 GPU 传输视频内容

加快推理工作负载的映像加载

- H2O.ai 如何使用 Karpenter 和 Bottlerocket 优化和保护其人工智能/机器学习基础架构

监控机器学习工作负载

- 使用 Amazon 托管的开源服务监控 Amazon EKS 上的 GPU 工作负载
- 在 Amazon CloudWatch 中启用基于容器组（pod）的 GPU 指标

Amazon EKS 工具

Amazon EKS 提供了一套 Amazon 工具，可帮助您更高效地开展管理、排查问题并与 EKS 集群进行交互。这类工具依托人工智能驱动能力与标准化接口，简化集群运维操作、加快问题排查速度，并支持以自然语言的方式与 Kubernetes 资源进行交互。当前可用的工具包括：

- Amazon EKS 模型上下文协议（MCP）服务器：这是一项完全托管的服务，可为 EKS 集群管理提供人工智能驱动的使用体验。EKS MCP 服务器可与兼容 MCP 协议的人工智能代码助手集成，输出关于集群及 Kubernetes 资源的实时上下文信息。您可以通过它以自然语言交互的方式，完成集群管理、应用部署、问题排查及 EKS 文档查询等操作。
- 部署于 Amazon EKS 控制台的 Amazon Q：这是一项直接集成在 Amazon 管理控制台中的人工智能驱动问题排查功能。当您在 EKS 控制台中遇到错误或告警信息时，Amazon Q 会自动分析集群问题，输出调查结果、根因分析报告及对应的缓解措施建议。

上述工具协同运作，为 EKS 集群的全生命周期提供全方位支持，覆盖从初始搭建配置到日常运维和问题排查的所有环节。无论是开展应用开发、管理基础设施，还是解决生产环境问题，这些 Amazon 工具都能帮助您更高效地实现 EKS 集群运维。

主题

- Amazon EKS 模型上下文协议（MCP）服务器
- 在 Amazon EKS 控制台使用 Amazon Q 开发者版

Amazon EKS 模型上下文协议（MCP）服务器

Amazon EKS MCP 服务器是一项完全托管的服务，可为开发与运维工作提供人工智能驱动的使用体验。模型上下文协议（MCP）提供了一套标准化接口，可为人工智能代理及应用程序赋予 EKS 集群与 Kubernetes 资源的实时上下文感知能力，从而在应用程序的全生命周期内（从初始部署、生产环境优化到问题排查的各个阶段），助力实现更精准的上下文感知型响应，以及人工智能驱动的工作流。

> **Note**
>
> Amazon EKS MCP 服务器在 Amazon EKS 中为预览版，且功能可能会发生变化。

概述

EKS MCP 服务器可轻松与任何兼容 MCP 协议的人工智能代码助手集成，以此优化开发工作流程，例如 Kiro、Amazon Q 开发者版 CLI，或是 Cursor、Cline 等第三方工具。在初始化阶段，EKS MCP 服务器会引导您完成集群创建流程，自动预置各项先决条件，并应用 Amazon 最佳实践方案。在开发阶段，其通过提供面向应用部署与集群管理的高阶工作流程，简化 EKS 与 Kubernetes 的操作流程。在调试与问题排查阶段，该服务器借助集成在 EKS 控制台及常用人工智能助手中的问题排查工具与知识库访问能力，加快问题解决效率。这些功能均支持通过自然语言交互调用，让您能够更直观、高效地执行复杂的 Kubernetes 操作。

完全托管的 Amazon EKS MCP 服务器部署在 Amazon 云端，无需进行本地安装与维护。其具备企业级功能特性，包括自动更新与补丁管理、基于 Amazon IAM 集成的集中式安全管控、借助 Amazon CloudTrail 实现的全面审计日志记录，以及依托 Amazon 构建的成熟可扩展性、可靠性与技术支持体系。部署于 Amazon 云端的完全托管的 EKS MCP 服务器具备以下核心优势：

- 无需安装与维护：由于 EKS MCP 服务器部署在 Amazon 云端，您无需再管理版本更新，也不必排查本地服务器问题。只需将您的人工智能助手配置为连接至到云端托管的 EKS MCP 服务器端点，即可开始操作 EKS 集群。
- 集中式访问管理：EKS MCP 服务器与 IAM 深度集成，提供了一种集中且安全的服务器访问控制方式。所有请求均通过轻量级代理，采用 Amazon SigV4 协议进行签名，可与您现有的 Amazon 凭证及 IAM 策略无缝对接。
- 增强型监控与可视化能力：借助 Amazon CloudTrail 的集成功能，可捕获通过该托管服务发起的初始化操作及全权限工具调用请求，从而生成详尽的审计跟踪记录与合规性报告。
- 始终保持最新状态：无需更新本地部署环境，即可自动获取新工具、新功能与漏洞修复补丁。该托管服务会基于用户反馈与 Amazon 最佳实践，持续进行迭代优化。

集成示例

EKS MCP 服务器提供了若干可用于执行以下操作的工具：

- 集群管理：遵循自动化最佳实践，创建、配置并管理 EKS 集群。
- Kubernetes 资源管理：部署应用程序、管理 Kubernetes 资源并检查集群状态。
- 集群问题排查：借助集成的问题排查工具与运行手册知识库，诊断各类问题
- 文档查询：基于上下文检索并获取相关的 EKS 文档内容。

探索集群

```
Show me all EKS clusters and their status
What insights does EKS have about my production-cluster?
Show me the VPC configuration for my staging cluster
```

检查 Kubernetes 资源

```
List all deployments in the production namespace
Show me pods that are not in Running state
Get the logs from the api-server pod in the last 30 minutes
```

排查问题

```
Why is my nginx-ingress-controller pod failing to start?
Search the EKS troubleshooting guide for pod networking issues
Show me events related to the failed deployment in the staging namespace
```

创建资源（若启用了“写入”模式）

```
Create a new EKS cluster named demo-cluster with VPC and Auto Mode
Deploy my containerized app from ECR to the production namespace with 3 replicas
Generate a Kubernetes deployment YAML for my Node.js app running on port 3000
```

开始使用

要开始使用，请参阅the section called “开始使用”。

Amazon EKS MCP 服务器入门

本指南将指导您完成通过人工智能代码助手设置和使用 EKS MCP 服务器的步骤。您将了解如何配置环境、连接到服务器以及通过自然语言交互开始管理 EKS 集群。

> **Note**
>
> Amazon EKS MCP 服务器在 Amazon EKS 中为预览版，且功能可能会发生变化。

先决条件

开始之前，请确保您已执行以下任务：

- 创建有权访问 Amazon EKS 的 Amazon 账户
- 使用凭证安装并配置 Amazon CLI
- 安装 Python 3.10+
- 安装 uv

设置

1. 验证 先决条件

```
# Check that your Python version is 3.10 or higher
python3 --version

# Check uv installation
uv --version

# Verify CLI configuration
aws configure list
```

2. 设置 IAM 权限

若要连接到 EKS MCP 服务器，您的 IAM 角色必须附加以下策略：**eks-mcp:InvokeMcp**（初始化及获取可用工具相关信息所需的权限），**eks-mcp:CallReadOnlyTool**（使用只读工具所需的权限），**eks-mcp:CallPrivilegedTool** [使用完全访问权限（写入）工具所需的权限]。上述这些 eks-mcp 权限，均已包含在下文所列的只读型与完全访问型 Amazon 托管策略中。

- 打开 IAM 控制台。
- 在左侧导航窗格中，根据要附加策略的身份类型，选择用户、用户组或角色，随后选中对应的具体用户、用户组或角色名称。
- 选择权限选项卡。
- 选择附加策略（若为第一次，则选择添加权限）。
- 在策略列表中，搜索并选择要附加的托管策略：
- 只读操作：AmazonEKSMCPReadOnlyAccess
- 选择附加策略（或选择下一步，再选择添加权限进行确认）。

执行此操作后，策略即被附加，相关权限会立即生效。您可以为同一个身份附加多个策略，且每个策略均可包含多种权限。要了解有关这些政策的更多信息，请参阅 Amazon Elastic Kubernetes Service 的 Amazon 托管策略。

3. 选择人工智能助手

选择以下任一兼容 MCP 的人工智能助手，或其他任意兼容 MCP 的工具：

- 安装 Amazon Q 开发者版 CLI
- 安装 Kiro
- 安装 Cursor
- 安装 Cline VS Code 扩展

步骤 1：配置人工智能助手

从以下任一选项中选择，完成人工智能代码助手配置。完成此步骤后，人工智能代码助手将通过适用于 Amazon 的 MCP 代理运行，而该代理是安全、经过身份验证访问 Amazon EKS MCP 服务器的必要组件。此过程需要添加或编辑 MCP 配置文件（例如，适用于 Amazon Q 开发者版 CLI 的配置文件路径为 `~/.aws/amazonq/mcp.json`）。该代理充当客户端侧的桥接组件，会利用本地的 Amazon 凭证执行 Amazon SigV4 身份验证，同时支持动态工具发现功能，以此实现与后端 Amazon MCP 服务器（如 EKS MCP 服务器）的交互。要了解更多信息，请参阅 MCP Proxy for Amazon 存储库。

选项 A：Amazon Q 开发者版 CLI

Q 开发者版 CLI 提供了与 EKS MCP 服务器集成度最高的体验。

1. 找到 MCP 配置文件

- macOS/Linux：`~/.aws/q/mcp.json`
- Windows：`%USERPROFILE%\.aws\q\mcp.json`

2. 添加 MCP 服务器配置

若配置文件尚不存在，则新建该文件。务必将区域 (`{region}`) 占位符替换为需要的区域。

对于 Mac/Linux：

```
{
  "mcpServers": {
```

```
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

对于 Windows：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

安全说明：可通过 `--read-only` 参数限制工具仅执行只读操作。

3. 验证配置

重启 Q 开发者版 CLI，然后查看可用工具：

```
q /tools
```

选项 B：Kiro IDE

Kiro 是一款原生支持人工智能的编码工作区，内置 MCP 支持功能。

1. 打开 Kiro 设置

- 打开 Kiro
- 前往 Kiro → 设置，搜索“MCP Config”
- 或者按下 Cmd+Shift+P,（Mac）或 Ctrl+Shift+P,（Windows/Linux），搜索“MCP Config”

2. 添加 MCP 服务器配置

- 单击“打开工作区 MCP 配置”或“打开用户 MCP 配置”，直接编辑 MCP 配置文件。

务必将区域 ({region}) 占位符替换为需要的区域。

对于 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
```

```
  }
}
```

对于 Windows：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

安全说明：可通过 `--read-only` 参数限制工具仅执行只读操作。

选项 C：Cursor IDE

Cursor 通过图形配置界面提供内置 MCP 支持功能。

1. 打开 Cursor 设置

- 打开 Cursor
- 前往设置 → Cursor 设置 → 工具和 MCP
- 或者按下 `Cmd+Shift+P`（Mac）或 `Ctrl+Shift+P`（Windows），搜索“MCP”

2. 添加 MCP 服务器配置

- 单击“新建 MCP 服务器”

若配置文件尚不存在，则新建该文件。务必将区域 ({region}) 占位符替换为需要的区域。

对于 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

对于 Windows：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
```

```
      ]
    }
  }
}
```

安全说明：可通过 `--read-only` 参数限制工具仅执行只读操作。

3. 重新启动 Cursor

关闭并重新打开 Cursor，使更改生效。

4. 在 Cursor 对话中验证

打开对话面板，尝试输入以下内容：

```
What EKS MCP tools are available?
```

此时应该会看到一份可用的 EKS 管理工具列表。

选项 D：Cline（VS Code 扩展）

Cline 是一款热门的 VS Code 扩展程序，可直接在编辑器中提供人工智能助手功能。

1. 打开 Cline设置

- 打开 Cline
- 按下 `Cmd+Shift+P`（Mac）或 `Ctrl+Shift+P`（Windows），搜索“MCP”

2. 添加 MCP 服务器配置

- 单击“添加服务器”
- 单击“打开用户配置”

若配置文件尚不存在，则新建该文件。务必将区域 (`{region}`) 占位符替换为需要的区域。

对于 Mac/Linux：

```
{
  "mcpServers": {
```

```
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

对于 Windows：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

安全说明：可通过 `--read-only` 参数限制工具仅执行只读操作。

2. 重新启动 VS Code

按下 Cmd+Shift+P/Ctrl+Shift+P，选择“开发者：重新加载窗口”

3. 验证配置

打开 Cline，询问：

```
List the available MCP tools for EKS
```

步骤 2：（可选）创建“写入”策略

您也可以选择创建一个客户管理型 IAM 策略，取得对 Amazon EKS MCP 服务器的完全访问权限。该策略授予使用 EKS MCP 服务器中所有工具的权限，包括可能涉及写入操作的特权工具以及只读工具。请注意，此策略包含高风险权限（所有以 Delete* 开头的权限，或不受限制的 IAM 资源权限），因为这些权限是 manage_eks_stacks 工具中集群资源部署与拆除操作所必需的。

```
aws iam create-policy \
 --policy-name EKSMcpWriteManagementPolicy \
 --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Effect
\": \"Allow\", \"Action\": [\"eks:DescribeCluster\", \"eks:ListClusters\",
 \"eks:DescribeNodegroup\", \"eks:ListNodegroups\", \"eks:DescribeAddon\",
 \"eks:ListAddons\", \"eks:DescribeAccessEntry\", \"eks:ListAccessEntries\",
 \"eks:DescribeInsight\", \"eks:ListInsights\", \"eks:AccessKubernetesApi\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"eks:CreateCluster
\", \"eks:DeleteCluster\", \"eks:CreateAccessEntry\", \"eks:TagResource\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"iam:GetRole\",
 \"iam:ListRolePolicies\", \"iam:ListAttachedRolePolicies\", \"iam:GetRolePolicy\",
 \"iam:GetPolicy\", \"iam:GetPolicyVersion\"], \"Resource\": \"*\"}, {\"Effect\":
 \"Allow\", \"Action\": [\"iam:TagRole\", \"iam:CreateRole\", \"iam:AttachRolePolicy
\", \"iam:PutRolePolicy\", \"iam:DetachRolePolicy\", \"iam:DeleteRole\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"iam:PassRole\"],
 \"Resource\": \"*\", \"Condition\": {\"StringEquals\": {\"iam:PassedToService
\": [\"eks.amazonaws.com\", \"ec2.amazonaws.com\"]}}}, {\"Effect\": \"Allow\",
 \"Action\": [\"ec2:CreateVpc\", \"ec2:CreateSubnet\", \"ec2:CreateRouteTable\",
 \"ec2:CreateRoute\", \"ec2:CreateInternetGateway\", \"ec2:CreateNatGateway\",
 \"ec2:CreateSecurityGroup\", \"ec2:AttachInternetGateway\", \"ec2:AssociateRouteTable
\", \"ec2:ModifyVpcAttribute\", \"ec2:ModifySubnetAttribute\", \"ec2:AllocateAddress
\", \"ec2:CreateTags\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"ec2:DeleteVpc\", \"ec2:DeleteSubnet\", \"ec2:DisassociateRouteTable\",
 \"ec2:DeleteRouteTable\", \"ec2:DeleteRoute\", \"ec2:DetachInternetGateway\",
 \"ec2:DeleteInternetGateway\", \"ec2:DeleteNatGateway\", \"ec2:ReleaseAddress\",
```

```
 \"ec2:DeleteSecurityGroup\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"ec2:DescribeVpcs\", \"ec2:DescribeSubnets\", \"ec2:DescribeRouteTables\",
 \"ec2:DescribeInternetGateways\", \"ec2:DescribeNatGateways\", \"ec2:DescribeAddresses
\", \"ec2:DescribeSecurityGroups\", \"ec2:DescribeAvailabilityZones\"], \"Resource
\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"cloudformation:CreateStack
\", \"cloudformation:UpdateStack\", \"cloudformation:DeleteStack\",
 \"cloudformation:DescribeStacks\", \"cloudformation:TagResource\"], \"Resource
\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"sts:GetCallerIdentity\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"logs:StartQuery\",
 \"logs:GetQueryResults\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\":
 [\"cloudwatch:GetMetricData\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"eks-mcp:*\"], \"Resource\": \"*\"}]}"
```

步骤 3：验证设置

测试连接

向您的 AI 助手提一个简单问题来验证连接：

```
List all EKS clusters in my {aws} account
```

此时应该会看到一份 EKS 集群列表。

步骤 4：运行第首批任务

示例 1：探索集群

```
Show me all EKS clusters and their status
What insights does EKS have about my production-cluster?
Show me the VPC configuration for my staging cluster
```

示例 2：检查 Kubernetes 资源

```
Get the details of all the kubernetes resources deployed in my EKS cluster
Show me pods that are not in Running state or pods with any restarts
Get the logs from the aws-node daemonset in the last 30 minutes
```

示例 3：问题排查

```
Why is my nginx-ingress-controller pod failing to start?
Search the EKS troubleshooting guide for pod networking issues
```

```
Show me events related to the failed deployment in the staging namespace
```

示例 4：创建资源（若启用了“写入”模式）

```
Create a new EKS cluster named demo-cluster with VPC and Auto Mode
Deploy my containerized app from ECR to the production namespace with 3 replicas
Generate a Kubernetes deployment YAML for my Node.js app running on port 3000
```

通用配置

场景 1：多个 Amazon 配置文件

如果您使用多个 Amazon 账户，请创建单独的 MCP 服务器配置。

对于 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp-prod": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2"
      ]
    },
    "eks-mcp-dev": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
```

```
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

对于 Windows：

```
{
  "mcpServers": {
    "eks-mcp-prod": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2"
      ]
    },
    "eks-mcp-dev": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
```

```
        "us-east-1"
      ]
    }
  }
}
```

场景 2：生产环境只读模式

为生产环境创建只读配置。

对于 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp-prod-readonly": {
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2",
        "--read-only"
      ],
      "autoApprove": [
        "list_k8s_resources",
        "get_pod_logs",
        "get_k8s_events"
      ]
    }
  }
}
```

对于 Windows：

```
{
  "mcpServers": {
    "eks-mcp-prod-readonly": {
      "command": "uvx",
```

```
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2",
        "--read-only"
      ],
      "autoApprove": [
        "list_k8s_resources",
        "get_pod_logs",
        "get_k8s_events"
      ]
    }
  }
}
```

场景 3：全权限开发模式

适用于具备完整写入权限的开发环境。

对于 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp-dev-full": {
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
```

```
}
```

对于 Windows：

```
{
  "mcpServers": {
    "eks-mcp-dev-full": {
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

注意事项

安全性

请勿通过允许的输入机制传递密钥或敏感信息：

- 请勿在通过 apply_yaml 执行的 YAML 文件中包含密钥或凭证。
- 请勿在提交给模型的提示词中直接传递敏感信息。
- 请勿在 CloudFormation 模板或应用程序清单中包含密钥。
- 避免使用 MCP 工具创建 Kubernetes 密钥，因为此操作需要向模型提供密钥数据。
- 避免在 Kubernetes 容器组（pod）的应用程序日志中记录敏感信息。

YAML 内容安全：

- 仅使用来源可信的 YAML 文件。

- 服务器依赖 Kubernetes API 对 YAML 内容进行验证，自身不执行验证操作。
- 在将 YAML 文件应用到集群前，需先对其进行审核。

替代 MCP 传递密钥的方案：

- 使用 Amazon Secrets Manager 或 Parameter Store 存储敏感信息。
- 为服务账户配置合规的 Kubernetes RBAC 权限。
- 对于容器组（pod）访问 Amazon 服务的场景，采用基于服务账户的 IAM 角色（IRSA）机制。

后续步骤

有关工具和配置的完整列表，请参阅 the section called “工具”。

Amazon EKS MCP 服务器工具配置参考

本指南详细列出了 mcp-proxy-for-aws 客户端工具的所有可用配置项。通过该工具，您可以从 IDE 连接到完全托管的 Amazon EKS MCP 服务器。

> **Note**
>
> Amazon EKS MCP 服务器在 Amazon EKS 中为预览版，且功能可能会发生变化。

示例

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
```

```
        "us-east-1",
        "--read-only"
      ]
    }
  }
}
```

IAM 权限

用于连接 MCP 服务器的角色，需具备 **eks-mcp:InvokeMcp** 权限，方可完成初始化操作并获取可用工具的相关信息。使用只读工具时，需配置 **eks-mcp:CallReadOnlyTool** 权限；而使用全权限（写入）工具时，则需配置 **eks-mcp:CallPrivilegedTool** 权限。

环境变量

AWS_PROFILE（可选）：要使用的 Amazon 凭证配置文件名称；可通过 `--profile` 命令行参数覆盖该配置。

- 示例：`export AWS_PROFILE=production`

AWS_REGION（可选）：用于 SigV4 签名的 Amazon 区域；若未设置，默认值为 `us-west-2`。

- 示例：`export AWS_REGION=us-east-1`

参数

SigV4 MCP 端点 URL（必填）：待连接的 MCP 端点 URL。

- 示例： https://eks-mcp.us-west-2.api.aws/mcp

--service（可选）：用于 SigV4 签名的 Amazon 服务名称；若未配置，将自动从端点主机名中识别。

- 示例：`--service eks-mcp`

--profile（可选）：要使用的 Amazon 凭证配置文件。如果未指定，则默认为 AWS_PROFILE 环境变量。

- 示例：`--profile production`

要使用的 **--region** Amazon 区域。如果未设置，则使用 AWS_REGION 环境变量，默认为 us-east-1。

- 示例：--region us-west-2

--read-only（选用）：禁用所有可能需要写入权限的工具（无需写入权限的工具均标注有 readOnlyHint=true 属性）。默认情况下，所有工具都已启用。

- 示例：--read-only

有关更多配置选项，请参阅配置参数。

工具

服务器公开了以下 MCP 工具。

只读工具

本节介绍适用于 EKS MCP 服务器的各类只读工具。请注意，所有只读类型的 Kubernetes API 操作均支持访问以下两类对象：

- 私有集群（参阅集群私有端点）
- 公共集群

search_eks_documentation 搜索 EKS 文档，获取最新信息和指引。这款工具可帮助您访问最新的 EKS 官方文档，其中涵盖了人工智能代理可能尚未收录的新功能与近期功能优化内容。

参数：

- query（必填）：与 EKS 文档、功能特性或最佳实践相关的具体问题或搜索语句。
- limit（可选）：返回文档结果的最大数量（取值范围 1-10）。默认值：5。

search_eks_troubleshooting_guide 根据检索语句在《EKS 问题排查指南》中搜索相关排障信息，助力定位常见问题并提供分步式解决方案。

参数：

- query（必填）：与 EKS 问题排查相关的具体问题或故障描述。

describe_eks_resource 检索有关特定 EKS 集群资源的详细信息，包括配置、状态和元数据。

参数：

- cluster_name（必填）：EKS 集群的名称（针对集群级资源为必填项）。
- resource_type（必填）：需要描述的 EKS 资源类型。有效值：
- accessentry（需指定 cluster_name，且需将 resource_name 填写为主体 ARN）
- addon（需指定 cluster_name，且需将 resource_name 填写为附加组件名称）
- cluster（需指定 cluster_name）、nodegroup（需指定 cluster_name，且需将 resource_name 填写为节点组名称）。
- resource_name（可选）：需要描述的特定资源名称（对于大多数资源类型为必填项）。

list_eks_resources 列出指定类型的 EKS 资源，返回所有可访问的该类型资源的摘要信息。

参数：

- resource_type（必填）：需要列出的 EKS 资源类型。有效值：
- accessentry（需指定 cluster_name）
- addon（需指定 cluster_name）
- cluster（无需额外参数）
- nodegroup（需指定 cluster_name）。
- cluster_name（可选）：EKS 集群的名称（针对集群级资源为必填项）。

get_eks_insights 检索 EKS 集群的洞察信息和优化建议。该工具基于 Amazon 最佳实践与集群分析结果，提供关于安全防护、性能调优及成本优化的可执行洞察方案。

参数：

- cluster_name（必填）：EKS 集群的名称。
- category（可选）：用于筛选洞察信息的可选分类（例如：MISCONFIGURATION 或 UPGRADE_READINESS）。
- insight_id（可选）：用于获取特定洞察详细信息的可选 ID。
- next_token（可选）：用于结果分页查询、获取下一组数据的可选令牌。

get_eks_vpc_config 获取 EKS 集群的 VPC 配置信息，包括子网、路由表及网络连通性相关内容。

参数：

- cluster_name（必填）：需获取 VPC 配置的 EKS 集群名称。
- vpc_id（可选）：待查询的特定 VPC 的 ID（可选参数，若未指定则使用该集群对应的 VPC）。

get_k8s_events 获取与特定资源相关的 Kubernetes 事件，用于问题排查与监控。

参数：

- cluster_name（必填）：目标资源所在的 EKS 集群名称。
- kind（必填）：关联对象的资源类型（例如：Pod、Deployment、Service）。必须与资源类型名称完全匹配。
- name（必填）：需查询事件的关联对象名称。
- namespace（可选）：关联对象所属的命名空间。针对命名空间级资源（如 Pod、Deployment）为必填项；针对集群级资源（如 Node、PersistentVolume）无需配置。

get_pod_logs 获取 EKS 集群中容器组（pod）的日志，并支持日志过滤功能。

参数：

- cluster_name（必填）：容器组（pod）运行所在的 EKS 集群名称。
- namespace（必填）：容器组（pod）所在的 Kubernetes 命名空间。
- pod_name（必填）：需获取日志的容器组（pod）名称。
- container_name（可选）：需获取日志的特定容器名称。仅当容器组（pod）包含多个容器时为必填项。
- limit_bytes（可选）：返回日志的最大字节数。默认值：10 KB（10240 字节）。
- previous（可选）：返回容器此前终止状态下的日志（默认值为 false）。该参数适用于获取重启中容器组（pod）的日志。
- since_seconds（可选）：仅返回指定秒数内的最新日志。适用于获取近期日志，无需调取完整日志历史。
- tail_lines（可选）：返回日志末尾的指定行数。默认值：100。

list_api_versions 列出指定 Kubernetes 集群中所有可用的 API 版本。

参数：

- cluster_name（必填）：EKS 集群的名称。

list_k8s_resources 列出 EKS 集群中特定类型的 Kubernetes 资源。

参数：

- cluster_name（必填）：标资源所在的 EKS 集群名称。
- kind（必填）：要列出的 Kubernetes 资源种类（例如：Pod、Service、Deployment）。可使用 `list_api_versions` 工具查询可用的资源类型。
- api_version（必填）：目标 Kubernetes 资源对应的 API 版本（例如：v1、apps/v1、networking.k8s.io/v1）。可使用 `list_api_versions` 工具查询可用的 API 版本。
- field_selector（可选）：用于筛选资源的字段选择器（例如：metadata.name=my-pod,status.phase=Running）。语法与 kubectl 命令的 --field-selector 参数一致。
- label_selector（可选）：用于筛选资源的标签选择器（例如：app=nginx,tier=frontend）。语法与 kubectl 命令的 `--selector` 参数一致。
- namespace（可选）：需列出的 Kubernetes 资源所属的命名空间。若未指定，将列出所有命名空间下的对应资源（仅适用于命名空间级资源）。

read_k8s_resource 获取 EKS 集群中特定 Kubernetes 资源的详细信息。

参数：

- api_version（必填）：目标 Kubernetes 资源对应的 API 版本（例如：v1、apps/v1、networking.k8s.io/v1）。
- cluster_name（必填）：目标资源所在的 EKS 集群名称。
- kind（必填）：目标 Kubernetes 资源的类型（例如：Pod、Service、Deployment）。
- name（必填）：需读取的 Kubernetes 资源名称。
- namespace（可选）：目标 Kubernetes 资源所属的命名空间。针对命名空间级资源为必填项；针对集群级资源（如 Node、PersistentVolume）无需配置。

generate_app_manifest 为容器化应用程序生成标准化的 Kubernetes 部署列表与服务清单。

参数：

- app_name（必填）：应用程序的名称。该名称将用于部署列表、服务列表的命名，同时会作为资源标签使用。

- image_uri（必填）：带标签的完整 ECR 映像 URI（例如：123456789012.dkr.ecr.region.amazonaws.com/repo:tag）。必须包含完整的映像存储库路径与标签。
- load_balancer_scheme（可选）：Amazon 负载均衡器方案。有效值：
- internal（仅限私有 VPC 网络）
- internet-facing（支持公网访问）。
- 默认值：internal。
- cpu（可选）：每个容器的 CPU 资源请求值（例如：100m 代表 0.1 个 CPU 核心，500m 代表 0.5 个 CPU 核心）。默认值：100m。
- memory（可选）：每个容器的内存资源请求值（例如：128Mi 代表 128 MiB，1Gi 代表 1 GiB）。默认值：128Mi
- namespace（可选）：应用程序将要部署到的 Kubernetes 命名空间。默认值：default。
- port（可选）：应用程序要监听的容器端口。默认值：80
- replicas（可选）：应用程序要部署的副本数量。默认值：2

get_cloudwatch_logs 基于输入参数筛选查询 CloudWatch 日志，同时支持 EKS 集群可观测性场景下的标准日志组。

参数：

- cluster_name（必填）：目标资源所在的 EKS 集群名称。该参数将用于构造 CloudWatch 日志组名称。
- resource_type（必填）：需检索日志对应的资源类型。有效值：pod、node、container、cluster。该参数将决定日志的筛选逻辑。
- log_type（必填）：需查询的日志类型。有效值：
- application：容器/应用程序日志
- host：节点级系统日志
- performance：性能指标日志
- control-plane：EKS 控制面板日志
- your-log-group-name：直接传入自定义的 CloudWatch 日志组名称。
- resource_name（可选）：需在日志消息中检索的资源名称（例如：容器组（pod）名称、节点名称、容器名称）。该参数将用于筛选特定资源的日志。

- minutes（可选）：日志回溯的时间范围（单位：分钟）。默认值：15。若传入 start_time 参数，此参数将失效。排查近期问题可设置较小值，进行历史分析可设置较大值。
- start_time（可选）：查询的起始时间（ISO 格式，例如：2023-01-01T00:00:00Z）。若传入此参数，将覆盖 minutes 参数的配置。
- end_time（可选）：查询的结束时间（ISO 格式，例如：2023-01-01T01:00:00Z）。若未指定，默认值为当前时间。
- fields（可选）：需包含在查询结果中的自定义字段（默认值：@timestamp、@message）。需遵循 CloudWatch Logs Insights 的字段语法。
- filter_pattern（可选）：需额外应用的 CloudWatch 日志筛选规则。遵循 CloudWatch Logs Insights 语法（例如：ERROR、field=value）。
- limit（可选）：返回日志条目的最大数量。设置较小值（10-50）可提升查询速度，设置较大值（100-1000）可获取更全面的结果。较高的值可能会影响性能。

get_cloudwatch_metrics 获取 CloudWatch 指标与数据点，用于 EKS 集群监控及性能分析。支持容器洞察指标、自定义指标，同时可配置时间范围与维度。

参数：

- cluster_name（必填）：需获取指标的 EKS 集群名称。
- dimensions（必填）：CloudWatch 指标查询所用的维度，格式为 JSON 字符串。必须包含与资源类型和指标匹配的对应维度（例如：{"ClusterName": "my-cluster", "PodName": "my-pod", "Namespace": "default"}）。
- metric_name（必填）：需获取的指标名称。常见示例：
- `cpu_usage_total`：CPU 总使用率
- `memory_rss`：常驻内存集（RSS）使用率
- `network_rx_bytes`：网络接收字节数
- `network_tx_bytes`：网络发送字节数
- namespace（必填）：指标存储的 CloudWatch 命名空间。常见取值：
- `"ContainerInsights"`：适用于容器指标
- `"AWS/EC2"`：适用于 EC2 实例指标
- `"AWS/EKS"`：适用于 EKS 控制面板指标
- minutes（可选）：指标数据的回溯时间范围（单位：分钟）。默认值：15。若传入 start_time 参数，此参数将失效。

- start_time（可选）：查询的起始时间（ISO 格式，例如：2023-01-01T00:00:00Z）。若传入此参数，将覆盖 minutes 参数的配置。
- end_time（可选）：查询的结束时间（ISO 格式，例如：2023-01-01T01:00:00Z）。若未指定，默认值为当前时间。
- limit（可选）：返回数据点的最大数量。设置较高数值（100-1000）可获取更精细的数据，但可能影响查询性能。默认值：50。
- period（可选）：指标数据点的统计周期（单位：秒）。默认值：60（1 分钟）。设置较低数值（1-60）可提升数据分辨率，但数据可用性可能会降低。
- stat（可选）：指标聚合计算所用的统计方式。默认值：Average。有效值：
- `Average`：统计周期内的平均值
- `Sum:`：统计周期内的累计值
- `Maximum`：统计周期内的最大值
- `Minimum`：统计周期内的最小值
- `SampleCount`：统计周期内的样本数量。

get_eks_metrics_guidance 获取 EKS 集群内特定资源类型对应的 CloudWatch 指标配置指引。该工具可辅助人工智能代理，在使用 get_cloudwatch_metrics 工具时确定应配置的正确维度。

参数：

- resource_type（必填）：需获取指标指引的资源类型（集群、节点、容器组（pod）、命名空间、服务）。

get_policies_for_role 获取附加至指定 IAM 角色的所有策略，包括代入角色策略、托管策略及内联策略。

参数：

- role_name（必填）：需查询策略的 IAM 角色名称。该角色必须存在于您的 Amazon 账户中。

全权限（写入）工具

本节介绍适用于 EKS MCP 服务器的各类只读工具。请注意，截至当前，所有 Kubernetes API 写入操作仅支持以下集群类型：

- 公共集群（endpointPublicAccess=true）

manage_k8s_resource 通过写入操作（create、update、patch 或 delete）管理单个 Kubernetes 资源。

参数：

- operation（必填）：需对资源执行的操作类型。有效值：
- `create:`：创建新资源
- `replace`：替换现有资源
- `patch`：更新现有资源的指定字段
- `delete`：删除现有资源
- 注意：如需读取资源信息，请使用 read_k8s_resource 工具；如需列出多个资源，请使用 list_k8s_resources 工具。
- cluster_name（必填）：目标资源所在或待创建的 EKS 集群名称。
- kind（必填）：目标 Kubernetes 资源的类型（例如：Pod、Service、Deployment）。
- api_version（必填）：目标 Kubernetes 资源对应的 API 版本（例如：v1、apps/v1、networking.k8s.io/v1）。
- body（可选）：资源定义内容，格式为 JSON 字符串。执行 create、replace、patch 操作时为必填项。其中，执行 create 和 replace 操作需传入完整的资源定义；执行 patch 操作仅需传入待更新的字段。
- name（可选）：目标 Kubernetes 资源的名称。除 create 操作（资源名称可在 body 中定义）外，其余操作均为必填项。
- namespace（可选）：目标 Kubernetes 资源所属的命名空间。针对命名空间级资源为必填项；针对集群级资源（如 Node、PersistentVolume）无需配置。

apply_yaml 将 Kubernetes YAML 清单配置应用到 EKS 集群。

参数：

- cluster_name（必填）：待创建或更新资源的目标 EKS 集群名称。
- namespace（必填）：应用资源的 Kubernetes 命名空间。该命名空间将适用于未在列表中指定命名空间的命名空间级资源。
- yaml_content（必填）：需应用到集群的 YAML 配置内容。可包含多个以 --- 分隔的配置文档。
- force（可选）：若资源已存在，是否执行更新操作（功能与 kubectl apply 命令类似）。设置为 false 时，仅会创建新资源。

manage_eks_stacks 通过生成模板、部署、描述及删除等操作，管理 EKS 相关的 CloudFormation 堆栈，涵盖 EKS 集群及其底层基础设施。集群创建操作通常需要 15-20 分钟完成。执行部署和删除操作时，目标堆栈必须由本工具创建（即带有 CreatedBy=EksMcpServer 标签）。

参数：

- cluster_name（必填）：EKS 集群名称（适用于生成、部署、描述及删除操作）。该名称将用于派生 CloudFormation 堆栈名称，同时会嵌入到集群资源。
- operation（必填）：需执行的操作类型。有效值：
- `generate`：创建 CloudFormation 模板
- `deploy`：部署 CloudFormation 堆栈（需 `template_content` 参数）
- `describe`：描述/读取 CloudFormation 堆栈（只读操作）
- `delete`：删除 CloudFormation 堆栈
- template_content（可选）：CloudFormation 模板内容（适用于部署操作）。需传入完整的 YAML 或 JSON 格式模板内容。支持单资源模板，也支持以 --- 分隔的多文档 YAML 内容。

add_inline_policy 为 IAM 角色添加新的内联策略。

参数：

- permissions（必填）：策略中包含的权限内容，格式为表示 IAM 策略语句的 JSON 字符串。可以是单个 JSON 字符串，也可以是 JSON 字符串数组。
- policy_name（必填）：待创建的内联策略名称。该名称在对应角色内必须具唯一性。
- role_name（必填）：需添加策略的 IAM 角色名称。该角色必须已存在。

在 Amazon EKS 控制台使用 Amazon Q 开发者版

Amazon Elastic Kubernetes Service（EKS）与 Amazon Q 集成，可直接在 Amazon 管理控制台中提供人工智能支持的问题排查。借助 Amazon Q 的人工智能辅助，此集成可帮助您快速调查和解决集群、控制面板、节点及工作负载问题。

工作原理

Amazon EKS 控制台会在整个控制台的错误或问题旁，根据上下文显示使用 Amazon Q 检查按钮。当您点击此按钮时，Amazon Q 会自动分析问题，并在控制台右侧打开聊天面板，其中包含调查结果、根本原因分析和建议的缓解措施。

集成将在 Amazon EKS 控制台的以下位置显示：

- 集群运行状况：在可观测性控制面板的“集群运行状况”选项卡中，调查集群运行状况问题和状态消息
- 控制面板：在可观测性控制面板的“控制面板监控”中，排查控制面板组件错误和警告
- 升级见解：在可观测性控制面板的“升级见解”中，分析潜在的升级阻碍和兼容性问题
- 节点运行状况：在可观测性控制面板的“节点运行状况问题”中，调查影响集群容量的节点级问题
- 工作负载：分析容器组（pod）上指示故障或问题的 Kubernetes 事件

使用 Amazon Q 进行问题排查

使用 Amazon Q 调查问题

1. 打开 Amazon EKS 控制台。
2. 选择要调查的集群名称。
3. 当您遇到错误消息或问题指示时，请查找使用 Amazon Q 检查按钮。该按钮会根据上下文，显示在问题旁边或错误状态详细信息视图中。
4. 选择使用 Amazon Q 检查。
5. Amazon Q 会自动调查问题，并在控制台右侧的聊天面板中显示分析结果。
6. 查看调查结果，包括根本原因分析和建议的缓解措施。
7. 您可以向 Amazon Q 询问有关该问题的后续问题，以继续对话。

注意：只有在出现错误、警告或需要调查的问题时，才会显示 Amazon Q 集成。如果资源运行状况良好，则不会显示。

注意事项

将 Amazon Q 与 Amazon ECS 结合使用时，请考虑以下事项：

- 只读操作：Amazon Q 集成仅对集群资源执行读取操作。它不会对集群配置或工作负载执行任何变异或写入操作。
- 跨区域处理：Amazon Q 可以跨 Amazon 区域处理数据，以提供人工智能支持的分析。有关跨区域处理的更多信息，有关跨区域处理的更多信息，请参阅《Amazon Q 开发者版用户指南》中的跨区域处理。
- 仅限 Amazon 管理控制台：此集成仅可通过 Amazon 管理控制台获得。无法通过 Amazon CLI、Amazon API 或基础设施即代码工具提供。

了解更多

有关使用 Amazon Q 的更多信息，请参阅《Amazon Q 开发者版用户指南》中的与 Amazon Q 聊天。

Amazon EKS 上的版本控制

本节旨在帮助您了解 Kubernetes 版本控制如何在 Amazon Elastic Kubernetes Service（EKS）中运行。您将详细了解支持的版本、我们的弃用和支持政策、升级流程以及管理集群版本的关键注意事项。

主题

- 了解 EKS 上的 Kubernetes 版本生命周期
- 查看每个 Kubernetes 版本的 Amazon EKS 平台版本

了解 EKS 上的 Kubernetes 版本生命周期

Kubernetes 发展迅速，会不断提供新功能、设计更新和错误修复。社区平均每四个月发布一次新的 Kubernetes 次要版本（例如 `1.33`）。Amazon EKS 会遵循次要版本的上游发布和弃用周期。随着新的 Kubernetes 版本在 Amazon EKS 中推出，我们建议您主动将集群更新为使用最新的可用版本。

在发布后的前 14 个月内，次要版本会在 Amazon EKS 中获得标准支持。一个版本超过标准支持终止日期后，将在接下来 12 个月自动进入扩展支持。延期支持允许您在特定 Kubernetes 版本上停留更长时间，但需要按小时为集群支付额外费用。如果您在延期支持期限结束之前未更新集群，则您的集群将自动升级到当前支持的最旧延期版本。

扩展支持会默认启用。要禁用，请参阅禁用 EKS 延期支持。

建议使用 Amazon EKS 支持的最新可用 Kubernetes 版本创建集群。如果应用程序需要特定版本的 Kubernetes，则可以选择较旧的版本。您可以在标准或延期支持中提供的任何版本上创建新的 Amazon EKS 集群。

标准支持上的可用版本

Amazon EKS 标准支持目前提供以下 Kubernetes 版本：

- `1.34`
- `1.33`
- `1.32`

有关标准支持中每个版本需要注意的重要更改，请参阅 Kubernetes 版本标准支持。

延期支持的可用版本

Amazon EKS 延期支持目前提供以下 Kubernetes 版本：

- 1.31
- 1.30
- 1.29

有关延期支持中每个版本需要注意的重要更改，请参阅 Kubernetes 版本延期支持。

Amazon EKS Kubernetes 发布日历

下表显示了每个 Kubernetes 版本需要考虑的重要发布和支持日期。延期支持从该版本的标准支持终止之日起开始计费（UTC+0 时区）。下表中的日期使用 UTC+0 时区。

> **Note**
>
> 只有月份和年份的发布日期均为大概日期，这些日期会在确切日期明确后进行更新。

要接收此特定文档页面的所有源文件更改通知，您可以通过 RSS 阅读器订阅以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/clusters/kubernetes-versions.adoc.atom
```

Kubernetes 版本	上游版本	Amazon EKS 版本	标准支持结束日期	延期支持结束日期
1.34	2025 年 8 月 27 日	2025 年 10 月 2 日	2026 年 12 月 2 日	2027 年 12 月 2 日
1.33	2025 年 4 月 23 日	2025 年 5 月 29 日	2026 年 7 月 29 日	2027 年 7 月 29 日
1.32	2024 年 12 月 11 日	2025 年 1 月 23 日	2026 年 3 月 23 日	2027 年 3 月 23 日

Kubernetes 版本	上游版本	Amazon EKS 版本	标准支持结束日期	延期支持结束日期
1.31	2024 年 8 月 13 日	2024 年 9 月 26 日	2025 年 11 月 26 日	2026 年 11 月 26 日
1.30	2024 年 4 月 17 日	2024 年 5 月 23 日	2025 年 7 月 23 日	2026 年 7 月 23 日
1.29	2023 年 12 月 13 日	2024 年 1 月 23 日	2025 年 3 月 23 日	2026 年 3 月 23 日

使用 Amazon CLI 获取版本信息

您可以使用 Amazon CLI 获取有关 EKS 上可用 Kubernetes 版本的信息，例如标准支持的终止日期。

使用 Amazon CLI 检索有关 EKS 上可用 Kubernetes 版本的信息

1. 打开终端。
2. 请确保您已经安装并配置好 Amazon CLI。有关更多信息，请参阅安装或更新到最新版本的 CLI。
3. 运行如下命令：

```
aws eks describe-cluster-versions
```

4. 该命令将返回一个 JSON 输出，其中包含有关可用集群版本的详细信息。输出示例如下：

```
{
    "clusterVersions": [
        {
            "clusterVersion": "1.31",
            "clusterType": "eks",
            "defaultPlatformVersion": "eks.21",
            "defaultVersion": true,
            "releaseDate": "2024-09-25T17:00:00-07:00",
            "endOfStandardSupportDate": "2025-11-25T16:00:00-08:00",
            "endOfExtendedSupportDate": "2026-11-25T16:00:00-08:00",
            "status": "STANDARD_SUPPORT",
            "kubernetesPatchVersion": "1.31.3"
        }
```

```
    ]
}
```

此输出提供了有关每个集群版本的下列信息：

- `clusterVersion`：EKS 集群的 Kubernetes 版本
- `clusterType`：该集群的类型（例如，“eks”）
- `defaultPlatformVersion`：默认 EKS 平台版本
- `defaultVersion`：此版本是否是默认版本
- `releaseDate`：此版本的发布日期
- `endOfStandardSupportDate`：标准支持终止日期
- `endOfExtendedSupportDate`：扩展支持终止日期
- `status`：此版本的当前支持状态，例如 `STANDARD_SUPPORT` 或 `EXTENDED_SUPPORT`
- `kubernetesPatchVersion`：特定的 Kubernetes 补丁版本

Amazon EKS 版本常见问题

标准支持提供多少个 Kubernetes 版本？

> 为了与 Kubernetes 社区对 Kubernetes 版本的支持保持一致，Amazon EKS 致力于在任何特定时间为至少三个 Kubernetes 版本提供支持。我们会提前至少 60 天公告特定 Kubernetes 次要版本的标准支持结束日期。考虑到 Kubernetes 新版本的 Amazon EKS 鉴定和发布流程，Amazon EKS 上对某个 Kubernetes 版本的标准支持的结束日期会是 Kubernetes 项目停止支持上游版本之后。

Kubernetes 可以获得多长时间的 Amazon EKS 标准支持？

> 首次在 Amazon EKS 上发布后，一个 Kubernetes 版本可获得 14 个月的标准支持。即使上游 Kubernetes 不再支持 Amazon EKS 上可用的版本，也是如此。我们向后移植适用于 Amazon EKS 上支持的 Kubernetes 版本的安全补丁。

当对 Amazon EKS 上的 Kubernetes 版本的标准支持结束时，我是否会收到通知？

> 可以。如果账户中有任何集群正在运行即将结束支持的版本，Amazon EKS 会在 Kubernetes 版本在 Amazon EKS 上发布大约 12 个月后通过 Amazon Health Dashboard 发出通知。该通知包括支持结束日期。为从通知发出之日起至少 60 天。

Amazon EKS 支持哪些 Kubernetes 功能？

Amazon EKS 支持 Kubernetes API 的所有正式发布（GA）功能。默认情况下集群中不启用新的测试版 API。但在默认情况下，先前存在的测试版 API 和现有测试版 API 的新版本将继续默认启用。不支持 Alpha 功能。

Amazon EKS 托管节点组是否会随集群控制面板版本一起自动更新？

否。托管节点组在您的账户中创建 Amazon EC2 实例。当您或 Amazon EKS 更新控制面板时，这些实例不会自动升级。有关更多信息，请参阅 the section called “更新”。我们建议您的控制层面和节点上保持相同的 Kubernetes 版本。

自行管理的节点组是否会随集群控制面板版本一起自动更新？

否。自行管理的节点组包括您的账户中的 Amazon EC2 实例。当您或 Amazon EKS 代表您更新控制面板版本时，这些实例不会自动升级。对于自行管理的节点组，控制台中没有任何其需要更新的标示。您可以在集群的 Overview（概览）选项卡上的 Nodes（节点）列表中选择节点查看其上安装的 `kubelet` 版本，确定哪些节点需要更新。您必须手动更新节点。有关更多信息，请参阅 the section called “更新方法”。

Kubernetes 项目最多测试三个次要版本的控制面板和节点之间的兼容性。例如，当 `1.30` 节点由 `1.33` 控制面板编排时，节点会继续运行。但是，不建议运行具有版本持续落后控制面板三个次要版本的节点的集群。有关更多信息，请参阅 Kubernetes 文档中的 Kubernetes 版本和版本倾斜支持政策。我们建议您的控制层面和节点上保持相同的 Kubernetes 版本。

Fargate 上运行的容器组（pod）是否会随着集群控制面板版本自动升级而自动升级？

不会。我们强烈建议将 Fargate 容器组（pod）作为复制控制器（如 Kubernetes 部署）的一部分运行。然后依次重启所有的 Fargate 容器组（pod）。新版本的 Fargate 容器组（pod）部署有一个 `kubelet` 版本，该版本与更新的集群控制面板版本相同。有关更多信息，请参阅 Kubernetes 文档中的部署。

> **Important**
>
> 如果您更新控制面板，则必须自行更新 Fargate 节点。要更新 Fargate 节点，请删除该节点表示的 Fargate 容器组（pod），然后重新部署容器组（pod）。部署新的容器组（pod）时使用 `kubelet` 版本，该版本与您的集群版本相同。

：哪些 Kubernetes 版本支持混合节点功能？

Amazon EKS 混合节点功能支持的 Kubernetes 版本与使用其他节点计算类型的 Amazon EKS 集群相同，包括标准和扩展 Kubernetes 版本支持。升级控制面板版本时，混合节点不会自动升级，您需要自行升级混合节点。有关更多信息，请参阅 the section called “升级混合节点”。

Amazon EKS 扩展支持常见问题

标准支持和延期支持术语对我来说是全新的。这些术语是什么意思？

对 Amazon EKS 中某个 Kubernetes 版本的标准支持从该 Kubernetes 版本在 Amazon EKS 上发布时开始，并将在发布日期 14 个月后结束。对某个 Kubernetes 版本的延期支持将在标准支持结束后立即开始，并将在接下来的 12 个月后结束。例如，Amazon EKS 版本 `1.23` 的标准支持将于 2023 年 10 月 11 日终止。对版本 `1.23` 的扩展支持于 2023 年 10 月 12 日开始，并将于 2024 年 10 月 11 日终止。

我需要做些什么才能获得对 Amazon EKS 集群的延期支持？

您需要通过将集群升级策略更改为 EXTENDED 来启用对集群的延期支持（请参阅 EKS 延期支持）。默认情况下，除非另有说明，否则对于所有新集群和现有集群，升级策略均设置为 EXTENDED。要查看您的集群的升级策略，请参阅集群升级策略。标准支持从 Kubernetes 版本在 Amazon EKS 上发布时开始，并将在发布日期 14 个月后结束。对某个 Kubernetes 版本的延期支持将在标准支持结束后立即开始，并将在接下来的 12 个月后结束。

哪些 Kubernetes 版本可以获得延期支持？

在任何版本的标准支持结束后，您最多可以在该版本上运行集群 12 个月。这意味着 Amazon EKS 将为每个版本提供 26 个月的支持（14 个月的标准支持加上 12 个月的延期支持）。

如果我不想使用延期支持怎么办？

如果您不想自动注册延期支持，则可以将集群升级到 Amazon EKS 标准支持的 Kubernetes 版本。要禁用延期支持，请参阅禁用 EKS 延期支持。注意：如果您禁用延期支持，则会在标准支持终止时自动将集群升级到下一个版本。

12 个月的延期支持结束后会发生什么？

在已完成 26 个月生命周期（14 个月的标准支持期加上 12 个月的延期支持期）的 Kubernetes 版本上运行的集群，会自动升级到下一个版本。自动升级仅包含 Kubernetes 控制面板。启用了 EKS 自动模式的节点可能会自动更新。自主管理型节点和 EKS 托管式节点组将继续使用之前的版本。

在延期支持结束之日，您将无法再使用不受支持的版本创建新的 Amazon EKS 集群。Amazon EKS 会在支持结束日期后通过逐步部署流程，自动将现有控制面板更新为最早的受支持版本。控

制面板自动更新后，确保手动更新集群附加组件和 Amazon EC2 节点。有关更多信息，请参阅 the section called “更新 Kubernetes 版本”。

在延期支持结束日期后，自动更新控制面板的确切时间是何时？

Amazon EKS 无法给出具体的时间范围。自动更新可以在延期支持结束日期后的任何时间进行。在更新之前，您不会收到任何通知。我们建议您主动更新您的控制面板，而不要依赖 Amazon EKS 自动更新流程。有关更多信息，请参阅 the section called “更新 Kubernetes 版本”。

我能否无限期地将控制面板留在某个 Kubernetes 版本上？

不能。Amazon 的云安全性具有最高优先级。一段时间后（通常为 1 年），Kubernetes 社区会停止发布通用漏洞披露（CVE）补丁程序，也不鼓励提交不受支持版本的 CVE。也就是说，可能不会报告旧版本 Kubernetes 的特定漏洞。这会在出现漏洞的情况下，集群被暴露且不会进行通知，也没有修复选项。鉴于此，Amazon EKS 不允许控制面板停留在延期支持已经结束的版本上。

获得延期支持是否需要额外费用？

需要，在延期支持下运行的 Amazon EKS 集群需要支付额外费用。有关定价详情，请参阅 Amazon 博客上的 对 Kubernetes 版本定价的 Amazon EKS 延期支持或我们的定价页面。

延期支持中包含哪些内容？

延期支持中的 Amazon EKS 集群会一直收到 Kubernetes 控制面板的安全补丁。此外，Amazon EKS 将发布 Amazon VPC CNI 的补丁 `kube-proxy` 和延期支持版本的 CoreDNS 附加组件。Amazon EKS 还将针对 Amazon 发布的适用于 Amazon Linux、Bottlerocket 和 Windows 的 Amazon EKS 优化版 AMI 以及这些版本的 Amazon EKS Fargate 节点发布补丁。延期支持中的所有集群将继续从 Amazon 中获得技术支持。

在延期支持中，非 Kubernetes 组件的补丁是否有任何限制？

虽然延期支持涵盖了来自 Amazon 的所有 Kubernetes 特定组件，但始终仅为 Amazon 发布的适用于 Amazon Linux、Bottlerocket 和 Windows 的 Amazon EKS 优化版 AMI 提供支持。这意味着，在使用延期支持时，您的 Amazon EKS 优化版 AMI 上可能会有更新的组件（例如操作系统或内核）。例如，一旦 Amazon Linux 2 在 2025 年结束其生命周期后，Amazon EKS 优化版 Amazon Linux AMI 将使用更新的 Amazon Linux 操作系统构建。Amazon EKS 将宣布并记录重要的支持生命周期差异，例如每个 Kubernetes 版本的差异。

我是否能使用扩展支持的版本创建新集群？

可以。

查看 Kubernetes 版本标准支持的发布说明

本主题介绍有关标准支持中每个 Kubernetes 版本需要注意的重要更改。升级时，请仔细检查集群新旧版本之间发生的变化。

Kubernetes 1.34

Kubernetes `1.34` 现已在 Amazon EKS 中可用。有关 Kubernetes `1.34` 的更多信息，请参阅官方发布公告。

Important

- Containerd 在版本 1.34 中更新到 2.1 以供发布。
  - 如果您在升级后遇到任何问题，请查看 containerd 2.1 发行说明。
- Amazon 没有发布适用于 Kubernetes 1.34 的 EKS 优化版 Amazon Linux 2 AMI。
  - Amazon 鼓励您迁移到 Amazon Linux 2023。了解如何the section called “升级至 AL2023”。
  - 有关更多信息，请参阅 the section called “Amazon Linux 2 AMI 弃用”。
- AppArmor 在 Kubernetes 1.34 中已被弃用。
  - 建议迁移到替代容器安全解决方案，例如 seccomp 或 Pod 安全标准。
- VolumeAttributesClass（VAC）在 Kubernetes 1.34 中升级到 GA，并从测试版 API (`storage.k8s.io/v1beta1`) 迁移到稳定版 API (`storage.k8s.io/v1`)。
  - 如果您将 EBS CSI 驱动程序与 Amazon 托管的 Sidecar 容器一起使用（来自 ECR 映像浏览馆上的 CSI 组件），则卷修改将继续在 EKS 1.31-1.33 集群上无缝进行。Amazon 将在 EKS 1.33 标准支持结束（2026 年 7 月 29 日）之前修补 Sidecar 以支持测试版 VAC API。
  - 如果您自行管理 CSI Sidecar 容器，则可能需要在 1.34 之前的集群上固定到较旧的 Sidecar 版本以维持 VAC 功能。
  - 要使用 GA VolumeAttributesClass 功能（例如修改回滚），请升级到 EKS 1.34 或更高版本。

- 动态资源分配（DRA）核心 API（GA）：动态资源分配已升级到稳定版，通过标准化的分配接口实现了对 GPU 等专业硬件的高效管理，从而简化了硬件加速器的资源管理并提高了专业资源的利用率。

- 适用于 Kubelet 的 ServiceAccount 投射式令牌（测试版）：此增强功能使用短期凭证（而不是长期密码）提取容器映像，以提高安全性，从而降低凭证泄露的风险并增强集群的整体安全状况。
- 容器组（pod）级别资源请求和限制（测试版）：此功能允许多容器容器组（pod）共享资源池，从而简化资源管理，并为具有多个容器的复杂应用实现更高效的资源分配和利用。
- 可变 CSI 节点可分配计数（测试版）：在 EKS 1.34 中，该 `MutableCSINodeAllocatableCount` 功能开关默认已启用，使 CSInode 最大可附加卷计数属性可变，并引入了一种在 CSI 驱动程序级别根据用户配置动态更新该属性的机制。这些更新可以按定期间隔触发，也可以在检测到故障时触发，从而通过解决节点上报告的附加容量与实际附加容量之间的不一致问题，来增强有状态容器组（pod）调度的可靠性。
  - 有关更多信息，请参阅 Kubernetes 博客上的 Kubernetes v1.34：可变的 CSI 节点可分配数。
- 弃用通知 – cgroup 驱动程序配置：已弃用手动 cgroup 驱动程序配置，转而使用自动检测。
  - 客户影响：如果您目前在 kubelet 配置中手动设置了 `--cgroup-driver` 标志，则应准备好移除此配置。
  - 所需操作：计划在未来的 Kubernetes 版本中移除该功能之前，更新节点引导脚本和自定义 AMI 配置以移除手动 cgroup 驱动程序设置。
  - 有关更多信息，请参阅 cgroup 驱动程序文档。

有关完整的 Kubernetes `1.34` 更改日志，请参阅 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.34.md

Kubernetes 1.33

Kubernetes `1.33` 现已在 Amazon EKS 中可用。有关 Kubernetes `1.33` 的更多信息，请参阅官方发布公告。

> **Important**
>
> - 动态资源分配测试版 Kubernetes API 已启用。
>   - 此测试版 API 改善了调度和监控需要 GPU 等资源的工作负载的体验。
>   - 此测试版 API 由 Kubernetes 社区定义，以后的 Kubernetes 版本中可能会有变化。
>   - 仔细查看 Kubernetes 文档中的功能阶段，了解使用测试版 API 的含义。
> - Amazon 没有发布适用于 Kubernetes 1.33 的 EKS 优化版 Amazon Linux 2 AMI。
>   - Amazon 鼓励您迁移到 Amazon Linux 2023。了解如何the section called “升级至 AL2023”。

- 有关更多信息，请参阅 the section called “Amazon Linux 2 AMI 弃用”。

- 就地容器组（pod）资源调整大小（测试版）：就地资源调整已升级为测试版，允许在不重启的情况下动态更新现有容器组（pod）的 CPU 和内存资源，从而可以在零停机时间的情况下垂直扩缩有状态工作负载，并根据流量模式进行无缝资源调整。
- 附加容器现已稳定：附加容器已升级到稳定状态，使用 `restartPolicy: Always` 将附加容器作为特殊的初始化容器实施，其在应用程序容器之前启动，在整个容器组（pod）生命周期中运行，并支持操作状态信令的探测器。
  - 有关更多信息，请参阅 Kubernetes 文档中的 Sidecar Containers。
- 端点 API 弃用：端点 API 现已正式弃用，访问时将返回警告 – 可迁移工作负载和脚本，改用 EndpointSlices API，后者支持双堆栈联网等现代功能，并且每项服务可以处理多个 EndPointSlice。
  - 有关更多信息，请参阅 Kubernetes 博客上的 Kubernetes v1.33: Continuing the transition from Endpoints to EndpointSlice。
- Elastic Fabric Adapter 支持：Amazon EKS 集群的默认安全组现在支持 Elastic Fabric Adapter（EFA）流量。默认安全组包含一条新的出站规则，该规则允许目标为同一安全组的 EFA 流量。这允许集群内的 EFA 流量。
  - 有关更多信息，请参阅《Amazon Elastic Compute Cloud 用户指南》中的适用于 Amazon EC2 上 AI/ML 和 HPC 工作负载的 Elastic Fabric Adapter。

有关完整的 Kubernetes `1.33` 更改日志，请参阅 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.33.md

Kubernetes 1.32

Kubernetes `1.32` 现已在 Amazon EKS 中可用。有关 Kubernetes `1.32` 的更多信息，请参阅官方发布公告。

Important

- FlowSchema 和 PriorityLevelConfiguration 的 `flowcontrol.apiserver.k8s.io/v1beta3` API 版本已在版本 `1.32` 中移除。如果您使用这些 API，则必须先更新配置以使用支持的最新版本，然后才能升级。

- ServiceAccount `metadata.annotations[kubernetes.io/enforce-mountable-secrets]` 已在版本 `1.32` 中弃用，并将在未来的 Kubernetes 次要版本发布时移除。建议使用单独的命名空间来隔离对已附加密钥的访问。
- Kubernetes 版本 `1.32` 是 Amazon EKS 将为其发布 Amazon Linux 2（AL2）AMI 的最后一个版本。从版本 `1.33` 起，Amazon EKS 将继续发布基于 Amazon Linux 2023（AL2023）和 Bottlerocket 的 AMI。

- 在 Kubernetes 版本 `1.32` 中，内存管理器功能已升级为正式发布（GA）功能。此增强功能可为容器化应用程序提供了更高效、更可预测的内存分配，对于具有特定内存要求的工作负载尤其实用。
- StatefulSets 创建的 PersistentVolumeClaims（PVC）现在包含自动清理功能。不再需要的 PVC 将被自动删除，同时在 StatefulSet 更新和节点维护操作期间保持数据持久性。此功能简化了存储管理，有助于防止集群中出现孤岛 PVC。
- 推出了自定义资源字段选择器功能，让开发人员能够向自定义资源添加字段选择器。此功能为自定义资源提供了与内置 Kubernetes 对象相同的筛选功能，从而实现了更精确、更高效的资源筛选，并促进了更好的 API 设计实践。

有关完整的 Kubernetes `1.32` 更改日志，请参阅 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.32.md

匿名身份验证更改

从 Amazon EKS `1.32` 开始，匿名身份验证将仅限于以下 API 服务器运行状况检查端点：

- `/healthz`
- `/livez`
- `/readyz`

使用 `system:unauthenticated` 用户向任何其他端点发出的请求都将收到 `401 Unauthorized` HTTP 响应。此安全增强功能有助于防止由于 RBAC 策略配置错误而可能发生的意外集群访问。

> **Note**
>
> `public-info-viewer` RBAC 角色继续适用于上面列出的运行状况检查端点。

Amazon Linux 2 AMI 弃用

Kubernetes 版本 1.32 是 Amazon EKS 将为其发布 AL2 AMI 的最后一个版本。从 1.33 起，Amazon EKS 将继续发布基于 AL2023 和 Bottlerocket 的 AMI。有关更多信息，请参阅 the section called “AL2 AMI 弃用”。

查看 Kubernetes 版本延期支持的发布说明

Amazon EKS 支持 Kubernetes 版本的时间长于上游支持的时间。从 Kubernetes 次要版本在 Amazon EKS 中发布之日起，标准支持期为 14 个月；Kubernetes 次要版本的扩展支持在此基础上提供额外 12 个月的支持（每个版本总计 26 个月）。

本主题介绍有关扩展支持中每个 Kubernetes 版本需要注意的重要更改。升级时，请仔细检查集群新旧版本之间发生的变化。

Kubernetes 1.31

Kubernetes 1.31 现已在 Amazon EKS 中可用。有关 Kubernetes 1.31 的更多信息，请参阅官方发布公告。

> **Important**
>
> - 自 2017 年以来已弃用的 kubelet 标志 `--keep-terminated-pod-volumes` 已在版本 1.31 发布时移除。此更改会影响 kubelet 处理已终止的容器组（pod）卷的方式。如果您在节点配置中使用此标志，则必须先更新引导脚本并启动模板，以在升级前删除它。

- 测试版 `VolumeAttributesClass` 功能门和 API 资源已在 Amazon EKS 版本 1.31 中启用。此功能允许集群操作员修改由兼容 CSI 驱动程序（包括 Amazon EBS CSI 驱动程序）管理的持久卷（PV）的可变属性。要利用此功能，请确保您的 CSI 驱动程序支持 `VolumeAttributesClass` 功能（对于 Amazon EBS CSI 驱动程序，请升级到版本 1.35.0 或更高版本以自动启用该功能）。您将能够创建 `VolumeAttributesClass` 对象来定义所需的卷属性，例如卷类型和吞吐量，并将它们与您的持久卷声明（PVC）相关联。有关更多信息，请参阅 Kubernetes 官方文档以及 CSI 驱动程序的文档。
  - 有关 Amazon EBS CSI 驱动程序更多信息，请参阅the section called “Amazon EBS”。
- Kubernetes 对 AppArmor 的支持已逐渐达到稳定状态，现已正式可供公众使用。此功能允许您通过在容器的 `securityContext` 中设置 `appArmorProfile.type` 字段来使用 Apparmor 保护您的

容器。在 Kubernetes 版本 1.30 之前，AppArmor 由注释控制。从版本 1.30 开始，它使用字段进行控制。要利用此功能，我们建议不要使用注释，而是使用 appArmorProfile.type 字段来确保您的工作负载兼容。

- PersistentVolume 最后阶段的转换时间功能已逐渐达到稳定状态，现已在 Kubernetes 版本 1.31 中正式可供公众使用。此功能在 PersistentVolumeStatus 中引入了一个新字段 .status.lastTransitionTime，该字段提供了 PersistentVolume 上次转换到不同阶段的时间戳。此增强功能可以更好地跟踪和管理 PersistentVolumes，尤其是在有必要了解卷生命周期的情况下。

有关完整的 Kubernetes 1.31 更改日志，请参阅 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.31.md

Kubernetes 1.30

Kubernetes 1.30 现已在 Amazon EKS 中可用。有关 Kubernetes 1.30 的更多信息，请参阅官方发布公告。

- 从 Amazon EKS 版本 1.30 或更高版本开始，任何新创建的托管节点组都将自动默认使用 Amazon Linux 2023（AL2023）作为节点操作系统。有关为托管节点组指定操作系统的更多信息，请参阅 the section called “创建”。
- 使用 Amazon EKS 1.30，将 topology.k8s.aws/zone-id 标签添加到工作线程节点中。您可以使用可用区 ID（AZ ID），以确定一个账户中的资源相对于另一个账户中的资源所在的位置。有关更多信息，请参阅《Amazon RAM 用户指南》中的适用于 Amazon 资源的可用区 ID。
- 从 1.30 开始，Amazon EKS 不再包含有关应用于新创建集群的 gp2 StorageClass 资源的 default 注释。如果您按名称引用此存储类，则不会产生任何影响。如果您依赖集群中具有默认 StorageClass，则必须采取措施。您应该通过名称 StorageClass 来引用 gp2。您也可以通过在安装版本 1.31.0 或更高版本 的 aws-ebs-csi-driver add-on 时将 defaultStorageClass.enabled 参数设置为 true，来部署 Amazon EBS 建议的默认存储类。
- Amazon EKS 集群 IAM 角色所需的最低 IAM 策略已更改。操作 ec2:DescribeAvailabilityZones 是必需的。有关更多信息，请参阅 the section called “集群 IAM 角色”。

有关完整的 Kubernetes 1.30 更改日志，请参阅 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.30.md。

Kubernetes 1.29

Kubernetes 1.29 现已在 Amazon EKS 中可用。有关 Kubernetes 1.29 的更多信息，请参阅官方发布公告。

> **Important**
>
> - Kubernetes 版本 1.29 中不再提供已弃用的 flowcontrol.apiserver.k8s.io/v1beta2 API 版本 FlowSchema 和 PriorityLevelConfiguration。如果您有清单或客户端软件在使用已弃用的测试版 API 组，则应该先更改这些清单或客户端软件，然后再升级版本 1.29。

- 节点对象的 .status.kubeProxyVersion 字段已弃用，而 Kubernetes 项目在未来版本中延迟删除此字段。已弃用的字段不正确，且过去一直由 kubelet 管理 - 并不真的知道 kube-proxy 版本，或 kube-proxy 是否正在运行。如果您一直在客户端软件中使用此字段，请停止使用 – 其中的信息不可靠且该字段已弃用。
- 在 Kubernetes 1.29 中用于减少潜在攻击面，LegacyServiceAccountTokenCleanUp 功能将自动生成的基于密钥的旧令牌标记为无效，如果这些令牌长时间未使用的话（默认为 1 年），且如果在标记为无效后长时间无人使用，则会自动删除（默认为额外增加 1 年）。可运行以下内容识别此类令牌：

```
kubectl get cm kube-apiserver-legacy-service-account-token-tracking -n kube-system
```

有关完整的 Kubernetes 1.29 更改日志，请参阅 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.29.md#changelog-since-v1280。

查看当前集群支持期

Amazon 控制台的集群支持期部分会显示集群当前处于标准支持还是扩展支持状态。如果集群的支持期为扩展支持，则需要支付 EKS 扩展支持费用。

有关标准支持和延期支持的更多信息，请参阅了解 EKS 上的 Kubernetes 版本生命周期。

1. 导航到 Amazon 控制台中 EKS 部分的集群页面。确认控制台的 Amazon 区域设置与您要查看的集群相同。

2. 检查支持期列。如果值为标准支持有效期至...，则当前不会收取延期支持费用。您还处于标准支持期内。如果值为延期支持...，则该集群当前正在产生延期支持费用。

> **Note**
>
> 无法使用 Amazon API 或 CLI 检索支持期。

查看当前集群升级策略

集群升级策略决定了集群在标准支持期结束时会发生什么。如果升级策略是 EXTENDED，则集群将不会自动升级，而是进入扩展支持状态。如果升级策略是 STANDARD，则将自动升级。

借助 Amazon EKS 的 Kubernetes 版本策略控件，您可以选择 EKS 集群的标准支持结束行为。借助这些控件，您可以决定哪些集群应在某个 Kubernetes 版本的标准支持结束时进入延期支持，哪些集群应自动升级。

在发布后的前 14 个月内，次要版本会在 Amazon EKS 中获得标准支持。一个版本超过标准支持终止日期后，将在接下来 12 个月自动进入扩展支持。延期支持允许您在特定 Kubernetes 版本上停留更长时间，但需要按小时为集群支付额外费用。您可以为 EKS 集群启用或禁用扩展支持。如果您禁用扩展支持，则 Amazon 会在标准支持终止时自动将集群升级到下一个版本。如果启用扩展支持，则可以在有限的时间内保持当前版本，但需要支付额外的费用。即使使用了延期支持，也要计划定期升级 Kubernetes 集群。

您可以使用 `supportType` 属性为新集群和现有集群设置版本策略。您可以通过两个选项来设置版本支持策略：

- **`STANDARD`** – 标准支持终止后，EKS 集群可以自动升级。使用此设置不会产生延期支持费用，但 EKS 集群将自动升级到处于标准支持期内的下一个受支持的 Kubernetes 版本。
- **`EXTENDED`** – Kubernetes 版本的标准支持结束后，EKS 集群会进入延期支持状态。使用此设置，您将需要支付扩展支持费用。您可以将集群升级到标准支持期内的 Kubernetes 版本，以免产生延期支持费用。依据扩展支持运行的集群在扩展支持终止时将可以自动升级。

默认情况下会为新集群和现有集群启用扩展支持。您可以在 Amazon Web Services 管理控制台中或使用 Amazon CLI 查看是否为集群启用了扩展支持。

> **Important**
>
> 如果希望集群保持当前 Kubernetes 版本以利用延期支持期，则必须在标准支持期结束之前启用延期支持升级策略。

仅当集群在标准支持期内的 Kubernetes 版本上运行时，才能为其设置版本支持策略。该版本进入扩展支持期后，除非在标准支持期内的版本上运行，否则您将无法更改此设置。

例如，假设将版本支持策略设置为 `standard`，则在集群上运行的 Kubernetes 版本在标准支持终止后，您将无法更改此设置。如果将版本支持策略设置为 `extended`，则在集群上运行的 Kubernetes 版本在标准支持终止后，您将无法更改此设置。要更改版本支持策略设置，集群必须在支持标准的 Kubernetes 版本上运行。

查看集群升级策略（Amazon 控制台）

1. 导航到 Amazon 控制台中 EKS 部分的集群页面。确认控制台的 Amazon 区域设置与您要查看的集群相同。
2. 检查升级策略列。如果值为标准支持，则集群将无法进入扩展支持。如果值为扩展支持，则集群将进入扩展支持。

查看集群升级策略（Amazon CLI）

1. 验证确认已安装了 Amazon CLI 并且您已登录。了解如何更新和安装 Amazon CLI。
2. 确定 EKS 集群的名称。将 CLI 设置为与 EKS 集群相同的 Amazon 区域。
3. 运行以下命令：

```
aws eks describe-cluster \
--name <cluster-name> \
--query "cluster.upgradePolicy.supportType"
```

4. 如果值为 `STANDARD`，则集群将无法进入扩展支持。如果值为 `EXTENDED`，则集群将进入扩展支持。

通过启用 EKS 扩展支持，可提高计划 Kubernetes 版本升级的灵活性

本主题介绍如何设置 EKS 集群的升级策略，以启用扩展支持。EKS 集群升级策略决定了集群在标准支持期结束时会发生什么。如果集群升级策略启用了扩展支持，则将在标准支持期结束时进入扩展支持期。标准支持期结束后，集群不会自动升级。

处于扩展支持期的集群实际上会产生更高的成本。如果集群仅将升级策略设置为启用扩展支持，而集群处于标准支持期内，则会产生标准成本。

如果您在 Amazon 控制台中创建集群，则会将升级策略设置为禁用延期支持。如果以别的方式创建集群，则会将升级策略设置为启用延期支持。例如，使用 Amazon API 创建的集群就已启用延期支持。

有关升级策略的更多信息，请参阅集群升级策略。

> **Important**
>
> 如果希望集群保持当前 Kubernetes 版本以利用延期支持期，则必须在标准支持期结束之前启用延期支持升级策略。
> 如果未启用扩展支持，集群将会自动升级。

启用 EKS 扩展支持（Amazon 控制台）

1. 在 Amazon 控制台中导航至您的 EKS 集群。在集群信息页面上，选择概述选项卡。
2. 在 Kubernetes 版本设置部分中，选择管理。
3. 选择扩展支持，然后选择保存更改。

启用 EKS 扩展支持（Amazon CLI）

1. 验证确认已安装了 Amazon CLI 并且您已登录。了解如何更新和安装 Amazon CLI。
2. 确定 EKS 集群的名称。
3. 运行以下命令：

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=EXTENDED
```

通过禁用 EKS 扩展支持来防止集群成本增加

本主题介绍如何设置 EKS 集群的升级策略，以禁用扩展支持。EKS 集群升级策略决定了集群在标准支持期结束时会发生什么。如果集群升级策略禁用了延期支持，则将自动升级到下一个 Kubernetes 版本。

有关升级策略的更多信息，请参阅集群升级策略。

> **Important**
>
> 集群进入扩展支持后，将无法禁用扩展支持。只能在标准支持下为集群禁用扩展支持。Amazon 建议将集群升级到处于标准支持期内的版本。

禁用 EKS 扩展支持（Amazon 控制台）

1. 在 Amazon 控制台中导航至您的 EKS 集群。在集群信息页面上，选择概述选项卡。
2. 在 Kubernetes 版本设置部分中，选择管理。
3. 选择标准支持，然后选择保存更改。

禁用 EKS 扩展支持（Amazon CLI）

1. 验证确认已安装了 Amazon CLI 并且您已登录。了解如何更新和安装 Amazon CLI。
2. 确定 EKS 集群的名称。
3. 运行以下命令：

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=STANDARD
```

查看每个 Kubernetes 版本的 Amazon EKS 平台版本

Amazon EKS 平台版本表示 Amazon EKS 集群控制面板的功能，例如已启用哪些 Kubernetes API 服务器标志以及当前的 Kubernetes 补丁版本。每个 Kubernetes 次要版本都有一个或多个关联的 Amazon EKS 平台版本。不同 Kubernetes 次要版本对应的平台版本相互独立。您可以使用 Amazon CLI 或 Amazon Web Services 管理控制台 检索集群的当前平台版本。如果您在 Amazon Outposts 上有一个本地集群，请参阅the section called “EKS 平台版本”而不是本主题。

当 Amazon EKS 中有新的 Kubernetes 次要版本（例如 1.33）可用时，则该 Kubernetes 次要版本的初始 Amazon EKS 平台版本从 `eks.1` 开始。不过，Amazon EKS 会定期发布新平台版本来支持新的 Kubernetes 控制层面设置并提供安全修复。

当次要版本的新 Amazon EKS 平台版本可用时：

- Amazon EKS 平台版本号将递增 (`eks.<n+1>`)。
- Amazon EKS 会自动将所有现有集群升级到其对应的 Kubernetes 次要版本的最新 Amazon EKS 平台版本。现有 Amazon EKS 平台版本的自动升级将增量推出。推出过程可能需要一段时间。如果您即刻就需要最新的 Amazon EKS 平台版本功能，您应创建新的 Amazon EKS 集群。

  如果您的集群比当前平台版本落后两个以上的平台版本，那么 Amazon EKS 可能无法自动更新您的集群。有关可能导致这种情况的原因的详细信息，请参阅 the section called “Amazon EKS 平台版本比当前平台版本落后两个版本以上”。
- Amazon EKS 可能会发布具有对应的修补程序版本的新工作线程 AMI。但是，所有补丁版本在指定 Kubernetes 次要版本的 EKS 控制层面与节点 AMI 之间兼容。

新 Amazon EKS 平台版本不会引发破坏性的更改或导致服务中断。

集群始终使用指定 Kubernetes 版本的最新可用 Amazon EKS 平台版本 (`eks.<n>`) 创建。如果您将集群更新为新的 Kubernetes 次要版本，您的集群将接收所更新到的 Kubernetes 次要版本的当前 Amazon EKS 平台版本。

下列各表列出了当前和最新的 Amazon EKS 平台版本。

> **Note**
>
> Amazon 最近禁用了 2024 年 6 月发布的某些平台版本。这些平台版本存在稳定性问题。无需任何操作。

要接收此特定文档页面的所有源文件更改通知，您可以通过 RSS 阅读器订阅以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/clusters/platform-versions.adoc.atom
```

Kubernetes 版1.34

所有 1.34 平台版本都启用了以下准入控制器：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.34.1	eks.9	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 18 日
1.34.1	eks.8	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 17 日
1.34.1	eks.7	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 6 日
1.34.1	eks.6	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 30 日
1.34.1	eks.5	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.34.1	eks.4	面向 EKS 的 Kubernetes 1.34 的初始版本。有关更多信息，请参阅 the section called “Kubernetes 1.34”。	2025 年 10 月 2 日

Kubernetes 版1.33

所有 1.33 平台版本都启用了以下准入控制器：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.33.5	eks.23	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 18 日
1.33.5	eks.22	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 17 日
1.33.5	eks.21	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 6 日
1.33.5	eks.20	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 30 日
1.33.5	eks.19	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 28 日
1.33.5	eks.18	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.33.5	eks.17	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.33.5	eks.16	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 2 日
1.33.4	eks.15	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.33.4	eks.14	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.33.4	eks.13	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 16 日
1.33.4	eks.12	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 5 日
1.33.3	eks.11	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 25 日
1.33.3	eks.10	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 17 日
1.33.2	eks.9	包含安全修复和增强功能的新平台版本。1.33 eks.7 和 1.33 eks.8 已在内部弃用，从未发布。	2025 年 7 月 30 日
1.33.1	eks.6	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 26 日
1.33.1	eks.5	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 11 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.33.1	eks.4	面向 EKS 的 Kubernetes 1.33 的初始版本。有关更多信息，请参阅 the section called “Kubernetes 1.33”。	2025 年 5 月 28 日

Kubernetes 版1.32

所有 1.32 平台版本都启用了以下准入控制器：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.32.9	eks.30	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 18 日
1.32.9	eks.29	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 17 日
1.32.9	eks.28	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 6 日
1.32.9	eks.27	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 30 日
1.32.9	eks.26	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 28 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.32.9	eks.25	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.32.9	eks.24	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.32.9	eks.23	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 2 日
1.32.8	eks.22	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.32.8	eks.21	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.32.8	eks.20	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 16 日
1.32.8	eks.19	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 5 日
1.32.7	eks.18	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 25 日
1.32.7	eks.17	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 17 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.32.6	eks.16	包含安全修复和增强功能的新平台版本。1.32 eks.14 和 1.32 eks.15 已在内部弃用，从未发布。	2025 年 7 月 30 日
1.32.5	eks.13	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 26 日
1.32.5	eks.12	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 11 日
1.32.5	eks.11	新的平台版本（包括安全修复和增强功能）。	2025 年 5 月 30 日
1.32.3	eks.10	新的平台版本（包括安全修复和增强功能）。	2025 年 5 月 16 日
1.32.3	eks.9	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 29 日
1.32.3	eks.8	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 18 日
1.32.3	eks.7	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 18 日
1.32.3	eks.6	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 2 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.32.2	eks.5	新的平台版本（包括安全修复和增强功能）。	2025 年 3 月 17 日
1.32.2	eks.4	新的平台版本（包括安全修复和增强功能）。	2025 年 3 月 4 日
1.32.1	eks.3	新的平台版本（包括安全修复和增强功能）。	2025 年 2 月 24 日
1.32.0	eks.2	面向 EKS 的 Kubernetes 1.32 的初始版本。有关更多信息，请参阅 the section called “Kubernetes 1.32”。	2025 年 1 月

Kubernetes 版1.31

所有 1.31 平台版本都启用了以下准入控制器：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.31.13	eks.46	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 18 日
1.31.13	eks.45	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 17 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.31.13	eks.44	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 6 日
1.31.13	eks.43	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 30 日
1.31.13	eks.42	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 28 日
1.31.13	eks.41	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.31.13	eks.40	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.31.13	eks.39	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 2 日
1.31.12	eks.38	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.31.12	eks.37	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.31.12	eks.36	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 16 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.31.12	eks.35	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 5 日
1.31.11	eks.34	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 25 日
1.31.11	eks.33	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 17 日
1.31.10	eks.32	包含安全修复和增强功能的新平台版本。1.31 eks.30 和 1.31 eks.31 已在内部弃用，从未发布。	2025 年 7 月 30 日
1.31.9	eks.29	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 26 日
1.31.9	eks.28	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 11 日
1.31.9	eks.27	新的平台版本（包括安全修复和增强功能）。	2025 年 5 月 30 日
1.31.7	eks.26	新的平台版本（包括安全修复和增强功能）。	2025 年 5 月 16 日
1.31.7	eks.25	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 29 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.31.7	eks.24	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 18 日
1.31.7	eks.23	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 18 日
1.31.7	eks.22	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 2 日
1.31.6	eks.21	新的平台版本（包括安全修复和增强功能）。	2025 年 3 月 17 日
1.31.6	eks.20	新的平台版本（包括安全修复和增强功能）。	2025 年 3 月 4 日
1.31.5	eks.19	新的平台版本（包括安全修复和增强功能）。	2025 年 2 月 24 日
1.31.5	eks.18	新的平台版本（包括安全修复和增强功能）。	2025 年 2 月 24 日
1.31.4	eks.17	新的平台版本（包括安全修复和增强功能）。	2025 年 1 月 17 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.31.2	eks.12	新平台版本支持 Amazon EKS 混合节点功能，并增强了控制面板可观测性。请分别参阅 the section called “混合节点”以及 Amazon EKS enhances performance observability。	2024 年 11 月 15 日
1.31.1	eks.6	新的平台版本（包括安全修复和增强功能）。	2024 年 10 月 21 日
1.31.0	eks.4	面向 EKS 的 Kubernetes 1.31 的初始版本。有关更多信息，请参阅 the section called “Kubernetes 1.31”。	2024 年 9 月 26 日

Kubernetes 版**1.30**

所有 1.30 平台版本都启用了以下准入控制器：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.30.14	eks.54	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 18 日
1.30.14	eks.53	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 17 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.30.14	eks.52	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 6 日
1.30.14	eks.51	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 30 日
1.30.14	eks.50	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 28 日
1.30.14	eks.49	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.30.14	eks.48	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.30.14	eks.47	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 2 日
1.30.14	eks.46	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.30.14	eks.45	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.30.14	eks.44	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 16 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.30.14	eks.43	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 5 日
1.30.14	eks.42	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 25 日
1.30.14	eks.41	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 17 日
1.30.14	eks.40	包含安全修复和增强功能的新平台版本。1.30 eks.38 和 1.30 eks.39 已在内部弃用，从未发布。	2025 年 7 月 30 日
1.30.13	eks.37	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 26 日
1.30.13	eks.36	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 11 日
1.30.13	eks.35	新的平台版本（包括安全修复和增强功能）。	2025 年 5 月 30 日
1.30.11	eks.34	新的平台版本（包括安全修复和增强功能）。	2025 年 5 月 16 日
1.30.11	eks.33	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 29 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.30.11	eks.32	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 18 日
1.30.11	eks.31	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 18 日
1.30.11	eks.30	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 2 日
1.30.10	eks.29	新的平台版本（包括安全修复和增强功能）。	2025 年 3 月 17 日
1.30.10	eks.28	新的平台版本（包括安全修复和增强功能）。	2025 年 3 月 4 日
1.30.9	eks.27	新的平台版本（包括安全修复和增强功能）。	2025 年 2 月 24 日
1.30.8	eks.25	新的平台版本（包括安全修复和增强功能）。	2025 年 1 月 17 日
1.30.8	eks.24	新的平台版本（包括安全修复和增强功能）。	2025 年 1 月 3 日
1.30.7	eks.23	新的平台版本（包括安全修复和增强功能）。	2024 年 12 月 13 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.30.6	eks.22	新的平台版本（包括安全修复和增强功能）。	2024 年 12 月 13 日
1.30.6	eks.21	新的平台版本（包括安全修复和增强功能）。	2024 年 12 月 13 日
1.30.6	eks.20	新平台版本支持 Amazon EKS 混合节点功能，并增强了控制面板可观测性。请分别参阅 the section called “混合节点”以及 Amazon EKS enhances performance observability。	2024 年 11 月 15 日
1.30.5	eks.12	新的平台版本（包括安全修复和增强功能）。	2024 年 10 月 21 日
1.30.4	eks.8	新的平台版本（包括安全修复和增强功能）。	2024 年 9 月 3 日
1.30.3	eks.7	新的平台版本（包括安全修复和增强功能）。	2024 年 8 月 28 日
1.30.3	eks.6	新的平台版本（包括安全修复和增强功能）。	2024 年 8 月 9 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.30.2	eks.5	新的平台版本（包括安全修复和增强功能）。	2024 年 7 月 2 日
1.30.0	eks.2	面向 EKS 的 Kubernetes 1.30 的初始版本。有关更多信息，请参阅 the section called “Kubernetes 1.30”。	2024 年 5 月 23 日

Kubernetes 版1.29

所有 1.29 平台版本都启用了以下准入控制器：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.29.15	eks.57	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 18 日
1.29.15	eks.56	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 17 日
1.29.15	eks.55	新的平台版本（包括安全修复和增强功能）。	2025 年 11 月 6 日
1.29.15	eks.54	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 30 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.29.15	eks.53	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 28 日
1.29.15	eks.52	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.29.15	eks.51	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 15 日
1.29.15	eks.50	新的平台版本（包括安全修复和增强功能）。	2025 年 10 月 2 日
1.29.15	eks.49	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.29.15	eks.48	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 29 日
1.29.15	eks.47	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 16 日
1.29.15	eks.46	新的平台版本（包括安全修复和增强功能）。	2025 年 9 月 5 日
1.29.15	eks.45	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 25 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.29.15	eks.44	新的平台版本（包括安全修复和增强功能）。	2025 年 8 月 17 日
1.29.15	eks.43	包含安全修复和增强功能的新平台版本。1.29 eks.41 和 1.29 eks.42 已在内部弃用，从未发布。	2025 年 7 月 30 日
1.29.15	eks.40	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 26 日
1.29.15	eks.39	新的平台版本（包括安全修复和增强功能）。	2025 年 6 月 11 日
1.29.15	eks.38	新的平台版本（包括安全修复和增强功能）。	2025 年 5 月 30 日
1.29.15	eks.37	新的平台版本（包括安全修复和增强功能）。	2025 年 5 月 16 日
1.29.15	eks.36	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 29 日
1.29.15	eks.35	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 18 日
1.29.15	eks.34	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 18 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.29.15	eks.33	新的平台版本（包括安全修复和增强功能）。	2025 年 4 月 2 日
1.29.14	eks.32	新的平台版本（包括安全修复和增强功能）。	2025 年 3 月 17 日
1.29.14	eks.31	新的平台版本（包括安全修复和增强功能）。	2025 年 3 月 4 日
1.29.13	eks.30	新的平台版本（包括安全修复和增强功能）。	2025 年 2 月 24 日
1.29.13	eks.29	新的平台版本（包括安全修复和增强功能）。	2025 年 2 月 24 日
1.29.12	eks.28	新的平台版本（包括安全修复和增强功能）。	2025 年 1 月 20 日
1.29.12	eks.27	新的平台版本（包括安全修复和增强功能）。	2025 年 1 月 3 日
1.29.11	eks.26	新的平台版本（包括安全修复和增强功能）。	2024 年 12 月 13 日
1.29.10	eks.25	新的平台版本（包括安全修复和增强功能）。	2024 年 12 月 13 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.29.10	eks.24	新的平台版本（包括安全修复和增强功能）。	2024 年 12 月 13 日
1.29.10	eks.23	新平台版本支持 Amazon EKS 混合节点功能，并增强了控制面板可观测性。请分别参阅 the section called “混合节点”以及 Amazon EKS enhances performance observability。	2024 年 11 月 15 日
1.29.9	eks.17	新的平台版本（包括安全修复和增强功能）。	2024 年 10 月 21 日
1.29.8	eks.13	新的平台版本（包括安全修复和增强功能）。	2024 年 9 月 3 日
1.29.7	eks.12	新的平台版本（包括安全修复和增强功能）。	2024 年 8 月 28 日
1.29.7	eks.11	新的平台版本（包括安全修复和增强功能）。	2024 年 8 月 9 日
1.29.6	eks.10	新的平台版本（包括安全修复和增强功能）。	2024 年 7 月 2 日

Kubernetes 版本	EKS 平台版本	发行说明	发行日期
1.29.4	eks.7	具有 CoreDNS 自动扩缩、安全修复和增强功能的新平台版本。有关 CoreDNS 自动扩缩的更多信息，请参阅 the section called “针对高流量扩展”。	2024 年 5 月 16 日
1.29.3	eks.6	新的平台版本（包括安全修复和增强功能）。	2024 年 4 月 18 日
1.29.1	eks.5	新的平台版本（包括安全修复和增强功能）。	2024 年 3 月 29 日
1.29.1	eks.4	新的平台版本（包括安全修复和增强功能）。	2024 年 3 月 20 日
1.29.1	eks.3	新的平台版本（包括安全修复和增强功能）。	2024 年 3 月 12 日
1.29.0	eks.1	面向 EKS 的 Kubernetes 1.29 的初始版本。有关更多信息，请参阅 the section called “Kubernetes 1.29”。	2024 年 1 月 23 日

获取当前平台版本

1. 打开 Amazon EKS 控制台。
2. 在导航窗格中，选择集群。

3. 在集群列表中，选择集群名称以检查平台版本。
4. 选择概述选项卡。
5. 可在详细信息部分下方查看平台版本。
6. 确定要检查平台版本的集群的名称。
7. 运行如下命令：

```
aws eks describe-cluster --name my-cluster --query cluster.platformVersion
```

示例输出如下。

```
"eks.10"
```

更改平台版本

无法更改 EKS 集群的平台版本。当某个 Kubernetes 版本有新的 Amazon EKS 平台版本时，EKS 会自动将所有现有集群升级到对应 Kubernetes 版本的最新 Amazon EKS 平台版本。现有 Amazon EKS 平台版本的自动升级将增量推出。无法使用 Amazon 控制台或 CLI 更改平台版本。

如果升级了 Kubernetes 版本，集群将迁移至该 Kubernetes 版本的最新平台版本。

排查 Amazon EKS 集群和节点问题

本章介绍使用 Amazon EKS 时可能遇到的一些常见错误以及相应的错误处理方式。如果您需要对特定的 Amazon EKS 区域进行问题排查，请参阅单独的 the section called “故障排除”、the section called “EKS Connector 问题排查”，以及使用 EKS 插件对 ADOT 进行问题排查主题。

有关其它故障排除信息，请参阅 Amazon re:Post 上的有关 Amazon Elastic Kubernetes Service 的知识中心内容。

容量不足

如果您在尝试创建 Amazon EKS 集群时收到以下错误，则表示所指定的某个可用区容量不足，无法支持集群。

```
Cannot create cluster 'example-cluster' because region-1d, the targeted
Availability Zone, does not currently have sufficient capacity to support
the cluster. Retry and choose from these Availability Zones: region-1a,
region-1b, region-1c
```

在集群 VPC 中使用此错误消息所返回的可用区中托管的子网重新尝试创建集群。

有些可用区是集群无法驻留的。将子网所在的可用区与子网要求和注意事项中的可用区列表进行比较。

节点未能加入集群

有几种常见原因会阻止节点加入集群：

- 如果节点是托管节点，Amazon EKS 会在您创建节点组时向 `aws-auth ConfigMap` 中添加条目。如果该条目已被移除或修改，则需要重新添加该条目。要了解更多信息，请在您的终端中输入 `eksctl create iamidentitymapping --help`。您可以通过将以下命令中的 *`my-cluster`* 替换为集群名称，然后运行修改后的命令来查看当前的 `aws-auth ConfigMap` 条目：`eksctl get iamidentitymapping --cluster` *`my-cluster`* 。您指定的角色的 ARN 不能包含 `/` 之外的路径。例如，假设角色名称为 `development/apps/my-role`，则需要在为该角色指定 ARN 时将其更改为 `my-role`。确保指定了节点 IAM 角色 ARN（非实例配置文件 ARN）。

  如果节点是自主管理型的，并且您尚未为该节点的 IAM 角色的 ARN 创建访问条目，请运行与托管式节点列出的相同命令。如果您为节点 IAM 角色的 ARN 创建了访问条目，则可能无法在访问条目中正确配置该条目。确保将节点 IAM 角色 ARN（非实例配置文件 ARN）指定为 `aws-auth ConfigMap` 条目或访问条目中的主体 ARN。有关访问条目的更多信息，请参阅 the section called “访问条目”。

- 节点 Amazon CloudFormation 模板中的 ClusterName 与您希望节点加入的集群的名称不完全匹配。将不正确的值传递到此字段会导致节点的 `/var/lib/kubelet/kubeconfig` 文件配置不正确，并且节点将无法加入集群。
- 节点不会标记为由集群拥有。您的节点必须应用了以下标签，其中的 *my-cluster* 替换为集群的名称。

键	值
kubernetes.io/cluster/*my-cluster*	owned

- 节点可能无法使用公有 IP 地址访问集群。确保向部署在公有子网中的节点分配了公有 IP 地址。如果没有分配，您可以在节点启动后为其关联弹性 IP 地址。有关更多信息，请参阅将弹性 IP 地址与正在运行的实例或网络接口关联。如果公有子网未设置为自动将公有 IP 地址分配给部署到其中的实例，我们建议启用该设置。有关更多信息，请参阅修改子网的公有 IPv4 寻址属性。如果节点部署到私有子网，则该子网必须具有指向分配了公有 IP 地址的 NAT 网关的路由。
- 您的账户未启用节点部署所在 Amazon 区域的 Amazon STS 端点。要启用该区域，请参阅在 Amazon 区域中激活和停用 Amazon STS。
- 节点没有私有 DNS 条目，从而导致 `kubelet` 日志中包含 `node "" not found` 错误。确保创建节点的 VPC 在 `DHCP options set` 中以 `Options` 的形式设置了 `domain-name` 和 `domain-name-servers` 的值。默认值为 `domain-name:<region>.compute.internal` 和 `domain-name-servers:AmazonProvidedDNS`。有关更多信息，请参阅《Amazon VPC 用户指南》中的 DHCP 选项集。
- 如果托管节点组中的节点未在 15 分钟内连接到集群，则将发出运行状况问题“NodeCreationFailure”，控制台状态将设置为 `Create failed`。对于启动时间较慢的 Windows AMI，可以使用快速启动来解决此问题。

要确定导致 Worker 节点无法加入集群的常见原因并进行问题排查，您可以使用 `AWSSupport-TroubleshootEKSWorkerNode` 运行手册。有关更多信息，请参阅 Amazon Systems Manager 自动化运行手册参考中 `AWSSupport-TroubleshootEKSWorkerNode` 。

未经授权或访问被拒绝 (`kubectl`)

如果您在运行 `kubectl` 命令时收到以下错误之一，则说明您未针对 Amazon EKS 正确配置 `kubectl`，或您使用的 IAM 主体（角色或用户）的凭证未映射到对 Amazon EKS 集群上的 Kubernetes 对象具有足够权限的 Kubernetes 用户名。

- `could not get token: AccessDenied: Access denied`
- `error: You must be logged in to the server (Unauthorized)`
- `error: the server doesn't have a resource type "svc"`

这可能是由以下原因之一导致的：

- 集群是使用一个 IAM 主体的凭证创建的，并且 `kubectl` 配置为使用另一个 IAM 主体的凭证。要解决此问题，请更新您的 `kube config` 文件以使用创建集群的凭证。有关更多信息，请参阅 the section called "使用 kubectl 访问集群"。
- 如果集群满足使用 EKS 访问条目向 IAM 用户授予对 Kubernetes 的访问权限的先决条件部分中的最低平台要求，则 IAM 主体中不存在访问条目。如果存在，则说明没有为其定义必要的 Kubernetes 组名称，或者没有与之关联的正确访问策略。有关更多信息，请参阅 the section called "访问条目"。
- 如果集群不符合使用 EKS 访问条目向 IAM 用户授予对 Kubernetes 的访问权限中的最低平台要求，则 `aws-auth ConfigMap` 中不存在具有您的 IAM 主体的条目。如果存在，则不会映射到 Kubernetes 组名称，这些名称已绑定到具有必要权限的 Kubernetes `Role` 或 `ClusterRole`。有关 Kubernetes 基于角色授权（RBAC）对象的更多信息，请参阅 Kubernetes 文档中的使用 RBAC 鉴权。您可以通过将以下命令中的 *`my-cluster`* 替换为集群名称，然后运行修改后的命令来查看当前的 `aws-auth ConfigMap` 条目：`eksctl get iamidentitymapping --cluster` *`my-cluster`* 。如果 `ConfigMap` 中不存在具有您的 IAM 主体的 ARN 的条目，请在您的终端中输入 `eksctl create iamidentitymapping --help` 以了解如何创建一个。

如果安装和配置 Amazon CLI，则可配置您使用的 IAM 凭证。有关更多信息，请参阅《Amazon 命令行界面用户指南》中的配置 Amazon CLI。如果担任 IAM 角色来访问集群上的 Kubernetes 对象，则也可以配置 `kubectl` 使用 IAM 角色。有关更多信息，请参阅 the section called "使用 kubectl 访问集群"。

`hostname doesn't match`

系统的 Python 版本必须为 `2.7.9` 或更高版本。否则，在 Amazon CLI 调用 Amazon EKS 时会收到 `hostname doesn't match` 错误。有关更多信息，请参阅《Python Requests Frequently Asked Questions》中的 What are "hostname doesn't match" errors?

`getsockopt: no route to host`

Docker 在 Amazon EKS 集群中的 `172.17.0.0/16` CIDR 范围内运行。我们建议您的集群的 VPC 子网不重叠此范围。否则，您将收到以下错误：

```
Error: : error upgrading connection: error dialing backend: dial tcp
 172.17.<nn>.<nn>:10250: getsockopt: no route to host
```

Instances failed to join the Kubernetes cluster

如果您在 Amazon Web Services 管理控制台 中收到 `Instances failed to join the Kubernetes cluster` 错误，请确保已启用集群的私有端点访问，或者您已正确配置 CIDR 块以用于公有端点访问。有关更多信息，请参阅 the section called “集群终端节点访问”。

托管节点组错误代码

如果您的托管节点组遇到硬件运行状况问题，则 Amazon EKS 返回错误代码以帮助您诊断问题。这些运行状况检查无法检测到软件问题，因为它们基于 Amazon EC2 运行状况检查。下面的列表介绍了错误代码。

AccessDenied

Amazon EKS 或者一个或多个托管节点无法向 Kubernetes 集群 API 服务器进行身份验证或授权。有关解决常见原因的更多信息，请参阅 the section called “修复托管节点组的 AccessDenied 错误的常见原因”。私有 Windows AMI 也可能导致出现此错误代码以及 `Not authorized for images` 错误消息。有关更多信息，请参阅 the section called “Not authorized for images”。

AmiIdNotFound

找不到与您的启动模板关联的 AMI ID。确保 AMI 存在并与您的账户共享。

AutoScalingGroupNotFound

找不到与托管节点组关联的自动扩缩组。您可以重新创建具有相同设置的自动扩缩组进行恢复。

ClusterUnreachable

Amazon EKS 或者一个或多个托管节点无法与 Kubernetes 集群 API 服务器通信。如果存在网络中断或 API 服务器处理请求时超时，则可能会发生这种情况。

Ec2SecurityGroupNotFound

找不到集群的集群安全组。您必须重新创建集群。

Ec2SecurityGroupDeletionFailure

无法删除托管节点组的远程访问安全组。从安全组中删除所有依赖关系。

Ec2LaunchTemplateNotFound

找不到托管节点组的 Amazon EC2 启动模板。您必须重新创建节点组才能恢复。

Ec2LaunchTemplateVersionMismatch

托管节点组的 Amazon EC2 启动模板版本与 Amazon EKS 创建的版本不匹配。您可以恢复到 Amazon EKS 创建的版本以进行恢复。

IamInstanceProfileNotFound

找不到托管节点组的 IAM 实例配置文件。您可以重新创建具有相同设置的实例配置文件进行恢复。

IamNodeRoleNotFound

找不到托管节点组的 IAM 角色。您可以重新创建具有相同设置的 IAM 角色进行恢复。

AsgInstanceLaunchFailures

自动扩缩组在尝试启动实例时出现故障。

NodeCreationFailure

启动的实例无法注册到 Amazon EKS 集群。此故障的常见原因是节点 IAM 角色权限不足，或节点缺少出站 Internet 访问权限。您的节点必须符合以下要求之一：

- 能够使用公有 IP 地址访问互联网。与节点所在子网关联的安全组必须允许进行通信。有关更多信息，请参阅the section called “子网要求和注意事项”和the section called “安全组要求”。
- 节点和 VPC 必须满足部署具有有限互联网访问权限的私有集群中的要求。

InstanceLimitExceeded

Amazon 账户无法启动指定实例类型的更多实例。您可以请求提高 Amazon EC2 实例限制以进行恢复。

InsufficientFreeAddresses

与托管节点组关联的一个或多个子网没有足够的 IP 地址供新节点使用。

InternalFailure

这些错误一般由 Amazon EKS 服务器端问题导致。

修复托管节点组的 **AccessDenied** 错误的常见原因

在托管节点组上执行操作时最常见的 `AccessDenied` 错误原因是缺少 `eks:node-manager` `ClusterRole` 或 `ClusterRoleBinding`。Amazon EKS 会在托管节点组启动的过程中在您的集群中设置这些资源，这些资源是管理节点组所必需的。

ClusterRole 可能随着时间的推移而改变，但它应该类似于以下示例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
```

ClusterRoleBinding 可能随着时间的推移而改变，但它应该类似于以下示例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
```

```
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

验证 eks:node-manager ClusterRole 是否存在。

```
kubectl describe clusterrole eks:node-manager
```

如果存在，则将输出与上一个 ClusterRole 示例进行比较。

验证 eks:node-manager ClusterRoleBinding 是否存在。

```
kubectl describe clusterrolebinding eks:node-manager
```

如果存在，则将输出与上一个 ClusterRoleBinding 示例进行比较。

如果您发现在请求托管节点组操作期间，由于 ClusterRole 或 ClusterRoleBinding 缺失或损坏造成了 AcessDenied 错误，则可还原它们。将以下内容保存到名为 *eks-node-manager-role.yaml* 的文件中。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
```

```
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

应用文件。

```
kubectl apply -f eks-node-manager-role.yaml
```

重试节点组操作，查看是否解决了您的问题。

Not authorized for images

Not authorized for images 错误消息的一个潜在原因是使用私有 Amazon EKS Windows AMI 启动 Windows 托管节点组。发布新的 Windows AMI 后，Amazon 会将已超过 4 个月的 AMI 设为私有，这会使得这些 AMI 不再可访问。如果您的托管节点组使用的是私有 Windows AMI，则请考虑更新 Windows 托管节点组。虽然我们不能保证可以提供对已设为私有的 AMI 的访问权限，但您可以通过向 Amazon Support 提交服务单来请求访问权限。有关更多信息，请参阅《Amazon EC2 用户指南》中的修补。

节点处于 NotReady 状态

如果节点进入 NotReady 状态，可能表示该节点运行状况不佳，无法调度新的容器组（pod）。这可能是由于各种原因造成的，例如节点缺少足够的 CPU、内存或可用磁盘空间资源。

对于 Amazon EKS 优化版 Windows AMI，在 kubelet 配置中没有默认指定的计算资源预留。为帮助防止出现资源问题，您可以为系统进程预留计算资源，方法是为 kubelet 提供 kube-reserved 和/或 system-reserved 的配置值。您可以使用引导脚本中的 -KubeletExtraArgs 命令行参数来执行此操作。有关更多信息，请参阅 Kubernetes 文档中的为系统守护进程预留计算资源和本用户指南中的the section called “引导脚本配置参数”。

EKS 日志收集器

要解决与 Amazon EKS 节点有关的问题，可以在位于 /etc/eks/log-collector-script/eks-log-collector.sh 的节点上使用预构建脚本。您可以使用该脚本收集有关支持案例和常规故障排除的诊断日志。

使用以下命令可在您的节点上运行脚本：

```
sudo bash /etc/eks/log-collector-script/eks-log-collector.sh
```

> **Note**
>
> 如果该位置没有脚本。可以使用以下命令手动下载并运行脚本：
>
> ```
> curl -O https://amazon-eks.s3.amazonaws.com/support/log-collector-script/linux/eks-log-collector.sh
> sudo bash eks-log-collector.sh
> ```

该脚本收集以下诊断信息。

```
$ sudo bash /etc/eks/log-collector-script/eks-log-collector.sh

      This is version 0.7.8. New versions can be found at https://github.com/awslabs/amazon-eks-ami/blob/main/log-collector-script/

Trying to collect common operating system logs...
Trying to collect kernel logs...
Trying to collect mount points and volume information...
...
...
```

```
 Done... your bundled logs are located in /var/log/eks_i-EXAMPLE_2025-03-25_0000-
UTC_0.7.8.tar.gz
```

诊断信息收集并存储在：

```
/var/log/eks_i-EXAMPLE_2025-03-25_0000-UTC_0.7.8.tar.gz
```

要检索 Bottlerocket 节点的日志捆绑包，请参阅 Bottlerocket Log 了解更多详细信息。

容器运行时网络未准备就绪

您可能会收到类似于以下内容的 `Container runtime network not ready` 错误和授权错误：

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
 uninitialized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
4191 kubelet_node_status.go:106] Unable to register node
 "ip-10-40-175-122.ec2.internal" with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

这可能是由以下原因之一导致的：

1. 您的集群上要么没有 `aws-auth` ConfigMap，要么其中不包含您为节点配置的 IAM 角色的条目。

   要解决此问题，可以通过将以下命令中的 *`my-cluster`* 替换为集群名称，然后运行修改后的命令 `eksctl get iamidentitymapping --cluster` *`my-cluster`* 来查看 ConfigMap 中的现有条目。如果您收到来自该命令的错误消息，则可能是因为集群没有 `aws-auth` ConfigMap。以下命令将条目添加到 ConfigMap。如果 ConfigMap 不存在，该命令也会创建一个。请将 *`111122223333`* 替换为 IAM 角色的 Amazon 账户 ID，将 *`myAmazonEKSNodeRole`* 替换为节点角色的名称。

   ```
   eksctl create iamidentitymapping --cluster my-cluster \
       --arn arn:aws-cn:iam::111122223333:role/myAmazonEKSNodeRole --group
    system:bootstrappers,system:nodes \
       --username system:node:{{EC2PrivateDNSName}}
   ```

您指定的角色的 ARN 不能包含 / 之外的路径。例如，如果您的角色名称为 `development/apps/my-role`，则需要在指定角色的 ARN 时将其更改为 `my-role`。确保指定了节点 IAM 角色 ARN（非实例配置文件 ARN）。

2. 您的自主管理型节点位于集群中，其平台版本为使用 EKS 访问条目向 IAM 用户授予对 Kubernetes 的访问权限主题中先决条件部分所列的最低版本，但该节点 IAM 角色的 `aws-auth` `ConfigMap` 中未列出条目（参见上一项），或者该角色不存在访问条目。要解决此问题，可以通过将以下命令中的 *`my-cluster`* 替换为集群名称，然后运行修改后的命令 `aws eks list-access-entries --cluster-name` *`my-cluster`* 来查看现有访问条目。以下命令将为节点的 IAM 角色添加访问条目。请将 *`111122223333`* 替换为 IAM 角色的 Amazon 账户 ID，将 *`myAmazonEKSNodeRole`* 替换为节点角色的名称。如果您有 Windows 节点，请将 *`EC2_LINUX`* 替换为 `EC2_Windows`。确保指定了节点 IAM 角色 ARN（非实例配置文件 ARN）。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws-cn:iam::111122223333:role/myAmazonEKSNodeRole --type EC2_LINUX
```

TLS 握手超时

当节点无法建立到公有 API 服务器端点的连接时，您可能会遇到类似如下的错误。

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error finding instance i-1111f2222f333e44c: "error listing Amazon instances: \"RequestError: send request failed\\ncaused by: Post  net/http: TLS handshake timeout\""
```

`kubelet` 进程将持续重新生成并测试 API 服务器终端节点。在控制面板中执行集群滚动更新（例如配置更改或版本更新）的任何过程中，也可能临时发生此错误。

要解决此问题，请检查路由表和安全组，以确保来自节点的流量可以到达公有端点。

InvalidClientTokenId

如果将服务账户的 IAM 角色用于部署到中国 Amazon 区域中集群的容器组（pod）或 DaemonSet，但尚未在规范中设置 `AWS_DEFAULT_REGION` 环境变量，则容器组（pod）或 DaemonSet 可能会收到以下错误：

```
An error occurred (InvalidClientTokenId) when calling the GetCallerIdentity operation: The security token included in the request is invalid
```

要解决此问题，您需要将 AWS_DEFAULT_REGION 环境变量添加到您的容器组（pod）或 DaemonSet 规范中，如以下示例容器组（pod）规范中所示。

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: AWS_DEFAULT_REGION
      value: "region-code"
```

在升级控制面板前，节点组必须匹配 Kubernetes 版本

集群中的托管节点和 Fargate 节点的 Kubernetes 次要版本必须与控制面板的当前版本相同，然后才能将控制面板更新为新的 Kubernetes 版本。Amazon EKS `update-cluster-version` API 会拒绝请求，直到您将所有 Amazon EKS 托管节点升级为当前集群版本。Amazon EKS 提供 API 来升级托管节点。有关升级托管节点组 Kubernetes 版本的信息，请参阅the section called “更新”。要升级 Fargate 节点的版本，请删除节点所表示的容器组（pod），然后在升级控制面板后重新部署容器组（pod）。有关更多信息，请参阅 the section called “更新 Kubernetes 版本”。

启动多个节点时，出现 **Too Many Requests** 错误

如果同时启动多个节点，您可能会在表示 `Too Many Requests` 的 Amazon EC2 用户数据执行日志中看到错误消息。发生这种情况的原因是控制面板被 `describeCluster` 调用过载。过载导致节流，节点无法运行引导脚本，节点无法完全加入集群。

确保 `--apiserver-endpoint`、`--b64-cluster-ca` 和 `--dns-cluster-ip` 参数正在传递给节点的引导脚本。加入这些参数后，引导脚本无需调用 `describeCluster`，这有助于防止控制面板过载。有关更多信息，请参阅 the section called “提供用户数据，将参数传递给随 Amazon EKS 优化版 Linux/Bottlerocket AMI 一起提供的 `bootstrap.sh` 文件”。

Kubernetes API 服务器请求的 HTTP 401 未经授权的错误响应

如果集群上容器组（pod）的服务账户令牌已到期，您就会看到这些错误。

Amazon EKS 集群的 Kubernetes API 服务器拒绝令牌超过 90 天的请求。在以前的 Kubernetes 版本中，令牌没有过期。这意味着依赖这些令牌的客户端必须在一小时内刷新它们。为防止 Kubernetes API 服务器因令牌无效而拒绝请求，工作负载使用的 Kubernetes 客户端开发工具包版本必须相同，或者高于以下版本：

- Go 版本 `0.15.7` 和更高版本
- Python 版本 `12.0.0` 和更高版本
- Java 版本 `9.0.0` 和更高版本
- JavaScript 版本 `0.10.3` 和更高版本
- Ruby `master` 分支
- Haskell 版本 `0.3.0.0`
- C# 版本 `7.0.5` 和更高版本

您可以识别使用过时令牌的集群中的所有现有容器组（pod）。有关更多信息，请参阅 the section called “服务账户令牌”。

Amazon EKS 平台版本比当前平台版本落后两个版本以上

当 Amazon EKS 无法自动更新集群的 platform-version 时，可能会发生这种情况。造成这种情况的原因有很多，而一些常见的原因如下。如果这些问题中的任何一个适用于您的集群，该集群可能仍然可以正常工作，但 Amazon EKS 不会更新其平台版本。

问题

集群 IAM 角色被删除 – 该角色是在创建集群时指定的。可使用以下命令查看所指定的角色。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-cluster --name my-cluster --query cluster.roleArn --output text | cut
 -d / -f 2
```

示例输出如下。

```
eksClusterRole
```

解决方案

创建具有相同名称的新集群 IAM 角色。

问题

集群创建期间指定的子网被删除 – 用于集群的子网是在集群创建期间指定的。可使用以下命令查看所指定的子网。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-cluster --name my-cluster --query cluster.resourcesVpcConfig.subnetIds
```

示例输出如下。

```
[
"subnet-EXAMPLE1",
"subnet-EXAMPLE2"
]
```

解决方案

确认您的账户中是否存在这些子网 ID。

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" --query
 "Subnets[*].SubnetId"
```

示例输出如下。

```
[
"subnet-EXAMPLE3",
"subnet-EXAMPLE4"
]
```

如果输出中返回的子网 ID 与创建集群时指定的子网 ID 不匹配，若您希望 Amazon EKS 更新集群，则需要更改集群使用的子网。这是因为，如果您在创建集群时指定了两个以上的子网，Amazon EKS 会随机选择您指定的子网，以便在其中创建新的弹性网络接口。这些网络接口使控制面板能够与您的节点进行通信。如果 Amazon EKS 选择的子网不存在，则不会更新集群。您在创建集群时指定了一些子网，但您无法控制 Amazon EKS 会选择哪个子网来创建新的网络接口。

当您对集群进行 Kubernetes 版本更新时，更新可能会因为同样的原因而失败。

问题

集群创建期间指定的安全组被删除 – 如果您在集群创建期间指定了安全组，则可以使用以下命令查看其 ID。将 *my-cluster* 替换为您的集群的名称。

```
aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.securityGroupIds
```

示例输出如下。

```
[
    "sg-EXAMPLE1"
]
```

如果返回 []，则在创建集群时未指定安全组，且缺少安全组也不是问题所在。如果返回安全组，请确认您的账户中存在这些安全组。

解决方案

确认您的账户中是否存在这些安全组。

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=$vpc_id" --query
 "SecurityGroups[*].GroupId"
```

示例输出如下。

```
[
"sg-EXAMPLE2"
]
```

如果输出中返回的安全组 ID 与创建集群时指定的安全组 ID 不匹配，若您希望 Amazon EKS 更新集群，则需要更改集群使用的安全组。如果创建集群时指定的安全组 ID 不存在，Amazon EKS 将不会更新集群。

当您对集群进行 Kubernetes 版本更新时，更新可能会因为同样的原因而失败。

- 在创建集群时指定的每个子网中，可用 IP 地址达不到至少 6 个（而我们建议 16 个）。如果子网中没有足够的可用 IP 地址，则需要释放子网中的 IP 地址，或者需要更改集群使用的子网，以使用具有足够可用 IP 地址的子网。
- 您在创建集群时启用了密钥加密，但您指定的 Amazon KMS 密钥已被删除。如果您希望 Amazon EKS 更新集群，则需要创建一个新集群

集群运行状况常见问题解答和错误代码以及解析路径

Amazon EKS 会检测 EKS 集群和集群基础设施存在的问题，并将其存储在 EKS 集群资源的运行状况对象中。借助集群运行状况信息，您可以更快地检测、排查并解决集群问题。这使您能够创建更安全、更新的应用程序环境。此外，由于必要的基础设施或集群配置存在问题，您可能无法升级到 Kubernetes 的较新版本或 Amazon EKS 无法在已降级的集群上安装安全更新。Amazon EKS 可能需要 3 小时来检测问题或检测问题是否已解决。

Amazon EKS 集群的运行状况由 Amazon EKS 及其用户共同负责。您负责管理 IAM 角色和 Amazon VPC 子网的必备基础设施，以及必须提前提供的其他必要基础设施。Amazon EKS 检测此基础设施和集群的配置发生的变化。

要通过 Amazon EKS 控制台访问集群的运行状况，请在 Amazon EKS 集群详细信息页面打开可观测性控制面板，然后在集群运行状况问题选项卡中查找名为运行状况问题的表。也可以通过在 EKS API 中调用 `DescribeCluster` 操作来获得这些数据，例如从 Amazon 命令行界面内部调用。

为什么应该使用此功能？

> 您可以更清楚地了解 Amazon EKS 集群的运行状况，快速诊断和修复任何问题，而无需花时间调试或提出 Amazon 支持案例。例如：您不小心删除了 Amazon EKS 集群的子网，Amazon EKS 将无法创建跨账户网络接口和 Kubernetes Amazon CLI 命令，例如 `kubectl exec` 或 `kubectl` 日志。这些将失败并显示错误：`Error from server: error dialing backend: remote error: tls: internal error.` 现在您将看到一个 Amazon EKS 运行状况问题，错误消息为：`subnet-da60e280 was deleted: could not create network interface`。

此功能如何与其它 Amazon 服务相关或结合使用？

> IAM 角色和 Amazon VPC 子网是集群运行状况检测到问题的两个必备基础设施示例。如果这些资源配置不正确，则此功能将返回详细信息。

存在运行状况问题的集群是否会产生费用？

> 是的，每个 Amazon EKS 集群都按标准的 Amazon EKS 定价计费。集群运行状况功能不收取额外费用。

此功能是否可以与 Amazon Outposts 上的 Amazon EKS 集群结合使用？

是的，已检测到 Amazon Cloud 中的 EKS 集群存在集群问题，包括 Amazon Outposts 上的扩展集群和 Amazon Outposts 上的本地集群。集群运行状况无法检测到 Amazon EKS Anywhere 或 Amazon EKS Distro（EKS-D）存在的问题。

当检测到新问题时，我能否收到通知？

是。Amazon 检测到新的集群运行状况问题时，会发送电子邮件和 Personal Health Dashboard 通知。

控制台是否会就运行状况问题向我发出警告？

是的，任何存在运行状况问题的集群都将在控制台顶部显示一个横幅。

前两列是 API 响应值所需的列。Health ClusterIssue 对象的第三个字段是 resourceIds，其返回值取决于问题类型。

代码	消息	ResourceIds	集群是否可恢复？
SUBNET_NOT_FOUND	我们找不到当前与您的集群关联的一个或多个子网。调用 Amazon EKS update-cluster-config API 来更新子网。	子网 ID	是
SECURITY_GROUP_NOT_FOUND	我们找不到当前与您的集群关联的一个或多个安全组。调用 Amazon EKS update-cluster-config API 来更新安全组	安全组 ID	是
IP_NOT_AVAILABLE	与集群关联的一个或多个子网没有足够的 IP 地址供 Amazon EKS 执行集群管理操作。使用 Amazon EKS update-cluster-	子网 ID	是

代码	消息	ResourceIds	集群是否可恢复？
	config API 释放子网中的地址，或者将不同的子网关联到您的集群。		
VPC_NOT_FOUND	我们找不到与您的集群关联的 VPC。您必须删除并重新创建集群。	VPC ID	否
ASSUME_ROLE_ACCESS_DENIED	您的集群未使用 Amazon EKS service-linked-role。我们无法分派与您的集群关联的角色来执行所需的 Amazon EKS 管理操作。检查该角色是否存在并且是否具有所需的信任策略。	集群 IAM 角色	是
PERMISSION_ACCESS_DENIED	您的集群未使用 Amazon EKS service-linked-role。与您的集群关联的角色未向 Amazon EKS 授予足够的权限来执行所需的管理操作。检查附加到集群角色的策略，以及是否应用了任何单独的拒绝策略。	集群 IAM 角色	是

代码	消息	ResourceIds	集群是否可恢复？
ASSUME_ROLE_ACCESS_ADCESS_DENIED_USING_SLR	我们无法分派 Amazon EKS 集群管理 service-linked-role。检查该角色是否存在并且是否具有所需的信任策略。	Amazon EKS service-linked-role	是
PERMISSION_ACCESS_DENIED_USING_SLR	Amazon EKS 集群管理 service-linked-role 未向 Amazon EKS 授予足够的权限来执行所需的管理操作。检查附加到集群角色的策略，以及是否应用了任何单独的拒绝策略。	Amazon EKS service-linked-role	是
OPT_IN_REQUIRED	您的账户没有订阅 Amazon EC2 服务。在您的账户设置页面更新您的账户订阅。	不适用	是
STS_REGIONAL_ENDPOINT_DISABLED	STS 区域端点已禁用。启用 Amazon EKS 的端点以执行所需的集群管理操作。	不适用	是
KMS_KEY_DISABLED	与您的集群关联的 Amazon KMS 密钥已禁用。重新启用密钥以恢复集群。	KMS 密钥 Arn	是
KMS_KEY_NOT_FOUND	我们找不到与您的集群关联的 Amazon KMS 密钥。您必须删除并重新创建集群。	KMS 密钥 ARN	否

代码	消息	ResourceIds	集群是否可恢复？
KMS_GRANT_REVOKED	已撤销对与您的集群关联的 Amazon KMS 密钥的授权。您必须删除并重新创建集群。	KMS 密钥 Arn	否

利用开源项目扩展 Amazon EKS 的功能

这些开源项目扩展了在Amazon上或在其外部运行的 Kubernetes 集群的功能，包括 Amazon EKS 管理的集群。

支持部署到 EKS 的软件

查看 Amazon EKS 文档时，您会在整个过程和示例中看到对各种开源工具和软件的引用。这些工具包括 Kubernetes Metrics Server 和 Cert Manager。

请注意，您选择部署的任何第三方或开源软件都不在您 Amazon Support 协议的范围内。使用 Kubernetes 的一个好处是可以加入活跃的开源社区。建议直接与相关的开源社区和项目维护者合作，为此类组件建立适当的支持渠道。有关更多信息，请参阅与云原生计算基金会（CNCF）相关的 Graduated and Incubating Projects。

Kubernetes 生态系统包括许多项目和组件，这些项目和组件具有不同级别的社区支持、响应时间和预期应用场景。与 EKS 一起实施这些技术时，要确保自己了解每个组件的支持矩阵。

Amazon 会维护我们集成到 EKS 控制面板中的开源组件。这包括我们全面的安全管道，涵盖我们分发的所有容器映像和二进制文件的版本验证、漏洞扫描、验证测试和补丁管理。例如，Amazon 负责 Kubernetes API 服务器。Kubernetes API 服务器在 Amazon EKS 服务水平协议的涵盖范围内。您可以使用 Amazon Web Services 支持计划来解决 Kubernetes API 服务器的问题，或者获取一般指导。

您需要仔细审查为各种 Amazon EKS 附加组件提供的支持。Amazon 附加组件是 Amazon 唯一完全支持的 Amazon EKS 附加组件类型。AmazonMarketplace 附加组件主要由 Amazon 合作伙伴提供支持。社区附加组件可从 Amazon 获得基本生命周期支持。有关更多信息，请参阅附加组件支持。

每个 EKS 附加组件，无论类型如何，都将获得 EKS 提供的基本生命周期支持，包括 Marketplace 附加组件。基本生命周期支持包括安装与卸载附加组件。有关可用的 Amazon EKS 附加组件类型和相关支持级别的更多信息，请参阅 Amazon EKS 附加组件的支持范围。要查看 Amazon 完全支持的附加组件，请参阅 Amazon Web Services 附加组件。

- 有关我们安全实践和支持范围的更多信息，请参阅 Amazon EKS 中的安全性。
- 有关通过 Amazon EKS 附加组件提供的社区和 Amazon Markertplace 附加组件的更多信息，请参阅 EKS 附加组件支持。

管理工具

Amazon EKS 和 Kubernetes 集群的相关管理工具。

eksctl

`eksctl` 是一个用于在 Amazon EKS 上创建集群的简单 CLI 工具。

- 项目 URL
- 项目文档
- Amazon开源博客：eksctl：使用一个命令的 Amazon EKS 集群

Amazon Controllers for Kubernetes

使用 Amazon Controllers for Kubernetes，您可以直接从 Kubernetes 集群创建和管理Amazon资源。

在 EKS 功能中提供。

- 项目 URL
- Amazon 开源博客：Kubernetes 的 Amazon 服务运算符现已推出

kro（Kube Resource Orchestrator）

kro 支持您创建自定义 Kubernetes API，将多个资源组合为更高级别的抽象资源。平台团队可以使用护栏定义可重复使用的模式，而应用程序团队则使用简单的高级别 API 来预调配和管理资源。

在 EKS 功能中提供。

- 项目 URL
- 项目文档

Argo CD

Argo CD 是一款适用于 Kubernetes 的声明式 GitOps 持续交付工具。它会持续监控您的 Git 存储库，并自动将更改同步到您的集群。

在 EKS 功能中提供。

- 项目 URL
- 项目文档

Flux CD

Flux 是一个工具，通过它，您可以使用 Git 来管理集群配置。它使用集群中的运算符来触发 Kubernetes 内部的部署。有关运算符的更多信息，请参阅 GitHub 上的 OperatorHub.io。

- 项目 URL
- 项目文档

CDK for Kubernetes

借助 CDK for Kubernetes (cdk8s)，您可以使用熟悉的编程语言定义 Kubernetes 应用程序和组件。cdk8s 应用程序合成为标准 Kubernetes 清单，可应用于任何 Kubernetes 集群。

- 项目 URL
- 项目文档
- Amazon 容器博客：推出 cdk8s+：面向 Kubernetes 对象的意图驱动 API

Networking

Amazon EKS 和 Kubernetes 集群的相关联网项目。

适用于 Kubernetes 的 Amazon VPC CNI 插件

Amazon EKS 支持通过适用于 Kubernetes 的 Amazon VPC CNI 插件进行的本机 VPC 联网。此插件会将 VPC 中的 IP 地址分配给每个容器组（pod）。

- 项目 URL
- 项目文档

Amazon适用于 Kubernetes 的负载均衡器控制器

Amazon负载均衡器控制器帮助管理 Kubernetes 集群的 Amazon Elastic Load Balancer。它通过配置 Amazon Application Load Balancer 来满足 Kubernetes 入口资源。该控制器通过预置 Amazon 网络负载均衡器来满足 Kubernetes 服务资源。

- 项目 URL
- 项目文档

ExternalDNS

ExternalDNS 将公开的 Kubernetes 服务和入口与 DNS 提供程序同步，包括 Amazon Route 53 和 Amazon服务发现。

- 项目 URL
- 项目文档

Machine Learning

Amazon EKS 和 Kubernetes 集群的相关 Machine Learning 项目。

Kubeflow

Kubernetes 的机器学习工具包。

- 项目 URL
- 项目文档
- Amazon 开源博客：Amazon EKS 上的 Kubeflow

Auto Scaling

Amazon EKS 和 Kubernetes 集群的相关 Auto Scaling 项目。

Cluster Autoscaler

Cluster Autoscaler 是一个根据 CPU 和内存压力自动调整 Kubernetes 集群大小的工具。

- 项目 URL
- 项目文档
- Amazon EKS 讲习会：Cluster Autoscaler

Karpenter

Karpenter 是一款专为提高灵活性、性能和简单性而设计的 Kubernetes 节点自动扩缩器。

- 项目 URL
- 项目文档
- Amazon EKS 讲习会：Karpenter

Escalator

Escalator 是 Kubernetes 的优化了批处理或作业的水平 Autoscaler。

- 项目 URL
- 项目文档

监控

Amazon EKS 和 Kubernetes 集群的相关监控项目。

Prometheus

Prometheus 是一个开源系统监控和警报工具包。

- 项目 URL
- 项目文档
- Amazon EKS 研讨会：https://eksworkshop.com/intermediate/240_monitoring/

持续集成/持续部署

Amazon EKS 和 Kubernetes 集群的相关 CI/CD 必要项目。

Jenkins X

Amazon EKS 和 Kubernetes 集群上现代云应用程序的 CI/CD 解决方案。

- 项目 URL
- 项目文档

了解 Amazon EKS 的新增功能和路线图

您可以通过滚动到 Amazon 新功能页面上的“新功能”源，了解 Amazon EKS 新功能。您也可以在 GitHub 上查看路线图，通过它了解即将推出的功能和优先工作，从而规划未来使用 Amazon EKS 的方式。您可以直接向我们提供有关路线图优先工作的反馈。

文档历史记录

下表描述了《Amazon EKS 用户指南》的部分主要更新和新功能。要在该表格有新条目时收到通知，您可以使用 RSS 阅读器订阅以下 URL：

```
https://docs.aws.amazon.com/eks/latest/userguide/doc-history.rss
```

变更	说明	日期
Amazon EKS 功能	Amazon EKS 现已通过 EKS 功能提供完全托管的集群功能，支持基于 Argo CD 的声明式持续部署，支持借助 Kubernetes Amazon 控制器实现 Amazon 资源管理，同时支持依托 Kube Resource Orchestrator 完成 Kubernetes 资源的组合编排。	2025 年 11 月 30 日
新 Amazon 托管式策略	Amazon EKS 已发布一项全新的托管策略（`AmazonEKSMCPReadOnlyAccess`），用于在 Amazon EKS MCP 服务器中启用只读工具，实现可观测性和问题排查功能。有关更多信息，请参阅 Amazon EKS 对 Amazon 托管策略的更新。	2025 年 11 月 21 日
网络可观测性	Amazon EKS 现已提供增强型容器网络可观测性能力，具备性能监控与工作负载流量可视化功能。	2025 年 11 月 19 日
Amazon 托管策略更新	已为 `AmazonEBSCSIDriverPolicy` 新增 `ec2:CopyV`	2025 年 11 月 17 日

	olumes 权限，支持 EBS CSI 驱动程序直接复制 EBS 卷。	
Amazon 托管策略更新	已为 AmazonEKSServiceRolePolicy 新增 ec2:DescribeRouteTables 和 ec2:DescribeNetworkAcls 权限。作为集群洞察功能的一部分，此项能力支持 Amazon EKS 检测混合节点的 VPC 路由表与网络 ACL 的配置问题。	2025 年 10 月 22 日
服务相关角色更新	已为 AmazonEKSConnectorServiceRolePolicy 新增 ssmmessages:OpenDataChannel 权限	2025 年 10 月 15 日
Kubernetes 1.34 版	增加了新集群和版本升级的 Kubernetes 1.34 版支持。	2025 年 10 月 2 日
可配置节点自动修复	Amazon EKS 现已支持对节点自动修复行为实施更精细化的管控。	2025 年 9 月 22 日
刷新集群见解	您现在可以手动刷新集群见解。	2025 年 8 月 27 日
Amazon 托管策略更新	已将权限添加至 AmazonEKSServiceRolePolicy 。此角色可以附加新的访问策略 AmazonEKSEventPolicy 。ec2:DeleteLaunchTemplate 和 ec2:TerminateInstances 的权限受限。	2025 年 8 月 26 日

防止跨服务混淆代理	添加了一个主题，其中包含您可以应用于防止跨服务混淆代理的策略信任示例。Amazon EKS 接受 EKS 集群角色的信任策略中的 `aws:SourceArn` 和 `aws:SourceAccount` 条件。	2025 年 8 月 19 日
Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这包括 Kubernetes `1.33.2`、`1.32.6`、`1.31.10` 和 `1.30.14` 的新补丁版本。	2025 年 7 月 30 日
适用于多宿主容器组（pod）的 VPC CNI 多 NIC 功能	Amazon EKS 向 VPC CNI 添加了多宿主容器组（pod）。现在，您可以配置工作负载，让 VPC CNI 将来自 EC2 实例上每个 NIC 的 IP 地址分配给每个容器组（pod）。应用程序可以建立并发连接，以使用每个 NIC 的带宽。每个网络接口都配置在与节点相同的子网和安全组中。以前，您需要使用 Multus CNI 运行多个其他 CNI 来创建多宿主容器组（pod）。执行此操作的文档和步骤已移至 eks/latest/userguide/pod-multus.html。	2025 年 7 月 15 日
VPC CNI 故障排除内容更新	扩展了 VPC CNI 中 Kubernetes 网络策略的故障排除页面。添加了 CRD 和 RBAC 权限以及 12 个已知问题。	2025 年 6 月 30 日

Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这些平台版本中没有任何新的 Kubernetes 补丁版本。	2025 年 6 月 26 日
Amazon 托管策略更新	已将 `ssmmessages:OpenDataChannel` 权限添加至 `AmazonEKSLocalOutpostServiceRolePolicy` 。	2025 年 6 月 26 日
Amazon 托管策略更新	添加了 `AmazonEKSServiceRolePolicy` 和 `AmazonEKSComputePolicy` 的权限以允许 Amazon EKS 自动模式使用您账户中的 EC2 按需容量预留来启动实例。此外，添加了 `AmazonEKSComputePolicy` 的权限，允许 Amazon EKS 代表您创建 EC2 竞价型实例服务相关角色。	2025 年 6 月 20 日
托管策略更新	添加了 `AmazonEKSDashboardConsoleReadOnly` 政策。	2025 年 6 月 19 日
有关 `NodeClass` 的 Amazon EKS 自动模式更新	自动模式节点的 `NodeClass` 模板为单独的容器组（pod）子网添加了配置。这将添加可选键 `podSubnetSelectorTerms` 和 `podSecurityGroupSelectorTerms` ，以便为容器组（pod）设置子网和安全组。	2025 年 6 月 13 日

使用 EKS 容器组身份定位辅助角色和跨账户角色	Amazon EKS 向 EKS 容器组身份添加目标 IAM 角色，实现自动角色链接。可以使用此功能自动担任另一个账户中的角色，EKS 容器组身份会轮换临时凭证。每个容器组身份关联必须先在同一个账户中拥有一个 IAM 角色进行担任，然后使用该角色担任目标角色。	2025 年 6 月 11 日
Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这些平台版本中没有任何新的 Kubernetes 补丁版本。	2025 年 6 月 11 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在亚太地区（台北）(ap-east-2) Amazon 区域推出。	2025 年 6 月 6 日
新 IPv6 集群双栈公共端点的 IPv6 访问控制	Amazon EKS 添加了 IPv6 CIDR 块，用于控制对新 IPv6 集群的公有集群端点的访问。以前，您只能添加 IPv4 CIDR 块以允许流量流向公有集群端点，即使是双栈集群端点也是如此。	2025 年 6 月 5 日
Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这包括 Kubernetes 1.32.5、1.31.9 和 1.30.13 的新补丁版本。	2025 年 5 月 30 日

EKS 混合节点的新集群见解	Amazon EKS 添加了新的集群见解，用于检查混合节点的配置。这些见解检查将警告您本地节点和容器组（pod）以及集群的远程网络配置存在问题。	2025 年 5 月 29 日
Kubernetes 1.33 版	增加了新集群和版本升级的 Kubernetes 1.33 版支持。	2025 年 5 月 29 日
对 Amazon FSx CSI 驱动程序的附加组件支持	您现在可以使用 Amazon 管理控制台、Amazon CLI 和 API 来管理 Amazon FSx CSI 驱动程序。	2025 年 5 月 23 日
在控制台中编辑 Prometheus 抓取程序	现在，您可以在 Amazon EKS 控制台中编辑 Amazon Managed Service for Prometheus 抓取程序。	2025 年 5 月 22 日
托管策略更新	添加了 AmazonEKS DashboardServiceRolePolicy 政策。	2025 年 5 月 21 日
Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这些平台版本中没有任何新的 Kubernetes 补丁版本。	2025 年 5 月 16 日
有关适用于 Lustre 的 Amazon FSx 性能的新页面	增加了新主题，包含有关适用于 Lustre 的 Amazon FSx 性能优化的详细信息。	2025 年 5 月 2 日
有关 NodeClass 的 Amazon EKS 自动模式更新	自动模式节点的 NodeClass 模板增加了有关转发网络代理的配置。这将添加可选的 advancedNetworking 键来设置 HTTPS 代理。	2025 年 4 月 30 日

用于混合节点的 Bottlerocket	Bottlerocket 现在支持 EKS 混合节点功能。	2025 年 4 月 29 日
Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这些平台版本中没有任何新的 Kubernetes 补丁版本。	2025 年 4 月 29 日
有关混合联网的新概念页面	添加了关于 EKS 混合节点概念的页面。这些页面以图表形式详细介绍了本地和云网络。	2025 年 4 月 18 日
Amazon 托管策略更新	添加了 `AmazonEKS ClusterPolicy` 权限，允许 Amazon EKS 访问由 VPC CNI 创建的弹性网络接口。这是必需项，以便 EKS 可以清理 VPC CNI 意外退出时留下的弹性网络接口。	2025 年 4 月 16 日
Amazon 托管策略更新	添加了 `AmazonEKS ServiceRolePolicy` 权限，允许 EKS AI/ML 客户向默认 EKS 集群安全组添加出口规则。	2025 年 4 月 14 日
EKS 混合节点的节点运行状况	从 `1.2.0-eksbuild.1` 版本开始，您可以在混合节点上使用 `eks-node-monitoring-agent` 。将 `eks-node-monitoring-agent` 作为 Amazon EKS 附加组件运行，从而检测和显示运行状况问题。	2025 年 3 月 31 日

适用于现有集群的 EKS 混合节点	现在，您可以添加、更改或移除现有集群的混合节点配置。以前，只有在创建新集群时才能将混合节点配置添加到新集群。借助 Amazon EKS 混合节点功能，您可以将本地和边缘基础设施用作 Amazon EKS 集群中的节点。Amazon 负责管理由 Amazon 托管的 Amazon EKS 集群的 Kubernetes 控制面板，而您负责管理在本地或边缘环境中运行的混合节点。	2025 年 3 月 31 日
回滚：利用集群见解防止意外升级	Amazon EKS 暂时回滚了一项功能，该功能要求您在遇到某些集群见解问题时使用 `--force` 标志升级集群。有关更多信息，请参阅 GitHub 上的 Temporary rollback of enforcing upgrade insights on update cluster version。	2025 年 3 月 28 日
Bottlerocket FIPS AMI	Bottlerocket FIPS AMI 现在可在标准托管节点组中使用。	2025 年 3 月 27 日
Amazon 托管策略更新	为 `AmazonEKSServiceRolePolicy` 增加了允许 Amazon EKS 终止由自动模式创建的 EC2 实例的权限。为 `AmazonEKSServiceRolePolicy` 增加了允许 Amazon EKS 终止由自动模式创建的 EC2 实例的权限。	2025 年 2 月 28 日

托管节点组的更新策略	现在可以使用更新策略来配置托管节点组的版本更新过程。其中引入了可在创建新节点之前终止节点的最低更新策略，该策略在容量受限的环境中很有用。默认更新策略会延续现有行为。	2025 年 1 月 27 日
Kubernetes 1.32 版	增加了新集群和版本升级的 Kubernetes 1.32 版支持。	2025 年 1 月 23 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在亚太地区（泰国）(ap-southeast-7 ）和墨西哥（中部）(mx-central-1 ）Amazon 区域推出。EKS 自动模式和适用于 EKS API 的 VPC 端点在这两个区域都未开放。	2025 年 1 月 14 日
Amazon 托管策略更新	为 AmazonEBSCSIDriverPolicy 增加了多项权限，以允许 Amazon EBS CSI 驱动程序还原所有快照、在 EBS 卷上启用快速快照还原（FSR）以及修改卷上的标签。	2025 年 1 月 13 日
Amazon 托管策略更新	已将权限添加至 AmazonEKSLoadBalancingPolicy 。	2024 年 12 月 26 日

更新了集群见解功能	Amazon EKS 升级见解功能现在将就更多集群运行状况和版本兼容性问题发出警告。可以检测不同 Kubernetes 和 Amazon EKS 组件之间的问题，例如 `kubelet`、`kube-proxy` 以及 Amazon EKS 附加组件。	2024 年 12 月 20 日
节点监控代理和自动修复功能	可以将新的 `eks-node-monitoring-agent` 作为 Amazon EKS 附加组件使用，从而检测和显示运行状况问题。您还可以启用节点自动修复功能，以便在检测到问题时自动替换节点。	2024 年 12 月 16 日
Amazon EKS 混合节点功能	您现在可以在本地运行连接到 Amazon EKS 集群的节点。借助 Amazon EKS 混合节点功能，您可以将本地和边缘基础设施用作 Amazon EKS 集群中的节点。Amazon 负责管理由 Amazon 托管的 Amazon EKS 集群的 Kubernetes 控制面板，而您负责管理在本地或边缘环境中运行的混合节点。	2024 年 12 月 1 日
Amazon EKS 自动模式	Amazon EKS 自动模式可完全自动管理 Amazon 上有关计算、存储和联网的 Kubernetes 集群基础设施。通过自动预置基础设施、选择最佳计算实例、动态扩展资源、持续优化成本、修补操作系统以及与 Amazon 安全服务集成，简化了 Kubernetes 的管理。	2024 年 12 月 1 日

Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这包括 Kubernetes 1.31.2、1.30.6、1.29.10 和 1.28.15 的新补丁版本。	2024 年 11 月 22 日
Amazon 托管策略更新	为兼容中国区域，更新了 AWSServiceRoleForAmazonEKSNodegroup 。	2024 年 11 月 22 日
Kubernetes 1.30 版现在可用于 Amazon Outposts 上的本地集群	现在可以使用 Kubernetes 1.30 版在 Amazon Outposts 上创建 Amazon EKS 本地集群。	2024 年 11 月 21 日
Amazon 托管策略更新	EKS 更新了 Amazon 托管策略 AmazonEKSLocalOutpostClusterPolicy 。添加了 ec2:DescribeAvailabilityZones 权限，以便集群控制面板上的 Amazon 云控制器管理器可以识别各个节点所在的可用区。	2024 年 11 月 21 日
使用 FIPS 140-3 的 Bottlerocket AMI	预配置为使用经过 FIPS 140-3 验证的加密模块的 Bottlerocket AMI 已可供使用。这包括 Amazon Linux 2023 内核加密 API 加密模块和 Amazon-LC 加密模块。	2024 年 11 月 20 日

Amazon 托管策略更新	更新了 AWSServiceRoleForAmazonEKSNodegroup 策略，允许 ec2:RebootInstances ，以实现由 Amazon EKS 托管节点组创建的实例。限制了 Amazon EC2 资源的 ec2:CreateTags 权限。	2024 年 11 月 20 日
可观测性仪表板	可观测性仪表板可帮助您进行快速检测、排除故障和修复问题。AWS/EKS 命名空间中还提供了新的 CloudWatch 提供的指标。	2024 年 11 月 18 日
Amazon 托管策略更新	EKS 更新了 Amazon 托管策略 AmazonEKSServiceRolePolicy 。添加了 EKS 访问策略、负载均衡器管理和自动集群资源清理的权限。	2024 年 11 月 16 日
在控制台中为支持 EKS 容器组身份的附加组件进行的新角色创建	在使用控制台创建或更新支持 EKS 容器组身份的附加组件时，存在部分新步骤，以供自动生成具有相应名称、角色策略和信任策略的 IAM 角色。	2024 年 11 月 15 日
Amazon Local Zones 中的托管节点组	托管节点组现已可在 Amazon Local Zones 中创建。	2024 年 11 月 15 日
新指标现已可用	API 组下的新指标现已可用 (metrics.eks.amazonaws.com)。	2024 年 11 月 11 日

Amazon 托管策略更新	EKS 更新了 Amazon 托管策略 AmazonEKSComputePolicy 。更新了 iam:AddRoleToInstanceProfile 操作的资源权限。	2024 年 11 月 7 日
Amazon 托管策略更新	EKS 添加了新的 Amazon 托管策略：AmazonEKSComputePolicy	2024 年 11 月 1 日
Amazon 托管策略更新	已将权限添加至 AmazonEKSClusterPolicy 。添加了 ec2:DescribeInstanceTopology 权限，允许 Amazon EKS 将拓扑信息作为标签附加到节点。	2024 年 11 月 1 日
Amazon 托管策略更新	EKS 添加了新的 Amazon 托管策略：AmazonEKSBlockStoragePolicy	2024 年 10 月 30 日
Amazon 托管策略更新	EKS 添加了新的 Amazon 托管策略：AmazonEKSLoadBalancingPolicy	2024 年 10 月 30 日
Amazon 托管策略更新	向 AmazonEKSServiceRolePolicy 添加了 cloudwatch:PutMetricData 权限，允许 Amazon EKS 向 Amazon CloudWatch 发布指标。	2024 年 10 月 29 日
Amazon 托管策略更新	EKS 添加了新的 Amazon 托管策略：AmazonEKSNetworkingPolicy	2024 年 10 月 28 日

新 IPv6 集群的双堆栈端点	使用双堆栈 eks-cluster. *region*.api.aws 端点连接到新的 IPv6 集群。当您描述这些集群时，将返回此端点。IPv4、IPv6 或双堆栈环境中的 kubectl 和其它 Kubernetes API 客户端可以解析并连接到公有或私有集群的这些端点。	2024 年 10 月 21 日
Amazon 托管策略更新	为中国区域向 AWSServiceRoleForAmazonEKSNodegroup 添加了 autoscaling:ResumeProcesses 、autoscaling:SuspendProcesses 和相关权限，可与适用于 EKS 的 Amazon 应用程序恢复控制器集成。其它地区没有变化。	2024 年 10 月 21 日
AL2023 加速了 AMI	现在，您可以将加速的 NVIDIA 和 Amazon Neuron 实例用于基于 AL2023 的 AMI。	2024 年 10 月 11 日
新的源格式	我们已切换到新的源格式，并进行了一些布局更改。我们正在解决一些暂时的小格式问题。	2024 年 10 月 10 日
Amazon 托管策略更新	添加了对 AmazonEKSServicePolicy 和 AmazonEKSServiceRolePolicy 的权限。添加了 ec2:GetSecurityGroupsForVpc 和相关的标签权限，以允许 EKS 读取安全组信息和更新相关标签。	2024 年 10 月 10 日

Amazon 托管式策略更新 - 新策略	EKS 添加了新的 Amazon 托管策略。	2024 年 10 月 3 日
Kubernetes 1.31 版	增加了新集群和版本升级的 Kubernetes 1.31 版支持。	2024 年 9 月 24 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2024 年 8 月 21 日
Kubernetes 1.29 版现在可用于 Amazon Outposts 上的本地集群	现在可以使用 Kubernetes 1.29 版在 Amazon Outposts 上创建 Amazon EKS 本地集群。	2024 年 8 月 20 日
Amazon GovCloud 中的 EKS 容器组身份（美国）	Amazon EKS 容器组身份将 IAM 角色与 Kubernetes 服务账户关联。借助此功能，您不再需要向节点 IAM 角色提供扩展的权限。这样，该节点上的容器组（pod）就可以调用 Amazon API 了。与服务账户的 IAM 角色不同，EKS 容器组身份完全在 EKS 内部；不需要 OIDC 身份提供者。	2024 年 8 月 14 日
场景驱动的内容更新	我们重命名和更新了整个指南中的主题，使其更加以场景为中心。	2024 年 8 月 9 日
适用于 Amazon EKS 的双栈 VPC 接口端点	现在，您可以为 Amazon EKS 创建同时具有 IPv4 和 IPv6 IP 地址和 DNS 名称的双栈 VPC 接口端点。	2024 年 8 月 7 日

适用于 Amazon EKS API 且具有 IPv6 地址的新双栈端点	用于创建和管理集群的 EKS API 以及集群的 OIDC 颁发者 URL 具有新的双栈端点。Amazon EKS API 的新 DNS 名称为 `eks.region.api.aws`，可以解析为 IPv4 地址和 IPv6 地址。新集群具有新的双栈 OIDC 颁发者 URL (`oidc-eks. region.api.aws`)。	2024 年 8 月 1 日
适用于托管式节点组的容量块	您现在可以将容量块用于托管式节点组。	2024 年 7 月 1 日
自动扩缩组指标收集功能会默认启用	现在，Amazon EKS 托管式节点组会默认启用 Amazon EC2 Auto Scaling 组指标，无需额外付费。以前，您必须执行多个步骤才能启用此功能。	2024 年 6 月 28 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2024 年 6 月 27 日
Kubernetes 1.26 版	Kubernetes 1.26 版现已提供延期支持。	2024 年 6 月 12 日
有关 AMI 信息参考的改进	我们改进了 AMI 信息参考，尤其是 Bottlerocket AMI 信息参考。	2024 年 6 月 12 日
Kubernetes 1.30 版	增加了新集群和版本升级的 Kubernetes 1.30 版支持。	2024 年 5 月 23 日

CoreDNS 自动扩缩	CoreDNS 自动扩缩器将根据节点数量和 CPU 核心数量，动态调整 EKS 集群中 CoreDNS 部署的副本数量。此功能适用于 CoreDNS v1.9 和 EKS 1.25 发行版及更高版本的最新平台版本。	2024 年 5 月 14 日
Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这包括 Kubernetes 1.29.4、1.28.9 和 1.27.13 的新补丁版本。	2024 年 5 月 14 日
Windows 的 CloudWatch Container Insights 支持	Amazon CloudWatch Observability Operator 附加组件现在还支持在集群中的 Windows Worker 节点上使用 Container Insights。	2024 年 4 月 10 日
Kubernetes 概念	添加了新的 Kubernetes 概念主题。	2024 年 4 月 5 日
重构访问权限和 IAM 内容	将与访问权限和 IAM 主题相关的现有页面（例如身份验证配置映射、访问条目、容器组（pod）ID 和 IRSA）移至新部分。修订概述内容。	2024 年 4 月 2 日
Amazon S3 CSI 驱动程序的 Bottlerocket 操作系统支持	适用于 Amazon S3 的 Mountpoint CSI 驱动程序现在与 Bottlerocket 兼容。	2024 年 3 月 13 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2024 年 3 月 4 日

Amazon Linux 2023	Amazon Linux 2023（AL2023）是一款新的基于 Linux 的操作系统，旨在为您的云应用程序提供安全、稳定和高性能的环境。	2024 年 2 月 29 日
EKS 容器组身份和 IRSA 支持 Kubernetes 1.29 中的边车容器	在 Kubernetes 1.29 中，Amazon EKS 集群中提供边车容器。服务账户或 EKS 容器组身份的 IAM 角色不支持附加容器。有关边车容器的更多信息，请参阅 Kubernetes 文档中的边车容器。	2024 年 2 月 26 日
Kubernetes 1.29 版	增加了新集群和版本升级的 Kubernetes 1.29 版支持。	2024 年 1 月 23 日
完整版：Kubernetes 版本的 Amazon EKS 延期支持	Kubernetes 版本延期支持允许在特定 Kubernetes 版本上停留超过 14 个月。	2024 年 1 月 16 日
Amazon 云中的 Amazon EKS 集群运行状况检测	Amazon EKS 会检测到您的 Amazon EKS 集群存在的问题，以及集群运行状况中集群先决条件的基础设施存在的问题。您可以在 Amazon Web Services 管理控制台 中以及 EKS API 中集群的 `health` 中查看您的 EKS 集群的问题。除了控制台检测到并显示的问题之外，还有这些问题。以前，集群运行状况仅适用于 Amazon Outposts 上的本地集群。	2023 年 12 月 28 日
集群见解	现在，您可以根据定期检查获得有关集群的建议。	2023 年 12 月 20 日

Amazon EKS Amazon 区域扩展	Amazon EKS 现已在加拿大西部（卡尔加里）(ca-west-1) Amazon 区域推出。	2023 年 12 月 20 日
现在，您可以使用访问条目授予 IAM 角色和用户访问您的集群的权限	在引入访问条目之前，您通过向 aws-auth ConfigMap 中添加条目来授予 IAM 角色和用户访问您的集群的权限。现在，每个集群都有访问模式，您可以切换到按计划使用访问条目。切换模式后，您可以通过在 Amazon CLI、Amazon CloudFormation 和 Amazon SDK 中添加访问条目来添加用户。	2023 年 12 月 18 日
Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这包括 Kubernetes 1.28.4、1.27.8、1.26.11 和 1.25.16 的新补丁版本。	2023 年 12 月 12 日
适用于 Amazon S3 的 Mountpoint CSI 驱动程序	现在可在 Amazon EKS 集群上安装适用于 Amazon S3 的 Mountpoint CSI 驱动程序。	2023 年 11 月 27 日
创建集群时开启 Prometheus 指标	在 Amazon Web Services 管理控制台 中，您现在可在创建集群时开启 Prometheus 指标。您还可以在可观测性选项卡中查看 Prometheus 抓取程序详细信息。	2023 年 11 月 26 日

Amazon EKS 容器组身份	Amazon EKS 容器组身份将 IAM 角色与 Kubernetes 服务账户关联。借助此功能，您不再需要向节点 IAM 角色提供扩展的权限。这样，该节点上的容器组（pod）就可以调用 Amazon API 了。与服务账户的 IAM 角色不同，EKS 容器组身份完全在 EKS 内部；不需要 OIDC 身份提供者。	2023 年 11 月 26 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2023 年 11 月 26 日
CSI 快照控制器	现在，您可以安装 CSI 快照控制器，以便与兼容的 CSI 驱动程序一起使用，例如 Amazon EBS CSI 驱动程序。	2023 年 11 月 17 日
ADOT Operator 主题重写	在适用于 OpenTelemetry 的 Amazon Distro 文档中，适用于 ADOT Operator 的 Amazon EKS 附加组件支持部分是多余的。我们将剩余的重要信息迁移到该资源，以减少过时和不一致的信息。	2023 年 11 月 14 日
Prometheus 指标的 CoreDNS EKS 附加组件支持	适用于 CoreDNS 的 EKS 附加组件版本 `v1.10.1-eksbuild.5` 、`v1.9.3-eksbuild.9` 和 `v1.8.7-eksbuild.8` 在 `kube-dns` 服务中公开了端口，以便 CoreDNS 向其发布指标。这样可以轻松将 CoreDNS 指标包含在监控系统中。	2023 年 11 月 10 日

Amazon EKS CloudWatch Observability Operator 插件	添加了 Amazon EKS CloudWatch Observability Operator 页面。	2023 年 11 月 6 日
美国东部（俄亥俄州）自行管理 P5 实例的容量块	在美国东部（俄亥俄州），您现在可以将容量块用于自我管理 P5 实例。	2023 年 10 月 31 日
集群支持修改子网和安全组	您可以更新集群以更改该集群使用的子网和安全组。您可以从 Amazon Web Services 管理控制台、Amazon CLI 的最新版本、Amazon CloudFormation 以及 `eksctl` 版本 `v0.164.0-rc.0` 或更高版本进行更新。您可能需要执行此操作，为子网提供更多可用 IP 地址，以便成功升级集群版本。	2023 年 10 月 24 日
集群角色和托管节点组角色支持客户管理型 Amazon Identity and Access Management 策略	您可以对集群角色使用自定义 IAM 策略，而不是 AmazonEKSClusterPolicy Amazon 托管策略。此外，您还可以对托管节点组中的节点角色使用自定义 IAM 策略，而不是 AmazonEKSWorkerNodePolicy Amazon 托管策略。这样做是为了创建具有最低权限的策略，以满足严格的合规性要求。	2023 年 10 月 23 日
修复 eksctl 安装链接	在移动页面后修复 eksctl 的安装链接。	2023 年 10 月 6 日

预览版：Kubernetes 版本的 Amazon EKS 延期支持	Kubernetes 版本延期支持允许在特定 Kubernetes 版本上停留超过 14 个月。	2023 年 10 月 4 日
删除对 Amazon App Mesh 集成的引用	Amazon EKS 与 Amazon App Mesh 的集成仅适用于 App Mesh 的现有客户。	2023 年 9 月 29 日
Kubernetes 1.28 版	增加了新集群和版本升级的 Kubernetes 1.28 版支持。	2023 年 9 月 26 日
现有集群支持在适用于 Kubernetes 的 Amazon VPC CNI 插件中强制执行 Kubernetes 网络策略	您可以将现有集群中的 Kubernetes 网络策略与适用于 Kubernetes 的 Amazon VPC CNI 插件结合使用，不必采用第三方解决方案。您可以将现有集群中的 Kubernetes 网络策略与适用于 Kubernetes 的 Amazon VPC CNI 插件结合使用，不必采用第三方解决方案。	2023 年 9 月 15 日
CoreDNS Amazon EKS 附加组件支持修改 PDB	在 v1.9.3-eksbuild.7 版及更高版本和 v1.10.1-eksbuild.4 版及更高版本中，您可以修改 CoreDNS EKS 附加组件的 PodDisruptionBudget 。	2023 年 9 月 15 日
Amazon EKS 对共享子网的支持	关于在共享子网中创建 Amazon EKS 集群的最新共享子网要求和注意事项。	2023 年 9 月 7 日
更新了“什么是 Amazon EKS？”	新增了常见用例和架构主题。更新了其他主题。	2023 年 9 月 6 日

在适用于 Kubernetes 的 Amazon VPC CNI 插件中强制执行 Kubernetes 网络策略	您可以将 Kubernetes 网络策略与适用于 Kubernetes 的 Amazon VPC CNI 插件结合使用，不必采用第三方解决方案。您可以将 Kubernetes 网络策略与适用于 Kubernetes 的 Amazon VPC CNI 插件结合使用，不必采用第三方解决方案。	2023 年 8 月 29 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在以色列（特拉维夫）(`il-central-1`) Amazon 区域推出。	2023 年 8 月 1 日
适用于 Fargate 的可配置临时存储	您可以增加在 Amazon EKS Fargate 上运行的每个容器组（pod）的临时存储总量。	2023 年 7 月 31 日
对 Amazon EFS CSI 驱动程序的附加组件支持	您现在可以使用 Amazon Web Services 管理控制台、Amazon CLI 和 API 来管理 Amazon EFS CSI 驱动程序。	2023 年 7 月 26 日
Amazon 托管策略更新：新策略	Amazon EKS 添加了新的 Amazon 托管策略。	2023 年 7 月 26 日
Kubernetes 的 1.27、1.26、1.25 和 1.24 版本更新现在可用于 Amazon Outposts 上的本地集群	Kubernetes 的 1.27.3、1.26.6、1.25.11 和 1.24.15 版本更新现在可用于 Amazon Outposts 上的本地集群	2023 年 7 月 20 日
Windows 节点的 IP 前缀支持	为节点分配 IP 前缀可令在节点上托管的容器组（pod）数量比为节点分配单个辅助 IP 地址时多得多。	2023 年 7 月 6 日

Amazon FSx for OpenZFS CSI 驱动程序	现在，您可以在 Amazon EKS 集群上安装适用于 OpenZFS CSI 驱动程序的 Amazon FSx。	2023 年 6 月 30 日
IPv4 集群中 Linux 节点上的容器组（pod）现在可以与 IPv6 端点通信。	为节点分配 IPv6 地址后，容器组（pod）的 IPv4 地址是转换为其所运行节点 IPv6 地址的网络地址。	2023 年 6 月 19 日
Amazon GovCloud（美国）区域中的 Windows 托管节点组	在 Amazon GovCloud（美国）区域中，Amazon EKS 托管节点组现在可以运行 Windows 容器。	2023 年 5 月 30 日
Kubernetes 1.27 版	增加了新集群和版本升级的 Kubernetes 1.27 版支持。	2023 年 5 月 24 日
Kubernetes 1.26 版	增加了新集群和版本升级的 Kubernetes 1.26 版支持。	2023 年 4 月 11 日
无域 gMSA	现在可将无域 gMSA 与容器组（pod）结合使用了。	2023 年 3 月 27 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在亚太地区（墨尔本）(ap-southeast-4) Amazon 区域推出。	2023 年 3 月 10 日
Amazon File Cache CSI 驱动程序	现在，您可以在 Amazon EKS 集群上安装 Amazon File Cache CSI 驱动程序。	2023 年 3 月 3 日
Kubernetes 1.25 版现在可用于 Amazon Outposts 上的本地集群	现在，您可以使用 Kubernetes 1.22 到 1.25 版在 Outpost 上创建 Amazon EKS 本地集群。	2023 年 3 月 1 日
Kubernetes 1.25 版	增加了新集群和版本升级的 Kubernetes 1.25 版支持。	2023 年 2 月 22 日

Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2023 年 2 月 7 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在亚太地区（海得拉巴）(ap-south-2)、欧洲（苏黎世）(eu-central-2)和欧洲（西班牙）(eu-south-2) Amazon 区域推出。	2023 年 2 月 6 日
Kubernetes 1.21 到 1.24 版现在可用于 Amazon Outposts 上的本地集群。	现在，您可以使用 Kubernetes 1.21 到 1.24 版在 Outpost 上创建 Amazon EKS 本地集群。以前，只有版本 1.21 可用。	2023 年 1 月 17 日
Amazon EKS 现在支持 Amazon PrivateLink	您可以使用 Amazon PrivateLink 在您的 VPC 和 Amazon EKS 之间创建私有连接。	2022 年 12 月 16 日
托管节点组 Windows 支持	您现在可以将 Windows 用于 Amazon EKS 托管节点组。	2022 年 12 月 15 日
来自独立软件供应商的 Amazon EKS 附加组件现已在 Amazon Marketplace 中提供	现在，您可以通过 Amazon Marketplace 浏览和订阅独立软件供应商提供的 Amazon EKS 附加组件。	2022 年 11 月 28 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2022 年 11 月 17 日
Kubernetes 1.24 版	增加了新集群和版本升级的 Kubernetes 1.24 版支持。	2022 年 11 月 15 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在中东（阿联酋）(me-central-1) Amazon 区域推出。	2022 年 11 月 3 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2022 年 10 月 24 日

Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2022 年 10 月 20 日
Amazon Outposts 上的本地集群现已推出	现在，您可以在 Outpost 上创建 Amazon EKS 本地集群。	2022 年 9 月 19 日
基于 Fargate vCPU 的配额	Fargate 将从基于容器组（pod）的配额过渡到基于 vCPU 的配额。	2022 年 9 月 8 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2022 年 8 月 31 日
成本监控	Amazon EKS 现在支持 Kubecost，让您能够监控按 Kubernetes 资源 [包括容器组（pod）、节点、命名空间和标签] 细分的成本。	2022 年 8 月 24 日
Amazon 托管策略更新：新策略	Amazon EKS 添加了新的 Amazon 托管策略。	2022 年 8 月 24 日
Amazon 托管策略更新：新策略	Amazon EKS 添加了新的 Amazon 托管策略。	2022 年 8 月 23 日
针对账单为资源添加标签	为所有集群增加了 `aws:eks:cluster-name` 生成的成本分配标签支持。	2022 年 8 月 16 日
Fargate 配置文件通配符	在命名空间、标签键和标签值的选择器标准中添加了对 Fargate 配置文件通配符的支持。	2022 年 8 月 16 日
Kubernetes `1.23` 版	增加了新集群和版本升级的 Kubernetes `1.23` 版支持。	2022 年 8 月 11 日

在 Amazon Web Services 管理控制台中查看 Kubernetes 资源	现在，您可以查看使用 Amazon Web Services 管理控制台 部署到集群的 Kubernetes 资源的相关信息。	2022 年 5 月 3 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在亚太地区（雅加达）(ap-southeast-3) Amazon 区域推出。	2022 年 5 月 2 日
可观测性页面和 ADOT 插件支持	添加了 Observability（可观察性）页面和 Amazon Distro for OpenTelemetry（ADOT）。	2022 年 4 月 21 日
Kubernetes 1.22 版	增加了新集群和版本升级的 Kubernetes 1.22 版支持。	2022 年 4 月 22 日
Amazon 托管策略更新：新策略	Amazon EKS 添加了新的 Amazon 托管策略。	2022 年 4 月 22 日
增加了 Fargate 容器组（pod）修补详细信息	升级 Fargate 容器组（pod）时，Amazon EKS 首先尝试根据您的容器组（pod）中断预算驱逐容器组（pod）。您可以创建事件规则，在删除容器组（pod）之前对失败的驱逐做出反应。	2022 年 4 月 1 日
完整版本：对 Amazon EBS CSI 驱动程序的附加组件支持	您现在可以使用 Amazon Web Services 管理控制台、Amazon CLI 和 API 来管理 Amazon EBS CSI 驱动程序。	2022 年 3 月 31 日
Amazon Outposts 内容更新	在 Amazon Outposts 上部署 Amazon EKS 集群的说明。	2022 年 3 月 22 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2022 年 3 月 21 日

Windows containerd 支持	现在，您可以选择 Windows 节点的 containerd 运行时。	2022 年 3 月 14 日
向安全文档中增加了 Amazon EKS Connector 考虑因素	描述与连接集群相关的责任共担模型。	2022 年 2 月 25 日
将 IPv6 地址分配给容器组（pod）和服务	您现在可以创建 1.21 或更高版本的集群，将 IPv6 地址分配给自己的容器组（pod）和服务。	2022 年 1 月 6 日
Amazon 托管式策略更新 — 对现有策略的更新	Amazon EKS 更新了现有的 Amazon 托管策略。	2021 年 12 月 13 日
预览版本：对 Amazon EBS CSI 驱动程序的附加组件支持	您现在可以使用 Amazon Web Services 管理控制台、Amazon CLI 和 API 进行预览，以管理 Amazon EBS CSI 驱动程序。	2021 年 12 月 9 日
Karpenter Autoscaler 支持	现在您可以使用 Karpenter 开源项目自动缩放节点。	2021 年 11 月 29 日
Fargate 日志记录中的 Fluent Bit Kubernetes 筛选支持	现在您可以将 Fluent Bit Kubernetes 筛选用于 Fargate 日志记录。	2021 年 11 月 10 日
控制面板的 Windows 支持	控制面板现在提供 Windows 支持。无需在数据层面中启用。	2021 年 11 月 9 日
添加 Bottlerocket 作为托管节点组的 AMI 类型	以前，Bottlerocket 仅作为自行管理节点选项提供。现在可以将其配置为托管节点组，从而减少满足节点合规性要求所需的工作量。	2021 年 10 月 28 日

DL1 驱动程序支持	自定义 Amazon Linux AMI 现在支持 Amazon Linux 2 的深度学习工作负载。这样支持通用本地部署或云基准配置。	2021 年 10 月 25 日
VT1 视频支持	自定义 Amazon Linux AMI 现在支持部分配的 VT1。此支持功能可在您的 Amazon EKS 集群上传播 Xilinx U30 设备。	2021 年 9 月 13 日
Amazon EKS Connector 现已发布	您可以使用 Amazon EKS Connector 注册并将任何符合要求的 Kubernetes 集群连接至 Amazon，并在 Amazon EKS 控制台中进行显示。	2021 年 9 月 8 日
Amazon EKS Anywhere 现已发布	Amazon EKS Anywhere 是 Amazon EKS 的新部署选项，让您能够在本地创建和操作 Kubernetes 集群。	2021 年 9 月 8 日
Amazon FSx for NetApp ONTAP CSI 驱动程序	添加了总结适用于 NetApp ONTAP 的 Amazon FSx CSI 驱动程序的主题，并提供了其他参考的链接。	2021 年 9 月 2 日
托管节点组现在会自动计算 Amazon EKS 建议的节点最大容器组（pod）数量	对于您未使用启动模板部署的节点，或使用未在其中指定 AMI ID 的启动模板部署的节点，托管节点组现在会自动为其计算 Amazon EKS 最大容器组（pod）数量。	2021 年 8 月 30 日
在不移除 Amazon EKS 附加组件软件的情况下移除附加组件设置的 Amazon EKS 管理	现在，您可以移除 Amazon EKS 附加组件，而无需从集群中移除附加组件软件。	2021 年 8 月 20 日

使用 Multus 创建多宿主容器组	您现在可以使用 Multus 向一个容器组添加多个网络接口了。	2021 年 8 月 2 日
向您的 Linux Amazon EC2 节点添加更多 IP 地址	您现在可以向您的 Linux Amazon EC2 节点添加大量 IP 地址。这意味着您可以在每个节点上运行更高密度的容器组（pod）。您现在可以向您的 Linux Amazon EC2 节点添加大量 IP 地址。这意味着您可以在每个节点上运行更高密度的容器组（pod）。	2021 年 7 月 27 日
`containerd` 运行时间引导	Amazon EKS 优化版加速型 Amazon Linux 亚马逊机器映像（AMI）现在包含一个引导标记，您可以用该标记在 Amazon EKS 优化版和 Bottlerocket AMI 中启用 `containerd` 运行时。此标记可用于所有受支持的 Kubernetes 版本的 AMI。	2021 年 7 月 19 日
Kubernetes `1.21` 版	增加了 Kubernetes `1.21` 版支持。	2021 年 7 月 19 日
已添加托管策略主题	所有 Amazon EKS IAM 托管策略以及自 2021 年 6 月 17 日以来对其做出的更改的列表。	2021 年 6 月 17 日
将容器组（pod）的安全组与 Fargate 一起使用	除了将容器组（pod）的安全组与 Amazon EC2 节点一起使用之外，您现在还可以将其与 Fargate 一起使用。	2021 年 6 月 1 日

已添加 CoreDNS 和 kube-proxy Amazon EKS 加载项	Amazon EKS 现在可以帮助您管理集群的 CoreDNS 和 kube-proxy Amazon EKS 加载项。	2021 年 5 月 19 日
Kubernetes 1.20 版	增加了新集群和版本升级的 Kubernetes 1.20 版支持。	2021 年 5 月 18 日
Amazon已发布 2.2.0 负载均衡器控制器	现在，您可以使用Amazon负载均衡器控制器，以利用实例或 IP 目标创建 Elastic Load Balancer。	2021 年 5 月 14 日
托管节点组的节点污点	Amazon EKS 现在支持向托管节点组添加节点污点。	2021 年 5 月 11 日
现有集群的密钥加密	Amazon EKS 现在支持向现有集群添加密钥加密。	2021 年 2 月 26 日
Kubernetes 1.19 版	增加了新集群和版本升级的 Kubernetes 1.19 版支持。	2021 年 2 月 16 日
Amazon EKS 现在支持 OpenID Connect（OIDC）身份提供商作为对 1.16 版或更高版本集群的用户进行身份验证的方法。	OIDC 身份提供者可与 Amazon Identity and Access Management（IAM）结合使用或作为其替代方法。	2021 年 2 月 12 日
在 Amazon Web Services 管理控制台 中查看节点和工作负载资源	您现在可以在 Amazon Web Services 管理控制台中查看有关托管节点、自主管理型节点和 Fargate 节点以及您部署的 Kubernetes 工作负载的详细信息。	2020 年 12 月 1 日
在托管节点组中部署竞价型实例类型	您现在可以将多种 Spot 或按需实例类型部署到托管节点组。	2020 年 12 月 1 日

Amazon EKS 现在可以管理集群的特定附加组件	您可以自行管理附加组件，也可以让 Amazon EKS 通过 Amazon EKS API 控制附加组件的启动和版本。	2020 年 12 月 1 日
跨多个入口共享 ALB	现在，您可以共享跨多个 Kubernetes 入口共享 Amazon Application Load Balancer (ALB)。过去，您必须为每个入口部署单独的 ALB。	2020 年 10 月 23 日
NLB IP 目标支持	您现在可以部署具有 IP 目标的网络负载均衡器。这意味着您可以使用 NLB 将网络流量负载均衡到 Fargate 容器组（pod）以及直接负载均衡到运行于 Amazon EC2 节点上的容器组（pod）。	2020 年 10 月 23 日
Kubernetes 1.18 版	增加了新集群和版本升级的 Kubernetes 1.18 版支持。	2020 年 10 月 13 日
为 Kubernetes 服务 IP 地址分配指定自定义 CIDR 块。	现在可以指定 Kubernetes 从中分配服务 IP 地址的自定义 CIDR 块。	2020 年 9 月 29 日
将安全组分配给各个容器组（pod	现在，您可以将不同的安全组与在许多 Amazon EC2 实例类型上运行的某些单独的容器组（pod）关联起来。	2020 年 9 月 9 日
在节点上部署 Bottlerocket	您现在可以部署运行 Bottlerocket 的节点。	2020 年 8 月 31 日
启动 Arm 节点的功能通常可用	现在，您可以在托管节点组和自行管理节点组中启动 Arm 节点。	2020 年 8 月 17 日

托管节点组启动模板和自定义 AMI	您现在可使用 Amazon EC2 启动模板部署托管节点组。如果您选择，则启动模板可指定自定义 AMI。	2020 年 8 月 17 日
EFS 支持 Amazon Fargate	您现在可以将 Amazon EFS 与 Amazon Fargate 结合使用	2020 年 8 月 17 日
Amazon EKS 平台版本更新	这是一个新的平台版本（包括安全修复和增强功能）。这包括在将网络负载均衡器用于 Kubernetes `1.15` 版或更高版本时 UDP 对 `LoadBalancer` 类型服务的支持。有关更多信息，请参阅 GitHub 上的允许 UDP 用于 Amazon 网络负载均衡器。	2020 年 8 月 12 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在非洲（开普敦）(`af-south-1` ）和欧洲地区（米兰）(`eu-south-1` ）Amazon 区域推出。	2020 年 8 月 6 日
Fargate 使用情况指标	Amazon Fargate 提供 CloudWatch 使用情况指标，通过该指标可以了解您对 Fargate 按需资源的账户使用情况。	2020 年 8 月 3 日
Kubernetes `1.17` 版	增加了新集群和版本升级的 Kubernetes `1.17` 版支持。	2020 年 7 月 10 日
使用适用于 Kubernetes 的 App Mesh 控制器，在 Kubernetes 中创建和管理 App Mesh 资源	您可以在 Kubernetes 中创建和管理 App Mesh 资源。控制器还会自动在您部署的容器组（pod）中注入 Envoy 代理和 init 容器。	2020 年 6 月 18 日

Amazon EKS 现在支持 Amazon EC2 Inf1 节点	您可以向集群添加 Amazon EC2 Inf1 节点。	2020 年 6 月 4 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在 Amazon GovCloud（美国东部）(us-gov-east-1) 和 Amazon GovCloud（美国西部）(us-gov-west-1) Amazon 区域推出。	2020 年 5 月 13 日
Amazon EKS 不再支持 Kubernetes 1.12	Amazon EKS 不再支持 Kubernetes 1.12 版。将所有 1.12 版集群更新到版本 1.13 或更高版本以避免服务中断。	2020 年 5 月 12 日
Kubernetes 1.16 版	增加了新集群和版本升级的 Kubernetes 1.16 版支持。	2020 年 4 月 30 日
添加了 AWSServiceRoleForAmazonEKS 服务相关角色	添加了 AWSServiceRoleForAmazonEKS 服务相关角色。	2020 年 4 月 16 日
Kubernetes 1.15 版	增加了新集群和版本升级的 Kubernetes 1.15 版支持。	2020 年 3 月 10 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在北京 (cn-north-1) 和宁夏 (cn-northwest-1) Amazon 区域推出。	2020 年 2 月 26 日
FSx for Lustre CSI 驱动程序	增加了在 Kubernetes 1.14 Amazon EKS 集群上安装 FSx for Lustre CSI 驱动程序的主题。	2019 年 12 月 23 日
限制对集群的公有访问端点的网络访问	此次更新后，您可以利用 Amazon EKS，限制可与 Kubernetes API 服务器的公有访问端点通信的 CIDR 范围。	2019 年 12 月 20 日

在 VPC 外部解析集群的私有访问端点地址	通过此项更新，您可以利用 Amazon EKS，在 VPC 外部解析 Kubernetes API 服务器的私有访问端点。	2019 年 12 月 13 日
（测试版）Amazon EC2 A1 Amazon EC2 实例节点	启动向 Amazon EKS 集群注册的 Amazon EC2 A1 Amazon EC2 实例节点。	2019 年 12 月 4 日
在 Amazon Outpost 上创建集群	Amazon EKS 现在支持在 Amazon Outposts 上创建集群。	2019 年 12 月 3 日
Amazon EKS 上的 Amazon Fargate	Amazon EKS Kubernetes 集群现在支持在 Fargate 上运行容器组（pod）。	2019 年 12 月 3 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在加拿大（中部）(`ca-central-1`) Amazon 区域推出。	2019 年 11 月 21 日
托管节点组	Amazon EKS 托管节点组为 Amazon EKS Kubernetes 集群自动对节点（Amazon EC2 实例）进行预置和生命周期管理。	2019 年 11 月 18 日
Amazon EKS 平台版本更新	新平台版本，以解决 CVE-2019-11253。	2019 年 11 月 6 日
Amazon EKS 不再支持 Kubernetes `1.11`	Amazon EKS 不再支持 Kubernetes `1.11` 版。请将所有 `1.11` 版集群更新到版本 `1.12` 或更高版本以避免服务中断。	2019 年 11 月 4 日

Amazon EKS Amazon 区域扩展	Amazon EKS 现已在南美洲（圣保罗）(sa-east-1) Amazon 区域推出。	2019 年 10 月 16 日
Windows 支持	现在，运行 Kubernetes 1.14 版的 Amazon EKS 集群支持 Windows 工作负载。	2019 年 10 月 7 日
Autoscaling	添加了一章，用于介绍 Amazon EKS 集群支持的不同类型的 Kubernetes 弹性伸缩。	2019 年 9 月 30 日
Kubernetes 控制面板更新	更新了在 Amazon EKS 集群上安装 Kubernetes 控制面板以使用 Beta 2.0 版的主题。	2019 年 9 月 28 日
Amazon EFS CSI 驱动程序	增加了在 Kubernetes 1.14 Amazon EKS 集群上安装 Amazon EFS CSI 驱动程序的主题。	2019 年 9 月 19 日
Amazon EKS 优化版 AMI ID 的 Amazon EC2 Systems Manager 参数	添加了使用 Amazon EC2 Systems Manager 参数检索 Amazon EKS 优化版 AMI ID 的主题。使用该参数，您无需查找 AMI ID。	2019 年 9 月 18 日
Amazon EKS 资源标记	您可以管理 Amazon EKS 集群的标记。	2019 年 9 月 16 日
Amazon EBS CSI 驱动程序	增加了在 Kubernetes 1.14 Amazon EKS 集群上安装 Amazon EBS CSI 驱动程序的主题。	2019 年 9 月 9 日

针对 CVE-2019-9512 和 CVE-2019-9514 进行修补的新的 Amazon EKS 优化版 AMI	Amazon EKS 更新了 Amazon EKS 优化版 AMI，用于解决 CVE-2019-9512 和 CVE-2019-9514。	2019 年 9 月 6 日
宣布在 Amazon EKS 中弃用 Kubernetes 1.11	Amazon EKS 已于 2019 年 11 月 4 日停止支持 Kubernetes 1.11 版。	2019 年 9 月 4 日
Kubernetes 1.14 版	增加了新集群和版本升级的 Kubernetes 1.14 版支持。	2019 年 9 月 3 日
服务账户的 IAM 角色	通过 Amazon EKS 集群上服务账户的 IAM 角色，您可以将 IAM 角色与 Kubernetes 服务账户关联。借助此功能，您不再需要向节点 IAM 角色提供扩展的权限。这样，该节点上的容器组（pod）就可以调用 Amazon API 了。	2019 年 9 月 3 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在中东（巴林）(me-south-1) Amazon 区域推出。	2019 年 8 月 29 日
Amazon EKS 平台版本更新	新平台版本，以针对 CVE-2019-9512 和 CVE-2019-9514。	2019 年 8 月 28 日
Amazon EKS 平台版本更新	新平台版本，以针对 CVE-2019-11247 和 CVE-2019-11249。	2019 年 8 月 5 日
Amazon EKS 区域扩展	Amazon EKS 现已在亚太地区（香港）(ap-east-1) Amazon 区域推出。	2019 年 7 月 31 日

Amazon EKS 不再支持 Kubernetes 1.10	Amazon EKS 不再支持 Kubernetes 1.10 版。将所有 1.10 版集群更新到版本 1.11 或更高版本以避免服务中断。	2019 年 7 月 30 日
增加了有关 ALB 入口控制器的主题	适用于 Kubernetes 的 Amazon ALB 入口控制器是一种控制器，在创建入口资源时，此控制器将触发 ALB 的创建。	2019 年 7 月 11 日
新的 Amazon EKS 优化版 AMI	从 AMI 删除不必要的 kubectl 二进制文件。	2019 年 7 月 3 日
Kubernetes 1.13 版	增加了新集群和版本升级的 Kubernetes 1.13 版支持。	2019 年 6 月 18 日
针对 Amazon-2019-005 进行修补的新的 Amazon EKS 优化版 AMI	Amazon EKS 已更新 Amazon EKS 优化版 AMI，解决了 link:security/security-bulletins/Amazon-2019-005/[Amazon-2019-005,type="marketing"] 中描述的漏洞。	2019 年 6 月 17 日
宣布在 Amazon EKS 中停止对 Kubernetes 1.10 的支持	2019 年 7 月 22 日，Amazon EKS 已停止支持 Kubernetes 1.10 版。	2019 年 5 月 21 日
Amazon EKS 平台版本更新	Kubernetes 1.11 和 1.10 集群的新平台版本支持在 kubelet 证书中使用自定义 DNS 名称，能够提高 etcd 性能。	2019 年 5 月 21 日

入门eksctl	此入门指南描述如何使用 eksctl 安转开始使用 Amazon EKS 所需的所有资源。这是一种用于在 Amazon EKS 上创建和管理 Kubernetes 集群的简单命令行实用程序。	2019 年 5 月 10 日
Amazon CLI get-token 命令	aws eks get-token 命令已添加到 Amazon CLI。您不再需要安装适用于 Kubernetes 的 Amazon IAM 身份验证器来创建用于集群 API 服务器通信的客户端安全令牌。将您的 Amazon CLI 安装升级到最新版本以使用此新功能。有关更多信息，请参阅 Amazon 命令行界面用户指南中的安装 Amazon 命令行界面。	2019 年 5 月 10 日
Amazon EKS 平台版本更新	Kubernetes 1.12 集群的新平台版本支持在 kubelet 证书中使用自定义 DNS 名称，能够提高 etcd 性能。这将修复导致节点 kubelet 守护程序每隔几秒便会请求新证书的漏洞。	2019 年 5 月 8 日
Prometheus 教程	增加了将 Prometheus 部署到 Amazon EKS 集群的主题。	2019 年 4 月 5 日
Amazon EKS 控制面板日志记录	通过此更新，您可以直接从 Amazon EKS 控制窗格中获取审计和诊断日志。您可以在账户中使用这些 CloudWatch 日志作为参考来保护和运行集群。	2019 年 4 月 4 日

Kubernetes 1.12 版	增加了新集群和版本升级的 Kubernetes 1.12 版支持。	2019 年 3 月 28 日
增加了 App Mesh 入门指南	增加了如何开始使用 App Mesh 和 Kubernetes 的文档。	2019 年 3 月 27 日
Amazon EKS API 服务器端点私有访问	增加了如何禁用 Amazon EKS 集群的 Kubernetes API 服务器端点的公有访问的文档。	2019 年 3 月 19 日
增加了安装 Kubernetes Metrics Server 的主题	Kubernetes Metrics Server 是集群中资源使用情况数据的聚合器。	2019 年 3 月 18 日
增加了相关开源项目的列表	这些开源项目扩展了在 Amazon 上运行的 Kubernetes 集群的功能，包括 Amazon EKS 管理的集群。	2019 年 3 月 15 日
增加了本地安装 Helm 的主题	Kubernetes 的 `helm` 包管理器帮助您在 Kubernetes 集群上安装和管理应用程序。本主题介绍了如何在本地安装和运行 `helm` 和 `tiller` 二进制文件。这样，您可以在本地系统中使用 Helm CLI 安装和管理图表。	2019 年 3 月 11 日
Amazon EKS 平台版本更新	新平台版本将 Amazon EKS Kubernetes 1.11 集群更新到补丁级别 `1.11.8`，解决了 CVE-2019-1002100 问题。	2019 年 3 月 8 日
提高了集群限制	Amazon EKS 将您可以在 Amazon 区域中创建的集群数从 3 增加到 50。	2019 年 2 月 13 日

Amazon EKS Amazon 区域扩展	Amazon EKS 现已在欧洲地区（伦敦）(eu-west-2)、欧洲地区（巴黎）(eu-west-3 ）和亚太地区（孟买）(ap-south-1 ）Amazon 区域推出。	2019 年 2 月 13 日
针对 ALAS-2019-1156 进行修补的新的 Amazon EKS 优化版 AMI	Amazon EKS 已更新 Amazon EKS 优化版 AMI，解决了 ALAS-2019-1156 中描述的漏洞。	2019 年 2 月 11 日
针对 ALAS2-2019-1141 进行修补的新的 Amazon EKS 优化版 AMI	Amazon EKS 更新了 Amazon EKS 优化版 AMI，以处理 ALAS2-2019-1141 中引用的 CVE。	2019 年 1 月 9 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在亚太地区（首尔）(ap-northeast-2 ）Amazon 区域推出。	2019 年 1 月 9 日
Amazon EKS 区域扩展	Amazon EKS 现已在以下其它 Amazon 区域推出：欧洲地区（法兰克福）(ap-southeast-1)、亚太地区（东京）(eu-central-1)、亚太地区（新加坡）(ap-northeast-1 ）和亚太地区（悉尼）(ap-southeast-2)。	2018 年 12 月 19 日
Amazon EKS 集群更新	增加了 Amazon EKS 集群 Kubernetes 版本更新和节点替换文档。	2018 年 12 月 12 日
Amazon EKS Amazon 区域扩展	Amazon EKS 现已在欧洲地区（斯德哥尔摩）(eu-north-1 ）Amazon 区域推出。	2018 年 12 月 11 日

Amazon EKS 平台版本更新	新平台版本将 Kubernetes 更新到补丁级别 1.10.11，解决了 link:security/security-bulletins/Amazon-2018-020/[CVE-2018-1002105,type="marketing"]。	2018 年 12 月 4 日
为 ALB 接收控制器添加了 1.0.0 版本支持	ALB 接收控制器发行版本 1.0.0，提供来自 Amazon 的正式支持。	2018 年 11 月 20 日
添加了对 CNI 网络配置的支持	适用于 Kubernetes 的 Amazon VPC CNI 插件版本 1.2.1 现在支持辅助容器组（pod）网络接口的自定义网络配置。	2018 年 10 月 16 日
添加了对 MutatingAdmissionWebhook 和 ValidatingAdmissionWebhook 的支持	Amazon EKS 平台版本 1.10-eks.2 现在支持 MutatingAdmissionWebhook 和 ValidatingAdmissionWebhook 准入控制器。	2018 年 10 月 10 日
添加了合作伙伴 AMI 信息	Canonical 还与 Amazon EKS 合作创建了可在您的集群中使用的节点 AMI。	2018 年 10 月 3 日
添加了有关 Amazon CLI update-kubeconfig 命令的说明	Amazon EKS 向 Amazon CLI 添加了 update-kubeconfig ，以简化用于访问集群的 kubeconfig 文件的创建过程。	2018 年 9 月 21 日
新的 Amazon EKS 优化版 AMI	Amazon EKS 更新了 Amazon EKS 优化版 AMI（使用和不使用 GPU 支持），以提供各种安全修复和 AMI 优化。	2018 年 9 月 13 日

Amazon EKS Amazon 区域扩展	Amazon EKS 现已在欧洲（爱尔兰）(eu-west-1 ）区域推出。	2018 年 9 月 5 日
Amazon EKS 平台版本更新	新平台版本支持 Kubernetes 聚合层和 Horizontal Pod Autoscaler(HPA)。	2018 年 8 月 31 日
新增了 Amazon EKS 优化版 AMI 和 GPU 支持	Amazon EKS 更新了 Amazon EKS 优化版 AMI，以使用新的 Amazon CloudFormation 节点模板和引导脚本。此外，还提供了新的具有 GPU 支持的 Amazon EKS 优化版 AMI。	2018 年 8 月 22 日
针对 ALAS2-2018-1058 进行修补的新的 Amazon EKS 优化版 AMI	Amazon EKS 更新了 Amazon EKS 优化版 AMI，以处理 ALAS2-2018-1058 中引用的 CVE。	2018 年 8 月 14 日
Amazon EKS 优化版 AMI 生成脚本	Amazon EKS 已公开用于生成 Amazon EKS 优化版 AMI 的生成脚本的源代码。GitHub 上现在提供了这些生成脚本。	2018 年 7 月 10 日
Amazon EKS 初始版本	服务的初始文档发布	2018 年 6 月 5 日

改进《EKS 用户指南》

Amazon 改进了《EKS 用户指南》的贡献体验。

现在，您可以直接在 GitHub 上编辑《EKS 用户指南》的源代码。

该文档现在使用 AsciiDoc，这是一种类似于 markdown 的强大创作语言。AsciiDoc 将简单的语法与高级格式、交叉引用和安全控制等企业文档功能相结合。

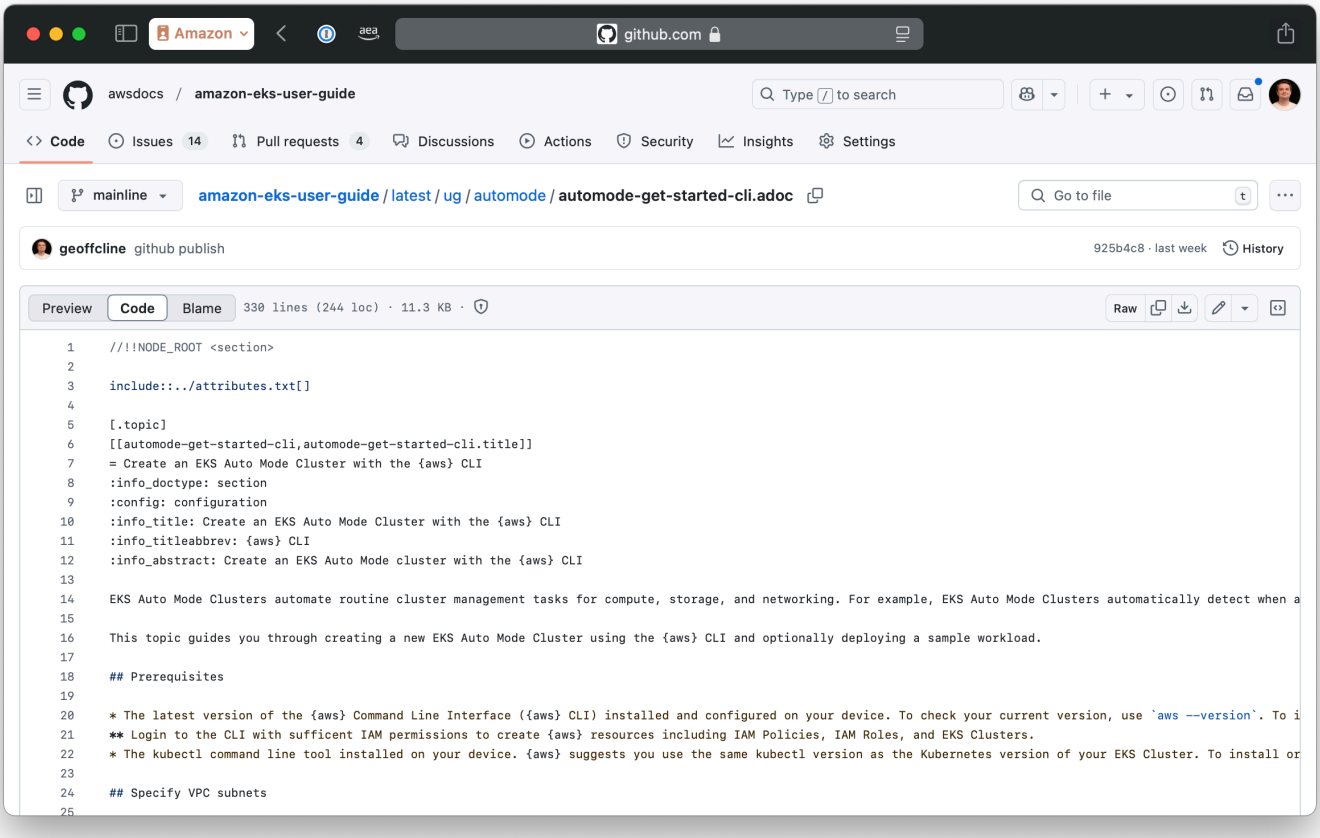
现在，您可以直接在 GitHub 上编辑 EKS 文档。我们的简化流程包括：

- 更快地处理拉取请求
- 减少手动步骤
- 自动内容质量检查

我们期待您的助力。

通过 Web 浏览器编辑单个页面

您可以直接通过 Web 浏览器轻松编辑《EKS 用户指南》中的单个页面。



要通过 Web 浏览器编辑多个页面，请参阅the section called “使用 GitHub 编辑文件”。

先决条件

- 要更改的文档页面已在 Web 浏览器中打开。
- 已登录到 GitHub。

过程

1. 导航到《Amazon EKS 用户指南》中要编辑的页面。
2. 在右侧窗格中选择在 GitHub 上编辑此页面链接。
3. 进入 GitHub 后，通过以下任一方式打开编辑器：
   - 按下键盘上的 e 键。
   - 点击铅笔图标，从下拉菜单中选择就地编辑。

- 如果您没有看到编辑选项，则需要登录到 GitHub。您的 GitHub 账户不需要任何特殊权限即可提出更改建议。但是，Amazon 内部贡献者应关联其 GitHub 个人资料。

4. 在 GitHub 编辑器中对内容进行必要的更改。
   - 该编辑器提供了语法突出显示和预览功能。
   - 您可以使用 AsciiDoc 标记安排更改内容的格式。
   - 您可以使用 `ctrl-f` 打开查找/替换界面。
5. （可选）预览更改。
   - 使用 `preview` 选项卡以丰富的格式预览更改。
   - 使用 `show diff` 选项可突出显示已更改的部分。删除的部分左边空白处有一个红色指示符。新增部分左边空白处有一个绿色指示符。
6. 编辑完成后，点击编辑器顶部的提交更改... 按钮
7. 在提交对话框中：
   - 验证电子邮件地址是否正确。
   - 添加一条简短的描述性提交消息来解释自己的更改。
   - （可选）根据需要添加更完整的描述。
   - 选择创建新分支和拉取请求。

您已经创建了一个包含建议更改的拉取请求。

拉取请求概述

创建 PR 时：

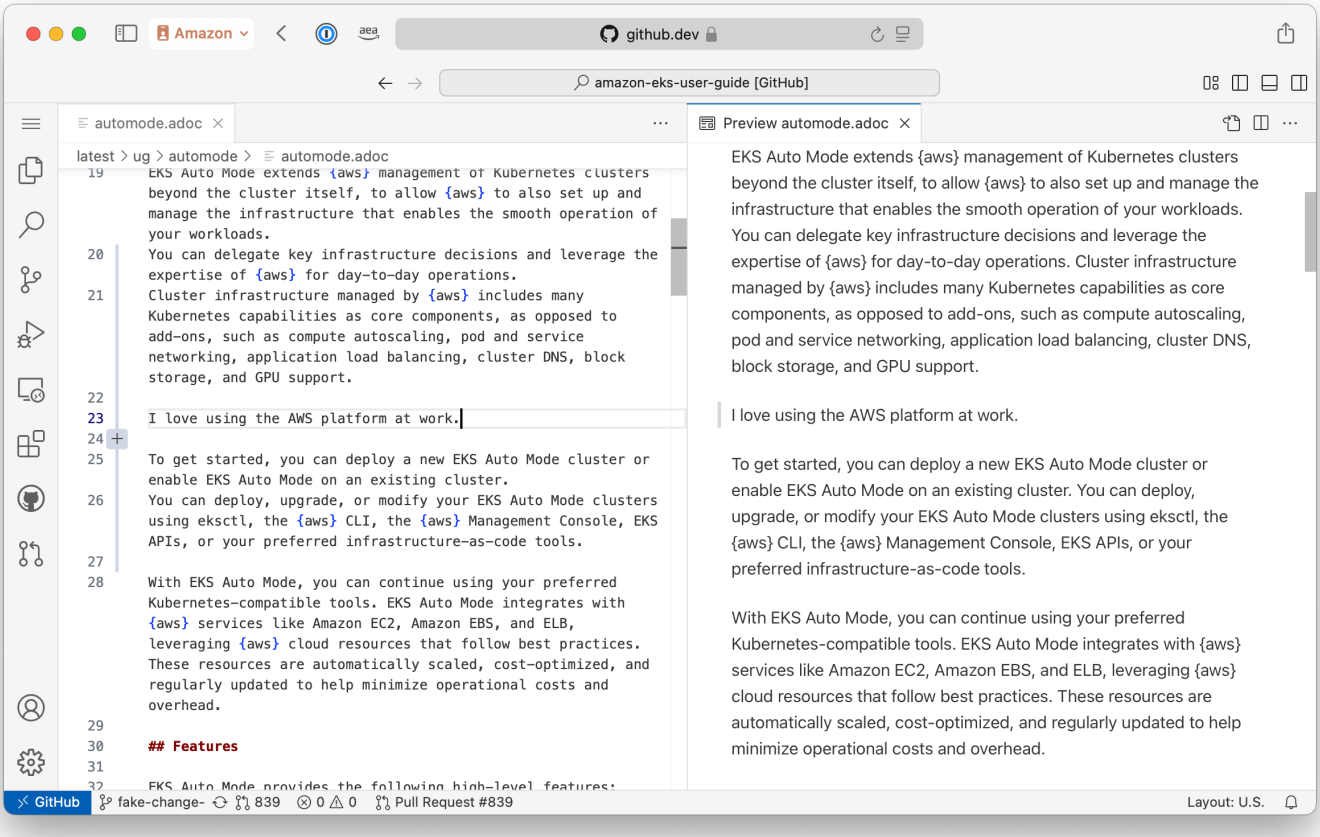
- 您的更改已提交给存储库维护者审阅。
- 审阅者可以对您的更改发表评论，要求进行相应修改。
- 可能会运行自动测试来验证您的更改。
- 一旦获得批准，您的更改就可以合并到主存储库中。

拉取请求有助于确保质量，并提供了在合并更改之前对其进行讨论的方式。

了解如何在 GitHub 文档中审阅和批准拉取请求。

使用 GitHub Web 编辑器通过 Web 浏览器编辑多个文件

如果需要对多个页面提出更改建议，或者创建一个新的文档页面，请使用 GitHub.dev Web 编辑器。此 Web 编辑器基于流行的 Visual Studio Code 文本编辑器。



先决条件

- 已登录到 GitHub
- 熟悉 Visual Studio Code 编辑器
- 熟悉 Git

过程

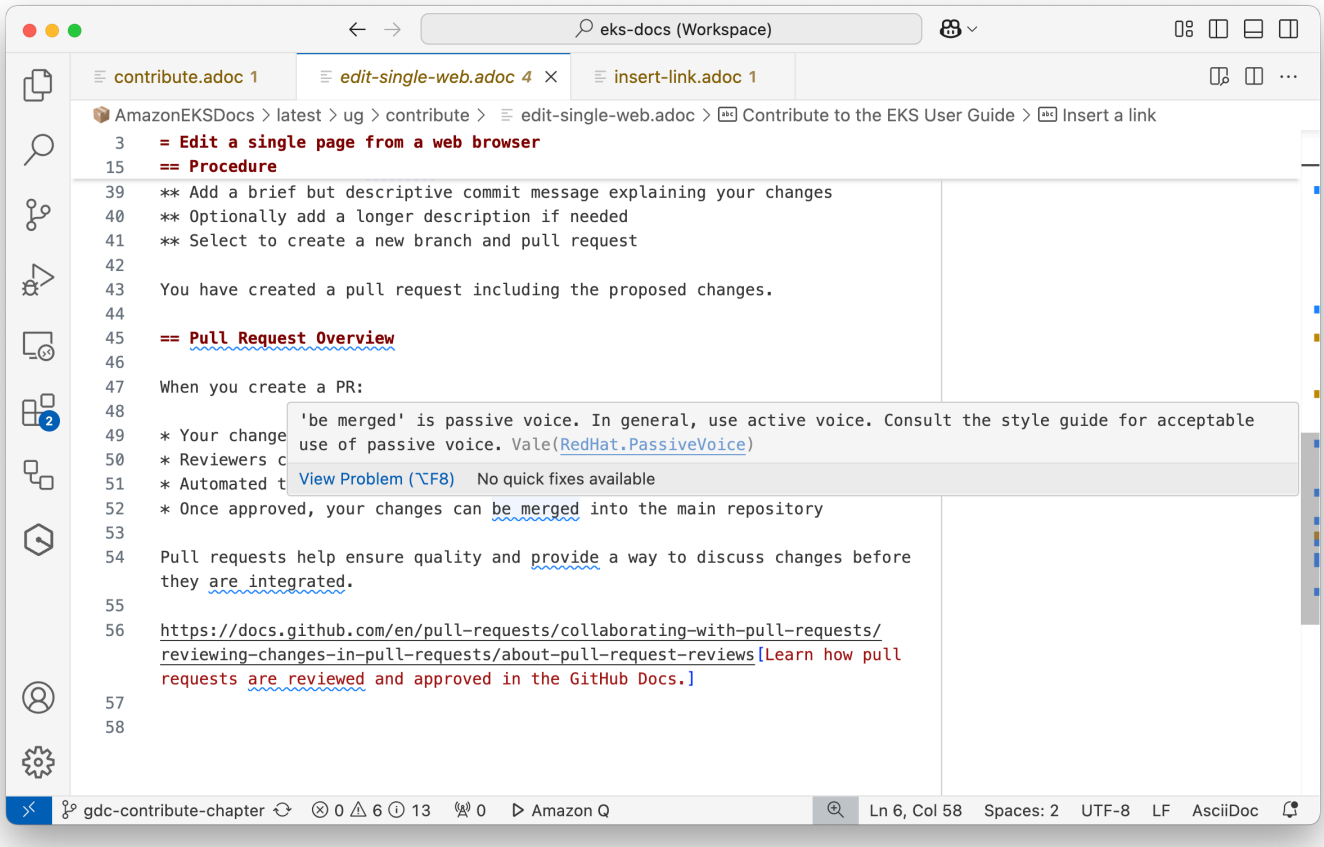
> **Note**
>
> EKS 文档团队创建了一个工作区文件，其中包含编辑器的建议配置，例如文本换行和 AsciiDoc 语法突出显示等。我们建议您加载此工作区文件。

1. 在 GitHub.dev 上打开工作区。
   - 您可以将书签添加到 URL `https://github.dev/awsdocs/amazon-eks-user-guide/blob/mainline/eks-docs.code-workspace?workspace=true`
2. （仅限首次设置时）系统可能会提示您在自己的 GitHub 账户中为存储库创建一个分支。接受此提示。有关更多信息，请参阅《GitHub 文档》中的关于分叉。
3. （仅限首次设置时）接受右下角的提示安装 AsciiDoc 扩展程序。
4. 导航到位于 `latest/ug` 的文档内容。
   - 文档文件按其顶层部分进行组织。例如，“安全”章节中页面的源文件位于“security/”目录下。
5. 要查看某个文档页面的预览，请使用右上角的在侧面打开预览按钮。该图标包括一个小放大镜。
6. 使用左侧的源控制选项卡提交更改。有关更多信息，请参阅 Visual Studio Code 文档：
   - 提交更改
   - 创建拉取请求

创建拉取请求后，文档团队将对其进行审阅。

在本地安装 Vale 以在键入时查看样式反馈

您可以在键入时查看样式反馈。这有助于识别写作错误和错别字。



概述：

- Vale CLI 会加载并针对源文件运行样式指南。
- EKS Docs 存储库包含一个用于加载样式指南和本地规则的 vale 配置文件。
- 适用于 Visual Studio（VS）Code 的 Vale 扩展会在编辑器中显示 vale 反馈。

安装 Vale

按照 Vale CLI 文档中的说明使用 Package Manager 安装 Vale。

安装 VS Code Vale 扩展

1. 打开 VS Code。
2. 单击活动栏中的扩展程序图标（或按 Ctrl+Shift+X）。
3. 搜索“Vale”。

4. 在 Chris Chinchilla 的“Vale VSCode”扩展程序上单击“安装”。
5. 出现提示时重新加载 VS 代码。

同步 Vale

Vale 会使用项目根目录中的 `.vale.ini` 配置文件来确定要应用哪些样式规则。

1. 打开 VS Code。
2. 单击查看 > 终端（或按 Ctrl+`）。
3. 如果需要，导航到项目的根目录。
4. 运行命令：

```
vale sync
```

5. 等待 Vale 完成样式规则的下载和同步

在 VS Code 中查看样式反馈

1. 在 VS Code 中打开一个 Markdown 或 AsciiDoc 文件。
2. Vale 扩展程序会根据样式规则自动检查文本。
3. 样式问题将在编辑器中用下划线表示。
4. 将鼠标悬停在带下划线的文本上可查看具体的样式建议。
5. 根据建议或样式指南修复问题。

创建新页面

了解如何创建新的文档页面。本主题包含有关创建初始页面元数据以及将页面添加到指南目录的说明。

创建页面

1. 导航到章节目录。例如，假设要在“安全”部分创建新页面，请导航到 `latest/ug/security` 目录。
2. 确定页面 ID。按照惯例，页面 ID 全部为小写并用 - 分隔。此页面的 ID 为 `create-page`。
3. 使用此页面 ID 和 adoc 扩展名创建一个新文件。例如 `create-page.adoc`。
4. 使用此模板插入页面元数据：

```
include::../attributes.txt[]

[.topic]
[#unique-page-id]
= Page title
:info_titleabbrev: Short title

[abstract]
--
Brief summary of the page's content.
--

Introduction paragraph goes here.
```

将页面添加到导航

1. 导航至父级页面。顶层版块的父级页面是 book.adoc。
2. 在父级页面底部，添加子页面。

   ```
   include::${filename}[leveloffset=+1]
   ```

   例如：

   ```
   include::create-page.adoc[leveloffset=+1]
   ```

插入链接

AsciiDoc 支持多种类型的链接。使用正确的链接类型很重要，这样链接才能在不同的环境中正常工作。

链接到《EKS 用户指南》中的页面或章节

使用交叉引用（xref）在同一文档网站（例如《EKS 用户指南》）中的页面或章节之间进行链接。如果目标章节移动或重命名，链接会自动更新。

使用页面标题作为链接文本

对于本用户指南中链接到其他 ID 的大多数情况，请使用以下方法让链接文本根据需要自动更新为最新标题。

```
For more information, see <<page-or-section-id>>.
```

定义自定义链接文本

对于必须有自定义链接文本的情况，请使用以下格式。

```
Here's an example of a <<page-or-section-id,link with custom text>>.
```

链接到 Amazon 文档中的其他指南

1. 查找指向 Amazon 文档页面的链接。
2. 移除 https://docs.aws.amazon.com/ 前缀，仅保留路径。路径应以字母开头。
3. 创建一个链接，如下所示：

```
link:AmazonS3/latest/userguide/create-bucket-overview.html[Create a bucket,
 type="documentation"]
```

链接到外部网页

这种格式会创建一个指向非 Amazon 托管页面的标准链接。例如，GitHub 链接可使用此格式。

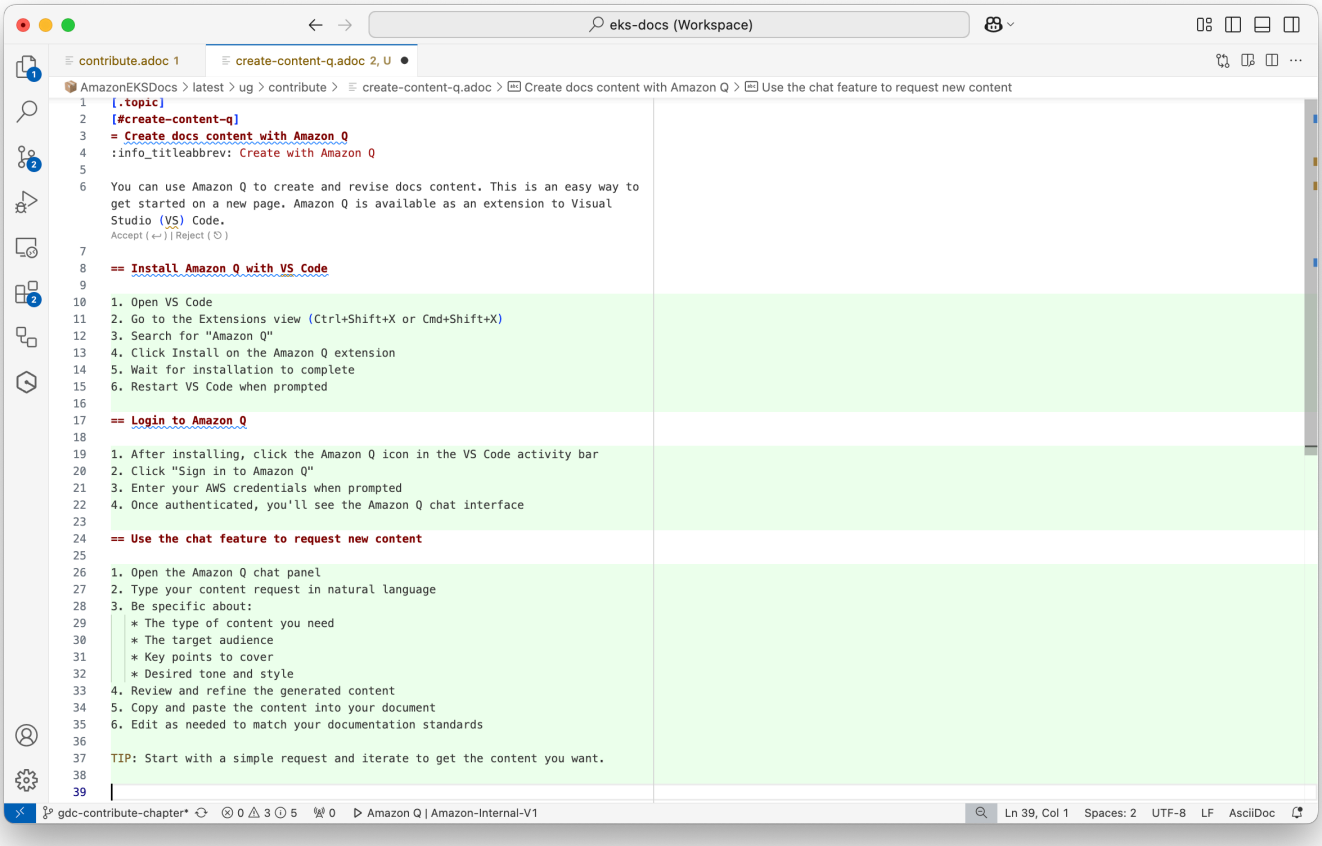
```
https://example.com[Link text]
```

> **Note**
>
> 我们已设置外部域允许列表。允许名单位于 `vale/styles/EksDocs/ExternalDomains.yml`

使用 Amazon Q 创建文档内容

您可以使用 Amazon Q 创建和修改文档内容。这是一种开始新页面的简单方法。Amazon Q 作为 Visual Studio（VS）Code 的扩展程序提供。

在下图中，Amazon Q 生成的行标为绿色。



使用 VS Code 安装 Amazon Q

1. 打开 VS Code
2. 前往“扩展程序”视图（Ctrl+Shift+X 或 Cmd+Shift+X）
3. 搜索“Amazon Q”
4. 单击 Amazon Q 扩展程序上的“安装”
5. 等待安装完成。
6. 出现提示时重启 VS Code

登录 Amazon Q

1. 安装完成后，选择 VS Code 活动栏中的 Amazon Q 图标。
2. 选择登录 Amazon Q。
3. 出现提示时，输入 Amazon 凭证。

4. 通过身份验证后，您将看到 Amazon Q 聊天界面。

使用 Amazon Q 创建内容

1. 在 VS Code 中打开要编辑的文件。
2. 选择要修改的文本或用于放置新内容的位置。
3. 按 Ctrl+I 或 Cmd+I。
4. 在提示中，具体说明：
   - 所需的内容类型。
   - 目标受众。
   - 需要涉及的要点。
   - 所需的语气和风格。
5. 在内联预览中检查生成的内容。
6. 使用 Enter 回车键接受更改，或使用 esc 键拒绝更改。
7. 根据需要进一步编辑。

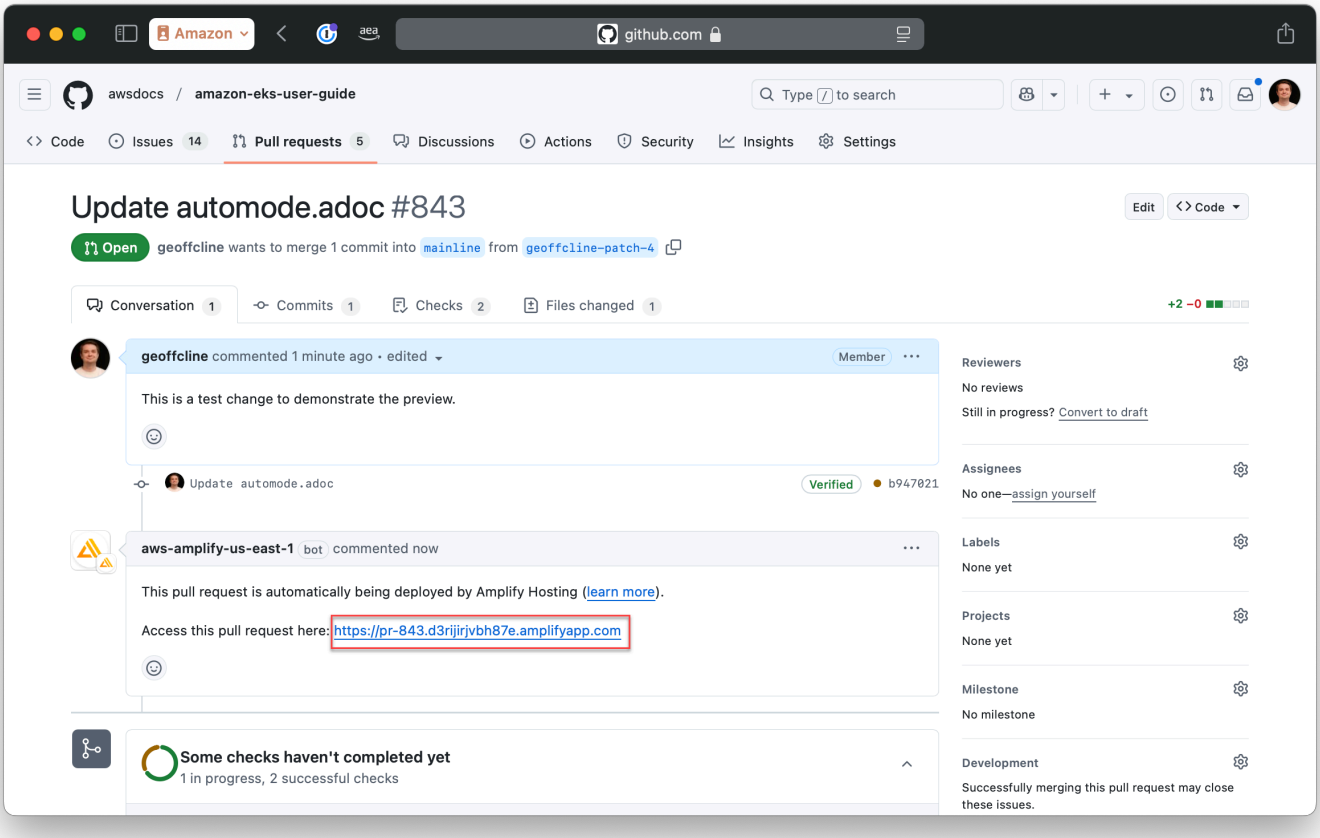
提示

- 从简单的请求开始，然后进行迭代以获取所需的内容。
- 创建页面标题的初稿，然后让 Q 填写。
- Amazon Q 可能会输出 Markdown。这是正常的。AsciiDoc 工具可以理解大多数 markdown 语法。

要了解有关 Amazon Q 开发者版的更多信息，请参阅 Using Amazon Q Developer in the IDE。

查看拉取请求内容的预览

Amazon EKS 用户指南 GitHub 已配置为可构建和生成文档站点的预览。此预览没有完整的 Amazon 主题，但确实可以检查内容是否正确构建以及链接是否有效。



此预览由 Amazon Amplify 在临时 URL 上托管。

查看预览

当您提交拉取请求时，Amazon Amplify 会尝试构建和部署内容的预览。

如果构建成功，aws-amplify-us-east-1 会向具有预览链接的拉取请求添加注释。选择“在此处访问此拉取请求”右侧的链接（如屏幕截图中红框标示出的内容）。

如果构建失败，存储库管理员可以查看日志并提供反馈。

> **Note**
>
> 如果您之前没有贡献过，则可能需要项目维护者批准后才能运行构建。

预览限制

此预览以单个大型 HTML 文件的形式构建。发布时将显示为多页。

哪些功能起作用：

- 交叉引用 (`xref`)
- 互联网链接
- 映像
- 来自 `samples/` 的托管内容

哪些功能不起作用：

- 使用 `type="documentation"` 链接到其他 Amazon 内容。这是因为此内容在预览环境中不存在。
- 属性 `{aws}` 不会正确显示。其值因环境而异。

AsciiDoc 语法参考

AsciiDoc 是 Amazon 文档的首选标记语言。虽然该工具部分支持 Markdown 语法（例如标题和列表），但我们建议对所有内容都使用 AsciiDoc 语法，以确保一致性和正确呈现。

本指南涵盖您在为 Amazon EKS 文档撰稿时所需的常用语法元素。有关更高级的语法和详细信息，请参阅 AsciiDoc 文档。

新页面

每个新的 AsciiDoc 文档都应以the section called “创建页面”中定义的结构开头。

标题

使用标题按层次组织内容：

```
= Page/topic title (level 1)
== Section title (level 2)
=== Level 3 heading
==== Level 4 heading
===== Level 5 heading
```

```
====== Level 6 heading
```

注意：Amazon 文档中的标题始终使用句子大小写。

文本格式设置

设置文本格式以强调重要信息：

```
Use *bold text* for UI labels.
Use _italic text_ for introducing terms or light emphasis.
Use `monospace text` for code, file names, and commands.
```

Lists

未排序的列表

为没有特定顺序的项目创建项目符号列表：

```
* First item
* Second item
** Nested item
** Another nested item
* Third item
```

排序后的列表

为顺序步骤或优先级项目创建编号列表：

```
. First step
. Second step
.. Substep 1
.. Substep 2
. Third step
```

链接

有关如何正确设置链接格式的详细信息，请参阅the section called “插入链接”。不支持 Markdown 样式的链接。

代码示例

内联代码

使用反引号表示内联代码：

```
Use the `kubectl get pods` command to list all pods.
```

代码块

创建具有语法突出显示并支持属性的代码块（类似于实体）：

```
[source,python,subs="verbatim,attributes"]
----
def hello_world():
    print("Hello, World!")
----
```

映像

插入带有替代文本的图像以实现可访问性：

```
image::images/image-file.png[Alt text description]
```

表

创建表格来整理信息：

```
[%header,cols="2"]
|===
|Header 1
|Header 2

|Cell 1,1
|Cell 1,2

|Cell 2,1
|Cell 2,2
|===
```

有关更复杂的表格，请参阅 AsciiDoc 表格文档。

标注

使用标注突出显示重要信息和警示内容：

```
NOTE: This is a note callout for general information.

TIP: This is a tip callout for helpful advice.

IMPORTANT: This is an important callout for critical information.
```

预览：

> **Note**
>
> 这是一个注释标注。

包含其他文件

包含来自其他文件的内容：

```
include::filename.adoc[]
```

属性（类似于实体）

使用预定义属性来保持一致性。特别是，必须使用 Amazon 和 `arn:aws-cn:` 的属性。

```
{aws} provides Amazon EKS as a managed Kubernetes service.
```

```
[source,bash,subs="verbatim,attributes"]
----
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn {arn-aws}iam::aws:policy/AmazonEKSClusterPolicy
----
```

如需属性列表，请在 `../attributes.txt` 文件中查看。

过程

格式化分步过程：

```
To create an Amaozon EKS cluster. do the following steps.

. Sign in to the {aws} Management Console.
. Open the Amazon EKS console.
. Choose *Create cluster*.

...
```