亚马逊云科技

用户指南

Amazon ECR

API 版本 2015-09-21

Amazon ECR: 用户指南

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 Elastic Container Registry？

Amazon Elastic Container Registry (Amazon ECR) Amazon 是一项安全、可扩展且可靠的托管容器镜像注册服务。Amazon ECR 使用 Amazon IAM 支持具有基于资源的权限的私有存储库。这样，指定的用户或 Amazon EC2 实例就可以访问您的容器存储库和镜像。您可以使用首选 CLI 推送、提取和管理 Docker 映像、Open Container Itistry (OCI) 映像和 OCI 兼容构件。

> Note
>
> Amazon ECR 也支持公共容器映像存储库。有关更多信息，请参阅 Amazon ECR Public 用户指南中的什么是 Amazon ECR Public。

Amazon 容器服务团队维护着一个公开的路线图 GitHub。它包含有关团队正在做的事情的信息，并允许所有 Amazon 客户直接提供反馈。有关更多信息，请参阅 Amazon 容器路线图。

Amazon ECR 的概念和组件

Amazon ECR 是 Amazon提供的一项完全托管式 Docker 容器注册表服务。它允许您安全可靠地存储、管理和部署 Docker 容器映像。这些概念和组件共同提供安全、可扩展和可靠的 Docker 容器注册表服务 Amazon，使您能够高效地管理和部署容器化应用程序。

以下是 Amazon ECR 的一些关键概念和组件：

注册表

Amazon ECR 注册表是提供给每个 Amazon 账户的私有存储库，您可以在其中创建一个或多个存储库。这些存储库允许您在环境中存储和分发 Docker 镜像、开放容器计划 (OCI) 镜像和其他与 OCI 兼容的工件。 Amazon 有关更多信息，请参阅 Amazon ECR 私有注册表。

授权令牌

您的客户端必须作为 Amazon 用户对 Amazon ECR 私有注册表进行身份验证，然后才能推送和提取映像。有关更多信息，请参阅 Amazon ECR 中的私有注册表身份验证。

存储库

Amazon ECR 中的存储库是一个逻辑集合，您可以在其中存储 Docker 映像、Open Container Initiative（OCI）映像和其他与 OCI 兼容的构件。在单个 Amazon ECR 注册表中，您可以有多个存储库来整理容器映像。有关更多信息，请参阅 Amazon ECR 私有存储库。

存储库策略

您可以通过存储库策略来控制对存储库及其中的内容的访问。有关更多信息，请参阅 Amazon ECR 中的私有存储库策略。

图像

您可以对存储库推送和提取容器映像。这些映像可以在开发系统中本地使用，也可以在 Amazon ECS 任务定义和 Amazon EKS Pod 规范中使用。有关更多信息，请参阅将 Amazon ECR 映像与 Amazon ECS 结合使用 和将 Amazon ECR 映像与 Amazon EKS 结合使用。

生命周期策略

Amazon ECR 生命周期策略允许您通过定义旧映像或未使用映像的修剪和过期规则来管理映像的生命周期。有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像。

映像扫描

Amazon ECR 提供一项集成的映像扫描功能，以帮助识别容器映像中的软件漏洞。有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在软件漏洞。

访问控制

Amazon ECR 使用 IAM 来控制对您存储库的访问。您可以创建具有推送、提取或管理 Amazon ECR 存储库的特定权限的 IAM 用户、组和角色。有关更多信息，请参阅 Amazon Elastic Container Registry 中的安全性。

跨账户和跨区域复制

Amazon ECR 支持跨多个 Amazon 账户和区域复制映像，以提高可用性并减少延迟。有关更多信息，请参阅 Amazon ECR 中的私有映像复制。

加密

Amazon ECR 支持使用 Amazon KMS对 Docker 映像进行静态服务器端加密。有关更多信息，请参阅 Amazon ECR 中的数据保护。

Amazon Command Line Interface 集成

Amazon CLI 提供了与 Amazon ECR 存储库交互的命令，例如创建、列出、推送和提取映像。

Amazon Web Services Management Console

Amazon ECR 也可以通过进行管理 Amazon Web Services Management Console，它提供了一个用户友好的网页界面，便于您处理存储库和映像。

Amazon CloudTrail

> 出于安全和合规目的 Amazon CloudTrail，Amazon ECR 与集成，允许您记录和审核向 Amazon ECR 发出的 API 调用。有关更多信息，请参阅 使用记录 Amazon ECR 操作 Amazon CloudTrail。

Amazon CloudWatch

> Amazon ECR 提供可使用 Amazon CloudWatch监控的指标和日志，使您能够跟踪 Amazon ECR 存储库的性能和使用情况。有关更多信息，请参阅 Amazon ECR 存储库指标。

Amazon ECR 中的常见用例

Amazon ECR 是 Amazon提供的一项完全托管式 Docker 容器注册表服务。它提供了一个安全且可扩展的存储库，用于存储和分发 Docker 容器映像，使其成为容器化应用程序部署中必不可少的组件。Amazon ECR 简化了在各种 Amazon 服务和本地环境中构建、分发和运行容器化应用程序的过程。

以下是 Amazon ECR 的一些关键用例：

容器映像存储和分发

> Amazon ECR 充当集中存储库，以在组织内存储和分发 Docker 容器映像或供公众使用。开发人员可以将其容器映像推送到 Amazon ECR，然后从其中的 Amazon任何计算环境（例如亚马逊 EC2或 Amazon EKS）中提取这些映像。 Amazon Fargate有关更多信息，请参阅 Amazon ECR 私有存储库。

持续集成和持续部署（CI/CD）

> Amazon ECR 与 Amazon CodeBuild Amazon CodePipeline、和其他CI/CD tools, enabling automated building, testing, and deployment of containerized applications. Container images can be automatically pushed to Amazon ECR as part of the CI/CD管道无缝集成，确保在不同环境中进行一致和可靠的部署。

微服务架构

> Amazon ECR 非常适合用于微服务架构，在这种微服务架构中，应用程序被分解成更小的解耦服务，打包成容器。每个微服务都可以将自己的容器映像存储在 Amazon ECR 中，从而实现各个服务的独立开发、部署和扩展。

混合云和多云部署

Amazon ECR 支持从其他容器注册表（例如 Docker Hub 或第三方注册表）中提取容器映像。这样允许组织使用 Amazon ECR 作为容器映像的中央存储库，在混合云或多云环境中保持一致的部署模式。

访问控制和安全性

Amazon ECR 提供了精细的访问控制机制，允许组织控制谁可以从注册表中推送或提取容器映像。它还与集成 Amazon Identity and Access Management 以进行身份验证和授权，从而确保对容器镜像的安全访问。有关更多信息，请参阅 Amazon Elastic Container Registry 中的安全性。

映像漏洞扫描

Amazon ECR 可自动扫描容器映像，以发现软件漏洞和潜在的配置错误，从而帮助维护安全合规的容器环境。有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在软件漏洞。

私有容器注册表

对于具有严格安全或合规要求的组织，Amazon ECR 可用作私有容器注册表，确保敏感的容器映像不会暴露给公共注册表，并且只能在组织的 Amazon 环境中访问。有关更多信息，请参阅 Amazon ECR 私有注册表。

使用 Amazon ECR 复制功能实现全球分布式应用程序部署

利用 Amazon ECR 复制功能，您可以将容器化 Web 应用程序映像集中到主存储库中，从而实现跨多个 Amazon 区域的自动分发，确保全球部署一致且延迟低，并减轻运营负担。有关更多信息，请参阅 Amazon ECR 中的私有映像复制

自动清理过时的容器映像

Amazon ECR 生命周期策略支持根据已定义的规则（例如使用年限、计数或标签）自动清理过时的容器映像，优化存储成本，维护有序的注册表，增强安全性和合规性，并通过自动化简化开发工作流。有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像

Amazon ECR 的功能

Amazon ECR 提供以下功能：

- 生命周期策略有助于管理存储库中映像的生命周期。您可以定义导致清理未使用映像的规则。您可以在将规则应用到存储库之前对其进行测试。有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像。

- 映像扫描有助于识别容器映像中的软件漏洞。每个存储库都可以配置为在推送时扫描。这可确保扫描推送到存储库的每个新映像。然后，您可以检索映像扫描的结果。有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在软件漏洞。
- 跨区域和跨账户复制使您可以更轻松地将映像放置在需要的位置。它配置为注册表设置，并基于每个区域。有关更多信息，请参阅 Amazon ECR 中的私有注册表设置。
- 缓存提取规则提供了在私有 Amazon ECR 注册表中缓存上游注册表中的存储库的方法。使用缓存提取规则时，Amazon ECR 将定期访问上游注册表，以确保 Amazon ECR 私有注册表中缓存的映像为最新状态。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步。

如何开始使用 Amazon ECR

如果您使用 Amazon Elastic Container Service（Amazon ECS）或 Amazon Elastic Kubernetes Service（Amazon EKS），请注意，这两项服务的设置与 Amazon ECR 的设置类似，因为 Amazon ECR 是这两项服务的扩展。

Amazon Command Line Interface 与 Amazon ECR 一起使用时，请使用支持最新 Amazon ECR 功能的版本。 Amazon CLI 如果您在中看不到对 Amazon ECR 功能的支持 Amazon CLI，请升级到最新版本的 Amazon CLI。有关安装最新版本的信息 Amazon CLI，请参阅《Amazon Command Line Interface 用户指南》 Amazon CLI中的安装或更新到最新版本的。

要了解如何使用 Amazon CLI 和 Docker 将容器映像推送到私有 Amazon ECR 存储库中，请参阅在 Amazon ECR 中移动映像的整个生命周期。

Amazon ECR 定价

使用 Amazon ECR，您只需为存储库中存储的数据量以及从映像推送和提取所传输的数据付费。有关更多信息，请参阅 Amazon ECR 定价。

Amazon ECR 中国区域

本指南中的示例反映了世界不同部分的区域。对于中国，将这些区域替换为以下两个中国区域之一：cn-northwest-1 或 cn-north-1。此外，由于在中国只有两个区域，因此与三个或以上区域相关的任何问题都不适用于中国。

在 Amazon ECR 中移动映像的整个生命周期

如果您是首次使用 Amazon ECR，请在 Docker CLI 和中使用以下步骤 Amazon CLI 来创建示例映像、向默认注册表进行身份验证并创建私有存储库。然后将映像推送到私有存储库并从中提取映像。完成示例映像后，删除示例映像和存储库。

要使用代 Amazon Web Services Management Console 替 Amazon CLI，请参阅the section called “创建存储库以存储映像”。

有关可用于管理 Amazon 资源的其他工具（包括不同 Amazon SDKs的 IDE 工具包和 Windows PowerShell 命令行工具）的更多信息，请参阅 http://aws.amazon.com/tools/。

先决条件

如果您尚未安装最新版本 Amazon CLI 和 Docker，也未准备就绪，请按照以下步骤安装这两个工具。

安装 Amazon CLI

要将 Amazon ECR Amazon CLI 与 Amazon ECR 一起使用，请安装最新 Amazon CLI 版本。有关信息，请参阅《Amazon Command Line Interface 用户指南》中的安装 Amazon Command Line Interface。

安装 Docker

Docker 适用于许多不同的操作系统，包括大多数现代 Linux 分发版 (如 Ubuntu) 甚至 MacOS 和 Windows。有关如何在特定的操作系统上安装 Docker 的更多信息，请转到 Docker 安装指南。

您无需本地开发系统即可使用 Docker。如果你 EC2已经在使用亚马逊，你可以启动亚马逊 Linux 2023 实例并安装 Docker 开始使用。

如果您已安装 Docker，请跳到步骤 1：创建 Docker 镜像。

使用亚马逊 Linux 2023 AMI 在亚马逊 EC2 实例上安装 Docker

1. 使用最新版 Amazon Linux 2023 AMI 启动实例。有关更多信息，请参阅 Amazon EC2 用户指南中的启动实例。
2. 连接到您的实例。有关更多信息，请参阅《亚马逊 EC2 用户指南》中的 “连接到您的 Linux 实例”。

3. 更新实例上已安装的程序包和程序包缓存。

   ```
   sudo yum update -y
   ```

4. 安装最新的 Docker Community Edition 程序包。

   ```
   sudo yum install docker
   ```

5. 启动 Docker 服务。

   ```
   sudo service docker start
   ```

6. 将 `ec2-user` 添加到 `docker` 组，以便您能够执行 Docker 命令，而无需使用 sudo。

   ```
   sudo usermod -a -G docker ec2-user
   ```

7. 退出，再重新登录以接受新的 `docker` 组权限。您可以关闭当前的 SSH 终端窗口并在新终端窗口中重新连接到实例，完成这一过程。您的新 SSH 会话将具有相应的 `docker` 组权限。

8. 验证 `ec2-user` 是否能在没有 sudo 的情况下运行 Docker 命令。

   ```
   docker info
   ```

   > **Note**
   >
   > 在某些情况下，您可能需要重新启动实例，以便为 `ec2-user` 提供访问 Docker 进程守护程序的权限。如果您看到以下错误，请尝试重启您的实例：
   >
   > ```
   > Cannot connect to the Docker daemon. Is the docker daemon running on this
   >  host?
   > ```

步骤 1：创建 Docker 镜像

在此步骤中，您将创建一个简单 Web 应用程序的 Docker 镜像，然后在本地系统或 Amazon EC2 实例上对其进行测试。

创建简单 Web 应用程序的 Docker 镜像

1. 创建名为 `Dockerfile` 的文件。Dockerfile 是一个清单文件，描述了用于 Docker 镜像的基本镜像以及要安装的项目以及在此项目上运行的内容。有关 Dockerfile 的更多信息，请转到 Dockerfile 参考。

   ```
   touch Dockerfile
   ```

2. 编辑您刚刚创建的 `Dockerfile` 并添加以下内容。

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest

   # Install dependencies
   RUN yum update -y && \
    yum install -y httpd

   # Install apache and write hello world message
   RUN echo 'Hello World!' > /var/www/html/index.html

   # Configure apache
   RUN echo 'mkdir -p /var/run/httpd' >> /root/run_apache.sh && \
    echo 'mkdir -p /var/lock/httpd' >> /root/run_apache.sh && \
    echo '/usr/sbin/httpd -D FOREGROUND' >> /root/run_apache.sh && \
    chmod 755 /root/run_apache.sh

   EXPOSE 80

   CMD /root/run_apache.sh
   ```

   此 Dockerfile 使用 Amazon ECR Public 上托管的 Amazon Linux 2 公有映像。RUN 指令更新包缓存，安装一些适用于 Web 服务器的软件包，然后将“Hello World!” 内容写入 Web 服务器的文档根目录。EXPOSE 指令在容器上公开端口 80，CMD 指令启动 Web 服务器。

3. 从您的 Dockerfile 生成 Docker 镜像。

   > **Note**
   >
   > Docker 的某些版本可能需要在以下命令中使用 Dockerfile 完整路径，而不是所示的相对路径。

```
docker build -t hello-world .
```

4. 列出容器映像。

```
docker images --filter reference=hello-world
```

输出：

```
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago
 194MB
```

5. 运行新构建的镜像。`-p 80:80` 选项将容器上公开的端口 80 映射到主机系统上的端口 80。有关 docker run 的更多信息，请转到 Docker 运行参考。

```
docker run -t -i -p 80:80 hello-world
```

> **Note**
>
> 来自 Apache Web 服务器的输出将显示在终端窗口中。您可以忽略“`Could not reliably determine the fully qualified domain name`”消息。

6. 打开浏览器并指向正在运行 Docker 并托管您的容器的服务器。

   - 如果您使用的是 EC2 实例，则这是服务器的公有 DNS 值，与您通过 SSH 连接实例时使用的地址相同。确保实例的安全组允许端口 80 上的入站流量。
   - 如果您正在本地运行 Docker，可将您的浏览器指向 http://localhost/。
   - 如果你在 Windows 或 Mac 计算机docker-machine上使用，请使用docker-machine ip命令查找托管 Docker 的 VirtualBox 虚拟机的 IP 地址，替换为你正在*`machine-name`*使用的 docker 计算机的名称。

   ```
   docker-machine ip machine-name
   ```

您应看到一个显示“Hello World!”语句的 网页。

7. 通过键入 Ctrl + c 来停止 Docker 容器。

步骤 2：创建存储库

现在您已拥有可推送到 Amazon ECR 的镜像，还必须创建一个存储库来保存它。在本示例中，您创建一个名称为 `hello-repository` 的存储库，稍后将推送 `hello-world:latest` 镜像到这里。要创建存储库，请运行以下命令：

```
aws ecr create-repository \
    --repository-name hello-repository \
    --region region
```

步骤 3：向您的默认注册表验证身份

安装并配置完毕后 Amazon CLI，请使用默认注册表对 Docker CLI 进行身份验证。这样一来，docker 命令可以通过 Amazon ECR 推送和提取镜像。 Amazon CLI 提供了简化身份验证过程的get-login-password命令。

要使用向 Amazon ECR 注册表对 Docker 进行身份验证 get-login-password，请运行命令。aws ecr get-login-password将身份验证令牌传递给 docker login 命令时，将值 `AWS` 用作用户名，并指定要对其进行身份验证的 Amazon ECR 注册表 URI。如果对多个注册表进行身份验证，则必须针对每个注册表重复该命令。

> ⚠ Important
>
> 如果收到错误，请安装或更新到最新版本的 Amazon CLI。有关更多信息，请参阅Amazon Command Line Interface 《用户指南》中的安装 Amazon Command Line Interface。

- get-login-password (Amazon CLI)

  ```
  aws ecr get-login-password --region region | docker login --username AWS --password-
  stdin aws_account_id.dkr.ecr.region.amazonaws.com
  ```

- Get-ECRLogin 命令 (Amazon Tools for Windows PowerShell)

  ```
  (Get-ECRLoginCommand).Password | docker login --username AWS --password-
  stdin aws_account_id.dkr.ecr.region.amazonaws.com
  ```

步骤 4：推送镜像到 Amazon ECR

现在您可以推送镜像到上一部分中创建的 Amazon ECR 存储库。在满足以下先决条件后，使用 docker CLI 推送映像：

- 安装最低版本的 docker：1.7。
- 已使用 docker login 配置 Amazon ECR 授权令牌。
- Amazon ECR 存储库存在且用户有向该存储库推送的权限。

在满足这些先决条件后，即可将镜像推送到您在帐户的默认注册表中新创建的存储库中。

标记镜像并推送到 Amazon ECR

1. 列出您存储在本地的镜像，以识别要标记和推送的镜像。

```
docker images
```

输出：

```
REPOSITORY          TAG                 IMAGE ID            CREATED
 VIRTUAL SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago
 241MB
```

2. 标记镜像并推送到存储库。

```
docker tag hello-world:latest aws_account_id.dkr.ecr.region.amazonaws.com/hello-
repository
```

3. 推送镜像。

```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository:latest
```

输出：

```
The push refers to a repository [aws_account_id.dkr.ecr.region.amazonaws.com/hello-
repository] (len: 1)
e9ae3c220b23: Pushed
a6785352b25c: Pushed
```

```
0998bf8fb9e9: Pushed
0a85502c06c9: Pushed
latest: digest: sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636EXAMPLE
 size: 6774
```

步骤 5：从 Amazon ECR 提取镜像

在推送映像到 Amazon ECR 存储库后，可以从其他位置提取该映像。在满足以下先决条件后，使用 docker CLI 提取映像：

- 安装最低版本的 docker：1.7。
- 已使用 docker login 配置 Amazon ECR 授权令牌。
- Amazon ECR 存储库存在且用户有从该存储库提取的权限。

在满足这些先决条件后，即可提取您的镜像。要从 Amazon ECR 提取示例镜像，请运行以下命令：

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository:latest
```

输出：

```
latest: Pulling from hello-repository
0a85502c06c9: Pull complete
0998bf8fb9e9: Pull complete
a6785352b25c: Pull complete
e9ae3c220b23: Pull complete
Digest: sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636EXAMPLE
Status: Downloaded newer image for aws_account_id.dkr.region.amazonaws.com/hello-
repository:latest
```

步骤 6：删除镜像

如果您不再需要一个存储库中的某个映像，则可以删除该映像。要删除映像，请指定它所在的存储库，并指定映像的 `imageTag` 或 `imageDigest` 值。以下示例删除 `hello-repository` 存储库中映像标签为 `latest` 的映像。要从存储库中删除示例映像，请运行以下命令：

```
aws ecr batch-delete-image \
      --repository-name hello-repository \
```

```
      --image-ids imageTag=latest \
      --region region
```

步骤 7：删除存储库

如果您不再需要整个映像存储库，您可以删除该存储库。以下示例使用 `--force` 标签删除包含映像的存储库。要删除包含映像的存储库 (及其中的所有映像)，请运行以下命令：

```
aws ecr delete-repository \
      --repository-name hello-repository \
      --force \
      --region region
```

优化 Amazon ECR 的性能

您可以使用以下关于设置和策略的建议在使用 Amazon ECR 时优化性能。

使用 Docker 1.10 及以上版本可利用同时层上传

Docker 镜像由层组成，是镜像的中间构建阶段。Dockerfile 的每一行都会创建新层。当使用 Docker 1.10 及以上版本时，Docker 在默认情况下会在上传至 Amazon ECR 的同时推送尽可能多的层，从而缩短上传时间。

使用较小基本镜像

通过 Docker Hub 提供的默认镜像，可能包含您的应用程序不需要的很多依赖项。请考虑使用其他人在 Docker 社区创建并维护的较小镜像，或使用 Docker 最小 Scratch 镜像构建您自己的基本镜像。有关更多信息，请参阅 Docker 文档中的创建基本镜像。

更早将更改最少的依赖性放入您的 Dockerfile

Docker 缓存层，可加速构建时间。如果从最后一次构建至今，某一层上没有任何更改，则 Docker 将使用缓存版本，而不重新构建层。但是，每层都依赖之前出现的层。如果层发生更改，则 Docker 不仅重新编译该层，也会重新编译该层之后出现的所有层。

为了尽量缩短重新构建 Dockerfile 并重新上传层所需的时间，可考虑早些时候将更改频率最低的依赖项放入 Dockerfile。将经常更改的依赖项 (如应用程序的源代码) 稍后放入堆栈。

链接命令以避免不必要文件的存储

在层中创建的中间文件会作为该层的一部分保留，即使该层在后续层中被删除。考虑以下示例：

```
WORKDIR /tmp
RUN wget http://example.com/software.tar.gz
RUN wget tar -xvf software.tar.gz
RUN mv software/binary /opt/bin/myapp
RUN rm software.tar.gz
```

在本示例中，第一个和第二个 RUN 命令创建的层包含原始 .tar.gz 文件及其所有解压内容。即使第四个 RUN 命令已删除 .tar.gz 文件。这些命令可以链接在一起，构成单独的运行语句，以确保最终 Docker 镜像中不包含不必要的文件。

```
WORKDIR /tmp
RUN wget http://example.com/software.tar.gz &&\
    wget tar -xvf software.tar.gz &&\
```

```
    mv software/binary /opt/bin/myapp &&\
    rm software.tar.gz
```

使用最近的区域终端节点

通过确保使用最靠近所运行应用程序的区域终端节点，可以减少从 Amazon ECR 提取镜像的延迟。如果您的应用程序在 Amazon EC2 实例上运行，则可以使用以下 shell 代码从该实例的可用区获取该区域：

```
REGION=$(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone
 |\
     sed -n 's/\(\d*\)[a-zA-Z]*$/\1/p')
```

可以使用--region参数将该区域传递给 Amazon CLI 命令，也可以使用命令将该区域设置为配置文件的默认区域。aws configure您还可以在使用 Amazon SDK 拨打电话时设置区域。有关更多信息，请参阅适用于特定编程语言的软件开发工具包文档。

向 Amazon ECR 注册管理机构提出请求

您可以使用 IPv4仅限终端节点或双堆栈（和）终端节点在 Amazon ECR 私有注册表中推送、拉取、删除、查看和管理 OCI 映像、Docker 映像和兼容 OCI 的项目。IPv4 IPv6要从 IPv4 网络发出请求，您可以使用双堆栈或 IPv4 终端节点。要从 IPv6 网络发出请求，请使用双堆栈终端节点。有关使用 IPv4 和双栈终端节点向 Amazon ECR 公共注册机构发出请求的更多信息，请参阅向 Amazon ECR 公共注册管理机构发出请求。访问 Amazon ECR 无需支付任何额外费用。 IPv6有关定价的更多信息，请参阅 Amazon 弹性容器注册表定价。

> Note
>
> Amazon ECR 不支持通过双堆栈 Amazon PrivateLink 终端节点传输流量。如果您需要 Amazon PrivateLink 支持，则必须使用 IPv4仅限使用的 Amazon ECR 终端节点。

Amazon ECR 终端节点由超出 IPv4仅限终端节点或双栈终端节点支持的属性指定。这些属性可以包括：

- 区域-每个终端节点都特定于一个区域。
- 类型 — 端点的选择取决于你使用的是软件开发工具包还是兼容 OCI 和 Amazon Docker 命令行接口。
- 安全 — 在部分地区，Amazon ECR 提供符合 FIPS 标准的终端节点。有关符合 FIPS 标准的 Amazon ECR 终端节点列表的更多信息，请参阅联邦信息处理标准 (FIPS) 140-3。

有关处理来 IPv4自 CLI 的 Amazon ECR API 调用的双栈客户端、Docker 和 OC Amazon I 客户端支持的服务终端节点的更多信息， Amazon SDKs 请参阅服务终端节点。

开始提出请求 IPv6

要向 Amazon ECR 注册表发出请求 IPv6，您需要使用双堆栈终端节点。在访问 Amazon ECR 注册表之前 IPv6，请验证以下要求：

- 您的客户端和网络必须支持 IPv6。
- Amazon ECR 支持以下请求类型： IPv6
  - OCI 和 Docker 客户端请求：

```
<registry-id>.dkr-ecr.<aws-region>.on.aws
```

- Amazon API 请求：

  ```
  ecr.<aws-region>.api.aws
  ```

- 您必须更新使用源 IP 地址筛选的任何 Amazon Identity and Access Management (IAM) 或注册表策略，以包括 IPv6 地址范围。有关更多信息，请参阅 在 IAM 策略中使用 IPv6 地址。
- 使用时 IPv6，服务器访问日志会以 IPv6 格式显示`Remote IP`地址。更新现有工具、脚本和软件以解析这些 IPv6格式化的 IP 地址。

> Note
>
> 如果您遇到与日志文件中存在 IPv6 地址有关的问题，请与联系Amazon Web Services 支持。

测试 IP 地址兼容性

如果您使用的是使用 Linux/Unix 或 Mac OS X，则可以使用以下示例所示的`curl`命令 IPv6 来测试是否可以访问双栈端点：

Example

```
curl --verbose https://ecr.us-west-2.api.aws
```

您获得的信息如下例所示。如果您通过连接的 IP 地址 IPv6 进行连接，则该地址将是一个 IPv6 地址。

```
* About to connect() to ecr.us-west-2.api.aws port 443 (#0)
* Trying IPv6 address... connected
* Connected to ecr.us-west-2.api.aws (IPv6 address) port 443 (#0)
> Host: ecr.us-west-2.api.aws
* Request completely sent off
```

如果你使用的是 Microsoft Windows 7 或 Windows 10，则可以测试是否 IPv6 可以通过 IPv4 或使用`ping`命令访问双栈端点，如下例所示。

```
ping ecr.us-west-2.api.aws
```

使用双栈 IPv6 终端节点发出请求

您可以使用双堆栈终端节点调用 Amazon ECR API。 IPv6 无论您使用 IPv4 还是，Amazon ECR API 操作的功能和性能都将保持一致。 IPv6

使用 Amazon Command Line Interface (Amazon CLI) 和时 Amazon SDKs，您可以通过使用参数或标志来启用 IPv6 以切换到双堆栈终端节点，或者直接在配置文件中指定双堆栈终端节点以覆盖默认 Amazon ECR 终端节点。您也可以使用命令更改配置，该命令在默认配置文件中设置`use_dualstack_endpoint`为 true。有关更多信息`use_dualstack_endpoint`，请参阅双栈和 FIPS 终端节点。

Example 使用命令更改配置

```
aws configure set default.ecr.use_dualstack_endpoint true
```

Example 提出请求而不是 IPv6 使用 Amazon CLI

```
aws ecr describe-repositories --region us-west-2 --endpoint-url https://
ecr.us-west-2.api.aws
```

使用 docker CLI 中的亚马逊 ECR 终端节点

登录您的 Amazon ECR 存储库并标记映像后，您可以向亚马逊 ECR 注册表推送和拉取 OCI 映像和 Docker 镜像。以下示例演示了使用两个双堆栈端点的 docker push 和 docker pull 命令。

Example 使用端点推送 docker 镜像 IPv4

```
docker push <registry-id>.dkr.ecr.us-west-1.amazonaws.com/my-repository:tag
```

Example 使用双栈端点推送 docker 镜像

```
docker push <registry-id>.dkr-ecr.us-west-1.on.aws/my-repository:tag
```

Example 使用端点提取 docker 镜像 IPv4

```
docker pull <registry-id>.dkr.ecr.us-west-1.amazonaws.com/my-repository:tag
```

Example 使用双栈端点提取 docker 镜像

```
docker pull <registry-id>.dkr-ecr.us-west-1.on.aws/my-repository:tag
```

在 IAM 策略中使用 IPv6 地址

在使用访问注册表之前 IPv6，请确保使用 IP 地址筛选的 IAM 用户和 Amazon ECR 注册表策略包含 IPv6 地址范围。如果未更新 IP 地址筛选策略来处理 IPv6 地址，则客户端在开始使用注册表时可能会错误地丢失或无法访问注册表 IPv6。有关使用 IAM 管理访问权限的更多信息，请参阅 适用于 Amazon Elastic Container Registry 的 Identity and Access Management。

筛选 IP 地址的 IAM 策略使用 IP 地址条件运算符。以下注册表策略示例说明如何使用 IP IPv4 地址条件运算符识别允许的地址`54.240.143.*`范围。任何超出此范围的 IP 地址都将被拒绝访问注册表 (`exampleregistry`)。由于所有 IPv6地址都超出了允许的范围，因此此策略会阻止 IPv6 地址访问`exampleregistry`。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "ecr:*",
      "Resource": "arn:aws:ecr:::exampleregistry/*",
      "Condition": {
         "IpAddress": {"aws:SourceIp": "54.240.143.0/24"}
      }
    }
  ]
}
```

要同时允许 IPv4 (`54.240.143.0/24`) 和 IPv6 (`2001:DB8:1234:5678::/64`) 地址范围，请修改注册表策略的 Condition 元素，如以下示例所示。您可以使用这种`Condition`区块格式来更新您的 IAM 用户和注册表策略。

```
        "Condition": {
          "IpAddress": {
             "aws:SourceIp": [
               "54.240.143.0/24",
                "2001:DB8:1234:5678::/64"
             ]
          }
        }
```

⚠ Important

在使用之前， IPv6 您必须更新所有使用 IP 地址筛选的相关 IAM 用户和注册表策略。我们不建议在注册表策略中使用 IP 地址筛选。

您可以使用 IAM 控制台查看您的 IAM 用户政策，网址为https://console.aws.amazon.com/iam/。有关 IAM 的更多信息，请参阅《IAM 用户指南》https://docs.amazonaws.cn/IAM/latest/UserGuide/。

Amazon ECR 私有注册表

Amazon ECR 私有注册表通过高度可用且可扩展的架构托管容器镜像。您可以使用私有注册表管理由 Docker 以及 Open Container Initiative(OCI) 镜像和构件组成的私有镜像存储库。每个 Amazon 账户都提供有原定设置的私有 Amazon ECR 注册表。有关 Amazon ECR 公有注册表的更多信息，请参阅 Amazon Elastic Container Registry Public 用户指南中的公有注册表。

私有注册表概念

- 您的原定设置私有注册表的 URL 为 `https://aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn`。
- 预设情况下，您的账户可以读取和写入私有注册表中的存储库。但是，用户需要权限才能调用 Amazon ECR， APIs 以及向您的私有存储库推送或拉取映像。Amazon ECR 提供了多个 托管策略来控制不同级别的用户访问。有关更多信息，请参阅 Amazon Elastic Container Registry 基于身份的策略示例。
- 您必须为 Docker 客户端授予注册表权限，以便使用 docker push 和 docker pull 命令对该私有注册表中的存储库执行推送和提取镜像操作。有关更多信息，请参阅 Amazon ECR 中的私有注册表身份验证。
- 可通过 用户访问策略及存储库策略对私有存储库加以控制。有关存储库策略的更多信息，请参阅Amazon ECR 中的私有存储库策略。
- 通过为私有注册表配置复制，可以在您自己的私有注册表中的 Amazon 区域之间复制您的私有注册表中的存储库，也可以跨不同的账户进行复制。有关更多信息，请参阅 Amazon ECR 中的私有映像复制。

Amazon ECR 中的私有注册表身份验证

您可以使用 Amazon Web Services Management Console Amazon CLI、或 Amazon SDKs 来创建和管理私有仓库。也可以使用这些方法对镜像执行某些操作，例如列出或删除镜像。这些客户端使用标准的 Amazon 身份验证方法。尽管在技术上可以使用 Amazon ECR API 推送和提取镜像，但您更有可能使用 Docker CLI 或特定语言的 Dockerf 库。

Docker CLI 不支持本机 IAM 身份验证方法。必须执行其他步骤， 以便 Amazon ECR 可以对 Docker 推送和提取请求进行身份验证和授权。

我们提供以下各节详细介绍的注册表身份验证方法。

使用 Amazon ECR 凭证辅助程序

Amazon ECR 提供了 Docker 凭证辅助程序，这使得在 Amazon ECR 中推送和提取镜像时更容易存储和使用 Docker 凭证。有关安装和配置步骤，请参阅 Amazon ECR Docker 凭证辅助程序。

> Note
>
> 目前，Amazon ECR Docker 凭证助手不支持多重身份验证 (MFA)。

使用授权令牌

授权令牌的权限范围与用于检索身份验证令牌的 IAM 委托人的权限范围相匹配。身份验证令牌用于访问您的 IAM 委托人有权访问且有效期为 12 小时的任何 Amazon ECR 注册表。要获取授权令牌，必须使用 GetAuthorizationTokenAPI 操作检索包含用户名AWS和编码密码的 base64 编码的授权令牌。该 Amazon CLI `get-login-password`命令通过检索和解码授权令牌来简化此操作，然后您可以将其传送到docker login命令中进行身份验证。

使用 get-login 针对 Amazon ECR 私有注册表验证 Docker

- 要使用向 Amazon ECR 注册表对 Docker 进行身份验证 get-login-password，请运行命令。aws ecr get-login-password将身份验证令牌传递给 docker login 命令时，将值 AWS 用作用户名，并指定要对其进行身份验证的 Amazon ECR 注册表 URI。如果对多个注册表进行身份验证，则必须针对每个注册表重复该命令。

  > Important
  >
  > 如果收到错误，请安装或更新到最新版本的 Amazon CLI。有关更多信息，请参阅Amazon Command Line Interface 《用户指南》中的安装 Amazon Command Line Interface。

  - get-login-password (Amazon CLI)

    ```
    aws ecr get-login-password --region region | docker login --username AWS --
    password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
    ```

  - Get-ECRLogin 命令 (Amazon Tools for Windows PowerShell)

```
(Get-ECRLoginCommand).Password | docker login --username AWS --password-
stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

使用 HTTP API 身份验证

Amazon ECR 支持 Docker 注册表 HTTP API。但是，由于 Amazon ECR 属于私有注册表，因此您必须为每个 HTTP 请求提供授权令牌。您可以使用-H选项添加 HTTP 授权标头，curl然后传递get-authorization-token Amazon CLI 命令提供的授权令牌。

使用 Amazon ECR HTTP API 进行身份验证

1. 使用检索授权令牌 Amazon CLI 并将其设置为环境变量。

   ```
   TOKEN=$(aws ecr get-authorization-token --output text --query
    'authorizationData[].authorizationToken')
   ```

2. 要向 API 进行身份验证，可将 `$TOKEN` 变量传递到 curl 命令的 `-H` 选项。例如，以下命令会列出 Amazon ECR 存储库中的镜像标签。有关更多信息，请参阅 Docker 注册表 HTTP API 参考文档。

   ```
   curl -i -H "Authorization: Basic $TOKEN"
    https://aws_account_id.dkr.ecr.region.amazonaws.com/v2/amazonlinux/tags/list
   ```

   您可以在一个 (扩展) 代码行中执行所有这些操作：

   ```
   HTTP/1.1 200 OK
   Content-Type: text/plain; charset=utf-8
   Date: Thu, 04 Jan 2018 16:06:59 GMT
   Docker-Distribution-Api-Version: registry/2.0
   Content-Length: 50
   Connection: keep-alive

   {"name":"amazonlinux","tags":["2017.09","latest"]}
   ```

Amazon ECR 中的私有注册表设置

Amazon ECR 使用私有注册表设置在注册表级别配置功能。为每个区域分别配置私有注册表设置。您可以使用私有注册表设置来配置以下功能。

- 注册表权限-注册表权限策略提供对复制和提取缓存权限的控制。有关更多信息，请参阅 Amazon ECR 中的私有注册表权限。
- 提取缓存规则-拉取缓存规则用于将来自上游注册表的图像缓存在您的 Amazon ECR 私有注册表中。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步。
- 复制配置-复制配置用于控制您的存储库是跨 Amazon 区域还是跨账户复制。有关更多信息，请参阅 Amazon ECR 中的私有映像复制。
- 存储库创建模板-存储库创建模板用于定义 Amazon ECR 代表您创建新存储库时要适用的标准设置。例如，通过缓存提取操作创建的存储库。有关更多信息，请参阅 用于控制在缓存提取或复制操作期间创建的存储库的模板。
- 扫描配置-默认情况下，您的注册表已启用基本扫描。您可以启用增强扫描功能，该功能提供一种自动持续的扫描模式，可扫描操作系统和编程语言包漏洞。有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在软件漏洞。

Amazon ECR 中的私有注册表权限

Amazon ECR 使用注册表策略在私有注册表级别向 Amazon 主体授予权限。

范围是通过选择注册表策略版本来设置的。有两个版本具有不同的注册表策略范围：版本 1 (V1) 和版本 2 (V2)。V2 是扩展的注册表策略范围，包括所有 ECR 权限。有关 API 操作的完整列表，请参阅 Amazon ECR API 指南。V2 版本是默认的注册表策略范围。有关查看或设置注册表策略范围的更多信息，请参阅切换到扩展注册表策略范围。有关 Amazon ECR 私有注册表常规设置的信息，请参阅Amazon ECR 中的私有注册表设置。

版本详情如下。

- V1 — 对于版本 1，Amazon ECR 仅在私有注册表级别强制执行以下权限。
  - `ecr:ReplicateImage` – 向其他账户（称为源注册表）授予将其镜像复制到您的注册表的权限。这仅用于跨账户复制。
  - `ecr:BatchImportUpstreamImage` – 授权检索外部镜像并将其导入到您的私有注册表。
  - `ecr:CreateRepository` – 授予在私有注册表中创建存储库的权限。如果该私有注册表中尚未存在用于存储复制镜像或缓存镜像的存储库，则需要此权限。

- V2 — 对于版本 2，Amazon ECR 允许策略中的所有 ECR 操作，并在所有 ECR 请求中强制执行注册策略。

您可以使用控制台或 CLI 来查看或更改您的注册表策略范围。

> Note
>
> 虽然可以将`ecr:*`操作添加到私有注册表策略中，但最佳做法是仅根据您正在使用的功能添加所需的特定操作，而不是使用通配符。

主题

- Amazon ECR 的私有注册表策略示例
- 切换到扩展注册表策略范围
- 授予在 Amazon ECR 中进行跨账户复制的注册表权限
- 授予在 Amazon ECR 中提取缓存的注册表权限

Amazon ECR 的私有注册表策略示例

以下示例显示了您可用于控制用户对 Amazon ECR 注册表所具备权限的注册表权限策略声明。

> Note
>
> 在每个示例中，如果该`ecr:CreateRepository`操作已从注册表策略中删除，则仍可以进行复制。但是，要成功复制，您需要在账户中创建具有相同名称的存储库。

示例：允许源账户的根用户复制所有存储库

以下注册表权限策略允许源账户的根用户复制所有存储库。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
```

```
                "AWS":"arn:aws:iam::source_account_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/*"
            ]
        }
    ]
}
```

示例：允许来自多个账户的根用户

以下注册表权限策略包含两个语句。每个语句都允许源账户的根用户复制所有存储库。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount1",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_1_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/*"
            ]
        },
        {
            "Sid":"ReplicationAccessCrossAccount2",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_2_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
```

```
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/*"
            ]
        }
    ]
}
```

示例：允许源账户的根用户复制带有前缀 **prod-** 的所有存储库。

以下注册表权限策略允许源账户的根用户复制以 `prod-` 开头的所有存储库。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/prod-*"
            ]
        }
    ]
}
```

切换到扩展注册表策略范围

⚠ Important

对于新用户，您的注册表将在创建时自动配置为使用V2注册表策略。您无需采取任何操作。Amazon ECR 不建议恢复到之前的注册表政策。V1

您可以使用控制台或 CLI 来查看或更改您的注册表策略范围。

Amazon Web Services Management Console

使用以下步骤查看您的账户设置。要查看或更新注册表策略范围，请参阅本页上的 CLI 程序。

为您的私有注册表启用增强版注册表策略

1. 在私有注册表/存储库中打开 Amazon ECR 控制台 https://console.aws.amazon.com/ecr/
2. 从导航栏中选择 “区域”。
3. 在导航窗格中，选择 “私有注册表”、“功能和设置”，然后选择 “权限”。
4. 在权限页面上，对于注册表策略，请查看您的策略 JSON。如果您有 V1 策略，则会显示一条横幅，其中包含更新到 V2 的说明。请选择启用。

   将显示一个横幅，表示注册表策略范围已更新为 V2。
5. 您也可以选择使用 CLI 配置权限。有关更多信息，请参阅 Amazon ECR 中的私有注册表设置。

   > Note
   >
   > 要查看或更新注册表策略范围，请参阅本页上的 CLI 程序。

Amazon CLI

Amazon ECR 生成 V2 注册表策略。使用以下步骤查看或更新注册表策略范围。您无法在控制台中查看或更改注册表策略范围

- 检索您当前正在使用的注册表策略。

  ```
  aws ecr get-account-setting --name REGISTRY_POLICY_SCOPE
  ```

  参数名称为必填字段。如果您不提供名称，则将收到以下错误：

  ```
   aws: error: the following arguments are required: --name
  ```

  查看您的注册表策略命令的输出。在以下示例输出中，注册表策略版本为 V1。

  ```
  {
      "name": "REGISTRY_POLICY_SCOPE",
      "value": "V1"
  ```

```
}
```

您可以将注册表策略版本从更改V1为V2。V1 不是推荐的注册表策略范围。

```
aws ecr put-account-setting --name REGISTRY_POLICY_SCOPE --value value
```

例如，使用以下命令更新到 V2。

```
aws ecr put-account-setting --name REGISTRY_POLICY_SCOPE --value V2
```

查看您的注册表策略命令的输出。在以下示例输出中，注册表策略版本已更新为 V2。

```
{
    "name": "REGISTRY_POLICY_SCOPE",
    "value": "V2"
}
```

授予在 Amazon ECR 中进行跨账户复制的注册表权限

跨账户策略类型用于向 Amazon 委托人授予权限，允许将存储库从源注册表复制到您的注册表。预设情况下，您有权在自己的注册表中配置跨区域复制。如果您授予其他账户将内容复制到注册表的权限，则只需配置注册表策略。

注册表策略必须授予 `ecr:ReplicateImage` API 操作权限。此 API 是一个内部 Amazon ECR API，可在区域或账户之间复制镜像。您还可以授予 `ecr:CreateRepository` 权限，该权限允许 Amazon ECR 在您的注册表中创建存储库 (如果存储库尚不存在)。如果未提供 `ecr:CreateRepository` 权限，则必须在注册表中手动创建与源存储库名称相同的存储库。如果两者均未完成，复制将失败。任何失败的操作`CreateRepository`或 `ReplicateImage` API 操作都会显示在中 CloudTrail。

要配置复制的权限策略 (Amazon Web Services Management Console)

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择区域以配置注册表策略。
3. 在导航窗格中，选择 “私有注册表”，选择 “功能和设置”，然后选择 “权限”。
4. 在 Registry permissions（注册表权限）页面上，选择 Generate statement（生成语句）。
5. 使用策略生成器完成以下步骤以定义策略声明。

   a. 对于策略类型，选择复制-跨账户。

   b. 在对账单编号中，输入唯一的对账单编号。此字段在注册表策略上用作 `Sid`。

   c. 对于帐户，输入您要向 IDs其授予权限的每个账户的帐户。指定多个帐户时 IDs，请用逗号分隔它们。

6. 选择保存。

要配置复制的权限策略 (Amazon CLI)

1. 创建名为 `registry_policy.json` 的文件并使用注册表策略填充它。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/*"
            ]
        }
    ]
}
```

2. 使用策略文件创建注册表策略。

```
aws ecr put-registry-policy \
      --policy-text file://registry_policy.json \
      --region us-west-2
```

3. 检索要确认的注册表策略。

```
aws ecr get-registry-policy \
```

```
        --region us-west-2
```

授予在 Amazon ECR 中提取缓存的注册表权限

Amazon ECR 私有注册表权限可用于限定各个 IAM 实体使用推送缓存的权限范围。如果 IAM policy 授予 IAM 实体的权限多于注册表权限策略授予的权限，则 IAM policy 优先。

要创建私有注册表的权限策略 (Amazon Web Services Management Console)

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择您在其中配置私有注册表权限语句的区域。
3. 在导航窗格中，选择 “私有注册表”，选择 “功能和设置”，然后选择 “权限”。
4. 在 Registry permissions（注册表权限）页面上，选择 Generate statement（生成语句）。
5. 对于要创建的每个缓存提取权限策略语句，请执行以下操作。
    a. 对于 Policy type（策略类型），请选择 Pull through cache policy（推送缓存策略）。
    b. 对于 Statement id（语句 ID），为推送缓存语句策略提供名称。
    c. 对于 IAM entities（IAM 实体），指定要包含在策略中的用户、组或角色。
    d. 对于缓存命名空间，选择要与策略关联的直通缓存规则。
    e. 对于 Repository names（存储库名称），指定要应用规则的存储库基本名称。例如，如果您想在 Amazon ECR Public 上指定 Amazon Linux 存储库，存储库名称将为 `amazonlinux`。

Amazon ECR 私有存储库

Amazon ECR 私有存储库包含您的 Docker 映像、Open Container Initiative（OCI）映像和 OCI 兼容构件。通过使用 Amazon ECR API 操作或 Amazon ECR 控制台的存储库部分，您可以创建、监控和删除映像存储库，并设置权限以管理谁可以访问存储库。Amazon ECR 还与 Docker CLI 集成，因此您可以将映像从开发环境推送到存储库，并提取。

主题

私有存储库概念

- 默认情况下，您的账户可以读取和写入默认注册表中的存储库 (*`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com`)。但是，用户需要权限才能调用 Amazon ECR， APIs 以及向您的存储库推送或拉取映像。Amazon ECR 提供了多个 托管策略来控制不同级别的用户访问。有关更多信息，请参阅 Amazon Elastic Container Registry 基于身份的策略示例。
- 通过用户访问策略及个别存储库策略均可控制存储库。有关更多信息，请参阅 Amazon ECR 中的私有存储库策略。
- 存储库名称可支持命名空间，您可以使用命名空间分组相似的存储库。例如，如果多个团队使用相同的注册表，团队 A 可以使用 `team-a` 命名空间，团队 B 可以使用 `team-b` 命名空间。这样，每个团队都有自己的名为 `web-app` 的镜像，每个镜像都以团队命名空间开头。这种配置可在不产生干扰的情况下同时使用每个团队上的这些镜像。团队 A 的镜像是 `team-a/web-app`，团队 B 的镜像是 `team-b/web-app`。
- 您的镜像可以在自己的注册表中跨区域和跨账户复制到其他存储库。您可以通过在注册表设置中指定复制配置来执行此操作。有关更多信息，请参阅 Amazon ECR 中的私有注册表设置。

创建 Amazon ECR 私有存储库以存储映像

> ⚠ Important
>
> 双层服务器端加密，使用（Amazon KMS DSSE-KMS) 仅在 Amazon GovCloud (US) 地区可用。

创建 Amazon ECR 私有存储库，然后使用存储库来存储您的容器映像。请按照以下步骤以使用 Amazon Web Services Management Console创建私有存储库。有关使用创建存储库的步骤 Amazon CLI，请参阅步骤 2：创建存储库。

创建存储库 (Amazon Web Services Management Console)

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。
2. 从导航栏中，选择您创建存储库的区域。
3. 在存储库页面，选择私有存储库，然后选择创建存储库。
4. 对于存储库名称，输入存储库的唯一名称。存储库名称可以自行指定 (例如 `nginx-web-app`)。或者，可以在其前面加上命名空间来将存储库分组到类别中 (例如 `project-a/nginx-web-app`)。

   > ⓘ Note
   >
   > 存储库名称最多可以包含 256 个字符。名称必须以字母开头，并且只能包含小写字母、数字、连字符、下划线和正斜杠。不支持使用双连字符、双下划线或双正斜杠。

5. 对于标签不变性，选择存储库的标签可变性设置。配置有不可变标签的存储库会阻止覆盖镜像标签。有关更多信息，请参阅 防止映像标签在 Amazon ECR 中被覆盖。
6. 要进行加密配置，请选择 AES-256 或 Amazon KMS。有关更多信息，请参阅 静态加密。
   a. 如果选中， Amazon KMS 则在单层加密和双层加密之间进行选择。使用 Amazon KMS 或双层加密需要支付额外费用。有关更多信息，请参阅 Amazon ECR 服务定价。
   b. 默认情况下，选择带有别`aws/ecr`名的 Amazon 托管密钥。此密钥是在您首次创建启用 Amazon KMS 加密的存储库时在您的账户中创建的。选择客户管理的密钥（高级）以选择您自己的 Amazon KMS 密钥。 Amazon KMS 密钥必须与集群位于同一个区域。选择 “创建 Amazon KMS 密钥”，导航到 Amazon KMS 控制台以创建自己的密钥。

7. 对于图像扫描设置，虽然您可以在存储库级别为基本扫描指定扫描设置，但最佳做法是在私有注册表级别指定扫描配置。通过在私有注册表级别配置扫描设置，您可以选择增强扫描或基本扫描，同时还可以定义用于指定应该扫描哪些存储库的筛选条件。
8. 选择创建。

后续步骤

要查看将映像推送到您的存储库的步骤，请选择存储库和查看推送命令。要详细了解如何将镜像推送到存储库，请参阅将映像推送至 Amazon ECR 私有存储库。

查看 Amazon ECR 中私有存储库的内容和详细信息

创建私有存储库后，您可以在 Amazon Web Services Management Console中查看与存储库相关的详细信息：

- 存储库中存储了哪些镜像
- 与存储库中存储的每个镜像相关的详细信息，包括每个镜像的大小和 SHA 摘要
- 为存储库内容指定的扫描频率
- 存储库是否具有与之关联的活动拉取缓存规则
- 存储库的加密设置

> **Note**
>
> 从 Docker 版本 1.9 开始，在将镜像推送至 V2 版本的 Docker 注册表之前，Docker 客户端会压缩镜像的分层。docker images 命令的输出显示未压缩的镜像大小。因此，请记住，Docker 可能会返回比 Amazon Web Services Management Console中所示更大的镜像。

查看存储库信息 (Amazon Web Services Management Console)

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。
2. 从导航栏中，选择包含要查看的存储库的区域。
3. 在导航窗格中，选择存储库。
4. 在 Repositories（存储库）页面上，选择 Private（私有）选项卡，然后选择要查看的存储库。

5. 在存储库详细信息页面上，控制台默认为 Images（镜像）视图。使用导航菜单查看其他存储库相关信息。
    - 选择 Summary（摘要）查看存储库详细信息并拉取该存储库的计数数据。
    - 选择 Images（镜像）以查看与存储库中的镜像相关的信息。要查看有关镜像的更多信息，请选择镜像标签。有关更多信息，请参阅 在 Amazon ECR 中查看映像详细信息。

      如果您要删除的镜像没有标签，您可以选择要删除的存储库左侧的框，然后选择删除。有关更多信息，请参阅 删除 Amazon ECR 中的映像。
    - 选择权限以查看适用于存储库的存储库策略。有关更多信息，请参阅 Amazon ECR 中的私有存储库策略。
    - 选择生命周期策略以查看适用于存储库的生命周期策略规则。此处还可查看生命周期事件历史记录。有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像。
    - 选择标签以查看适用于存储库的元数据标签。

在 Amazon ECR 中删除私有存储库

如果存储库已使用完毕，您可以删除它。在中删除存储库时 Amazon Web Services Management Console，存储库中包含的所有图像也会被删除；此操作无法撤消。

> ⚠ Important
>
> 已删除存储库中的映像也会被删除。您无法撤消此操作。

删除存储库（Amazon Web Services Management Console）

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。
2. 从导航栏中，选择包含要删除的存储库的区域。
3. 在导航窗格中，选择存储库。
4. 在 Repositories（存储库）页面上，选择 Private（私有）选项卡，然后选择要删除的存储库，并选择 Delete（删除）。
5. 在 *repository_name* “删除” 窗口中，确认是否应删除选定的存储库，然后选择 “删除”。

Amazon ECR 中的私有存储库策略

Amazon ECR 使用基于资源的权限控制对存储库的访问。基于资源的权限让您可以指定能够访问存储库的用户或角色，以及这些用户或角色可以对该存储库执行的操作。默认情况下，只有创建存储库的 Amazon 账户才有权访问存储库。您可以应用存储库策略来允许针对存储库的其他权限。

主题

- 存储库策略与 IAM policy
- Amazon ECR 中的私有存储库策略示例
- 在 Amazon ECR 中设置私有存储库策略声明

存储库策略与 IAM policy

Amazon ECR 存储库策略是 IAM policy 的一部分，这些策略专用于控制对单个 Amazon ECR 存储库的访问。IAM policy 通常用于应用针对整个 Amazon ECR 服务的权限，但也可用于控制对特定资源的访问。

在确定某个特定用户或角色可对存储库执行的操作时，将同时使用 Amazon ECR 存储库策略和 IAM policy。如果通过存储库策略允许某个用户或角色执行某个操作但通过 IAM policy 拒绝其执行该操作 (或反过来)，则将拒绝该操作。用户或角色只需通过存储库策略或 IAM policy 之一获得执行某个操作的许可，而不需要同时通过这两个策略来获得执行该操作的许可。

⚠ Important

Amazon ECR 要求用户有权通过 IAM policy 调用 `ecr:GetAuthorizationToken` API，然后才能对注册表进行身份验证并对任意 Amazon ECR 存储库推送或提取任意镜像。Amazon ECR 提供了多个托管 IAM 策略，用于控制不同级别的用户访问权限。有关更多信息，请参阅 Amazon Elastic Container Registry 基于身份的策略示例。

您可以使用这两个策略类型之一来控制对您的存储库的访问，如以下示例中所示。

此示例显示了一个 Amazon ECR 存储库策略，该策略允许某个特定用户描述存储库及存储库内的镜像。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ECRRepositoryPolicy",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::account-id:user/username"},
            "Action": [
                "ecr:DescribeImages",
                "ecr:DescribeRepositories"
            ]
        }
    ]
}
```

此示例显示了一个 IAM policy，该策略可实现与上面相同的目标，方法是使用资源参数将策略范围限定为存储库 (由存储库的完整 ARN 指定)。有关 Amazon Resource Name (ARN) 格式的更多信息，请参阅 资源。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDescribeRepoImage",
            "Effect": "Allow",
            "Action": [
                "ecr:DescribeImages",
                "ecr:DescribeRepositories"
            ],
            "Resource": ["arn:aws:ecr:region:account-id:repository/repository-name"]
        }
    ]
}
```

Amazon ECR 中的私有存储库策略示例

> ⚠ Important
>
> 此页面上的存储库策略示例旨在应用于 Amazon ECR 私有存储库。如果直接与 IAM 主体结合使用，这些存储库策略将无法正常工作，除非进行修改，将 Amazon ECR 存储库指定为资源。有关如何设置存储库策略的更多信息，请参阅在 Amazon ECR 中设置私有存储库策略声明。

Amazon ECR 存储库策略是 IAM policy 的一部分，这些策略专用于控制对单个 Amazon ECR 存储库的访问。IAM policy 通常用于应用针对整个 Amazon ECR 服务的权限，但也可用于控制对特定资源的访问。有关更多信息，请参阅 存储库策略与 IAM policy。

以下存储库策略示例显示了可用于控制对 Amazon ECR 私有存储库的访问权限的权限声明。

> **Important**
>
> Amazon ECR 要求用户有权通过 IAM policy 调用 `ecr:GetAuthorizationToken` API，然后才能对注册表进行身份验证并从任意 Amazon ECR 存储库推送或提取任意镜像。Amazon ECR 提供了多个托管 IAM 策略，用于控制不同级别的用户访问权限。有关更多信息，请参阅 Amazon Elastic Container Registry 基于身份的策略示例。

示例：允许一个或多个 用户

以下存储库策略允许一个或多个 用户向存储库推送和从存储库提取映像。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPushPull",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id:user/push-pull-user-1",
                    "arn:aws:iam::account-id:user/push-pull-user-2"
                ]
            },
            "Action": [
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "ecr:CompleteLayerUpload",
                "ecr:GetDownloadUrlForLayer",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ]
        }
    ]
```

```
}
```

示例：允许其他账户

以下存储库策略允许特定账户推送镜像。

> **Important**
>
> 您授予权限的账户必须启用所创建存储库策略的区域，否则将发生错误。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCrossAccountPush",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:root"
            },
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:CompleteLayerUpload",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ]
        }
    ]
}
```

以下存储库策略允许某些用户拉取图像（*pull-user-1*和*pull-user-2*），同时提供对另一个存储库的完全访问权限（*admin-user*）。

> **Note**
>
> 对于目前不支持的更复杂的存储库策略 Amazon Web Services Management Console，您可以使用set-repository-policy Amazon CLI 命令应用该策略。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPull",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id:user/pull-user-1",
                    "arn:aws:iam::account-id:user/pull-user-2"
                ]
            },
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ]
        },
        {
            "Sid": "AllowAll",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:user/admin-user"
            },
            "Action": [
                "ecr:*"
            ]
        }
    ]
}
```

示例：拒绝所有

以下存储库策略拒绝所有账户中的所有用户提取镜像。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyPull",
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
```

```
                ]
            }
        ]
}
```

示例：限制对特定 IP 地址的访问权限

以下示例拒绝向任何用户授予在应用于来自特定地址范围的存储库时执行任何 Amazon ECR 操作的权限。

此语句中的条件确定了允许的 Internet 协议版本 4 (IPv4) IP 地址的`54.240.143.*`范围。

该`Condition`块使用`NotIpAddress`条件和`aws:SourceIp`条件键，后者是一个 Amazon宽范围的条件键。有关这些条件键的更多信息，请参阅 Amazon 全局条件上下文键。这些`aws:sourceIp` IPv4 值使用标准的 CIDR 表示法。有关更多信息，请参阅 《IAM 用户指南》中的 IP 地址条件运算符。

```
{
    "Version": "2012-10-17",
    "Id": "ECRPolicyId1",
    "Statement": [
        {
            "Sid": "IPAllow",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "ecr:*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "54.240.143.0/24"
                }
            }
        }
    ]
}
```

示例：允许某项 Amazon 服务

以下存储库策略允许 Amazon CodeBuild 访问与该服务集成所需的 Amazon ECR API 操作。使用以下示例时，您应该使用 `aws:SourceArn` 和 `aws:SourceAccount` 条件键来限定哪些资源可以使用这些权限。有关更多信息，请参阅《Amazon CodeBuild 用户指南》 CodeBuild中的 Amazon ECR 示例。

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CodeBuildAccess",
         "Effect":"Allow",
         "Principal":{
            "Service":"codebuild.amazonaws.com"
         },
         "Action":[
            "ecr:BatchGetImage",
            "ecr:GetDownloadUrlForLayer"
         ],
         "Condition":{
            "ArnLike":{
               "aws:SourceArn":"arn:aws:codebuild:region:123456789012:project/project-name"
            },
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            }
         }
      }
   ]
}
```

在 Amazon ECR 中设置私有存储库策略声明

您可以按照以下步骤向中的存储库添加访问策略声明。 Amazon Web Services Management Console 您可以为每个存储库添加多个策略声明。有关示例策略，请参阅 Amazon ECR 中的私有存储库策略示例。

> ⚠ Important
>
> Amazon ECR 要求用户有权通过 IAM policy 调用 `ecr:GetAuthorizationToken` API，然后才能对注册表进行身份验证并从任意 Amazon ECR 存储库推送或提取任意镜像。Amazon ECR 提供了多个托管 IAM 策略，用于控制不同级别的用户访问权限。有关更多信息，请参阅 Amazon Elastic Container Registry 基于身份的策略示例。

设置存储库策略声明

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。

2. 从导航栏中，选择包含要对其设置策略声明的存储库的区域。
3. 在导航窗格中，选择存储库。
4. 在存储库页面上，选择要对其设置策略声明的存储库，以查看存储库的内容。
5. 从存储库镜像列表视图的导航窗格中，选择权限、编辑。

    > **Note**
    >
    > 如果您未看到权限选项，请确保您处于存储库镜像列表视图中。

6. 在编辑权限页面上，选择添加声明。
7. 对于声明名称，输入声明的名称。
8. 对于效果，选择策略语句产生的结果是允许还是明确拒绝。
9. 对于委托人，选择策略声明应用到的范围。有关更多信息，请参阅 IAM 用户指南中的 Amazon JSON 策略元素：委托人。

    - 通过选中 Everyone (*) 复选框，可以将该语句应用于所有经过身份验证的 Amazon 用户。
    - 对于服务委托人，指定服务委托人名称 (例如 `ecs.amazonaws.com`) 以将声明应用到特定的服务。
    - 在 “Amazon 帐户” 中 IDs，指定一个 Amazon 账号（例如111122223333），以将该账单应用于特定 Amazon 账户下的所有用户。可以使用逗号分隔的列表指定多个账户。

        > **Important**
        >
        > 您授予权限的账户必须启用所创建存储库策略的区域，否则将发生错误。

    - 对于 IAM 实体，请选择您的 Amazon 账户下要应用声明的角色或用户。

        > **Note**
        >
        > 对于目前不支持的更复杂的存储库策略 Amazon Web Services Management Console，您可以使用set-repository-policy Amazon CLI 命令应用该策略。

10. 对于操作，从各个 API 操作的列表中选择应当应用策略声明的 Amazon ECR API 操作的范围。
11. 完成后，选择保存设置策略。
12. 对要添加的每个存储库策略重复以上步骤。

在 Amazon ECR 中标记私有存储库

为了帮助您管理 Amazon ECR 存储库，您可以使用 Amazon 资源标签将自己的元数据分配给新的或现有的 Amazon ECR 存储库。例如，您可以为账户的 Amazon ECR 存储库定义一组标签以帮助您跟踪每个存储库的拥有者。

标签基本知识

标签对 Amazon ECR 没有任何语义意义，应严格按字符串进行解析。标签不会自动分配至资源。您可以修改标签的密钥和值，还可以随时删除资源的标签。您可以将标签的值设为空的字符串，但是不能将其设为空值。如果您添加的标签的值与该实例上现有标签的值相同，新的值就会覆盖旧值。如果删除资源，资源的所有标签也会被删除。

您可以使用 Amazon ECR 控制台 Amazon CLI、和 Amazon ECR API 来处理标签。

使用 Amazon Identity and Access Management (IAM)，您可以控制 Amazon 账户中哪些用户有权创建、编辑或删除标签。有关 IAM 策略中的标签的信息，请参阅the section called “使用基于标签的访问控制”。

标记资源以便于计费

您为 Amazon ECR 存储库添加的标签在成本和使用率报告中启用标签后查看成本分配时非常有帮助。有关更多信息，请参阅 Amazon ECR 用量报告。

如需查看组合资源的成本，请按具有相同标签键值的资源组织您的账单信息。例如，您可以将特定的应用程序名称用作几个资源的标签，然后组织账单信息，以查看在数个服务中的使用该应用程序的总成本。有关设置带有标签的成本分配报告的更多信息，请参阅 Amazon Billing 用户指南中的月度成本分配报告。

> **Note**
>
> 如果您已启用报告，则可以在 24 小时后查看当月的数据。

在 Amazon ECR 中为私有存储库添加标签

您可以将标签添加到私有存储库中。

有关标签名称和最佳做法的信息，请参阅《标签 Amazon 资源用户指南》中的标签命名限制和要求以及最佳实践。

向存储库添加标签（Amazon Web Services Management Console）

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择要使用的区域。
3. 在导航窗格中，选择存储库。
4. 在存储库页面上，选中要标记的存储库旁边的复选框。
5. 从操作菜单中，选择存储库标签。
6. 在存储库标签页面上，选择添加标签、添加标签。
7. 在编辑存储库标签页面上，为每个标签指定键和值，然后选择保存。

向存储库（Amazon CLI 或 API）添加标签

您可以使用或 API 添加或覆盖一个 Amazon CLI 或多个标签。

- Amazon CLI -标签资源
- API 操作-TagResource

以下示例演示如何使用 Amazon CLI来添加标签。

示例 1：标记存储库

以下命令标记存储库。

```
aws ecr tag-resource \
     --resource-arn arn:aws:ecr:region:account_id:repository/repository_name \
     --tags Key=stack,Value=dev
```

示例 2：使用多个标签标记存储库

以下命令将三个标签添加到存储库。

```
aws ecr tag-resource \
     --resource-arn arn:aws:ecr:region:account_id:repository/repository_name \
     --tags Key=key1,Value=value1 Key=key2,Value=value2 Key=key3,Value=value3
```

示例 3：列出存储库的标签

以下命令列出与存储库关联的标签。

```
aws ecr list-tags-for-resource \
     --resource-arn arn:aws:ecr:region:account_id:repository/repository_name
```

示例 4：创建存储库并添加标签

以下命令创建一个名为 `test-repo` 的存储库并添加键为 `team`、值为 `devs` 的标签。

```
aws ecr create-repository \
     --repository-name test-repo \
     --tags Key=team,Value=devs
```

在 Amazon ECR 中从私有存储库删除标签

您可以从私有存储库中删除标签。

要从私有存储库中删除标签（Amazon Web Services Management Console）

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择要使用的区域。
3. 在存储库页面上，选中要从中删除标签的存储库旁边的复选框。
4. 从操作菜单中，选择存储库标签。
5. 在存储库标签页面上，选择编辑。
6. 在编辑存储库标签页面上，选择要删除的每个标签对应的删除，然后选择保存。

要从私有存储库中删除标签（Amazon CLI）

您可以使用或 API 删除一个 Amazon CLI 或多个标签。

- Amazon CLI -取消标记资源
- API 操作-UntagResource

以下示例演示了如何使用 Amazon CLI从存储库中删除标签。

```
aws ecr untag-resource \
     --resource-arn arn:aws:ecr:region:account_id:repository/repository_name \
     --tag-keys tag_key
```

Amazon ECR 中的私有映像

Amazon ECR 在私有存储库中存储 Docker 映像、Open Container Initiative（OCI）映像和 OCI 兼容构件。您可以使用 Docker CLI 或首选客户端从存储库推送和提取镜像。

借助 Amazon ECR 对 OCI v1.1 的支持，您可以存储和管理由 OCI 引用站点 API 定义的参考构件。工件包括签名、软件物料清单 (SBoMs)、Helm 图表、扫描结果和证明。容器映像的构件组随该容器一起传输，并作为单独的映像进行存储，该映像算作存储库消耗的映像。

对存储在 Amazon ECR 私有存储库中的映像进行签名 和 从 Amazon ECR 私有存储库中删除签名和其他构件 页面提供了如何使用与签名相关的构件的示例。有关对容器映像签名的更多信息，请参阅 Amazon Signer Developer Guide 中的 Signing container images。

主题

将映像推送至 Amazon ECR 私有存储库

您可以将 Docker 镜像、清单列表和 Open Container Initiative (OCI) 镜像以及兼容的构件推送到您的私有存储库。

Amazon ECR 还提供了一种可将您的映像复制到其他存储库的方法。通过在您的私有注册表设置中指定复制配置，您可以在自己注册表中跨区域和跨不同账户进行复制。有关更多信息，请参阅 Amazon ECR 中的私有注册表设置。

主题

- 将映像推送至 Amazon ECR 私有存储库的 IAM 权限
- 将 Docker 映像推送到 Amazon ECR 私有存储库
- 将多架构映像推送到 Amazon ECR 私有存储库
- 将 Helm 图表推送到 Amazon ECR 私有存储库

将映像推送至 Amazon ECR 私有存储库的 IAM 权限

用户需要将映像推送至 Amazon ECR 私有存储库的 IAM 权限。按照授予最低权限的最佳实践，您可以授予对特定存储库的访问权限。您还可以授予对所有存储库的访问权限。

用户必须通过请求授权令牌，向其推送镜像的每个 Amazon ECR 注册表进行身份验证。Amazon ECR 提供了多种 Amazon 托管策略来控制不同级别的用户访问权限。有关更多信息，请参阅 Amazon Amazon 弹性容器注册表的托管策略。

您还可以创建自己的 IAM 策略。以下 IAM 策略授予将映像推送到特定存储库所需的权限。要限制特定存储库的权限，请使用该存储库的完整 Amazon 资源名称 (ARN)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:CompleteLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:InitiateLayerUpload",
                "ecr:BatchCheckLayerAvailability",
                "ecr:PutImage",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:region:111122223333:repository/repository-name"
        },
        {
            "Effect": "Allow",
            "Action": "ecr:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
```

```
}
```

以下 IAM 策略授予将映像推送到所有存储库所需的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:CompleteLayerUpload",
                "ecr:GetAuthorizationToken",
                "ecr:UploadLayerPart",
                "ecr:InitiateLayerUpload",
                "ecr:BatchCheckLayerAvailability",
                "ecr:PutImage"
            ],
             "Resource": "arn:aws:ecr:region:111122223333:repository/*"
        }
    ]
}
```

将 Docker 映像推送到 Amazon ECR 私有存储库

您可以使用 docker push 命令将容器镜像推送到 Amazon ECR 存储库。

Amazon ECR 还支持创建和推送用于多架构映像的 Docker 清单列表。有关信息，请参阅将多架构映像推送到 Amazon ECR 私有存储库。

推送 Docker 镜像到 Amazon ECR 存储库

在推送镜像之前，Amazon ECR 存储库必须存在。有关更多信息，请参阅 the section called “创建存储库以存储映像”。

1. 向要向其推送镜像的 Amazon ECR 注册表验证 Docker 客户端的身份。必须针对每个注册表获得授权令牌，令牌有效期为 12 小时。有关更多信息，请参阅 Amazon ECR 中的私有注册表身份验证。

   要对 Amazon ECR 注册表验证 Docker，请运行 aws ecr get-login-password 命令。将身份验证令牌传递给 docker login 命令时，将值 `AWS` 用作用户名，并指定要对其进行身份验证的 Amazon ECR 注册表 URI。如果对多个注册表进行身份验证，则必须针对每个注册表重复该命令。

> ⚠ Important
>
> 如果收到错误，请安装或更新到最新版本的 Amazon CLI。有关更多信息，请参阅Amazon Command Line Interface 《用户指南》中的安装 Amazon Command Line Interface。

```
aws ecr get-login-password --region region | docker login --username AWS --
password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

2. 如果在要推送的注册表中还没有您的镜像存储库，请创建它。有关更多信息，请参阅 创建 Amazon ECR 私有存储库以存储映像。
3. 识别要推送的本地镜像。运行 docker images 命令列出系统中的容器镜像。

   ```
   docker images
   ```

   您可以在生成的命令输出中使用该*repository:tag*值或图像 ID 来识别图像。
4. 通过要使用的 Amazon ECR 注册表、存储库和可选镜像标签名称组合标记您的镜像。注册表格式为 *aws_account_id*`.dkr.ecr.`*cn-north-1*`.amazonaws.com.cn`。存储库名称应与您为镜像创建的存储库一致。如果省略镜像标签，我们将假定标签为 `latest`。

   以下示例将 ID 为的本地图像标记*e9ae3c220b23*为*aws_account_id*`.dkr.ecr.`*cn-north-1*`.amazonaws.com.cn/my-repository:tag`。

   ```
   docker tag e9ae3c220b23 aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn/my-
   repository:tag
   ```
5. 使用 docker push 命令推送镜像：

   ```
   docker push aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn/my-repository:tag
   ```
6. (可选) 通过重复 Step 4 和 Step 5，向镜像应用任何其他标签并将这些标签推送到 Amazon ECR。

将多架构映像推送到 Amazon ECR 私有存储库

通过创建和推送 Docker 清单列表，您可以将多架构映像推送到 Amazon ECR 存储库。清单列表是通过指定一个或多个镜像名称创建的镜像列表。大多数情况下，清单列表是从提供相同功能但适用于不同操作系统或架构的映像创建的。清单列表不是必需项。有关更多信息，请参阅 Docker 清单。

清单列表可以像其他 Amazon ECR 镜像一样在 Amazon ECS 任务定义或 Amazon EKS Pod 规范中提取或引用。

先决条件

- 在您的 Docker CLI 中，开启实验性功能。有关实验性功能的信息，请参阅 Docker 文档中的 Experimental features。
- 在推送镜像之前，Amazon ECR 存储库必须存在。有关更多信息，请参阅 the section called “创建存储库以存储映像”。
- 在创建 Docker 清单之前，必须将映像推送到您的存储库。有关如何推送镜像的信息，请参阅 将 Docker 映像推送到 Amazon ECR 私有存储库。

将多架构 Docker 镜像推送到 Amazon ECR 存储库

1. 向要向其推送镜像的 Amazon ECR 注册表验证 Docker 客户端的身份。必须针对每个注册表获得授权令牌，令牌有效期为 12 小时。有关更多信息，请参阅 Amazon ECR 中的私有注册表身份验证。

   要对 Amazon ECR 注册表验证 Docker，请运行 aws ecr get-login-password 命令。将身份验证令牌传递给 docker login 命令时，将值 AWS 用作用户名，并指定要对其进行身份验证的 Amazon ECR 注册表 URI。如果对多个注册表进行身份验证，则必须针对每个注册表重复该命令。

   > Important
   >
   > 如果收到错误，请安装或更新到最新版本的 Amazon CLI。有关更多信息，请参阅Amazon Command Line Interface 《用户指南》中的安装 Amazon Command Line Interface。

   ```
   aws ecr get-login-password --region region | docker login --username AWS --
   password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
   ```

2. 列出存储库中的镜像，确认镜像标签。

   ```
   aws ecr describe-images --repository-name my-repository
   ```

3. 创建 Docker 清单列表。`manifest create` 命令验证引用的镜像是否已存在于您的存储库中，并在本地创建清单。

```
docker manifest create aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn/my-repository aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn/my-repository:image_one_tag aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn/my-repository:image_two
```

4. (可选) 检查 Docker 清单列表。这使您能够确认清单列表中引用的每个镜像清单的大小和摘要。

```
docker manifest inspect aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn/my-repository
```

5. 将 Docker 清单列表推送到您的 Amazon ECR 存储库。

```
docker manifest push aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn/my-repository
```

将 Helm 图表推送到 Amazon ECR 私有存储库

您可以将 Open Container Initiative（OCI）构件推送到 Amazon ECR 存储库。要查看此功能的示例，请使用以下步骤将 Helm 图表推送到 Amazon ECR。

有关在 Amazon EKS 中使用 Amazon ECR 托管的 Helm 图表的信息，请参阅在 Amazon EKS 集群中安装 Helm 图表。

将 Helm Chart 推送到 Amazon ECR 存储库

1. 安装最新版本的 Helm 客户端。这些步骤是使用 Helm 版本 `3.8.2` 编写的。有关更多信息，请参阅安装 Helm。
2. 请按照以下步骤创建测试 Helm Chart。有关更多信息，请参阅 Helm 文档 - 入门。

   a. 创建名为 `helm-test-chart` 的 Helm Chart 并清除 `templates` 目录的内容。

   ```
   helm create helm-test-chart
   rm -rf ./helm-test-chart/templates/*
   ```

   b. 在`templates`文件夹`ConfigMap`中创建。

   ```
   cd helm-test-chart/templates
   cat <<EOF > configmap.yaml
   apiVersion: v1
   ```

```
kind: ConfigMap
metadata:
  name: helm-test-chart-configmap
data:
  myvalue: "Hello World"
EOF
```

3. 打包图表。输出将包含您在推送 Helm Chart 时使用的打包图表的文件名。

   ```
   cd ../..
   helm package helm-test-chart
   ```

   输出

   ```
   Successfully packaged chart and saved it to: /Users/username/helm-test-
   chart-0.1.0.tgz
   ```

4. 创建存储库以存储 Helm Chart。存储库的名称应与步骤 2 中创建 Helm Chart 时使用的名称匹配。有关更多信息，请参阅 创建 Amazon ECR 私有存储库以存储映像。

   ```
   aws ecr create-repository \
        --repository-name helm-test-chart \
        --region us-west-2
   ```

5. 对要向其推送 Helm Chart 的 Amazon ECR 注册表验证 Docker 客户端的身份。必须针对每个注册表获得授权令牌，令牌有效期为 12 小时。有关更多信息，请参阅 Amazon ECR 中的私有注册表身份验证。

   ```
   aws ecr get-login-password \
        --region us-west-2 | helm registry login \
        --username AWS \
        --password-stdin aws_account_id.dkr.ecr.cn-north-1.amazonaws.com.cn
   ```

6. 使用 helm push 命令推送 Helm Chart。输出应包括 Amazon ECR 存储库 URI 和 SHA 摘要。

   ```
   helm push helm-test-chart-0.1.0.tgz oci://aws_account_id.dkr.ecr.cn-
   north-1.amazonaws.com.cn/
   ```

7. 描述您的 Helm Chart。

   ```
   aws ecr describe-images \
   ```

```
    --repository-name helm-test-chart \
    --region us-west-2
```

在输出中，验证 `artifactMediaType` 参数指示正确的构件类型。

```
{
    "imageDetails": [
        {
            "registryId": "aws_account_id",
            "repositoryName": "helm-test-chart",
            "imageDigest":
 "sha256:dd8aebdda7df991a0ffe0b3d6c0cf315fd582cd26f9755a347a52adEXAMPLE",
            "imageTags": [
                "0.1.0"
            ],
            "imageSizeInBytes": 1620,
            "imagePushedAt": "2021-09-23T11:39:30-05:00",
            "imageManifestMediaType": "application/vnd.oci.image.manifest.v1+json",
            "artifactMediaType": "application/vnd.cncf.helm.config.v1+json"
        }
    ]
}
```

8. （可选）有关其他步骤，请安装 Helm `ConfigMap` 并开始使用 Amazon EKS。有关更多信息，请参阅 在 Amazon EKS 集群中安装 Helm 图表。

对存储在 Amazon ECR 私有存储库中的映像进行签名

Amazon ECR Amazon Signer 与集成，为您提供了一种签署容器映像的方法。您可以将容器镜像和签名存储在私有存储库中。

注意事项

使用 Amazon ECR 镜像签名时应注意以下事项。

- 存储库中存储的签名计入每个存储库的最大镜像数量的服务限额。有关更多信息，请参阅 Amazon ECR 服务配额。
- 当存储库中存在参考项目时，Amazon ECR 生命周期策略将在主题图片删除后的 24 小时内自动清理这些项目。

先决条件

在您开始之前，必须满足以下先决条件。

- 安装并配置最新版本的 Amazon CLI。有关更多信息，请参阅《Amazon Command Line Interface 用户指南》中的安装或更新最新版本的 Amazon CLI。
- 安装符号CLI和符号 Amazon Signer 插件。有关更多信息，请参阅《Amazon Signer 开发人员指南》中的对容器镜像进行签名的先决条件。
- 签署存储在 Amazon ECR 私有存储库中的容器镜像。有关更多信息，请参阅 将映像推送至 Amazon ECR 私有存储库。

为 Notary 客户端配置身份验证

在使用 Notafication CLI 创建签名之前，必须先配置客户端，使其能够向 Amazon ECR 进行身份验证。如果您在安装 Notation 客户端的同一主机上安装了 Docker，那么 Notation 将重用您为 Docker 客户端使用的相同身份验证方法。Docker `login` 和 `logout` 命令将允许 Notation `sign` 和 `verify` 命令使用相同的凭证，并且您不必单独对 Notation 进行身份验证。有关配置 Notation 客户端进行身份验证的详细信息，请参阅 Notary Project 文档中的 Authenticate with OCI-compliant registries。

如果您没有使用 Docker 或其他使用 Docker 凭证的工具，那么我们建议使用 Amazon ECR Docker 凭证助手作为您的凭证存储。有关如何安装和配置 Amazon ECR 凭证助手的更多信息，请参阅 Amazon ECR Docker 凭证助手。

签署镜像

可以使用以下步骤创建签署容器镜像所需的资源，并将签名存储到 Amazon ECR 私有存储库中。Notation 使用摘要对镜像进行签名。

要签署映像

1. 使用 Amazon Signer 签名平台创建`Notation-OCI-SHA384-ECDSA`签名配置文件。您可以选择性地使用 `--signature-validity-period` 参数指定签名有效期。可以使用 `DAYS`、`MONTHS` 或 `YEARS` 指定此值。如果未指定有效期，则使用默认值 135 个月。

```
aws signer put-signing-profile --profile-name ecr_signing_profile --platform-id
 Notation-OCI-SHA384-ECDSA
```

> **Note**
>
> 签名配置文件名称仅支持字母数字字符和下划线（_）。

2. 根据您的默认注册表进行 Notation 客户端身份验证。以下示例使用对 Amazon ECR 私有注册表的 Lotation CLI 进行身份验证。 Amazon CLI

```
aws ecr get-login-password --region region | notation login --username AWS --
password-stdin 111122223333.dkr.ecr.region.amazonaws.com
```

3. 使用 Notation CLI 对映像进行签名，并且使用存储库名称和 SHA 摘要指定映像。这将创建签名并将其推送到正在签名的镜像所在的 Amazon ECR 私有存储库。

   在以下示例中，我们正在对`curl`带 SHA 摘要的存储库`sha256:ca78e5f730f9a789ef8c63bb55275ac12dfb9e8099e6EXAMPLE`中的映像进行签名。

```
notation
 sign 111122223333.dkr.ecr.region.amazonaws.com/
curl@sha256:ca78e5f730f9a789ef8c63bb55275ac12dfb9e8099e6EXAMPLE --plugin
 "com.amazonaws.signer.notation.plugin" --id "arn:aws:signer:region:111122223333:/
signing-profiles/ecrSigningProfileName"
```

后续步骤

对容器器映像进行签名后，可以在本地验证签名。有关验证映像的说明，请参阅 Amazon Signer Developer Guide 中的 Verify an image locally after signing。

从 Amazon ECR 私有存储库中删除签名和其他构件

您可以使用 ORAS 客户端列出和删除 Amazon ECR 私有存储库中的签名和其他引用类型构件。删除签名和其他引用构件与删除映像的方式类似（请参阅 删除 Amazon ECR 中的映像）。以下是列出构件和删除签名的方法：

要使用 ORAS CLI 管理映像构件

1. 安装和配置 ORAS 客户端。

有关安装和配置 ORAS 客户端的信息，请参阅 ORAS 文档中的 Installation。

2. 要列出 Amazon ECR 映像的可用构件，请使用 `oras discover`，后面加上映像名称：

   ```
   oras discover 111222333444.dkr.ecr.us-east-1.amazonaws.com/oci:helloworld
   ```

   输出应如下所示：

   ```
   111222333444.dkr.ecr.us-east-1.amazonaws.com/
   oci@sha256:88c0c54329bfdc1d94d6f58cd3fcb1226d46f58670f44a8c689cb3c9b37b6925
   ### application/vnd.cncf.notary.signature
       ### sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42
       ### sha256:6527bcec87adf1d55460666183b9d0968b3cd4e4bc34602d485206a219851171
   ```

3. 要使用 ORAS CLI 删除签名，如上例所示，请执行以下命令：

   ```
   oras manifest delete 111222333444.dkr.ecr.us-east-1.amazonaws.com/
   oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42
   ```

   输出应如下所示：

   ```
   Are you sure you want to delete the manifest "111222333444.dkr.ecr.us-
   east-1.amazonaws.com/
   oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42" and
    all tags associated with it? [y/N] y
   ```

4. 按 `y`。应删除构件。

要排查构件删除问题

如果签名删除（例如刚才显示的签名）失败，则会出现类似以下内容的输出。

```
Error response from registry: failed to delete 111222333444.dkr.ecr.us-
east-1.amazonaws.com/
oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42:
```

```
unsupported: Requested image referenced by manifest list:
 [sha256:005e2c97a6373e483799fa4ff29ac64a42dd10f08efcc166d6775f9b74943b5b]
```

删除在 OCI 1.1 发布之前推送的映像时，可能会发生此故障。如错误中所述，您必须先删除引用该映像的清单，然后才能删除该映像，如下所示：

1. 要删除与要删除的签名关联的清单，请键入：

   ```
    oras manifest delete 111222333444.dkr.ecr.us-east-1.amazonaws.com/
   oci@sha256:005e2c97a6373e483799fa4ff29ac64a42dd10f08efcc166d6775f9b74943b5b
   ```

   输出应如下所示：

   ```
   Are you sure you want to delete the manifest
    "sha256:005e2c97a6373e483799fa4ff29ac64a42dd10f08efcc166d6775f9b74943b5b" and all
    tags associated with it? [y/N] y
   ```

2. 按 y。应删除清单。
3. 清单消失后，您可以删除签名：

   ```
    oras manifest delete 111222333444.dkr.ecr.us-east-1.amazonaws.com/
   oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42
   ```

   输出应如下所示：按 y。

   ```
   Are you sure you want to delete the manifest
    "sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42" and all
    tags associated with it? [y/N] y
   Deleted [registry] 111222333444.dkr.ecr.us-east-1.amazonaws.com/
   oci@sha256:387c10c1598ee18aae81dcfc86d0d06d116e46461d1c3cda8927e69c48108c42
   ```

4. 要查看签名是否已删除，请键入：

   ```
   oras discover 111222333444.dkr.ecr.us-east-1.amazonaws.com/oci:helloworld
   ```

输出应如下所示：

```
111222333444.dkr.ecr.us-east-1.amazonaws.com/
oci@sha256:88c0c54329bfdc1d94d6f58cd3fcb1226d46f58670f44a8c689cb3c9b37b6925
### application/vnd.cncf.notary.signature
    ### sha256:6527bcec87adf1d55460666183b9d0968b3cd4e4bc34602d485206a219851171
```

在 Amazon ECR 中查看映像详细信息

将映像推送到存储库后，可以查看其信息。所包括的详细信息如下：

- 镜像 URI
- 镜像标签
- 构件媒体类型
- 镜像清单类型
- 扫描状态
- 镜像的大小 (以 MiB 为单位)
- 镜像推送到存储库的时间
- 复制状态

查看镜像详细信息 (Amazon Web Services Management Console)

1. 在[https://console.aws.amazon.com/ecr/存储库](https://console.aws.amazon.com/ecr/)中打开 Amazon ECR 控制台。
2. 从导航栏中，选择包含镜像所在存储库的区域。
3. 在导航窗格中，选择存储库。
4. 在存储库页面上，选择要查看的存储库。
5. 在 “存储库：***`repository_name`***” 页面上，选择要查看其详细信息的图像。

从 Amazon ECR 私有存储库中将映像提取到您的本地环境

如果希望运行 Amazon ECR 中可用的 Docker 镜像，可以使用 docker pull 命令将其提取到本地环境。您可以从默认注册表或与其他 Amazon 账户关联的注册表中执行此操作。

要在 Amazon ECS 任务定义中使用 Amazon ECR 镜像，请参阅 。将 Amazon ECR 映像与 Amazon ECS 结合使用

> **Important**
>
> Amazon ECR 要求用户有权通过 IAM policy 调用 `ecr:GetAuthorizationToken` API，然后才能对注册表进行身份验证并从任意 Amazon ECR 存储库推送或提取任意镜像。Amazon ECR 提供了多种 Amazon 托管策略来控制不同级别的用户访问权限。有关 Amazon ECR Amazon 托管策略的信息，请参阅Amazon Amazon 弹性容器注册表的托管策略。

从 Amazon ECR 存储库提取 Docker 镜像

1. 将您的 Docker 客户端验证到要从中提取镜像的 Amazon ECR 注册表。必须针对每个注册表获得授权令牌，令牌有效期为 12 小时。有关更多信息，请参阅 Amazon ECR 中的私有注册表身份验证。
2. (可选) 识别要提取的镜像。
    - 可以使用 aws ecr describe-repositories 命令列出注册表中的存储库：

      ```
      aws ecr describe-repositories
      ```

      上述示例注册表包含一个名为 `amazonlinux` 的存储库。
    - 可以使用 aws ecr describe-images 命令描述存储库中的镜像：

      ```
      aws ecr describe-images --repository-name amazonlinux
      ```

      上述示例存储库具有带标签 `latest` 和 `2016.09` 的镜像，并且镜像摘要为 `sha256:f1d4ae3f7261a72e98c6ebefe9985cf10a0ea5bd762585a43e0700ed99863807`。
3. 使用 docker pull 命令提取镜像。镜像名称格式应为 `registry/repository[:tag]` 以便按标签提取，或为 `registry/repository[@digest]` 以便按摘要提取。

```
docker pull aws_account_id.dkr.ecr.us-west-2.amazonaws.com/amazonlinux:latest
```

> ⚠ Important
>
> 如果您收到 *`repository-url`* `not found: does not exist or no pull access` 错误，您可能需要向 Amazon ECR 验证您的 Docker 客户端。有关更多信息，请参阅 Amazon ECR 中的私有注册表身份验证。

提取 Amazon Linux 容器映像

构建 Amazon Linux 容器镜像的软件组件与 Amazon Linux AMI 中包含的软件组件相同。Amazon Linux 容器映像作为 Docker 工作负载的基本映像，可用在任何环境中。如果您将亚马逊 Linux AMI 用于亚马逊中的应用程序 EC2，则可以使用亚马逊 Linux 容器镜像对应用程序进行容器化。

您可以在本地开发环境中使用 Amazon Linux 容器镜像，然后使用 Amazon ECS 将您的应用程序推送到 Amazon 使用 Amazon ECS。有关更多信息，请参阅 将 Amazon ECR 映像与 Amazon ECS 结合使用。

Amazon Linux 容器镜像在 和 Docker Hub 上可用。要获得 Amazon Linux 容器映像的支持，请转到 Amazon 开发人员论坛。

从 Docker Hub 提取 Amazon Linux 容器镜像

1. 使用 docker pull 命令提取 Amazon Linux 容器镜像。

```
docker pull amazonlinux
```

2. （可选）在本地运行容器。

```
docker run -it amazonlinux:latest /bin/bash
```

删除 Amazon ECR 中的映像

如果您结束使用镜像，可以从存储库中删除它。如果您结束使用存储库，可以删除整个存储库以及其中的所有镜像。有关更多信息，请参阅 在 Amazon ECR 中删除私有存储库。

作为手动删除镜像的替代方法，您可以创建存储库生命周期策略，以便更好地控制存储库中镜像的生命周期管理。生命周期策略自动执行此过程。有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像。

> **Note**
>
> 如果您的存储库中包含混合映像，其中一些映像是在 Amazon ECR 支持 OCI v1.1 之前推送的，则某些签名会有指向它们的映像索引或清单列表。因此，当您删除 OCI v1.1 之前的映像时，可能需要手动删除引用该映像的清单列表才能删除该构件。

删除镜像 (Amazon Web Services Management Console)

1. 在[https://console.aws.amazon.com/ecr/存储库](https://console.aws.amazon.com/ecr/)中打开 Amazon ECR 控制台。
2. 从导航栏中，选择包含要删除的镜像的区域。
3. 在导航窗格中，选择存储库。
4. 在存储库页面上，选择包含要删除的镜像的存储库。
5. 在 “存储库：***repository_name***” 页面上，选中要删除的图像左侧的复选框，然后选择 “删除”。
6. 在删除镜像对话框中，验证选定的镜像是否应被删除，然后选择删除。

删除镜像 (Amazon CLI)

1. 列出存储库中的镜像。带标签的镜像将具有镜像摘要以及相关标签的列表。不带标签的镜像仅具有镜像摘要。

   ```
   aws ecr list-images \
        --repository-name my-repo
   ```

2. (可选) 通过指定要删除镜像的关联标签来删除镜像的任何不需要的标签。从镜像中删除最后一个标签后，也会删除该镜像。

   ```
   aws ecr batch-delete-image \
        --repository-name my-repo \
        --image-ids imageTag=tag1 imageTag=tag2
   ```

3. 通过指定镜像摘要删除带标签或不带标签的镜像。在通过引用镜像摘要来删除镜像时，镜像及其所有标签都会被删除。

   ```
   aws ecr batch-delete-image \
        --repository-name my-repo \
        --image-ids imageDigest=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304EXAMPLE
   ```

要删除多个镜像，您可以在请求中指定多个镜像标签或镜像摘要。

```
aws ecr batch-delete-image \
     --repository-name my-repo \
     --image-ids imageDigest=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304EXAMPLE
 imageDigest=sha256:f5t0e245ssffc302b13e25962d8f7a0bd304EXAMPLE
```

重新标记 Amazon ECR 中的映像

借助 Docker Image Manifest V2 Schema 2 镜像，可以使用 `--image-tag` 命令的 put-image 选项重新为现有镜像添加标签。无需使用 Docker 提取或推送镜像，即可重新添加标签。对于大型镜像，此过程可大大节省重新为镜像添加标签所需的网络带宽和时间。

重新为镜像添加标签 (Amazon CLI)

要使用重新标记图像 Amazon CLI

1. 使用 batch-get-image 命令可获取要重新添加标签的镜像的镜像清单并将其写入文件中。在此示例中，存储库中带有标签的图像的清单被写入名为的环境变量*MANIFEST*。*latest* *amazonlinux*

   ```
   MANIFEST=$(aws ecr batch-get-image --repository-name amazonlinux --image-ids
    imageTag=latest --output text --query 'images[].imageManifest')
   ```

2. 使用 put-image 命令的 `--image-tag` 选项将镜像清单与新标签一起放置到 Amazon ECR 中。在此示例中，图像被标记为*2017.03*。

   > **Note**
   >
   > 如果该`--image-tag`选项在您的版本中不可用 Amazon CLI，请升级到最新版本。有关更多信息，请参阅Amazon Command Line Interface 《用户指南》中的安装 Amazon Command Line Interface。

   ```
   aws ecr put-image --repository-name amazonlinux --image-tag 2017.03 --image-
   manifest "$MANIFEST"
   ```

3. 验证您的新镜像标签是否已附加到您的镜像。在以下输出中，镜像具有标签 `latest` 和 `2017.03`。

```
aws ecr describe-images --repository-name amazonlinux
```

您可以在一个 (扩展) 代码行中执行所有这些操作：

```
{
    "imageDetails": [
        {
            "imageSizeInBytes": 98755613,
            "imageDigest":
 "sha256:8d00af8f076eb15a33019c2a3e7f1f655375681c4e5be157a26EXAMPLE",
            "imageTags": [
                "latest",
                "2017.03"
            ],
            "registryId": "aws_account_id",
            "repositoryName": "amazonlinux",
            "imagePushedAt": 1499287667.0
        }
    ]
}
```

重新为镜像添加标签 (Amazon Tools for Windows PowerShell)

要使用重新标记图像 Amazon Tools for Windows PowerShell

1. 使用获取Get-ECRImageBatchcmdlet要重新标记的图像的描述并将其写入环境变量。在此示例中，存储库中带有标签的图像被写入环境变量*$Image*。*latest* *amazonlinux*

   > **Note**
   >
   > 如果您的系统上没有Get-ECRImageBatchcmdlet可用的，请参阅《Amazon Tools for Windows PowerShell 用户指南》 Amazon Tools for Windows PowerShell中的设置。

   ```
   $Image = Get-ECRImageBatch -ImageId @{ imageTag="latest" } -
   RepositoryName amazonlinux
   ```

2. 将图像的清单写入*$Manifest*环境变量。

```
$Manifest = $Image.Images[0].ImageManifest
```

3. 使用`-ImageTag`选项将带有新标签Write-ECRImagecmdlet的图像清单放入 Amazon ECR。在此示例中，图像被标记为*2017.09*。

```
Write-ECRImage -RepositoryName amazonlinux -ImageManifest $Manifest -
ImageTag 2017.09
```

4. 验证您的新镜像标签是否已附加到您的镜像。在以下输出中，镜像具有标签 `latest` 和 `2017.09`。

```
Get-ECRImage -RepositoryName amazonlinux
```

   您可以在一个 (扩展) 代码行中执行所有这些操作：

```
ImageDigest                                                             ImageTag
-----------                                                             --------
sha256:359b948ea8866817e94765822787cd482279eed0c17bc674a7707f4256d5d497 latest
sha256:359b948ea8866817e94765822787cd482279eed0c17bc674a7707f4256d5d497 2017.09
```

防止映像标签在 Amazon ECR 中被覆盖

您可以在存储库中开启标签不可变性，以防止覆盖映像标签。为开启标签不可变性后，如果您推送某个标签已经在存储库中的映像，将返回 `ImageTagAlreadyExistsException` 错误。标签不可变性会影响所有标签。您不能使某些标签不可变，而另一些则不是。

您可以使用 Amazon Web Services Management Console 和 Amazon CLI 工具为新存储库或现有存储库设置图像标签的可变性。要使用控制台步骤创建存储库，请参阅创建 Amazon ECR 私有存储库以存储映像。

设置映像标签的可变性（Amazon Web Services Management Console）

要设置映像标签的可变性

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。
2. 从导航栏中，选择包含要编辑的存储库的区域。
3. 在导航窗格中，选择存储库。

4. 在 Repositories（存储库）页面上，选择 Private（私有）选项卡，然后选择要编辑的存储库，并选择 Edit（编辑）。
5. 对于标签不变性，选择存储库的标签可变性设置。配置有不可变标签的存储库会阻止覆盖镜像标签。有关更多信息，请参阅 防止映像标签在 Amazon ECR 中被覆盖。
6. 对于 Image scan settings（镜像扫描设置），尽管您可以在存储库级别为基础扫描指定扫描设置，但最佳实践是在私有注册表级别指定扫描配置。通过在私有注册表级别指定扫描设置，您可以启用增强扫描或基本扫描，并定义用于指定扫描哪些存储库的筛选条件。有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在软件漏洞。
7. 对于 Encryption settings（加密设置），此字段仅供查看，因为存储库的加密设置在存储库创建完成之后无法更改。
8. 选择保存以更新存储库设置。

设置映像标签的可变性（Amazon CLI）

创建配置有不可变标签的存储库

使用以下命令之一创建配置有不可变标签的新镜像存储库。

- create-repository (Amazon CLI)

```
aws ecr create-repository --repository-name name --image-tag-mutability IMMUTABLE --
region us-east-2
```

- 新-ECRRepository (Amazon Tools for Windows PowerShell)

```
New-ECRRepository -RepositoryName name -ImageTagMutability IMMUTABLE -Region us-
east-2 -Force
```

要更新存储库的映像标签可变性设置

使用以下命令之一更新现有存储库的镜像标签可变性设置。

- put-image-tag-mutability (Amazon CLI)

```
aws ecr put-image-tag-mutability --repository-name name --image-tag-
mutability IMMUTABLE --region us-east-2
```

- 写-ECRImage TagMutability (Amazon Tools for Windows PowerShell)

```
Write-ECRImageTagMutability -RepositoryName name -ImageTagMutability IMMUTABLE -
Region us-east-2 -Force
```

Amazon ECR 中的容器映像清单格式支持

Amazon ECR 支持以下容器镜像清单格式：

- Docker Image Manifest V2 Schema 1 (与 Docker 版本 1.9 和更早版本配合使用)
- Docker Image Manifest V2 Schema 2 (与 Docker 版本 1.10 和更新版本配合使用)
- Open Container Initiative（OCI）规范（v1.0 和 v1.1）

对 Docker Image Manifest V2 Schema 2 的支持可提供以下功能：

- 能够为单个镜像使用多个标签。
- 支持存储 Windows 容器镜像。

Amazon ECR 镜像清单转换

在 Amazon ECR 中推送和提取镜像时，您的容器引擎客户端 (例如 Docker) 将与注册表进行通信以就客户端了解的清单格式以及要用于镜像的注册表达成一致。

在使用 Docker 版本 1.9 或更旧版本将镜像推送到 Amazon ECR 时，镜像清单格式将存储为 Docker Image Manifest V2 Schema 1。在使用 Docker 版本 1.10 或更新版本将镜像推送到 Amazon ECR 时，镜像清单格式将存储为 Docker Image Manifest V2 Schema 2。

在从 Amazon ECR 按标签提取镜像时，Amazon ECR 将返回存储在存储库中的镜像清单格式。仅当客户端理解该格式时，才会将其返回。如果客户端不理解所存储的镜像清单格式，则 Amazon ECR 会将镜像清单转换为客户端能够理解的格式。例如，如果 Docker 1.9 客户端请求的镜像清单存储格式为 Docker Image Manifest V2 Schema 2，那么 Amazon ECR 将以 Docker Image Manifest V2 Schema 1 格式返回该清单。下表介绍了按标签提取镜像时 Amazon ECR 支持的可用转换：

客户端请求的架构	作为 V2 Schema 1 推送到 ECR	作为 V2 Schema 2 推送到 ECR	作为 OCI 推送到 ECR
V2 Schema 1	无需转换	已转换为 V2 Schema 1	无可用转换
V2 Schema 2	无可用转换，客户端将回退到 V2 Schema 1	无需转换	已转换为 V2 Schema 2
OCI	无可用转换	已转换为 OCI	无需转换

⚠ Important

如果按摘要提取镜像，则没有可用的转换。您的客户端必须了解存储在 Amazon ECR 中的镜像清单格式。如果您在 Docker 1.9 或更旧版本的客户端上按摘要请求 Docker Image Manifest V2 Schema 2 镜像，则无法提取镜像。有关更多信息，请参阅 Docker 文档中的注册表兼容性。

在此示例中，如果按标签请求同一镜像，Amazon ECR 会将镜像清单转换为客户端能够理解的格式。镜像提取成功。

将 Amazon ECR 映像与 Amazon ECS 结合使用

您可以使用 Amazon ECR 私有存储库来托管容器镜像和构件，您的 Amazon ECS 任务可能会从中提取这些镜像和构件。要使其发挥作用，Amazon ECS 或 Fargate 容器代理必须具有创建`ecr:BatchGetImage``ecr:GetDownloadUrlForLayer`、和的权限。`ecr:GetAuthorizationToken` APIs

所需的 IAM 权限

下表显示了每种启动类型要使用的 IAM 角色，该角色为您的任务从 Amazon ECR 私有存储库提取提供了所需的权限。Amazon ECS 提供包括所需权限的托管 IAM 策略。

启动类型	IAM 角色	Amazon 托管 IAM 策略
亚马逊实例上的 Amazon EC2 ECS	使用容器实例 IAM 角色，该角色与注册到您的 Amazon ECS 集群的亚马逊 EC2 实例相关联。有关更多信息，请参阅 Amazon Elastic Container Service 开发人员指南中的容器实例 IAM 角色。	`AmazonEC2ContainerServiceforEC2Role` 有关更多信息，请参阅 AmazonEC2ContainerServiceforEC2Role在 Amazon 弹性容器服务开发者指南中
Fargate 上的 Amazon ECS	使用您在 Amazon ECS 任务定义中引用的任务执行 IAM 角色。有关更多信息，请参阅 Amazon Elastic Container Service 开发人员指南中的任务执行 IAM 角色。	`AmazonECSTaskExecutionRolePolicy` 有关更多信息，请参阅 AmazonECSTaskExecutionRolePolicy在 Amazon 弹性容器服务开发者指南中。
外部实例上的 Amazon ECS	使用容器实例 IAM 角色，该角色与注册到 Amazon ECS 集群的本地服务器或虚拟机（VM）相关联。有关更多信息，请参阅 Amazon Elastic Container Service 开发人员指南中的容器实例 Amazon ECS 角色。	`AmazonEC2ContainerServiceforEC2Role` 有关更多信息，请参阅 AmazonEC2ContainerServiceforEC2Role在 Amazon 弹性容器服务开发者指南中。

Important

Amazon 托管 IAM 策略包含您可能不需要的额外权限即可使用。在这种情况下，这些是从 Amazon ECR 私有存储库提取所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
```

```
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

在 Amazon ECS 任务定义中指定 Amazon ECR 镜像

创建 Amazon ECS 任务定义时，您可以指定在 Amazon ECR 私有存储库中托管的容器镜像。在任务定义中，确保对您的 Amazon ECR 镜像使用完整的 `registry/repository:tag` 命名。例如：`aws_account_id.dkr.ecr.region.amazonaws.com/my-repository:latest`。

以下任务定义代码段显示了用于指定在 Amazon ECS 任务定义中的 Amazon ECR 中托管的容器映像的语法。

```
{
    "family": "task-definition-name",
    ...
    "containerDefinitions": [
        {
            "name": "container-name",
            "image": "aws_account_id.dkr.ecr.region.amazonaws.com/my-repository:latest",
            ...
        }
    ],
    ...
}
```

将 Amazon ECR 映像与 Amazon EKS 结合使用

您可以将 Amazon ECR 映像与 Amazon EKS 结合使用。

从 Amazon ECR 中引用映像时，您必须为映像使用完整的 `registry/repository:tag` 命名。例如，`aws_account_id.dkr.ecr.region.amazonaws.com/my-repository:latest`。

所需的 IAM 权限

如果您将 Amazon EKS 工作负载托管在托管节点、自管理节点或上 Amazon Fargate，请查看以下内容：

- 在托管式节点或自行管理的节点上托管的 Amazon EKS 工作负载：必须提供 Amazon EKS Worker 节点 IAM 角色（`NodeInstanceRole`）。Amazon EKS Worker 节点 IAM 角色必须包含以下适用于 Amazon ECR 的 IAM policy 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 如果您使用`eksctl`或 Amazon EKS 入门中的 Amazon CloudFormation 模板来创建集群和工作节点组，则默认情况下，这些 IAM 权限将应用于您的工作节点 IAM 角色。

- 托管于 Amazon EKS 工作负载 Amazon Fargate：使用 Fargate 容器执行角色，该角色允许你的 pod 从私有 Amazon ECR 存储库中提取映像。有关更多信息，请参阅创建 Fargate Pod 执行角色。

在 Amazon EKS 集群中安装 Helm 图表

在 Amazon ECR 中托管的 Helm Chart 可以安装在您的 Amazon EKS 集群上。

先决条件

- 安装最新版本的 Helm 客户端。这些步骤是使用 Helm 版本 `3.9.0` 编写的。有关更多信息，请参阅安装 Helm。
- 您至少已在计算机上安装了 Amazon CLI 的版本 `1.23.9` 或 `2.6.3`。有关更多信息，请参阅安装或更新 Amazon CLI的最新版本。
- 您已将 Helm Chart 推送到您的 Amazon ECR 存储库。有关更多信息，请参阅 将 Helm 图表推送到 Amazon ECR 私有存储库。
- 您已配置 `kubectl` 以使用 Amazon EKS。有关更多信息，请参阅 Amazon EKS 用户指南中的为 Amazon EKS 创建 `kubeconfig`。如果集群的以下命令成功，说明您已正确配置。

```
kubectl get svc
```

要在 Amazon EKS 集群中安装 Helm 图表

1. 向托管 Helm Chart 的 Amazon ECR 注册表验证您的 Helm 客户端。必须针对每个注册表获得授权令牌，令牌有效期为 12 小时。有关更多信息，请参阅 Amazon ECR 中的私有注册表身份验证。

```
aws ecr get-login-password \
     --region us-west-2 | helm registry login \
     --username AWS \
     --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

2. 安装图表。*helm-test-chart*用你的存储库和 *0.1.0* Helm 图表的标签替换。

```
helm install ecr-chart-demo oci://aws_account_id.dkr.ecr.region.amazonaws.com/helm-test-chart --version 0.1.0
```

输出应如下所示：

```
NAME: ecr-chart-demo
LAST DEPLOYED: Tue May 31 17:38:56 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

3. 验证图表安装。

```
helm list -n default
```

输出示例：

```
NAME            NAMESPACE       REVISION        UPDATED
     STATUS          CHART                      APP VERSION
ecr-chart-demo  default         1               2022-06-01 15:56:40.128669157 +0000
 UTC deployed        helm-test-chart-0.1.0   1.16.0
```

4. （可选）查看已安装的 Helm 图表 ConfigMap。

```
kubectl describe configmap helm-test-chart-configmap
```

5. 完成后，您可以从集群中删除图表版本。

```
helm uninstall ecr-chart-demo
```

在 Amazon ECR 中扫描映像是否存在软件漏洞

Amazon ECR 映像扫描有助于识别容器映像中的软件漏洞。提供以下扫描类型。

> ⚠ Important
>
> 在增强扫描、基本扫描和改进的基本扫描版本之间切换将导致先前建立的扫描不再可用。您必须重新设置扫描。但是，如果您切换回之前的扫描版本，则已建立的扫描将可用。

- 增强扫描 — Amazon ECR 与 Amazon Inspector 集成，可自动持续扫描您的存储库。将会扫描容器映像是否存在操作系统和编程语言包漏洞。当出现新的漏洞时，扫描结果会更新，Amazon Inspector 会发出一个事件 EventBridge 来通知您。增强扫描提供以下功能：
  - 操作系统和编程语言包漏洞
  - 两种扫描频率：按下扫描和连续扫描
- 基本扫描 — Amazon ECR 提供两个版本的基本扫描，它们使用常见漏洞和风险敞口 (CVEs) 数据库:.
  - Amazon 本机基本扫描-使用 Amazon 本机技术，现已推出 GA，建议使用。默认情况下，所有新的客户注册表都选择使用此改进版本。
  - Clair 基础扫描 — 使用开源 Clair 项目，已弃用。

  通过基本扫描，您可以将存储库配置为在推送时扫描，也可以执行手动扫描，而 Amazon ECR 会提供扫描结果列表。基本扫描提供以下功能：

  - 操作系统扫描
  - 两种扫描频率：手动和按下扫描

  > ⚠ Important
  >
  > 新版本的 Amazon ECR 基本扫描不使用 `DescribeImages` API 响应中的`imageScanFindingsSummary`和`imageScanStatus`属性来返回扫描结果。请改用 `DescribeImageScanFindings` API。有关更多信息，请参阅 [`DescribeImageScanFindings`](DescribeImageScanFindings)。

用于选择在 Amazon ECR 中扫描哪些存储库的筛选条件

为私有注册表配置映像扫描后，您可以使用筛选条件来选择扫描哪些存储库。

使用基本扫描时，您可以指定推送时扫描筛选条件，以指定将哪些存储库设置为在推送新映像时进行映像扫描。对于任何不符合基本扫描的推送筛选条件的存储库，都将设置为 manual（手动）扫描频率，这意味着必须手动触发扫描才能执行扫描。

使用 enhanced（增强）扫描时，您可以为推送扫描和连续扫描分别指定筛选条件。任何不符合增强扫描筛选条件的存储库都将禁用扫描。如果您使用增强扫描并为推送扫描和连续扫描分别指定了筛选条件，并且同一存储库符合多个筛选条件，则 Amazon ECR 会强制执行该存储库的连续扫描筛选条件，而不是推送扫描筛选条件。

筛选条件通配符

指定筛选条件后，没有通配符的筛选条件将匹配包含该筛选条件的所有存储库名称。带通配符 (*) 的筛选条件匹配任何存储库名称，通配符会在其中替换存储库名称中的零个或多个字符。

下表提供了示例，其中存储库名称在水平轴上表示，示例筛选条件在垂直轴上指定。

	prod	repo-prod	prod-repo	repo-prod-repo	prodrepo
prod	支持	是	是	是	是
*prod	支持	是	否	否	否
prod*	是	否	是	否	是
prod	支持	是	是	是	是
prod*repo	否	否	是	否	是

扫描映像以查找 Amazon ECR 中的操作系统和编程语言包漏洞

Amazon ECR 增强扫描是与 Amazon Inspector 的集成，它为您的容器镜像提供漏洞扫描。将会扫描容器映像是否存在操作系统和编程语言包漏洞。您可以使用 Amazon ECR 和 Amazon Inspector 直接查看扫描结果。有关 Amazon Inspector 的更多信息，请参阅 Amazon Inspector 用户指南中的使用 Amazon Inspector 扫描容器镜像。

借助增强扫描功能，您可以选择配置哪些存储库进行自动连续扫描，配置哪些存储库在推送时扫描。此操作通过设置扫描筛选条件完成。

增强扫描的注意事项

启用 Amazon ECR 增强扫描时考虑以下因素。

- 使用此功能不会收取 Amazon ECR 的额外费用，但是 Amazon Inspector 会收取扫描映像的费用。有关更多信息，请参阅 Amazon Inspector 定价。
- 以下区域不支持增强扫描：
  - 中东（阿联酋）(`me-central-1`)
  - 亚太地区（海得拉巴）（`ap-south-2`）
  - 以色列（特拉维夫）（`il-central-1`）
  - 亚太地区（墨尔本）(`ap-southeast-4`)
  - 欧洲（西班牙）（`eu-south-2`）
- Amazon Inspector 支持扫描特定操作系统。获取完整列表，请参阅 Amazon Inspector 用户指南中的支持的操作系统 - Amazon ECR 扫描。
- Amazon Inspector 使用服务相关 IAM 角色，该角色提供了为存储库提供增强扫描所需的权限。为私有注册表开启增强扫描后，Amazon Inspector 会自动创建服务相关 IAM 角色。有关更多信息，请参阅 Amazon Inspector 用户指南中的将服务相关角色用于 Amazon Inspector。
- 当您为私有注册表初始开启增强扫描时，Amazon Inspector 将仅根据映像推送时间戳识别过去 30 天内推送到 Amazon ECR 的映像，或提取过去 90 天的映像。更早映像的扫描状态将为 `SCAN_ELIGIBILITY_EXPIRED`。如果您需要让 Amazon Inspector 扫描这些映像，则应将其重新推送到您的存储库。
- 开启增强扫描后推送到 Amazon ECR 的所有映像都会在配置的持续时间内持续扫描。默认情况下，持续时间为生命周期。此设置可通过 Amazon Inspector 控制台来配置。有关更多信息，请参阅 更改 Amazon Inspector 中映像增强扫描的持续时间。
- 如果您的 Amazon ECR 私有注册表开启了增强扫描，则只能使用增强扫描功能来扫描符合扫描筛选条件的所有存储库。不符合任何筛选条件的任何存储库将使用 `Off` 扫描频率，并且不会被扫描。不支持使用增强扫描进行手动扫描。有关更多信息，请参阅 用于选择在 Amazon ECR 中扫描哪些存储库的筛选条件。
- 如果您为推送扫描和连续扫描分别指定了筛选条件，并且同一存储库符合多个筛选条件，则 Amazon ECR 会强制执行该存储库的连续扫描筛选条件，而不是推送扫描筛选条件。

- 启用增强扫描后，当存储库的扫描频率发生变化 EventBridge 时，Amazon ECR 会向发送一个事件。 EventBridge 当初始扫描完成以及创建、更新或关闭图像扫描结果时，Amazon Inspector 会发出事件。

在 Amazon ECR 中进行增强扫描所需的 IAM 权限

Amazon ECR 增强型扫描需要一个与 Amazon Inspector 服务相关的 IAM 角色，并且启用和使用增强扫描的 IAM 委托人有权调用扫描 APIs 所需的亚马逊 Inspector。为私有注册表开启增强扫描后，Amazon Inspector 会自动创建 Amazon Inspector 服务相关 IAM 角色。有关更多信息，请参阅 Amazon Inspector 用户指南中的将服务相关角色用于 Amazon Inspector。

以下 IAM policy 授予启用和使用增强扫描所需的权限。它包括 Amazon Inspector 创建服务相关 IAM 角色所需的权限，以及开启和关闭增强扫描和检索扫描结果所需的 Amazon Inspector API 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "inspector2:Enable",
                "inspector2:Disable",
                "inspector2:ListFindings",
                "inspector2:ListAccountPermissions",
                "inspector2:ListCoverage"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "inspector2.amazonaws.com"
                    ]
                }
            }
        }
    ]
```

```
}
```

在 Amazon ECR 中配置映像的增强扫描

为您的私有注册表配置每个区域的增强扫描。

验证您是否拥有配置增强扫描的适当 IAM 权限。有关信息，请参阅在 Amazon ECR 中进行增强扫描所需的 IAM 权限。

Amazon Web Services Management Console

要为私有注册表开启增强扫描

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。
2. 从导航栏中，选择要为其设置扫描配置的区域。
3. 在导航窗格中，选择私有注册表，然后选择设置。
4. 在 Scanning configuration（扫描配置）页面中，为 Scan type（扫描类型）选择 Enhanced scanning（增强扫描）。

   默认情况下，当增强扫描处于选中状态时，所有存储库都被连续扫描。
5. 要选择连续扫描的特定存储库，请清除连续扫描所有存储库复选框，然后定义筛选条件：

   > ⚠ Important
   >
   > 没有通配符的筛选条件将匹配包含该筛选条件的所有存储库名称。带通配符 (*) 的筛选条件匹配存储库名称，通配符会替换存储库名称中的零个或多个字符。要查看筛选条件行为的示例，请参阅the section called “筛选条件通配符”。

   a. 输入基于存储库名称的筛选条件，然后选择添加筛选条件。
   b. 决定推送映像时要扫描哪些存储库：
      - 要在推送时扫描所有存储库，请选择推送时扫描所有存储库。
      - 要选择在推送时扫描的特定存储库，请输入基于存储库名称的筛选条件，然后选择添加筛选条件。
6. 选择保存。
7. 在您要开启增强扫描的每个区域中重复这些步骤。

Amazon CLI

使用以下 Amazon CLI 命令开启对私有注册表的增强扫描 Amazon CLI。您可以使用 `rules` 对象指定扫描筛选条件。

- put-registry-scanning-configuration (Amazon CLI)

  以下示例将为您的私有注册表开启增强扫描。默认情况下，如果没有指定 `rules`，Amazon ECR 会将扫描配置设置为对所有存储库进行持续扫描。

```
aws ecr put-registry-scanning-configuration \
     --scan-type ENHANCED \
     --region us-east-2
```

  以下示例将为您的私有注册表开启增强扫描并指定扫描筛选条件。示例中的扫描筛选条件将对名称中带有 `prod` 的所有存储库开启持续扫描。

```
aws ecr put-registry-scanning-configuration \
     --scan-type ENHANCED \
     --rules '[{"repositoryFilters" : [{"filter":"prod","filterType" :
 "WILDCARD"}],"scanFrequency" : "CONTINUOUS_SCAN"}]' \
     --region us-east-2
```

  以下示例将为您的私有注册表开启增强扫描并指定多个扫描筛选条件。示例中的扫描筛选条件将对名称中带有 `prod` 的所有存储库开启持续扫描，并且对所有其他存储库开启仅在推送时扫描。

```
aws ecr put-registry-scanning-configuration \
     --scan-type ENHANCED \
     --rules '[{"repositoryFilters" : [{"filter":"prod","filterType" :
 "WILDCARD"}],"scanFrequency" : "CONTINUOUS_SCAN"},{"repositoryFilters" :
 [{"filter":"*","filterType" : "WILDCARD"}],"scanFrequency" : "SCAN_ON_PUSH"}]' \
     --region us-west-2
```

更改 Amazon Inspector 中映像增强扫描的持续时间

您可以更改 Amazon Inspector 连续扫描您 Amazon ECR 私有存储库中的映像的天数。默认情况下，当您的 Amazon ECR 私有注册表开启增强扫描时，Amazon Inspector 服务会持续监控您的存储库，直到删除映像或禁用增强扫描。可以使用 Amazon Inspector 设置更改 Amazon Inspector 扫描图像的持续时间。可用扫描持续时间为生命周期（默认）、180 天和 30 天。当存储库的扫描持续时间已过时，

在列出扫描漏洞时，将会显示 `SCAN_ELIGIBILITY_EXPIRED` 的扫描状态。有关更多信息，请参阅 Amazon Inspector 用户指南中的更改 Amazon ECR 自动重新扫描持续时间。

更改增强扫描持续时间设置

1. 在 https://console.aws.amazon.com/inspector/v2/ home 中打开 Amazon Inspector 控制台。
2. 在左侧导航中，展开 Settings（设置），然后选择 General（通用）。
3. 在 Settings（设置）页面上的 ECR re-scan duration（ECR 重新扫描持续时间）下，选择一个设置，然后选择 Save（保存）。

EventBridge 为了在 Amazon ECR 中进行增强扫描而发送的事件

启用增强扫描后，当存储库的扫描频率发生变化 EventBridge 时，Amazon ECR 会向发送一个事件。EventBridge 当初始扫描完成以及创建、更新或关闭图像扫描结果时，Amazon Inspector 会向其发送事件。

存储库扫描频率更改事件

为注册表开启增强扫描后，当开启了增强扫描的资源发生更改时，Amazon ECR 将发送以下事件。这包括正在创建新存储库、正在更改存储库的扫描频率，或者在开启了增强扫描功能的存储库中创建或删除映像。有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在软件漏洞。

```
{
 "version": "0",
 "id": "0c18352a-a4d4-6853-ef53-0abEXAMPLE",
 "detail-type": "ECR Scan Resource Change",
 "source": "aws.ecr",
 "account": "123456789012",
 "time": "2021-10-14T20:53:46Z",
 "region": "us-east-1",
 "resources": [],
 "detail": {
  "action-type": "SCAN_FREQUENCY_CHANGE",
  "repositories": [{
    "repository-name": "repository-1",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-1",
    "scan-frequency": "SCAN_ON_PUSH",
    "previous-scan-frequency": "MANUAL"
   },
   {
    "repository-name": "repository-2",
```

```
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-2",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   },
   {
    "repository-name": "repository-3",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-3",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   }
  ],
  "resource-type": "REPOSITORY",
  "scan-type": "ENHANCED"
 }
}
```

初始镜像扫描的事件（增强扫描）

为注册表开启增强扫描后，当初始映像扫描完成时，Amazon Inspector 会发送以下事件。`finding-severity-counts` 参数仅返回严重性级别的值 (如果存在)。例如，如果镜像不包含任何 `CRITICAL` 级别的结果，则不会返回任何关键计数。有关更多信息，请参阅 扫描映像以查找 Amazon ECR 中的操作系统和编程语言包漏洞。

事件模式：

```
{
  "source": ["aws.inspector2"],
  "detail-type": ["Inspector2 Scan"]
}
```

输出示例：

```
{
    "version": "0",
    "id": "739c0d3c-4f02-85c7-5a88-94a9EXAMPLE",
    "detail-type": "Inspector2 Scan",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2021-12-03T18:03:16Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-sample"
    ],
```

```
    "detail": {
        "scan-status": "INITIAL_SCAN_COMPLETE",
        "repository-name": "arn:aws:ecr:us-east-2:123456789012:repository/amazon/
amazon-ecs-sample",
        "finding-severity-counts": {
            "CRITICAL": 7,
            "HIGH": 61,
            "MEDIUM": 62,
            "TOTAL": 158
        },
        "image-digest":
 "sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77e5EXAMPLE",
        "image-tags": [
            "latest"
        ]
    }
}
```

镜像扫描结果更新的事件（增强扫描）

为注册表开启增强扫描后，在创建、更新或关闭映像扫描结果时，Amazon Inspector 会发送以下事件。有关更多信息，请参阅 扫描映像以查找 Amazon ECR 中的操作系统和编程语言包漏洞。

事件模式：

```
{
  "source": ["aws.inspector2"],
  "detail-type": ["Inspector2 Finding"]
}
```

输出示例：

```
{
    "version": "0",
    "id": "42dbea55-45ad-b2b4-87a8-afaEXAMPLE",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2021-12-03T18:02:30Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-sample/
sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77eEXAMPLE"
```

```
    ],
    "detail": {
        "awsAccountId": "123456789012",
        "description": "In libssh2 v1.9.0 and earlier versions, the SSH_MSG_DISCONNECT
 logic in packet.c has an integer overflow in a bounds check, enabling an attacker to
 specify an arbitrary (out-of-bounds) offset for a subsequent memory read. A crafted
 SSH server may be able to disclose sensitive information or cause a denial of service
 condition on the client system when a user connects to the server.",
        "findingArn": "arn:aws:inspector2:us-east-2:123456789012:finding/
be674aaddd0f75ac632055EXAMPLE",
        "firstObservedAt": "Dec 3, 2021, 6:02:30 PM",
        "inspectorScore": 6.5,
        "inspectorScoreDetails": {
            "adjustedCvss": {
                "adjustments": [],
                "cvssSource": "REDHAT_CVE",
                "score": 6.5,
                "scoreSource": "REDHAT_CVE",
                "scoringVector": "CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:N",
                "version": "3.0"
            }
        },
        "lastObservedAt": "Dec 3, 2021, 6:02:30 PM",
        "packageVulnerabilityDetails": {
            "cvss": [
                {
                    "baseScore": 6.5,
                    "scoringVector": "CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:N",
                    "source": "REDHAT_CVE",
                    "version": "3.0"
                },
                {
                    "baseScore": 5.8,
                    "scoringVector": "AV:N/AC:M/Au:N/C:P/I:N/A:P",
                    "source": "NVD",
                    "version": "2.0"
                },
                {
                    "baseScore": 8.1,
                    "scoringVector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:H",
                    "source": "NVD",
                    "version": "3.1"
                }
            ],
```

```
            "referenceUrls": [
                "https://access.redhat.com/errata/RHSA-2020:3915"
            ],
            "source": "REDHAT_CVE",
            "sourceUrl": "https://access.redhat.com/security/cve/CVE-2019-17498",
            "vendorCreatedAt": "Oct 16, 2019, 12:00:00 AM",
            "vendorSeverity": "Moderate",
            "vulnerabilityId": "CVE-2019-17498",
            "vulnerablePackages": [
                {
                    "arch": "X86_64",
                    "epoch": 0,
                    "name": "libssh2",
                    "packageManager": "OS",
                    "release": "12.amzn2.2",
                    "sourceLayerHash":
 "sha256:72d97abdfae3b3c933ff41e39779cc72853d7bd9dc1e4800c5294dEXAMPLE",
                    "version": "1.4.3"
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "Update all packages in the vulnerable packages section to
 their latest versions."
            }
        },
        "resources": [
            {
                "details": {
                    "awsEcrContainerImage": {
                        "architecture": "amd64",
                        "imageHash":
 "sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77e5EXAMPLE",
                        "imageTags": [
                            "latest"
                        ],
                        "platform": "AMAZON_LINUX_2",
                        "pushedAt": "Dec 3, 2021, 6:02:13 PM",
                        "registry": "123456789012",
                        "repositoryName": "amazon/amazon-ecs-sample"
                    }
                },
```

```
                "id": "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-
sample/sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77EXAMPLE",
                "partition": "N/A",
                "region": "N/A",
                "type": "AWS_ECR_CONTAINER_IMAGE"
            }
        ],
        "severity": "MEDIUM",
        "status": "ACTIVE",
        "title": "CVE-2019-17498 - libssh2",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Dec 3, 2021, 6:02:30 PM"
    }
}
```

检索 Amazon ECR 中增强扫描的调查发现

您可以检索上次完成的增强映像扫描的扫描调查发现，然后在 Amazon Inspector 中打开调查发现以查看更多详细信息。已发现的软件漏洞是根据常见漏洞和风险敞口 (CVEs) 数据库按严重程度列出的。

有关扫描镜像时的常见问题的排查详细信息，请参阅 排查 Amazon ECR 中的映像扫描问题。

Amazon Web Services Management Console

通过 Amazon Web Services Management Console使用以下步骤检索镜像扫描结果。

要检索映像扫描调查发现

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择您的存储库所在的区域。
3. 在导航窗格中，选择存储库。
4. 在存储库页面上，选择包含要扫描检索结果的镜像的存储库。
5. 在映像页面的映像标签列下，选择映像标签以检索扫描调查发现。
6. 要在 Amazon Inspector 控制台中查看更多详细信息，请在名称列中选择漏洞名称。

Amazon CLI

使用以下 Amazon CLI 命令使用检索图像扫描结果 Amazon CLI。您可以使用 `imageTag` 或 `imageDigest` 指定镜像，这两者都可以使用 list-images CLI 命令获取。

- describe-image-scan-findings (Amazon CLI)

  以下示例使用镜像标签。

  ```
  aws ecr describe-image-scan-findings \
       --repository-name name \
       --image-id imageTag=tag_name \
       --region us-east-2
  ```

  以下示例使用镜像摘要。

  ```
  aws ecr describe-image-scan-findings \
       --repository-name name \
       --image-id imageDigest=sha256_hash \
       --region us-east-2
  ```

在 Amazon ECR 中扫描映像是否存在操作系统漏洞

Amazon ECR 提供了两个版本的基本扫描，它们使用常见漏洞和暴露 (CVEs) 数据库：

- Amazon 本机基本扫描-使用 Amazon 本机技术，现已推出 GA，建议使用。这种经过改进的基本扫描旨在为客户提供更好的扫描结果和跨各种流行操作系统的漏洞检测。这使客户能够进一步增强其容器镜像的安全性。默认情况下，所有新的客户注册表都选择使用此改进版本。
- Clair 基本扫描 — 以前的基本扫描版本，使用开源 Clair 项目，现已弃用。有关 Clair 的更多信息，请参阅上的 Clair。 GitHub

“按地区划分的Amazon 服务” 中列出的所有区域均支持 Amazon 原生扫描和 Clair 基本扫描，但 2024 年 9 月之后添加的区域除外。由于 Clair 支持已被弃用，因此新增区域将不支持 Clair，从 2025 年 10 月 1 日起，所有地区都将不再支持 Clair。

Amazon ECR 使用来自上游分配源的 CVE 的严重性（如果可用）。否则，将使用常见漏洞评分系统（CVSS）评分。CVSS 评分可用于获取 NVD 漏洞严重性评级。有关更多信息，请参阅 NVD 漏洞严重性评级。

这两个版本的 Amazon ECR 基本扫描都支持筛选条件，以指定推送时要扫描哪些存储库。任何不符合推送时扫描筛选条件的存储库都将设置为手动扫描频率，这意味着必须手动开始扫描。每 24 小时扫描一次映像。24 小时包括初始推送时扫描（如已配置）以及任何手动扫描。使用基本扫描，在给定注册

表中，您每 24 小时最多可以扫描 100,000 张图像。100,000 的限制包括推送式初始扫描和手动扫描，包括 Clair 和改进版的基本扫描。

可以为每个镜像检索上次完成的镜像扫描结果。图像扫描完成后，Amazon ECR 会向亚马逊 EventBridge发送一个事件。有关更多信息，请参阅 亚马逊 ECR 事件和 EventBridge。

操作系统支持基本扫描及改进的基本扫描

作为一项安全最佳实践并为了持续提供覆盖，建议您继续使用受支持的操作系统版本。根据供应商政策，停用的操作系统不再更新修补程序，而且在许多情况下，也不再发布新的安全公告。此外，当受影响的操作系统的标准支持期结束时，一些供应商会从他们的信息源中删除现有的安全公告和检测。分发失去其供应商的支持后，Amazon ECR 可能不再支持扫描它是否存在漏洞。Amazon ECR 仍为已停用操作系统生成的所有调查发现都应仅供参考。下方列出了当前支持的操作系统和版本。

操作系统	版本	Amazon 原生基础版	克莱尔基本款
Alpine Linux (Alpine)	3.18	是	是
Alpine Linux (Alpine)	3.19	是	是
Alpine Linux (Alpine)	3.20	是	是
Alpine Linux (Alpine)	3.21	是	没有
AlmaLinux	8	是	没有
AlmaLinux	9	是	没有
亚马逊 Linux (2AL2)	AL2	是	是
亚马逊 Linux 2023 (AL2023)	AL2023	是	是

操作系统	版本	Amazon 原生基础版	克莱尔基本款
Debian 服务器 (Bookworm)	12	是	是
Debian 服务器 (Bullseye)	11	是	是
Fedora	40	是	没有
Fedora	41	是	没有
OpenSUSE Leap	15.6	是	没有
Oracle Linux (Oracle)	9	是	是
Oracle Linux (Oracle)	8	是	是
Photon OS	4	是	没有
Photon OS	5	是	没有
Red Hat Enterprise Linux (RHEL)	8	是	是
Red Hat Enterprise Linux (RHEL)	9	是	是
Rocky Linux	8	是	没有
Rocky Linux	9	是	没有

操作系统	版本	Amazon 原生基础版	克莱尔基本款
SUSE Linux Enterprise Server (SLES)	15.6	是	没有
Ubuntu (Xenial)	16.04 (ESM)	是	是
Ubuntu (Bionic)	18.04 (ESM)	是	是
Ubuntu (Focal)	20.04 (LTS)	是	是
Ubuntu (Jammy)	22.04 (LTS)	是	是
Ubuntu（Noble Numbat）	24.04	是	没有
Ubuntu（Oracular Oriole））	24.10	是	没有

在 Amazon ECR 中配置映像的基本扫描

默认情况下，Amazon ECR 为所有私有注册表开启基本扫描。因此，除非您更改了私有注册表上的扫描设置，否则不需要开启基本扫描。基本扫描使用开源 Clair 项目。

您可以使用以下步骤定义一个或多个推送时扫描筛选条件。

要为私有注册表开启基本扫描

1. 在私有注册表/存储库中打开 Amazon ECR 控制台 https://console.aws.amazon.com/ecr/
2. 从导航栏中，选择要为其设置扫描配置的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Scanning（扫描）。
4. 在 Scanning configuration（扫描配置）页面中，为 Scan type（扫描类型）选择 Basic scanning（基本扫描）。

5. 默认情况下，您的所有存储库都设置为手动扫描。您可以选择通过指定推送时扫描筛选条件来开启推送时扫描。您可以为所有存储库或单个存储库设置推送时扫描。有关更多信息，请参阅 用于选择在 Amazon ECR 中扫描哪些存储库的筛选条件。

在 Amazon ECR 中切换到改进的映像基本扫描功能

Amazon ECR 通过使用 Amazon 原生技术的改进版基本扫描来提供增强的容器图像扫描功能。此功能可帮助您识别容器镜像中的软件漏洞。如果您使用的是使用`CLAIR`技术的先前版本的基本扫描，则以下步骤可帮助您切换到此基本扫描的改进版本。

> ⚠ Important
>
> 对于新用户，您的注册表将在创建时自动配置为使用 `AWS_NATIVE` 扫描技术。您无需采取任何操作。Amazon ECR 不建议恢复到以前的扫描技术 `CLAIR`。

Amazon Web Services Management Console

要为私有注册表开启改进的基本扫描

1. 在私有注册表/存储库中打开 Amazon ECR 控制台 https://console.aws.amazon.com/ecr/
2. 从导航栏中，选择要为其设置扫描配置的区域。
3. 在导航窗格中，选择私有注册表、功能和设置、扫描。
4. 在扫描配置页面上，选择加入（推荐）以选择基本扫描的改进版本。
5. 默认情况下，您的所有存储库都设置为手动扫描。您可以选择通过指定推送时扫描筛选条件来开启推送时扫描。您可以为所有存储库或单个存储库设置推送时扫描。有关更多信息，请参阅 用于选择在 Amazon ECR 中扫描哪些存储库的筛选条件。

Amazon CLI

Amazon ECR 为所有私有注册表启用了基本扫描。使用以下命令查看您当前的基本扫描类型，并更改您的基本扫描类型。

- 要检索您当前使用的基本扫描类型版本。

```
aws ecr get-account-setting --name BASIC_SCAN_TYPE_VERSION
```

参数名称为必填字段。如果您不提供名称，则将收到以下错误：

```
aws: error: the following arguments are required: --name
```

要将您的基本扫描类型版本从 `CLAIR` 更改为 `AWS_NATIVE`。一旦您将基本扫描类型版本从 `CLAIR` 更改为 `AWS_NATIVE` 后，建议不要恢复为 `CLAIR`。

```
aws ecr put-account-setting --name BASIC_SCAN_TYPE_VERSION --value value
```

在 Amazon ECR 中手动扫描映像是否存在操作系统漏洞

如果您的存储库未配置为推送时扫描，则可以手动启动映像扫描。每 24 小时扫描一次映像。24 小时包括初始推送时扫描（如已配置）以及任何手动扫描。

有关扫描镜像时的常见问题的排查详细信息，请参阅 排查 Amazon ECR 中的映像扫描问题。

Amazon Web Services Management Console

通过 Amazon Web Services Management Console使用以下步骤开始手动镜像扫描。

1. 在私有注册表/存储库中打开 Amazon ECR 控制台 https://console.aws.amazon.com/ecr/
2. 从导航栏中，选择您创建存储库的区域。
3. 在导航窗格中，选择存储库。
4. 在存储库页面上，选择包含要扫描的镜像的存储库。
5. 在镜像页面上，选择要扫描的镜像，然后选择扫描。

Amazon CLI

- start-image-scan (Amazon CLI)

  以下示例使用镜像标签。

```
aws ecr start-image-scan --repository-name name --image-id imageTag=tag_name --
region us-east-2
```

  以下示例使用镜像摘要。

```
aws ecr start-image-scan --repository-name name --image-id imageDigest=sha256_hash
 --region us-east-2
```

Amazon Tools for Windows PowerShell

- 获取-ECRImage ScanFinding (Amazon Tools for Windows PowerShell)

  以下示例使用镜像标签。

  ```
  Start-ECRImageScan -RepositoryName name -ImageId_ImageTag tag_name -Region us-
  east-2 -Force
  ```

  以下示例使用镜像摘要。

  ```
  Start-ECRImageScan -RepositoryName name -ImageId_ImageDigest sha256_hash -
  Region us-east-2 -Force
  ```

检索 Amazon ECR 中基本扫描的调查发现

您可以检索上次完成的基本映像扫描的扫描调查发现。已发现的软件漏洞是根据常见漏洞和风险敞口 (CVEs) 数据库按严重程度列出的。

有关扫描镜像时的常见问题的排查详细信息，请参阅 排查 Amazon ECR 中的映像扫描问题。

Amazon Web Services Management Console

通过 Amazon Web Services Management Console使用以下步骤检索镜像扫描结果。

要检索映像扫描调查发现

1. 在私有注册表/存储库中打开 Amazon ECR 控制台 https://console.aws.amazon.com/ecr/
2. 从导航栏中，选择您创建存储库的区域。
3. 在导航窗格中，选择存储库。
4. 在存储库页面上，选择包含要扫描检索结果的镜像的存储库。
5. 在映像页面的映像标签列下，选择映像标签以检索扫描调查发现。

Amazon CLI

使用以下 Amazon CLI 命令使用检索图像扫描结果 Amazon CLI。您可以使用 `imageTag` 或 `imageDigest` 指定镜像，这两者都可以使用 list-images CLI 命令获取。

- describe-image-scan-findings (Amazon CLI)

  以下示例使用镜像标签。

  ```
  aws ecr describe-image-scan-findings --repository-name name --image-id
   imageTag=tag_name --region us-east-2
  ```

  以下示例使用镜像摘要。

  ```
  aws ecr describe-image-scan-findings --repository-name name --image-id
   imageDigest=sha256_hash --region us-east-2
  ```

Amazon Tools for Windows PowerShell

- 获取-ECRImage ScanFinding (Amazon Tools for Windows PowerShell)

  以下示例使用镜像标签。

  ```
  Get-ECRImageScanFinding -RepositoryName name -ImageId_ImageTag tag_name -
  Region us-east-2
  ```

  以下示例使用镜像摘要。

  ```
  Get-ECRImageScanFinding -RepositoryName name -ImageId_ImageDigest sha256_hash -
  Region us-east-2
  ```

排查 Amazon ECR 中的映像扫描问题

以下是常见的镜像扫描失败。您可以在 Amazon ECR 控制台中显示图片详情、通过 API 或使用 API 查看此类错误。 Amazon CLI `DescribeImageScanFindings`

UnsupportedImageError

尝试对使用 Amazon ECR 不支持其基础镜像扫描的操作系统构建的镜像进行基础扫描时，您可能会得到 `UnsupportedImageError` 错误。Amazon ECR 支持 Amazon Elastic Container Storage Storage Simple Storage、Amazon Linux 2、Debian、Ubuntu、CentOS、Oracle Linux、Alpine 和 RHEL Linux 发行版的主要版本。一旦分发失去其供应商的支持，Amazon ECR 可能不再支持扫描它是否存在漏洞。Amazon ECR 不支持扫描从 Docker scratch 镜像构建的镜像。

> ⚠ Important
>
> 使用增强型扫描时，Amazon Inspector 支持扫描特定操作系统和媒体类型。有关完整列表，请参阅 Amazon Inspector 用户指南中的支持的操作系统和媒体类型。

返回 `UNDEFINED` 严重性级别

您可能会收到严重性级别为 `UNDEFINED` 的扫描结果。造成这种情况的常见原因如下：

- CVE 源未向该漏洞分配优先级。
- 该漏洞被分配了一个 Amazon ECR 无法识别的优先级。

要确定漏洞的严重性和描述，您可以直接从源查看 CVE。

了解扫描状态 `SCAN_ELIGIBILITY_EXPIRED`

如果您的私有注册表启用了使用 Amazon Inspector 的增强扫描，并且您正在查看扫描漏洞，则可能会看到扫描状态为 `SCAN_ELIGIBILITY_EXPIRED`。造成这种情况的常见原因如下。

- 当您为私有注册表初始开启增强扫描时，Amazon Inspector 将仅根据映像推送时间戳识别过去 30 天内推送到 Amazon ECR 的映像。更早映像的扫描状态将为 `SCAN_ELIGIBILITY_EXPIRED`。如果您需要让 Amazon Inspector 扫描这些映像，则应将其重新推送到您的存储库。
- 如果在 Amazon Inspector 控制台中更改了 ECR 重新扫描持续时间并且该时间已过，则镜像的扫描状态将会更改为 `inactive` 并显示原因代码 `expired`，并且将会计划关闭该镜像的所有关联调查结果。这会导致 Amazon ECR 控制台将扫描状态列为 `SCAN_ELIGIBILITY_EXPIRED`。

将上游注册表与 Amazon ECR 私有注册表同步

使用缓存提取规则，您可以将上游注册表的内容与 Amazon ECR 私有注册表同步。

Amazon ECR 目前支持为以下上游注册表创建直通缓存规则：

- Amazon ECR Public、Kubernetes 容器镜像注册表和 Quay（不需要身份验证）
- Docker Hub、Microsoft Azure GitHub 容器注册表、容器注册表和 GitLab 容器注册表（需要使用 Amazon Secrets Manager 密钥进行身份验证）
- Amazon ECR（需要使用 Amazon IAM 角色进行身份验证）

对于 GitLab 容器注册表，Amazon ECR 仅支持通过软件即服务 (SaaS) 产品提取缓存。 GitLab有关使用 GitLab的 SaaS 产品的更多信息，请参阅 GitLab.com。

对于需要使用密钥进行身份验证的上游注册表（例如 Docker Hub），您必须将凭据存储在密钥中。 Amazon Secrets Manager 您可以使用 Amazon ECR 控制台为每个经过身份验证的上游注册表创建 Secrets Manager 密钥。有关使用 Secrets Manager 控制台创建 Secrets Manager 密钥的更多信息，请参阅将您的上游存储库凭证存储在 Amazon Secrets Manager 密钥中。

对于 Amazon ECR，如果上游和下游 Amazon ECR 注册表属于不同的账户，则必须创建一个 IAM 角色。 Amazon 有关创建 IAM 角色的更多信息，请参阅跨账户 ECR 到 ECR 通过缓存提取所需的 IAM 策略。

为上游注册表创建了拉取缓存规则后，使用您的 Amazon ECR 私有注册表 URI 从该上游注册表中提取映像。然后，Amazon ECR 会创建一个存储库，在私有注册表中缓存该映像。对于使用给定标签的缓存图像的后续拉取请求，Amazon ECR 会检查上游注册表中是否有带有该特定标签的镜像的新版本，并尝试至少每 24 小时更新一次私有注册表中的镜像。

存储库创建模板

Amazon ECR 已支持存储库创建模板。这样的话，在为 Amazon ECR 使用缓存提取规则代表您创建的新存储库指定初始配置时，您有控制权。每个模板都包含存储库命名空间前缀，用于将新存储库与特定模板匹配。模板可以指定所有存储库设置的配置，包括基于资源的访问策略、标签不变性、加密和生命周期策略。存储库创建模板中的设置仅在存储库创建期间应用，对现有存储库或使用任何其他方法创建的存储库没有任何影响。有关更多信息，请参阅 用于控制在缓存提取或复制操作期间创建的存储库的模板。

使用缓存提取规则的注意事项

使用 Amazon ECR 缓存提取规则时，请考虑以下内容。

- 以下区域不支持创建缓存提取规则。
  - 中国（北京）(`cn-north-1`)
  - 中国（宁夏）(`cn-northwest-1`)
  - Amazon GovCloud （美国东部）(`us-gov-east-1`)
  - Amazon GovCloud （美国西部）(`us-gov-west-1`)
- Amazon Lambda 不支持使用拉取缓存规则从 Amazon ECR 提取容器映像。
- 如果使用提取缓存提取镜像，首次提取镜像时不支持 Amazon ECR FIPS 服务终端节点。但是，使用 Amazon ECR FIPS 服务终端节点可以处理后续提取。
- 当通过 Amazon ECR 私有注册表 URI 提取缓存图像时，图像提取将由 Amazon IP 地址启动。这可以确保映像提取不被计入上游注册表所实施的任何提取率配额。
- 当通过 Amazon ECR 私有注册表 URI 提取缓存的映像时，Amazon ECR 至少每 24 小时检查一次上游存储库，以验证缓存的映像是否为最新版本。如果上游注册表中有较新的映像，Amazon ECR 会尝试更新缓存的映像。此计时器基于缓存镜像的最后一次提取。
- 如果 Amazon ECR 出于任何原因无法从上游注册表更新映像，并且映像已提取，则系统仍会提取最后缓存的映像。
- 在创建包含上游注册表凭证的 Secrets Manager 密钥时，密钥名称必须使用 `ecr-pullthroughcache/` 前缀。密钥还必须与缓存提取规则位于相同的账户和区域。
- 当使用缓存提取规则提取多架构镜像时，清单列表和清单列表中引用的每个镜像都会被提取到 Amazon ECR 存储库中。如果您只想提取特定架构，则可以使用与架构关联的镜像摘要或标签，而不是与清单列表关联的标签来提取镜像。
- Amazon ECR 使用服务相关的 IAM 角色，为 Amazon ECR 提供所需的权限，以创建存储库、检索用于身份验证的 Secrets Manager 密钥，以及代表您推送缓存映像。服务相关 IAM 角色在创建缓存提取规则时自动创建。有关更多信息，请参阅 用于缓存提取的 Amazon ECR 服务相关角色。
- 默认情况下，提取缓存映像的 IAM 主体具有通过其 IAM policy 授予的权限。您可以使用 Amazon ECR 私有注册表权限策略进一步限定 IAM 实体的权限范围。有关更多信息，请参阅 使用注册表权限。
- 使用缓存提取工作流创建的 Amazon ECR 存储库与任何其他 Amazon ECR 存储库受到同等对待。支持所有存储库功能，例如复制和镜像扫描。
- 当 Amazon ECR 使用提取缓存操作代表您创建新存储库时，除非存在匹配的存储库创建模板，否则以下默认设置将应用于存储库。您可以使用存储库创建模板，定义 Amazon ECR 代表您创建的存

储库所应用的设置。有关更多信息，请参阅 用于控制在缓存提取或复制操作期间创建的存储库的模板。

  - 标签不可变性-关闭后，标签是可变的，可以被覆盖。
  - 加密-使用默认AES256加密。
  - 存储库权限-省略，不应用任何存储库权限策略。
  - 生命周期策略-省略，不应用任何生命周期策略。
  - 资源标签-省略，不应用任何资源标签。
- 使用缓存提取规则为存储库启用映像标签不变性，这将阻止 Amazon ECR 使用相同的标签更新映像。
- 首次使用缓存提取规则提取映像时，可能需要通往互联网的路由。在某些情况下，需要通往互联网的路由，因此最好设置一条这样的路由，以免出现任何故障。因此，如果您已将 Amazon ECR 配置为使用接口 VPC 终端节点， Amazon PrivateLink 则需要确保第一次拉取具有通往互联网的路由。一种方法是在同个 VPC 中创建带有互联网网关的公有子网，然后将流至互联网的所有出站流量从私有子网路由到公有子网。使用缓存提取规则进行的后续映像提取不需要此操作。有关更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的示例路由选项。

将上游注册表与 Amazon ECR 私有注册表同步所需的 IAM 权限

除了对私有注册表进行身份验证以及推送和提取映像时所需的 Amazon ECR API 权限之外，还需要以下其他权限才能有效使用缓存提取规则。

- `ecr:CreatePullThroughCacheRule` – 授予创建拉取缓存规则的权限。此权限必须通过基于身份的 IAM policy 授予。
- `ecr:BatchImportUpstreamImage` – 授权检索外部镜像并将其导入到您的私有注册表。可以通过使用私有注册表权限策略、基于身份的 IAM policy 或通过使用基于资源的存储库权限策略授予此权限。有关使用存储库权限的更多信息，请参阅 Amazon ECR 中的私有存储库策略。
- `ecr:CreateRepository` – 授予在私有注册表中创建存储库的权限。如果存储缓存图像的存储库不存在，则需要此权限。可以通过基于身份的 IAM policy 或私有注册表权限策略授予此权限。

使用注册表权限

Amazon ECR 私有注册表权限可用于限定各个 IAM 实体使用缓存提取的权限范围。如果 IAM policy 授予 IAM 实体的权限多于注册表权限策略授予的权限，则 IAM policy 优先。例如，如果用户已授予 `ecr:*` 权限，则无需额外的注册表级别权限。

要创建私有注册表的权限策略 (Amazon Web Services Management Console)

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择您在其中配置私有注册表权限语句的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Registry permissions（注册表权限）。
4. 在 Registry permissions（注册表权限）页面上，选择 Generate statement（生成语句）。
5. 对于要创建的每个缓存提取权限策略语句，请执行以下操作。
    a. 对于 Policy type（策略类型），请选择 Pull through cache policy（推送缓存策略）。
    b. 对于 Statement id（语句 ID），为推送缓存语句策略提供名称。
    c. 对于 IAM entities（IAM 实体），指定要包含在策略中的用户、组或角色。
    d. 对于 Repository namespace（存储库命名空间），选择要与策略关联的推送缓存规则。
    e. 对于 Repository names（存储库名称），指定要应用规则的存储库基本名称。例如，如果您想在 Amazon ECR Public 上指定 Amazon Linux 存储库，存储库名称将为 `amazonlinux`。

要创建私有注册表的权限策略 (Amazon CLI)

使用以下 Amazon CLI 命令通过指定私有注册表权限 Amazon CLI。

1. 创建名为 `ptc-registry-policy.json` 的本地文件，其中包含注册表策略的内容。以下示例授予创建存储库并从 Amazon ECR Public 中拉取镜像的 `ecr-pull-through-cache-user` 权限，Amazon ECR Public 是与之前创建的拉取缓存规则相关联的上游源。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PullThroughCacheFromReadOnlyRole",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/ecr-pull-through-cache-user"
      },
      "Action": [
        "ecr:CreateRepository",
        "ecr:BatchImportUpstreamImage"
      ],
      "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/ecr-public/*"
    }
  ]
```

```
}
```

⚠ Important

仅当存储缓存镜像的存储库不存在时才需要 `ecr-CreateRepository` 权限。例如，如果存储库创建操作和镜像拉取操作是由单独的 IAM 主体（例如管理员和开发人员）完成。

2. 使用put-registry-policy命令设置注册表策略。

```
aws ecr put-registry-policy \
    --policy-text file://ptc-registry.policy.json
```

后续步骤

准备好开始使用缓存提取规则后，请执行以下后续步骤。

- 创建缓存提取规则。有关更多信息，请参阅 在 Amazon ECR 中创建缓存提取规则。
- 创建存储库创建模板。有了存储库创建模板，在为 Amazon ECR 在缓存提取操作期间代表您创建的新存储库定义设置时，您能拥有控制权。有关更多信息，请参阅 用于控制在缓存提取或复制操作期间创建的存储库的模板。

设置跨账户 ECR 到 ECR PTC 的权限

Amazon ECR 到 Amazon ECR（ECR 到 ECR）直通缓存功能支持在区域、 Amazon 账户或两者之间自动同步图像。使用 ECR 到 ECR PTC，您可以将图像推送到主要 Amazon ECR 注册表，并配置拉取缓存规则，将图像缓存在下游 Amazon ECR 注册表中。

跨账户 ECR 到 ECR 通过缓存提取所需的 IAM 策略

要在不同 Amazon 账户的 Amazon ECR 注册表之间缓存映像，请在下游账户中创建 IAM 角色并配置本节中的策略以提供以下权限：

- Amazon ECR 需要权限才能代表您从上游 Amazon ECR 注册表中提取图片。您可以通过创建 IAM 角色然后在拉取缓存规则中指定该角色来授予这些权限。
- 上游注册表所有者还必须向缓存注册表所有者授予将映像拉入资源策略所需的权限。

策略

- 创建 IAM 角色来定义直通缓存权限
- 为 IAM 角色创建信任策略
- 在上游 Amazon ECR 注册表中创建资源策略

创建 IAM 角色来定义直通缓存权限

以下示例显示了一个权限策略，该策略授予 IAM 角色代表您从上游 Amazon ECR 注册表中提取图像的权限。当 Amazon ECR 担任该角色时，它将获得本策略中指定的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken",
                "ecr:BatchImportUpstreamImage",
                "ecr:BatchGetImage",
                "ecr:GetImageCopyStatus",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage"
            ],
            "Resource": "*"
        }
    ]
}
```

为 IAM 角色创建信任策略

以下示例显示了一个信任策略，该策略将 Amazon ECR 穿过缓存确定为可以担任该角色的 Amazon 服务主体。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "pullthroughcache.ecr.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

在上游 Amazon ECR 注册表中创建资源策略

上游 Amazon ECR 注册表所有者还必须添加注册表策略或存储库策略，以授予下游注册表所有者执行以下操作所需的权限。

```
{
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::444455556666:root"
    },
    "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchImportUpstreamImage",
        "ecr:GetImageCopyStatus"
    ],
    "Resource": "arn:aws:ecr:region:111122223333:repository/*"
}
```

在 Amazon ECR 中创建缓存提取规则

对于包含您要在 Amazon ECR 私有注册表中缓存的映像的每个上游注册表，您必须创建缓存提取规则。

对于需要使用密钥进行身份验证的上游注册表，您必须将凭据存储在 Secrets Manager 密钥中。您可以使用现有的 密钥或创建一个新密钥。您可以在 Amazon ECR 控制台或 Secrets Manager 控制台中创建 Secrets Manager 密钥。要使用 Secrets Manager 控制台而不是 Amazon ECR 控制台创建 Secrets Manager 密钥，请参阅将您的上游存储库凭证存储在 Amazon Secrets Manager 密钥中。

先决条件

- 请验证您是否拥有创建缓存提取规则的适当 IAM 权限。有关信息，请参阅将上游注册表与 Amazon ECR 私有注册表同步所需的 IAM 权限。
- 对于需要使用密钥进行身份验证的上游注册表：如果要使用现有密钥，请验证 Secrets Manager 密钥是否满足以下要求：
  - 密钥的名称以 `ecr-pullthroughcache/` 开头。 Amazon Web Services Management Console 仅显示带有 `ecr-pullthroughcache/` 前缀的 Secrets Manager 密钥。
  - 密钥所在的账户和区域必须与缓存提取规则所在的账户和区域相匹配。

要创建缓存提取规则 (Amazon Web Services Management Console)

以下步骤演示如何使用 Amazon ECR 控制台创建缓存提取规则和 Secrets Manager 密钥。要使用 Secrets Manager 控制台创建密钥，请参阅将您的上游存储库凭证存储在 Amazon Secrets Manager 密钥中。

对于 Amazon ECR Public、Kubernetes 容器注册表或 Quay

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择要配置私有注册表设置的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Pull through cache（缓存提取）。
4. 在 Pull through cache configuration（缓存提取配置）页面上，选择 Add rule（添加规则）。
5. 在步骤 1：指定来源页面上，对于注册表，请从上游注册表的列表中选择 Amazon ECR Public、Kubernetes 或 Quay，然后选择下一步。
6. 在步骤 2：指定目标页面上，对于 Amazon ECR 存储库前缀，请指定在缓存从源公有注册表中提取的映像时要使用的存储库命名空间前缀，然后选择下一步。默认情况下，已填充命名空间，但也可以指定自定义命名空间。
7. 在步骤 3：审核并创建页面上，审核缓存提取规则配置，然后选择创建。
8. 对要创建的每个缓存提取重复上一步骤。单独为每个区域创建了缓存提取规则。

对于 Docker Hub

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择要配置私有注册表设置的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Pull through cache（缓存提取）。

4. 在 Pull through cache configuration（缓存提取配置）页面上，选择 Add rule（添加规则）。
5. 在步骤 1：指定来源页面上，对于注册表，选择 Docker Hub，然后选择下一步。
6. 在步骤 2：配置身份验证页面上，对于上游凭证，您必须以 Amazon Secrets Manager 密钥存储 Docker Hub 的身份验证凭证。您可以指定现有密钥，也可以使用 Amazon ECR 控制台创建新密钥。

   a. 要使用现有密钥，请选择使用现有 Amazon 密钥。对于密钥名称，使用下拉列表选择现有的密钥，然后选择下一步。

      > **Note**
      >
      > Amazon Web Services Management Console 仅显示名称使用`ecr-pullthroughcache/`前缀的 Secrets Manager 密钥。密钥还必须与缓存提取规则位于相同的账户和区域。

   b. 要创建新密钥，请选择创建 Amazon 密钥，执行以下操作，然后选择下一步。

      i. 在密钥名称中，请为密钥指定一个描述性名称。密钥名称必须包含 1-512 个 Unicode 字符。
      ii. 在 Docker Hub 电子邮件中，指定您的 Docker Hub 电子邮件。
      iii. 在 Docker Hub 访问令牌中，请指定 Docker Hub 访问令牌。有关如何创建 Docker Hub 访问令牌的更多信息，请参阅 Docker 文档中的 Create and manage access tokens。
7. 在步骤 3：指定目标页面上，对于 Amazon ECR 存储库前缀，请指定在缓存从源公有注册表中提取的映像时要使用的存储库命名空间，然后选择下一步。

   默认情况下，已填充命名空间，但也可以指定自定义命名空间。
8. 在步骤 4：审核并创建页面上，审核缓存提取规则配置，然后选择创建。
9. 对要创建的每个缓存提取重复上一步骤。单独为每个区域创建了缓存提取规则。

对于 GitHub 容器注册表

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择要配置私有注册表设置的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Pull through cache（缓存提取）。
4. 在 Pull through cache configuration（缓存提取配置）页面上，选择 Add rule（添加规则）。
5. 在 “步骤 1：指定来源” 页面上，对于 “注册表”，选择 “GitHub 容器注册表”，然后选择 “下一步”。

6. 在 “步骤 2：配置身份验证” 页面上，对于上游凭证，您必须将 GitHub 容器注册表的身份验证凭据存储在 Amazon Secrets Manager 密钥中。您可以指定现有密钥，也可以使用 Amazon ECR 控制台创建新密钥。

    a. 要使用现有密钥，请选择使用现有 Amazon 密钥。对于密钥名称，使用下拉列表选择现有的密钥，然后选择下一步。

        Note

        Amazon Web Services Management Console 仅显示名称使用`ecr-pullthroughcache/`前缀的 Secrets Manager 密钥。密钥还必须与缓存提取规则位于相同的账户和区域。

    b. 要创建新密钥，请选择创建 Amazon 密钥，执行以下操作，然后选择下一步。

        i. 在密钥名称中，请为密钥指定一个描述性名称。密钥名称必须包含 1-512 个 Unicode 字符。

        ii. 对于GitHub 容器注册表用户名，请指定您的 GitHub 容器注册表用户名。

        iii. 对于GitHub 容器注册表访问令牌，请指定您的 GitHub 容器注册表访问令牌。有关创建 GitHub 访问令牌的更多信息，请参阅 GitHub 文档中的管理您的个人访问令牌。

7. 在步骤 3：指定目标页面上，对于 Amazon ECR 存储库前缀，请指定在缓存从源公有注册表中提取的映像时要使用的存储库命名空间，然后选择下一步。

    默认情况下，已填充命名空间，但也可以指定自定义命名空间。

8. 在步骤 4：审核并创建页面上，审核缓存提取规则配置，然后选择创建。

9. 对要创建的每个缓存提取重复上一步骤。单独为每个区域创建了缓存提取规则。

对于 Microsoft Azure 容器注册表

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择要配置私有注册表设置的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Pull through cache（缓存提取）。
4. 在 Pull through cache configuration（缓存提取配置）页面上，选择 Add rule（添加规则）。
5. 在步骤 1：指定来源页面上，执行以下操作。

    a. 在注册表中，请选择Microsoft Azure 容器注册表

   b. 在源注册表 URL 中，请指定 Microsoft Azure 容器注册表的名称，然后选择下一步。

      > ⚠ **Important**
      >
      > 您只需要指定前缀，因为系统已代表您填充 `.azurecr.io` 后缀。

6. 在步骤 2：配置身份验证页面上，对于上游凭证，您必须以 Amazon Secrets Manager 密钥存储 Microsoft Azure 容器注册表的身份验证凭证。您可以指定现有密钥，也可以使用 Amazon ECR 控制台创建新密钥。

   a. 要使用现有密钥，请选择使用现有 Amazon 密钥。对于密钥名称，使用下拉列表选择现有的密钥，然后选择下一步。

      > ⓘ **Note**
      >
      > Amazon Web Services Management Console 仅显示名称使用`ecr-pullthroughcache/`前缀的 Secrets Manager 密钥。密钥还必须与缓存提取规则位于相同的账户和区域。

   b. 要创建新密钥，请选择创建 Amazon 密钥，执行以下操作，然后选择下一步。

      i. 在密钥名称中，请为密钥指定一个描述性名称。密钥名称必须包含 1-512 个 Unicode 字符。

      ii. 在 Microsoft Azure 容器注册表用户名中，请指定 Microsoft Azure 容器注册表用户名。

      iii. 在 Microsoft Azure 容器注册表访问令牌中，请指定 Microsoft Azure 容器注册表访问令牌。有关如何创建 Microsoft Azure 容器注册表访问令牌的更多信息，请参阅 Microsoft Azure 文档中的创建令牌 - 门户。

7. 在步骤 3：指定目标页面上，对于 Amazon ECR 存储库前缀，请指定在缓存从源公有注册表中提取的映像时要使用的存储库命名空间，然后选择下一步。

   默认情况下，已填充命名空间，但也可以指定自定义命名空间。

8. 在步骤 4：审核并创建页面上，审核缓存提取规则配置，然后选择创建。

9. 对要创建的每个缓存提取重复上一步骤。单独为每个区域创建了缓存提取规则。

对于 GitLab 容器注册表

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。

2. 从导航栏中，选择要配置私有注册表设置的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Pull through cache（缓存提取）。
4. 在 Pull through cache configuration（缓存提取配置）页面上，选择 Add rule（添加规则）。
5. 在 “步骤 1：指定来源” 页面上，对于 “注册表”，选择 “ GitLab 容器注册表”，然后选择 “下一步”。
6. 在 “步骤 2：配置身份验证” 页面上，对于上游凭证，您必须将 GitLab 容器注册表的身份验证凭据存储在 Amazon Secrets Manager 密钥中。您可以指定现有密钥，也可以使用 Amazon ECR 控制台创建新密钥。

   a. 要使用现有密钥，请选择使用现有 Amazon 密钥。对于密钥名称，使用下拉列表选择现有的密钥，然后选择下一步。有关使用 Secrets Manager 控制台创建 Secrets Manager 密钥的更多信息，请参阅将您的上游存储库凭证存储在 Amazon Secrets Manager 密钥中。

      > **Note**
      >
      > Amazon Web Services Management Console 仅显示名称使用`ecr-pullthroughcache/`前缀的 Secrets Manager 密钥。密钥还必须与缓存提取规则位于相同的账户和区域。

   b. 要创建新密钥，请选择创建 Amazon 密钥，执行以下操作，然后选择下一步。

      i. 在密钥名称中，请为密钥指定一个描述性名称。密钥名称必须包含 1-512 个 Unicode 字符。
      ii. 对于GitLab 容器注册表用户名，请指定您的 GitLab 容器注册表用户名。
      iii. 对于GitLab 容器注册表访问令牌，请指定您的 GitLab 容器注册表访问令牌。有关创建 GitLab 容器注册表访问令牌的更多信息，请参阅 GitLab 文档中的个人访问令牌、群组访问令牌或项目访问令牌。

7. 在步骤 3：指定目标页面上，对于 Amazon ECR 存储库前缀，请指定在缓存从源公有注册表中提取的映像时要使用的存储库命名空间，然后选择下一步。

   默认情况下，已填充命名空间，但也可以指定自定义命名空间。
8. 在步骤 4：审核并创建页面上，审核缓存提取规则配置，然后选择创建。
9. 对要创建的每个缓存提取重复上一步骤。单独为每个区域创建了缓存提取规则。

对于您的 Amazon 账户中的 Amazon ECR 私有注册表

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。

2. 从导航栏中，选择要在其中配置私有注册表设置的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Pull through cache（缓存提取）。
4. 在 Pull through cache configuration（缓存提取配置）页面上，选择 Add rule（添加规则）。
5. 在 “步骤 1：指定上游” 页面上，在 “注册” 中选择 Amazon ECR Pri vate 和 “此账户”。对于区域，选择上游 Amazon ECR 注册表的区域，然后选择下一步。
6. 在 “步骤 2：指定命名空间” 页面上，对于缓存命名空间，选择是创建带特定前缀还是不带前缀的直通缓存存储库。如果选择特定前缀，则必须指定一个前缀名称，该名称将用作命名空间的一部分，用于缓存来自上游注册表的图像。
7. 对于上游命名空间，选择是否从上游注册表中存在的特定前缀中提取。如果不选择前缀，则可以从上游注册表中的任何存储库中提取数据。如果出现提示，请指定上游存储库前缀，然后选择 “下一步”。

   > Note
   >
   > 要了解有关自定义缓存和上游命名空间的更多信息，请参阅。自定义 ECR 到 ECR 的存储库前缀拉取缓存

8. 在步骤 3：审核并创建页面上，审核缓存提取规则配置，然后选择创建。
9. 对要创建的每个直通缓存重复这些步骤。单独为每个区域创建了缓存提取规则。

对于来自其他 Amazon 账户的 Amazon ECR 私有注册表

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择要配置私有注册表设置的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Pull through cache（缓存提取）。
4. 在 Pull through cache configuration（缓存提取配置）页面上，选择 Add rule（添加规则）。
5. 在 “步骤 1：指定上游” 页面上，在 “注册” 中选择 Amazon ECR 私有和跨账户。对于区域，选择上游 Amazon ECR 注册表的区域。在 “账户” 中，指定上游 Amazon ECR 注册表的 Amazon 账户 ID，然后选择 “下一步”。
6. 在 “步骤 2：指定权限” 页面上，对于 IAM 角色，选择用于跨账户拉取缓存访问权限的角色，然后选择创建。

> Note
>
> 请务必选择使用中创建的权限的 IAM 角色跨账户 ECR 到 ECR 通过缓存提取所需的 IAM 策略。

7. 在 “步骤 3：指定命名空间” 页面上，对于缓存命名空间，选择是创建带特定前缀还是不带前缀的直通缓存存储库。如果选择特定前缀，则必须指定一个前缀名称，该名称将用作命名空间的一部分，用于缓存来自上游注册表的图像。
8. 对于上游命名空间，选择是否从上游注册表中存在的特定前缀中提取。如果不选择前缀，则可以从上游注册表中的任何存储库中提取数据。如果出现提示，请指定上游存储库前缀，然后选择 “下一步”。

> Note
>
> 要了解有关自定义缓存和上游命名空间的更多信息，请参阅。自定义 ECR 到 ECR 的存储库前缀拉取缓存

9. 在步骤 4：审核并创建页面上，审核缓存提取规则配置，然后选择创建。
10. 对要创建的每个直通缓存重复这些步骤。单独为每个区域创建了缓存提取规则。

要创建缓存提取规则 (Amazon CLI)

使用 create-pull-through-cache- Amazon CLI rule 命令为 Amazon ECR 私有注册表创建直通缓存规则。对于需要使用密钥进行身份验证的上游注册表，您必须将凭据存储在 Secrets Manager 密钥中。要使用 Secrets Manager 控制台创建密钥，请参阅将您的上游存储库凭证存储在 Amazon Secrets Manager 密钥中。

针对每个支持的上游注册表提供以下示例。

对于 Amazon ECR Public

以下示例为 Amazon ECR 公有注册表创建一个缓存提取规则。它指定了存储库前缀 `ecr-public`，这导致使用缓存提取规则创建的每个存储库都具有 `ecr-public/`*`upstream-repository-name`* 命名方案。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix ecr-public \
     --upstream-registry-url public.ecr.aws \
```

```
    --region us-east-2
```

适用于 Kubernetes 容器注册表

以下示例为 Kubernetes 公有注册表创建了一个缓存提取规则。它指定了存储库前缀 `kubernetes`，这导致使用缓存提取规则创建的每个存储库都具有 `kubernetes/upstream-repository-name` 命名方案。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix kubernetes \
     --upstream-registry-url registry.k8s.io \
     --region us-east-2
```

对于 Quay

以下示例为 Quay 公有注册表创建了一个缓存提取规则。它指定了存储库前缀 `quay`，这导致使用推送缓存规则创建的每个存储库都具有命名方案 `quay/upstream-repository-name`。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix quay \
     --upstream-registry-url quay.io \
     --region us-east-2
```

对于 Docker Hub

以下示例为 Docker Hub 注册表创建了一个缓存提取规则。它指定了存储库前缀 `docker-hub`，这导致使用缓存提取规则创建的每个存储库都具有 `docker-hub/upstream-repository-name` 命名方案。您必须指定包含 Docker Hub 凭证的密钥的完整 Amazon 资源名称（ARN）。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix docker-hub \
     --upstream-registry-url registry-1.docker.io \
     --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-2
```

对于 GitHub 容器注册表

以下示例为 GitHub 容器注册表创建了通过缓存规则。它指定了存储库前缀 `github`，这导致使用缓存提取规则创建的每个存储库都具有 `github/upstream-repository-name` 命名方案。您必须指定包含您的 GitHub 容器注册凭证的密钥的完整 Amazon 资源名称 (ARN)。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix github \
     --upstream-registry-url ghcr.io \
     --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-2
```

对于 Microsoft Azure 容器注册表

以下示例为 Microsoft Azure 容器注册表创建了一个缓存提取规则。它指定了存储库前缀 `azure`，这导致使用缓存提取规则创建的每个存储库都具有 `azure/upstream-repository-name` 命名方案。您必须指定包含 Microsoft Azure 容器注册表凭证的密钥的完整 Amazon 资源名称（ARN）。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix azure \
     --upstream-registry-url myregistry.azurecr.io \
     --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-2
```

对于 GitLab 容器注册表

以下示例为 GitLab 容器注册表创建了通过缓存规则。它指定了存储库前缀 `gitlab`，这导致使用缓存提取规则创建的每个存储库都具有 `gitlab/upstream-repository-name` 命名方案。您必须指定包含您的 GitLab 容器注册凭证的密钥的完整 Amazon 资源名称 (ARN)。

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix gitlab \
     --upstream-registry-url registry.gitlab.com \
     --credential-arn arn:aws:secretsmanager:us-east-2:111122223333:secret:ecr-
pullthroughcache/example1234 \
     --region us-east-2
```

对于您的 Amazon 账户中的 Amazon ECR 私有注册表

以下示例在同一 Amazon 账户中为跨区域的 Amazon ECR 私有注册表创建直通缓存规则。它指定了存储库前缀 `ecr`，这导致使用缓存提取规则创建的每个存储库都具有 `ecr/upstream-repository-name` 命名方案。

```
aws ecr create-pull-through-cache-rule \
```

```
     --ecr-repository-prefix ecr \
     --upstream-registry-url aws_account_id.dkr.ecr.region.amazonaws.com \
     --region us-east-2
```

对于来自其他 Amazon 账户的 Amazon ECR 私有注册表

以下示例在同一 Amazon 账户中为跨区域的 Amazon ECR 私有注册表创建直通缓存规则。它指定了存储库前缀 `ecr`，这导致使用缓存提取规则创建的每个存储库都具有 `ecr/upstream-repository-name` 命名方案。您必须使用中创建的权限指定 IAM 角色的完整亚马逊资源名称 (ARN)。在 Amazon ECR 中创建缓存提取规则

```
aws ecr create-pull-through-cache-rule \
 --ecr-repository-prefix ecr \
 --upstream-registry-url aws_account_id.dkr.ecr.region.amazonaws.com \
 --custom-role-arn arn:aws:iam::aws_account_id:role/example-role \
 --region us-east-2
```

后续步骤

创建缓存提取规则后，请执行以下后续步骤：

- 创建存储库创建模板。有了存储库创建模板，在为 Amazon ECR 在缓存提取操作期间代表您创建的新存储库定义设置时，您能拥有控制权。有关更多信息，请参阅 用于控制在缓存提取或复制操作期间创建的存储库的模板。
- 验证缓存提取规则。在验证缓存提取规则时，Amazon ECR 会与上游注册表建立网络连接，验证它是否可以访问包含上游注册表凭证的 Secrets Manager 密钥，以及身份验证是否成功。有关更多信息，请参阅 验证 Amazon ECR 中的缓存提取规则。
- 开始使用缓存提取规则。有关更多信息，请参阅 在 Amazon ECR 中使用缓存提取规则来提取映像。

验证 Amazon ECR 中的缓存提取规则

创建缓存提取规则后，对于需要身份验证的上游注册表，您可以验证该规则是否正常工作。在验证缓存提取规则时，Amazon ECR 会与上游注册表建立网络连接，验证它是否可以访问包含上游注册表凭证的 Secrets Manager 密钥，并确认身份验证是否成功。

在开始使用缓存提取规则之前，请确认您拥有适当的 IAM 权限。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步所需的 IAM 权限。

删除缓存提取规则（Amazon Web Services Management Console）

以下步骤演示如何使用 Amazon ECR 控制台验证缓存提取规则。

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 在导航栏中，选择包含要验证的缓存提取规则的区域。
3. 在导航窗格中，选择 Private registry（私有注册表）、Pull through cache（缓存提取）。
4. 在缓存提取配置页面上，选择要验证的缓存提取规则。然后，使用操作下拉菜单并选择查看详细信息。
5. 在缓存提取规则详细信息页面上，使用操作下拉菜单并选择验证身份验证。Amazon ECR 将显示带结果的横幅。
6. 对要验证的每个缓存提取规则重复上述步骤。

删除缓存提取规则（Amazon CLI）

validate-pull-through-cache- Amazon CLI rule 命令用于验证 Amazon ECR 私有注册表的直通缓存规则。以下示例使用 `ecr-public` 命名空间前缀。将该值替换为要验证的缓存提取规则的前缀值。

```
aws ecr validate-pull-through-cache-rule \
     --ecr-repository-prefix ecr-public \
     --region us-east-2
```

在响应中，`isValid` 参数指示验证是否成功。如果为 `true`，则表示 Amazon ECR 能够访问上游注册表，并且身份验证成功。如果为 `false`，则表示存在问题且验证失败。`failure` 参数指示原因。

在 Amazon ECR 中使用缓存提取规则来提取映像

以下示例演示在使用缓存提取规则来提取映像时要使用的命令语法。如果您在使用拉取缓存规则拉取上游镜像时收到错误，请参阅 排查 Amazon ECR 中的缓存提取问题，以查看最常见的错误以及如何解决这些错误。

在开始使用缓存提取规则之前，请确认您拥有适当的 IAM 权限。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步所需的 IAM 权限。

> **Note**
>
> 以下示例使用使用的默认 Amazon ECR 存储库命名空间值。 Amazon Web Services Management Console 确保您使用已配置的 Amazon ECR 私有存储库 URI。

对于 Amazon ECR Public

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/ecr-public/repository_name/
image_name:tag
```

Kubernetes 容器注册表

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/kubernetes/repository_name/
image_name:tag
```

Quay

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/quay/repository_name/
image_name:tag
```

Docker Hub

对于 Docker Hub 官方映像：

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/docker-hub/
library/image_name:tag
```

> **Note**
>
> 对于 Docker Hub 官方映像，必须包含 `/library` 前缀。对于所有其他 Docker Hub 存储库，应省略 `/library` 前缀。

对于所有其他 Docker Hub 映像：

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/docker-hub/repository_name/
image_name:tag
```

GitHub 容器注册表

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/github/repository_name/
image_name:tag
```

Microsoft Azure 容器注册表

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/azure/repository_name/
image_name:tag
```

GitLab 容器注册表

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/gitlab/repository_name/
image_name:tag
```

将您的上游存储库凭证存储在 Amazon Secrets Manager 密钥中

为需要身份验证的上游存储库创建缓存提取规则时，您必须以 Secrets Manager 密钥存储凭证。使用 Secrets Manager 密钥可能会产生费用。有关更多信息，请参阅Amazon Secrets Manager 定价。

以下过程引导您了解如何为每个支持的上游存储库创建 Secrets Manager 密钥。您可以选择在 Amazon ECR 控制台中使用“创建缓存提取规则”工作流程来创建密钥，而不是使用 Secrets Manager 控制台来创建密钥。有关更多信息，请参阅 在 Amazon ECR 中创建缓存提取规则。

Docker Hub

为 Docker Hub 凭证创建 Secrets Manager 密钥（Amazon Web Services Management Console）

1. 打开 Secrets Manager 控制台，网址为https://console.aws.amazon.com/secretsmanager/。
2. 选择存储新密钥。
3. 在选择密钥类型页面上，执行以下操作：
    a. 对于密钥类型，请选择其他密钥类型。
    b. 在键/值对中，为 Docker Hub 凭证创建两行键/值对。您可以在密钥中存储最多 65536 个字节。
        i. 对于第一个键/值对，指定 `username` 为键，指定 Docker Hub 用户名为值。

      ii. 对于第二个键/值对，指定 `accessToken` 为键，指定 Docker Hub 访问令牌为值。有关如何创建 Docker Hub 访问令牌的更多信息，请参阅 Docker 文档中的 Create and manage access tokens。

   c. 对于加密密钥，保留默认的 aws/secretsmanager Amazon KMS key 值，然后选择下一步。使用此密钥不产生任何费用。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的 Secrets Manager 中的密钥加密和解密。

> ⚠ Important
>
> 您必须使用默认 `aws/secretsmanager` 加密密钥来加密密钥。Amazon ECR 不支持为此密钥使用客户自主管理型密钥（CMK）。

4. 在配置密钥页面上，执行以下操作：

   a. 输入一个描述性的 Secret name（密钥名称）和 Description（说明）。密钥名称必须包含 1-512 个 Unicode 字符，并且以 `ecr-pullthroughcache/` 为前缀。

> ⚠ Important
>
> Amazon ECR Amazon Web Services Management Console 仅显示名称使用`ecr-pullthroughcache/`前缀的 Secrets Manager 机密。

   b. （可选）在标签部分中，在您的密钥中添加一个或多个标签。有关标记策略，请参阅《Amazon Secrets Manager 用户指南》中的标记 Secrets Manager 密钥。请不要将敏感信息存储在标签中，因为它们未加密。

   c. （可选）在资源权限，要将资源策略添加到您的密钥中，请选择编辑权限。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的将权限策略附加到 Secrets Manager 密钥。

   d. （可选）在复制密钥中，要将您的密钥复制到另一个密钥 Amazon Web Services 区域，请选择复制密钥。您可以现在复制密钥，也可以回头再复制。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的将密钥复制到其他区域。

   e. 选择下一步。

5. （可选）在 Configure rotation（配置轮换）页面上，您可以启用自动轮换。您也可以现在保持关闭轮换，然后稍后将其打开。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的轮换 Secrets Manager 密钥。选择下一步。

6. 在 Review (审核) 页上，审核您的密钥详细信息，然后选择 Store (存储)。

Secrets Manager 将返回到密钥列表。如果您的新密钥未显示，请选择 Refresh（刷新）按钮。

GitHub Container Registry

为你的 GitHub 容器注册表凭证创建 Secrets Manager 密钥 (Amazon Web Services Management Console)

1. 打开 Secrets Manager 控制台，网址为https://console.aws.amazon.com/secretsmanager/。
2. 选择存储新密钥。
3. 在选择密钥类型页面上，执行以下操作：
    a. 对于密钥类型，请选择其他密钥类型。
    b. 在键/值对中，为您的 GitHub 凭据创建两行。您可以在密钥中存储最多 65536 个字节。
        i. 对于第一个键/值对，指定`username`为密钥，将您的 GitHub用户名指定为值。
        ii. 对于第二个密钥/值对，指定`accessToken`为密钥，将 GitHub 访问令牌指定为值。有关创建 GitHub 访问令牌的更多信息，请参阅 GitHub 文档中的管理您的个人访问令牌。
    c. 对于加密密钥，保留默认的 aws/secretsmanager Amazon KMS key 值，然后选择下一步。使用此密钥不产生任何费用。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的 Secrets Manager 中的密钥加密和解密。

> ⚠ Important
>
> 您必须使用默认 `aws/secretsmanager` 加密密钥来加密密钥。Amazon ECR 不支持为此密钥使用客户自主管理型密钥（CMK）。

4. 在 Configure secret（配置密钥）页面上，执行以下操作：
    a. 输入一个描述性的 Secret name（密钥名称）和 Description（说明）。密钥名称必须包含 1-512 个 Unicode 字符，并且以 `ecr-pullthroughcache/` 为前缀。

> ⚠ Important
>
> Amazon ECR Amazon Web Services Management Console 仅显示名称使用`ecr-pullthroughcache/`前缀的 Secrets Manager 机密。

   b. （可选）在标签部分中，在您的密钥中添加一个或多个标签。有关标记策略，请参阅《Amazon Secrets Manager 用户指南》中的标记 Secrets Manager 密钥。请不要将敏感信息存储在标签中，因为它们未加密。
   c. （可选）在资源权限，要将资源策略添加到您的密钥中，请选择编辑权限。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的将权限策略附加到 Secrets Manager 密钥。
   d. （可选）在复制密钥中，要将您的密钥复制到另一个密钥 Amazon Web Services 区域，请选择复制密钥。您可以现在复制密钥，也可以回头再复制。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的将密钥复制到其他区域。
   e. 选择下一步。

5. （可选）在 Configure rotation（配置轮换）页面上，您可以启用自动轮换。您也可以现在保持关闭轮换，然后稍后将其打开。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的轮换 Secrets Manager 密钥。选择下一步。
6. 在 Review (审核) 页上，审核您的密钥详细信息，然后选择 Store (存储)。

   Secrets Manager 将返回到密钥列表。如果您的新密钥未显示，请选择 Refresh（刷新）按钮。

Microsoft Azure Container Registry

为 Microsoft Azure 容器注册表凭证创建 Secrets Manager 密钥（Amazon Web Services Management Console）

1. 打开 Secrets Manager 控制台，网址为https://console.aws.amazon.com/secretsmanager/。
2. 选择存储新密钥。
3. 在选择密钥类型页面上，执行以下操作：
   a. 对于密钥类型，请选择其他密钥类型。
   b. 在键/值对中，为 Microsoft Azure 凭证创建两行键/值对。您可以在密钥中存储最多 65536 个字节。

    i. 对于第一个键/值对，指定 `username` 为键，指定 Microsoft Azure 容器注册表用户名为值。
    ii. 对于第二个键/值对，指定 `accessToken` 为键，Microsoft Azure 容器注册表访问令牌为值。有关如何创建 Microsoft Azure 访问令牌的更多信息，请参阅 Microsoft Azure 文档中的创建令牌 - 门户。

  c. 对于加密密钥，保留默认的 aws/secretsmanager Amazon KMS key 值，然后选择下一步。使用此密钥不产生任何费用。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的 Secrets Manager 中的密钥加密和解密。

> ⚠ Important
>
> 您必须使用默认 `aws/secretsmanager` 加密密钥来加密密钥。Amazon ECR 不支持为此密钥使用客户自主管理型密钥（CMK）。

4. 在 Configure secret（配置密钥）页面上，执行以下操作：

  a. 输入一个描述性的 Secret name（密钥名称）和 Description（说明）。密钥名称必须包含 1-512 个 Unicode 字符，并且以 `ecr-pullthroughcache/` 为前缀。

> ⚠ Important
>
> Amazon ECR Amazon Web Services Management Console 仅显示名称使用`ecr-pullthroughcache/`前缀的 Secrets Manager 机密。

  b. （可选）在标签部分中，在您的密钥中添加一个或多个标签。有关标记策略，请参阅《Amazon Secrets Manager 用户指南》中的标记 Secrets Manager 密钥。请不要将敏感信息存储在标签中，因为它们未加密。

  c. （可选）在资源权限，要将资源策略添加到您的密钥中，请选择编辑权限。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的将权限策略附加到 Secrets Manager 密钥。

  d. （可选）在复制密钥中，要将您的密钥复制到另一个密钥 Amazon Web Services 区域，请选择复制密钥。您可以现在复制密钥，也可以回头再复制。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的将密钥复制到其他区域。

  e. 选择下一步。

5. （可选）在 Configure rotation（配置轮换）页面上，您可以启用自动轮换。您也可以现在保持关闭轮换，然后稍后将其打开。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的轮换 Secrets Manager 密钥。选择下一步。
6. 在 Review (审核) 页上，审核您的密钥详细信息，然后选择 Store (存储)。

   Secrets Manager 将返回到密钥列表。如果您的新密钥未显示，请选择 Refresh（刷新）按钮。

GitLab Container Registry

为你的 GitLab 容器注册表凭证创建 Secrets Manager 密钥 (Amazon Web Services Management Console)

1. 打开 Secrets Manager 控制台，网址为https://console.aws.amazon.com/secretsmanager/。
2. 选择存储新密钥。
3. 在选择密钥类型页面上，执行以下操作：
   a. 对于密钥类型，请选择其他密钥类型。
   b. 在键/值对中，为您的 GitLab 凭据创建两行。您可以在密钥中存储最多 65536 个字节。
      i. 对于第一个键/值对，指定`username`为密钥，将您的 GitLab 容器注册表用户名指定为值。
      ii. 对于第二个密钥/值对，指定`accessToken`为密钥，将您的 GitLab 容器注册表访问令牌指定为值。有关创建 GitLab 容器注册表访问令牌的更多信息，请参阅 GitLab 文档中的个人访问令牌、群组访问令牌或项目访问令牌。
   c. 对于加密密钥，保留默认的 aws/secretsmanager Amazon KMS key 值，然后选择下一步。使用此密钥不产生任何费用。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的 Secrets Manager 中的密钥加密和解密。

      Important

      您必须使用默认 `aws/secretsmanager` 加密密钥来加密密钥。Amazon ECR 不支持为此密钥使用客户自主管理型密钥（CMK）。
4. 在 Configure secret（配置密钥）页面上，执行以下操作：

   a. 输入一个描述性的 Secret name（密钥名称）和 Description（说明）。密钥名称必须包含 1-512 个 Unicode 字符，并且以 `ecr-pullthroughcache/` 为前缀。

      > ⚠ Important
      >
      > Amazon ECR Amazon Web Services Management Console 仅显示名称使用`ecr-pullthroughcache/`前缀的 Secrets Manager 机密。

   b. （可选）在标签部分中，在您的密钥中添加一个或多个标签。有关标记策略，请参阅《Amazon Secrets Manager 用户指南》中的标记 Secrets Manager 密钥。请不要将敏感信息存储在标签中，因为它们未加密。

   c. （可选）在资源权限，要将资源策略添加到您的密钥中，请选择编辑权限。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的将权限策略附加到 Secrets Manager 密钥。

   d. （可选）在复制密钥中，要将您的密钥复制到另一个密钥 Amazon Web Services 区域，请选择复制密钥。您可以现在复制密钥，也可以回头再复制。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的将密钥复制到其他区域。

   e. 选择下一步。

5. （可选）在 Configure rotation（配置轮换）页面上，您可以启用自动轮换。您也可以现在保持关闭轮换，然后稍后将其打开。有关更多信息，请参阅《Amazon Secrets Manager 用户指南》中的轮换 Secrets Manager 密钥。选择下一步。

6. 在 Review (审核) 页上，审核您的密钥详细信息，然后选择 Store (存储)。

   Secrets Manager 将返回到密钥列表。如果您的新密钥未显示，请选择 Refresh（刷新）按钮。

自定义 ECR 到 ECR 的存储库前缀拉取缓存

直通缓存规则同时支持 ecr 存储库前缀和上游存储库前缀。ecr 存储库前缀是 Amazon ECR 缓存注册表中与规则关联的存储库命名空间前缀。所有使用此前缀的存储库都将成为规则中定义的上游注册表的支持缓存的存储库。例如，前缀`prod`适用于以开头的所有存储库`prod/`。要将模板应用于注册表中所有没有关联的直通缓存规则的仓库，请使用`ROOT`作为前缀。

⚠ Important

前缀末尾始终应用假定的 `/`。如果您指定 `ecr-public` 为前缀，Amazon ECR 会将其视为 `ecr-public/`。

上游存储库前缀与上游存储库名称匹配。默认情况下，它设置为`ROOT`，允许与任何上游存储库进行匹配。只有当 Amazon ECR 存储库前缀为非**`ROOT`**值时，您才能设置上游存储库前缀。

下表根据拉取缓存规则中的前缀配置，显示了缓存存储库名称和上游存储库名称之间的映射。

缓存命名空间	上游命名空间	映射关系（缓存存储库 → 上游存储库）
ecr-public	根（默认）	`ecr-public/my-app/image1` → `my-app/image1` `ecr-public/my-app/image2` → `my-app/image2`
根	根	`my-app/image1` → `my-app/image1`
team-a	team-a	`team-a/myapp/image1` → `team-a/myapp/image1`
我的应用程序	上游应用程序	`my-app/image1` → `upstream-app/image1`

排查 Amazon ECR 中的缓存提取问题

使用拉取缓存规则拉取上游镜像时，您可能会收到以下常见错误。

存储库不存在

指示存储库不存在的错误通常是由该存储库不存在于 Amazon ECR 私有注册表中或者未向拉取上游镜像的 IAM 主体授予 `ecr:CreateRepository` 权限所致。要解决此错误，您应该验证拉取命令中的存储库 URI 正确，已向拉取上游镜像的 IAM 主体授予所需的 IAM 权限，或者在执行上游镜像拉取之前，已在您的 Amazon ECR 私有注册表中创建要向其推送上游镜像的存储库。有关所需 IAM 权限的更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步所需的 IAM 权限。

以下是此错误的示例。

```
Error response from daemon: repository 111122223333.dkr.ecr.us-east-1.amazonaws.com/
ecr-public/amazonlinux/amazonlinux not found: name unknown: The repository with
 name 'ecr-public/amazonlinux/amazonlinux' does not exist in the registry with id
 '111122223333'
```

找不到请求的映像

指示找不到镜像的错误通常是由该镜像不存在于上游注册表中或者未向拉取上游镜像的 IAM 主体授予 `ecr:BatchImportUpstreamImage` 权限但已在 Amazon ECR 私有注册表中创建存储库所致。要解决此错误，您应验证上游镜像和镜像标签名称正确且存在，并且已向拉取上游镜像的 IAM 主体授予所需的 IAM 权限。有关所需 IAM 权限的更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步所需的 IAM 权限。

以下是此错误的示例。

```
Error response from daemon: manifest for 111122223333.dkr.ecr.us-
east-1.amazonaws.com/ecr-public/amazonlinux/amazonlinux:latest not found: manifest
 unknown: Requested image not found
```

从 Docker Hub 存储库中提取时发生 403 Forbidden 错误

从标记为 Docker 官方映像的 Docker Hub 存储库中提取时，您必须在使用的 URI 中包含 `/library/`。例如 *aws_account_id*.dkr.ecr.*region*.amazonaws.com/docker-hub**/library/***image_name:tag*。如果您省略 Docker Hub 官方镜像，则当您尝试使用拉取缓存规则提取镜像时，将返回`403 Forbidden`错误。`/library/`有关更多信息，请参阅 在 Amazon ECR 中使用缓存提取规则来提取映像。

以下是此错误的示例。

```
Error response from daemon: failed to resolve reference "111122223333.dkr.ecr.us-
west-2.amazonaws.com/docker-hub/amazonlinux:2023": pulling from host
```

```
111122223333.dkr.ecr.us-west-2.amazonaws.com failed with status code [manifests 
2023]: 403 Forbidden
```

Amazon ECR 中的私有映像复制

您可以配置 Amazon ECR 私有注册表以支持存储库的复制。Amazon ECR 同时适用于跨区域和跨账户复制。要进行跨账户复制，目标账户必须配置注册表权限策略，以允许从源注册表进行复制。有关更多信息，请参阅 Amazon ECR 中的私有注册表权限。

主题

私有镜像复制的注意事项

使用私有镜像复制时应注意以下事项。

- 只有在配置复制之后，推送到存储库的内容才会被复制。存储库中任何先前存在的内容都不会复制。为存储库配置复制后，Amazon ECR 将保持目标和源同步。
- 复制完成后，跨区域和账户的存储库名称将保持不变。Amazon ECR 不支持在复制过程中更改存储库名称。
- 首次配置私有注册表以进行复制时，Amazon ECR 会代表您创建服务相关 IAM 角色。服务相关 IAM 角色授予 Amazon ECR 复制服务在注册表中创建存储库和复制镜像所需的权限。有关更多信息，请参阅 对 Amazon ECR 使用服务相关角色。
- 要进行跨账户复制，私有注册表目标必须授予允许源注册表复制其镜像的权限。通过设置私有注册表权限策略来完成此授权。有关更多信息，请参阅 Amazon ECR 中的私有注册表权限。
- 如果更改私有注册表的权限策略以删除权限，则以前授予权限的任何进行中复制都可能完成。
- 要进行跨区域复制，在该区域内或向该区域进行任何复制操作之前，源账户和目标账户都必须选择加入该区域。有关更多信息，请参阅《Amazon Web Services 一般参考》中的管理 Amazon 区域。
- 不支持在 Amazon 分区之间进行跨区域复制。例如，`us-west-2` 中的存储库无法复制到 `cn-north-1`。有关 Amazon 分区的更多信息，请参阅《Amazon 一般参考》中的 ARN 格式。
- 私有注册表的复制配置最多可以包含 25 个跨所有规则的唯一目标，最多共有 10 个规则。每个规则最多可包含 100 个筛选条件。这允许为包含用于生产和测试的镜像的存储库指定单独的规则。
- 复制配置支持通过指定存储库前缀来筛选私有注册表中复制的存储库。有关示例，请参阅 示例：使用存储库筛选条件配置跨区域复制。

- 每次镜像推送只会执行一次复制操作。例如，如果您配置了从 `us-west-2` 到 `us-east-1` 以及从 `us-east-1` 到 `us-east-2` 的跨区域复制，则推送到 `us-west-2` 的镜像仅复制到 `us-east-1`，它不会再复制到 `us-east-2`。此行为同时适用于跨区域和跨账户复制。
- 大多数映像会在不到 30 分钟的时间内复制，但在极少数情况下，复制可能需要更长的时间。
- 注册表复制不执行任何删除操作。复制镜像和存储库不再使用时，可以手动删除它们。
- 存储库策略 (包括 IAM policy) 和生命周期策略不会被复制，而且除了对为其定义的存储库之外，不会产生任何影响。
- 默认情况下不会复制存储库设置，您可以使用存储库创建模板来复制存储库设置。这些设置包括标签可变性、加密、存储库权限和生命周期策略。有关仓库创建模板的更多信息，请参阅用于控制在缓存提取或复制操作期间创建的存储库的模板。
- 如果在存储库上启用了标签不变性，并且复制了与现有镜像使用相同标签的镜像，则该镜像将被复制，但不包含重复的标签。这可能会形成未标记的镜像。

Amazon ECR 的私有映像复制示例

以下各示例演示了私有映像复制的常见应用场景。如果您使用配置复制 Amazon CLI，则可以在创建 JSON 文件时使用 JSON 示例作为起点。如果您使用配置复制 Amazon Web Services Management Console，则在查看并提交页面上查看复制规则时，您将看到类似的 JSON。

示例：配置跨区域复制到单个目标区域

下面显示了在单个注册表中配置跨区域复制的示例。此示例假定您的账户 ID 为 `111122223333`，并且您正在区域 (而不是 `us-west-2`) 中指定此复制配置。

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-2",
                    "registryId": "111122223333"
                }
            ]
        }
    ]
}
```

示例：使用存储库筛选条件配置跨区域复制

下面显示了为与前缀名称值匹配的存储库配置跨区域复制的示例。此示例假定您的账户 ID 为`111122223333`，您正在区域 (而不是 `us-west-1`) 中指定此复制配置，并且具有前缀为 `prod` 的存储库。

```
{
 "rules": [{
  "destinations": [{
   "region": "us-west-1",
   "registryId": "111122223333"
  }],
  "repositoryFilters": [{
   "filter": "prod",
   "filterType": "PREFIX_MATCH"
  }]
 }]
}
```

示例：配置跨区域复制到多个目标区域

下面显示了在单个注册表中配置跨区域复制的示例。此示例假定您的账户 ID 为 `111122223333`，并且您正在区域 (而不是 `us-west-1` 或 `us-west-2`) 中指定此复制配置。

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-1",
                    "registryId": "111122223333"
                },
                {
                    "region": "us-west-2",
                    "registryId": "111122223333"
                }
            ]
        }
    ]
}
```

示例：配置跨账户复制

下面显示了为注册表配置跨账户复制的示例。此示例将配置复制到 `444455556666` 账户和 `us-west-2` 区域。

Important

要进行跨账户复制，目标账户必须配置注册表权限策略，以允许进行复制。有关更多信息，请参阅 Amazon ECR 中的私有注册表权限。

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-2",
                    "registryId": "444455556666"
                }
            ]
        }
    ]
}
```

示例：在配置中指定多个规则

以下显示了配置注册表的多个复制规则的示例。此示例配置 *111122223333* 账户的复制，其具备一个规则，即将前缀为 `prod` 的存储库复制到 `us-west-2` 区域，并将前缀为 `test` 的存储库复制到 `us-east-2` 区域。复制配置最多可以包含 10 个规则，每个规则最多指定 25 个目标。

```
{
 "rules": [{
   "destinations": [{
    "region": "us-west-2",
    "registryId": "111122223333"
   }],
   "repositoryFilters": [{
    "filter": "prod",
    "filterType": "PREFIX_MATCH"
   }]
```

```
    },
    {
     "destinations": [{
      "region": "us-east-2",
      "registryId": "111122223333"
     }],
     "repositoryFilters": [{
      "filter": "test",
      "filterType": "PREFIX_MATCH"
     }]
    }
   ]
  }
```

在 Amazon ECR 中配置私有映像复制

为您的私有注册表配置每个区域的复制。您可以配置跨区域复制或跨账户复制。

有关常用复制的示例，请参阅 Amazon ECR 的私有映像复制示例。

配置注册表复制设置 (Amazon Web Services Management Console)

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。
2. 从导航栏中，选择要配置注册表复制设置的区域。
3. 在导航窗格中，选择私有注册表。
4. 在私有注册表页面上，选择设置，然后选择复制配置下的编辑。
5. 在复制页面上，选择添加复制规则。
6. 在目标类型页面上，选择是启用跨区域复制、跨账户复制还是两者，然后选择下一步。
7. 如果启用了跨区域复制，则在配置目标区域中，选择一个或多个目标区域，然后选择下一步。
8. 如果启用了跨账户复制，则在跨账户复制中，选择注册表的跨账户复制设置。对于目标帐户，输入目标账户的账户 ID 以及复制到其中的一个或多个目标区域。选择目标账户 + 以将其他账户配置为复制目标。

> **Important**
>
> 要进行跨账户复制，目标账户必须配置注册表权限策略，以允许执行复制。有关更多信息，请参阅 Amazon ECR 中的私有注册表权限。

9. (可选) 在添加筛选条件页面上，为复制规则指定一个或多个筛选条件，然后选择添加。对要与复制操作相关联的每个筛选条件重复此步骤。必须将筛选条件指定为存储库名称前缀。如果未添加筛选条件，则复制所有存储库的内容。添加所有筛选条件后，选择下一步。
10. 在存储库的审核和提交页面上，查看复制规则配置，然后选择提交规则。

配置注册表复制设置 (Amazon CLI)

1. 创建包含要为注册表定义的复制规则的 JSON 文件。复制配置最多可以包含 10 个规则，所有规则最多包含 25 个唯一目标，每个规则最多包含 100 个筛选条件。要在自己的账户中配置跨区域复制，请指定自己的账户 ID。有关更多示例，请参阅Amazon ECR 的私有映像复制示例。

```
{
 "rules": [{
  "destinations": [{
   "region": "destination_region",
   "registryId": "destination_accountId"
  }],
  "repositoryFilters": [{
   "filter": "repository_prefix_name",
   "filterType": "PREFIX_MATCH"
  }]
 }]
}
```

2. 创建注册表的复制配置。

```
aws ecr put-replication-configuration \
     --replication-configuration file://replication-settings.json \
     --region us-west-2
```

3. 确认您的注册表设置。

```
aws ecr describe-registry \
     --region us-west-2
```

用于控制在缓存提取或复制操作期间创建的存储库的模板

使用 Amazon ECR 存储库创建模板，定义 Amazon ECR 代表您创建的存储库的设置。存储库创建模板中的设置仅在存储库创建期间应用，对现有存储库或使用任何其他方法创建的存储库没有任何影响。目前，存储库创建模板可用于在存储库创建过程中为以下功能应用设置：

- 缓存提取
- 复制

以下区域不支持存储库创建模板：

- 中国（北京）(`cn-north-1`)
- 中国（宁夏）(`cn-northwest-1`)
- Amazon GovCloud （美国东部）(`us-gov-east-1`)
- Amazon GovCloud （美国西部）(`us-gov-west-1`)

存储库创建模板的工作原理

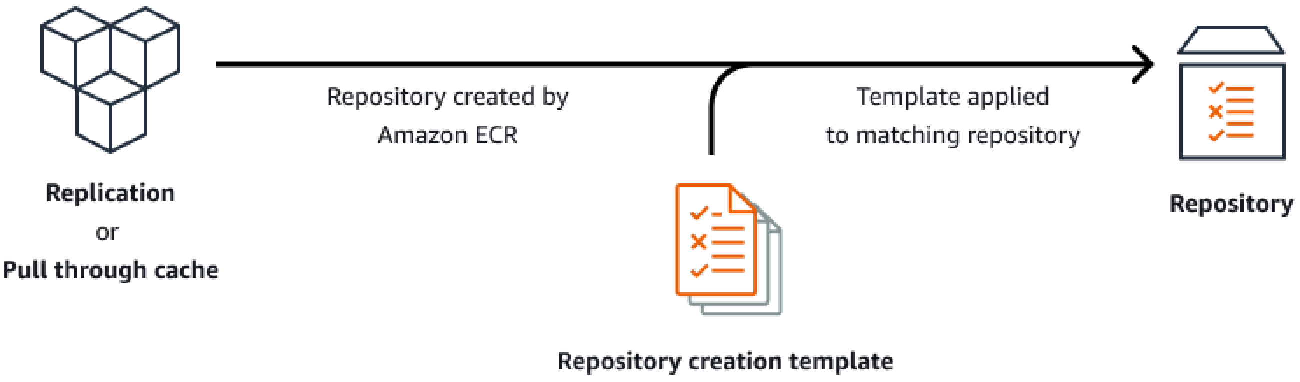
有时，Amazon ECR 需要代表您创建新的私有存储库。例如：

- 您首次使用缓存提取规则来检索上游存储库的内容并将其存储于 Amazon ECR 私有注册表时。
- 当您想要 Amazon ECR 将存储库复制到另一个区域或账户时。

当没有与您的缓存提取规则或复制的存储库匹配的存储库创建模板时，Amazon ECR 会为新存储库使用默认设置。这些默认设置包括关闭标签不变性、使用 `AES-256` 加密，以及不应用任何存储库或生命周期策略。

使用存储库创建模板，您可以定义 Amazon ECR 为通过缓存提取和复制操作创建的新存储库应用的设置。您可以为新存储库定义标签不变性、加密配置、存储库权限、生命周期策略和资源标签。

下图演示了当存储库创建模板与缓存提取操作结合使用时 Amazon ECR 使用的工作流。



以下内容详细描述了存储库创建模板中的每个参数。

Prefix

Prefix 是与模板关联的存储库命名空间前缀。使用此前缀创建的所有存储库都将应用此模板中定义的设置。例如，前缀 `prod` 将应用于以 `prod/` 开头的所有存储库。同理，前缀 `prod/team` 将应用于以 `prod/team/` 开头的所有存储库。在包含两个模板的注册表中，如果一个模板的前缀为 “prod”，另一个模板的前缀为 “prod/team", the template with the prefix "prod/team" will be applied to all repositories whose names start with "prod/team/”。

要将某模板应用于注册表中没有关联创建模板的所有存储库，您可以使用 `ROOT` 作为前缀。

Important

前缀末尾始终应用假定的 `/`。如果您指定 `ecr-public` 为前缀，Amazon ECR 会将其视为 `ecr-public/`。使用缓存提取规则时，您在创建规则时指定的存储库前缀也应指定为存储库创建模板前缀。

描述

此模板描述可选，用于描述存储库创建模板的用途。

应用对象

应用的设置决定了使用此模板创建的 Amazon ECR 创建的存储库。有效值为 `PULL_THROUGH_CACHE` 和 `REPLICATION`。例如，您首次使用缓存提取规则来检索上游存储库的内容并将其存储于 Amazon ECR 私有注册表时。当没有与您的缓存提取规则匹配的存储库创建模板时，Amazon ECR 会为新存储库使用默认设置。

存储库创建角色

存储库创建角色是一个 IAM 角色 ARN，在通过存储库创建模板创建和配置存储库时，该角色将由 Amazon ECR 担任。在模板中使用存储库标签和/或 KMS 时，必须提供此角色，否则存储库创建将失败。

镜像标签可变性

使用模板创建的存储库要使用的标签可变性设置。如果省略此参数，将使用 MUTABLE 的默认设置，该设置将允许覆盖映像标签。对于通过缓存提取操作创建的存储库所使用的模板，建议使用此设置。这样可以确保当标签相同时，Amazon ECR 可以更新缓存的映像。

如果指定 IMMUTABLE，则存储库中的所有映像标签都将不可变，从而防止这些标签被覆盖。

加密配置

> ⚠ Important
>
> 双层服务器端加密，使用（Amazon KMS DSSE-KMS) 仅在 Amazon GovCloud (US) 地区可用。

使用模板创建的存储库要使用的加密配置。

如果您使用 KMS 加密类型，则使用具有 Amazon KMS中存储的 Amazon Key Management Service 密钥的服务器端加密来加密存储库的内容。当您使用 Amazon KMS 加密数据时，可以使用 Amazon ECR 的默认 Amazon 托管 Amazon KMS 密钥，也可以指定您自己的 Amazon KMS 密钥，该密钥是您已经创建的。您还可以选择使用单层或双层加密。 Amazon KMS有关更多信息，请参阅 静态加密。如果您使用的是 KMS 加密类型并将其用于跨区域复制，则可能还需要额外的权限。有关更多信息，请参阅 Creating a KMS key policy for replication。

如果您使用 AES256 加密类型，Amazon ECR 将使用具有 Amazon S3 托管加密密钥的服务器端加密，从而使用 AES-256 加密算法对存储库中的映像进行加密。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用采用 Amazon S3 托管加密密钥的服务器端加密 (SSE-S3) 保护数据。

存储库权限

使用模板创建的存储库要应用的存储库策略。存储库策略使用基于资源的权限来控制对存储库的访问权限。基于资源的权限让您可以指定能够访问存储库的 IAM 用户或角色，以及这些用户或角色可以对该存储库执行的操作。默认情况下，只有创建仓库的 Amazon 账户才有权访问仓库。您可以应

用策略文档来授予或拒绝针对您的存储库的其他权限。有关更多信息，请参阅 Amazon ECR 中的私有存储库策略。

存储库生命周期策略

使用模板创建的存储库要使用的生命周期策略。生命周期策略提供了对私有存储库中映像的生命周期管理的更多控制。生命周期策略是一组规则，其中的每个规则为 Amazon ECR 定义一个操作。这提供了一种自动清理容器器镜像的方法，例如根据使用期限或计数过期的镜像。有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像。

资源标签

资源标签是应用于存储库的元数据，旨在帮助您对其进行分类和整理。每个标签都包含定义的一个键和一个可选值。如果您使用的是具有跨区域复制功能的存储库创建模板，则需要将此权限应用于目标注册表策略。

在 Amazon ECR 中创建存储库创建模板

您可以创建存储库创建模板，定义 Amazon ECR 在缓存提取或复制操作期间代表您创建的存储库要使用的设置。创建存储库创建模板后，所有新建的存储库都将应用这些设置。这不会对之前创建的存储库产生任何影响。

在使用模板设置存储库时，您可以选择指定 KMS 密钥和资源标签。如果您打算同时在一个或多个模板中使用 KMS 密钥、资源标签或两者的组合，则需要：

- 为存储库创建模板创建自定义策略.
- 为存储库创建模板创建 IAM 角色.

配置完成后，您可以将自定义角色附加到注册表中的特定模板。

创建存储库创建模板的 IAM 权限

IAM 主体需要具有以下权限，才能管理存储库创建模板。此这些权限必须通过基于身份的 IAM policy 授予。

- `ecr:CreateRepositoryCreationTemplate` – 授权创建存储库创建模板。
- `ecr:UpdateRepositoryCreationTemplate`：授予更新存储库创建模板的权限。
- `ecr:DescribeRepositoryCreationTemplates`：授予在注册表中列出存储库创建模板的权限。

- `ecr:DeleteRepositoryCreationTemplate` – 授权删除存储库创建模板。
- `ecr:CreateRepository`：授予创建 Amazon ECR 存储库的权限。
- `ecr:PutLifecyclePolicy` – 授权创建生命周期策略并将其应用于存储库。仅当存储库创建模板包含生命周期策略时，才需要此权限。
- `ecr:SetRepositoryPolicy` – 授权创建存储库的权限策略。仅当存储库创建模板包含存储库策略时，才需要此权限。
- `iam:PassRole`：授予允许实体将角色传递给服务或应用程序的权限。此权限对于需要代入角色以代表您执行操作的服务和应用程序是必需的。

为存储库创建模板创建自定义策略

您可以使用 Amazon Web Services Management Console 来定义随后将与 IAM 角色关联的策略。之后，在配置存储库创建模板时，可以将此 IAM 角色用作存储库创建角色。

Amazon Web Services Management Console

要使用 JSON 策略编辑器为存储库创建模板创建自定义策略。

1. 登录 Amazon Web Services Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择策略。
3. 选择创建策略。
4. 在策略编辑器部分，选择 JSON 选项。
5. 在 JSON 字段中输入以下策略。

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "ecr:CreateRepository",
                    "ecr:ReplicateImage",
                    "ecr:TagResource"
                ],
                "Resource": "*"
            },
            {
```

```
                  "Effect": "Allow",
                  "Action": [
                      "kms:CreateGrant",
                      "kms:RetireGrant",
                      "kms:DescribeKey"
                  ],
                  "Resource": "*"
              }
          ]
}
```

6. 解决策略验证过程中生成的任何安全警告、错误或常规警告，然后选择下一步。
7. 向策略添加完权限后，选择下一步。
8. 在查看和创建页面上，为创建的策略键入策略名称和描述（可选）。查看此策略中定义的权限以查看策略授予的权限。
9. 选择创建策略可保存新策略。
10. 创建为创建模板分配此策略的角色，请参阅为存储库创建模板创建 IAM 角色。

为存储库创建模板创建 IAM 角色

当您在使用存储库标签的存储库创建模板中指定存储库创建角色或在模板中使用 KMS 时，您可以使用创建可由 Amazon ECR 使用的角色。 Amazon Web Services Management Console

Amazon Web Services Management Console

要创建角色。

1. 登录 Amazon Web Services Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 在控制台的导航窗格中，选择 Roles，然后选择 Create role。
3. 选择自定义信任策略角色类型。
4. 在自定义信任策略部分，粘贴下面列出的自定义信任策略：

```
{
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Principal": {
```

```
            "Service": "ecr.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
    ]
}
```

5. 选择下一步。
6. 在添加权限页面中，从权限策略列表中选中您之前创建的自定义策略旁的复选框，然后选择下一步。
7. 对于角色名称，请为您的角色输入一个名称。角色名称在您的角色中必须是唯一的 Amazon Web Services 账户。在策略中使用角色名称或将其作为 ARN 的一部分时，角色名称区分大小写。在控制台中向客户显示角色名称时（例如在登录过程中），角色名称不区分大小写。由于多个实体可能引用该角色，因此，角色创建完毕后，您将无法编辑角色名称。
8. （可选）对于 Description（描述），输入新角色的描述。
9. 检查角色，然后选择创建角色。

创建存储库创建模板

完成模板的必要先决条件后，即可继续创建存储库创建模板。

Amazon Web Services Management Console

创建存储库创建模板（Amazon Web Services Management Console）

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 在导航栏中，选择存储库创建模板的创建区域。
3. 在导航窗格中，依次选择私有注册表、存储库创建模板。
4. 在存储库创建模板页面上，选择创建模板。
5. 在步骤 1：定义模板页面上，对于模板详细信息，选择特定前缀，将模板应用于特定的存储库命名空间前缀，或者选择 ECR 注册表中的任意前缀，将模板应用于与该区域中任何其他模板都不匹配的所有存储库。

    a. 如果选择特定前缀，则在前缀中指定要应用模板的存储库命名空间前缀。前缀末尾始终应用假定的 `/`。例如，前缀 `prod` 将应用于以 `prod/` 开头的所有存储库。同理，前缀 `prod/team` 将应用于以 `prod/team/` 开头的所有存储库。
    b. 如果选择 ECR 注册表中的任意前缀，则前缀将设置为 `ROOT`。

6. 在应用对象中，指定此模板将应用于哪些 Amazon ECR 工作流。选项为 `PULL_THROUGH_CACHE` 和 `REPLICATION`。
7. 在模板描述中，为模板指定可选描述，然后选择下一步。
8. 在步骤 2：添加存储库创建配置页面上，指定使用模板创建的存储库要应用的存储库设置配置。

   a. 在映像标签可变性中，选择要使用的标签可变性设置。有关更多信息，请参阅 防止映像标签在 Amazon ECR 中被覆盖。

      选择可变后，映像标签可以被覆盖。对于通过复制操作创建的存储库所使用的模板，建议使用此设置。这样可以确保当标签相同时，Amazon ECR 可以更新缓存的映像。

      选择不可变后，将防止映像标签被覆盖。为存储库配置了不可变标签后，如果尝试推送某个映像但其标签在存储库中已存在，系统将返回 `ImageTagAlreadyExistsException` 错误。当存储库开启标签不可变性时，这会影响所有标签，您不能将某些标签配置为不可变，而将其他标签配置为可变。

   b. 在加密配置中，选择要使用的加密设置。有关更多信息，请参阅 静态加密。

      选择 AES-256 时，Amazon ECR 使用具有 Amazon Simple Storage Service 托管加密密钥的服务器端加密，从而使用 AES-256 加密算法对静态数据进行加密。此服务不会产生额外费用。

      选择 Amazon KMS 时，Amazon ECR 使用具有 Amazon Key Management Service（Amazon KMS）中存储的密钥的服务器端加密。当您使用 Amazon KMS 加密数据时，您可以使用默认的 Amazon 托管密钥（由 Amazon ECR 管理），也可以指定自己的 Amazon KMS 密钥（称为客户托管密钥）。

      > **Note**
      >
      > 存储库创建之后，存储库的加密设置无法更改。

   c. 在存储库权限中，请指定使用此模板创建的存储库要应用的存储库权限策略。您可以选择使用下拉菜单，为最常见的使用案例选择一个 JSON 示例。有关更多信息，请参阅 Amazon ECR 中的私有存储库策略。

   d. 在存储库生命周期策略中，指定使用此模板创建的存储库要应用的存储库生命周期策略。您可以选择使用下拉菜单，为最常见的使用案例选择一个 JSON 示例。有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像。

   e. 对于存储库 Amazon 标签，以键值对的形式指定要与使用此模板创建的存储库关联的元数据，然后选择下一步。有关更多信息，请参阅 在 Amazon ECR 中标记私有存储库。

   f. 对于存储库创建角色，请从下拉菜单中选择自定义 IAM 角色，以便在模板中使用存储库标签或 KMS 时用于存储库创建模板（详情请参阅为存储库创建模板创建 IAM 角色）。然后选择下一步。

9. 在步骤 3：审核并创建页面上，审核您为存储库创建模板指定的设置。选择编辑选项以进行更改。完成后，选择创建。

Amazon CLI

该create-repository-creation-template Amazon CLI 命令用于为您的私有注册表创建存储库模板。

创建存储库创建模板（Amazon CLI）

1. Amazon CLI 使用生成create-repository-creation-template命令的骨架。

```
aws ecr create-repository-creation-template \
    --generate-cli-skeleton
```

   该命令的输出显示了存储库创建模板的完整语法。

```
{
    "appliedFor":[""], // string array, but valid are PULL_THROUGH_CACHE and
 REPLICATION
    "prefix": "string",
    "description": "string",
    "imageTagMutability": "MUTABLE"|"IMMUTABLE",
    "repositoryPolicy": "string",
    "lifecyclePolicy": "string"
    "encryptionConfiguration": {
        "encryptionType": "AES256"|"KMS",
        "kmsKey": "string"
    },
    "resourceTags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "customRoleArn": "string", // must be a valid IAM Role ARN
```

```
}
```

2. 使用上一步中的输出创建名为 `repository-creation-template.json` 的文件。此模板为根据`prod/*`存储库策略创建的任何存储库设置 KMS 加密密钥，该策略允许将映像推送和拉取到未来的存储库，设置生命周期策略，使超过两周的图像过期，并设置一个自定义角色以允许 ECR 访问 KMS 密钥并将资源标签分配`examplekey`给未来的存储库。

```
{
    "prefix": "prod",
    "description": "For repositories cached from my PTC rule and in my
 replication configuration that start with 'prod/'",
    "appliedFor": ["PULL_THROUGH_CACHE","REPLICATION"],
    "encryptionConfiguration": {
        "encryptionType": "KMS",
        "kmsKey": "arn:aws:kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-
cdef-example11111"
    },
    "resourceTags": [
        {
            "Key": "examplekey",
            "Value": "examplevalue"
        }
    ],
    "imageTagMutability": "MUTABLE",
    "repositoryPolicy": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Sid
\":\"AllowPushPullIAMRole\",\"Effect\":\"Allow\",\"Principal\":{\"AWS\":
\"arn:aws:iam::111122223333:user\/IAMusername\"},\"Action\":[\"ecr:BatchGetImage
\",\"ecr:BatchCheckLayerAvailability\",\"ecr:CompleteLayerUpload\",
\"ecr:GetDownloadUrlForLayer\",\"ecr:InitiateLayerUpload\",\"ecr:PutImage\",
\"ecr:UploadLayerPart\"]}]}",
    "lifecyclePolicy": "{\"rules\":[{\"rulePriority\":1,\"description\":\"Expire
 images older than 14 days\",\"selection\":{\"tagStatus\":\"any\",\"countType
\":\"sinceImagePushed\",\"countUnit\":\"days\",\"countNumber\":14},\"action\":
{\"type\":\"expire\"}}]}",
    "customRoleArn": "arn:aws:iam::111122223333:role/myRole"
}
```

3. 使用以下命令来创建存储库创建模板。请务必指定在上一步中创建的配置文件的名称来代替以下示例中的 `repository-creation-template.json`。

```
aws ecr create-repository-creation-template \
    --cli-input-json file://repository-creation-template.json
```

更新仓库创建模板

如果需要更改存储库创建模板配置，则可以对其进行编辑。编辑存储库创建模板后，新配置即应用于现有模板。

> ⚠ Important
>
> 这不会对之前创建的存储库产生任何影响。

Amazon Web Services Management Console

要编辑存储库创建模板（Amazon Web Services Management Console）

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 在导航栏中，选择要编辑的存储库创建模板所在的区域。
3. 在导航窗格中，选择私有注册表，然后选择设置。
4. 从导航栏中选择存储库创建模板。
5. 在存储库创建模板页面上，选择要编辑的存储库创建模板。
6. 从操作下拉菜单中，选择编辑。
7. 查看和更新配置设置。
8. 选择更新以应用新的创建模板配置。

Amazon CLI

要编辑存储库创建模板（Amazon CLI）

- 使用update-repository-creation-template命令更新现有的存储库创建模板。必须指定模板的 `prefix` 值。以下示例更新了带有 `prod` 前缀的存储库创建模板。

  ```
  aws ecr update-repository-creation-template \
       --prefix prod \
       --image-tag-mutability="IMMUTABLE"
  ```

  命令的输出显示了更新的存储库创建模板的详细信息。

在 Amazon ECR 中删除存储库创建模板

使用完存储库创建模板后，您可以将其删除。删除存储库创建模板后，除非找到其他匹配的模板，否则在提取缓存或复制操作期间在关联前缀下新创建的任何存储库都将继承默认设置，请参阅存储库创建模板的工作原理。

⚠ Important

这不会对之前创建的存储库产生任何影响。

Amazon Web Services Management Console

删除存储库创建模板（Amazon Web Services Management Console）

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 在导航栏中，选择要删除的存储库创建模板所在的区域。
3. 在导航窗格中，依次选择私有注册表、存储库创建模板。
4. 在存储库创建模板页面上，选择要删除的存储库创建模板。
5. 在操作下拉菜单中，选择删除。

Amazon CLI

删除存储库创建模板（Amazon CLI）

- 使用 delete-repository-creation-template.html 命令删除现有的存储库创建模板。必须指定模板的 `prefix` 值。以下示例删除了带有 `prod` 前缀的存储库创建模板。

  ```
  aws ecr delete-repository-creation-template \
       --prefix prod
  ```

  命令的输出显示了已删除存储库创建模板的详细信息。

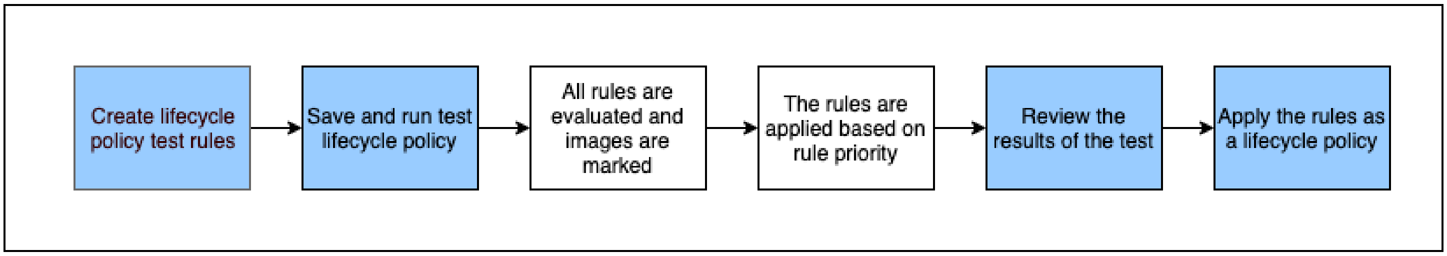
在 Amazon ECR 中使用生命周期策略自动清理映像

Amazon ECR 生命周期策略提供了对私有存储库中镜像的生命周期管理的更多控制。生命周期策略是一项或多项规则，每项规则都为 Amazon ECR 定义一个操作。根据生命周期策略中的过期条件，映像将根据使用时限或计数在 24 小时内过期。当 Amazon ECR 基于生命周期策略执行操作时，此操作将在 Amazon CloudTrail中记录为一个事件。有关更多信息，请参阅 使用记录 Amazon ECR 操作 Amazon CloudTrail。

生命周期策略工作原理

生命周期策略由一条或多条规则组成，这些规则确定存储库中的哪些镜像应过期。在考虑使用生命周期策略时，务必使用生命周期策略预览来确认生命周期策略视为过期的镜像，然后再将其应用到存储库。将生命周期策略应用到存储库后，镜像将在达到到期标准后 24 小时内过期。当 Amazon ECR 基于生命周期策略执行操作时，这将在 Amazon CloudTrail中记录为一个事件。有关更多信息，请参阅 使用记录 Amazon ECR 操作 Amazon CloudTrail。

下图显示了生命周期策略工作流程。



1. 创建一个或多个测试规则。
2. 保存测试规则并运行预览。
3. 生命周期策略评估程序遍历所有规则，并标记每个规则影响的镜像。
4. 然后，生命周期策略评估程序根据规则优先级应用规则，并显示存储库中的哪些镜像设置为过期。规则优先级数字越低意味着优先级越高。例如，优先级为 1 的规则优先于优先级为 2 的规则。
5. 查看测试结果，确保标记为过期的镜像符合您预期的要求。
6. 将测试规则应用为存储库的生命周期策略。
7. 创建生命周期策略后，镜像将在达到过期标准后 24 小时内过期。

生命周期策略评估规则

生命周期策略评估器负责解析生命周期策略的明文 JSON，评估所有规则，然后根据规则优先级将这些规则应用于存储库中的镜像。下文更详细地解释了生命周期策略评估器采用的逻辑。有关示例，请参阅 Amazon ECR 中的生命周期策略的示例。

- 当存储库中存在参考项目时，Amazon ECR 生命周期策略会在主题图片删除后的 24 小时内自动清理这些项目。
- 无论规则优先级如何，评估器都会同时评估所有规则。评估所有规则后，将根据规则优先级应用这些规则。
- 镜像由一条或零条规则设为过期。
- 与规则的标记要求匹配的镜像不能被优先级较低的规则设为过期。
- 规则永远不能标记已由较高优先级规则标记的镜像，但仍然可以将其识别为未过期。
- 规则集必须包含一组唯一的标签前缀。
- 只允许一个规则选择未标记的镜像。
- 如果清单列表引用了映像，则在未先删除清单列表的情况下，该映像无法过期。
- 过期始终按 `pushed_at_time` 排序，并且始终是较早的镜像在较新的镜像之前过期。
- 生命周期策略规则可以指定 `tagPatternList` 或 `tagPrefixList`，但不能同时指定这两个选项。但是，一个生命周期策略可能包含多个规则，而不同的规则可能同时使用模式和前缀列表。如果 `tagPatternList` 或 `tagPrefixList` 值中的全部标签均与任何映像的标签匹配，则表示该映像成功匹配。
- 仅在 `tagStatus` 为 `tagged` 时才能使用 `tagPatternList` 或 `tagPrefixList` 参数。
- 使用时 `tagPatternList`，只要与通配符筛选条件匹配，即表示该映像成功匹配。例如，假设应用的筛选条件为 `prod*`，则将匹配名称以 `prod` 开头的映像标签，例如 `prod`、`prod1` 或 `production-team1`。同样，如果应用的筛选条件为 `*prod*`，则将匹配名称包含 `prod` 的映像标签，例如 `repo-production` 或 `prod-team`。

> ⚠ Important
>
> 每个字符串最多可以使用四个通配符（*）。例如，`["*test*1*2*3", "test*1*2*3*"]` 有效，而 `["test*1*2*3*4*5*6"]` 无效。

- 使用 `tagPrefixList` 时，如果 `tagPrefixList` 值中的全部通配符筛选条件均与任何映像的标签匹配，则表示该映像成功匹配。

- 仅当 `countType` 为 `sinceImagePushed` 时，才能使用 `countUnit` 参数。
- 使用 `countType = imageCountMoreThan`，镜像基于 `pushed_at_time` 从最新到早排序，然后所有大于指定计数的镜像都将过期。
- 使用 `countType = sinceImagePushed`，其 `pushed_at_time` 早于指定天数 (基于 `countNumber`) 的所有镜像均会过期。

在 Amazon ECR 中创建生命周期策略预览

您可以使用生命周期策略预览在应用生命周期策略之前查看其对映像存储库影响的方法。在将生命周期策略应用到存储库之前进行预览被认为是一种最佳实践。

> **Note**
>
> 如果您使用 Amazon ECR 复制功能跨不同的区域或账户创建存储库的副本，请注意，生命周期策略可能仅对创建该策略的区域中的存储库执行操作。因此，如果您开启了复制功能，则建议在要将存储库复制到的每个区域和账户中创建一个生命周期策略。

创建生命周期策略预览 (Amazon Web Services Management Console)

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。
2. 从导航栏中，选择包含要对其执行生命周期策略预览的存储库的区域。
3. 在导航窗格中的私有注册表下，选择存储库。
4. 在私有存储库页面上选择一个存储库，然后使用操作下拉列表选择生命周期策略。
5. 在该存储库的生命周期策略页面上，选择编辑测试规则、创建规则。
6. 为每个测试生命周期策略规则指定以下详细信息。

    a. 对于规则优先级，输入该规则优先级的编号。规则优先级决定了生命周期策略规则的应用顺序。数字越小意味着优先级越高。例如，优先级为 1 的规则优先于优先级为 2 的规则。

    b. 对于规则描述，输入该生命周期策略规则的说明。

    c. 对于映像状态，选择已标记（通配符匹配）、已标记（前缀匹配）、未标记或任意。

    > **Important**
    >
    > 如果指定多个标签，则仅选择具有所有指定标签的镜像。

d. 如果映像状态选择的是已标记（通配符匹配），则对于指定用于通配符匹配的标签，您可以指定带有通配符（*）的映像标签列表，以便根据您的生命周期策略执行操作。例如，假设您的映像标记为 `prod`、`prod1`、`prod2` 等，则可以指定 `prod*` 来对所有这些映像执行操作。如果指定多个标签，则仅选择具有所有指定标签的镜像。

> **Important**
>
> 每个字符串最多可以使用四个通配符（*）。例如，`["*test*1*2*3", "test*1*2*3*"]` 有效，而 `["test*1*2*3*4*5*6"]` 无效。

e. 如果映像状态选择的是已标记（前缀匹配），则对于指定用于前缀匹配的标签，您可以指定一个映像标签列表，以便根据您的生命周期策略执行操作。

f. 对于匹配条件，选择自从推送映像以来或映像计数超过，然后指定一个值。

g. 选择保存。

7. 重复第 5-7 步以创建其他测试生命周期策略规则。
8. 要运行生命周期策略预览，请选择保存并运行测试。
9. 在测试生命周期规则的镜像匹配下方，查看生命周期策略预览的影响。
10. 如果对预览结果满意，请选择应用为生命周期策略以创建具有指定规则的生命周期策略。应用生命周期策略后，受影响的镜像会在 24 小时内过期。
11. 如果您对预览结果不满意，可以删除一个或多个测试生命周期规则，创建一个或多个规则来替换它们，然后再重复测试。

在 Amazon ECR 中为存储库创建生命周期策略

使用生命周期策略创建一组规则，这些规则会让未使用的存储库映像过期。创建生命周期策略后，受影响的映像会在 24 小时内过期。

> **Note**
>
> 如果您使用 Amazon ECR 复制功能跨不同的区域或账户创建存储库的副本，请注意，生命周期策略可能仅对创建该策略的区域中的存储库执行操作。因此，如果您开启了复制功能，则建议在要将存储库复制到的每个区域和账户中创建一个生命周期策略。

先决条件

最佳实践：创建生命周期策略预览，以验证在生命周期策略规则下过期的映像是否符合预期。有关说明，请参阅在 Amazon ECR 中创建生命周期策略预览。

创建生命周期策略 (Amazon Web Services Management Console)

1. 在https://console.aws.amazon.com/ecr/存储库中打开 Amazon ECR 控制台。
2. 从导航栏中，选择包含要对其创建生命周期策略的存储库的区域。
3. 在导航窗格中的私有注册表下，选择存储库。
4. 在私有存储库页面上选择一个存储库，然后使用操作下拉列表选择生命周期策略。
5. 在该存储库的生命周期策略页面上，选择创建规则。
6. 输入生命周期策略规则的以下详细信息。

    a. 对于规则优先级，输入该规则优先级的编号。规则优先级决定了生命周期策略规则的应用顺序。规则优先级数字越低意味着优先级越高。例如，优先级为 1 的规则优先于优先级为 2 的规则。

    b. 对于规则描述，输入该生命周期策略规则的说明。

    c. 对于映像状态，选择已标记（通配符匹配）、已标记（前缀匹配）、未标记或任意。

    > ⚠ Important
    >
    > 如果指定多个标签，则仅选择具有所有指定标签的镜像。

    d. 如果映像状态选择的是已标记（通配符匹配），则对于指定用于通配符匹配的标签，您可以指定带有通配符（*）的映像标签列表，以便根据您的生命周期策略执行操作。例如，假设您的映像标记为 `prod`、`prod1`、`prod2` 等，则可以指定 `prod*` 来对所有这些映像执行操作。如果指定多个标签，则仅选择具有所有指定标签的镜像。

    > ⚠ Important
    >
    > 每个字符串最多可以使用四个通配符（*）。例如，`["*test*1*2*3", "test*1*2*3*"]` 有效，而 `["test*1*2*3*4*5*6"]` 无效。

    e. 如果映像状态选择的是已标记（前缀匹配），则对于指定用于前缀匹配的标签，您可以指定一个映像标签列表，以便根据您的生命周期策略执行操作。

    f. 对于匹配条件，选择自从推送映像以来或映像计数超过，然后指定一个值。

   g. 选择保存。

7. 重复步骤 5-7 以创建其他生命周期策略规则。

创建生命周期策略 (Amazon CLI)

1. 获取要为其创建生命周期策略的存储库的名称。

   ```
   aws ecr describe-repositories
   ```

2. 创建名为 `policy.json` 的本地文件，其中包含生命周期策略的内容。有关生命周期策略示例，请参阅 Amazon ECR 中的生命周期策略的示例。
3. 通过指定存储库名称并引用创建的生命周期策略 JSON 文件来创建生命周期策略。

   ```
   aws ecr put-lifecycle-policy \
         --repository-name repository-name \
         --lifecycle-policy-text file://policy.json
   ```

Amazon ECR 中的生命周期策略的示例

以下是示例生命周期策略，其中显示了语法。

要查看有关策略属性的更多信息，请参阅Amazon ECR 中的生命周期策略属性。有关使用创建生命周期策略的说明 Amazon CLI，请参阅创建生命周期策略 (Amazon CLI)。

生命周期策略模板

在与存储库关联之前评估生命周期策略的内容。以下是生命周期策略的 JSON 语法模板。

```
{
        "rules": [
            {
                "rulePriority": integer,
                "description": "string",
                "selection": {
                    "tagStatus": "tagged"|"untagged"|"any",
                    "tagPatternList": list<string>,
                    "tagPrefixList": list<string>,
                    "countType": "imageCountMoreThan"|"sinceImagePushed",
```

```
                    "countUnit": "string",
                    "countNumber": integer
                },
                "action": {
                    "type": "expire"
                }
            }
        ]
    }
```

根据镜像使用期限筛选

以下示例演示了一个策略的生命周期策略语法，该策略使用 `prod*` 的 `tagPatternList` 来确保具有以 `prod` 开头的标签并且存在时间超过 14 天的映像会过期。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Expire images older than 14 days",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["prod*"],
                "countType": "sinceImagePushed",
                "countUnit": "days",
                "countNumber": 14
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

根据镜像计数筛选

以下示例演示了一个策略的生命周期策略语法，该策略将仅保留一个未标记的映像并使所有其他映像过期。

```
{
    "rules": [
        {
```

```
            "rulePriority": 1,
            "description": "Keep only one untagged image, expire all others",
            "selection": {
                "tagStatus": "untagged",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

根据多个规则筛选

以下示例在生命周期策略中使用多条规则。在此提供了一个示例存储库和生命周期策略以及结果的说明。

示例 A

存储库内容：

- 镜像 A，标签列表：["beta-1", "prod-1"]，已推送：10 天前
- 镜像 B，标签列表：["beta-2", "prod-2"]，已推送：9 天前
- 镜像 C，标签列表：["beta-3"]，已推送：8 天前

生命周期策略文本：

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["prod*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
```

```
                "type": "expire"
            }
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["beta*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

此生命周期策略的逻辑是：

- 规则 1 标识带有前缀 `prod` 标签的镜像。它应该从最早的镜像开始标记镜像，直到剩下一个或更少的匹配镜像。它将镜像 A 标记为过期。
- 规则 2 标识带有前缀 `beta` 标签的镜像。它应该从最早的镜像开始标记镜像，直到剩下一个或更少的匹配镜像。它将镜像 A 和镜像 B 标记为过期。但是，规则 1 已标记镜像 A，如果镜像 B 已过期，则会违反规则 1，因此跳过。
- 结果：镜像 A 已过期。

示例 B

这与前面的示例相同，但规则优先级顺序会更改以展示结果。

存储库内容：

- 镜像 A，标签列表：["beta-1", "prod-1"]，已推送：10 天前
- 镜像 B，标签列表：["beta-2", "prod-2"]，已推送：9 天前
- 镜像 C，标签列表：["beta-3"]，已推送：8 天前

生命周期策略文本：

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["beta*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["prod*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

此生命周期策略的逻辑是：

- 规则 1 标识带有前缀 beta 标签的镜像。它应该从最早的镜像开始标记镜像，直到剩下一个或更少的匹配镜像。它可以看到所有三个镜像，并将镜像 A 和镜像 B 标记为过期。
- 规则 2 标识带有前缀 prod 标签的镜像。它应该从最早的镜像开始标记镜像，直到剩下一个或更少的匹配镜像。它不会看到任何镜像，因为所有可用镜像已经被规则 1 标记，因此不会标记其他镜像。
- 结果：镜像 A 和 B 已过期。

在单个规则中筛选多个标签

以下示例为单个规则中的多个标签模式指定了生命周期策略语法。在此提供了一个示例存储库和生命周期策略以及结果的说明。

示例 A

在单个规则上指定多个标签模式时，映像必须与所有列出的标签模式匹配。

存储库内容：

- 镜像 A，标签列表：["alpha-1"]，已推送：12 天前
- 镜像 B，标签列表：["beta-1"]，已推送：11 天前
- 镜像 C，标签列表：["alpha-2", "beta-2"]，已推送：10 天前
- 镜像 D，标签列表：["alpha-3"]，已推送：4 天前
- 图片 E，标签列表：["beta-3"]，已推送：3 天前
- 镜像 F，标签列表：["alpha-4", "beta-4"]，已推送：2 天前

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["alpha*", "beta*"],
                "countType": "sinceImagePushed",
                "countNumber": 5,
                "countUnit": "days"
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

此生命周期策略的逻辑是：

- 规则 1 用于识别标记有前缀 `alpha` 和`beta` 的映像。它可以看到镜像 C 和 F。它应该标记超过五天的镜像，镜像 C 符合此条件。
- 结果：镜像 C 已过期。

示例 B

以下示例说明标签不是独占的。

存储库内容：

- 镜像 A，标签列表：["alpha-1", "beta-1", "gamma-1"]，已推送：10 天前
- 镜像 B，标签列表：["alpha-2", "beta-2"]，已推送：9 天前
- 镜像 C，标签列表：["alpha-3", "beta-3", "gamma-2"]，已推送：8 天前

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["alpha*", "beta*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

此生命周期策略的逻辑是：

- 规则 1 用于识别标记有前缀 `alpha` 和`beta` 的映像。它可以看到所有的镜像。它应该从最早的镜像开始标记镜像，直到剩下一个或更少的匹配镜像。它将镜像 D 标记为过期。
- 结果：镜像 A 和 B 已过期。

筛选所有镜像

以下生命周期策略示例指定具有不同筛选条件的所有镜像。此处提供了一个示例存储库和生命周期策略以及结果的说明。

示例 A

下面显示了应用于所有规则但仅保留一个镜像并使所有其他镜像过期的策略的生命周期策略语法。

存储库内容：

- 镜像 A，标签列表：["alpha-1"]，已推送：4 天前
- 镜像 B，标签列表：["beta-1"]，已推送：3 天前
- 镜像 C，标签列表：[]，已推送：2 天前
- 镜像 D，标签列表：["alpha-2"]，已推送：1 天前

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

此生命周期策略的逻辑是：

- 规则 1 标识所有镜像。它可以看到镜像 A、B、C 和 D。它应该使最新镜像之外的所有其他镜像过期。它将镜像 A、B 和 C 标记为过期。
- 结果：镜像 A、B 和 C 已过期。

示例 B

以下示例说明了将所有规则类型组合在一个策略中的生命周期策略。

存储库内容：

- 镜像 A，标签列表：["alpha-1", "beta-1"]，已推送：4 天前
- 图片 B，标签列表：[]，已推送：3 天前
- 镜像 C，标签列表：["alpha-2"]，已推送：2 天前
- 镜像 D，标签列表：["git hash"]，已推送：1 天前
- 镜像 E，标签列表：[]，已推送：1 天前

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPatternList": ["alpha*"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
                "tagStatus": "untagged",
                "countType": "sinceImagePushed",
                "countUnit": "days",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
```

```
            "rulePriority": 3,
            "description": "Rule 3",
            "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

此生命周期策略的逻辑是：

- 规则 1 标识带有前缀 `alpha` 标签的镜像。它识别镜像 A 和 C。它应该保留最新镜像并将其余镜像标记为过期。它将镜像 A 标记为过期。
- 规则 2 标识未标记的镜像。它可以识别镜像 B 和 E。它应该将所有超过一天的镜像标记为过期。它将镜像 B 标记为过期。
- 规则 3 标识所有镜像。它识别镜像 A、B、C、D 和 E。它应该保留最新镜像并将其余镜像标记为过期。但是，它无法标记镜像 A、B、C 或 E，因为它们由优先级更高的规则识别。它将镜像 D 标记为过期。
- 结果：镜像 A、B 和 D 已过期。

Amazon ECR 中的生命周期策略属性

生命周期策略具有以下属性。

要查看生命周期策略的示例，请参阅Amazon ECR 中的生命周期策略的示例。有关使用创建生命周期策略的说明 Amazon CLI，请参阅创建生命周期策略 (Amazon CLI)。

规则优先级

`rulePriority`

类型：整数

必需：是

设置应用规则的顺序，从低到高。优先级为 1 的生命周期策略规则首先应用，优先级为 2 的规则下一个应用，依此类推。当您向某个生命周期策略添加规则时，必须为每个规则赋予一个唯一的 `rulePriority` 值。但是，在策略中的各规则之间，值不需要顺序。具有 `tagStatus` 值 any 的规则必须具有最大的 `rulePriority` 值并且最后被评估。

描述

`description`

类型：字符串

必需：否

(可选) 描述生命周期策略中规则的用途。

标签状态

`tagStatus`

类型：字符串

必需：是

确定要添加的生命周期策略规则是否为镜像指定标签。可接受的选项包括 `tagged`、`untagged` 或 `any`。如果您指定 `any`，则所有镜像都会根据它们评估规则。如果指定 `tagged`，还必须指定 `tagPrefixList` 值。如果指定 `untagged`，那么必须省略 `tagPrefixList`。

标签模式列表

`tagPatternList`

类型：list[string]

必填项：`tagStatus` 设置为“已标记”且未指定 `tagPrefixList` 时，是必填项

为已标记的映像创建生命周期策略时，最佳实践是使用 `tagPatternList` 来指定要过期的标签。您必须指定以逗号分隔的可能包含通配符（`*`）的映像标签模式列表，以便根据此列表执行生命周期策略操作。例如，假设映像标记为 `prod`、`prod1`、`prod2` 等，则可以使用标签模式列表 `prod*` 来指定所有这些映像。如果指定多个标签，则仅选择具有所有指定标签的镜像。

> ⚠ Important
>
> 每个字符串最多可以使用四个通配符（`*`）。例如，`["*test*1*2*3", "test*1*2*3*"]` 有效，而 `["test*1*2*3*4*5*6"]` 无效。

标签前缀列表

`tagPrefixList`

类型：list[string]

必填项：`tagStatus` 设置为“已标记”且未指定 `tagPatternList` 时，是必填项

仅在您指定了 `"tagStatus": "tagged"` 但未指定 `tagPatternList` 时才使用。您必须指定以逗号分隔的镜像标签前缀列表，以便根据此列表执行生命周期策略操作。例如，如果您的镜像被标记为 `prod`、`prod1`、`prod2` 等，则可以使用标签前缀 `prod` 以指定所有这些标签。如果指定多个标签，则仅选择具有所有指定标签的镜像。

计数类型

`countType`

类型：字符串

必需：是

指定要应用于镜像的计数类型。

如果 `countType` 设置为 `imageCountMoreThan`，您还可以指定 `countNumber` 以创建一个规则，用于设置存储库中存在的镜像数量限制。如果 `countType` 设置为 `sinceImagePushed`，您还可以指定 `countUnit` 和 `countNumber`，以指定存储库中存在的镜像的时间限制。

计数单位

`countUnit`

类型：字符串

必需：是，仅当 `countType` 设置为 `sinceImagePushed` 时

指定计数单位 `days` 作为时间单位，除此之外，还指定 `countNumber` 表示天数。

只有在 `countType` 为 `sinceImagePushed` 时才能指定；如果您在 `countType` 是任何其他值时指定计数单位，将发生错误。

计数

`countNumber`

类型：整数

必需：是

指定计数数量。可接受的值为正整数 (`0` 不是可接受的值)。

如果使用的 `countType` 是 `imageCountMoreThan`，则该值为您希望在存储库中保留的镜像的最大数量。如果使用的 `countType` 是 `sinceImagePushed`，则该值为镜像的最大使用期限。

操作

`type`

类型：字符串。

必需：是

指定操作类型。支持的值为 `expire`。

Amazon Elastic Container Registry 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。 Amazon 还为您提供可以安全使用的服务。作为 Amazon 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于 Amazon ECR 的合规性计划，请参阅合规性计划范围内的Amazon 服务。
- 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。

该文档帮助您了解如何在使用 Amazon ECR 时应用责任共担模式。以下主题说明如何配置 Amazon ECR 以实现您的安全性和合规性目标。您还将学习如何使用其他 Amazon 服务来帮助您监控和保护您的 Amazon ECR 资源。

主题

适用于 Amazon Elastic Container Registry 的 Identity and Access Management

Amazon Identity and Access Management (IAM) Amazon Web Services 服务 可帮助管理员安全地控制对 Amazon 资源的访问权限。IAM 管理员控制谁可以通过身份验证（登录）和授权（具有权限）使用 Amazon ECR 资源。您可以使用 IAM Amazon Web Services 服务 ，无需支付额外费用。

主题

- 使用身份进行身份验证
- 使用策略管理访问
- Amazon Elastic Container Registry 如何与 IAM 结合使用
- Amazon Elastic Container Registry 基于身份的策略示例
- 使用基于标签的访问控制
- Amazon Amazon 弹性容器器注册表的托管策略
- 对 Amazon ECR 使用服务相关角色
- 排查 Amazon Elastic Container Registry 的身份和访问权限问题

受众

您的使用方式 Amazon Identity and Access Management (IAM) 会有所不同，具体取决于您在 Amazon ECR 中所做的工作。

服务用户 - 如果您使用 Amazon ECR 服务来完成工作，您的管理员会为您提供所需的凭证和权限。随着您使用更多 Amazon ECR 功能来完成工作，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon ECR 中的功能，请参阅排查 Amazon Elastic Container Registry 的身份和访问权限问题。

服务管理员 – 如果您在公司负责管理 Amazon ECR 资源，您可能对 Amazon ECR 具有完全访问权限。您有责任确定您的服务用户应访问哪些 Amazon ECR 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 Amazon ECR 搭配使用的更多信息，请参阅Amazon Elastic Container Registry 如何与 IAM 结合使用。

IAM 管理员 – 如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 Amazon ECR 的访问的详细信息。要查看您可在 IAM 中使用的 Amazon ECR 基于身份的策略示例，请参阅 Amazon Elastic Container Registry 基于身份的策略示例。

使用身份进行身份验证

身份验证是您 Amazon 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 Amazon Web Services 账户根用户任 IAM 角色进行身份验证（登录 Amazon）。

如果您 Amazon 以编程方式访问，则会 Amazon 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 Amazon 工具，则必须自己签署请求。有关使用推

荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的用于签署 API 请求的Amazon 签名版本 4。

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如， Amazon 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 中的Amazon 多重身份验证。

Amazon Web Services 账户 root 用户

创建时 Amazon Web Services 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 Amazon Web Services 服务 和资源。此身份被称为 Amazon Web Services 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

IAM 用户和群组

I AM 用户是您 Amazon Web Services 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的用例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可以拥有一个名为的群组，IAMAdmins并向该群组授予管理 IAM 资源的权限。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 用户的使用案例。

IAM 角色

I AM 角色是您内部具有特定权限 Amazon Web Services 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。要在中临时担任 IAM 角色 Amazon Web Services Management Console，您可以从用户切换到 IAM 角色（控制台）。您可以通过调用 Amazon CLI 或 Amazon API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的代入角色的方法。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。
- 临时 IAM 用户权限：IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 Amazon Web Services 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南中的 IAM 中的跨账户资源访问。
- 跨服务访问 — 有些 Amazon Web Services 服务 使用其他 Amazon Web Services 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 Amazon，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 Amazon Web Services 服务 向下游服务发出请求的请求。 Amazon Web Services 服务只有当服务收到需要与其他 Amazon Web Services 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 Amazon Web Services 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种链接到的服务角色。 Amazon Web Services 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 Amazon Web Services 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon 上运行的应用程序 EC2 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 Amazon CLI 或 Amazon API 请求的应用程序的临时证书。这比在 EC2 实例中存储访问密钥更可取。要为 EC2 实例分配 Amazon 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含该角色，并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的使用 IAM 角色向在 A mazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

您可以 Amazon 通过创建策略并将其附加到 Amazon 身份或资源来控制中的访问权限。策略是其中的一个对象 Amazon ，当与身份或资源关联时，它会定义其权限。 Amazon 在委托人（用户、root 用

户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 Amazon 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 Amazon JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 Amazon Web Services Management Console Amazon CLI、或 Amazon API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户托管策略定义自定义 IAM 权限。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 Amazon Web Services 账户。托管策略包括 Amazon 托管策略和客户托管策略。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 Amazon Web Services 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 Amazon 托管策略。

其他策略类型

Amazon 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCPs) — SCPs 是 JSON 策略，用于指定中组织或组织单位 (OU) 的最大权限 Amazon Organizations。 Amazon Organizations 是一项用于对您的企业拥有的多 Amazon Web Services 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。SCP 限制成员账户中的实体（包括每个 Amazon Web Services 账户根用户实体）的权限。有关 Organization SCPs s 和的更多信息，请参阅《Amazon Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs) — RCPs 是 JSON 策略，您可以使用它来设置账户中资源的最大可用权限，而无需更新附加到您拥有的每个资源的 IAM 策略。RCP 限制成员账户中资源的权限，并可能影响身份（包括身份）的有效权限 Amazon Web Services 账户根用户，无论这些身份是否属于您的组织。有关 Organizations 的更多信息 RCPs，包括 Amazon Web Services 服务 该支持的列表 RCPs，请参阅《Amazon Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略：会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 Amazon 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

Amazon Elastic Container Registry 如何与 IAM 结合使用

在使用 IAM 管理对 Amazon ECR 的访问权限之前，您应该了解哪些 IAM 功能可用于 Amazon ECR。要全面了解 Amazon ECR 和其他 Amazon 服务如何与 IAM 配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的Amazon 服务。

主题

Amazon ECR 基于身份的策略

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。Amazon ECR 支持特定的操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》 中的 IAM JSON 策略元素参考。

操作

管理员可以使用 Amazon JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 Amazon API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

Amazon ECR 中的策略操作在操作前面使用以下前缀：`ecr:`。例如，要授予某人使用 Amazon ECR `CreateRepository` API 操作创建 Amazon ECR 存储库的权限，您应将 `ecr:CreateRepository` 操作纳入其策略中。策略语句必须包含 `Action` 或 `NotAction` 元素。Amazon ECR 定义了一组自己的操作，以描述您可以使用该服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
      "ecr:action1",
      "ecr:action2"
```

您也可以使用通配符（*) 指定多个操作。例如，要指定以单词 `Describe` 开头的所有操作，包括以下操作：

```
"Action": "ecr:Describe*"
```

要查看 Amazon ECR 操作的列表，请参阅 IAM 用户指南中的 Amazon Elastic Container Registry 的操作、资源和条件键。

资源

管理员可以使用 Amazon JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符（*）指示语句应用于所有资源。

```
"Resource": "*"
```

Amazon ECR 存储库资源具有以下 ARN：

```
arn:${Partition}:ecr:${Region}:${Account}:repository/${Repository-name}
```

有关格式的更多信息 ARNs，请参阅 Amazon 资源名称 (ARNs) 和 Amazon 服务命名空间。

例如，要在语句中指定 `us-east-1` 区域中的 `my-repo` 存储库，请使用以下 ARN：

```
"Resource": "arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
```

要指定属于特定账户的所有存储库，请使用通配符 (*)：

```
"Resource": "arn:aws:ecr:us-east-1:123456789012:repository/*"
```

要在单个语句中指定多个资源，请 ARNs 用逗号分隔。

```
"Resource": [
      "resource1",
      "resource2"
```

要查看 Amazon ECR 资源类型及其列表 ARNs，请参阅 IAM 用户指南中的亚马逊弹性容器注册表定义的资源。要了解您可以使用哪些操作指定每个资源的 ARN，请参阅 Amazon Elastic Container Registry 定义的操作。

条件键

管理员可以使用 Amazon JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 `Condition` 元素，或在单个 `Condition` 元素中指定多个键，则 Amazon 使用逻辑 `AND` 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑`OR`运算来 Amazon 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

Amazon 支持全局条件密钥和特定于服务的条件密钥。要查看所有 Amazon 全局条件键，请参阅 IAM 用户指南中的Amazon 全局条件上下文密钥。

Amazon ECR 定义了自己的一组条件键，还支持使用一些全局条件键。要查看所有 Amazon 全局条件键，请参阅 IAM 用户指南中的Amazon 全局条件上下文密钥。

大多数 Amazon ECR 操作都支持 `aws:ResourceTag` 和 `ecr:ResourceTag` 条件键。有关更多信息，请参阅 使用基于标签的访问控制。

要查看 Amazon ECR 条件键的列表，请参阅 IAM 用户指南中的 Amazon Elastic Container Registry 定义的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon Elastic Container Registry 定义的操作。

示例

要查看 Amazon ECR 基于身份的策略的示例，请参阅 Amazon Elastic Container Registry 基于身份的策略示例。

Amazon ECR 基于资源的策略

基于资源的策略是 JSON 策略文档，它们指定了指定委托人可在 Amazon ECR 资源上执行的操作以及在什么条件下可执行。Amazon ECR 支持针对 Amazon ECR 存储库的基于资源的权限策略。基于资源的策略允许您基于资源向其他账户授予使用权限。您也可以使用基于资源的策略以允许 Amazon 服务访问您的 Amazon ECR 存储库。

要启用跨账户访问，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的委托人。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源位于不同的 Amazon 账户中时，您还必须向委托人实体授予访问资源的权限。通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一账户中的委托人授予访问权限，则您不需要在基于身份的策略中添加其他 Amazon ECR 存储库权限。有关更多信息，请参阅《IAM 用户指南》 中的 IAM 角色与基于资源的策略有何不同。

Amazon ECR 服务仅支持一种类型的基于资源的策略（称为容器策略），这种策略附加到存储库。这个策略定义哪些委托人实体（账户、用户、角色和联合身份用户）可以在存储库上执行操作。要了解如何将基于资源的策略附加到存储库，请参阅Amazon ECR 中的私有存储库策略。

> **Note**
>
> 在 Amazon ECR 存储库策略中，策略元素 `Sid` 支持 IAM policy 所不支持的其他字符和间距。

示例

要查看 Amazon ECR 基于资源的策略的示例，请参阅 Amazon ECR 中的私有存储库策略示例，

基于 Amazon ECR 标签的授权

您可以将标签附加到 Amazon ECR 资源，或者在请求中将标签传递给 Amazon ECR。要基于标签控制访问，您需要使用 `ecr:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关标记 Amazon ECR 资源的更多信息，请参阅 在 Amazon ECR 中标记私有存储库。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅使用基于标签的访问控制。

Amazon ECR IAM 角色

I AM 角色是您的 Amazon 账户中具有特定权限的实体。

将临时凭证用于 Amazon ECR

可以使用临时凭证进行联合身份验证登录，分派 IAM 角色或分派跨账户角色。您可以通过调用AssumeRole或之类的 Amazon STS API 操作来获取临时安全证书GetFederationToken。

Amazon ECR 支持使用临时凭证。

服务相关角色

服务相关角色允许 Amazon 服务访问其他服务中的资源以代表您完成操作。服务相关角色显示在 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

Amazon ECR 支持服务相关角色。有关更多信息，请参阅 对 Amazon ECR 使用服务相关角色。

Amazon Elastic Container Registry 基于身份的策略示例

默认情况下，用户和角色没有创建或修改 Amazon ECR 资源的权限。他们也无法使用 Amazon Web Services Management Console、 Amazon Command Line Interface (Amazon CLI) 或 Amazon API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略（控制台）。

有关 Amazon ECR 定义的操作和资源类型（包括每种资源类型的格式）的详细信息，请参阅《 ARNs 服务授权参考》中的 Amazon Elastic Container Registry 的操作、资源和条件密钥。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的在 JSON 选项卡上创建策略。

主题

- 策略最佳实践
- 使用 Amazon ECR 控制台
- 允许用户查看他们自己的权限
- 访问一个 Amazon ECR 存储库

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon ECR 资源。这些操作可能会使 Amazon Web Services 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 Amazon 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的Amazon 托管策略。它们在你的版本中可用 Amazon Web Services 账户。我们建议您通过定义针对您的用例的 Amazon 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 Amazon 托管式策略或工作职能的Amazon 托管式策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使

用的，则也可以使用条件来授予对服务操作的访问权限 Amazon Web Services 服务，例如 Amazon CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。

- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 Amazon Web Services 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 Amazon ECR 控制台

要访问 Amazon Elastic Container Registry 控制台，您必须具有一组最低的权限。这些权限必须允许您列出和查看有关您 Amazon 账户中的 Amazon ECR 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

为确保这些实体仍然可以使用 Amazon ECR 控制台，请向实体添加`AmazonEC2ContainerRegistryReadOnly` Amazon 托管策略。有关更多信息，请参阅 IAM 用户指南中的为用户添加权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

对于仅调用 Amazon CLI 或 Amazon API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 Amazon CLI 或 Amazon API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

访问一个 Amazon ECR 存储库

在本示例中，您想向 Amazon 账户中的用户授予访问您的 Amazon ECR 存储库的权限。`my-repo`您还希望允许用户推送、提取和列出镜像。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"GetAuthorizationToken",
         "Effect":"Allow",
         "Action":[
            "ecr:GetAuthorizationToken"
         ],
         "Resource":"*"
      },
      {
         "Sid":"ManageRepositoryContents",
         "Effect":"Allow",
         "Action":[
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage"
         ],
         "Resource":"arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
      }
   ]
}
```

使用基于标签的访问控制

Amazon ECR `CreateRepository` API 操作允许您在创建存储库时指定标签。有关更多信息，请参阅 在 Amazon ECR 中标记私有存储库。

要使用户能够在创建存储桶时标记存储桶，用户必须有权使用创建资源的操作（例如，`ecr:CreateRepository`）。如果在资源创建操作中指定了标签，则 Amazon 会对 `ecr:CreateRepository` 操作执行额外的授权，以验证用户是否具备创建标签的权限。

您可以通过 IAM policy 来使用基于标签的访问控制。示例如下。

以下策略仅允许用户创建存储库或将其标记为 `key=environment,value=dev`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateTaggedRepository",
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/environment": "dev"
                }
            }
        },
        {
            "Sid": "AllowTagRepository",
            "Effect": "Allow",
            "Action": [
                "ecr:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/environment": "dev"
                }
            }
        }
    ]
```

```
}
```

以下策略将允许用户从所有存储库中提取图像，除非它们被标记为`key=environment,value=prod`。

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecr:ResourceTag/environment": "prod"
                }
            }
        }
    ]
}
```

Amazon Amazon 弹性容器注册表的托管策略

Amazon 托管策略是由创建和管理的独立策略 Amazon。 Amazon 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， Amazon 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 Amazon 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 Amazon 托管策略中定义的权限。如果 Amazon 更新 Amazon 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 Amazon 最有可能在启动新的 API 或现有服务可以使用新 Amazon Web Services 服务 的 API 操作时更新 Amazon 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 Amazon 托管策略。

Amazon ECR 提供了多个托管策略，您可以将这些策略附加到 IAM 身份或亚马逊 EC2 实例。借助这些策略，您可对 Amazon ECR 资源和 API 操作的访问权限进行不同级别的控制。有关这些策略中提到的每个 API 操作的更多信息，请参阅 Amazon Elastic Container Registry API 参考中的操作。

主题

- AmazonEC2ContainerRegistryFullAccess
- AmazonEC2ContainerRegistryPowerUser
- AmazonEC2ContainerRegistryPullOnly
- AmazonEC2ContainerRegistryReadOnly
- AWSECRPullThroughCache_ServiceRolePolicy
- ECRReplicationServiceRolePolicy
- ECRTemplateServiceRolePolicy
- Amazon ECR 更新了托 Amazon 管政策

`AmazonEC2ContainerRegistryFullAccess`

您可以将 `AmazonEC2ContainerRegistryFullAccess` 策略附加到 IAM 身份。

您可以使用此托管策略作为起点，根据您的具体要求创建自己的 IAM policy。例如，您可以创建一个策略，专门为用户或角色提供完全管理员访问权限，以管理 Amazon ECR 的使用。借助 Amazon ECR 生命周期策略功能，您可以指定存储库中镜像的生命周期管理。生命周期策略事件作为 CloudTrail 事件报告。Amazon ECR 已与集成 Amazon CloudTrail ，因此它可以直接在 Amazon ECR 控制台中显示您的生命周期策略事件。`AmazonEC2ContainerRegistryFullAccess` 托管 IAM policy 包含促进此行为的 `cloudtrail:LookupEvents` 权限。

权限详细信息

该策略包含以下权限：

- `ecr`— 允许委托人完全访问所有 Amazon EC APIs R。
- `cloudtrail`— 允许委托人查找由 CloudTrail捕获的管理事件或 Amazon CloudTrail Insights 事件。

`AmazonEC2ContainerRegistryFullAccess` 策略如下所示。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:*",
                "cloudtrail:LookupEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "replication.ecr.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

`AmazonEC2ContainerRegistryPowerUser`

您可以将 `AmazonEC2ContainerRegistryPowerUser` 策略附加到 IAM 身份。

此策略授予管理权限，以允许 IAM 用户读写存储库，但不允许他们删除存储库或更改应用于存储库的策略文档。

权限详细信息

该策略包含以下权限：

- `ecr`：允许主体读取和写入存储库，以及读取生命周期策略。委托人不会被授予删除存储库或更改应用于它们的生命周期策略的权限。

`AmazonEC2ContainerRegistryPowerUser` 策略如下所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage"
            ],
            "Resource": "*"
        }
    ]
}
```

AmazonEC2ContainerRegistryPullOnly

您可以将 `AmazonEC2ContainerRegistryPullOnly` 策略附加到 IAM 身份。

此策略授予从 Amazon ECR 提取容器映像的权限。如果注册表启用了缓存提取，则它还将允许从上游注册表导入映像的提取。

权限详细信息

此策略包含以下权限：

- `ecr` - 允许委托人读取存储库及其各自的生命周期策略。

`AmazonEC2ContainerRegistryPullOnly` 策略如下所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchImportUpstreamImage"
            ],
            "Resource": "*"
        }
    ]
}
```

`AmazonEC2ContainerRegistryReadOnly`

您可以将 `AmazonEC2ContainerRegistryReadOnly` 策略附加到 IAM 身份。

此策略授予 Amazon ECR 的只读权限。这包括列出存储库及其中镜像的功能。它还包括使用 Docker CLI 从 Amazon ECR 提取镜像的功能。

权限详细信息

此策略包含以下权限：

- `ecr` - 允许委托人读取存储库及其各自的生命周期策略。

`AmazonEC2ContainerRegistryReadOnly` 策略如下所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
```

```
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings"
            ],
            "Resource": "*"
        }
    ]
}
```

`AWSECRPullThroughCache_ServiceRolePolicy`

您不能将 `AWSECRPullThroughCache_ServiceRolePolicy` 托管的 IAM policy 附加到您的 IAM 实体。此策略附加到服务相关角色，该角色允许 Amazon ECR 通过缓存提取工作流将镜像推送到存储库。有关更多信息，请参阅 用于缓存提取的 Amazon ECR 服务相关角色。

`ECRReplicationServiceRolePolicy`

您不能将 `ECRReplicationServiceRolePolicy` 托管的 IAM policy 附加到您的 IAM 实体。此附加到服务相关角色的策略允许 Amazon ECR 代表您执行操作。有关更多信息，请参阅 对 Amazon ECR 使用服务相关角色。

`ECRTemplateServiceRolePolicy`

您不能将 `ECRTemplateServiceRolePolicy` 托管的 IAM policy 附加到您的 IAM 实体。此附加到服务相关角色的策略允许 Amazon ECR 代表您执行操作。有关更多信息，请参阅 对 Amazon ECR 使用服务相关角色。

Amazon ECR 更新了托 Amazon 管政策

查看自该服务开始跟踪这些更改以来对 Amazon ECR Amazon 托管政策更新的详细信息。有关此页面更改的自动提示，请订阅 Amazon ECR 文档历史记录页面上的 RSS 源。

更改	描述	日期
用于缓存提取的 Amazon ECR 服务相关角色 – 对现有策略的更新	Amazon ECR 将新权限添加到 `AWSECRPullThroughCache_ServiceRolePo`	2025 年 3 月 12 日

更改	描述	日期
	`licy` 策略。这些权限允许 Amazon ECR 从 ECR 私有注册表中提取映像。使用拉取缓存规则缓存来自其他 Amazon ECR 私有注册表的图像时，这是必需的。	
亚马逊 EC2 Container RegistryPullOnly-新政策	Amazon ECR 添加了一项新政策，向亚马逊 ECR 授予仅限拉取的权限。	2024 年 10 月 10 日
ECRTemplateServiceRolePolicy：新策略	Amazon ECR 添加了新策略。此策略与用于存储库创建模板功能的 `ECRTemplateServiceRolePolicy` 服务相关角色关联。	2024 年 6 月 20 日
AWSECRPullThroughCache_ServiceRolePolicy – 对现有策略的更新	Amazon ECR 将新权限添加到 `AWSECRPullThroughCache_ServiceRolePolicy` 策略。这些权限允许 Amazon ECR 检索 Secrets Manager 密钥的加密内容。当使用缓存提取规则来缓存需要身份验证的上游注册表中的映像时，需要此类权限。	2023 年 11 月 15 日
AWSECRPullThroughCache_ServiceRolePolicy：新策略	Amazon ECR 添加了新策略。此策略与用于缓存提取功能的 `AWSServiceRoleForECRPullThroughCache` 服务相关角色相关。	2021 年 11 月 29 日

更改	描述	日期
ECRReplicationServiceRolePolicy：新策略	Amazon ECR 添加了新策略。此策略与用于复制功能的 `AWSServiceRoleForECRReplication` 服务相关角色相关。	2020 年 12 月 4 日
亚马逊 EC2 Container RegistryFullAccess-对现有政策的更新	Amazon ECR 将新权限添加到 `AmazonEC2ContainerRegistryFullAccess` 策略。这些权限允许委托人创建 Amazon ECR 服务相关角色。	2020 年 12 月 4 日
亚马逊 EC2 Container RegistryReadOnly-对现有政策的更新	Amazon ECR 将新权限添加到 `AmazonEC2ContainerRegistryReadOnly` 策略，该策略允许委托人读取生命周期策略、列出标签以及描述镜像的扫描结果。	2019 年 12 月 10 日
亚马逊 EC2 Container RegistryPowerUser-对现有政策的更新	Amazon ECR 将新权限添加到 `AmazonEC2ContainerRegistryPowerUser` 策略。这些权限允许委托人读取生命周期策略、列出标签以及描述镜像的扫描结果。	2019 年 12 月 10 日
亚马逊 EC2 Container RegistryFullAccess-对现有政策的更新	Amazon ECR 将新权限添加到 `AmazonEC2ContainerRegistryFullAccess` 策略。它们允许校长查找由 CloudTrail捕获的管理事件或 Amazon CloudTrail Insights 事件。	2017 年 11 月 10 日

更改	描述	日期
亚马逊 EC2 Container RegistryReadOnly-对现有政策的更新	Amazon ECR 将新权限添加到 `AmazonEC2ContainerRegistryReadOnly` 策略。这些权限允许委托人描述 Amazon ECR 镜像。	2016 年 10 月 11 日
亚马逊 EC2 Container RegistryPowerUser-对现有政策的更新	Amazon ECR 将新权限添加到 `AmazonEC2ContainerRegistryPowerUser` 策略。这些权限允许委托人描述 Amazon ECR 镜像。	2016 年 10 月 11 日
亚马逊 EC2 Container RegistryReadOnly-新政策	Amazon ECR 添加了一项新政策，向亚马逊 ECR 授予只读权限。这些权限包括列出存储库及其中镜像的功能。它们还包括使用 Docker CLI 从 Amazon ECR 提取镜像的功能。	2015 年 12 月 21 日
亚马逊 EC2 Container RegistryPowerUser-新政策	Amazon ECR 添加了一项新政策，该策略授予管理权限，允许用户读取和写入存储库，但不允许他们删除存储库或更改应用于他们的策略文档。	2015 年 12 月 21 日
亚马逊 EC2 Container RegistryFullAccess-新政策	Amazon ECR 添加了新策略。此策略授予 Amazon ECR 的完全访问权限。	2015 年 12 月 21 日
Amazon ECR 开始跟踪更改	Amazon ECR 已开始跟踪 Amazon 托管策略的更改。	2021 年 6 月 24 日

对 Amazon ECR 使用服务相关角色

Amazon Elastic Container Registry (Amazon ECR) Amazon Identity and Access Management 使用 (IAM) 服务相关角色来提供使用复制和提取缓存功能所需的权限。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon ECR 直接相关。服务相关角色由 Amazon ECR 预定义。它包含该服务支持私有注册表的复制和缓存提取功能所需的所有权限。为注册表配置复制或缓存提取之后，系统会自动创建服务相关角色。有关更多信息，请参阅 Amazon ECR 中的私有注册表设置。

服务相关角色可让您更轻松地通过 Amazon ECR 设置复制和缓存提取。这是因为，使用该角色，您就不必手动添加所有必要的权限。Amazon ECR 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon ECR 可以代入该角色。定义的权限包括信任策略和权限策略。不能将该权限策略附加到任何其他 IAM 实体。

只有先禁用注册表上的复制或缓存提取之后，才能删除相应的服务相关角色。这可以确保您不会无意中删除 Amazon ECR 对这些功能所需的权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的Amazon 服务。在此链接页面上，查找在服务相关角色列中具有是值的服务。选择是与查看该服务的服务相关角色文档的链接。

主题

- Amazon ECR 服务相关角色支持的区域
- 用于复制的 Amazon ECR 服务相关角色
- 用于缓存提取的 Amazon ECR 服务相关角色
- 用于存储库创建模板的 Amazon ECR 服务相关角色

Amazon ECR 服务相关角色支持的区域

Amazon ECR 在提供 Amazon ECR 服务的所有区域支持使用服务相关角色。有关 Amazon ECR 区域可用性的更多信息，请参阅 Amazon 区域和端点。

用于复制的 Amazon ECR 服务相关角色

Amazon ECR 使用名为的服务相关角色 AWSServiceRoleForECRReplication，该角色允许 Amazon ECR 跨多个账户复制映像。

Amazon ECR 的服务相关角色权限

AWSServiceRoleForECRReplication 服务相关角色信任以下服务来代入该角色：

- `replication.ecr.amazonaws.com`

以下 `ECRReplicationServiceRolePolicy` 角色权限策略允许 Amazon ECR 对资源使用以下操作：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> `ReplicateImage` 是 Amazon ECR 用于复制的内部 API，不能直接调用。

必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅 IAM 用户指南中的服务相关角色权限。

为 Amazon ECR 创建服务相关角色

无需手动创建 Amazon ECR 服务相关角色。当您在 Amazon Web Services Management Console、或 Amazon API 中为注册表配置复制设置时 Amazon CLI，Amazon ECR 会为您创建服务相关角色。

如果您删除了此服务相关角色然后需要再次创建它，则可以使用相同的流程在您的账户中重新创建此角色。当您为注册表配置复制设置时，Amazon ECR 将再次为您创建服务相关角色。

为 Amazon ECR 编辑服务相关角色

Amazon ECR 不允许手动编辑 AWSServiceRoleForECRReplication 服务相关角色。在创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。不过，您可以使用 IAM 编辑角色的说明。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

为 Amazon ECR 删除服务相关角色

如果不再需要使用某个需要服务相关角色的特征或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须先删除每个区域中的注册表复制配置，才能手动删除服务相关角色。

> **Note**
>
> 如果您尝试删除资源，而 Amazon ECR 服务仍在使用角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟，然后重试。

删除使用的 Amazon ECR 资源 AWSService RoleFor ECRReplication

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择为其设置复制配置的区域。
3. 在导航窗格中，选择私有注册表。
4. 在 Private registry（私有注册表）页面上的 Replication configuration（复制配置）部分，选择 Edit（编辑）。
5. 要删除所有复制规则，请选择 Delete all（全部删除）。此步骤需要确认。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台 Amazon CLI、或 Amazon API 删除AWSServiceRoleForECRReplication服务相关角色。有关更多信息，请参见《IAM 用户指南》中的删除服务相关角色。

用于缓存提取的 Amazon ECR 服务相关角色

Amazon ECR 使用名为的服务相关角色 AWSServiceRoleForECRPullThroughCache，该角色允许 Amazon ECR 代表您执行操作以完成缓存提取操作。有关缓存提取的更多信息，请参阅用于控制在缓存提取或复制操作期间创建的存储库的模板。

Amazon ECR 的服务相关角色权限

AWSServiceRoleForECRPullThroughCache服务相关角色信任以下服务来代入该角色。

- `pullthroughcache.ecr.amazonaws.com`

权限详细信息

`AWSECRPullThroughCache_ServiceRolePolicy` 权限策略附加到服务相关角色。此托管策略授权 Amazon ECR 执行以下操作。有关更多信息，请参阅 [`AWSECRPullThroughCache_ServiceRolePolicy`]。

- `ecr`— 允许 Amazon ECR 服务将图像提取并推送到私有存储库。
- `secretsmanager:GetSecretValue`— 允许 Amazon ECR 服务检索 Amazon Secrets Manager 密钥的加密内容。当使用缓存提取规则在私有注册表中缓存需要身份验证的上游注册表中的映像时，需要此类权限。此权限仅适用于带 `ecr-pullthroughcache/` 名称前缀的密钥。

`AWSECRPullThroughCache_ServiceRolePolicy` 策略包含以下 JSON。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ECR",
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage",
                "ecr:BatchGetImage",
                "ecr:BatchImportUpstreamImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetImageCopyStatus"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManager",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:ecr-pullthroughcache/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
```

```
                }
            }
        }
    ]
}
```

必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon ECR 创建服务相关角色

无需手动为缓存提取创建 Amazon ECR 服务相关角色。当您在 Amazon Web Services Management Console、或 Amazon API 中为私有注册表创建直通缓存规则时 Amazon CLI，Amazon ECR 会为您创建服务相关角色。

如果您删除了此服务相关角色然后需要再次创建它，则可以使用相同的流程在您的账户中重新创建此角色。当您为私有注册表创建缓存提取规则时，如果服务相关角色尚不存在，则 Amazon ECR 将为您再次创建该角色。

为 Amazon ECR 编辑服务相关角色

Amazon ECR 不允许手动编辑AWSServiceRoleForECRPullThroughCache服务相关角色。在创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。不过，您可以使用 IAM 编辑角色的说明。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

为 Amazon ECR 删除服务相关角色

如果不再需要使用某个需要服务相关角色的特征或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须先删除每个区域中的注册表缓存提取规则，才能手动删除服务相关角色。

> ⓘ Note
>
> 如果您尝试删除资源，而 Amazon ECR 服务仍在使用角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟，然后重试。

要删除 AWSServiceRoleForECRPullThroughCache 服务相关角色所使用的 Amazon ECR 资源

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择创建缓存提取规则所在的区域。

3. 在导航窗格中，选择私有注册表。
4. 在 Private registry（私有注册表）页面上的 Pull through cache configuration（缓存提取配置）部分，选择 Edit（编辑）。
5. 对于您创建的每个缓存提取规则，选择该规则，然后选择 Delete rule（删除规则）。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台 Amazon CLI、或 Amazon API 删除AWSServiceRoleForECRPullThroughCache服务相关角色。有关更多信息，请参见《IAM 用户指南》中的删除服务相关角色。

用于存储库创建模板的 Amazon ECR 服务相关角色

Amazon ECR 使用名为的服务相关角色 AWSServiceRoleForECRTemplate，该角色授予 Amazon ECR 代表您执行操作以完成存储库创建模板操作的权限。

Amazon ECR 的服务相关角色权限

AWSServiceRoleForECRTemplate服务相关角色信任以下服务来代入该角色。

- `ecr.amazonaws.com`

权限详细信息

`ECRTemplateServiceRolePolicy` 权限策略附加到服务相关角色。此托管策略授予 Amazon ECR 代表您执行存储库创建操作的权限。

`ECRTemplateServiceRolePolicy` 策略包含以下 JSON。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
     "Sid": "CreateRepositoryWithTemplate",
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon ECR 创建服务相关角色

您无需为存储库创建模板手动创建 Amazon ECR 服务相关角色。当您在 Amazon Web Services Management Console、或 Amazon API 中为私有注册表创建存储库模板规则时 Amazon CLI，Amazon ECR 会为您创建服务相关角色。

如果您删除了此服务相关角色然后需要再次创建它，则可以使用相同的流程在您的账户中重新创建此角色。当您为私有注册表创建存储库创建规则时，如果服务相关角色尚不存在，则 Amazon ECR 将为您再次创建该角色。

为 Amazon ECR 编辑服务相关角色

Amazon ECR 不允许手动编辑AWSServiceRoleForECRTemplate服务相关角色。在创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。不过，您可以使用 IAM 编辑角色的说明。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

为 Amazon ECR 删除服务相关角色

如果不再需要使用某个需要服务相关角色的特征或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须先删除每个区域中的注册表创建规则，才能手动删除服务相关角色。

> Note
>
> 如果您尝试删除资源，而 Amazon ECR 服务仍在使用角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟，然后重试。

要删除 AWSServiceRoleForECRTemplate 服务相关角色所使用的 Amazon ECR 资源

1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
2. 从导航栏中，选择您创建存储库创建规则所在的区域。
3. 在导航窗格中，选择私有注册表。
4. 在私有注册表页面上的存储库创建模板部分，选择编辑。
5. 对于您创建的每个存储库创建规则，选择该规则，然后选择删除规则。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台 Amazon CLI、或 Amazon API 删除AWSServiceRoleForECRTemplate服务相关角色。有关更多信息，请参见《IAM 用户指南》中的删除服务相关角色。

排查 Amazon Elastic Container Registry 的身份和访问权限问题

可以使用以下信息，以帮助您诊断和修复在使用 Amazon ECR 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Amazon ECR 中执行操作
- 我无权执行 iam：PassRole
- 我想允许我以外的人访问我 Amazon Web Services 账户 的 Amazon ECR 资源

我无权在 Amazon ECR 中执行操作

如果您收到错误提示，指明您无权执行某个操作，则必须更新策略以允许执行该操作。

当 `mateojackson` IAM 用户尝试使用控制台查看有关虚构 *`my-example-widget`* 资源的详细信息，但不拥有虚构 `ecr:`*`GetWidget`* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 ecr:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 `mateojackson` 用户的策略，以允许使用 `ecr:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

如果您需要帮助，请联系您的 Amazon 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Amazon ECR。

有些 Amazon Web Services 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 Amazon ECR 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 Amazon 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人访问我 Amazon Web Services 账户 的 Amazon ECR 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon ECR 是否支持这些功能，请参阅 Amazon Elastic Container Registry 如何与 IAM 结合使用。
- 要了解如何提供对您拥有的资源的访问权限 Amazon Web Services 账户 ，请参阅 IAM 用户指南中的向您拥有 Amazon Web Services 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 Amazon Web Services 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 Amazon Web Services 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

Amazon ECR 中的数据保护

分担责任模型适用于亚马逊弹性容器服务中的数据保护。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，请参阅数据隐私常见问题。

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）。

- 使用 SSL/TLS 与资源通信。 Amazon 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》。

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 Amazon Web Services 服务 使用控制台、API 或与 Amazon ECS 或其他机构 Amazon CLI合作时 Amazon SDKs。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网址，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。

主题

- 静态加密

静态加密

Important

使用 Amazon KMS (DSSE-KMS) 的双层服务器端加密仅在区域中 Amazon GovCloud (US) 可用。

Amazon ECR 将镜像存储在 Amazon ECR 管理的 Amazon S3 存储桶中。默认情况下，Amazon ECR 使用具有 Amazon S3 托管加密密钥的服务器端加密，从而使用 AES-256 加密算法对静态数据进行加密。这不需要您采取任何行动，且不会另外收取费用。有关更多信息，请参阅 Amazon Simple Storage Service 用户指南中的借助使用 Amazon S3 托管式加密密钥的服务器端加密 (SSE-S3) 保护数据。

为了更好地控制 Amazon ECR 存储库的加密，您可以使用服务器端加密，并使用存储在 Amazon Key Management Service ()Amazon KMS中的 KMS 密钥。在使用 Amazon KMS 加密数据时，您可以使用由 Amazon ECR 管理的默认 Amazon 托管式密钥密钥，也可以指定自己的 KMS 密钥（称为客户托

管密钥）。有关更多信息，请参阅《亚马逊简单存储服务用户指南》中的使用服务器端加密和存储在 Amazon KMS (SSE-KMS) 中的 KMS 密钥保护数据。

您可以选择使用双层服务器端加密（ ）对您的 Amazon ECR 映像应用两层加 Amazon KMS 密。DSSE-KMS DSSE-KMS选项类似于SSE-KMS，但应用的是两个单独的加密层，而不是一层。有关更多信息，请参阅使用带 Amazon KMS 密钥的双层服务器端加密 (DSS E-KMS)。

每个 Amazon ECR 存储库都有一个加密配置，该配置在创建存储库时进行设置。您可以在每个存储库上使用不同的加密配置。有关更多信息，请参阅 创建 Amazon ECR 私有存储库以存储映像。

在启用 Amazon KMS 加密的情况下创建存储库时，将使用 KMS 密钥对存储库的内容进行加密。此外，Amazon ECR 还会向 KMS 密钥添加 Amazon KMS 授权，并将 Amazon ECR 存储库作为被授权者委托人。

以下内容提供了对 Amazon ECR 如何与 Amazon KMS 集成以加密和解密存储库的高层级理解：

1. 创建存储库时，Amazon ECR 会向发送DescribeKey调用， Amazon KMS 以验证和检索加密配置中指定的 KMS 密钥的亚马逊资源名称 (ARN)。
2. Amazon ECR 向发送了两个CreateGrant请求 Amazon KMS ，要求在 KMS 密钥上创建授权，以允许 Amazon ECR 使用数据密钥加密和解密数据。
3. 推送图像时，系统会向发出GenerateDataKey请求 Amazon KMS ，指定用于加密图像层和清单的 KMS 密钥。
4. Amazon KMS 生成新的数据密钥，使用指定的 KMS 密钥对其进行加密，然后发送要与图像层元数据和图像清单一起存储的加密数据密钥。
5. 拉取图像时，会向发出解密请求 Amazon KMS，指定加密的数据密钥。
6. Amazon KMS 解密加密的数据密钥并将解密后的数据密钥发送到 Amazon S3。
7. 数据密钥用于在提取镜像层之前对其进行解密。
8. 删除存储库后，Amazon ECR 会向发送两个RetireGrant请求，要求停 Amazon KMS 用为该存储库创建的授权。

注意事项

在 Amazon ECR 中使用 Amazon KMS 基于基础的加密（SSE-KMS或DSSE-KMS）时，应考虑以下几点。

- 如果您创建采用 KMS 加密的 Amazon ECR 存储库，但未指定 KMS 密钥，则 Amazon ECR 会默认使用 Amazon 托管式密钥 带有别名`aws/ecr`的。首次创建启用 KMS 加密的存储库时，在您的账户中创建此 KMS 密钥。

- 创建存储库后，将无法更改存储库加密配置。
- 当您将 KMS 加密与自己的 KMS 密钥结合使用时，密钥必须与您的存储库位于同一个区域中。
- Amazon ECR 代表您创建的授权不应被撤销。如果您撤销授予 Amazon ECR 使用您账户中 Amazon KMS 密钥的权限，Amazon ECR 将无法访问这些数据、加密推送到存储库的新映像，也无法在提取时对其进行解密。当您撤销 Amazon ECR 授权时，更改将立即生效。要撤销访问权限，则应删除存储库，而不是撤销该授权。删除存储库后，Amazon ECR 会代表您停用授权。
- 使用 Amazon KMS 密钥需要付费。有关更多信息，请参阅Amazon Key Management Service 定价。
- 使用双层服务器端加密将产生费用。有关更多信息，请参阅 Amazon ECR 定价

所需的 IAM 权限

创建或删除使用 Amazon KMS进行服务器端加密的 Amazon ECR 存储库时，所需的权限取决于您正在使用的特定 KMS 密钥。

使用适用于 Amazon ECR 时需要 Amazon 托管式密钥 的 IAM 权限

默认情况下，如果为 Amazon ECR 存储库启用了 Amazon KMS 加密，但未指定 KMS 密钥，则使用 Amazon 托管式密钥 适用于 Amazon ECR 的。当使用 Amazon ECR 的 Amazon托管 KMS 密钥加密存储库时，任何有权创建存储库的委托人也可以在存储库上启用 Amazon KMS 加密。但是，删除存储库的 IAM 委托人必须具有 `kms:RetireGrant` 权限。这样可以停用创建存储库时添加到 Amazon KMS 密钥中的授权。

以下示例 IAM policy 可作为内联策略添加到用户，以确保用户具有删除启用加密的存储库所需的最低权限。可以使用资源参数指定用于加密存储库的 KMS 密钥。

```
{
    "Version": "2012-10-17",
    "Id": "ecr-kms-permissions",
    "Statement": [
        {
            "Sid": "AllowAccessToRetireTheGrantsAssociatedWithTheKey",
            "Effect": "Allow",
            "Action": [
                "kms:RetireGrant"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/b8d9ae76-080c-4043-92EXAMPLE"
        }
```

```
    ]
}
```

使用客户托管密钥时所需的 IAM 权限

使用客户托管密钥创建启用 Amazon KMS 加密功能的存储库时，创建存储库的用户或角色必须拥有 KMS 密钥策略和 IAM 策略的权限。

创建自己的 KMS 密钥时，您可以使用原定设置密钥策略 Amazon KMS 创建，也可以指定自己的密钥策略。为确保账户所有者可以管理客户托管的密钥，KMS 密钥的密钥策略应允许账户的根用户 Amazon KMS 执行所有操作。可以向密钥策略添加额外的作用域权限，但至少应向根用户授予管理 KMS 密钥的权限。要允许 KMS 密钥仅用于源自 Amazon ECR 的请求，您可以将 kms: ViaService 条件密钥与值一起使用。`ecr.<region>.amazonaws.com`

以下示例密钥策略为拥有 KMS 密钥的 Amazon 账户（根用户）提供了对 KMS 密钥的完全访问权限。有关此示例密钥策略的更多信息，请参阅Amazon Key Management Service 开发人员指南中的允许访问 Amazon 账户和启用 IAM 策略。

```
{
    "Version": "2012-10-17",
    "Id": "ecr-key-policy",
    "Statement": [
        {
            "Sid": "EnableIAMUserPermissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        }
    ]
}
```

除了必要的 Amazon ECR 权限外，创建存储库的 IAM 用户`kms:CreateGrantkms:RetireGrant`、IAM 角色或 Amazon 账户还必须拥有、和`kms:DescribeKey`权限。

> **Note**
>
> `kms:RetireGrant` 权限必须添加到创建存储库的用户或角色的 IAM policy 中。`kms:CreateGrant` 和 `kms:DescribeKey` 权限可以添加到 KMS 密钥的密钥策略或创建存储库的用户或角色的 IAM policy 中。有关 Amazon KMS 权限工作原理的更多信息，请参阅《Amazon Key Management Service 开发者指南》中的 Amazon KMS API 权限：操作和资源参考。

以下示例 IAM policy 可作为内联策略添加到用户，以确保用户拥有创建启用加密的存储库所需的最低权限，并在完成存储库时删除存储库。可以使用资源参数指定用于加密存储库的 Amazon KMS key 。

```
{
    "Version": "2012-10-17",
    "Id": "ecr-kms-permissions",
    "Statement": [
        {
            "Sid":
 "AllowAccessToCreateAndRetireTheGrantsAssociatedWithTheKeyAsWellAsDescribeTheKey",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:RetireGrant",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/b8d9ae76-080c-4043-92EXAMPLE"
        }
    ]
}
```

创建存储库时，允许用户在控制台中列出 KMS 密钥

使用 Amazon ECR 控制台创建存储库时，您可以授予权限，允许用户在启用存储库加密时于区域中列出客户托管的 KMS 密钥。以下 IAM policy 示例显示了使用控制台时列出 KMS 密钥和别名所需的权限。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
```

```
    "Action": [
      "kms:ListKeys",
      "kms:ListAliases",
      "kms:DescribeKey"
    ],
    "Resource": "*"
  }
}
```

监控 Amazon ECR 与 Amazon KMS的集成

您可以使用 Amazon CloudTrail 来跟踪 Amazon ECR 代表您发送 Amazon KMS 的请求。日志中的 CloudTrail 日志条目包含加密上下文密钥，便于识别。

Amazon ECR 加密上下文

加密上下文 是一组包含任意非机密数据的键值对。当您在加密数据的请求中包含加密上下文时，会以加密 Amazon KMS 方式将加密上下文绑定到加密数据。要解密数据，您必须传入相同的加密上下文。

在对的请求GenerateDataKey和解密请求中， Amazon KMS Amazon ECR 使用具有两个名称值对的加密上下文，用于标识存储库和正在使用的 Amazon S3 存储桶。如以下示例所示。名称不会变化，但与其组合的加密上下文会因每个值而异。

```
"encryptionContext": {
    "aws:s3:arn": "arn:aws:s3:::us-west-2-starport-manifest-bucket/EXAMPLE1-90ab-cdef-fedc-ba987BUCKET1/sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df",
    "aws:ecr:arn": "arn:aws:ecr:us-west-2:111122223333:repository/repository-name"
}
```

您可以使用加密上下文在审计记录和日志（例如和 Amazon CloudWatch Logs）中识别这些加密操作，并作为策略和授权中的授权条件。Amazon CloudTrail

Amazon ECR 加密上下文包含两个名称–值对。

- aws:s3:arn – 第一个名称 - 值对标识存储桶。键是 `aws:s3:arn`。值是 Amazon S3 存储桶的 Amazon Resource Name (ARN)。

  ```
  "aws:s3:arn": "ARN of an Amazon S3 bucket"
  ```

例如，如果存储桶的 ARN 是 `arn:aws:s3:::`*`us-west-2`*`-starport-manifest-bucket/`*`EXAMPLE1-90ab-cdef-fedc-ba987BUCKET1`*`/sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df`，加密上下文将包括以下对。

```
"arn:aws:s3:::us-west-2-starport-manifest-bucket/EXAMPLE1-90ab-cdef-fedc-
ba987BUCKET1/sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df"
```

- aws:ecr:arn – 第二个名称 - 值对标识存储库的 Amazon Resource Name (ARN)。键是 `aws:ecr:arn`。值是存储库的 ARN。

```
"aws:ecr:arn": "ARN of an Amazon ECR repository"
```

例如，如果储存库的 ARN 是 `arn:aws:ecr:`*`us-west-2`*`:`*`111122223333`*`:repository/`*`repository-name`*，加密上下文将包括以下对。

```
"aws:ecr:arn": "arn:aws:ecr:us-west-2:111122223333:repository/repository-name"
```

故障排除

使用控制台删除 Amazon ECR 存储库时，如果存储库已成功删除，但 Amazon ECR 无法停用添加到存储库 KMS 密钥的授权，您将收到以下错误消息。

```
The repository [{repository-name}] has been deleted successfully but the grants created
 by the kmsKey [{kms_key}] failed to be retired
```

发生这种情况时，您可以自己取消对存储库的 Amazon KMS 授权。

手动取消对存储库的 Amazon KMS 授权

1. 列出用于存储库的 Amazon KMS 密钥的授权。`key-id` 值包含在您从控制台收到的错误中。您还可以使用`list-keys`命令列出账户中特定区域中的客户托管的 KMS 密钥 Amazon 托管式密钥 和客户托管的 KMS 密钥。

```
aws kms list-grants \
     --key-id b8d9ae76-080c-4043-9237-c815bfc21dfc
     --region us-west-2
```

输出包括 `EncryptionContextSubset` 以及存储库的 Amazon Resource Name (ARN)。这可用于确定添加到密钥中的哪个授权是您想要停用的授权。`GrantId` 值将在下一步中停用授权时使用。

2. 取消为存储库添加的 Amazon KMS 密钥的每项授权。将的值替换为*`GrantId`*上一步输出中的授权 ID。

```
aws kms retire-grant \
     --key-id b8d9ae76-080c-4043-9237-c815bfc21dfc \
     --grant-id GrantId \
     --region us-west-2
```

Amazon Elastic Container Registry 的合规性验证

要了解是否属于特定合规计划的范围，请参阅Amazon Web Services 服务 “” Amazon Web Services 服务 中的 “按合规计划划分的范围”，然后选择您感兴趣的合规计划。 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 Web Services 服务 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：

- Security & Compliance：这些解决方案实施指南讨论了架构考虑因素，并提供了部署安全性和合规性功能的步骤。
- 合规资源 — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- 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— 这 Amazon Web Services 服务 可以全面了解您的安全状态 Amazon。Security Hub 通过安全控制措施评估您的 Amazon 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控制措施的列表，请参阅 Security Hub 控制措施参考。
- Amazon GuardDuty — 它通过监控您的 Amazon Web Services 账户环境中是否存在可疑和恶意活动，来 Amazon Web Services 服务 检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。

Amazon Elastic Registry 中的基础设施安全性

作为一项托管服务，Amazon Elastic Container Registry 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 Framework 中的基础设施保护。

您可以使用 Amazon 已发布的 API 调用通过网络访问 Amazon ECR。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密（PFS）的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 Amazon Security Token Service（Amazon STS）生成临时安全凭证来对请求进行签名。

您可以从任何网络位置调用这些 API 操作，但 Amazon ECR 不支持基于资源的访问策略，其中可以包含基于源 IP 地址的限制。您还可以使用 Amazon ECR 策略来控制来自特定亚马逊虚拟私有云（亚马逊 VPC）终端节点或特定 VPCs终端节点的访问。实际上，这可以将对给定 Amazon ECR 资源的网络访问与网络中的特定 VPC 隔离开来。 Amazon 有关更多信息，请参阅 Amazon ECR 接口 VPC 终端节点 ()Amazon PrivateLink。

Amazon ECR 接口 VPC 终端节点 ()Amazon PrivateLink

您可以将 Amazon ECR 配置为使用接口 VPC 终端节点以改善 VPC 的安全状况。VPC 终端节点由一项技术提供支持 Amazon PrivateLink，该技术使您能够 APIs 通过私有 IP 地址私密访问 Amazon ECR。 Amazon PrivateLink 将您的 VPC 和 Amazon ECR 之间的所有网络流量限制到亚马逊网络。您无需互联网网关、NAT 设备或虚拟私有网关。

有关 Amazon PrivateLink 和 VPC 终端节点的更多信息，请参阅 Amazon VPC 用户指南中的 VPC 终端节点。

Amazon ECR VPC 终端节点的注意事项

在为 Amazon ECR 配置 VPC 终端节点之前，请注意以下事项：

- 要允许托管在亚马逊 EC2 实例上的 Amazon ECS 任务从 Amazon ECR 提取私有镜像，请为亚马逊 ECS 创建接口 VPC 终端节点。有关更多信息，请参阅《亚马逊弹性容器服务开发者指南》中的接口 VPC 终端节点 (Amazon PrivateLink)。

- 从 Amazon ECR 提取容器镜像的 Amazon ECS 任务（在 Fargate 上托管）可以通过向其任务的任务执行 IAM 角色添加条件键，限制对其任务使用的特定 VPC 和服务使用的 VPC 终端节点的访问。有关更多信息，请参阅 Amazon Elastic Container Service 开发人员指南中的用于通过接口终端节点提取 Amazon ECR 镜像的 Fargate 任务的可选 IAM 权限。
- 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。
- VPC 端点当前不支持跨区域请求。确保在计划向 Amazon ECR 发出 API 调用的同一区域中创建 VPC 终端节点。
- VPC 端点目前不支持 Amazon ECR 公有存储库。考虑使用缓存提取规则将公有映像托管在与 VPC 端点位于同一区域的私有存储库中。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步。
- VPC 终端节点仅支持通过 Amazon Route 53 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。
- 如果您的容器具有与 Amazon S3 的现有连接，则在添加 Amazon S3 网关端点时，其连接可能会短暂中断。要避免此中断，请创建一个使用 Amazon S3 网关端点的新 VPC，然后将 Amazon ECS 集群及其容器迁移到该新 VPC。
- 首次使用缓存提取规则提取镜像时，如果您已将 Amazon ECR 配置为通过 Amazon PrivateLink 使用接口 VPC 终端节点，您需要在同一个 VPC 中创建一个带有 NAT 网关的公有子网，然后将至互联网的所有出站流量从私有子网路由到 NAT 网关，才能使提取操作正常工作。后续的镜像提取不需要此操作。有关更多信息，请参阅 Amazon Virtual Private Cloud 用户指南中的场景：从私有子网访问互联网。

Windows 镜像注意事项

基于 Windows 操作系统的镜像包括受许可证限制而无法分发的构件。预设情况下，当您将 Windows 镜像推送到 Amazon ECR 存储库时，包含这些构件的层不会被推送，因为它们被视为外部层。当构件由 Microsoft 提供时，将从 Microsoft Azure 基础设施中检索外部层。因此，要使容器能够从 Azure 中提取这些外部层，除了创建 VPC 终端节点之外，还需要执行其他步骤。

当您将 Windows 镜像推送到 Amazon ECR 时，可以通过使用 Docker 守护进程中的 `--allow-nondistributable-artifacts` 标记覆盖此行为。启用后，此标记会将许可层推送到 Amazon ECR，从而使这些镜像可以通过 VPC 终端节点从 Amazon ECR 提取，而无需额外访问 Azure。

> ⚠ Important
>
> 使用 `--allow-nondistributable-artifacts` 标记不会排除您遵守 Windows 容器基础镜像许可证条款的义务；您不能发布 Windows 内容以供公共或第三方重新分发。允许在您自己的环境中使用。

要在 Docker 安装中启用此标记，您必须修改 Docker 守护进程配置文件，根据您的 Docker 安装，该配置文件通常可以在 Docker Engine 部分下的设置或首选项菜单中配置，也可以通过直接编辑 `C:\ProgramData\docker\config\daemon.json` 文件配置。

以下是所需配置的示例。将值替换为要将镜像推送到其中的存储库 URI。

```
{
    "allow-nondistributable-artifacts": [
        "111122223333.dkr.ecr.us-west-2.amazonaws.com"
    ]
}
```

修改 Docker 守护进程配置文件后，您必须在尝试推送镜像之前重新启动 Docker 守护进程。通过验证基础层是否推送到您的存储库，确认推送工作运转正常。

> ⓘ Note
>
> Windows 镜像的基础层很庞大。层太大将导致延长推送时间，并在 Amazon ECR 中增加这些镜像的存储成本。出于这些原因，我们建议仅在严格要求此选项以减少构建时间和持续存储成本时使用此选项。例如，在 Amazon ECR 中压缩之后，`mcr.microsoft.com/windows/servercore` 镜像的大小约为 1.7 GiB。

为 Amazon ECR 创建 VPC 终端节点

要为 Amazon ECR 服务创建 VPC 终端节点，请使用 Amazon VPC 用户指南中的创建接口终端节点过程。

托管在亚马逊 EC2 实例上的 Amazon ECS 任务需要同时使用 Amazon ECR 终端节点和 Amazon S3 网关终端节点。

使用平台版本 `1.4.0` 或更高版本在 Fargate 上托管的 Amazon ECS 任务需要 Amazon ECR VPC 终端节点和 Amazon S3 网关终端节点。

使用平台版本 `1.3.0` 或更早版本在 Fargate 上托管的 Amazon ECS 任务只需要 cn.com.amazonaws.cn-northwest-1.ecr.dkr Amazon ECR VPC 终端节点和 Amazon S3 网关终端节点。

> **Note**
>
> 创建终端节点的顺序无关紧要。

cn.com.amazonaws.cn-northwest-1.ecr.dkr

此端点用于 Docker 注册表 APIs。诸如 `push` 和 `pull` 这样的 Docker 客户端命令使用此终端节点。

在创建此终端节点时，您必须启用私有 DNS 主机名。为此，请确保在创建 VPC 终端节点时，在 Amazon VPC 控制台中选择启用私有 DNS 名称选项。

cn.com.amazonaws.cn-northwest-1.ecr.api

此终端节点用于对 Amazon ECR API 执行的调用。API 操作（如 `DescribeImages` 和 `CreateRepository`）转到此终端节点。

在创建此终端节点时，您可以选择启用私有 DNS 主机名。在创建 VPC 终端节点时，通过在 VPC 控制台中选择启用私有 DNS 名称，可启用此设置。如果您为 VPC 终端节点启用私有 DNS 主机名，请将您的 SDK 或更新 Amazon CLI 到最新版本，以便在使用 SDK 或 Amazon CLI 时无需指定终端节点 URL。

如果您启用私有 DNS 主机名并且使用的是 2019 年 1 月 24 日之前发布的 SDK 或 2019 年 1 月 24 日之前发布的 Amazon CLI 版本，则必须使用`--endpoint-url`参数来指定接口终端节点。以下示例显示了终端节点 URL 的格式。

```
aws ecr create-repository --repository-name name --endpoint-url https://
api.ecr.region.amazonaws.com
```

如果您不为 VPC 终端节点启用私有 DNS 主机名，则必须使用 `--endpoint-url` 参数并指定接口终端节点的 VPC 终端节点 ID。以下示例显示了终端节点 URL 的格式。

```
aws ecr create-repository --repository-name name --endpoint-url
 https://VPC_endpoint_ID.api.ecr.region.vpce.amazonaws.com
```

创建 Amazon S3 网关终端节点

对于从 Amazon ECR 提取私有镜像的 Amazon ECS 任务，您必须为 Amazon S3 创建网关终端节点。必须创建网关终端节点，因为 Amazon ECR 使用 Amazon S3 来存储您的镜像层。当容器从 Amazon ECR 下载镜像时，它们必须访问 Amazon ECR 才能获取镜像清单，然后 Amazon S3 才能下载实际镜像层。以下是包含每个 Docker 镜像层的 Amazon S3 存储桶的 Amazon Resource Name (ARN)。

```
arn:aws:s3:::prod-region-starport-layer-bucket/*
```

使用 Amazon VPC 用户指南中的创建网关终端节点过程为 Amazon ECR 创建以下 Amazon S3 网关终端节点。在创建终端节点时，请务必为您的 VPC 选择路由表。

com.amazonaws。 ***region***.s3

> Amazon S3 网关终端节点使用 IAM policy 文档来限制对服务的访问。可以使用完全访问权限策略，因为您在任务 IAM 角色或其他 IAM 用户策略中设置的任何限制仍然适用于此策略。如果要将 Amazon S3 存储桶访问权限限制为使用 Amazon ECR 所需的最低权限，请参阅 Amazon ECR 的 Amazon S3 存储桶最低权限。

Amazon ECR 的 Amazon S3 存储桶最低权限

Amazon S3 网关终端节点使用 IAM policy 文档来限制对服务的访问。要仅允许 Amazon ECR 的 Amazon S3 存储桶最低权限，请在为终端节点创建 IAM policy 文档时，限制对 Amazon ECR 使用的 Amazon S3 存储桶的访问权限。

下表描述了 Amazon ECR 所需的 Amazon S3 存储桶策略权限。

权限	描述
`arn:aws:s3:::prod-region-starport-layer-bucket/*` `arn:aws-cn:s3:::prod-region-starport-layer-bucket/*`	提供对包含每个 Docker 镜像层的 Amazon S3 存储桶的访问权限。表示 Amazon ECR 支持的 Amazon 区域的区域标识符，例如美国东部（俄亥俄）区域的 `us-east-2` 。

示例

以下示例说明如何提供对 Amazon ECR 操作所需的 Amazon S3 存储桶的访问权限。

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::prod-region-starport-layer-bucket/*"]
    }
  ]
}
```

创建 CloudWatch 日志端点

使用 Fargate 启动类型的 Amazon ECS 任务需要你创建 com.amazonaws，这些任务使用没有互联网网关的 VPC，也使用**awslogs** CloudWatch 日志驱动程序向日志发送日志信息。 ***region***.logs 接口 VPC CloudWatch 日志终端节点。有关更多信息，请参阅 Amazon CloudWatch Logs 用户指南中的将 CloudWatch 日志与接口 VPC 终端节点配合使用。

为 Amazon ECR VPC 端点创建终端节点策略

VPC 端点策略是一种 IAM 资源策略，您在创建或修改端点时可将它附加到端点。如果您在创建终端节点时未附加策略， Amazon 则会为您附加允许完全访问服务的默认策略。端点策略不会覆盖或替换用户策略或服务特定的策略。这是一个单独的策略，用于控制从端点中对指定服务进行的访问。端点策略必须采用 JSON 格式编写。有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问权限。

我们建议您创建一个 IAM 资源策略，并将该策略同时附加到两个 Amazon ECR VPC 终端节点。

下面是用于 Amazon ECR 的端点策略示例。此策略允许特定 IAM 角色从 Amazon ECR 中提取镜像。

```
{
 "Statement": [{
  "Sid": "AllowPull",
  "Principal": {
   "AWS": "arn:aws:iam::1234567890:role/role_name"
  },
  "Action": [
   "ecr:BatchGetImage",
```

```
    "ecr:GetDownloadUrlForLayer",
                     "ecr:GetAuthorizationToken"
   ],
   "Effect": "Allow",
   "Resource": "*"
  }]
}
```

下面的终端节点策略示例阻止删除指定的存储库。

```
{
 "Statement": [{
   "Sid": "AllowAll",
   "Principal": "*",
   "Action": "*",
   "Effect": "Allow",
   "Resource": "*"
  },
  {
   "Sid": "PreventDelete",
   "Principal": "*",
   "Action": "ecr:DeleteRepository",
   "Effect": "Deny",
   "Resource": "arn:aws:ecr:region:1234567890:repository/repository_name"
  }
 ]
}
```

下面的终端节点策略示例将前面的两个示例组合到一个策略中。

```
{
 "Statement": [{
   "Sid": "AllowAll",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "*",
   "Resource": "*"
  },
  {
   "Sid": "PreventDelete",
   "Effect": "Deny",
   "Principal": "*",
   "Action": "ecr:DeleteRepository",
```

```
   "Resource": "arn:aws:ecr:region:1234567890:repository/repository_name"
  },
  {
   "Sid": "AllowPull",
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::1234567890:role/role_name"
   },
   "Action": [
    "ecr:BatchGetImage",
    "ecr:GetDownloadUrlForLayer",
                          "ecr:GetAuthorizationToken"
   ],
   "Resource": "*"
  }
 ]
}
```

修改 Amazon ECR 的 VPC 终端节点策略

1. 打开位于 https://console.aws.amazon.com/vpc/ 的 Amazon VPC 控制台。
2. 在导航窗格中，选择端点。
3. 如果您没有为 Amazon ECR 创建 VPC 终端节点，请参阅 为 Amazon ECR 创建 VPC 终端节点。
4. 选择要将策略添加到的 Amazon ECR VPC 终端节点，然后在屏幕的下半部分中选择策略选项卡。
5. 选择编辑策略并对策略进行更改。
6. 选择保存以保存策略。

共享子网

您无法在与您共享的子网中创建、描述、修改或删除 VPC 端点。但是，您可以在与您共享的子网中使用 VPC 端点。

防止跨服务混淆座席

混淆代理问题是一个安全性问题，即不具有操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在中 Amazon，跨服务模仿可能会导致混乱的副手问题。一个服务（呼叫服务）调用另一项服务（所谓的服务）时，可能会发生跨服务模拟。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况， Amazon 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务主体有权限访问账户中的资源。

我们建议在资源策略中使用 `aws:SourceArn` 或 `aws:SourceAccount` 全局条件上下文键，以限制 Amazon ECR 为其他服务提供的资源访问权限。如果您只希望将一个资源与跨服务访问相关联，请使用 `aws:SourceArn`。如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 `aws:SourceAccount`。

防范混淆代理问题最有效的方法是使用 `aws:SourceArn` 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符 (*) 的 `aws:SourceArn` 全局上下文条件键。例如 `arn:aws:servicename:region:123456789012:*`。

如果 `aws:SourceArn` 值不包含账户 ID，例如 Amazon S3 存储桶 ARN，您必须使用两个全局条件上下文键来限制权限。

`aws:SourceArn` 的值必须为 ResourceDescription。

以下示例展示了如何使用 Amazon ECR 存储库策略中的`aws:SourceArn`和`aws:SourceAccount`全局条件上下文密钥来允许 Amazon CodeBuild 访问与该服务集成所必需的 Amazon ECR API 操作，同时还可以防止出现混乱的代理问题。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CodeBuildAccess",
         "Effect":"Allow",
         "Principal":{
            "Service":"codebuild.amazonaws.com"
         },
         "Action":[
            "ecr:BatchGetImage",
            "ecr:GetDownloadUrlForLayer"
         ],
         "Condition":{
            "ArnLike":{
               "aws:SourceArn":"arn:aws:codebuild:region:123456789012:project/project-
name"
            },
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            }
         }
      }
   ]
```

```
}
```

Amazon ECR 监控

您可以通过亚马逊监控您的 Amazon ECR API 使用情况 CloudWatch，亚马逊会收集来自亚马逊 ECR 的原始数据并将其处理为可读的、近乎实时的指标。这些统计数据会保存两周，以便您访问历史信息并更好地了解 API 使用情况。Amazon ECR 指标数据将在一分钟 CloudWatch 内自动发送到。有关的更多信息 CloudWatch，请参阅 Amazon CloudWatch 用户指南。

Amazon ECR 基于您的 API 用量提供指标，这些指标适用于授权、镜像推送和镜像提取操作。

监控是维护 Amazon ECR 和您的 Amazon 解决方案的可靠性、可用性和性能的重要组成部分。我们建议您从构成 Amazon 解决方案的资源中收集监控数据，以便在出现多点故障时可以更轻松地进行调试。不过，在开始监控 Amazon ECR 之前，您应制定一个监控计划并在计划中回答下列问题：

- 监控目的是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您将使用哪些监控工具？
- 谁负责执行监控任务？
- 出现错误时应通知谁？

下一步，通过在不同时间和不同负载条件下测量性能，在您的环境中建立正常 Amazon ECR 性能的基准。在监控 Amazon ECR 时，存储历史监控数据，以便将此数据与新的性能数据进行比较，确定正常性能模式和性能异常，并设计解决问题的方法。

主题

可视化 Service Quotas 并设置警报

您可以使用 CloudWatch 控制台直观显示您的服务配额，并查看您的当前使用量与服务配额的对比情况。还可以设置警报，从而在接近配额时向您发送通知。

可视化服务配额并选择性地设置警报

1. 打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择指标。
3. 在所有指标选项卡上，选择用量，然后选择按 Amazon 资源。

   这将显示服务配额用量指标的列表。
4. 选中其中一个指标旁边的复选框。

   该图表显示您当前对该 Amazon 资源的使用情况。
5. 要将服务配额添加到图表，请执行以下操作：
   a. 选择 Graphed metrics（绘制的指标）选项卡。
   b. 选择数学表达式、从空表达式开始。然后在新行中，在详细信息下，输入 **`SERVICE_QUOTA(m1)`**。

      这将向图表中添加一个新行，并显示指标中表示的资源的服务配额。
6. 要以配额百分比的形式查看您的当前用量，请添加新的表达式或更改当前 SERVICE_QUOTA 表达式。对于新的表达式，请使用 **`m1/60/SERVICE_QUOTA(m1)*100`**。
7. (可选) 要设置一个警报，以便在接近服务配额时向您发送通知，请执行以下操作：
   a. 在 **`m1/60/SERVICE_QUOTA(m1)*100`** 行上的操作下，选择警报图标。该图标看起来像一个铃铛。

      这将显示警报创建页面。
   b. 在条件下，确保阈值类型为静态，并将当 Expression1 为设置为大于。在多于下，输入 **`80`**。这将创建一个警报，当用量超过配额的 80% 时，该警报将进入 ALARM 状态。
   c. 选择下一步。
   d. 在下一页上，选择一个 Amazon SNS 主题或创建一个新主题。当警报进入 ALARM 状态时，会向此主题发送通知。然后选择下一步。
   e. 在下一页上，输入警报的名称和描述，然后选择下一步。
   f. 选择创建警报。

Amazon ECR 用量指标

您可以使用 CloudWatch 使用量指标来了解您的账户的资源使用情况。使用这些指标在 CloudWatch 图表和仪表板上可视化您当前的服务使用情况。

Amazon ECR 使用率指标与 Amazon 服务配额相对应。您可以配置警报，以在用量接近服务配额时向您发出警报。有关 Amazon ECR 默认服务配额的更多信息，请参阅 Amazon ECR 服务配额。

Amazon ECR 在 `Amazon/Usage` 命名空间中发布以下指标。

指标	描述
`CallCount`	从您的账户调用 API 操作的次数。资源由与指标关联的维度定义。 此指标最有用的统计数据是 `SUM`，表示定义时段内来自所有贡献者的值的总和。

以下维度用于优化由 Amazon ECR 发布的用量指标。

维度	描述
`Service`	包含资源的 Amazon 服务的名称。对于 Amazon ECR 用量指标，此维度的值为 `ECR`。
`Type`	正在报告的实体的类型。目前，Amazon ECR 用量指标的唯一有效值为 `API`。
`Resource`	正在运行的资源的类型。目前，Amazon ECR 会返回有关以下 API 操作的 API 用量的信息。 • `GetAuthorizationToken` • `BatchCheckLayerAvailability` • `InitiateLayerUpload` • `UploadLayerPart` • `CompleteLayerUpload` • `PutImage`

维度	描述
	- `BatchGetImage` - `GetDownloadUrlForLayer`
`Class`	所跟踪的资源的类。Amazon ECR 目前不使用类维度。

Amazon ECR 用量报告

Amazon 提供名为 Cost Explorer 的免费报告工具，使您能够分析 Amazon ECR 资源的成本和使用情况。

使用 Cost Explorer 查看用量和成本的图表。您可以查看之前 13 个月的数据，并预测您在接下来三个月内可能产生的费用。您可以使用 Cost Explorer 查看有关您一段时间内在 Amazon 资源方面的费用的模式、确定需要进一步查询的方面以及查看可用于了解您的成本的趋势。您还可以指定数据的时间范围，并按天或按月查看时间数据。

成本和用量报告中的计量数据显示跨所有 Amazon ECR 存储库的用量。有关更多信息，请参阅 标记资源以便于计费。

有关创建 Amazon 成本和使用情况报告的更多信息，请参阅《Amazon Billing 用户指南》中的Amazon 成本和使用情况报告。

Amazon ECR 存储库指标

Amazon ECR 向亚马逊 CloudWatch发送存储库提取计数指标。Amazon ECR 指标数据会 CloudWatch 在 1 分钟内自动发送到。有关的更多信息 CloudWatch，请参阅 Amazon CloudWatch 用户指南。

主题

- 启用 CloudWatch 指标
- 可用指标和维度
- 使用控制台查看 Amazon ECR 指标 CloudWatch

启用 CloudWatch 指标

Amazon ECR 自动发送所有存储库的存储库指标。无需执行任何手动步骤。

可用指标和维度

以下部分列出了 Amazon ECR 发送给亚马逊 CloudWatch的指标和维度。

Amazon ECR 指标

Amazon ECR 提供了可用于监控存储库的指标。您可以测量拉取计数。

`AWS/ECR` 命名空间包括以下指标。

`RepositoryPullCount`

对存储库中映像的总拉取次数。

有效维度：`RepositoryName`。

有效统计数据：平均值、最小值、最大值、总计和样本数。最有用的统计指标是和。

单位：整数。

Amazon ECR 指标的维度

Amazon ECR 指标使用 `AWS/ECR` 命名空间并提供以下维度的指标。

`RepositoryName`

此维度将筛选您为指定存储库中所有容器器映像请求的数据。

使用控制台查看 Amazon ECR 指标 CloudWatch

您可以在 CloudWatch 控制台上查看 Amazon ECR 存储库指标。 CloudWatch 控制台可对您的资源进行精细且可自定义的显示。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

亚马逊 ECR 事件和 EventBridge

Amazon EventBridge 使您能够实现 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 服务的事件几乎是实时的。 EventBridge 您可以编写简单规则来指示您关注的事件，并包括要在事件匹配规则时执行的自动化操作。可自动触发的操作包括：

- 将事件添加到日志中的 CloudWatch 日志组
- 调用函数 Amazon Lambda

- 调用 Amazon EC2 运行命令
- 将事件中继到 Amazon Kinesis Data Streams
- 激活 Amazon Step Functions 状态机
- 通知 Amazon SNS 主题或 Amazon SQS 队列

有关更多信息，请参阅《亚马逊 EventBridge 用户指南》 EventBridge中的 “亚马逊入门”。

来自 Amazon ECR 的示例事件

以下是来自 Amazon ECR 的示例事件。尽最大努力发出事件。

已完成镜像推送的事件

每个镜像推送完成后，将发送以下事件。有关更多信息，请参阅 将 Docker 映像推送到 Amazon ECR 私有存储库。

```
{
    "version": "0",
    "id": "13cde686-328b-6117-af20-0e5566167482",
    "detail-type": "ECR Image Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-11-16T01:54:34Z",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "result": "SUCCESS",
        "repository-name": "my-repository-name",
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "action-type": "PUSH",
        "image-tag": "latest"
    }
}
```

缓存提取操作的事件

尝试缓存提取操作时，将发送以下事件。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步。

```
{
```

```
    "version": "0",
    "id": "85fc3613-e913-7fc4-a80c-a3753e4aa9ae",
    "detail-type": "ECR Pull Through Cache Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2023-02-29T02:36:48Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecr:us-west-2:123456789012:repository/docker-hub/alpine"
    ],
    "detail": {
        "rule-version": "1",
        "sync-status": "SUCCESS",
        "ecr-repository-prefix": "docker-hub",
        "repository-name": "docker-hub/alpine",
        "upstream-registry-url": "public.ecr.aws",
        "image-tag": "3.17.2",
        "image-digest":
 "sha256:4aa08ef415aecc80814cb42fa41b658480779d80c77ab15EXAMPLE",
    }
}
```

已完成镜像扫描的事件（基本扫描）

为注册表启用基本扫描后，当每个镜像扫描完成时，会发送以下事件。`finding-severity-counts` 参数仅返回严重性级别的值 (如果存在)。例如，如果镜像不包含任何 `CRITICAL` 级别的结果，则不会返回任何关键计数。有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在操作系统漏洞。

> **Note**
>
> 有关启用增强扫描后 Amazon Inspector 发出的事件的详细信息，请参阅 EventBridge 为了在 Amazon ECR 中进行增强扫描而发送的事件。

```
{
    "version": "0",
    "id": "85fc3613-e913-7fc4-a80c-a3753e4aa9ae",
    "detail-type": "ECR Image Scan",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-10-29T02:36:48Z",
    "region": "us-east-1",
```

```
    "resources": [
        "arn:aws:ecr:us-east-1:123456789012:repository/my-repository-name"
    ],
    "detail": {
        "scan-status": "COMPLETE",
        "repository-name": "my-repository-name",
        "finding-severity-counts": {
        "CRITICAL": 10,
        "MEDIUM": 9
      },
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "image-tags": []
    }
}
```

启用了增强扫描的资源的变更通知事件（增强扫描）

为注册表启用增强扫描后，当启用了增强扫描的资源发生更改时，Amazon ECR 将发送以下事件。这包括正在创建的新存储库、正在更改的存储库的扫描频率，或者在启用了增强扫描功能的存储库中创建或删除镜像的时间。有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在软件漏洞。

```
{
 "version": "0",
 "id": "0c18352a-a4d4-6853-ef53-0ab8638973bf",
 "detail-type": "ECR Scan Resource Change",
 "source": "aws.ecr",
 "account": "123456789012",
 "time": "2021-10-14T20:53:46Z",
 "region": "us-east-1",
 "resources": [],
 "detail": {
  "action-type": "SCAN_FREQUENCY_CHANGE",
  "repositories": [{
    "repository-name": "repository-1",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-1",
    "scan-frequency": "SCAN_ON_PUSH",
    "previous-scan-frequency": "MANUAL"
   },
   {
    "repository-name": "repository-2",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-2",
    "scan-frequency": "CONTINUOUS_SCAN",
```

```
    "previous-scan-frequency": "SCAN_ON_PUSH"
   },
   {
    "repository-name": "repository-3",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-3",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   }
  ],
  "resource-type": "REPOSITORY",
  "scan-type": "ENHANCED"
 }
}
```

镜像删除的事件

删除镜像时将发送以下事件。有关更多信息，请参阅 删除 Amazon ECR 中的映像。

```
{
    "version": "0",
    "id": "dd3b46cb-2c74-f49e-393b-28286b67279d",
    "detail-type": "ECR Image Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-11-16T02:01:05Z",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "result": "SUCCESS",
        "repository-name": "my-repository-name",
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "action-type": "DELETE",
        "image-tag": "latest"
    }
}
```

已完成映像复制的事件

每个映像复制完成后，将发送以下事件。有关更多信息，请参阅 Amazon ECR 中的私有映像复制。

```
{
```

```
  "version": "0",
  "id": "c8b133b1-6029-ee73-e2a1-4f466b8ba999",
  "detail-type": "ECR Replication Action",
  "source": "aws.ecr",
  "account": "123456789012",
  "time": "2024-05-08T20:44:54Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ecr:us-east-1:123456789012:repository/docker-hub/alpine"
  ],
  "detail": {
    "result": "SUCCESS",
    "repository-name": "docker-hub/alpine",
    "image-digest": 
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
    "source-account": "123456789012",
    "action-type": "REPLICATE",
    "source-region": "us-west-2",
    "image-tag": "3.17.2"
  }
}
```

使用记录 Amazon ECR 操作 Amazon CloudTrail

Amazon ECR 与 Amazon CloudTrail一项服务集成，该服务提供用户、角色或 Amazon 服务在 Amazon ECR 中采取的操作的记录。 CloudTrail 将以下 Amazon ECR 操作捕获为事件：

- 所有 API 调用，包括来自 Amazon ECR 控制台的调用
- 由于存储库上的加密设置而采取的所有操作
- 由于生命周期策略规则而采取的所有操作，包括成功和不成功的操作

Important

由于单个 CloudTrail 事件的大小限制，对于 10 张或更多图像过期的生命周期策略操作，Amazon ECR 会向发送多个事件。 CloudTrail此外，Amazon ECR 中每个镜像最多可包含 100 个标签。

创建跟踪后，您可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括 Amazon ECR 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

此信息，您可以确定向 Amazon ECR 发出了什么请求、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。

有关更多信息，请参阅 Amazon CloudTrail 《用户指南》。

亚马逊 ECR 信息位于 CloudTrail

CloudTrail 在您创建 Amazon 账户时已在您的账户上启用。当 Amazon ECR 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。您可以在自己的 Amazon 账户中查看、搜索和下载最近发生的事件。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 账户中的事件，包括 Amazon ECR 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。在控制台中创建跟踪时，您可以将跟踪应用到单个区域或所有区域。跟踪记录 Amazon 分区中的事件，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 服务来分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅：

- 为您的 Amazon 账户创建跟踪
- Amazon 与日志的服务集成 CloudTrail
- 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 ECR API 操作均由《亚马逊弹性容器注册表 API 参考》记录 CloudTrail 并记录在案。执行常见任务时，会在 CloudTrail 日志文件中为该任务中的每个 API 操作生成部分。例如，当您创建存储库时 `GetAuthorizationTokenCreateRepository`，会在 CloudTrail 日志文件中生成`SetRepositoryPolicy`章节。当您将某个镜像推送到存储库中时，则将生成 `InitiateLayerUpload`、`UploadLayerPart`、`CompleteLayerUpload` 和 `PutImage` 部分。当您提取镜像时，则将生成 `GetDownloadUrlForLayer` 和 `BatchGetImage` 部分。当支持该 OCI 1.1规范的OCI客户端使用 Referrers API 获取图片的反向链接或引用工件列表时，会触发一个`ListImageReferrers` CloudTrail 事件。有关这些常见任务的示例，请参阅 CloudTrail 日志条目示例。

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：

- 请求是使用根用户凭证还是 用户凭证发出的
- 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。

了解 Amazon ECR 日志文件条目

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，包括有关所请求操作的信息、操作的日期和时间、请求参数以及其他信息。 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顺序出现。

CloudTrail 日志条目示例

以下是一些常见 Amazon ECR 任务的 CloudTrail 日志条目示例。

为提高可读性，这些示例已进行格式化处理。在 CloudTrail 日志文件中，所有条目和事件都连接成一行。此外，该示例限于一个 Amazon ECR 条目。在真实的 CloudTrail 日志文件中，您可以看到来自多个 Amazon 服务的条目和事件。

> **Important**
>
> 来源IPAddress是发出请求的 IP 地址。对于源自服务控制台的操作，报告的地址针对的是基础资源而不是控制台 Web 服务器。对于中的服务 Amazon，仅显示 DNS 名称。即使对 Amazon 服务 DNS 名称进行了编辑，我们仍会使用客户端源 IP 来评估身份验证。

主题

- 示例：创建存储库操作
- 示例：创建 Amazon ECR 存储库时的 Amazon KMSCreateGrant API 操作
- 示例：镜像推送操作
- 示例：镜像提取操作
- 示例：镜像生命周期策略操作
- 示例：映像引用站点操作

示例：创建存储库操作

以下示例显示了演示该`CreateRepository`操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.04",
```

```
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-07-11T21:54:07Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            }
        }
    },
    "eventTime": "2018-07-11T22:17:43Z",
    "eventSource": "ecr.amazonaws.com",
    "eventName": "CreateRepository",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "repositoryName": "testrepo"
    },
    "responseElements": {
        "repository": {
            "repositoryArn": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
            "repositoryName": "testrepo",
            "repositoryUri": "123456789012.dkr.ecr.us-east-2.amazonaws.com/testrepo",
            "createdAt": "Jul 11, 2018 10:17:44 PM",
            "registryId": "123456789012"
        }
    },
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "resources": [
        {
            "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
            "accountId": "123456789012"
```

```
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

示例：创建 Amazon ECR 存储库时的 Amazon KMS`CreateGrant` API 操作

以下示例显示了一个 CloudTrail 日志条目，该条目演示了在启用了 KMS 加密的情况下创建 Amazon ECR 存储库时的 Amazon KMS `CreateGrant`操作。对于每个启用 KMS 加密创建的存储库，您应该会在中看到两个`CreateGrant`日志条目 CloudTrail。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAIEP6W46J43IG7LXAQ",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "sessionIssuer": {

            },
            "webIdFederationData": {

            },
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-06-10T19:22:10Z"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-06-10T19:22:10Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "keyId": "4b55e5bf-39c8-41ad-b589-18464af7758a",
```

```
        "granteePrincipal": "ecr.us-west-2.amazonaws.com",
        "operations": [
            "GenerateDataKey",
            "Decrypt"
        ],
        "retiringPrincipal": "ecr.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecr:arn": "arn:aws:ecr:us-west-2:123456789012:repository/testrepo"
            }
        }
    },
    "responseElements": {
        "grantId": "3636af9adfee1accb67b83941087dcd45e7fadc4e74ff0103bb338422b5055f3"
    },
    "requestID": "047b7dea-b56b-4013-87e9-a089f0f6602b",
    "eventID": "af4c9573-c56a-4886-baca-a77526544469",
    "readOnly": false,
    "resources": [
        {
            "accountId": "123456789012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:123456789012:key/4b55e5bf-39c8-41ad-
b589-18464af7758a"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

示例：镜像推送操作

以下示例显示了一个 CloudTrail 日志条目，该条目演示了使用该`PutImage`操作的图像推送。

> **Note**
>
> 推送图像时，您还将在 CloudTrail 日志中看到`InitiateLayerUpload``UploadLayerPart`、和`CompleteLayerUpload`引用。

```
{
    "eventVersion": "1.04",
```

```
    "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
    "arn": "arn:aws:sts::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major",
  "sessionContext": {
   "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2019-04-15T16:42:14Z"
   }
  }
 },
 "eventTime": "2019-04-15T16:45:00Z",
 "eventSource": "ecr.amazonaws.com",
 "eventName": "PutImage",
 "awsRegion": "us-east-2",
 "sourceIPAddress": "Amazon Internal",
 "userAgent": "Amazon Internal",
 "requestParameters": {
  "repositoryName": "testrepo",
  "imageTag": "latest",
  "registryId": "123456789012",
  "imageManifest": "{\n   \"schemaVersion\": 2,\n   \"mediaType\": \"application/
vnd.docker.distribution.manifest.v2+json\",\n   \"config\": {\n      \"mediaType\":
 \"application/vnd.docker.container.image.v1+json\",\n      \"size\": 5543,\n
 \"digest\": \"sha256:000b9b805af1cdb60628898c9f411996301a1c13afd3dbef1d8a16ac6dbf503a
\"\n   },\n   \"layers\": [\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 43252507,\n
 \"digest\": \"sha256:3b37166ec61459e76e33282dda08f2a9cd698ca7e3d6bc44e6a6e7580cdeff8e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 846,\n         \"digest
\": \"sha256:504facff238fde83f1ca8f9f54520b4219c5b8f80be9616ddc52d31448a044bd
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 615,\n         \"digest
\": \"sha256:ebbcacd28e101968415b0c812b2d2dc60f969e36b0b08c073bf796e12b1bb449\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 850,\n         \"digest
\": \"sha256:c7fb3351ecad291a88b92b600037e2435c84a347683d540042086fe72c902b8a
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 168,\n         \"digest\":
 \"sha256:2e3debadcbf7e542e2aefbce1b64a358b1931fb403b3e4aeca27cb4d809d56c2\"\n      },
\n      {\n         \"mediaType\": \"application/vnd.docker.image.rootfs.diff.tar.gzip
```

```
\",\n         \"size\": 37720774,\n         \"digest\":
 \"sha256:f8c9f51ad524d8ae9bf4db69cd3e720ba92373ec265f5c390ffb21bb0c277941\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 30432107,\n
 \"digest\": \"sha256:813a50b13f61cf1f8d25f19fa96ad3aa5b552896c83e86ce413b48b091d7f01b
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 197,\n         \"digest
\": \"sha256:7ab043301a6187ea3293d80b30ba06c7bf1a0c3cd4c43d10353b31bc0cecfe7d
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 154,\n         \"digest
\": \"sha256:67012cca8f31dc3b8ee2305e7762fee20c250513effdedb38a1c37784a5a2e71\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 176,\n         \"digest
\": \"sha256:3bc892145603fffc9b1c97c94e2985b4cb19ca508750b15845a5d97becbd1a0e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 183,\n         \"digest
\": \"sha256:6f1c79518f18251d35977e7e46bfa6c6b9cf50df2a79d4194941d95c54258d18\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest
\": \"sha256:b7bcfbc2e2888afebede4dd1cd5eebf029bb6315feeaf0b56e425e11a50afe42\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest\":
 \"sha256:2b220f8b0f32b7c2ed8eaafe1c802633bbd94849b9ab73926f0ba46cdae91629\"\n      }\n
   ]\n}"
 },
 "responseElements": {
  "image": {
   "repositoryName": "testrepo",
   "imageManifest": "{\n   \"schemaVersion\": 2,\n   \"mediaType\": \"application/
vnd.docker.distribution.manifest.v2+json\",\n   \"config\": {\n      \"mediaType\":
 \"application/vnd.docker.container.image.v1+json\",\n      \"size\": 5543,\n
 \"digest\": \"sha256:000b9b805af1cdb60628898c9f411996301a1c13afd3dbef1d8a16ac6dbf503a
\"\n   },\n   \"layers\": [\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 43252507,\n
 \"digest\": \"sha256:3b37166ec61459e76e33282dda08f2a9cd698ca7e3d6bc44e6a6e7580cdeff8e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 846,\n         \"digest
\": \"sha256:504facff238fde83f1ca8f9f54520b4219c5b8f80be9616ddc52d31448a044bd
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 615,\n         \"digest
\": \"sha256:ebbcacd28e101968415b0c812b2d2dc60f969e36b0b08c073bf796e12b1bb449\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 850,\n         \"digest
\": \"sha256:c7fb3351ecad291a88b92b600037e2435c84a347683d540042086fe72c902b8a
```

```
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 168,\n         \"digest\":
 \"sha256:2e3debadcbf7e542e2aefbce1b64a358b1931fb403b3e4aeca27cb4d809d56c2\"\n      },
\n      {\n         \"mediaType\": \"application/vnd.docker.image.rootfs.diff.tar.gzip
\",\n         \"size\": 37720774,\n         \"digest\":
 \"sha256:f8c9f51ad524d8ae9bf4db69cd3e720ba92373ec265f5c390ffb21bb0c277941\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 30432107,\n
 \"digest\": \"sha256:813a50b13f61cf1f8d25f19fa96ad3aa5b552896c83e86ce413b48b091d7f01b
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 197,\n         \"digest
\": \"sha256:7ab043301a6187ea3293d80b30ba06c7bf1a0c3cd4c43d10353b31bc0cecfe7d
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 154,\n         \"digest
\": \"sha256:67012cca8f31dc3b8ee2305e7762fee20c250513effdedb38a1c37784a5a2e71\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 176,\n         \"digest
\": \"sha256:3bc892145603fffc9b1c97c94e2985b4cb19ca508750b15845a5d97becbd1a0e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 183,\n         \"digest
\": \"sha256:6f1c79518f18251d35977e7e46bfa6c6b9cf50df2a79d4194941d95c54258d18\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest
\": \"sha256:b7bcfbc2e2888afebede4dd1cd5eebf029bb6315feeaf0b56e425e11a50afe42\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest\":
 \"sha256:2b220f8b0f32b7c2ed8eaafe1c802633bbd94849b9ab73926f0ba46cdae91629\"\n      }\n
   ]\n}",
   "registryId": "123456789012",
   "imageId": {
    "imageDigest":
 "sha256:98c8b060c21d9adbb6b8c41b916e95e6307102786973ab93a41e8b86d1fc6d3e",
    "imageTag": "latest"
   }
  }
 },
 "requestID": "cf044b7d-5f9d-11e9-9b2a-95983139cc57",
 "eventID": "2bfd4ee2-2178-4a82-a27d-b12939923f0f",
 "resources": [{
  "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
  "accountId": "123456789012"
 }],
 "eventType": "AwsApiCall",
 "recipientAccountId": "123456789012"
```

```
}
```

示例：镜像提取操作

以下示例显示了一个 CloudTrail 日志条目，该条目演示了使用该`BatchGetImage`操作的图像拉取。

> **Note**
>
> 当拉取映像时，如果本地尚没有映像，您在 CloudTrail 日志中还将看到 `GetDownloadUrlForLayer` 引用。

```
{
    "eventVersion": "1.04",
    "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
    "arn": "arn:aws:sts::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major",
  "sessionContext": {
   "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2019-04-15T16:42:14Z"
   }
  }
 },
 "eventTime": "2019-04-15T17:23:20Z",
 "eventSource": "ecr.amazonaws.com",
 "eventName": "BatchGetImage",
 "awsRegion": "us-east-2",
 "sourceIPAddress": "ecr.amazonaws.com",
 "userAgent": "ecr.amazonaws.com",
 "requestParameters": {
  "imageIds": [{
   "imageTag": "latest"
  }],
  "acceptedMediaTypes": [
   "application/json",
   "application/vnd.oci.image.manifest.v1+json",
   "application/vnd.oci.image.index.v1+json",
```

```
    "application/vnd.docker.distribution.manifest.v2+json",
    "application/vnd.docker.distribution.manifest.list.v2+json",
    "application/vnd.docker.distribution.manifest.v1+prettyjws"
   ],
   "repositoryName": "testrepo",
   "registryId": "123456789012"
  },
  "responseElements": null,
  "requestID": "2a1b97ee-5fa3-11e9-a8cd-cd2391aeda93",
  "eventID": "c84f5880-c2f9-4585-9757-28fa5c1065df",
  "resources": [{
   "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
   "accountId": "123456789012"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
 }
```

示例：镜像生命周期策略操作

以下示例显示了一个 CloudTrail 日志条目，该条目演示了图像何时因生命周期策略规则而过期。可通过筛选事件名称字段的 `PolicyExecutionEvent` 来定位此事件类型。

当您测试生命周期策略预览时，Amazon ECR 会生成一个 CloudTrail日志条目，其事件名称字段为`DryRunEvent`，其结构与完全相同。`PolicyExecutionEvent`通过将事件名称更改为`DryRunEvent`，您可以改为筛选试运行事件。

> ⚠ Important
>
> 由于单个 CloudTrail 事件的大小限制，对于 10 张或更多图像过期的生命周期策略操作，Amazon ECR 会向发送多个事件。 CloudTrail此外，Amazon ECR 中每个镜像最多可包含 100 个标签。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "accountId": "123456789012",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-03-12T20:22:12Z",
    "eventSource": "ecr.amazonaws.com",
```

```
    "eventName": "PolicyExecutionEvent",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "9354dd7f-9aac-4e9d-956d-12561a4923aa",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:ecr:us-west-2:123456789012:repository/testrepo",
            "accountId": "123456789012",
            "type": "AWS::ECR::Repository"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "123456789012",
    "serviceEventDetails": {
        "repositoryName": "testrepo",
        "lifecycleEventPolicy": {
            "lifecycleEventRules": [
                {
                    "rulePriority": 1,
                    "description": "remove all images > 2",
                    "lifecycleEventSelection": {
                        "tagStatus": "Any",
                        "tagPrefixList": [],
                        "countType": "Image count more than",
                        "countNumber": 2
                    },
                    "action": "expire"
                }
            ],
            "lastEvaluatedAt": 0,
            "policyVersion": 1,
            "policyId": "ceb86829-58e7-9498-920c-aa042e33037b"
        },
        "lifecycleEventImageActions": [
            {
                "lifecycleEventImage": {
                    "digest":
"sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45",
                    "tagStatus": "Tagged",
                    "tagList": [
```

```
                        "alpine"
                    ],
                    "pushedAt": 1584042813000
                },
                "rulePriority": 1
            },
            {
                "lifecycleEventImage": {
                    "digest":
 "sha256:6ab380c5a5acf71c1b6660d645d2cd79cc8ce91b38e0352cbf9561e050427baf",
                    "tagStatus": "Tagged",
                    "tagList": [
                        "centos"
                    ],
                    "pushedAt": 1584042842000
                },
                "rulePriority": 1
            }
        ],
        "lifecycleEventFailureDetails": [
            {
                "lifecycleEventImage": {
                    "digest":
 "sha256:9117e1bc28cd20751e584b4ccd19b1178d14cf02d134b04ce6be0cc51bff762a",
                    "tagStatus": "Untagged",
                    "tagList": [],
                    "pushedAt": 1584042844000
                },
                "rulePriority": 1,
                "failureCode": "ImageReferencedByManifestList",
                "failureReason": "Requested image referenced by manifest list:
 [sha256:4b27c83d44a18c31543039d9e8b2786043ec6c8d00804d5800c5148d6b6f65bc]"
            }
        ]
    }
}
```

示例：映像引用站点操作

以下示例显示了一个 Amazon CloudTrail 日志条目，该条目演示了OCI 1.1合规的客户端何时使用 API 获取图片的反向链接或参考构件列表。`Referrers`

```
{
```

```
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-10-08T16:38:39Z",
                "mfaAuthenticated": "false"
            },
            "ec2RoleDelivery": "2.0"
        },
        "invokedBy": "ecr.amazonaws.com"
    },
    "eventTime": "2024-10-08T17:22:51Z",
    "eventSource": "ecr.amazonaws.com",
    "eventName": "ListImageReferrers",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "ecr.amazonaws.com",
    "userAgent": "ecr.amazonaws.com",
    "requestParameters": {
        "registryId": "123456789012",
        "repositoryName": "testrepo",
        "subjectId": {
            "imageDigest":
 "sha256:000b9b805af1cdb60628898c9f411996301a1c13afd3dbef1d8a16ac6dbf503a"
        },
        "nextToken": "urD72mdD/mC8b5-EXAMPLE"
    },
    "responseElements": null,
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "readOnly": true,
    "resources": [
```

```
        {
            "accountId": "123456789012",
            "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```

将 Amazon ECR 与软件开发工具包一起使用 Amazon

Amazon 软件开发套件 (SDKs) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档
Amazon CLI
适用于 Java 的 Amazon SDK
适用于 JavaScript 的 Amazon SDK
适用于 .NET 的 Amazon SDK
适用于 PHP 的 Amazon SDK
Amazon Tools for PowerShell
适用于 Python (Boto3) 的 Amazon SDK
适用于 Ruby 的 Amazon SDK
适用于 SAP ABAP 的 Amazon SDK

使用 Amazon ECR 的代码示例 Amazon SDKs

以下代码示例展示了如何将 Amazon ECR 与 Amazon 软件开发套件 (SDK) 一起使用。

基础知识是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

Hello Amazon ECR

以下代码示例展示了如何开始使用 Amazon ECR。

Java

适用于 Java 的 SDK 2.x

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ecr.EcrClient;
import software.amazon.awssdk.services.ecr.model.EcrException;
import software.amazon.awssdk.services.ecr.model.ListImagesRequest;
import software.amazon.awssdk.services.ecr.paginators.ListImagesIterable;

public class HelloECR {

    public static void main(String[] args) {
        final String usage = """
            Usage:    <repositoryName>
```

```
            Where:
               repositoryName - The name of the Amazon ECR repository.
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String repoName = args[0];
        EcrClient ecrClient = EcrClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listImageTags(ecrClient, repoName);
    }
    public static void listImageTags(EcrClient ecrClient, String repoName){
        ListImagesRequest listImagesPaginator = ListImagesRequest.builder()
            .repositoryName(repoName)
            .build();

        ListImagesIterable imagesIterable =
 ecrClient.listImagesPaginator(listImagesPaginator);
        imagesIterable.stream()
            .flatMap(r -> r.imageIds().stream())
            .forEach(image -> System.out.println("The docker image tag is: "
 +image.imageTag()));
    }
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for Java 2.x API 参考》中的 listImages。

Kotlin

适用于 Kotlin 的 SDK

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import aws.sdk.kotlin.services.ecr.EcrClient
import aws.sdk.kotlin.services.ecr.model.ListImagesRequest
import kotlin.system.exitProcess

suspend fun main(args: Array<String>) {
    val usage = """
            Usage: <repositoryName>

            Where:
               repositoryName - The name of the Amazon ECR repository.

    """.trimIndent()

    if (args.size != 1) {
        println(usage)
        exitProcess(1)
    }

    val repoName = args[0]
    listImageTags(repoName)
}

suspend fun listImageTags(repoName: String?) {
    val listImages =
        ListImagesRequest {
            repositoryName = repoName
        }

    EcrClient { region = "us-east-1" }.use { ecrClient ->
        val imageResponse = ecrClient.listImages(listImages)
        imageResponse.imageIds?.forEach { imageId ->
            println("Image tag: ${imageId.imageTag}")
        }
    }
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for Kotlin API 参考》中的 listImages。

Python

适用于 Python 的 SDK（Boto3）

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import boto3
import argparse
from boto3 import client


def hello_ecr(ecr_client: client, repository_name: str) -> None:
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon Elastic Container
 Registry (Amazon ECR)
    client and list the images in a repository.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param ecr_client: A Boto3 Amazon ECR Client object. This object wraps
                         the low-level Amazon ECR service API.
    :param repository_name: The name of an Amazon ECR repository in your account.
    """
    print(
        f"Hello, Amazon ECR! Let's list some images in the repository
 '{repository_name}':\n"
    )
    paginator = ecr_client.get_paginator("list_images")
    page_iterator = paginator.paginate(
        repositoryName=repository_name, PaginationConfig={"MaxItems": 10}
    )

    image_names: [str] = []
    for page in page_iterator:
        for schedule in page["imageIds"]:
            image_names.append(schedule["imageTag"])

    print(f"{len(image_names)} image(s) retrieved.")
```

```
    for schedule_name in image_names:
        print(f"\t{schedule_name}")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Run hello Amazon ECR.")
    parser.add_argument(
        "--repository-name",
        type=str,
        help="the name of an Amazon ECR repository in your account.",
        required=True,
    )
    args = parser.parse_args()

    hello_ecr(boto3.client("ecr"), args.repository_name)
```

- 有关 API 的详细信息，请参阅 Python Amazon SDK 中的 ListImages (Boto3) API 参考 API 参考。

代码示例

- 使用 Amazon ECR 的基本示例 Amazon SDKs
  - Hello Amazon ECR
  - 使用软件开发工具包学习 Amazon ECR 的基础知识 Amazon
  - 使用 Amazon ECR 执行的操作 Amazon SDKs
    - CreateRepository与 Amazon SDK 或 CLI 配合使用
    - DeleteRepository与 Amazon SDK 或 CLI 配合使用
    - DescribeImages与 Amazon SDK 或 CLI 配合使用
    - DescribeRepositories与 Amazon SDK 或 CLI 配合使用
    - GetAuthorizationToken与 Amazon SDK 或 CLI 配合使用
    - GetRepositoryPolicy与 Amazon SDK 或 CLI 配合使用
    - ListImages与 Amazon SDK 或 CLI 配合使用
    - 与 Amazon SDK PushImageCmd 配合使用
    - PutLifeCyclePolicy与 Amazon SDK 或 CLI 配合使用
    - SetRepositoryPolicy与 Amazon SDK 或 CLI 配合使用
    - StartLifecyclePolicyPreview与 Amazon SDK 或 CLI 配合使用

使用 Amazon ECR 的基本示例 Amazon SDKs

以下代码示例展示了如何使用 Amazon 弹性容器注册表的基础知识 Amazon SDKs。

示例

- Hello Amazon ECR
- 使用软件开发工具包学习 Amazon ECR 的基础知识 Amazon
- 使用 Amazon ECR 执行的操作 Amazon SDKs
  - CreateRepository与 Amazon SDK 或 CLI 配合使用
  - DeleteRepository与 Amazon SDK 或 CLI 配合使用
  - DescribeImages与 Amazon SDK 或 CLI 配合使用
  - DescribeRepositories与 Amazon SDK 或 CLI 配合使用
  - GetAuthorizationToken与 Amazon SDK 或 CLI 配合使用
  - GetRepositoryPolicy与 Amazon SDK 或 CLI 配合使用
  - ListImages与 Amazon SDK 或 CLI 配合使用
  - 与 Amazon SDK PushImageCmd 配合使用
  - PutLifeCyclePolicy与 Amazon SDK 或 CLI 配合使用
  - SetRepositoryPolicy与 Amazon SDK 或 CLI 配合使用
  - StartLifecyclePolicyPreview与 Amazon SDK 或 CLI 配合使用

Hello Amazon ECR

以下代码示例展示了如何开始使用 Amazon ECR。

Java

适用于 Java 的 SDK 2.x

> Note
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.ecr.EcrClient;
import software.amazon.awssdk.services.ecr.model.EcrException;
import software.amazon.awssdk.services.ecr.model.ListImagesRequest;
import software.amazon.awssdk.services.ecr.paginators.ListImagesIterable;

public class HelloECR {

    public static void main(String[] args) {
        final String usage = """
            Usage:    <repositoryName>

            Where:
               repositoryName - The name of the Amazon ECR repository.
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String repoName = args[0];
        EcrClient ecrClient = EcrClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listImageTags(ecrClient, repoName);
    }
    public static void listImageTags(EcrClient ecrClient, String repoName){
        ListImagesRequest listImagesPaginator = ListImagesRequest.builder()
            .repositoryName(repoName)
            .build();

        ListImagesIterable imagesIterable =
 ecrClient.listImagesPaginator(listImagesPaginator);
        imagesIterable.stream()
            .flatMap(r -> r.imageIds().stream())
            .forEach(image -> System.out.println("The docker image tag is: "
 +image.imageTag()));
    }
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for Java 2.x API 参考》中的 listImages。

Kotlin

适用于 Kotlin 的 SDK

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import aws.sdk.kotlin.services.ecr.EcrClient
import aws.sdk.kotlin.services.ecr.model.ListImagesRequest
import kotlin.system.exitProcess

suspend fun main(args: Array<String>) {
    val usage = """
            Usage: <repositoryName>

            Where:
               repositoryName - The name of the Amazon ECR repository.

    """.trimIndent()

    if (args.size != 1) {
        println(usage)
        exitProcess(1)
    }

    val repoName = args[0]
    listImageTags(repoName)
}

suspend fun listImageTags(repoName: String?) {
    val listImages =
        ListImagesRequest {
            repositoryName = repoName
        }

    EcrClient { region = "us-east-1" }.use { ecrClient ->
        val imageResponse = ecrClient.listImages(listImages)
        imageResponse.imageIds?.forEach { imageId ->
            println("Image tag: ${imageId.imageTag}")
```

```
        }
    }
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for Kotlin API 参考》中的 listImages。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import boto3
import argparse
from boto3 import client


def hello_ecr(ecr_client: client, repository_name: str) -> None:
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon Elastic Container
 Registry (Amazon ECR)
    client and list the images in a repository.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param ecr_client: A Boto3 Amazon ECR Client object. This object wraps
                             the low-level Amazon ECR service API.
    :param repository_name: The name of an Amazon ECR repository in your account.
    """
    print(
        f"Hello, Amazon ECR! Let's list some images in the repository
 '{repository_name}':\n"
    )
    paginator = ecr_client.get_paginator("list_images")
    page_iterator = paginator.paginate(
        repositoryName=repository_name, PaginationConfig={"MaxItems": 10}
    )
```

```
    image_names: [str] = []
    for page in page_iterator:
        for schedule in page["imageIds"]:
            image_names.append(schedule["imageTag"])

    print(f"{len(image_names)} image(s) retrieved.")
    for schedule_name in image_names:
        print(f"\t{schedule_name}")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Run hello Amazon ECR.")
    parser.add_argument(
        "--repository-name",
        type=str,
        help="the name of an Amazon ECR repository in your account.",
        required=True,
    )
    args = parser.parse_args()

    hello_ecr(boto3.client("ecr"), args.repository_name)
```

- 有关 API 的详细信息，请参阅 Python Amazon SDK 中的 ListImages (Boto3) API 参考 API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包学习 Amazon ECR 的基础知识 Amazon

以下代码示例演示了如何：

- 创建 Amazon ECR 存储库。
- 设置存储库策略。
- 检索存储库 URIs。
- 获取 Amazon ECR 授权令牌。
- 设置 Amazon ECR 存储库的生命周期策略。
- 将 Docker 映像推送到 Amazon ECR 存储库。

- 验证 Amazon ECR 存储库中是否存在映像。
- 列出您账户的 Amazon ECR 存储库，并获取有关它们的详细信息。
- 删除 Amazon ECR 存储库。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

运行一个交互式场景，演示 Amazon ECR 的功能。

```
import software.amazon.awssdk.services.ecr.model.EcrException;
import
 software.amazon.awssdk.services.ecr.model.RepositoryPolicyNotFoundException;

import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This Java code example requires an IAM Role that has permissions to interact
 with the Amazon ECR service.
 *
 * To create an IAM role, see:
 *
 * https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create.html
 *
 * This Java scenario example requires a local docker image named echo-text.
 Without a local image,
```

```
 * this Java program will not successfully run. For more information including
 how to create the local
 * image, see:
 *
 * /scenarios/basics/ecr/README
 *
 */
public class ECRScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");
    public static void main(String[] args) {
        final String usage = """
            Usage: <iamRoleARN> <accountId>

            Where:
               iamRoleARN - The IAM role ARN that has the necessary permissions
 to access and manage the Amazon ECR repository.
               accountId - Your AWS account number.
            """;

        if (args.length != 2) {
            System.out.println(usage);
            return;
        }

        ECRActions ecrActions = new ECRActions();
        String iamRole = args[0];
        String accountId = args[1];
        String localImageName;

        Scanner scanner = new Scanner(System.in);
        System.out.println("""
             The Amazon Elastic Container Registry (ECR) is a fully-managed
 Docker container registry
             service provided by AWS. It allows developers and organizations to
 securely
             store, manage, and deploy Docker container images.
             ECR provides a simple and scalable way to manage container images
 throughout their lifecycle,
             from building and testing to production deployment.\s

             The `EcrAsyncClient` interface in the AWS SDK for Java 2.x provides
 a set of methods to
```

```
             programmatically interact with the Amazon ECR service. This allows
developers to
             automate the storage, retrieval, and management of container images
as part of their application
             deployment pipelines. With ECR, teams can focus on building and
deploying their
             applications without having to worry about the underlying
infrastructure required to
             host and manage a container registry.

            This scenario walks you through how to perform key operations for
this service.
            Let's get started...

            You have two choices:
            1 - Run the entire program.
            2 - Delete an existing Amazon ECR repository named echo-text (created
from a previous execution of
            this program that did not complete).
            """);

        while (true) {
            String input = scanner.nextLine();
            if (input.trim().equalsIgnoreCase("1")) {
                System.out.println("Continuing with the program...");
                System.out.println("");
                break;
            } else if (input.trim().equalsIgnoreCase("2")) {
                String repoName = "echo-text";
                ecrActions.deleteECRRepository(repoName);
                return;
            } else {
                // Handle invalid input.
                System.out.println("Invalid input. Please try again.");
            }
        }

        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println("""
           1. Create an ECR repository.
```

```
            The first task is to ensure we have a local Docker image named echo-
text.
            If this image exists, then an Amazon ECR repository is created.

            An ECR repository is a private Docker container repository provided
            by Amazon Web Services (AWS). It is a managed service that makes it
 easy
            to store, manage, and deploy Docker container images.\s
            """ );

        // Ensure that a local docker image named echo-text exists.
        boolean doesExist = ecrActions.isEchoTextImagePresent();
        String repoName;
        if (!doesExist){
            System.out.println("The local image named echo-text does not exist");
            return;
        } else {
            localImageName = "echo-text";
            repoName = "echo-text";
        }

        try {
            String repoArn = ecrActions.createECRRepository(repoName);
            System.out.println("The ARN of the ECR repository is " + repoArn);

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid repository name: " + e.getMessage());
            return;
        } catch (RuntimeException e) {
            System.err.println("An error occurred while creating the ECR
 repository: " + e.getMessage());
            e.printStackTrace();
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
        2. Set an ECR repository policy.

        Setting an ECR repository policy using the `setRepositoryPolicy` function
 is crucial for maintaining
        the security and integrity of your container images. The repository
 policy allows you to
```

```
        define specific rules and restrictions for accessing and managing the
images stored within your ECR
        repository.
        """);
        waitForInputToContinue(scanner);
        try {
            ecrActions.setRepoPolicy(repoName, iamRole);

        } catch (RepositoryPolicyNotFoundException e) {
            System.err.println("Invalid repository name: " + e.getMessage());
            return;
        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;
        } catch (RuntimeException e) {
            System.err.println("An error occurred while creating the ECR
repository: " + e.getMessage());
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
        3. Display ECR repository policy.

        Now we will retrieve the ECR policy to ensure it was successfully set.
        """);
        waitForInputToContinue(scanner);
        try {
            String policyText = ecrActions.getRepoPolicy(repoName);
            System.out.println("Policy Text:");
            System.out.println(policyText);

        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;
        } catch (RuntimeException e) {
            System.err.println("An error occurred while creating the ECR
repository: " + e.getMessage());
            return;
        }

        waitForInputToContinue(scanner);
```

```
        System.out.println(DASHES);
        System.out.println("""
        4. Retrieve an ECR authorization token.

        You need an authorization token to securely access and interact with the
 Amazon ECR registry.
        The `getAuthorizationToken` method of the `EcrAsyncClient` is responsible
 for securely accessing
        and interacting with an Amazon ECR repository. This operation is
 responsible for obtaining a
        valid authorization token, which is required to authenticate your
 requests to the ECR service.

        Without a valid authorization token, you would not be able to perform any
 operations on the
        ECR repository, such as pushing, pulling, or managing your Docker images.

        """);
        waitForInputToContinue(scanner);
        try {
             ecrActions.getAuthToken();

        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;
        } catch (RuntimeException e) {
            System.err.println("An error occurred while retrieving the
 authorization token: " + e.getMessage());
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
        5. Get the ECR Repository URI.

        The URI  of an Amazon ECR repository is important. When you want to
 deploy a container image to
        a container orchestration platform like Amazon Elastic Kubernetes Service
 (EKS)
        or Amazon Elastic Container Service (ECS), you need to specify the full
 image URI,
        which includes the ECR repository URI. This allows the container runtime
 to pull the
```

```
        correct container image from the ECR repository.
       """);
        waitForInputToContinue(scanner);

        try {
            ecrActions.getRepositoryURI(repoName);

        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;

        } catch (RuntimeException e) {
            System.err.println("An error occurred while retrieving the URI: " +
e.getMessage());
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
            6. Set an ECR Lifecycle Policy.

            An ECR Lifecycle Policy is used to manage the lifecycle of Docker
images stored in your ECR repositories.
            These policies allow you to automatically remove old or unused Docker
images from your repositories,
            freeing up storage space and reducing costs.

            This example policy helps to maintain the size and efficiency of the
container registry
            by automatically removing older and potentially unused images,
ensuring that the
            storage is optimized and the registry remains up-to-date.
            """);
        waitForInputToContinue(scanner);
        try {
            ecrActions.setLifeCyclePolicy(repoName);

        } catch (RuntimeException e) {
            System.err.println("An error occurred while setting the lifecycle
policy: " + e.getMessage());
            e.printStackTrace();
            return;
        }
```

```
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("""
        7. Push a docker image to the Amazon ECR Repository.

        The `pushImageCmd()` method pushes a local Docker image to an Amazon ECR
 repository.
        It sets up the Docker client by connecting to the local Docker host using
 the default port.
        It then retrieves the authorization token for the ECR repository by
 making a call to the AWS SDK.

        The method uses the authorization token to create an `AuthConfig` object,
 which is used to authenticate
        the Docker client when pushing the image. Finally, the method tags the
 Docker image with the specified
        repository name and image tag, and then pushes the image to the ECR
 repository using the Docker client.
        If the push operation is successful, the method prints a message
 indicating that the image was pushed to ECR.
        """);
        waitForInputToContinue(scanner);

        try {
            ecrActions.pushDockerImage(repoName, localImageName);

        } catch (RuntimeException e) {
            System.err.println("An error occurred while pushing a local Docker
 image to Amazon ECR: " + e.getMessage());
            e.printStackTrace();
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("8. Verify if the image is in the ECR Repository.");
        waitForInputToContinue(scanner);
        try {
            ecrActions.verifyImage(repoName, localImageName);

        } catch (EcrException e) {
            System.err.println("An ECR exception occurred: " + e.getMessage());
            return;
```

```
        } catch (RuntimeException e) {
            System.err.println("An error occurred " + e.getMessage());
            e.printStackTrace();
            return;
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("9. As an optional step, you can interact with the
 image in Amazon ECR by using the CLI.");
        System.out.println("Would you like to view instructions on how to use the
 CLI to run the image? (y/n)");
        String ans = scanner.nextLine().trim();
        if (ans.equalsIgnoreCase("y")) {
            String instructions = """
            1. Authenticate with ECR - Before you can pull the image from Amazon
 ECR, you need to authenticate with the registry. You can do this using the AWS
 CLI:

                aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin %s.dkr.ecr.us-east-1.amazonaws.com

            2. Describe the image using this command:

               aws ecr describe-images --repository-name %s --image-ids imageTag=
%s

            3. Run the Docker container and view the output using this command:

               docker run --rm %s.dkr.ecr.us-east-1.amazonaws.com/%s:%s
            """;

            instructions = String.format(instructions, accountId, repoName,
 localImageName, accountId, repoName, localImageName);
            System.out.println(instructions);
        }
        waitForInputToContinue(scanner);

        System.out.println(DASHES);
        System.out.println("10. Delete the ECR Repository.");
        System.out.println(
        """
        If the repository isn't empty, you must either delete the contents of the
 repository
```

```
       or use the force option (used in this scenario) to delete the repository
and have Amazon ECR delete all of its contents
       on your behalf.
       """);
       System.out.println("Would you like to delete the Amazon ECR Repository?
(y/n)");
       String delAns = scanner.nextLine().trim();
       if (delAns.equalsIgnoreCase("y")) {
           System.out.println("You selected to delete the AWS ECR resources.");

           try {
               ecrActions.deleteECRRepository(repoName);

           } catch (EcrException e) {
               System.err.println("An ECR exception occurred: " +
e.getMessage());
               return;
           } catch (RuntimeException e) {
               System.err.println("An error occurred while deleting the Docker
image: " + e.getMessage());
               e.printStackTrace();
               return;
           }
       }

       System.out.println(DASHES);
       System.out.println("This concludes the Amazon ECR SDK scenario");
       System.out.println(DASHES);
   }

  private static void waitForInputToContinue(Scanner scanner) {
      while (true) {
          System.out.println("");
          System.out.println("Enter 'c' followed by <ENTER> to continue:");
          String input = scanner.nextLine();

          if (input.trim().equalsIgnoreCase("c")) {
              System.out.println("Continuing with the program...");
              System.out.println("");
              break;
          } else {
              // Handle invalid input.
              System.out.println("Invalid input. Please try again.");
         }
```

```
        }
    }
}
```

Amazon ECR SDK 方法的封装类。

```
import com.github.dockerjava.api.DockerClient;
import com.github.dockerjava.api.exception.DockerClientException;
import com.github.dockerjava.api.model.AuthConfig;
import com.github.dockerjava.api.model.Image;
import com.github.dockerjava.core.DockerClientBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration;
import software.amazon.awssdk.http.async.SdkAsyncHttpClient;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ecr.EcrAsyncClient;
import software.amazon.awssdk.services.ecr.model.AuthorizationData;
import software.amazon.awssdk.services.ecr.model.CreateRepositoryRequest;
import software.amazon.awssdk.services.ecr.model.CreateRepositoryResponse;
import software.amazon.awssdk.services.ecr.model.DeleteRepositoryRequest;
import software.amazon.awssdk.services.ecr.model.DeleteRepositoryResponse;
import software.amazon.awssdk.services.ecr.model.DescribeImagesRequest;
import software.amazon.awssdk.services.ecr.model.DescribeImagesResponse;
import software.amazon.awssdk.services.ecr.model.DescribeRepositoriesRequest;
import software.amazon.awssdk.services.ecr.model.DescribeRepositoriesResponse;
import software.amazon.awssdk.services.ecr.model.EcrException;
import software.amazon.awssdk.services.ecr.model.GetAuthorizationTokenResponse;
import software.amazon.awssdk.services.ecr.model.GetRepositoryPolicyRequest;
import software.amazon.awssdk.services.ecr.model.GetRepositoryPolicyResponse;
import software.amazon.awssdk.services.ecr.model.ImageIdentifier;
import software.amazon.awssdk.services.ecr.model.Repository;
import
 software.amazon.awssdk.services.ecr.model.RepositoryPolicyNotFoundException;
import software.amazon.awssdk.services.ecr.model.SetRepositoryPolicyRequest;
import software.amazon.awssdk.services.ecr.model.SetRepositoryPolicyResponse;
import
 software.amazon.awssdk.services.ecr.model.StartLifecyclePolicyPreviewRequest;
import
 software.amazon.awssdk.services.ecr.model.StartLifecyclePolicyPreviewResponse;
import com.github.dockerjava.api.command.DockerCmdExecFactory;
```

```
import com.github.dockerjava.netty.NettyDockerCmdExecFactory;
import java.time.Duration;
import java.util.Base64;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.CompletionException;

public class ECRActions {
    private static EcrAsyncClient ecrClient;

    private static DockerClient dockerClient;

    private static Logger logger = LoggerFactory.getLogger(ECRActions.class);

    /**
     * Creates an Amazon Elastic Container Registry (Amazon ECR) repository.
     *
     * @param repoName the name of the repository to create.
     * @return the Amazon Resource Name (ARN) of the created repository, or an
 empty string if the operation failed.
     * @throws IllegalArgumentException     If repository name is invalid.
     * @throws RuntimeException             if an error occurs while creating the
 repository.
     */
    public String createECRRepository(String repoName) {
        if (repoName == null || repoName.isEmpty()) {
            throw new IllegalArgumentException("Repository name cannot be null or
 empty");
        }

        CreateRepositoryRequest request = CreateRepositoryRequest.builder()
            .repositoryName(repoName)
            .build();

        CompletableFuture<CreateRepositoryResponse> response =
 getAsyncClient().createRepository(request);
        try {
            CreateRepositoryResponse result = response.join();
            if (result != null) {
                System.out.println("The " + repoName + " repository was created
 successfully.");
                return result.repository().repositoryArn();
            } else {
                throw new RuntimeException("Unexpected response type");
```

```
            }
        } catch (CompletionException e) {
            Throwable cause = e.getCause();
            if (cause instanceof EcrException ex) {
                if
("RepositoryAlreadyExistsException".equals(ex.awsErrorDetails().errorCode())) {
                    System.out.println("The Amazon ECR repository already exists,
moving on...");
                    DescribeRepositoriesRequest describeRequest =
DescribeRepositoriesRequest.builder()
                        .repositoryNames(repoName)
                        .build();
                    DescribeRepositoriesResponse describeResponse =
getAsyncClient().describeRepositories(describeRequest).join();
                    return
describeResponse.repositories().get(0).repositoryArn();
                } else {
                    throw new RuntimeException(ex);
                }
            } else {
                throw new RuntimeException(e);
            }
        }
    }

    /**
     * Deletes an ECR (Elastic Container Registry) repository.
     *
     * @param repoName the name of the repository to delete.
     * @throws IllegalArgumentException if the repository name is null or empty.
     * @throws EcrException if there is an error deleting the repository.
     * @throws RuntimeException if an unexpected error occurs during the deletion
process.
     */
    public void deleteECRRepository(String repoName) {
        if (repoName == null || repoName.isEmpty()) {
            throw new IllegalArgumentException("Repository name cannot be null or
empty");
        }

        DeleteRepositoryRequest repositoryRequest =
DeleteRepositoryRequest.builder()
            .force(true)
            .repositoryName(repoName)
```

```
           .build();

       CompletableFuture<DeleteRepositoryResponse> response =
getAsyncClient().deleteRepository(repositoryRequest);
       response.whenComplete((deleteRepositoryResponse, ex) -> {
           if (deleteRepositoryResponse != null) {
               System.out.println("You have successfully deleted the " +
repoName + " repository");
           } else {
               Throwable cause = ex.getCause();
               if (cause instanceof EcrException) {
                   throw (EcrException) cause;
               } else {
                   throw new RuntimeException("Unexpected error: " +
cause.getMessage(), cause);
               }
           }
       });

       // Wait for the CompletableFuture to complete
       response.join();
   }



   private static DockerClient getDockerClient() {
       String osName = System.getProperty("os.name");
       if (osName.startsWith("Windows")) {
           // Make sure Docker Desktop is running.
           String dockerHost = "tcp://localhost:2375"; // Use the Docker Desktop
default port.
           DockerCmdExecFactory dockerCmdExecFactory = new
NettyDockerCmdExecFactory().withReadTimeout(20000).withConnectTimeout(20000);
           dockerClient =
DockerClientBuilder.getInstance(dockerHost).withDockerCmdExecFactory(dockerCmdExecFactor
       } else {
           dockerClient = DockerClientBuilder.getInstance().build();
       }
       return dockerClient;
   }

   /**
    * Retrieves an asynchronous Amazon Elastic Container Registry (ECR) client.
    *
```

```
     * @return the configured ECR asynchronous client.
     */
    private static EcrAsyncClient getAsyncClient() {

        /*
         The `NettyNioAsyncHttpClient` class is part of the AWS SDK for Java,
version 2,
         and it is designed to provide a high-performance, asynchronous HTTP
client for interacting with AWS services.
         It uses the Netty framework to handle the underlying network
communication and the Java NIO API to
         provide a non-blocking, event-driven approach to HTTP requests and
responses.
         */
        SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
            .maxConcurrency(50)  // Adjust as needed.
            .connectionTimeout(Duration.ofSeconds(60))  // Set the connection
timeout.
            .readTimeout(Duration.ofSeconds(60))  // Set the read timeout.
            .writeTimeout(Duration.ofSeconds(60))  // Set the write timeout.
            .build();

        ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
            .apiCallTimeout(Duration.ofMinutes(2))  // Set the overall API call
timeout.
            .apiCallAttemptTimeout(Duration.ofSeconds(90))  // Set the individual
call attempt timeout.
            .build();

        if (ecrClient == null) {
            ecrClient = EcrAsyncClient.builder()
                .region(Region.US_EAST_1)
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return ecrClient;
    }

    /**
     * Sets the lifecycle policy for the specified repository.
     *
```

```
     * @param repoName the name of the repository for which to set the lifecycle
 policy.
     */
    public void setLifeCyclePolicy(String repoName) {
        /*
           This policy helps to maintain the size and efficiency of the container
 registry
           by automatically removing older and potentially unused images,
           ensuring that the storage is optimized and the registry remains up-to-
date.
         */
        String polText = """
             {
             "rules": [
                 {
                     "rulePriority": 1,
                     "description": "Expire images older than 14 days",
                     "selection": {
                         "tagStatus": "any",
                         "countType": "sinceImagePushed",
                         "countUnit": "days",
                         "countNumber": 14
                     },
                     "action": {
                         "type": "expire"
                     }
                 }
            ]
            }
            """;

        StartLifecyclePolicyPreviewRequest lifecyclePolicyPreviewRequest =
 StartLifecyclePolicyPreviewRequest.builder()
            .lifecyclePolicyText(polText)
            .repositoryName(repoName)
            .build();

        CompletableFuture<StartLifecyclePolicyPreviewResponse> response =
 getAsyncClient().startLifecyclePolicyPreview(lifecyclePolicyPreviewRequest);
        response.whenComplete((lifecyclePolicyPreviewResponse, ex) -> {
            if (lifecyclePolicyPreviewResponse != null) {
                System.out.println("Lifecycle policy preview started
 successfully.");
            } else {
```

```
                if (ex.getCause() instanceof EcrException) {
                    throw (EcrException) ex.getCause();
                } else {
                    String errorMessage = "Unexpected error occurred: " +
 ex.getMessage();
                    throw new RuntimeException(errorMessage, ex);
                }
            }
        });
        // Wait for the CompletableFuture to complete.
        response.join();
    }

    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
 (Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     * @throws EcrException             if there is an error retrieving the image
 information from Amazon ECR.
     * @throws CompletionException      if the asynchronous operation completes
 exceptionally.
     */
    public void verifyImage(String repositoryName, String imageTag) {
        DescribeImagesRequest request = DescribeImagesRequest.builder()
            .repositoryName(repositoryName)
            .imageIds(ImageIdentifier.builder().imageTag(imageTag).build())
            .build();

        CompletableFuture<DescribeImagesResponse> response =
 getAsyncClient().describeImages(request);
        response.whenComplete((describeImagesResponse, ex) -> {
            if (ex != null) {
                if (ex instanceof CompletionException) {
                    Throwable cause = ex.getCause();
                    if (cause instanceof EcrException) {
                        throw (EcrException) cause;
                    } else {
                        throw new RuntimeException("Unexpected error: " +
 cause.getMessage(), cause);
                    }
                } else {
```

```
                    throw new RuntimeException("Unexpected error: " +
 ex.getCause());
                }
            } else if (describeImagesResponse != null && !
describeImagesResponse.imageDetails().isEmpty()) {
                System.out.println("Image is present in the repository.");
            } else {
                System.out.println("Image is not present in the repository.");
            }
        });

        // Wait for the CompletableFuture to complete.
        response.join();
    }

    /**
     * Retrieves the repository URI for the specified repository name.
     *
     * @param repoName the name of the repository to retrieve the URI for.
     * @return the repository URI for the specified repository name.
     * @throws EcrException        if there is an error retrieving the repository
 information.
     * @throws CompletionException if the asynchronous operation completes
 exceptionally.
     */
    public void getRepositoryURI(String repoName) {
        DescribeRepositoriesRequest request =
 DescribeRepositoriesRequest.builder()
            .repositoryNames(repoName)
            .build();

        CompletableFuture<DescribeRepositoriesResponse> response =
 getAsyncClient().describeRepositories(request);
        response.whenComplete((describeRepositoriesResponse, ex) -> {
            if (ex != null) {
                Throwable cause = ex.getCause();
                if (cause instanceof InterruptedException) {
                    Thread.currentThread().interrupt();
                    String errorMessage = "Thread interrupted while waiting for
 asynchronous operation: " + cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                } else if (cause instanceof EcrException) {
                    throw (EcrException) cause;
                } else {
```

```
                    String errorMessage = "Unexpected error: " +
cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                }
            } else {
                if (describeRepositoriesResponse != null) {
                    if (!describeRepositoriesResponse.repositories().isEmpty()) {
                        String repositoryUri =
describeRepositoriesResponse.repositories().get(0).repositoryUri();
                        System.out.println("Repository URI found: " +
repositoryUri);
                    } else {
                        System.out.println("No repositories found for the given
name.");
                    }
                } else {
                    System.err.println("No response received from
describeRepositories.");
                }
            }
        });
        response.join();
    }

    /**
     * Retrieves the authorization token for Amazon Elastic Container Registry
(ECR).
     * This method makes an asynchronous call to the ECR client to retrieve the
authorization token.
     * If the operation is successful, the method prints the token to the
console.
     * If an exception occurs, the method handles the exception and prints the
error message.
     *
     * @throws EcrException     if there is an error retrieving the authorization
token from ECR.
     * @throws RuntimeException if there is an unexpected error during the
operation.
     */
    public void getAuthToken() {
        CompletableFuture<GetAuthorizationTokenResponse> response =
getAsyncClient().getAuthorizationToken();
        response.whenComplete((authorizationTokenResponse, ex) -> {
            if (authorizationTokenResponse != null) {
```

```
                AuthorizationData authorizationData =
authorizationTokenResponse.authorizationData().get(0);
                String token = authorizationData.authorizationToken();
                if (!token.isEmpty()) {
                    System.out.println("The token was successfully retrieved.");
                }
            } else {
                if (ex.getCause() instanceof EcrException) {
                    throw (EcrException) ex.getCause();
                } else {
                    String errorMessage = "Unexpected error occurred: " +
ex.getMessage();
                    throw new RuntimeException(errorMessage, ex); // Rethrow the
exception
                }
            }
        });
        response.join();
    }

    /**
     * Gets the repository policy for the specified repository.
     *
     * @param repoName the name of the repository.
     * @throws EcrException if an AWS error occurs while getting the repository
policy.
     */
    public String getRepoPolicy(String repoName) {
        if (repoName == null || repoName.isEmpty()) {
            throw new IllegalArgumentException("Repository name cannot be null or
empty");
        }

        GetRepositoryPolicyRequest getRepositoryPolicyRequest =
GetRepositoryPolicyRequest.builder()
            .repositoryName(repoName)
            .build();

        CompletableFuture<GetRepositoryPolicyResponse> response =
getAsyncClient().getRepositoryPolicy(getRepositoryPolicyRequest);
        response.whenComplete((resp, ex) -> {
            if (resp != null) {
                System.out.println("Repository policy retrieved successfully.");
            } else {
```

```
                if (ex.getCause() instanceof EcrException) {
                    throw (EcrException) ex.getCause();
                } else {
                    String errorMessage = "Unexpected error occurred: " +
ex.getMessage();
                    throw new RuntimeException(errorMessage, ex);
                }
            }
        });

        GetRepositoryPolicyResponse result = response.join();
        return result != null ? result.policyText() : null;
    }

    /**
     * Sets the repository policy for the specified ECR repository.
     *
     * @param repoName the name of the ECR repository.
     * @param iamRole  the IAM role to be granted access to the repository.
     * @throws RepositoryPolicyNotFoundException if the repository policy does
not exist.
     * @throws EcrException                      if there is an unexpected error
setting the repository policy.
     */
    public void setRepoPolicy(String repoName, String iamRole) {
        /*
          This example policy document grants the specified AWS principal the
permission to perform the
          `ecr:BatchGetImage` action. This policy is designed to allow the
specified principal
          to retrieve Docker images from the ECR repository.
         */
        String policyDocumentTemplate = """
             {
              "Version" : "2012-10-17",
              "Statement" : [ {
                "Sid" : "new statement",
                "Effect" : "Allow",
                "Principal" : {
                  "AWS" : "%s"
                },
                "Action" : "ecr:BatchGetImage"
              } ]
            }
```

```
             """;

        String policyDocument = String.format(policyDocumentTemplate, iamRole);
        SetRepositoryPolicyRequest setRepositoryPolicyRequest =
SetRepositoryPolicyRequest.builder()
            .repositoryName(repoName)
            .policyText(policyDocument)
            .build();

        CompletableFuture<SetRepositoryPolicyResponse> response =
getAsyncClient().setRepositoryPolicy(setRepositoryPolicyRequest);
        response.whenComplete((resp, ex) -> {
            if (resp != null) {
                System.out.println("Repository policy set successfully.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof RepositoryPolicyNotFoundException) {
                    throw (RepositoryPolicyNotFoundException) cause;
                } else if (cause instanceof EcrException) {
                    throw (EcrException) cause;
                } else {
                    String errorMessage = "Unexpected error: " +
cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                }
            }
        });
        response.join();
    }

    /**
     * Pushes a Docker image to an Amazon Elastic Container Registry (ECR)
repository.
     *
     * @param repoName  the name of the ECR repository to push the image to.
     * @param imageName the name of the Docker image.
     */
    public void pushDockerImage(String repoName, String imageName) {
        System.out.println("Pushing " + imageName + " to Amazon ECR will take a
few seconds.");
        CompletableFuture<AuthConfig> authResponseFuture =
getAsyncClient().getAuthorizationToken()
            .thenApply(response -> {
```

```
                String token =
 response.authorizationData().get(0).authorizationToken();
                String decodedToken = new
 String(Base64.getDecoder().decode(token));
                String password = decodedToken.substring(4);

                DescribeRepositoriesResponse descrRepoResponse =
 getAsyncClient().describeRepositories(b -> b.repositoryNames(repoName)).join();
                Repository repoData =
 descrRepoResponse.repositories().stream().filter(r ->
 r.repositoryName().equals(repoName)).findFirst().orElse(null);
                assert repoData != null;
                String registryURL = repoData.repositoryUri().split("/")[0];

                AuthConfig authConfig = new AuthConfig()
                    .withUsername("AWS")
                    .withPassword(password)
                    .withRegistryAddress(registryURL);
                return authConfig;
            })
            .thenCompose(authConfig -> {
                DescribeRepositoriesResponse descrRepoResponse =
 getAsyncClient().describeRepositories(b -> b.repositoryNames(repoName)).join();
                Repository repoData =
 descrRepoResponse.repositories().stream().filter(r ->
 r.repositoryName().equals(repoName)).findFirst().orElse(null);
                getDockerClient().tagImageCmd(imageName + ":latest",
 repoData.repositoryUri() + ":latest", imageName).exec();
                try {

 getDockerClient().pushImageCmd(repoData.repositoryUri()).withTag("echo-
text").withAuthConfig(authConfig).start().awaitCompletion();
                    System.out.println("The " + imageName + " was pushed to
 ECR");

                } catch (InterruptedException e) {
                    throw (RuntimeException) e.getCause();
                }
                return CompletableFuture.completedFuture(authConfig);
            });

        authResponseFuture.join();
    }
```

```
    // Make sure local image echo-text exists.
    public boolean isEchoTextImagePresent() {
        try {
            List<Image> images = getDockerClient().listImagesCmd().exec();
            boolean helloWorldFound = false;
            for (Image image : images) {
                String[] repoTags = image.getRepoTags();
                if (repoTags != null) {
                    for (String tag : repoTags) {
                        if (tag.startsWith("echo-text")) {
                            System.out.println(tag);
                            helloWorldFound = true;
                        }
                    }
                }
            }
            if (helloWorldFound) {
                System.out.println("The local image named echo-text exists.");
                return true;
            } else {
                System.out.println("The local image named echo-text does not
 exist.");
                return false;
            }
        } catch (DockerClientException ex) {
            logger.error("ERROR: " + ex.getMessage());
            return false;
        }
    }
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for Java 2.x API 参考》中的以下主题。
  - CreateRepository
  - DeleteRepository
  - DescribeImages
  - DescribeRepositories
  - GetAuthorizationToken
  - GetRepositoryPolicy
  - SetRepositoryPolicy

- StartLifecyclePolicyPreview

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

运行一个交互式场景，演示 Amazon ECR 的功能。

```
import java.util.Scanner

/**
 * Before running this Kotlin code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 *
 * This code example requires an IAM Role that has permissions to interact with
 the Amazon ECR service.
 *
 * To create an IAM role, see:
 *
 * https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create.html
 *
 * This code example requires a local docker image named echo-text. Without a
 local image,
 * this program will not successfully run. For more information including how to
 create the local
 * image, see:
 *
 * /scenarios/basics/ecr/README
 *
 */

val DASHES = String(CharArray(80)).replace("\u0000", "-")
```

```
suspend fun main(args: Array<String>) {
    val usage =
        """
        Usage: <iamRoleARN> <accountId>

        Where:
           iamRoleARN - The IAM role ARN that has the necessary permissions to
 access and manage the Amazon ECR repository.
           accountId - Your AWS account number.

        """.trimIndent()

    if (args.size != 2) {
        println(usage)
        return
    }

    var iamRole = args[0]
    var localImageName: String
    var accountId = args[1]
    val ecrActions = ECRActions()
    val scanner = Scanner(System.`in`)

    println(
        """
        The Amazon Elastic Container Registry (ECR) is a fully-managed Docker
 container registry
        service provided by AWS. It allows developers and organizations to
 securely
        store, manage, and deploy Docker container images.
        ECR provides a simple and scalable way to manage container images
 throughout their lifecycle,
        from building and testing to production deployment.

        The `EcrClient` service client that is part of the AWS SDK for Kotlin
 provides a set of methods to
        programmatically interact with the Amazon ECR service. This allows
 developers to
        automate the storage, retrieval, and management of container images as
 part of their application
        deployment pipelines. With ECR, teams can focus on building and deploying
 their
```

```
        applications without having to worry about the underlying infrastructure
 required to
        host and manage a container registry.

        This scenario walks you through how to perform key operations for this
 service.
        Let's get started...

         You have two choices:
            1 - Run the entire program.
            2 - Delete an existing Amazon ECR repository named echo-text (created
 from a previous execution of
            this program that did not complete).

        """.trimIndent(),
    )

    while (true) {
        val input = scanner.nextLine()
        if (input.trim { it <= ' ' }.equals("1", ignoreCase = true)) {
            println("Continuing with the program...")
            println("")
            break
        } else if (input.trim { it <= ' ' }.equals("2", ignoreCase = true)) {
            val repoName = "echo-text"
            ecrActions.deleteECRRepository(repoName)
            return
        } else {
            // Handle invalid input.
            println("Invalid input. Please try again.")
        }
    }

    waitForInputToContinue(scanner)
    println(DASHES)
    println(
        """
        1. Create an ECR repository.

        The first task is to ensure we have a local Docker image named echo-
text.
        If this image exists, then an Amazon ECR repository is created.

        An ECR repository is a private Docker container repository provided
```

```
        by Amazon Web Services (AWS). It is a managed service that makes it easy
        to store, manage, and deploy Docker container images.

        """.trimIndent(),
    )

    // Ensure that a local docker image named echo-text exists.
    val doesExist = ecrActions.listLocalImages()
    val repoName: String
    if (!doesExist) {
        println("The local image named echo-text does not exist")
        return
    } else {
        localImageName = "echo-text"
        repoName = "echo-text"
    }

    val repoArn = ecrActions.createECRRepository(repoName).toString()
    println("The ARN of the ECR repository is $repoArn")
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        2. Set an ECR repository policy.

        Setting an ECR repository policy using the `setRepositoryPolicy` function
 is crucial for maintaining
        the security and integrity of your container images. The repository
 policy allows you to
        define specific rules and restrictions for accessing and managing the
 images stored within your ECR
        repository.

        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    ecrActions.setRepoPolicy(repoName, iamRole)
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        3. Display ECR repository policy.
```

```
        Now we will retrieve the ECR policy to ensure it was successfully set.
        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    val policyText = ecrActions.getRepoPolicy(repoName)
    println("Policy Text:")
    println(policyText)
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        4. Retrieve an ECR authorization token.

        You need an authorization token to securely access and interact with the
 Amazon ECR registry.
        The `getAuthorizationToken` method of the `EcrAsyncClient` is responsible
 for securely accessing
        and interacting with an Amazon ECR repository. This operation is
 responsible for obtaining a
        valid authorization token, which is required to authenticate your
 requests to the ECR service.

        Without a valid authorization token, you would not be able to perform any
 operations on the
        ECR repository, such as pushing, pulling, or managing your Docker images.


        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    ecrActions.getAuthToken()
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        5. Get the ECR Repository URI.

        The URI  of an Amazon ECR repository is important. When you want to
 deploy a container image to
        a container orchestration platform like Amazon Elastic Kubernetes Service
 (EKS)
```

```
        or Amazon Elastic Container Service (ECS), you need to specify the full 
image URI,
        which includes the ECR repository URI. This allows the container runtime 
to pull the
        correct container image from the ECR repository.

        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    val repositoryURI: String? = ecrActions.getRepositoryURI(repoName)
    println("The repository URI is $repositoryURI")
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        6. Set an ECR Lifecycle Policy.

        An ECR Lifecycle Policy is used to manage the lifecycle of Docker images 
stored in your ECR repositories.
        These policies allow you to automatically remove old or unused Docker 
images from your repositories,
        freeing up storage space and reducing costs.

        """.trimIndent(),
    )
    waitForInputToContinue(scanner)
    val pol = ecrActions.setLifeCyclePolicy(repoName)
    println(pol)
    waitForInputToContinue(scanner)

    println(DASHES)
    println(
        """
        7. Push a docker image to the Amazon ECR Repository.

        The `pushImageCmd()` method pushes a local Docker image to an Amazon ECR 
repository.
        It sets up the Docker client by connecting to the local Docker host using 
the default port.
        It then retrieves the authorization token for the ECR repository by 
making a call to the AWS SDK.
```

```
        The method uses the authorization token to create an `AuthConfig` object,
 which is used to authenticate
        the Docker client when pushing the image. Finally, the method tags the
 Docker image with the specified
        repository name and image tag, and then pushes the image to the ECR
 repository using the Docker client.
        If the push operation is successful, the method prints a message
 indicating that the image was pushed to ECR.

        """.trimIndent(),
    )

    waitForInputToContinue(scanner)
    ecrActions.pushDockerImage(repoName, localImageName)
    waitForInputToContinue(scanner)

    println(DASHES)
    println("8. Verify if the image is in the ECR Repository.")
    waitForInputToContinue(scanner)
    ecrActions.verifyImage(repoName, localImageName)
    waitForInputToContinue(scanner)

    println(DASHES)
    println("9. As an optional step, you can interact with the image in Amazon
 ECR by using the CLI.")
    println("Would you like to view instructions on how to use the CLI to run the
 image? (y/n)")
    val ans = scanner.nextLine().trim()
    if (ans.equals("y", true)) {
        val instructions = """
        1. Authenticate with ECR - Before you can pull the image from Amazon ECR,
 you need to authenticate with the registry. You can do this using the AWS CLI:

            aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin $accountId.dkr.ecr.us-east-1.amazonaws.com

        2. Describe the image using this command:

           aws ecr describe-images --repository-name $repoName --image-ids
 imageTag=$localImageName

        3. Run the Docker container and view the output using this command:
```

```
            docker run --rm $accountId.dkr.ecr.us-east-1.amazonaws.com/$repoName:
$localImageName
        """
        println(instructions)
    }
    waitForInputToContinue(scanner)

    println(DASHES)
    println("10. Delete the ECR Repository.")
    println(
        """
        If the repository isn't empty, you must either delete the contents of the
 repository
        or use the force option (used in this scenario) to delete the repository
 and have Amazon ECR delete all of its contents
        on your behalf.

        """.trimIndent(),
    )
    println("Would you like to delete the Amazon ECR Repository? (y/n)")
    val delAns = scanner.nextLine().trim { it <= ' ' }
    if (delAns.equals("y", ignoreCase = true)) {
        println("You selected to delete the AWS ECR resources.")
        waitForInputToContinue(scanner)
        ecrActions.deleteECRRepository(repoName)
    }

    println(DASHES)
    println("This concludes the Amazon ECR SDK scenario")
    println(DASHES)
}

private fun waitForInputToContinue(scanner: Scanner) {
    while (true) {
        println("")
        println("Enter 'c' followed by <ENTER> to continue:")
        val input = scanner.nextLine()
        if (input.trim { it <= ' ' }.equals("c", ignoreCase = true)) {
            println("Continuing with the program...")
            println("")
            break
        } else {
            // Handle invalid input.
            println("Invalid input. Please try again.")
```

```
        }
    }
}
```

Amazon ECR SDK 方法的封装类。

```
import aws.sdk.kotlin.services.ecr.EcrClient
import aws.sdk.kotlin.services.ecr.model.CreateRepositoryRequest
import aws.sdk.kotlin.services.ecr.model.DeleteRepositoryRequest
import aws.sdk.kotlin.services.ecr.model.DescribeImagesRequest
import aws.sdk.kotlin.services.ecr.model.DescribeRepositoriesRequest
import aws.sdk.kotlin.services.ecr.model.EcrException
import aws.sdk.kotlin.services.ecr.model.GetRepositoryPolicyRequest
import aws.sdk.kotlin.services.ecr.model.ImageIdentifier
import aws.sdk.kotlin.services.ecr.model.RepositoryAlreadyExistsException
import aws.sdk.kotlin.services.ecr.model.SetRepositoryPolicyRequest
import aws.sdk.kotlin.services.ecr.model.StartLifecyclePolicyPreviewRequest
import com.github.dockerjava.api.DockerClient
import com.github.dockerjava.api.command.DockerCmdExecFactory
import com.github.dockerjava.api.model.AuthConfig
import com.github.dockerjava.core.DockerClientBuilder
import com.github.dockerjava.netty.NettyDockerCmdExecFactory
import java.io.IOException
import java.util.Base64

class ECRActions {
    private var dockerClient: DockerClient? = null

    private fun getDockerClient(): DockerClient? {
        val osName = System.getProperty("os.name")
        if (osName.startsWith("Windows")) {
            // Make sure Docker Desktop is running.
            val dockerHost = "tcp://localhost:2375" // Use the Docker Desktop
 default port.
            val dockerCmdExecFactory: DockerCmdExecFactory =

 NettyDockerCmdExecFactory().withReadTimeout(20000).withConnectTimeout(20000)
            dockerClient =
 DockerClientBuilder.getInstance(dockerHost).withDockerCmdExecFactory(dockerCmdExecFactor
        } else {
            dockerClient = DockerClientBuilder.getInstance().build()
        }
```

```
        return dockerClient
    }



    /**
     * Sets the lifecycle policy for the specified repository.
     *
     * @param repoName the name of the repository for which to set the lifecycle
 policy.
     */
    suspend fun setLifeCyclePolicy(repoName: String): String? {
        val polText =
            """
             {
             "rules": [
                 {
                     "rulePriority": 1,
                     "description": "Expire images older than 14 days",
                     "selection": {
                         "tagStatus": "any",
                         "countType": "sinceImagePushed",
                         "countUnit": "days",
                         "countNumber": 14
                     },
                     "action": {
                         "type": "expire"
                     }
                 }
            ]
            }

            """.trimIndent()
        val lifecyclePolicyPreviewRequest =
            StartLifecyclePolicyPreviewRequest {
                lifecyclePolicyText = polText
                repositoryName = repoName
            }

        // Execute the request asynchronously.
        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val response =
 ecrClient.startLifecyclePolicyPreview(lifecyclePolicyPreviewRequest)
            return response.lifecyclePolicyText
        }
```

```
    }


    /**
     * Retrieves the repository URI for the specified repository name.
     *
     * @param repoName the name of the repository to retrieve the URI for.
     * @return the repository URI for the specified repository name.
     */
    suspend fun getRepositoryURI(repoName: String?): String? {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
 cannot be null or empty" }
        val request =
            DescribeRepositoriesRequest {
                repositoryNames = listOf(repoName)
            }

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val describeRepositoriesResponse =
 ecrClient.describeRepositories(request)
            if (!describeRepositoriesResponse.repositories?.isEmpty()!!) {
                return
 describeRepositoriesResponse?.repositories?.get(0)?.repositoryUri
            } else {
                println("No repositories found for the given name.")
                return ""
            }
        }
    }


    /**
     * Retrieves the authorization token for Amazon Elastic Container Registry
 (ECR).
     *
     */
    suspend fun getAuthToken() {
        EcrClient { region = "us-east-1" }.use { ecrClient ->
            // Retrieve the authorization token for ECR.
            val response = ecrClient.getAuthorizationToken()
            val authorizationData = response.authorizationData?.get(0)
            val token = authorizationData?.authorizationToken
            if (token != null) {
                println("The token was successfully retrieved.")
```

```
            }
        }
    }


    /**
     * Gets the repository policy for the specified repository.
     *
     * @param repoName the name of the repository.
     */
    suspend fun getRepoPolicy(repoName: String?): String? {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name 
cannot be null or empty" }

        // Create the request
        val getRepositoryPolicyRequest =
            GetRepositoryPolicyRequest {
                repositoryName = repoName
            }
        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val response = 
ecrClient.getRepositoryPolicy(getRepositoryPolicyRequest)
            val responseText = response.policyText
            return responseText
        }
    }


    /**
     * Sets the repository policy for the specified ECR repository.
     *
     * @param repoName the name of the ECR repository.
     * @param iamRole the IAM role to be granted access to the repository.
     */
    suspend fun setRepoPolicy(
        repoName: String?,
        iamRole: String?,
    ) {
        val policyDocumentTemplate =
            """
             {
              "Version" : "2012-10-17",
              "Statement" : [ {
                "Sid" : "new statement",
```

```
                 "Effect" : "Allow",
                 "Principal" : {
                   "AWS" : "$iamRole"
                 },
                 "Action" : "ecr:BatchGetImage"
               } ]
             }

             """.trimIndent()
         val setRepositoryPolicyRequest =
             SetRepositoryPolicyRequest {
                 repositoryName = repoName
                 policyText = policyDocumentTemplate
             }

         EcrClient { region = "us-east-1" }.use { ecrClient ->
             val response =
 ecrClient.setRepositoryPolicy(setRepositoryPolicyRequest)
             if (response != null) {
                 println("Repository policy set successfully.")
             }
         }
     }



     /**
      * Creates an Amazon Elastic Container Registry (Amazon ECR) repository.
      *
      * @param repoName the name of the repository to create.
      * @return the Amazon Resource Name (ARN) of the created repository, or an
 empty string if the operation failed.
      * @throws RepositoryAlreadyExistsException if the repository exists.
      * @throws EcrException         if an error occurs while creating the
 repository.
      */
     suspend fun createECRRepository(repoName: String?): String? {
         val request =
             CreateRepositoryRequest {
                 repositoryName = repoName
             }

         return try {
             EcrClient { region = "us-east-1" }.use { ecrClient ->
                 val response = ecrClient.createRepository(request)
```

```
                response.repository?.repositoryArn
            }
        } catch (e: RepositoryAlreadyExistsException) {
            println("Repository already exists: $repoName")
            repoName?.let { getRepoARN(it) }
        } catch (e: EcrException) {
            println("An error occurred: ${e.message}")
            null
        }
    }

    suspend fun getRepoARN(repoName: String): String? {
        // Fetch the existing repository's ARN.
        val describeRequest =
            DescribeRepositoriesRequest {
                repositoryNames = listOf(repoName)
            }
        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val describeResponse =
 ecrClient.describeRepositories(describeRequest)
            return describeResponse.repositories?.get(0)?.repositoryArn
        }
    }

    fun listLocalImages(): Boolean = try {
        val images = getDockerClient()?.listImagesCmd()?.exec()
        images?.any { image ->
            image.repoTags?.any { tag -> tag.startsWith("echo-text") } ?: false
        } ?: false
    } catch (ex: Exception) {
        println("ERROR: ${ex.message}")
        false
    }


    /**
     * Pushes a Docker image to an Amazon Elastic Container Registry (ECR)
 repository.
     *
     * @param repoName the name of the ECR repository to push the image to.
     * @param imageName the name of the Docker image.
     */
    suspend fun pushDockerImage(
        repoName: String,
```

```
        imageName: String,
    ) {
        println("Pushing $imageName to $repoName will take a few seconds")
        val authConfig = getAuthConfig(repoName)

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val desRequest =
                DescribeRepositoriesRequest {
                    repositoryNames = listOf(repoName)
                }

            val describeRepoResponse = ecrClient.describeRepositories(desRequest)
            val repoData =
                describeRepoResponse.repositories?.firstOrNull
{ it.repositoryName == repoName }
                    ?: throw RuntimeException("Repository not found: $repoName")

            val tagImageCmd = getDockerClient()?.tagImageCmd("$imageName",
"${repoData.repositoryUri}", imageName)
            if (tagImageCmd != null) {
                tagImageCmd.exec()
            }
            val pushImageCmd =
                repoData.repositoryUri?.let {
                    dockerClient?.pushImageCmd(it)
                        // ?.withTag("latest")
                        ?.withAuthConfig(authConfig)
                }

            try {
                if (pushImageCmd != null) {
                    pushImageCmd.start().awaitCompletion()
                }
                println("The $imageName was pushed to Amazon ECR")
            } catch (e: IOException) {
                throw RuntimeException(e)
            }
        }
    }


    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
(Amazon ECR) repository asynchronously.
```

```
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     */
    suspend fun verifyImage(
        repoName: String?,
        imageTagVal: String?,
    ) {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
 cannot be null or empty" }
        require(!(imageTagVal == null || imageTagVal.isEmpty())) { "Image tag
 cannot be null or empty" }

        val imageId =
            ImageIdentifier {
                imageTag = imageTagVal
            }
        val request =
            DescribeImagesRequest {
                repositoryName = repoName
                imageIds = listOf(imageId)
            }

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val describeImagesResponse = ecrClient.describeImages(request)
            if (describeImagesResponse != null && !
describeImagesResponse.imageDetails?.isEmpty()!!) {
                println("Image is present in the repository.")
            } else {
                println("Image is not present in the repository.")
            }
        }
    }


    /**
     * Deletes an ECR (Elastic Container Registry) repository.
     *
     * @param repoName the name of the repository to delete.
     */
    suspend fun deleteECRRepository(repoName: String) {
        if (repoName.isNullOrEmpty()) {
            throw IllegalArgumentException("Repository name cannot be null or
 empty")
```

```
        }

        val repositoryRequest =
            DeleteRepositoryRequest {
                force = true
                repositoryName = repoName
            }

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            ecrClient.deleteRepository(repositoryRequest)
            println("You have successfully deleted the $repoName repository")
        }
    }

    // Return an AuthConfig.
    private suspend fun getAuthConfig(repoName: String): AuthConfig {
        EcrClient { region = "us-east-1" }.use { ecrClient ->
            // Retrieve the authorization token for ECR.
            val response = ecrClient.getAuthorizationToken()
            val authorizationData = response.authorizationData?.get(0)
            val token = authorizationData?.authorizationToken
            val decodedToken = String(Base64.getDecoder().decode(token))
            val password = decodedToken.substring(4)

            val request =
                DescribeRepositoriesRequest {
                    repositoryNames = listOf(repoName)
                }

            val descrRepoResponse = ecrClient.describeRepositories(request)
            val repoData = descrRepoResponse.repositories?.firstOrNull
 { it.repositoryName == repoName }
            val registryURL: String =
 repoData?.repositoryUri?.split("/")?.get(0) ?: ""

            return AuthConfig()
                .withUsername("AWS")
                .withPassword(password)
                .withRegistryAddress(registryURL)
        }
    }
}
```

- 有关 API 详细信息，请参阅《Amazon SDK for Kotlin API 参考》中的以下主题。
  - CreateRepository
  - DeleteRepository
  - DescribeImages
  - DescribeRepositories
  - GetAuthorizationToken
  - GetRepositoryPolicy
  - SetRepositoryPolicy
  - StartLifecyclePolicyPreview

Python

适用于 Python 的 SDK（Boto3）

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

在命令提示符中运行交互式场景。

```
class ECRGettingStarted:
    """
    A scenario that demonstrates how to use Boto3 to perform basic operations
 using
    Amazon ECR.
    """

    def __init__(
        self,
        ecr_wrapper: ECRWrapper,
        docker_client: docker.DockerClient,
    ):
        self.ecr_wrapper = ecr_wrapper
        self.docker_client = docker_client
        self.tag = "echo-text"
        self.repository_name = "ecr-basics"
```

```
        self.docker_image = None
        self.full_tag_name = None
        self.repository = None

    def run(self, role_arn: str) -> None:
        """
        Runs the scenario.
        """
        print(
            """
The Amazon Elastic Container Registry (ECR) is a fully-managed Docker container
 registry
service provided by AWS. It allows developers and organizations to securely
store, manage, and deploy Docker container images.
ECR provides a simple and scalable way to manage container images throughout
 their lifecycle,
from building and testing to production deployment.

The `ECRWrapper' class is a wrapper for the Boto3 'ecr' client. The 'ecr' client
 provides a set of methods to
programmatically interact with the Amazon ECR service. This allows developers to
automate the storage, retrieval, and management of container images as part of
 their application
deployment pipelines. With ECR, teams can focus on building and deploying their
applications without having to worry about the underlying infrastructure required
 to
host and manage a container registry.

This scenario walks you through how to perform key operations for this service.
Let's get started...
        """
        )
        press_enter_to_continue()
        print_dashes()
        print(
            f"""
* Create an ECR repository.

An ECR repository is a private Docker container repository provided
by Amazon Web Services (AWS). It is a managed service that makes it easy
to store, manage, and deploy Docker container images.
        """
        )
        print(f"Creating a repository named {self.repository_name}")
```

```
        self.repository =
 self.ecr_wrapper.create_repository(self.repository_name)
        print(f"The ARN of the ECR repository is
 {self.repository['repositoryArn']}")
        repository_uri = self.repository["repositoryUri"]
        press_enter_to_continue()
        print_dashes()

        print(
            f"""
* Build a Docker image.

Create a local Docker image if it does not already exist.
A Python Docker client is used to execute Docker commands.
You must have Docker installed and running.
            """
        )
        print(f"Building a docker image from 'docker_files/Dockerfile'")
        self.full_tag_name = f"{repository_uri}:{self.tag}"
        self.docker_image = self.docker_client.images.build(
            path="docker_files", tag=self.full_tag_name
        )[0]
        print(f"Docker image {self.full_tag_name} successfully built.")
        press_enter_to_continue()
        print_dashes()

        if role_arn is None:
            print(
                """
* Because an IAM role ARN was not provided, a role policy will not be set for
 this repository.
            """
            )
        else:
            print(
                """
* Set an ECR repository policy.

Setting an ECR repository policy using the `setRepositoryPolicy` function is
 crucial for maintaining
the security and integrity of your container images. The repository policy allows
 you to
define specific rules and restrictions for accessing and managing the images
 stored within your ECR
```

```
repository.
        """
            )

            self.grant_role_download_access(role_arn)
            print(f"Download access granted to the IAM role ARN {role_arn}")
            press_enter_to_continue()
            print_dashes()

            print(
                """
* Display ECR repository policy.

Now we will retrieve the ECR policy to ensure it was successfully set.
            """
            )

            policy_text =
 self.ecr_wrapper.get_repository_policy(self.repository_name)
            print("Policy Text:")
            print(f"{policy_text}")
            press_enter_to_continue()
            print_dashes()

        print(
            """
* Retrieve an ECR authorization token.

You need an authorization token to securely access and interact with the Amazon
 ECR registry.
The `get_authorization_token` method of the `ecr` client is responsible for
 securely accessing
and interacting with an Amazon ECR repository. This operation is responsible for
 obtaining a
valid authorization token, which is required to authenticate your requests to the
 ECR service.

Without a valid authorization token, you would not be able to perform any
 operations on the
ECR repository, such as pushing, pulling, or managing your Docker images.
        """
        )

        authorization_token = self.ecr_wrapper.get_authorization_token()
```

```
        print("Authorization token retrieved.")
        press_enter_to_continue()
        print_dashes()
        print(
            """
* Get the ECR Repository URI.

The URI  of an Amazon ECR repository is important. When you want to deploy a
 container image to
a container orchestration platform like Amazon Elastic Kubernetes Service (EKS)
or Amazon Elastic Container Service (ECS), you need to specify the full image
 URI,
which includes the ECR repository URI. This allows the container runtime to pull
 the
correct container image from the ECR repository.
        """
        )
        repository_descriptions = self.ecr_wrapper.describe_repositories(
            [self.repository_name]
        )
        repository_uri = repository_descriptions[0]["repositoryUri"]
        print(f"Repository URI found: {repository_uri}")
        press_enter_to_continue()
        print_dashes()

        print(
            """
* Set an ECR Lifecycle Policy.

An ECR Lifecycle Policy is used to manage the lifecycle of Docker images stored
 in your ECR repositories.
These policies allow you to automatically remove old or unused Docker images from
 your repositories,
freeing up storage space and reducing costs.

This example policy helps to maintain the size and efficiency of the container
 registry
by automatically removing older and potentially unused images, ensuring that the
storage is optimized and the registry remains up-to-date.
            """
        )
        press_enter_to_continue()
        self.put_expiration_policy()
        print(f"An expiration policy was added to the repository.")
```

```
        print_dashes()

        print(
            """
* Push a docker image to the Amazon ECR Repository.

The Docker client uses the authorization token is used to authenticate the when
 pushing the image to the
ECR repository.
        """
        )
        decoded_authorization =
 base64.b64decode(authorization_token).decode("utf-8")
        username, password = decoded_authorization.split(":")

        resp = self.docker_client.api.push(
            repository=repository_uri,
            auth_config={"username": username, "password": password},
            tag=self.tag,
            stream=True,
            decode=True,
        )
        for line in resp:
            print(line)

        print_dashes()

        print("* Verify if the image is in the ECR Repository.")
        image_descriptions = self.ecr_wrapper.describe_images(
            self.repository_name, [self.tag]
        )
        if len(image_descriptions) > 0:
            print("Image found in ECR Repository.")
        else:
            print("Image not found in ECR Repository.")
        press_enter_to_continue()
        print_dashes()

        print(
            "* As an optional step, you can interact with the image in Amazon ECR
 by using the CLI."
        )
        if q.ask(
```

```
            "Would you like to view instructions on how to use the CLI to run the
 image? (y/n)",
            q.is_yesno,
        ):
            print(
                f"""
1. Authenticate with ECR - Before you can pull the image from Amazon ECR, you
 need to authenticate with the registry. You can do this using the AWS CLI:

    aws ecr get-login-password --region us-east-1 | docker login --username AWS
 --password-stdin {repository_uri.split("/")[0]}

2. Describe the image using this command:

   aws ecr describe-images --repository-name {self.repository_name} --image-ids
 imageTag={self.tag}

3. Run the Docker container and view the output using this command:

   docker run --rm {self.full_tag_name}
"""
            )

        self.cleanup(True)

    def cleanup(self, ask: bool):
        """
        Deletes the resources created in this scenario.
        :param ask: If True, prompts the user to confirm before deleting the
 resources.
        """
        if self.repository is not None and (
            not ask
            or q.ask(
                f"Would you like to delete the ECR repository
 '{self.repository_name}? (y/n) "
            )
        ):
            print(f"Deleting the ECR repository '{self.repository_name}'.")
            self.ecr_wrapper.delete_repository(self.repository_name)

        if self.full_tag_name is not None and (
            not ask
            or q.ask(
```

```
                f"Would you like to delete the local Docker image
'{self.full_tag_name}? (y/n) "
            )
        ):
            print(f"Deleting the docker image '{self.full_tag_name}'.")
            self.docker_client.images.remove(self.full_tag_name)

    def grant_role_download_access(self, role_arn: str):
        """
        Grants the specified role access to download images from the ECR
repository.

        :param role_arn: The ARN of the role to grant access to.
        """
        policy_json = {
            "Version": "2008-10-17",
            "Statement": [
                {
                    "Sid": "AllowDownload",
                    "Effect": "Allow",
                    "Principal": {"AWS": role_arn},
                    "Action": ["ecr:BatchGetImage"],
                }
            ],
        }

        self.ecr_wrapper.set_repository_policy(
            self.repository_name, json.dumps(policy_json)
        )


    def put_expiration_policy(self):
        """
        Puts an expiration policy on the ECR repository.
        """
        policy_json = {
            "rules": [
                {
                    "rulePriority": 1,
                    "description": "Expire images older than 14 days",
                    "selection": {
                        "tagStatus": "any",
                        "countType": "sinceImagePushed",
                        "countUnit": "days",
```

```
                        "countNumber": 14,
                    },
                    "action": {"type": "expire"},
                }
            ]
        }

        self.ecr_wrapper.put_lifecycle_policy(
            self.repository_name, json.dumps(policy_json)
        )



if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="Run Amazon ECR getting started scenario."
    )
    parser.add_argument(
        "--iam-role-arn",
        type=str,
        default=None,
        help="an optional IAM role ARN that will be granted access to download
 images from a repository.",
        required=False,
    )
    parser.add_argument(
        "--no-art",
        action="store_true",
        help="accessibility setting that suppresses art in the console output.",
    )
    args = parser.parse_args()
    no_art = args.no_art
    iam_role_arn = args.iam_role_arn
    demo = None
    a_docker_client = None
    try:
        a_docker_client = docker.from_env()
        if not a_docker_client.ping():
            raise docker.errors.DockerException("Docker is not running.")
    except docker.errors.DockerException as err:
        logging.error(
            """
        The Python Docker client could not be created.
        Do you have Docker installed and running?
```

```
        Here is the error message:
        %s
        """,
            err,
        )
        sys.exit("Error with Docker.")
    try:
        an_ecr_wrapper = ECRWrapper.from_client()
        demo = ECRGettingStarted(an_ecr_wrapper, a_docker_client)
        demo.run(iam_role_arn)

    except Exception as exception:
        logging.exception("Something went wrong with the demo!")
        if demo is not None:
            demo.cleanup(False)
```

ECRWrapper 封装 Amazon ECR 操作的类。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def create_repository(self, repository_name: str) -> dict[str, any]:
        """
        Creates an ECR repository.

        :param repository_name: The name of the repository to create.
        :return: A dictionary of the created repository.
        """
```

```
       try:
           response = 
self.ecr_client.create_repository(repositoryName=repository_name)
           return response["repository"]
       except ClientError as err:
           if err.response["Error"]["Code"] == 
"RepositoryAlreadyExistsException":
               print(f"Repository {repository_name} already exists.")
               response = self.ecr_client.describe_repositories(
                   repositoryNames=[repository_name]
               )
               return self.describe_repositories([repository_name])[0]
           else:
               logger.error(
                   "Error creating repository %s. Here's why %s",
                   repository_name,
                   err.response["Error"]["Message"],
               )
               raise


   def delete_repository(self, repository_name: str):
       """
       Deletes an ECR repository.

       :param repository_name: The name of the repository to delete.
       """
       try:
           self.ecr_client.delete_repository(
               repositoryName=repository_name, force=True
           )
           print(f"Deleted repository {repository_name}.")
       except ClientError as err:
           logger.error(
               "Couldn't delete repository %s.. Here's why %s",
               repository_name,
               err.response["Error"]["Message"],
           )
           raise


   def set_repository_policy(self, repository_name: str, policy_text: str):
       """
       Sets the policy for an ECR repository.
```

```
        :param repository_name: The name of the repository to set the policy for.
        :param policy_text: The policy text to set.
        """
        try:
            self.ecr_client.set_repository_policy(
                repositoryName=repository_name, policyText=policy_text
            )
            print(f"Set repository policy for repository {repository_name}.")
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "RepositoryPolicyNotFoundException":
                logger.error("Repository does not exist. %s.", repository_name)
                raise
            else:
                logger.error(
                    "Couldn't set repository policy for repository %s. Here's why
 %s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise


    def get_repository_policy(self, repository_name: str) -> str:
        """
        Gets the policy for an ECR repository.

        :param repository_name: The name of the repository to get the policy for.
        :return: The policy text.
        """
        try:
            response = self.ecr_client.get_repository_policy(
                repositoryName=repository_name
            )
            return response["policyText"]
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "RepositoryPolicyNotFoundException":
                logger.error("Repository does not exist. %s.", repository_name)
                raise
            else:
                logger.error(
```

```
                "Couldn't get repository policy for repository %s. Here's why 
%s",
                repository_name,
                err.response["Error"]["Message"],
            )
            raise


    def get_authorization_token(self) -> str:
        """
        Gets an authorization token for an ECR repository.

        :return: The authorization token.
        """
        try:
            response = self.ecr_client.get_authorization_token()
            return response["authorizationData"][0]["authorizationToken"]
        except ClientError as err:
            logger.error(
                "Couldn't get authorization token. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise


    def describe_repositories(self, repository_names: list[str]) -> list[dict]:
        """
        Describes ECR repositories.

        :param repository_names: The names of the repositories to describe.
        :return: The list of repository descriptions.
        """
        try:
            response = self.ecr_client.describe_repositories(
                repositoryNames=repository_names
            )
            return response["repositories"]
        except ClientError as err:
            logger.error(
                "Couldn't describe repositories. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

```
    def put_lifecycle_policy(self, repository_name: str, lifecycle_policy_text:
str):
        """
        Puts a lifecycle policy for an ECR repository.

        :param repository_name: The name of the repository to put the lifecycle
policy for.
        :param lifecycle_policy_text: The lifecycle policy text to put.
        """
        try:
            self.ecr_client.put_lifecycle_policy(
                repositoryName=repository_name,
                lifecyclePolicyText=lifecycle_policy_text,
            )
            print(f"Put lifecycle policy for repository {repository_name}.")
        except ClientError as err:
            logger.error(
                "Couldn't put lifecycle policy for repository %s. Here's why %s",
                repository_name,
                err.response["Error"]["Message"],
            )
            raise


    def describe_images(
        self, repository_name: str, image_ids: list[str] = None
    ) -> list[dict]:
        """
        Describes ECR images.

        :param repository_name: The name of the repository to describe images
for.
        :param image_ids: The optional IDs of images to describe.
        :return: The list of image descriptions.
        """
        try:
            params = {
                "repositoryName": repository_name,
            }
            if image_ids is not None:
                params["imageIds"] = [{"imageTag": tag} for tag in image_ids]

            paginator = self.ecr_client.get_paginator("describe_images")
```

```
            image_descriptions = []
            for page in paginator.paginate(**params):
                image_descriptions.extend(page["imageDetails"])
            return image_descriptions
        except ClientError as err:
            logger.error(
                "Couldn't describe images. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

- 有关 API 详细信息，请参阅《Amazon SDK for Python (Boto3) API Reference》中的以下主题。
  - CreateRepository
  - DeleteRepository
  - DescribeImages
  - DescribeRepositories
  - GetAuthorizationToken
  - GetRepositoryPolicy
  - SetRepositoryPolicy
  - StartLifecyclePolicyPreview

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Amazon ECR 执行的操作 Amazon SDKs

以下代码示例演示了如何使用执行单个 Amazon ECR 操作。 Amazon SDKs每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅《Amazon Elastic Container Registry API 参考》https://docs.amazonaws.cn/AmazonECR/latest/APIReference/Welcome.html。

示例

- CreateRepository与 Amazon SDK 或 CLI 配合使用
- DeleteRepository与 Amazon SDK 或 CLI 配合使用
- DescribeImages与 Amazon SDK 或 CLI 配合使用
- DescribeRepositories与 Amazon SDK 或 CLI 配合使用
- GetAuthorizationToken与 Amazon SDK 或 CLI 配合使用
- GetRepositoryPolicy与 Amazon SDK 或 CLI 配合使用
- ListImages与 Amazon SDK 或 CLI 配合使用
- 与 Amazon SDK PushImageCmd 配合使用
- PutLifeCyclePolicy与 Amazon SDK 或 CLI 配合使用
- SetRepositoryPolicy与 Amazon SDK 或 CLI 配合使用
- StartLifecyclePolicyPreview与 Amazon SDK 或 CLI 配合使用

`CreateRepository`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateRepository`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

CLI

Amazon CLI

示例 1：创建存储库

以下 `create-repository` 示例将在账户默认注册表中的指定命名空间内创建存储库。

```
aws ecr create-repository \
    --repository-name project-a/sample-repo
```

输出：

```
{
    "repository": {
        "registryId": "123456789012",
```

```
        "repositoryName": "project-a/sample-repo",
        "repositoryArn": "arn:aws:ecr:us-west-2:123456789012:repository/project-
a/sample-repo"
    }
}
```

有关更多信息，请参阅《Amazon ECR 用户指南》中的创建存储库。

示例 2：创建配置了映像标签不可变性的存储库

以下 `create-repository` 示例将在账户的默认注册表中创建已配置标签不可变性的存储库。

```
aws ecr create-repository \
    --repository-name project-a/sample-repo \
    --image-tag-mutability IMMUTABLE
```

输出：

```
{
    "repository": {
        "registryId": "123456789012",
        "repositoryName": "project-a/sample-repo",
        "repositoryArn": "arn:aws:ecr:us-west-2:123456789012:repository/project-
a/sample-repo",
        "imageTagMutability": "IMMUTABLE"
    }
}
```

有关更多信息，请参阅《Amazon ECR 用户指南》中的映像标签可变性。

示例 3：创建已配置扫描配置的存储库

以下 `create-repository` 示例将在账户的默认注册表中创建配置为在映像推送中执行漏洞扫描的存储库。

```
aws ecr create-repository \
    --repository-name project-a/sample-repo \
    --image-scanning-configuration scanOnPush=true
```

输出：

```
{
```

```
    "repository": {
        "registryId": "123456789012",
        "repositoryName": "project-a/sample-repo",
        "repositoryArn": "arn:aws:ecr:us-west-2:123456789012:repository/project-
a/sample-repo",
        "imageScanningConfiguration": {
            "scanOnPush": true
        }
    }
}
```

有关更多信息，请参阅《Amazon ECR 用户指南》中的映像扫描。

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考CreateRepository中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Creates an Amazon Elastic Container Registry (Amazon ECR) repository.
     *
     * @param repoName the name of the repository to create.
     * @return the Amazon Resource Name (ARN) of the created repository, or an
 empty string if the operation failed.
     * @throws IllegalArgumentException     If repository name is invalid.
     * @throws RuntimeException             if an error occurs while creating the
 repository.
     */
    public String createECRRepository(String repoName) {
        if (repoName == null || repoName.isEmpty()) {
            throw new IllegalArgumentException("Repository name cannot be null or
 empty");
        }

        CreateRepositoryRequest request = CreateRepositoryRequest.builder()
```

```
            .repositoryName(repoName)
            .build();

        CompletableFuture<CreateRepositoryResponse> response = 
getAsyncClient().createRepository(request);
        try {
            CreateRepositoryResponse result = response.join();
            if (result != null) {
                System.out.println("The " + repoName + " repository was created 
successfully.");
                return result.repository().repositoryArn();
            } else {
                throw new RuntimeException("Unexpected response type");
            }
        } catch (CompletionException e) {
            Throwable cause = e.getCause();
            if (cause instanceof EcrException ex) {
                if 
("RepositoryAlreadyExistsException".equals(ex.awsErrorDetails().errorCode())) {
                    System.out.println("The Amazon ECR repository already exists, 
moving on...");
                    DescribeRepositoriesRequest describeRequest = 
DescribeRepositoriesRequest.builder()
                        .repositoryNames(repoName)
                        .build();
                    DescribeRepositoriesResponse describeResponse = 
getAsyncClient().describeRepositories(describeRequest).join();
                    return 
describeResponse.repositories().get(0).repositoryArn();
                } else {
                    throw new RuntimeException(ex);
                }
            } else {
                throw new RuntimeException(e);
            }
        }
    }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考CreateRepository中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Creates an Amazon Elastic Container Registry (Amazon ECR) repository.
     *
     * @param repoName the name of the repository to create.
     * @return the Amazon Resource Name (ARN) of the created repository, or an
 empty string if the operation failed.
     * @throws RepositoryAlreadyExistsException if the repository exists.
     * @throws EcrException         if an error occurs while creating the
 repository.
     */
    suspend fun createECRRepository(repoName: String?): String? {
        val request =
            CreateRepositoryRequest {
                repositoryName = repoName
            }

        return try {
            EcrClient { region = "us-east-1" }.use { ecrClient ->
                val response = ecrClient.createRepository(request)
                response.repository?.repositoryArn
            }
        } catch (e: RepositoryAlreadyExistsException) {
            println("Repository already exists: $repoName")
            repoName?.let { getRepoARN(it) }
        } catch (e: EcrException) {
            println("An error occurred: ${e.message}")
            null
        }
    }
```

- 有关 API 的详细信息，请参阅适用[CreateRepository](CreateRepository)于 Kotlin 的Amazon SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def create_repository(self, repository_name: str) -> dict[str, any]:
        """
        Creates an ECR repository.

        :param repository_name: The name of the repository to create.
        :return: A dictionary of the created repository.
        """
        try:
            response =
 self.ecr_client.create_repository(repositoryName=repository_name)
            return response["repository"]
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "RepositoryAlreadyExistsException":
```

```
                print(f"Repository {repository_name} already exists.")
                response = self.ecr_client.describe_repositories(
                    repositoryNames=[repository_name]
                )
                return self.describe_repositories([repository_name])[0]
            else:
                logger.error(
                    "Error creating repository %s. Here's why %s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise
```

- 有关 API 的详细信息，请参阅适用CreateRepository于 Python 的Amazon SDK (Boto3) API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteRepository`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteRepository`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

CLI

Amazon CLI

删除存储库

以下 `delete-repository` 示例命令将强制删除账户默认注册表中指定的存储库。如果存储库包含映像，则需要 `--force` 标志。

```
aws ecr delete-repository \
    --repository-name ubuntu \
```

```
    --force
```

输出：

```
{
    "repository": {
        "registryId": "123456789012",
        "repositoryName": "ubuntu",
        "repositoryArn": "arn:aws:ecr:us-west-2:123456789012:repository/ubuntu"
    }
}
```

有关更多信息，请参阅《Amazon ECR 用户指南》中的删除存储库。

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考DeleteRepository中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Deletes an ECR (Elastic Container Registry) repository.
     *
     * @param repoName the name of the repository to delete.
     * @throws IllegalArgumentException if the repository name is null or empty.
     * @throws EcrException if there is an error deleting the repository.
     * @throws RuntimeException if an unexpected error occurs during the deletion
 process.
     */
    public void deleteECRRepository(String repoName) {
        if (repoName == null || repoName.isEmpty()) {
            throw new IllegalArgumentException("Repository name cannot be null or
 empty");
        }
```

```
        DeleteRepositoryRequest repositoryRequest =
DeleteRepositoryRequest.builder()
            .force(true)
            .repositoryName(repoName)
            .build();

        CompletableFuture<DeleteRepositoryResponse> response =
getAsyncClient().deleteRepository(repositoryRequest);
        response.whenComplete((deleteRepositoryResponse, ex) -> {
            if (deleteRepositoryResponse != null) {
                System.out.println("You have successfully deleted the " +
repoName + " repository");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof EcrException) {
                    throw (EcrException) cause;
                } else {
                    throw new RuntimeException("Unexpected error: " +
cause.getMessage(), cause);
                }
            }
        });

        // Wait for the CompletableFuture to complete
        response.join();
    }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考DeleteRepository中的。

Kotlin

适用于 Kotlin 的 SDK

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
```

```
     * Deletes an ECR (Elastic Container Registry) repository.
     *
     * @param repoName the name of the repository to delete.
     */
    suspend fun deleteECRRepository(repoName: String) {
        if (repoName.isNullOrEmpty()) {
            throw IllegalArgumentException("Repository name cannot be null or
 empty")
        }

        val repositoryRequest =
            DeleteRepositoryRequest {
                force = true
                repositoryName = repoName
            }

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            ecrClient.deleteRepository(repositoryRequest)
            println("You have successfully deleted the $repoName repository")
        }
    }
```

- 有关 API 的详细信息，请参阅适用DeleteRepository于 K otlin 的Amazon SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
```

```
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon 
ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def delete_repository(self, repository_name: str):
        """
        Deletes an ECR repository.

        :param repository_name: The name of the repository to delete.
        """
        try:
            self.ecr_client.delete_repository(
                repositoryName=repository_name, force=True
            )
            print(f"Deleted repository {repository_name}.")
        except ClientError as err:
            logger.error(
                "Couldn't delete repository %s.. Here's why %s",
                repository_name,
                err.response["Error"]["Message"],
            )
            raise
```

- 有关 API 的详细信息，请参阅适用DeleteRepository于 Python 的Amazon SDK (Boto3) API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DescribeImages`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DescribeImages`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

CLI

Amazon CLI

要描述存储库中的映像

以下 `describe-images` 示例显示具有标签 `v1.13.6` 的 `cluster-autoscaler` 存储库中某个映像的相关详细信息。

```
aws ecr describe-images \
    --repository-name cluster-autoscaler \
    --image-ids imageTag=v1.13.6
```

输出：

```
{
    "imageDetails": [
        {
            "registryId": "012345678910",
            "repositoryName": "cluster-autoscaler",
            "imageDigest":
 "sha256:4a1c6567c38904384ebc64e35b7eeddd8451110c299e3368d2210066487d97e5",
            "imageTags": [
                "v1.13.6"
            ],
            "imageSizeInBytes": 48318255,
            "imagePushedAt": 1565128275.0
        }
    ]
}
```

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考DescribeImages中的。

Java

适用于 Java 的 SDK 2.x

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
 (Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     * @throws EcrException            if there is an error retrieving the image
 information from Amazon ECR.
     * @throws CompletionException      if the asynchronous operation completes
 exceptionally.
     */
    public void verifyImage(String repositoryName, String imageTag) {
        DescribeImagesRequest request = DescribeImagesRequest.builder()
            .repositoryName(repositoryName)
            .imageIds(ImageIdentifier.builder().imageTag(imageTag).build())
            .build();

        CompletableFuture<DescribeImagesResponse> response =
 getAsyncClient().describeImages(request);
        response.whenComplete((describeImagesResponse, ex) -> {
            if (ex != null) {
                if (ex instanceof CompletionException) {
                    Throwable cause = ex.getCause();
                    if (cause instanceof EcrException) {
                        throw (EcrException) cause;
                    } else {
                        throw new RuntimeException("Unexpected error: " +
 cause.getMessage(), cause);
                    }
                } else {
                    throw new RuntimeException("Unexpected error: " +
 ex.getCause());
```

```
                }
            } else if (describeImagesResponse != null && !
describeImagesResponse.imageDetails().isEmpty()) {
                System.out.println("Image is present in the repository.");
            } else {
                System.out.println("Image is not present in the repository.");
            }
        });

        // Wait for the CompletableFuture to complete.
        response.join();
    }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考DescribeImages中的。

Kotlin

适用于 Kotlin 的 SDK

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
 (Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     */
    suspend fun verifyImage(
        repoName: String?,
        imageTagVal: String?,
    ) {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
 cannot be null or empty" }
        require(!(imageTagVal == null || imageTagVal.isEmpty())) { "Image tag
 cannot be null or empty" }
```

```
        val imageId =
            ImageIdentifier {
                imageTag = imageTagVal
            }
        val request =
            DescribeImagesRequest {
                repositoryName = repoName
                imageIds = listOf(imageId)
            }

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val describeImagesResponse = ecrClient.describeImages(request)
            if (describeImagesResponse != null && !
describeImagesResponse.imageDetails?.isEmpty()!!) {
                println("Image is present in the repository.")
            } else {
                println("Image is not present in the repository.")
            }
        }
    }
```

- 有关 API 的详细信息，请参阅适用DescribeImages于 K otlin 的Amazon SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
```

```
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon 
ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def describe_images(
        self, repository_name: str, image_ids: list[str] = None
    ) -> list[dict]:
        """
        Describes ECR images.

        :param repository_name: The name of the repository to describe images 
for.
        :param image_ids: The optional IDs of images to describe.
        :return: The list of image descriptions.
        """
        try:
            params = {
                "repositoryName": repository_name,
            }
            if image_ids is not None:
                params["imageIds"] = [{"imageTag": tag} for tag in image_ids]

            paginator = self.ecr_client.get_paginator("describe_images")
            image_descriptions = []
            for page in paginator.paginate(**params):
                image_descriptions.extend(page["imageDetails"])
            return image_descriptions
        except ClientError as err:
            logger.error(
                "Couldn't describe images. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

- 有关 API 的详细信息，请参阅适用DescribeImages于 Python 的Amazon SDK (Boto3) API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DescribeRepositories`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DescribeRepositories`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

CLI

Amazon CLI

描述注册表中的存储库

此示例将描述账户默认注册表中的存储库。

命令:

```
aws ecr describe-repositories
```

输出：

```
{
    "repositories": [
        {
            "registryId": "012345678910",
            "repositoryName": "ubuntu",
            "repositoryArn": "arn:aws:ecr:us-west-2:012345678910:repository/
ubuntu"
        },
        {
            "registryId": "012345678910",
            "repositoryName": "test",
            "repositoryArn": "arn:aws:ecr:us-west-2:012345678910:repository/test"
        }
    ]
}
```

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考DescribeRepositories中的。

Java

适用于 Java 的 SDK 2.x

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Retrieves the repository URI for the specified repository name.
     *
     * @param repoName the name of the repository to retrieve the URI for.
     * @return the repository URI for the specified repository name.
     * @throws EcrException        if there is an error retrieving the repository
 information.
     * @throws CompletionException if the asynchronous operation completes
 exceptionally.
     */
    public void getRepositoryURI(String repoName) {
        DescribeRepositoriesRequest request =
 DescribeRepositoriesRequest.builder()
            .repositoryNames(repoName)
            .build();

        CompletableFuture<DescribeRepositoriesResponse> response =
 getAsyncClient().describeRepositories(request);
        response.whenComplete((describeRepositoriesResponse, ex) -> {
            if (ex != null) {
                Throwable cause = ex.getCause();
                if (cause instanceof InterruptedException) {
                    Thread.currentThread().interrupt();
                    String errorMessage = "Thread interrupted while waiting for
 asynchronous operation: " + cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                } else if (cause instanceof EcrException) {
                    throw (EcrException) cause;
                } else {
```

```
                    String errorMessage = "Unexpected error: " +
cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                }
            } else {
                if (describeRepositoriesResponse != null) {
                    if (!describeRepositoriesResponse.repositories().isEmpty()) {
                        String repositoryUri =
describeRepositoriesResponse.repositories().get(0).repositoryUri();
                        System.out.println("Repository URI found: " +
repositoryUri);
                    } else {
                        System.out.println("No repositories found for the given
name.");
                    }
                } else {
                    System.err.println("No response received from
describeRepositories.");
                }
            }
        });
        response.join();
    }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考DescribeRepositories中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Retrieves the repository URI for the specified repository name.
```

```
     *
     * @param repoName the name of the repository to retrieve the URI for.
     * @return the repository URI for the specified repository name.
     */
    suspend fun getRepositoryURI(repoName: String?): String? {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
 cannot be null or empty" }
        val request =
            DescribeRepositoriesRequest {
                repositoryNames = listOf(repoName)
            }

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val describeRepositoriesResponse =
 ecrClient.describeRepositories(request)
            if (!describeRepositoriesResponse.repositories?.isEmpty()!!) {
                return
 describeRepositoriesResponse?.repositories?.get(0)?.repositoryUri
            } else {
                println("No repositories found for the given name.")
                return ""
            }
        }
    }
```

- 有关 API 的详细信息，请参阅适用DescribeRepositories于 K otlin 的Amazon SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
```

```
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def describe_repositories(self, repository_names: list[str]) -> list[dict]:
        """
        Describes ECR repositories.

        :param repository_names: The names of the repositories to describe.
        :return: The list of repository descriptions.
        """
        try:
            response = self.ecr_client.describe_repositories(
                repositoryNames=repository_names
            )
            return response["repositories"]
        except ClientError as err:
            logger.error(
                "Couldn't describe repositories. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

- 有关 API 的详细信息，请参阅适用DescribeRepositories于 Python 的Amazon SDK (Boto3) API 参考。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
async fn show_repos(client: &aws_sdk_ecr::Client) -> Result<(),
 aws_sdk_ecr::Error> {
    let rsp = client.describe_repositories().send().await?;

    let repos = rsp.repositories();

    println!("Found {} repositories:", repos.len());

    for repo in repos {
        println!("  ARN:  {}", repo.repository_arn().unwrap());
        println!("  Name: {}", repo.repository_name().unwrap());
    }

    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeRepositories于 Rust 的Amazon SDK API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetAuthorizationToken`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetAuthorizationToken`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

CLI

Amazon CLI

要获取默认注册表的授权令牌

以下 `get-authorization-token` 示例命令可获取默认注册表的授权令牌。

```
aws ecr get-authorization-token
```

输出：

```
{
    "authorizationData": [
        {
            "authorizationToken": "QVdTOkN...",
            "expiresAt": 1448875853.241,
            "proxyEndpoint": "https://123456789012.dkr.ecr.us-
west-2.amazonaws.com"
        }
    ]
}
```

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考GetAuthorizationToken中的。

Java

适用于 Java 的 SDK 2.x

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Retrieves the authorization token for Amazon Elastic Container Registry
 (ECR).
     * This method makes an asynchronous call to the ECR client to retrieve the
 authorization token.
```

```
     * If the operation is successful, the method prints the token to the
 console.
     * If an exception occurs, the method handles the exception and prints the
 error message.
     *
     * @throws EcrException     if there is an error retrieving the authorization
 token from ECR.
     * @throws RuntimeException if there is an unexpected error during the
 operation.
     */
    public void getAuthToken() {
        CompletableFuture<GetAuthorizationTokenResponse> response =
 getAsyncClient().getAuthorizationToken();
        response.whenComplete((authorizationTokenResponse, ex) -> {
            if (authorizationTokenResponse != null) {
                AuthorizationData authorizationData =
 authorizationTokenResponse.authorizationData().get(0);
                String token = authorizationData.authorizationToken();
                if (!token.isEmpty()) {
                    System.out.println("The token was successfully retrieved.");
                }
            } else {
                if (ex.getCause() instanceof EcrException) {
                    throw (EcrException) ex.getCause();
                } else {
                    String errorMessage = "Unexpected error occurred: " +
 ex.getMessage();
                    throw new RuntimeException(errorMessage, ex); // Rethrow the
 exception
                }
            }
        });
        response.join();
    }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考GetAuthorizationToken中的。

Kotlin

适用于 Kotlin 的 SDK

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/**
 * Retrieves the authorization token for Amazon Elastic Container Registry
(ECR).
 *
 */
suspend fun getAuthToken() {
    EcrClient { region = "us-east-1" }.use { ecrClient ->
        // Retrieve the authorization token for ECR.
        val response = ecrClient.getAuthorizationToken()
        val authorizationData = response.authorizationData?.get(0)
        val token = authorizationData?.authorizationToken
        if (token != null) {
            println("The token was successfully retrieved.")
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用GetAuthorizationToken于 K otlin 的Amazon SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def get_authorization_token(self) -> str:
        """
        Gets an authorization token for an ECR repository.

        :return: The authorization token.
        """
        try:
            response = self.ecr_client.get_authorization_token()
            return response["authorizationData"][0]["authorizationToken"]
        except ClientError as err:
            logger.error(
                "Couldn't get authorization token. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

- 有关 API 的详细信息，请参阅适用GetAuthorizationToken于 Python 的Amazon SDK (Boto3) API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetRepositoryPolicy`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetRepositoryPolicy`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

CLI

Amazon CLI

要检索用于存储库的存储库策略

以下 `get-repository-policy` 示例显示 `cluster-autoscaler` 存储库的存储库策略的相关详细信息。

```
aws ecr get-repository-policy \
    --repository-name cluster-autoscaler
```

输出：

```
{
    "registryId": "012345678910",
    "repositoryName": "cluster-autoscaler",
    "policyText": "{\n  \"Version\" : \"2008-10-17\",\n  \"Statement\" :
 [ {\n    \"Sid\" : \"allow public pull\",\n    \"Effect\" : \"Allow\",\n
 \"Principal\" : \"*\",\n    \"Action\" : [ \"ecr:BatchCheckLayerAvailability\",
 \"ecr:BatchGetImage\", \"ecr:GetDownloadUrlForLayer\" ]\n  } ]\n}"
}
```

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考GetRepositoryPolicy中的。

Java

适用于 Java 的 SDK 2.x

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Gets the repository policy for the specified repository.
     *
     * @param repoName the name of the repository.
     * @throws EcrException if an AWS error occurs while getting the repository
 policy.
     */
    public String getRepoPolicy(String repoName) {
        if (repoName == null || repoName.isEmpty()) {
            throw new IllegalArgumentException("Repository name cannot be null or
 empty");
        }

        GetRepositoryPolicyRequest getRepositoryPolicyRequest =
 GetRepositoryPolicyRequest.builder()
            .repositoryName(repoName)
            .build();

        CompletableFuture<GetRepositoryPolicyResponse> response =
 getAsyncClient().getRepositoryPolicy(getRepositoryPolicyRequest);
        response.whenComplete((resp, ex) -> {
            if (resp != null) {
                System.out.println("Repository policy retrieved successfully.");
            } else {
                if (ex.getCause() instanceof EcrException) {
                    throw (EcrException) ex.getCause();
                } else {
                    String errorMessage = "Unexpected error occurred: " +
 ex.getMessage();
                    throw new RuntimeException(errorMessage, ex);
                }
            }
```

```
        });

        GetRepositoryPolicyResponse result = response.join();
        return result != null ? result.policyText() : null;
    }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考GetRepositoryPolicy中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Gets the repository policy for the specified repository.
     *
     * @param repoName the name of the repository.
     */
    suspend fun getRepoPolicy(repoName: String?): String? {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name 
cannot be null or empty" }

        // Create the request
        val getRepositoryPolicyRequest =
            GetRepositoryPolicyRequest {
                repositoryName = repoName
            }
        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val response = 
ecrClient.getRepositoryPolicy(getRepositoryPolicyRequest)
            val responseText = response.policyText
            return responseText
        }
```

```
    }
```

- 有关 API 的详细信息，请参阅适用GetRepositoryPolicy于 K otlin 的Amazon SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

Note

还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def get_repository_policy(self, repository_name: str) -> str:
        """
        Gets the policy for an ECR repository.

        :param repository_name: The name of the repository to get the policy for.
        :return: The policy text.
        """
        try:
            response = self.ecr_client.get_repository_policy(
                repositoryName=repository_name
            )
```

```
            return response["policyText"]
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "RepositoryPolicyNotFoundException":
                logger.error("Repository does not exist. %s.", repository_name)
                raise
            else:
                logger.error(
                    "Couldn't get repository policy for repository %s. Here's why
 %s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise
```

- 有关 API 的详细信息，请参阅适用GetRepositoryPolicy于 Python 的Amazon SDK (Boto3) API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListImages`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListImages`。

CLI

Amazon CLI

列出存储库的映像

以下 `list-images` 示例将显示 `cluster-autoscaler` 存储库的映像列表。

```
aws ecr list-images \
    --repository-name cluster-autoscaler
```

输出：

```
{
    "imageIds": [
```

```
        {
            "imageDigest": 
 "sha256:99c6fb4377e9a420a1eb3b410a951c9f464eff3b7dbc76c65e434e39b94b6570",
            "imageTag": "v1.13.8"
        },
        {
            "imageDigest": 
 "sha256:99c6fb4377e9a420a1eb3b410a951c9f464eff3b7dbc76c65e434e39b94b6570",
            "imageTag": "v1.13.7"
        },
        {
            "imageDigest": 
 "sha256:4a1c6567c38904384ebc64e35b7eeddd8451110c299e3368d2210066487d97e5",
            "imageTag": "v1.13.6"
        }
    ]
}
```

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考ListImages中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
async fn show_images(
    client: &aws_sdk_ecr::Client,
    repository: &str,
) -> Result<(), aws_sdk_ecr::Error> {
    let rsp = client
        .list_images()
        .repository_name(repository)
        .send()
        .await?;

    let images = rsp.image_ids();
```

```
    println!("found {} images", images.len());

    for image in images {
        println!(
            "image: {}:{}",
            image.image_tag().unwrap(),
            image.image_digest().unwrap()
        );
    }

    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用[ListImages](#)于 Rust 的Amazon SDK API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅[将 Amazon ECR 与软件开发工具包一起使用 Amazon](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

与 Amazon SDK **`PushImageCmd`** 配合使用

以下代码示例演示如何使用 `PushImageCmd`。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 [Amazon 代码示例存储库](#)中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Pushes a Docker image to an Amazon Elastic Container Registry (ECR)
 repository.
     *
     * @param repoName  the name of the ECR repository to push the image to.
     * @param imageName the name of the Docker image.
     */
    public void pushDockerImage(String repoName, String imageName) {
```

```
        System.out.println("Pushing " + imageName + " to Amazon ECR will take a
 few seconds.");
        CompletableFuture<AuthConfig> authResponseFuture =
 getAsyncClient().getAuthorizationToken()
            .thenApply(response -> {
                String token =
 response.authorizationData().get(0).authorizationToken();
                String decodedToken = new
 String(Base64.getDecoder().decode(token));
                String password = decodedToken.substring(4);

                DescribeRepositoriesResponse descrRepoResponse =
 getAsyncClient().describeRepositories(b -> b.repositoryNames(repoName)).join();
                Repository repoData =
 descrRepoResponse.repositories().stream().filter(r ->
 r.repositoryName().equals(repoName)).findFirst().orElse(null);
                assert repoData != null;
                String registryURL = repoData.repositoryUri().split("/")[0];

                AuthConfig authConfig = new AuthConfig()
                    .withUsername("AWS")
                    .withPassword(password)
                    .withRegistryAddress(registryURL);
                return authConfig;
            })
            .thenCompose(authConfig -> {
                DescribeRepositoriesResponse descrRepoResponse =
 getAsyncClient().describeRepositories(b -> b.repositoryNames(repoName)).join();
                Repository repoData =
 descrRepoResponse.repositories().stream().filter(r ->
 r.repositoryName().equals(repoName)).findFirst().orElse(null);
                getDockerClient().tagImageCmd(imageName + ":latest",
 repoData.repositoryUri() + ":latest", imageName).exec();
                try {

 getDockerClient().pushImageCmd(repoData.repositoryUri()).withTag("echo-
text").withAuthConfig(authConfig).start().awaitCompletion();
                    System.out.println("The " + imageName + " was pushed to
 ECR");

                } catch (InterruptedException e) {
                    throw (RuntimeException) e.getCause();
                }
                return CompletableFuture.completedFuture(authConfig);
```

```
            });

        authResponseFuture.join();
    }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考PushImageCmd中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Pushes a Docker image to an Amazon Elastic Container Registry (ECR) 
 repository.
     *
     * @param repoName the name of the ECR repository to push the image to.
     * @param imageName the name of the Docker image.
     */
    suspend fun pushDockerImage(
        repoName: String,
        imageName: String,
    ) {
        println("Pushing $imageName to $repoName will take a few seconds")
        val authConfig = getAuthConfig(repoName)

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val desRequest =
                DescribeRepositoriesRequest {
                    repositoryNames = listOf(repoName)
                }

            val describeRepoResponse = ecrClient.describeRepositories(desRequest)
            val repoData =
```

```
                describeRepoResponse.repositories?.firstOrNull
{ it.repositoryName == repoName }
                    ?: throw RuntimeException("Repository not found: $repoName")

            val tagImageCmd = getDockerClient()?.tagImageCmd("$imageName",
"${repoData.repositoryUri}", imageName)
            if (tagImageCmd != null) {
                tagImageCmd.exec()
            }
            val pushImageCmd =
                repoData.repositoryUri?.let {
                    dockerClient?.pushImageCmd(it)
                        // ?.withTag("latest")
                        ?.withAuthConfig(authConfig)
                }

            try {
                if (pushImageCmd != null) {
                    pushImageCmd.start().awaitCompletion()
                }
                println("The $imageName was pushed to Amazon ECR")
            } catch (e: IOException) {
                throw RuntimeException(e)
            }
        }
    }
```

- 有关 API 的详细信息，请参阅适用PushImageCmd于 K otlin 的Amazon SDK API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`PutLifeCyclePolicy`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `PutLifeCyclePolicy`。

CLI

Amazon CLI

创建生命周期策略

以下 `put-lifecycle-policy` 示例为账户的默认注册表中的指定存储库创建生命周期策略。

```
aws ecr put-lifecycle-policy \
    --repository-name "project-a/amazon-ecs-sample" \
    --lifecycle-policy-text "file://policy.json"
```

`policy.json` 的内容：

```
{
   "rules": [
       {
           "rulePriority": 1,
           "description": "Expire images older than 14 days",
           "selection": {
               "tagStatus": "untagged",
               "countType": "sinceImagePushed",
               "countUnit": "days",
               "countNumber": 14
           },
           "action": {
               "type": "expire"
           }
       }
   ]
}
```

输出：

```
{
   "registryId": "<aws_account_id>",
   "repositoryName": "project-a/amazon-ecs-sample",
   "lifecyclePolicyText": "{\"rules\":[{\"rulePriority\":1,\"description\":
\"Expire images older than 14 days\",\"selection\":{\"tagStatus\":\"untagged\",
\"countType\":\"sinceImagePushed\",\"countUnit\":\"days\",\"countNumber\":14},
\"action\":{\"type\":\"expire\"}}]}"
}
```

有关更多信息，请参阅《Amazon ECR 用户指南》中的生命周期策略。

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考PutLifeCyclePolicy中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def put_lifecycle_policy(self, repository_name: str, lifecycle_policy_text:
 str):
        """
        Puts a lifecycle policy for an ECR repository.

        :param repository_name: The name of the repository to put the lifecycle
 policy for.
        :param lifecycle_policy_text: The lifecycle policy text to put.
        """
        try:
            self.ecr_client.put_lifecycle_policy(
                repositoryName=repository_name,
                lifecyclePolicyText=lifecycle_policy_text,
            )
            print(f"Put lifecycle policy for repository {repository_name}.")
        except ClientError as err:
```

```
            logger.error(
                "Couldn't put lifecycle policy for repository %s. Here's why %s",
                repository_name,
                err.response["Error"]["Message"],
            )
            raise
```

设置到期日政策的示例。

```
    def put_expiration_policy(self):
        """
        Puts an expiration policy on the ECR repository.
        """
        policy_json = {
            "rules": [
                {
                    "rulePriority": 1,
                    "description": "Expire images older than 14 days",
                    "selection": {
                        "tagStatus": "any",
                        "countType": "sinceImagePushed",
                        "countUnit": "days",
                        "countNumber": 14,
                    },
                    "action": {"type": "expire"},
                }
            ]
        }

        self.ecr_wrapper.put_lifecycle_policy(
            self.repository_name, json.dumps(policy_json)
        )
```

- 有关 API 的详细信息，请参阅适用PutLifeCyclePolicy于 Python 的Amazon SDK (Boto3) API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`SetRepositoryPolicy`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `SetRepositoryPolicy`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

CLI

Amazon CLI

要为存储库设置存储库策略

以下 `set-repository-policy` 示例将文件中包含的存储库策略附加到 `cluster-autoscaler` 存储库。

```
aws ecr set-repository-policy \
    --repository-name cluster-autoscaler \
    --policy-text file://my-policy.json
```

`my-policy.json` 的内容：

```
{
    "Version" : "2008-10-17",
    "Statement" : [
        {
            "Sid" : "allow public pull",
            "Effect" : "Allow",
            "Principal" : "*",
            "Action" : [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ]
        }
    ]
}
```

输出：

```
{
    "registryId": "012345678910",
    "repositoryName": "cluster-autoscaler",
    "policyText": "{\n  \"Version\" : \"2008-10-17\",\n  \"Statement\" :
 [ {\n    \"Sid\" : \"allow public pull\",\n    \"Effect\" : \"Allow\",\n
 \"Principal\" : \"*\",\n    \"Action\" : [ \"ecr:BatchCheckLayerAvailability\",
 \"ecr:BatchGetImage\", \"ecr:GetDownloadUrlForLayer\" ]\n  } ]\n}"
}
```

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考SetRepositoryPolicy中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Sets the repository policy for the specified ECR repository.
     *
     * @param repoName the name of the ECR repository.
     * @param iamRole  the IAM role to be granted access to the repository.
     * @throws RepositoryPolicyNotFoundException if the repository policy does
 not exist.
     * @throws EcrException                      if there is an unexpected error
 setting the repository policy.
     */
    public void setRepoPolicy(String repoName, String iamRole) {
        /*
          This example policy document grants the specified AWS principal the
 permission to perform the
          `ecr:BatchGetImage` action. This policy is designed to allow the
 specified principal
          to retrieve Docker images from the ECR repository.
         */
        String policyDocumentTemplate = """
             {
```

```
              "Version" : "2012-10-17",
              "Statement" : [ {
                "Sid" : "new statement",
                "Effect" : "Allow",
                "Principal" : {
                  "AWS" : "%s"
                },
                "Action" : "ecr:BatchGetImage"
              } ]
            }
             """;

        String policyDocument = String.format(policyDocumentTemplate, iamRole);
        SetRepositoryPolicyRequest setRepositoryPolicyRequest = 
SetRepositoryPolicyRequest.builder()
            .repositoryName(repoName)
            .policyText(policyDocument)
            .build();

        CompletableFuture<SetRepositoryPolicyResponse> response = 
getAsyncClient().setRepositoryPolicy(setRepositoryPolicyRequest);
        response.whenComplete((resp, ex) -> {
            if (resp != null) {
                System.out.println("Repository policy set successfully.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof RepositoryPolicyNotFoundException) {
                    throw (RepositoryPolicyNotFoundException) cause;
                } else if (cause instanceof EcrException) {
                    throw (EcrException) cause;
                } else {
                    String errorMessage = "Unexpected error: " + 
cause.getMessage();
                    throw new RuntimeException(errorMessage, cause);
                }
            }
        });
        response.join();
   }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考SetRepositoryPolicy中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/**
 * Sets the repository policy for the specified ECR repository.
 *
 * @param repoName the name of the ECR repository.
 * @param iamRole the IAM role to be granted access to the repository.
 */
suspend fun setRepoPolicy(
    repoName: String?,
    iamRole: String?,
) {
    val policyDocumentTemplate =
        """
         {
          "Version" : "2012-10-17",
          "Statement" : [ {
            "Sid" : "new statement",
            "Effect" : "Allow",
            "Principal" : {
              "AWS" : "$iamRole"
            },
            "Action" : "ecr:BatchGetImage"
          } ]
        }

        """.trimIndent()
    val setRepositoryPolicyRequest =
        SetRepositoryPolicyRequest {
            repositoryName = repoName
            policyText = policyDocumentTemplate
        }
```

```
        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val response =
 ecrClient.setRepositoryPolicy(setRepositoryPolicyRequest)
            if (response != null) {
                println("Repository policy set successfully.")
            }
        }
    }
```

- 有关 API 的详细信息，请参阅适用SetRepositoryPolicy于 K otlin 的Amazon SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ECRWrapper:
    def __init__(self, ecr_client: client):
        self.ecr_client = ecr_client

    @classmethod
    def from_client(cls) -> "ECRWrapper":
        """
        Creates a ECRWrapper instance with a default Amazon ECR client.

        :return: An instance of ECRWrapper initialized with the default Amazon
 ECR client.
        """
        ecr_client = boto3.client("ecr")
        return cls(ecr_client)


    def set_repository_policy(self, repository_name: str, policy_text: str):
        """
        Sets the policy for an ECR repository.
```

```
        :param repository_name: The name of the repository to set the policy for.
        :param policy_text: The policy text to set.
        """
        try:
            self.ecr_client.set_repository_policy(
                repositoryName=repository_name, policyText=policy_text
            )
            print(f"Set repository policy for repository {repository_name}.")
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"RepositoryPolicyNotFoundException":
                logger.error("Repository does not exist. %s.", repository_name)
                raise
            else:
                logger.error(
                    "Couldn't set repository policy for repository %s. Here's why
%s",
                    repository_name,
                    err.response["Error"]["Message"],
                )
                raise
```

授予 IAM 角色下载权限的示例。

```
    def grant_role_download_access(self, role_arn: str):
        """
        Grants the specified role access to download images from the ECR
repository.

        :param role_arn: The ARN of the role to grant access to.
        """
        policy_json = {
            "Version": "2008-10-17",
            "Statement": [
                {
                    "Sid": "AllowDownload",
                    "Effect": "Allow",
                    "Principal": {"AWS": role_arn},
                    "Action": ["ecr:BatchGetImage"],
                }
            ],
```

```
        }

        self.ecr_wrapper.set_repository_policy(
            self.repository_name, json.dumps(policy_json)
        )
```

- 有关 API 的详细信息，请参阅适用SetRepositoryPolicy于 Python 的Amazon SDK (Boto3) API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`StartLifecyclePolicyPreview`与 Amazon SDK 或 CLI 配合使用

以下代码示例演示如何使用 `StartLifecyclePolicyPreview`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

CLI

Amazon CLI

要创建生命周期策略预览

以下 `start-lifecycle-policy-preview` 示例创建了由指定存储库的 JSON 文件定义的生命周期策略预览。

```
aws ecr start-lifecycle-policy-preview \
    --repository-name "project-a/amazon-ecs-sample" \
    --lifecycle-policy-text "file://policy.json"
```

`policy.json` 的内容：

```
{
   "rules": [
       {
```

```
           "rulePriority": 1,
           "description": "Expire images older than 14 days",
           "selection": {
               "tagStatus": "untagged",
               "countType": "sinceImagePushed",
               "countUnit": "days",
               "countNumber": 14
           },
           "action": {
               "type": "expire"
           }
       }
   ]
}
```

输出：

```
{
   "registryId": "012345678910",
   "repositoryName": "project-a/amazon-ecs-sample",
   "lifecyclePolicyText": "{\n    \"rules\": [\n        {\n
 \"rulePriority\": 1,\n            \"description\": \"Expire images older than 14
 days\",\n            \"selection\": {\n                \"tagStatus\": \"untagged
\",\n                \"countType\": \"sinceImagePushed\",\n
 \"countUnit\": \"days\",\n                \"countNumber\": 14\n            },\n
           \"action\": {\n                \"type\": \"expire\"\n            }\n
      }\n    ]\n}\n",
   "status": "IN_PROGRESS"
}
```

- 有关 API 的详细信息，请参阅Amazon CLI 命令参考StartLifecyclePolicyPreview中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
 (Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     * @throws EcrException             if there is an error retrieving the image
 information from Amazon ECR.
     * @throws CompletionException      if the asynchronous operation completes
 exceptionally.
     */
    public void verifyImage(String repositoryName, String imageTag) {
        DescribeImagesRequest request = DescribeImagesRequest.builder()
            .repositoryName(repositoryName)
            .imageIds(ImageIdentifier.builder().imageTag(imageTag).build())
            .build();

        CompletableFuture<DescribeImagesResponse> response =
 getAsyncClient().describeImages(request);
        response.whenComplete((describeImagesResponse, ex) -> {
            if (ex != null) {
                if (ex instanceof CompletionException) {
                    Throwable cause = ex.getCause();
                    if (cause instanceof EcrException) {
                        throw (EcrException) cause;
                    } else {
                        throw new RuntimeException("Unexpected error: " +
 cause.getMessage(), cause);
                    }
                } else {
                    throw new RuntimeException("Unexpected error: " +
 ex.getCause());
                }
            } else if (describeImagesResponse != null && !
describeImagesResponse.imageDetails().isEmpty()) {
                System.out.println("Image is present in the repository.");
            } else {
                System.out.println("Image is not present in the repository.");
            }
        });

        // Wait for the CompletableFuture to complete.
        response.join();
```

```
    }
```

- 有关 API 的详细信息，请参阅 Amazon SDK for Java 2.x API 参考StartLifecyclePolicyPreview中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 Amazon 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /**
     * Verifies the existence of an image in an Amazon Elastic Container Registry
 (Amazon ECR) repository asynchronously.
     *
     * @param repositoryName The name of the Amazon ECR repository.
     * @param imageTag       The tag of the image to verify.
     */
    suspend fun verifyImage(
        repoName: String?,
        imageTagVal: String?,
    ) {
        require(!(repoName == null || repoName.isEmpty())) { "Repository name
 cannot be null or empty" }
        require(!(imageTagVal == null || imageTagVal.isEmpty())) { "Image tag
 cannot be null or empty" }

        val imageId =
            ImageIdentifier {
                imageTag = imageTagVal
            }
        val request =
            DescribeImagesRequest {
                repositoryName = repoName
                imageIds = listOf(imageId)
```

```
            }

        EcrClient { region = "us-east-1" }.use { ecrClient ->
            val describeImagesResponse = ecrClient.describeImages(request)
            if (describeImagesResponse != null && !
describeImagesResponse.imageDetails?.isEmpty()!!) {
                println("Image is present in the repository.")
            } else {
                println("Image is not present in the repository.")
            }
        }
    }
```

- 有关 API 的详细信息，请参阅适用StartLifecyclePolicyPreview于 K otlin 的Amazon SDK API 参考。

有关 S Amazon DK 开发者指南和代码示例的完整列表，请参阅将 Amazon ECR 与软件开发工具包一起使用 Amazon。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

Amazon ECR 服务配额

下表提供了 Amazon Elastic Container Registry (Amazon ECR) 的默认服务配额。

名称	默认值	可调整	描述
每 24 小时进行一次基本图像扫描	每个受支持的区域：10 万个	否	使用基本扫描在当前账户和区域内 24 小时内可以扫描的最大图像数量。此限制包括推送扫描和手动扫描。
复制配置中每条规则的筛选器	每个受支持的区域：100 个	否	复制配置中每条规则的筛选器的最大数量。
每个存储库的镜像数	每个支持的区域：2 万个	是	每个存储库的最大镜像数。
层分段	每个受支持的区域：4,200 个	否	层分段数上限。仅当您使用 Amazon ECR API 操作直接启动镜像推送操作的分段上传时才适用。
生命周期策略长度	每个受支持的区域：30,720 个	否	生命周期策略中的最大字符数。
层分段大小上限	每个受支持的区域：10 个	否	层分段的最大大小 (MiB)。仅当您使用 Amazon ECR API 操作直接启动镜像推送操作的分段上传时才适用。
层大小上限	每个受支持的区域：52000 个	否	层的最大大小 (MiB)。

名称	默认值	可调整	描述
层分段大小下限	每个受支持的区域：5 个	否	层分段的最小大小 (MiB)。仅当您使用 Amazon ECR API 操作直接启动镜像推送操作的分段上传时才适用。
每个注册表的缓存提取规则	每个受支持的区域：50 个	否	缓存提取规则的最大数量。
BatchCheckLayerAvailability 请求率	每个受支持的区域：每秒 1000 个	是	您在当前区域内每秒可以发出的最大 BatchCheckLayerAvailability 请求数。将镜像推送到存储库时，会检查每个镜像层以验证之前是否已上传它。如果已上传，则会跳过镜像层。
BatchGetImage 请求率	每个受支持的区域：每秒 2,000 个	是	您在当前区域内每秒可以发出的最大 BatchGetImage 请求数。拉取图像时，系统会调用一次 BatchGetImage API 来检索图像清单。如果您请求增加此 API 的配额，请同时查看您的 GetDownloadUrlForLayer 使用情况。

名称	默认值	可调整	描述
CompleteLayerUpload 请求率	每个受支持的区域：每秒 100 个	是	您在当前区域内每秒可以发出的最大 CompleteLayerUpload 请求数。推送图像时，每个新的图像层都会调用一次 CompleteLayerUpload API，以验证上传是否已完成。
GetAuthorizationToken 请求率	每个受支持的区域：每秒 500 个	是	您在当前区域内每秒可以发出的最大 GetAuthorizationToken 请求数。
GetDownloadUrlForLayer 请求率	每个受支持的区域：每秒 3,000 个	是	您在当前区域内每秒可以发出的最大 GetDownloadUrlForLayer 请求数。拉取图像时，每个尚未缓存的图像层都会调用一次 GetDownloadUrlForLayer API。如果您请求增加此 API 的配额，请同时查看您的 BatchGetImage 使用情况。
InitiateLayerUpload 请求率	每个受支持的区域：每秒 100 个	是	您在当前区域内每秒可以发出的最大 InitiateLayerUpload 请求数。推送图像时，每个尚未上传的图像层都会调用一次 InitiateLayerUpload API。图像层是否已上传由 BatchCheckLayerAvailability API 操作决定。

名称	默认值	可调整	描述
PutImage 请求率	每个受支持的区域：每秒 10 个	是	您在当前区域内每秒可以发出的最大 PutImage 请求数。推送图像并上传所有新的图像层后，系统会调用一次 PutImage API 来创建或更新图像清单以及与该图像关联的标签。
UploadLayerPart 请求率	每个受支持的区域：每秒 500 个	是	您在当前区域内每秒可以发出的最大 UploadLayerPart 请求数。推送图像时，每个新的图像层都会分段上传，并且每个新的图像层部分都会调用一次 UploadLayerPart API。
镜像扫描速率	每个受支持的区域：1 个	否	每 24 小时每个镜像的最大镜像扫描次数。
已注册的存储库	每个受支持的区域：10 万个	是	您可以在当前区域中的此账户中创建的存储库的最大数量。
每个生命周期策略的规则数	每个受支持的区域：50 个	否	生命周期策略中的最大规则数量
每个复制配置的规则数	每个受支持的区域：10 个	否	复制配置中的最大规则数。
每个镜像的标签数	每个受支持的区域：1,000 个	否	每个镜像的最大标签数。
复制配置中所有规则的唯一目标	每个受支持的区域：25 个	否	复制配置中所有规则的最大唯一目标数

Amazon ECR 故障排除

本章帮助您查找 Amazon ECR 的诊断信息，并为常见问题和错误消息提供故障排查步骤。

主题

- 使用 Amazon ECR 时排查 Docker 命令和问题
- 排查 Amazon ECR 错误消息问题

使用 Amazon ECR 时排查 Docker 命令和问题

有时，针对 Amazon ECR 运行 Docker 命令可能导致错误消息。一些常见错误消息和可能的解决办法解释如下。

主题

- Docker 日志不包含预期的错误消息
- 从 Amazon ECR 存储库提取镜像时，出现错误：“Filesystem Verification Failed”(文件系统验证失败) 或“404: Image Not Found”(404：找不到镜像)
- 从 Amazon ECR 提取镜像时，出现错误：“Filesystem Layer Verification Failed”(文件系统分层验证失败)
- 推送到存储库时出现 HTTP 403 错误或“no basic auth credentials”(没有基础级验证凭证) 错误

Docker 日志不包含预期的错误消息

要开始调试任何 Docker 相关问题，首先在您的主机实例上运行的 Docker 守护程序中开启 Docker 调试输出。如果您使用从 Amazon ECS 容器实例上的 Amazon ECR 中提取的映像，则请参阅 Amazon Elastic Container Service Developer Guide 中的 Configuring verbose output from the Docker daemon。

从 Amazon ECR 存储库提取镜像时，出现错误：“Filesystem Verification Failed”(文件系统验证失败) 或“404: Image Not Found”(404：找不到镜像)

在 Docker 1.9 或更高版本中使用 docker pull 命令从 Amazon ECR 存储库提取镜像时，可能会收到错误 `Filesystem verification failed`。如果使用的是 1.9 之前的 Docker 版本，则可能收到错误 `404: Image not found`。

以下为一些可能的原因及它们的解释。

本地磁盘已满

> 如果运行 docker pull 命令的本地磁盘已满，那么对本地文件计算的 SHA-1 哈希值可能与 Amazon ECR 计算的 SHA-1 哈希值不同。确保本地磁盘有足够的剩余空间可存储所提取的 Docker 镜像。为腾出空间存储新镜像，可以删除旧镜像。使用 docker images 命令可查看所有已下载到本地的 Docker 镜像的列表及这些镜像的大小。

由于网络错误，客户端无法连接到远程存储库

> 调用 Amazon ECR 存储库需要 Internet 连接正常。验证网络设置，然后验证其他工具和应用程序是否可以访问 Internet 上的资源。如果您在私有子网中的 Amazon EC2 实例docker pull上运行，请验证该子网是否有通往互联网的路由。可使用网络地址转换 (NAT) 服务器或托管的 NAT 网关。
>
> 目前，对 Amazon ECR 存储库的调用还要求通过您的公司防火墙访问 Amazon Simple Storage Service (Amazon S3)。如果贵企业或组织使用的是允许服务终端节点的防火墙软件或 NAT 设备，请确保当前区域的 Amazon S3 服务终端节点在允许范围内。
>
> 如果您通过 HTTP 代理使用 Docker，可以对 Docker 进行相应的代理设置。有关更多信息，请参阅 Docker 文档中的 HTTP 代理。

从 Amazon ECR 提取镜像时，出现错误："Filesystem Layer Verification Failed"(文件系统分层验证失败)

您可能在使用 `image image-name not found` 命令提取镜像时收到错误 docker pull。如果检查 Docker 日志，可能会看到与下面类似的错误：

```
filesystem layer verification failed for digest sha256:2b96f...
```

此错误表示镜像的一个或多个层下载失败。以下为一些可能的原因及它们的解释。

您正在使用旧版本的 Docker

> 在使用低于 1.10 的 Docker 版本时，有少数情况会出现此错误。请将您的 Docker 客户端升级至 1.10 或更高版本。

您的客户端遇到网络错误或磁盘错误

如前文对 `Filesystem verification failed` 消息的讨论中所述，磁盘已满或网络问题可能会导致一个或多个层无法下载。请遵循上述建议确保您的文件系统未满，并且您在网络中有对 Amazon S3 的访问权限。

推送到存储库时出现 HTTP 403 错误或“no basic auth credentials”(没有基础级验证凭证) 错误

有时，即使您已使用 aws ecr get-login-password 命令成功通过 Docker 身份验证，也可能会从 docker push 或 docker pull 命令收到 `HTTP 403 (Forbidden)` 错误或者错误消息 `no basic auth credentials`。以下是此问题的一些已知的原因：

您已验证到其他区域

身份验证请求与特定的区域相关联，不能跨区域使用。例如，如果您从美国西部 (俄勒冈) 获得授权令牌，不能使用它对您在美国东部 (弗吉尼亚北部) 的存储库进行身份验证。要解决此问题，请确保您已从存储库所在的同一区域检索了身份验证令牌。有关更多信息，请参阅 the section called “注册表身份验证”。

您已进行身份验证以推送到您没有权限的存储库

您没有必要的权限来推送到存储库。有关更多信息，请参阅 Amazon ECR 中的私有存储库策略。

您的令牌已过期。

对于使用 `GetAuthorizationToken` 操作获取的令牌，默认授权令牌有效期为 12 小时。

`wincred` 凭证管理器中的错误

适用于 Windows 的 Docker 的某些版本使用名为的凭据管理器`wincred`，该管理器无法正确处理由生成的 Docker 登录命令aws ecr get-login-password（有关更多信息，请参阅私有存储库`CredsStore`失败）。可以运行作为输出的 Docker 登录命令，但如果尝试推送或提取镜像，这些命令会失败。修复这个错误的方法是，对于从 aws ecr get-login-password 输出的 Docker 登录命令，删除其注册表参数中的 `https://` 方案。如下所示为不带 HTTPS 方案的 Docker 登录命令示例。

```
docker login -u AWS -p <password> <aws_account_id>.dkr.ecr.<region>.amazonaws.com
```

排查 Amazon ECR 错误消息问题

有时，通过 Amazon ECR 控制台或 Amazon CLI 发起的 API 调用会存在错误消息。一些常见错误消息和可能的解决办法解释如下。

HTTP 429：请求太多或 ThrottleException

您可能会从一个或多个 Amazon ECR 操作或 API 调用收到 `429: Too Many Requests` 错误或 `ThrottleException` 错误。这表示由于您在短时间内重复调用 Amazon ECR 中的单个终端节点，您的请求已受限制。单个用户在一段时间内，调用单个终端节点的次数超过特定阈值时，就会产生限制。

Amazon ECR 中的每个 API 操作都有一个与之相关的速率限制。例如，`GetAuthorizationToken` 操作的限制为每秒 20 个事务 (TPS)，允许高达 200 TPS 的突增。在每个区域，每个账户会收到一个可存储多达 200 点 `GetAuthorizationToken` 积分的存储桶。这些积分以每秒 20 点的速度补充。如果您的存储桶有 200 点积分，则可实现每秒 200 个 `GetAuthorizationToken` API 事务 (持续一秒)，然后无限期地维持每秒 20 个事务。有关 Amazon ECR 速率限制的更多信息 APIs，请参阅Amazon ECR 服务配额。

要处理限制错误，请在代码中实施增量退避重试函数。有关更多信息，请参阅《工具参考指南》Amazon SDKs 和《工具参考指南》中的重试行为。另一种选择是请求提高速率限制，您可以使用服务配额控制台执行此操作。 。

HTTP 403：“User [arn] is not authorized to perform [operation]”(用户 [arn] 没有执行 [operation] 的权限)

尝试通过 Amazon ECR 执行操作时，您可能会收到以下错误：

```
$ aws ecr get-login-password
A client error (AccessDeniedException) occurred when calling the GetAuthorizationToken
 operation:
    User: arn:aws:iam::account-number:user/username is not authorized to perform:
    ecr:GetAuthorizationToken on resource: *
```

这表示您的用户没有获得使用 Amazon ECR 的权限，或者这些权限设置不正确。尤其是在对 Amazon ECR 执行操作时，请验证是否已授予用户访问该存储库的权限。有关创建和验证 Amazon ECR 权限的更多信息，请参阅 适用于 Amazon Elastic Container Registry 的 Identity and Access Management。

HTTP 404：“Repository Does Not Exist”(存储库不存在) 错误

如果您指定了当前不存在的 Docker Hub 存储库，Docker Hub 会自动创建存储库。但在使用 Amazon ECR 时，新存储库必须在使用前显式创建。这会防止意外创建新存储库 (例如，由于输入错误)，也可确保为所有新存储库明确分配适当的安全访问策略。有关创建存储库的更多信息，请参阅 Amazon ECR 私有存储库。

错误：无法从非 TTY 设备执行交互式登录

如果您收到错误 `Cannot perform an interactive login from a non TTY device`，则以下故障排除步骤应该会有所帮助。

- 确认您使用的是 Amazon CLI 版本 2，并且您的系统上没有 Amazon CLI 版本 1 的冲突版本。有关更多信息，请参阅安装或更新 Amazon CLI的最新版本。
- 确认您已 Amazon CLI 使用有效的凭据配置了您的证书。有关更多信息，请参阅安装或更新 Amazon CLI的最新版本。
- 验证您的 Amazon CLI 命令语法是否正确。

将 Podman 与 Amazon ECR 结合使用

将 Podman 与 Amazon ECR 结合使用使组织能够利用 Podman 的安全性和简单性，同时受益于用于容器映像管理的 Amazon ECR 的可扩展性和可靠性。通过遵循概述的步骤和命令，开发人员和管理员可以简化其容器工作流，增强安全性并优化资源利用率。随着容器化的发展势头不断增强，使用 Podman 和 Amazon ECR 为管理和部署容器化应用程序提供了强大而灵活的解决方案。

使用 Podman 通过 Amazon ECR 进行身份验证

在使用 Podman 与 Amazon ECR 进行交互之前，需要进行身份验证。可以通过以下方式来完成此操作：运行 `aws ecr get-login-password` 命令来检索身份验证令牌，然后将该令牌与 `podman login` 命令一起使用，以通过 Amazon ECR 进行身份验证。

```
aws ecr get-login-password --region region | podman login --username AWS --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

在 Podman 中使用亚马逊 ECR 凭证助手

Amazon ECR 提供了可与 Podman 配合使用的 Docker 凭证助手。在向 Amazon ECR 推送和拉取映像时，凭证助手可以更轻松地存储和使用 Docker 证书。有关安装和配置步骤，请参阅 Amazon ECR Docker 凭证辅助程序。

> **Note**
>
> Amazon ECR Docker 凭证助手目前不支持多因素身份验证 (MFA)。

使用 Podman 从 Amazon ECR 提取映像

成功进行身份验证后，可以使用带有完整 Amazon ECR 存储库 URI 的 `podman pull` 命令从 Amazon ECR 中提取容器映像。

```
podman pull aws_account_id.dkr.ecr.region.amazonaws.com/repository_name:tag
```

使用 Podman 运行 Amazon ECR 的容器

提取所需映像后，可以使用 `podman run` 命令实例化容器。

```
podman run -d aws_account_id.dkr.ecr.region.amazonaws.com/repository_name:tag
```

使用 Podman 将映像推送到 Amazon ECR

要将本地映像推送到 Amazon ECR，必须首先使用 `podman tag` 为映像标记 Amazon ECR 存储库 URI，然后才能使用 `podman push` 命令将映像上传到 Amazon ECR。

```
podman
 tag local_image:tag aws_account_id.dkr.ecr.region.amazonaws.com/repository_name:tag
podman push aws_account_id.dkr.ecr.region.amazonaws.com/repository_name:tag
```

文档历史记录

下表列出了自 Amazon ECR 上一次发布以来对文档所做的重要更改。我们还经常更新文档来处理发送给我们的反馈意见。

更改	描述	日期
IPv6 支持	增加了对使用 IPv4仅限和双堆栈（IPv4 和）终端节点向 Amazon ECR 注册表发出请求的支持。 IPv6有关更多信息，请参阅 向 Amazon ECR 注册管理机构提出请求。	2025 年 4 月 30 日
添加了 Amazon ECR 私有注册表支持以提取缓存	Amazon ECR 增加了对为 Amazon ECR 私有注册表创建拉取缓存规则的支持。有关更多信息，请参阅将上游注册表与 Amazon ECR 私有注册表同步和用于缓存提取的 Amazon ECR 服务相关角色。	2025 年 3 月 12 日
增加了对设置注册表策略范围的支持	Amazon ECR 增加了对为您的私有注册表配置注册策略范围的支持。有关更多信息，请参阅 Amazon ECR 中的私有注册权限和 Amazon ECR 私有注册表。	2024 年 12 月 23 日
亚马逊 EC2 Container RegistryPullOnly-新政策	Amazon ECR 添加了一项新政策，向亚马逊 ECR 授予仅限拉取的权限。	2024 年 10 月 10 日
事件中的 docker/OCI 客户端代理操作现在指向 CloudTrail `ecr.amazonaws.com`	与 Docker/OCI 客户端终端节点关联 CloudTrail 的事件的用户代理 (`userAgent sourceIPAddress`) 和源 IP 地址 () 字段`AWS Internal`中的值`ecr.amazonaws.com` 替换。有关示例，请参阅 示例：镜像提取操作 和 示例：镜像推送操作。	2024 年 7 月 1 日
添加了用于存储库创建模板的新 Amazon ECR 服务相关角色的描述。	Amazon ECR 使用名为的服务相关角色 AWSServiceRoleForECRTemplate，该角色授予 Amazon ECR 代表您执行操作以完成存储库创建模板操作的权限。有关更多信息，请参阅 用于存储库创建模板的 Amazon ECR 服务相关角色。	2024 年 6 月 20 日

更改	描述	日期
已添加 `ECRTemplateServiceRolePolicy` 服务相关角色。	已添加 `ECRTemplateServiceRolePolicy` 服务相关角色。有关更多信息，请参阅 [`ECRTemplateServiceRolePolicy`]。	2024 年 6 月 20 日
已增加跨区域和跨账户复制到中国区域。	Amazon ECR 添加了对中国区域筛选要复制的存储库的支持。有关更多信息，请参阅 Amazon ECR 中的私有映像复制。	2024 年 5 月 15 日
添加了 GitLab 容器注册表以查看缓存规则	Amazon ECR 增加了对为 GitLab 容器注册表创建拉取缓存规则的支持。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步。	2024 年 5 月 8 日
Amazon ECR 生命周期策略更新，增加了对使用通配符的支持	Amazon ECR 增加了对在生命周期策略中使用通配符的支持，方法是在生命周期策略规则中使用 `tagPatternList` 参数。有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像。	2023 年 12 月 18 日
Amazon ECR 存储库创建模板	Amazon ECR 增加了对存储库创建模板的支持 有关更多信息，请参阅 用于控制在缓存提取或复制操作期间创建的存储库的模板。	2023 年 11 月 15 日
Amazon ECR 缓存提取添加了针对经过身份验证的上游注册表的支持	Amazon ECR 增加了对使用需要对拉取缓存规则进行身份验证的上游注册表的支持。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步。	2023 年 11 月 15 日
AWSECRPullThroughCache_ServiceRolePolicy – 对现有策略的更新	Amazon ECR 将新权限添加到 `AWSECRPullThroughCache_ServiceRolePolicy` 策略。这些权限允许 Amazon ECR 检索 Secrets Manager 密钥的加密内容。当使用缓存提取规则来缓存需要身份验证的上游注册表中的映像时，需要此类权限。	2023 年 11 月 15 日
Amazon ECR 镜像签名	Amazon ECR，并 Amazon Signer 增加了对使用公证客户端创建和推送容器镜像签名的支持。有关更多信息，请参阅 对存储在 Amazon ECR 私有存储库中的映像进行签名。	2023 年 6 月 6 日

更改	描述	日期
增加了 Kubernetes 容器注册表以提取缓存规则	Amazon ECR 增加了对创建 Kubernetes 容器注册表的缓存提取规则的支持。有关更多信息，请参阅 将上游注册表与 Amazon ECR 私有注册表同步。	2023 年 6 月 1 日
Amazon ECR 增强扫描持续时间支持	Amazon Inspector 增加了对启用增强扫描时监控存储库的持续时间设置支持。有关更多信息，请参阅 更改 Amazon Inspector 中映像增强扫描的持续时间。	2022 年 6 月 28 日
Amazon ECR 向亚马逊发送存储库提取计数指标 CloudWatch	Amazon ECR 向亚马逊 CloudWatch发送存储库提取计数指标。有关更多信息，请参阅 Amazon ECR 存储库指标。	2022 年 1 月 6 日
扩展了复制支持	Amazon ECR 添加了要复制的存储库进行筛选的支持。有关更多信息，请参阅 Amazon ECR 中的私有映像复制。	2021 年 9 月 21 日
Amazon 亚马逊 ECR 的托管策略	Amazon ECR 添加了 Amazon 托管策略的文档。有关更多信息，请参阅 Amazon Amazon 弹性容器注册表的托管策略。	2021 年 6 月 24 日
跨区域和跨账户复制	Amazon ECR 添加了对配置私有注册表复制设置的支持。有关更多信息，请参阅 Amazon ECR 中的私有注册表设置。	2020 年 12 月 8 日
OCI 构件支持	Amazon ECR 添加了对推送和提取 Open Container Itistry (OCI) 构件的支持。新的参数 `artifactMediaType` 添加到 `DescribeImages` API 响应中以指示工件类型。 有关更多信息，请参阅 将 Helm 图表推送到 Amazon ECR 私有存储库。	2020 年 8 月 24 日
静态加密	Amazon ECR 增加了对结合使用服务器端加密和 Amazon Key Management Service (Amazon KMS) 中所存储客户托管密钥配置加密的支持。 有关更多信息，请参阅 静态加密。	2020 年 7 月 29 日

更改	描述	日期
多架构镜像	Amazon ECR 添加了对创建和推送用于多架构镜像的 Docker 清单列表的支持。 有关更多信息，请参阅 将多架构映像推送到 Amazon ECR 私有存储库。	2020 年 4 月 28 日
Amazon ECR 使用情况指标	Amazon ECR 增加了 CloudWatch 使用量指标，可让您了解账户的资源使用情况。您还可以从 Service Quotas 控制台 CloudWatch 和 Service Quotas 控制台创建 CloudWatch警报，以便在使用量接近已申请的服务配额时收到提醒。 有关更多信息，请参阅 Amazon ECR 用量指标。	2020 年 2 月 28 日
更新了 Amazon ECR 服务配额	更新了 Amazon ECR 服务配额，以包含每个 API 的配额。 有关更多信息，请参阅 Amazon ECR 服务配额。	2020 年 2 月 19 日
已添加 `get-login-password` 命令	增加了对 get-login-password 的支持，它提供了一个简单而安全的方法来检索授权令牌。 有关更多信息，请参阅 使用授权令牌。	2020 年 2 月 4 日
镜像扫描	增加了对镜像扫描的支持，这有助于识别容器镜像中的软件漏洞。Amazon ECR 使用开源 CoreOS Clair 项目中的常见漏洞和风险敞口 (CVEs) 数据库，并为您提供扫描结果列表。 有关更多信息，请参阅 在 Amazon ECR 中扫描映像是否存在软件漏洞。	2019 年 10 月 24 日
VPC 终端节点策略	增加了对在 Amazon ECR 接口 VPC 终端节点上设置 IAM policy 的支持。 有关更多信息，请参阅 为 Amazon ECR VPC 端点创建终端节点策略。	2019 年 9 月 26 日

更改	描述	日期
镜像标签可变性	增加了对将存储库配置为不可变的支持，以防止覆盖镜像标签。 有关更多信息，请参阅 防止映像标签在 Amazon ECR 中被覆盖。	2019 年 7 月 25 日
接口 VPC 端点 (Amazon PrivateLink)	增加了对配置由提供支持的接口 VPC 终端节点的支持 Amazon PrivateLink。这能让您在您的 VPC 和 Amazon ECR 之间创建私有连接，而无需通过 Internet、NAT 实例、VPN 连接或 Amazon Direct Connect进行访问。 有关更多信息，请参阅 Amazon ECR 接口 VPC 终端节点 ()Amazon PrivateLink。	2019 年 1 月 25 日
为资源加标签	Amazon ECR 增加了对为存储库添加元数据标签的支持。 有关更多信息，请参阅 在 Amazon ECR 中标记私有存储库。	2018 年 12 月 18 日
Amazon ECR 名称更改	亚马逊弹性容器注册表已重命名（之前为亚马逊 EC2 容器注册表）。	2017 年 11 月 21 日
生命周期策略	Amazon ECR 生命周期策略使您能够指定存储库中镜像的生命周期管理。 有关更多信息，请参阅 在 Amazon ECR 中使用生命周期策略自动清理映像。	2017 年 10 月 11 日
Amazon ECR 支持 Docker Image Manifest 2、Schema 2	Amazon ECR 现已支持 Docker Image Manifest V2 Schema 2 (与 Docker 版本 1.10 和更高版本配合使用) 有关更多信息，请参阅 Amazon ECR 中的容器映像清单格式支持。	2017 年 1 月 27 日

更改	描述	日期
Amazon ECR 正式发布	Amazon Elastic Container Registry (Amazon ECR) 是一项安全、可扩展和可靠的 Amazon 托管 Docker 注册服务。	2015 年 12 月 21 日