亚马逊云科技

用户指南

Amazon 私有证书颁发机构

版本 latest

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

Amazon 私有证书颁发机构: 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

什么是 Amazon 私有 CA？ 1
区域可用性 1
集成服务 2
支持的算法 2
符合 RFC 5280 3
定价 4
的术语和概念 Amazon 私有 CA 4
信任 5
TLS 服务器证书 5
证书签名 5
证书颁发机构 6
根 CA 6
CA 证书 6
根 CA 证书 7
终端实体证书 7
自签名证书 7
私有证书 8
证书路径 9
路径长度约束 9
什么是我需要的最佳证书服务？ 10
最佳实践 11
记录 CA 结构和策略 11
尽可能减少对根 CA 的使用 11
给根 CA 自己的 CA Amazon Web Services 账户 12
管理员和颁发者角色分开 12
实施证书的托管吊销 12
开启 Amazon CloudTrail 12
管理 CA 密钥和证书 13
延长 CA 有效期 13
轮换 CA 密钥 13
删除未使用的 CAs 13
阻止公众访问您的 CRLs 13
Amazon EKS 应用程序最佳实践 14
Amazon 私有 CA 搭配使用 适用于 Java 的 Amazon SDK 15

API 示例 15
以编程方式创建并激活根 CA 16
以编程方式创建并激活从属 CA 25
CreateCertificateAuthority 34
CreateCertificateAuthority 使用支持活动目录 35
CreateCertificateAuthorityAuditReport 44
CreatePermission 46
DeleteCertificateAuthority 49
DeletePermission 51
DeletePolicy 53
DescribeCertificateAuthority 55
DescribeCertificateAuthorityAuditReport 57
GetCertificate 60
GetCertificateAuthorityCertificate 62
GetCertificateAuthorityCsr 64
GetPolicy 67
ImportCertificateAuthorityCertificate 69
IssueCertificate 72
ListCertificateAuthorities 75
ListPermissions 79
ListTags 81
PutPolicy 83
RestoreCertificateAuthority 86
RevokeCertificate 87
TagCertificateAuthorities 89
UntagCertificateAuthority 92
UpdateCertificateAuthority 94
使用自定义主题名称创建 CAs 和证书 96
使用自定义扩展创建证书 104
问题示例 119
激活产品认证机构 (PAA) 120
激活产品认证中间体 (PAI) 130
创建设备认证证书 (DAC) 141
激活节点操作证书 (NOC) 的根 CA。 145
为节点操作证书 (NOC) 激活从属 CA 155
创建节点操作证书 (NOC) 165

mdL 示例 170
激活颁发机构证书颁发机构 (IACA) 证书 170
创建文档签名者证书 179
设计您的解决方案 Amazon 私有 CA 184
设计 CA 层次结构 184
验证最终实体证书 186
规划 CA 层次结构的结构 187
在认证路径上设置长度限制 190
管理 CA 生命周期 192
选择有效期 192
管理 CA 继任 193
撤销 CA 195
计划吊销证书 195
要求 196
设置 CRL 197
自定义 OCSP 网址 203
CA 模式 205
通用型（默认） 206
短期证书 206
做好韧性计划 206
冗余和灾难恢复 207
证书颁发机构 208
设置 209
注册获取 Amazon Web Services 账户 209
保护 IAM 用户 209
安装 Amazon Command Line Interface 210
创建私有 CA 210
CLI 示例 217
安装 CA 证书 228
兼容的签名算法 229
安装根 CA 证书 231
安装由托管的从属 CA 证书 Amazon 私有 CA 238
安装由外部父 CA 签名的从属 CA 证书 239
控制 访问 239
为 IAM 用户创建单账户权限 240
附加跨账户存取策略 242

列出私有 CA 244
查看私有 CA 246
添加标签 249
CA 状态 251
CA 状态与 CA 生命周期之间的关系 253
更新 CA 254
更新 CA（控制台） 254
更新 CA（CLI） 257
删除 CA 265
还原 CA 266
还原私有 CA（控制台） 266
恢复私有 CA (Amazon CLI) 267
外部签名的 CA 证书 268
颁发和管理证书 272
颁发私有终端实体证书 272
颁发标准证书 (Amazon CLI) 273
使用 APIPassthrough 模板颁发带有自定义主题名称的证书 275
使用 APIPassthrough 模板颁发带有自定义扩展名的证书 278
检索私有证书 279
列出私有证书 280
导出证书 285
吊销私有证书 285
已吊销的证书和 OCSP 287
CRL 中的已吊销证书 287
审核报告中的已吊销证书 288
自动导出 289
证书模板 289
模板种类 289
模板操作顺序 300
模板定义 301
安全性 340
IAM 340
API 权限 341
Amazon 托管策略 346
客户托管策略 348
内联策略 350

Cross-account 访问 355
客户托管权限 356
Resource-based 政策 356
数据保护 360
Amazon 私有 CA 私钥的存储和安全合规性 361
活动目录 Amazon 私有 CA 连接器中的数据加密 361
合规性验证 361
创建审计报告 362
基础结构安全性 369
VPC 终端节点 (Amazon PrivateLink) 369
Dual-stack 端点支持 373
在 IAM 中使用 IPv6 地址和 Amazon 私有 CA 373
CP/CPS 375
CP/CPS 要求和责任 375
监控 资源 383
Amazon 私有 CA CloudWatch 指标 383
Amazon 私有 CA 使用 CloudWatch 事件进行监控 385
创建私有 CA 时成功或失败 385
颁发证书时成功或失败 386
吊销证书时成功 387
生成 CRL 时成功或失败 388
创建 CA 审计报告时成功或失败 390
CloudTrail 日志 391
Amazon 私有 CA 信息在 CloudTrail 391
Amazon 私有 CA 管理事件 392
示例 Amazon 私有 CA 事件 393
故障排除 396
证书吊销问题 396
OCSP 响应延迟 396
撤销自签名证书 396
异常消息 396
符合 Matter 的证书错误 398
使用 Kubernetes 保护 Amazon 私有 CA 401
概念 401
注意事项 403
跨账户使用证书管理器 403

开始使用 404
安装证书管理器 407
配置 IAM 权限 407
安装和配置 Amazon 私有 CA 集群发行者 410
使用证书管理器管理 Amazon 私有 CA 客户证书 414
颁发您的第一个 TLS 证书 415
示例 416
监控 416
故障排除 417
活动目录连接器 361
您是初次接触 Connector for AD 的用户吗？ 419
AD 接入连接器 419
定价 419
设置 420
步骤 1：使用创建私有 CA Amazon 私有 CA 420
步骤 2：设置活动目录 420
（仅限 Active Directory 连接器）第 3 步：将权限委托给服务帐户 420
步骤 4：创建 IAM 策略 421
第 5 步：与 Connector for AD 共享您的私有 CA 424
步骤 6：创建目录注册 424
步骤 7：配置安全组 425
步骤 8：为目录对象配置网络访问权限 425
开始使用 426
开始前的准备工作 426
步骤 1：创建连接器 426
步骤 2：配置微软 Active Directory 策略 426
步骤 3：创建模板 428
步骤 4：配置微软群组权限 428
活动目录的连接器 428
创建连接器 428
创建模板 430
更新模板 434
列出连接器 435
列出模板 436
查看连接器 437
查看模板 438

目录注册 440
模板访问控制条目 442
服务委托人名称 443
标签 444
与集成 EventBridge 445
如何为 AD 事件 EventBridge 路由连接器 445
用于 AD 事件的连接器 446
创建事件模式 446
接收事件 447
对活动目录连接器进行故障排除 447
AD 错误代码的连接器 447
连接器创建失败 451
创建 SPN 失败 454
模板更新问题 455
适用于 SCEP 的连接器 457
功能 457
如何开始使用适用于 SCEP 的连接器 458
相关服务 458
适用于 SCEP 的接入连接器 458
定价 458
概念 459
注意事项和限制 460
注意事项 460
限制 461
设置 461
步骤 1：创建一个 Amazon Identity and Access Management policy 461
步骤 2：创建私有 CA 463
步骤 3：创建资源共享 463
开始使用 464
开始前的准备工作 464
步骤 1：创建连接器 465
步骤 2：将连接器详细信息复制到 MDM 系统中 466
配置您的 MDM 系统 467
General-purpose 连接器 467
Amazon 私有 CA 适用于微软 Intune 的 SCEP 连接器 468
配置 Jamf Pro 469

配置微软 Intune 475
配置 Omnissa 工作空间 ONE 477
安全性 482
VPC 终端节点 (Amazon PrivateLink) 483
监控 486
自动使用 EventBridge 487
CloudTrail 日志 492
故障排除 500
HTTP 错误 500
客户端错误 514
服务配额 515
文档历史记录 516
早期更新 523
..... dxxiv

什么是 Amazon 私有 CA？

Amazon 私有 CA 允许创建私有证书颁发机构 (CA) 层次结构，包括根和下级结构 CAs，而无需运营本地 CA 的投资和维护成本。您的私有 CAs 可以颁发终端实体 X.509 证书，这些证书在以下场景中很有用：

- 创建加密的 TLS 通信通道
- 对用户、计算机、API 终端节点和 IoT 设备进行身份验证
- 加密签名代码
- 实施在线证书状态协议 (OCSP) 以获取证书吊销状态

Amazon 私有 CA 可以从 Amazon Web Services 管理控制台、使用 Amazon 私有 CA API 或使用 Amazon CLI。

主题

- 的地区可用性 Amazon 私有证书颁发机构
- 与之集成的服务 Amazon 私有证书颁发机构
- 中支持的加密算法 Amazon 私有证书颁发机构
- 符合 RFC 5280 的要求 Amazon 私有证书颁发机构
- 的定价 Amazon 私有证书颁发机构
- 的术语和概念 Amazon 私有 CA

的地区可用性 Amazon 私有证书颁发机构

与大多数 Amazon 资源一样，私有证书颁发机构 (CAs) 是区域资源。要 CAs 在多个区域中使用私有功能，您必须在这些区域 CAs 中创建您的。您不能在区域 CAs 之间复制私有内容。访问 Amazon Web Services 一般参考 中的 Amazon 区域和端点或 Amazon 区域表以查看 Amazon 私有 CA的区域可用性。

> **Note**
>
> ACM 目前在某些地区可用， Amazon 私有 CA 但尚未提供。

与之集成的服务 Amazon 私有证书颁发机构

如果您使用 Amazon Certificate Manager 请求私有证书，则可以将该证书与任何与 ACM 集成的服务相关联。这既适用于链接到 Amazon 私有 CA 根的证书，也适用于链接到外部根的证书。有关更多信息，请参阅《 Amazon Certificate Manager 用户指南》中的集成服务。

您还可以将私有 CAs 集成到亚马逊 Elastic Kubernetes Service 中，以便在 Kubernetes 集群内提供证书颁发服务。有关更多信息，请参阅 使用 Kubernetes 保护 Amazon 私有证书颁发机构。

> **Note**
>
> Amazon Elastic Kubernetes Service 并非 ACM 集成服务。

如果您使用 Amazon 私有 CA API 或 Amazon CLI 颁发证书或从 ACM 导出私有证书，则可以将证书安装在所需的任何地方。

中支持的加密算法 Amazon 私有证书颁发机构

Amazon 私有 CA 支持以下用于私钥生成和证书签名的加密算法。

支持的算法

私有密钥算法	签名算法
ML_DSA_44	ML_DSA_44
ML_DSA_65	ML_DSA_65
ML_DSA_87	ML_DSA_87
RSA_2048	SHA256WITHRSA
RSA_3072	SHA384WITHRSA
RSA_4096	SHA512WITHRSA
EC_prime256v1	SHA256WITHECDSA
EC_secp384r1	SHA384WITHECDSA
	SHA512WITHECDSA

私有密钥算法	签名算法
ec_secp521r1	SM3WITHSM2
SM2（仅限中国地区）	

此列表仅适用于 Amazon 私有 CA 通过其控制台、API 或命令行直接颁发的证书。当使用来自的 CA Amazon Certificate Manager 颁发证书时 Amazon 私有 CA，它支持部分但不是全部算法。有关更多信息，请参阅 Amazon Certificate Manager 用户指南中的申请私有证书。

> **Note**
>
> 对于 RSA 或 ECDSA，指定的签名算法系列必须与 CA 私钥的密钥算法系列相匹配。
> 对于 ML-DSA，哈希函数被定义为算法本身的一部分。无法使用 ML-DSA 选择不同的哈希函数。为了保持与的向后兼容性 APIs，密钥算法和签名算法使用相同的值。

符合 RFC 5280 的要求 Amazon 私有证书颁发机构

Amazon 私有 CA 不强制执行 RFC 528 0 中定义的某些限制。相反的情况也是如此：强制实施某些适用于私有 CA 的附加约束。

强制实施

- “不迟于”日期。根据 RFC 5280， Amazon 私有 CA 防止颁发 `Not After` 日期晚于颁发 CA 证书的 `Not After` 日期的证书。
- 基本限制。 Amazon 私有 CA 在导入的 CA 证书中强制执行基本限制和路径长度。

  基本约束指示证书所标识的资源是否为 CA 并可以颁发证书。导入到 Amazon 私有 CA 的 CA 证书必须包含基本约束扩展，并且该扩展必须标记为 `critical`。除了`critical`旗帜外，还`CA=true`必须设置。 Amazon 私有 CA 由于以下原因而失败并出现验证异常，从而强制执行基本约束：

  - CA 证书中不包含该扩展。
  - 该扩展未标记为 `critical`。

  路径长度 (pathLenConstraint) 决定了导入的 CA 证书的下游 CAs 可能存在多少从属证书。 Amazon 私有 CA 由于以下原因，由于验证异常而失败，从而强制执行路径长度：

  - 导入 CA 证书将违反 CA 证书或链中任何 CA 证书中的路径长度约束。

  - 颁发证书将违反路径长度约束。
- 名称限制表示一个命名空间，认证路径中后续证书中的所有使用者名称都必须位于该命名空间内。限制适用于主题可分辨名称和主题备用名称。

未强制实施

- 证书政策。证书政策规定了 CA 颁发证书的条件。
- 禁止任何政策。用于颁发给的证书CAs。
- 发行人备用名称。允许将其他身份与 CA 证书的颁发者相关联。
- 政策限制。这些约束限制 CA 颁发从属 CA 证书的能力。
- 策略映射。用于 CA 证书。列出一对或多对 OIDs；每对包括 a issuerDomainPolicy 和 a subjectDomainPolicy。
- 主题目录属性。用于传达拍摄对象的识别属性。
- 主题信息访问。如何访问包含扩展程序的证书主体的信息和服务。
- 主题密钥标识符 (SKI) 和授权密钥标识符 (AKI)。RFC 需要 CA 证书才能包含 SKI 扩展。CA 颁发的证书必须包含与 CA 证书的 SKI 匹配的 AKI 扩展名。 Amazon 不强制执行这些要求。如果您的 CA 证书不包含 SKI，则颁发的终端实体或从属 CA 证书 AKI 将改为颁发者公有密钥的 SHA-1 哈希。
- SubjectPublicKeyInfo和主题备用名称 (SAN)。颁发证书时，无需执行验证，即可从提供的 CSR 中 Amazon 私有 CA 复制 SubjectPublicKeyInfo 和 SAN 扩展。

的定价 Amazon 私有证书颁发机构

从您创建私有 CA 的时间开始，每月将为每个私有 CA 向您的账户收取费用。您还需要为您颁发的每个证书付费。此费用包括您从 ACM 导出的证书和通过 Amazon 私有 CA API 或 Amazon 私有 CA CLI 创建的证书。删除私有 CA 后，您无需再为其付费。但是，如果您还原私有 CA，则需支付删除到还原期间内的费用。您无法访问其私有密钥的私有证书是免费的。其中包括用于集成服务（例如 Elastic Load Balancing 和 API Gateway）的证书。 CloudFront

有关最新的定 Amazon 私有 CA 价信息，请参阅Amazon 私有证书颁发机构 定价。您也可以使用定 Amazon 价计算器来估算成本。

的术语和概念 Amazon 私有 CA

以下术语和概念可以在您使用时为您提供帮助 Amazon 私有证书颁发机构。

主题

- 信任
- TLS 服务器证书
- 证书签名
- 证书颁发机构
- 根 CA
- CA 证书
- 根 CA 证书
- 终端实体证书
- 自签名证书
- 私有证书
- 证书路径
- 路径长度约束

信任

要让 Web 浏览器信任网站的身份，该浏览器必须能够验证网站的证书。不过，浏览器仅信任称为“CA 根证书”的少量证书。称为证书颁发机构 (CA) 的可信第三方将验证该网站的身份并向网站运营商颁发签名的数字证书。随后，浏览器可以检查数字签名以验证网站的身份。如果验证成功，浏览器会在地址栏中显示一个锁定图标。

TLS 服务器证书

HTTPS 事务需要服务器证书来对服务器进行身份验证。服务器证书是 X.509 v3 数据结构，用于将证书中的公有密钥绑定到证书的使用者。TLS 证书由证书颁发机构（CA）签名。该证书包含服务器的名称、有效期、公有密钥、签名算法等。

证书签名

数字签名是证书的已加密哈希。签名用于确认证书数据的完整性。您的私有 CA 使用加密哈希函数（例如可变大小的证书 SHA256 内容）创建签名。此哈希函数会生成一个实际上不可伪造的固定大小的数据字符串。此字符串称为哈希。然后，CA 使用其私有密钥对哈希值进行加密，并将已加密的哈希与证书相连接。

证书颁发机构

证书颁发机构 (CA) 颁发和 (如有必要) 吊销数字证书。最常见的证书类型基于 ISO X.509 标准。X.509 证书确认证书主题的身份并将该身份绑定到公有密钥。主题可以是用户、应用程序、计算机或其他设备。CA 可以对证书进行签名，方法是对内容进行哈希处理，然后使用与证书中的公有密钥相关的私有密钥对哈希进行加密。需要确认主题的身份的客户端应用程序 (如 Web 浏览器) 使用公有密钥来解密证书签名。然后，它对证书内容进行哈希处理，并将哈希值与已解密的签名进行比较来确定它们是否匹配。有关证书签名的信息，请参阅证书签名。

您可以使用 Amazon 私有 CA 创建私有 CA 并使用私有 CA 颁发证书。您的私有 CA 仅颁 SSL/TLS 发私有证书供组织内部使用。有关更多信息，请参阅 私有证书。您的私有 CA 还需要证书，然后才能使用它。有关更多信息，请参阅 CA 证书。

根 CA

颁发证书时可依据的加密构建数据块和信任根。它由用于签署（颁发）证书的私有密钥和标识根 CA 并将私有密钥绑定到 CA 名称的根证书组成。根证书将分发到环境中每个实体的信任存储区。管理员构建信任存储库，使其仅 CAs 包含他们信任的存储库。管理员将信任存储区更新或构建到其环境中实体的操作系统、实例和主机映像中。资源尝试相互连接时，它们会检查每个实体提供的证书。客户端检查证书的有效性，以及是否存在从证书到信任存储中安装的根证书的链。如果满足这些条件，资源之间就会完成“握手”。该握手以加密方式证明每个实体彼此的身份，并在它们之间创建加密的通信通道 (TLS/SSL)。

CA 证书

证书颁发机构 (CA) 证书确认 CA 的身份并将它绑定到证书中包含的公有密钥。

您可以使用 Amazon 私有 CA 创建私有根 CA 或私有从属 CA 证书。从属 CA 证书由信任链中较高的另一个 CA 证书签名。但是，对于根 CA，证书是自签名的。您还可以建立外部根颁发机构（例如，在本地托管）。然后，您可以使用根颁发机构对 Amazon 私有 CA托管的从属根 CA 证书进行签名。

以下示例显示了 Amazon 私有 CA X.509 CA 证书中包含的典型字段。请注意，对于 CA 证书，`CA:` 字段中的 `Basic Constraints` 值设置为 `TRUE`。

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4121 (0x1019)
    Signature Algorithm: sha256WithRSAEncryption
```

```
        Issuer: C=US, ST=Washington, L=Seattle, O=Example Company Root CA, OU=Corp,
CN=www.example.com/emailAddress=corp@www.example.com
        Validity
            Not Before: Feb 26 20:27:56 2018 GMT
            Not After : Feb 24 20:27:56 2028 GMT
        Subject: C=US, ST=WA, L=Seattle, O=Examples Company Subordinate CA,
OU=Corporate Office, CN=www.example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c0: ... a3:4a:51
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                F8:84:EE:37:21:F2:5E:0B:6C:40:C2:9D:C6:FE:7E:49:53:67:34:D9
            X509v3 Authority Key Identifier:
                keyid:0D:CE:76:F2:E3:3B:93:2D:36:05:41:41:16:36:C8:82:BC:CB:F8:A0

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         6:bb:94: ... 80:d8
```

根 CA 证书

证书颁发机构 (CA) 通常存在于一个分层结构中，该分层结构包含多个其他机构，它们之间 CAs 有明确定义的父子关系。子女或下属 CAs 由其父母进行认证 CAs，从而创建证书链。位于层次结构顶部的 CA 称为“根 CA”，而其证书称为“根证书”。此证书通常是自签名的。

终端实体证书

终端实体证书可标识服务器、实例、容器或设备等资源。与 CA 证书不同，终端实体证书不能用于颁发证书。终端实体证书的其他常用术语是 “客户端” 或 “叶子” 证书。

自签名证书

由颁发者而不是更高级别 CA 签名的证书。与由 CA 维护的安全根颁发的证书不同，自签名证书充当自己的根，因此其有很大的局限性：这些证书可用于提供在线加密，但不能用于验证身份，并且无法吊销。从安全角度来看，它们是不可接受的。但是，组织仍然使用它们，因为它们易于生成，不需要专业

知识或基础设施，而且为许多应用程序所接受。市场上没有针对颁发自签名证书的相应控制体系。使用这些证书的组织会因证书到期产生更大的中断风险，因为它们无法跟踪到期日期。

私有证书

Amazon 私有 CA 证书是可在组织内使用的私有 SSL/TLS 证书，但在公共 Internet 上不受信任。使用这些证书来标识资源，如客户端、服务器、应用程序、服务、设备和用户。在建立安全加密的通信通道时，每个资源都使用证书 (如下所示) 以及加密技术来向另一个资源证明其身份。内部 API 终端节点、Web 服务器、VPN 用户、IoT 设备和许多其他应用程序使用私有证书来建立其安全操作所需的加密的通信通道。默认情况下，私有证书不是公开信任的。内部管理员必须显式配置应用程序才能信任私有证书并分发证书。

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            e8:cb:d2:be:db:12:23:29:f9:77:06:bc:fe:c9:90:f8
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=WA, L=Seattle, O=Example Company CA, OU=Corporate,
 CN=www.example.com
        Validity
            Not Before: Feb 26 18:39:57 2018 GMT
            Not After : Feb 26 19:39:57 2019 GMT
        Subject: C=US, ST=Washington, L=Seattle, O=Example Company, OU=Sales,
 CN=www.example.com/emailAddress=sales@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00...c7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:AA:6E:C1:8A:EC:2F:8F:21:BC:BE:80:3D:C5:65:93:79:99:E7:71:65

            X509v3 Subject Key Identifier:
                C6:6B:3C:6F:0A:49:9E:CC:4B:80:B2:8A:AB:81:22:AB:89:A8:DA:19
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
```

```
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://NA/crl/12345678-1234-1234-1234-123456789012.crl

    Signature Algorithm: sha256WithRSAEncryption
         58:32:...:53
```

证书路径

依赖证书的客户端会验证是否存在从终端实体证书（可能通过中间证书链）到受信任根的路径。该客户端会检查路径上的每个证书是否均有效（未吊销）。它还会检查终端实体证书是否未到期、是否完整（未被篡改或修改），以及是否强制实施了证书中的约束。

路径长度约束

CA 证书pathLenConstraint的基本限制设置了其下方链中可能存在的从属 CA 证书的数量。例如，路径长度约束为零的 CA 证书不能有任何从属证书 CAs。路径长度约束为 1 的 CA CAs 下方最多可以有一个从属级别。RFC 5280 将其定义为 “在有效认证路径中可以跟随此证书的最大 non-self-issued中间证书数量”。路径长度值不包括终端实体证书，尽管关于验证链“长度”或“深度”的非正式语言可能包括它...导致混乱。

什么是我需要的最佳证书服务？

有两种 Amazon 服务用于颁发和部署 X.509 证书。选择最适合您需求的一种服务。注意事项包括您是否需要面向公共或私人的证书、自定义证书、要部署到其他 Amazon 服务的证书，还是需要自动证书管理和续订。

1. Amazon 私有 CA - 此服务适用于在 Amazon 云中构建公有密钥基础设施 (PKI) 的企业客户，并且供组织内部私人使用。使用 Amazon 私有 CA，您可以创建自己的 CA 层次结构并使用它颁发证书，用于对内部用户、计算机、应用程序、服务、服务器和其他设备进行身份验证以及对计算机代码进行签名。私有 CA 颁发的证书仅在您的组织内受信任，而在互联网上不受信任。

   创建私有 CA 后，您可以直接颁发证书（即，无需从第三方 CA 获得验证），并可以自定义证书以满足您组织的内部需求。例如，您可能需要：

   - 创建具有任何主题名称的证书。
   - 创建具有任何到期日期的证书。
   - 使用任何受支持的私有密钥算法和密钥长度。
   - 使用任何受支持的签名算法。
   - 使用模板控制证书颁发。

   此服务正好适用于您。要开始使用，请登录 https://console.aws.amazon.com/acm-pca/ 控制台。
2. Amazon Certificate Manager (ACM)-此服务为需要使用 TLS 建立公众信任的安全网站的企业客户管理证书。您可以将 ACM 证书部署到 Ela Amazon stic Load Balancing、Amazon CloudFront、Amazon API Gateway 和其他集成服务中。此类最常见的应用是一个需要大量流量的安全公共网站。

   借助此服务，您可以使用由 ACM 提供的证书（ACM 证书）（公有）或您导入到 ACM 的证书。如果您使用 Amazon 私有 CA 创建 CA，ACM 可以管理该私有 CA 的证书颁发并自动续订证书。

   有关更多信息，请参阅 Amazon Certificate Manager 用户指南。

Amazon 私有 CA 最佳实践

最佳实践是可以帮助您 Amazon 私有 CA 有效使用的建议。以下最佳实践基于当前 Amazon Certificate Manager 和 Amazon 私有 CA 客户的实际经验。

记录 CA 结构和策略

Amazon 建议记录您在运营 CA 时的所有策略和实践。这可能包括：

- 关于 CA 结构的决策的推理
- 一张显示你 CAs 和他们之间关系的图表
- 关于 CA 有效期的策略
- CA 继承规划
- 关于路径长度的策略
- 权限目录
- 管理控制结构说明
- 安全性

您可以在称为“认证策略 (CP)”和“认证实践声明 (CPS)”的这两个文档中捕获这些信息。有关捕获关于 CA 操作的重要信息的框架，请参阅 RFC 3647。

尽可能减少对根 CA 的使用

一般而言，根 CA 只能用于为中间证书颁发证书 CAs。这允许将根 CA 存储在远离危险的地方，而中间证书可以 CAs执行签发最终实体证书的日常任务。

但是，如果您的组织目前的做法是直接从根 CA 颁发最终实体证书，则 Amazon 私有 CA 可以在改善安全性和运营控制的同时支持此工作流程。在这种情况下，颁发终端实体证书需要 IAM 权限策略，该策略允许您的根 CA 使用终端实体证书模板。有关 IAM 策略的信息，请参阅 身份和访问管理 (IAM) Access Management 适用于 Amazon 私有证书颁发机构。

> **Note**
>
> 此配置施加了可能带来操作挑战的限制。例如，如果您的根 CA 被破坏或丢失，则必须创建一个新的根 CA 并将其分发给您环境中的所有客户端。在此恢复过程完成之前，您将无法颁发新

证书。直接从根 CA 颁发证书还可以防止您限制访问和限制从根 CA 颁发的证书的数量，这都是管理根 CA 的最佳实践。

给根 CA 自己的 CA Amazon Web Services 账户

在两个不同的 Amazon 账户中创建根 CA 和从属 CA 是推荐的最佳做法。这样做可以为您的根 CA 提供更多保护和访问控制。为此，您可以从一个账户中的从属 CA 导出 CSR，然后使用另一个账户中的根 CA 对其进行签名。这种方法的好处是，您可以 CAs 按账户单独控制自己的控制权。缺点是您无法使用向 Amazon Web Services 管理控制台 导来简化从根 CA 签署从属 CA 的 CA 证书的过程。

Important

我们强烈建议您在访问时使用多因素身份验证 (MFA)。 Amazon 私有 CA

管理员和颁发者角色分开

CA 管理员角色应与只需要颁发终端实体证书的用户分开。如果您的 CA 管理员和证书颁发者住在同一个地方 Amazon Web Services 账户，则可以通过专门为此目的创建 IAM 用户来限制颁发者的权限。

实施证书的托管吊销

证书被吊销后，托管吊销会自动向证书客户发送通知。如果证书的加密信息已泄露或颁发有误，则可能需要吊销证书。客户通常拒绝接受已撤销的证书。 Amazon 私有 CA 为托管吊销提供了两个标准选项：在线证书状态协议 (OCSP) 和证书吊销列表 ()。CRLs有关更多信息，请参阅 规划您的 Amazon 私有 CA 证书吊销方法。

开启 Amazon CloudTrail

在创建和开始操作私有 CA 之前，请开启 CloudTrail 日志功能。借 CloudTrail助，您可以检索账户 Amazon 的 API 调用历史记录，以监控您的 Amazon 部署。此历史记录包括从 Amazon Web Services 管理控制台、 Amazon Command Line Interface、和更高级别的 Amazon SDKs Amazon 服务发出的 API 调用。您还可以确定哪些用户和账户调用了 PCA API 操作、发出调用的源 IP 地址以及发生调用的时间。您可以使用 API CloudTrail 集成到应用程序中，为您的组织自动创建跟踪，检查跟踪的状态，并控制管理员如何开启和关闭 CloudTrail 登录功能。有关更多信息，请参阅创建跟踪。转使用记录 Amazon 私有证书颁发机构 API 调用 Amazon CloudTrail至查看 Amazon 私有 CA 操作的示例路径。

管理 CA 密钥和证书

您可以通过延长私有 CA 的有效期或轮换 CA 密钥来管理私有 CA 的生命周期。

延长 CA 有效期

您可以通过导入到期日期更长的新 CA 证书来延长 CA 的有效期。

轮换 CA 密钥

使用新的 CA 替换私有 CA，定期轮换 CA 密钥。创建新 CA 后，将应用程序和服务中对 CA ARN 的所有引用更新为新创建的 CA ARN，并使用新的 CA 证书更新基础架构中的信任存储。

> **Note**
>
> 如果您更换 CA 本身，请注意 CA 的 ARN 会发生变化。您的自动化系统中硬编码的 ARN 参考文献将会中断。

删除未使用的 CAs

您可以永久删除私有 CA。如果您不再需要 CA 或者要将其替换为具有较新私有密钥的 CA，您可能要执行此操作。要安全删除 CA，我们建议您执行 删除私有 CA 中所述的流程。

> **Note**
>
> Amazon 在 CA 被删除之前向您收费。

阻止公众访问您的 CRLs

Amazon 私有 CA 建议对包含以下内容的存储桶使用 Amazon S3 阻止公共访问 (BPA) 功能。 CRLs这样可以避免不必要地将您的私有 PKI 的详细信息暴露给潜在的对手。BPA 是 S3 的最佳实践，在新桶上默认处于启用状态。在某些情况下，需要进行其他设置。有关更多信息，请参阅 使用启用 S3 阻止公共访问 (BPA) CloudFront。

Amazon EKS 应用程序最佳实践

使用 Amazon 私有 CA 为 Amazon EKS 配置 X.509 证书时，请按照《A mazon EK S 最佳实践指南》中有关保护多租户环境的建议。有关将 Amazon 私有 CA 与 Kubernetes 集成的一般信息，请参阅 使用 Kubernetes 保护 Amazon 私有证书颁发机构。

Amazon 私有 CA 搭配使用 适用于 Java 的 Amazon SDK

您可以使用 Amazon 私有证书颁发机构 API 通过发送 HTTP 请求以编程方式与服务进行交互。该服务会返回 HTTP 响应。有关更多信息，请参阅《Amazon 私有证书颁发机构 API 参考》https://docs.amazonaws.cn/privateca/latest/APIReference/。

除了 HTTP API 之外，您还可以使用 Amazon SDKs 和命令行工具与之交互 Amazon 私有 CA。建议通过 HTTP API 进行此交互。有关更多信息，请参阅用于 Amazon Web Services 的工具。以下主题向您演示如何使用 适用于 Java 的 Amazon SDK 为 Amazon 私有 CA API 编程。

GetCertificateAuthorityCsrGetCertificate、和DescribeCertificateAuthorityAuditReport操作为服务员提供支持。您可以使用 Waiter 根据某些资源的存在性或状态来控制代码的进度。有关更多信息，请参阅以下主题以及Amazon 开发者博客 适用于 Java 的 Amazon SDK中的 “服务员”。

Amazon 私有 CA API 示例

以下代码示例展示了如何将选定 Amazon 私有 CA API 操作和数据类型与配合使用 适用于 Java 的 Amazon SDK。

主题

- 以编程方式创建并激活根 CA
- 以编程方式创建并激活从属 CA
- CreateCertificateAuthority
- CreateCertificateAuthority 使用支持活动目录
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority
- DeletePermission
- DeletePolicy
- DescribeCertificateAuthority
- DescribeCertificateAuthorityAuditReport
- GetCertificate
- GetCertificateAuthorityCertificate
- GetCertificateAuthorityCsr

- GetPolicy
- ImportCertificateAuthorityCertificate
- IssueCertificate
- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthorities
- UntagCertificateAuthority
- UpdateCertificateAuthority
- 使用自定义主题名称创建 CAs 和证书
- 使用自定义扩展创建证书

以编程方式创建并激活根 CA

此 Java 示例展示了如何使用以下 Amazon 私有 CA API 操作激活根 CA：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.KeyStorageSecurityStandard;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
```

```
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region"; // Substitute your region here, e.g. "us-
west-2"

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setOrganization("Example Organization");
        subject.setOrganizationalUnit("Example");
        subject.setCountry("US");
        subject.setState("Virginia");
        subject.setLocality("Arlington");
        subject.setCommonName("www.example.com");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, crlConfigure, CAtype,
 client);
```

```
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
 format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com.cn/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
 configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
 client)

{
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);
```

```
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withRevocationConfiguration(revokeConfig);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

createCARequest.withKeyStorageSecurityStandard(KeyStorageSecurityStandard.CCPC_LEVEL_1_OR_HIGH

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Root CA Arn: " + rootCAArn);

        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<  GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
```

```
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws-cn:acm-pca:::template/RootCACertificate/
V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);
```

```
        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
```

```
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {
```

```
        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
```

```
}
```

以编程方式创建并激活从属 CA

此 Java 示例展示了如何使用以下 Amazon 私有 CA API 操作激活从属 CA：

- GetCertificateAuthorityCertificate
- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.KeyStorageSecurityStandard;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Objects;
```

```
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class SubordinateCAActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String rootCAArn = "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566";

        // Define the endpoint region for your sample.
```

```
        String endpointRegion = "region"; // Substitute your region here, e.g. "us-
west-2"

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setOrganization("Example Organization");
        subject.setOrganizationalUnit("Example");
        subject.setCountry("US");
        subject.setState("Virginia");
        subject.setLocality("Arlington");
        subject.setCommonName("www.example.com");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
 CAtype, client);
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
 rootCAArn, client);
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
 subordinateCAArn, client);

    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
```

```
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
 format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com.cn/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
 AWSACMPCA client)

{
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
```

```
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Retrieve and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Root CA Certificate / Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withRevocationConfiguration(revokeConfig);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

createCARequest.withKeyStorageSecurityStandard(KeyStorageSecurityStandard.CCPC_LEVEL_1_OR_HIGH

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
```

```
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Subordinate CA Arn: " + subordinateCAArn);

        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
```

```
        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws-cn:acm-pca:::template/
SubordinateCACertificate_PathLen0/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
```

```
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }
```

```
        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
```

```
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Subordinate CA certificate successfully imported.");
        System.out.println("Subordinate CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

CreateCertificateAuthority

以下 Java 示例显示了如何使用该CreateCerticateAuthority操作。

此操作可创建私有从属证书颁发机构 (CA)。您必须指定 CA 配置、吊销配置、CA 类型和可选的等幂令牌。

CA 配置可指定以下内容：

- 要用于创建 CA 私有密钥的算法名称和密钥大小
- CA 用来签署自己的证书签名请求和 OCSP 响应的签名算法类型 CRLs
- X.500 主题信息

CRL 配置可指定以下内容：

- CRL 有效期，以天为单位 (CRL 的有效期)
- 将包含 CRL 的 Amazon S3 桶
- CA 颁发的证书中包含的 S3 存储桶的 CNAME 别名

如果成功，此函数将返回 CA 的 Amazon 资源名称 (ARN)。

您的输出应类似于以下内容：

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

CreateCertificateAuthority 使用支持活动目录

以下 Java 示例显示了如何使用该CreateCerticateAuthority操作创建可安装在 Microsoft Active Directory (AD) 企业 NTAuth 商店中的 CA。

该操作使用自定义对象标识符 () 创建私有根证书颁发机构 (CAOIDs)。有关更多信息以及等效操作的 Amazon CLI 示例，请参阅创建用于 Active Directory 登录的 CA。

如果成功，此函数将返回 CA 的 Amazon 资源名称 (ARN)。

```
package com.amazonaws.samples.appstream;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
```

```
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
```

```
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.util.io.pem.PemReader;

import lombok.SneakyThrows;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // OID for Common Name
                .withValue("root CA"),
            new CustomAttribute()
                .withObjectIdentifier("0.9.2342.19200300.100.1.25") // OID for Domain
 Component
                .withValue("example"),
            new CustomAttribute()
                .withObjectIdentifier("0.9.2342.19200300.100.1.25") // OID for Domain
 Component
                .withValue("com")

        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);
```

```
        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }
```

```
   private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
       // Create the request object.
       CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
       createCARequest.withCertificateAuthorityConfiguration(configCA);
       createCARequest.withIdempotencyToken("123987");
       createCARequest.withCertificateAuthorityType(CAtype);

       // Create the private CA.
       CreateCertificateAuthorityResult createCAResult = null;
       try {
           createCAResult = client.createCertificateAuthority(createCARequest);
       } catch (InvalidArgsException ex) {
           throw ex;
       } catch (InvalidPolicyException ex) {
           throw ex;
       } catch (LimitExceededException ex) {
           throw ex;
       }

       // Retrieve the ARN of the private CA.
       String rootCAArn = createCAResult.getCertificateAuthorityArn();
       System.out.println("Root CA Arn: " + rootCAArn);

       return rootCAArn;
   }

   private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

       // Create the CSR request object and set the CA ARN.
       GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
       csrRequest.withCertificateAuthorityArn(rootCAArn);

       // Create waiter to wait on successful creation of the CSR file.
       Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
       try {
           getCSRWaiter.run(new WaiterParameters<>(csrRequest));
       } catch (WaiterUnrecoverableException e) {
           //Explicit short circuit when the recourse transitions into
           //an undesired state.
```

```
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/RootCACertificate/
V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
```

```
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);
```

```
        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {
```

```
        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

您的输出应类似于以下内容：

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

CreateCertificateAuthorityAuditReport

以下 Java 示例显示了如何使用该CreateCertificateAuthorityAuditReport操作。

该操作会创建一个审计报告，该报告会列出每次的证书颁发和吊销。该报告保存在您通过输入指定的 Amazon S3 桶中。您可以每 30 分钟生成一次新报告。

> **Note**
>
> 已颁发超过 1 亿份证书的证书颁发机构不支持生成审计报告。如果您为这样的 CA 调用此操作，则该服务会返回LimitExceededException。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.CreateCertificateAuthorityAuditReportRequest;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityAuditReportResult;

import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
```

```
public class CreateCertificateAuthorityAuditReport {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object and set the certificate authority ARN.
      CreateCertificateAuthorityAuditReportRequest req =
          new CreateCertificateAuthorityAuditReportRequest();

      // Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Specify the S3 bucket name for your report.
      req.setS3BucketName("your-bucket-name");

      // Specify the audit response format.
      req.setAuditReportResponseFormat("JSON");

      // Create a result object.
      CreateCertificateAuthorityAuditReportResult result = null;
```

```
        try {
           result = client.createCertificateAuthorityAuditReport(req);
        } catch (RequestInProgressException ex) {
           throw ex;
        } catch (RequestFailedException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidArgsException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        } catch (LimitExceededException ex) {
           throw ex;
        }

        String ID = result.getAuditReportId();
        String S3Key = result.getS3Key();

        System.out.println(ID);
        System.out.println(S3Key);

    }
}
```

您的输出应类似于以下内容：

```
58904752-7de3-4bdf-ba89-6953e48c3cc7
audit-report/16075838-061c-4f7a-b54b-49bbc111bcff/58904752-7de3-4bdf-
ba89-6953e48c3cc7.json
```

CreatePermission

以下 Java 示例显示了如何使用该CreatePermission操作。

该操作将私有 CA 的访问权限分配给指定的 Amazon 服务主体。可以向服务授予以下权限：从私有 CA 创建和检索证书，以及列出私有 CA 已授予的活动权限。要通过 ACM 自动续订证书，必须将 CA 中所有可能的权限（IssueCertificateGetCertificate、和ListPermissions）分配给 ACM 服务主体 (acm.amazonaws.com)。您可以通过调用函数找到 CA 的 ARN。ListCertificateAuthorities

创建权限后，您可以使用ListPermissions函数对其进行检查，也可以使用该DeletePermission函数将其删除。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.CreatePermissionRequest;
import com.amazonaws.services.acmpca.model.CreatePermissionResult;

import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.PermissionAlreadyExistsException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

import java.util.ArrayList;

public class CreatePermission {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
```

```
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      CreatePermissionRequest req =
          new CreatePermissionRequest();

      //  Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the permissions to give the user.
      ArrayList<String> permissions = new ArrayList<>();
      permissions.add("IssueCertificate");
      permissions.add("GetCertificate");
      permissions.add("ListPermissions");

      req.setActions(permissions);

      // Set the Principal.
      req.setPrincipal("acm.amazonaws.com");

      // Create a result object.
      CreatePermissionResult result = null;
      try {
         result = client.createPermission(req);
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (PermissionAlreadyExistsException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
```

```
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        }
    }
}
```

DeleteCertificateAuthority

以下 Java 示例显示了如何使用该DeleteCertificateAuthority操作。

此操作将删除您使用该CreateCertificateAuthority操作创建的私有证书颁发机构 (CA)。DeleteCertificateAuthority 操作要求您提供要删除的 CA 的 ARN。您可以通过调用操作来找到 ARN。ListCertificateAuthorities如果私有 CA 的状态为 CREATING 或 PENDING_CERTIFICATE，则可立即将其删除。但是，如果您已经导入此证书，则无法立即将其删除。必须先通过调用UpdateCertificateAuthority操作来禁用 CA，然后将Status参数设置为DISABLED。然后，您可以使用 DeleteCertificateAuthority 操作中的 PermanentDeletionTimeInDays 参数指定天数（7 到 30 天）。在此期间，私有 CA 可以还原到 disabled 状态。默认情况下，如果未设置 PermanentDeletionTimeInDays 参数，则还原期为 30 天。在此期间后，私有 CA 将被永久删除，无法还原。有关更多信息，请参阅 还原 CA。

有关向您展示如何使用该RestoreCertificateAuthority操作的 Java 示例，请参阅RestoreCertificateAuthority。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.DeleteCertificateAuthorityRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
```

```
public class DeleteCertificateAuthority {

   public static void main(String[] args) throws Exception{

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a requrest object and set the ARN of the private CA to delete.
      DeleteCertificateAuthorityRequest req = new DeleteCertificateAuthorityRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the recovery period.
      req.withPermanentDeletionTimeInDays(12);

      // Delete the CA.
      try {
         client.deleteCertificateAuthority(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
```

```
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      }
   }
}
```

DeletePermission

以下 Java 示例显示了如何使用该DeletePermission操作。

该操作会删除私有 CA 使用该CreatePermissions操作委托给 Amazon 服务委托人的权限。您可以通过调用函数找到 CA 的 ARN。ListCertificateAuthorities您可以通过调用ListPermissions函数来检查 CA 授予的权限。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.DeletePermissionRequest;
import com.amazonaws.services.acmpca.model.DeletePermissionResult;

import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class DeletePermission {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
```

```
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      DeletePermissionRequest req =
          new DeletePermissionRequest();

      //  Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the AWS service principal.
      req.setPrincipal("acm.amazonaws.com");

      // Create a result object.
      DeletePermissionResult result = null;
      try {
         result = client.deletePermission(req);
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
```

```
      } catch (ResourceNotFoundException ex) {
         throw ex;
      }
   }
}
```

DeletePolicy

以下 Java 示例显示了如何使用该DeletePolicy操作。

该操作删除附加到私有 CA 的基于资源的策略。基于资源的策略用于启用跨账户 CA 共享。您可以通过调用操作来找到私有 CA 的 ARN。ListCertificateAuthorities

相关的 API 操作包括PutPolicy和GetPolicy。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.DeletePolicyRequest;
import com.amazonaws.services.acmpca.model.DeletePolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.LockoutPreventedException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class DeletePolicy {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
```

```
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "us-west-2";  // Substitute your Region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        DeletePolicyRequest req = new DeletePolicyRequest();

        // Set the resource ARN.
        req.withResourceArn("arn:aws:acm-pca:us-west-2:111122223333:certificate-
authority/11223344-44ee-aa22-bb33-4cd2d13f1f18");

        // Retrieve a list of your CAs.
        DeletePolicyResult result = null;
        try {
            result = client.deletePolicy(req);
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (LockoutPreventedException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
```

```
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (AWSACMPCAException ex) {
            throw ex;
        }
    }
}
```

DescribeCertificateAuthority

以下 Java 示例显示了如何使用该DescribeCertificateAuthority操作。

此操作列出有关您的私有证书颁发机构 (CA) 的信息。您必须指定私有 CA 的 ARN (Amazon 资源名称)。输出包含 CA 的状态。这可以是以下任一种：

- CREATING— Amazon 私有 CA 正在创建您的私有证书颁发机构。
- PENDING_CERTIFICATE – 证书正在等待处理。您必须使用本地根或从属 CA 来签署您的私有 CA CSR，然后将其导入 PCA。
- ACTIVE – 您的私有 CA 处于活动状态。
- DISABLED – 您的私有 CA 已被禁用。
- EXPIRED – 您的私有 CA 证书已过期。
- FAILED – 无法创建您的私有 CA。
- DELETED – 您的私有 CA 处于还原期，经过该期限之后，它将被永久删除。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.CertificateAuthority;
import com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityResult;
```

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class DescribeCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object
      DescribeCertificateAuthorityRequest req = new
 DescribeCertificateAuthorityRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create a result object.
      DescribeCertificateAuthorityResult result = null;
      try {
         result = client.describeCertificateAuthority(req);
```

```
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      }

      // Retrieve and display information about the CA.
      CertificateAuthority PCA = result.getCertificateAuthority();
      String strPCA = PCA.toString();
      System.out.println(strPCA);
   }
}
```

DescribeCertificateAuthorityAuditReport

以下 Java 示例显示了如何使用该DescribeCertificateAuthorityAuditReport操作。

该操作列出了有关您通过调用该CreateCertificateAuthorityAuditReport操作创建的特定审计报告的信息。每次使用证书颁发机构 (CA) 私有密钥时，都会创建审核信息。当您颁发证书、签署 CRL 或吊销证书时，会使用私有密钥。

```
package com.amazonaws.samples;

import java.util.Date;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityAuditReportRequest;
import
 com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityAuditReportResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
```

```
import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class DescribeCertificateAuthorityAuditReport {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      DescribeCertificateAuthorityAuditReportRequest req =
            new DescribeCertificateAuthorityAuditReportRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the audit report ID.
      req.withAuditReportId("11111111-2222-3333-4444-555555555555");
```

```
      // Create waiter to wait on successful creation of the audit report file.
      Waiter<DescribeCertificateAuthorityAuditReportRequest> waiter =
 client.waiters().auditReportCreated();
      try {
         waiter.run(new WaiterParameters<>(req));
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
      } catch (AWSACMPCAException e) {
          //Unexpected service exception.
      }

      // Create a result object.
      DescribeCertificateAuthorityAuditReportResult result = null;
      try {
         result = client.describeCertificateAuthorityAuditReport(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      }

      String status = result.getAuditReportStatus();
      String S3Bucket = result.getS3BucketName();
      String S3Key = result.getS3Key();
      Date createdAt = result.getCreatedAt();

      System.out.println(status);
      System.out.println(S3Bucket);
      System.out.println(S3Key);
      System.out.println(createdAt);
   }
}
```

您的输出应类似于以下内容：

```
SUCCESS
your-audit-report-bucket-name
audit-report/a4119411-8153-498a-a607-2cb77b858043/25211c3d-f2fe-479f-b437-
fe2b3612bc45.json
```

```
Tue Jan 16 13:07:58 PST 2018
```

GetCertificate

以下 Java 示例显示了如何使用该GetCertificate操作。

此操作可从您的私有 CA 检索证书。当您调用该操作时，将返回证书的 ARN。IssueCertificate在调用 GetCertificate 操作时，您必须同时指定私有 CA 的 ARN 和已颁发证书的 ARN。如果证书处于 ISSUED 状态，则可以检索该证书。您可以调用该CreateCertificateAuthorityAuditReport操作来创建一份报告，其中包含有关您的私有 CA 颁发和吊销的所有证书的信息。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException ;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import com.amazonaws.services.acmpca.model.AWSACMPCAException;

public class GetCertificate {

   public static void main(String[] args) throws Exception{
```

```
      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      GetCertificateRequest req = new GetCertificateRequest();

      // Set the certificate ARN.
      req.withCertificateArn("arn:aws:acm-pca:region:account:certificate-
authority/CA_ID/certificate/certificate_ID");

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create waiter to wait on successful creation of the certificate file.
      Waiter<GetCertificateRequest> waiter = client.waiters().certificateIssued();
      try {
         waiter.run(new WaiterParameters<>(req));
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
```

```
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult result = null;
        try {
           result = client.getCertificate(req);
        } catch (RequestInProgressException ex) {
           throw ex;
        } catch (RequestFailedException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String strCert = result.getCertificate();
        System.out.println(strCert);
    }
}
```

对于证书颁发机构 (CA) 和您指定的证书，您的输出应该是类似如下的证书链。

```
-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----
```

GetCertificateAuthorityCertificate

以下 Java 示例显示了如何使用该GetCertificateAuthorityCertificate操作。

此操作可检索您的私有证书颁发机构 (CA) 的证书和证书链。证书和证书链均为 PEM 格式的 base64 PEM 编码字符串。证书链不包含 CA 证书。该链中的每个证书都对它前面的证书签名。

```
package com.amazonaws.samples;
```

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class GetCertificateAuthorityCertificate {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();
```

```
      // Create a request object
      GetCertificateAuthorityCertificateRequest req =
            new GetCertificateAuthorityCertificateRequest();

      // Set the certificate authority ARN,
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create a result object.
      GetCertificateAuthorityCertificateResult result = null;
      try {
         result = client.getCertificateAuthorityCertificate(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      }

      // Retrieve and display the certificate information.
      String strPcaCert = result.getCertificate();
      System.out.println(strPcaCert);
      String strPCACChain = result.getCertificateChain();
      System.out.println(strPCACChain);
   }
}
```

对于您指定的证书颁发机构 (CA)，您的输出应该是类似如下的证书和链。

```
-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----
```

GetCertificateAuthorityCsr

以下 Java 示例显示了如何使用该GetCertificateAuthorityCsr操作。

此操作可检索您的私有证书颁发机构 (CA) 的证书签名请求 (CSR)。CSR 是在您调用CreateCertificateAuthority操作时创建的。将 CSR 放至您的本地 X.509 基础设施，

并使用根或从属 CA 对其进行签名。然后通过调用操作将签名的证书重新导入 ACM PCA。ImportCertificateAuthorityCertificate以 PEM 格式的 base64 编码字符串形式返回 CSR。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class GetCertificateAuthorityCsr {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
```

```
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create the request object and set the CA ARN.
      GetCertificateAuthorityCsrRequest req = new GetCertificateAuthorityCsrRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create waiter to wait on successful creation of the CSR file.
      Waiter<GetCertificateAuthorityCsrRequest> waiter =
 client.waiters().certificateAuthorityCSRCreated();
      try {
         waiter.run(new WaiterParameters<>(req));
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
      } catch (AWSACMPCAException e) {
          //Unexpected service exception.
      }

      // Retrieve the CSR.
      GetCertificateAuthorityCsrResult result = null;
      try {
         result = client.getCertificateAuthorityCsr(req);
      } catch (RequestInProgressException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
```

```
            throw ex;
        }

        // Retrieve and display the CSR;
        String Csr = result.getCsr();
        System.out.println(Csr);
    }
}
```

对于您指定的证书颁发机构 (CA)，您的输出应类似如下。证书签名请求 (CSR) 采用 PEM 格式进行 base64 编码。将它保存到本地文件中，然后将它置于本地 X.509 基础设施，并使用根或从属 CA 对它进行签名。

```
-----BEGIN CERTIFICATE REQUEST----- base64-encoded request -----END CERTIFICATE
 REQUEST-----
```

GetPolicy

以下 Java 示例显示了如何使用该GetPolicy操作。

该操作检索附加到私有 CA 的基于资源的策略。基于资源的策略用于启用跨账户 CA 共享。您可以通过调用操作来找到私有 CA 的 ARN。ListCertificateAuthorities

相关的 API 操作包括PutPolicy和DeletePolicy。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.GetPolicyRequest;
import com.amazonaws.services.acmpca.model.GetPolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
```

```
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class GetPolicy {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        GetPolicyRequest req = new GetPolicyRequest();

        // Set the resource ARN.
        req.withResourceArn("arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566");

        // Retrieve a list of your CAs.
        GetPolicyResult result= null;
        try {
            result = client.getPolicy(req);
        } catch (InvalidArnException ex) {
```

```
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (AWSACMPCAException ex) {
            throw ex;
        }

        // Display the policy.
        System.out.println(result.getPolicy());
    }
}
```

ImportCertificateAuthorityCertificate

以下 Java 示例显示了如何使用该ImportCertificateAuthorityCertificate操作。

此操作将您签名的私有 CA 证书导入 Amazon 私有 CA。在调用此操作之前，必须先通过调用该CreateCertificateAuthority操作创建私有证书颁发机构。然后，您必须通过调用GetCertificateAuthorityCsr操作来生成证书签名请求 (CSR)。将该 CSR 放至本地 CA 并使用您的根或从属证书对其签名。创建证书链并将签名的证书和证书链复制到您的工作目录。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
```

```
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.RequestFailedException;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Objects;

public class ImportCertificateAuthorityCertificate {

   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
      }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();
```

```
      // Create the request object and set the signed certificate, chain and CA ARN.
      ImportCertificateAuthorityCertificateRequest req =
            new ImportCertificateAuthorityCertificateRequest();

      // Set the signed certificate.
      String strCertificate =
            "-----BEGIN CERTIFICATE-----\n" +
            "base64-encoded certificate\n" +
            "-----END CERTIFICATE-----\n";
      ByteBuffer certByteBuffer = stringToByteBuffer(strCertificate);
      req.setCertificate(certByteBuffer);

      // Set the certificate chain.
      String strCertificateChain =
            "-----BEGIN CERTIFICATE-----\n" +
            "base64-encoded certificate\n" +
            "-----END CERTIFICATE-----\n";
      ByteBuffer chainByteBuffer = stringToByteBuffer(strCertificateChain);
      req.setCertificateChain(chainByteBuffer);

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Import the certificate.
      try {
         client.importCertificateAuthorityCertificate(req);
      } catch (CertificateMismatchException ex) {
         throw ex;
      } catch (MalformedCertificateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (RequestInProgressException ex) {
         throw ex;
      } catch (ConcurrentModificationException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      }
   }
```

```
}
```

IssueCertificate

以下 Java 示例显示了如何使用该IssueCertificate操作。

此操作使用您的私有证书颁发机构（CA）来颁发终端实体证书。此操作将返回证书的 Amazon 资源名称 (ARN)。您可以通过调用GetCertificate并指定 ARN 来检索证书。

> **Note**
>
> 该IssueCertificate操作要求您指定证书模板。此示例使用 EndEntityCertificate/V1 模板。有关所有可用模板的信息，请参阅 使用 Amazon 私有 CA 证书模板。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
```

```
public class IssueCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
      }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
```

```
        String strCSR =
        "-----BEGIN CERTIFICATE REQUEST-----\n" +
        "base64-encoded certificate\n" +
        "-----END CERTIFICATE REQUEST-----\n";
        ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
        req.setCsr(csrByteBuffer);

        // Specify the template for the issued certificate.
        req.withTemplateArn("arn:aws:acm-pca:::template/EndEntityCertificate/V1");

        // Set the signing algorithm.
        req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(<<3650L>>);
        validity.withType("DAYS");
        req.withValidity(validity);

        // Set the idempotency token.
        req.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult result = null;
        try {
           result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidArgsException ex) {
           throw ex;
        } catch (MalformedCSRException ex) {
           throw ex;
        }

        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
```

```
}
```

您的输出应类似于以下内容：

```
arn:aws:acm-pca:region:account:certificate-authority/CA_ID/certificate/certificate_ID
```

ListCertificateAuthorities

以下 Java 示例显示了如何使用该ListCertificateAuthorities操作。

此操作列出了您使用该CreateCertificateAuthority操作创建的私有证书颁发机构 (CAs)。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ListCertificateAuthoritiesRequest;
import com.amazonaws.services.acmpca.model.ListCertificateAuthoritiesResult;
import com.amazonaws.services.acmpca.model.InvalidNextTokenException;

public class ListCertificateAuthorities {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from file.",
 e);
      }
```

```
      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create the request object.
      ListCertificateAuthoritiesRequest req = new ListCertificateAuthoritiesRequest();
      req.withMaxResults(10);

      // Retrieve a list of your CAs.
      ListCertificateAuthoritiesResult result= null;
      try {
          result = client.listCertificateAuthorities(req);
      } catch (InvalidNextTokenException ex) {
          throw ex;
      }

      // Display the CA list.
      System.out.println(result.getCertificateAuthorities());
   }
}
```

如果您有任何要列出的证书颁发机构，则输出应类似于以下内容：

```
[{
 Arn: arn: aws: acm-pca: region: account: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: TueNov0712: 05: 39PST2017,
 LastStateChangeAt: WedJan1012: 35: 39PST2018,
 Type: SUBORDINATE,
 Serial: 4109,
 Status: DISABLED,
 NotBefore: TueNov0712: 19: 15PST2017,
 NotAfter: FriNov0513: 19: 15PDT2027,
```

```
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Organization: ExampleCorp,
   OrganizationalUnit: HR,
   State: Washington,
   CommonName: www.example.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: true,
   ExpirationInDays: 3650,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: WedSep1312: 54: 52PDT2017,
 LastStateChangeAt: WedSep1312: 54: 52PDT2017,
 Type: SUBORDINATE,
 Serial: 4100,
 Status: ACTIVE,
 NotBefore: WedSep1314: 11: 19PDT2017,
 NotAfter: SatSep1114: 11: 19PDT2027,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Country: US,
   Organization: ExampleCompany,
   OrganizationalUnit: Sales,
   State: Washington,
   CommonName: www.example.com,
   Locality: Seattle,

  }
 },
```

```
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: false,
   ExpirationInDays: 5,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: FriJan1213: 57: 11PST2018,
 LastStateChangeAt: FriJan1213: 57: 11PST2018,
 Type: SUBORDINATE,
 Status: PENDING_CERTIFICATE,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Country: US,
   Organization: Examples-R-Us Ltd.,
   OrganizationalUnit: corporate,
   State: WA,
   CommonName: www.examplesrus.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: true,
   ExpirationInDays: 365,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: FriJan0511: 14: 21PST2018,
 LastStateChangeAt: FriJan0511: 14: 21PST2018,
 Type: SUBORDINATE,
```

```
 Serial: 4116,
 Status: ACTIVE,
 NotBefore: FriJan0512: 12: 56PST2018,
 NotAfter: MonJan0312: 12: 56PST2028,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Country: US,
   Organization: ExamplesLLC,
   OrganizationalUnit: CorporateOffice,
   State: WA,
   CommonName: www.example.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: true,
   ExpirationInDays: 3650,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
}]
```

ListPermissions

以下 Java 示例显示了如何使用该ListPermissions操作。

此操作列出您的私有 CA 已分配的权限（如果有）。权限（包括IssueCertificateGetCertificateListPermissions、和）可以通过操作分配给 Amazon 服务主体，也可以通过CreatePermission操作将其撤销。DeletePermissions

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ListPermissionsRequest;
import com.amazonaws.services.acmpca.model.ListPermissionsResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidNextTokenException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RequestFailedException;

public class ListPermissions {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object and set the CA ARN.
      ListPermissionsRequest req = new ListPermissionsRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
```

```
      // List the tags.
      ListPermissionsResult result = null;
      try {
         result = client.listPermissions(req);
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch(RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      }

      // Retrieve and display the permissions.
      System.out.println(result);
   }
}
```

如果指定的私有 CA 已向服务委托人分配权限，则输出应类似于以下内容：

```
[{
       Arn: arn:aws:acm-
pca:region:account:permission/12345678-1234-1234-1234-123456789012,
       CreatedAt: WedFeb0317: 05: 39PST2019,
       Prinicpal: acm.amazonaws.com,
       Permissions: {
            ISSUE_CERTIFICATE,
            GET_CERTIFICATE,
            DELETE,CERTIFICATE
       },
       SourceAccount: account
}]
```

ListTags

以下 Java 示例显示了如何使用该ListTags操作。

此操作可列出与您的私有 CA 关联的标签 (如果有)。标签是可用于识别和整理您的标签的标签 CAs。每个标签都由一个键和一个可选值组成。调用TagCertificateAuthority操作将一个或多个标签添加到您的 CA。调用该UntagCertificateAuthority操作以移除标签。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ListTagsRequest;
import com.amazonaws.services.acmpca.model.ListTagsResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class ListTags {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
```

```
            .build();

        // Create a request object and set the CA ARN.
        ListTagsRequest req = new ListTagsRequest();
        req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

        // List the tags
        ListTagsResult result = null;
        try {
            result = client.listTags(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        }

        // Retrieve and display the tags.
        System.out.println(result);
    }
}
```

如果您有任何要列出的标签，则输出应类似于以下内容：

```
{Tags: [{Key: Admin,Value: Alice}, {Key: Purpose,Value: WebServices}],}
```

PutPolicy

以下 Java 示例显示了如何使用该PutPolicy操作。

该操作将基于资源的策略附加到私有 CA，从而实现跨账户共享。获得政策授权后，居住在其他 Amazon 账户中的委托人可以使用其不拥有的私有 CA 签发和续订私有终端实体证书。您可以通过调用操作来找到私有 CA 的 ARN。ListCertificateAuthorities有关策略的示例，请参阅基于资源的策略的 Amazon 私有 CA 指导。

将策略附加到 CA 后，您可以使用GetPolicy操作对其进行检查，也可以使用DeletePolicy操作将其删除。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
```

```
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.PutPolicyRequest;
import com.amazonaws.services.acmpca.model.PutPolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.LockoutPreventedException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class PutPolicy {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
```

```
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        PutPolicyRequest req = new PutPolicyRequest();

        // Set the resource ARN.
        req.withResourceArn("arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566");

        // Import and set the policy.
        // Note: This code assumes the file "ShareResourceWithAccountPolicy.json" is in
 a folder titled policy.
        String policy = new String(Files.readAllBytes(Paths.get("policy",
 "ShareResourceWithAccountPolicy.json")));
        req.withPolicy(policy);

        // Retrieve a list of your CAs.
        PutPolicyResult result = null;
        try {
            result = client.putPolicy(req);
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LockoutPreventedException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (AWSACMPCAException ex) {
            throw ex;
        }
```

```
    }
}
```

RestoreCertificateAuthority

以下 Java 示例显示了如何使用该RestoreCertificateAuthority操作。私有 CA 在其还原期内可随时还原。当前，此期间可持续 7 到 30 天（自删除之日起），并且您可以在删除 CA 时对其进行定义。有关更多信息，请参阅 还原 CA。另请参阅 DeleteCertificateAuthority Java 示例。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.RestoreCertificateAuthorityRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class RestoreCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from file.",
 e);
      }

      // Define the endpoint for your sample.
```

```
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create the request object.
      RestoreCertificateAuthorityRequest req = new
 RestoreCertificateAuthorityRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Restore the CA.
      try {
          client.restoreCertificateAuthority(req);
      } catch (InvalidArnException ex) {
          throw ex;
      } catch (InvalidStateException ex) {
          throw ex;
      } catch (ResourceNotFoundException ex) {
          throw ex;
      }
   }
}
```

RevokeCertificate

以下 Java 示例显示了如何使用该RevokeCertificate操作。

此操作会吊销您通过调用该IssueCertificate操作颁发的证书。如果您在创建或更新私有 CA 时启用了证书吊销列表 (CRL)，则有关已吊销证书的信息将包含在 CRL 中。 Amazon 私有 CA 将 CRL 写入您指定的 Amazon S3 存储桶。有关更多信息，请参阅CrlConfiguration结构。

```
package com.amazonaws.samples;
```

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.RevokeCertificateRequest;
import com.amazonaws.services.acmpca.model.RevocationReason;

import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestAlreadyProcessedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;

public class RevokeCertificate {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
```

```
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create a request object.
        RevokeCertificateRequest req = new RevokeCertificateRequest();

        // Set the certificate authority ARN.
        req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

        // Set the certificate serial number.
        req.setCertificateSerial("79:3f:0d:5b:6a:04:12:5e:2c:9c:fb:52:37:35:98:fe");

        // Set the RevocationReason.
        req.withRevocationReason(RevocationReason.<<KEY_COMPROMISE>>);

        // Revoke the certificate.
        try {
            client.revokeCertificate(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestAlreadyProcessedException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
    }
}
```

TagCertificateAuthorities

以下 Java 示例显示了如何使用该TagCertificateAuthority操作。

此操作可向您的私有 CA 添加一个或多个标签。标签是可用于识别和组织 Amazon 资源的标签。每个标签都由一个键和一个可选值组成。调用此操作时，可以通过私有 CA 的 Amazon 资源名称 (ARN) 指定私有 CA。您使用键-值对指定标签。要标识该 CA 的特定特征，可以将标签仅应用于一个私有 CA。

或者，要筛选它们之间的共同关系 CAs，您可以将相同的标签应用于多个私有关系 CAs。要移除一个或多个标签，请使用UntagCertificateAuthority操作。调用ListTags操作以查看哪些标签与您的 CA 关联。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.TagCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.Tag;

import java.util.ArrayList;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidTagException;
import com.amazonaws.services.acmpca.model.TooManyTagsException;

public class TagCertificateAuthorities {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
```

```
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create a tag - method 1
        Tag tag1 = new Tag();
        tag1.withKey("Administrator");
        tag1.withValue("Bob");

        // Create a tag - method 2
        Tag tag2 = new Tag()
            .withKey("Purpose")
            .withValue("WebServices");

        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag1);
        tags.add(tag2);

        // Create a request object and specify the certificate authority ARN.
        TagCertificateAuthorityRequest req = new TagCertificateAuthorityRequest();
        req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
        req.setTags(tags);

        // Add a tag
        try {
            client.tagCertificateAuthority(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidTagException ex) {
            throw ex;
        } catch (TooManyTagsException ex) {
            throw ex;
        }
```

```
    }
}
```

UntagCertificateAuthority

以下 Java 示例显示了如何使用该UntagCertificateAuthority操作。

此操作可从私有 CA 中删除一个或多个标签。一个标签由一个键值对组成。如果您在调用此操作时未指定标签的值部分，则会删除标签，而不管值如何。如果您指定了值，则仅删除与指定值关联的标签。要向私有 CA 添加标签，请使用TagCertificateAuthority操作。调用ListTags操作以查看哪些标签与您的 CA 关联。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.util.ArrayList;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.UntagCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.Tag;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidTagException;

public class UntagCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
```

```
            throw new AmazonClientException("Cannot load your credentials from disk", e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create a Tag object with the tag to delete.
        Tag tag = new Tag();
        tag.withKey("Administrator");
        tag.withValue("Bob");

        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag);

        // Create a request object and specify the certificate authority ARN.
        UntagCertificateAuthorityRequest req = new UntagCertificateAuthorityRequest();
        req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
        req.withTags(tags);

        // Delete the tag
        try {
            client.untagCertificateAuthority(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidTagException ex) {
            throw ex;
        }
    }
```

```
}
```

UpdateCertificateAuthority

以下 Java 示例显示了如何使用该UpdateCertificateAuthority操作。

此操作可更新私有证书颁发机构 (CA) 的状态或配置。您的私有 CA 必须处于 ACTIVE 或 DISABLED 状态，您才能更新它。您可以禁用处于 ACTIVE 状态的私有 CA，或使处于 DISABLED 状态的 CA 再次变为活动的。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.UpdateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CertificateAuthorityStatus;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;

public class UpdateCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
```

```
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        UpdateCertificateAuthorityRequest req = new UpdateCertificateAuthorityRequest();

        // Set the ARN of the private CA that you want to update.
        req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

        // Define the certificate revocation list configuration. If you do not want to
        // update the CRL configuration, leave the CrlConfiguration structure alone and
        // do not set it on your UpdateCertificateAuthorityRequest object.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname("your-custom-name");
        crlConfigure.withS3BucketName("your-bucket-name");

        // Set the CRL configuration onto your UpdateCertificateAuthorityRequest object.
        // If you do not want to change your CRL configuration, do not use the
        // setCrlConfiguration method.
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);
        req.setRevocationConfiguration(revokeConfig);

        // Set the status.
        req.withStatus(CertificateAuthorityStatus.<<ACTIVE>>);
```

```
        // Create the result object.
        try {
            client.updateCertificateAuthority(req);
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        }
    }
}
```

使用自定义主题名称创建 CAs 和证书

该CustomAttribute对象允许管理员将自定义对象标识符 (OIDs) 传递给私有对象 CAs 和证书。自定义 OIDs 可用于创建专门的主题名称层次结构，以反映您的组织结构和需求。必须使用其中一个 ApiPassthrough 模板创建自定义证书。有关模板的更多信息，请参阅Amazon 私有 CA 模板品种。有关使用自定义属性的更多信息，请参阅 颁发私有终端实体证书 和 在中创建私有 CA Amazon 私有 CA。

不能将 StandardAttributes 与 CustomAttributes 结合使用。但是，您可以 OIDs 作为标准的一部分通过标准CustomAttributes。下表 OIDs 列出了默认主题名称：

使用者名称	对象 ID
国家/地区	2.5.4.6
CommonName	2.5.4.3
DistinguishedNameQualifier	2.5.4.46
GenerationQualifier	2.5.4.44

使用者名称	对象 ID
GivenName	2.5.4.42
Initials	2.5.4.43
区域	2.5.4.7
Organization（组织）	2.5.4.10
OrganizationalUnit	2.5.4.11
化名	2.5.4.65
SerialNumber	2.5.4.5
州	2.5.4.8
Surname	2.5.4.4
标题	2.5.4.12

主题

- 使用创建 CA CustomAttribute
- 颁发证书 CustomAttribute

使用创建 CA CustomAttribute

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
```

```
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;


public class CreateCertificateAuthorityWithCustomAttributes {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                   "Cannot load the credentials from the credential profiles file. " +
                   "Please make sure that your credentials file is at the correct " +
                   "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
 format.",
                   e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
```

```
        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.6") // Country
                .withValue("US"),
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("CommonName"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                .withValue("ABCDEFG"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                .withValue("BCDEFGH")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
```

```
        crlConfigure.withS3BucketName("your-bucket-name");

        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Define a certificate authority type: ROOT or SUBORDINATE
        CertificateAuthorityType caType = CertificateAuthorityType.SUBORDINATE;

        // Create a tag - method 1
        Tag tag1 = new Tag();
        tag1.withKey("PrivateCA");
        tag1.withValue("Sample");

        // Create a tag - method 2
        Tag tag2 = new Tag()
            .withKey("Purpose")
            .withValue("WebServices");

        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag1);
        tags.add(tag2);

        // Create the request object.
        CreateCertificateAuthorityRequest req = new
CreateCertificateAuthorityRequest();
        req.withCertificateAuthorityConfiguration(configCA);
        req.withRevocationConfiguration(revokeConfig);
        req.withIdempotencyToken("1234");
        req.withCertificateAuthorityType(caType);
        req.withTags(tags);


        // Create the private CA.
        CreateCertificateAuthorityResult result = null;
        try {
            result = client.createCertificateAuthority(req);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }
```

```
        // Retrieve the ARN of the private CA.
        String arn = result.getCertificateAuthorityArn();
        System.out.println(arn);
    }
}
```

颁发证书 CustomAttribute

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
```

```
public class IssueCertificateWithCustomAttributes {
   private static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
         "certificate-authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
```

```
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded CSR\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
EndEntityCertificate_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(100L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Define custom attributes
      List<CustomAttribute> customAttributes = Arrays.asList(
         new CustomAttribute()
               .withObjectIdentifier("2.5.4.6") // Country
               .withValue("US"),
         new CustomAttribute()
               .withObjectIdentifier("2.5.4.3") // CommonName
               .withValue("CommonName"),
         new CustomAttribute()
               .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
               .withValue("ABCDEFG"),
         new CustomAttribute()
               .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
               .withValue("BCDEFGH")
      );

      // Define certificate subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setCustomAttributes(customAttributes);

      // Add subject to api-passthrough
      ApiPassthrough apiPassthrough = new ApiPassthrough();
```

```
        apiPassthrough.setSubject(subject);
        req.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult result = null;
        try {
           result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidArgsException ex) {
           throw ex;
        } catch (MalformedCSRException ex) {
           throw ex;
        }

        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

使用自定义扩展创建证书

该CustomExtension对象允许管理员在私有证书中设置自定义 X.509 扩展。必须使用其中一个 ApiPassthrough 模板创建自定义证书。有关模板的更多信息，请参阅Amazon 私有 CA 模板品种。有关使用自定义扩展的更多信息，请参阅 颁发私有终端实体证书。

主题

- 使用 NameConstraints 扩展名激活从属 CA
- 颁发带有 QC 声明扩展的证书

使用 NameConstraints 扩展名激活从属 CA

```
package com.amazonaws.samples;
```

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;
import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;
import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
```

```
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;

import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.GeneralSubtree;
import org.bouncycastle.asn1.x509.NameConstraints;

import lombok.SneakyThrows;

public class SubordinateCAActivationWithNameConstraints {
   public static void main(String[] args) throws Exception {
      // Place your own Root CA ARN here.
      String rootCAArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";

      // Define the endpoint region for your sample.
      String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"

      // Define a CA subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setOrganization("Example Organization");
      subject.setOrganizationalUnit("Example");
      subject.setCountry("US");
      subject.setState("Virginia");
      subject.setLocality("Arlington");
      subject.setCommonName("SubordinateCA");

      // Define the CA configuration.
      CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
      configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
      configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
```

```
      configCA.withSubject(subject);

      // Define a certificate revocation list configuration.
      CrlConfiguration crlConfigure = new CrlConfiguration();
      crlConfigure.withEnabled(true);
      crlConfigure.withExpirationInDays(365);
      crlConfigure.withCustomCname(null);
      crlConfigure.withS3BucketName("your-bucket-name");

      // Define a certificate authority type
      CertificateAuthorityType caType = CertificateAuthorityType.SUBORDINATE;

      // ** Execute core code samples for Subordinate CA activation in sequence **
      AWSACMPCA client = ClientBuilder(endpointRegion);
      String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
      String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
caType, client);
      String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
      String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
      String subordinateCertificate = GetCertificate(subordinateCertificateArn,
rootCAArn, client);
      ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);
  }

  private static AWSACMPCA ClientBuilder(String endpointRegion) {
      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException(
                  "Cannot load the credentials from the credential profiles file. " +
                  "Please make sure that your credentials file is at the correct " +
                  "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                  e);
      }

      String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
      EndpointConfiguration endpoint =
```

```
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      return client;
  }

  private static String GetCertificateAuthorityCertificate(String rootCAArn, AWSACMPCA
client) {
      // ** GetCertificateAuthorityCertificate **

      // Create a request object and set the certificate authority ARN,
      GetCertificateAuthorityCertificateRequest getCACertificateRequest =
           new GetCertificateAuthorityCertificateRequest();
      getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

      // Create a result object.
      GetCertificateAuthorityCertificateResult getCACertificateResult = null;
      try {
          getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
      } catch (ResourceNotFoundException ex) {
          throw ex;
      } catch (InvalidStateException ex) {
          throw ex;
      } catch (InvalidArnException ex) {
          throw ex;
      }

      // Retrieve and display the certificate information.
      String rootCertificate = getCACertificateResult.getCertificate();
      System.out.println("Root CA Certificate / Certificate Chain:");
      System.out.println(rootCertificate);

      return rootCertificate;
  }
```

```
  private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType caType, AWSACMPCA
client) {
      RevocationConfiguration revokeConfig = new RevocationConfiguration();
      revokeConfig.setCrlConfiguration(crlConfigure);

      // Create the request object.
      CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
      createCARequest.withCertificateAuthorityConfiguration(configCA);
      createCARequest.withRevocationConfiguration(revokeConfig);
      createCARequest.withIdempotencyToken("1234");
      createCARequest.withCertificateAuthorityType(caType);

      // Create the private CA.
      CreateCertificateAuthorityResult createCAResult = null;
      try {
          createCAResult = client.createCertificateAuthority(createCARequest);
      } catch (InvalidArgsException ex) {
          throw ex;
      } catch (InvalidPolicyException ex) {
          throw ex;
      } catch (LimitExceededException ex) {
          throw ex;
      }

      // Retrieve the ARN of the private CA.
      String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
      System.out.println("Subordinate CA Arn: " + subordinateCAArn);

      return subordinateCAArn;
  }

  private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

      // Create the CSR request object and set the CA ARN.
      GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
      csrRequest.withCertificateAuthorityArn(subordinateCAArn);

      // Create waiter to wait on successful creation of the CSR file.
      Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
```

```
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
```

```
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
SubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(100L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded Nameconstraints extension value
        String base64EncodedExtValue = getNameConstraintExtensionValue();

        // Generate custom extension
        CustomExtension customExtension = new CustomExtension();
        customExtension.setCritical(true);
        customExtension.setObjectIdentifier("2.5.29.30"); // NameConstraints Extension
 OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
```

```
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " + subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    @SneakyThrows
    private static String getNameConstraintExtensionValue() {
       // Generate Base64 encoded Nameconstraints extension value
       GeneralSubtree dnsPrivate = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".private"));
       GeneralSubtree dnsLocal = new GeneralSubtree(new GeneralName(GeneralName.dNSName,
".local"));
       GeneralSubtree dnsCorp = new GeneralSubtree(new GeneralName(GeneralName.dNSName,
".corp"));
       GeneralSubtree dnsSecretCorp = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".secret.corp"));
       GeneralSubtree dnsExample = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".example.com"));
       GeneralSubtree[] permittedSubTree = new GeneralSubtree[] { dnsPrivate, dnsLocal,
dnsCorp };
       GeneralSubtree[] excludedSubTree = new GeneralSubtree[] { dnsSecretCorp,
dnsExample };
       NameConstraints nameConstraints = new NameConstraints(permittedSubTree,
excludedSubTree);

       return new String(Base64.getEncoder().encode(nameConstraints.getEncoded()));
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();
```

```
        // Set the certificate ARN.
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }
```

```
  private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

      // Create the request object and set the signed certificate, chain and CA ARN.
      ImportCertificateAuthorityCertificateRequest importRequest =
          new ImportCertificateAuthorityCertificateRequest();

      ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
      importRequest.setCertificate(certByteBuffer);

      ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
      importRequest.setCertificateChain(rootCACertByteBuffer);

      // Set the certificate authority ARN.
      importRequest.withCertificateAuthorityArn(subordinateCAArn);

      // Import the certificate.
      try {
          client.importCertificateAuthorityCertificate(importRequest);
      } catch (CertificateMismatchException ex) {
          throw ex;
      } catch (MalformedCertificateException ex) {
          throw ex;
      } catch (InvalidArnException ex) {
          throw ex;
      } catch (ResourceNotFoundException ex) {
          throw ex;
      } catch (RequestInProgressException ex) {
          throw ex;
      } catch (ConcurrentModificationException ex) {
          throw ex;
      } catch (RequestFailedException ex) {
          throw ex;
      }
      System.out.println("Subordinate CA certificate successfully imported.");
      System.out.println("Subordinate CA activated successfully.");
  }

  private static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
```

```
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

颁发带有 QC 声明扩展的证书

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.ASN1EncodableVector;
import org.bouncycastle.asn1.ASN1ObjectIdentifier;
import org.bouncycastle.asn1.DERSequence;
import org.bouncycastle.asn1.DERUTF8String;
```

```
import org.bouncycastle.asn1.x509.qualified.ETSIQCObjectIdentifiers;
import org.bouncycastle.asn1.x509.qualified.QCStatement;

import lombok.SneakyThrows;

public class IssueCertificateWithQCStatement {
   private static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
   private static String generateQCStatementBase64ExtValue() {
      DERSequence qcTypeSeq = new DERSequence(ETSIQCObjectIdentifiers.id_etsi_qct_web);
      QCStatement qcType = new QCStatement(ETSIQCObjectIdentifiers.id_etsi_qcs_QcType,
 qcTypeSeq);

      ASN1EncodableVector pspAIVector = new ASN1EncodableVector(2);
      pspAIVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.3"));
      pspAIVector.add(new DERUTF8String("PSP_AI"));
      DERSequence pspAISeq = new DERSequence(pspAIVector);

      ASN1EncodableVector pspASVector = new ASN1EncodableVector(2);
      pspASVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.1"));
      pspASVector.add(new DERUTF8String("PSP_AS"));
      DERSequence pspASSeq = new DERSequence(pspASVector);

      ASN1EncodableVector pspPIVector = new ASN1EncodableVector(2);
      pspPIVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.2"));
      pspPIVector.add(new DERUTF8String("PSP_PI"));
      DERSequence pspPISeq = new DERSequence(pspPIVector);

      ASN1EncodableVector pspICVector = new ASN1EncodableVector(2);
      pspICVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.4"));
      pspICVector.add(new DERUTF8String("PSP_IC"));
      DERSequence pspICSeq = new DERSequence(pspICVector);

      ASN1EncodableVector pspSeqVector = new ASN1EncodableVector(4);
      pspSeqVector.add(pspPISeq);
      pspSeqVector.add(pspICSeq);
      pspSeqVector.add(pspASSeq);
```

```
      pspSeqVector.add(pspAISeq);
      DERSequence pspSeq = new DERSequence(pspSeqVector);

      ASN1EncodableVector pspVector = new ASN1EncodableVector(3);
      pspVector.add(pspSeq);
      pspVector.add(new DERUTF8String("Your Financial Authority"));
      pspVector.add(new DERUTF8String("AB-CD"));
      DERSequence psp = new DERSequence(pspVector);
      QCStatement qcPSP = new QCStatement(new ASN1ObjectIdentifier("0.4.0.19495.2"),
 psp);

      DERSequence qcSeq = new DERSequence(new QCStatement[] { qcType, qcPSP });

      byte[] qcExtValueInBytes = qcSeq.getEncoded();
      return Base64.getEncoder().encodeToString(qcExtValueInBytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();
```

```
      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
         "certificate-authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded CSR\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
EndEntityCertificate_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(30L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Generate Base64 encoded extension value for QC Statement
      String base64EncodedExtValue = generateQCStatementBase64ExtValue();

      // Generate custom extension
      CustomExtension customExtension = new CustomExtension();
      customExtension.setObjectIdentifier("1.3.6.1.5.5.7.1.3"); // QC Statement
 Extension OID
      customExtension.setValue(base64EncodedExtValue);

      // Add custom extension to api-passthrough
      ApiPassthrough apiPassthrough = new ApiPassthrough();
      Extensions extensions = new Extensions();
      extensions.setCustomExtensions(Arrays.asList(customExtension));
      apiPassthrough.setExtensions(extensions);
```

```
        req.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult result = null;
        try {
           result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidArgsException ex) {
           throw ex;
        } catch (MalformedCSRException ex) {
           throw ex;
        }

        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

用于 Amazon 私有 CA 实现案件证书

您可以使用 Amazon 私有证书颁发机构 API 创建符合 Matt er 连接标准的证书。Matter 指定了可提高跨多个工程平台的物联网（IoT）设备安全性和一致性的证书配置。有关 Matter 的更多信息，请参阅 buildwithmatter.com。

2023 年 10 月发布的 Matter 1.2 支持使用证书吊销列表 () CRLs 撤销 DAC。为了帮助您遵守当前 Matter 标准，当您为 CAs 该颁发的 Matter 证书启用 CRL 撤销时，在`CrlConfiguration`对象中，在`CrlDistributionPointExtensionConfiguration`结构中，设置为`OmitExtension`。`true`

通常，在他们颁发的证书中 CAs 嵌入 CRL 分发点 (CDP)，以便执行证书链验证的依赖方可以获取 CRL 并检查证书状态。在 Matter 中，CDP URI 未写入证书。取而代之的是，用户 CDPs 从 Matter 分布式合规账本 (DCL)（可信的 Matter 数据存储）中获取。你必须将 CDP URI 上传到 Matter DCL，这样在验证时才能发现它。 DACs有关确定 CDP URI 的更多信息，请参阅确定 CRL 分发点 (CDP) URI。有关 Matter 的更多信息，请参阅 M atter 标准主页。

主题

- 激活产品认证机构 (PAA)
- 激活产品认证中间体 (PAI)
- 创建设备认证证书 (DAC)
- 激活节点操作证书 (NOC) 的根 CA。
- 为节点操作证书 (NOC) 激活从属 CA
- 创建节点操作证书 (NOC)

激活产品认证机构 (PAA)

此 Java 示例展示了如何使用根 CACertificate _ APIPassthrough /V1 的定义模板创建和安装用于产品认证的 M at ter Root CA (PAA) 证书。 AuthorityKeyIdentifier (AKI) 扩展名是可选的 PAAs。要设置 AKI，必须生成一个 Base64 编码的 AKI 值并将其传递给。 CustomExtension

该示例调用了以下 Amazon 私有 CA API 操作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

如果您遇到问题，请参阅“故障排除”部分中的 对 Amazon 私有 CA Matter 兼容的证书错误进行故障排除。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
```

```
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CrlDistributionPointExtensionConfiguration;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
```

```
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.util.io.pem.PemReader;

import lombok.SneakyThrows;

public class ProductAttestationAuthorityActivation {

    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("Matter Test PAA"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.1") // Vendor ID
                .withValue("FFF1")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);
```

```
        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a CRL distribution point extension configuration
        CrlDistributionPointExtensionConfiguration CDPConfigure = new
CrlDistributionPointExtensionConfiguration();
        CDPConfigure.withOmitExtension(true);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");
        crlConfigure.withS3ObjectAcl("BUCKET_OWNER_FULL_CONTROL");
        crlConfigure.withCrlDistributionPointExtensionConfiguration(CDPConfigure);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, crlConfigure, CAtype,
client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
```

```
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);
        createCARequest.withRevocationConfiguration(revokeConfig);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
```

```
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Product Attestation Authority (PAA) Arn: " + rootCAArn);

        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
```

```
        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    @SneakyThrows
    private static String generateAuthorityKeyIdentifier(final String csrPEM) {
        PKCS10CertificationRequest csr = getPKCS10CertificationRequest(csrPEM);
        SubjectPublicKeyInfo spki = csr.getSubjectPublicKeyInfo();

        JcaX509ExtensionUtils extensionUtils = new JcaX509ExtensionUtils();
        byte[] akiBytes =
extensionUtils.createAuthorityKeyIdentifier(spki).getEncoded();

        return Base64.getEncoder().encodeToString(akiBytes);
    }

    @SneakyThrows
    private static PKCS10CertificationRequest getPKCS10CertificationRequest(final
String csrPEM) {
        ByteArrayInputStream bais = new ByteArrayInputStream(csrPEM.getBytes());
        PemReader pemReader = new PemReader(new InputStreamReader(bais));
        PEMParser parser = new PEMParser(pemReader);
        Object o = parser.readObject();
        if (o instanceof PKCS10CertificationRequest) {
            return (PKCS10CertificationRequest) o;
        }
        return null;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
```

```
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
RootCACertificate_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded extension value for AuthorityKeyIdentifier
        String base64EncodedExtValue = generateAuthorityKeyIdentifier(csr);

        // Generate custom extension
        CustomExtension customExtension = new CustomExtension();
        customExtension.setObjectIdentifier("2.5.29.35"); // AuthorityKeyIdentifier
 Extension OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
```

```
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Product Attestation Authority (PAA) Certificate Arn: " +
rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
```

```
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
```

```
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Product Attestation Authority (PAA) certificate
 successfully imported.");
        System.out.println("Product Attestation Authority (PAA) activated
 successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

激活产品认证中间体 (PAI)

此 Java 示例展示了如何使用BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1定义模板创建和安装用于产品认证的 Matt er Sublerity CA (PAI) 证书。必须生成一个 Base64 编码的 KeyUsage 值并将其传递给。 CustomExtension

该示例调用了以下 Amazon 私有 CA API 操作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate
- GetCertificateAuthorityCertificate

如果您遇到问题，请参阅“故障排除”部分中的 对 Amazon 私有 CA Matter 兼容的证书错误进行故障排除。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CrlDistributionPointExtensionConfiguration;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
```

```
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import lombok.SneakyThrows;

public class ProductAttestationIntermediateActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String paaArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";

        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
```

```
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("Matter Test PAI"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.1") // Vendor ID
                .withValue("FFF1"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.2") // Product ID
                .withValue("8000")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a CRL distribution point extension configuration
        CrlDistributionPointExtensionConfiguration CDPConfigure = new
CrlDistributionPointExtensionConfiguration();
        CDPConfigure.withOmitExtension(true);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");
        crlConfigure.withS3ObjectAcl("BUCKET_OWNER_FULL_CONTROL");
        crlConfigure.withCrlDistributionPointConfiguration(CDPConfigure);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(paaArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
CAtype, client);
```

```
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(paaArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
paaArn, client);
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);

    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
AWSACMPCA client) {
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
```

```
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Retrieve and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Product Attestation Authority (PAA) Certificate /
Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);
        createCARequest.withRevocationConfiguration(revokeConfig);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
```

```
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Product Attestation Intermediate (PAI) Arn: " +
subordinateCAArn);

        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
```

```
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");
```

```
        ApiPassthrough apiPassthrough = new ApiPassthrough();

        // Generate Base64 encoded extension value for ExtendedKeyUsage
        String base64EncodedKUValue = generateKeyUsageValue();

        // Generate custom extension
        CustomExtension customKeyUsageExtension = new CustomExtension();
        customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
        customKeyUsageExtension.setValue(base64EncodedKUValue);
        customKeyUsageExtension.setCritical(true);

        // Set KeyUsage extension to api passthrough
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    @SneakyThrows
```

```
   private static String generateKeyUsageValue() {
      KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign |
X509KeyUsage.cRLSign);
      byte[] kuBytes = keyUsage.getEncoded();
      return Base64.getEncoder().encodeToString(kuBytes);
   }

   private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

       // Create a request object.
       GetCertificateRequest certificateRequest = new GetCertificateRequest();

       // Set the certificate ARN.
       certificateRequest.withCertificateArn(subordinateCertificateArn);

       // Set the certificate authority ARN.
       certificateRequest.withCertificateAuthorityArn(rootCAArn);

       // Create waiter to wait on successful creation of the certificate file.
       Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
       try {
           getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
       } catch (WaiterUnrecoverableException e) {
           //Explicit short circuit when the recourse transitions into
           //an undesired state.
       } catch (WaiterTimedOutException e) {
           //Failed to transition into desired state even after polling.
       } catch (AWSACMPCAException e) {
           //Unexpected service exception.
       }

       // Retrieve the certificate and certificate chain.
       GetCertificateResult certificateResult = null;
       try {
           certificateResult = client.getCertificate(certificateRequest);
       } catch (RequestInProgressException ex) {
           throw ex;
       } catch (RequestFailedException ex) {
           throw ex;
       } catch (ResourceNotFoundException ex) {
           throw ex;
       } catch (InvalidArnException ex) {
```

```
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
```

```
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Product Attestation Intermediate (PAI) certificate
 successfully imported.");
        System.out.println("Product Attestation Intermediate (PAI) activated
 successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

创建设备认证证书 (DAC)

此 Java 示例展示了如何使用 BlankEndEntityCertificate_ CriticalBasicConstraints _ APIPassthrough / V1 模板创建 Matt er Device 认证证书。您必须生成一个 Base64 编码的 KeyUsage 值，然后将其传递给。 CustomExtension

该示例调用以下 Amazon 私有 CA API 操作：

- IssueCertificate

如果您遇到问题，请参阅“故障排除”部分中的 对 Amazon 私有 CA Matter 兼容的证书错误进行故障排除。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
```

```
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssueDeviceAttestationCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
   private static String generateKeyUsageValue() {
      KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
      byte[] kuBytes = keyUsage.getEncoded();
```

```
        return Base64.getEncoder().encodeToString(kuBytes);
    }

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from disk", e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create a certificate request:
        IssueCertificateRequest req = new IssueCertificateRequest();

        // Set the CA ARN.
        req.withCertificateAuthorityArn("arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012");

        // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
        String strCSR =
        "-----BEGIN CERTIFICATE REQUEST-----\n" +
        "base64-encoded certificate\n" +
        "-----END CERTIFICATE REQUEST-----\n";
        ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
        req.setCsr(csrByteBuffer);
```

```
      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(10L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Define custom attributes
      List<CustomAttribute> customAttributes = Arrays.asList(
         new CustomAttribute()
            .withObjectIdentifier("2.5.4.3")
            .withValue("Matter Test DAC 0001"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.2.1")
            .withValue("FFF1"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.2.2")
            .withValue("8000")
      );

      // Define a cert subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setCustomAttributes(customAttributes);

      ApiPassthrough apiPassthrough = new ApiPassthrough();
      apiPassthrough.setSubject(subject);

      // Generate Base64 encoded extension value for ExtendedKeyUsage
      String base64EncodedKUValue = generateKeyUsageValue();

      // Generate custom extension
      CustomExtension customKeyUsageExtension = new CustomExtension();
      customKeyUsageExtension.setObjectIdentifier("2.5.29.15"); // KeyUsage Extension
 OID
      customKeyUsageExtension.setValue(base64EncodedKUValue);
```

```
        customKeyUsageExtension.setCritical(true);

        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
        apiPassthrough.setExtensions(extensions);
        req.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult result = null;
        try {
           result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidArgsException ex) {
           throw ex;
        } catch (MalformedCSRException ex) {
           throw ex;
        }

        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

激活节点操作证书 (NOC) 的根 CA。

此 Java 示例展示了如何使用根 CACertificate _ APIPassthrough /V1 的定义模板创建和安装要颁发的 M at ter Root CA 证书 NOCs。 AuthorityKeyIdentifier (AKI) 扩展名对于 NOC 根 CA 证书是可选的。要设置 AKI，必须生成一个 Base64 编码的 AKI 值并将其传递给。 CustomExtension

该示例调用了以下 Amazon 私有 CA API 操作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate

- GetCertificate
- ImportCertificateAuthorityCertificate

如果您遇到问题，请参阅“故障排除”部分中的 对 Amazon 私有 CA Matter 兼容的证书错误进行故障排除。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;
```

```
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.util.io.pem.PemReader;

import lombok.SneakyThrows;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
```

```
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.1.4")
                .withValue("CACACACA00000001")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
```

```
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Root CA Arn: " + rootCAArn);
```

```
        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);
```

```
        return csr;
    }

    @SneakyThrows
    private static String generateAuthorityKeyIdentifier(final String csrPEM) {
        PKCS10CertificationRequest csr = getPKCS10CertificationRequest(csrPEM);
        SubjectPublicKeyInfo spki = csr.getSubjectPublicKeyInfo();

        JcaX509ExtensionUtils extensionUtils = new JcaX509ExtensionUtils();
        byte[] akiBytes =
 extensionUtils.createAuthorityKeyIdentifier(spki).getEncoded();

        return Base64.getEncoder().encodeToString(akiBytes);
    }

    @SneakyThrows
    private static PKCS10CertificationRequest getPKCS10CertificationRequest(final
 String csrPEM) {
        ByteArrayInputStream bais = new ByteArrayInputStream(csrPEM.getBytes());
        PemReader pemReader = new PemReader(new InputStreamReader(bais));
        PEMParser parser = new PEMParser(pemReader);
        Object o = parser.readObject();
        if (o instanceof PKCS10CertificationRequest) {
            return (PKCS10CertificationRequest) o;
        }
        return null;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
RootCACertificate_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);
```

```
        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded extension value for AuthorityKeyIdentifier
        String base64EncodedExtValue = generateAuthorityKeyIdentifier(csr);

        // Generate custom extension
        CustomExtension customExtension = new CustomExtension();
        customExtension.setObjectIdentifier("2.5.29.35"); // AuthorityKeyIdentifier
Extension OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
```

```
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
```

```
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
```

```
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

为节点操作证书 (NOC) 激活从属 CA

此 Java 示例展示了如何使用BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1定义模板来颁发和安装要颁发的 Matt er Sublerity CA 证书 NOCs。您必须生成一个 Base64 编码的 KeyUsage 值并将其传递给。 CustomExtension

该示例调用了以下 Amazon 私有 CA API 操作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate
- GetCertificateAuthorityCertificate

如果出现问题，请参阅 对 Amazon 私有 CA Matter 兼容的证书错误进行故障排除 “故障排除” 部分。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
```

```
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
```

```
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import lombok.SneakyThrows;

public class IntermediateCAActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String rootCAArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";

        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.1.3")
                .withValue("CACACACA00000003")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
```

```
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
rootCAArn, client);
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);

    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
```

```
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
AWSACMPCA client) {
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Get and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Root CA Certificate / Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);
```

```
        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Subordinate CA Arn: " + subordinateCAArn);

        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Get the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
```

```
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Get and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);
```

```
        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        ApiPassthrough apiPassthrough = new ApiPassthrough();

        // Generate base64 encoded extension value for ExtendedKeyUsage
        String base64EncodedKUValue = generateKeyUsageValue();

        // Generate custom extension
        CustomExtension customKeyUsageExtension = new CustomExtension();
        customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
        customKeyUsageExtension.setValue(base64EncodedKUValue);
        customKeyUsageExtension.setCritical(true);

        // Set KeyUsage extension to api passthrough
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Get and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
```

```
    }

    @SneakyThrows
    private static String generateKeyUsageValue() {
       KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign |
X509KeyUsage.cRLSign);
       byte[] kuBytes = keyUsage.getEncoded();
       return Base64.getEncoder().encodeToString(kuBytes);
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Get the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
```

```
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
```

```
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Subordinate CA certificate successfully imported.");
        System.out.println("Subordinate CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

创建节点操作证书 (NOC)

此 Java 示例展示了如何使用 BlankEndEntityCertificateCriticalBasicConstraints_ _ APIPassthrough / V1 模板创建案件节点操作证书。您必须生成一个 Base64 编码的 KeyUsage 值，然后将其传递给。CustomExtension

该示例调用以下 Amazon 私有 CA API 操作：

- IssueCertificate

如果您遇到问题，请参阅“故障排除”部分中的 对 Amazon 私有 CA Matter 兼容的证书错误进行故障排除。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.ExtendedKeyUsage;
import org.bouncycastle.asn1.x509.KeyPurposeId;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssueNodeOperatingCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
```

```
   private static String generateExtendedKeyUsageValue() {
      KeyPurposeId[] keyPurposeIds = new KeyPurposeId[]
 { KeyPurposeId.id_kp_clientAuth, KeyPurposeId.id_kp_serverAuth };
      ExtendedKeyUsage eku = new ExtendedKeyUsage(keyPurposeIds);
      byte[] ekuBytes = eku.getEncoded();
      return Base64.getEncoder().encodeToString(ekuBytes);
   }

   @SneakyThrows
   private static String generateKeyUsageValue() {
      KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
      byte[] kuBytes = keyUsage.getEncoded();
      return Base64.getEncoder().encodeToString(kuBytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
```

```
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded certificate\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(10L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Define custom attributes
      List<CustomAttribute> customAttributes = Arrays.asList(
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.1.1")
            .withValue("DEDEDEDE00010001"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.1.5")
            .withValue("FAB000000000001D")
      );

      // Define a cert subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setCustomAttributes(customAttributes);

      ApiPassthrough apiPassthrough = new ApiPassthrough();
      apiPassthrough.setSubject(subject);
```

```
     // Generate Base64 encoded extension value for ExtendedKeyUsage
     String base64EncodedKUValue = generateKeyUsageValue();

     // Generate custom extension
     CustomExtension customKeyUsageExtension = new CustomExtension();
     customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
     customKeyUsageExtension.setValue(base64EncodedKUValue);
     customKeyUsageExtension.setCritical(true);

     // Generate Base64 encoded extension value for ExtendedKeyUsage
     String base64EncodedEKUValue = generateExtendedKeyUsageValue();

     CustomExtension customExtendedKeyUsageExtension = new CustomExtension();
     customExtendedKeyUsageExtension.setObjectIdentifier("2.5.29.37"); //
ExtendedKeyUsage Extension OID
     customExtendedKeyUsageExtension.setValue(base64EncodedEKUValue);
     customExtendedKeyUsageExtension.setCritical(true);

     // Set KeyUsage and ExtendedKeyUsage extension to api-passthrough
     Extensions extensions = new Extensions();
     extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension,
customExtendedKeyUsageExtension));
     apiPassthrough.setExtensions(extensions);
     req.setApiPassthrough(apiPassthrough);

     // Issue the certificate.
     IssueCertificateResult result = null;
     try {
        result = client.issueCertificate(req);
     } catch (LimitExceededException ex) {
        throw ex;
     } catch (ResourceNotFoundException ex) {
        throw ex;
     } catch (InvalidStateException ex) {
        throw ex;
     } catch (InvalidArnException ex) {
        throw ex;
     } catch (InvalidArgsException ex) {
        throw ex;
     } catch (MalformedCSRException ex) {
        throw ex;
     }
```

```
        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

用于 Amazon 私有 CA 实现 mDL 证书

您可以使用该 Amazon 私有证书颁发机构 API 创建符合 ISO/IEC 移动驾驶执照 (mDL) 标准的证书。该标准为实施与移动设备相关的驾驶执照制定了接口规范，包括证书配置。

主题

- 激活颁发机构证书颁发机构 (IACA) 证书
- 创建文档签名者证书

激活颁发机构证书颁发机构 (IACA) 证书

此 Java 示例演示如何使用BlankRootCACertificate_ PathLen 0_ APIPassthrough /V1 的定义模板创建和安装符合 ISO/IEC mDL 标准的颁发机构证书颁发机构 (IACA) 证书。必须为、和生成 base64 编码的值 `KeyUsage` `IssuerAlternativeNameCRLDistributionPoint`，然后将其传递。`CustomExtensions`

> **Note**
>
> IACA 链接证书建立了从旧 IACA 根证书到新 IACA 根证书的信任路径。颁发机构可以在IACA 重新密钥过程中生成和分发IACA链接证书。您不能使用已设置的 IACA 根证书来颁发 IACA 链接证书。`pathLen=0`

该示例调用了以下 Amazon 私有 CA API 操作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples.mdl;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
```

```
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssuingAuthorityCertificateAuthorityActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = null;  // Substitute your region here, e.g. "ap-
southeast-2"
        if (endpointRegion == null) throw new Exception("Region cannot be null");

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject()
                .withCountry("US") // mDL spec requires ISO 3166-1-alpha-2 country code
 e.g. "US"
                .withCommonName("mDL Test IACA");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration()
                .withKeyAlgorithm(KeyAlgorithm.EC_prime256v1)
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withSubject(subject);

        // Define a certificate authority type
```

```
        CertificateAuthorityType CAType = CertificateAuthorityType.ROOT;

        // Execute core code samples for Root CA activation in sequence
        AWSACMPCA client = buildClient(endpointRegion);
        String rootCAArn = createCertificateAuthority(configCA, CAType, client);
        String csr = getCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = issueCertificate(rootCAArn, csr, client);
        String rootCertificate = getCertificate(rootCertificateArn, rootCAArn, client);
        importCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA buildClient(String endpointRegion) {
        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from disk",
e);
        }

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
                .withRegion(endpointRegion)
                .withCredentials(new AWSStaticCredentialsProvider(credentials))
                .build();

        return client;
    }

    private static String createCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest()
                .withCertificateAuthorityConfiguration(configCA)
                .withIdempotencyToken("123987")
                .withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
```

```
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Get the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Issuing Authority Certificate Authority (IACA) Arn: " +
rootCAArn);

        return rootCAArn;
    }

    private static String getCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest()
                .withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            // Explicit short circuit when the recourse transitions into
            // an undesired state.
        } catch (WaiterTimedOutException e) {
            // Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            // Unexpected service exception.
        }

        // Get the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
```

```
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Get and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("CSR:");
        System.out.println(csr);

        return csr;
    }

    @SneakyThrows
    private static String issueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {
        IssueCertificateRequest issueRequest = new IssueCertificateRequest()
                .withCertificateAuthorityArn(rootCAArn)
                .withTemplateArn("arn:aws:acm-pca:::template/
BlankRootCACertificate_PathLen0_APIPassthrough/V1")
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withIdempotencyToken("1234");

        // Set the CSR.
        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity()
                .withValue(3650L)
                .withType("DAYS");
        issueRequest.setValidity(validity);

        // Generate base64 encoded extension value for KeyUsage
        KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign +
 X509KeyUsage.cRLSign);
        byte[] kuBytes = keyUsage.getEncoded();
        String base64EncodedKUValue = Base64.getEncoder().encodeToString(kuBytes);

        CustomExtension keyUsageCustomExtension = new CustomExtension()
                .withObjectIdentifier("2.5.29.15") // KeyUsage Extension OID
```

```
                .withValue(base64EncodedKUValue)
                .withCritical(true);

        // Generate base64 encoded extension value for IssuerAlternativeName
        GeneralNames issuerAlternativeName = new GeneralNames(new
 GeneralName(GeneralName.uniformResourceIdentifier, "https://issuer-alternative-
name.com"));
        String base64EncodedIANValue =
 Base64.getEncoder().encodeToString(issuerAlternativeName.getEncoded());

        CustomExtension ianCustomExtension = new CustomExtension()
                .withValue(base64EncodedIANValue)
                .withObjectIdentifier("2.5.29.18"); // IssuerAlternativeName Extension
 OID

        // Generate base64 encoded extension value for CRLDistributionPoint
        CRLDistPoint crlDistPoint = new CRLDistPoint(new DistributionPoint[]{new
 DistributionPoint(new DistributionPointName(
                new GeneralNames(new GeneralName(GeneralName.uniformResourceIdentifier,
 "dummycrl.crl"))), null, null)});
        String base64EncodedCDPValue =
 Base64.getEncoder().encodeToString(crlDistPoint.getEncoded());

        CustomExtension cdpCustomExtension = new CustomExtension()
                .withValue(base64EncodedCDPValue)
                .withObjectIdentifier("2.5.29.31"); // CRLDistributionPoint Extension
 OID

        // Add custom extension to api-passthrough
        Extensions extensions = new Extensions()
                .withCustomExtensions(Arrays.asList(keyUsageCustomExtension,
 ianCustomExtension, cdpCustomExtension));
        ApiPassthrough apiPassthrough = new ApiPassthrough()
                .withExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
```

```
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Get and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("mDL IACA Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String getCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest()
                .withCertificateArn(rootCertificateArn)
                .withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            // Explicit short circuit when the recourse transitions into
            // an undesired state.
        } catch (WaiterTimedOutException e) {
            // Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            // Unexpected service exception.
        }

        // Get the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
```

```
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void importCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
                new ImportCertificateAuthorityCertificateRequest()
                        .withCertificateChain(null)
                        .withCertificateAuthorityArn(rootCAArn);

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
```

```
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

创建文档签名者证书

此 Java 示例演示如何使用 BlankEndEntityCertificate_ APIPassthrough /V1 模板创建符合 ISO/IEC mDL 标准的文档签名者证书。必须为、和生成基于 base64 编码的值 KeyUsageIssuerAlternativeName，CRLDistributionPoint然后将其传递。CustomExtensions

该示例调用以下 Amazon 私有 CA API 操作：

- IssueCertificate

```
package com.amazonaws.samples.mdl;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.ExtendedKeyUsage;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

public class IssueDocumentSignerCertificate {
    public static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }

    public static void main(String[] args) throws Exception {

        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
```

```
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from disk",
 e);
        }

        // Create a client that you can use to make requests.
        String endpointRegion = null;  // Substitute your region here, e.g. "ap-
southeast-2"
        if (endpointRegion == null) throw new Exception("Region cannot be null");

        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
                .withRegion(endpointRegion)
                .withCredentials(new AWSStaticCredentialsProvider(credentials))
                .build();

        // Create a certificate request:
        String caArn = null;
        if (caArn == null) throw new Exception("Certificate authority ARN cannot be
 null");

        IssueCertificateRequest req = new IssueCertificateRequest()
                .withCertificateAuthorityArn(caArn)
                .withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V1")
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withIdempotencyToken("1234");

        // Specify the certificate signing request (CSR) for the certificate to be
 signed and issued.
        // Format: "-----BEGIN CERTIFICATE REQUEST-----\n" +
        //         "base64-encoded certificate\n" +
        //         "-----END CERTIFICATE REQUEST-----\n";
        String strCSR = null;
        if (strCSR == null) throw new Exception("CSR string cannot be null");

        ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
        req.setCsr(csrByteBuffer);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity()
                .withValue(365L)
                .withType("DAYS");
        req.setValidity(validity);
```

```
        // Define a cert subject.
        ASN1Subject subject = new ASN1Subject()
                .withCountry("US") // mDL spec requires ISO 3166-1-alpha-2 country code
 e.g. "US"
                .withCommonName("mDL Test DS");

        ApiPassthrough apiPassthrough = new ApiPassthrough()
                .withSubject(subject);

        // Generate base64 encoded extension value for KeyUsage
        KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
        byte[] kuBytes = keyUsage.getEncoded();
        String base64EncodedKUValue = Base64.getEncoder().encodeToString(kuBytes);

        CustomExtension customKeyUsageExtension = new CustomExtension()
                .withObjectIdentifier("2.5.29.15") // KeyUsage Extension OID
                .withValue(base64EncodedKUValue)
                .withCritical(true);

        // Generate base64 encoded extension value for IssuerAlternativeName
        GeneralNames issuerAlternativeName = new GeneralNames(new
 GeneralName(GeneralName.uniformResourceIdentifier, "https://issuer-alternative-
name.com"));
        String base64EncodedIANValue =
 Base64.getEncoder().encodeToString(issuerAlternativeName.getEncoded());

        CustomExtension ianCustomExtension = new CustomExtension()
                .withValue(base64EncodedIANValue)
                .withObjectIdentifier("2.5.29.18"); // IssuerAlternativeName Extension
 OID

        // Generate base64 encoded extension value for CRLDistributionPoint
        CRLDistPoint crlDistPoint = new CRLDistPoint(new DistributionPoint[]{new
 DistributionPoint(new DistributionPointName(
                new GeneralNames(new GeneralName(GeneralName.uniformResourceIdentifier,
 "dummycrl.crl"))), null, null)});
        String base64EncodedCDPValue =
 Base64.getEncoder().encodeToString(crlDistPoint.getEncoded());

        CustomExtension cdpCustomExtension = new CustomExtension()
                .withValue(base64EncodedCDPValue)
                .withObjectIdentifier("2.5.29.31"); // CRLDistributionPoint Extension
 OID
```

```
        // Generate EKU
        ExtendedKeyUsage eku = new ExtendedKeyUsage()
                .withExtendedKeyUsageObjectIdentifier("1.0.18013.5.1.2"); // EKU value
 reserved for mDL DS

        // Set KeyUsage, ExtendedKeyUsage, IssuerAlternativeName, CRL Distribution
 Point extensions to api-passthrough
        Extensions extensions = new Extensions()
                .withCustomExtensions(Arrays.asList(customKeyUsageExtension,
 ianCustomExtension, cdpCustomExtension))
                .withExtendedKeyUsage(Arrays.asList(eku));
        apiPassthrough.setExtensions(extensions);
        req.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult result = null;
        try {
            result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Get and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println("mDL DS Certificate Arn: " + arn);
    }
}
```

设计您的解决方案 Amazon 私有 CA

Amazon 私有 CA 让您可以基于云对组织的私有 PKI（公钥基础架构）进行全面的控制，从根证书颁发机构 (CA) 到下级证书 CAs，再到终端实体证书。要想实现安全、可维护、可扩展且适合组织需求的 PKI，全面的规划至关重要。本节提供了有关设计 CA 层次结构、管理私有 CA 和私有终端实体证书生命周期的指导，以及如何应用最佳安全实践。

本节介绍如何在创建私有证书颁发机构 (CA) 之前做好使用准备 Amazon 私有 CA 。它还说明了通过在线证书状态协议（OCSP）或证书吊销列表（CRL）添加吊销支持的选项。

此外，您还应确定您的组织是否更愿意将其私有根 CA 凭证托管在本地而不是使用 Amazon托管。在这种情况下，您需要在使用前设置并保护自我管理的私有 PKI。 Amazon 私有 CA在这种情况下，您随后在中创建一个由外部的父 CA Amazon 私有 CA 支持的从属 CA Amazon 私有 CA。有关更多信息，请参阅安装由外部父 CA 签名的从属 CA 证书。

主题

- 设计 CA 层次结构
- 管理私有 CA 生命周期
- 规划您的 Amazon 私有 CA 证书吊销方法
- 了解 Amazon 私有 CA CA 模式
- 规划适应能力 Amazon 私有 CA

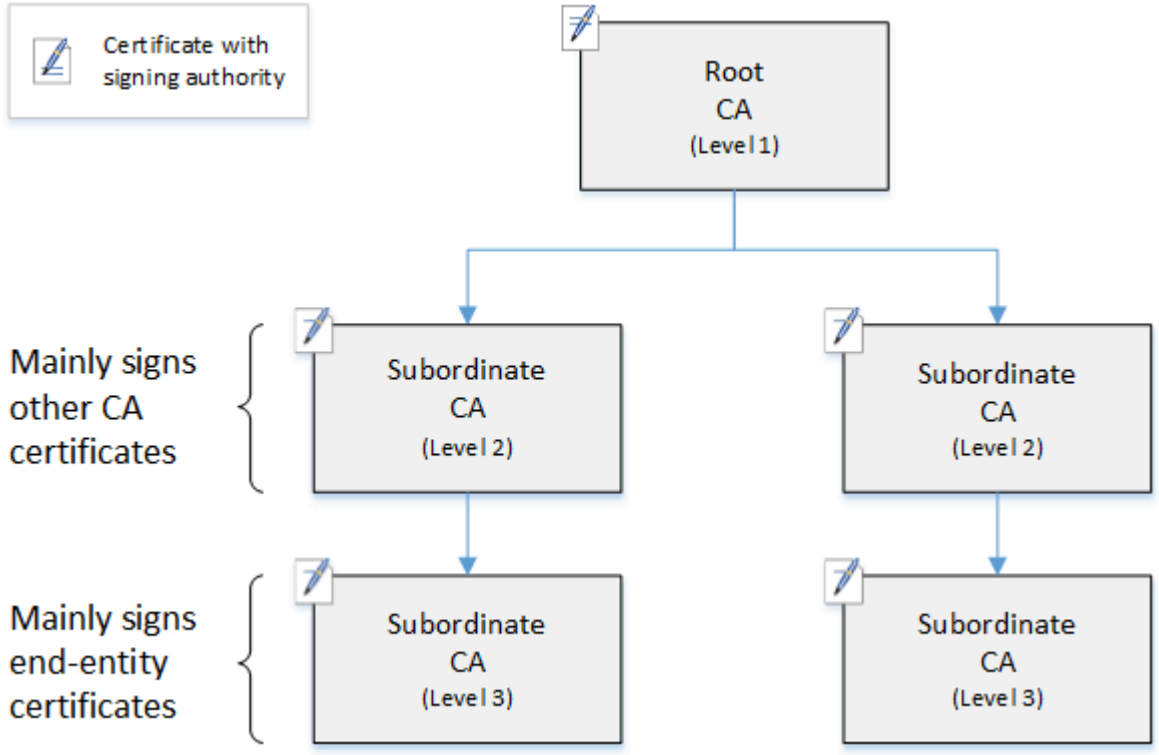
设计 CA 层次结构

使用 Amazon 私有 CA，您可以创建最多五个级别的证书颁发机构层次结构。位于层次结构树顶部的根 CA 可以有任意数量的分支。根 CA 的每个分支 CAs 上最多可以有四个从属级别。您还可以创建多个层次结构，每个层次结构都有自己的根。

良好设计的 CA 层次结构提供以下优势：

- 适用于每个 CA 的精细安全控制
- 划分管理任务以实现更好的负载平衡和安全性
- 在有限的 CAs 、可撤销的信任下使用，用于日常运营
- 有效期和证书路径限制

下图说明了简单的三级 CA 层次结构。



树中的每个 CA 都由具有签名权限的 X.509 v3 证书支持（由图标表示）。 pen-and-paper这意味着 CAs，他们可以签署从属于他们的其他证书。当 CA 对较低级别的 CA 证书签名时，它会授予对签名证书的有限、可吊销的权限。级别 1 中的根 CA 签署级别 2 中的高级别从属 CA 证书。反过来 CAs，它们会签署管理终端实体证书的 PKI（公钥基础架构）管理员使用的第 3 级证书。 CAs

CA 层次结构中的安全性应在树顶部配置为最强。此设置保护根 CA 证书及其私有密钥。根 CA 将所有从属证书 CAs 和终端实体证书的信任锚定在其下面。虽然终端实体证书的受损会导致本地损坏，但根目录的受损会破坏整个 PKI 中的信任。根证书和高级从属证书 CAs 很少使用（通常用于签署其他 CA 证书）。因此，它们受到严格的控制和审计，以确保降低受损风险。在层次结构的较低级别，安全性的限制性较小。此方法允许为用户、计算机主机和软件服务执行例行管理任务，包括颁发和吊销终端实体证书。

> **Note**
>
> 使用根 CA 对从属证书签名是一种罕见的事件，仅在少数情况下发生：
>
> - 创建 PKI 时
> - 需要替换高级证书颁发机构时
> - 需要配置证书吊销列表 (CRL) 或联机证书状态协议 (OCSP) 响应程序时

Root 和其他高级用户 CAs 需要高度安全的操作流程和访问控制协议。

主题

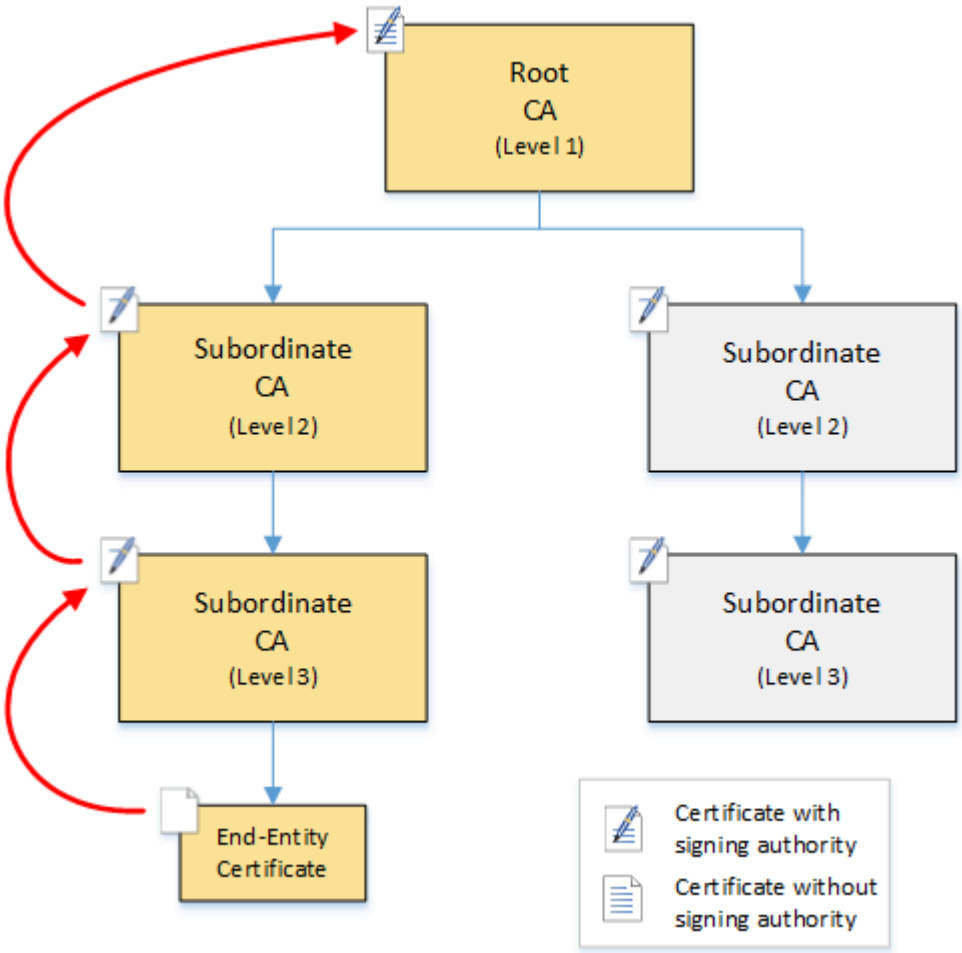
- 验证最终实体证书
- 规划 CA 层次结构的结构
- 在认证路径上设置长度限制

验证最终实体证书

终端实体证书的信任源于一条从属证书路径返回 CAs 到根 CA。向 Web 浏览器或其他客户端呈现了终端实体证书时，它会尝试构建信任链。例如，它可能会检查证书的颁发者可分辨名称 和主题可分辨名称 是否与颁发 CA 证书的相应字段匹配。匹配将在层次结构上的每个后续级别继续进行，直到客户端到达其信任存储中包含的受信任根。

信任存储库是浏览器或操作系统包含的可 CAs 信库。对于私有 PKI，组织的 IT 部门必须确保每个浏览器或系统之前已将私有根 CA 添加到其信任存储中。否则将无法验证证书路径，导致客户端错误。

下图显示了在向浏览器提供终端实体 X.509 证书时，浏览器遵循的验证路径。请注意，终端实体证书缺乏签名颁发机构，仅用于对拥有该证书的实体进行身份验证。



浏览器检查终端实体证书。浏览器发现证书提供了来自从属 CA（级别 3）的签名作为其信任凭证。从属证书 CAs 必须包含在同一 PEM 文件中。或者，它们也可以位于包含构成信任链的证书的单独文件中。找到这些内容后，浏览器会检查从属 CA（级别 3）的证书，并发现它提供了来自从属 CA（级别 2）的签名。接下来，从属 CA（级别 2）提供了来自根 CA（级别 1）的签名作为其信任凭证。如果浏览器发现其信任存储中预装的私有根 CA 证书的副本，它会确认终端实体证书可信。

通常，浏览器还会根据证书吊销列表 (CRL) 检查每个证书。已过期、已吊销或配置错误的证书将被拒绝，验证失败。

规划 CA 层次结构的结构

通常，CA 层次结构应反映组织的结构。路径深度（即 CA 的级别数）的目标是不超过委派管理和安全角色所需的级数。将 CA 添加到层次结构意味着增加证书路径中的证书数量，这会增加验证时间。将路径长度保持在最短水平还可以减少验证终端实体证书时从服务器发送到客户端的证书数量。

从理论上讲，没有pathLenConstraint参数的根 CA 可以授权无限级别的下属 CAs。从属 CA 可以拥有其内部配置 CAs 所允许的任意数量的子从属机构。 Amazon 私有 CA 托管层次结构支持多达五个级别的 CA 认证路径。

良好设计的 CA 结构有几个优势：

- 为不同组织部门分离管理控制
- 能够将访问权限委托给下属 CAs
- 一种分层结构，可通过额外的安全控制来保护更高级别 CAs 的安全

两种常见的 CA 结构可以实现所有这些优势：

- 两个 CA 级别：根 CA 和从属 CA

  这是最简单的 CA 结构，允许对根 CA 和从属 CA 实施单独的管理、控制和安全策略。您可以为根 CA 维护限制性控制和策略，同时为从属 CA 允许更多的访问权限。后者用于批量颁发终端实体证书。

- 三个 CA 级别：根 CA 和两个从属 CA级别

  与上述内容类似，此结构添加了一个额外的 CA 级别，以进一步将根 CA 与低级 CA 操作分开。中间的 CA 层仅用于签发终端实体证书的下属 CAs 机构。

较不常见的 CA 结构如下：

- 四个或更多 CA 级别

  虽然相比三级层次结构不太常见，但具有四个或更多级别的 CA 层次也存在，并且可能需要允许管理委派。

- 一个 CA 级别：仅根 CA

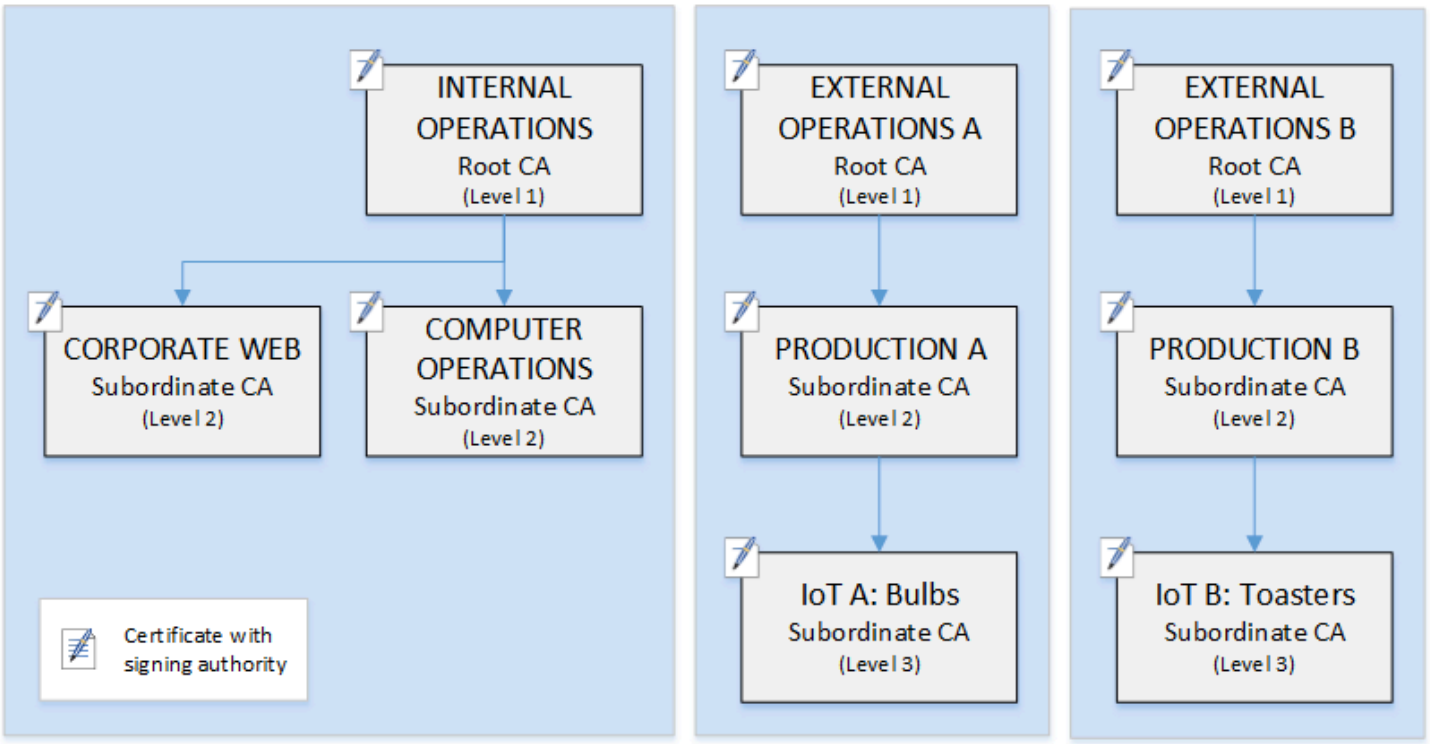
  此结构通常用于不需要完整信任链的开发和测试。用于生产是不合规则的。此外，它违反了为根 CA 和颁发终端实体证书的 CA 分别维护安全策略的最佳实践。 CAs

  但是，如果您已经直接从根 CA 颁发证书，则可以迁移到 Amazon 私有 CA。与使用 OpenSSL 或其他软件管理的根 CA 相比，这样做具有安全和控制优势。

制造商的私有 PKI 示例

在此示例中，一家虚构的技术公司生产两种物联网 (IoT) 产品：智能灯泡和智能烤面包机。在生产过程中会向每台设备颁发终端实体证书，以便它可以通过 Internet 与制造商安全地通信。该公司的 PKI 还保护其计算机基础设施，包括内部网站和各种自行托管的计算机服务，负责财务和业务运营。

因此，CA 层次结构将密切围绕业务的这些管理和运营层面建模。



此层次结构包含三个根，一个用于内部运营，两个用于外部运营（每个产品线一个根 CA）。它还展示了多种认证路径长度，其中两个 CA 级别用于内部运营，三个级别用于外部运营。

在外部操作端使用独立的根 CAs 和额外的从属 CA 层是一项满足业务和安全需求的设计决策。采用多个 CA 树，PKI 可以适应未来的企业重组、剥离或收购。发生更改时，整个根 CA 层次结构可以随其保护的部门彻底移动。而且，由于 CAs 有两个级别的下属 CA，根目录与负责批量签署成千上万个制成品证书的 3 CAs 级有很高的隔离度。

在内部一端，企业 Web 和内部计算机操作构成了两个层次结构。这些级别允许 Web 管理员和运营工程师为自己的工作域独立管理证书颁发。将 PKI 划分为不同的功能域是一种最佳安全实践，可保护每个领域免受可能影响对方的损坏。Web 管理员颁发终端实体证书，供整个公司的 Web 浏览器使用，对内部网站上的通信进行身份验证和加密。运营工程师颁发终端实体证书，用于对数据中心主机和计算机服务彼此进行身份验证。该系统对局域网上的敏感数据进行加密，有助于保护敏感数据的安全。

在认证路径上设置长度限制

CA 层次结构的结构由每个证书包含的基本约束扩展定义和强制实施。扩展定义了两个约束：

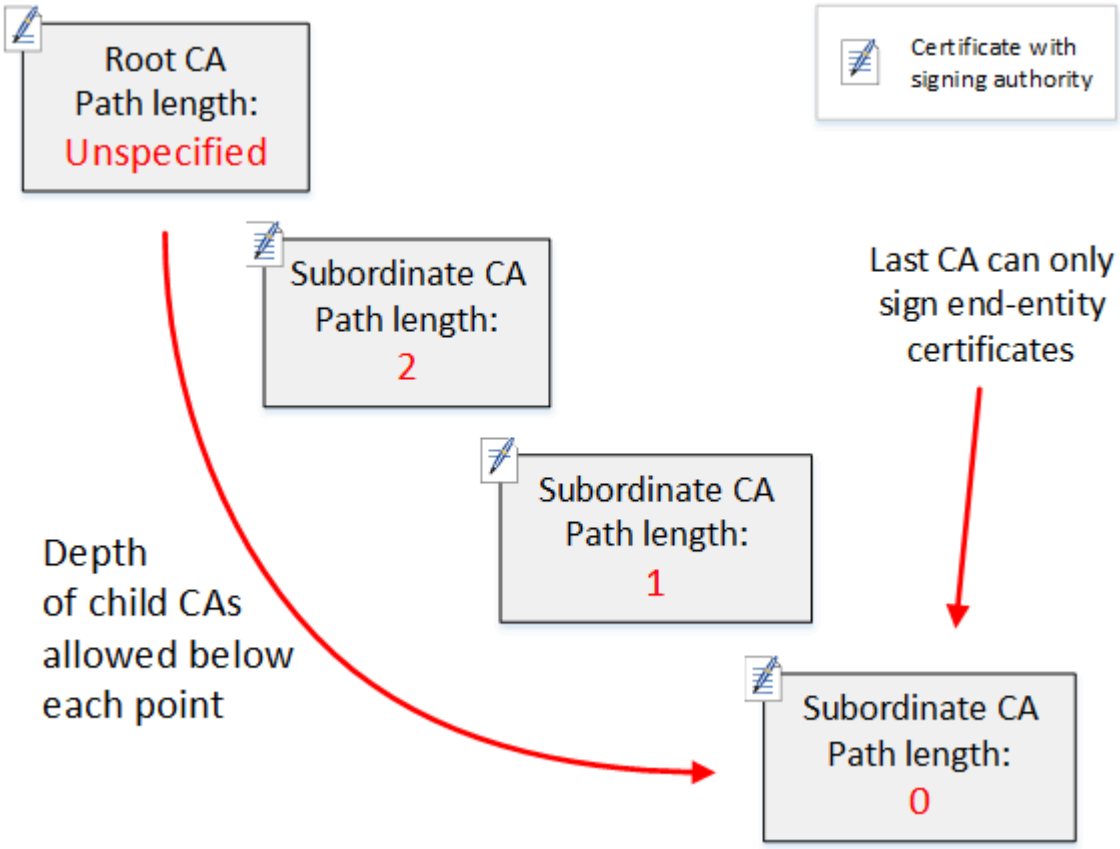
- `cA` – 证书是否定义 CA。如果此值为 false（默认值），则证书是终端实体证书。
- `pathLenConstraint`— 有效信任链中可以 CAs 存在的最大下级下属人数。终端实体证书不计入，因为它不是 CA 证书。

根 CA 证书需要最大的灵活性，不包括路径长度约束。这允许根定义任意长度的证书路径。

> **Note**
>
> Amazon 私有 CA 将认证路径限制为五个级别。

从属`pathLenConstraint`服务器 CAs 的值等于或大于零，具体取决于层次结构中的位置和所需的要素。例如，在包含三个的层次结构中 CAs，没有为根 CA 指定路径约束。第一个从属 CA 的路径长度为 1，因此可以签名子级 CAs。每个孩子的`pathLenConstraint`值都 CAs 必须为零。这意味着它们可以签署终端实体证书，但无法颁发其他 CA 证书。限制创建新内容的权力 CAs 是一项重要的安全控制措施。

下图说明了这种有限权限在层次结构中向下传播的情况。



在这个四级层次结构中，根不受约束（一如既往）。但是第一个从属 CA 的`pathLenConstraint`值为 2，这限制了其子 CAs 级的深度不能超过两个等级。因此，对于有效的证书路径，约束值必须在接下来的两级中减少为零。如果 Web 浏览器遇到来自此分支的终端实体证书的路径长度大于 4，验证将失败。此类证书可能是由于意外创建的 CA、错误配置的 CA 或未授权颁发造成的。

使用模板管理路径长度

Amazon 私有 CA 提供用于颁发根证书、从属证书和终端实体证书的模板。这些模板封装了基本约束值的最佳实践，包括路径长度。模板包括以下内容：

- root CACertificate /V1
- 下属 CACertificate _ PathLen 0/V1
- 下属 CACertificate _ PathLen 1/V1
- 下属 CACertificate _ PathLen 2/V1
- 下属 CACertificate _ PathLen 3/V1
- EndEntityCertificate/V1

如果您尝试创建的 CA，其路径长度大于或等于其颁发证书 CA 的路径长度，`IssueCertificate` API 将返回错误。

有关证书模板的详细信息，请参阅使用 Amazon 私有 CA 证书模板。

使用自动设置 CA 层次结构 Amazon CloudFormation

当你确定了 CA 层次结构的设计后，你可以使用 Amazon CloudFormation 模板对其进行测试并投入生产。有关此类模板的示例，请参阅《Amazon CloudFormation 用户指南》中的声明私有 CA 层次结构。

管理私有 CA 生命周期

CA 证书具有固定的生命周期，即有效期。当 CA 证书到期时，CA 层次结构中由其 CAs 下属机构直接或间接颁发的所有证书都将失效。您可以通过提前规划避免 CA 证书过期。

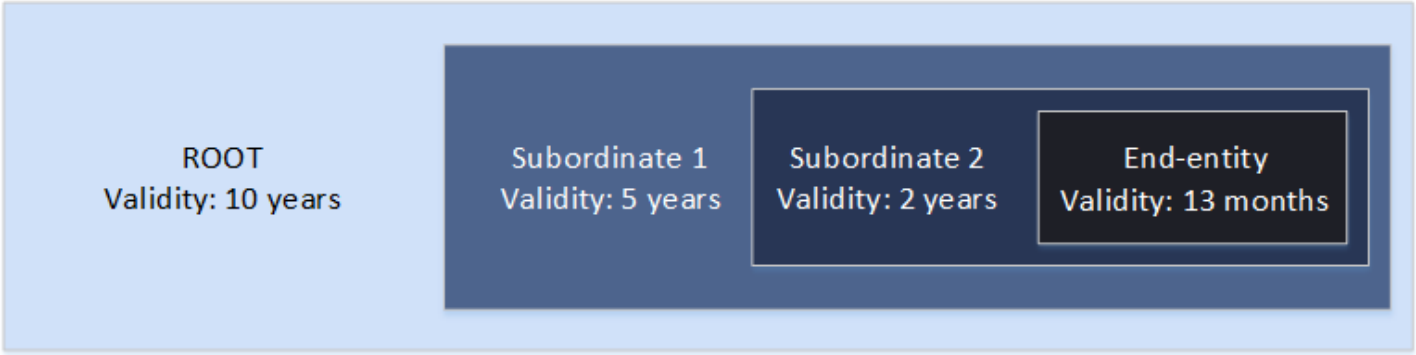
选择有效期

X.509 证书的有效期是必填的基本证书字段。它确定颁发证书 CA 证明证书可信的时间范围，但吊销除外。（根证书是自签名的，证明其自身的有效期。）

Amazon 私有 CA 并 Amazon Certificate Manager 协助配置证书有效期，但须遵守以下限制：

- 由管理的证书的有效期 Amazon 私有 CA 必须短于或等于颁发该证书的 CA 的有效期。换句话说，子证书 CAs 和最终实体证书的寿命不能超过其父证书。尝试使用 `IssueCertificate` API 颁发有效期大于或等于其父 CA 的 CA 证书将失败。
- 由 Amazon Certificate Manager（ACM 生成私钥的证书）颁发和管理的证书的有效期为 13 个月（395 天）。ACM 管理这些证书的续订过程。如果您使用直接颁 Amazon 私有 CA 发证书，则可以选择任何有效期。

下图显示了嵌套有效期的典型配置。根证书的寿命最长；终端实体证书的寿命相对较短；从属证书介于这些极端 CAs 之间。



规划 CA 层次结构时，请确定 CA 证书的最佳生命周期。从要颁发的终端实体证书的所需生命周期反推。

终端实体证书

终端实体证书应具有与使用案例对应的有效期。较短的生命周期可在证书的私有密钥丢失或被盗的情况下，最大限度地减少证书的风险。然而，较短的生命周期意味着经常续订。未能续订即将到期的证书可能会导致停机。

如果发生安全漏洞，分布式使用的终端实体证书也会造成逻辑问题。您在规划时应考虑证书的续订和分发、受损证书的吊销以及吊销传播到依赖证书的客户端的速度。

通过 ACM 颁发的终端实体证书的默认有效期为 13 个月（395 天）。在中 Amazon 私有 CA，您可以使用 `IssueCertificate` API 来应用任何有效期，前提是该有效期少于签发的 CA 的有效期。

从属 CA 证书

从属 CA 证书的有效期应比其颁发的证书长得多。CA 证书比较合适的有效性范围是其颁发的任何子 CA 证书或终端实体证书的两到五倍。例如，假设您具有两级 CA 层次结构（根 CA 和一个从属 CA）。如果您要颁发具有一年生命周期的终端实体证书，则可以将从属颁发证书 CA 的生命周期配置为三年。这是中从属 CA 证书的默认有效期 Amazon 私有 CA。从属 CA 证书可以更改而无需替换根 CA 证书。

根证书

对根 CA 证书的更改会影响整个 PKI（公有密钥基础设施），并要求您更新所有依赖客户端操作系统和浏览器信任存储。为了最大限度地减少操作影响，您应为根证书选择较长的有效期。根证书的 Amazon 私有 CA 默认期限为十年。

管理 CA 继任

您可以通过两种方法管理 CA 继承：替换旧 CA，或重新颁发具有新有效期的 CA。

更换旧的 CA

要替换旧 CA，请创建新 CA 并将其链接到同一父 CA。之后，您从该新 CA 颁发证书。

从新 CA 颁发的证书具有新的 CA 链。建立新 CA 后，您可以禁用旧 CA 以阻止其颁发新证书。禁用后，旧 CA 支持吊销 CA 颁发的旧证书，而且，如果配置为这样做，它将继续通过 OCSP 证书吊销列表 () 来验证 and/or 证书。CRLs当从旧 CA 颁发的最后一个证书过期时，您可以删除旧 CA。您可以为从该 CA 颁发的所有证书生成审计报告，以确认颁发的所有证书都已过期。如果旧 CA 有下属 CA CAs，则还必须替换它们，因为从属 CAs证书会同时过期，或者在其父 CA 之前过期。首先替换层次结构中需要替换的最高级别 CA。然后 CAs 在随后的每个较低级别创建新的替换下属。

Amazon 建议您根据需要在的名称中包含 CA 生成标识符。 CAs 例如，假设您将第一代 CA 命名为“公司根 CA”。创建第二代 CA 时，将其命名为“公司根 CA G2”。当两者都 CAs 未过期时，这种简单的命名约定可以帮助避免混淆。

这种 CA 继承方法是首选的，因为它轮换 CA 的私有密钥。轮换私有密钥是 CA 密钥的最佳实践。轮换频率应与密钥使用频率成正比：颁发的证书越多 CAs，则应更频繁地轮换。

> **Note**
>
> 如果您替换 CA，则通过 ACM 颁发的私有证书无法续订。如果将 ACM 用于颁发和续订，则您必须重新颁发 CA 证书以延长 CA 的生命周期。

补发旧的 CA

当 CA 接近到期时，延长其使用寿命的另一种方法是重新颁发具有新到期日期的 CA 证书。重新颁发会保留所有 CA 元数据，并保留现有的私有密钥和公有密钥。在这种情况下，现有证书链和 CA 颁发的未到期的终端实体证书在到期之前一直有效。新证书的颁发也可以不间断地继续进行。要使用重新颁发的证书更新 CA，请按照 安装 CA 证书 中所述的常规安装过程进行操作。

> **Note**
>
> 建议更换即将到期的 CA，而不是重新颁发其证书，因为轮换到新密钥对可以获得安全优势。

撤销 CA

您可以通过吊销 CA 的基础证书来吊销 CA。这也有效地吊销了该 CA 颁发的所有证书。吊销信息通过 OCSP 或 CRL 的方式分发给客户端。只有当您希望吊销 CA 证书颁发的所有终端实体和从属 CA 证书时，才应吊销该 CA 证书。

规划您的 Amazon 私有 CA 证书吊销方法

在规划私有 PKI 时 Amazon 私有 CA，应考虑如何处理不再希望端点信任已颁发的证书的情况，例如端点的私钥被泄露的情况。解决此问题的常见方法是使用短期证书或配置证书吊销。短期证书将在很短的时间（几小时或几天）内过期，因此吊销没有任何意义，证书失效的时间与通知端点吊销证书的时间相差无几。本节介绍 Amazon 私有 CA 客户的吊销选项，包括配置和最佳实践。

寻找吊销方法的客户可以选择在线证书状态协议 (OCSP)、证书吊销列表 (CRLs) 或两者兼而有之。

> **Note**
>
> 如果您在未配置吊销的情况下创建 CA，以后可以随时对其进行配置。有关更多信息，请参阅在中更新私有 CA Amazon 私有证书颁发机构。

- 在线证书状态协议（OCSP）

  Amazon 私有 CA 提供完全托管的 OCSP 解决方案，无需客户自己操作基础架构，即可通知端点证书已被吊销。客户可以使用 Amazon 私有 CA 控制台、API、CLI 或通过 Amazon CloudFormation单个操作在新的或现有 CAs 版本上启用 OCSP。尽管 CRLs 在端点上存储和处理并且可能会过时，但 OCSP 存储和处理要求是在响应者后端同步处理的。

  为证书颁发机构启用 OCSP 时，会在颁发的每个新证书的授权信息访问 (AIA) 扩展中 Amazon 私有 CA 包含 OCSP 响应者的 URL。该扩展允许 Web 浏览器等客户端查询响应程序并确定是否可以信任终端实体或从属 CA 证书。响应程序返回经过加密签名的状态消息，以确保其真实性。

  Amazon 私有 CA OCSP 响应器符合 RF C 5019。

  OCSP 注意事项

  - OCSP 状态消息的签名算法与发放 CA 配置使用的签名算法相同。 CAs 在 Amazon 私有 CA 控制台中创建，默认使用 SHA256 WITHRSA 签名算法。其他支持的算法可以在 CertificateAuthorityConfigurationAPI 文档中找到。

- APIPassthrough 如果启用 OCSP 响应器，则 CSRPassthrough证书模板将不适用于 AIA 扩展。
- 托管 OCSP 服务的端点可在公共互联网上访问。想要使用 OCSP 但又不想拥有公共端点的客户需要运行自己的 OCSP 基础架构。

- 证书吊销清单 () CRLs

  证书吊销列表 (CRL) 是一个包含在预定到期日期之前已撤销的证书列表的文件。CRL 包含一份不应再信任的证书列表、吊销原因和其他相关信息。

  配置证书颁发机构 (CA) 时，可以选择 Amazon 私有 CA 创建完整的 CRL 还是分区的 CRL。您的选择决定了证书颁发机构可以颁发和吊销的最大证书数量。有关更多信息，请参阅 Amazon 私有 CA 配额。

  CRL 注意事项

  - 内存和带宽注意事项：由于本地下载和处理要求， CRLs 需要比 OCSP 更多的内存。但是，与 OCSP 相比，通过缓存吊销列表而不是检查每个连接的状态， CRLs可能会减少网络带宽。对于内存受限的设备，例如某些物联网设备，可以考虑使用分区。 CRLs
  - 更改 CRL 类型：从完整的 CRL 更改为分区 CRL 时，根据需要 Amazon 私有 CA 创建新分区，并将 IDP 扩展名添加到所有分区 CRLs，包括原始分区。从分区更改为完成仅更新一个 CRL，并防止将来撤消与先前分区关联的证书。

> **Note**
>
> OCSP 和 OCS CRLs P 在撤销和状态更改可用性之间都存在一定的延迟。
>
> - 当您吊销证书时，OCSP 响应最多可能需要 60 分钟才能反映新状态。通常，OCSP 倾向于支持更快地分发撤销信息，因为与客户端 CRLs 可以缓存数天的撤销信息不同，OCSP 响应通常不会被客户端缓存。
> - 通常在吊销证书大约 30 分钟后更新 CRL。如果 CRL 更新因任何原因失败， Amazon 私有 CA 则每 15 分钟再尝试一次。

吊销配置的一般要求

以下要求适用于所有吊销配置。

- 禁用 CRLs 或 OCSP 的配置必须仅包含该`Enabled=False`参数，如果包含`CustomCname`或等其他参数，`ExpirationInDays`则配置将失败。

- 在 CRL 配置中，S3BucketName 参数必须符合 Amazon Simple Storage Service 桶命名规则。
- 包含 CRLs 或 OCSP 的自定义规范名称 (CNAME) 参数的配置必须符合对在 CNAME 中使用特殊字符的RFC7230限制。
- 在 CRL 或 OCSP 配置中，CNAME 参数的值不得包含协议前缀，例如“http://”或“https://”。

主题

- 为以下各项设置 CRL Amazon 私有 CA
- 自定义 OCSP 网址 Amazon 私有 CA

为以下各项设置 CRL Amazon 私有 CA

在将证书吊销列表 (CRL) 配置为 C A 创建过程的一部分之前，可能需要事先进行一些设置。本节说明在创建附有 CRL 的 CA 之前应了解的先决条件和选项。

有关使用在线证书状态协议（OCSP）作为 CRL 的备选或补充的信息，请参阅 Certificate revocation options 和 自定义 OCSP 网址 Amazon 私有 CA。

主题

- CRL 类型
- CRL 结构
- 亚马逊 S3 CRLs 中的访问策略
- 使用启用 S3 阻止公共访问 (BPA) CloudFront
- 确定 CRL 分发点 (CDP) URI

CRL 类型

- 完成-默认设置。 Amazon 私有 CA 为 CA 颁发的所有已被吊销的未过期证书维护一个未分区的 CRL 文件。根据 RFC 5280 中的定义， Amazon 私有 CA 颁发的每个证书都通过其 CRL 分发点 (CDP) 扩展绑定到特定 CRL。启用完整 CRL 后，每个 CA 最多可以有 100 万个私有证书。有关更多信息，请参阅Amazon 私有 CA 配额。
- 已分区-与完整版相比 CRLs，分区可以 CRLs 显著增加您的私有 CA 可以颁发的证书数量，并使您不必频繁轮换。 CAs

> **Important**
>
> 使用分区时 CRLs，必须验证 CRL 的关联颁发分发点 (IDP) URI 是否与证书的 CDP URI 相匹配，以确保已获取正确的 CRL。 Amazon 私有 CA 将 IDP 扩展标记为关键，您的客户必须能够处理该扩展。

CRL 结构

每个 CRL 是一个 DER 编码文件。要下载文件并使用 OpenSSL 进行查看，请使用类似如下的命令：

```
openssl crl -inform DER -in path-to-crl-file -text -noout
```

CRLs 格式如下：

```
Certificate Revocation List (CRL):
          Version 2 (0x1)
      Signature Algorithm: sha256WithRSAEncryption
          Issuer: /C=US/ST=WA/L=Seattle/O=Example Company CA/OU=Corporate/
CN=www.example.com
          Last Update: Feb 26 19:28:25 2018 GMT
          Next Update: Feb 26 20:28:25 2019 GMT
          CRL extensions:
              X509v3 Authority Key Identifier:
                  keyid:AA:6E:C1:8A:EC:2F:8F:21:BC:BE:80:3D:C5:65:93:79:99:E7:71:65

              X509v3 CRL Number:
                  1519676905984
  Revoked Certificates:
      Serial Number: E8CBD2BEDB122329F97706BCFEC990F8
          Revocation Date: Feb 26 20:00:36 2018 GMT
          CRL entry extensions:
              X509v3 CRL Reason Code:
                  Key Compromise
      Serial Number: F7D7A3FD88B82C6776483467BBF0B38C
          Revocation Date: Jan 30 21:21:31 2018 GMT
          CRL entry extensions:
              X509v3 CRL Reason Code:
                  Key Compromise
      Signature Algorithm: sha256WithRSAEncryption
           82:9a:40:76:86:a5:f5:4e:1e:43:e2:ea:83:ac:89:07:49:bf:
```

```
        c2:fd:45:7d:15:d0:76:fe:64:ce:7b:3d:bb:4c:a0:6c:4b:4f:
        9e:1d:27:f8:69:5e:d1:93:5b:95:da:78:50:6d:a8:59:bb:6f:
        49:9b:04:fa:38:f2:fc:4c:0d:97:ac:02:51:26:7d:3e:fe:a6:
        c6:83:34:b4:84:0b:5d:b1:c4:25:2f:66:0a:2e:30:f6:52:88:
        e8:d2:05:78:84:09:01:e8:9d:c2:9e:b5:83:bd:8a:3a:e4:94:
        62:ed:92:e0:be:ea:d2:59:5b:c7:c3:61:35:dc:a9:98:9d:80:
        1c:2a:f7:23:9b:fe:ad:6f:16:7e:22:09:9a:79:8f:44:69:89:
        2a:78:ae:92:a4:32:46:8d:76:ee:68:25:63:5c:bd:41:a5:5a:
        57:18:d7:71:35:85:5c:cd:20:28:c6:d5:59:88:47:c9:36:44:
        53:55:28:4d:6b:f8:6a:00:eb:b4:62:de:15:56:c8:9c:45:d7:
        83:83:07:21:84:b4:eb:0b:23:f2:61:dd:95:03:02:df:0d:0f:
        97:32:e0:9d:38:de:7c:15:e4:36:66:7a:18:da:ce:a3:34:94:
        58:a6:5d:5c:04:90:35:f1:8b:55:a9:3c:dd:72:a2:d7:5f:73:
        5a:2c:88:85
```

> **Note**
>
> 只有在颁发了引用它的证书时，CRL 才会存放到 Amazon S3 中。在此之前，Amazon S3 桶中只有显示一个 `acm-pca-permission-test-key` 文件。

亚马逊 S3 CRLs 中的访问策略

如果您计划创建 CRL，则需要准备一个 Amazon S3 存储桶来存储它。 Amazon 私有 CA 自动将 CRL 存入您指定的 Amazon S3 存储桶中，并定期对其进行更新。有关更多信息，请参阅创建存储桶。

您的 S3 桶必须通过附加的 IAM 权限策略进行保护。授权用户和服务委托人需要`Put`权限 Amazon 私有 CA 才能在存储桶中放置对象，以及检索对象的`Get`权限。

> **Note**
>
> IAM 策略配置取决于 Amazon Web Services 区域 所涉及的内容。区域分为两类：
>
> - 默认启用区域-默认情况下为所有区域启用的区域。 Amazon Web Services 账户
> - 默认禁用的区域 – 默认情况下禁用但可以由客户手动启用的区域。
>
> 有关更多信息以及默认禁用区域的列表，请参阅管理 Amazon Web Services 区域。有关在 IAM 上下文中对服务主体的讨论，请参阅选择加入区域的Amazon 服务主体。

当您配置 CRLs 为证书吊销方法时， Amazon 私有 CA 会创建 CRL 并将其发布到 S3 存储桶。S3 存储桶需要一个允许 Amazon 私有 CA 服务委托人写入存储桶的 IAM 策略。服务主体的名称因所使用的区域而有所不同，且并非支持所有可能性。

PCA	S3	服务主体
两者位于同一区域		`acm-pca.amazonaws.com`
已启用	已启用	`acm-pca.amazonaws.com`
已禁用	已启用	`acm-pca.Region.amazonaws.com`
已启用	已禁用	不支持

默认策略对 CA 不施加任何 `SourceArn` 限制。我们建议您采用宽松程度较低的政策，例如以下政策，该政策限制对特定 Amazon 账户和特定私有 CA 的访问权限。或者，您可以使用 a ws: SourceOrg ID 条件键来限制对中特定组织的访问权限。 Amazon Organizations有关存储桶策略的更多信息，请参阅 Amazon 简单存储服务的存储桶策略。

如果您选择允许默认策略，以后可以随时修改。

使用启用 S3 阻止公共访问 (BPA) CloudFront

默认情况下，新 Amazon S3 桶是在激活屏蔽公共访问权限（BPA）功能的情况下配置的。BPA 包含在 Amazon S3 安全最佳实践中，是一组访问控制，客户可以使用这些控制来微调对 S3 桶中对象和整个桶的访问权限。当 BPA 处于活动状态且配置正确时，只有经过授权和身份验证的 Amazon 用户才能访问存储桶及其内容。

Amazon 建议在所有 S3 存储桶上使用 BPA，以避免敏感信息泄露给潜在的对手。但是，如果您的PKI 客户通过公共互联网（即未登录 Amazon 帐户）进行检索 CRLs ，则需要进行额外的规划。本节介绍如何使用 Amazon CloudFront（内容分发网络 (CDN)）配置私有 PKI 解决方案，使其 CRLs 无需经过身份验证的客户端访问 S3 存储桶即可提供服务。

> **Note**
>
> 使用 CloudFront 会给您的 Amazon 账户带来额外费用。有关更多信息，请参阅 Amazon CloudFront 定价。

如果您选择将 CRL 存储在启用 BPA 的 S3 存储桶中，但不使用 CloudFront，则必须构建另一个 CDN 解决方案，以确保您的 PKI 客户端可以访问您的 CRL。

为 BPA CloudFront 做好准备

创建一个可以访问您的私有 S3 存储桶的 CloudFront 分配，并且可以提供 CRLs 给未经身份验证的客户端。

为 CR CloudFront L 配置发行版

1. 使用《Amazon CloudFront 开发者指南》中创建分配中的步骤创建新 CloudFront 分配。

   完成该过程时，请应用以下设置：

   - 在源域名中，选择您的 S3 桶。
   - 为限制存储桶访问选择是。
   - 为源访问身份选择创建新身份。
   - 在授予对存储桶的读取权限下选择是，更新存储桶策略。

   > **Note**
   >
   > 在此过程中， CloudFront 修改您的存储桶策略以允许其访问存储桶对象。考虑编辑此策略，使其仅允许访问 `crl` 文件夹下的对象。

2. 初始化发行版后，在 CloudFront 控制台中找到其域名并将其保存以供下一个步骤使用。

   > **Note**
   >
   > 如果您的 S3 存储桶是在 us-east-1 以外的区域新创建的，则当您通过访问已发布的应用程序时，可能会出现 HTTP 307 临时重定向错误。 CloudFront桶的地址可能需要几个小时才能传播。

为 BPA 设置 CA

在配置新 CA 时，请将别名添加到您的 CloudFront发行版中。

为你的 CA 配置别名记录 CloudFront

- 使用 在中创建私有 CA Amazon 私有 CA 创建您的 CA。

  执行该过程时，撤销文件revoke_config.txt应包含以下几行，以指定非公共 CRL 对象并在中提供分发端点的 URL： CloudFront

  ```
  "S3ObjectAcl":"BUCKET_OWNER_FULL_CONTROL",
   "CustomCname":"abcdef012345.cloudfront.net"
  ```

  之后，当您使用此 CA 颁发证书时，这些证书将包含如下所示的块：

  ```
  X509v3 CRL Distribution Points:
   Full Name:
   URI:http://abcdef012345.cloudfront.net/crl/01234567-89ab-
  cdef-0123-456789abcdef.crl
  ```

> **Note**
>
> 如果您拥有由此 CA 颁发的较旧证书，则他们将无法访问 CRL。

确定 CRL 分发点 (CDP) URI

如果您需要在工作流程中使用 CRL 分发点 (CDP) URI，则可以使用该证书上的 CRL URI 颁发证书，也可以使用以下方法。这仅适用于完整版 CRLs。分区后面 CRLs 会随机附加一个 GUID。

如果您使用 S3 存储桶作为 CA 的 CRL 分发点 (CDP)，则 CDP URI 可以采用以下格式之一。

- http://*amzn-s3-demo-bucket*.s3.*region-code*.amazonaws.com/crl/*CA-ID*.crl
- http://s3.*region-code*.amazonaws.com/*amzn-s3-demo-bucket*/crl/*CA-ID*.crl

如果您为自己的 CA 配置了自定义 CNAME，则 CDP URI 将包含别名记录，例如，http://*alternative.example.com*/crl/*CA-ID*.crl

默认情况下，使用 IPv4仅限amazonaws.com区域的终端节点编 Amazon 私有 CA 写 CDP 扩展。要 CRLs 重新使用 IPv6，请执行以下步骤之一，以便 CDPs 在编写时指向 S3 的双栈端点： URLs

- 将您的 CRL 自定义名称设置为 S3 双堆栈终端节点域。例如，*bucketname*.s3.dualstack.*region-code*.amazonaws.com
- 设置你自己的 CNAME DNS 记录，指向相关的 S3 双栈端点，然后将其用作 CRL 自定义名称

自定义 OCSP 网址 Amazon 私有 CA

> **Note**
>
> 本主题适用于想要为品牌或其他目的自定义在线证书状态协议 (OCSP) 响应器端点的公共 URL 的客户。如果您计划使用 Amazon 私有 CA 托管 OCSP 的默认配置，则可以跳过本主题并按照配置吊销中的配置说明进行操作。

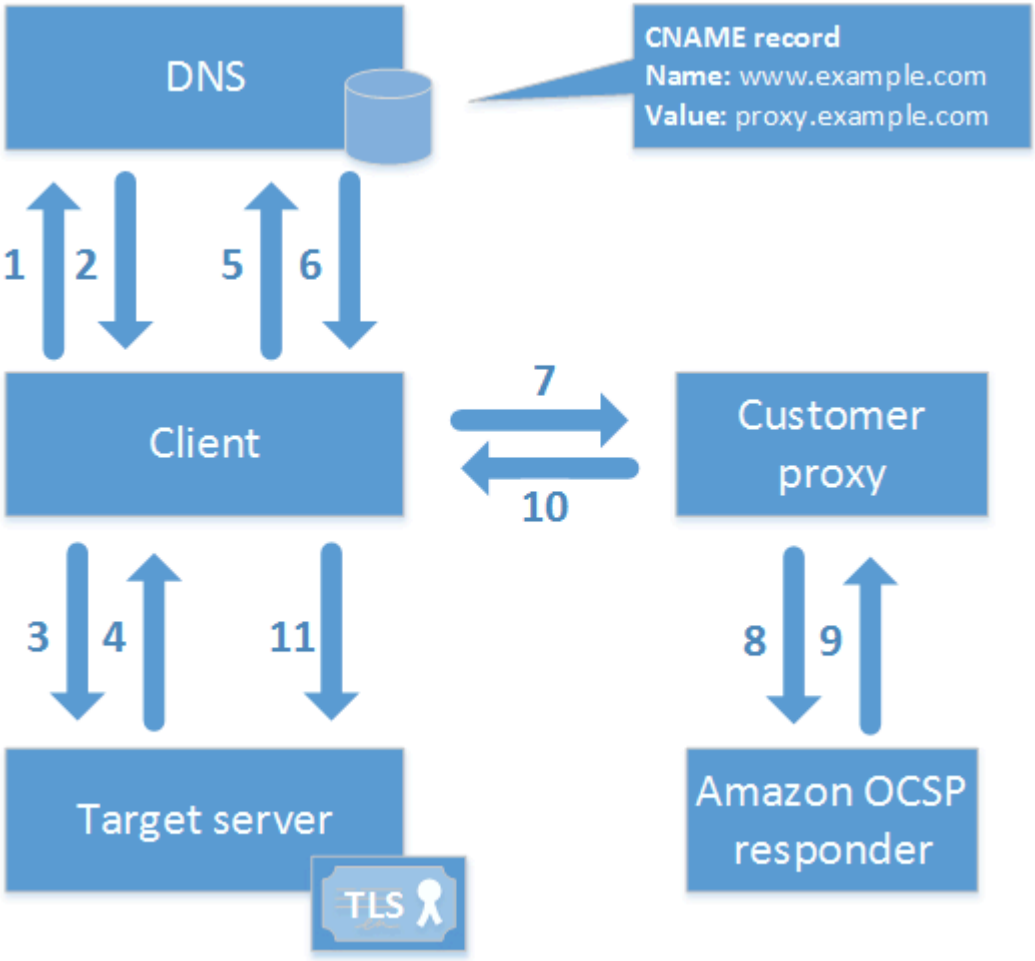
默认情况下，当您为启用 OCSP 时 Amazon 私有 CA，您颁发的每个证书都包含 Amazon OCSP 响应者的 URL。这允许请求加密安全连接的客户端直接向 Amazon发送 OCSP 验证查询。但是，在某些情况下，最好在证书中注明不同的 URL，同时最终仍向 Amazon提交 OCSP 查询。

> **Note**
>
> 有关使用证书吊销列表（CRL）作为 OCSP 备选或补充的信息，请参阅配置吊销和规划证书吊销列表（CRL）。

为 OCSP 配置自定义 URL 涉及三个元素。

- CA 配置 – 在 `RevocationConfiguration` 中为您的 CA 指定自定义 OCSP URL，如 在中创建私有 CA Amazon 私有 CA 中的 示例 2：创建启用 OCSP 和自定义 CNAME 的 CA 中所述。
- DNS – 将 CNAME 记录添加到您的域配置，以将证书中显示的 URL 映射到代理服务器 URL。有关更多信息，请参阅在中创建私有 CA Amazon 私有 CA中的示例 2：创建启用 OCSP 和自定义 CNAME 的 CA。
- 转发代理服务器 – 设置代理服务器，使其能够透明地将收到的 OCSP 流量转发给 Amazon OCSP 响应程序。

下图说明了这些元素协同工作的方式。



如图所示，自定义 OCSP 验证过程包括以下步骤：

1. 客户端查询目标域的 DNS。
2. 客户端收到目标 IP。
3. 客户端打开与目标的 TCP 连接。
4. 客户端收到目标 TLS 证书。
5. 客户端查询 DNS 以获得证书中列出的 OCSP 域。
6. 客户端收到代理 IP。
7. 客户端向代理发送 OCSP 查询。
8. 代理将查询转发到 OCSP 响应程序。
9. 响应程序将证书状态返回给代理。
10. 代理将证书状态转发到客户端。
11. 如果证书有效，则客户端将开始 TLS 握手。

> **Tip**
>
> 在按照上述方式配置 CA 之后，可以使用亚马逊 CloudFront和 Amazon Route 53 实现此示例。
>
> 1. 在中 CloudFront，创建发行版并按如下方式对其进行配置：
>   - 创建与您的自定义 CNAME 相匹配的备用名称。
>   - 将您的证书与其绑定。
>   - 设置`ocsp.acm-pca.<region>.amazonaws.com`为原点。
>   - 要使用 IPv6 连接，请使用双栈端点 `acm-pca-ocsp.<region>.api.aws`
>   - 应用 `Managed-CachingDisabled` 策略。
>   - 将查看器协议策略更改为 HTTP 和 HTTPS。
>   - 将允许的 HTTP 方法设置为 GET、HEAD、OPTIONS、PUT、POST、PATCH、DELETE。
> 2. 在 Route 53 中，创建一个 DNS 记录，将您的自定义 CNAME 映射到 CloudFront 分配的网址。

通过使用 OCSP IPv6

默认 Amazon 私有 CA OCSP 响应器网址仅为 IPv4-only。要使用 OCSP IPv6，请为您的 CA 配置自定义 OCSP 网址。网址可以是：

- 双栈 PCA OCSP 响应器的 FQDN，其形式为 `acm-pca-ocsp.region-name.api.aws`
- 如上所述，您已将其配置为指向双堆栈 OCSP 响应器的别名记录。

了解 Amazon 私有 CA CA 模式

Amazon 私有 CA 支持以两种模式中的任何一种创建证书颁发机构 (CA)。通用证书和短期证书模式会影响 CA 颁发的证书的允许有效期。

> **Note**
>
> Amazon 私有 CA 不对根 CA 证书执行有效性检查。

通用型（默认）

此模式允许 CA 颁发任何有效期的证书。大多数应用程序都使用这种类型的证书。通常，CA 还会指定吊销机制。

短期证书

此模式定义专门颁发最长有效期为七天的证书的 CA。这些短期证书很快便会过期，因此可在没有吊销机制的情况下进行部署。对于某些应用程序来说，频繁部署短期证书比产生吊销的网络和处理开销更有意义。

短期证书必须是证书层次结构中的最后一个 CA。由于私有 CA 必须每七天续订一次，因此会产生很大的开销。

CAs 短期证书模式的成本低于通用 CAs模式。有关更多信息，请参阅Amazon 私有证书颁发机构 定价。

要创建颁发短期证书的 CA，请使用创建 CA 过程将UsageMode参数设置为短期证书。

> **Note**
>
> Amazon Certificate Manager 无法颁发由短期模式的私有 CA 签名的证书。

以下 Amazon 服务支持使用短期证书：

- Amazon AppStream
- Amazon WorkSpaces

规划适应能力 Amazon 私有 CA

Amazon 全球基础设施是围绕 Amazon 区域和可用区构建的。 Amazon 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。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失效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设施相比，可用区具有更高的可用性、容错能力和可扩展性。

有关 Amazon 区域和可用区的更多信息，请参阅Amazon 全球基础设施。

冗余和灾难恢复

在规划 CA 层次结构时，请考虑冗余和灾难恢复。 Amazon 私有 CA 在多个区域中可用，这允许您在多个区域 CAs 中创建冗余。该 Amazon 私有 CA 服务按照 99.9% 可用性的服务级别协议 (SLA) 运行。您至少有两种方法可以考虑用于冗余和灾难恢复。您可以在根 CA 或最高从属 CA 配置冗余。每种方法都各有优缺点。

1. 您可以在两个不同的 Amazon 区域 CAs 中创建两个根以实现冗余和灾难恢复。使用此配置，每个根 CA 可以在一个 Amazon 区域中独立运行，从而在发生单区域灾难时为您提供保护。但是，创建冗余根 CAs 确实会增加操作的复杂性：您需要将根 CA 证书分发到环境中浏览器和操作系统的信任存储区。
2. 您还可以创建冗余的从属服务器 CAs 以部署在每个 Amazon 区域，并将它们链接到单个 Amazon 区域中同一个唯一的根 CA。此方法的好处是，您只需要将单个根 CA 证书分发给环境中的信任存储。限制在于，如果发生影响您的根 CA 所在 Amazon 区域的灾难，则您没有冗余的根 CA。

中的证书颁发机构 Amazon 私有 CA

使用 Amazon 私有证书颁发机构，您可以创建完全 Amazon 托管的根证书颁发机构和从属证书颁发机构 (CA) 层次结构，供组织内部使用。要管理证书吊销，您可以启用在线证书状态协议 (OCSP)、证书吊销列表 (CRL) 或两者。 Amazon 私有 CA 存储和管理您的 CA 证书、CRL 和 OCSP 响应，根机构的私钥由安全存储。 Amazon

> **Note**
>
> 中的 OCSP 实现 Amazon 私有 CA 不支持 OCSP 请求扩展。如果您提交包含多个证书的 OCSP 批量查询， Amazon OCSP 响应器将仅处理队列中的第一个证书，而丢弃其他证书。吊销最多可能需要一个小时才会出现在 OCSP 响应中。

您可以 Amazon 私有 CA 使用 Amazon Web Services 管理控制台 Amazon CLI、和 Amazon 私有 CA API 进行访问。以下主题向您介绍如何使用控制台和 CLI。要了解有关 API 的更多信息，请参阅 Amazon 私有证书颁发机构 API Reference。如需演示如何使用 API 的 Java 示例，请参阅 Amazon 私有 CA 搭配使用 适用于 Java 的 Amazon SDK。

创建活动私有 CA 并配置其访问权限后，即可颁发和检索证书，如中所述颁发和管理证书 Amazon 私有 CA。

主题

- 设置为使用 Amazon 私有 CA
- 在中创建私有 CA Amazon 私有 CA
- 安装 CA 证书
- 控制对私有 CA 的访问权限
- 列出私有 CA
- 查看私有 CA
- 为您的私有 CA 添加标签
- 明白 Amazon 私有 CA CA 状态
- 在中更新私有 CA Amazon 私有证书颁发机构
- 删除私有 CA
- 恢复私有 CA
- 使用外部签名的私有 CA 证书

设置为使用 Amazon 私有 CA

如果您还不是 Amazon Web Services（Amazon）客户，则必须先进行注册才能使用 Amazon 私有 CA。您的账户会自动拥有所有可用服务的访问权限，但您只需为所使用的服务付费。

主题

- 注册获取 Amazon Web Services 账户
- 保护 IAM 用户
- 安装 Amazon Command Line Interface

注册获取 Amazon Web Services 账户

如果您没有 Amazon Web Services 账户，请完成以下步骤来创建一个。

要注册 Amazon Web Services 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 Amazon Web Services 账户，就会创建Amazon Web Services 账户根用户一个。根用户有权访问该账户中的所有 Amazon Web Services 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

Amazon 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择 My Account (我的账户) 来查看当前的账户活动并管理您的账户。

保护 IAM 用户

注册后 Amazon Web Services 账户，开启多重身份验证 (MFA)，保护您的管理用户。有关说明，请参阅《IAM 用户指南》中的 为 IAM 用户启用虚拟 MFA 设备（控制台）。

要允许其他用户访问您的 Amazon Web Services 账户 资源，请创建 IAM 用户。为了保护您的 IAM 用户，请启用 MFA 并仅向 IAM 用户授予执行任务所需的权限。

有关创建和保护 IAM 用户的更多信息，请参阅《IAM 用户指南》中的以下主题：

- 在你的 IAM 用户中创建 Amazon Web Services 账户
- 适用于 Amazon 资源的访问权限管理
- 基于 IAM 身份的策略示例

安装 Amazon Command Line Interface

如果您尚未安装 Amazon CLI 但想使用它，请按照中的说明进行操作Amazon Command Line Interface。在本指南中，我们假设您已配置端点、区域和身份验证详细信息，并且我们在示例命令中省略了这些参数。

在中创建私有 CA Amazon 私有 CA

您可以使用本节中的过程创建根 CA 或从属 CA，从而得到符合组织需要的可审计信任关系层次结构。您可以使用或 Amazon CloudFormation的 Amazon Web Services 管理控制台、PCA 部分创建 CA。Amazon CLI

有关更新已创建 CA 配置的信息，请参阅 在中更新私有 CA Amazon 私有证书颁发机构。

有关使用 CA 为用户、设备和应用程序签署终端实体证书的信息，请参阅 颁发私有终端实体证书。

> **Note**
>
> 从您创建私有 CA 的时间开始，每月将为每个私有 CA 向您的账户收取费用。
> 有关最新的定 Amazon 私有 CA 价信息，请参阅Amazon 私有证书颁发机构 定价。您也可以使用定 Amazon 价计算器来估算成本。

主题

- 创建私有 CA 的 CLI 示例

Console

使用 控制台创建私有 CA

1. 完成以下步骤以使用 Amazon Web Services 管理控制台创建私有 CA。

   开始使用控制台

登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为**https://console.aws.amazon.com/acm-pca/home**。

- 如果您在没有私有 CA 的区域中打开控制台，则会显示介绍页面。选择创建私有 CA。
- 如果您在已创建 CA 的区域中打开控制台，则会打开私有证书颁发机构页面，其中包含您的 CA 列表。选择创建 CA。

2. 在“模式选项”下，选择您的 CA 颁发的证书的到期模式。

- General-purpose— 颁发可配置为任何到期日期的证书。这是默认值。
- Short-lived 证书 — 颁发最长有效期为七天的证书。在某些情况下，较短的有效期可以取代吊销机制。

3. 在控制台的类型选项部分，选择您要创建的私有证书颁发机构的类型。

- 选择根可建立新的 CA 层次结构。此 CA 由自签名证书提供支持。它用作层次结构中其他 CA 和终端实体证书的最终签名颁发机构。
- 选择从属将创建一个 CA，该 CA 必须由层次结构中在其上方的父 CA 签名。从属 CA 通常用于创建其他从属 CA 或向用户、计算机和应用程序颁发终端实体证书。

> **Note**
>
> Amazon 私有 CA 当您的下属 CA 的父 CA 也由托管时，会提供自动签名流程 Amazon 私有 CA。您只需选择要使用的父 CA。
> 您的从属 CA 可能需要由外部信任服务提供商签名。如果是，则会 Amazon 私有 CA 为您提供证书签名请求 (CSR)，您必须下载该请求并使用该请求才能获得签名的 CA 证书。有关更多信息，请参阅 安装由外部父 CA 签名的从属 CA 证书。

4. 在使用者可分辨名称选项下，配置您的私有 CA 的使用者名称。您必须至少输入以下选项之一的值：

- 组织（O）– 例如，公司名称
- 组织单位（OU）– 例如，公司内部的部门
- 国家/地区名称（C）– 两个字母的国家/地区代码
- 州或省名称 – 州或省的全名
- 所在地名称 – 城市的名称

- 公用名 (CN) — 用于标识 CA 的人类可读字符串。

> **Note**
>
> 您可以通过在颁发时应用 APIPassthrough 模板来进一步自定义证书的使用者名称。有关更多信息和详细示例，请参阅 使用 APIPassthrough 模板颁发带有自定义主题名称的证书。

由于支持证书是自签名的，因此您为私有 CA 提供的使用者信息可能比公有 CA 包含的使用者信息更少。有关构成使用者可分辨名称的每个值的更多信息，请参阅 RFC 5280。

5. 在 “密钥算法选项” 下，选择密钥算法和算法强度。默认值为 RSA 2048。可从以下算法中进行选择：

   - ML-DSA-44
   - ML-DSA-65
   - ML-DSA-87
   - RSA 2048
   - RSA 3072
   - RSA 4096
   - ECDSA P256
   - ECDSA P384
   - ECDSA P521

6. 在证书吊销选项下，您可以从两种与使用您的证书的客户端共享吊销状态的方法中进行选择：

   - 激活 CRL 分配
   - 打开 OCSP

   您可以为 CA 配置这两个吊销选项中的任一个、两个都不配置或两个都配置。尽管是可选的，但建议将托管吊销作为最佳实践。在完成此步骤之前，请参阅 规划您的 Amazon 私有 CA 证书吊销方法，了解有关每种方法的优点、可能需要的初步设置以及其他吊销功能的信息。

> **Note**
>
> 如果您在未配置吊销的情况下创建 CA，以后可以随时对其进行配置。有关更多信息，请参阅 在中更新私有 CA Amazon 私有证书颁发机构。

要配置证书吊销选项，请执行以下步骤。

a. 在证书吊销选项下，选择激活 CRL 分配。

b. 在 S3 存储桶 URI 下，从列表中选择一个现有存储桶。

   指定现有存储桶时，必须确保为该账户和存储桶禁用 BPA。否则，创建 CA 的操作将失败。如果 CA 已成功创建，则必须仍手动为其附加策略，然后才能开始生成 CRL。使用 亚马逊 S3 CRLs 中的访问策略 中所述的策略模式之一。有关更多信息，请参阅使用 Amazon S3 控制台添加桶策略。

c. 展开 CRL 设置以获取其他配置选项。

   - 选择启用分区以启用 CRL 的分区。如果您不启用分区，则您的 CA 将遵守吊销证书的最大数量。有关更多信息，请参阅 Amazon 私有证书颁发机构 配额。有关分区 CRL 的更多信息，请参阅 CRL 类型。
   - 添加自定义 CRL 名称可为 Amazon S3 桶创建别名。此名称包含在由 CA 颁发的证书的“CRL 分配点”扩展中（由 RFC 5280 定义）。要通过 IPv6 使用 CRL，请将其设置为存储桶的双栈 S3 端点，如在 IPv6 上使用 CR L 中所述。
   - 添加自定义路径，为您的 Amazon S3 存储桶中的文件路径创建 DNS 别名。
   - 键入有效期（以天为单位），您的 CRL 将保持有效。默认值为 7 天。对于联机 CRL，有效期通常为 2-7 天。 Amazon 私有 CA 将在指定时间段的中点尝试重新生成 CRL。

7. 对于证书吊销选项，请选择启用 OCSP。

   - 在自定义 OCSP 端点 – 可选字段中，您可以为非 Amazon OCSP 端点提供完全限定的域名（FQDN）。要通过 IPv6 使用 OCSP，请将此字段设置为双栈端点，如在 IPv6 上使用 OCS P 中所述。

     在此字段中提供 FQDN 时，将 FQDN Amazon 私有 CA 插入到每个已颁发证书的授权信息访问扩展插件中，以代替 Amazon OCSP 响应者的默认 URL。当端点收到包含自定义 FQDN 的证书时，它会查询该地址以获取 OCSP 响应。要使此机制发挥作用，您需要采取另外两个操作：

- 使用代理服务器将到达您的自定义 FQDN 的流量转发给 Amazon OCSP 响应器。
- 将相应的 CNAME 记录添加到您的 DNS 数据库。

> **Tip**
>
> 有关使用自定义 CNAME 实现完整 OCSP 解决方案的更多信息，请参阅 自定义 OCSP 网址 Amazon 私有 CA。

例如，以下是自定义 OCSP 的 CNAME 记录，该记录将在 Amazon Route 53 中显示。

记录名称	Type	路由策略	优势	Value/Route 前往的交通
alternative.example.com	别名记录	简便	-	proxy.example.com

> **Note**
>
> CNAME 的值不得包含协议前缀，例如“http://”或“https://”。

8. 在添加标签下，您可以选择标记您的 CA。标签是键值对，用作标识和组织 Amazon 资源的元数据。有关 Amazon 私有 CA 标签参数列表以及如何在创建后向 CA 添加标签的说明，请参阅为您的私有 CA 添加标签。

> **Note**
>
> 要在创建过程中将标签附加到私有 CA，CA 管理员必须先将内联 IAM policy 与 `CreateCertificateAuthority` 操作关联并显式允许标记。有关更多信息，请参阅 Tag-on-create: 在创建 CA 时将标签附加到 CA。

9.

在 CA 权限选项下，您可以选择将自动续订权限委托给 Amazon Certificate Manager 服务委托人。如果授予此权限，ACM 只能自动续订此 CA 生成的私有终端实体证书。您可以随时使用 Amazon 私有 CA CreatePermissionAPI 或 create-permission C LI 命令分配续订权限。

这些权限默认启用。

> **Note**
>
> Amazon Certificate Manager 不支持自动续订短期证书。

10. 在定价下，确认您了解私有 CA 的定价。

> **Note**
>
> 有关最新的定 Amazon 私有 CA 价信息，请参阅Amazon 私有证书颁发机构 定价。您也可以使用定 Amazon 价计算器来估算成本。

11. 检查所有输入信息的准确性后，选择创建 CA。CA 的详细信息页面将打开，其状态显示为待处理证书。

> **Note**
>
> 在详细信息页面上，您可以通过选择操作、安装 CA 证书来完成 CA 的配置，也可以稍后返回私有证书颁发机构列表并完成适用于您的情况的安装过程：
>
> - 安装根 CA 证书
> - 安装由托管的从属 CA 证书 Amazon 私有 CA
> - 安装由外部父 CA 签名的从属 CA 证书

CLI

使用 create-certificate-authority 命令可创建私有 CA。必须指定 CA 配置（包含算法和主题名称信息）、撤销配置（如果您计划使用 OCSP 作为 CRL）和 C and/or A 类型（根或从属）。配置和吊销配置详细信息包含在您作为命令参数提供的两个文件中。或者，您还可以配置 CA 使用模式（用于颁发标准或短期证书）、附加标签和提供幂等性令牌。

如果您正在配置 CRL，则在发出 create-certificate-authority 命令之前，必须准备好安全的 Amazon S3 桶。有关更多信息，请参阅 亚马逊 S3 CRLs 中的访问策略。

CA 配置文件可指定以下信息：

- 算法的名称
- 要用于创建 CA 私钥的密钥大小
- CA 用来签署自己的证书签名请求、CRL 和 OCSP 响应的签名算法类型
- X.500 主题信息

OCSP 的吊销配置定义了一个包含以下信息的 `OcspConfiguration` 对象：

- `Enabled` 标签设置为“true”。
- （可选）声明为 `OcspCustomCname` 值的自定义 CNAME。

CRL 的吊销配置定义了一个包含以下信息的 `CrlConfiguration` 对象：

- `Enabled` 标签设置为“true”。
- CRL 有效期，以天为单位（CRL 的有效期）。
- 将包含 CRL 的 Amazon S3 桶。
- （可选）确定 CRL 是否可公共访问的 S3ObjectAcl 值。在此处提供的示例中，阻止公共访问。有关更多信息，请参阅 使用启用 S3 阻止公共访问 (BPA) CloudFront。
- （可选）CA 颁发的证书中包含的 S3 桶的 CNAME 别名。如果 CRL 不可公开访问，则将指向诸如 Amazon CloudFront 之类的分发机制。
- （可选）包含以下信息的`CrlDistributionPointExtensionConfiguration`对象：
  - 该`OmitExtension`标志设置为 “真” 或 “假”。这控制是否将 CDP 扩展的默认值写入 CA 颁发的证书。有关 CDP 扩展的更多信息，请参阅确定 CRL 分发点 (CDP) URI。如果为 “true” OmitExtension ，则 CustomCname 无法设置 A。
- （可选）S3 存储桶中 CRL 的自定义路径。
- （可选）确定 CRL 是完整还是分区的CrlType值。如果未提供，则 CRL 将默认为完成。

> **Note**
>
> 通过定义 `OcspConfiguration` 对象和 `CrlConfiguration` 对象，可以在同一 CA 上启用两种吊销机制。如果不提供任何 --revocation-configuration 参数，则默认情况下两种机制均处于禁用状态。如果您以后需要吊销验证支持，请参阅 更新 CA（CLI）。

有关 CLI 示例，请参阅以下部分。

创建私有 CA 的 CLI 示例

以下示例假设您已使用有效的默认区域、端点和凭证设置了 `.aws` 配置目录。有关配置 Amazon CLI 环境的信息，请参阅配置和凭证文件设置。为了便于阅读，我们在示例命令中以 JSON 文件的形式提供 CA 配置和吊销输入。根据需要修改示例文件以供您使用。

除非另有说明，否则所有示例都使用以下 `ca_config.txt` 配置文件。

文件：ca_config.txt

```
{
   "KeyAlgorithm":"RSA_2048",
   "SigningAlgorithm":"SHA256WITHRSA",
   "Subject":{
      "Country":"US",
      "Organization":"Example Corp",
      "OrganizationalUnit":"Sales",
      "State":"WA",
      "Locality":"Seattle",
      "CommonName":"www.example.com"
   }
}
```

示例 1：创建启用 OCSP 的 CA

在此示例中，吊销文件启用默认 OCSP 支持，即使用 Amazon 私有 CA 响应器检查证书状态。

文件：适用于 OCSP 的 revoke_config.txt

```
{
```

```
   "OcspConfiguration":{
      "Enabled":true
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
     --certificate-authority-configuration file://ca_config.txt \
     --revocation-configuration file://revoke_config.txt \
     --certificate-authority-type "ROOT" \
     --idempotency-token 01234567 \
     --tags Key=Name,Value=MyPCA
```

如果成功，此命令将输出新 CA 的 Amazon 资源名称（ARN）。

```
{
 "CertificateAuthorityArn":"arn:aws:acm-pca:region:account:
       certificate-authority/CA_ID"
}
```

命令

```
$ aws acm-pca create-certificate-authority \
 --certificate-authority-configuration file://ca_config.txt \
 --revocation-configuration file://revoke_config.txt \
 --certificate-authority-type "ROOT" \
 --idempotency-token 01234567 \
 --tags Key=Name,Value=MyPCA-2
```

如果成功，此命令将输出 CA 的 Amazon 资源名称（ARN）。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用以下命令检查 CA 的配置。

```
$ aws acm-pca describe-certificate-authority \
```

```
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述应包含以下部分。

```
"RevocationConfiguration": {
   ...
   "OcspConfiguration": {
      "Enabled": true
   }
   ...
}
```

示例 2：创建启用 OCSP 和自定义 CNAME 的 CA

在此示例中，吊销文件启用了自定义 OCSP 支持。`OcspCustomCname` 参数采用完全限定域名（FQDN）作为其值。

在此字段中提供 FQDN 时，将 FQDN Amazon 私有 CA 插入到每个已颁发证书的授权信息访问扩展插件中，以代替 Amazon OCSP 响应者的默认 URL。当端点收到包含自定义 FQDN 的证书时，它会查询该地址以获取 OCSP 响应。要使此机制发挥作用，您需要采取另外两个操作：

- 使用代理服务器将到达您的自定义 FQDN 的流量转发给 Amazon OCSP 响应器。
- 将相应的 CNAME 记录添加到您的 DNS 数据库。

> **Tip**
>
> 有关使用自定义 CNAME 实现完整 OCSP 解决方案的更多信息，请参阅 自定义 OCSP 网址 Amazon 私有 CA。

例如，以下是自定义 OCSP 的 CNAME 记录，该记录将在 Amazon Route 53 中显示。

记录名称	Type	路由策略	优势	Value/Route 前往的交通
alternative.example.com	别名记录	简便	-	proxy.example.com

> **Note**
>
> CNAME 的值不得包含协议前缀，例如“http://”或“https://”。

文件：适用于 OCSP 的 revoke_config.txt

```
{
   "OcspConfiguration":{
      "Enabled":true,
      "OcspCustomCname":"alternative.example.com"
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
 --certificate-authority-configuration file://ca_config.txt \
 --revocation-configuration file://revoke_config.txt \
 --certificate-authority-type "ROOT" \
 --idempotency-token 01234567 \
 --tags Key=Name,Value=MyPCA-3
```

如果成功，此命令将输出 CA 的 Amazon 资源名称（ARN）。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用以下命令检查 CA 的配置。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述应包含以下部分。

```
"RevocationConfiguration": {
   ...
```

```
   "OcspConfiguration": {
      "Enabled": true,
      "OcspCustomCname": "alternative.example.com"
   }
   ...
}
```

示例 3：创建带有附加 CRL 的 CA

在此示例中，吊销配置定义了 CRL 参数。

文件：revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"amzn-s3-demo-bucket"
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

如果成功，此命令将输出 CA 的 Amazon 资源名称（ARN）。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用以下命令检查 CA 的配置。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
```

```
      --output json
```

此描述应包含以下部分。

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
   },
   ...
}
```

示例 4：创建带有附加 CRL 并启用自定义 CNAME 的 CA

在此示例中，吊销配置定义了包含自定义 CNAME 的 CRL 参数。

文件：revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "CustomCname": "alternative.example.com",
      "S3BucketName":"amzn-s3-demo-bucket"
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

如果成功，此命令将输出 CA 的 Amazon 资源名称（ARN）。

```
{
```

```
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用以下命令检查 CA 的配置。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述应包含以下部分。

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "CustomCname": "alternative.example.com",
      "S3BucketName": "amzn-s3-demo-bucket",
   ...
   }
}
```

示例 5：创建 CA 并指定使用模式

在此示例中，CA 使用模式是在创建 CA 时指定的。如果未指定，则使用模式参数默认为 GENERAL_PURPOSE。在此示例中，参数设置为 SHORT_LIVED_CERTIFICATE，这意味着 CA 将颁发最长有效期为七天的证书。在配置吊销不方便的情况下，已被泄露的短期证书很快就会过期，这是正常操作的一部分。因此，此示例 CA 缺少吊销机制。

> **Note**
>
> Amazon 私有 CA 不对根 CA 证书执行有效性检查。

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
      --certificate-authority-type "ROOT" \
      --usage-mode SHORT_LIVED_CERTIFICATE \
```

```
      --tags Key=usageMode,Value=SHORT_LIVED_CERTIFICATE
```

使用中的describe-certificate-authority命令显示 Amazon CLI 有关生成的 CA 的详细信息，如以下命令所示：

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn arn:aws:acm:region:account:certificate-
authority/CA_ID
```

```
{
    "CertificateAuthority":{
       "Arn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
       "CreatedAt":"2022-09-30T09:53:42.769000-07:00",
       "LastStateChangeAt":"2022-09-30T09:53:43.784000-07:00",
       "Type":"ROOT",
       "UsageMode":"SHORT_LIVED_CERTIFICATE",
       "Serial":"serial_number",
       "Status":"PENDING_CERTIFICATE",
       "CertificateAuthorityConfiguration":{
          "KeyAlgorithm":"RSA_2048",
          "SigningAlgorithm":"SHA256WITHRSA",
          "Subject":{
             "Country":"US",
             "Organization":"Example Corp",
             "OrganizationalUnit":"Sales",
             "State":"WA",
             "Locality":"Seattle",
             "CommonName":"www.example.com"
          }
       },
       "RevocationConfiguration":{
          "CrlConfiguration":{
             "Enabled":false
          },
          "OcspConfiguration":{
             "Enabled":false
          }
       },
 ...
```

示例 6：创建用于 Active Directory 登录的 CA

您可以创建适合在 Microsoft Active Directory（AD）的 Enterprise NTAuth 存储中使用的私有 CA，它可以在其中颁发卡登录或域控制器证书。有关将 CA 证书导入 AD 的信息，请参阅如何将第三方证书颁发机构（CA）证书导入企业 NTAuth 存储。

通过调用 -dspublish 选项，可以使用 Microsoft certutil 工具在 AD 中发布 CA 证书。使用 certutil 发布到 AD 的证书在整个林中都受信任。使用组策略，您还可以将信任限制为整个林的子集，例如单个域或域中的一组计算机。要使登录生效，还必须在 NTauth 存储中发布颁发 CA。有关更多信息，请参阅使用组策略将证书分发到客户端计算机。

此示例使用以下 `ca_config_AD.txt` 配置文件。

文件：ca_config_ AD.txt

```
{
   "KeyAlgorithm":"RSA_2048",
   "SigningAlgorithm":"SHA256WITHRSA",
   "Subject":{
      "CustomAttributes":[
         {
            "ObjectIdentifier":"2.5.4.3",
            "Value":"root CA"
         },
         {
            "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
            "Value":"example"
         },
         {
            "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
            "Value":"com"
         }
      ]
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config_AD.txt \
      --certificate-authority-type "ROOT" \
      --tags Key=application,Value=ActiveDirectory
```

如果成功，此命令将输出 CA 的 Amazon 资源名称（ARN）。

```
{
 "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
 }
```

使用以下命令检查 CA 的配置。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述应包含以下部分。

```
...

"Subject":{
   "CustomAttributes":[
      {
         "ObjectIdentifier":"2.5.4.3",
         "Value":"root CA"
      },
      {
         "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
         "Value":"example"
      },
      {
         "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
         "Value":"com"
      }
   ]
}
...
```

示例 7：创建一个 Matter CA，附带一个 CRL，且已颁发的证书中省略了 CDP 扩展名

您可以创建适合颁发 Matter 智能家居标准证书的私有 CA。在此示例中，中的 CA 配置`ca_config_PAA.txt`定义了将供应商 ID (VID) 设置为 FFF1 的 Matter 产品认证机构 (PAA)。

文件：ca_config_ PAA.txt

```
{
   "KeyAlgorithm":"EC_prime256v1",
   "SigningAlgorithm":"SHA256WITHECDSA",
   "Subject":{
      "Country":"US",
      "Organization":"Example Corp",
      "OrganizationalUnit":"SmartHome",
      "State":"WA",
      "Locality":"Seattle",
      "CommonName":"Example Corp Matter PAA",
   "CustomAttributes":[
      {
        "ObjectIdentifier":"1.3.6.1.4.1.37244.2.1",
        "Value":"FFF1"
      }
    ]
  }
}
```

撤销配置启用 CRL，并将 CA 配置为省略所有已颁发的证书中的默认 CDP URL。

文件：revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"amzn-s3-demo-bucket",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config_PAA.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

如果成功，此命令将输出 CA 的 Amazon 资源名称（ARN）。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用以下命令检查 CA 的配置。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述应包含以下部分。

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   },
   ...
}
...
```

安装 CA 证书

请完成以下过程以创建和安装私有 CA 证书。然后您就可以使用 CA。

Amazon 私有 CA 支持安装 CA 证书的三种方案：

- 为托管的根 CA 安装证书 Amazon 私有 CA
- 安装由 Amazon 私有 CA托管其父颁发机构的从属 CA 证书
- 安装其父颁发机构在外部托管的从属 CA 证书

以下各节介绍了每个方案的过程。控制台过程从控制台页面私有 CA 开始。

兼容的签名算法

CA 证书的签名算法支持取决于父 CA 的签名算法和 Amazon Web Services 区域。以下限制适用于控制台和 Amazon CLI 操作。

- 采用 RSA 密钥算法的父 CA 可以使用以下签名算法颁发证书：
  - SHA256 RSA
  - SHA384 RSA
  - SHA512 RSA
- 在旧版中 Amazon Web Services 区域，采用 EDCSA 密钥算法的父 CA 可以使用以下签名算法颁发证书：
  - SHA256 ECDSA
  - SHA384 ECDSA
  - SHA512 ECDSA

  遗产 Amazon Web Services 区域 包括：

区域名称	地理位置
eu-north-1	欧洲地区（斯德哥尔摩）
me-south-1	中东（巴林）
ap-south-1	亚太地区（孟买）
eu-west-3	欧洲地区（巴黎）
us-east-2	美国东部（俄亥俄州）
af-south-1	非洲（开普敦）

区域名称	地理位置
eu-west-1	欧洲地区（爱尔兰）
eu-central-1	欧洲地区（法兰克福）
sa-east-1	南美洲（圣保罗）
ap-east-1	亚太地区（香港）
us-east-1	美国东部（弗吉尼亚州北部）
ap-northeast-2	亚太地区（首尔）
eu-west-2	欧洲地区（伦敦）
ap-northeast-1	亚太地区（东京）
us-gov-east-1	Amazon GovCloud (US-East)
us-gov-west-1	Amazon GovCloud (US-West)
us-west-2	美国西部（俄勒冈州）
us-west-1	美国西部（北加利福尼亚）
ap-southeast-1	亚太地区（新加坡）
ap-southeast-2	亚太地区（悉尼）

- 在非旧版中 Amazon Web Services 区域，以下规则适用于 EDCSA：

- 采用 EC_prime256v1 签名算法的父 CA 可以使用 ECDSA P256 颁发证书。
- 采用 EC_secp384r1 签名算法的父 CA 可以使用 ECDSA P384 颁发证书。

- 在每一项中 Amazon Web Services 区域，以下规则都适用于 EDCSA：
  - 采用 ec_secp521r1 签名算法的父 CA 可以使用 ECDSA P521 颁发证书。

安装根 CA 证书

您可以从 Amazon Web Services 管理控制台 或安装根 CA 证书 Amazon CLI。

为私有根 CA 创建和安装证书（控制台）

1. （可选）如果您尚未进入 CA 的详细信息页面，请打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。在私有证书颁发机构页面上，选择状态为待处理证书或活动的根 CA。
2. 选择操作、安装 CA 证书以打开安装根 CA 证书页面。
3. 在指定根 CA 证书参数下，指定以下证书参数：

   - 有效期 – 指定 CA 证书的到期日期和时间。根 CA 证书的 Amazon 私有 CA 默认有效期为 10 年。
   - 签名算法 – 指定根 CA 颁发新证书时使用的签名算法。可用选项因您创建 CA 的 Amazon Web Services 区域 位置而异。有关更多信息，请参阅兼容的签名算法中支持的加密算法 Amazon 私有证书颁发机构、和SigningAlgorithmCertificateAuthorityConfiguration。
     - SHA256 RSA
     - SHA384 RSA
     - SHA512 RSA

   检查您的设置是否正确，然后选择 “确认并安装”。 Amazon 私有 CA 为您的 CA 导出 CSR，使用根 CA 证书模板生成证书，并对证书进行自签名。 Amazon 私有 CA 然后导入自签名的根 CA 证书。
4. CA 的详细信息页面顶部显示安装状态（成功或失败）。如果安装成功，新完成的根 CA 将在常规窗格中显示为活动。

为您的私有根 CA 创建和安装证书 (Amazon CLI)

1. 生成证书签名请求（CSR）。

```
$ aws acm-pca get-certificate-authority-csr \
     --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
     --output text \
     --region region > ca.csr
```

生成的文件 ca.csr 是以 base64 格式编码的 PEM 文件，其内容显示如下。

```
-----BEGIN CERTIFICATE REQUEST-----
MIIC1DCCAbwCAQAwbTELMAkGA1UEBhMCVVMxFTATBgNVBAoMDEV4YW1wbGUgQ29y
cDEOMAwGA1UECwwFU2FsZXMxCzAJBgNVBAgMAldBMRgwFgYDVQQDDA93d3cuZXhh
bXBsZS5jb20xEDAOBgNVBAcMB1NlYXR0bGUwggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQDD+7eQChWUO2m6pHslI7AVSFkWvbQofKIHvbvy7wm8VO9/BuI7
LE/jrnd1jGoyI7jaMHKXPtEP3uNlCzv+oEza07OjgjqPZVehtA6a3/3vdQ1qCoD2
rXpv6VIzcq2onx2X7m+Zixwn2oY1l1ELXP7I5g0GmUStymq+pY5VARPy3vTRMjgC
JEiz8w7VvC15uIsHFAWa2/NvKyndQMPaCNft238wesV5s2cXOUS173jghIShg99o
ymf0TRUgvAGQMCXvsW07MrP5VDmBU7k/AZ9ExsUfMe2OB++fhfQWr2N7/lpC4+DP
qJTfXTEexLfRTLeLuGEaJL+c6fMyG+Yk53tZAgMBAAGgIjAgBgkqhkiG9w0BCQ4x
EzARMA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAA7xxLVI5s1B
qmXMMT44y1DZtQx3RDPanMNGLGO1TmLtyqqnUH49Tla+2p7nrl0tojUf/3PaZ52F
QN09SrFk8qtYSKnMGd5PZL0A+NFsNW+w4BAQNKlg9m617YEsnkztbfKRloaJNYoA
HZaRvbA0lMQ/tU2PKZR2vnao444Ugm0O/t3jx5rj817b31hQcHHQ0lQuXV2kyTrM
ohWeLf2fL+K0xJ9ZgXD4KYnY0zarpreA5RBeO5xs3Ms+oGWc13qQfMBx33vrrz2m
dw5iKjg71uuUUmtDV6ewwGa/VO5hNinYAfogdu5aGuVbnTFT3n45B8WHz2+9r0dn
bA7xUel1SuQ=
-----END CERTIFICATE REQUEST-----
```

您可以使用 OpenSSL 查看和验证 CSR 的内容。

```
openssl req -text -noout -verify -in ca.csr
```

这会生成类似以下内容的输出。

```
verify OK
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
 L=Seattle
        Subject Public Key Info:
```

```
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c3:fb:b7:90:0a:15:94:3b:69:ba:a4:7b:25:23:
                    b0:15:48:59:16:bd:b4:28:7c:a2:07:bd:bb:f2:ef:
                    09:bc:54:ef:7f:06:e2:3b:2c:4f:e3:ae:77:75:8c:
                    6a:32:23:b8:da:30:72:97:3e:d1:0f:de:e3:65:0b:
                    3b:fe:a0:4c:da:d3:b3:a3:82:3a:8f:65:57:a1:b4:
                    0e:9a:df:fd:ef:75:0d:6a:0a:80:f6:ad:7a:6f:e9:
                    52:33:72:ad:a8:9f:1d:97:ee:6f:99:8b:1c:27:da:
                    86:35:97:51:0b:5c:fe:c8:e6:0d:06:99:44:ad:ca:
                    6a:be:a5:8e:55:01:13:f2:de:f4:d1:32:38:02:24:
                    48:b3:f3:0e:d5:bc:2d:79:b8:8b:07:14:05:9a:db:
                    f3:6f:2b:29:dd:40:c3:da:08:d7:ed:db:7f:30:7a:
                    c5:79:b3:67:17:39:44:b5:ef:78:e0:84:84:a1:83:
                    df:68:ca:67:f4:4d:15:20:bc:01:90:30:25:ef:b1:
                    6d:3b:32:b3:f9:54:39:81:53:b9:3f:01:9f:44:c6:
                    c5:1f:31:ed:8e:07:ef:9f:85:f4:16:af:63:7b:fe:
                    5a:42:e3:e0:cf:a8:94:df:5d:31:1e:c4:b7:d1:4c:
                    b7:8b:b8:61:1a:24:bf:9c:e9:f3:32:1b:e6:24:e7:
                    7b:59
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         0e:f1:c4:b5:48:e6:cd:41:aa:65:cc:31:3e:38:cb:50:d9:b5:
         0c:77:44:33:da:9c:c3:46:2c:63:b5:4e:62:ed:ca:aa:a7:50:
         7e:3d:4e:56:be:da:9e:e7:ae:5d:2d:a2:35:1f:ff:73:da:67:
         9d:85:40:dd:3d:4a:b1:64:f2:ab:58:48:a9:cc:19:de:4f:64:
         bd:00:f8:d1:6c:35:6f:b0:e0:10:10:34:a9:60:f6:6e:b5:ed:
         81:2c:9e:4c:ed:6d:f2:91:96:86:89:35:8a:00:1d:96:91:bd:
         b0:34:94:c4:3f:b5:4d:8f:29:94:76:be:76:a8:e3:8e:14:82:
         6d:0e:fe:dd:e3:c7:9a:e3:f3:5e:db:df:58:50:70:71:d0:d2:
         54:2e:5d:5d:a4:c9:3a:cc:a2:15:9e:2d:fd:9f:2f:e2:b4:c4:
         9f:59:81:70:f8:29:89:d8:d3:36:ab:a6:b7:80:e5:10:5e:3b:
         9c:6c:dc:cb:3e:a0:65:9c:d7:7a:90:7c:c0:71:df:7b:eb:af:
         3d:a6:77:0e:62:2a:38:3b:d6:eb:94:52:6b:43:57:a7:b0:c0:
         66:bf:54:ee:61:36:29:d8:01:fa:20:76:ee:5a:1a:e5:5b:9d:
         31:53:de:7e:39:07:c5:87:cf:6f:bd:af:47:67:6c:0e:f1:51:
         e9:75:4a:e4
```

2. 使用上一步中的 CSR 作为 `--csr` 参数的参数，颁发根证书。

> **Note**
>
> 以下示例使用前缀加载`fileb://`所需的输入文件，确保正确 Amazon 私有 CA 解析 Base64-encoded 数据。如果您使用的是低于 1.6.3 的 Amazon CLI 版本，请`file://`改用。

```
$ aws acm-pca issue-certificate \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --csr fileb://ca.csr \
     --signing-algorithm SHA256WITHRSA \
     --template-arn arn:aws:acm-pca:::template/RootCACertificate/V1 \
     --validity Value=365,Type=DAYS
```

3. 检索根证书。

```
$ aws acm-pca get-certificate \
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
 --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID \
 --output text > cert.pem
```

生成的文件 `cert.pem` 是以 base64 格式编码的 PEM 文件，其内容显示如下。

```
-----BEGIN CERTIFICATE-----
MIIDpzCCAo+gAwIBAgIRAIIuOarlQETlUQEOZJGZYdIwDQYJKoZIhvcNAQELBQAw
bTELMAkGA1UEBhMCVVMxFTATBgNVBAoMDEV4YW1wbGUgQ29ycDEOMAwGA1UECwwF
U2FsZXMxCzAJBgNVBAgMAldBMRgwFgYDVQQDDA93d3cuZXhhbXBsZS5jb20xEDAO
BgNVBAcMB1NlYXR0bGUwHhcNMjEwMzA4MTU0NjI3WhcNMjIwMzA4MTY0NjI3WjBt
MQswCQYDVQQGEwJVUzEVMBMGA1UECgwMRXhhbXBsZSBDb3JwMQ4wDAYDVQQLDAVT
YWxlczELMAkGA1UECAwCV0ExGDAWBgNVBAMMD3d3dy5leGFtcGxlLmNvbTEQMA4G
A1UEBwwHU2VhdHRsZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMP7
t5AKFZQ7abqkeyUjsBVIWRa9tCh8oge9u/LvCbxU738G4jssT+Oud3WMajIjuNow
cpc+0Q/e42ULO/6gTNrTs6OCOo9lV6G0Dprf/e91DWoKgPatem/pUjNyraifHZfu
b5mLHCfahjWXUQtc/sjmDQaZRK3Kar6ljlUBE/Le9NEyOAIkSLPzDtW8LXm4iwcU
BZrb828rKd1Aw9oI1+3bfzB6xXmzZxc5RLXveOCEhKGD32jKZ/RNFSC8AZAwJe+x
bTsys/lUOYFTuT8Bn0TGxR8x7Y4H75+F9BavY3v+WkLj4M+olN9dMR7Et9FMt4u4
YRokv5zp8zIb5iTne1kCAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4E
```

```
FgQUaW3+r328uTLokog2TklmoBK+yt4wDgYDVR0PAQH/BAQDAgGGMA0GCSqGSIb3
DQEBCwUAA4IBAQAXjd/7UZ8RDE+PLWSDNGQdLemOBTcawF+tK+PzA4Evlmn9VuNc
g+x3oZvVZSDQBANUz0b9oPeo54aE38dW1zQm2qfTab8822aqeWMLyJ1dMsAgqYX2
t9+u6w3NzRCw8Pvz18V69+dFE5AeXmNP0Z5/gdz8H/NSpctjlzopbScRZKCSlPid
Rf3ZOPm9QP92YpWyYDkfAU04xdDo1vR0MYjKPkl4LjRqSU/tcCJnPMbJiwq+bWpX
2WJoEBXB/p15Kn6JxjI0ze2SnSI48JZ8it4fvxrhOo0VoLNIuCuNXJOwU17Rdl1W
YJidaq7je6k18AdgPA0Kh8y1XtfUH3fTaVw4
-----END CERTIFICATE-----
```

您可以使用 OpenSSL 查看和验证证书的内容。

```
openssl x509 -in cert.pem -text -noout
```

这会生成类似以下内容的输出。

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            82:2e:39:aa:e5:40:44:e5:51:01:0e:64:91:99:61:d2
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
 L=Seattle
        Validity
            Not Before: Mar  8 15:46:27 2021 GMT
            Not After : Mar  8 16:46:27 2022 GMT
        Subject: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
 L=Seattle
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c3:fb:b7:90:0a:15:94:3b:69:ba:a4:7b:25:23:
                    b0:15:48:59:16:bd:b4:28:7c:a2:07:bd:bb:f2:ef:
                    09:bc:54:ef:7f:06:e2:3b:2c:4f:e3:ae:77:75:8c:
                    6a:32:23:b8:da:30:72:97:3e:d1:0f:de:e3:65:0b:
                    3b:fe:a0:4c:da:d3:b3:a3:82:3a:8f:65:57:a1:b4:
                    0e:9a:df:fd:ef:75:0d:6a:0a:80:f6:ad:7a:6f:e9:
                    52:33:72:ad:a8:9f:1d:97:ee:6f:99:8b:1c:27:da:
                    86:35:97:51:0b:5c:fe:c8:e6:0d:06:99:44:ad:ca:
                    6a:be:a5:8e:55:01:13:f2:de:f4:d1:32:38:02:24:
                    48:b3:f3:0e:d5:bc:2d:79:b8:8b:07:14:05:9a:db:
```

```
                    f3:6f:2b:29:dd:40:c3:da:08:d7:ed:db:7f:30:7a:
                    c5:79:b3:67:17:39:44:b5:ef:78:e0:84:84:a1:83:
                    df:68:ca:67:f4:4d:15:20:bc:01:90:30:25:ef:b1:
                    6d:3b:32:b3:f9:54:39:81:53:b9:3f:01:9f:44:c6:
                    c5:1f:31:ed:8e:07:ef:9f:85:f4:16:af:63:7b:fe:
                    5a:42:e3:e0:cf:a8:94:df:5d:31:1e:c4:b7:d1:4c:
                    b7:8b:b8:61:1a:24:bf:9c:e9:f3:32:1b:e6:24:e7:
                    7b:59
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                69:6D:FE:AF:7D:BC:B9:32:E8:92:88:36:4E:49:66:A0:12:BE:CA:DE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         17:8d:df:fb:51:9f:11:0c:4f:8f:2d:64:83:34:64:1d:2d:e9:
         8e:05:37:1a:c0:5f:ad:2b:e3:f3:03:81:2f:96:69:fd:56:e3:
         5c:83:ec:77:a1:9b:d5:65:20:d0:04:03:54:cf:46:fd:a0:f7:
         a8:e7:86:84:df:c7:56:d7:34:26:da:a7:d3:69:bf:3c:db:66:
         aa:79:63:0b:c8:9d:5d:32:c0:20:a9:85:f6:b7:df:ae:eb:0d:
         cd:cd:10:b0:f0:fb:f3:d7:c5:7a:f7:e7:45:13:90:1e:5e:63:
         4f:d1:9e:7f:81:dc:fc:1f:f3:52:a5:cb:63:97:3a:29:6d:27:
         11:64:a0:92:94:f8:9d:45:fd:d9:38:f9:bd:40:ff:76:62:95:
         b2:60:39:1f:01:4d:38:c5:d0:e8:d6:f4:74:31:88:ca:3e:49:
         78:2e:34:6a:49:4f:ed:70:22:67:3c:c6:c9:8b:0a:be:6d:6a:
         57:d9:62:68:10:15:c1:fe:9d:79:2a:7e:89:c6:32:34:cd:ed:
         92:9d:22:38:f0:96:7c:8a:de:1f:bf:1a:e1:3a:8d:15:a0:b3:
         48:b8:2b:8d:5c:93:b0:53:5e:d1:76:5d:56:60:98:9d:6a:ae:
         e3:7b:a9:35:f0:07:60:3c:0d:0a:87:cc:b5:5e:d7:d4:1f:77:
         d3:69:5c:38
```

4. 导入根 CA 证书以将其安装在 CA 上。

> **Note**
>
> 以下示例使用前缀加载`fileb://`所需的输入文件，确保正确 Amazon 私有 CA 解析 Base64-encoded 数据。如果您使用的是低于 1.6.3 的 Amazon CLI 版本，请`file://`改用。

```
$ aws acm-pca import-certificate-authority-certificate \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --certificate fileb://cert.pem
```

检查 CA 的新状态。

```
$ aws acm-pca describe-certificate-authority \
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
 --output json
```

状态现在显示为“活动”。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T12:37:14.235000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
```

```
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

安装由托管的从属 CA 证书 Amazon 私有 CA

您可以使用 Amazon Web Services 管理控制台 为 Amazon 私有 CA 托管的从属 CA 创建和安装证书。

为您创建和安装证书 Amazon 私有 CA 托管的下属 CA

1. （可选）如果您尚未进入 CA 的详细信息页面，请打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。在私有证书颁发机构页面上，选择状态为待处理证书或活动的从属 CA。
2. 选择操作、安装 CA 证书以打开安装从属 CA 证书页面。
3. 在 “安装从属 CA 证书” 页面的 “选择 CA 类型” 下，选择Amazon 私有 CA安装由管理的证书 Amazon 私有 CA。
4. 在选择父 CA 下，从父私有 CA 列表中选择 CA。此列表筛选为显示满足以下条件的 CA：
   - 您拥有使用该 CA 的权限。
   - 该 CA 不会自签名。
   - 该 CA 处于 ACTIVE 状态。
   - CA 模式为 GENERAL_PURPOSE。
5. 在指定从属 CA 证书参数下，指定以下证书参数：
   - 有效期 – 指定 CA 证书的到期日期和时间。
   - 签名算法 – 指定根 CA 颁发新证书时使用的签名算法。选项包括：
     - SHA256 RSA
     - SHA384 RSA
     - SHA512 RSA

- 路径长度 – 从属 CA 在签署新证书时可以添加的信任层数。路径长度为零（默认值）表示只能创建终端实体证书，不能创建 CA 证书。路径长度为一或更多表示从属 CA 可以颁发证书以创建附属于它的其他 CA。
- 模板 ARN – 显示此 CA 证书配置模板的 ARN。如果您更改指定的路径长度，模板也会更改。如果您使用 CLI 颁发证书命令或 API IssueCertificate操作创建证书，则必须手动指定 ARN。有关可用 CA 证书模板的信息，请参阅使用 Amazon 私有 CA 证书模板。

6. 检查您的设置是否正确，然后选择 “确认并安装”。 Amazon 私有 CA 导出 CSR，使用从属 CA 证书模板生成证书，并使用选定的父 CA 签署证书。 Amazon 私有 CA 然后导入已签名的从属 CA 证书。
7. CA 的详细信息页面顶部显示安装状态（成功或失败）。如果安装成功，新完成的从属 CA 将在常规窗格中显示为活动。

安装由外部父 CA 签名的从属 CA 证书

按照中所述创建从属私有 CA 后在中创建私有 CA Amazon 私有 CA，您可以选择通过安装由外部签名机构签名的 CA 证书来激活它。使用外部 CA 签署从属 CA 证书需要您先将外部信任服务提供商设置为签名颁发机构，或者安排使用第三方提供商。

> **Note**
>
> 创建或获取外部信任服务提供商的过程不在本指南的讨论范围内。

在创建从属 CA 并拥有外部签名颁发机构访问权限之后，请完成以下任务：

1. 从获取证书签名请求 (CSR) Amazon 私有 CA。
2. 将 CSR 提交给您的外部签名颁发机构，并获取签名的 CA 证书以及所有链证书。
3. 将 CA 证书和链导入 Amazon 私有 CA 以激活您的从属 CA。

有关详细步骤，请参阅 使用外部签名的私有 CA 证书。

控制对私有 CA 的访问权限

任何对私有 CA 具有必要权限的用户都 Amazon 私有 CA 可以使用该 CA 签署其他证书。CA 所有者可以颁发证书或将颁发证书所需的权限委托给居住在相同的 Amazon Identity and Access Management

(IAM) 用户 Amazon Web Services 账户。如果 CA 所有者通过基于资源的策略授权，居住在不同 Amazon 账户中的用户也可以颁发证书。

授权用户，无论是单账户还是跨账户，都可以在颁发证书时使用 Amazon 私有 CA 或 Amazon Certificate Manager 资源。 Amazon 私有 CA IssueCertificate通过 API 或 issue-certification CL I 命令颁发的证书处于非托管状态。此类证书需要在目标设备上手动安装，并在到期时手动续订。管理从 ACM 控制台、ACM RequestCertificateAPI 或请求证书 CL I 命令颁发的证书。此类证书可以轻松地安装在与 ACM 集成的服务中。如果 CA 管理员允许，并且颁发者的账户拥有 ACM 的服务相关角色，则托管证书将在到期时自动续订。

主题

- 为 IAM 用户创建单账户权限
- 附加跨账户存取策略

为 IAM 用户创建单账户权限

当 CA 管理员（即 CA 的所有者）和证书颁发者居住在同一个 Amazon 账户中时，最佳做法是通过创建一个权限有限的 Amazon Identity and Access Management (IAM) 用户，将颁发者和管理员角色分开。有关将 IAM 与 Amazon 私有 CA权限一起使用的信息以及示例权限，请参阅身份和访问管理 (IAM) Access Management 适用于 Amazon 私有证书颁发机构。

Single-account 案例 1：颁发非托管证书

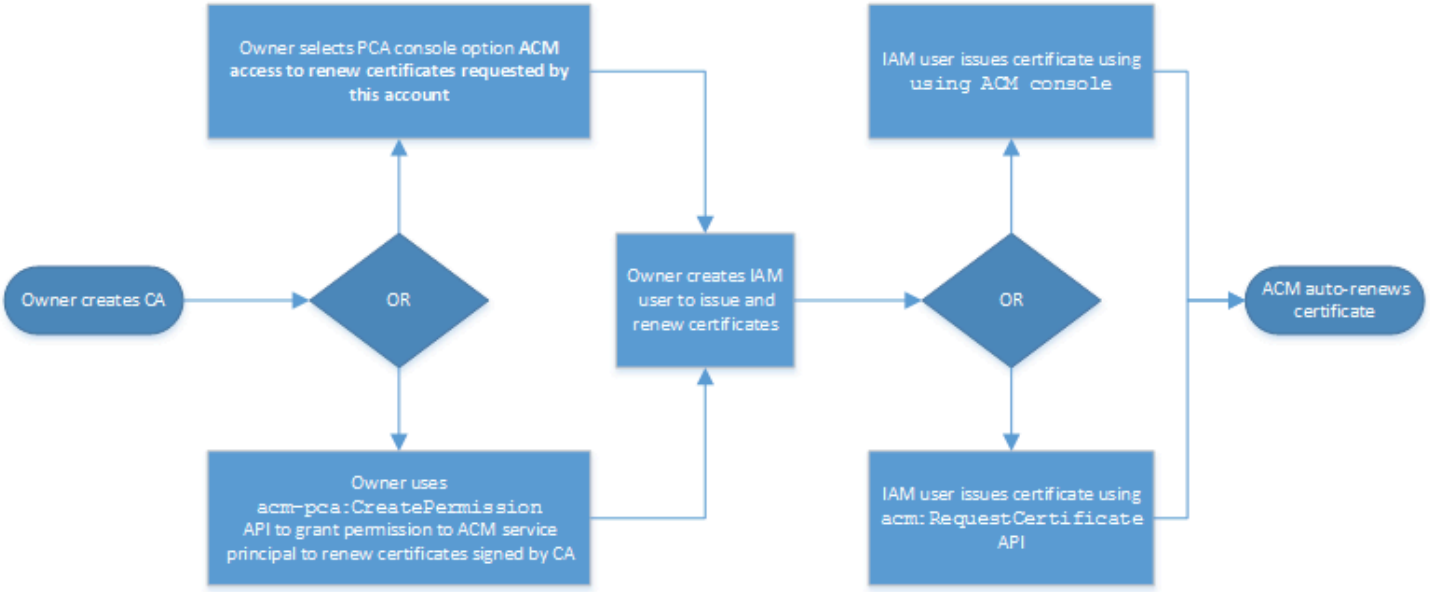
在这种情况下，账户拥有者创建一个私有 CA，然后创建一个具有颁发由私有 CA 签名证书权限的 IAM 用户。IAM 用户通过调用 Amazon 私有 CA `IssueCertificate` API 来颁发证书。

以这种方式颁发的证书未托管，这意味着管理员必须将其导出并安装在要使用的设备上。这些证书过期时还必须手动续订。使用此 API 颁发证书需要证书签名请求 (CSR) 和 Amazon 私有 CA 由 OpenSSL 或类似程序之外生成的密钥对。有关更多信息，请参阅 `IssueCertificate` 文档。

Single-account 案例 2：通过 ACM 颁发托管证书

第二种情况涉及来自 ACM 和 PCA 的 API 操作。账户拥有者像以前一样创建私有 CA 和 IAM 用户。然后，账户拥有者向 ACM 服务主体授予权限以自动续订由此 CA 签名的所有证书。IAM 用户再次颁

发证书，但这次是通过调用处理 CSR 和密钥生成的 ACM `RequestCertificate` API。该证书到期后，ACM 会自动执行续订工作流程。



账户拥有者可以选择在创建 CA 期间或之后或使用 PCA `CreatePermission` API 通过管理控制台授予续订权限。通过此工作流程创建的托管证书可用于与 ACM 集成的 Amazon 服务。

下一节包含授予续订权限的过程。

将证书续订权限分配给 ACM

借助 Amazon Certificate Manager （ACM）中的托管续订，您可以自动执行公有和私有证书的证书续订流程。为了让 ACM 自动续订私有 CA 生成的证书，CA 本身必须向 ACM 服务主体授予所有可能的权限。如果 ACM 不存在这些续订权限，则 CA 的拥有者（或授权代表）必须在每个私有证书过期时手动重新颁发该证书。

> **Important**
>
> 这些分配续订权限的过程仅在 CA 所有者和证书颁发者居住在同一个 Amazon 账户中时适用。有关跨账户场景，请参阅 附加跨账户存取策略。

续订权限可在私有 CA 创建期间委派，并且只要 CA 处于 `ACTIVE` 状态，即可在任何时间更改。

您可以从 Amazon 私有 CA 控制台、Amazon Command Line Interface (Amazon CLI) 或 Amazon 私有 CA API 管理私有 CA 权限：

向 ACM 分配私有 CA 权限（控制台）

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。
2. 在私有证书颁发机构页面上，从列表中选择您的私有 CA。
3. 选择操作、配置 CA 权限。
4. 选择授权 ACM 访问以续订此账户请求的证书。
5. 选择保存。

要在中管理 ACM 权限 Amazon 私有 CA (Amazon CLI)

使用 create-permission 命令将权限分配给 ACM。您必须分配必要的权限（IssueCertificate、GetCertificate 和 ListPermissions）以使 ACM 能自动续订证书。

```
$ aws acm-pca create-permission \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --actions IssueCertificate GetCertificate ListPermissions \
     --principal acm.amazonaws.com
```

使用 list-permissions 命令列出 CA 委派的权限。

```
$ aws acm-pca list-permissions \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID
```

使用 delete-per mission 命令撤消 CA 分配给服务主体的权限。 Amazon

```
$ aws acm-pca delete-permission \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --principal acm.amazonaws.com
```

附加跨账户存取策略

当 CA 管理员和证书颁发者位于不同的 Amazon 账户中时，CA 管理员必须共享 CA 访问权限。这是通过将基于资源的策略附加到 CA 来实现的。该策略向特定的委托人授予签发权限，该委托人可以是 Amazon 账户所有者、IAM 用户、 Amazon Organizations ID 或组织单位 ID。

CA 管理员可以通过以下方式附加和管理策略：

- 在管理控制台中，使用 Amazon Resource Access Manager (RAM)，这是跨账户共享 Amazon 资源的标准方法。当您与其他账户中的 Amazon RAM 委托人共享 CA 资源时，所需的基于资源的策略会自动附加到 CA。有关 RAM 的更多信息，请参阅《Amazon RAM 用户指南》https://docs.amazonaws.cn/ram/latest/userguide/。

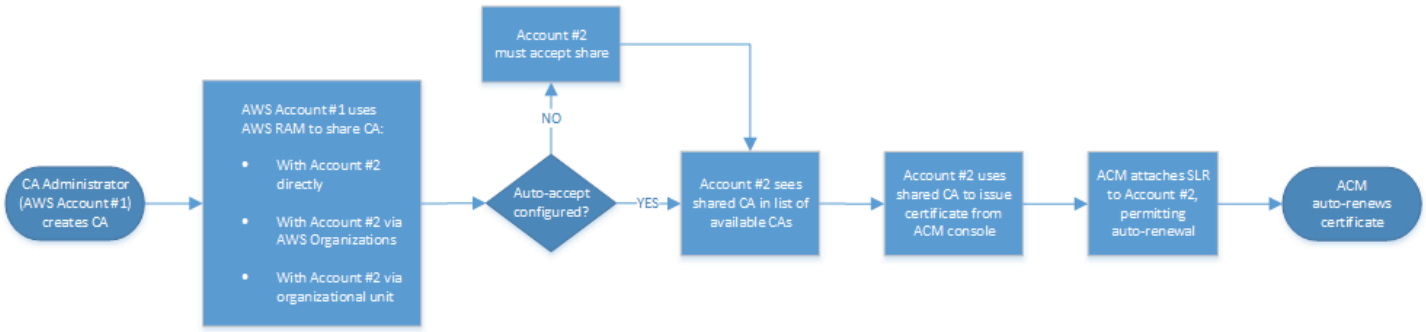
> **Note**
>
> 选择 CA，然后选择操作、管理资源共享，即可轻松打开 RAM 控制台。

- 以编程方式，使用 PCA API PutPolicyGetPolicy、和。DeletePolicy
- 在 Amazon CLI中手动使用 PCA 命令 put-policy、get-policy 和 delete-policy。

只有控制台方法需要 RAM 访问权限。

Cross-account 案例 1：从控制台颁发托管证书

在这种情况下，CA 管理员使用 Amazon Resource Access Manager (Amazon RAM) 与其他 Amazon 账户共享 CA 访问权限，从而允许该账户颁发托管 ACM 证书。该图显示 Amazon RAM 可以直接与账户共享 CA，也可以通过账户所属的 Amazon Organizations ID 间接共享 CA。



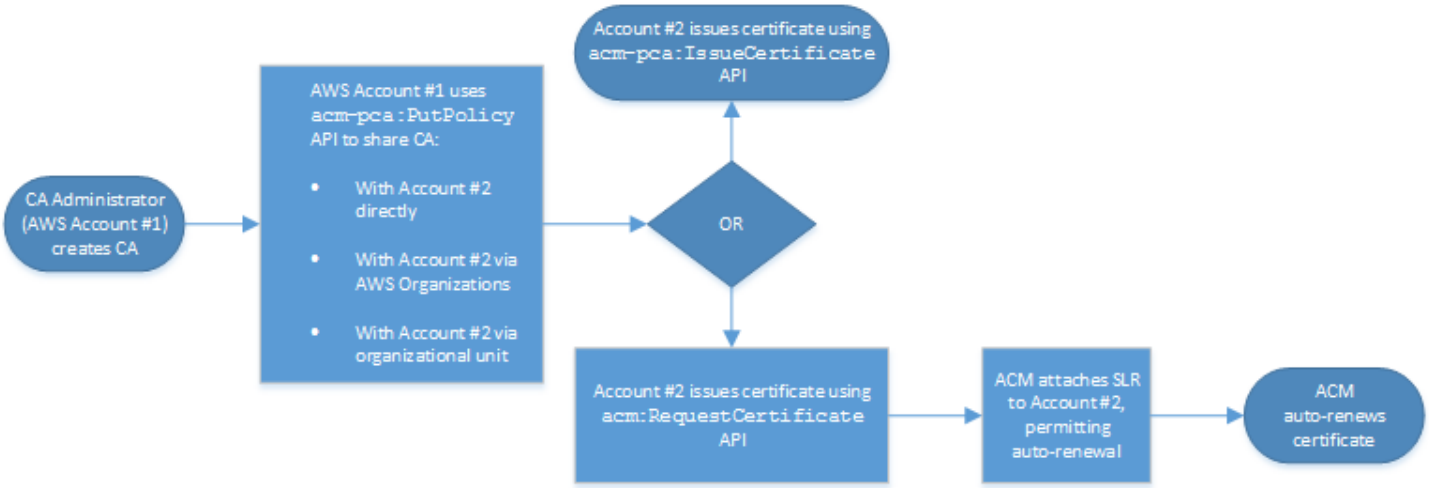
RAM 通过共享资源后 Amazon Organizations，接收方委托人必须接受该资源才能使其生效。收件人可以配置 Amazon Organizations 为自动接受所发行的股票。

> **Note**
>
> 接收方账户负责在 ACM 中配置自动续订。通常，在首次使用共享 CA 时，ACM 会安装一个服务相关角色，允许其对 Amazon 私有 CA进行无人值守的证书调用。如果此操作失败（通常是

由于缺少权限），则该 CA 的证书不会自动续订。只有 ACM 用户可以解决问题，CA 管理员无法解决问题。有关更多信息，请参阅将服务相关角色（SLR）用于 ACM。

Cross-account 案例 2：使用 API 或 CLI 颁发托管和非托管证书

第二个案例演示了使用和 Amazon 私有 CA API 可能的共享 Amazon Certificate Manager 和发行选项。所有这些操作也可以使用相应的 Amazon CLI 命令来执行。



由于在此示例中直接使用了 API 操作，因此证书颁发者可以选择两个 API 操作来颁发证书。PCA API 操作 `IssueCertificate` 会生成非托管证书，该证书不会自动续订，并且必须导出和手动安装。ACM API 操作会RequestCertificate生成托管证书，该证书可以轻松安装在 ACM 集成服务上并自动续订。

> **Note**
>
> 接收方账户负责在 ACM 中配置自动续订。通常，在首次使用共享 CA 时，ACM 会安装一个服务相关角色，允许其对 Amazon 私有 CA进行无人值守的证书调用。如果此操作失败（通常是由于缺少权限），则该 CA 的证书不会自动续订，并且只有 ACM 用户可以解决问题，CA 管理员无法解决问题。有关更多信息，请参阅将服务相关角色（SLR）用于 ACM。

列出私有 CA

您可以使用 Amazon 私有 CA 控制台或 Amazon CLI 列出您拥有或有权访问的私有 CA。

使用控制台列出可用的 CA

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。
2. 查看私有证书颁发机构列表中的信息。您可以使用右上角的页码浏览多页 CA。每个 CA 占据一行，每个 CA 都显示以下部分或全部列：

- 使用者 – CA 的可分辨名称信息摘要。
- ID – CA 的 32 字节十六进制唯一标识符。
- 状态 – CA 状态。可能的值为正在创建、待处理证书、活动、已删除、已禁用、已过期和失败。
- 类型 – CA 的类型。可能的值为根和从属。
- 模式 – CA 的模式。可能的值是 General-purpose（颁发可配置为任何到期日期的证书）和Short-lived证书（颁发最长有效期为七天的证书）。在某些情况下，较短的有效期可以取代吊销机制。默认值为 General-purpose。
- 所有者-拥有 CA 的 Amazon 账户。这可能是您的账户，也可能是向您委派 CA 管理权限的账户。
- 密钥算法 – CA 支持的公有密钥算法。可能的值是 ML_DSA_44、ML_DSA_65、ML_DSA_87、RSA_204 8、RSA_3072、RSA_4096、ec_prime256 v1、ec_secp384r1 和 ec_secp 521r1。
- 签名算法 – CA 签署证书请求所用的算法。（不要与颁发证书时用于签署证书的 `SigningAlgorithm` 参数混淆。）可能的值是 ML_DSA_44、ML_DSA _ 65、ML_DSA_87、SHA256WITHRSA、SHA384WITHRSA、SHA512WITHRSA、SHA256WITHECDSA、和 SHA512WITHECDSA。

> **Note**
>
> 您可以通过选择控制台右上角的设置图标来自定义要显示的列以及其他设置。

要列出可用的 CA，请使用 Amazon CLI

使用 list-certificate-authorities 命令列出可用的 CA，如以下示例中所示：

```
$ aws acm-pca list-certificate-authorities --max-items 10
```

命令返回类似于下文的信息：

```
{
   "CertificateAuthorities":[
      {
         "Arn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
         "CreatedAt":"2022-05-02T11:59:02.022000-07:00",
         "LastStateChangeAt":"2022-05-02T11:59:18.498000-07:00",
         "Type":"ROOT",
         "Serial":"serial_number",
         "Status":"ACTIVE",
         "NotBefore":"2022-05-02T10:59:17-07:00",
         "NotAfter":"2032-05-02T11:59:17-07:00",
         "CertificateAuthorityConfiguration":{
            "KeyAlgorithm":"RSA_2048",
            "SigningAlgorithm":"SHA256WITHRSA",
            "Subject":{
               "Organization":"testing_com"
            }
         },
         "RevocationConfiguration":{
            "CrlConfiguration":{
               "Enabled":false
            }
         }
      }
 ...
   ]
}
```

查看私有 CA

您可以使用 ACM 控制台或查看有关私有 CA 的详细元数据，并根据需要更改其中的几个值。 Amazon CLI 有关更新 CA 的详细信息，请参阅 在中更新私有 CA Amazon 私有证书颁发机构。

在控制台中查看 CA 详细信息

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。
2. 查看私有证书颁发机构列表。您可以使用右上角的页码浏览多页 CA。
3. 要显示所列 CA 的详细元数据，请选择您要检查的 CA 旁边的单选按钮。这将打开一个包含以下选项卡式视图的详细信息窗格：

- 使用者选项卡 – 有关 CA 的可分辨名称的信息。有关更多信息，请参阅 Subject distinguished name。显示的字段包括：
  - 使用者 – 提供的名称信息字段摘要
  - 组织（O）– 例如，公司名称
  - 组织单位（OU）– 例如，公司内部的部门
  - 国家/地区名称（C）– 两个字母的国家/地区代码
  - 州或省名称 – 州或省的全名
  - 所在地名称 – 城市的名称
  - 公用名 (CN) — 用于标识 CA 的人类可读字符串。
- CA 证书选项卡 – 有关 CA 证书有效性的信息
  - 有效期至 – CA 证书在此之前有效的日期和时间
  - 到期时间 – 证书到期前的天数
- 吊销配置选项卡 – 您当前选择的证书吊销选项。选择编辑进行更新。
  - 证书吊销列表（CRL）分配 – 状态为已启用或已禁用
  - 在线证书状态协议（OCSP）– 状态为已启用或已禁用
- 权限选项卡 – 您当前通过 Amazon Certificate Manager （ACM）为此 CA 选择的证书续订权限。选择编辑进行更新。
- ACM 续订授权 – 状态为已授权或未授权
- 标签选项卡 – 您当前为此 CA 分配的可自定义标签。选择管理标签以更新。
- “资源共享” 选项卡 — 您当前通过 Amazon Resource Access Manager (RAM) 为此 CA 分配的资源共享。选择管理资源共享以更新。
  - 名称 – 资源共享的名称
  - 状态 – 资源共享的状态

4. 选择要检查的 CA 的 ID 字段以打开常规窗格。CA 的 32 字节十六进制唯一标识符将在顶部显示。该窗格提供以下附加信息：

- 状态 – CA 状态。可能的值为正在创建、待处理证书、活动、已删除、已禁用、已过期和失败。
- ARN – CA 的 Amazon 资源名称。
- 所有者-拥有 CA 的 Amazon 账户。这可能是您的账户（自己），也可能是向您委派 CA 管理权限的账户。
- CA 类型 – CA 的类型。可能的值为根和从属。

- 创建时间 – 创建 CA 的日期和时间。
- 过期日期 – CA 证书过期的日期和时间。
- 模式 – CA 的模式。可能的值是 General-purpose （可以配置为任何到期日期的证书）和Short-lived证书（最长有效期为七天的证书）。在某些情况下，较短的有效期可以取代吊销机制。默认值为 General-purpose。
- 密钥算法 – CA 支持的公有密钥算法。可能的值是ML-DSA-44、、ML-DSA-65、RSA 2048 ML-DSA-87、RSA 3072、RSA 4096、ECDSA P256、ECDSA P384 和 ECDSA P 521。
- 签名算法 — CA 用来签署自己的证书签名请求、CRL 和 OCSP 响应的算法（不要与 IssueCertificate API 中使用的`SigningAlgorithm`参数混淆。）可能的值是ML-DSA-44、、ML-DSA-65、SHA256 RSA ML-DSA-87、SHA384 RSA 和 SHA512 RSA、SHA256 ECDSA 、SHA384 ECDSA 、SHA512 ECDSA、ECDSA
- 密钥存储安全标准-符合联邦信息处理标准 (FIPS) 的级别。你可以从以下值中进行选择：FIPS 140-2 等级 2 或更高、FIPS 140-2 3 级或更高，以及 CCPC 等级 1 或更高。此参数因 Amazon 地区而异。

> **Note**
>
> 从 2023 年 1 月 26 日起， Amazon 私有证书颁发机构使用符合 FIPS PUB 140-2 Level 3 的硬件安全模块 (HSM) 保护非中国地区的所有 CA 私钥。

要查看和修改 CA 详细信息，请使用 Amazon CLI

使用 Amazon CLI 中的 describe-certificate-authority 命令显示有关 CA 的详细信息，如以下命令所示：

```
$ aws acm-pca describe-certificate-authority --certificate-authority-arn
 arn:aws:acm:region:account:certificate-authority/CA_ID
```

命令返回类似于下文的信息：

```
{
   "CertificateAuthority":{
      "Arn":"arn:aws:acm:region:account:certificate-authority/CA_ID",
      "CreatedAt":"2022-05-02T11:59:02.022000-07:00",
      "LastStateChangeAt":"2022-05-02T11:59:18.498000-07:00",
      "Type":"ROOT",
      "Serial":"serial_number",
```

```
      "Status":"ACTIVE",
      "NotBefore":"2022-05-02T10:59:17-07:00",
      "NotAfter":"2031-05-02T11:59:17-07:00",
      "CertificateAuthorityConfiguration":{
         "KeyAlgorithm":"RSA_2048",
         "SigningAlgorithm":"SHA256WITHRSA",
         "Subject":{
            "Organization":"testing_com"
         }
      },
      "RevocationConfiguration":{
         "CrlConfiguration":{
            "Enabled":false
         }
      }
   }
}
```

有关通过命令行更新私有 CA 的信息，请参阅 更新 CA（CLI）。

为您的私有 CA 添加标签

标签是一些充当元数据的词和短语，用于标识和组织 Amazon 资源。每个标签均包含一个键 和一个值。您可以使用 Amazon 私有 CA 控制台、 Amazon Command Line Interface (Amazon CLI) 或 PCA API 来添加、查看或删除私有 CA 的标签。

您可以随时为私有 CA 添加或删除自定义标签。例如，您可以使用 `Environment=Prod` 或 `Environment=Beta` 等键值对来为私有 CA 添加标签以确定 CA 适合的环境。有关更多信息，请参阅创建私有 CA。

> **Note**
>
> 要在创建过程中将标签附加到私有 CA，CA 管理员必须先将内联 IAM policy 与 `CreateCertificateAuthority` 操作关联并显式允许标记。有关更多信息，请参阅 Tag-on-create: 在创建 CA 时将标签附加到 CA。

其他 Amazon 资源也支持标记。您可以将同一标签分配给不同的资源以指示这些资源是否相关。例如，您可以将标签（如 `Website=example.com`）分配给 CA、Elastic Load Balancing 负载均衡器以及其他相关资源。有关为 Amazon 资源添加标签的更多信息，请参阅 Amazon EC2 用户指南中的为你的 Amazon EC2 资源添加标签。

以下基本限制适用于 Amazon 私有 CA 标签：

- 每个私有 CA 的最大标签数是 50。
- 标签键的最大长度是 128 个字符。
- 标签值的最大长度是 256 个字符。
- 标签键和值可以包含以下字符： A-Z、a-z 和。 :+= @_ %-（连字符）。
- 标签键和值区分大小写。
- 保留 `aws:` 和 `rds:` 前缀以供 Amazon 使用；您无法添加、编辑或删除其键以 `aws:` 或 `rds:` 开头的标签。以 `aws:` 和 `rds:` 开头的默认标签不计入每个资源的标签数配额。
- 如果您计划在多个服务和资源中使用添加标签方案，请记得其他服务可能对允许使用的字符有不同限制。请参阅该服务对应的文档。
- Amazon 私有 CA 标签不可用于中的 Res ource Groups 和标签编辑器中 Amazon Web Services 管理控制台。

您可以为从 Amazon 私有 CA 控制台、Amazon Command Line Interface (Amazon CLI) 或 Amazon 私有 CA API 为私有 CA 添加标签。

标记私有 CA（控制台）

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。
2. 在私有证书颁发机构页面上，从列表中选择您的私有 CA。
3. 在列表下方的详细信息区域中，选择标签选项卡。此时会显示现有标签列表。
4. 选择管理标签。
5. 选择 Add new tag（添加新标签）。
6. 键入键值对。
7. 选择保存。

标记私有 CA (Amazon CLI)

使用 tag-certificate-authority 命令可为 CA 添加标签。

```
$ aws acm-pca tag-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
```

```
     --tags Key=Admin,Value=Alice
```

使用 list-tags 命令可为私有 CA 列出标签。

```
$ aws acm-pca list-tags \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --max-results 10
```

使用 untag-certificate-authority 命令可从私有 CA 中删除标签。

```
$ aws acm-pca untag-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:aregion:account:certificate-
authority/CA_ID \
     --tags Key=Purpose,Value=Website
```

明白 Amazon 私有 CA CA 状态

由 Amazon 私有 CA 用户操作管理的 CA 的状态源于用户操作，或者在某些情况下来自服务操作。例如，CA 状态在过期时会发生变化。对 CA 管理员可用的状态选项取决于 CA 的当前状态。

Amazon 私有 CA 可以报告以下状态值。下表显示每种状态下可用的 CA 功能。

> **Note**
>
> 对于除 DELETED 和 FAILED 之外的所有状态值，您需要支付 CA 的费用。

Status	颁发证书	使用 OCSP 验证证书	生成 CRL	生成审计	您可以更新 CA 证书	可以吊销证书	您需要支付 CA 费用
CREATING – 正在创建 CA。	否	否	否	否	否	否	是
PENDING_CERTIFICATE – CA	否	否	否	否	否	否	是

Status	颁发证书	使用 OCSP 验证证书	生成 CRL	生成审计	您可以更新 CA 证书	可以吊销证书	您需要支付 CA 费用
已创建，需要证书才能正常运行。*							
ACTIVE	是	是	是	是	是	是	是
DISABLED – 您已手动禁用 CA。	否	是	是	是	否	是	是
EXPIRED – CA 证书已过期。**	否	否	否	否	是	否	是
FAILED	CreateCertificateAuthority 操作失败。这可能是由于网络中断、后端 Amazon 故障或其他错误造成的。出现故障的 CA 无法恢复。请删除 CA 并创建新 CA。						否
DELETED	您的 CA 处于还原期内，时长可能为 7-30 天。在此期间之后，它将被永久删除。 • 如果您在状态为 DELETED 且具有过期证书的 CA 上调用 RestoreCertificateAuthority API，则 CA 将设置为 EXPIRED。 • 有关删除 CA 的更多信息，请参阅删除私有 CA。						否

要完成激活，您需要生成 CSR，从 CA 获取签名的 CA 证书，然后将证书导入 Amazon 私有 CA。CSR 可以提交给您的新 CA（用于自签名），也可以提交给本地根或从属 CA。有关更多信息，请参阅 安装 CA 证书。

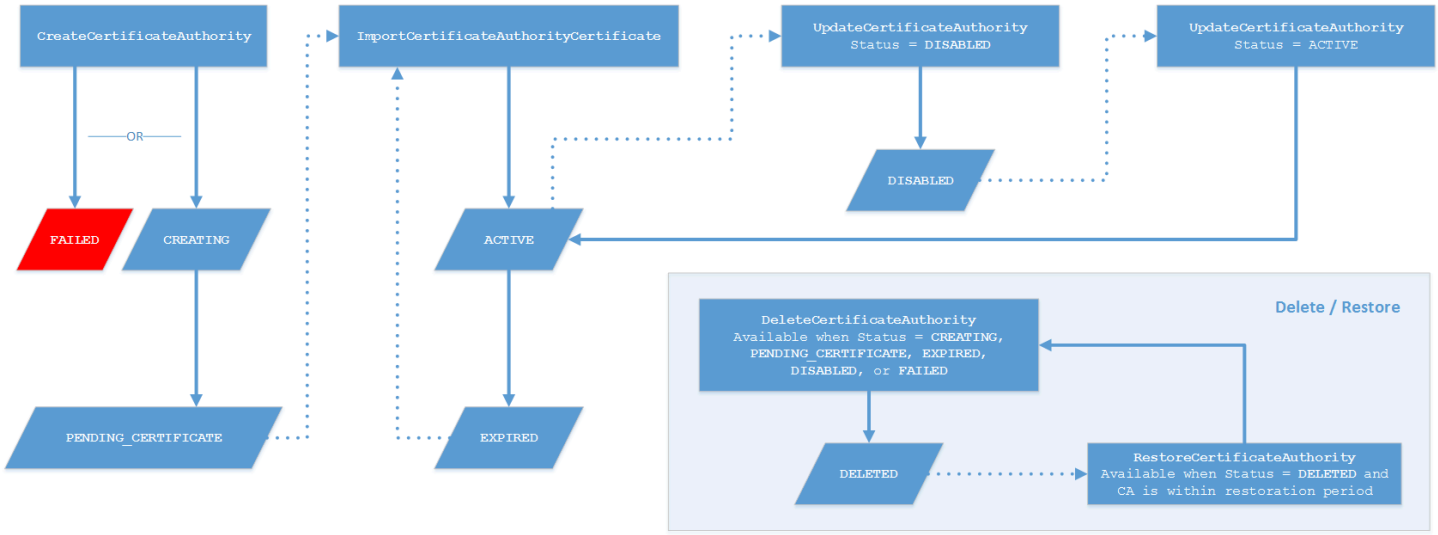
您不能直接更改过期 CA 的状态。如果您为 CA 导入新证书，则ACTIVE除非在证书过期DISABLED之前将其设置为，否则状态会 Amazon 私有 CA 重置为。

有关过期 CA 证书的其他注意事项：

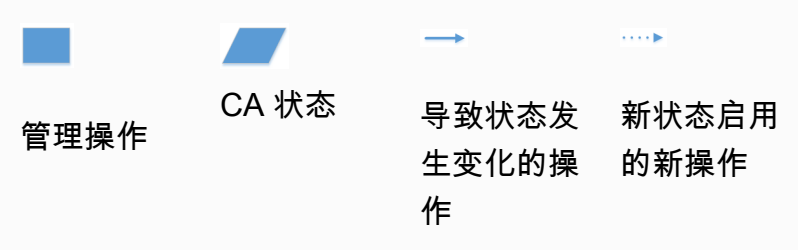
- CA 证书不会自动续订。有关通过自动续订的信息 Amazon Certificate Manager，请参阅将证书续订权限分配给 ACM。
- 如果您尝试使用过期 CA 颁发新证书，`IssueCertificate` API 将返回 `InvalidStateException`。过期的根 CA 必须先自行签名新的根 CA 证书，然后才能颁发新的从属证书。
- 如果 CA 证书已过期，则 The `ListCertificateAuthorities` 和 `DescribeCertificateAuthority` API 返回状态 `EXPIRED`，无论 CA 状态设置为 `ACTIVE` 还是 `DISABLED`。但是，如果已过期 CA 设置为 `DELETED`，则返回状态 `DELETED`。
- `UpdateCertificateAuthority` API 无法更新已过期 CA 的状态。
- `RevokeCertificate` API 无法用于吊销任何已过期证书，包括 CA 证书。

CA 状态与 CA 生命周期之间的关系

下图通过管理操作与 CA 状态的交互说明了 CA 生命周期。



图键



在图的顶部，管理操作通过 Amazon 私有 CA 控制台、CLI 或 API 应用。这些操作将引导 CA 完成创建、激活、过期和续订的过程。CA 状态会随着手动操作或自动更新而变化（如实线所示）。在大多数情况下，新状态会带来 CA 管理员可以应用的新操作（以虚线显示）。右下角的嵌入图显示允许删除和恢复操作的可能状态值。

在中更新私有 CA Amazon 私有证书颁发机构

创建私有 CA 后，您可以更新其状态或更改其吊销配置。本主题提供有关 CA 状态和 CA 生命周期的详细信息，以及 CA 控制台和 CLI 更新的示例。

更新 CA（控制台）

以下过程说明如何使用 Amazon Web Services 管理控制台更新现有 CA 配置。

更新 CA 状态（控制台）

在此示例中，已启用 CA 的状态更改为已禁用。

更新 CA 的状态

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为 https://console.aws.amazon.com/acm-pca/home
2. 在私有证书颁发机构页面上，从列表中选择当前处于活动状态的私有 CA。
3. 在操作菜单上，选择禁用以禁用私有 CA。

更新 CA 的吊销配置（控制台）

您可以更新私有 CA 的吊销配置，例如，通过添加或删除 OCSP 或 CRL 支持，或者通过修改其设置。

> **Note**
>
> 对 CA 吊销配置的更改不会影响已经颁发的证书。要使托管吊销生效，必须重新颁发较旧的证书。

对于 OCSP，您可以更改以下设置：

- 启用或禁用 OCSP。
- 启用或禁用自定义 OCSP 完全限定域名（FQDN）。

- 更改 FQDN。

对于 CRL，您可以更改以下任何设置：

- CRL 类型（完整或分区）
- 私有 CA 是否生成证书吊销列表 (CRL)
- CRL 过期前的天数。请注意， Amazon 私有 CA 开始尝试在您指定的天数的 ½ 时重新生成 CRL。
- 保存了您的 CRL 的 Amazon S3 桶的名称。
- 用于在公共视图中隐藏 Amazon S3 桶名称的别名。

> **Important**
>
> 更改上述任何参数可能具有负面影响。示例包括在将私有 CA 投入生产后禁用 CRL 生成、更改有效期或更改 S3 桶。此类更改可能会破坏依赖于 CRL 和当前 CRL 配置的现有证书。更改别名可以安全地完成，只要旧别名保持链接到正确的存储桶。

更新吊销设置

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。
2. 在私有证书颁发机构页面上，从列表中选择一个私有 CA。这将打开 CA 的详细信息面板。
3. 选择吊销配置选项卡，然后选择编辑。
4. 在证书吊销选项下，显示两个选项：

   - 激活 CRL 分配
   - 打开 OCSP

   您可以为 CA 配置这两个吊销机制中的任一个、两个都不配置或两个都配置。尽管是可选的，但建议将托管吊销作为最佳实践。在完成此步骤之前，请参阅 规划您的 Amazon 私有 CA 证书吊销方法，了解有关每种方法的优点、可能需要的初步设置以及其他吊销功能的信息。

配置 CRL

1. 选择激活 CRL 分配。

2. 要为您的 CRL 条目创建 Amazon S3 桶，请选择创建新的 S3 桶。提供唯一的桶名称。（不需要包括存储桶的路径。）否则，请取消选中此选项，然后从 S3 存储桶名称列表中选择现有桶。

   如果您创建了新的存储桶，则 Amazon 私有 CA 会创建所需的访问策略并将其附加到该存储桶。如果您决定使用现有桶，则必须先为其附加访问策略，然后才能开始生成 CRL。使用 亚马逊 S3 CRLs 中的访问策略 中所述的策略模式之一。有关附加策略的信息，请参阅使用 Amazon S3 控制台添加桶策略。

   > **Note**
   >
   > 使用 Amazon 私有 CA 控制台时，如果满足以下两个条件，则尝试创建 CA 将失败：
   >
   > - 您正在对您的 Amazon S3 桶或账户强制执行阻止公共访问设置。
   > - 您要求 Amazon 私有 CA 自动创建 Amazon S3 存储桶。
   >
   > 在这种情况下，控制台默认会尝试创建可公共访问的桶，而 Amazon S3 会拒绝此操作。如果发生这种情况，请检查您的 Amazon S3 设置。有关更多信息，请参阅阻止对您的 Amazon S3 存储的公共访问。

3. 展开高级以获取其他配置选项。

   - 选择启用分区以启用 CRL 的分区。如果您不启用分区，则您的 CA 将遵守配额上显示的已吊销证书的最大数量。Amazon 私有证书颁发机构有关分区 CRL 的更多信息，请参阅 CRL 类型。
   - 添加自定义 CRL 名称可为 Amazon S3 桶创建别名。此名称包含在由 CA 颁发的证书的“CRL 分配点”扩展中（由 RFC 5280 定义）。要通过 IPv6 使用 CRL，请将其设置为存储桶的双栈 S3 端点，如在 IPv6 上使用 CR L 中所述。
   - 添加自定义路径，为您的 Amazon S3 存储桶中的文件路径创建 DNS 别名。
   - 键入有效期（以天为单位），您的 CRL 将保持有效。默认值为 7 天。对于联机 CRL，有效期通常为 2-7 天。 Amazon 私有 CA 将在指定时间段的中点尝试重新生成 CRL。

4. 完成后，选择保存更改。

配置 OCSP

1. 在证书吊销页面上，选择打开 OCSP。
2. （可选）在自定义 OCSP 端点字段中，为您的 OCSP 端点提供完全限定的域名（FQDN）。要通过 IPv6 使用 OCSP，请将此字段设置为双栈端点，如在 IPv6 上使用 OCS P 中所述。

在此字段中提供 FQDN 时，将 FQDN Amazon 私有 CA 插入到每个已颁发证书的授权信息访问扩展插件中，以代替 Amazon OCSP 响应者的默认 URL。当端点收到包含自定义 FQDN 的证书时，它会查询该地址以获取 OCSP 响应。要使此机制发挥作用，您需要采取另外两个操作：

- 使用代理服务器将到达您的自定义 FQDN 的流量转发给 Amazon OCSP 响应器。
- 将相应的 CNAME 记录添加到您的 DNS 数据库。

> **Tip**
>
> 有关使用自定义 CNAME 实现完整 OCSP 解决方案的更多信息，请参阅 自定义 OCSP 网址 Amazon 私有 CA。

例如，以下是自定义 OCSP 的 CNAME 记录，该记录将在 Amazon Route 53 中显示。

记录名称	Type	路由策略	优势	Value/Route 前往的交通
alternative.example.com	别名记录	简便	-	proxy.example.com

> **Note**
>
> CNAME 的值不得包含协议前缀，例如“http://”或“https://”。

3. 完成后，选择保存更改。

更新 CA（CLI）

以下过程说明如何使用 Amazon CLI更新现有 CA 的状态和吊销配置。

> **Note**
>
> 对 CA 吊销配置的更改不会影响已经颁发的证书。要使托管吊销生效，必须重新颁发较旧的证书。

更新您的私有 CA 的状态 (Amazon CLI)

使用 update-certificate-authority 命令。

当您的现有 CA 状态为 DISABLED 且您希望将其设置为 ACTIVE 时，这非常有用。首先，使用以下命令确认 CA 的初始状态。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

这会产生类似于以下内容的输出。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T13:17:40.221000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "DISABLED",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
```

```
    },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

以下命令将私有 CA 的状态设置为 ACTIVE。仅当在 CA 上安装了有效证书时，才可能实现此目的。

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --status "ACTIVE"
```

检查 CA 的新状态。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

状态现在显示为 ACTIVE。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T13:23:09.352000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
```

```
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

在某些情况下，您的活动 CA 可能没有配置吊销机制。如果要开始使用证书吊销列表（CRL），请按以下过程操作。

向现有 CA 添加 CRL (Amazon CLI)

1. 使用以下命令检查 CA 的当前状态。

```
$ aws acm-pca describe-certificate-authority
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

输出确认 CA 的状态为 ACTIVE 但未配置为使用 CRL。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
```

```
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": false
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

2. 创建并保存一个名为 `revoke_config.txt` 的文件来定义 CRL 配置参数。

```
{
   "CrlConfiguration":{
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
   }
}
```

> **Note**
>
> 更新 Matter 设备认证 CA 以启用 CRL 时，必须将其配置为在已颁发的证书中省略 CDP 扩展，以帮助符合当前 Matter 标准。为此，请定义您的 CRL 配置参数，如下所示：
>
> ```
> {
>    "CrlConfiguration":{
> ```

```
        "Enabled": true,
        "ExpirationInDays": 7,
        "S3BucketName": "amzn-s3-demo-bucket"
        "CrlDistributionPointExtensionConfiguration":{
           "OmitExtension": true
        }
     }
}
```

3. 使用 update-certificate-authority 命令和吊销配置文件更新 CA。

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
      --revocation-configuration file://revoke_config.txt
```

4. 再次检查 CA 的状态。

```
$ aws acm-pca describe-certificate-authority
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

输出确认 CA 现已配置为使用 CRL。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
```

```
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "S3BucketName": "amzn-s3-demo-bucket",
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

在某些情况下，您可能希望添加 OCSP 吊销支持，而不是像前面的过程那样启用 CRL。在这种情况下，请使用以下步骤。

为现有 CA 添加 OCSP 支持 (Amazon CLI)

1. 创建并保存一个名为 `revoke_config.txt` 的文件来定义 OCSP 参数。

```
{
   "OcspConfiguration":{
      "Enabled":true
   }
}
```

2. 使用 update-certificate-authority 命令和吊销配置文件更新 CA。

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
      --revocation-configuration file://revoke_config.txt
```

3. 再次检查 CA 的状态。

```
$ aws acm-pca describe-certificate-authority
```

```
 --certificate-authority-arnarn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

输出确认 CA 现已配置为使用 OCSP。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": false
            },
            "OcspConfiguration": {
                "Enabled": true
            }
        }
    }
}
```

> **Note**
>
> 您也可以在 CA 上同时配置 CRL 和 OCSP 支持。

删除私有 CA

您可以从 Amazon Web Services 管理控制台 或中 Amazon CLI 永久删除私有 CA。您可能想删除一个 CA，例如，用具有新私钥的新 CA 来替换它。要安全删除 CA，请按照下列步骤操作：

1. 创建替换 CA。
2. 一旦新的私有 CA 投入使用，则禁用旧版，但不要立即将其删除。
3. 将旧版 CA 保持禁用状态，直到其颁发的所有证书过期。
4. 删除旧版 CA。

Amazon 私有 CA 在处理删除请求之前，不会检查所有已颁发的证书是否都已过期。您可以生成审核报告来确定哪些证书已过期。当 CA 被禁用时，您可以吊销证书，但不能颁发新证书。

如果您必须在私有 CA 颁发的所有证书过期之前删除该 CA，建议您同时吊销 CA 证书。该 CA 证书将列在父 CA 的 CRL 中，客户端将不信任该私有 CA。

> **Important**
>
> 私有 CA 在处于 PENDING_CERTIFICATE、CREATING、EXPIRED、DISABLED 或 FAILED 状态时可被删除。要删除处于 ACTIVE 状态的 CA，必须先将其禁用，否则删除请求将引发异常。如果您要删除处于 PENDING_CERTIFICATE 或 DISABLED 状态的私有 CA，可将其还原期设置为 7-30 天（默认值为 30 天）。在此期间，状态设置为 DELETED，CA 可恢复。处于 CREATING 或 FAILED 状态时删除的私有 CA 没有分配的还原期，因此无法还原。有关更多信息，请参阅 恢复私有 CA。
>
> 删除私有 CA 后，您无需再为其付费。但是，如果还原已删除的 CA，则需支付删除到还原这个时段内的费用。有关更多信息，请参阅 的定价 Amazon 私有证书颁发机构。

删除私有 CA（控制台）

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。

2. 在私有证书颁发机构页面上，从列表中选择您的私有 CA。
3. 如果您的 CA 处于 ACTIVE 状态，则必须先将其禁用。在 Actions (操作) 菜单上，选择 Disable (禁用)。出现提示时，选择我了解风险，继续。
4. 对于未处于 ACTIVE 状态的 CA，请选择操作、删除。
5. 如果您的 CA 处于 DISABLED、EXPIRED、或 PENDING_CERTIFICATE 状态，通过删除 CA 页面可让您指定 7-30 天的还原期。如果您的私有 CA 未处于其中任一状态，则它稍后将无法还原，删除为永久性。
6. 选择删除。
7. 如果您确定要删除私有 CA，请在系统提示您时，选择 Permanently delete (永久删除)。私有 CA 的状态将更改为 DELETED。不过，在还原期结束前，您可以还原私有 CA。要查看该DELETED州私有 CA 的恢复周期，请调用DescribeCerticateAuthority或 ListCertificateAuthoritiesAPI 操作。

要删除私有 CA (Amazon CLI)

使用 delete-certificate-authority 命令可删除私有 CA。

```
$ aws acm-pca delete-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --permanent-deletion-time-in-days 16
```

恢复私有 CA

对于已被删除的私有 CA，只要该 CA 仍在您指定的还原期内，就可将其还原。还原期为 7–30 天。此期间结束时，私有 CA 将被永久删除。有关更多信息，请参阅 删除私有 CA。已被永久删除的私有 CA 无法还原。

> **Note**
>
> 删除私有 CA 后，您无需再为其付费。但是，如果还原已删除的 CA，则需支付删除到还原这个时段内的费用。有关更多信息，请参阅 的定价 Amazon 私有证书颁发机构。

还原私有 CA（控制台）

您可以使用 Amazon Web Services 管理控制台 恢复私有 CA。

还原私有 CA（控制台）

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。
2. 在私有证书颁发机构页面上，从列表中选择已删除的私有 CA。
3. 在 Actions (操作) 菜单上，选择 Restore (还原)。
4. 在还原 CA 页面上，再次选择恢复。
5. 如果成功，私有 CA 的状态将设置为它被删除前的状态。选择操作、启用，然后再次选择启用，将其状态更改为 ACTIVE。如果私有 CA 在被删除时已处于 PENDING_CERTIFICATE 状态，则必须先向私有 CA 导入 CA 证书，然后才能激活它。

恢复私有 CA (Amazon CLI)

使用 restore-certificate-authority 命令可还原处于 DELETED 状态的已删除的私有 CA。以下步骤讨论依次删除、还原和重新激活私有 CA 所需的整个过程。

删除、恢复和重新激活私有 CA (Amazon CLI)

1. 删除私有 CA。

   运行 delete-certificate-authority 命令可删除私有 CA。如果私有 CA 的状态为 DISABLED 或 PENDING_CERTIFICATE，则可以设置 --permanent-deletion-time-in-days 参数以指定私有 CA 的还原期（7-30 天）。如果未指定还原期，则默认为 30 天。如果成功，此命令会将私有 CA 的状态设置为 DELETED。

   > **Note**
   >
   > 要使私有 CA 可还原，它在被删除时的状态必须为 DISABLED 或 PENDING_CERTIFICATE。

   ```
   $ aws acm-pca delete-certificate-authority \
        --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
        --permanent-deletion-time-in-days 16
   ```

2. 还原私有 CA。

运行 restore-certificate-authority 命令可还原私有 CA。您必须在使用 delete-certificate-authority 命令设置的还原期过期前运行此命令。如果成功，此命令会将私有 CA 的状态设置为它被删除前的状态。

```
$ aws acm-pca restore-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID
```

3. 使私有 CA 处于 ACTIVE 状态。

   运行 update-certificate-authority 命令可将私有 CA 的状态更改为 ACTIVE。

```
$ aws acm-pca update-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --status ACTIVE
```

使用外部签名的私有 CA 证书

如果您的私有 CA 层次结构的信任根必须是外部的 CA Amazon 私有 CA，则可以创建自己的根 CA 并对其进行自签名。或者，您可以获取由组织运营的外部私有 CA 签名的私有 CA 证书。无论其来源如何，您都可以使用此外部获得的 CA 来签署 Amazon 私有 CA 管理的私有从属 CA 证书。

> **Note**
>
> 创建或获取外部信任服务提供商的过程不在本指南的讨论范围内。

使用外部父 CA Amazon 私有 CA 允许您强制执行 RFC 5280 的 “名称约束” 部分中定义的 CA 名称约束。名称约束为 CA 管理员提供了一种限制证书中使用者名称的方法。

如果您计划使用外部 CA 签署私有从属 CA 证书，则在 Amazon 私有 CA中拥有工作 CA 之前需要完成三项任务：

1. 生成证书签名请求（CSR）。
2. 将 CSR 提交给您的外部签名颁发机构，然后返回签名的证书和证书链。
3. 在中安装签名证书 Amazon 私有 CA。

以下过程介绍如何使用 Amazon Web Services 管理控制台 或 Amazon CLI完成这些任务。

获取并安装外部签名的 CA 证书（控制台）

1. （可选）如果您尚未进入 CA 的详细信息页面，请打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。在私有证书颁发机构页面上，选择状态为待处理证书、活动、已禁用或已过期的从属 CA。
2. 选择操作、安装 CA 证书以打开安装从属 CA 证书页面。
3. 在安装从属 CA 证书页面的选择 CA 类型下，选择外部私有 CA。
4. 在此 CA 的 CSR 下，控制台会显示 CSR 的 Base64-encoded ASCII 文本。您可以使用复制按钮复制文本，也可以选择将 CSR 导出到文件并将其保存在本地。

   > **Note**
   >
   > 复制和粘贴时，必须保留 CSR 文本的确切格式。

5. 如果您无法立即执行离线步骤从外部签名颁发机构获取签名证书，可以关闭该页面，并在拥有签名证书和证书链后返回该页面。

   否则，如果您已准备就绪，请执行以下任一操作：

   - 将证书正文和证书链的 Base64-encoded ASCII 文本粘贴到各自的文本框中。
   - 选择上传将证书正文和证书链从本地文件加载到其各自的文本框中。
6. 选择确认并安装。

获取并安装外部签名的 CA 证书（CLI）

1. 使用 get-certificate-authority-csr 命令检索您的私有 CA 的证书签名请求 (CSR)。如果要将 CSR 发送到显示屏上，请使用该`--output text`选项从每行末尾删除 CR/LF 字符。要将 CSR 发送到文件，请使用重定向选项 (>)，后跟文件名。

   ```
   $ aws acm-pca get-certificate-authority-csr \
   --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
   --output text
   ```

   将 CSR 保存为本地文件后，您可以使用以下 OpenSSL 命令对其进行检查：

```
openssl req -in path_to_CSR_file -text -noout
```

此命令将生成类似于以下内容的输出。请注意，CA 扩展为 TRUE，表示 CSR 用于 CA 证书。

```
Certificate Request:
Data:
Version: 0 (0x0)
Subject: O=ExampleCompany, OU=Corporate Office, CN=Example CA 1
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:d4:23:51:b3:dd:01:09:01:0b:4c:59:e4:ea:81:
            1d:7f:48:36:ef:2a:e9:45:82:ec:95:1d:c6:d7:c9:
            7f:19:06:73:c5:cd:63:43:14:eb:c8:03:82:f8:7b:
            c7:89:e6:8d:03:eb:b6:76:58:70:f2:cb:c3:4c:67:
            ea:50:fd:b9:17:84:b8:60:2c:64:9d:2e:d5:7d:da:
            46:56:38:34:a9:0d:57:77:85:f1:6f:b8:ce:73:eb:
            f7:62:a7:8e:e6:35:f5:df:0c:f7:3b:f5:7f:bd:f4:
            38:0b:95:50:2c:be:7d:bf:d9:ad:91:c3:81:29:23:
            b2:5e:a6:83:79:53:f3:06:12:20:7e:a8:fa:18:d6:
            a8:f3:a3:89:a5:a3:6a:76:da:d0:97:e5:13:bc:84:
            a6:5c:d6:54:1a:f0:80:16:dd:4e:79:7b:ff:6d:39:
            b5:67:56:cb:02:6b:14:c3:17:06:0e:7d:fb:d2:7e:
            1c:b8:7d:1d:83:13:59:b2:76:75:5e:d1:e3:23:6d:
            8a:5e:f5:85:ca:d7:e9:a3:f1:9b:42:9f:ed:8a:3c:
            14:4d:1f:fc:95:2b:51:6c:de:8f:ee:02:8c:0c:b6:
            3e:2d:68:e5:f8:86:3f:4f:52:ec:a6:f0:01:c4:7d:
            68:f3:09:ae:b9:97:d6:fc:e4:de:58:58:37:09:9a:
            f6:27
        Exponent: 65537 (0x10001)
Attributes:
Requested Extensions:
    X509v3 Basic Constraints:
        CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
 c5:64:0e:6c:cf:11:03:0b:b7:b8:9e:48:e1:04:45:a0:7f:cc:
 a7:fd:e9:4d:c9:00:26:c5:6e:d0:7e:69:7a:fb:17:1f:f3:5d:
 ac:f3:65:0a:96:5a:47:3c:c1:ee:45:84:46:e3:e6:05:73:0c:
 ce:c9:a0:5e:af:55:bb:89:46:21:92:7b:10:96:92:1b:e6:75:
 de:02:13:2d:98:72:47:bd:b1:13:1a:3d:bb:71:ae:62:86:1a:
 ee:ae:4e:f4:29:2e:d6:fc:70:06:ac:ca:cf:bb:ee:63:68:14:
```

```
8e:b2:8f:e3:8d:e8:8f:e0:33:74:d6:cf:e2:e9:41:ad:b6:47:
f8:2e:7d:0a:82:af:c6:d8:53:c2:88:a0:32:05:09:e0:04:8f:
79:1c:ac:0d:d4:77:8e:a6:b2:5f:07:f8:1b:e3:98:d4:12:3d:
28:32:82:b5:50:92:a4:b2:4c:28:fc:d2:73:75:75:ff:10:33:
2c:c0:67:4b:de:fd:e6:69:1c:a8:bb:e8:31:93:07:35:69:b7:
d6:53:37:53:d5:07:dd:54:35:74:50:50:f9:99:7d:38:b7:b6:
7f:bd:6c:b8:e4:2a:38:e5:04:00:a8:a3:d9:e5:06:38:e0:38:
4c:ca:a9:3c:37:6d:ba:58:38:11:9c:30:08:93:a5:62:00:18:
d1:83:66:40
```

2. 将 CSR 提交给您的外部签名机构，并获取包含 Base64 PEM-encoded 签名证书和证书链的文件。
3. 使用 import-certificate-authority-certificate 命令将私有 CA 证书文件和证书链文件导入到 Amazon 私有 CA中。

```
$ aws acm-pca import-certificate-authority-certificate \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--certificate file://C:\example_ca_cert.pem \
--certificate-chain file://C:\example_ca_cert_chain.pem
```

颁发和管理证书 Amazon 私有 CA

创建并激活私有证书颁发机构 (CA) 并配置其访问权限后，您或您的授权用户可以颁发和管理证书。如果您尚未为 CA 设置 Amazon Identity and Access Management (IAM) 策略，则可以在本指南的 “身份和访问管理” 部分中详细了解如何配置这些策略。有关在单账户和跨账户场景中配置 CA 访问权限的信息，请参阅 控制对私有 CA 的访问权限。

主题

- 颁发私有终端实体证书
- 检索私有证书
- 列出私有证书
- 导出私有证书及其密钥
- 吊销私有证书
- 自动导出续订的证书
- 使用 Amazon 私有 CA 证书模板

颁发私有终端实体证书

有了私有 CA 后，您可以向 Amazon Certificate Manager (ACM) 或申请私有终端实体证书。 Amazon 私有 CA下表对两项服务的功能进行了比较。

能力	ACM	Amazon 私有 CA
颁发终端实体证书	✓（使用 RequestCertificate 或控制台）	✓（使用 IssueCertificate）
与负载均衡器和面向互联网的服务 Amazon 关联	✓	不支持
托管证书续订	✓	通过 ACM 间接支持
控制台支持	✓	不支持
API 支持	✓	✓
CLI 支持	✓	✓

Amazon 私有 CA 创建证书时，它遵循指定证书类型和路径长度的模板。如果未向创建证书的 API 或 CLI 语句提供模板 ARN，则默认情况下会EndEntityCertificate应用 /V1 模板。有关可用证书模板的更多信息，请参阅 使用 Amazon 私有 CA 证书模板。

虽然 ACM 证书是围绕公共信任设计的，但 Amazon 私有 CA 可以满足您的私有 PKI 的需求。因此，您可以使用 Amazon 私有 CA API 和 CLI 以 ACM 不允许的方式配置证书。这些功能包括：

- 创建具有任何使用者名称的证书。
- 使用任何支持的私有密钥算法和密钥长度。
- 使用任何支持的签名算法。
- 指定私有 CA 和私有证书的任何有效期。

使用创建私有 TLS 证书后 Amazon 私有 CA，您可以将其导入 ACM 并与支持的 Amazon 服务一起使用。

> **Note**
>
> 使用以下步骤、使用issue-certificate命令或 IssueCertificateAPI 操作创建的证书不能直接导出以供外部使用 Amazon。但是，您可以使用私有 CA 签署通过 ACM 颁发的证书，并且这些证书可以与其密钥一起导出。有关请求 ACM 证书的更多信息，请参阅《ACM 用户指南》中的请求私有证书和导出私有证书。

颁发标准证书 (Amazon CLI)

您可以使用 Amazon 私有 CA CLI 命令 issue-certificate 或 API 操作IssueCertificate来请求最终实体证书。此命令需要要用于颁发证书的私有 CA 的 Amazon 资源名称 (ARN)。您还必须使用 OpenSSL 之类的程序生成证书签名请求（CSR）。

如果您使用 Amazon 私有 CA API 或 Amazon CLI 颁发私有证书，则证书处于非托管状态，这意味着您无法使用 ACM 控制台、ACM CLI 或 ACM API 来查看或导出证书，也不会自动续订证书。但是，您可以使用 PCA get-certificate 命令来检索证书详细信息，如果您拥有 CA，则可以创建审计报告。

创建证书时的注意事项

- 为满足 RFC 5280 要求，您提供的域名（技术术语为公用名）长度不能超过 64 个八位字节（字符），包括句点。要添加更长的域名，请在“使用者备用名称”字段中指定该名称，该字段支持长度不超过 253 个八位字节的名称。

- 如果您使用的是 1.6.3 或更高 Amazon CLI 版本，请在指定 base64 编码的输入文件`fileb://`时使用前缀，例如。 CSRs这样可以确保正确 Amazon 私有 CA 解析数据。

以下 OpenSSL 命令为证书生成 CSR 和私有密钥：

```
$ openssl req -out csr.pem -new -newkey rsa:2048 -nodes -keyout private-key.pem
```

您可以按如下方式检查 CSR 的内容：

```
$ openssl req -in csr.pem -text -noout
```

生成的输出应与以下简短示例类似：

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, O=Big Org, CN=example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ca:85:f4:3a:b7:5f:e2:66:be:fc:d8:97:65:3d:
                    a4:3d:30:c6:02:0a:9e:1c:ca:bb:15:63:ca:22:81:
                    00:e1:a9:c0:69:64:75:57:56:53:a1:99:ee:e1:cd:
                    ...
                    aa:38:73:ff:3d:b7:00:74:82:8e:4a:5d:da:5f:79:
                    5a:89:52:e7:de:68:95:e0:16:9b:47:2d:57:49:2d:
                    9b:41:53:e2:7f:e1:bd:95:bf:eb:b3:a3:72:d6:a4:
                    d3:63
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         74:18:26:72:33:be:ef:ae:1d:1e:ff:15:e5:28:db:c1:e0:80:
         42:2c:82:5a:34:aa:1a:70:df:fa:4f:19:e2:5a:0e:33:38:af:
         21:aa:14:b4:85:35:9c:dd:73:98:1c:b7:ce:f3:ff:43:aa:11:
         ....
         3c:b2:62:94:ad:94:11:55:c2:43:e0:5f:3b:39:d3:a6:4b:47:
         09:6b:9d:6b:9b:95:15:10:25:be:8b:5c:cc:f1:ff:7b:26:6b:
         fa:81:df:e4:92:e5:3c:e5:7f:0e:d8:d9:6f:c5:a6:67:fb:2b:
         0b:53:e5:22
```

以下命令创建证书。由于未指定模板，因此默认情况下会颁发基本终端实体证书。

```
$ aws acm-pca issue-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
      --csr fileb://csr.pem \
      --signing-algorithm "SHA256WITHRSA" \
      --validity Value=365,Type="DAYS"
```

将返回已颁发证书的 ARN：

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/certificate/certificate_ID"
}
```

> **Note**
>
> Amazon 私有 CA 收到命令后，会立即返回带有序列号的 ARN。issue-certificate但是，证书处理是异步进行的，仍然可能失败。如果发生这种情况，使用新 ARN 的 get-certificate 命令也会失败。

使用 APIPassthrough 模板颁发带有自定义主题名称的证书

在此示例中，颁发的证书包含自定义使用者名称元素。除了提供像中的那样的 CSR 之外颁发标准证书 (Amazon CLI)，您还可以向issue-certificate命令传递两个额外的参数： APIPassthrough 模板的 ARN 和指定自定义属性及其对象标识符的 JSON 配置文件（ ）OIDs。您不能与一起使用`CustomAttributes`。但是，您可以通过标准 OIDs 作为其`StandardAttributes`中的一部分`CustomAttributes`。下表 OIDs 列出了默认主题名称（来自 RFC 4519 和全局 OID 参考数据库的信息）：

使用者名称	缩写	对象 ID
countryName	c	2.5.4.6
commonName	cn	2.5.4.3
dnQualifier [可分辨名称限定符]		2.5.4.46

使用者名称	缩写	对象 ID
generationQualifier		2.5.4.44
givenName		2.5.4.42
initials		2.5.4.43
locality	l	2.5.4.7
organizationName	o	2.5.4.10
organizationalUnitName	ou	2.5.4.11
pseudonym		2.5.4.65
serialNumber		2.5.4.5
st [状态]		2.5.4.8
surname	sn	2.5.4.4
删除实例快照		2.5.4.12
domainComponent	dc	0.9.2342.19200300.100.1.25
userid		0.9.2342.19200300.100.1.1

示例配置文件 `api_passthrough_config.txt` 包含以下代码：

```
{
  "Subject": {
    "CustomAttributes": [
      {
        "ObjectIdentifier": "2.5.4.6",
        "Value": "US"
      },
      {
        "ObjectIdentifier": "1.3.6.1.4.1.37244.1.1",
        "Value": "BCDABCDA12341234"
      },
      {
```

```
        "ObjectIdentifier": "1.3.6.1.4.1.37244.1.5",
        "Value": "CDABCDAB12341234"
      }
    ]
  }
}
```

使用以下命令颁发证书：

```
$ aws acm-pca issue-certificate \
      --validity Type=DAYS,Value=10
      --signing-algorithm "SHA256WITHRSA" \
      --csr fileb://csr.pem \
      --api-passthrough file://api_passthrough_config.txt \
      --template-arn arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V1 \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

将返回已颁发证书的 ARN：

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

按如下方式在本地检索证书：

```
$ aws acm-pca get-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID | \
      jq -r .'Certificate' > cert.pem
```

您可以使用 OpenSSL 检查证书的内容：

```
$ openssl x509 -in cert.pem -text -noout
```

> **Note**
>
> 也可以创建一个私有 CA，将自定义属性传递给它颁发的每个证书。

使用 APIPassthrough 模板颁发带有自定义扩展名的证书

在此示例中，颁发的证书包含自定义扩展。为此，您需要向issue-certificate命令传递三个参数：APIPassthrough 模板的 ARN、指定自定义扩展的 JSON 配置文件，以及如中所示的 CSR。颁发标准证书 (Amazon CLI)

示例配置文件 `api_passthrough_config.txt` 包含以下代码：

```
{
  "Extensions": {
    "CustomExtensions": [
      {
        "ObjectIdentifier": "2.5.29.30",
        "Value": "MBWgEzARgg8ucGVybWl0dGVkLnRlc3Q=",
        "Critical": true
      }
    ]
  }
}
```

自定义证书的颁发方式如下：

```
$ aws acm-pca issue-certificate \
      --validity Type=DAYS,Value=10
      --signing-algorithm "SHA256WITHRSA" \
      --csr fileb://csr.pem \
      --api-passthrough file://api_passthrough_config.txt \
      --template-arn arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1
 \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

将返回已颁发证书的 ARN：

```
{
```

```
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

按如下方式在本地检索证书：

```
$ aws acm-pca get-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID | \
      jq -r .'Certificate' > cert.pem
```

您可以使用 OpenSSL 检查证书的内容：

```
$ openssl x509 -in cert.pem -text -noout
```

检索私有证书

您可以使用 Amazon 私有 CA API 和 Amazon CLI 来颁发私有证书。如果这样做，则可以使用 Amazon CLI 或 Amazon 私有 CA API 来检索该证书。如果您使用 ACM 创建私有 CA 并请求证书，则必须使用 ACM 导出证书和已加密的私有密钥。有关更多信息，请参阅导出私有证书。

检索终端实体证书

使用 get-certi Amazon CLI ficate 命令检索私有终端实体证书。您也可以使用 GetCertificateAPI 操作。建议使用类似 sed 的解析器 jq 设置输出格式。

> **Note**
>
> 如果要吊销证书，可以使用 get-certificate 命令检索十六进制格式的序列号。您还可以创建审核报告来检索十六进制序列号。有关更多信息，请参阅 将审计报告与您的私有 CA 一起使用。

```
$ aws acm-pca get-certificate \
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 | \
```

```
      jq -r '.Certificate, .CertificateChain'
```

此命令按以下标准格式输出证书和证书链。

```
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
```

检索 CA 证书

您可以使用 Amazon 私有 CA API 和 Amazon CLI 来检索私有 CA 的证书颁发机构 (CA) 证书。运行 get-certificate-authority-certificate 命令。您还可调用 GetCertificateAuthorityCertificate 操作。建议使用类似 sed 的解析器 jq 设置输出格式。

```
$ aws acm-pca get-certificate-authority-certificate \
     --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
     | jq -r '.Certificate'
```

此命令按以下标准格式输出 CA 证书。

```
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
```

列出私有证书

要列出您的私有证书，请生成审计报告，从其 S3 桶中检索该报告，然后根据需要解析报告内容。有关创建 Amazon 私有 CA 审计报告的信息，请参阅 将审计报告与您的私有 CA 一起使用。有关从 S3 桶检索对象的信息，请参阅《Amazon Simple Storage Service 用户指南》中的下载对象。

以下示例说明了创建审计报告并对其进行解析以获取有用数据的方法。结果采用 JSON 格式，使用类似 sed 的解析器 jq 筛选数据。

1. 创建审计报告。

以下命令为指定 CA 生成审计报告。

```
$ aws acm-pca create-certificate-authority-audit-report \
     --region region \
     --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
     --s3-bucket-name bucket_name \
     --audit-report-response-format JSON
```

如果成功，则该命令将返回新审计报告的 ID 和位置。

```
{
   "AuditReportId":"audit_report_ID",
   "S3Key":"audit-report/CA_ID/audit_report_ID.json"
}
```

2. 检索审计报告并设置其格式。

此命令检索审计报告，在标准输出中显示其内容，并筛选结果以仅显示 2020-12-01 当天或之后颁发的证书。

```
$ aws s3api get-object \
     --region region \
     --bucket bucket_name \
     --key audit-report/CA_ID/audit_report_ID.json \
     /dev/stdout | jq '.[] | select(.issuedAt >= "2020-12-01")'
```

返回的项目如下所示：

```
{
   "awsAccountId":"account",
   "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial":"serial_number",
   "subject":"CN=pca.alpha.root2.leaf5",
   "notBefore":"2020-12-21T21:28:09+0000",
   "notAfter":"9999-12-31T23:59:59+0000",
   "issuedAt":"2020-12-21T22:28:09+0000",
   "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
```

```
}
```

3. 在本地保存审计报告。

如果要执行多个查询，可以方便地将审计报告保存到本地文件中。

```
$ aws s3api get-object \
     --region region \
     --bucket bucket_name \
     --key audit-report/CA_ID/audit_report_ID.json > my_local_audit_report.json
```

与以前相同的筛选条件将产生相同的输出：

```
$ cat my_local_audit_report.json | jq '.[] | select(.issuedAt >= "2020-12-01")'
{
   "awsAccountId":"account",
   "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial":"serial_number",
   "subject":"CN=pca.alpha.root2.leaf5",
   "notBefore":"2020-12-21T21:28:09+0000",
   "notAfter":"9999-12-31T23:59:59+0000",
   "issuedAt":"2020-12-21T22:28:09+0000",
   "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

4. 在某个日期范围内查询

您可以查询在某个日期范围内颁发的证书，如下所示：

```
$ cat my_local_audit_report.json | jq '.[] | select(.issuedAt >= "2020-11-01"
 and .issuedAt <= "2020-11-10")'
```

筛选后的内容在标准输出中显示：

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf1",
   "notBefore": "2020-11-06T19:18:21+0000",
```

```
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:18:22+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.rsa2048sha256",
   "notBefore": "2020-11-06T19:15:46+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:15:46+0000",
   "templateArn": "arn:aws:acm-pca:::template/RootCACertificate/V1"
}
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf2",
   "notBefore": "2020-11-06T20:04:39+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T21:04:39+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

5. 按照指定模板搜索证书。

以下命令使用模板 ARN 筛选报告内容：

```
$ cat my_local_audit_report.json | jq '.[] | select(.templateArn == "arn:aws:acm-
pca:::template/RootCACertificate/V1")'
```

输出显示匹配的证书记录：

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.rsa2048sha256",
   "notBefore": "2020-11-06T19:15:46+0000",
```

```
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:15:46+0000",
   "templateArn": "arn:aws:acm-pca:::template/RootCACertificate/V1"
}
```

6. 筛选已吊销的证书

要找出所有已吊销的证书，请使用以下命令：

```
$ cat my_local_audit_report.json | jq '.[] | select(.revokedAt != null)'
```

已吊销的证书显示如下：

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf2",
   "notBefore": "2020-11-06T20:04:39+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T21:04:39+0000",
   "revokedAt": "2021-05-27T18:57:32+0000",
   "revocationReason": "UNSPECIFIED",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

7. 使用正则表达式进行筛选。

以下命令搜索包含字符串“leaf”的使用者名称：

```
$ cat my_local_audit_report.json | jq '.[] | select(.subject|test("leaf"))'
```

匹配的证书记录返回如下：

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.roo2.leaf4",
   "notBefore": "2020-11-16T18:17:10+0000",
```

```
    "notAfter": "9999-12-31T23:59:59+0000",
    "issuedAt": "2020-11-16T19:17:12+0000",
    "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
{
    "awsAccountId": "account",
    "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
    "serial": "serial_number",
    "subject": "CN=pca.alpha.root2.leaf5",
    "notBefore": "2020-12-21T21:28:09+0000",
    "notAfter": "9999-12-31T23:59:59+0000",
    "issuedAt": "2020-12-21T22:28:09+0000",
    "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
{
    "awsAccountId": "account",
    "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
    "serial": "serial_number",
    "subject": "CN=pca.alpha.root2.leaf1",
    "notBefore": "2020-11-06T19:18:21+0000",
    "notAfter": "9999-12-31T23:59:59+0000",
    "issuedAt": "2020-11-06T20:18:22+0000",
    "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

导出私有证书及其密钥

Amazon 私有 CA 无法直接导出其已签署并颁发的私有证书。但是，您可以使用导 Amazon Certificate Manager 出此类证书及其加密密钥。然后该证书便可完全移植，以部署在您的私有 PKI 中的任意位置。有关更多信息，请参阅《 Amazon Certificate Manager 用户指南》中的导出私有证书。

作为一项额外好处， Amazon Certificate Manager 可以为使用 ACM 控制台、ACM API 的`RequestCertificate`操作或的 ACM 部分中的request-certificate命令颁发的私有证书提供托管续订。 Amazon CLI有关续订的更多信息，请参阅在私有 PKI 中续订证书。

吊销私有证书

您可以使用 revoke -c Amazon 私有 CA ertificate Amazon CLI 命令或 API 操作吊销证书。RevokeCertificate例如，如果证书的密钥泄露或其关联的域失效，则可能需要在证书的预定到期之

前将其吊销。为了使吊销生效，使用证书的客户端在尝试建立安全的网络连接时需要一种方法来检查吊销状态。

Amazon 私有 CA 提供了两种完全托管的机制来支持吊销状态检查：在线证书状态协议 (OCSP) 和证书吊销列表 ()。CRLs使用 OCSP，客户端可以查询实时返回状态的权威吊销数据库。使用 CRL，客户端根据其定期下载和存储的已吊销证书列表检查证书。客户端拒绝接受已吊销的证书。

OCSP 和都 CRLs 依赖于证书中嵌入的验证信息。因此，在颁发之前，必须将颁发 CA 配置为支持其中一种或两种机制。有关通过选择和实施托管撤销的信息 Amazon 私有 CA，请参阅规划您的 Amazon 私有 CA 证书吊销方法。

已撤销的证书始终记录在 Amazon 私有 CA 审计报告中。

> **Note**
>
> 对于跨账户来电者，撤销权限不包括在中。`AWSRAMDefaultPermissionCertificateAuthority`要允许跨账户发卡机构撤销，CA 管理员可以使用以下任一方法：
>
> - 客户托管权限（推荐）— 创建 RAM 客户托管权限，其中包括单个资源共享中的操作和其他所需操作。`acm-pca:RevokeCertificate`有关更多信息，请参阅 RAM 中的客户托管权限。
> - Amazon 托管权限 — 创建两个 RAM 共享，均指向同一 CA：
>
> 1. 具有 `AWSRAMRevokeCertificateCertificateAuthority` 权限的共享。
> 2. 具有 `AWSRAMDefaultPermissionCertificateAuthority` 权限的共享。

吊销证书

使用 RevokeCertificateAPI 操作或吊销证书命令吊销私有 PKI 证书。序列号必须使用十六进制格式。您可以通过调用 get-certificate 命令来检索序列号。`revoke-certificate` 命令不返回响应。

```
$ aws acm-pca revoke-certificate \
     --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
     --certificate-serial serial_number \
     --revocation-reason "KEY_COMPROMISE"
```

已吊销的证书和 OCSP

当您吊销证书时，OCSP 响应最多可能需要 60 分钟才能反映新状态。通常，OCSP 倾向于支持更快地分发撤销信息，因为与客户端 CRLs 可以缓存数天的撤销信息不同，OCSP 响应通常不会被客户端缓存。

CRL 中的已吊销证书

通常在吊销证书大约 30 分钟后更新 CRL。如果 CRL 更新因任何原因失败， Amazon 私有 CA 则每 15 分钟再尝试一次。

借助 Amazon CloudWatch，您可以为指标创建警报，`CRLGenerated`以及`MisconfiguredCRLBucket`。有关更多信息，请参阅支持的 CloudWatch指标。有关创建和配置的更多信息 CRLs，请参阅为以下各项设置 CRL Amazon 私有 CA。

以下示例显示证书吊销列表 (CRL) 中的已吊销证书。

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: /C=US/ST=WA/L=Seattle/O=Examples LLC/OU=Corporate Office/
CN=www.example.com
        Last Update: Jan 10 19:28:47 2018 GMT
        Next Update: Jan  8 20:28:47 2028 GMT
        CRL extensions:
            X509v3 Authority key identifier:
                keyid:3B:F0:04:6B:51:54:1F:C9:AE:4A:C0:2F:11:E6:13:85:D8:84:74:67

            X509v3 CRL Number:
                1515616127629
Revoked Certificates:
    Serial Number: B17B6F9AE9309C51D5573BCA78764C23
        Revocation Date: Jan  9 17:19:17 2018 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Key Compromise
    Signature Algorithm: sha256WithRSAEncryption
         21:2f:86:46:6e:0a:9c:0d:85:f6:b6:b6:db:50:ce:32:d4:76:
         99:3e:df:ec:6f:c7:3b:7e:a3:6b:66:a7:b2:83:e8:3b:53:42:
         f0:7a:bc:ba:0f:81:4d:9b:71:ee:14:c3:db:ad:a0:91:c4:9f:
         98:f1:4a:69:9a:3f:e3:61:36:cf:93:0a:1b:7d:f7:8d:53:1f:
         2e:f8:bd:3c:7d:72:91:4c:36:38:06:bf:f9:c7:d1:47:6e:8e:
```

```
        54:eb:87:02:33:14:10:7f:b2:81:65:a1:62:f5:fb:e1:79:d5:
        1d:4c:0e:95:0d:84:31:f8:5d:59:5d:f9:2b:6f:e4:e6:60:8b:
        58:7d:b2:a9:70:fd:72:4f:e7:5b:e4:06:fc:e7:23:e7:08:28:
        f7:06:09:2a:a1:73:31:ec:1c:32:f8:dc:03:ea:33:a8:8e:d9:
        d4:78:c1:90:4c:08:ca:ba:ec:55:c3:00:f4:2e:03:b2:dd:8a:
        43:13:fd:c8:31:c9:cd:8d:b3:5e:06:c6:cc:15:41:12:5d:51:
        a2:84:61:16:a0:cf:f5:38:10:da:a5:3b:69:7f:9c:b0:aa:29:
        5f:fc:42:68:b8:fb:88:19:af:d9:ef:76:19:db:24:1f:eb:87:
        65:b2:05:44:86:21:e0:b4:11:5c:db:f6:a2:f9:7c:a6:16:85:
        0e:81:b2:76
```

审核报告中的已吊销证书

包括已吊销证书在内的所有证书都包含在私有 CA 的审核报告中。以下示例显示包含一个已颁发证书和一个已吊销证书的审核报告。有关更多信息，请参阅 将审计报告与您的私有 CA 一起使用。

```
[
   {
      "awsAccountId":"account",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"serial_number",

 "Subject":"1.2.840.113549.1.9.1=#161173616c6573406578616d706c652e636f6d,CN=www.example1.com,OU
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2018-02-26T18:39:57+0000",
      "notAfter":"2019-02-26T19:39:57+0000",
      "issuedAt":"2018-02-26T19:39:58+0000",
      "revokedAt":"2018-02-26T20:00:36+0000",
      "revocationReason":"KEY_COMPROMISE"
   },
   {
      "awsAccountId":"account",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"serial_number",

 "Subject":"1.2.840.113549.1.9.1=#161970726f64407777772e70616c6f75736573616c65732e636f6d,CN=www
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2018-01-22T20:10:49+0000",
      "notAfter":"2019-01-17T21:10:49+0000",
      "issuedAt":"2018-01-22T21:10:49+0000"
   }
```

```
]
```

自动导出续订的证书

使用 Amazon 私有 CA 创建 CA 时，可以将该 CA 导入 Amazon Certificate Manager 并让 ACM 管理证书的颁发和续订。如果正在续订的证书与集成服务相关联，则该服务将无缝应用新证书。但是，如果证书最初是为了在 PKI 环境中的其他地方（例如本地服务器或设备）使用而导出的，则需要在续订后再次将其导出。

有关使用 Amazon 和 EventBridge Lambda Amazon 自动执行 ACM 导出流程的示例解决方案，请参阅自动导出续订的证书。

使用 Amazon 私有 CA 证书模板

Amazon 私有 CA 使用配置模板颁发 CA 证书和终端实体证书。从 PCA 控制台颁发 CA 证书时，会自动应用相应的根或从属 CA 证书模板。

如果使用 CLI 或 API 颁发证书，则可以提供模板 ARN 作为 `IssueCertificate` 操作的参数。如果您未提供 ARN，则默认应用 `EndEntityCertificate/V1` 模板。有关更多信息，请参阅 IssueCertificateAPI 和颁发证书命令文档。

> **Note**
>
> Amazon Certificate Manager (ACM) 对私有 CA 具有跨账户共享访问权限的用户可以颁发由 CA 签署的托管证书。当您授予`IssueCertificate`操作权限时，可以通过向策略中添加`acm-pca:TemplateArn`条件来限制用于证书颁发的证书模板。
> 有关更多信息，请参阅 Resource-based 政策。

主题

- Amazon 私有 CA 模板品种
- Amazon 私有 CA 模板操作顺序
- Amazon 私有 CA 模板定义

Amazon 私有 CA 模板品种

Amazon 私有 CA 支持四种模板。

- 基础模板

  不允许使用传递参数的预定义模板。

- CSRPassthrough 模板

  通过允许 CSR 传递来扩展其相应基础模板版本的模板。用于颁发证书的 CSR 中的扩展将复制到颁发的证书中。如果 CSR 包含与模板定义冲突的扩展值，则模板定义将始终具有更高的优先级。有关优先级的详细信息，请参阅 Amazon 私有 CA 模板操作顺序。

- APIPassthrough 模板

  通过允许 API 传递来扩展其相应基础模板版本的模板。管理员或其他中间系统已知的动态值可能对请求证书的实体未知，可能无法在模板中定义，也可能在 CSR 中不可用。但是，CA 管理员可以从其他数据来源（例如 Active Directory）检索其他信息来完成请求。例如，如果一台计算机不知道自己属于哪个组织单位，则管理员可以在 Active Directory 中查找信息，然后通过在 JSON 结构中包含该信息来将其添加到证书请求中。

  `IssueCertificate` 操作 的 `ApiPassthrough` 参数中的值将复制到颁发的证书中。如果 `ApiPassthrough` 参数包含与模板定义冲突的信息，则模板定义将始终具有更高的优先级。有关优先级的详细信息，请参阅 Amazon 私有 CA 模板操作顺序。

- APICSRPassthrough 模板

  通过允许 API 和 CSR 传递来扩展其相应基础模板版本的模板。用于颁发证书的 CSR 中的扩展将复制到颁发的证书中，且 `IssueCertificate` 操作的 `ApiPassthrough` 参数中的值也将复制过来。如果模板定义、API 传递值和 CSR 传递扩展存在冲突，则模板定义的优先级最高，其次是 API 传递值，最后是 CSR 传递扩展。有关优先级的详细信息，请参阅 Amazon 私有 CA 模板操作顺序。

下表列出了支持的所有模板类型，并 Amazon 私有 CA 附有指向其定义的链接。

> **Note**
>
> 有关 GovCloud 区域模板 ARNs 的信息，请参阅Amazon GovCloud (US) 用户指南Amazon 私有证书颁发机构中的。

基础模板

模板名称	模板 ARN	证书类型
CodeSigningCertificate/V1	arn:aws:acm-pca:::template/CodeSigningCertificate/V1	代码签名
EndEntityCertificate/V1	arn:aws:acm-pca:::template/EndEntityCertificate/V1	终端实体
EndEntityClientAuthCertificate/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate/V1	终端实体
EndEntityServerAuthCertificate/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate/V1	终端实体
OCSPSigning证书/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate/V1	OCSP 签名
root CACertificate /V1	arn:aws:acm-pca:::template/RootCACertificate/V1	CA
下属 CACertificate _ PathLen 0/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0/V1	CA
下属 CACertificate _ PathLen 1/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1/V1	CA

模板名称	模板 ARN	证书类型
下属 CACertificate _ PathLen 2/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2/V1	CA
下属 CACertificate _ PathLen 3/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3/V1	CA

CSRPassthrough 模板

模板名称	模板 ARN	证书类型
BlankEndEntityCertificate_ CSRPassthrough /V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CSRPassthrough/V1	终端实体
BlankEndEntityCertificate_ CriticalBasicConstraints _ CSRPassthrough /V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1	终端实体
BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertifica	CA

模板名称	模板 ARN	证书类型
	te_PathLen1_CSRPas sthrough/V1	
BlankSubordinateCACertifica te_PathLen2_CSRPassthrough/V1	arn:aws:acm-pca::: template/BlankSubo rdinateCACertifica te_PathLen2_CSRPas sthrough/V1	CA
BlankSubordinateCACertifica te_PathLen3_CSRPassthrough/V1	arn:aws:acm-pca::: template/BlankSubo rdinateCACertifica te_PathLen3_CSRPas sthrough/V1	CA
CodeSigningCertificate_ CSRPassthrough /V1	arn:aws:acm-pca::: template/CodeSigni ngCertificate_CSRP assthrough/V1	代码签名
EndEntityCertificate_ CSRPassth rough /V1	arn:aws:acm-pca::: template/EndEntity Certificate_CSRPas sthrough/V1	终端实体
EndEntityClientAuthCertificate_ CSRPassthrough /V1	arn:aws:acm-pca::: template/EndEntity ClientAuthCertific ate_CSRPassthrough/V1	终端实体
EndEntityServerAuthCertificate_ CSRPassthrough /V1	arn:aws:acm-pca::: template/EndEntity ServerAuthCertific ate_CSRPassthrough/V1	终端实体

模板名称	模板 ARN	证书类型
OCSPSigning证书_ /V1 CSRPassthrough	arn:aws:acm-pca:::template/OCSPSigningCertificate_CSRPassthrough/V1	OCSP 签名
下属 CACertificate _ PathLen 0_ / V1 CSRPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_CSRPassthrough/V1	CA
下属 CACertificate _ PathLen 1_ / V1 CSRPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_CSRPassthrough/V1	CA
下属 CACertificate _ PathLen 2_ / V1 CSRPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_CSRPassthrough/V1	CA
下属 CACertificate _ PathLen 3_ / V1 CSRPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_CSRPassthrough/V1	CA

APIPassthrough 模板

模板名称	模板 ARN	证书类型
BlankEndEntityCertificate_ APIPassthrough /V1	arn:aws:acm-pca:::template/BlankEndE	终端实体

模板名称	模板 ARN	证书类型
	ntityCertificate_APIPassthrough/V1	
BlankEndEntityCertificate_CriticalBasicConstraints _APIPassthrough /V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1	终端实体
CodeSigningCertificate_APIPassthrough /V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APIPassthrough/V1	代码签名
EndEntityCertificate_ APIPassthrough /V1	arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1	终端实体
EndEntityClientAuthCertificate_APIPassthrough /V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate_APIPassthrough/V1	终端实体
EndEntityServerAuthCertificate_APIPassthrough /V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_APIPassthrough/V1	终端实体
OCSPSigning证书_ /V1 APIPassthrough	arn:aws:acm-pca:::template/OCSPSigningCertificate_APIPassthrough/V1	OCSP 签名

模板名称	模板 ARN	证书类型
root CACertificate _ APIPassthrough /V1	arn:aws:acm-pca:::template/RootCACertificate_APIPassthrough/V1	CA
BlankRootCACertificate_ APIPassthrough /V1	arn:aws:acm-pca:::template/BlankRootCACertificate_APIPassthrough/V1	CA
BlankRootCACertificate_ PathLen0_ /V1 APIPassthrough	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen0_APIPassthrough/V1	CA
BlankRootCACertificate_ PathLen1_ /V1 APIPassthrough	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen1_APIPassthrough/V1	CA
BlankRootCACertificate_ PathLen2_ /V1 APIPassthrough	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen2_APIPassthrough/V1	CA
BlankRootCACertificate_ PathLen3_ /V1 APIPassthrough	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen3_APIPassthrough/V1	CA
下属 CACertificate _ PathLen 0_ / V1 APIPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_APIPassthrough/V1	CA

模板名称	模板 ARN	证书类型
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	CA
下属 CACertificate _ PathLen 1_ / V1 APIPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1	CA
下属 CACertificate _ PathLen 2_ / V1 APIPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1	CA
下属 CACertificate _ PathLen 3_ / V1 APIPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_APIPassthrough/V1	CA

模板名称	模板 ARN	证书类型
BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	CA

APICSRPassthrough 模板

模板名称	模板 ARN	证书类型
BlankEndEntityCertificate_APICSRPassthrough /V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_APICSRPassthrough/V1	终端实体
BlankEndEntityCertificate_CriticalBasicConstraints _APICSRPassthrough /V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1	终端实体
CodeSigningCertificate_APICSRPassthrough /V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APICSRPassthrough/V1	代码签名
EndEntityCertificate_ APICSRPassthrough /V1	arn:aws:acm-pca:::template/EndEntityCertificate_APICSRPassthrough/V1	终端实体
EndEntityClientAuthCertificate_APICSRPassthrough /V1	arn:aws:acm-pca:::template/EndEntity	终端实体

模板名称	模板 ARN	证书类型
	ClientAuthCertificate_APICSRPassthrough/V1	
EndEntityServerAuthCertificate_APICSRPassthrough /V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_APICSRPassthrough/V1	终端实体
OCSPSigning证书_ /V1 APICSRPassthrough	arn:aws:acm-pca:::template/OCSPSigningCertificate_APICSRPassthrough/V1	OCSP 签名
下属 CACertificate _ PathLen 0_ / V1 APICSRPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1	CA
下属 CACertificate _ PathLen 1_ / V1 APICSRPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_APICSRPassthrough/V1	CA

模板名称	模板 ARN	证书类型
BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	CA
下属 CACertificate _ PathLen 2_APICSRPassthrough/PathLen3_V1 APIPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1	CA
下属 CACertificate _ PathLen 3_ / V1 APICSRPassthrough	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1	CA

Amazon 私有 CA 模板操作顺序

颁发的证书中包含的信息可能来自四个来源：模板定义、API 传递、CSR 传递和 CA 配置。

只有在使用 API 传递或 APICSR 传递模板时，才会重视 API 传递值。只有在使用 CSRPassthrough 或 APICSR 直通模板时，才会尊重 CSR 直通。当这些信息来源发生冲突时，通常适用一般规则：对于每个扩展值，模板定义的优先级最高，其次是 API 传递值，最后是 CSR 传递扩展。

示例

1. EndEntityClientAuthCertificate_ 的模板定义使用值APIPassthrough为 “TLS Web 服务器身份验证、TLS Web 客户端身份验证” 的 ExtendedKeyUsage 扩展名。如果在 CSR 或`IssueCertificateApiPassthrough`参数中定义， ExtendedKeyUsage 则 ExtendedKeyUsage 将忽略的`ApiPassthrough`值，因为模板定义具有优先级；值的 CSR ExtendedKeyUsage 值将被忽略，因为该模板不是 CSR 直通变体。

   > **Note**
   >
   > 尽管如此，模板定义还是复制了 CSR 中的其他值，例如使用者和使用者备用名称。尽管模板并非 CSR 传递种类，但这些值仍取自 CSR，因为模板定义始终具有最高优先级。

2. EndEntityClientAuthCertificate_ 的模板APICSRPassthrough定义将主题备用名称 (SAN) 扩展定义为从 API 或 CSR 中复制。如果在 CSR 中定义了 SAN 扩展并在 `IssueCertificate ApiPassthrough` 参数中提供，则 API 传递值将优先，因为 API 传递值优先于 CSR 传递值。

Amazon 私有 CA 模板定义

以下各节提供了有关支持的 Amazon 私有 CA 证书模板的配置详细信息。

BlankEndEntityCertificate_ APIPassthrough /V1 的定义

使用空白的终端实体证书模板，您可以颁发仅存在 X.509 基本约束的终端实体证书。这是 Amazon 私有 CA 可以颁发的最简单的最简单的最终实体证书，但可以使用 API 结构对其进行自定义。基本约束扩展定义该证书是否为 CA 证书。空白的终端实体证书模板将基本约束的值强制设置为 FALSE，以确保颁发的是终端实体证书，而不是 CA 证书。

您可以使用空白的直通模板来颁发需要密钥用法 (KU) 和扩展密钥用法 (EKU) 特定值的智能卡证书。例如，扩展密钥用法可能需要“客户端身份验证”和“智能卡登录”，而密钥用法可能需要“数字签名”、“不可否认”和“密钥加密”。与其他直通模板不同，空白的终端实体证书模板允许配置 KU 和 EKU 扩展，其中 KU 可以是九个支持的值（DigitalSignature、NonRepudiation、KeyenCipherment、DataEncipherMent、KeyEncipherMent、、cRLSign、encipherOnly 和 DecipherOnly），而 EKU 可以是任何支持的值

（ServerAuth、ClientAuth、codesigning keyCertSign、EmailProtection），Eku 可以是任何支持的值（ServerAuth、Client、时间戳和）以及自定义扩展程序。 OCSPSigning

BlankEndEntityCertificate_ APIPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankEndEntityCertificate_ APICSRPassthrough /V1 的定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankEndEntityCertificate_ APICSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankEndEntityCertificate_ CriticalBasicConstraints _ APICSRPassthrough /V1 的定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankEndEntityCertificate_ CriticalBasicConstraints _ APICSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置、API 或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankEndEntityCertificate_ CriticalBasicConstraints _ APIPassthrough /V1 的定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankEndEntityCertificate_ CriticalBasicConstraints _ APIPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 API 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankEndEntityCertificate_ CriticalBasicConstraints _ CSRPassthrough /V1 的定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankEndEntityCertificate_ CriticalBasicConstraints _ CSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankEndEntityCertificate_ CSRPassthrough /V1 的定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankEndEntityCertificate_ CSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 0
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 0
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]

X509v3 参数	值
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、`CA:TRUE、pathlen: 0`
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置传递]

BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、`CA:TRUE、pathlen: 1`
授权密钥标识符	[来自 CA 证书的 SKI]

X509v3 参数	值
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、`CA:TRUE`、`pathlen: 1`
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]

X509v3 参数	值
基本约束	Critical、CA:TRUE、pathlen: 1
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 2
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 2
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 2
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 3
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 3
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1定义

有关空白模板的一般信息，请参阅 BlankEndEntityCertificate_ APIPassthrough /V1 的定义。

BlankSubordinateCACertificate_PathLen3_APICSRPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 3
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

CodeSigningCertificate/V1 定义

使用此模板可以创建用于代码签名的证书。您可以将来自的代码签名证书 Amazon 私有 CA 与任何基于私有 CA 基础架构的代码签名解决方案一起使用。例如，使用代码签名的客户 Amazon IoT 可以使用生成代码签名证书 Amazon 私有 CA 并将其导入到。 Amazon Certificate Manager有关更多信息，请参阅代码签名的用途 Amazon IoT？ 以及获取并导入代码签名证书。

CodeSigningCertificate/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]

X509v3 参数	值
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature
扩展密钥用法	Critical、code signing
CRL 分发点*	[从 CA 配置传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

CodeSigningCertificate_ APICSRPassthrough /V1 的定义

此模板扩展了 CodeSigningCertificate /V1 以支持 API 和 CSR 直通值。

CodeSigningCertificate_ APICSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	`CA:FALSE`
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature
扩展密钥用法	Critical、code signing
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

CodeSigningCertificate_ APIPassthrough /V1 的定义

此模板与CodeSigningCertificate模板相同，但有一个区别：在此模板中，如果模板中未指定扩展名，则通过API将其他扩展 Amazon 私有 CA 传递给证书。此模板中指定的扩展始终覆盖 API 中的扩展。

CodeSigningCertificate_ APIPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature
扩展密钥用法	Critical、code signing
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

CodeSigningCertificate_ CSRPassthrough /V1 的定义

此模板与CodeSigningCertificate模板相同，但有一个区别：在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

CodeSigningCertificate_ CSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]

X509v3 参数	值
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature
扩展密钥用法	Critical、code signing
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityCertificate/V1 定义

此模板用于为终端实体（如操作系统或 Web 服务器）创建证书。

EndEntityCertificate/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 服务器身份验证、TLS Web 客户端身份验证
CRL 分发点*	[从 CA 配置传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityCertificate_ APICSRPassthrough /V1 的定义

此模板扩展了 EndEntityCertificate /V1 以支持 API 和 CSR 直通值。

EndEntityCertificate_ APICSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 服务器身份验证、TLS Web 客户端身份验证
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityCertificate_ APIPassthrough /V1 的定义

此模板与`EndEntityCertificate`模板相同，但有一个区别：在此模板中，如果模板中未指定扩展名，则通过API将其他扩展 Amazon 私有 CA 传递给证书。此模板中指定的扩展始终覆盖 API 中的扩展。

EndEntityCertificate_ APIPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]

X509v3 参数	值
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 服务器身份验证、TLS Web 客户端身份验证
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityCertificate_ CSRPassthrough /V1 的定义

此模板与`EndEntityCertificate`模板相同，但有一个区别：在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

EndEntityCertificate_ CSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 服务器身份验证、TLS Web 客户端身份验证

X509v3 参数	值
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityClientAuthCertificate/V1 定义

此模板与 `EndEntityCertificate` 仅在扩展密钥用法值上不同，此模板将值限制为 TLS Web 客户端身份验证。

EndEntityClientAuthCertificate/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 客户端身份验证
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityClientAuthCertificate_ APICSRPassthrough /V1 的定义

此模板扩展了 EndEntityClientAuthCertificate /V1 以支持 API 和 CSR 直通值。

EndEntityClientAuthCertificate_ APICSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 客户端身份验证
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityClientAuthCertificate_ APIPassthrough /V1 的定义

此模板与 `EndEntityClientAuthCertificate` 模板相同，但有一点区别。在此模板中，如果模板中未指定扩展名，则通过 API 将其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 API 中的扩展。

EndEntityClientAuthCertificate_ APIPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]

X509v3 参数	值
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 客户端身份验证
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityClientAuthCertificate_ CSRPassthrough /V1 的定义

此模板与 `EndEntityClientAuthCertificate` 模板相同，但有一点区别。在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

EndEntityClientAuthCertificate_ CSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 客户端身份验证
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityServerAuthCertificate/V1 定义

此模板与 EndEntityCertificate 仅在扩展密钥用法值上不同，此模板将值限制为 TLS Web 服务器身份验证。

EndEntityServerAuthCertificate/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 服务器身份验证
CRL 分发点*	[从 CA 配置传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityServerAuthCertificate_ APICSRPassthrough /V1 的定义

此模板扩展了 EndEntityServerAuthCertificate /V1 以支持 API 和 CSR 直通值。

EndEntityServerAuthCertificate_ APICSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]

X509v3 参数	值
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 服务器身份验证
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityServerAuthCertificate_ APIPassthrough /V1 的定义

此模板与 `EndEntityServerAuthCertificate` 模板相同，但有一点区别。在此模板中，如果模板中未指定扩展名，则通过 API 将其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 API 中的扩展。

EndEntityServerAuthCertificate_ APIPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 服务器身份验证
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

EndEntityServerAuthCertificate_ CSRPassthrough /V1 的定义

此模板与 EndEntityServerAuthCertificate 模板相同，但有一点区别。在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

EndEntityServerAuthCertificate_ CSRPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、key encipherment
扩展密钥用法	TLS Web 服务器身份验证
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

OCSPSigning证书/V1 定义

使用此模板可以创建用于 OCSP 响应签名的证书。此模板与 CodeSigningCertificate 模板相同，只是扩展密钥用法值指定 OCSP 签名而不是代码签名。

OCSPSigning证书/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	CA:FALSE

X509v3 参数	值
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature
扩展密钥用法	Critical、OCSP signing
CRL 分发点*	[从 CA 配置传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

OCSPSigning证书_ /V1 的定义 APICSRPassthrough

此模板扩展了 OCSPSigning证书/V1 以支持 API 和 CSR 直通值。

OCSPSigning证书_ /V1 APICSRPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	`CA:FALSE`
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature
扩展密钥用法	Critical、OCSP signing
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

OCSPSigning证书_ /V1 的定义 APIPassthrough

此模板与 `OCSPSigningCertificate` 模板相同，但有一点区别。在此模板中，如果模板中未指定扩展名，则通过 API 将其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 API 中的扩展。

OCSPSigning证书_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	`CA:FALSE`
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature
扩展密钥用法	Critical、OCSP signing
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

OCSPSigning证书_ /V1 的定义 CSRPassthrough

此模板与 `OCSPSigningCertificate` 模板相同，但有一点区别。在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

OCSPSigning证书_ /V1 CSRPassthrough

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]

X509v3 参数	值
基本约束	CA:FALSE
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature
扩展密钥用法	Critical、OCSP signing
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

根 CACertificate /V1 定义

此模板用于颁发自签名根 CA 证书。CA 证书包括一个关键的基本约束扩展，该扩展中的 CA 字段设置为 TRUE 以指定证书可用于颁发 CA 证书。模板未指定路径长度 (pathLenConstraint)，因为这可能会阻碍层次结构的未来扩展。排除扩展密钥用法，以防止将 CA 证书用作 TLS 客户端或服务器证书。未指定 CRL 信息，因为无法吊销自签名证书。

root CACertificate /V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE
使用者密钥标识符	[派生自 CSR]
密钥用法	关键签名、数字签名 keyCertSign、CRL 签名
CRL 分发点	不适用

根 CACertificate _ APIPassthrough /V1 的定义

此模板扩展了 Root CACertificate /V1 以支持 API 直通值。

root CACertificate _ APIPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE
授权密钥标识符	[从 API 传递]
使用者密钥标识符	[派生自 CSR]
密钥用法	关键签名、数字签名 keyCertSign、CRL 签名
CRL 分发点*	不适用

BlankRootCACertificate_ APIPassthrough /V1 的定义

如果根证书模板为空，则可以在仅存在 X.509 基本限制的情况下颁发根证书。这是 Amazon 私有 CA 可以颁发的最简单的根证书，但可以使用 API 结构对其进行自定义。基本约束扩展定义证书是否为 CA 证书。为确保颁发根 CA 证书，空白TRUE的根证书模板会强制使用基本约束的值。

您可以使用空白的直通根模板来颁发需要特定密钥用法 (KU) 值的根证书。例如，密钥的使用可能需要keyCertSign和cRLSign，但不需要digitalSignature。与其他非空白根直通证书模板不同，空白根证书模板允许配置 KU 扩展，其中 KU 可以是九个支持的值（digitalSignature、、、、、、nonRepudiation、keyEncipherment、dataEnciphermentkeyA和decipherOnly）中的任何一个。

BlankRootCACertificate_ APIPassthrough /V1

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]

X509v3 参数	值
基本约束	Critical、CA:TRUE
使用者密钥标识符	[派生自 CSR]

BlankRootCACertificate_ PathLen 0_ APIPassthrough /V1 的定义

有关空白根 CA 模板的一般信息，请参阅???。

BlankRootCACertificate_ PathLen 0_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 0
使用者密钥标识符	[派生自 CSR]

BlankRootCACertificate_ PathLen 1_ APIPassthrough /V1 的定义

有关空白根 CA 模板的一般信息，请参阅???。

BlankRootCACertificate_ PathLen 1_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 1
使用者密钥标识符	[派生自 CSR]

BlankRootCACertificate_ PathLen 2_ APIPassthrough /V1 的定义

有关空白根 CA 模板的一般信息，请参阅???。

BlankRootCACertificate_ PathLen 2_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 2
使用者密钥标识符	[派生自 CSR]

BlankRootCACertificate_ PathLen 3_ APIPassthrough /V1 的定义

有关空白根 CA 模板的一般信息，请参阅???。

BlankRootCACertificate_ PathLen 3_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 3
使用者密钥标识符	[派生自 CSR]

下属 CACertificate _ PathLen 0/V1 的定义

此模板用于颁发路径长度为的从属 CA 证书0。CA 证书包括一个关键的基本约束扩展，该扩展中的 CA 字段设置为 TRUE 以指定证书可用于颁发 CA 证书。不包括扩展密钥用法，以防止 CA 证书用作 TLS 客户端或服务器证书。

有关认证路径的更多信息，请参阅设置认证路径的长度约束。

下属 CACertificate _ PathLen 0/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、`CA:TRUE`、`pathlen: 0`
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、`keyCertSi gn` 、CRL sign
CRL 分发点*	[从 CA 配置传递]

*仅当 CA 配置为启用 CRL 生成时，才会将 CRL 分发点包含在使用此模板颁发的证书中。

下属 CACertificate _ PathLen 0_ APICSRPassthrough /V1 定义

此模板扩展了从属关系 CACertificate _ PathLen 0/V1 以支持 API 和 CSR 直通值。

下属 CACertificate _ PathLen 0_ /V1 APICSRPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、`CA:TRUE`、`pathlen: 0`
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、`keyCertSi gn` 、CRL sign

X509v3 参数	值
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

下属 CACertificate _ PathLen 0_ APIPassthrough /V1 定义

此模板扩展了从属关系 CACertificate _ PathLen 0/V1 以支持 API 直通值。

下属 CACertificate _ PathLen 0_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、`CA:TRUE`、`pathlen: 0`
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、`keyCertSi gn` 、CRL sign
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

下属 CACertificate _ PathLen 0_ CSRPassthrough /V1 定义

此模板与`SubordinateCACertificate_PathLen0`模板相同，但有一个区别：在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

> **Note**
>
> 包含自定义附加扩展的 CSR 必须在 Amazon 私有 CA外部创建。

下属 CACertificate _ PathLen 0_ /V1 CSRPassthrough

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 0
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在使用此模板颁发的证书中。

下属 CACertificate _ PathLen 1/V1 的定义

此模板用于颁发路径长度为的从属 CA 证书1。CA 证书包括一个关键的基本约束扩展，该扩展中的 CA 字段设置为 TRUE 以指定证书可用于颁发 CA 证书。不包括扩展密钥用法，以防止 CA 证书用作 TLS 客户端或服务器证书。

有关认证路径的更多信息，请参阅设置认证路径的长度约束。

下属 CACertificate _ PathLen 1/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 1
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]

X509v3 参数	值
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在使用此模板颁发的证书中。

下属 CACertificate _ PathLen 1_ APICSRPassthrough /V1 定义

此模板扩展了从属关系 CACertificate _ PathLen 1/V1 以支持 API 和 CSR 直通值。

下属 CACertificate _ PathLen 1_ /V1 APICSRPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 1
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

下属 CACertificate _ PathLen 1_ APIPassthrough /V1 定义

此模板扩展了从属关系 CACertificate _ PathLen 0/V1 以支持 API 直通值。

下属 CACertificate _ PathLen 1_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 1
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

下属 CACertificate _ PathLen 1_ CSRPassthrough /V1 定义

此模板与SubordinateCACertificate_PathLen1模板相同，但有一个区别：在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

> **Note**
>
> 包含自定义附加扩展的 CSR 必须在 Amazon 私有 CA外部创建。

下属 CACertificate _ PathLen 1_ /V1 CSRPassthrough

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 1

X509v3 参数	值
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在使用此模板颁发的证书中。

下属 CACertificate _ PathLen 2/V1 的定义

此模板用于颁发路径长度为 2 的从属 CA 证书。CA 证书包括一个关键的基本约束扩展，该扩展中的 CA 字段设置为 TRUE 以指定证书可用于颁发 CA 证书。不包括扩展密钥用法，以防止 CA 证书用作 TLS 客户端或服务器证书。

有关认证路径的更多信息，请参阅设置认证路径的长度约束。

下属 CACertificate _ PathLen 2/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 2
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在使用此模板颁发的证书中。

下属 CACertificate _ PathLen 2_ APICSRPassthrough /V1 定义

此模板扩展了从属关系 CACertificate _ PathLen 2/V1 以支持 API 和 CSR 直通值。

下属 CACertificate _ PathLen 2_ /V1 APICSRPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 2
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

下属 CACertificate _ PathLen 2_ APIPassthrough /V1 定义

此模板扩展了从属关系 CACertificate _ PathLen 2/V1 以支持 API 直通值。

下属 CACertificate _ PathLen 2_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 2
授权密钥标识符	[来自 CA 证书的 SKI]

X509v3 参数	值
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

下属 CACertificate _ PathLen 2_ CSRPassthrough /V1 定义

此模板与SubordinateCACertificate_PathLen2模板相同，但有一个区别：在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

> **Note**
>
> 包含自定义附加扩展的 CSR 必须在 Amazon 私有 CA外部创建。

下属 CACertificate _ PathLen 2_ /V1 CSRPassthrough

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 2
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在使用此模板颁发的证书中。

下属 CACertificate _ PathLen 3/V1 的定义

此模板用于颁发路径长度为 3 的从属 CA 证书。CA 证书包括一个关键的基本约束扩展，该扩展中的 CA 字段设置为 TRUE 以指定证书可用于颁发 CA 证书。不包括扩展密钥用法，以防止 CA 证书用作 TLS 客户端或服务器证书。

有关认证路径的更多信息，请参阅设置认证路径的长度约束。

下属 CACertificate _ PathLen 3/V1

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 3
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在使用此模板颁发的证书中。

下属 CACertificate _ PathLen 3_ APICSRPassthrough /V1 定义

此模板扩展了从属关系 CACertificate _ PathLen 3/V1 以支持 API 和 CSR 直通值。

下属 CACertificate _ PathLen 3_ /V1 APICSRPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]

X509v3 参数	值
基本约束	Critical、CA:TRUE、pathlen: 3
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置或 CSR 传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

下属 CACertificate _ PathLen 3_ APIPassthrough /V1 定义

此模板扩展了从属关系 CACertificate _ PathLen 3/V1 以支持 API 直通值。

下属 CACertificate _ PathLen 3_ /V1 APIPassthrough

X509v3 参数	值
使用者备用名称	[从 API 或 CSR 传递]
主题	[从 API 或 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 3
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置传递]

* 只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在模板中。

下属 CACertificate _ PathLen 3_ CSRPassthrough /V1 定义

此模板与SubordinateCACertificate_PathLen3模板相同，但有一个区别：在此模板中，如果未在模板中指定扩展名，则将证书签名请求 (CSR) 中的其他扩展 Amazon 私有 CA 传递到证书中。此模板中指定的扩展始终覆盖 CSR 中的扩展。

> **Note**
>
> 包含自定义附加扩展的 CSR 必须在 Amazon 私有 CA外部创建。

下属 CACertificate _ PathLen 3_ /V1 CSRPassthrough

X509v3 参数	值
使用者备用名称	[从 CSR 传递]
主题	[从 CSR 传递]
基本约束	Critical、CA:TRUE、pathlen: 3
授权密钥标识符	[来自 CA 证书的 SKI]
使用者密钥标识符	[派生自 CSR]
密钥用法	Critical、digital signature、keyCertSign 、CRL sign
CRL 分发点*	[从 CA 配置或 CSR 传递]

*只有在配置 CA 时启用了 CRL 生成，CRL 分发点才会包含在使用此模板颁发的证书中。

安全性 Amazon 私有证书颁发机构

云安全 Amazon 是重中之重。作为 Amazon 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 Amazon 的责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — Amazon 负责保护在 Amazon 云中运行 Amazon 服务的基础架构。 Amazon 还为您提供可以安全使用的服务。 Third-party作为的一部分，审计师定期测试和验证我们安全的有效性。要了解适用于的合规计划 Amazon 私有证书颁发机构，请参阅 “按合规计划划分划分的范围”。
- 云端安全-您的责任由您使用的 Amazon 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档可帮助您了解在使用时如何应用分担责任模型 Amazon 私有 CA。以下主题向您介绍如何进行配置 Amazon 私有 CA 以满足您的安全和合规性目标。您还将学习如何使用其他方法 Amazon Web Services 服务 来帮助您监控和保护您的 Amazon 私有 CA 资源。

主题

- 身份和访问管理 (IAM) Access Management 适用于 Amazon 私有证书颁发机构
- Cross-account 访问私有 CA 的安全最佳实践
- 中的数据保护 Amazon 私有证书颁发机构
- 合规性验证 Amazon 私有证书颁发机构
- 中的基础设施安全 Amazon 私有证书颁发机构
- Amazon 私有证书颁发机构 客户 CP/CPS 框架

身份和访问管理 (IAM) Access Management 适用于 Amazon 私有证书颁发机构

访问 Amazon 私有 CA 需要 Amazon 可用于对您的请求进行身份验证的证书。以下主题提供详细信息来说明如何使用 Amazon Identity and Access Management (IAM) 控制谁能访问您的 Private Certificate Authority (CA)，从而帮助保护它们。

在中 Amazon 私有 CA，您使用的主要资源是证书颁发机构 (CA)。您拥有或控制的每个私有 CA 均由 Amazon 资源名称 (ARN) 来标识，形式如下。

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

资源所有者是创建 Amazon 资源的 Amazon 账户的委托人实体。以下示例说明了它的工作原理。

- 如果您使用您的 Amazon Web Services 账户根用户 证书创建私有 CA，则您的 Amazon 账户拥有该 CA。

  > **Important**
  > - 我们不建议使用 Amazon Web Services 账户根用户 来创建 CA。
  > - 我们强烈建议您在访问时使用多因素身份验证 (MFA)。 Amazon 私有 CA

- 如果您在 Amazon 账户中创建 IAM 用户，则可以向该用户授予创建私有 CA 的权限。但是，该用户所属的账户拥有该 CA。
- 如果您在 Amazon 账户中创建 IAM 角色并授予其创建私有 CA 的权限，则任何能够担任该角色的人都可以创建 CA。但是，该角色所属的账户拥有该私有 CA。

权限策略规定谁可以访问哪些内容。以下讨论介绍创建权限策略时的可用选项。

> **Note**
> 本文档讨论了在的上下文中使用 IAM Amazon 私有 CA。这里不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅 IAM 用户指南。有关 IAM policy 语法和说明的信息，请参阅 Amazon IAM Policy 参考。

Amazon 私有 CA API 操作和权限

在设置您计划附加到 IAM 身份的访问控制和权限策略（基于身份的策略）时，可将下表作为参考。表中的第一列列出了每个 Amazon 私有 CA API 操作。您可以在策略的 `Action` 元素中指定操作。剩余的列将提供额外的信息。

Amazon 私有 CA API 操作	所需的权限	资源
CreateCertificateAuthority	`acm-pca:CreateCertificateAuthority`	`arn:aws:acm-pca: us-east-1 :111122223`

Amazon 私有 CA API 操作	所需的权限	资源
	acm-pca:TagCertificateAuthority （仅在创建带有标签的 CA 时才需要。）	*333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
CreateCertificateAuthorityAuditReport	acm-pca:CreateCertificateAuthorityAuditReport	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
CreatePermission	acm-pca:CreatePermission	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DeleteCertificateAuthority	acm-pca:DeleteCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DeletePermission	acm-pca:DeletePermission	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

Amazon 私有 CA API 操作	所需的权限	资源
DeletePolicy	acm-pca:DeletePolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DescribeCertificateAuthority	acm-pca:DescribeCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DescribeCertificateAuthorityAuditReport	acm-pca:DescribeCertificateAuthorityAuditReport	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetCertificate	acm-pca:GetCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetCertificateAuthorityCertificate	acm-pca:GetCertificateAuthorityCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

Amazon 私有 CA API 操作	所需的权限	资源
GetCertificateAuthorityCsr	acm-pca:GetCertificateAuthorityCsr	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetPolicy	acm-pca:GetPolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ImportCertificateAuthorityCertificate	acm-pca:ImportCertificateAuthorityCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
IssueCertificate	acm-pca:IssueCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListCertificateAuthorities	acm-pca:ListCertificateAuthorities	N/A

Amazon 私有 CA API 操作	所需的权限	资源
ListPermissions	acm-pca:ListPermissions	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListTags	acm-pca:ListTags	N/A
PutPolicy	acm-pca:PutPolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
RevokeCertificate	acm-pca:RevokeCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
TagCertificateAuthority	acm-pca:TagCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

Amazon 私有 CA API 操作	所需的权限	资源
UntagCertificateAuthority	acm-pca:UntagCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
UpdateCertificateAuthority	acm-pca:UpdateCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

要提供访问权限，请为您的用户、组或角色添加权限：

- 通过身份提供者在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供者创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

Amazon 托管策略

Amazon 私有 CA 包括一组适用于 Amazon 管理 Amazon 私有 CA 员、用户和审计员的预定义托管策略。了解这些策略可以帮助您实施 客户托管策略。

选择下面列出的任何策略，以查看详细信息和示例策略代码。

AWSPrivateCAFullAccess

授予不受限制的管理控制。

有关策略详细信息的 JSON 列表，请参阅AWSPrivateCAFullAccess。

AWSPrivateCAReadOnly

授予限于只读 API 操作的访问权限。

有关策略详细信息的 JSON 列表，请参阅AWSPrivateCAReadOnly。

AWSPrivateCAPrivilegedUser

授予颁发和吊销 CA 证书的功能。此策略没有其他管理功能，不能颁发终端实体证书。权限与User 策略相互排斥。

有关策略详细信息的 JSON 列表，请参阅AWSPrivateCAPrivilegedUser。

AWSPrivateCAUser

授予颁发和吊销终端实体证书的功能。此策略没有管理功能，不能颁发 CA 证书。权限与PrivilegedUser策略相互排斥。

有关策略详细信息的 JSON 列表，请参阅AWSPrivateCAUser。

AWSPrivateCAAuditor

授予对只读 API 操作的访问权限和生成 CA 审计报告的权限。

有关策略详细信息的 JSON 列表，请参阅AWSPrivateCAAuditor。

AWSPrivateCAConnectorForKubernetesPolicy

为 Kubernetes Amazon 私有 CA 连接器授予基本权限。

有关策略详细信息的 JSON 列表，请参阅AWSPrivateCAConnectorForKubernetesPolicy。

的托 Amazon 管策略更新 Amazon 私有 CA

在下表中，查看自服务开始跟踪这些更改以 Amazon 私有 CA 来的 Amazon 托管策略更新的详细信息。要获得有关所有更改的自动提醒 Amazon 私有 CA，请订阅文档历史记录页面上的 RSS feed。

托管式策略更改

更改	描述	日期
新政策： Amazon PrivateCA ConnectorForKubernetesPolicy	引入了新的托管策略，用于适用于 Kubernetes 的 Amazon 私有 CA 连接器。	2025 年 5 月 19 日
Amazon PrivateCAPrivilegedUser 和 Amazon PrivateCauser-更新了政策	`StringLike` 替换为`ArnLike`、`StringNotLike` 和`ArnNotLike` 。 更新了模板 arn 以包含通配符`arn:aws:acm-pca:::template` 。`arn:aws:acm-pca:*:*:template`	2025 年 1 月 22 日
新策略名称： • `Amazon PrivateCAFullAccess` • `Amazon PrivateCAReadOnly` • `Amazon PrivateCAPrivilegedUser` • `Amazon PrivateCAAuditor` • `Amazon PrivateCAUser`	策略名称前缀已从 `Amazon CertificateManager PrivateCA` 更改为 `Amazon PrivateCA` 。 功能保持不变。	2023 年 2 月 13 日

客户托管策略

作为最佳实践，请勿使用您的 Amazon Web Services 账户根用户 与之互动 Amazon，包括 Amazon 私有 CA。而是使用 Amazon Identity and Access Management (IAM) 创建 IAM 用户、IAM 角色或联合用户。创建管理员组并将自己添加到其中。然后作为管理员登录。根据需要向组添加其他用户。

另一个最佳实践是创建可分配给用户的客户托管的 IAM policy。客户托管的策略是您可创建的基于身份的独立策略，您可以将这些策略附加到 Amazon 账户中的多个用户、组或角色。此类策略可限制用户仅执行您指定的 Amazon 私有 CA 操作。

下面的示例客户托管策略允许用户创建 CA 审计报告。这只是一个示例。你可以选择任何你想要的 Amazon 私有 CA 操作。有关更多示例，请参阅内联策略。

创建客户托管策略

1. 使用 Amazon 管理员的凭证登录 IAM 控制台。
2. 在控制台的导航窗格中，选择策略。
3. 选择创建策略。
4. 选择 JSON 选项卡。
5. 复制以下策略并将其粘贴到编辑器中。

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:CreateCertificateAuthorityAuditReport",
         "Resource":"*"
      }
   ]
}
```

6. 选择查看策略。
7. 对于名称，键入 PcaListPolicy。
8. (可选) 键入描述。
9. 选择创建策略。

管理员可以将策略附加到任何 IAM 用户以限制用户可以执行的 Amazon 私有 CA 操作。有关应用权限策略的方法，请参阅《IAM 用户指南》中的更改 IAM 用户的权限。

内联策略

内联策略是由您创建和管理的策略，它们直接嵌入在用户、组或角色中。以下策略示例说明如何分配执行 Amazon 私有 CA 操作的权限。有关内联策略的一般信息，请参阅《IAM 用户指南》https://docs.amazonaws.cn/IAM/latest/UserGuide/中的使用内联策略。您可以使用 Amazon Web Services 管理控制台、 Amazon Command Line Interface (Amazon CLI) 或 IAM API 来创建和嵌入内联策略。

> **Important**
>
> 我们强烈建议您在访问时使用多因素身份验证 (MFA)。 Amazon 私有 CA

主题

- 列出私有 CA
- 检索私有 CA 证书
- 导入私有 CA 证书
- 删除私有 CA
- Tag-on-create: 在创建 CA 时将标签附加到 CA
- Tag-on-create: 受限标记
- 使用标签控制对私有 CA 的访问权限
- Read-only 访问权限 Amazon 私有 CA
- 完全访问权限 Amazon 私有 CA

列出私有 CA

以下策略允许用户列出账户中的所有私有 CA。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:ListCertificateAuthorities",
         "Resource":"*"
```

```
      }
   ]
}
```

检索私有 CA 证书

以下策略允许用户检索特定的私有 CA 证书。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:GetCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"
   }
}
```

导入私有 CA 证书

以下策略允许用户导入私有 CA 证书。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:ImportCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"
   }
}
```

删除私有 CA

以下策略允许用户删除特定的私有 CA。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:DeleteCertificateAuthority",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"   }
}
```

Tag-on-create: 在创建 CA 时将标签附加到 CA

以下策略允许用户在创建 CA 期间应用标签。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Action": [
            "acm-pca:CreateCertificateAuthority",
            "acm-pca:TagCertificateAuthority"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

Tag-on-create: 受限标记

以下 tag-on-create 策略禁止在创建 CA 期间使用键值对 Environment=Prod。允许使用其他键值对进行标记。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:*",
         "Resource":"*"
      },
      {
         "Effect":"Deny",
         "Action":"acm-pca:TagCertificateAuthority",
         "Resource":"*",
         "Condition":{
            "StringEquals":{
               "aws:ResourceTag/Environment":[
                  "Prod"
               ]
            }
         }
      }
   ]
}
```

使用标签控制对私有 CA 的访问权限

以下策略仅允许访问带有键值对 Environment= 的 CA。PreProd它还要求新 CA 包含此标签。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*",
         "Condition":{
```

```
            "StringEquals":{
               "aws:ResourceTag/Environment":[
                  "PreProd"
               ]
            }
         }
      }
   ]
}
```

Read-only 访问权限 Amazon 私有 CA

以下策略允许用户描述和列出私有证书颁发机构并检索私有 CA 证书和证书链。

JSON

```
{
    "Version":"2012-10-17",
    "Statement":{
       "Effect":"Allow",
       "Action":[
          "acm-pca:DescribeCertificateAuthority",
          "acm-pca:DescribeCertificateAuthorityAuditReport",
          "acm-pca:ListCertificateAuthorities",
          "acm-pca:ListTags",
          "acm-pca:GetCertificateAuthorityCertificate",
          "acm-pca:GetCertificateAuthorityCsr",
          "acm-pca:GetCertificate"
       ],
       "Resource":"*"
    }
}
```

完全访问权限 Amazon 私有 CA

以下策略允许用户执行任何 Amazon 私有 CA 操作。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*"
      }
   ]
}
```

Cross-account 访问私有 CA 的安全最佳实践

Amazon 私有 CA 管理员可以与另一 Amazon Web Services 账户位管理员中的委托人（用户、角色等）共享 CA。收到并接受股票后，委托人可以使用 Amazon 私有 CA 或 Amazon Certificate Manager 资源使用证书颁发最终实体证书。委托人可以使用 CA 通过颁发从属 CA 证书 Amazon 私有 CA。

> **Important**
>
> 与跨账户场景中颁发的证书相关的费用由颁发证书的 Amazon 账户收取。

要共享对 CA 的访问权限， Amazon 私有 CA 管理员可以选择以下任一方法：

- 使用 Amazon Resource Access Manager (RAM) 将 CA 作为资源与其他账户的委托人共享，或者与之共享 Amazon Organizations。RAM 是跨账户共享 Amazon 资源的标准方法。有关 RAM 的更多信息，请参阅《Amazon RAM 用户指南》https://docs.amazonaws.cn/ram/latest/userguide/。有关 Amazon Organizations的更多信息，请参阅 Amazon Organizations 用户指南。
- 使用 Amazon 私有 CA API 或 CLI 将基于资源的策略附加到 CA，从而向其他账户中的委托人授予访问权限。有关更多信息，请参阅 Resource-based 政策。

本指南的 控制对私有 CA 的访问权限 部分提供了在单账户和跨账户场景中授予 CA 访问权限的工作流程。

RAM 中的客户托管权限

除了 RAM 中可用的 Amazon 托管权限外，还 Amazon 私有 CA 支持 RAM 客户托管权限 (CMP)。客户托管权限允许 CA 所有者定义跨账户委托人可以在共享 CA 上执行的一组自定义操作，从而提供比默认托管权限更精细的访问控制。 Amazon

以下操作适用于客户对`acm-pca:certificate-authority`资源类型的托管权限：

读取动作

- `acm-pca:DescribeCertificateAuthority`— 查看 CA 配置和状态。
- `acm-pca:GetCertificate`— 检索已颁发的证书。
- `acm-pca:GetCertificateAuthorityCertificate`— 检索 CA 证书和证书链。
- `acm-pca:ListPermissions`— 列出分配给 CA 的权限。
- `acm-pca:ListTags`— 列出与 CA 相关的标签。

写入动作

- `acm-pca:IssueCertificate`— 从共享 CA 颁发证书。
- `acm-pca:RevokeCertificate`— 撤销先前颁发的证书。

您可以创建包含这些操作的任意组合的客户托管权限。例如，您可以创建排除`IssueCertificate`和的只读权限`RevokeCertificate`，或者创建包含所有七个操作的完全访问权限。

有关创建客户托管权限的更多信息，请参阅《Amazon RAM 用户指南》中的创建客户托管权限。

Resource-based 政策

Resource-based 策略是您创建并手动附加到资源（在本例中为私有 CA）而不是用户身份或角色的权限策略。或者，与其创建自己的策略，不如使用 Amazon 托管策略 Amazon 私有 CA。通过 Amazon RAM 应用基于资源的策略， Amazon 私有 CA 管理员可以直接或通过 Amazon Organizations与其他 Amazon 账户中的用户共享对 CA 的访问权限。或者， Amazon 私有 CA 管理员可以使用 PCA API PutPolicy、GetPolicy和或相应的 Amazon CLI 命令 p ut-policy DeletePolicy、get- policy 和 del ete-policy 来应用和管理基于资源的策略。

有关基于资源的策略的一般信息，请参阅Identity-Based策略和 Resource-Based 策略以及使用策略控制访问权限。

要查看的基于资源的 Amazon 托管策略列表 Amazon 私有 CA，请导航到 Amazon Resource Access Manager 控制台中的托管权限库，然后搜索。CertificateAuthority与任何策略一样，在应用之前，我们建议在测试环境中应用该策略，以确保其符合您的要求。

Amazon 私有 CA 还支持 RAM 客户托管权限，允许您定义以下集合中的自定义操作组合：DescribeCertificateAuthorityGetCertificate、GetCertificateAuthorityCertificat和RevokeCertificate。客户托管权限使您可以灵活地授予最低权限访问权限，例如，向某些账户授予只读访问权限，同时允许其他账户颁发和吊销证书。有关更多信息，请参阅 RAM 中的客户托管权限。

Amazon Certificate Manager (ACM) 对私有 CA 具有跨账户共享访问权限的用户可以颁发由 CA 签署的托管证书。当您授予IssueCertificate操作权限时，可以通过向策略中添加acm-pca:TemplateArn条件来限制用于证书颁发的证书模板。

策略示例

本节提供了满足各种需求的跨账户策略示例。在所有情况下，都使用以下命令模式来应用策略：

```
$ aws acm-pca put-policy \
   --region region \
   --resource-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
   --policy file:///[path]/policyN.json
```

除了指定 CA 的 ARN 之外，管理员还提供一个 Amazon 账户 ID 或一个将被授予对 CA 的访问权限的 Amazon Organizations ID。为了便于阅读，以下每项策略的 JSON 都被格式化为文件，但也可以作为内联 CLI 参数提供。

> **Note**
>
> 必须严格遵循下面显示的基于 JSON 资源的策略的结构。客户只能配置委托人的 ID 字段（ Amazon 账号或 Org Amazon anizations ID ）和 CA ARN。

1. 文件：policy1.json – 与不同账户中的用户共享对 CA 的访问权限

   *555555555555*替换为共享 CA 的 Amazon 账户 ID。

   对于资源 ARN，请用您自己的值替换以下内容：

   - *aws*-分 Amazon 区。例如，、awsaws-us-gov、aws-cn、等。

- *us-east-1*-资源可用 Amazon 的地区，例如us-west-1。
- *111122223333*-资源所有者的 Amazon 账户 ID。
- *11223344-1234-1122-2233-112233445566*-证书颁发机构的资源 ID。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Sid": "ExampleStatementID",
            "Effect": "Allow",
            "Principal": {
                "AWS": "555555555555"
            },
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource": "arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID"
        },
        {
            "Sid": "ExampleStatementID2",
            "Effect": "Allow",
            "Principal": {
                "AWS": "555555555555"
            },
            "Action": [
                "acm-pca:IssueCertificate"
            ],
            "Resource": "arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
EndEntityCertificate/V1"
                }
            }
        }
    ]
```

```
}
```

2. 文件：policy2.json — 通过共享对 CA 的访问权限 Amazon Organizations

*o-a1b2c3d4z5*替换为 Amazon Organizations 身份证。

对于资源 ARN，请用您自己的值替换以下内容：

- *aws*-分 Amazon 区。例如，、awsaws-us-gov、aws-cn、等。
- *us-east-1*-资源可用 Amazon 的地区，例如us-west-1。
- *111122223333*-资源所有者的 Amazon 账户 ID。
- *11223344-1234-1122-2233-112233445566*-证书颁发机构的资源 ID。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatementID3",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "acm-pca:IssueCertificate",
            "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
EndEntityCertificate/V1",
                    "aws:PrincipalOrgID": "o-a1b2c3d4z5"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        },
        {
            "Sid": "ExampleStatementID4",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
```

```
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4z5"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        }
    ]
}
```

中的数据保护 Amazon 私有证书颁发机构

分 Amazon 担责任模型适用于中的数据保护 Amazon 私有证书颁发机构。如本模型所述 Amazon ，负责保护运行所有内容的全球基础架构 Amazon Web Services 云。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 Amazon Web Services 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅中国地区法律条款。

出于数据保护目的，我们建议您保护 Amazon Web Services 账户 凭证并使用 Amazon IAM Identity Center 或 Amazon Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 Amazon 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 Amazon CloudTrail。有关使用 CloudTrail 跟踪捕获 Amazon 活动的信息，请参阅Amazon CloudTrail 用户指南中的使用跟 CloudTrail 踪。
- 使用 Amazon 加密解决方案以及其中的所有默认安全控件 Amazon Web Services 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。

- 如果您在 Amazon 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》https://www.amazonaws.cn/compliance/fips/。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API Amazon 私有 CA 或 SDK 或以其他 Amazon Web Services 服务 方式使用控制台 Amazon CLI、API 或 Amazon SDK 的情况。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

Amazon 私有 CA 私钥的存储和安全合规性

私有 CA 的私钥存储在 Amazon 托管硬件安全模块 (HSM) 中。HSM 符合 FIPS PUB 140-3 加密模块的 3 级安全要求。

活动目录 Amazon 私有 CA 连接器中的数据加密

Amazon 私有 CA Connector for AD 存储有关连接器、模板、目录注册、服务主体名称和模板组访问控制条目的客户配置数据。此数据在传输中和静态时均加已密。使用 Amazon 私有 CA API 中的GetCertificate操作可以发现有关通过 Connector for AD 颁发的证书的信息。不存储有关颁发的证书或请求证书的客户端或计算机的信息 Amazon。

合规性验证 Amazon 私有证书颁发机构

Third-party Amazon 私有证书颁发机构 作为多个合规计划的一部分，审计师对安全性和 Amazon 合规性进行评估。其中包括 SOC、PCI、FedRAMP、HIPAA 及其他。

有关特定合规计划范围内的 Amazon 服务列表，请参阅按合规计划划分的规计划划分）。Amazon Web Services 服务 有关一般信息，请参阅合规计划。

您可以使用下载第三方审计报告 Amazon Artifact。有关更多信息，请参阅中的 “下载报告” Amazon Artifact。

您在使用 Amazon 私有 CA 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 Amazon 提供了以下资源来帮助实现合规性：

- 对于需要加密其 Amazon S3 桶的组织，以下主题介绍如何配置加密以容纳 Amazon 私有 CA 资产：
  - 加密审计报告

  - 加密 CRL
- 安全性与合规性快速入门指南安全性与合规性快速入门指南 – 这些部署指南讨论了架构注意事项，并提供了在 Amazon上部署基于安全性和合规性的基准环境的步骤。
- HIPAA 安全与合规架构白皮书 — 本白皮书描述了公司如何使用来创建应用程序。 Amazon HIPAA-compliant
- 合规资源 — 此工作簿和指南集可能适用于您的行业和所在地区。
- 使用Amazon Config 开发人员指南中的规则评估资源 — 该 Amazon Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- Amazon Security Hub CSPM— 此 Amazon 服务可全面了解您的安全状态 Amazon ，帮助您检查是否符合安全行业标准和最佳实践。

将审计报告与您的私有 CA 一起使用

您可以创建审核报告，以列出您的私有 CA 已颁发和吊销的所有证书。该报告将保存在您通过输入指定的新的或现有 S3 存储桶中。

> **Note**
>
> 已颁发超过 1 亿份证书的证书颁发机构不支持生成审计报告。

有关向审计报告添加加密保护的信息，请参阅 加密审计报告。

审计报告文件具有以下路径和文件名。Amazon S3 桶的 ARN 是 `amzn-s3-demo-bucket` 的值。`CA_ID` 是颁发 CA 的唯一标识符。`UUID` 是审计报告的唯一标识符。

```
amzn-s3-demo-bucket/audit-report/CA_ID/UUID.[json|csv]
```

您可以每 30 分钟生成一份新报告，并从存储桶中下载该报告。以下示例显示了一 CSV-separated 份报告。

```
awsAccountId,requestedByServicePrincipal,certificateArn,serial,subject,notBefore,notAfter,issue
123456789012,,arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/
certificate_ID,00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff,"2.5.4.5=#012345678901,2.5.4.44=
 Company,L=Seattle,ST=Washington,C=US",2020-03-02T21:43:57+0000,2020-04-07T22:43:57+0000,2020-0
pca:::template/EndEntityCertificate/V1
```

```
123456789012,acm.amazonaws.com,arn:aws:acm-pca:region:account:certificate-
authority/CA_ID/
certificate/
certificate_ID,ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00,"2.5.4.5=#012345678901,2.5.4.44=
 Company,L=Seattle,ST=Washington,C=US",2020-03-02T20:53:39+0000,2020-04-07T21:53:39+0000,2020-0
pca:::template/EndEntityCertificate/V1
```

以下示例显示了一 JSON-formatted 份报告。

```
[
   {
      "awsAccountId":"123456789012",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff",

 "subject":"2.5.4.5=#012345678901,2.5.4.44=#0a1b3c4d,2.5.4.65=#0a1b3c4d5e6f,2.5.4.43=#0a1b3c4d5
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2020-02-26T18:39:57+0000",
      "notAfter":"2021-02-26T19:39:57+0000",
      "issuedAt":"2020-02-26T19:39:58+0000",
      "revokedAt":"2020-02-26T20:00:36+0000",
      "revocationReason":"UNSPECIFIED",
      "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
   },
   {
      "awsAccountId":"123456789012",
      "requestedByServicePrincipal":"acm.amazonaws.com",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00",

 "subject":"2.5.4.5=#012345678901,2.5.4.44=#0a1b3c4d,2.5.4.65=#0a1b3c4d5e6f,2.5.4.43=#0a1b3c4d5
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2020-01-22T20:10:49+0000",
      "notAfter":"2021-01-17T21:10:49+0000",
      "issuedAt":"2020-01-22T21:10:49+0000",
      "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
   }
]
```

> **Note**
>
> Amazon Certificate Manager 续订证书时，私有 CA 审计报告会在该`requestedByServicePrincipal`字段中填充。`acm.amazonaws.com`这表示该 Amazon Certificate Manager 服务代表客户调 Amazon 私有 CA 用了 API 的`IssueCertificate`操作来续订证书。

为审计报告准备一个 Amazon S3 存储桶

> **Important**
>
> Amazon 私有 CA 不支持使用 A mazon S3 对象锁。如果您在存储桶上启用对象锁定，则 Amazon 私有 CA 无法向存储桶写入审计报告。

要存储您的审计报告，您需要准备 Amazon S3 桶。有关更多信息，请参阅如何创建 S3 桶？

您的 S3 存储桶必须受权限策略的保护，该策略 Amazon 私有 CA 允许访问和写入您指定的 S3 存储桶。授权用户和服务委托人需要`Put`权限 Amazon 私有 CA 才能在存储桶中放置对象，以及检索对象的`Get`权限。我们建议您应用如下所示的政策，该政策限制了对 Amazon 账户和私有 CA 的 ARN 的访问权限。或者，您可以使用 a ws: SourceOrg ID 条件键来限制对中特定组织的访问权限。 Amazon Organizations有关存储桶策略的更多信息，请参阅 Amazon 简单存储服务的存储桶策略。

创建审计报告

可从控制台或 Amazon CLI创建审计报告。

创建审计报告（控制台）

1. 登录您的 Amazon 账户并打开 Amazon 私有 CA 控制台，网址为https://console.aws.amazon.com/acm-pca/home。
2. 在私有证书颁发机构页面，从列表中选择您的私有 CA。
3. 从操作菜单中，选择生成审计报告。
4. 在审计报告目标下，对于创建新的 S3 桶？，选择是并键入唯一桶名称，或选择否并从列表中选择现有的桶。

如果您选择“是”，则 Amazon 私有 CA 会创建默认策略并将其附加到您的存储桶。默认策略包括`aws:SourceAccount`条件密钥，该密钥限制了对特定 Amazon 账户的访问权限。如果您想进一步限制访问权限，则可以在策略中添加其他条件密钥，如前面的示例所示。

如果您选择否，则必须先将策略附加到桶，然后才能生成审计报告。使用 为审计报告准备一个 Amazon S3 存储桶 中所述的策略模式。有关附加策略的信息，请参阅使用 Amazon S3 控制台添加存储桶策略。

5. 在输出格式下，为 JavaScript 对象表示法选择 JSON，为逗号分隔值选择 CSV。
6. 选择 Generate audit report (生成审核报告)。

创建审计报告 (Amazon CLI)

1. 如果您没有 S3 桶可用，则请创建一个。
2. 将策略附加到您的桶。使用 为审计报告准备一个 Amazon S3 存储桶 中所述的策略模式。有关附加策略的信息，请参阅使用 Amazon S3 控制台添加桶策略
3. 使用 create-certificate-authority-authity-au dit-report 命令创建审计报告并将其放入准备好的 S3 存储桶中。

```
$ aws acm-pca create-certificate-authority-audit-report \
--certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
--s3-bucket-name amzn-s3-demo-bucket \
--audit-report-response-format JSON
```

检索审计报告

要检索审计报告以进行检查，请使用 Amazon S3 控制台、API、CLI 或软件开发工具包。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的下载对象。

加密审计报告

您可以选择在包含审计报告的 Amazon S3 存储桶上配置加密。 Amazon 私有 CA 支持 S3 中资产的两种加密模式：

- 使用 Amazon 密 S3-managed AES-256钥自动进行服务器端加密。
- 客户使用管理加密 Amazon Key Management Service ，并根据您的规格 Amazon KMS key 进行配置。

> **Note**
>
> Amazon 私有 CA 不支持使用 S3 自动生成的默认 KMS 密钥。

以下过程介绍如何设置每个加密选项。

配置自动加密

完成以下步骤以启用 S3 服务器端加密。

1. 打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。
2. 在 Buckets 表中，选择用于存放您的 Amazon 私有 CA 资产的存储桶。
3. 在存储桶页面上，选择属性选项卡。
4. 选择默认加密卡。
5. 请选择启用。
6. 选择亚马逊 S3 密钥 (SSE-S3)。
7. 选择保存更改。

配置自定义加密

完成以下步骤以启用使用自定义密钥的加密。

1. 打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。
2. 在 Buckets 表中，选择用于存放您的 Amazon 私有 CA 资产的存储桶。
3. 在存储桶页面上，选择属性选项卡。
4. 选择默认加密卡。
5. 请选择启用。
6. 选择Amazon Key Management Service 密钥 (SSE-KMS)。
7. 选择 “从 Amazon KMS 密钥中选择” 或 “输入 Amazon KMS key ARN”。
8. 选择保存更改。
9. （可选）如果您还没有 KMS 密钥，请使用以下 Amazon CLI create-key 命令创建一个：

```
$ aws kms create-key
```

输出包含 KMS 密钥的密钥 ID 和 Amazon 资源名称（ARN）。下面是一个示例输出：

```
{
    "KeyMetadata": {
        "KeyId": "01234567-89ab-cdef-0123-456789abcdef",
        "Description": "",
        "Enabled": true,
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1478910250.94,
        "Arn": "arn:aws:kms:us-west-2:123456789012:key/01234567-89ab-
cdef-0123-456789abcdef",
        "AWSAccountId": "123456789012"
    }
}
```

10. 使用以下步骤，您可以向 Amazon 私有 CA 服务主体授予使用 KMS 密钥的权限。默认情况下，所有 KMS 密钥均为私有；只有资源拥有者可以使用 KMS 密钥加密和解密数据。但是，资源拥有者可以将 KMS 密钥的访问权限授予其他用户和资源。该服务主体必须位于存储 KMS 密钥的相同区域内。

    a. 首先，使用以下 `policy.json`get-key-policy 命令将您的 KMS 密钥默认策略保存为 ：

    ```
    $ aws kms get-key-policy --key-id key-id --policy-name default --output text
     > ./policy.json
    ```

    b. 在文本编辑器中打开 `policy.json` 文件。选择以下策略声明之一，并将其添加到现有策略中。

    如果您的 Amazon S3 桶密钥已启用，则请使用以下语句：

    ```
    {
       "Sid":"Allow ACM-PCA use of the key",
       "Effect":"Allow",
       "Principal":{
          "Service":"acm-pca.amazonaws.com"
       },
       "Action":[
          "kms:GenerateDataKey",
          "kms:Decrypt"
       ],
    ```

```
   "Resource":"*",
   "Condition":{
      "StringLike":{
         "kms:EncryptionContext:aws:s3:arn":"arn:aws:s3:::bucket-name"
      }
   }
}
```

如果您的 Amazon S3 桶密钥已禁用，则请使用以下语句：

```
{
   "Sid":"Allow ACM-PCA use of the key",
   "Effect":"Allow",
   "Principal":{
      "Service":"acm-pca.amazonaws.com"
   },
   "Action":[
      "kms:GenerateDataKey",
      "kms:Decrypt"
   ],
   "Resource":"*",
   "Condition":{
      "StringLike":{
         "kms:EncryptionContext:aws:s3:arn":[
            "arn:aws:s3:::bucket-name/acm-pca-permission-test-key",
            "arn:aws:s3:::bucket-name/acm-pca-permission-test-key-private",
            "arn:aws:s3:::bucket-name/audit-report/*",
            "arn:aws:s3:::bucket-name/crl/*"
         ]
      }
   }
}
```

c. 最后，使用以下 put-key-policy 命令应用更新后的策略：

```
$ aws kms put-key-policy --key-id key_id --policy-name default --policy file://
policy.json
```

中的基础设施安全 Amazon 私有证书颁发机构

作为一项托管服务 Amazon 私有证书颁发机构 ，受 Amazon 全球网络安全的保护。有关 Amazon 安全服务以及如何 Amazon 保护基础设施的信息，请参阅Amazon 云安全。要使用基础设施安全的最佳实践来设计您的 Amazon 环境，请参阅 S Amazon ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 Amazon 已发布的 API 调用 Amazon 私有 CA 通过网络进行访问。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（短暂的）或 ECDHE（椭圆曲线短暂的 Diffie-Hellman）。 Diffie-Hellman大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

Amazon 私有 CA VPC 终端节点 (Amazon PrivateLink)

您可以通过配置接口 VPC 终端节点在您 Amazon 私有 CA 的 VPC 和之间创建私有连接。接口端点由Amazon PrivateLink一种用于私密访问 Amazon 私有 CA API 操作的技术提供支持。 Amazon PrivateLink 在您的 VPC 之间以及 Amazon 私有 CA 通过 Amazon 网络路由所有网络流量，避免在开放的互联网上暴露。每个 VPC 终端节点都由您的 VPC 子网中一个或多个使用私有 IP 地址的弹性网络接口代表。

接口 VPC 终端节点直接连接您的 VPC， Amazon 私有 CA 无需互联网网关、NAT 设备、VPN Amazon Direct Connect 连接或连接。VPC 中的实例不需要公有 IP 地址便可与 Amazon 私有 CA API 进行通信。

要 Amazon 私有 CA 通过您的 VPC 使用，您必须从 VPC 内部的实例进行连接。或者，您可以使用 Amazon Virtual Private Network (Amazon VPN) 或将您的私有网络连接到 VPC Amazon Direct Connect。有关信息 Amazon VPN，请参阅 Amazon VPC 用户指南中的 VPN 连接。有关 Amazon Direct Connect的信息，请参阅《Amazon Direct Connect 用户指南》中的创建连接。

Amazon 私有 CA 不需要使用 Amazon PrivateLink，但我们建议将其作为额外的安全层。有关 Amazon PrivateLink 和 VPC 终端节点的更多信息，请参阅通过访问服务 Amazon PrivateLink。

Amazon 私有 CA VPC 终端节点的注意事项

在为设置接口 VPC 终端节点之前 Amazon 私有 CA，请注意以下注意事项：

- Amazon 私有 CA 在某些可用区域中可能不支持 VPC 终端节点。创建 VPC 端点时，请先在管理控制台中查看是否支持。不支持的可用区标记为“此可用区不支持服务”。
- VPC 终端节点不支持跨区域请求。确保在计划向 Amazon 私有 CA发出 API 调用的同一区域中创建端点。
- VPC 端点仅通过 Amazon Route 53 支持 Amazon 提供的 DNS。如果您希望使用自己的 DNS，可以使用条件 DNS 转发。有关更多信息，请参阅 Amazon VPC 用户指南中的 DHCP 选项集。
- 附加到 VPC 端点的安全组必须允许端口 443 上来自 VPC 的私有子网的传入连接。

Amazon 私有 CA API 目前支持以下方面的 VPC 终端节点 Amazon Web Services 区域：

- 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）
- 非洲（开普敦）
- 亚太地区（香港）
- 亚太地区（海得拉巴）
- 亚太地区（雅加达）
- 亚太地区（墨尔本）
- 亚太地区（孟买）
- 亚太地区（大阪）
- 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）
- 加拿大西部（卡尔加里）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- 欧洲地区（米兰）
- 欧洲地区（巴黎）

- 欧洲（西班牙）
- 欧洲地区（斯德哥尔摩）
- 欧洲（苏黎世）
- 以色列（特拉维夫）
- Middle East (Bahrain)
- 中东（阿联酋）：
- 南美洲（圣保罗）

为创建 VPC 终端节点 Amazon 私有 CA

您可以使用位于https://console.aws.amazon.com/vpc/或的 VPC 控制台为 Amazon 私有 CA 服务创建 VPC 终端节点 Amazon Command Line Interface。有关更多信息，请参阅 Amazon VPC 用户指南中的创建接口终端节点程序。 Amazon 私有 CA 支持在您的 VPC 内调用其所有 API 操作。

如果您已为端点启用私有 DNS 主机名，则默认的 Amazon 私有 CA 端点现在解析为您的 VPC 端点。有关默认服务终端节点的完整列表，请参阅服务终端节点和配额。

如果您尚未启用私有 DNS 主机名，则 Amazon VPC 将提供一个您可以使用的 DNS 端点名称，格式如下：

```
vpc-endpoint-id.acm-pca.region.vpce.amazonaws.com
```

> **Note**
>
> 该值*region*表示所 Amazon 私有 CA支持的 Amazon 区域（例如us-east-2美国东部（俄亥俄州）地区的区域标识符。有关 Amazon 私有 CA的列表，请参阅 Amazon Certificate Manager Private Certificate Authority 端点和配额。

有关更多信息，请参阅 Amazon Amazon 私有 CA VPC 用户指南中的 VPC 终端节点 (Amazon PrivateLink)。

为创建 VPC 终端节点策略 Amazon 私有 CA

您可以为的 Amazon VPC 终端节点创建策略 Amazon 私有 CA 以指定以下内容：

- 可执行操作的主体

- 可执行的操作
- 可对其执行操作的资源

有关更多信息，请参阅 Amazon VPC 指南中的使用 VPC 终端节点控制对服务的访问。

示例-用于 Amazon 私有 CA 操作的 VPC 终端节点策略

连接到终端节点时，以下策略授予所有委托人访问 Amazon 私有 CA 操作IssueCertificate、、、DescribeCertificateAuthorityGetCertificateGetCertificateAu和ListTags的权限。每个 Stanza 中的资源都是一个私有 CA。第一个 Stanza 授权使用指定的私有 CA 和证书模板创建终端实体证书。如果您不想控制要使用的模板，则不需要 Condition 部分。但是，删除此部分后将允许所有委托人创建 CA 证书以及终端实体证书。

```
{
      "Statement":[
         {
            "Principal":"*",
            "Effect":"Allow",
            "Action":[
               "acm-pca:IssueCertificate"
            ],
            "Resource":[
               "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
            ],
            "Condition":{
               "StringEquals":{
                  "acm-pca:TemplateArn":"arn:aws:acm-pca:::template/
EndEntityCertificate/V1"
               }
            }
         },
         {
            "Principal":"*",
            "Effect":"Allow",
            "Action":[
               "acm-pca:DescribeCertificateAuthority",
               "acm-pca:GetCertificate",
               "acm-pca:GetCertificateAuthorityCertificate",
               "acm-pca:ListPermissions",
               "acm-pca:ListTags"
            ],
```

```
            "Resource":[
               "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
            ]
         }
      ]
   }
```

Dual-stack 端点支持

Amazon 私有证书颁发机构 提供了同时支持 IPv4 和 IPv6 客户端的双栈公共端点。双栈端点使客户端能够 Amazon 私有 CA 使用 IPv4 或 IPv6 地址与之通信。 Amazon 私有 CA 适用于 Active Directory 和适用于 SCEP 的 Amazon 私有 CA 连接器也支持双堆栈

的 Amazon 私有 CA 双栈公共终端节点同时`https://acm-pca.your-region.api.aws`支持 IPv4 和 IPv6 客户端。 Amazon 私有 CA 也可以使用 IPv4 和 IPv6 从您的虚拟私有云 (VPC) 通过 IPv4 和 IPv6 进行 Amazon PrivateLink私密访问。有关为其创建私有接口 VPC 终端节点的更多信息 Amazon 私有 CA，请参阅Amazon 私有 CA VPC 终端节点 (Amazon PrivateLink)。

有关更多信息，请参阅以下资源：

- 为 VPC 和子网分配 IP 地址
- 为你的 VPC 提供 IPv6 支持

在 IAM 中使用 IPv6 地址和 Amazon 私有 CA

在尝试 Amazon 私有证书颁发机构 通过 IPv6 进行访问之前，请确保所有包含 IP 地址限制的 IAM 策略都已更新为包含 IPv6 地址范围。如果未将基于 IP 的策略更新为处理 IPv6 地址，则可能在开始使用 IPv6 时导致客户端错误地失去或获得访问权限。要了解有关 Amazon 私有 CA 双栈支持的更多信息，请参阅Dual-stack 端点支持。

> **Important**
>
> 这些语句并未允许任何操作。请将这些语句与允许特定操作的其他语句结合使用。

以下语句明确拒绝对来自 IPv4 地址`192.0.2.*`范围的请求访问所有 Amazon 私有 CA 权限。任何超出此范围的 IP 地址都不会被明确拒绝 Amazon 私有 CA 权限。由于所有 IPv6 地址都在拒绝范围之外，因此此语句并未明确拒绝任何 IPv6 地址的 Amazon 私有 CA 权限。

```
{
    "Sid": "DenyPrivateCAPermissions",
    "Effect": "Deny",
    "Action": [
        "acm-pca:*"
    ],
    "Resource": "*",
    "Condition": {
        "NotIpAddress": {
            "aws:SourceIp": [
                "192.0.2.0/24"
            ]
        }
    }
}
```

您可以修改Condition元素以拒绝 IPv4 (192.0.2.0/24) 和 IPv6 (2001:db8::/32) 地址范围，如以下示例所示：

```
{
    "Sid": "DenyPrivateCAPermissions",
    "Effect": "Deny",
    "Action": [
        "acm-pca:*"
    ],
    "Resource": "*",
    "Condition": {
        "NotIpAddress": {
            "aws:SourceIp": [
                "192.0.2.0/24",
                "2001:db8::/32"
            ]
        }
    }
}
```

Amazon 私有证书颁发机构 客户 CP/CPS 框架

Amazon 私有证书颁发机构 提供基础设施服务，使您能够创建证书颁发机构 (CA) 层次结构，包括根和从属层次 CAs，而无需支付运营本地 CA 的投资和维护成本。当您使用 Amazon 私有 CA 创建 CA 层次结构时，您和 Amazon 私有 CA之间需要共同承担责任。在 Amazon 运营、管理和控制服务运营设施的人身安全时，分担责任模式可以帮助减轻您的运营负担。您负责并管理证书颁发机构（包括创建和删除 CA 资源；分配信任锚；创建 PKI 层次结构；认证策略和实践；允许或拒绝 CA 共享的配置 Amazon Web Services 账户；模板使用策略；审计；访问控制，包括职责分离；以及其他 CA 配置和策略）。您应仔细考虑所选择的服务，因为您的责任因所使用的服务、这些服务与您的 IT 环境的集成以及适用的法律和法规而异。有关更多信息，请参阅Amazon Web Services 云 安全分担责任模型。

为私有证书颁发机构创建证书策略 (CP) 或认证实践声明 (CPS) 是管理公钥基础设施 (PKI) 的关键部分。CP 定义了您的 PKI 的所有要求/规则，CPS 解释了您如何满足 CP 要求。您负责创建 CP 和 CPS 作为 PKI 的证书颁发机构。 Amazon 私有 CA 为您提供 Amazon 控制和合规文档，例如Amazon 系统和组织控制 (SOC) 2 报告，可用于帮助您创建 CP 和 CPS，并根据需要执行控制评估和验证程序。Amazon SOC 报告是独立的第三方检查报告，用于展示如何 Amazon 实现关键合规控制和目标。这些报告的目的是帮助您和您的审计师了解为支持运营和合规而建立的 Amazon 控制措施。

本文档提供了一个符合 RFC 3647 的框架，以帮助您编写 CP 和 CPS，并确定了您与之间的共同责任。 Amazon 私有 CA CP/CPS 要求中负 Amazon 私有 CA 有合规责任的部分用 “共享” 或 “” 标识，并提供相应的 “补充信息”，以帮助您了解如何 Amazon 私有 CA 满足相关的 CP/CPS 要求。Amazon 私有证书颁发机构例如，要求 5 (4.5.1) 是一项 Amazon 私有 CA 责任，您可以在 Amazon SOC 2 报告的第 D.6 节中找到相应的控制语言，以帮助完成 CP/CPS。有关 Amazon SOC 报告以及如何申请访问 SOC 报告的更多信息，请访问我们的 SOC FAQs 页面。

CP/CPS 要求和责任

CP/CPS 要求	责任	补充信息
1. 简介 (全部)	您	您负责记录与您的 PKI 相关的概述、文档名称和身份、PKI 参与者、证书使用、策略管理以及定义和首字母缩略词。
2. 出版物和存储库职责 (全部)	您	您负责记录与您的 PKI 相关的定义。

CP/CPS 要求	责任	补充信息
3. 身份验证和认证 (全部)	您	在证书颁发之前，您负责记录用于向 CA 或注册机构 (RA) 验证最终用户证书申请人的身份和/或其他属性的程序。
4. 证书生命周期操作要求（4.4.1 — 4.4.6、4.4.9 — 4.4.11）	已共享	您有责任具体说明颁发 CA、主体 CAs RAs、订阅者或其他参与者在证书生命周期方面的要求。 Amazon 私有 CA 为您提供两种完全托管的机制来帮助支持吊销状态检查：在线证书状态协议 (OCSP) 和证书吊销列表 (CRLs)，可帮助您满足 4.4.9 和 4.4.10 的要求。
4. 证书生命周期操作要求（4.4.7、4.4.8、4.4.12）	不适用	Amazon 私有 CA 不支持证书重新密钥、证书修改或密钥托管和恢复。
5. 设施、管理和运营控制 (4.5.1)	Amazon 私有 CA	您可以继承访问控制，以帮助您满足本节中在 Amazon 私有 CA SOC 2 第 2 类报告范围内的要求（参见第 D.6 节 “物理安全和环境保护”）。 **Note** 您对导出或传出环境的 CA 数据的物理安全和数据分类负责，但不负责存储在 Amazon 环境中的 CA 数据的物理安全 Amazon。

CP/CPS 要求	责任	补充信息
5. 设施、管理和运营控制 (4.5.2)	已共享	您有责任满足本节中关于为公用钥匙基础结构环境的操作定义可信角色的要求。 Amazon 私有 CA 维护特定于加密模块物理访问的可信角色。
5. 设施、管理和运营控制 (4.5.3)	已共享	您有责任满足本节中针对您可信人员的背景调查、培训和纪律处分程序的要求。 对于属于 Amazon 私有 CA SOC 2 类型 2 报告范围的 Amazon 员工，您可以继承与背景调查、培训和纪律处分程序相关的控制措施（参见 A 部分、政策、A.1 控制环境、B. 通信以及 D.1 安全组织和 D.2 员工用户访问权限）。

CP/CPS 要求	责任	补充信息
5. 设施、管理和运营控制 (4.5.4)	已共享	您负责启用、配置保留以及保护 CloudTrail 和审计报告日志和 CloudWatch 警报。此外，您还负责创建日志处理程序，并对您使用 Amazon 私有 CA 服务进行漏洞评估，以满足本节的要求。 您可以继承与 Amazon 私有 CA SOC 2 第 2 类报告范围内的日志可用性、物理access/site security, CA/RA配置管理、 Amazon 基础设施日志安全以及 Amazon 基础设施漏洞评估相关的控制措施（参见 A.1 节控制环境、C.1 节服务承诺、D.2 员工用户访问权限、D.3 逻辑安全、D.6 物理安全和环境保护、D.7 变更管理、D.8 数据完整性、可用性和冗余以及 E.1 监控活动）。
5. 设施、管理和运营控制 (4.5.5)	已共享	您负责配置满足本节要求的备份和保留期。 您继承与 Amazon 私有 CA SOC 2 类型 2 报告范围内的日志可用性（配置时）相关的控制措施（请参阅 D.8 数据完整性、可用性和冗余）。
5. 设施、管理和运营控制 (4.5.6)	不适用	Amazon 私有 CA 不支持密钥切换。

CP/CPS 要求	责任	补充信息
5. 设施、管理和运营控制 (4.5.7)	已共享	您负责实施符合本节要求的事件和泄露处理程序 Amazon 私有 CA ，具体取决于您的使用情况。 您可以继承特定于物理站点住房和基础设施运营的事件、危害处理程序、业务连续性和灾难恢复程序，这些程序可帮助您满足本节中属于 Amazon 私有 CA SOC 2 第 2 类隐私报告范围内的要求（请参阅 D.8 数据完整性、可用性和冗余性以及 D.10 部分 “隐私”）。
5. 设施、管理和运营控制 (4.5.8)	您	您需要记录与终止和终止CA或RA的程序相关的要求，包括CA和RA档案记录保管人的身份。
6. 技术控制 (4.6.1)	已共享	您负责记录您的 PKI 的密钥生成和安装需求。 Amazon 私有 CA 为您提供经过 FIPS 140-3 级别 3 认证的加密模块，可生成 CA 密钥。
6. 技术控制 (4.6.2)	已共享	您负责记录私钥保护和加密模块工程控制，例如加密标准要求和多人控制。 Amazon 私有 CA 为您提供经过 FIPS 140-3 级别 3 认证的加密模块，可生成 CA 密钥和两方物理访问控制。 HSMs

CP/CPS 要求	责任	补充信息
6. 技术控制 (4.6.3)	您	您负责记录密钥对管理的其他方面，例如公钥的存档和证书的使用期限。
6. 技术控制 (4.6.4)	不适用	Amazon Amazon 私有 CA HSMs 始终在线，没有 “激活数据” 的概念。 **Note** 您负责实施对私有 CA 的用户访问控制，以适当限制创建 CA 和颁发证书的能力。
6. 技术控制 (4.6.5)	已共享	您有责任记录您使用私有 CA 的计算机安全控制措施。 您可以继承与 Amazon 私有 CA SOC 2 第 2 类报告范围内的 Amazon 员工逻辑访问权限、 Amazon 基础设施的网络和计算机安全控制以及 Amazon 员工帐户的密码参数控制相关的控制（参见第 D.2 节 “员工用户访问权限”、“D.3 逻辑安全” 和 D.6 物理安全和环境保护）。

CP/CPS 要求	责任	补充信息
6. 技术控制 (4.6.6)	已共享	您有责任记录与您使用私有 CA 相关的安全管理控制措施。 您可以继承与 Amazon 私有 CA SOC 2 第 2 类报告范围内的 Amazon 私有 CA 服务系统开发控制相关的控制措施（参见第 D.7 节变更管理）。
6. 技术控制 (4.6.7)	已共享	如果适用于您的 PKI 环境，则您有责任记录您使用私有 CA 的网络安全控制措施。 您可以继承与 Amazon 私有 CA SOC 2 第 2 类报告范围内的 Amazon 基础设施网络安全控制相关的控制措施（参见第 C.1 节服务承诺、D.3 逻辑安全和 E.1 监控活动）。
6. 技术控制 (4.6.8)	Amazon 私有 CA	Amazon 私有 CA 使用可信的时间源为 CA 数据加上时间戳。
7. 证书、CRL 和 OCSP 配置文件 (全部)	已共享	您负责记录符合您的 PKI 环境需求的配置文件要求和证书输入。 Amazon 私有 CA 为您提供个人资料模板，以帮助满足您的个人资料要求。

CP/CPS 要求	责任	补充信息
8. 合规审计和其他评估 (全部)	已共享	您负责记录合规性审计和其他评估。 Amazon 私有 CA 为您提供 SOC 2 报告，以帮助您和您的审计员了解为支持运营和合规而建立的 Amazon 控制措施。
9. 其他商业和法律事务	您	您负责记录涵盖您的私有 CA 的一般业务和法律事务。

监控 Amazon 私有 CA 资源

监控是维护和其他 Amazon 解决方案的可靠性、可用性和性能的重要组成部分。 Amazon 私有 CA Amazon 提供以下监控工具 Amazon 私有 CA，供您监视、报告问题并在适当时自动采取措施：

- Amazon 会实时 CloudWatch监控您的 Amazon 资源和您运行 Amazon 的应用程序。您可以收集和跟踪指标，创建自定义的控制面板，以及设置警报以在指定的指标达到您指定的阈值时通知您或采取措施。例如，您可以 CloudWatch 跟踪您的 Amazon EC2 实例的 CPU 使用率或其他指标，并在需要时自动启动新实例。有关更多信息，请参阅 Amazon CloudWatch 用户指南。
- Amazon CloudWatch Logs 允许您监控、存储和访问来自 Amazon EC2 实例和其他来源的日志文件。 CloudTrail CloudWatch 日志可以监视日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch 日志用户指南。
- Amazon CloudTrail 捕获由您的 Amazon Web Services 账户 或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 桶。您可以标识哪些用户和账户调用了 Amazon、发出调用的源 IP 地址以及调用的发生时间。有关更多信息，请参阅 Amazon CloudTrail 《用户指南》。
- Amazon EventBridge 是一项无服务器事件总线服务，可以轻松地将您的应用程序与来自各种来源的数据连接起来。 EventBridge 提供来自您自己的应用程序、 Software-as-a-Service (SaaS) 应用程序和 Amazon 服务的实时数据流，并将这些数据路由到 Lambda 等目标。这使您能够监控服务中发生的事件，并构建事件驱动的架构。有关更多信息，请参阅 Amazon EventBridge 用户指南。

以下主题描述了可用于的 Amazon 云监控工具。 Amazon 私有 CA

Amazon 私有 CA CloudWatch 指标

Amazon CloudWatch 是一项 Amazon 资源监控服务。您可以使用 CloudWatch 来收集和跟踪指标、设置警报以及自动对 Amazon 资源变化做出反应。 CloudWatch 指标至少发布一次。

Amazon 私有 CA 支持以下 CloudWatch 指标。

指标	说明
CRLGenerated	已生成证书吊销列表 (CRL)。此指标仅适用于私有 CA。

指标	说明
MisconfiguredCRLBucket	为 CRL 指定的 S3 存储桶的配置不正确。请检查存储桶策略。此指标仅适用于私有 CA。
Time	从颁发请求到颁发完成（或失败）之间的时间（以毫秒为单位）。此指标仅适用于该IssueCertificate操作。
Success	已成功颁发证书。此指标仅适用于该IssueCertificate操作。
Failure	操作失败。此指标仅适用于该IssueCertificate操作。
CertificateAuthority	您账户中当前区域的私有证书颁发机构总数。该指标包括所有状态（“活动”、“已禁用”、“创建 CAs 中”等），可用于监控 CA 库存并根据账户配额跟踪使用情况。
CertificatesPerCA	特定私有证书颁发机构自创建以来颁发的证书总数。该指标会随着证书的成功颁发而增加，并且在证书到期或被吊销时不会减少。该指标适用于个人私 CAs 人，可用于跟踪证书颁发量和容量规划。此指标仅在所有者账户中发布，不适用于与 CA 共享的账户。

> **Note**
>
> 对于在 2026 年 3 月 26 日之前 CAs 创建的任何私有内容，CertificateAuthority和CertificatesPerCA指标可能要等到 2026 年 4 月 30 日才能在中公布。 CloudWatch

有关 CloudWatch 指标的更多信息，请参阅以下主题：

- 使用 Amazon CloudWatch 指标
- 创建 Amazon CloudWatch 警报

Amazon 私有 CA 使用 CloudWatch 事件进行监控

您可以使用 Amazon CloudWatch Events 实现 Amazon 服务自动化，并自动响应系统事件，例如应用程序可用性问题或资源更改。来自 Amazon 服务的事件以近乎实时的方式传递到 CloudWatch 活动。您可以编写简单的规则来指明您感兴趣的事件，以及当事件与规则匹配时要采取的自动操作。CloudWatch 活动至少发布一次。有关更多信息，请参阅创建在 CloudWatch 事件上触发的事件规则。

CloudWatch 使用 Amazon 将事件转化为操作 EventBridge。借 EventBridge助，您可以使用事件触发目标，包括 Amazon Lambda 函数、 Amazon Batch 作业、Amazon SNS 主题等。有关更多信息，请参阅什么是亚马逊 EventBridge？

创建私有 CA 时成功或失败

这些事件由CreateCertificateAuthority操作触发。

成功

成功时，该操作将返回新 CA 的 ARN。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Creation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:14:56Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"success"
   }
}
```

Failure

失败时，该操作将返回原 CA 的 ARN。使用 ARN，您可以致电DescribeCertificateAuthority确定 CA 的状态。

```
{
```

```
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Creation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:14:56Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure"
   }
}
```

颁发证书时成功或失败

这些事件由IssueCertificate操作触发。

成功

成功后，该操作将 ARNs 返回 CA 和新证书的。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

Failure

失败时，该操作将返回证书 ARN 和 CA 的 ARN。使用证书 ARN，您可以致电GetCertificate查看失败原因。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"failure"
   }
}
```

吊销证书时成功

此事件由RevokeCertificate操作触发。

如果吊销失败或证书已被吊销，则不会发送任何事件。

成功

成功后，该操作将 ARNs 返回 CA 和已吊销证书的。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Revocation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-05T20:25:19Z",
   "region":"region",
```

```
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

生成 CRL 时成功或失败

这些事件由操作触发，该RevokeCertificate操作应导致创建证书吊销列表 (CRL)。

成功

成功时，该操作将返回与 CRL 关联的 CA 的 ARN。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:07:08Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"success"
   }
}
```

失败 1 – 由于权限错误，CRL 无法保存到 Amazon S3

如果发生此错误，请检查您的 Amazon S3 桶权限。

```
{
   "version":"0",
```

```
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to write CRL to S3. Check your S3 bucket permissions."
   }
}
```

失败 2 – 由于内部错误，CRL 无法保存到 Amazon S3

如果发生此错误，请重试该操作。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to write CRL to S3. Internal failure."
   }
}
```

失败 3-创建 CRL 失 Amazon 私有 CA 败

要解决此错误，请检查您的 CloudWatch 指标。

```
{
```

```
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to generate CRL. Internal failure."
   }
}
```

创建 CA 审计报告时成功或失败

这些事件由CreateCertificateAuthorityAuditReport操作触发。

成功

成功时，该操作将返回 CA 的 ARN 和审计报告的 ID。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Audit Report Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:54:20Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "audit_report_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

Failure

在您的 Amazon S3 存储桶上 Amazon 私有 CA 缺乏PUT权限、在存储桶上启用加密或其他原因时，审计报告可能会失败。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Audit Report Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:54:20Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "audit_report_ID"
   ],
   "detail":{
      "result":"failure"
   }
}
```

使用记录 Amazon 私有证书颁发机构 API 调用 Amazon CloudTrail

Amazon 私有证书颁发机构 与 Amazon CloudTrail一项服务集成，该服务提供用户、角色或 Amazon 服务在中执行的操作的记录 Amazon 私有 CA。 CloudTrail 捕获 Amazon 私有 CA 作为事件的 API 调用和签名操作。捕获的调用包括来自 Amazon 私有 CA 控制台的调用和对 Amazon 私有 CA API 操作的代码调用。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括的事件 Amazon 私有 CA。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向哪个请求发出 Amazon 私有 CA、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅《Amazon CloudTrail 用户指南》。

Amazon 私有 CA 信息在 CloudTrail

CloudTrail 在您创建账户 Amazon Web Services 账户 时已在您的账户上启用。当活动发生在中时 Amazon 私有 CA，该活动会与其他 Amazon 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在中查看、搜索和下载最近发生的事件 Amazon Web Services 账户。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的 Amazon Web Services 账户事件（包括的事件） Amazon 私有 CA，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 Amazon Web Services 区域。跟踪记录 Amazon 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 Amazon 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

所有 Amazon 私有 CA 操作均由 API 参考记录 CloudTrail 并记录在 Amazon 私有 CA API 参考中。例如，调用`IssueCertificate`和`CreateAuditReport`操作会在 CloudTrail 日志文件中生成条目。`ImportCACertificate`

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 Amazon Identity and Access Management (IAM) 用户凭证发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 Amazon 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

Amazon 私有 CA 管理事件

Amazon 私有 CA 与集成 CloudTrail 以记录用户、角色或 Amazon 服务在中执行的 API 操作 Amazon 私有 CA。您可以使用 CloudTrail 实时监控 Amazon 私有 CA API 请求并将日志存储在亚马逊简单存储服务、Amazon Log CloudWatch s 和 Amazon Ev CloudWatch ents 中。 Amazon 私有 CA 支持将以下操作和操作作为事件记录在 CloudTrail 日志文件中：

- CreateCertificateAuthority
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority
- DeletePermission
- DeletePolicy

- DescribeCertificateAuthority
- DescribeCertificateAuthorityReport
- GetCertificate
- GetCertificateAuthorityCertificate
- GetCertificateAuthorityCsr
- GetPolicy
- ImportCertificateAuthorityCertificate
- IssueCertificate
- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthority
- UntagCertificateAuthority
- UpdateCertificateAuthority
- `GenerateOCSPResponse`- Amazon 私有 CA 生成 OCSP 响应时触发。
- `SignCertificate`-在您的客户来电时生成IssueCertificate。
- `SignOCSPResponse`-在 Amazon 私有 CA 签署 OCSP 响应时生成。
- `GenerateCRL`-生成证书吊销列表 (CRL) 时生 Amazon 私有 CA 成。
- `SignCACSR`-在 Amazon 私有 CA 签署证书颁发机构 (CA) 证书签名请求 (CSR) 时生成。
- `SignCRL`-在 Amazon 私有 CA 签署 CRL 时生成。

示例 Amazon 私有 CA 事件

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下是 Amazon 私有 CA CloudTrail 事件的示例。

示例 1：管理事件，IssueCertificate

以下示例显示了演示该IssueCertificate操作的 CloudTrail 日志条目。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

示例 2：管理事件，ImportCertificateAuthorityCertificate

以下示例显示了演示该ImportCertificateAuthorityCertificate操作的 CloudTrail 日志条目。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:53:28Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"ImportCertificateAuthorityCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
```

```
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "certificate":{
         "hb":[
            45,
            45,
            ...10
         ],
         "offset":0,
         "isReadOnly":false,
         "bigEndian":true,
         "nativeByteOrder":false,
         "mark":-1,
         "position":1257,
         "limit":1257,
         "capacity":1257,
         "address":0
      },
      "certificateChain":{
         "hb":[
            45,
            45,
            ...10
         ],
         "offset":0,
         "isReadOnly":false,
         "bigEndian":true,
         "nativeByteOrder":false,
         "mark":-1,
         "position":1139,
         "limit":1139,
         "capacity":1139,
         "address":0
      }
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

对问题进行故障排除 Amazon 私有证书颁发机构

如果您在使用 Amazon 私有证书颁发机构时遇到问题，请参阅以下主题。

主题

- 解决 Amazon 私有 CA 证书吊销问题
- Amazon 私有证书颁发机构 异常消息疑难解答
- 对 Amazon 私有 CA Matter 兼容的证书错误进行故障排除

解决 Amazon 私有 CA 证书吊销问题

OCSP 响应延迟

如果调用方远离区域边缘缓存或颁发 CA 的区域，则 OCSP 的响应速度可能会变慢。有关区域边缘缓存可用性的更多信息，请参阅全球边缘网络。建议在靠近证书使用地点的区域颁发证书。

撤销自签名证书

您无法撤销自签名的 CA 证书。要从功能上吊销证书，请删除 CA。

Amazon 私有证书颁发机构 异常消息疑难解答

Amazon 私有 CA 命令可能由于多种原因而失败。有关每个异常以及关于解决这些异常的建议的信息，请参阅下表。

Amazon 私有 CA 例外情况

返回的异常 Amazon 私有 CA	说明	修复
`AccessDeniedException`	私有 CA 尚未将使用给定命令所需的权限委派给调用账户。	有关在中委派权限的信息 Amazon 私有 CA，请参阅将证书续订权限分配给 ACM。
`InvalidArgsException`	使用无效参数发出了证书创建或续订请求。	检查命令的单个文档，以确保您的输入参数有效。如果您要创建新证书，请确保所请求的

返回的异常 Amazon 私有 CA	说明	修复
		签名算法可以与 CA 的密钥类型一起使用。
InvalidStateException	关联的私有 CA 因不处于 ACTIVE 状态而无法续订证书。	尝试还原私有 CA。如果私有 CA 不在其还原期，则无法还原该 CA，并且无法续订证书。
LimitExceededException	每个证书颁发机构 (CA) 都有其可颁发的证书配额。与指定证书关联的私有 CA 已达到其配额。有关更多信息，请参阅《 Amazon Web Services 一般参考 指南》中的服务限额。	请联系Amazon Web Services 支持 中心申请增加配额。
MalformedCSRException	无法验证或确认已提交给 Amazon 私有 CA 的证书签名请求 (CSR)。	确认已正确生成和配置您的 CSR。
OtherException	内部错误导致了请求失败。	尝试再次运行命令。如果问题仍然存在，请联系Amazon Web Services 支持 中心。
RequestFailedException	您的 Amazon 环境中的网络问题导致请求失败。	重试请求。如果仍然失败，请检查您的 Amazon VPC（VPC）配置。
ResourceNotFoundException	颁发证书的私有 CA 已删除，不再存在。	从另一个活动 CA 请求新证书。

返回的异常 Amazon 私有 CA	说明	修复
`ThrottlingException`	因超出配额而导致请求的 API 操作失败。	确认发出的调用数量不超过 Amazon 私有 CA允许的数量。 如果您遇到了瞬态条件而不是超出配额，也可能发生 `ThrottlingException` 错误。如果您遇到该错误并且发出的调用数量未超过配额，请重试您的请求。 如果您即将接近配额，则可以请求增加配额。有关更多信息，请参阅 Amazon Web Services 一般参考 指南中的 Service Quotas。
`ValidationException`	请求的输入参数格式不正确，或者根证书的有效期在所请求证书的有效期之前结束。	检查命令输入参数的语法要求以及 CA 根证书的有效期。有关更改有效期的信息，请参阅在中更新私有 CA Amazon 私有证书颁发机构。

对 Amazon 私有 CA Matter 兼容的证书错误进行故障排除

Matter 连接标准规定了可提高物联网（IoT）设备安全性和一致性的证书配置。有关创建符合 Matter 标准的根 CA、中间 CA 和终端实体证书的 Java 示例，请访问 用于 Amazon 私有 CA 实现案件证书。

为了帮助进行故障排除，Matter 开发人员提供了一种名为 chip-cert 的证书验证工具。下表列出了此工具报告的错误以及修正措施。

错误代码	含义	修复
0x0000035	`BasicConstraints` 、`KeyUsage`、和	确保为使用案例选择正确的模板。

错误代码	含义	修复
	`ExtensionKeyUsage` 扩展必须标记为重要。	
0x0000000	必须存在授权密钥标识符扩展。	Amazon 私有 CA 不在根证书上设置授权密钥标识符扩展名。您CSR 生成一个 Base64 编码的AuthorityKeyIdentifier 值，然后将。CustomExtension有关更多信息，请参阅激活节点操作证书 (NCA。和激活产品认证机构 (PAA)。
0x000000E	证书已过期。	确保您使用的证书未过期。

错误代码	含义	修复

0x00000004	证书链验证失败。	如果您在不使用所提供的 Java 示例的情况下尝试创建符合 Matte 实体证书，则可能会遇到此错误，这些示例使用 Amazon 私有 C 传递正确配置的。 KeyUsage
		
		默认情况下， Amazon 私有 CA 生成九位 KeyUsage 扩展值，第 成一个额外的字节。在格式转换期间，Matter 会忽略额外的字节 验证失败。但是，APIPassthrough 模板CustomExtension中 于设置KeyUsage值中的确切字节数。有关示例，请参阅创建节点 书 (NOC)。
		
		如果您修改示例代码或使用 OpenSSL 等其他 X.509 实用程序， 行手动验证以避免链验证错误。
		
		验证转换是否无损
		
		1. 使用 openssl 验证节点（终端实体）证书是否包含有效的链 例中，rcac.pem 是根 CA 证书，icac.pem 是中间 CA 证 noc.pem 是节点证书。
		
		`openssl verify -verbose -CAfile <(cat rcac.pem icac noc.pem`
		
		2. 使用 chip-cert 将 PEM 格式的节点证书转换为 TLV（标签、 值）格式，然后再次转换回来。
		
		`./chip-cert convert-cert noc.pem noc.chip -c ./chip-cert convert-cert noc.chip noc_converted.pem`
		
		文件 noc.pem 和 noc_converted.pem 应与字符串比较 的完全相同。

使用 Kubernetes 保护 Amazon 私有证书颁发机构

您可以使用提供证书 Amazon 私有证书颁发机构 ，以便通过 TLS 和 mTLS 进行安全身份验证和加密。 Amazon 私有 CA 为 Kubernetes 广泛采用的证书管理器插件提供了一个开源插件，即 aws-privateca-issuer Kubernetes Amazon 私有 CA 连接器，用于请求证书、将其分发到 Kubernetes 密钥并自动续订证书。

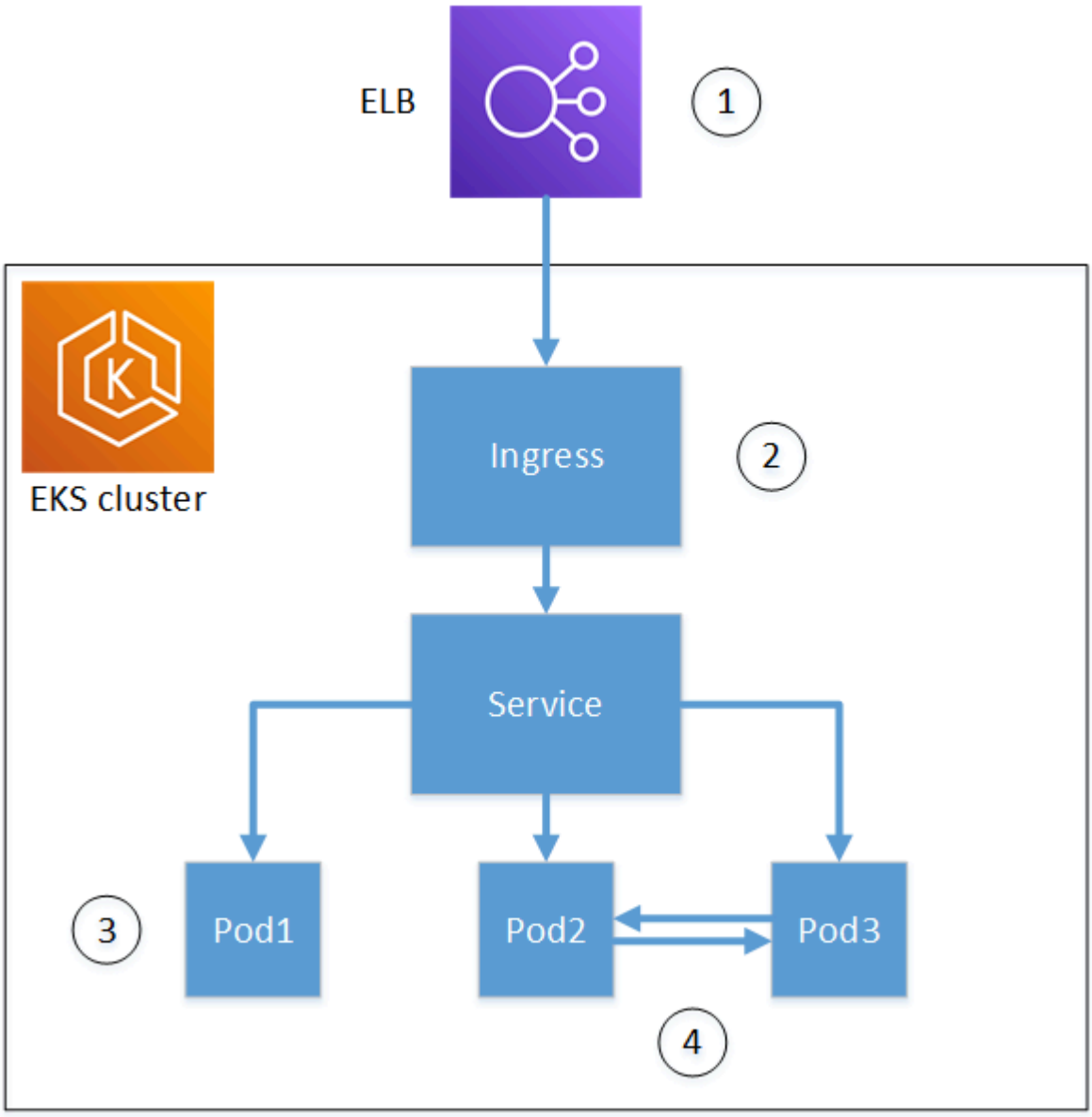
该aws-privateca-issuer插件允许您通过 Amazon 私有 CA 颁发证书cert-manager。你可以将该插件与亚马逊 Elastic Kubernetes Service（Amazon EKS）一起使用，该集群是一个自行管理的 Kubernetes 集群，也可以在本地 Kubernetes 集群 Amazon中使用。该插件同时适用于 x86 和 ARM 架构。

Amazon 私有 CA 有 HSM 支持的密钥无法导出。如果您有控制访问权限和审计 CA 运营的监管要求，则可以使用 Amazon 私有 CA 来提高可审计性并支持合规性。

> **Note**
>
> 如果您在 Amazon EKS 上运行，我们建议您使用cert-manager和aws-privateca-connector-for-kubernetes插件来获得托管安装体验。有关更多信息，请参阅Amazon 附加组件。

概念

下图显示了在 Amazon EKS 集群中使用 TLS 的一些可用选项。示例集群位于负载均衡器后面。这些数字标识了 TLS 安全通信的可能端点。



1. 在负载均衡器处终止

   Elastic Load Balancing（Elastic Load Balancing）已与该 Amazon Certificate Manager 服务集成。您无需在负载均衡器`cert-manager`上安装。您可以使用 Elastic Load Balancing 控制台为私有 CA 配置 ACM，使用私有 CA 签署证书，然后使用 Elastic Load Balancing 控制台安装证书。 Amazon 私有 CA 证书会自动续订。

   或者，您可以向非Amazon 负载均衡器提供私有证书以终止 TLS。

   这提供了远程客户端和负载均衡器之间的加密通信。负载均衡器之后的数据以未加密方式传递到 Amazon EKS 集群。

2. 在 Kubernetes 入口控制器处终止

   入口控制器位于 Amazon EKS 集群内，充当负载均衡器和路由器。要使用入口控制器作为集群的终端节点进行外部通信，您必须：

   - 同时安装`cert-manager`和 `aws-privateca-issuer`
   - 为控制器提供来自的 TLS 私有证书 Amazon 私有 CA。

   负载均衡器和入口控制器之间的通信已加密，数据未加密地传递到集群的资源。

3. 在 pod 处终止

   每个 Pod 是一组共享存储和网络资源的一个或多个容器。如果您同时安装了`cert-manager`和`aws-privateca-issuer`并为集群配置了私有 CA，Kubernetes 可以根据需要在 Pod 上安装签名的 TLS 私有证书。默认情况下，集群中的其他 Pod 无法使用在某个 Pod 上终止的 TLS 连接。

4. 容器组（pod）之间的安全通信。

   您可以为多个 Pod 配置证书，以允许它们相互通信。以下是可能的情况：

   - 使用 Kubernetes 进行配置会生成自签名证书。这可以保护 pod 之间的通信，但是自签名证书不满足 HIPAA 或 FIPS 的要求。
   - 使用由签名的证书进行配置 Amazon 私有 CA。这需要同时安装`cert-manager`和`aws-privateca-issuer`。然后，Kubernetes 可以根据需要在容器上安装签名的 mTLS 证书。

注意事项

Amazon 私有证书颁发机构 与 Kubernetes 一起使用时，请记住以下注意事项。

跨账户使用证书管理器

对 CA 具有跨账户访问权限的管理员可以使用 Kubernetes 的插件为使用共享 CA 的集群配置证书。`cert-manager`有关更多信息，请参阅Cross-account 访问私有 CA 的安全最佳实践。

在跨账户场景中，您只能使用某些 Amazon 私有 CA 证书模板。

下表列出了您可以与证书管理器一起使用来配置 Kubernetes 集群的 Amazon 私有 CA 模板。

Kubernetes 支持的模板	支持跨账户使用
BlankEndEntityCertificate_ CSRPassthrough / V1 的定义	否

Kubernetes 支持的模板	支持跨账户使用
CodeSigningCertificate/V1 定义	否
EndEntityCertificate/V1 定义	是
EndEntityClientAuthCertificate/V1 定义	是
EndEntityServerAuthCertificate/V1 定义	是
OCSPSigning证书/V1 定义	否

开始使用适用于 Kubernetes 的 Amazon 私有 CA 连接器。

以下主题介绍如何使用 Amazon 私有 CA 来保护 Kubernetes 集群中的通信。有关另一个示例，请参阅开启的 Kubernetes 传输中的加密。 GitHub

您可以使用私有证书颁发机构来保护与 Amazon EKS 集群的通信。在您开始之前，请确保您已拥有以下各项：

- 具有适当权限的 Amazon 账户，其范围仅限于您的安全策略。

  Amazon EKS clusters

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAM",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy",
        "iam:GetRole"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EKS",
      "Effect": "Allow",
```

```
      "Action": [
        "eks:CreateAddon",
        "eks:DescribeAddon",
        "eks:CreatePodIdentityAssociation",
        "eks:DescribeCluster"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::*:role/CertManagerPrivateCARole"
    }
  ]
}
```

Other clusters

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "GetAndIssuePCACertificates",
      "Effect": "Allow",
      "Action": [
        "acm-pca:GetCertificate",
        "acm-pca:IssueCertificate"
      ],
      "Resource": "*"
    },
    {
      "Sid": "RolesAnywhere",
      "Effect": "Allow",
      "Action": [
        "rolesanywhere:CreateProfile"
      ],
      "Resource": "*"
    },
    {
```

```
      "Sid": "IAM",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::*:role/CertManagerPrivateCARole"
    }
  ]
}
```

- 一个 Kubernetes 集群。要创建亚马逊 Elastic Kubernetes Service 集群，请参阅亚马逊 EKS 快速入门指南。为简单起见，创建一个环境变量来保存集群名称：

```
export CLUSTER=aws-privateca-demo
```

- 您的 Amazon Web Services 区域 CA 和 Amazon EKS 集群所在的位置。为简单起见，创建一个环境变量来保存区域：

```
export REGION=aws-region
```

- Amazon 私有 CA 私有证书颁发机构的亚马逊资源名称 (ARN)。为简单起见，创建一个环境变量来保存私有 CA ARN：

```
export CA_ARN="arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
```

  要创建私有 CA，请参阅中的https://docs.amazonaws.cn/privateca/latest/userguide/create-CA.html创建私有 CA Amazon 私有 CA

- 一台安装了以下软件的计算机：
  - Amazon CLI 已@@ 配置 v2
  - kubectl v1.13+

- 对于非亚马逊 EKS 集群，Helm v3

安装证书管理器

要使用私有 CA，必须安装请求证书、分发证书和自动续订证书的cert-manager>插件。您还必须安装允许您从中颁发私有证书的aws-private-ca-issuer插件 Amazon 私有 CA。使用以下步骤安装插件和插件。

Amazon EKS clusters

cert-manager作为 Amazon EKS 附加组件安装：

```
aws eks create-addon \
  --cluster-name $CLUSTER \
  --addon-name cert-manager \
  --region $REGION
```

Other clusters

cert-manager使用 Helm 进行安装：

```
helm repo add jetstack https://charts.jetstack.io
helm repo update

helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --set crds.enabled=true
```

配置 IAM 权限

该aws-privateca-issuer插件需要与之交互的权限 Amazon 私有 CA。对于 Amazon EKS 集群，您可以使用容器身份。对于您使用的其他集群 Amazon Identity and Access Management Roles Anywhere。

首先，创建一个 IAM 策略。该策略使用AWSPrivateCAConnectorForKubernetesPolicy托管策略。有关该策略的更多信息，请参阅AWSPrivateCAConnectorForKubernetesPolicy《Amazon 托管策略参考指南》。

Amazon EKS clusters

1. 创建一个名为的文件，trust-policy.json其中包含以下信任策略：

   JSON

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

2. 运行以下命令创建 IAM 角色：

```
ROLE_ARN=$(aws iam create-role \
  --role-name CertManagerPrivateCARole \
  --assume-role-policy-document file://trust-policy.json \
  --region $REGION \
  --output text \
  --query "Role.Arn")

 aws iam attach-role-policy \
  --role-name CertManagerPrivateCARole \
  --policy-arn arn:aws:iam::aws:policy/AWSPrivateCAConnectorForKubernetesPolicy
```

Other clusters

1. 创建信任存储在中的私有 CA 的信任锚CA_ARN。有关说明，请参阅入门 IAM Roles Anywhere。创建一个环境变量来存储信任锚点 ARN：

```
export TRUST_ANCHOR_ARN=trustAnchorArn
```

2. 创建一个名为的文件，trust-policy.json其中包含以下信任策略：

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "TrustPolicyForSelfManagedOrOnPremiseClusters",
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:SetSourceIdentity",
                "sts:TagSession"
            ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TRUST_ANCHOR_ARN"
                    ]
                },
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "aws-privateca-
issuer"
                }
            }
        }
    ]
}
```

3. 运行以下命令创建 IAM 角色：

```
ROLE_ARN=$(aws iam create-role \
  --role-name CertManagerPrivateCARole \
  --assume-role-policy-document file://trust-policy.json \
```

```
  --query "Role.Arn" \
  --region $REGION \
  --output text)

 aws iam attach-role-policy \
  --role-name CertManagerPrivateCARole \
  --region $REGION \
  --policy-arn arn:aws:iam::aws:policy/AWSPrivateCAConnectorForKubernetesPolicy
```

安装和配置 Amazon 私有 CA 集群发行者

要安装该aws-privateca-connector-for-kubernetes插件，请使用以下命令：

Amazon EKS clusters

创建插件：

```
aws eks create-addon --region $REGION \
  --cluster-name $CLUSTER \
  --addon-name aws-privateca-connector-for-kubernetes \
  --pod-identity-associations "[{
    \"serviceAccount\": \"aws-privateca-issuer\",
    \"roleArn\": \"$ROLE_ARN\"
  }]"
```

然后等待插件激活：

```
aws eks describe-addon \
  --cluster-name $CLUSTER \
  --addon-name aws-privateca-connector-for-kubernetes \
  --region $REGION \
  --query 'addon.status'
```

Other clusters

1. 在以下位置创建个人资料 IAM Roles Anywhere：

```
PROFILE_ARN=$(aws rolesanywhere create-profile \
  --name "privateca-profile" \
  --role-arns "$ROLE_ARN" \
  --region "$REGION" \
```

```
  --query 'profile.profileArn' \
  --enabled \
  --output text)
```

2. 生成客户端证书，用于 Kubernetes 的连接器使用，并通过以下方式 IAM Roles Anywhere 进行身份验证： Amazon 私有 CA

   a. 为客户证书生成私钥：

   ```
   openssl genrsa -out client.key 2048
   ```

   b. 为客户端证书生成证书签名请求 (CSR)：

   ```
   openssl req -new \
     -key client.key \
     -out client.csr \
     -subj "/CN=aws-privateca-issuer"
   ```

   c. 从 Amazon 私有 CA以下地址颁发客户证书：

   ```
   CERT_ARN=$(aws acm-pca issue-certificate \
     --signing-algorithm SHA256WITHRSA \
     --csr fileb://client.csr \
     --validity Value=1,Type=DAYS \
     --certificate-authority-arn "$CA_ARN" \
     --region "$REGION" \
     --query 'CertificateArn' \
     --output text)
   ```

   d. 将客户端证书存储在本地：

   ```
   aws acm-pca get-certificate \
     --certificate-authority-arn $CA_ARN \
     --certificate-arn $CERT_ARN \
     --region $REGION \
     --query 'Certificate'
     --output text > pca-issuer-client-cert.pem
   ```

3. 使用客户端证书在集群中安装 Amazon 私有 CA 发行者：

   a. 添加 awspca Helm 存储库：

   ```
   helm repo add awspca https://cert-manager.github.io/aws-privateca-issuer
   ```

```
helm repo update
```

b. 创建命名空间：

```
kubectl create namespace aws-privateca-issuer
```

c. 将之前创建的证书放入密钥中：

```
kubectl create secret tls aws-privateca-credentials \
  -n aws-privateca-issuer \
  --cert=pca-issuer-client-cert.pem \
  --key=client.key
```

4. 使用以下命令安装 Amazon 私有 CA 发行者 IAM Roles Anywhere：

a. 创建一个名为的文件`values.yaml`，将 Amazon 私有 CA 发行者插件配置为与以下内容一起使用 IAM Roles Anywhere：

```
cat > values.yaml <<EOF
env:
  AWS_EC2_METADATA_SERVICE_ENDPOINT: "http://127.0.0.1:9911"

extraContainers:
  - name: "rolesanywhere-credential-helper"
    image: "public.ecr.aws/rolesanywhere/credential-helper:latest"
    command: ["aws_signing_helper"]
    args:
      - "serve"
      - "--private-key"
      - "/etc/cert/tls.key"
      - "--certificate"
      - "/etc/cert/tls.crt"
      - "--role-arn"
      - "$ROLE_ARN"
      - "--profile-arn"
      - "$PROFILE_ARN"
      - "--trust-anchor-arn"
      - "$TRUST_ANCHOR_ARN"
    volumeMounts:
      - name: cert
        mountPath: /etc/cert/
        readOnly: true
```

```
volumes:
  - name: cert
    secret:
      secretName: aws-privateca-credentials
EOF
```

b. 使用以下命令安装 Amazon 私有 CA 发行者 IAM Roles Anywhere：

```
helm install aws-privateca-issuer awspca/aws-privateca-issuer \
  -n aws-privateca-issuer \
  -f values.yaml
```

等待发行人准备就绪。使用以下命令：

```
kubectl wait --for=condition=ready pods --all -n aws-privateca-issuer --timeout=120s
```

然后验证安装以确保所有 pod 都已达到READY状态：

```
kubectl -n aws-privateca-issuer get all
```

要配置aws-private-ca-cluster-issuer，请创建一个名为的 YAML 文件，cluster-issuer.yaml其中包含颁发者的配置：

```
cat > cluster-issuer.yaml <<EOF
apiVersion: awspca.cert-manager.io/v1beta1
kind: AWSPCAClusterIssuer
metadata:
  name: aws-privateca-cluster-issuer
spec:
  arn: "$CA_ARN"
  region: "$REGION"
EOF
```

接下来，应用集群配置：

```
kubectl apply -f cluster-issuer.yaml
```

检查发行人的状态：

```
kubectl describe awspcaclusterissuer aws-privateca-cluster-issuer
```

您可以看到类似以下内容的响应：

```
Status:
  Conditions:
    Last Transition Time:  2025-08-13T21:00:00Z
    Message:               AWS PCA Issuer is ready
    Reason:                Verified
    Status:                True
    Type:                  Ready
```

使用证书管理器管理 Amazon 私有 CA 客户证书

如果您未使用 Amazon EKS 集群，则在中手动引导可信证书后，aws-privateca-issuer您可以过渡到由cert-manager管理的客户端身份验证证书。这cert-manager允许自动续订客户端身份验证证书。

1. 创建一个名为pca-auth-cert.yaml：的文件

```
cat > pca-auth-cert.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: aws-privateca-client-cert
  namespace: aws-privateca-issuer
spec:
  secretName: aws-privateca-credentials
  duration: 168h
  renewBefore: 48h
  commonName: aws-privateca-issuer
  privateKey:
    algorithm: ECDSA
    size: 256
    rotationPolicy: Always
  usages:
    - client auth
  issuerRef:
    name: aws-privateca-cluster-issuer
    kind: AWSPCAClusterIssuer
    group: awspca.cert-manager.io
EOF
```

2. 创建新的托管客户端身份验证证书：

```
kubectl apply -f pca-auth-cert.yaml
```

3. 验证证书是否已创建：

```
kubectl get certificate aws-privateca-client-cert -n aws-privateca-issuer
```

您可以看到类似以下内容的响应：

```
NAME                         READY   SECRET                        AGE
aws-privateca-client-cert    True    aws-privateca-credentials     19m
```

颁发您的第一个 TLS 证书

现在cert-manager和已安装aws-privateca-issuer完毕，您可以颁发证书。

创建一个名为的 YAML 文件，certificate.yaml其中包含证书资源：

```
cat > certificate.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-certificate
  namespace: default
spec:
  secretName: example-certificate-tls
  issuerRef:
    name: aws-privateca-cluster-issuer
    kind: AWSPCAClusterIssuer
    group: awspca.cert-manager.io
  commonName: example.internal
  dnsNames:
    - example.internal
    - api.example.internal
  duration: 2160h # 90 days
  renewBefore: 360h # 15 days
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

使用以下命令应用证书：

```
kubectl apply -f certificate.yaml
```

然后，您可以使用以下命令检查证书的状态：

```
kubectl get certificate example-certificate
kubectl describe certificate example-certificate
```

您应该会看到类似于以下内容的响应：

```
NAME                  READY   SECRET                     AGE
example-certificate   True    example-certificate-tls    30s
```

您可以使用以下命令检查已颁发的证书：

```
kubectl get secret example-certificate-tls -o yaml
```

您也可以使用以下命令解码和检查证书：

```
kubectl get secret example-certificate-tls -o jsonpath='{.data.tls\.crt}' | base64 -d |
 openssl x509 -text -noout
```

示例

以下示例展示了 Amazon 私有 CA 如何与 Kubernetes 集群一起使用。

- 示例：Kubernetes 传输中的加密
- TLS-enabled Kubernetes clusters with Amazon 私有 CA and Amazon EKS
- 使用新的 end-to-end Loa Amazon d Balancer 控制器在 Amazon EKS 上设置 TLS 加密

使用监控 Kubernetes Amazon 私有 CA

要监控 Kubernetes 集群的私有 CA，请使用中概述的技术。监控 Amazon 私有 CA 资源您可以使用以下方法来监控私有 CA：

- Amazon 私有 CA CloudWatch 指标

- Amazon 私有 CA 使用 CloudWatch 事件进行监控
- 使用记录 Amazon 私有证书颁发机构 API 调用 Amazon CloudTrail

使用 Kubernetes 进行故障排除 Amazon 私有 CA

您可以按照以下步骤获取`aws-private-ca-issuer`日志：

1. 获取 pod 的名称：

```
kubectl get pods -A
```

2. 要查看发行者日志，请使用以下命令：

```
kubectl logs -n aws-privateca-issuer <pod-name> aws-privateca-issuer
```

3. 要查看日 IAM Roles Anywhere 志，请使用以下命令：

```
kubectl logs -n aws-privateca-issuer <pod-name> rolesanywhere-credentials-helper
```

要查看您的 Amazon 私有 CA 发卡机构的状态，请使用以下方法之一：

要检查您的发卡机构是否准备就绪，请使用以下命令：

```
kubectl get AWSPCAClusterIssuers -o json | jq '.items[].status
```

响应应类似于以下内容：

```
{
  "conditions": [
    {
      "lastTransitionTime": "2024-07-03T13:56:37Z",
      "message": "Issuer verified",
      "reason": "Verified",
      "status": "True",
      "type": "Ready"
    }
  ]
}
```

如果发行人不在该Ready州，则该message字段将提供有关发行人无法到达该Ready州的原因的信息。

要检查您的证书是否已准备就绪，请使用以下命令：

```
kubectl get certificates -o json | jq '.items[].status'
```

响应应类似于以下内容：

```
{
  "conditions": [
    {
      "lastTransitionTime": "2024-07-03T13:58:13Z",
      "message": "Certificate is up to date and has not expired",
      "observedGeneration": 1,
      "reason": "Ready",
      "status": "True",
      "type": "Ready"
    }
  ],
  "notAfter": "2024-10-01T13:58:12Z",
  "notBefore": "2024-07-03T12:58:12Z",
  "renewalTime": "2024-09-16T13:58:12Z",
  "revision": 1
}
```

如果证书不在该Ready州，则该message字段将提供有关证书无法到达该Ready州的原因的信息。

Amazon 私有 CA 活动目录连接器

Amazon 私有 CA 可以颁发和管理所需的证书 Amazon Managed Microsoft AD。使用适用于 Active Directory 的连接器（AD 连接器），您可以将本地企业或其他第三方 CAs 替换为您拥有的托管私有 CA，从而为由 AD 管理的用户、组和计算机提供证书注册。 Amazon 私有 CA

您可以使用适用于 AD 的连接器，通过将 AD 和公钥基础设施迁移到云端来消除本地基础架构。 Amazon Managed Microsoft AD 对于希望与本地 AD Amazon 私有 CA 一起使用的客户，此功能还与 Conn Amazon Managed Microsoft AD ector 集成。

主题

- 您是初次接触 Connector for AD 的用户吗？
- 设置 Connector for AD
- 开始使用活动目录 Amazon 私有 CA 连接器
- Amazon 私有 CA 活动目录的连接器
- 使用 Amazon 将适用于 AD 的 Connector 集成到事件驱动的应用程序中 EventBridge
- 解决活动目录 Amazon 私有 CA 连接器的问题

您是初次接触 Connector for AD 的用户吗？

如果您是首次接触 Connector for AD 的用户，则建议您先阅读以下部分：

- 什么是 Amazon 私有 CA？
- 什么是 Amazon Directory Service？

AD 接入连接器

您可以通过控制台、 Amazon CLI和访问适用于 AD 的连接器 APIs。您可以通过控制 Amazon 私有 CA 台、您的主机或在搜索栏中搜索 Connector for AD 来访问 Amazon Directory Service 控制台中的 Amazon Web Services 管理控制台 连接器。

定价

AD 连接器作为一项功能提供 Amazon 私有 CA ，无需额外付费。您只需为私有证书颁发机构以及通过这些机构颁发的证书付费。

有关最新的定 Amazon 私有 CA 价信息，请参阅Amazon 私有证书颁发机构 定价。您也可以使用定 Amazon 价计算器来估算成本。

设置 Connector for AD

本节中的步骤是使用适用于 AD 的连接器的先决条件。它假设你已经创建了一个 Amazon 账户。完成本页上的步骤后，就可以开始为 AD 创建连接器了。

步骤 1：使用创建私有 CA Amazon 私有 CA

设置私有证书颁发机构 (CA)，用于向目录对象颁发证书。有关更多信息，请参阅 中的证书颁发机构 Amazon 私有 CA。

私有 CA 必须处于`Active`状态才能为 AD 创建连接器。私有 CA 的使用者名称必须包含公用名。如果您尝试使用不带公用名的私有 CA 创建连接器，则连接器创建将失败。

步骤 2：设置活动目录

除了私有 CA 之外，您还需要虚拟私有云 (VPC) 中的活动目录。Connector for AD 支持由 Amazon Directory Service提供的以下目录类型：

- Amazon 托管微软 Active Direct ory：有了它， Amazon Directory Service 你可以将微软活动目录 (AD) 作为托管服务运行。 Amazon Directory Service for Microsoft Active Directory 也称为 Amazon Managed Microsoft AD，由 Windows Server 2019 提供支持。使用 Amazon Managed Microsoft AD，你可以在中运行目录感知型工作负载，包括微软 Sharepoint 以及基于.Net 和 SQL Server 的自定义应用程序。 Amazon Web Services 云
- Active Directory Connector：AD Connector 是一种目录网关，可以将目录请求重定向到本地 Microsoft Active Directory，而无需在云中缓存任何信息。AD Connector 支持连接到 Amazon EC2 上托管的域

（仅限 Active Directory 连接器）第 3 步：将权限委托给服务帐户

> **Note**
>
> 如果您正在使用，则当您在目录 Amazon Managed Microsoft AD 中授权 Connector for AD 服务时，系统会自动委派其他权限。您可以跳过此先决条件步骤。

使用 Directory Service AD Connector 时，您需要向服务账户委派其他权限。在服务账户上设置访问控制列表（ACL）以允许以下功能：

- 向其自身添加和删除服务主体名称（SPN）
- 在以下容器中创建和更新证书颁发机构：

```
#containers
CN=Public Key Services,CN=Services,CN=Configuration
CN=AIA,CN=Public Key Services,CN=Services,CN=Configuration
CN=Certification Authorities,CN=Public Key Services,CN=Services,CN=Configuration
```

- 创建和更新 NTAuth证书颁发机构 (CA) 对象。注意：如果存在 Cer NTAuth tificates CA 对象，则必须为其委派权限。如果对象不存在，则必须委派在公钥服务容器上创建子对象的权限。

```
#objects
CN=NTAuthCertificates,CN=Public Key Services,CN=Services,CN=Configuration
```

官方 Connector fo r Active Directory 存储库中提供的 PowerShell 脚本可用于委派目录服务 AD Connector 服务帐户所需的额外权限。

此脚本创建 NTAuth证书证书颁发机构对象。

有关最新版本的脚本和用法详情，请参阅GitHub 存储库中的自述文件。

步骤 4：创建 IAM 策略

要为 AD 创建连接器，您需要一个 IAM policy，该策略允许您创建连接器资源，与 Connector for AD 服务共享您的私有 CA，并使用您的目录中授权 Connector for AD 服务。

以下是用户托管策略的示例：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "pca-connector-ad:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListCertificateAuthorities",
                "acm-pca:ListTags",
                "acm-pca:PutPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "acm-pca:IssueCertificate",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V*"
                },
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "pca-connector-ad.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ds:AuthorizeApplication",
                "ds:DescribeDirectories",
                "ds:ListTagsForResource",
                "ds:UnauthorizeApplication",
                "ds:UpdateAuthorizedApplication"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint",
```

```
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcs",
                "ec2:DeleteVpcEndpoints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeTags",
                "ec2:DeleteTags",
                "ec2:CreateTags"
            ],
            "Resource": "arn:*:ec2:*:*:vpc-endpoint/*"
        }
    ]
}
```

AD 连接器需要额外的 Amazon RAM 权限，才能使用控制台和命令行。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ram:CreateResourceShare",
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "ram:Principal": "pca-connector-ad.amazonaws.com",
                    "ram:RequestedResourceType": "acm-pca:CertificateAuthority"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ram:GetResourcePolicies",
```

```
                "ram:GetResourceShareAssociations",
                "ram:GetResourceShares",
                "ram:ListPrincipals",
                "ram:ListResources",
                "ram:ListResourceSharePermissions",
                "ram:ListResourceTypes"
            ],
            "Resource": "*"
        }
    ]
}
```

第 5 步：与 Connector for AD 共享您的私有 CA

您需要使用 Amazon Resource Access Manager 服务主体共享与连接器服务共享您的私有 CA。

在 Amazon 控制台中创建连接器时，系统会自动为您创建资源共享。

使用创建资源共享时 Amazon CLI，将使用 Amazon RAM create-resource-share命令。

以下命令创建资源共享：

```
$  aws ram create-resource-share \
    --region us-east-1 \
    --name MyPcaConnectorAdResourceShare \
    --permission-arns arn:aws:ram::aws:permission/
AWSRAMBlankEndEntityCertificateAPIPassthroughIssuanceCertificateAuthority \
    --resource-arns arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
    --principals pca-connector-ad.amazonaws.com \
    --sources account
```

调用的服务主体在 PCA 上 CreateConnector 拥有证书颁发权限。要防止使用 Connector for AD 的服务主体拥有对您的 Amazon 私有 CA 资源的常规访问权限，请使用 `CalledVia` 限制其权限。

步骤 6：创建目录注册

您授权 Connector for AD 服务使用您的目录，以便连接器可以与您的目录通信。要授权 Connector for AD 服务，您需要创建目录注册。有关创建目录注册的更多信息，请参阅 管理目录注册

步骤 7：配置安全组

您的 VPC 与 AD 连接器之间的通信是通过的 Amazon PrivateLink，这需要一个或多个安全组，其入站规则可在您的 VPC 上打开 443 TCP 端口。当您创建连接器时，系统会要求您输入此安全组。您可以将源指定为自定义，然后选择 VPC 的 CIDR 块。您可以选择进一步限制此项（即 IP、CIDR 和安全组 ID）。

步骤 8：为目录对象配置网络访问权限

目录对象需要公共 Internet 访问权限才能验证来自以下域的在线证书状态协议 (OCSP) 和证书吊销列表 (CRLs)：

```
*.windowsupdate.com
*.amazontrust.com
```

所需的最低访问规则：

- OCSP 和 CRL 通信所必需的：

  ```
  TCP 80: (HTTP) to 0.0.0.0/0
  ```

- 对于 AD 的连接器是必需的：

  ```
  TCP 443: (HTTPS) to 0.0.0.0/0
  ```

- 活动目录必填项：

  ```
  TCP 88: (Kerberos) to Domain Controller IP range
  TCP/UDP 389/636: (LDAP/LDAPS) to Domain Controller IP range, depending on Domain
   Controller configuration
  TCP/UDP 53: (DNS) to 0.0.0.0/0
  ```

如果设备无法访问公共互联网，则证书颁发将间歇性失败，并显示错误代码 `WS_E_OPERATION_TIMED_OUT`.

> **Note**
>
> 如果您正在为 Amazon EC2 实例配置安全组，则该安全组不必与步骤 7 中的安全组相同。

开始使用活动目录 Amazon 私有 CA 连接器

使用 Active Directory Amazon 私有 CA 连接器，您可以将私有 CA 中的证书颁发给您的 Active Directory 对象进行身份验证和加密。创建连接器时， Amazon 私有证书颁发机构 会在您的 VPC 中为您创建一个端点，以便您的目录对象请求证书。

要颁发证书，您需要创建连接器以及该连接器的 AD 兼容模板。创建模板时，您可以设置 AD 组的注册权限。

主题

- 开始前的准备工作
- 步骤 1：创建连接器
- 步骤 2：配置微软 Active Directory 策略
- 步骤 3：创建模板
- 步骤 4：配置微软群组权限

开始前的准备工作

以下教程将指导您完成为 AD 创建连接器和连接器模板的过程。要学习本教程，您必须首先满足本节中列出的先决条件。

步骤 1：创建连接器

要创建连接器，请参阅为活动目录创建连接器。

步骤 2：配置微软 Active Directory 策略

Connector for AD 无法查看或管理客户的组策略对象（GPO）配置。GPO 控制向客户或其他身份验证 Amazon 私有 CA 或证书自动售卖服务器发送 AD 请求的路由。无效的 GPO 配置可能会导致您的请求路由不正确。由客户来配置和测试 Connector for AD 配置。

组策略与连接器关联，您可以选择为单个 AD 创建多个连接器。如果每个连接器的组策略配置不同，则由您来管理对每个连接器的访问控制。

数据面板调用的安全性取决于 Kerberos 和您的 VPC 配置。只要通过相应 AD 的身份验证，有权访问 VPC 的任何人都可以进行数据面板调用。这存在于边界之外，管理授权和身份验证由您（客户）决定。 AWSAuth

使用时 Amazon Managed Microsoft AD，使用 Amazon Directory Service enable-ca-enrollment-policy命令在 Amazon Managed Microsoft AD 实例的域控制器 GPOs 上进行配置。

以下命令允许注册域控制器：

```
$  aws ds enable-ca-enrollment-policy \
    --pca-connector-arn MyPcaConnectorAdArn \
    --directory-id MyDirectoryId
```

使用 AD Connector 时，请按照以下步骤创建指向创建连接器时生成的 URI 的 GPO。要通过控制台或命令行使用 Connector for AD，需要执行此步骤。

配置 GPOs。

1. 在 DC 上打开服务器管理器
2. 转到工具，然后选择控制台右上角的组策略管理。
3. 转到林 > 域。选择您的域名，然后右键单击您的域。选择在此域中创建 GPO，并将其链接到此处...，然后输入 PCA GPO 的名称。
4. 现在，新创建的 GPO 将在您的域名下列出。
5. 选择 PCA GPO，然后选择编辑。如果打开的对话框显示警报消息这是一个链接，更改将在全球范围内传播，请确认该消息以继续。组策略管理编辑器应打开。
6. 在组策略管理编辑器中，转到计算机配置 > 策略 > Windows 设置 > 安全设置 > 公有密钥策略（选择文件夹）。
7. 转到对象类型并选择证书服务客户端 – 证书注册策略
8. 在选项中，将配置模型更改为启用。
9. 确认已选中并启用 Active Directory 注册策略。选择添加。
10. 此时应打开证书注册策略服务器窗口。
11. 在输入注册服务器策略 URI 字段中输入创建连接器时生成的证书注册策略服务器端点。
12. 将身份验证类型保留为 Windows 集成。
13. 选择验证。验证成功后，选择添加。对话框关闭。
14. 返回证书服务客户端 – 证书注册策略并选中新创建的连接器旁边的复选框以确保连接器为默认注册策略
15. 选择 Active Directory 注册策略，然后选择删除。
16. 在确认对话框中，选择是以删除基于 LDAP 的身份验证。
17. 在证书服务客户端>证书注册策略窗口中选择应用和确定，然后将其关闭。

18. 转到公有密钥策略文件夹，然后选择证书服务客户端 – 自动注册。
19. 将配置模型选项更改为启用。
20. 确认续订过期的证书和更新证书均已选中。保持其他设置不变。
21. 选择 “应用”，然后选择 “确定”，然后关闭对话框。

接下来，配置用户配置的公有密钥策略。转到用户配置 > 策略 > Windows 设置 > 安全设置 > 公有密钥策略。按照步骤 6 到步骤 21 中概述的步骤配置用户配置的公有密钥策略。

配置 GPOs 完公钥策略后，域中的对象将从 Conn Amazon 私有 CA ector for AD 请求证书，并获得由颁发的证书 Amazon 私有 CA。

步骤 3：创建模板

要创建模板，请参阅创建连接器模板。

步骤 4：配置微软群组权限

要配置 Microsoft 群组权限，请参阅管理 AD 模板访问控制条目的连接器。

Amazon 私有 CA 活动目录的连接器

本节中的过程介绍如何创建 Active Directory (AD) 连接器、配置模板以及如何与 Amazon 私有 CA Active Directory 集成。您可以从 AD Amazon 私有 CA 连接器控制台、使用的 AD 连接器部分或使用适用于 AD 的 Amazon CLI Amazon 私有 CA 连接器 API 执行这些操作。

> **Note**
>
> 尽管 AD Amazon 私有 CA 连接器与 Connector 紧密集成 Amazon 私有 CA，但这两个服务是分开 APIs的。有关更多信息，请参阅《Amazon 私有证书颁发机构 API 参考》https://docs.amazonaws.cn/privateca/latest/APIReference/和《Amazon 私有 CA Connector for Active Directory API 参考》https://docs.amazonaws.cn/pca-connector-ad/latest/APIReference/。

为活动目录创建连接器

使用以下过程使用控制台、命令行或 Active Directory Amazon 私有 CA 连接器的 API 创建连接器。

Console

使用控制台创建连接器

登录您的 Amazon 账户，然后打开 Active Directory Amazon 私有 CA 连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-ad/home**。

1. 在首次服务登录页面或适用于 Active Directory 的连接器页面上，选择创建连接器。
2. 在创建 Private CA Connector for Active Directory 页面的 Active Directory 部分中提供信息。
   - 在选择您的 Active Directory 类型下，选择两种可用类型之一：
     - Amazon Directory Service for Microsoft Active Directory— 指定由管理的活动目录 Amazon Directory Service。
     - 带有 Amazon AD Connector 的本地 Active Directory – 使用 AD Connector 访问您在本地托管的 Active Directory。
   - 在选择您的目录下，从列表中选择您的目录。

     或者，您可以选择 “创建目录”，这将在新窗口中打开 Amazon Directory Service 控制台。创建完新目录后，返回 Active Directory Amazon 私有 CA 连接器控制台并刷新目录列表。您的新目录应该可供选择。

     > **Note**
     >
     > 创建目录时，请注意，Connector for AD 仅支持 Amazon Directory Service 控制台中提供的以下目录类型：
     >
     > - Amazon 微软 AD 托管
     > - AD Connector

   - 在为 VPC 端点选择安全组下，从列表中选择一个安全组。

     或者，您可以选择 “创建安全组”，这将在新窗口中打开 Amazon EC2 控制台的 “创建安全组” 页面。创建完安全组后，返回 Active Directory Amazon 私有 CA 连接器控制台并刷新安全组列表。您的新安全组应该可供选择。
3. 在 IP 地址类型部分，从以下选项中进行选择：
   - IPv4-启用与服务的 IPv4 连接。只有当托管您的目录的所有子网都有 IPv4 地址范围时，才选择此选项。

- Dualstack-同时启用两者 IPv4 并 IPv6 连接到服务。只有当托管您的目录的所有子网同时具有 IPv4 和 IPv6 地址范围时，才选择此选项。

4. 在私有证书颁发机构部分，从列表中选择一个私有 CA。

   或者，您可以选择 “创建私有 CA”，这将在新窗口中打开私有证书颁发机构页面的 Amazon 私有 CA 控制台。完成创建 CA 后，返回 Active Directory Amazon 私有 CA 连接器控制台并刷新列表 CAs。您的新 CA 应该可供选择。

5. 在标签：可选窗格中，您可以在 AD 资源上应用和移除元数据。标签是键值字符串对，其中键对于资源必须是唯一的，而值是可选的。该窗格在表中显示资源的任何现有标签。支持以下操作。

   - 选择管理标签以打开管理标签页面。
   - 选择“添加新标签”以创建标签。填写键字段和（可选）值字段。选择保存更改以应用标签。
   - 选择标签旁边的删除按钮将其标记为删除，然后选择保存更改进行确认。

6. 提供所需信息并检查您的选择后，选择创建连接器。这将打开适用于 Active Directory 的连接器详细信息页面，可在其中查看连接器的创建进度。

   创建连接器的过程完成后，为其分配服务主体名称。

API

使用 API 创建连接器

要使用 API 为 Active Directory 创建连接器，请使用 Active Directory Amazon 私有 CA 连接器 API 中的 CreateConnector操作。

CLI

要使用创建连接器 Amazon CLI

要使用 CLI 为 Active Directory 创建连接器，请使用 Active D irectory 连接器部分的 Amazon 私有 CA create- connector 命令。 Amazon CLI

创建连接器模板

模板是证书颁发后的外观以及客户端应如何处理证书的配置列表。以下过程说明了如何创建模板。

Console

使用控制台创建模板

1. 登录您的 Amazon 账户，然后打开 Active Directory Amazon 私有 CA 连接器控制台，网址为https://console.aws.amazon.com/pca-connector-ad/home。
2. 从适用于 Active Directory 的连接器列表中选择一个连接器，然后选择查看详细信息。
3. 在连接器的详细信息页面上，找到模板部分，然后选择创建模板。
4. 在创建模板页面的模板创建方法部分，选择其中一个方法选项。
   - 从预定义的模板开始（默认值）– 从 AD 应用程序的预定义模板列表中进行选择：
     - 代码签名
     - 计算机
     - 域控制器身份验证
     - EFS 恢复代理
     - 注册代理
     - 注册代理（计算机）
     - IPSec
     - Kerberos 身份验证
     - RAS 和 IAS 服务器
     - 智能卡登录
     - 信任列表签名
     - 用户签名
     - 工作站身份验证
   - 从您创建的现有模板开始 – 从您之前创建的自定义模板列表中进行选择。
   - 从空白模板开始 – 选择此选项可开始创建全新的模板。
5. 在证书设置部分，根据此模板定义证书的以下设置。
   - 证书类型 – 指定是创建用户证书还是计算机证书。
   - 自动注册 – 根据此模板选择是否激活证书的自动注册。
   - 有效期 – 将证书有效期指定为小时、天、周、月或年的整数值。最小值为 2 小时。
   - 续订期限 – 将证书续订期限指定为小时、天、周、月或年的整数值。续订期限不得超过有效期的 75%。

- 使用者名称 – 根据 Active Directory 中包含的信息，选择要包含在使用者名称中的一个或多个选项。

  > **Note**
  >
  > 必须至少指定一个使用者名称或使用者备用名称选项。

  - 公用名
  - 将 DNS 作为公用名
  - 目录路径
  - 电子邮件
- 使用者备用名称 – 根据 Active Directory 中包含的信息，选择要包含在使用者备用名称中的一个或多个选项。

  > **Note**
  >
  > 必须至少指定一个使用者名称或使用者备用名称选项。

  - 目录 GUID
  - DNS 名称
  - 域 DNS
  - 电子邮件
  - 服务主体名称（SPN）
  - 用户主体名称（UPN）

6. 在证书请求处理和注册选项部分，根据模板指定证书的用途，选择以下选项之一。

   - 签名
   - 加密
   - 签名和加密
   - 签名和智能卡登录

   接下来，选择要激活以下哪些功能。选项因证书用途而有所不同。

- 删除无效的证书（不存档）
- 包括对称算法
- 可导出的私有密钥

最后，选择证书注册选项。选项因证书用途而有所不同。

- 无需用户输入
- 在注册期间提示用户
- 在注册期间提示用户并需要用户输入

7. 在应用程序策略部分，选择所有适用的应用程序策略。可用策略在多个页面中列出。某些策略可能是由于之前的设置而预先选择的。
8. 在自定义应用程序策略部分，您可以 OIDs 向模板添加自定义策略，并指定应用程序策略扩展是否重要。
9. 在加密设置部分，根据此模板为证书选择以下类别的加密设置。
10. 在组和权限部分，您可以查看模板现有组和注册权限，也可以选择添加新的组和权限按钮来添加新的组和权限。该按钮将打开一个需要以下信息的表单：

    - 显示名称
    - 安全标识符（SID）
    - 注册，选项为：允许 | 拒绝 | 未设置
    - 自动注册，选项为：允许 | 拒绝 | 未设置

11. 在替代模板部分中，您可以通知 Active Directory 当前模板取代在 AD 中创建的一个或多个模板。通过选择添加 Active Directory 模板以取代并指定取代模板的通用名称来应用取代模板。
12. 在标签：可选窗格中，您可以在 AD 资源上应用和移除元数据。标签是键值字符串对，其中键对于资源必须是唯一的，而值是可选的。该窗格在表中显示资源的任何现有标签。支持以下操作。

    - 选择管理标签以打开管理标签页面。
    - 选择“添加新标签”以创建标签。填写键字段和（可选）值字段。选择保存更改以应用标签。
    - 选择标签旁边的删除按钮将其标记为删除，然后选择保存更改进行确认。

13. 提供所需信息并检查您的选择后，选择创建模板。这将打开模板详细信息，您可以在其中查看新模板的设置、编辑或删除模板、管理组和权限、管理被取代的模板、管理标签以及为证书持有者设置自动重新注册。

API

使用 API 创建连接器模板

使用 Active Directory Amazon 私有 CA 连接器 API 中的 CreateTemplate操作。

CLI

要使用创建连接器模板 Amazon CLI

在 Activ e Directory 的 Amazon 私有 CA 连接器部分使用创建模板命令。 Amazon CLI

更新活动目录的模板

使用以下过程使用控制台、命令行或 Active Directory Amazon 私有 CA 连接器的 API 更新模板。

Console

使用控制台更新模板

登录您的 Amazon 账户，然后打开 Active Directory Amazon 私有 CA 连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-ad/home**。

1. 在 A ctive Directory 的连接器列表中，选择要更新其模板的连接器。选择 “编辑” 以查看和修改连接器的模板。
2. 在连接器的模板详细信息页面中，选择编辑。按照提示进行更新。编辑完区域后，选择 “保存” 以保存所做的更改。

API

使用 API 更新模板

要使用 API 更新 Active Directory 的模板，请使用 Active Directory Amazon 私有 CA 连接器 API 中的UpdateTemplate操作。

CLI

要使用更新模板 Amazon CLI

要使用 CLI 更新 Active Directory 的连接器，请使用 Active Directory Amazon 私有 CA 连接器部分的 update-tem plate 命令。 Amazon CLI

Active Directory 连接器如何传播您的模板更改

Amazon 私有 CA 当您的客户端刷新策略缓存（每八小时刷新一次）时，将模板应用于您的策略。这包括对模板组访问控制条目的更改。当您的客户端刷新缓存时，它会向连接器查询可用模板。如果是自动注册刷新，则客户端会颁发符合以下任一或两个条件的证书：

- 证书在续订期内。
- 证书不存在于客户端设备上。

要进行手动刷新，客户端将查询连接器，并且您必须将模板设置为发布。

如果您正在调试，则可以手动清除策略缓存以立即查看模板的更改。为此，请在您的客户端上运行以下 Powershell 命令。

```
certutil -f -user -policyserver * -policycache delete
```

列出活动目录的连接器

您可以使用 Active Directory Amazon 私有 CA 连接器控制台或 Amazon CLI 列出您拥有的连接器。

Console

使用控制台列出连接器

1. 登录您的 Amazon 账户，然后打开 Active Directory Amazon 私有 CA 连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 查看适用于 Active Directory 的连接器列表中的信息。您可以使用右上角的页码浏览多页连接器。默认情况下，每个连接器占据一行，显示以下列的信息。

- 连接器 ID – 连接器的唯一 ID。
- 目录名称 – 与连接器关联的 Active Directory 资源。
- 连接器状态 – 连接器状态。可能的值为：正在创建 | 活动 | 正在删除 | 失败。
- 服务主体名称状态 – 与连接器关联的服务主体名称（SPN）的状态。可能的值为：正在创建 | 活动 | 正在删除 | 失败。
- 目录注册状态 – 关联目录的注册状态。可能的值为：正在创建 | 活动 | 正在删除 | 失败。
- 创建时间 – 连接器创建时的时间戳。

通过选择控制台右上角的齿轮图标，您可以使用页面大小首选项来自定义页面上显示的连接器数量。

API

使用 API 列出连接器

使用 Active Directory Amazon 私有 CA 连接器 API 中的ListConnectors操作。

CLI

要列出您的连接器，请使用 Amazon CLI

使用 list-connectors 命令列出您的连接器。

列出连接器模板

您可以使用 Active Directory 的 Amazon 私有 CA 连接器控制台或 Amazon CLI 列出您拥有的连接器的模板。连接器模板基于 Amazon 私有 CA BlankEndEntityCertificate_ APIPassthrough /V1 模板。

Console

使用控制台列出模板

1. 登录您的 Amazon 账户，然后打开 Active Directory Amazon 私有 CA 连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 从适用于 Active Directory 的连接器列表中选择一个连接器，然后选择查看详细信息。
3. 在连接器详细信息页面上，查看模板部分中的信息。您可以使用右上角的页码浏览多页模板。每个模板占据一行，显示以下列的信息。

- 模板名称 – 用户可读的模板名称。
- 模板状态 – 模板的状态。可能的值为：活动 | 正在删除。
- 模板 ID – 模板的唯一标识符。

API

使用 API 列出连接器

使用 Active Directory Amazon 私有 CA 连接器 API 中的 ListTemplates操作列出指定连接器的模板。

CLI

要列出您的连接器，请使用 Amazon CLI

使用 list-templates 命令列出指定连接器的模板。

查看连接器详细信息

使用以下过程在控制台、命令行或 Active Directory 连接器的 API 中查看 Amazon 私有 CA 连接器的配置详细信息。

Console

使用控制台查看连接器的详细信息

1. 登录您的 Amazon 账户，然后打开 Active Directory Amazon 私有 CA 连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 从适用于 Active Directory 的连接器列表中选择一个连接器，然后选择查看详细信息。
3. 在连接器详细信息页面上，查看“连接器详细信息”窗格中的信息，其中包含以下内容：
   - 连接器 ID
   - 连接器状态
   - 其他状态详细信息
   - 连接器 ARN
   - 证书注册策略服务器端点
   - 目录名称
   - 目录 ID
   - Amazon 私有 CA 主题
   - Amazon 私有 CA 状态
   - IP 地址类型
   - VPC 端点和安全组
4. 在模板窗格中，您可以创建或管理与连接器关联的模板。
5. 在服务主体名称（SPN）窗格中，您可以查看与连接器关联的服务主体名称。
6. 在目录注册窗格中，您可以查看或更改与连接器关联的目录注册。
7. 在标签：可选窗格中，您可以创建或管理与连接器关联的标签。

API

使用 API 列出连接器

使用 Active Directory Amazon 私有 CA 连接器 API 中的GetConnector操作。

CLI

要列出您的连接器，请使用 Amazon CLI

在 Active Direc tory 的连接 Amazon 私有 CA 器部分中使用 get- connector 命令。 Amazon CLI

查看连接器模板的详细信息

使用控制台、命令行或 Active Directory 连接器的 API，使用以下过程查看 Amazon 私有 CA 连接器模板的配置详细信息

Console

使用控制台查看连接器模板的详细信息

1. 登录您的 Amazon 账户，然后打开 Active Directory Amazon 私有 CA 连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 从适用于 Active Directory 的连接器列表中选择一个连接器，然后选择查看详细信息。
3. 在连接器详细信息页面上，查看模板部分中的信息，然后选择要检查的模板。请选择查看详细信息。
4. 在详细信息页面上，模板详细信息窗格显示有关该模板的以下信息：
   - 模板名称
   - 模板 ID
   - 模板状态
   - 模板架构版本
   - 模板版本
   - 模板 ARN
   - 证书类型
   - 自动注册已开启
   - 有效期
   - 续订期限

- 使用者名称要求
- 使用者备用名称要求
- 证书申请和注册设置
- 加密提供程序类别
- 密钥算法
- 最小密钥大小（位）
- 哈希算法
- 加密提供程序
- 密钥用法扩展设置

在此窗格中，您还可以使用编辑、删除和操作按钮执行以下操作。

- 编辑
- 删除
- 管理组和权限 – 有关更多信息，请参阅配置组和权限。
- 管理被取代的模板 – 有关更多信息，请参阅查看并创建。
- 管理标签 – 有关更多信息，请参阅 标记适用于 AD 的连接器资源。
- 重新注册所有证书持有者 – 此设置允许自动增加模板的主要版本。允许使用模板注册的 Active Directory 组的所有成员都将收到使用该模板颁发的新证书。有关更多信息，请参阅 UpdateTemplate API。

5. 下方窗格显示一行选项卡，允许更改模板的配置。

- 组和权限 – 查看和管理 Active Directory 组使用此模板注册证书的权限。有关更多信息，请参阅配置组和权限
- 应用程序策略 – 查看和管理模板应用程序策略。有关更多信息，请参阅分配应用程序策略。
- 被取代的模板 – 查看和管理被取代的模板。有关更多信息，请参阅查看并创建。
- 标签可选 – 查看和管理此模板上的标记。有关更多信息，请参阅 标记适用于 AD 的连接器资源。

API

使用 API 列出连接器

使用 Active Directory Amazon 私有 CA 连接器 API 中的 GetTemplate操作。

CLI

要列出您的连接器，请使用 Amazon CLI

在 Ac tive Directory 的 Amazon 私有 CA 连接器部分使用获取模板命令。 Amazon CLI

管理目录注册

Console

使用控制台管理目录注册

可以从 Amazon 私有 CA Connector for Active Directory 控制台的顶层管理连接器的目录注册。本主题将介绍可用的管理选项。

1. 登录您的 Amazon 账户，然后打开 Active Directory Amazon 私有 CA 连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 在左侧导航区域中，选择目录注册。
3. 目录注册页面显示包含以下字段的注册目录表：

   - 目录 ID – 目录的唯一 ID
   - 目录名称 – 目录域站点名称
   - 目录类型
   - 已注册 – 注册的状态。支持的值为：正在创建 | 活动 | 正在删除 | 失败。
   - 目录状态 – 目录的状态

   用户可以使用注册目录创建新的注册。
4. 您可以选择列出的注册之一进行管理。这将启用查看注册详细信息和取消注册目录按钮。使用查看注册详细信息按钮可打开注册的详细信息页面。
5. 目录注册详细信息窗格，其中显示以下信息：

   - 目录域站点名称
   - 目录 ID – 目录的唯一 ID。选择该链接会将您带到 Amazon Directory Service 控制台。
   - 目录类型
   - 状态 – 目录的状态
   - 目录注册 ARN – 目录注册的 Amazon 资源名称

- 其他状态信息

6. 在 “连接器和服务主体名称” (SPNs) 窗格中，您可以管理 SPNs 连接器。有关更多信息，请参阅查看连接器详细信息。
7. 在标签：可选窗格中，您可以在 AD 资源上应用和移除元数据。标签是键值字符串对，其中键对于资源必须是唯一的，而值是可选的。该窗格在表中显示资源的任何现有标签。支持以下操作。

   - 选择管理标签以打开管理标签页面。
   - 选择“添加新标签”以创建标签。填写键字段和（可选）值字段。选择保存更改以应用标签。
   - 选择标签旁边的删除按钮将其标记为删除，然后选择保存更改进行确认。

API

使用 API 管理目录注册

创建：Active Directory API Amazon 私有 CA 连接器中的CreateDirectoryRegistration操作。

检索：Active Directory API Amazon 私有 CA 连接器中的GetDirectoryRegistration操作。

列表：Active Directory API Amazon 私有 CA 连接器中的ListDirectoryRegistrations操作。

删除：Active Directory API Amazon 私有 CA 连接器中的DeleteDirectoryRegistration操作。

CLI

使用 CLI 管理目录注册

创建：使用 Active Directory Amazon 私有 CA 连接器部分中的create-directory-registration命令 Amazon CLI。

“检索：” 的 “活动目录 Amazon 私有 CA 连接器” 部分中的get-directory-registration命令 Amazon CLI。

列表：位于 “活动目录 Amazon 私有 CA 连接器” 部分中的list-directory-registrations命令 Amazon CLI。

删除：位于 “活动目录 Amazon 私有 CA 连接器” 部分中的delete-directory-registration命令 Amazon CLI。

管理 AD 模板访问控制条目的连接器

访问控制条目允许控制哪些 Active Directory 组可以或不能为特定的 Connector for AD 模板注册证书。当您可以在 Connector for AD 中创建或管理组和权限时，必须提供 Active Directory 中组对象的安全标识符 (SID)。您可以使用以下 PowerShell 命令获取 SID。有关信息 SIDs，请参阅 Microsoft 目录域服务文档中的安全标识符的工作原理。

```
$ Get-ADGroup -Identity "my_active_directory_group_name"
```

以下过程说明如何为 AD 模板访问组条目创建和管理 Connector。

Console

使用控制台管理模板组权限

您可以从模板的详细信息页面管理现有模板的群组和权限。有关更多信息，请参阅查看连接器模板详细信息。

设置哪些组可以或不能为特定模板注册证书的权限。您提供组的安全标识符（SID）。然后为该组设置注册和自动注册权限。对于自动注册，注册和自动注册必须均设置为“允许”。

API

使用 API 管理模板组权限

创建：Active Directory API Amazon 私有 CA 连接器中的 CreateTemplateGroupAccessControlEntry操作。

更新：Active Directory API Amazon 私有 CA 连接器中的 UpdateTemplateGroupAccessControlEntry操作。

检索：Active Directory API Amazon 私有 CA 连接器中的 GetTemplateGroupAccessControlEntry操作。

列表：Active Directory API Amazon 私有 CA 连接器中的 ListTemplateGroupAccessControlEntries操作。

删除：Active Directory API Amazon 私有 CA 连接器中的 DeleteTemplateGroupAccessControlEntry操作。

CLI

使用 CLI 管理模板组权限

在 Active Directory create-template-group-access的 Amazon 私有 CA 连接器部分@@ 创建:-control- entry 命令。 Amazon CLI

更新： update-template-group-access-control-entry 命令位于 Active Directory 的 Amazon 私有 CA 连接器部分。 Amazon CLI

在 Active Directory 的 Amazon 私有 CA 连接器部分中@@ 检索: get-template-group-access-control- entry 命令。 Amazon CLI

L@@ ist: list-template-group-access-control-entries 命令位于 Active Directory 的 Amazon 私有 CA 连接器部分。 Amazon CLI

“活动目录 Amazon 私有 CA 连接器” 部分中的 “删除 delete-template-group-access:-control-entries” 命令。 Amazon CLI

配置服务主体名称

了解如何配置连接器的服务主体名称。

Console

使用控制台管理服务主体名称

可从连接器的详细信息页面管理现有 AD 连接器的服务主体名称（SPN）。有关更多信息，请参阅管理目录注册查看连接器详细信息

API

使用 API 管理服务主体名称

创建：Active Directory API Amazon 私有 CA 连接器中的 CreateServicePrincipalName操作。

检索：Active Directory API Amazon 私有 CA 连接器中的 GetServicePrincipalName操作。

列表：Active Directory API Amazon 私有 CA 连接器中的 ListServicePrincipalNames操作。

删除：Active Directory API Amazon 私有 CA 连接器中的 DeleteServicePrincipalName操作。

CLI

使用 CLI 管理服务主体名称

在 Active Directory 的 Amazon 私有 CA 连接器部分中@@ 创建： create-service-principal-name命令 Amazon CLI。

“检索：” 的 “活动目录 Amazon 私有 CA 连接器” 部分中的 get-service-principal-name命令 Amazon CLI。

列表：位于 “活动目录 Amazon 私有 CA 连接器” 部分中的 list-service-principal-names命令 Amazon CLI。

删除：位于 “活动目录 Amazon 私有 CA 连接器” 部分中的 delete-service-principal-name命令 Amazon CLI。

标记适用于 AD 的连接器资源

您可以将标签应用到连接器、模板和目录注册。标记可将元数据添加到资源，从而有助于组织和管理。

Console

使用控制台管理资源标记

在资源的详细信息页面上管理现有资源的标记。有关更多信息，请参阅以下流程：

- 查看连接器模板详细信息
- 管理目录注册

API

使用 API 管理资源标记

标签：Active Directory API Amazon 私有 CA 连接器中的 TagResource操作。

列出标签：Active Directory API Amazon 私有 CA 连接器中的 ListTagsForResource操作。

取消标记：Active Directory API 的 Amazon 私有 CA 连接器中的 UntagResource操作。

重要提示 – 使用标签来标记包含机密数据的对象是可以接受的。但标签本身不应包含任何个人身份信息（PII）、敏感信息或机密信息。

CLI

使用 CLI 管理资源标记

Tag: tag-resou rce 命令位于 Active Directory 的 Amazon 私有 CA 连接器部分。 Amazon CLI

列出标签：在 Active Directory 的 Amazon 私有 CA 连接器部分中的 list-tags-for-resource命令 Amazon CLI。

取消标记：在 Ac tive Directory 的 Amazon 私有 CA 连接器部分中取消标记资源命令。 Amazon CLI

使用 Amazon 将适用于 AD 的 Connector 集成到事件驱动的应用程序中 EventBridge

您可以将 Connector for AD 整合到事件驱动的应用程序 (EDAs) 中，这些应用程序使用 Connector for AD 中发生的事件在应用程序组件之间进行通信并启动下游进程。

例如，当您的账户中发生以下 Connector for AD 事件时，您可以调用其他 Amazon 服务或自定义组件：

- 证书已创建或创建失败时。
- 证书已注册或注册失败。

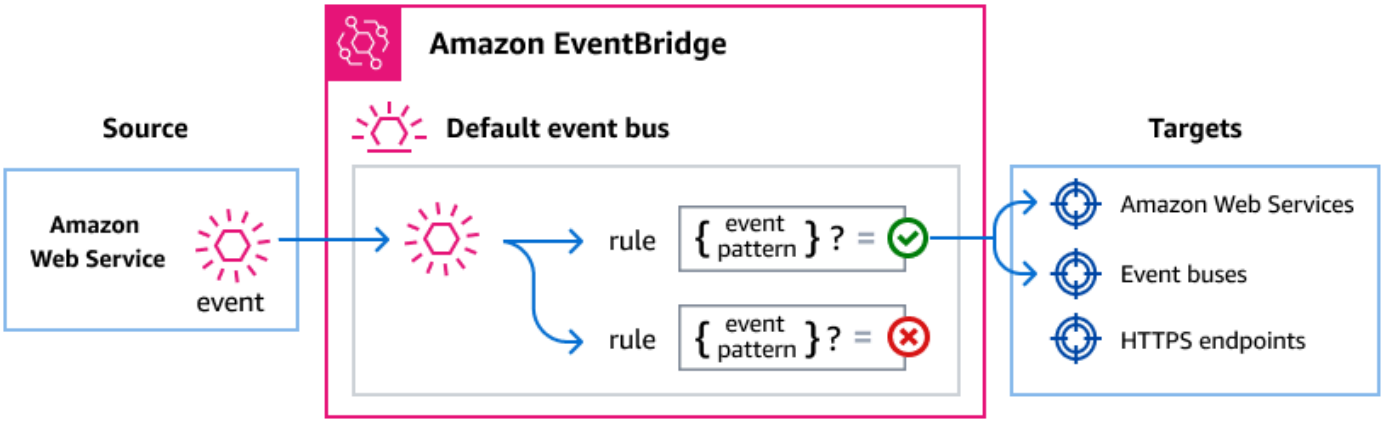
为此，您可以使用 Amazon EventBridge 将事件从 Connector for AD 路由到其他软件组件。Amazon EventBridge 是一项无服务器服务，它使用事件将应用程序组件连接在一起，这样您就可以更轻松地将诸如 Connector for AD 之类的 Amazon 服务集成到事件驱动的架构中，而无需额外的代码和操作。

如何为 AD 事件 EventBridge 路由连接器

以下是 EventBridge 与 Connector for AD 事件的配合方式：

与许多 Amazon 服务一样，Connector for AD 会生成事件并将其发送到 EventBridge 默认事件总线。事件总线是接收事件并将其路由到您所指定的目的地或目标的路由器。目标可以包括其他 Amazon 服务、定制应用程序和 SaaS 合作伙伴应用程序。

EventBridge 根据您在事件总线上创建的规则对事件进行路由。对于每条规则，您可以指定筛选条件或事件模式，以便仅选择所需的事件。每当向事件总线发送事件时，都要将其 EventBridge与每条规则进行比较。如果事件符合规则，则将事件 EventBridge 路由到指定的目标。



用于 AD 事件的连接器

有关发送到的 AD 连接器事件列表 EventBridge，请参阅《EventBridge 事件参考》中的 AD 连接器主题。

事件结构

来自 Amazon 服务的所有事件都包含两种类型的数据：

- 一组公共字段，其中包含有关事件的元数据，例如作为事件来源的 Amazon 服务、事件的生成时间、事件发生的账户和区域以及其他信息。有关这些常规字段的定义，请参阅 Amazon Events 参考中的 EventBridge 事件结构。
- `detail` 字段包含该特定服务事件专有的数据。

为 AD 事件创建与 Connector 匹配的事件模式

事件模式是过滤器，用于指定要选择的事件应包含哪些数据。

每个事件模式是一个 JSON 对象，其中包含：

- 标识发送事件的服务的 `source` 属性。对于 AD 连接器事件，源为`aws.pca-connector-ad`。
- （可选）：包含要匹配的事件名称数组的 `detail-type` 属性。
- （可选）：包含要匹配的其他事件数据的 `detail` 属性。

例如，以下事件模式将从 Connector for AD 中选择所有证书策略注册成功事件：

```
{
  "source": ["aws.pca-connector-ad"],
  "detail-type": ["Certificate Policy Enrollment Succeeded"]
}
```

有关写入事件模式的更多信息，请参阅《EventBridge 用户指南》中的事件模式。

接收来自的事件 EventBridge

您可以将 AD 证书的连接器指定为规则的目标。这使得 Connector for AD 能够接收来自各种来源的事件，包括其他 Amazon 服务、自定义应用程序和 SaaS 合作伙伴。有关更多信息，请参阅EventBridge 用户指南中的创建对事件做出反应的规则。

有关可以指定为目标的 Amazon 服务的完整列表，请参阅《EventBridge 事件参考》中的目标类型。

解决活动目录 Amazon 私有 CA 连接器的问题

使用此处的信息来帮助您诊断和修复 Conn Amazon 私有证书颁发机构 ector 的 AD 问题。

主题

- 对 AD 错误代码的连接器进行故障排除
- 排除 AD 连接器创建失败的连接器故障
- 对 AD SPN 创建失败的连接器进行故障排除
- 针对 AD 模板更新问题进行连接器故障排除

对 AD 错误代码的连接器进行故障排除

AD 连接器出于多种原因发送错误消息。有关每个错误的信息以及解决这些错误的建议，请参阅下表。您可以通过订阅 Amazon S EventBridge cheduler 事件（事件来源：`aws.pca-connector-ad`）或在 Windows 中使用手动注册来收到这些错误。

错误代码	根本原因	修复
0x8FFFA000	Kerberos 身份验证失败。	确保您的目录可以访问，并且客户端是用户或计算机。如果您使用的是自动注册，请修复您的 Amazon 资源服

错误代码	根本原因	修复
		务主体。如果您使用 Active Directory UI 获取证书，请运行 gpupdate /force 。
0x8FFFA001	SOAP 消息必须包含操作标头。	添加操作标头。
0x8FFFA002	连接器无法访问其所连接的私有 CA。	通过创建 Amazon Resource Access Manager（RAM）在私有 CA 与 Connector for AD 服务之间共享，从而与连接器共享您的私有 CA。
0x8FFFA003	此连接器的私有 CA 未激活。	将私有 CA 转为活动状态。如果您的私有 CA 处于待处理证书状态，则请安装 CA 证书。
0x8FFFA004	此连接器的私有 CA 不存在。	如果您的证书颁发机构处于“已删除”状态，则请将其转为“活动”状态。如果您的私有 CA 被永久删除，则请使用其他 CA 创建一个新的连接器。
0x8FFFA005	模板为证书使用者或使用者备用名称指定了 directoryGuid 属性，但在请求者的 AD 对象中找不到该属性。	Active Directory 没有为您的目录生成 directoryGuid 。在 Active Directory 中进行故障排除。
0x8FFFA006	模板为证书使用者或使用者备用名称指定了 dnsHostName 属性，但在请求者的 AD 对象中找不到该属性。	将 dnsHostName 属性添加到您的 AD 对象。

错误代码	根本原因	修复
0x8FFFA007	模板指定了要包含在证书使用者或使用者备用名称中的电子邮件属性，但在请求者的 AD 对象中找不到该属性。	将电子邮件属性添加到您的 AD 对象
0x8FFFA008	SOAP 消息必须有 `http://schemas.microsoft.com/windows/pki/2009/01/enrollmentpolicy/IPolicy/GetPolicies` 或 `http://schemas.microsoft.com/windows/pki/2009/01/enrollment/RST/wstep` 的操作标头。	更新操作标头以使用其中一个指定值。
0x8FFFA009	BinarySecurityToken 必须进行编码。`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#base64binary`	更新二进制安全令牌类型。
0x8FFFA00A	BinarySecurityToken 无效。	检查 CSR 是否正确生成。
0x8FFFA00B	的值类型 BinarySecurityToken 必须为`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#PKCS7` 或`http://schemas.microsoft.com/windows/pki/2009/01/enrollment#PKCS10` 。	将二进制安全令牌值类型更新为有效值。

错误代码	根本原因	修复
0x8FFFA00C	BinarySecurityToken 包含的内容管理系统无效。	Base64 有效，但加密消息语法（CMS）无效。检查 CMS 语法。
0x8FFFA00D	BinarySecurityToken 包含无效的 CSR。	检查 CSR 是否正确生成。
0x8FFFA00E	私有 CA 无法使用特定模板颁发证书。	查看来自的验证例外 Amazon 私有 CA。您可以在 Amazon EventBridge 或 Amazon 上查看验证异常 Amazon CloudTrail。
0x8FFFA00F	SOAP 消息的请求类型必须为 `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` 。	将请求类型设置为 `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` 。
0x8FFFA010	SOAP 消息必须有连接器 `CertificateEnrollmentPolicyServerEndpoint` 字段或 XCEP 响应中的 URI 字段的 to 标头。	将请求安全令牌的标头设置为 `CertificateEnrollmentPolicyServerEndpoint` 字段或 XCEP 响应中的 URI 字段。
0x8FFFA011	SOAP 消息必须只有一个操作标头。	查看请求安全令牌的 SOAP 消息标头并正确设置标头。
0x8FFFA012	SOAP 消息必须只有一个 `messageId` 标头。	查看请求安全令牌的 SOAP 消息标头并正确设置标头。
0x8FFFA013	SOAP 消息必须只有一个 to 标头。	查看请求安全令牌的 SOAP 消息标头并正确设置标头。

错误代码	根本原因	修复
0x8FFFA014	请求者无权访问所请求的模板。	通过创建访问控制条目，允许请求者的组使用请求的模板进行注册。
0x8FFFA015	`CertificateTemplateInformation` 或`CertificateTemplateName` 扩展名必须存在于中 BinarySecurityToken。	将安全扩展添加到您的 CSR。
0x8FFFA016	找不到给定连接器请求的模板。	模板是每个连接器的子资源。使用 `createTemplate` 为连接器创建模板。
0x8FFFA017	由于请求限制而导致请求被拒绝。	降低请求速率。
0x8FFFA018	SOAP 消息必须包含 `to` 标头。	查看 SOAP 消息的标头。
0x8FFFA019	由于标头无法识别，无法处理 SOAP 消息。	查看 SOAP 消息的标头。
0x8FFFA01A	模板指定了要包含在证书使用者或使用者备用名称中的 UPN 属性，但在请求者的 AD 对象中找不到该属性。	将 UPN 添加到 Active Directory 对象。

排除 AD 连接器创建失败的连接器故障

创建 AD 连接器的连接器可能由于各种原因而失败。连接器创建失败时，您将在 API 响应中收到失败原因。如果您使用的是控制台，则失败原因将显示在连接器详细信息页面的 “连接器详细信息” 容器中 “其他状态详细信息” 字段下。下表描述了失败原因和建议的解决步骤。

失败状态	描述	修复
CA_CERTIFICATE_REGISTRATION_FAILED	AD 连接器无法将 CA 证书导入您的目录。	查看 “先决条件” 页面，并检查您的服务帐号是否具有正确的权限。将正确的权限委派给您的服务帐号后，删除失败的连接器并创建一个新的连接器。有关委派权限的信息，请参阅《Amazon Directory Service 管理指南》中的向服务帐号委派权限。
DIRECTORY_ACCESS_DENIED	AD 连接器无法访问您的目录。	您必须授予 Connector for AD 访问您的目录的权限。请查看该步骤 4：创建 IAM 策略部分，确保与您的 Amazon 账户关联的 IAM 策略允许您访问和描述目录。向您的 Amazon 角色授予正确的权限后，删除失败的连接器并创建一个新的连接器。 如果将 Connector for A Amazon Directory Service D 与 AD 连接器一起使用，请确保 AD Connector 服务帐户的密码未过期且有效。有关 AD Connector 服务帐户的信息，请参阅《AD 连接器管理指南》中的 AD Connec tor 入门。
INTERNAL_FAILURE	AD 连接器出现内部故障。	请稍后重试。删除失败的连接器并创建一个新的连接器。

失败状态	描述	修复
INSUFFICIENT_FREE_ADDRESSES	VPC 子网必须至少有一个可用的私有 IP 地址。	确保子网中有可用的私有 IP 地址。删除失败的连接器并创建一个新的连接器。
INVALID_SUBNET_IP_PROTOCOL	AD 连接器无法在您的 VPC 上创建终端节点，因为与您的目录关联的子网不支持指定的 IP 地址类型。	确保托管您的目录的 VPC 和子网支持您选择的 IP 地址类型。有关更多信息，请参阅 IP 地址类型。删除失败的连接器，然后使用支持的 IP 地址类型创建一个新的连接器。
PRIVATECA_ACCESS_DENIED	AD 连接器无法访问您的私有 CA。	查看 “先决条件” 页面，并检查您是否具有创建连接器的权限。有关信息，请参阅步骤 4：创建 IAM 策略。 如果您通过 Amazon CLI 或 API 创建连接器，请查看 “先决条件” 页面，并检查您是否已使用与 Connector for AD 共享私有 CA Amazon Resource Access Manager。 检查并修复 IAM 权限和 Amazon RAM 资源共享后，删除失败的连接器并创建一个新的连接器。
PRIVATECA_RESOURCE_NOT_FOUND	AD 连接器找不到指定的私有 CA。	请确保指定正确的私有 CA Amazon 资源名称 (ARN)，然后删除失败的连接器，并使用您想要的私有 CA ARN 创建一个新的连接器。

失败状态	描述	修复
SECURITY_GROUP_NOT_IN_VPC	安全组不在托管您的目录的 VPC 中。	使用托管目录的 VPC 中的安全组。有关更多信息，请参阅 步骤 7：配置安全组。删除失败的连接器，然后使用位于 VPC 中的安全组创建一个新的连接器。
VPC_ACCESS_DENIED	AD 连接器无法访问托管您的目录的 Amazon VPC。	检查您的 IAM 权限。删除失败的连接器并创建一个新的连接器。有关包含访问权限的 IAM 策略示例，请参阅 步骤 4：创建 IAM 策略
VPC_ENDPOINT_LIMIT_EXCEEDED	AD 连接器无法在您的 Amazon VPC 中创建终端节点。您已达到可以为您的账户创建 VPC 终端节点的上限。	删除 Amazon VPC 终端节点，或请求提高限制。完成两个步骤之一后，删除失败的连接器并创建一个新的连接器。有关配额的信息，请参阅 Amazon Virtual Private Cloud 服务配额。
VPC_RESOURCE_NOT_FOUND	AD 连接器找不到指定的 VPC。	请确保您指定的 VPC 正确且该 VPC 存在。然后删除失败的连接器，并使用正确的 VPC ID 创建一个新的连接器。

对 AD SPN 创建失败的连接器进行故障排除

服务主体名称 (SPN) 创建可能由于各种原因而失败。当 SPN 创建失败时，您将在 API 响应中收到失败原因。如果您使用的是控制台，则失败原因将显示在连接器详细信息页面的服务主体名称 (SPN) 容器内其他状态详细信息字段下。下表描述了失败原因和建议的解决步骤。

失败状态	描述	修复
DIRECTORY_ACCESS_DENIED	AD 连接器无法访问您的目录。	授予连接器让 AD 访问您的目录的权限。有关包含授予目录访问权限的权限的 IAM 策略示例，请参阅步骤 4：创建 IAM 策略。
DIRECTORY_NOT_REACHABLE	AD 连接器无法访问您的目录。	检查与您的目录 Amazon 之间的网络，然后尝试再次创建 SPN。
DIRECTORY_RESOURCE_NOT_FOUND	AD 连接器找不到指定的目录。	请确保指定正确的目录 ID，然后删除失败的连接器，并使用预期的目录 ID 创建一个新的连接器。
INTERNAL_FAILURE	AD 连接器出现内部故障。	请稍后重试。
SPN_EXISTS_ON_DIFFERENT_AD_OBJECT	服务主体名称 (SPN) 存在于另一个 Active Directory 对象上。	从 Active Directory 对象中删除 SPN，然后尝试再次创建 SPN。
SPN_LIMIT_EXCEEDED	AD 连接器无法创建 SPN，因为您已达到 SPNs 每个目录的限制。 SPNs 每个目录的最大数量为 10。	SPNs 从您的账户中删除一个或多个，然后尝试再次创建 SPN。

针对 AD 模板更新问题进行连接器故障排除

如果您对模板或组访问控制条目进行了更改，但看不到更改，则可能是由于策略缓存所致。 Amazon 私有 CA 当您的客户端刷新策略缓存（每八小时刷新一次）时，将模板应用于您的策略。当您的客户端刷新缓存时，它会向连接器查询可用模板。如果是自动注册刷新，则客户端会颁发符合以下任一或两个条件的证书：

- 证书在续订期内。
- 证书不存在于客户端设备上。

要进行手动刷新，客户端将查询连接器，并且您必须将模板设置为发布。

如果您正在调试，则可以手动清除策略缓存以立即查看模板的更改。为此，请在您的客户端上运行以下 Powershell 命令。

```
certutil -f -user -policyserver * -policycache delete
```

Amazon 私有 CA 适用于 SCEP 的连接器

简单证书注册协议 (SCEP) 连接器 Amazon 私有证书颁发机构 可链接到您的 SCEP-enabled 移动设备和网络设备。使用适用于 SCEP 的连接器，您可以使用 Amazon 私有 CA 颁发证书和注册您的 SCEP 设备。SCEP 连接器可用于流行的移动设备管理 (MDM) 系统，专为与支持 SCEP 的客户端或端点配合使用而设计。

主题

- 功能
- 如何开始使用适用于 SCEP 的连接器
- 相关服务
- 适用于 SCEP 的接入连接器
- 定价
- SCEP 概念的连接器
- 了解 Connector 的 SCEP 注意事项和限制
- 为 SCEP 设置连接器
- 开始使用适用于 SCEP 的连接器
- 为适用于 SCEP 的连接器配置 MDM 系统
- SCEP 连接器中的安全性
- 适用于 SCEP 的显示器连接器
- 故障排除 Amazon 私有证书颁发机构 用于 SCEP 问题的连接器

功能

Support for SCEP 协议-SCEP 是一种广泛采用的协议，用于从证书颁发机构 (CA) 获取数字身份证书并将其分发到移动设备和网络设备。您可以使用适用于 SCEP 的连接器来帮助您使用 SCEP 注册终端。

移动设备注册 ——你可以将 Connector for SCEP 用于常用 MDM 系统，包括微软 Intune 和 Jamf Pro。

大规模颁发证书-将 SCEP-enabled 设备配置为通过连接器的 SCEP 端点请求证书后，您的客户端可以自动向其申请证书。 Amazon 私有 CA

如何开始使用适用于 SCEP 的连接器

要开始使用，请从 Connector for SCEP 管理控制台启动向导，该向导可帮助您创建连接器并指定要与该连接器一起使用的私有 CA。完成这些步骤后，Connector for SCEP 将提供端点和其他配置参数，您可以将其输入到 MDM 系统或网络设备中。配置 MDM 系统或网络设备后，您的客户将自动向请求证书。 Amazon 私有 CA要详细了解如何开始使用适用于 SCEP 的连接器，请参阅开始使用适用于 SCEP 的连接器。

相关服务

SCEP 的连接器与以下 Amazon 服务相关。

- Amazon 私有证书颁发机构- Amazon 私有 CA 为您提供高度可用的私有 CA 服务，无需支付运营自己的私有 CA 的前期投资和持续维护成本。
- Amazon 私有 CA A@@ ctive Directory 连接器-AD 连接器将您的活动目录 (AD) 链接到 Amazon 私有 CA。连接器负责将证书交换 Amazon 私有 CA 给您的 AD 管理的用户和计算机。

适用于 SCEP 的接入连接器

您可以使用以下任何接口创建、访问和管理 SCEP 连接器的连接器：

- Amazon Web Services 管理控制台-提供可用于访问 SCEP 连接器的 Web 界面。参见 SCEP 管理控制台的连接器。
- Amazon Command Line Interface-为各种 Amazon 服务提供命令，包括用于 SCEP 的连接器。在 Amazon CLI Windows、macOS 和 Linux 上都支持。有关更多信息，请参阅 Amazon Command Line Interface。
- Amazon 软件开发工具包-提供特定语言的 API 并处理许多连接细节，例如计算签名、处理请求重试和错误处理。有关更多信息，请参阅 Amazon Command Line Interface。
- SCEP API 连接器-提供您使用 HTTPS 请求调用的低级 API 操作。使用适用于 SCEP 的连接器 API 是访问该服务的最直接方式。但是，适用于 SCEP 的 Connector API 要求您的应用程序处理低级细节，例如生成哈希值以签署请求以及错误处理。有关更多信息，请参阅 SCEP 连接器 API 参考。

定价

SCEP 连接器作为一项功能提供，无需 Amazon 私有 CA 额外付费。您只需为用于创建和更新连接器的 Amazon 私有证书颁发机构 操作和证书付费。

有关最新的定 Amazon 私有 CA 价信息，请参阅Amazon 私有证书颁发机构 定价。您也可以使用定 Amazon 价计算器来估算成本。

SCEP 概念的连接器

SCEP 连接器是的附加功能。 Amazon 私有证书颁发机构

以下是 SCEP 连接器的关键概念：

证书签名请求 (CSR)

为颁发数字证书而向 CA 提供的必要信息。此信息包含公钥和身份。

质询密码

在从 CA 颁发证书之前，SCEP 协议使用质询密码对请求进行身份验证。SCEP 连接器根据连接器类型处理 SCEP 质询密码。有关更多信息，请参阅 为适用于 SCEP 的连接器配置 MDM 系统。

证书吊销

证书吊销是在已颁发的证书到期之前将其吊销的过程。您可以通过调用 API、 Amazon SDK 或 Amazon CloudFormation来吊销与连接器关联RevokeCertificate的私有 CA 证书。 Amazon Command Line Interface

适用于 SCEP 的连接器

SCEP 的连接器 Amazon 私有 CA 可链接到您的 SCEP-enabled 设备。

移动设备管理

移动设备管理 (MDM) 允许 IT 管理员在智能手机、平板电脑和其他端点或设备上控制、保护和实施策略。许多 MDM 系统都为 SCEP-based 证书注册提供内置集成。

SCEP

SCEP 是一种用于自动分发证书的标准化协议 (RFC 8894)。该协议为设备提供了向 CA 请求证书的端点。SCEP 使用质询密码授权向设备颁发证书。SCEP 通常应用于移动设备管理 (MDM) 系统和网络设备。MDM 解决方案允许 IT 管理员在智能手机、平板电脑和 Apple 工作站等其他实体上控制、保护和执行策略。大多数 MDM 解决方案都支持 SCEP，例如微软 Intune、Apple MDM 和 Jamf Pro。大多数网络设备，例如路由器、负载均衡器、 Wi-Fi 集线器、VPN 设备和防火墙，都使用 SCEP 进行自动证书注册。

SCEP 简介

SCEP 配置文件包含用于定义证书配置文件的配置参数。这包括证书有效期、密钥大小、SCEP 配置名称、质询密码、失败的重试次数和重试间隔，以及其他与证书颁发相关的信息。MDM 系统和证书管理平台通常会将 SCEP 配置文件发送给将请求证书进行身份验证的客户端。

了解 Connector 的 SCEP 注意事项和限制

使用适用于 SCEP 的连接器时，请记住以下注意事项和限制。

注意事项

CA 操作模式

您只能将 Connector for SCEP 与使用通用操作模式的私有 CA 一起使用。SCEP 的连接器默认为颁发有效期为一年的证书。使用短期证书模式的私有 CA 不支持颁发有效期大于七天的证书。有关操作模式的信息，请参见了解 Amazon 私有 CA CA 模式。

质疑密码

- 非常谨慎地分发您的挑战密码，并且仅与高度信任的个人和客户共享。单个质询密码可用于颁发任何证书，包括任何主体和 SAN，这会带来安全风险。
- 如果使用通用连接器，我们建议您经常手动轮换质询密码。

符合 RFC 8894

SCEP 连接器通过提供 HTTPS 端点而不是 HTTP 端点来偏离 RFC 8894 协议。

企业社会责任

- 如果发送到 Connector for SCEP 的证书签名请求 (CSR) 不包括扩展密钥使用 (EKU) 扩展，我们会将 EKU 值设置为。`clientAuthentication`有关信息，请参阅 4.2.1.12。RFC 528@@ 0 中扩展了密钥的用法。
- 我们支持 CSR 中的属性`ValidityPeriod`并对其进行`ValidityPeriodUnits`自定义。如果您的 CSR 不包括`ValidityPeriod`，我们会颁发有效期为一年的证书。请记住，您可能无法在 MDM 系统中设置这些属性。但是，如果你能设置它们，我们就会支持它们。有关这些属性的信息，请参阅 szenrolment_name_value_pair。

端点共享

仅将连接器的端点分发给可信方。将终端节点视为机密，因为任何能够找到您唯一的完全限定域名和路径的人都可以检索您的 CA 证书。

限制

以下限制适用于 SCEP 的连接器。

动态质询密码

您只能使用通用连接器创建静态质询密码。要在通用连接器上使用动态密码，必须构建自己的轮换机制，使用连接器的静态密码。适用于 Microsoft Intune 的 SCEP 连接器类型支持动态密码，你可以使用 Microsoft Intune 管理动态密码。

HTTP

SCEP 连接器仅支持 HTTPS，并且可以为 HTTP 调用创建重定向。如果您的系统依赖于 HTTP，请确保它能够容纳 Connector for SCEP 提供的 HTTP 重定向。

共享私有 CA

您只能将适用于 SCEP 的 Connector 与您拥有的私有 CA 一起使用。

为 SCEP 设置连接器

本节中的步骤可帮助您开始使用适用于 SCEP 的连接器。它假设你已经创建了一个 Amazon 账户。完成本页上的步骤后，您可以继续为 SCEP 创建连接器。

主题

- 步骤 1：创建一个 Amazon Identity and Access Management policy
- 步骤 2：创建私有 CA
- 步骤 3：使用创建资源共享 Amazon Resource Access Manager

步骤 1：创建一个 Amazon Identity and Access Management policy

要为 SCEP 创建连接器，您需要创建一个 IAM 策略，授予 Connector for SCEP 创建和管理连接器所需的资源以及代表您颁发证书的能力。有关 IAM 的更多信息，请参阅什么是 IAM？ 在 IAM 用户指南中。

以下示例是一个客户托管策略，您可以将其用于 Connector for SCEP。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "pca-connector-scep:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListCertificateAuthorities",
                "acm-pca:ListTags",
                "acm-pca:PutPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "acm-pca:IssueCertificate",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APICSRPassthrough/V*"
                },
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "pca-connector-scep.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ram:CreateResourceShare",
                "ram:GetResourcePolicies",
                "ram:GetResourceShareAssociations",
                "ram:GetResourceShares",
```

```
                "ram:ListPrincipals",
                "ram:ListResources",
                "ram:ListResourceSharePermissions",
                "ram:ListResourceTypes"
            ],
            "Resource": "*"
        }
    ]
}
```

步骤 2：创建私有 CA

要将连接器用于 SCEP，您需要将私有 CA 与该连接器关联起来。 Amazon 私有证书颁发机构 由于 SCEP 协议中存在固有的安全漏洞，因此我们建议您使用仅用于连接器的私有 CA。

私有 CA 必须满足以下要求：

- 它必须处于活动状态并使用通用操作模式。
- 您必须拥有私有 CA。您不能使用通过跨账户共享与您共享的私有 CA。

将私有 CA 配置为与 SCEP 连接器一起使用时，请注意以下注意事项：

- DNS 名称限制-考虑使用 DNS 名称限制来控制为你的 SCEP 设备颁发的证书中允许或禁止哪些域。有关更多信息，请参阅中的如何强制执行 DNS 名称限制 Amazon 私有证书颁发机构。
- 撤销 — 在私有 CA 上启用 OCSP 或 CRL 以允许撤销。有关更多信息，请参阅 规划您的 Amazon 私有 CA 证书吊销方法。
- PII — 我们建议您不要在 CA 证书中添加个人身份信息 (PII) 或其他机密或敏感信息。如果出现安全漏洞，这有助于限制敏感信息的泄露。
- 将@@ 根证书存储在信任存储中 — 将根 CA 证书存储在设备信任存储中，以便您可以验证证书和的返回值GetCertificateAuthorityCertificate。有关与之相关的信任存储的信息 Amazon 私有 CA，请参阅根 CA。

有关如何创建私有 CA 的信息，请参阅在中创建私有 CA Amazon 私有 CA。

步骤 3：使用创建资源共享 Amazon Resource Access Manager

如果您使用 Amazon Command Line Interface、 Amazon SDK 或适用于 SCEP 的连接器 API 以编程方式使用适用于 SCEP 的连接器，则需要使用 Amazon Resource Access Manager 服务主体共享与

适用于 SCEP 的连接器共享您的私有 CA。这为 SCEP 的 Connector 提供了对您的私有 CA 的共享访问权限。当您在 Amazon 控制台中创建连接器时，我们会自动为您创建资源共享。有关资源共享的信息，请参阅《Amazon RAM 用户指南》中的创建资源共享。

要使用创建资源共享 Amazon CLI，可以使用 Amazon RAM create-resource-share命令。以下命令创建资源共享。将要共享的私有 CA 的 ARN 指定为的值。*resource-arns*

```
$  aws ram create-resource-share \
--region us-east-1 \
--name MyPcaConnectorScepResourceShare \
--permission-arns arn:aws:ram::aws:permission/
AWSRAMBlankEndEntityCertificateAPICSRPassthroughIssuanceCertificateAuthority \
--resource-arns arn:aws:acm-pca:Region:account:certificate-authority/CA_ID \
--principals pca-connector-scep.amazonaws.com \
--sources account
```

调用的服务主体`CreateConnector`拥有私有 CA 的证书颁发权限。要防止使用 Connector for SCEP 的服务主体对您的 Amazon 私有 CA 资源拥有一般访问权限，请使用限制他们的权限。`CalledVia`

开始使用适用于 SCEP 的连接器

使用适用于 SCEP 的 Conn Amazon 私有证书颁发机构 ector，您可以从私有 CA 向 SCEP-enabled 设备和移动设备管理 (MDM) 系统颁发证书。创建连接器时， Amazon 私有证书颁发机构 会创建一个 SCEP URL 供您申请证书，还会为您提供可用于集成到 MDM 系统的信息。

要颁发证书，必须创建 Amazon 私有证书颁发机构 私有 CA，创建连接器，然后将 SCEP-enabled MDM 系统和设备配置为向连接器请求证书。

主题

- 开始前的准备工作
- 步骤 1：创建连接器
- 步骤 2：将连接器详细信息复制到 MDM 系统中

开始前的准备工作

以下教程将指导您完成为 SCEP 创建连接器的过程。

要学习本教程，你需要一个私有 CA 和一 SCEP-enabled 台设备。您还必须首先满足本为 SCEP 设置连接器节中列出的先决条件。

以下过程指导您如何使用 Amazon 控制台创建连接器。

任务

- 步骤 1：创建连接器
- 步骤 2：将连接器详细信息复制到 MDM 系统中

步骤 1：创建连接器

你要么创建一个用于通用用途的连接器，要么为 Microsoft Intune 创建用于 SCEP 的连接器。General-purpose 连接器专为与 SCEP-enabled 端点配合使用而设计，您可以管理 SCEP 质询密码。适用于微软 Intune 的 SCEP 连接器适用于微软 Intune，你可以使用 Microsoft Intune 管理质询密码。

General-purpose

创建用于通用用途的连接器

登录您的 Amazon 账户，打开适用于 SCEP 的连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-scep/home**。

1. 选择 Create connector (创建连接器)。
2. 在 “创建连接器” 页面中，可以选择在 “名称标记” 字段中为连接器指定一个友好名称。该名称将显示在您的连接器列表中。如果您愿意，可以通过选择 “添加更多标签” 向连接器添加更多标签。标签是您分配给 Amazon 资源的标签。每个标签都由一个键和一个可选值组成。您可以使用标签来搜索和筛选资源或跟踪 Amazon 成本。
3. 在 “连接器类型” 下，选择General-purpose。
4. 在 “私有 CA” 下，选择要用于此连接器的私有 CA。或者，通过选择 “创建私有 CA” 来创建一个新的 CA。由于 SCEP 协议中存在固有的漏洞，我们建议使用专用于此连接器的私有 CA。如果您创建了新 CA，则在中 Amazon 私有 CA完成创建后，请返回 Connector for SCEP 控制台并刷新私有 CA 列表。您的新私有 CA 应该可供选择。
5. 在 “质询密码” 下，选择 “自动生成质询密码”。创建此连接器时，我们将为您生成一个静态质询密码。
6. 在 “连接” 下，选择 “公用” 以创建可通过公共互联网访问的连接器。或者，选择私有并指定 VPC 终端节点，将此连接器限制为只能通过该特定的 VPC 终端节点进行访问。
7. 选择 “创建连接器”。

Microsoft Intune

为微软 Intune 的 SCEP 创建 Connector

登录您的 Amazon 账户，打开适用于 SCEP 的连接器控制台，网址为**https://console.aws.amazon.com/pca-connector-scep/home**。

1. 选择 Create connector (创建连接器)。
2. 在创建连接器页面上，可以选择在名称标签字段中为连接器指定一个友好名称。该名称将显示在您的连接器列表中。如果您愿意，可以通过选择 “添加更多标签” 向连接器添加更多标签。标签是您分配给 Amazon 资源的标签。每个标签都由一个键和一个可选值组成。您可以使用标签来搜索和筛选资源或跟踪 Amazon 成本。
3. 在 “连接器类型” 下，选择 Microsoft Intune。
   a. 对于应用程序（客户端）ID，请输入您的 Microsoft Entra ID 应用程序注册中的应用程序（客户端）ID。有关使用带连接器的 Microsoft Intune for SCEP 的信息，请参阅。为适用于 SCEP 的连接器配置 MDM 系统
   b. 对于目录（租户）ID 或主域，请输入您的 Microsoft Entra ID 应用程序注册中的目录（租户）ID 或主域。
4. 在 “私有 CA” 下，选择要用于此连接器的私有 CA。或者，通过选择 “创建私有 CA” 来创建一个新的 CA。由于 SCEP 协议中存在固有的漏洞，我们建议使用专用于此连接器的私有 CA。如果您创建了新 CA，则在中 Amazon 私有 CA完成创建后，请返回 Connector for SCEP 控制台并刷新私有 CA 列表。您的新私有 CA 应该可供选择。
5. 在 “连接” 下，选择 “公用” 以创建可通过公共互联网访问的连接器。或者，选择私有并指定 VPC 终端节点，将此连接器限制为只能通过该特定的 VPC 终端节点进行访问。
6. 选择 “创建连接器”。

步骤 2：将连接器详细信息复制到 MDM 系统中

创建连接器后，您需要将以下详细信息从连接器复制到 MDM 系统中。要使用控制台查看连接器的详细信息，请从 SCEP 连接器控制台页面的列表中选择该连接器。

- SCEP URL-这是连接器的端点，您的 SCEP 客户端将从中请求证书。注意仅将此端点提供给可信实体。
- (General-purpose) 质询密码-在 “质询密码” 下，选择您在上述过程中自动生成的密码，然后选择 “查看密码” 以查看密码。要创建其他密码，请选择创建密码。请注意谨慎分发密码，并且仅向高度信任

的个人和客户分发密码。单个质询密码可用于颁发任何证书，包括任何主题和 SAN，因此应谨慎处理。

- （Microsoft Intune）Ope n ID 值——如果你要与微软 Intune 集成，则必须将开放身份证颁发者、开放身份主题和开放身份受众复制到微软 Entra 应用程序注册的 OpenID Connect (OIDC) 凭证中。有关更多信息，请参阅 为适用于 SCEP 的连接器配置 MDM 系统。

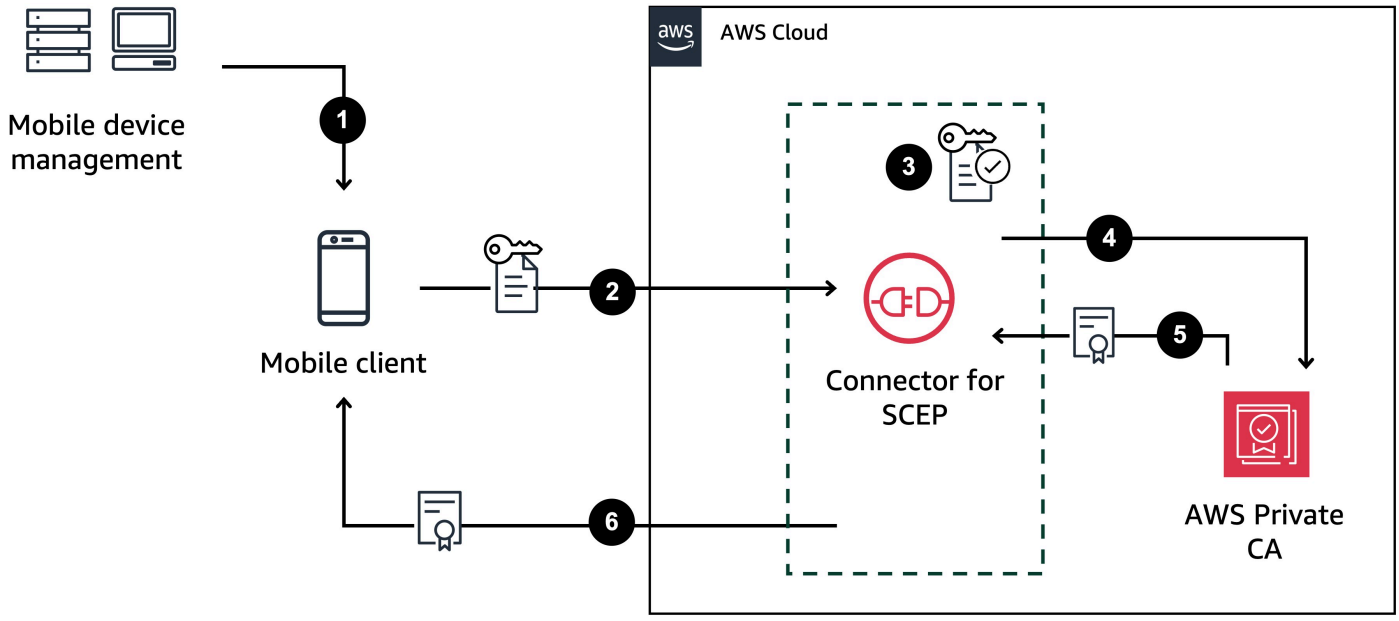
为适用于 SCEP 的连接器配置 MDM 系统

简单证书注册协议 (SCEP) 是用于证书注册和续订的标准协议。SCEP 连接器是一款基于 RFC 8894 的 SCEP 服务器，可自动 Amazon 私有证书颁发机构 向你的 SCEP 客户端颁发证书。创建连接器时，适用于 SCEP 的连接器会为 SCEP 客户端提供一个 HTTPS 端点，供其请求证书。客户端使用质询密码进行身份验证，该密码包含在向服务提出的证书签名请求 (CSR) 中。你可以将 Connector for SCEP 与流行的移动设备管理 (MDM) 系统（包括微软 Intune、Omnissa Workspace ONE 和 Jamf Pro）一起使用来注册移动设备。它旨在与任何支持 SCEP 的客户端或端点配合使用。

SCEP 连接器提供两种类型的连接器——通用连接器和适用于 Microsoft Intune 的 SCEP 连接器。以下各节介绍它们的工作原理，以及如何配置您的 MDM 系统以使用它们。

General-purpose 连接器

通用连接器旨在与支持 SCEP 的移动设备端点配合使用，但具有专用连接器的 Microsoft Intune 除外。使用通用连接器，例如 Jamf Pro 或 Omnissa Workspace ONE，您可以管理 SCEP 挑战密码。下图以移动设备管理 (MDM) 系统为例，但同样的功能适用于其他 SCEP-enabled 系统或设备。

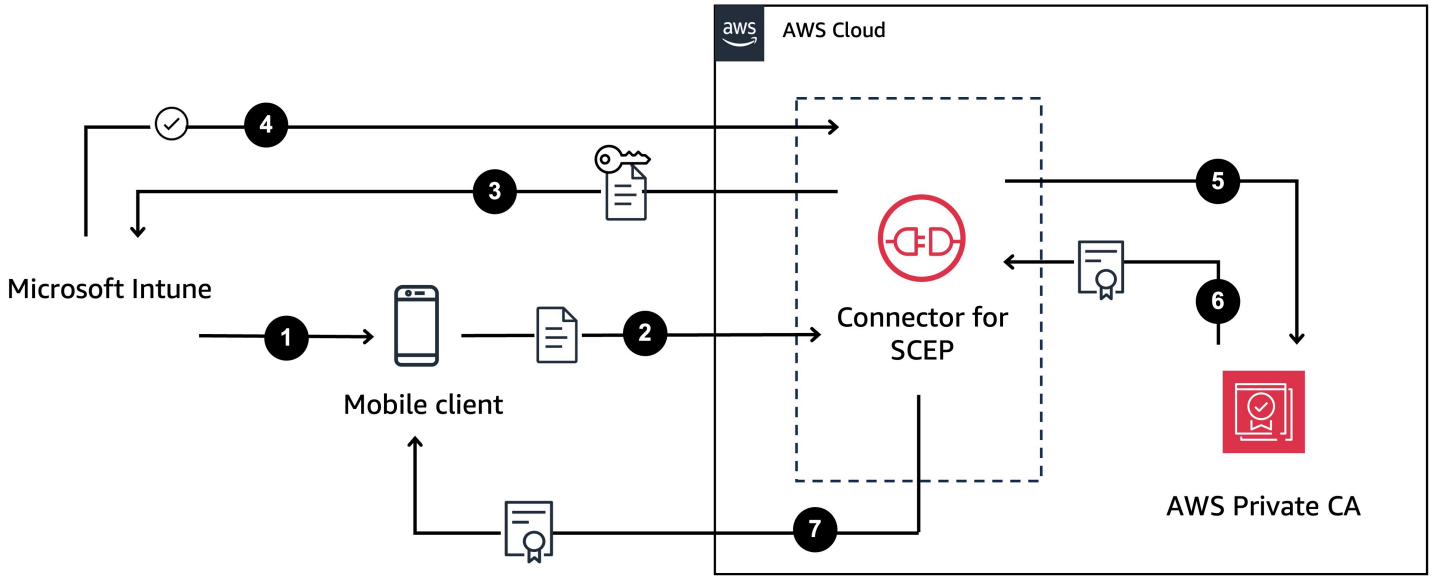


1. MDM 系统（或其他设备或系统）向移动客户端发送 SCEP 配置文件。SCEP 配置文件包含定义证书配置文件的配置参数，例如证书有效期、质询密码以及与证书颁发相关的其他信息。
2. 移动客户端请求证书，还会发送包含质询密码的证书签名请求 (CSR)。
3. SCEP 连接器验证质询密码。如果证书有效，则该服务将 Amazon 私有 CA 代表移动客户端请求证书。
4. Amazon 私有 CA 颁发证书并将其发送到连接器进行 SCEP。
5. SCEP 连接器将颁发的证书发送到移动客户端。

Amazon 私有 CA 适用于微软 Intune 的 SCEP 连接器

Amazon 私有 CA 适用于微软 Intune 的 SCEP 连接器专为与微软 Intune 配合使用而设计。使用适用于 Microsoft Intune 的 SCEP 连接器类型，你将使用 Microsoft Intune 来管理你的 SCEP 挑战密码。有关在 Microsoft Intune 中使用适用于 SCEP 的 Connector 的更多信息，请参阅。为 SCEP 的 Connector 配置微软 Intune

要在微软 Intune 上使用 Connector for SCEP，你必须使用微软 Intune API 启用特定功能，并拥有有效的微软 Intune 许可证。你还应该查看 Microsoft Intune® 应用程序保护政策。



1. Microsoft Intune 向移动客户端发送 SCEP 配置文件。该配置文件包含一个加密的质询密码，移动客户端将其放入 CSR 中。
2. 移动客户端请求证书并将 CSR 发送给 Connector 以获取 SCEP。
3. SCEP 连接器将 CSR 发送给 Microsoft Intune 进行授权。

4. Microsoft Intune 解密了 CSR 中的质询密码。如果证书有效，Microsoft Intune 会向 Connector 发送批准，让 SCEP 向移动客户端颁发证书。
5. SCEP 连接器 Amazon 私有 CA 代表移动客户端请求证书。
6. Amazon 私有 CA 颁发证书并将其发送到连接器进行 SCEP。
7. SCEP 连接器将颁发的证书发送到移动客户端。

主题

- 为 SCEP 的连接器配置 Jamf Pro
- 为 SCEP 的 Connector 配置微软 Intune
- 为 SCEP 连接器配置 Omnissa Workspace ONE

为 SCEP 的连接器配置 Jamf Pro

您可以在 Jamf Pro 移动设备管理 (MDM) 系统中 Amazon 私有 CA 用作外部证书颁发机构 (CA)。本指南提供有关在创建通用连接器后如何配置 Jamf Pro 的说明。

为 SCEP 的连接器配置 Jamf Pro

本指南提供有关如何配置 Jamf Pro 以与适用于 SCEP 的连接器一起使用的说明。成功为 SCEP 配置 Jamf Pro 和 Connector 后，您将能够向托管设备颁 Amazon 私有 CA 发证书。

Jamf Pro 要求

您的 Jamf Pro 实施必须满足以下要求。

- 您必须在 Jamf Pro 中启用 “启用基于证书的身份验证” 设置。您可以在 Jamf Pro 文档的 Jamf Pro 安全设置页面上找到有关此设置的详细信息。

第 1 步：（可选-推荐）获取您的私有 CA 的指纹

指纹是您的私有 CA 的唯一标识符，可用于在与其他系统或应用程序建立信任时验证您的 CA 的身份。通过使用证书颁发机构 (CA) 指纹，托管设备可以对其连接的 CA 进行身份验证，并仅向预期的 CA 请求证书。我们建议在 Jamf Pro 上使用 CA 指纹。

为您的私有 CA 生成指纹

1. 从 Amazon 私有 CA 控制台或使用获取私有 CA 证书GetCertificateAuthorityCertificate。将其另存为`ca.pem`文件。

2. 安装 OpenSSL 命令行实用程序。
3. 在 OpenSSL 中，运行以下命令来生成指纹：

```
openssl x509 -in ca.pem -sha256 -fingerprint
```

步骤 2：配置 Amazon 私有 CA 在 Jamf Pro 中作为外部 CA

为 SCEP 创建连接器后，必须在 Jamf Pro 中设置 Amazon 私有 CA 为外部证书颁发机构 (CA)。您可以设置 Amazon 私有 CA 为全局的外部 CA。或者，您可以使用 Jamf Pro 配置文件 Amazon 私有 CA 为不同的用例颁发不同的证书，例如向组织中的一部分设备颁发证书。有关实现 Jamf Pro 配置文件的指导超出了本文档的范围。

要配置 Amazon 私有 CA 在 Jamf Pro 中作为外部证书颁发机构 (CA)

1. 在 Jamf Pro 控制台中，前往 “设置” > “全局” > “PKI 证书”，进入 P KI 证书设置页面。
2. 选择 “管理证书模板” 选项卡。
3. 选择外部 CA。
4. 选择编辑。
5. （可选）为配置文件选择 “启用 Jamf Pro 作为 SCEP 代理”。您可以使用 Jamf Pro 配置文件颁发针对特定用例量身定制的不同证书。有关如何在 Jamf Pro 中使用配置文件的指导，请参阅 Jamf Pro 文档中的启用 Jamf Pro 作为配置文件的 SCEP 代理。
6. 选择 “使用 SCEP-enabled 外部 CA 注册计算机和移动设备”。
7. （可选）选择使用 Jamf Pro 作为 SCEP 代理进行计算机和移动设备注册。如果您遇到配置文件安装失败的情况，请参阅解决配置文件安装失败的问题。
8. 将 SCEP 连接器 SCE P 网址从连接器的详细信息中复制并粘贴到 Jamf Pro 的 URL 字段。要查看连接器的详细信息，请从 SCEP 的连接器列表中选择该连接器。或者，您可以通过调用GetConnector并复制响应中的`Endpoint`值来获取 URL。
9. （可选）在名称字段中输入实例的名称。例如，你可以给它起个名字Amazon 私有 CA。
10. 为挑战类型选择 “静态”。
11. 从连接器中复制质询密码，然后将其粘贴到挑战字段中。一个连接器可以有多个质询密码。要查看连接器的质询密码，请在 Amazon 控制台中导航到连接器的详细信息页面，然后选择查看密码按钮。或者，您可以通过调用GetChallengePassword并复制响应中的`Password`值来获取连接器的质询密码。有关使用质询密码的信息，请参阅了解 Connector 的 SCEP 注意事项和限制。
12. 将质询密码粘贴到验证质询字段中。

13. 选择密钥大小。我们建议密钥大小为 2048 或更高。
14. （可选）选择 “用作数字签名”。选择此项进行身份验证，以授予设备对诸如 Wi-Fi 和 VPN 之类的资源的安全访问权限。
15. （可选）选择 “用于密钥加密”。
16. （可选-推荐）在 “指纹” 字段中输入十六进制字符串。我们建议您添加 CA 指纹以允许托管设备验证 CA，并且仅向 CA 申请证书。有关如何为私有 CA 生成指纹的说明，请参阅第 1 步：（可选-推荐）获取您的私有 CA 的指纹。
17. 选择保存。

步骤 3：设置配置文件签名证书

要使用带连接器的 Jamf Pro for SCEP，您必须提供与您的连接器关联的私有 CA 的签名和 CA 证书。为此，您可以将包含两个证书的配置文件签名证书密钥库上传到 Jamf Pro。

以下是创建证书密钥库并将其上传到 Jamf Pro 的步骤：

- 使用内部流程生成证书签名请求 (CSR)。
- 获取由与您的连接器关联的私有 CA 签署的 CSR。
- 创建包含配置文件签名和 CA 证书的配置文件签名证书密钥库。
- 将证书密钥库上传到 Jamf Pro。

通过执行这些步骤，您可以确保您的设备可以验证和验证由您的私有 CA 签名的配置文件，从而允许在 Jamf Pro 中使用 Connector for SCEP。

1. 以下示例使用 OpenSSL 和 Amazon Certificate Manager，但您可以使用首选方法生成证书签名请求。

   Amazon Certificate Manager console

   使用 ACM 控制台创建配置文件签名证书

   1. 使用 ACM 申请私有 PKI 证书。包括以下内容：

      - 类型-使用与 MDM 系统的 SCEP 证书颁发机构相同的私有 CA 类型。
      - 在证书颁发机构详细信息部分，选择证书颁发机构菜单，然后选择用作 Jamf Pro CA 的私有 CA。

- 域名-提供要嵌入到证书中的域名。您可以使用完全限定的域名 (FQDN)，例如`www.example.com`，也可以使用裸域名或顶点域名，例如`example.com`（不`www.`包括）。

2. 使用 ACM 导出您在上一步中创建的私有证书。选择导出证书、证书链和加密密钥的文件。请随身携带密码短语，因为下一步你将需要它。
3. 在终端中，在包含导出文件的文件夹中运行以下命令，将 PKCS #12 bundle 写入由您在上一步中创建的密码编码`output.p12`的文件中。

```
openssl pkcs12 -export \
  -in "Exported Certificate.txt" \
  -certfile "Certificate Chain.txt" \
  -inkey "Exported Certificate Private Key.txt" \
  -name example \
  -out output.p12 \
  -passin pass:your-passphrase \
  -passout pass:your-passphrase
```

Amazon Certificate Manager CLI

使用 ACM CLI 创建配置文件签名证书

- 以下命令显示如何在 ACM 中创建证书，然后将文件导出为 PKCS #12 捆绑包。

```
PCA=<Enter your Private CA ARN>

CERTIFICATE=$(aws acm request-certificate \
    --certificate-authority-arn $PCA \
    --domain-name <any valid domain name, such as test.name> \
    | jq -r '.CertificateArn')

while [[ $(aws acm describe-certificate \
  --certificate-arn $CERTIFICATE \
  | jq -r '.Certificate.Status') != "ISSUED" ]] do sleep 1; done

aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.Certificate' > Certificate.pem
aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.CertificateChain' > CertificateChain.pem
```

```
aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.PrivateKey' > PrivateKey.pem

openssl pkcs12 -export \
  -in "Certificate.pem" \
  -certfile "CertificateChain.pem" \
  -inkey "PrivateKey.pem" \
  -name example \
  -out output.p12 \
  -passin pass:passphrase \
  -passout pass:passphrase
```

OpenSSL CLI

使用 OpenSSL CLI 创建配置文件签名证书

1. 使用 OpenSSL，通过运行以下命令生成私钥。

```
openssl genrsa -out local.key 2048
```

2. 生成证书签名请求 (CSR)：

```
openssl req -new -key local.key -sha512 -out local.csr -
subj "/CN=MySigningCertificate/O=MyOrganization" -addext
 keyUsage=critical,digitalSignature,nonRepudiation
```

3. 使用 Amazon CLI，使用您在上一步中生成的 CSR 颁发签名证书。运行以下命令，并在响应中记下证书 ARN。

```
aws acm-pca issue-certificate --certificate-authority-arn <SAME CA AS
 USED ABOVE, SO IT'S TRUSTED> --csr fileb://local.csr --signing-algorithm
 SHA512WITHRSA --validity Value=365,Type=DAYS
```

4. 通过运行以下命令获取签名证书。指定上一步中的证书 ARN。

```
aws acm-pca get-certificate --certificate-authority-arn <SAME CA AS USED
 ABOVE, SO IT'S TRUSTED> --certificate-arn <ARN OF NEW CERTIFICATE> | jq -r
 '.Certificate' >local.crt
```

5. 通过运行以下命令获取 CA 证书。

```
aws acm-pca get-certificate-authority-certificate --certificate-authority-
arn <SAME CA AS USED ABOVE, SO IT'S TRUSTED> | jq -r '.Certificate' > ca.crt
```

6. 使用 OpenSSL，以 p12 格式输出签名证书密钥库。使用您在步骤四和步骤五中生成的 CRT 文件。

```
openssl pkcs12 -export -in local.crt -inkey local.key -certfile ca.crt -name
 "CA Chain" -out local.p12
```

7. 出现提示时，输入导出密码。此密码是您提供给 Jamf Pro 的密钥库密码。

2. 在 Jamf Pro 中，导航到管理证书模板并转到外部 CA 窗格。
3. 在 “外部 CA” 窗格的底部，选择 “更改签名和 CA 证书”。
4. 按照屏幕上的说明上传外部 CA 的签名证书和 CA 证书。

步骤 4：（可选）在用户启动的注册过程中安装证书

要在您的客户端设备和私有 CA 之间建立信任，必须确保您的设备信任 Jamf Pro 颁发的证书。当客户端设备在User-Initiated 注册过程中申请证书时，您可以使用 Jamf Pro Amazon 私有 CA的注册设置自动在客户端设备上安装您的 CA 证书。

解决配置文件安装失败的问题

如果您在为计算机和移动设备注册启用 “使用 Jamf Pro 作为 SCEP 代理” 后遇到配置文件安装失败，请查阅您的设备日志并尝试以下操作。

设备日志错误消息	缓解方法
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-endpoint>.jamfcloud.com". <MDM-SCEP:15001>`	如果您在尝试注册时收到此错误消息，请重试注册。注册成功可能需要几次尝试。
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-`	您的质询密码可能配置错误。确认 Jamf Pro 中的质询密码是否与连接器的质询密码相匹配。

设备日志错误消息	缓解方法
`endpoint>.jamfcloud.com". <MDM-SCEP:14006>`	

为 SCEP 的 Connector 配置微软 Intune

你可以在 Microsoft Intune 移动设备管理 (MDM) 系统中 Amazon 私有 CA 用作外部证书颁发机构 (CA)。本指南提供有关在为微软 Intune 创建 SCEP 连接器后如何配置 Microsoft Intune 的说明。

先决条件

在为 Microsoft Intune 的 SCEP 创建连接器之前，必须完成以下先决条件。

- 创建入口 ID。
- 创建微软 Intune 租户。
- 在你的 Microsoft Entra ID 中创建应用程序注册。有关如何管理应用程序注册的应用程序级权限的信息，请参阅 Microsoft Entra 文档中的 Microsoft Entra ID 中更新应用程序请求的权限。应用程序注册必须具有以下权限：
  - 在 Intune 下设置 sc ep_challenge_provider。
  - 适用于微软 Graph 集Application.Read.All和User.Read。
- 您必须在应用程序注册管理员同意中授予该应用程序。有关信息，请参阅 Microsoft Entra 文档中的授予整个租户对应用程序的管理员同意。

> **Tip**
>
> 创建应用程序注册时，请记下应用程序（客户端）ID 和目录（租户）ID 或主域。当你为 Microsoft Intune 的 SCEP 创建连接器时，你需要输入这些值。有关如何获取这些值的信息，请参阅 Microsoft E ntra 文档中的创建可以访问资源的 Microsoft Entra 应用程序和服务主体。

第 1 步：授予 Amazon 私有 CA 使用你的 Microsoft Entra ID 应用程序的权限

为 Microsoft Intune 创建 SCEP 连接器后，必须在 Microsoft 应用程序注册下创建联合凭据，这样 SCEP 连接器才能与微软 Intune 通信。

要配置 Amazon 私有 CA 作为 Microsoft Intune 中的外部 CA

1. 在 Microsoft Entra ID 控制台中，导航到应用程序注册。
2. 选择您创建的用于连接器 for SCEP 的应用程序。您单击的应用程序的应用程序（客户端）ID 必须与您在创建连接器时指定的 ID 相匹配。
3. 从 “托管” 下拉菜单中选择 “证书和密钥”。
4. 选择 “联合证书” 选项卡。
5. 选择 “添加凭据”。
6. 从 “联邦证书方案” 下拉菜单中，选择 “其他颁发者”。
7. 将你的 Connector for SCEP for Microsoft Intune 详细信息中的 OpenID 发行者值复制并粘贴到发行者字段中。要查看连接器的详细信息，请从 Amazon 控制台的 SCEP 连接器列表中选择该连接器。或者，您可以通过调用获取 URL，GetConnector然后从响应中复制该`Issuer`值。
8. 在 “类型” 中，选择 “显式主题标识符”。
9. 将 OpenID 主题值从您的连接器复制并粘贴到值字段中。您可以在控制台的连接器详细信息页面中查看 OpenID 颁发者 Amazon 值。或者，您可以通过调用获取 URL，GetConnector然后从响应中复制该`Audience`值。
10. （可选）在名称字段中输入实例的名称。例如，你可以给它起个名字Amazon 私有 CA。
11. （可选）在描述字段中输入描述。
12. 将 OpenID 受众值从 Connector for Microsoft Intune 的 SCEP 详细信息复制并粘贴到 “受众” 字段中。要查看连接器的详细信息，请从 Amazon 控制台的 SCEP 连接器列表中选择该连接器。或者，您可以通过调用获取 URL，GetConnector然后从响应中复制该`Subject`值。
13. 选择添加。

第 2 步：设置微软 Intune 配置文件

在你授予 Amazon 私有 CA 调用 Microsoft Intune 的权限后，你必须使用 Microsoft Intune 创建微软 Intune 配置文件，指示设备联系 Connector 获取 SCEP 以获取证书。

1. 创建可信证书配置文件。你必须将与 Connector for SCEP 一起使用的链的根 CA 证书上传到 Microsoft Intune 才能建立信任。有关如何创建可信证书配置文件的信息，请参阅 Microsoft Intune 文档中的 Microsoft Intune 的可信根证书配置文件。
2. 创建 SCEP 证书配置文件，当您的设备需要新证书时，该配置文件可将设备指向连接器。配置文件的配置文件类型应为 SCEP 证书。对于配置文件的根证书，请确保使用在上一步中创建的可信证书。

对于 SCEP 服务器 URL，请将连接器详细信息中的 SCEP 网址复制并粘贴到 SCEP 服务器网址字段中。要查看连接器的详细信息，请从 SCEP 的连接器列表中选择该连接器。或者，您可以通过调用获取 URL ListConnectors，然后从响应中复制该Endpoint值。有关在 Microsoft Intune 中创建配置文件的指南，请参阅微软 Intune 文档中的在 Microsoft Intune 中创建和分配 SCEP 证书配置文件。

> **Note**
>
> 对于非 Mac OS 和 iOS 设备，如果您未在配置文件中设置有效期，则适用于 SCEP 的 Connector 会颁发有效期为一年的证书。如果您未在配置文件中设置扩展密钥用法 (EKU) 值，则 SCEP 的 Connector 将颁发一个 EKU 设置为的证书。`Client Authentication (Object Identifier: 1.3.6.1.5.5.7.3.2)`对于 macOS `ExtendedKeyUsage` 或 iOS 设备，Microsoft Intune 不尊重你的`Validity`配置文件中的任何参数。对于这些设备，Connector for SCEP 通过客户端身份验证向这些设备颁发有效期为一年的证书。

步骤 3：验证与 SCEP 连接器的连接

创建指向 SCEP 连接器端点的 Microsoft Intune 配置文件后，请确认注册的设备可以申请证书。要进行确认，请确保没有任何策略分配失败。要进行确认，请在 Intune 门户中导航到 “设备” > “管理设备” > “配置”，并确认配置策略分配失败下没有列出任何内容。如果有，请使用上述过程中的信息确认您的设置。如果您的设置正确但仍然出现故障，请查阅从移动设备收集可用数据。

有关设备注册的信息，请参阅什么是设备注册？ 在微软 Intune 文档中。

为 SCEP 连接器配置 Omnissa Workspace ONE

您可以在 Omnissa Workspace ONE UEM（统一端点管理）系统中 Amazon 私有 CA 用作外部证书颁发机构 (CA)。本指南提供有关在中创建 SCEP 连接器后如何配置 Omnissa Workspace ONE 的说明。Amazon

先决条件

在为 Omnissa Workspace ONE 创建 SCEP 连接器之前，必须完成以下先决条件：

- 在 Amazon 控制台中创建私有 CA。有关更多信息，请参阅 在中创建私有 CA Amazon 私有 CA。
- 创建通用的 SCEP 连接器。有关更多信息，请参阅创建连接器。

- 拥有一个活跃的 Omnissa Workspace ONE 环境管理员账户和组织组 ID。
- 如果您要注册 Apple 设备，请为 MDM 配置苹果推送通知服务 (APN)。有关更多信息，请参阅 Omnis sa 文档中的 APNs 证书。

第 1 步：在 Omnissa Workspace ONE 中定义证书颁发机构和模板

在 Amazon 控制台中创建私有 CA 和 SCEP 连接器后，在 Omnissa Workspace ONE 中定义证书颁发机构和模板。

添加 Amazon 私有 CA 作为证书颁发机构

1. 从 “系统” 菜单中选择 “企业集成”，然后选择 “证书颁发机构”。
2. 选择 + ADD 并提供以下信息：
   - 名称: Amazon-Private-CA。
   - 描述： Amazon 私有 CA 用于颁发设备证书。
   - 权限类型：选择通用 SCEP。
   - SCEP 网址：输入来自的 SCEP 网址。 Amazon 私有 CA
   - 挑战类型：选择静态。
   - 静态质询：在控制台中输入连接器中用于 SCEP 配置的 SCEP 静态质询密码。 Amazon
   - 输入重试超时和最大重试次数值。
3. 保存配置。

创建证书模板

1. 从 “系统” 菜单中选择 “企业集成”，选择 “证书颁发机构”，然后选择 “模板”。
2. 选择添加模板并提供以下信息：
   - 模板名称: Device-Cert-Template.
   - 证书颁发机构：选择 Amazon-Private-CA。
   - 主题名称：这是一个可自定义的字段。您可以从属性列表中选择变量值。例如，CN= {DeviceReportedName}、O= {}、OU= {1DevicePlatform} CustomAttribute
   - 私钥长度：2048 位。
   - 私钥类型：根据需要选择签名和加密
   - 自动续订： Enabled/Disabled （根据您的需求）。

3. 保存 模板。

第 2 步：设置 Omnissa Workspace ONE UEM 配置文件配置

在 Omnissa Workspace ONE UEM 中创建配置文件，将设备定向到 Connector，让 SCEP 颁发证书。

为证书分发创建 SCEP 设备配置文件

1. 从 “资源” 菜单中选择 “配置文件和基准”，然后选择 “配置文件”。
2. 选择添加，然后选择添加个人资料
3. 选择设备平台（安卓、iOS、macOS、Windows）。
4. 根据需要设置管理类型和上下文。
5. 设置名称: Device-Cert-Profile.
6. 滚动到 SCEP 有效负载。
7. 选择 SCEP，然后选择 + 添加。
8. 使用以下配置：
   - SCEP：
     - 对于凭据来源，选择定义的证书颁发机构（默认）。
     - 对于证书颁发机构，请选择 Amazon-Private-CA
     - 对于证书模板，请选择步骤 1 中Device-Cert-Template定义的。
9. 选择 “下一步”，然后在 “任务” 部分中，从列表中选择正确的智能组（设备的任务组）。
10. 选择 “分配类型” 为 “自动” 以启用自动续订。
11. 保存并发布个人资料。

> **Note**
>
> 有关更多信息，请参阅 Omnis sa 文档中的 SCEP。

第 3 步：在 Omnissa Workspace ONE 中注册设备

创建或验证智能群组

1. 从 “群组和设置” 中选择 “群组”，然后选择 “任务小组”。

2. 创建或编辑 POC-Devices 智能群组：

   - 姓名: POC-Devices。
   - 设备类型：选择全部或特定平台（例如 Android 或 iOS）。
   - 标准：使用 UserGroup“平台和操作系统”、“OEM” 和 “型号” 来指定对目标设备进行分组的标准。
   - 所有权：为个人或公司设备选择 “任意”。

3. 保存并验证目标设备是否显示在 “预览” 选项卡中。

手动注册设备

Android

- 从 Google Play 下载 Workspace ONE 智能中心应用程序。
- 打开应用程序并输入注册网址或扫描二维码。
- 登录并按照提示注册为 MDM-managed设备。

iOS/macOS

- 在设备上，打开 Safari 浏览并导航到注册网址（<WorkspaceONEUEMHostname>例如 https: /// enroll）。
- 使用用户凭据登录。
- 从 App St ore 下载并安装 Workspace ONE 智能中心应用程序。
- 按照提示在 “设置” > “常规” > “VPN 和设备管理” > “配置文件” > “安装” 中安装 MDM 配置文件。

Windows

- 从 Workspace ONE 服务器或微软商店下载 Workspace ONE 智能中心。
- 使用注册 URL 和凭据通过 Hub 注册。

在 “设备” > “列表视图” > “更多操作” > “分配到 POC-Devices 智能组” 中将注册的设备分配给智能组。

有关更多信息，请参阅 Omnissa 文档中的自动设备注册。

验证注册

1. 在 Omnissa Workspace ONE UEM 控制台中，前往 “设备”，然后转到 “列表视图”。
2. 确认已注册设备的状态设置为 “已注册”。

3. 在 “设备详细信息” 的 “群组” 选项卡中，验证设备是否在 POC-Devices 智能组中。

步骤 4：颁发证书

触发颁发证书

1. 在设备列表视图中，选择已注册的设备。
2. 选择 “查询” 按钮以提示办理登机手续。
3. Device-Cert-Profile 应通过以下方式签发证书。 Amazon 私有 CA

验证证书安装

Android

依次选择设置、安全、可信凭证和用户来验证证书。

iOS

前往 “设置”，然后选择 “常规”、“VPN 和设备管理”，然后选择 “配置文件”。验证来自的证书 AWS-Private-CA 是否存在。

macOS

打开 “钥匙串访问权限”，然后打开 “系统钥匙串” 并验证证书。

Windows

打开 certmgr.msc，然后打开 “个人”，然后打开 “证书” 以验证证书。

问题排查

SCEP 错误（例如 “22013-SCEP 服务器返回的响应无效”）

- 验证 Workspace ONE 中的 SCEP 网址和静态质询密码是否匹配 Amazon 私有 CA。
- <SCEP_URL>测试 SCEP 端点连接：curl。
- 检查 Amazon CloudTrail 日志中是否有 Amazon 私有 CA 错误（例如`IssueCertificate`失败）。

APN 问题 () iOS/macOS

- 确保 APNs 证书有效且已分配给正确的组织组。

- 测试 APN 的连接：telnet gateway.push.apple.com 2195。

配置文件安装失败

- 确认设备位于正确的智能组中（依次选择 “设备”、“列表视图” 和 “群组”）。
- 强制同步个人资料：依次选择 “更多操作”、“发送” 和 “个人资料列表”。

日志

- 安卓系统：使用 Logcat 或 Workspace ONE
- iOS/macOS: log show--predicate 'process == “mdmclient” '--最后 1 小时（通过配置器）。Xcode/Apple
- Windows：“事件查看器”，然后选择 “应用程序和服务日志”，然后是Microsoft-Windows-DeviceManagement。
- Workspace ONE UEM：监控，然后是报告和分析，然后是事件，然后是设备事件。

有关中用于 SCEP 监控的连接器的详细信息 Amazon，请参阅 SCEP 的监视器连接器。

安全注意事项

- 安全地存储 SCEP 网址和密钥。有关更多信息，请参阅Amazon Secrets Manager 服务。
- 将智能组标准仅限于目标设备。
- 定期续订 Apple 推送通知 (APN) 证书（有效期为 1 年）。
- 为概念验证项目设置较短的证书有效期，以最大限度地降低风险。
- 对于个人设备，请确保清理会删除所有配置文件和证书。

有关如何使用 SCEP 连接器配置 Omnissa Workspace ONE UEM 和 CA 集成的信息，请参阅 Omnissa Workspace ONE 文档中的 SCEP。

SCEP 连接器中的安全性

云安全 Amazon 是重中之重。作为 Amazon 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 Amazon 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — Amazon 负责保护在 Amazon 云中运行 Amazon 服务的基础架构。

- 云端安全-您的责任由您使用的 Amazon 服务决定。您还需要对其它因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档可帮助您了解在使用 Connector for SCEP 时如何应用分担责任模型。以下主题介绍如何为 SCEP 配置连接器以满足您的安全和合规性目标。

主题

- SCEP VPC 终端节点的连接器 (Amazon PrivateLink)

SCEP VPC 终端节点的连接器 (Amazon PrivateLink)

您可以通过配置接口 VPC 终端节点在您的 VPC 和适用于 SCEP 的连接器之间创建私有连接。接口端点由Amazon PrivateLink一种用于私密访问连接器以进行 SCEP API 操作的技术提供支持。 Amazon PrivateLink 通过亚马逊网络路由您的 VPC 和 Connector for SCEP 之间的所有网络流量，避免暴露在开放的互联网上。每个 VPC 终端节点都由您的 VPC 子网中一个或多个使用私有 IP 地址的弹性网络接口代表。

接口 VPC 终端节点将您的 VPC 直接连接到 SCEP 连接器，无需互联网网关、NAT 设备、VPN 连接或 Amazon Direct Connect 连接。您的 VPC 中的实例不需要公有 IP 地址即可与 SCEP API 的连接器通信。

要通过您的 VPC 使用 Connector for SCEP，您必须从 VPC 内部的实例进行连接。或者，您可以使用 Amazon Virtual Private Network (Amazon VPN) 或将您的私有网络连接到 VPC Amazon Direct Connect。有关信息 Amazon VPN，请参阅 Amazon VPC 用户指南中的 VPN 连接。有关 Amazon Direct Connect的信息，请参阅《Amazon Direct Connect 用户指南》中的创建连接。

SCEP 连接器不需要使用 Amazon PrivateLink，但我们建议将其作为额外的安全层。有关 Amazon PrivateLink 和 VPC 终端节点的更多信息，请参阅通过访问服务 Amazon PrivateLink。

SCEP VPC 终端节点连接器的注意事项

在为 SCEP 连接器设置接口 VPC 终端节点之前，请注意以下注意事项：

- SCEP 连接器可能不支持某些可用区中的 VPC 终端节点。创建 VPC 端点时，请先在管理控制台中查看是否支持。不支持的可用区标记为“此可用区不支持服务”。
- VPC 终端节点不支持跨区域请求。确保在创建连接器的同一区域创建终端节点。
- VPC 端点仅通过 Amazon Route 53 支持 Amazon 提供的 DNS。如果您希望使用自己的 DNS，可以使用条件 DNS 转发。有关更多信息，请参阅 Amazon VPC 用户指南中的 DHCP 选项集。

- 附加到 VPC 端点的安全组必须允许端口 443 上来自 VPC 的私有子网的传入连接。

为 SCEP 连接器创建 VPC 终端节点

您可以使用 VPC 控制台为适用于 SCEP 的连接器服务创建 VPC 终端节点。https://console.aws.amazon.com/vpc/ Amazon Command Line Interface有关更多信息，请参阅 Amazon VPC 用户指南中的创建接口终端节点程序。SCEP 连接器支持在您的 VPC 内调用其所有 API 操作。

创建终端节点时，请指定`com.amazonaws.region.pca-connector-scep`为服务名称。

如果您为终端节点启用了私有 DNS 主机名，那么 SCEP 终端节点的默认连接器现在会解析到您的 VPC 终端节点。有关默认服务终端节点的完整列表，请参阅服务终端节点和配额。

如果您尚未启用私有 DNS 主机名，则 Amazon VPC 将提供一个您可以使用的 DNS 端点名称，格式如下：

```
vpc-endpoint-id.pca-connector-scep.region.vpce.amazonaws.com
```

有关更多信息，请参阅 Amazon VPC 用户指南中的 VPC 终端节点 (Amazon PrivateLink)。

为 SCEP 连接器创建 VPC 终端节点策略

您可以为 Connector for SCEP 的 Amazon VPC 终端节点创建策略，以指定以下内容：

- 可执行操作的主体
- 可执行的操作
- 可对其执行操作的资源

有关更多信息，请参阅 Amazon VPC 指南中的使用 VPC 终端节点控制对服务的访问。

示例 — 用于 SCEP 操作的连接器的 VPC 终端节点策略

连接到端点后，以下策略授予所有委托人访问列出的连接器的权限，以便对指定连接器资源进行 SCEP 操作。

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
```

```
            "Action": [
               "pca-connector-scep:GetConnector",
               "pca-connector-scep:ListConnectors"
            ],
            "Resource": "arn:aws:pca-connector-scep:region:account:connector/connector-id"
         }
      ]
}
```

为连接器创建 VPC 终端节点以进行 SCEP 注册操作

SCEP 连接器为注册操作（例如`GetCACaps`和`PKIOperation`）提供单独的 VPC 终端节点服务。

创建注册端点时，请指定`com.amazonaws.region.pca-connector-scep.enroll`为服务名称。

创建连接器时，您可以选择指定，将连接器限制`VpcEndpointId`为只能通过该特定 VPC 终端节点进行访问。

如果您尚未启用私有 DNS 主机名，则 Amazon VPC 将提供一个您可以使用的 DNS 端点名称，格式如下：

```
vpc-endpoint-id.enroll.pca-connector-scep.region.vpce.amazonaws.com
```

> **Note**
>
> 要访问您的连接器，您必须使用连接器详细信息中包含的终端节点 URL，而不是直接使用 VPC 终端节点 DNS 名称。但是，您可以将连接器终端节点 URL 的 DNS 名称部分替换为任何有效的 VPC 终端节点 DNS 名称，例如 AZ-specific DNS 名称。
> 例如，要使用特定于可用区的 DNS 名称，您可以替换
>
> ```
> https://vpc-endpoint-id.enroll.pca-connector-
> scep.region.vpce.amazonaws.com/account-id-connector-id/UUID
> ```
>
> 用
>
> ```
> https://vpc-endpoint-id-availability-zone.enroll.pca-connector-
> scep.region.vpce.amazonaws.com/account-id-connector-id/UUID
> ```

示例-用于 SCEP 注册操作的连接器的 VPC 终端节点策略

您可以附加 VPC 终端节点策略来控制对注册操作的访问权限。当连接到终端节点时，以下策略授予所有委托人访问GetCACaps和PKIOperation操作的权限。节中的资源是一个连接器。

SCEP 注册操作的连接器未使用 Sigv4 进行身份验证。因此，它们不与 IAM 委托人关联，而是被 VPC 终端节点策略视为匿名的。因此，您的 VPC 终端节点策略必须允许所有委托人执行这些操作。

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "pca-connector-scep:GetCACaps",
            "pca-connector-scep:GetCACert",
            "pca-connector-scep:PKIOperation"
         ],
         "Resource": [
            arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566
         ]
      }
   ]
}
```

适用于 SCEP 的显示器连接器

监控是维护 Connector for SCEP 和其他 Amazon 解决方案的可靠性、可用性和性能的重要组成部分。Amazon 提供了以下监视工具，用于监视 Connector 的 SCEP，在出现问题时进行报告，并在适当时自动采取措施：

- Amazon CloudTrail 捕获由您的 Amazon Web Services 账户 或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 桶。您可以识别哪些用户和账户调用了 Amazon API、发出呼叫的源 IP 地址以及调用的发生时间。

  如果您监控 CloudTrail 数据事件，则日志将包含来自客户端设备的所有最近请求的列表。数据事件附带可识别的客户端设备信息，例如 IP 地址、执行的操作类型以及错误代码和详细消息（如果操作导致failed状态）。有关更多信息，请参阅 Amazon CloudTrail 《用户指南》。
- Amazon EventBridge 是一项无服务器事件总线服务，可以轻松地将您的应用程序与来自各种来源的数据连接起来。 EventBridge 提供来自您自己的应用程序、 Software-as-a-Service (SaaS) 应用程序和 Amazon 服务的实时数据流，并将这些数据路由到 Lambda 和 CloudWatch 日志等目标。这使您

能够监控服务中发生的事件，并构建事件驱动的架构。有关更多信息，请参阅 Amazon EventBridge 用户指南。

主题

- 使用 SCEP 自动连接器 EventBridge
- 使用 SCEP API 调用的日志连接器 Amazon CloudTrail

使用 SCEP 自动连接器 EventBridge

您可以使用 Amazon EventBridge 实现 Amazon 服务自动化，并自动响应系统事件，例如应用程序可用性问题或资源更改。来自 Amazon 服务的事件几乎实时 EventBridge 地传送到。您可以编写简单的规则来指明您感兴趣的事件，以及当事件与规则匹配时要采取的自动操作。 EventBridge 至少发布一次。有关更多信息，请参阅中的创建对事件做出反应的规则 EventBridge。

CloudWatch 使用将事件转化为操作 EventBridge。使用 EventBridge，您可以使用事件来触发目标。有关更多信息，请参阅什么是亚马逊 EventBridge？

SCEP 事件类型的连接器

证书颁发成功

EventBridge 当我们为响应PkiOperationPost请求而颁发证书时，SCEP 连接器会向发送一个Certificate Issuance Succeeded事件。

以下是该事件的示例数据。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Issuance Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
```

```
   ],
   "detail": {
       "result": "success",
       "requestType": "PkiOperationPost",
       "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   }
}
```

证书颁发失败

EventBridge 当我们无法为响应PkiOperationPost请求颁发证书时，SCEP 连接器会向发送一个Certificate Issuance Failed事件。

以下是该事件的示例数据。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Issuance Failed",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "failure",
       "requestType": "PkiOperationPost",
       "reason": "The certificate authority is not active."
   }
}
```

证书颁发机构证书检索成功

EventBridge 当我们收到GetCACert请求并成功检索连接器的私有 CA 证书时，SCEP 连接器会向发送一个Certificate Authority Certificate Retrieval Succeeded事件。

以下是该事件的示例数据。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "GetCACert"
   }
}
```

证书颁发机构证书检索失败

EventBridge 当我们收到GetCACert请求但无法检索连接器的私有 CA 证书时，SCEP 连接器会向发送一个Certificate Authority Certificate Retrieval Failed事件。该事件包括失败的原因。

以下是该事件的示例数据。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Failed",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
```

```
       "result": "failure",
       "requestType": "GetCACert",
       "reason": "The certificate authority certificate validity must be at least one
 year from today."
   }
}
```

证书颁发机构证书检索成功

EventBridge 当我们收到GetCACert请求并成功检索连接器的私有 CA 证书时，SCEP 连接器会向发送一个Certificate Authority Certificate Retrieval Succeeded事件。

以下是该事件的示例数据。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "GetCACert"
   }
}
```

证书颁发机构功能检索成功

EventBridge 当我们收到 SCEP GetCACaps 请求并成功检索 CA 的功能时，SCEP 连接器会向发送一个Certificate Authority Capabilities Retrieval Succeeded事件。

以下是该事件的示例数据。

证书颁发机构功能检索失败

EventBridge 当我们收到 SCEP GetCACaps 请求但无法检索 CA 的功能时，SCEP 连接器会向发送一个Certificate Authority Capabilities Retrieval Failed事件。我们在事件中注明失败的原因。

以下是该事件的示例数据。

```
{
 "resources":
     [
     "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
     "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector11223344-1234-1122-2233-112233445566"
     ],
 "detailType":"Certificate Authority Capabilities Retrieval Failed",
 "detail": {
     "result":"failure",
     "requestType":"GetCACaps",
     "reason":"The request was denied due to request throttling."
 },
 "source":"aws.pca-connector-scep","accountId":"111122223333"
 }
```

已调用不支持的操作

已调用不支持的操作

EventBridge 如果发送到连接器端点的操作不受支持或未知，则适用于 SCEP 的连接器会向发送Unsupported Operation Invoked事件。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Unsupported Operation Invoked",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
```

```
        "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
    ],
    "detail": {}
}
```

创建 EventBridge 规则

在中 EventBridge，您可以创建响应所记录的事件的规则 CloudTrail。要创建包含连接器为 SCEP 记录的所有事件的规则，请将源设置为。`aws.pca-connector-scep`有关规则的更多信息，请参阅在 Amazon 中创建规则 EventBridge。

使用 SCEP API 调用的日志连接器 Amazon CloudTrail

简单证书注册协议 (SCEP) 连接器与 Amazon CloudTrail一项服务集成，该服务提供用户、角色、客户机或 Amazon 服务所执行操作的记录。 CloudTrail 捕获所有 SCEP 连接器的 API 调用作为事件。捕获的调用包括来自连接器用于 SCEP 控制台的调用和对连接器进行 SCEP API 操作的代码调用。如果您创建跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括适用于 SCEP 的 Connector 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向 Connector 发出 SCEP 的请求、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅Amazon CloudTrail 用户指南。

用于存放 SCEP 信息的连接器 CloudTrail

CloudTrail 在您创建账户 Amazon Web Services 账户 时已在您的账户上启用。当 Connector for SCEP 中发生活动时，该活动会与其他 Amazon 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在中查看、搜索和下载最近发生的事件 Amazon Web Services 账户。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的事件 Amazon Web Services 账户，包括适用于 SCEP 的 Connector 的事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 Amazon Web Services 区域。跟踪记录 Amazon 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 Amazon 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail

- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

所有用于 SCEP 的连接器操作都由记录 CloudTrail 并记录在《适用于 SCEP 的连接器 API 参考手册》中。例如，调用`GetConnector`和`CreateChallenge`操作会在 CloudTrail 日志文件中生成条目。`CreateConnector`

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 Amazon Identity and Access Management (IAM) 用户凭证发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 Amazon 服务发出。
- 请求是否由 SCEP 客户端设备发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

SCEP 管理事件的连接器

SCEP 连接器与 CloudTrail 集成，用于记录用户、角色或 Amazon 服务在 SCEP 连接器中执行的 API 操作。您可以使用 CloudTrail 实时监控 Connector 的 SCEP API 请求，并将日志存储在亚马逊简单存储服务、亚马逊 CloudWatch 日志和亚马逊 CloudWatch 事件中。SCEP 连接器支持将以下操作作为事件记录在 CloudTrail 日志文件中：

- CreateChallenge
- CreateConnector
- GetConnector
- GetChallengeMetadata
- GetChallengePassword
- DeleteConnector
- DeleteChallenge

用于 SCEP 数据事件的连接器 CloudTrail

数据事件提供有关在资源上或资源中执行的资源操作的信息，例如，当您的客户端向连接器端点发送 SCEP `GetCACaps` 消息时。这些也称为数据面板操作。数据事件通常是高容量活动。默认情况下，CloudTrail 不记录任何数据事件， CloudTrail 事件历史记录也不记录这些事件。

记录数据事件将收取额外费用。有关 CloudTrail 定价的更多信息，请参阅Amazon CloudTrail 定价。

您可以使用 CloudTrail 控制台或 CloudTrail API 操作记录AWS::PCAConnectorSCEP::Connector资源类型的数据事件。 Amazon CLI有关如何记录数据事件的更多信息，请参阅《Amazon CloudTrail 用户指南》中的使用 Amazon Web Services 管理控制台记录数据事件和使用 Amazon Command Line Interface记录数据事件。

下表列出了您可以记录数据事件的 SCEP 连接器资源类型。数据事件类型（控制台）列显示要从控制 CloudTrail 台上的数据事件类型列表中选择的值。res ources.type 值列显示该resources.type值，您将在使用或 API 配置高级事件选择器时指定该值。 Amazon CLI CloudTrail “记录到的数据 API CloudTrail” 列显示了 CloudTrail 针对该资源类型记录的 API 调用。

数据事件类型（控制台）	resources.type 值	数据 API 已登录到 CloudTrail
连接器	AWS::PCAConnectorSCEP::Connector	• PKIOperationGet -如果向连接器的数据平面端点发出包含PKCSReq消息的 HTTP GET SCEP 请求，并且该消息的操作设置为，则生成。PKIOperation • PKIOperationPost -如果向连接器的数据平面端点发出包含PKCSReq消息的 HTTP POST SCEP 请求，并且该消息的操作设置为，则生成。PKIOperation • GetCACaps -如果向连接器的数据平面端点发出包含 GetCACaps 消息的 SCEP 请求，则生成。 • GetCACert -如果向连接器的数据平面端点发出包含 GetCACert 消息的 SCEP 请求，则生成。

您可以将高级事件选择器配置为在 eventName、readOnly 和 resources.ARN 字段上进行筛选，从而仅记录那些对您很重要的事件。以下示例是数据事件配置的 JSON 视图，该视图仅记录特定函数的事件。有关这些字段的更多信息，请参阅《Amazon CloudTrail API 参考》中的 AdvancedFieldSelector。

```
[
  {
    "name": "connector-scep-events",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
      {
        "field": "resources.type",
        "equals": [
          "AWS::PCAConnectorSCEP::Connector"
        ]
      },
      {
        "field": "resources.ARN",
        "equals": [
          "arn:aws:pca-connector-scep:US West (N.
 California):111122223333:connector/11223344-1122-2233-3344-cae95a00d2a7"
        ]
      }
    ]
  }
]
```

示例条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

示例 1：管理事件，**CreateConnector**

以下示例显示了演示该CreateConnector操作的 CloudTrail 日志条目。

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "111122223333",
                  "userName": "my-user-name"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:46:41Z",
                  "mfaAuthenticated": "false"
              }
          }
        },
        "eventTime": "2024-08-16T17:48:07Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "CreateConnector",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ClientToken": "11223344-2222-3333-4444-666555444555",
          "CertificateAuthorityArn": "arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
        },
        "responseElements": {
          "ConnectorArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
```

```
        "eventCategory": "Management"
        }
```

示例 2：管理事件，**CreateChallenge**

以下示例显示了演示该CreateChallenge操作的 CloudTrail 日志条目。

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "111122223333",
                  "userName": "user-name"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:46:41Z",
                  "mfaAuthenticated": "false"
              }
          }
        },
        "eventTime": "2024-08-16T17:47:52Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "CreateChallenge",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ConnectorArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
          "ClientToken": "11223344-2222-3333-4444-666555444555"
        },
        "responseElements": {
          "Challenge": {
```

```
            "Arn": "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/9cac40bc-acba-412e-9a24-f255ef2fe79a/a1b2c3d4-5678-90ab-
cdef-EXAMPLE22222",
            "ConnectorArn": "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "CreatedAt": 1723830472.942,
            "Password": "***",
            "UpdatedAt": 1723830472.942
          }
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```

示例 3：管理事件，**GetChallengePassword**

以下示例显示了演示该GetChallengePassword操作的 CloudTrail 日志条目。

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "905418114790",
                  "userName": "111122223333"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:55:01Z",
                  "mfaAuthenticated": "false"
              }
          },
```

```
          "invokedBy": "signin.amazonaws.com"
        },
        "eventTime": "2024-08-16T17:55:54Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "GetChallengePassword",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ChallengeArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:challenge/
a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
        },
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```


示例 4：数据事件，**PkiOperationPost**

以下示例显示了演示失败PkiOperationPost呼叫的 CloudTrail 日志条目。该日志包括错误代码和错误消息，并说明了失败。


```
{
        "eventVersion": "1.10",
        "userIdentity": {
          "type": "FederatedUser",
          "principalId": "111122223333",
          "accountId": "111122223333"
        },
        "eventTime": "2024-08-16T17:40:09Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "PkiOperationPost",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "errorCode": "BadRequestException",
```

```
        "errorMessage": "The certificate authority is not in a valid state for issuing
 certificates (Service: AcmPca, Status Code: 400, Request ID: a1b2c3d4-5678-90ab-cdef-
EXAMPLE55555)",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "resources": [
          {
              "accountId": "111122223333",
              "type": "AWS::PCAConnectorSCEP::Connector",
              "ARN": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
          }
        ],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "905418114790",
        "eventCategory": "Data",
        "tlsDetails": {
          "clientProvidedHostHeader": "111122223333-a1b2c3d4-5678-90ab-cdef-
EXAMPLE33333.enroll.pca-connector-scep.us-east-1.api.aws"
        }
        }
```


故障排除 Amazon 私有证书颁发机构 用于 SCEP 问题的连接器

您可能需要解决与 SCEP 实现连接器相关的问题。本章提供有关服务发送的 HTTP 和客户端错误的详细信息。

主题

- 对 SCEP 连接器的 HTTP 错误进行故障排除
- 对 SCEP 客户端错误进行连接器故障排除

对 SCEP 连接器的 HTTP 错误进行故障排除

当您的客户端触发 SCEP 数据平面连接器 API 操作并导致错误时，SCEP 连接器会向请求客户端发送包含错误信息的 HTTP 响应代码。

除了直接提供给客户端的服务响应外，您还可以使用适用于 SCEP 的显示器连接器本节中描述的监控工具来查看和调试导致 HTTP 错误的错误。

以下是服务向 SCEP 客户端返回的错误消息、潜在原因以及为解决这些问题可以采取的步骤。

HTTP 400 错误的请求

HTTP 400 响应代码意味着由于明显的客户端错误（例如请求中缺少数据或无效数据），SCEP 连接器无法处理请求。如果错误是由 SCEP-protocol 特定错误引起的，则 SCEP 连接器会将 SCEP 响应作为二进制文件包含在消息中。出于以下任何原因，适用于 SCEP API 的连接器可以返回 400 个响应。

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
LimitExceededException	已超过证书颁发机构颁发限制。	与连接器关联的私有证书颁发机构 (CA) 已超过其可以颁发的证书数量的配额。	SCEP 连接器在其生命周期内只能连接到一个私有 CA。如果您已用尽私有 CA 的限制，请创建新的连接器或申请增加配额。有关私有 CA 配额的更多信息，请参阅Amazon 私有证书颁发机构 配额。	否
ValidationException	请求必须包含 base64。	SCEP 连接器无法处理 HTTP GET 请求，因为正文无效 Base64。	如果可能，请将您的客户端配置为使用 HTTP POST 消息而不是 HTTP GET 消息。如果必须使用 HTTP GET，则消息必须使用 Base64 格式。如果您的客户不符合这些要求，请联系Amazon	否

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
			Web Services 支持以寻求帮助。	
ValidationException	证书颁发机构未激活。	与连接器关联的私有 CA 处于非活动状态。	重新激活私有 CA。有关信息，请参阅在中更新私有 CA Amazon 私有证书颁发机构。	否
ValidationException	从今天起，证书颁发机构证书的有效期必须至少为一年。	从今天起，与通用连接器关联的私有 CA 的有效期必须为一年。	从今天起补发有效期超过一年的证书。有关管理证书的信息，请参阅管理私有 CA 生命周期。	否
ValidationException	请求中包含的证书已过期。	客户端设备在每笔交易中生成的临时证书在服务收到时已过期。	很可能是您的客户端设备没有正确配置其时间设置，并且它们正在创建日期晚于实时的证书。如果您无法解决此问题，请联系Amazon Web Services 支持寻求帮助。	否

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
ValidationException	该请求包含无效的加密消息语法。	该服务无法解码 SCEP 请求消息。	检查您的 SCEP 消息是否符合 SCE P RFC 8894 中定义的加密消息语法。如果您无法解决此问题，请联系Amazon Web Services 支持寻求帮助。	否
ValidationException	连接器未激活。	连接器的状态为 “未激活”。	您可以在控制台或 API 的状态字段中找到连接器的状态。连接器的状态可以是创建、活动、正在删除或失败。如果状态为创建中，请稍后再试您的请求。如果状态为失败，请查看状态原因以解决问题，然后创建新的连接器。	否
ValidationException	申请中必须包含有效的证书。	客户端请求消息中包含的临时证书要么丢失，要么无效。	SCEP-compatible 客户端必须提供自签名证书才能对自己进行身份验证。如果您的客户无法提供所需的自签名证书，请联系Amazon Web Services 支持以寻求帮助。	否

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
ValidationException	请求的 URI 无效。	SCEP 连接器无法解析请求，因为请求的 URI 路径或查询无效。	管理员应验证客户端设备的配置设置，这些设备通常通过移动设备管理 (MDM) 系统进行管理。有关更多信息，请参阅 步骤 2：将连接器详细信息复制到 MDM 系统中。	否
ValidationException	请求中只需要一个主机标头。	客户端未在请求中提供有效的 HTTP Host 标头，这是处理请求所必需的。	需要使用 HTTP 主机标头来区分来自不同连接器的请求。如果您的客户端无法提供所需的 HTTP 主机标头，请联系Amazon Web Services 支持以寻求帮助。	否
ValidationException	无法对请求进行解码。请发送有效的 SCEP 请求。	该服务无法解码和处理您的客户端发送的加密消息语法 (CMS) 请求。	如果您的客户在我们实施 SCEP 时遇到问题，请记下回复中的请求 ID (`x-amzn-requestid`) 并联系我们Amazon Web Services 支持。	否

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
ValidationException	无法使用从请求中派生的值对响应进行编码。请发送有效的 SCEP 请求。	该服务无法对 SCEP 响应进行编码。	当服务无法使用提供的请求者证书对 SCEP 响应消息进行正确编码时，通常会出现此问题。例如，如果请求者证书具有椭圆曲线数字签名算法 (ECDSA) 密钥，而适用于 SCEP 的 Connector 不支持该密钥，则可能会发生这种情况。 如果遇到此问题，请先将 MDM 或 SCEP 客户端配置为使用 RSA。如果您仍然无法解决问题，请记下回复中的请求编号 (`x-amzn-requestid`)，并联系Amazon Web Services 支持寻求帮助。	否

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
ValidationException	不支持的算法：<OID>	该请求由不支持的加密算法签名或加密。	我们的服务不支持某些过时且较弱的加密算法。这些信息通过 `GetCACaps` 请求传达给客户。但是，某些客户端可能不会使用此方法来检查支持的算法。 如果您的客户似乎与我们的服务支持的加密算法不兼容，请联系Amazon Web Services 支持以寻求帮助。	否

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
ValidationException	不支持的 PkiOperation 消息类型。	请求消息包含无效的 `PkiOperation` 消息类型，因此服务无法处理。	我们的服务仅支持 RFC 8894 中定义的 SCEP 协议消息类型的子集。具体而言，我们会识别和处理以下消息类型：CertRep、pkcsReq、GetCert、getCRL 和。 CertPoll 我们通过 getCacaps 方法向客户端传达支持的消息类型。不幸的是，有些客户可能没有使用这种方法，并且可能不符合我们的服务能力。 如果您的客户似乎与我们的服务支持的 SCEP 消息类型不兼容，请联系Amazon Web Services 支持。	否

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
BadRequestException	质询密码无效。	客户端提供的质询密码对于已联系的服务端点及其关联的连接器无效。质询密码是 SCEP 协议中定义的一项必需安全措施，以确保只有经过授权的客户端才能访问该服务。	请确保您的客户在其请求中提供了正确的质询密码。您可以在控制台或通过 GetChallengePasswordAPI 的连接器详细信息中找到。有关更多信息，请参阅 步骤 2：将连接器详细信息复制到 MDM 系统中。	是
BadRequestException	证书签名请求中只需要一个质询密码。	客户端在其请求中提供了零个或多个质询密码。	请确保您的客户在其请求中提供了一个质询密码。您可以在控制台的连接器详细信息中或通过 GetChallengePasswordAPI 找到质询密码。有关更多信息，请参阅 步骤 2：将连接器详细信息复制到 MDM 系统中。	是
BadRequestException	连接器无权访问 Azure。	微软 Intune 连接器通过微软 Intune 授权客户请求。这需要你授予 SCEP 连接器访问你的 Azure 资源的权限。	配置权限，详见中第 1 步：授予 Amazon 私有 CA 使用你的 Microsoft Entra ID 应用程序的权限。	是

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
BadRequestException	Azure 应用程序没有执行权限<action>。	微软 Intune 连接器通过微软 Intune 授权客户请求。这需要你授予 SCEP 连接器访问你的 Azure 资源的权限。	配置权限，详见中第 1 步：授予 Amazon 私有 CA 使用你的 Microsoft Entra ID 应用程序的权限。	是
BadRequestException	找不到 Azure 应用程序。	微软 Intune 连接器通过微软 Intune 授权客户请求。此错误表示你的 Microsoft Entra ID 中没有应用程序注册，或者你的连接器的 Intune 详细信息配置错误。	按照为 SCEP 的 Connector 配置微软 Intune主题中的指导进行操作。	是
BadRequestException	Intune 证书签名请求验证失败。原因：<reason>	微软 Intune 连接器通过微软 Intune 授权客户请求。此错误消息表明 Intune 验证过程失败，并提供了相应的 Intune 错误代码。	按照为 SCEP 的 Connector 配置微软 Intune主题中的指导进行操作。如果问题仍然存在，请联系 Microsoft Support。	是

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
BadRequestException	<message type>不支持的 PkiOperation 消息类型：。	请求消息包含无效的消息类型，因此服务无法处理。	我们的服务仅支持 RFC 8894 中定义的 SCEP 协议消息类型的子集。具体而言，我们会识别和处理以下消息类型：CertRep、pkcsReq、GetCert、getCRL 和。 CertPoll 我们通过 getCacaps 方法向客户端传达支持的消息类型。不幸的是，有些客户可能没有使用这种方法，并且可能不符合我们的服务能力。 如果您的客户似乎与我们的服务支持的 SCEP 消息类型不兼容，请联系Amazon Web Services 支持。	是

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
BadRequestException	不支持密钥算法或长度。	该服务不支持证书签名请求中包含的提供的公钥。	我们的服务仅支持最多 16,384 位的标准 RSA 密钥和最大 521 位的 ECDSA 密钥。如果您的客户需要使用当前不支持的算法，请联系Amazon Web Services 支持以寻求帮助。	是

HTTP 401 未经授权

401 未授权响应状态代码指示客户端请求尚未完成，因为该请求缺少所请求资源的有效身份验证凭证。

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
AccessDeniedException	连接器无权访问证书颁发机构。	SCEP 的连接器无权访问连接器的关联私有 CA。	使用 Amazon Resource Access Manager与 SCEP 连接器共享您的私有 CA。	否
AccountDoesNotExistException	该 Amazon 账户不存在。	SCEP 的连接器资源已不存在。	拥有目标资源的账户已被删除。如果这是错误的，请在关闭后的 90 天Amazon Web Services 支持内联系。	否

未找到 HTTP 404

HTTP 404 响应代码通常意味着找不到您要查找的资源。

响应标头 (x-amzn-ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
ResourceNotFoundException	证书颁发机构不存在。	连接器的关联私有 CA 已被删除。	如果私有证书颁发机构 (CA) 被误删除，则可以在宽限期内将其恢复。有关更多信息，请参阅 恢复私有 CA。	否
ResourceNotFoundException	带有端点的连接器<URL>不存在。	客户端设备试图连接到不属于任何现有连接器的 URL。	确保您的客户端为连接器提供了正确的端点。要查看连接器Endpoint，请调用 GetConnectorAPI 或在控制台的连接器详细信息页面中查看。	否

HTTP 409 冲突

HTTP 409 冲突响应表示自请求启动以来，与连接器关联的私有 CA 已更改。

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
ConflictException	自请求发起以来，连接器已更改。	与连接器关联的私有 CA 已更新，从而触发连接器内部证书的轮	请过几分钟重试您的请求。如果问题仍未解决，请联系Amazon	否

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
		换，该证书用于通过 SCEP 与客户端设备通信。 在部署新证书时，这种证书轮换可能会在更新期间导致临时问题。但是，应及时自动解决此错误。	Web Services 支持寻求帮助。	

HTTP 429 请求太多

SCEP 的连接器在每个区域都有账户级别的配额。如果您超过了对连接器的请求限制，则您的请求将被拒绝，并出现 HTTP 429 错误。如果您需要增加配额，请参阅Amazon 私有证书颁发机构 终端节点和配额。

响应标头 (x-amzn-) ErrorType	错误消息 (x-amzn-) ErrorMessage	根本原因	修复	包括 SCEP 的回应？
ThrottlingException	由于请求限制而导致请求被拒绝。	已向此 Connector 发出的请求过多，导致某些请求被拒绝。 在部署新证书时，这种证书轮换可能会在更新期间导致临时问题。但是，应及时自动解决此错误。	如果您超过了对连接器的请求限制，则您的请求将被拒绝。如果您需要增加配额，请参阅适用于 SCEP 端点和配额的连接器。	否

对 SCEP 客户端错误进行连接器故障排除

使用以下指南对与 SCEP 连接器相关的客户端错误进行故障排除。

消息示例	根本原因	解决方案
不支持 ECDSA 密钥	连接器连接到使用 ECDSA 密钥而不是 RSA 的私有 CA。虽然此服务支持 ECDSA 密钥，但并非所有客户端设备都与该算法兼容。	考虑使用 RSA-encrypted 私有 CA 而不是 ECDSA。如果您创建使用 RSA 的私有 CA，则还需要创建一个新的连接器。一个连接器在其生命周期内只能绑定到一个私有 CA。
加密或签名证书不存在	根据 RFC 8894，SCEP 服务将中间 CA 证书返回给客户端。作为 SCEP 协议的一部分，客户端使用这些证书来执行加密和签名验证操作。 SCEP 连接器使用相同的证书进行加密和签名验证，这是一种常见的方法。但是，有些客户可能希望改为使用两个单独的证书。	如果您无法使用兼容的客户端，请联系Amazon Web Services 支持寻求帮助。

Amazon 私有 CA 服务配额

Amazon 私有 CA 为您允许的证书数量和证书颁发机构分配配额。API 操作的请求速率也受配额限制。Amazon 私有 CA 配额特定于 Amazon 账户和地区。

Amazon 私有 CA 根据 API 操作以不同的速率限制 API 请求。限制意味着 Amazon 私有 CA 拒绝原本有效的请求，因为该请求超过了操作的每秒请求数配额。当请求受到限制时，会 Amazon 私有 CA 返回错误。ThrottlingException Amazon 私有 CA 不能保证的最低请求速率 APIs。

要了解可以调整哪些配额，请参阅中的Amazon 私有 CA 配额表Amazon Web Services 一般参考。

您可以使用 Amazon 服务限额查看当前配额并请求增加配额。

查看您的 Amazon 私有 CA 配额 up-to-date列表

1. 登录您的 Amazon 账户。
2. 在 https://console.aws.amazon.com/servicequotas/ 打开 Service Quotas 控制台。
3. 在服务列表中，选择 Amazon Certificate Manager Private Certificate Authority（ACM PCA）。服务限额列表中的每个配额都会显示您当前应用的配额值、默认配额值以及配额是否可以调整。您可以选择配额的名称以了解有关该配额的详细信息。

要请求提高限额

1. 在服务限额列表中，选择可调整配额的单选按钮。
2. 选择请求增加配额按钮。
3. 填写并提交申请增加配额表格。

Amazon 私有 CA 已与 Amazon Certificate Manager。您可以使用 ACM 控制台或 ACM API 向现有私有 CA 申请私有证书。 Amazon CLI这些由 ACM 管理的私有 PKI 证书既受 PCA 配额的约束，也受 ACM 对公共证书和导入证书设置的配额的约束。有关 ACM 要求的更多信息，请参阅 Amazon Certificate Manager 用户指南中的申请私有证书和配额。

文档历史记录

下表介绍了自 2018 年 1 月起对此文档的一些重要更改。除了此处列出的主要更改以外，我们还会经常更新文档，以改进说明和示例以及处理您发送给我们的反馈意见。要获得有关重要更改的通知，请使用右上角的链接来订阅 RSS 源。

变更	说明	日期
更新了 CA 密钥管理的最佳实践	将 CA 密钥轮换最佳实践重组为延长 CA 有效期和轮换 CA 密钥的单独指南。有关更多信息，请参阅管理 CA 密钥和证书。	2026年4月10日
文档更新	使用新的入门程序、示例以及监控和故障 Amazon 私有证书颁发机构排除主题更新了 Secure Kubernetes。	2025 年 10 月 1 日
双栈支持	Amazon 私有证书颁发机构 支持双堆栈。	2025 年 6 月 23 日
适用于 AD 的 Connector 的子域支持现已正式推出	现在，您可以使用子域设置适用于 AD 的连接器。	2025 年 6 月 2 日
新的托管策略：AWSPrivateCAConnectoForKubernetesPolicy	引入了新的托管策略，用于适用于 Kubernetes 的 Amazon 私有 CA 连接器。	2025 年 5 月 19 日
已更新AWSPrivateCAPrivilegedUser AWSPrivateCAUser 和管理的策略	替换StringLike 为 ArnLike in AWSPrivateCAUser and AWSPrivateCAPrivilegedUser 。更新了模板 ARN，使其包含通配符arn:aws:acm-pca:::template 。arn:aws:acm-pca:*:*:template	2025 年 1 月 22 日

SCEP 连接器现已正式上市	SCEP 连接器现已正式上市。	2024 年 9 月 16 日
新的疑难解答主题	添加了一个新主题，可帮助您解决与更新 Active Directory 模板的连接器相关的问题。	2024 年 7 月 31 日
添加了如何更新 AD 模板的连接器	添加了一个描述如何更新适用于 AD 模板的连接器以及如何 Amazon 私有 CA 传播这些更新的过程。	2024 年 7 月 31 日
适用于 Omnissa Workspace ONE 的 SCEP 连接器现已正式上市	适用于 Omnissa Workspace ONE 的 SCEP 连接器现已正式上市。	2024 年 7 月 23 日
为审计报告添加了约束条件	Amazon 私有 CA 不支持将 Amazon S3 对象锁与用于审计报告的存储桶一起使用。	2024 年 7 月 3 日
现在 SM2 支持中国区域	Amazon 私有 CA 现在支持 SM2 签名算法，仅适用于中国地区。	2024 年 6 月 27 日
Amazon 私有 CA 现在支持 SCEP 连接器（预览）	使用 SCEP 连接器链接 Amazon 私有 CA 到支持 SCEP 的客户端和设备。	2024 年 6 月 11 日
新的连接器故障排除指南	添加了有关排除连接器和 SPN 创建失败故障的新章节。	2024 年 4 月 4 日
为 Matter 添加 CDP 扩展名	添加对 Matter 证书吊销列表分发点 (CDP) 扩展的支持。	2024 年 1 月 25 日
Amazon 私有 CA 对 mdL 的 API 支持	添加了 API 支持，用于创建符合 ISO/IEC 移动驾驶执照 (mDL) 标准的证书。	2024 年 1 月 16 日

Amazon 私有 CA 活动目录连接器	Connector for AD 的用户指南、API 和 CLI 支持。有关更多信息，请参阅 Connector for AD 文档。	2023 年 8 月 24 日
更改安全策略名称以匹配新服务名称	为指定标准权限的 Amazon 托管 IAM 策略采用新名称 Amazon 私有 CA。有关更多信息，请参阅 Amazon 托管策略。	2023 年 2 月 13 日
为 Amazon 托管策略添加更改跟踪器	添加了文档，用于跟踪指定标准权限的 Amazon 托管 IAM 策略的更改 Amazon 私有 CA。有关更多信息，请参阅更新 Amazon 私有 CA的 Amazon 托管策略。	2022 年 11 月 11 日
对颁发短期证书 CAs 的 API 和 CLI 支持	随着 CA 使用模式的引入，可以将 CA 配置为颁发通用证书或专门颁发短期证书。有关更多信息，请参阅证书颁发机构模式。	2022 年 10 月 24 日
服务品牌重塑和控制台更新	该服务已重命名为 Amazon 私有证书颁发机构 (Amazon 私有 CA)。 Amazon 私有 CA 控制台的可用性得到了改进，包括链接到完整文档的集成帮助面板。	2022 年 9 月 27 日
符合 Matter 标准的证书支持	三个新的证书模板增加了对符合 Matter 标准的 CA 和终端实体证书的支持。有关更多信息，请参阅了解证书模板。	2022 年 7 月 20 日

新区域支持	为亚太地区（雅加达）添加了端点。有关 Amazon 私有 CA 终端节点的完整列表，请参阅 ACM 私有证书颁发机构端点和配额。	2022 年 5 月 4 日
支持自定义属性和扩展	使用CustomAttribute对象配置自定义证书 CAs 和证书，使用CustomExtension 对象配置自定义证书。	2022 年 3 月 16 日
支持托管 OCSP	有关包括 OCSP 在内的吊销选项，请参阅设置证书吊销方法。	2021 年 8 月 18 日
Support 支持 S3 阻止公共访问功能 CRLs	请参阅启用 S3 屏蔽公共访问权限功能。	2021 年 5 月 27 日
新的和更新的 Java 实现示例	请参阅使用 ACM Private CA API（Java 示例）。	2020 年 9 月 9 日
新区域支持	为非洲（开普敦）和欧洲地区（米兰）添加了端点。有关 Amazon 私有 CA 端点的完整列表，请参阅 Amazon Certificate Manager Private Certificate Authority 端点和配额。	2020 年 8 月 27 日
支持跨账户私有 CA 访问	Amazon Certificate Manager 可以授权用户使用他们不拥有的私 CAs 有证书颁发证书。有关更多信息，请参阅跨账户访问私人 CAs账户。	2020 年 8 月 17 日

VPC 终端节点 (PrivateLink) 支持	增加了对使用 VPC 终端节点 (Amazon PrivateLink) 的支持，以增强网络安全。有关更多信息，请参阅 ACM 私有 CA VPC 终端节点 (Amazon PrivateLink)。	2020 年 3 月 26 日
添加了专用安全性部分	的安全文档 Amazon 已合并到专门的安全部分。有关安全性的信息，请参阅Amazon Certificate Manager 私有证书颁发机构的安全。	2020 年 3 月 26 日
模板 ARN 已添加到审计报告。	有关更多信息，请参阅为您的私有 CA 创建审计报告。	2020 年 3 月 6 日
CloudFormation 支持	添加了对 Support 的支持 Amazon CloudFormation。有关更多信息，请参阅《 Amazon CloudFormation User Guide》中 ACMPCA Resource Type Reference。	2020 年 1 月 22 日
CloudWatch 活动集成	与异步 CloudWatch 事件集成，包括 CA 创建、证书颁发和 CRL 创建。有关更多信息，请参阅使用 CloudWatch 事件。	2019 年 12 月 23 日
FIPS 端点	为 Amazon GovCloud （美国东部）和 Amazon GovCloud （美国西部）添加了 FIPS 端点。有关 Amazon 私有 CA 终端节点的完整列表，请参阅Amazon Certificate Manager 私有证书颁发机构端点和配额。	2019 年 12 月 13 日

基于标签的权限	使用新的 APIs TagResource UntagResource 、和ListTagsForResource 支持基于标签的权限。有关基于标签的控制的一般信息，请参阅使用 IAM 资源标签控制对 IAM 用户和角色的访问以及这些用户和角色的访问权限。	2019 年 11 月 5 日
名称约束强制实施	添加了对在导入的 CA 证书上强制实施主题名称约束的支持。有关更多信息，请参阅在私有 CA 上强制实施名称约束。	2019 年 10 月 28 日
新证书模板	添加了新的证书模板，包括用于进行代码签名的模板 Amazon Signer。有关更多信息，请参阅使用模板。	2019 年 10 月 1 日
规划您的 CA	添加了关于使用 Amazon 私有 CA规划 PKI 的新部分。有关更多信息，请参阅规划 ACM 私有 CA 部署。	2019 年 9 月 30 日
增加了区域支持	增加了对 Amazon 亚太地区（香港）区域的区域支持。有关受支持区域的完整列表，请参阅 Amazon Certificate Manager Private Certificate Authority 端点和配额。	2019 年 7 月 24 日
添加了完整的私有 CA 层次结构支持	Support 对创建和托管 root 的支持 CAs 消除了对外部父级的需求。	2019 年 6 月 20 日

增加了区域支持	增加了对 Amazon GovCloud（美国西部和美国东部）区域的区域支持。有关受支持区域的完整列表，请参阅 Amazon Certificate Manager Private Certificate Authority 端点和配额。	2019 年 5 月 8 日
增加了区域支持	增加了对 Amazon 亚太地区（孟买和首尔）、美国西部（加利福尼亚北部）和欧洲（巴黎和斯德哥尔摩）区域的区域支持。有关受支持区域的完整列表，请参阅 Amazon Certificate Manager Private Certificate Authority 端点和配额。	2019 年 4 月 4 日
测试证书续订工作流程	客户现在可以手动测试 ACM 托管续订工作流的配置。有关更多信息，请参阅测试 ACM 的托管续订配置。	2019 年 3 月 14 日
增加了区域支持	增加了对 Amazon 欧洲（伦敦）区域的区域支持。有关受支持区域的完整列表，请参阅 Amazon Certificate Manager Private Certificate Authority 端点和配额。	2018 年 8 月 1 日
恢复已删除 CAs	私有 CA 恢复允许客户在证书颁发机构 (CAs) 被删除后最长 30 天内将其恢复。有关更多信息，请参阅还原私有 CA。	2018 年 6 月 20 日

早期更新

下表描述了 2018 年 6 月 Amazon 私有证书颁发机构 之前的文档发布历史。

更改	描述	日期
新指南	此版本引入了。 Amazon 私有证书颁发机构	2018 年 04 月 4 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。