CodeArtifact 用户指南

CodeArtifact

CodeArtifact: CodeArtifact 用户指南

Table of Contents

什么是 AWS CodeArtifact？

AWS CodeArtifact 是一项安全、高度可扩展的托管构件存储库服务，有助于组织存储和共享用于应用程序开发的软件包。您可以将 CodeArtifact 与常用的构建工具和程序包管理器（例如 NuGet CLI、Maven、Gradle、npm、yarn、pip 和 twine）配合使用。CodeArtifact 有助于减少管理自己的构件存储系统的需求，也不需要担心其基础设施的扩展。您可以在 CodeArtifact 存储库中存储的程序包数量或总大小没有限制。

您可以在私有 CodeArtifact 存储库和外部公有存储库（例如 npmjs.com 或 Maven Central）之间创建连接。然后，当程序包管理器请求程序包时，CodeArtifact 将按需从公有存储库中提取和存储程序包。这样就可以更方便地使用应用程序所用的开源依赖项，并有助于确保这些依赖项始终可用于构建和开发。您也可以将私有程序包发布到 CodeArtifact 存储库。从而有助于在组织中的多个应用程序和开发团队之间分享专有软件组件。

有关更多信息，请参阅 AWS CodeArtifact。

CodeArtifact 如何工作？

CodeArtifact 将软件包存储在存储库中。存储库是多种语言的，单个存储库可以包含任何受支持类型的程序包。每个 CodeArtifact 存储库都是单个 CodeArtifact 域的成员。我们建议您为组织使用一个生产域，其中包含一个或多个存储库。例如，可为不同的开发团队使用各个不同的存储库。然后，可以发现存储库中的程序包，并在各个开发团队之间共享。

要将程序包添加到存储库，请将程序包管理器（例如 npm 或 Maven）配置为使用存储库端点 (URL)。然后，您可以使用程序包管理器将程序包发布到存储库。您还可以通过将存储库配置为与公有存储库（例如 npmjs、NuGet Gallery、Maven Central 或 PyPI）建立外部连接，从而将开源程序包导入存储库。有关更多信息，请参阅将 CodeArtifact 存储库连接到公有存储库。

您可以将一个存储库中的程序包提供给同一域中的另一个存储库。为此，请将一个存储库配置为另一个存储库的上游。上游存储库可用的所有程序包版本也可供下游存储库使用。此外，通过与公有存储库建立外部连接，上游存储库可用的所有程序包均可提供给下游存储库使用。有关更多信息，请参阅在 CodeArtifact 中使用上游存储库。

AWS CodeArtifact 概念

以下是您在使用 CodeArtifact 时需要了解的一些概念和术语。

主题

资产

资产 是在 CodeArtifact 中存储且与程序包版本相关联的单个文件，例如 npm `.tgz` 文件或 Maven POM 和 JAR 文件。

域

存储库聚合到一个名为域 的更高级别实体中。所有程序包资产和元数据都存储在域中，但通过存储库来使用资产和元数据。给定的程序包资产（例如 Maven JAR 文件）在每个域中存储一次，而不管域中有多少个存储库。域中的所有资产和元数据都使用存储在 AWS Key Management Service (AWS KMS) 中的相同 AWS KMS key（KMS 密钥）进行加密。

每个存储库都是单个域的成员，不能移动到另一个域。

借助域，您可以将组织策略应用于多个存储库。通过这种方法，您可以确定哪些账户可以访问域中的存储库，以及哪些公有存储库可以用作程序包的来源。

尽管一个组织可以有多个域，但我们建议使用一个包含所有已发布构件的生产域。这样，团队就可以找到和共享整个组织中的程序包。

存储库

CodeArtifact 存储库 包含一组程序包版本，每个版本都映射到一组资产。存储库是多种语言的，单个存储库可以包含任何受支持类型的程序包。每个存储库都公开端点，用于使用 nuget CLI、npm CLI、Maven CLI (mvn) 和 pip 等工具来提取和发布程序包。您最多可以为每个域创建 1000 个存储库。

程序包

程序包 是解析依赖关系和安装软件所需的软件和元数据的捆绑包。在 CodeArtifact 中，程序包由程序包名称、可选的命名空间（例如，`@types/node` 中的 `@types`）、一组程序包版本和程序包级元数据（例如 npm 标签）组成。

AWS CodeArtifact 支持 npm、PyPI、Maven、NuGet 和通用程序包格式。

程序包命名空间

某些程序包格式支持分层程序包名称，用于将程序包组织成逻辑组，有助于避免名称冲突。例如，npm 支持作用域。有关更多信息，请参阅 npm 作用域文档。npm 程序包 `@types/node` 的作用域为 `@types`，名称为 `node`。`@types` 作用域中还有许多其他的程序包名称。在 CodeArtifact 中，作用域（“types”）称为程序包命名空间，名称（“node”）称为程序包名称。对于 Maven 程序包，程序包命名空间与 Maven GroupId 相对应。Maven 程序包 `org.apache.logging.log4j:log4j` 的 groupID（程序包命名空间）为 `org.apache.logging.log4j`，artifactID（程序包名称）为 `log4j`。对于通用程序包，需要命名空间。某些程序包格式（例如 pyPI）不支持其概念类似于 npm 作用域或 Maven groupID 的分层名称。如果无法对程序包名称进行分组，则可能更难避免名称冲突。

程序包版本

程序包版本 标识程序包的特定版本，例如 `@types/node 12.6.9`。版本号格式和语义因不同的程序包格式而异。例如，npm 程序包版本必须符合语义版本控制规范。在 CodeArtifact 中，程序包版本由版本标识符、程序包版本级别元数据和一组资产组成。

程序包版本修订

程序包版本修订 是一个字符串，用于标识程序包版本的一组特定资产和元数据。每次更新程序包版本时，都会创建一个新的程序包版本修订。例如，您可以发布 Python 程序包版本的源分配存档 (sdist)，然后将包含已编译代码的 Python wheel 添加到同一个版本中。当您发布 wheel 时，会创建一个新的程序包版本修订。

上游存储库

当可以从下游存储库的存储库端点访问其中的程序包版本时，则一个存储库位于另一个存储库的上游。从客户的角度来看，这种方法有效地合并了两个存储库的内容。使用 CodeArtifact，您可以在两个存储库之间创建上游关系。

如何开始使用 CodeArtifact？

我们建议您完成以下步骤：

1. 通过阅读AWS CodeArtifact 概念来了解有关 CodeArtifact 的更多信息。
2. 按照设置 AWS CodeArtifact中的步骤设置您的 AWS 账户、AWS CLI 和 IAM 用户。
3. 按照开始使用 CodeArtifact中的说明使用 CodeArtifact。

设置 AWS CodeArtifact

如果您已注册了 Amazon Web Services (AWS)，则可以立即开始使用 AWS CodeArtifact。您可以打开 CodeArtifact 控制台，选择创建域和存储库，然后按照启动向导中的步骤创建您的第一个域和存储库。

如果您尚未注册 AWS，或者在创建第一个域和存储库时需要协助，请完成以下任务来进行设置，以便可以使用 CodeArtifact：

主题

注册 AWS

当您注册 Amazon Web Services（AWS）时，系统只对您使用的服务和资源（包括 AWS CodeArtifact）收费。

如果您已有 AWS 账户，请跳到下一个任务 安装或升级 AWS CLI，然后对其进行配置。如果您还没有 AWS 账户，请使用以下步骤创建。

创建 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册 AWS 账户时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

安装或升级 AWS CLI，然后对其进行配置

要从本地开发计算机上的 AWS Command Line Interface (AWS CLI) 调用 CodeArtifact 命令，您必须安装 AWS CLI。

如果您安装的是旧版 AWS CLI，则必须进行升级，以使 CodeArtifact 命令可用。CodeArtifact 命令可在以下 AWS CLI 版本中使用：

1. AWS CLI 1：1.18.77 和更新版本
2. AWS CLI 2：2.0.21 和更新版本

要检查版本，请使用 `aws --version` 命令。

安装和配置 AWS CLI

1. 按照安装 AWS Command Line Interface 中的说明安装或升级 AWS CLI。
2. 使用 configure 命令配置 AWS CLI，如下所示。

   ```
   aws configure
   ```

   出现提示时，指定要用于 CodeArtifact 的 IAM 用户的 AWS 访问密钥和 AWS 秘密访问密钥。当系统提示提供默认 AWS 区域 名称时，请指定您要创建管道的区域，例如 `us-east-2`。系统提示指定默认输出格式时，指定 `json`。

   > **Important**
   >
   > 在配置 AWS CLI 时，系统会提示您指定 AWS 区域。选择《AWS 一般参考》的区域和端点中列出的受支持区域之一。

   有关更多信息，请参阅配置 AWS Command Line Interface和管理 IAM 用户的访问密钥。
3. 要验证安装或升级，请从 AWS CLI 调用以下命令：

   ```
   aws codeartifact help
   ```

   如果成功，此命令将显示可用 CodeArtifact 命令的列表。

接下来，您可以创建 IAM 用户并授予该用户对 CodeArtifact 的访问权限。有关更多信息，请参阅预置 IAM 用户。

预置 IAM 用户

按照以下说明让 IAM 用户做好使用 CodeArtifact 的准备。

预置 IAM 用户

1. 创建 IAM 用户，或使用与您的 AWS 账户 关联的用户。有关更多信息，请参阅《IAM 用户指南》中的创建 IAM 用户和 AWS IAM 策略概述。
2. 授予 IAM 用户访问 CodeArtifact 的权限。

   - 选项 1：创建自定义 IAM 策略。使用自定义 IAM 策略，您可以提供所需的最低权限并更改身份验证令牌的持续时间。有关更多信息以及示例策略，请参阅基于身份的策略示例 AWS CodeArtifact。
   - 选项 2：使用 `AWSCodeArtifactAdminAccess` AWS 托管的策略。下面的代码段显示了此策略的内容。

     > **Important**
     >
     > 此策略授予对所有 CodeArtifact API 的访问权限。建议您始终使用完成任务所需的最低权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 最佳实践。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Action": [
            "codeartifact:*"
         ],
         "Effect": "Allow",
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
            "Condition": {
               "StringEquals": {
                  "sts:AWSServiceName": "codeartifact.amazonaws.com"
               }
```

```
            }
        }
    ]
}
```

调用 CodeArtifact `GetAuthorizationToken` API 需要 `sts:GetServiceBearerToken` 权限。此 API 返回一个令牌，在将程序包管理器（例如 npm 或 `pip`）与 CodeArtifact 配合使用时，必须使用该令牌。要将程序包管理器与 CodeArtifact 存储库配合使用，您的 IAM 用户或角色必须允许 `sts:GetServiceBearerToken`，如前面的策略示例所示。

如果您尚未安装计划与 CodeArtifact 一起使用的程序包管理器或构建工具，请参阅安装程序包管理器或构建工具。

安装程序包管理器或构建工具

要从 CodeArtifact 发布或使用程序包，必须使用程序包管理器。每种程序包类型都有不同的程序包管理器。以下列表包含一些可以与 CodeArtifact 一起使用的程序包管理器。如果尚未安装程序包管理器，请为要使用的程序包类型安装程序包管理器。

- 对于 npm，请使用 npm CLI 或 pnpm。
- 对于 Maven，请使用 Apache Maven (`mvn`) 或 Gradle。
- 对于 Python，使用 pip 来安装程序包，使用 twine 来发布程序包。
- 对于 NuGet，请使用 Visual Studio 中的 Toolkit for Visual Studio 或 nuget 或 dotnet CLI。
- 对于通用程序包，请使用 AWS CLI 或 SDK 来发布和下载程序包内容。

后续步骤

后续步骤取决于您与 CodeArtifact 一起使用的程序包类型，以及 CodeArtifact 资源的状态。

如果您是首次为自己、您的团队或组织开始使用 CodeArtifact，请参阅以下文档，了解一般入门信息和协助创建所需资源。

- 通过控制台开始使用
- 开始使用 AWS CLI

如果您的资源已经创建，并且您已准备好将程序包管理器配置为将程序包推送到 CodeArtifact 存储库或从 CodeArtifact 存储库安装程序包，请参阅与您的程序包类型或程序包管理器对应的文档。

- 将 CodeArtifact 与 npm 结合使用
- 将 CodeArtifact 与 Python 结合使用
- 将 CodeArtifact 与 Maven 结合使用
- 将 CodeArtifact 与 NuGet 结合使用
- 在 CodeArtifact 中使用通用程序包

开始使用 CodeArtifact

在本入门教程中，您将使用 CodeArtifact 来创建以下项：

- 一个名为 `my-domain` 的域。
- 包含在 `my-domain` 中的名为 `my-repo` 的存储库。
- 包含在 `my-domain` 中的名为 `npm-store` 的存储库。`npm-store` 与 npm 公有存储库建立了外部连接。此连接用于将 npm 程序包摄取到 `my-repo` 存储库中。

在开始本教程之前，我们建议您查看 CodeArtifact AWS CodeArtifact 概念。

> **Note**
>
> 本教程要求您创建的资源可能会导致您的 AWS 账户产生相关费用。有关更多信息，请参阅 CodeArtifact 定价。

主题

先决条件

您可以使用 AWS Management Console 或 AWS Command Line Interface (AWS CLI) 完成此教程。要按照教程操作，您必须先满足以下先决条件：

- 完成 设置 AWS CodeArtifact 中的步骤。
- 安装 npm CLI。有关更多信息，请参阅 npm 文档中的下载和安装 Node.js 和 npm。

通过控制台开始使用

使用 AWS Management Console，按照以下步骤来开始使用 CodeArtifact。本指南使用 npm 程序包管理器，如果您使用的是不同的程序包管理器，则需要修改以下某些步骤。

1. 通过以下网址登录 AWS Management Console 并打开 AWS CodeArtifact 控制台：https://console.aws.amazon.com/codesuite/codeartifact/start。有关更多信息，请参阅设置 AWS CodeArtifact。
2. 选择创建存储库。
3. 在存储库名称中，输入 **my-repo**。
4. （可选）在存储库描述中，输入存储库的可选描述。
5. 在公有上游存储库中，选择 npm-store 来创建一个连接到 npmjs 的存储库，该存储库是 `my-repo` 存储库的上游存储库。

   CodeArtifact 会为这个存储库分配名称 `npm-store`。上游存储库 `npm-store` 中提供的所有可用程序包也可用于其下游存储库 `my-repo`。
6. 选择 Next（下一步）。
7. 在 AWS 账户中，选择此 AWS 账户。
8. 在域名中，输入 **my-domain**。
9. 展开 Additional configuration (其他配置)。
10. 您必须使用 AWS KMS key（KMS 密钥）来加密域中的所有资产。您可以使用 AWS 托管式密钥或自己管理的 KMS 密钥：

    - 如果您想使用默认 AWS 托管式密钥，请选择 AWS 托管式密钥。
    - 如果您想使用自己管理的 KMS 密钥，请选择客户管理的密钥。要使用自己管理的 KMS 密钥，请在客户管理的密钥 ARN 中搜索并选择 KMS 密钥。

    有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS 托管式密钥和客户管理的密钥。
11. 选择 Next（下一步）。
12. 在查看并创建中，查看 CodeArtifact 正在为您创建的内容。

    - 程序包流显示了 `my-domain`、`my-repo` 和 `npm-store` 之间如何相互关联。
    - 步骤 1：创建存储库显示了有关 `my-repo` 和 `npm-store` 的详细信息。
    - 步骤 2: 选择域显示有关 `my-domain` 的详细信息。

    当您准备好后，选择创建存储库。
13. 在 my-repo 页面上，选择查看连接说明，然后选择 npm。

14. 使用 AWS CLI 来运行使用此 AWS CLI CodeArtifact 命令配置 npm 客户端下所示的 `login` 命令。

```
aws codeartifact login --tool npm --repository my-repo --domain my-domain --domain-owner 111122223333
```

您应该会收到确认登录成功的输出。

```
Successfully configured npm to use AWS CodeArtifact repository https://my-domain-111122223333.d.codeartifact.us-east-2.amazonaws.com/npm/my-repo/
Login expires in 12 hours at 2020-10-08 02:45:33-04:00
```

如果您收到错误消息 `Could not connect to the endpoint URL`，请确保 AWS CLI 已配置完毕且您的默认区域名称设置为在创建存储库时使用的相同区域，请参阅配置 AWS 命令行界面。

有关更多信息，请参阅在 CodeArtifact 中配置和使用 npm。

15. 使用 npm CLI 来安装 npm 程序包。例如，要安装常见的 npm 程序包 `lodash`，请使用以下命令。

```
npm install lodash
```

16. 返回 CodeArtifact 控制台。如果您的 my-repo 存储库已打开，请刷新页面。否则，在导航窗格中，选择存储库，然后选择 my-repo。

在程序包下面，您应该会看到您安装的 npm 库或程序包。您可以选择程序包的名称来查看其版本和状态。您可以选择其最新版本来查看程序包的详细信息，例如依赖项、资产等。

> **Note**
>
> 从安装程序包到将其提取到存储库之间可能会有一段延迟。

17. 为避免产生进一步的 AWS 费用，请删除您在本教程中使用的资源：

> **Note**
>
> 您无法删除包含存储库的域，因此必须先删除 `my-repo` 和 `npm-store`，然后才能删除 `my-domain`。

a. 在导航窗格中，选择存储库。

b. 选择 npm-store，选择删除，然后按照步骤来删除存储库。

c. 选择 my-repo，选择删除，然后按照步骤来删除存储库。

d. 在导航窗格中，选择域。

e. 选择 my-domain，选择删除，然后按照步骤删除域。

开始使用 AWS CLI

使用 AWS Command Line Interface (AWS CLI)，按照以下步骤来开始使用 CodeArtifact。有关更多信息，请参阅安装或升级 AWS CLI，然后对其进行配置。本指南使用 npm 程序包管理器，如果您使用的是不同的程序包管理器，则需要修改以下某些步骤。

1. 使用 AWS CLI 运行 create-domain 命令。

```
aws codeartifact create-domain --domain my-domain
```

输出中会显示 JSON 格式的数据，并包含有关新域的详细信息。

```
{
    "domain": {
        "name": "my-domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my-domain",
        "status": "Active",
        "createdTime": "2020-10-07T15:36:35.194000-04:00",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0
    }
}
```

如果您收到错误消息 `Could not connect to the endpoint URL`，请确保 AWS CLI 已配置完毕且您的默认区域名称设置为在创建存储库时使用的相同区域，请参阅配置 AWS 命令行界面。

2. 使用 create-repository 命令在您的域中创建存储库。

```
aws codeartifact create-repository --domain my-domain --domain-owner 111122223333
 --repository my-repo
```

输出中会显示 JSON 格式的数据，并包含有关新存储库的详细信息。

```
{
    "repository": {
        "name": "my-repo",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/my-repo",
        "upstreams": [],
        "externalConnections": []
    }
}
```

3. 使用 create-repository 命令为您的 `my-repo` 存储库创建上游存储库。

```
aws codeartifact create-repository --domain my-domain --domain-owner 111122223333
 --repository npm-store
```

输出中会显示 JSON 格式的数据，并包含有关新存储库的详细信息。

```
{
    "repository": {
        "name": "npm-store",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/npm-store",
        "upstreams": [],
        "externalConnections": []
    }
}
```

4. 使用 associate-external-connection 命令将 npm 公有存储库的外部连接添加到您的 `npm-store` 存储库中。

```
aws codeartifact associate-external-connection --domain my-domain --domain-
owner 111122223333 --repository npm-store --external-connection "public:npmjs"
```

输出中会显示 JSON 格式的数据，并包含有关存储库及其新外部连接的详细信息。

```
{
    "repository": {
        "name": "npm-store",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/npm-store",
        "upstreams": [],
        "externalConnections": [
            {
                "externalConnectionName": "public:npmjs",
                "packageFormat": "npm",
                "status": "AVAILABLE"
            }
        ]
    }
}
```

有关更多信息，请参阅将 CodeArtifact 存储库连接到公有存储库。

5. 使用 update-repository 命令将 npm-store 存储库作为上游存储库关联到 my-repo 存储库。

```
aws codeartifact update-repository --repository my-repo --domain my-domain --
domain-owner 111122223333 --upstreams repositoryName=npm-store
```

输出中会显示 JSON 格式的数据，并包含有关已更新的存储库（包括其新的上游存储库）的详细信息。

```
{
    "repository": {
        "name": "my-repo",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
```

```
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/my-repo",
        "upstreams": [
            {
                "repositoryName": "npm-store"
            }
        ],
        "externalConnections": []
    }
}
```

有关更多信息，请参阅添加或删除上游存储库 (AWS CLI)。

6. 使用 login 命令对 my-repo 存储库配置 npm 程序包管理器。

```
aws codeartifact login --tool npm --repository my-repo --domain my-domain --domain-
owner 111122223333
```

您应该会收到确认登录成功的输出。

```
Successfully configured npm to use AWS CodeArtifact repository https://my-
domain-111122223333.d.codeartifact.us-east-2.amazonaws.com/npm/my-repo/
Login expires in 12 hours at 2020-10-08 02:45:33-04:00
```

有关更多信息，请参阅在 CodeArtifact 中配置和使用 npm。

7. 使用 npm CLI 来安装 npm 程序包。例如，要安装常见的 npm 程序包 lodash，请使用以下命令。

```
npm install lodash
```

8. 使用 list-packages 命令来查看您刚刚在 my-repo 存储库中安装的程序包。

> **Note**
>
> 从 npm install 安装命令完成到存储库中显示程序包之间可能会有一段延迟。有关从公有存储库提取程序包时的典型延迟的详细信息，请参阅外部连接延迟。

```
aws codeartifact list-packages --domain my-domain --repository my-repo
```

输出中会显示 JSON 格式的数据，包括您安装的程序包的格式和名称。

```
{
    "packages": [
        {
            "format": "npm",
            "package": "lodash"
        }
    ]
}
```

您现在有三个 CodeArtifact 资源：

- 域 my-domain。
- my-domain 中包含的存储库 my-repo。这个存储库有一个 npm 程序包。
- my-domain 中包含的存储库 npm-store。这个存储库与公有 npm 存储库建立了外部连接，并作为上游存储库与 my-repo 存储库相关联。

9. 为避免产生进一步的 AWS 费用，请删除您在本教程中使用的资源：

> **Note**
>
> 您无法删除包含存储库的域，因此必须先删除 my-repo 和 npm-store，然后才能删除 my-domain。

a. 使用 delete-repository 命令来删除 npm-store 存储库。

```
aws codeartifact delete-repository --domain my-domain --domain-
owner 111122223333 --repository my-repo
```

输出中会显示 JSON 格式的数据，并包含有关已删除存储库的详细信息。

```
{
    "repository": {
        "name": "my-repo",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
```

```
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/my-repo",
        "upstreams": [
            {
                "repositoryName": "npm-store"
            }
        ],
        "externalConnections": []
    }
}
```

b. 使用 delete-repository 命令来删除 npm-store 存储库。

```
aws codeartifact delete-repository --domain my-domain --domain-
owner 111122223333 --repository npm-store
```

输出中会显示 JSON 格式的数据，并包含有关已删除存储库的详细信息。

```
{
    "repository": {
        "name": "npm-store",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/npm-store",
        "upstreams": [],
        "externalConnections": [
            {
                "externalConnectionName": "public:npmjs",
                "packageFormat": "npm",
                "status": "AVAILABLE"
            }
        ]
    }
}
```

c. 使用 delete-domain 命令来删除 my-domain 存储库。

```
aws codeartifact delete-domain --domain my-domain --domain-owner 111122223333
```

输出中会显示 JSON 格式的数据，并包含有关已删除域的详细信息。

```
{
    "domain": {
        "name": "my-domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my-domain",
        "status": "Deleted",
        "createdTime": "2020-10-07T15:36:35.194000-04:00",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0
    }
}
```

在中使用存储库 CodeArtifact

这些主题向您展示如何使用 CodeArtifact 控制台 AWS CLI、和 CodeArtifact API 创建、列出、更新和删除存储库。

主题

创建存储库

由于中的所有软件包 CodeArtifact 都存储在存储库中，因此要使用 CodeArtifact，您必须创建一个。您可以使用 CodeArtifact 控制台、 AWS Command Line Interface (AWS CLI) 或创建存储库 AWS CloudFormation。每个存储库都与您在创建存储库时使用的 AWS 账户相关联。您可以有多个存储库，按域来创建存储库并进行分组。创建存储库时，存储库中不包含任何程序包。存储库是多语言的，这意味着单个存储库可以包含任何受支持类型的程序包。

有关 CodeArtifact 服务限制的信息，例如单个域中允许的最大存储库数量，请参阅AWS CodeArtifact 中的配额。如果达到了允许的最大存储库数量，则可以删除存储库来腾出空间，用于容纳更多存储库。

一个存储库可以有一个或多个与之关联的 CodeArtifact 仓库作为上游存储库。这样，程序包管理器客户端就可以使用单个 URL 端点访问多个存储库中包含的程序包。有关更多信息，请参阅 在 CodeArtifact 中使用上游存储库。

有关使用管理 CodeArtifact 仓库的更多信息 CloudFormation，请参阅使用 AWS CloudFormation 创建 CodeArtifact 资源。

> **Note**
>
> 创建存储库后，便无法更改其名称、关联的 AWS 账户或域。

主题

创建存储库（控制台）

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择存储库，然后选择创建存储库。
3. 在存储库名称中，输入存储库的名称。
4. （可选）在存储库描述中，输入存储库的可选描述。
5. （可选）在发布上游存储库中，添加可用于将您的存储库与程序包发布机构（例如 Maven Central 或 npmjs.com）连接起来的中间存储库。
6. 选择下一步。
7. 在 AWS 账户中，如果您登录的是拥有该域的账户，请选择此 AWS 账户。如果另一个 AWS 账户拥有该域，请选择不同 AWS 账户。
8. 在域中，选择要在其中创建存储库的域。

   如果账户中没有域，则必须创建一个域。在域名中，输入新域的名称。

   展开其他配置。

   您必须使用 AWS KMS key（KMS 密钥）来加密您域中的所有资产。您可以使用自己管理的 AWS 托管式密钥 或 KMS 密钥：

   > **Important**
   >
   > CodeArtifact 仅支持对称 KMS 密钥。您不能使用非对称 KMS 密钥来加密您的 CodeArtifact 域名。要获取确定 KMS 密钥是对称还是非对称的帮助，请参阅识别对称密钥和非对称密钥。

   - 如果您想使用默认 AWS 托管式密钥，请选择 AWS 托管式密钥。

- 如果您想使用自己管理的 KMS 密钥，请选择客户管理的密钥。要使用自己管理的 KMS 密钥，请在客户管理的密钥 ARN 中搜索并选择 KMS 密钥。

有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS 托管式密钥和客户管理的密钥。

9. 选择下一步。
10. 在 “查看并创建” 中，查看 CodeArtifact 正在为您创建的内容。

- 程序包流显示您的域和存储库如何连接。
- 步骤 1: 创建存储库显示有关存储库以及将要创建的可选上游存储库的详细信息。
- 步骤 2: 选择域显示有关 `my_domain` 的详细信息。

当您准备好后，选择创建存储库。

创建存储库 (AWS CLI)

使用 `create-repository` 命令在您的域中创建存储库。

```
aws codeartifact create-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo --description "My new repository"
```

输出示例：

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo",
        "description": "My new repository",
        "upstreams": "[]",
        "externalConnections"" "[]"
    }
}
```

新的存储库不包含任何程序包。每个存储库都与您在创建存储库时经过身份验证的 AWS 账户相关联。

创建带标签的存储库

要创建带标签的存储库，请在 `create-domain` 命令中添加 `--tags` 参数。

```
aws codeartifact create-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo --tags key=k1,value=v1 key=k2,value=v2
```

创建具有上游存储库的存储库

在创建存储库时，可以指定一个或多个上游存储库。

```
aws codeartifact create-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo \
  --upstreams repositoryName=my-upstream-repo --repository-description "My new
 repository"
```

输出示例：

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo",
        "description": "My new repository",
        "upstreams": [
            {
                "repositoryName": "my-upstream-repo"
            }
        ],
        "externalConnections"" "[]"
    }
}
```

> **Note**
>
> 要创建带有上游存储库的存储库，您必须拥有对上游存储库执行 `AssociateWithDownstreamRepository` 操作的权限。

要在创建后向存储库中添加上游存储库，请参阅添加或删除上游存储库（控制台）和添加或删除上游存储库 (AWS CLI)。

连接存储库

将您的个人资料和凭据配置为对您的 AWS 账户进行身份验证后，请决定要在哪个存储库中使用 CodeArtifact。您有以下选项：

- 创建存储库。有关更多信息，请参阅创建存储库。
- 使用账户中已存在的存储库。您可以使用 `list-repositories` 命令来查找在您的 AWS 账户中创建的存储库。有关更多信息，请参阅 列出存储库。
- 使用其他 AWS 账户中的存储库。有关更多信息，请参阅存储库策略。

使用程序包管理器客户端

在知道要使用哪个存储库后，请参阅以下主题之一。

- CodeArtifact 与 Maven 一起使用
- CodeArtifact 与 npm 一起使用
- CodeArtifact 与一起使用 NuGet
- CodeArtifact 与 Python 一起使用

删除存储库

您可以使用 CodeArtifact 控制台或删除存储库 AWS CLI。删除存储库后，即不能再向存储库推送程序包或从中提取程序包。存储库中的所有程序包都变为永久不可用且无法还原。您可以创建一个同名的存储库，但其内容将为空。

主题

- 删除存储库（控制台）
- 删除存储库 (AWS CLI)

删除存储库（控制台）

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格上，选择存储库，然后选择要删除的存储库。
3. 选择删除，然后按照步骤删除域。

删除存储库 (AWS CLI)

使用 `delete-repository` 命令来删除存储库。

```
aws codeartifact delete-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

输出示例：

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "123456789012",
        "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain/my_repo",
        "description": "My new repository",
        "upstreams": [],
        "externalConnections": []
    }
}
```

列出存储库

使用本主题中的命令列出 AWS 账户或域中的存储库。

列出 AWS 账户中的存储库

使用此命令列出您 AWS 账户中的所有存储库。

```
aws codeartifact list-repositories
```

示例输出：

```
{
    "repositories": [
        {
            "name": "repo1",
            "administratorAccount": "123456789012",
            "domainName": "my_domain",
            "domainOwner": "123456789012",
            "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain/repo1",
            "description": "Description of repo1"
        },
        {
            "name": "repo2",
            "administratorAccount": "123456789012",
            "domainName": "my_domain",
            "domainOwner": "123456789012",
            "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain/repo2",
            "description": "Description of repo2"

        },
        {
            "name": "repo3",
            "administratorAccount": "123456789012",
            "domainName": "my_domain2",
            "domainOwner": "123456789012",
            "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain2/repo3",
            "description": "Description of repo3"
        }
    ]
}
```

您可以使用 `--max-results` 和 `--next-token` 参数对来自 `list-repositories` 的响应进行分页。对于 `--max-results`，指定一个 1 至 1000 之间的整数，用来指定在一页中返回的结果数。其默认值为 50。要返回后续页面，请再次运行 `list-repositories` 并将上一个命令输出中接收到的 `nextToken` 值传递给 `--next-token`。如果未使用 `--next-token` 选项，则始终返回结果的第一页。

列出域中的存储库

使用 `list-repositories-in-domain` 来获取域中所有存储库的列表。

```
aws codeartifact list-repositories-in-domain --domain my_domain --domain-
owner 123456789012 --max-results 3
```

输出显示，有些存储库由不同的 AWS 账户管理。

```
{
    "repositories": [
        {
            "name": "repo1",
            "administratorAccount": "123456789012",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo1",
            "description": "Description of repo1"
        },
        {
            "name": "repo2",
            "administratorAccount": "444455556666",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo2",
            "description": "Description of repo2"
        },
        {
            "name": "repo3",
            "administratorAccount": "444455556666",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo3",
            "description": "Description of repo3"
        }
    ]
}
```

您可以使用 --max-results 和 --next-token 参数对来自 list-repositories-in-domain 的响应进行分页。对于 --max-results，指定一个 1 至 1000 之间的整数，用来指定在一页中返回的结果数。其默认值为 50。要返回后续页面，请再次运行 list-repositories-in-domain 并将上一个命令输出中接收到的 nextToken 值传递给 --next-token。如果未使用 --next-token 选项，则始终返回结果的第一页。

要以更精简的列表输出存储库名称，请尝试以下命令。

```
aws codeartifact list-repositories-in-domain --domain my_domain --domain-
owner 111122223333 \
  --query 'repositories[*].[name]' --output text
```

示例输出：

```
repo1
repo2
repo3
```

以下示例除了输出存储库名称外，还输出账户 ID。

```
aws codeartifact list-repositories-in-domain --domain my_domain --domain-
owner 111122223333  \
  --query 'repositories[*].[name,administratorAccount]' --output text
```

示例输出：

```
repo1 710221105108
repo2 710221105108
repo3 532996949307
```

有关--query参数的更多信息，请参阅 CodeArtifact API 参考ListRepositories中的。

查看或修改存储库配置

您可以使用 CodeArtifact 控制台或 AWS Command Line Interface (AWS CLI) 查看和更新有关存储库的详细信息。

> **Note**
>
> 创建存储库后，便无法更改其名称、关联的 AWS 账户或域。

主题

查看或修改存储库配置（控制台）

您可以使用 CodeArtifact 控制台查看有关存储库的详细信息并对其进行更新。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择存储库，然后选择要查看或修改的存储库名称。
3. 展开详细信息，查看以下信息：
   - 存储库的域。选择域名来了解有关域的更多信息。
   - 存储库的资源策略。选择应用存储库策略来添加存储库策略。
   - 存储库的 Amazon 资源名称 (ARN)。
   - 如果您的存储库有外部连接，则可以选择该连接来了解更多信息。存储库只能有一个外部连接。有关更多信息，请参阅 将 CodeArtifact 存储库连接到公有存储库。
   - 如果您的存储库有上游存储库，则可以选择一个存储库来查看其详细信息。存储库最多可以有 10 个直接上游存储库。有关更多信息，请参阅 在 CodeArtifact 中使用上游存储库。

   > **Note**
   >
   > 存储库可以有外部连接或上游存储库，但不能两者兼而有之。
4. 在程序包中，您可以看到此存储库可用的所有程序包。选择程序包来了解有关程序包的更多信息。
5. 选择 “查看连接说明”，然后选择软件包管理器以了解如何对其进行配置 CodeArtifact。
6. 选择应用存储库策略来更新策略或向存储库添加资源策略。有关更多信息，请参阅 存储库策略。
7. 选择编辑来添加或更新以下项。

- 存储库说明。
- 与存储库关联的标签。
- 如果您的存储库有外部连接，则可以更改它连接到哪个公有存储库。另外，您可以将一个或多个现有存储库添加为上游存储库。按照您希望在申请包裹 CodeArtifact 时按优先顺序排列它们。有关更多信息，请参阅 上游存储库优先顺序。

查看或修改存储库配置 (AWS CLI)

要在中查看存储库的当前配置 CodeArtifact，请使用`describe-repository`命令。

```
aws codeartifact describe-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

输出示例：

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012,
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo"
        "upstreams": [],
        "externalConnections": []
    }
}
```

修改存储库上游配置

借助上游存储库，程序包管理器客户端可以使用单个 URL 端点访问多个存储库中包含的程序包。要添加或更改存储库的上游关系，请使用 `update-repository` 命令。

```
aws codeartifact update-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo \
       --upstreams repositoryName=my-upstream-repo
```

输出示例：

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012,
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo"
        "upstreams": [
            {
                "repositoryName": "my-upstream-repo"
            }
        ],
        "externalConnections": []
    }
}
```

> **Note**
>
> 要添加上游存储库，您必须拥有对上游存储库执行 `AssociateWithDownstreamRepository` 操作的权限。

要移除存储库的上游关系，请使用空列表作为 `--upstreams` 选项的参数。

```
aws codeartifact update-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo --upstreams []
```

输出示例：

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012,
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo"
        "upstreams": [],
        "externalConnections": []
    }
```

```
}
```

存储库策略

CodeArtifact 使用基于资源的权限来控制访问权限。基于资源的权限让您可以指定能够访问存储库的用户，以及这些用户可以对该存储库执行的操作。默认情况下，只有存储库所有者有权访问存储库。您可以应用策略文档，让其他 IAM 主体可以访问您的存储库。

有关更多信息，请参阅基于资源的策略以及基于身份的策略和基于资源的策略。

创建资源策略来授予读取访问权限

资源策略是 JSON 格式的文本文件。该文件必须指定主体 (actor)、一个或多个操作以及效果（`Allow` 或 `Deny`）。例如，以下资源策略向账户 `123456789012` 授予从存储库下载程序包的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:ReadFromRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "*"
        }
    ]
}
```

由于策略仅针对策略所附加到的存储库的操作进行评估，因此无需指定资源。由于资源是隐含的，因此可以将 `Resource` 设置为 `*`。为了让程序包管理器可从这个存储库下载程序包，还需要创建域策略来实现跨账户访问。域策略必须至少向主体授予 `codeartifact:GetAuthorizationToken` 和 `sts:GetServiceBearerToken` 权限。有关授予跨账户访问权限的完整域策略的示例，请参阅这个域策略示例。

> **Note**
>
> codeartifact:ReadFromRepository 操作只能在存储库资源上使用。您不能将程序包的 Amazon 资源名称 (ARN) 作为资源，将 codeartifact:ReadFromRepository 作为操作来允许对存储库中的程序包子集进行读取访问。给定的主体可以读取存储库中的所有程序包，也可以不读取任何程序包。

由于存储库中指定的唯一操作是 ReadFromRepository，因此账户 1234567890 中的用户和角色可以从存储库下载程序包。但是，他们无法对其执行其他操作（例如，列出程序包名称和版本）。通常，因为从存储库下载程序包的用户也需要以其他方式与存储库进行交互，所以除了 ReadFromRepository 之外，您还需要在以下策略中授予权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:DescribePackageVersion",
                "codeartifact:DescribeRepository",
                "codeartifact:GetPackageVersionReadme",
                "codeartifact:GetRepositoryEndpoint",
                "codeartifact:ListPackages",
                "codeartifact:ListPackageVersions",
                "codeartifact:ListPackageVersionAssets",
                "codeartifact:ListPackageVersionDependencies",
                "codeartifact:ReadFromRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "*"
        }
    ]
}
```

设置策略

创建策略文档后，使用 put-repository-permissions-policy 命令将其附加到存储库：

```
aws codeartifact put-repository-permissions-policy --domain my_domain --domain-
owner 111122223333 \
          --repository my_repo --policy-document file:///PATH/TO/policy.json
```

如果调用 put-repository-permissions-policy，则在评估权限时会忽略存储库上的资源策略。这样可以确保域的所有者不会将自己锁定在存储库之外，因而使他们无法更新资源策略。

> **Note**
>
> 您不能向其他 AWS 账户授予使用资源策略更新仓库资源策略的权限，因为调用时会忽略资源策略 put-repository-permissions-policy。

示例输出：

```
{
    "policy": {
        "resourceArn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo",
        "document": "{ ...policy document content...}",
        "revision": "MQlyyTQRASRU3HB58gBtSDHXG7Q3hvxxxxxxx="
    }
}
```

命令输出包含存储库资源的 Amazon 资源名称 (ARN)、策略文档的完整内容以及修订标识符。您可以使用 --policy-revision 选项将修订标识符传递给 put-repository-permissions-policy。这样可以确保覆盖文档的已知修订版，而不是由另一个作者设置的较新版本。

读取策略

使用 get-repository-permissions-policy 命令来读取策略文档的现有版本。要格式化输出来提高可读性，请将 --output 和 --query policy.document 与 Python json.tool 模块一起使用。

```
aws codeartifact get-repository-permissions-policy --domain my_domain --domain-
owner 111122223333 \
          --repository my_repo --output text --query policy.document | python -m
 json.tool
```

示例输出：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": [
                "codeartifact:DescribePackageVersion",
                "codeartifact:DescribeRepository",
                "codeartifact:GetPackageVersionReadme",
                "codeartifact:GetRepositoryEndpoint",
                "codeartifact:ListPackages",
                "codeartifact:ListPackageVersions",
                "codeartifact:ListPackageVersionAssets",
                "codeartifact:ListPackageVersionDependencies",
                "codeartifact:ReadFromRepository"
            ],
            "Resource": "*"
        }
    ]
}
```

删除策略

使用 `delete-repository-permissions-policy` 命令从存储库中删除策略。

```
aws codeartifact delete-repository-permissions-policy --domain my_domain --domain-
owner 111122223333 \
          --repository my_repo
```

输出的格式与 `get-repository-permissions-policy` 命令的输出格式相同。

向主体授予读取访问权限

当您在策略文档中将账户的根用户指定为主体时，即向该账户中的所有用户和角色授予访问权限。要限制选定用户或角色的访问权限，请在策略的 `Principal` 部分使用他们的 ARN。例如，使用以下方法向账户 123456789012 中的 IAM 用户 bob 授予读取权限。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:ReadFromRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:user/bob"
            },
            "Resource": "*"
        }
    ]
}
```

授予对程序包的写入权限

codeartifact:PublishPackageVersion 操作用于控制发布程序包新版本的权限。与此操作一起使用的资源必须是程序包。 CodeArtifact 软件包 ARN 的格式如下所示。

```
arn:aws:codeartifact:region-id:111122223333:package/my_domain/my_repo/package-format/package-namespace/package-name
```

以下示例显示了域 my_domain 的 example-repo 存储库中作用域为 @parity 且名称为 ui 的 npm 程序包的 ARN。

```
arn:aws:codeartifact:region-id:111122223333:package/my_domain/example-repo/npm/parity/ui
```

如果一个 npm 程序包没有作用域，那么它的 ARN 的命名空间字段为空字符串。以下示例显示了域 my_domain 的 example-repo 存储库中没有作用域且名称为 react 的程序包的 ARN。

```
arn:aws:codeartifact:region-id:111122223333:package/my_domain/example-repo/npm//react
```

以下策略向账户 123456789012 授予在 example-repo 存储库中发布 @parity/ui 版本的权限。

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

```
                "codeartifact:PublishPackageVersion"
            ],
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "arn:aws:codeartifact:region-
id:111122223333:package/my_domain/example-repo/npm/parity/ui"
        }
    ]
}
```

> **Important**
>
> 要授予发布 Maven 和 NuGet 软件包版本的权限，请除添加以下权限外。`codeartifact:PublishPackageVersion`
>
> 1. NuGet: `codeartifact:ReadFromRepository` 并指定存储库资源
> 2. Maven：`codeartifact:PutPackageMetadata`

由于此策略将域和存储库指定为资源的一部分，所以仅当连接到该存储库时才允许发布。

授予对存储库的写入权限

您可以使用通配符来授予对存储库中所有程序包的写入权限。例如，使用以下策略向账户授予对 `example-repo` 存储库中所有程序包的写入权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:PublishPackageVersion"
            ],
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "arn:aws:codeartifact:region-
id:111122223333:package/my_domain/my_repo/*"
```

```
        }
    ]
}
```

在中标记存储库 CodeArtifact

标签是与 AWS 资源关联的键/值对。您可以将标签应用于中的仓库 CodeArtifact。有关 CodeArtifact 资源标记、用例、标签键和值限制以及支持的资源类型的信息，请参阅标记资源。

您可以使用 CLI 在创建存储库时指定标签。您可以使用控制台或 CLI 来添加或删除标签，以及更新存储库中标签的值。您最多可以为每个存储库添加 50 个标签。

主题

- 标记存储库 (CLI)
- 标记存储库（控制台）

标记存储库 (CLI)

您可以使用 CLI 来管理存储库标签。

主题

- 向存储库添加标签 (CLI)
- 查看存储库的标签 (CLI)
- 编辑存储库的标签 (CLI)
- 移除存储库的标签 (CLI)

向存储库添加标签 (CLI)

您可以使用控制台或 AWS CLI 来标记存储库。

要在创建存储库为其添加标签，请参阅创建存储库。

在这些步骤中，我们假设您已安装最新版本的 AWS CLI 或已更新到当前版本。有关更多信息，请参阅安装 AWS Command Line Interface。

在终端或命令行中运行 tag-resource 命令，并指定要添加标签的存储库的 Amazon 资源名称 (ARN) 以及要添加的标签的键和值。

> **Note**
>
> 要获取存储库的 ARN，请运行 `describe-repository` 命令：
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

您可以将多个标签添加到一个存储库中。例如，要在名为 *my_domain* 的域中为名为 *my_repo* 的存储库添加两个标签，一个标签键名为 *key1*，标签值为 *value1*，另一个标签键名为 *key2*，标签值为 *value2*，请执行以下命令：

```
aws codeartifact tag-resource --resource-arn arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo --tags key=key1,value=value1
 key=key2,value=value2
```

如果成功，此命令没有输出。

查看存储库的标签 (CLI)

按照以下步骤使用 AWS CLI 来查看存储库的 AWS 标签。如果尚未添加标签，则返回的列表为空。

在终端或命令行中，运行 list-tags-for-resource 命令。

> **Note**
>
> 要获取存储库的 ARN，请运行 `describe-repository` 命令：
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

例如，要查看名为 *my_domain* 且具有 `arn:aws:codeartifact:us-west-2:111122223333:repository/my_domain/my_repo` ARN 值的域中名为 *my_repo* 的存储库的标签键和标签值列表，请运行以下命令：

```
aws codeartifact list-tags-for-resource --resource-arn arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo
```

如果成功，该命令返回类似以下内容的信息：

```
{
    "tags": {
        "key1": "value1",
        "key2": "value2"
    }
}
```

编辑存储库的标签 (CLI)

按照以下步骤 AWS CLI 使用编辑存储库的标签。您可以更改现有键的值或添加另一个键。

在终端或命令行中运行 tag-resource 命令，指定要为其更新标签的存储库的 ARN 并指定标签键和标签值。

> **Note**
>
> 要获取存储库的 ARN，请运行 `describe-repository` 命令：
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

```
aws codeartifact tag-resource --resource-arn arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo --tags key=key1,value=newvalue1
```

如果成功，此命令没有输出。

移除存储库的标签 (CLI)

按照以下步骤使用从 AWS CLI 存储库中移除标签。

> **Note**
>
> 如果删除 存储库，则会从删除的存储库中删除所有标签关联。您无需在删除存储库之前移除标签。

在终端或命令行中运行 untag-resource 命令，指定要从中删除标签的存储库的 ARN 以及要删除的标签的标签键。

> **Note**
>
> 要获取存储库的 ARN，请运行 `describe-repository` 命令：
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

例如，要在名为 *my_domain* 的域中名为 *my_repo* 的存储库中移除标签键为 *key1* 和 *key2* 的多个标签，请运行以下命令：

```
aws codeartifact untag-resource --resource-arn arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo --tag-keys key1 key2
```

如果成功，此命令没有输出。移除标签后，您可以使用 `list-tags-for-resource` 命令查看存储库中剩余的标签。

标记存储库（控制台）

您可以使用控制台或 CLI 来标记资源。

主题

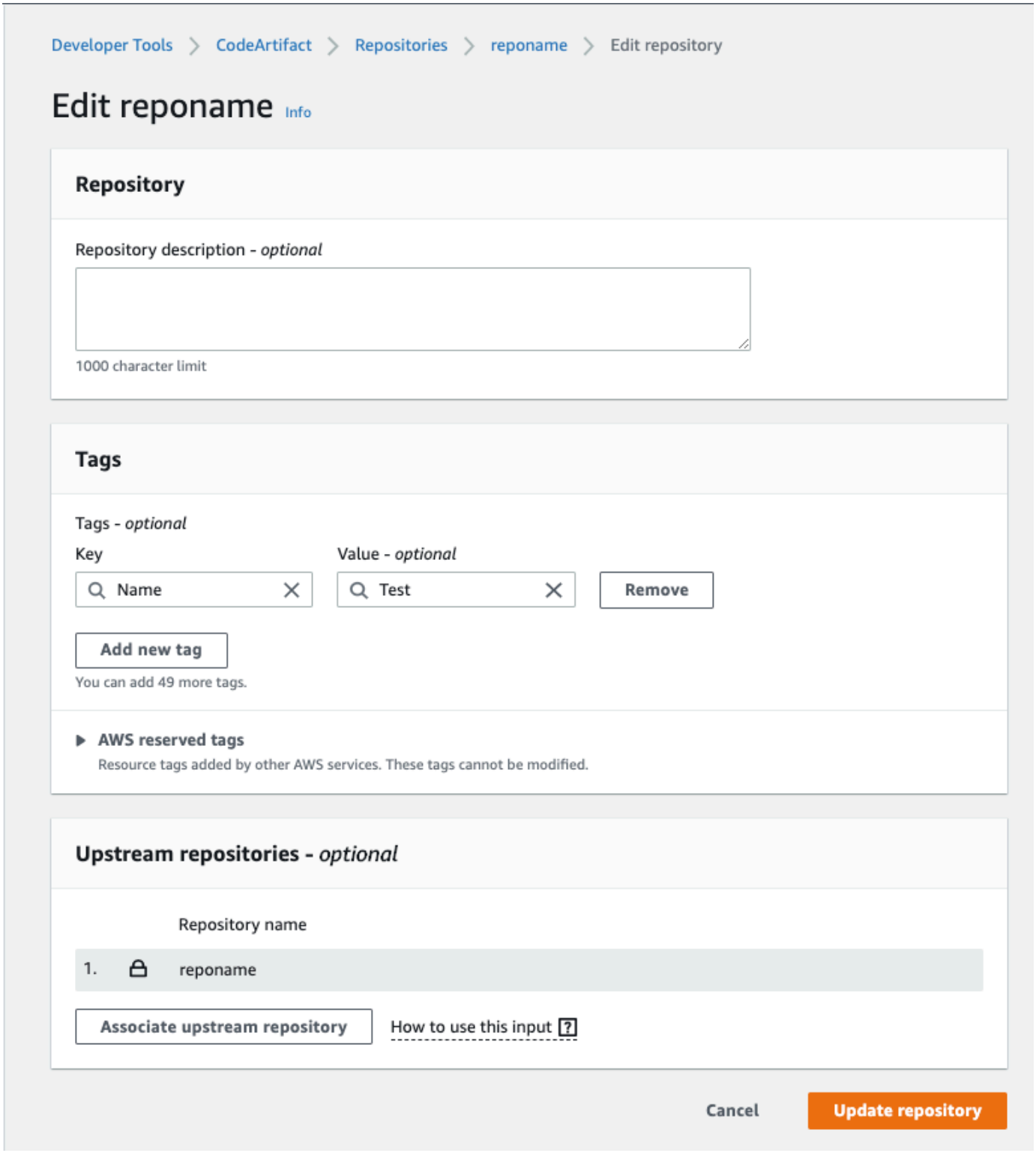
- 向存储库添加标签（控制台）
- 查看存储库的标签（控制台）
- 编辑存储库的标签（控制台）
- 从存储库中移除标签（控制台）

向存储库添加标签（控制台）

您可以使用控制台向现有存储库添加标签。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在存储库页面上，选择要为其添加标签的存储库。

3. 展开详细信息部分。
4. 在存储库标签下面，如果存储库没有标签，请选择添加存储库标签。如果存储库有标签，请选择查看和编辑存储库标签。
5. 选择添加新标签。
6. 在键和值字段中，输入要添加的每个标签的文本。（值字段为可选项。）例如，在键中，输入 **Name**。在值中，输入 **Test**。



7. （可选）选择添加标签以添加多行并输入多个标签。
8. 选择更新存储库。

查看存储库的标签（控制台）

您可以使用控制台列出现有存储库的标签。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在存储库页面上，选择要在其中查看标签的存储库。
3. 展开详细信息部分。
4. 在存储库标签下面，选择查看和编辑存储库标签。

> **Note**
>
> 如果未向此存储库添加任何标签，则控制台会显示添加存储库标签。

编辑存储库的标签（控制台）

您可以使用控制台来编辑已添加到存储库的标签。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在存储库页面上，选择要在其中更新标签的存储库。
3. 展开详细信息部分。
4. 在存储库标签下面，选择查看和编辑存储库标签。

> **Note**
>
> 如果未向此存储库添加任何标签，则控制台会显示添加存储库标签。

5. 在键和值字段中，根据需要更新每个字段的值。例如，对于 **Name** 键，在值中，将 **Test** 更改为 **Prod**。
6. 选择更新存储库。

从存储库中移除标签（控制台）

您可以使用控制台从存储库删除标签。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在存储库页面上，选择要在其中删除标签的存储库。
3. 展开详细信息部分。
4. 在存储库标签下面，选择查看和编辑存储库标签。

> **Note**
>
> 如果未向此存储库添加任何标签，则控制台会显示添加存储库标签。

5. 接下来，对于您要删除的每个标签的键和值，选择删除。
6. 选择更新存储库。

在 CodeArtifact 中使用上游存储库

一个存储库可以将其他 AWS CodeArtifact 存储库作为上游存储库。这样，程序包管理器客户端就可以使用单个存储库端点访问多个存储库中包含的程序包。

您可以使用 AWS Management Console、AWS CLI 或 SDK 将一个或多个上游存储库添加到 AWS CodeArtifact 存储库。要将一个存储库与上游存储库关联，您必须拥有对上游存储库执行 `AssociateWithDownstreamRepository` 操作的权限。有关更多信息，请参阅 创建具有上游存储库的存储库 和 添加或删除上游存储库：

如果上游存储库与公有存储库建立了外部连接，则其下游存储库可以从该公有存储库提取程序包。例如，假设存储库 `my_repo` 有一个名为 `upstream` 的上游存储库，并且 `upstream` 与公有 npm 存储库建立了外部连接。在这种情况下，连接到 `my_repo` 的程序包管理器可以从 npm 公有存储库提取程序包。有关从上游存储库或通过外部连接请求程序包的更多信息，请参阅请求包含上游存储库的程序包版本或从外部连接请求程序包。

主题

上游存储库和外部连接之间有什么区别？

在 CodeArtifact 中，上游存储库和外部连接的行为基本相同，但有一些重要的区别。

1. 您最多可将 10 个上游存储库添加到 CodeArtifact 存储库。您只能添加一个外部连接。
2. 有单独的 API 调用可用来添加上游存储库或外部连接。
3. 程序包保留行为略有不同，因为从上游存储库请求的程序包会保留在这些存储库中。有关更多信息，请参阅中间存储库中的程序包保留。

添加或删除上游存储库

按照以下各节中的步骤，在 CodeArtifact 存储库中添加或移除上游存储库。有关上游存储库的更多信息，请参阅在 CodeArtifact 中使用上游存储库。

本指南包含有关将其他 CodeArtifact 存储库配置为上游存储库的信息。有关配置与 npmjs.com、Nuget Gallery、Maven Central 或 PyPI 等公有存储库的外部连接的信息，请参阅添加外部连接。

添加或删除上游存储库（控制台）

执行以下程序中的步骤，使用 CodeArtifact 控制台将存储库添加为上游存储库。有关使用 AWS CLI 添加上游存储库的信息，请参阅添加或删除上游存储库 (AWS CLI)。

使用 CodeArtifact 控制台添加上游存储库

1. 打开 AWS CodeArtifact 控制台，网址为：https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择域，然后选择包含您的存储库的域名。
3. 选择您的存储库的名称。
4. 选择编辑。
5. 在上游存储库中，选择关联上游存储库，然后添加要作为上游存储库添加的存储库。您只能在与上游存储库相同的域中添加存储库。
6. 选择更新存储库。

使用 CodeArtifact 控制台删除上游存储库

1. 打开 AWS CodeArtifact 控制台，网址为：https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择域，然后选择包含您的存储库的域名。
3. 选择您的存储库的名称。
4. 选择编辑。
5. 在上游存储库中，找到要删除的上游存储库的列表条目，然后选择取消关联。

> **Important**
>
> 从 CodeArtifact 存储库中删除上游存储库后，程序包管理器将无法访问上游存储库中的程序包或其任何上游存储库。

6. 选择更新存储库。

添加或删除上游存储库 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 添加或删除 CodeArtifact 存储库的上游存储库。为此，请使用 `update-repository` 命令并使用 `--upstreams` 参数指定上游存储库。

您只能在与上游存储库相同的域中添加存储库。

添加上游存储库 (AWS CLI)

1. 如果还没有设置和配置，请按照设置 AWS CodeArtifact 中的步骤设置和配置 AWS CLI 与 CodeArtifact。
2. 使用带有 `--upstreams` 标记的 `aws codeartifact update-repository` 命令来添加上游存储库。

   > **Note**
   >
   > 调用 `update-repository` 命令会将现有已配置的上游存储库替换为使用 `--upstreams` 标记提供的存储库列表。如果要添加上游存储库并保留现有存储库，则必须在调用中包括现有的上游存储库。

   以下示例命令将两个上游存储库添加到名为 `my_repo` 的存储库，该存储库位于名为 `my_domain` 的域中。当 CodeArtifact 从 `my_repo` 存储库请求程序包时，`--upstreams` 参数中指定的上游存储库顺序决定了它们的搜索优先顺序。有关更多信息，请参阅上游存储库优先顺序。

   ```
   aws codeartifact update-repository --repository my_repo --domain my_domain --
   domain-owner 111122223333 \
    --upstreams repositoryName=upstream-1 repositoryName=upstream-2
   ```

   输出包含上游存储库，如下所示。

```
{
       "repository": {
           "name": "my_repo",
           "administratorAccount": "123456789012",
           "domainName": "my_domain",
           "domainOwner": "111122223333",
           "arn": "arn:aws:codeartifact:us-
east-2:111122223333:repository/my_domain/my_repo",
           "upstreams": [
               {
                   "repositoryName": "upstream-1"
               },
               {
                   "repositoryName": "upstream-2"
               }
           ],
           "externalConnections": []
       }
   }
```

删除上游存储库 (AWS CLI)

1. 如果还没有设置和配置，请按照设置 AWS CodeArtifact 中的步骤设置和配置 AWS CLI 与 CodeArtifact。
2. 要从 CodeArtifact 存储库中删除上游存储库，请使用带有 `update-repository` 标记的 `--upstreams` 命令。提供给该命令的存储库列表将是 CodeArtifact 存储库的新上游存储库集。包括您要保留的现有上游存储库，并省略要删除的上游存储库。

   要从存储库中删除所有上游存储库，请使用 `update-repository` 命令并包括不带参数的 `--upstreams` 选项。以下示例命令从名为 `my_domain` 的域中包含的名为 `my_repo` 的存储库中删除上游存储库。

   ```
   aws codeartifact update-repository --repository my_repo --domain my_domain --
   domain-owner 111122223333 --upstreams
   ```

   输出显示 `upstreams` 的列表为空。

   ```
   {
          "repository": {
   ```

```
            "name": "my_repo",
            "administratorAccount": "123456789012",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:us-
east-2:111122223333:repository/my_domain/my_repo",
            "upstreams": [],
            "externalConnections": []
        }
    }
```

将 CodeArtifact 存储库连接到公有存储库

您可以在 CodeArtifact 存储库和外部公有存储库（例如 https://npmjs.com 或 Maven Central 存储库）之间添加外部连接。然后，当您从 CodeArtifact 存储库中请求一个在该存储库中尚不存在的程序包时，可以从外部连接提取该程序包。这样就可以使用由应用程序所用的开源依赖项。

在 CodeArtifact 中，使用外部连接的预期方法是为每个域设置一个存储库，并与给定的公有存储库建立外部连接。例如，如果要连接到 npmjs.com，请将域中的一个存储库配置为与 npmjs.com 建立外部连接，并将所有其他存储库配置为该存储库的上游存储库。这样，所有存储库都可以使用已经从 npmjs.com 提取的程序包，而不必再次提取和存储程序包。

主题

- 连接到外部存储库（控制台）
- 连接到外部存储库 (CLI)
- 支持的外部连接存储库
- 移除外部连接 (CLI)

连接到外部存储库（控制台）

使用控制台向外部存储库添加连接时，会发生以下情况：

1. 如果 `-store` 存储库尚不存在，则将在您的 CodeArtifact 域中为外部存储库创建一个这样的存储库。这些 `-store` 存储库充当您的存储库和外部存储库之间的中间存储库，让您可以连接到多个外部存储库。
2. 相应的 `-store` 存储库将作为上游存储库添加到您的存储库中。

以下列表包含 CodeArtifact 中的每个 `-store` 存储库以及它们所连接的相应外部存储库。

1. `commonsware-store` 连接到 CommonsWare Android 存储库。
2. `google-android-store` 连接到 Google Android。
3. `gradle-plugins-store` 连接到 Gradle 插件。
4. `maven-central-store` 连接到 Maven Central 存储库。
5. `clojars-store` 连接到 Clojars 存储库。
6. `npm-store` 连接到 npmjs.com。
7. `nuget-store` 连接到 nuget.org。
8. `pypi-store` 连接到 Python 打包权威机构。

连接到外部存储库（控制台）

1. 打开 AWS CodeArtifact 控制台，网址为：https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择域，然后选择包含您的存储库的域名。
3. 选择您的存储库的名称。
4. 选择编辑。
5. 在上游存储库中，选择关联上游存储库，然后添加作为上游存储库连接的相应 `-store` 存储库。
6. 选择更新存储库。

将 `-store` 存储库作为上游存储库添加之后，连接到您的 CodeArtifact 存储库的程序包管理器可以从相应的外部存储库提取程序包。

连接到外部存储库 (CLI)

您可以使用 AWS CLI 将外部连接直接添加到存储库，从而将 CodeArtifact 存储库连接到外部存储库。这样就可以让连接到 CodeArtifact 存储库或其任何下游存储库的用户从已配置的外部存储库提取程序包。每个 CodeArtifact 存储库只能有一个外部连接。

建议每个域都有一个存储库与给定的公有存储库建立外部连接。要将其他存储库连接到公有存储库，请将具有外部连接的存储库作为其上游添加进来。如果您或域中的其他人已经在控制台中配置了外部连接，则您的域可能已经有一个 `-store` 存储库，该存储库与您要连接的公有存储库建立了外部连接。有关 `-store` 存储库以及与控制台连接的更多信息，请参阅连接到外部存储库（控制台）。

向 CodeArtifact 存储库添加外部连接 (CLI)

- 使用 `associate-external-connection` 来添加外部连接。以下示例将存储库连接到 npm 公有注册表 npmjs.com。有关支持的外部存储库的列表，请参阅支持的外部连接存储库。

```
aws codeartifact associate-external-connection --external-connection public:npmjs \
    --domain my_domain --domain-owner 111122223333 --repository my_repo
```

输出示例：

```
{
    "repository": {
        "name": my_repo
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo",
        "description": "A description of my_repo",
        "upstreams": [],
        "externalConnections": [
            {
                "externalConnectionName": "public:npmjs",
                "packageFormat": "npm",
                "status": "AVAILABLE"
            }
        ]
    }
}
```

添加外部连接后，有关通过外部连接从外部存储库请求程序包的信息，请参阅从外部连接请求程序包。

支持的外部连接存储库

CodeArtifact 支持与以下公有存储库建立外部连接。要使用 CodeArtifact CLI 来指定外部连接，请在运行 `associate-external-connection` 命令时为 `--external-connection` 参数使用名称列中的值。

存储库类型	描述	名称
npm	npm 公有注册表	`public:npmjs`
Python	Python 包索引	`public:pypi`
Maven	Maven Central	`public:maven-central`
Maven	Google Android 存储库	`public:maven-googleandroid`
Maven	Gradle 插件存储库	`public:maven-gradleplugins`
Maven	CommonsWare Android 存储库	`public:maven-commonsware`
Maven	Clojars 存储库	`public:maven-clojars`
NuGet	NuGet 库	`public:nuget-org`

移除外部连接 (CLI)

要移除在 AWS CLI 中使用 `associate-external-connection` 命令添加的外部连接，请使用 `disassociate-external-connection`。

```
aws codeartifact disassociate-external-connection --external-connection public:npmjs \
    --domain my_domain --domain-owner 111122223333 --repository my_repo
```

输出示例：

```
{
    "repository": {
        "name": my_repo
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
```

```
        "arn": "arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo",
        "description": "A description of my_repo",
        "upstreams": [],
        "externalConnections": []
    }
}
```

请求包含上游存储库的程序包版本

当客户端（例如 npm）从包含多个上游存储库且名为 my_repo 的 CodeArtifact 存储库请求程序包版本时，会发生以下情况：

- 如果 my_repo 包含请求的程序包版本，则将该版本返回给客户端。
- 如果 my_repo 不包含请求的程序包版本，则 CodeArtifact 会在 my_repo 的上游存储库中查找该版本。如果找到了程序包版本，则会将对该版本的引用复制到 my_repo，并将程序包版本返回给客户端。
- 如果 my_repo 和其上游存储库都不包含程序包版本，则会向客户端返回 HTTP 404 Not Found 响应。

使用 create-repository 或 update-repository 命令添加上游存储库时，它们传递给 --upstreams 参数的顺序决定了在请求程序包版本时它们的优先级。按照在请求程序包版本时您希望 CodeArtifact 使用的顺序，使用 --upstreams 指定上游存储库。有关更多信息，请参阅上游存储库优先顺序。

一个存储库允许的直接上游存储库的最大数量为 10。由于直接上游存储库也可以有自己的直接上游存储库，因此 CodeArtifact 会在 10 个以上的存储库中搜索程序包版本。当请求程序包版本时，CodeArtifact 可以查找的最大存储库数量为 25。

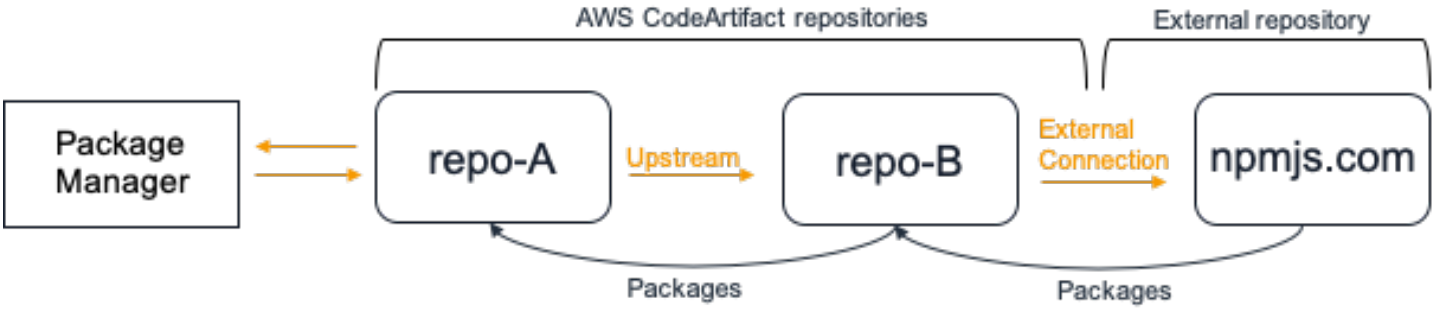
上游存储库的程序包保留

如果在上游存储库中找到了请求的程序包版本，则会保留对该版本的引用，并且始终可以从下游存储库中使用。保留的程序包版本不受以下任何因素的影响：

- 删除上游存储库。
- 断开上游存储库与下游存储库的连接。
- 从上游存储库中删除程序包版本。
- 在上游存储库中编辑程序包版本（例如，向其中添加新资产）。

通过上游关系提取程序包

如果 CodeArtifact 存储库与具有外部连接的存储库存在上游关系，则在上游存储库中找不到请求的程序包时，会从外部存储库复制该程序包。例如，考虑以下配置：名为 repo-A 的存储库有一个名为 repo-B 的上游存储库。repo-B 与 https://npmjs.com 建立了外部连接。



如果将 npm 配置为使用 repo-A 存储库，则运行 npm install 会触发将程序包从 https://npmjs.com 复制到 repo-B。已安装的版本也会提取到 repo-A。下面的示例会安装 lodash。

```
$ npm config get registry
https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my-
downstream-repo/
$ npm install lodash
+ lodash@4.17.20
added 1 package from 2 contributors in 6.933s
```

运行 npm install 后，repo-A 仅包含最新版本 (lodash 4.17.20)，因为这是 npm 从 repo-A 提取的版本。

```
aws codeartifact list-package-versions --repository repo-A --domain my_domain \
            --domain-owner 111122223333 --format npm --package lodash
```

输出示例：

```
{
    "package": "lodash",
    "format": "npm",
    "versions": [
        {
            "version": "4.17.15",
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
```

```
    ]
}
```

由于 repo-B 与 https://npmjs.com 建立了外部连接，所以从 https://npmjs.com 导入的所有程序包版本都存储在 repo-B 中。任何与 repo-B 建立了上游关系的下游存储库可能已经提取这些程序包版本。

repo-B 中记录了随着时间推移从 https://npmjs.com 导入的所有程序包及程序包版本。例如，要查看一段时间内导入的 lodash 程序包的所有版本，可以使用 list-package-versions，如下所示。

```
aws codeartifact list-package-versions --repository repo-B --domain my_domain \
            --domain-owner 111122223333 --format npm --package lodash --max-results 5
```

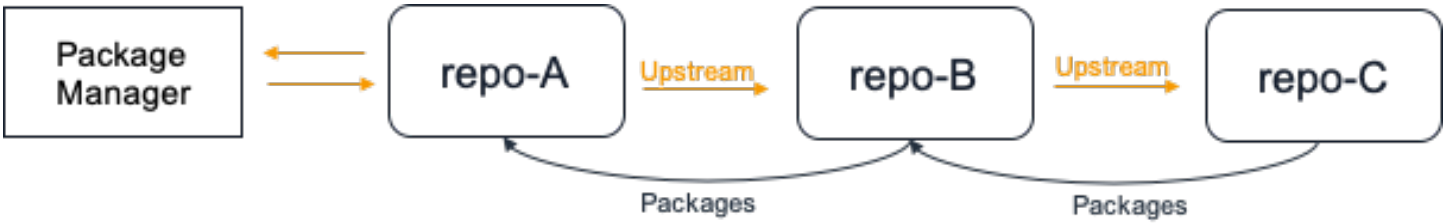
输出示例：

```
{
    "package": "lodash",
    "format": "npm",
    "versions": [
        {
            "version": "0.10.0",
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.2",
            "revision": "REVISION-2-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.0",
            "revision": "REVISION-3-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.1",
            "revision": "REVISION-4-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.1.0",
            "revision": "REVISION-5-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
```

```
    ],
    "nextToken": "eyJsaXN0UGFja2FnZVZlcnNpb25zVG9rZW4iOiIwLjIuMiJ9"
}
```

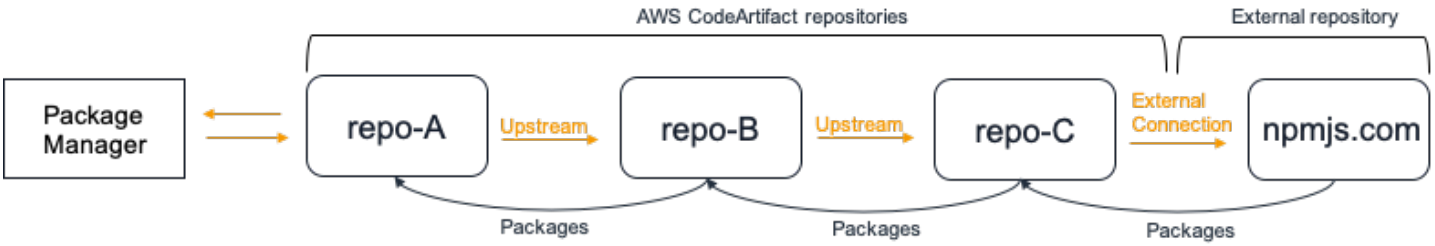
中间存储库中的程序包保留

CodeArtifact 允许链接上游存储库。例如，repo-A 可以使用 repo-B 作为上游，而 repo-B 也可以使用 repo-C 作为上游。这个配置意味着在 repo-A 中可以获取 repo-B 和 repo-C 中的程序包版本。



当程序包管理器连接到存储库 repo-A 并从存储库 repo-C 中提取程序包版本时，程序包版本将不会保留在存储库 repo-B 中。程序包版本将仅保留在最下游的存储库中，在此示例中为 repo-A。程序包版本不会保留在任何中间存储库中。对于较长的链也是如此；例如，如果有四个存储库 repo-A、repo-B、repo-C 和 repo-D，一个连接到 repo-A 的程序包管理器从 repo-D 提取程序包版本，则程序包版本将保留在 repo-A 中，但不会保留在 repo-B 或 repo-C 中。

从外部存储库提取程序包版本时，程序包保留行为与之类似，不同之处在于程序包版本始终保留在连接了外部连接的存储库中。例如，repo-A 使用 repo-B 作为上游。repo-B 使用 repo-C 作为上游，并且 repo-C 还配置了 npmjs.com 作为外部连接；请参见下图。



如果连接到 repo-A 的程序包管理器请求程序包版本（例如，lodash 4.17.20），而三个存储库中的任何一个都不存在该程序包版本，则将从 npmjs.com 提取该程序包版本。当提取 lodash 4.17.20 时，该版本将保留在 repo-A（因为这是最下游的存储库）和 repo-C（因为它与 npmjs.com 建立了外部连接）中。lodash 4.17.20 不会保留在 repo-B 中（因为这是中间存储库）。

从外部连接请求程序包

以下各节介绍如何通过外部连接请求程序包以及请求程序包时的预期 CodeArtifact 行为。

主题

从外部连接提取程序包

如将 CodeArtifact 存储库连接到公有存储库中所述，将程序包添加到 CodeArtifact 存储库后，要从外部连接提取程序包，请将程序包管理器配置为使用您的存储库并安装这些程序包。

> **Note**
>
> 以下说明使用 npm，要查看其他程序包类型的配置和使用说明，请参阅将 CodeArtifact 与 Maven 结合使用、将 CodeArtifact 与 NuGet 结合使用或将 CodeArtifact 与 Python 结合使用。

从外部连接提取程序包

1. 使用您的 CodeArtifact 存储库配置程序包管理器并对其进行身份验证。对于 npm，使用以下 `aws codeartifact login` 命令。

   ```
   aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333 --repository my_repo
   ```

2. 从公有存储库请求程序包。对于 npm，请使用以下 `npm install` 命令，将 *`lodash`* 替换为要安装的程序包。

   ```
   npm install lodash
   ```

3. 将程序包复制到您的 CodeArtifact 存储库后，您可以使用 `list-packages` 和 `list-package-versions` 命令来查看程序包。

   ```
   aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --repository my_repo
   ```

   输出示例：

```
{
    "packages": [
        {
            "format": "npm",
            "package": "lodash"
        }
    ]
}
```

`list-package-versions` 命令列出了已复制到您的 CodeArtifact 存储库中的程序包的所有版本。

```
aws codeartifact list-package-versions --domain my_domain --domain-
owner 111122223333 --repository my_repo --format npm --package lodash
```

输出示例：

```
{
    "defaultDisplayVersion: "1.2.5"
    "format": "npm",
    "package": "lodash",
    "namespace": null,
    "versions": [
        {
            "version": "1.2.5",
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    ]
}
```

外部连接延迟

使用外部连接从公有存储库提取程序包时，从公有存储库提取软件包到将其存储在 CodeArtifact 存储库中存在一定的延迟。例如，假设您已经安装了 npm 程序包“lodash”的版本 1.2.5，如从外部连接提取程序包中所述。尽管 `npm install lodash` lodash 命令成功完成，但程序包版本可能尚未出现在您的 CodeArtifact 存储库中。程序包版本通常需要大约 3 分钟才会出现在您的存储库中，但偶尔可能需要更长的时间。

由于存在这种延迟，您可能已经成功取回了程序包版本，但在 CodeArtifact 控制台的存储库中或调用 ListPackages 和 ListPackageVersions API 操作时可能还无法看到该版本。CodeArtifact 异步保存了程序包版本之后，即可在控制台中以及通过 API 请求来显示该版本。

外部存储库不可用时的 CodeArtifact 行为

有时，外部存储库会出现中断，这意味着 CodeArtifact 无法从存储库中提取程序包，或者提取程序包的速度比平时慢得多。发生这种情况时，已经从外部存储库（例如 npmjs.com）提取并存储在 CodeArtifact 存储库中的程序包版本将继续可供从 CodeArtifact 下载。但是，即使配置了与该存储库的外部连接，尚未存储在 CodeArtifact 中的程序包也可能不可用。例如，您的 CodeArtifact 存储库可能包含 npm 程序包版本 `lodash 4.17.19` ，因为这是您迄今为止在应用程序中使用的版本。当您想升级到 `4.17.20` 时，CodeArtifact 通常会从 npmjs.com 提取新版本并将其存储在您的 CodeArtifact 存储库中。但是，如果 npmjs.com 发生中断，则这个新版本将不可用。唯一的解决方法是等 npmjs.com 恢复后再试。

外部存储库中断也会影响将新程序包版本发布到 CodeArtifact。在配置了外部连接的存储库中，CodeArtifact 不允许发布外部存储库中已存在的程序包版本。有关更多信息，请参阅程序包概览。但在极少数情况下，外部存储库中断可能意味着 CodeArtifact 不了解有关外部存储库中存在哪些程序包和程序包版本的最新信息。在这种情况下，CodeArtifact 可能允许发布它通常会拒绝的程序包版本。

新程序包版本的可用性

要使公有存储库（例如 npmjs.com）中的程序包版本可通过 CodeArtifact 存储库来提供，必须先将其添加到区域程序包元数据缓存中。此缓存由 CodeArtifact 在每个 AWS 区域中维护，并包含描述受支持公有存储库内容的元数据。由于存在这个缓存，在将新程序包版本发布到公有存储库和从 CodeArtifact 提供新程序包版本之间会有延迟。延迟时间因程序包类型而异。

对于 npm、Python 和 Nuget 程序包，从将新程序包版本发布到 npmjs.com、pypi.org 或 nuget.org 到可以从 CodeArtifact 存储库进行安装为止，最长可能有 30 分钟的延迟。CodeArtifact 会自动同步来自这两个存储库的元数据，从而确保缓存是最新的。

对于 Maven 程序包，从将新程序包版本发布到公有存储库到可以从 CodeArtifact 存储库进行安装为止，最长可能有 3 小时的延迟。CodeArtifact 至多每 3 小时检查一次程序包的新版本。在 3 小时缓存寿命到期后首次请求给定程序包名称会导致该程序包的所有新版本都导入到区域缓存中。

对于常用的 Maven 程序包，通常每 3 小时导入一次新版本，因为高请求率意味着缓存通常会在缓存寿命到期后立即更新。对于不常使用的程序包，在从 CodeArtifact 存储库请求程序包版本之前，缓存将不会有最新版本。在第一个请求中，CodeArtifact 只能提供之前导入的版本，但此请求会导致缓存更新。在随后的请求中，程序包的新版本将添加到缓存中并可供下载。

导入包含多个资产的程序包版本

Maven 和 Python 程序包的每个程序包版本中都可以有多个资产。因为 npm 和 NuGet 程序包的每个版本只有一个资产，这使得导入这些格式的程序包比导入 npm 和 NuGet 程序包更加复杂。有关为这些程序包类型导入哪些资产以及如何提供新添加资产的说明，请参阅从上游和外部连接请求 Python 程序包和从上游和外部连接请求 Maven 程序包。

上游存储库优先顺序

当您从包含一个或多个上游存储库的存储库请求程序包版本时，它们的优先级与调用 `create-repository` 或 `update-repository` 命令时列出的顺序相对应。找到请求的程序包版本后，即使该搜索并未搜索所有上游存储库，搜索也会停止。有关更多信息，请参阅添加或删除上游存储库 (AWS CLI)。

使用 `describe-repository` 命令来查看优先顺序。

```
aws codeartifact describe-repository --repository my_repo --domain my_domain --domain-
owner 111122223333
```

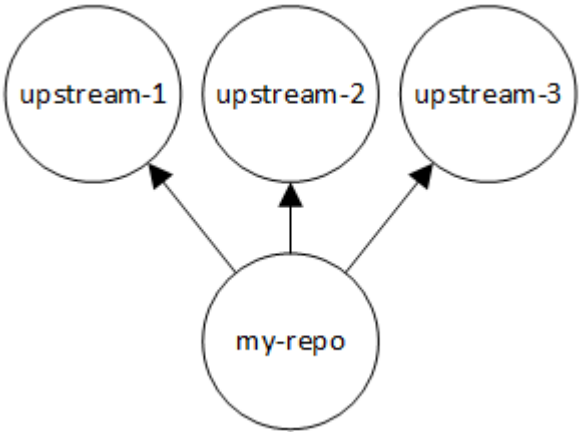
结果可能如下所示。结果显示，上游存储库的优先级是 `upstream-1` 第一、`upstream-2` 第二，`upstream-3` 第三。

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-
east-1:111122223333:repository/my_domain/my_repo",
        "description": "My new repository",
        "upstreams": [
            {
                "repositoryName": "upstream-1"
            },
            {
                "repositoryName": "upstream-2"
            },
            {
                "repositoryName": "upstream-3"
            }
```

```
        ],
        "externalConnections": []
    }
}
```

简单的优先顺序示例

在下图中，my_repo 存储库有三个上游存储库。上游存储库的优先顺序为 upstream-1、upstream-2、upstream-3。



对 my_repo 中的程序包版本发出的请求会按以下顺序搜索存储库，直到找到该版本，或直至向客户端返回 HTTP 404 Not Found 响应：

1. my_repo
2. upstream-1
3. upstream-2
4. upstream-3

如果找到了程序包版本，则即使搜索并未在所有上游存储库中查找，搜索也会停止。例如，如果在 upstream-1 中找到了程序包版本，则搜索会停止，且 CodeArtifact 不会在 upstream-2 或 upstream-3 中查找。

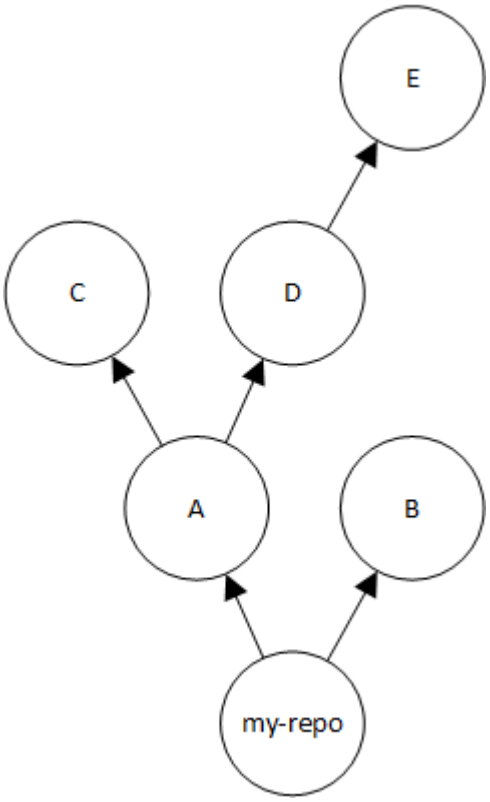
当您使用 AWS CLI 命令 list-package-versions 列出 my_repo 中的程序包版本时，该命令只会在 my_repo 中查找。该命令不会列出上游存储库中的程序包版本。

复杂优先顺序示例

如果上游存储库有自己的上游存储库，则在移动到下一个上游存储库之前，将使用相同的逻辑来查找程序包版本。例如，假设您的 my_repo 存储库有两个上游存储库，A 和 B。如果存储库 A 有上游存储

库，则对 my_repo 中的程序包版本的请求会首先在 my_repo 中查找，其次在 A 中查找，然后在 A 的上游存储库中查找，依此类推。

在下图中，my_repo 存储库包含上游存储库。上游存储库 A 有两个上游存储库，D 有一个上游存储库。图中同一级别的上游存储库按其优先顺序从左到右显示（存储库 A 的优先顺序高于存储库 B，存储库 C 的优先级顺序高于存储库 D）。



在此示例中，对 my_repo 中的程序包版本发出的请求会按以下顺序在存储库中查找，直到找到该版本，或者直至程序包管理器向客户端返回 HTTP 404 Not Found 响应：

1. my_repo
2. A
3. C
4. D
5. E
6. B

上游存储库的 API 行为

当您对连接到上游存储库的存储库调用某些 CodeArtifact API 时，行为可能会有所不同，具体取决于程序包或程序包版本是存储在目标仓库还是上游存储库中。此处记录了这些 API 的行为。

有关 CodeArtifact API 的更多信息，请参阅 CodeArtifact API 参考。

如果目标存储库中不存在指定的程序包版本，则大多数引用该程序包或程序包版本的 API 都会返回 `ResourceNotFound` 错误。即使在上游存储库中存在该程序包或程序包版本，也会返回错误。实际上，在调用这些 API 时会忽略上游存储库。这些 API 是：

- DeletePackageVersions
- DescribePackageVersion
- GetPackageVersionAsset
- GetPackageVersionReadme
- ListPackages
- ListPackageVersionAssets
- ListPackageVersionDependencies
- ListPackageVersions
- UpdatePackageVersionsStatus

为了演示这种行为，我们使用两个存储库：`target-repo` 和 `upstream-repo`。`target-repo` 为空存储库且已配置 `upstream-repo` 作为上游存储库。`upstream-repo` 包含 npm 程序包 `lodash`。

当对 `upstream-repo`（包含 `lodash` 程序包）调用 `DescribePackageVersion` API 时，我们会得到以下输出：

```
{
    "packageVersion": {
        "format": "npm",
        "packageName": "lodash",
        "displayName": "lodash",
        "version": "4.17.20",
        "summary": "Lodash modular utilities.",
        "homePage": "https://lodash.com/",
        "sourceCodeRepository": "https://github.com/lodash/lodash.git",
        "publishedTime": "2020-10-14T11:06:10.370000-04:00",
        "licenses": [
```

```
            {
                "name": "MIT"
            }
        ],
        "revision": "Ciqe5/9yicvkJT13b5/LdLpCyE6fqA7poa9qp+FilPs=",
        "status": "Published"
    }
```

在对 target-repo（该存储库为空但已将 upstream-repo 配置为上游存储库）调用相同的 API 时，我们会得到以下输出：

```
An error occurred (ResourceNotFoundException) when calling the DescribePackageVersion
 operation:
Package not found in repository. RepoId: repo-id, Package =
 PackageCoordinate{packageType=npm, packageName=lodash},
```

CopyPackageVersions API 的行为有所不同。默认情况下，CopyPackageVersions API 仅复制存储在目标存储库中的程序包版本。如果程序包版本存储在上游存储库中，但未存储在目标存储库中，则不会复制该版本。要包括仅存储在上游存储库中的程序包的程序包版本，请在 API 请求中将 includeFromUpstream 的值设置为 true。

有关 CopyPackageVersions API 的更多信息，请参阅在存储库之间复制程序包。

处理中的软件包 CodeArtifact

这些主题向您展示如何使用 CLI 和 AP CodeArtifact I 列出、复制、删除和搜索软件包。

主题

程序包概览

程序包 是解析依赖关系和安装软件所需的软件和元数据的捆绑包。在中 CodeArtifact，软件包由软件包名称、可选命名空间（如`@types`在）`@types/node`、一组软件包版本以及包级元数据（例如 npm 标签）组成。

目录

- 程序包名称、程序包版本和资产名称规范化

支持的软件包格式

AWS CodeArtifact 支持 npm、pyP I 、 Maven 和NuGet通用包格式。

程序包发布

您可以使用诸如npm、、、、twine和之类的工具将任何支持的软件包格式的新版本发布到 CodeArtifact 存储库dotnet。Maven Gradle nuget

发布权限

您的 AWS Identity and Access Management (IAM) 用户或角色必须具有发布到目标存储库的权限。发布程序包需要以下权限：

- Maven：codeartifact:PublishPackageVersion 和 codeartifact:PutPackageMetadata
- npm：codeartifact:PublishPackageVersion
- NuGet: codeartifact:PublishPackageVersion 和 codeartifact:ReadFromRepository
- Python：codeartifact:PublishPackageVersion
- 通用：codeartifact:PublishPackageVersion

在前面的权限列表中，IAM 策略必须为 codeartifact:PublishPackageVersion 和 codeartifact:PutPackageMetadata 权限指定 package 资源。还必须指定 codeartifact:ReadFromRepository 权限的 repository 资源。

有关中权限的更多信息 CodeArtifact，请参阅AWS CodeArtifact 权限参考。

覆盖程序包资产

您无法重新发布已存在且包含不同内容的程序包资产。例如，假定您已经发布了一个具有 JAR 资产 mypackage-1.0.jar 的 Maven 程序包。仅当新旧资产的校验和完全相同时，您才能再次发布该资产。要重新发布包含新内容的相同资产，请先使用 delete-package-versions 命令删除该程序包版本。尝试重新发布具有不同内容但名称相同的资产会导致出现 HTTP 409 冲突错误。

对于支持多种资产（通用、PyPI 和 Maven）的程序包格式，您可以向现有程序包版本添加使用不同名称的新资产，前提是您拥有所需的权限。对于通用程序包，只要程序包版本处于 Unfinished 状态，

就可以添加新资产。由于 npm 的每个程序包版本仅支持单个资产，因此若要以任何方式修改已发布的程序包版本，都必须先使用 delete-package-versions 删除版本。

如果您尝试重新发布已存在的资产（例如 `mypackage-1.0.jar`），并且已发布资产和新资产的内容相同，则因为该操作具有幂等性，所以操作会成功。

私有程序包和公有存储库

CodeArtifact 不会将存储在存储 CodeArtifact 库中的软件包发布到公共存储库，例如 npmjs.com 或 Maven Central。 CodeArtifact 将软件包从公共存储库导入 CodeArtifact 存储库，但它从不将软件包向另一个方向移动。您发布到 CodeArtifact 存储库的软件包将保持私密状态，并且仅供您授予访问权限的 AWS 账户、角色和用户使用。

发布经过修补的程序包版本

有时，您可能想要发布修改后的程序包版本，可能是在公有存储库中可用的版本。例如，您可能在名为 `mydep 1.1` 的关键应用程序依赖项中发现了一个错误，您需要在程序包供应商审查和接受更改之前尽快修复该错误。如前所述，如果可以通过上游 CodeArtifact存储库和外部连接从您的存储库访问公共 CodeArtifact 存储库，则 CodeArtifact 禁止您在存储库中发布`mydep 1.1`。

要解决此问题，请将软件包版本发布到无法访问公共存储库的其他存储库。 CodeArtifact 然后使用 `copy-package-versions` API 将的修补版本复制到您`mydep 1.1`要从中使用该版本的 CodeArtifact 存储库中。

发布的资产大小限制

可以发布的程序包资产的最大大小受资产文件大小最大配额的限制，如AWS CodeArtifact 中的配额中所示。例如，您发布的 Maven JAR 或 Python Wheel 不能超过当前资产文件大小的最大配额。如果您需要在中存储更大的资产 CodeArtifact，请申请增加配额。

除了资产文件大小的最大配额外，npm 程序包发布请求的最大大小为 2 GB。此限制与资产文件大小的最大配额无关，不能随着配额的增加而提高。在 npm 发布请求 (HTTP PUT) 中，程序包元数据和 npm 程序包 tar 存档的内容捆绑在一起。因此，可以发布的 npm 程序包的实际最大大小会有所不同，具体取决于所包含元数据的大小。

> **Note**
>
> 已发布的 npm 包的最大大小限制为小于 2 GB。

发布延迟

发布到 CodeArtifact 存储库的 Package 版本通常可以在不到一秒钟的时间内下载。例如，如果您将 npm 包版本发布到 wit CodeArtifact hnpm publish，则该版本应在不到一秒钟的时间内可供npm install命令使用。但是，发布可能不一致，有时可能需要更长的时间。如果您必须在发布后立即使用程序包版本，请使用重试来确保下载可靠。例如，发布程序包版本后，如果第一次尝试下载时刚刚发布的程序包版本最初不可用，请重复下载最多三次。

> **Note**
>
> 从公有存储库导入程序包版本通常要花比发布更长的时间。有关更多信息，请参阅 外部连接延迟。

程序包版本状态

中的每个软件包版本都 CodeArtifact 有一个描述软件包版本的当前状态和可用性的状态。您可以使用 AWS CLI 和 SDK 来更改程序包版本状态。有关更多信息，请参阅 更新程序包版本状态。

程序包版本状态的可能值如下所示：

- 已发布 - 已成功发布程序包版本，可以使用程序包管理器来请求版本。在返回给程序包管理器的程序包版本列表中将包括该程序包版本，例如，在 npm view <package-name> versions 的输出中。程序包版本的所有资产均可从存储库中获得。
- 未完成 - 客户端已上传程序包版本的一个或多个资产，但尚未通过将其变为 Published 状态来最终完成上传。当前，只有通用程序包版本和 Maven 程序包版本可以处于 Unfinished 状态。对于 Maven 程序包，当客户端上传程序包版本的一个或多个资源但没有为包括该版本的程序包发布 maven-metadata.xml 文件时，就会发生这种情况。当 Maven 程序包版本是未完成时，向客户端（例如 mvn 或 gradle）返回的版本列表中不会包括该版本，因此在构建时不能使用该版本。通过在调用 PublishPackageVersionAPI 时提供unfinished标志，可以故意将通用包保持在Unfinished状态。可以通过省略unfinished标志或调用 UpdatePackageVersionsStatusAPI 将通用包更改为Published状态。
- 未列出 - 可以从存储库下载程序包版本的资产，但向程序包管理器返回的版本列表中未包括该程序包版本。例如，对于 npm 程序包，npm view <package-name> versions 的输出将不包括该程序包版本。因为在可用版本列表中未显示该版本，这意味着 npm 的依赖项解析逻辑不会选择该程序包版本。但是，如果 npm package-lock.json 文件中已经引用了未列出的程序包版本，则仍然可以下载和安装该版本，例如在运行 npm ci 时。

- 已存档 - 无法再下载该程序包版本的资产。在返回给程序包管理器的版本列表中不会包括该程序包版本。由于资产不可用，因此会阻止客户端使用程序包版本。如果您的应用程序构建依赖于更新为已存档的版本，那么构建就会出问题，前提是该程序包版本尚未在本地缓存。您不能使用包管理器或构建工具重新发布存档包版本，因为该版本仍存在于存储库中，但您可以使用 API 将包版本的状态更改回 “未上市” 或 “UpdatePackageVersionsStatus 已发布”。
- 已处置 - 程序包版本未出现在列表中，也无法从存储库下载资产。它们在已处置和已存档之间的主要区别在于，如果状态为 “已处置”，则软件包版本的资产将被永久删除 CodeArtifact。因此，您无法将程序包版本从已处置更改为已存档、未列出或已发布。由于已删除资产，因此无法再使用该程序包版本。将程序包版本标记为已处置后，我们将不再向您收取程序包资产的存储费用。

不 list-package-versions 带`--status`参数调用时，将默认返回所有状态的 Package 版本。

除了前面列出的状态外，还可以使用 DeletePackageVersionsAPI 删除软件包版本。删除程序包版本后，存储库中将不再存在该版本，您可以使用程序包管理器或构建工具随意地重新发布该程序包版本。删除程序包版本后，我们将不再向您收取程序包版本的资产的存储费用。

程序包名称、程序包版本和资产名称规范化

CodeArtifact 在存储软件包名称、软件包版本和资源名称之前对其进行标准化，这意味着中的名称或版本 CodeArtifact 可能与发布软件包时提供的名称或版本不同。有关如何标准化每种软件包类型的名称和版本 CodeArtifact 的更多信息，请参阅以下文档：

- Python 程序包名称规范化
- NuGet 程序包名称、版本和资产名称规范化

CodeArtifact 不对其他包格式执行标准化。

列出程序包名称

使用中的`list-packages`命令 CodeArtifact 获取存储库中所有软件包名称的列表。此命令仅返回程序包名称，不返回版本。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

示例输出：

```
{
```

```
    "nextToken": "eyJidWNrZXRJZCI6I...",
    "packages": [
        {
            "package": "acorn",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "acorn-dynamic-import",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "ajv",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "ajv-keywords",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "anymatch",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
```

```
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "ast",
            "namespace": "webassemblyjs",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        }
    ]
}
```

列出 npm 程序包名称

要仅列出 npm 程序包的名称，请将 `--format` 选项的值设置为 `npm`。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format npm
```

要列出命名空间 (npm scope) 中的 npm 程序包，请使用 `--namespace` 和 `--format` 选项。

> **Important**
>
> `--namespace` 选项的值不得包括前导 `@`。要搜索命名空间 `@types`，请将值设置为 *types*。

> **Note**
>
> 按命名空间前缀来筛选 `--namespace` 选项。作用域从传递给 `--namespace` 选项的值开始的任何 npm 包都将在 `list-packages` 响应中返回。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format npm --namespace types
```

示例输出：

```
{
    "nextToken": "eyJidWNrZXRJZ...",
    "packages": [
        {
            "package": "3d-bin-packing",
            "namespace": "types",
            "format": "npm"

        },
        {
            "package": "a-big-triangle",
            "namespace": "types",
            "format": "npm"

        },
        {
            "package": "a11y-dialog",
            "namespace": "types",
            "format": "npm"

        }
    ]
}
```

列出 Maven 程序包名称

若要仅列出 Maven 程序包的名称，请将 `--format` 选项的值设置为 `maven`。您还必须在 `--namespace` 选项中指定 Maven 组 ID。

> **Note**
>
> 按命名空间前缀来筛选 `--namespace` 选项。作用域从传递给 `--namespace` 选项的值开始的任何 npm 包都将在 `list-packages` 响应中返回。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format maven --namespace org.apache.commons
```

示例输出：

```
{
    "nextToken": "eyJidWNrZXRJZ...",
    "packages": [
        {
            "package": "commons-lang3",
            "namespace": "org.apache.commons",
            "format": "maven"

        },
        {
            "package": "commons-collections4",
            "namespace": "org.apache.commons",
            "format": "maven"

        },
        {
            "package": "commons-compress",
            "namespace": "org.apache.commons",
            "format": "maven"

        }
    ]
}
```

列出 Python 程序包名称

要仅列出 Python 程序包的名称，请将 `--format` 选项的值设置为 `pypi`。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format pypi
```

按程序包名称前缀筛选

要返回以指定字符串开头的程序包，可以使用 `--package-prefix` 选项。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format npm --package-prefix pat
```

示例输出：

```
{
    "nextToken": "eyJidWNrZXRJZ...",
    "packages": [
        {
            "package": "path",
            "format": "npm"

        },
        {
            "package": "pat-test",
            "format": "npm"

        },
        {
            "package": "patch-math3",
            "format": "npm"

        }
    ]
}
```

支持的搜索选项组合

您可以按照任意组合来使用 `--format`、`--namespace` 和 `--package-prefix` 选项，但不能单独使用 `--namespace`。搜索作用域从 `@types` 开始的所有 npm 程序包需要指定 `--format` 选项。单独使用 `--namespace` 会引发错误。

`list-packages` 不支持使用这三个选项中的任何一个，并将返回存储库中的所有格式的所有程序包。

格式输出

您可以使用所有 AWS CLI 命令都可用的参数来使`list-packages`响应更紧凑且更具可读性。使用 `--query` 参数来指定每个返回的程序包版本的格式。使用 `--output` 参数将响应格式化为纯文本。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo \
  --output text --query 'packages[*].[package]'
```

示例输出：

```
accepts
array-flatten
body-parser
bytes
content-disposition
content-type
cookie
cookie-signature
```

有关更多信息，请参阅 AWS Command Line Interface 用户指南中的从 AWS CLI控制命令输出。

默认值和其他选项

默认情况下，list-packages 返回的最大结果数为 100。您可以使用 --max-results 选项来更改此结果限制。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo --max-results 20
```

--max-results 允许的最大值为 1,000。要允许列出包含 1,000 个以上程序包的存储库中的程序包，list-packages 支持在响应中使用 nextToken 字段进行分页。如果存储库中的程序包数量超过 --max-results 的值，则可以将 nextToken 的值传递给另一个 list-packages 调用，来获得下一页的结果。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo \
  --next-token rO0ABXNyAEdjb...
```

列出程序包版本

使用中的list-package-versions命令 AWS CodeArtifact 获取存储库中软件包名称的所有版本的列表。

```
aws codeartifact list-package-versions --package kind-of \
--domain my_domain --domain-owner 111122223333 \
--repository my_repository --format npm
```

示例输出：


```
{
  "defaultDisplayVersion": "1.0.1",
  "format": "npm",
  "package": "kind-of",
  "versions": [
      {
          "version": "1.0.1",
          "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      },
      {
          "version": "1.0.0",
          "revision": "REVISION-SAMPLE-2-C752BEEF6D2CFC",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      },
      {
          "version": "0.1.2",
          "revision": "REVISION-SAMPLE-3-654S65A5C5E1FC",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      },
      {
          "version": "0.1.1",
          "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"",
          "status": "Published",
```

```
            "origin": {
                  "domainEntryPoint": {
                      "externalConnectionName": "public:npmjs"
                  },
                  "originType": "EXTERNAL"
                }
          },
          {
              "version": "0.1.0",
              "revision": "REVISION-SAMPLE-4-AF669139B772FC",
              "status": "Published",
              "origin": {
                  "domainEntryPoint": {
                      "externalConnectionName": "public:npmjs"
                  },
                  "originType": "EXTERNAL"
                }
          }
    ]
}
```

您可以将 `--status` 参数添加到 `list-package-versions` 调用中，从而根据程序包版本状态筛选结果。有关程序包版本状态的更多信息，请参阅程序包版本状态。

您可以使用 `--max-results` 和 `--next-token` 参数对来自 `list-package-versions` 的响应进行分页。对于 `--max-results`，指定一个 1 至 1000 之间的整数，用来指定在一页中返回的结果数。其默认值为 50。要返回后续页面，请再次运行 `list-package-versions` 并将上一个命令输出中接收到的 `nextToken` 值传递给 `--next-token`。如果未使用 `--next-token` 选项，则始终返回结果的第一页。

`list-package-versions` 命令不会列出上游存储库中的程序包版本。但会列出对上游存储库中程序包版本的引用，这些版本在程序包版本请求过程中复制到您的存储库。有关更多信息，请参阅 在 CodeArtifact 中使用上游存储库。

列出 npm 程序包版本

要列出 npm 程序包的所有程序包版本，请将 `--format` 选项的值设置为 `npm`。

```
aws codeartifact list-package-versions --package my_package --domain my_domain \
 --domain-owner 111122223333 --repository my_repo --format npm
```

要列出特定命名空间 (npm scope) 中的 npm 程序包版本，请使用 --namespace 选项。--namespace 选项的值不得包括前导 @。要搜索命名空间 @types，请将值设置为 *types*。

```
aws codeartifact list-package-versions --package my_package --domain my_domain \
 --domain-owner 111122223333 --repository my_repo --format npm \
 --namespace types
```

列出 Maven 程序包版本

要列出 Maven 程序包的所有程序包版本，请将 --format 选项的值设置为 maven。您还必须在 --namespace 选项中指定 Maven 组 ID。

```
aws codeartifact list-package-versions --package my_package --domain my_domain \
 --domain-owner 111122223333 --repository my_repo --format maven \
 --namespace org.apache.commons
```

对版本进行排序

list-package-versions 可以根据发布时间按降序排列输出版本（最先列出最近发布的版本）。使用值为 PUBLISHED_TIME 的 --sort-by 参数，如下所示。

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333
 --repository my_repository  \
--format npm --package webpack --max-results 5 --sort-by PUBLISHED_TIME
```

示例输出：

```
{

  "defaultDisplayVersion": "4.41.2",
  "format": "npm",
  "package": "webpack",
  "versions": [
      {
        "version": "5.0.0-beta.7",
        "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
        "status": "Published"
      },
      {
```

```
        "version": "5.0.0-beta.6",
        "revision": "REVISION-SAMPLE-2-C752BEEF6D2CFC",
        "status": "Published"
      },
      {
        "version": "5.0.0-beta.5",
        "revision": "REVISION-SAMPLE-3-654S65A5C5E1FC",
        "status": "Published"
      },
      {
        "version": "5.0.0-beta.4",
        "revision": "REVISION-SAMPLE-4-AF669139B772FC",
        "status": "Published"
      },
      {
        "version": "5.0.0-beta.3",
        "revision": "REVISION-SAMPLE-5-C752BEE9B772FC",
        "status": "Published"
      }
  ],
  "nextToken": "eyJsaXN0UGF...."
}
```

默认显示版本

defaultDisplayVersion 的返回值取决于程序包格式：

- 对于通用、Maven 和 PyPI 程序包，返回值是最新发布的程序包版本。
- 对于 npm 程序包，返回值是 latest 标签引用的版本。如果未设置 latest 标签，则返回值是最近发布的程序包版本。

格式输出

您可以使用所有 AWS CLI 命令都可用的参数来使list-package-versions响应更紧凑且更具可读性。使用 --query 参数来指定每个返回的程序包版本的格式。使用 --output 参数将响应格式化为纯文本。

```
aws codeartifact list-package-versions --package my-package-name --domain my_domain --
domain-owner 111122223333 \
--repository my_repo --format npm --output text --query 'versions[*].[version]'
```

示例输出：

```
0.1.1
0.1.2
0.1.0
3.0.0
```

有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的从 AWS CLI控制命令输出。

列出程序包版本资产

资产是存储在其中与软件包版本关联的单个文件（例如 npm `.tgz` 文件或 Maven POM 或 JAR 文件）。 CodeArtifact 您可以使用 `list-package-version-assets` 命令列出每个程序包版本中的资产。

运行`list-package-version-assets`命令以返回有关您 AWS 账户中每项资产和当前 AWS 地区的以下信息：

- 它的名称。
- 它的大小（以字节为单位）。
- 一组用于校验和验证的哈希值。

例如，使用以下命令来列出 Python 程序包 `flatten-json` 版本 `0.1.7` 的资产。

```
aws codeartifact list-package-version-assets --domain my_domain --domain-
owner 111122223333 \
 --repository my_repo --format pypi --package flatten-json \
 --package-version 0.1.7
```

下面显示了输出。

```
{
    "format": "pypi",
    "package": "flatten-json",
    "version": "0.1.7",
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
    "assets": [
        {
```

```
            "name": "flatten_json-0.1.7-py3-none-any.whl",
            "size": 31520,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f086
SHA-512"
            }
        },
        {
            "name": "flatten_json-0.1.7.tar.gz",
            "size": 2865,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f086
SHA-512"
            }
        }
    ]
}
```

列出 npm 程序包的资产

npm 程序包总是有一个名为 `package.tgz` 的资产。要列出限定范围的 npm 程序包的资产，请在 `--namespace` 选项中包括作用域。

```
aws codeartifact list-package-version-assets --domain my_domain --domain-
owner 111122223333 \
 --repository my_repo --format npm --package webpack \
 --namespace types --package-version 4.9.2
```

列出 Maven 程序包的资产

要列出 Maven 程序包的资产，请在 `--namespace` 选项中包括程序包命名空间。要列出 Maven 程序包 `commons-cli:commons-cli` 的资产，请执行以下操作：

```
aws codeartifact list-package-version-assets --domain my_domain --domain-
owner 111122223333 \
 --repository my_repo --format maven --package commons-cli \
 --namespace commons-cli --package-version 1.0
```

下载程序包版本资源

资产是存储在其中与软件包版本关联的单个文件（例如 npm .tgz 文件或 Maven POM 或 JAR 文件）。 CodeArtifact 您可以使用 get-package-version-assets command 下载程序包资产。这样您就可以取回资产，而无需使用程序包管理器客户端（如 npm 或 pip）。要下载资产，必须提供可使用 list-package-version-assets 命令获取的资产名称，有关更多信息，请参阅列出程序包版本资产。使用您指定的文件名将资源下载到本地存储。

以下示例从版本为 *27.1-jre* 的 Maven 程序包 *com.google.guava:guava* 中下载 *guava-27.1-jre.jar* 资产。

```
aws codeartifact get-package-version-asset --domain my_domain --domain-
owner 111122223333 --repository my_repo \
    --format maven --namespace com.google.guava --package guava --package-version 27.1-
jre \
    --asset guava-27.1-jre.jar \
    guava-27.1-jre.jar
```

在此示例中，上述命令的最后一个参数将文件名指定为 *guava-27.1-jre.jar*，因此下载的资产将命名为 *guava-27.1-jre.jar*。

命令的输出如下：

```
{
    "assetName": "guava-27.1-jre.jar",
    "packageVersion": "27.1-jre",
    "packageVersionRevision": "YGp9ck2tmy03PGSxioclfYzQ0BfTLR9zzhQJtERv62I="
}
```

> **Note**
>
> 要从限定范围的 npm 程序包下载资产，请在 --namespace 选项中包括作用域。使用 --namespace 时必须省略 @ 符号。例如，如果作用域是 @types，则使用 --namespace types。

使用 `get-package-version-asset` 下载资源需要获得程序包资源的 `codeartifact:GetPackageVersionAsset` 权限。有关基于资源的权限策略的更多信息，请参阅《AWS Identity and Access Management 用户指南》中的基于资源的策略。

在存储库之间复制程序包

您可以在中将软件包版本从一个存储库复制到另一个存储库 CodeArtifact。这对于程序包提升工作流或在团队或项目之间共享程序包版本等场景很有用。要复制程序包版本，源存储库和目标存储库必须位于同一个域中。

复制程序包所需的 IAM 权限

要在中复制软件包版本 CodeArtifact，调用用户必须具有所需的 IAM 权限，并且附加到源存储库和目标存储库的基于资源的策略必须具有所需的权限。有关基于资源的权限策略和 CodeArtifact 存储库的更多信息，请参阅 存储库策略。

调用 `copy-package-versions` 的用户必须具有源存储库的 `ReadFromRepository` 权限和目标存储库的 `CopyPackageVersions` 权限。

源存储库必须具有 `ReadFromRepository` 权限，目标存储库必须具有 `CopyPackageVersions` 权限，这些权限分配给 IAM 账户或复制程序包的用户。以下策略是使用 `put-repository-permissions-policy` 命令添加到源存储库或目标存储库的示例存储库策略。将 *111122223333* 替换为调用 `copy-package-versions` 的账户的 ID。

> **Note**
>
> 调用 `put-repository-permissions-policy` 会替换当前存储库策略（如果存在策略）。您可以使用 `get-repository-permissions-policy` 命令来查看是否存在策略，有关更多信息，请参阅读取策略。如果存在策略，则可能需要向策略中添加这些权限，而不是替换策略。

示例源存储库权限策略

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

```
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::111122223333:root"
            },
            "Resource": "*"
        }
    ]
}
```

示例目标存储库权限策略

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:CopyPackageVersions"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::111122223333:root"
            },
            "Resource": "*"
        }
    ]
}
```

复制程序包版本

使用中的`copy-package-versions`命令 CodeArtifact 将一个或多个软件包版本从源存储库复制到同一域中的目标存储库。以下示例会将名为 `my-package` 的 npm 程序包的 6.0.2 和 4.0.0 版本从 `my_repo` 存储库复制到 `repo-2` 存储库。

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 --source-repository my_repo \
 --destination-repository repo-2 --package my-package --format npm \
 --versions 6.0.2 4.0.0
```

您可以在单个操作中复制同一程序包名称的多个版本。要复制不同程序包名称的版本，必须为每个版本调用 `copy-package-versions`。

假设可以成功复制两个版本，则上述命令会生成以下输出。

```
{
   "successfulVersions": {
        "6.0.2": {
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        "4.0.0": {
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    },
    "failedVersions": {}
}
```

从上游存储库复制程序包

通常，copy-package-versions 仅在 --source-repository 选项指定的存储库中查找要复制的版本。但是，您可以使用 --include-from-upstream 选项从源存储库及其上游存储库中复制版本。如果您使用 CodeArtifact 软件开发工具包，请在includeFromUpstream参数设置为 true 的情况下调用 CopyPackageVersions API。有关更多信息，请参阅 在 CodeArtifact 中使用上游存储库。

复制限定范围的 npm 程序包

要复制作用域中的 npm 程序包版本，请使用 --namespace 选项指定作用域。例如，要复制程序包 @types/react，请使用 --namespace types。使用 --namespace 时必须省略 @ 符号。

```
aws codeartifact copy-package-versions  --domain my_domain --domain-owner 111122223333
 --source-repository repo-1 \
 --destination-repository repo-2 --format npm --namespace types \
 --package react --versions 0.12.2
```

复制 Maven 程序包版本

要在存储库之间复制 Maven 程序包版本，请通过使用 --namespace 选项来传递 Maven 组 ID 并使用 --name 选项来传递 Maven artifactID，从而指定要复制的程序包。例如，要复制 com.google.guava:guava 的单个版本：

```
 aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 \
```

```
 --source-repository my_repo --destination-repository repo-2 --format maven --
namespace com.google.guava \
 --package guava --versions 27.1-jre
```

如果成功复制了程序包版本，则输出将类似于以下内容。

```
{
    "successfulVersions": {
        "27.1-jre": {
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    },
    "failedVersions": {}
}
```

源存储库中不存在的版本

如果您指定的版本在源存储库中不存在，则复制会失败。如果源存储库中存在某些版本，而有些版本不存在，则所有版本都将无法复制。在以下示例中，源存储库中存在 `array-unique` npm 程序包的版本 0.2.0，但不存在版本 5.6.7：

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333 \
        --source-repository my_repo --destination-repository repo-2 --format npm \
        --package array-unique --versions 0.2.0 5.6.7
```

此场景中的输出将类似于以下内容。

```
{
    "successfulVersions": {},
    "failedVersions": {
        "0.2.0": {
            "errorCode": "SKIPPED",
            "errorMessage": "Version 0.2.0 was skipped"
        },
        "5.6.7": {
            "errorCode": "NOT_FOUND",
            "errorMessage": "Could not find version 5.6.7"
        }
    }
}
```

SKIPPED 错误代码用于表示因为无法复制另一个版本，所以未将该版本复制到目标存储库。

目标存储库中已存在的版本

将软件包版本复制到已存在的存储库时，会 CodeArtifact比较两个存储库中的软件包资产和软件包版本级别的元数据。

如果源存储库和目标存储库中的程序包版本资产和元数据相同，则不会执行复制，而是认为该操作已成功。这意味着 copy-package-versions 是幂等的。发生这种情况时，源存储库和目标存储库中已经存在的版本将不会在 copy-package-versions 的输出中列出。

在以下示例中，源存储库 repo-1 中存在 npm 程序包 array-unique 的两个版本。目标存储库 dest-repo 中也存在版本 0.2.1，但不存在版本 0.2.0。

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333 \
         --source-repository my_repo --destination-repository repo-2 --format npm --
package array-unique \
         --versions 0.2.1 0.2.0
```

此场景中的输出将类似于以下内容。

```
{
    "successfulVersions": {
        "0.2.0": {
            "revision": "Yad+B1QcBq2kdEVrx1E1vSfHJVh8Pr61hBUkoWPGWX0=",
            "status": "Published"
        }
    },
    "failedVersions": {}
}
```

因为版本 0.2.0 已成功地从源存储库复制到目标存储库，所以在 successfulVersions 中列出该版本。因为目标存储库中已经存在版本 0.2.1，所以在输出中未显示该版本。

如果源存储库和目标存储库中的程序包版本资产或元数据不同，则复制操作会失败。您可以使用 --allow-overwrite 参数来强制覆盖。

如果目标存储库中存在某些版本，而有些版本不存在，则所有版本都将无法复制。在以下示例中，源存储库和目标存储库中都存在 array-unique npm 程序包的版本 0.3.2，但程序包版本的内容不同。源存储库中存在版本 0.2.1，但目标存储库中不存在该版本。

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333 \
        --source-repository my_repo --destination-repository repo-2 --format npm --
package array-unique \
        --versions 0.3.2 0.2.1
```

此场景中的输出将类似于以下内容。

```
{
    "successfulVersions": {},
    "failedVersions": {
        "0.2.1": {
            "errorCode": "SKIPPED",
            "errorMessage": "Version 0.2.1 was skipped"
        },
        "0.3.2": {
            "errorCode": "ALREADY_EXISTS",
            "errorMessage": "Version 0.3.2 already exists"
        }
    }
}
```

因为版本 0.2.1 未复制到目标存储库，所以该版本标记为 SKIPPED。因为目标存储库中已经存在版本 0.3.2，但该版本在源存储库和目标存储库中不完全相同，所以该版本复制失败，没有复制该版本。

指定程序包版本修订

程序包版本修订是一个字符串，它为程序包版本指定一组特定的资产和元数据。您可以指定程序包版本修订，用来复制处于特定状态的程序包版本。要指定程序包版本修订，请使用 `--version-revisions` 参数将一个或多个逗号分隔的程序包版本和程序包版本修订对传递给 `copy-package-versions` 命令。

> **Note**
>
> 必须使用 `copy-package-versions` 指定 `--versions` 或 `--version-revisions` 参数。不能同时指定两者。

仅在源存储库中存在程序包 `my-package` 的版本 0.3.2，且程序包版本修订为 `REVISION-1-SAMPLE-6C81EFF7DA55CC` 时，以下示例才会复制该版本。

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 --source-repository repo-1 \
 --destination-repository repo-2 --format npm --namespace my-namespace \
 --package my-package --version-revisions 0.3.2=REVISION-1-SAMPLE-6C81EFF7DA55CC
```

以下示例复制程序包 my-package 的两个版本，即 0.3.2 和 0.3.13。仅当在源存储库中，my-package 的版本 0.3.2 具有修订 REVISION-1-SAMPLE-6C81EFF7DA55CC，且版本 0.3.13 具有修订 REVISION-2-SAMPLE-55C752BEE772FC 时，复制才会成功。

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 --source-repository repo-1 \
 --destination-repository repo-2 --format npm --namespace my-namespace \
 --package my-package --version-revisions 0.3.2=REVISION-1-
SAMPLE-6C81EFF7DA55CC,0.3.13=REVISION-2-SAMPLE-55C752BEE772FC
```

要查找程序包版本的修订，请使用 describe-package-version 或 list-package-versions 命令。

有关更多信息，请参阅 CodeArtifact API 参考CopyPackageVersion中的程序包版本修订和。

复制 npm 程序包

有关 npm 包copy-package-versions行为的更多信息，请参阅 npm 标签和 API。 CopyPackageVersions

删除包

可以使用 delete-package-versions 命令一次删除一个或多个程序包版本。要从存储库中完全删除程序包（包括所有关联的版本和配置），请使用 delete-package 命令。存储库中的程序包可以没有任何程序包版本。当使用 delete-package-versions 命令删除所有版本时，或者使用 put-package-origin-configuration API 操作创建没有任何版本的程序包时（请参阅编辑程序包来源控制），可能会出现这种情况。

主题

- 删除软件包 (AWS CLI)
- 删除程序包版本 (AWS CLI)
- 删除软件包（控制台）

- 删除软件包版本（控制台）
- 删除 npm 程序包
- 删除 Maven 程序包

删除软件包 (AWS CLI)

您可以使用 delete-package 命令删除程序包，包括其所有程序包版本和配置。以下示例删除 my_domain 域的存储库 my_repo 中名为 my-package 的 PyPI 程序包：

```
aws codeartifact delete-package --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format pypi \
--package my-package
```

示例输出：

```
{
   "deletedPackage": {
      "format": "pypi",
      "originConfiguration": {
         "restrictions": {
            "publish": "ALLOW",
            "upstream": "BLOCK"
         }
      },
      "package": "my-package"
   }
}
```

您可以对同一个程序包运行 describe-package 来确认已删除程序包：

```
aws codeartifact describe-package --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format pypi --package my-package
```

删除程序包版本 (AWS CLI)

可以使用 delete-package-versions 命令一次删除一个或多个程序包版本。以下示例删除 my_domain 域的 my_repo 中名为 my-package 的 PyPI 程序包的版本 4.0.0、4.0.1 和 5.0.0：

```
aws codeartifact delete-package-versions --domain my_domain --domain-owner 111122223333
 \
--repository my_repo --format pypi \
--package my-package --versions 4.0.0 4.0.1 5.0.0
```

示例输出：

```
{
   "successfulVersions": {
      "4.0.0": {
         "revision": "oxwwYC9dDeuBoCt6+PDSwL6OMZ7rXeiXy44BM32Iawo=",
            "status": "Deleted"
      },
      "4.0.1": {
         "revision": "byaaQR748wrsdBaT+PDSwL6OMZ7rXeiBKM0551aqWmo=",
            "status": "Deleted"
      },
      "5.0.0": {
         "revision": "yubm34QWeST345ts+ASeioPI354rXeiSWr734PotwRw=",
            "status": "Deleted"
      }
   },
   "failedVersions": {}
}
```

您可以对同一个程序包运行 `list-package-versions` 来确认已删除版本：

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format pypi --package my-package
```

删除软件包（控制台）

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择存储库。
3. 选择要从中删除程序包的存储库。
4. 选择要删除的程序包。
5. 选择删除程序包。

删除软件包版本（控制台）

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择存储库。
3. 选择要从中删除程序包版本的存储库。
4. 选择要从中删除版本的程序包。
5. 选择要删除的程序包版本。
6. 选择 Delete（删除）。

> **Note**
>
> 在控制台中，可以一次仅删除一个程序包版本。要一次删除多个版本，请使用 CLI。

删除 npm 程序包

要删除 npm 程序包或单个程序包版本，请将 `--format` 选项设置为 `npm`。要删除限定范围的 npm 程序包中的程序包版本，请使用 `--namespace` 选项来指定作用域。例如，要删除程序包 `@types/react`，请使用 `--namespace types`。使用 `--namespace` 时省略 `@` 符号。

```
aws codeartifact delete-package-versions --domain my_domain --domain-owner 111122223333
 \
--repository my_repo --format npm --namespace types \
--package react --versions 0.12.2
```

要删除程序包 `@types/react`，包括其所有版本，请执行以下操作：

```
aws codeartifact delete-package --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format npm --namespace types \
--package react
```

删除 Maven 程序包

要删除 Maven 程序包或单个程序包版本，请将 `--format` 选项设置为 `maven`，并使用 `--namespace` 选项传递 Maven 组 ID 和使用 `--name` 选项传递 Maven artifactID，从而指定要删除的程序包。例如，下面说明了如何删除 `com.google.guava:guava` 的单个版本。

```
 aws codeartifact delete-package-versions --domain my_domain --domain-
owner 111122223333 \
--repository my_repo --format maven --namespace com.google.guava \
--package guava --versions 27.1-jre
```

以下示例说明如何删除程序包 com.google.guava:guava，包括其所有版本：

```
 aws codeartifact delete-package --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format maven --namespace com.google.guava \
--package guava
```

查看和更新程序包版本详细信息和依赖项

您可以在中查看有关软件包版本的信息，包括依赖关系 CodeArtifact。您也可以更新程序包版本的状态。有关程序包版本状态的更多信息，请参阅程序包版本状态。

查看程序包版本详细信息

使用 describe-package-version 命令来查看有关程序包版本的详细信息。Package 版本详细信息是在软件包发布到时从包中提取的 CodeArtifact。不同程序包中的详细信息各不相同，且取决于程序包的格式以及作者向其中添加了多少信息。

describe-package-version 命令输出中的大多数信息都取决于程序包的格式。例如，describe-package-version 从其 package.json 文件中提取 npm 程序包的信息。修订版由创建 CodeArtifact。有关更多信息，请参阅 指定程序包版本修订。

如果两个同名的程序包版本位于不同的命名空间中，则它们可以位于同一个存储库中。使用可选的 --namespace 参数来指定命名空间。有关更多信息，请参阅查看 npm 程序包版本详细信息或查看 Maven 程序包版本详细信息。

以下示例返回有关 my_repo 存储库中名为 pyhamcrest 的 Python 程序包的版本 1.9.0 的详细信息。

```
aws codeartifact describe-package-version --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format pypi --package pyhamcrest --package-version 1.9.0
```

输出可能看起来类似以下内容。

```
{
  "format": "pypi",
  "package": "PyHamcrest",
  "displayName": "PyHamcrest",
  "version": "1.9.0",
  "summary": "Hamcrest framework for matcher objects",
  "homePage": "https://github.com/hamcrest/PyHamcrest",
  "publishedTime": 1566002944.273,
  "licenses": [
    {
      "id": "license-id",
      "name": "license-name"
    }
  ],
  "revision": "REVISION-SAMPLE-55C752BEE9B772FC"
}
```

查看 npm 程序包版本详细信息

要查看有关 npm 程序包版本的详细信息，请将 `--format` 选项的值设置为 **npm**。（可选）在 `--namespace` 选项中包括程序包版本命名空间 (npm scope)。`--namespace` 选项的值不得包括前导 `@`。要搜索命名空间 `@types`，请将值设置为 *types*。

下面返回有关 `@types` 作用域中名为 `webpack` 的 npm 程序包的版本 `4.41.5` 的详细信息。

```
aws codeartifact describe-package-version --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format npm --package webpack --namespace types --package-version 4.41.5
```

输出可能看起来类似以下内容。

```
{
  "format": "npm",
  "namespace": "types",
  "package": "webpack",
  "displayName": "webpack",
  "version": "4.41.5",
  "summary": "Packs CommonJs/AMD modules for the browser. Allows ... further output
 omitted for brevity",
  "homePage": "https://github.com/webpack/webpack",
  "sourceCodeRepository": "https://github.com/webpack/webpack.git",
```

```
  "publishedTime": 1577481261.09,
  "licenses": [
    {
      "id": "license-id",
      "name": "license-name"
    }
  ],
  "revision": "REVISION-SAMPLE-55C752BEE9B772FC",
  "status": "Published",
  "origin": {
            "domainEntryPoint": {
                "externalConnectionName": "public:npmjs"
            },
            "originType": "EXTERNAL"
  }
}
```

查看 Maven 程序包版本详细信息

要查看有关 Maven 程序包版本的详细信息，请将 `--format` 选项的值设置为 `maven`，并在 `--namespace` 选项中包括程序包版本命名空间。

以下示例返回有关 `org.apache.commons` 命名空间和 `my_repo` 存储库中名为 `commons-rng-client-api` 的 Maven 程序包的版本 `1.2` 的详细信息。

```
aws codeartifact describe-package-version --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format maven --namespace org.apache.commons --package commons-rng-client-api --
package-version 1.2
```

输出可能看起来类似以下内容。

```
{
  "format": "maven",
  "namespace": "org.apache.commons",
  "package": "commons-rng-client-api",
  "displayName": "Apache Commons RNG Client API",
  "version": "1.2",
  "summary": "API for client code that uses random numbers generators.",
  "publishedTime": 1567920624.849,
  "licenses": [],
  "revision": "REVISION-SAMPLE-55C752BEE9B772FC"
```

```
}
```

> **Note**
>
> CodeArtifact 不会从父 POM 文件中提取软件包版本详细信息。给定程序包版本的元数据将仅包括该确切程序包版本的 POM 中的信息，而不包括父 POM 或使用 POM `parent` 标签以传递方式引用的任何其他 POM 的信息。这意味着，`describe-package-version` 的输出会省略依赖于 `parent` 引用来包含此元数据的 Maven 程序包版本的元数据（例如许可证信息）。

查看程序包版本依赖项

使用 `list-package-version-dependencies` 命令来获取程序包版本依赖项的列表。以下命令列出 `my_domain` 域的 `my_repo` 存储库中名为 `my-package`、版本为 `4.41.5` 的 npm 程序包的依赖项。

```
aws codeartifact list-package-version-dependencies --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format npm --package my-package --package-version 4.41.5
```

输出可能看起来类似以下内容。

```
{
  "dependencies": [
    {
      "namespace": "webassemblyjs",
      "package": "ast",
      "dependencyType": "regular",
      "versionRequirement": "1.8.5"
    },
    {
      "namespace": "webassemblyjs",
      "package": "helper-module-context",
      "dependencyType": "regular",
      "versionRequirement": "1.8.5"
    },
    {
      "namespace": "webassemblyjs",
      "package": "wasm-edit",
      "dependencyType": "regular",
```

```
      "versionRequirement": "1.8.5"
    }
  ],
  "versionRevision": "REVISION-SAMPLE-55C752BEE9B772FC"
}
```

有关 DependencyType 字段支持的值范围，请参阅 API 中的PackageDependency数据类型。CodeArtifact

查看程序包版本自述文件

某些程序包格式（例如 npm）包括一个 README 文件。使用 `get-package-version-readme` 来获取程序包版本的 README 文件。以下命令返回 `my_domain` 域的 `my_repo` 存储库中名为 `my-package`、版本为 `4.41.5` 的 npm 程序包的 README 文件。

> **Note**
>
> CodeArtifact 不支持显示来自通用包或 Maven 包的自述文件。

```
aws codeartifact get-package-version-readme --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format npm --package my-package --package-version 4.41.5
```

输出可能看起来类似以下内容。

```
{
  "format": "npm",
  "package": "my-package",
  "version": "4.41.5"
  "readme": "<div align=\"center\">\n   <a href=\https://github.com/webpack/webpack
\"> ... more content ... \n",
  "versionRevision": "REVISION-SAMPLE-55C752BEE9B772FC"
}
```

更新程序包版本状态

中的每个软件包版本都 CodeArtifact 有一个描述软件包版本的当前状态和可用性的状态。您可以使用 AWS CLI 和控制台更改软件包版本状态。

> **Note**
>
> 有关程序包版本状态的更多信息（包括可用状态列表），请参阅程序包版本状态。

更新程序包版本状态

通过设置程序包版本的状态，可以控制如何使用程序包版本，而无需完全从存储库中删除版本。例如，当程序包版本的状态为 `Unlisted` 时，仍然可以正常下载版本，但使用 `npm view` 等命令返回的程序包版本列表中不会显示该版本。该 UpdatePackageVersionsStatus API 允许在单个 API 调用中设置同一个软件包的多个版本的软件包版本状态。有关不同状态的说明，请参阅程序包概览。

使用 `update-package-versions-status` 命令将程序包版本的状态更改为 `Published`、`Unlisted` 或 `Archived`。要查看使用命令所需的 IAM 权限，请参阅更新程序包版本状态所需的 IAM 权限。以下示例将 npm 程序包 `chalk` 版本 4.1.0 的状态设置为 `Archived`。

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format npm --package chalk
--versions 4.1.0 --target-status Archived
```

示例输出：

```
{
    "successfulVersions": {
        "4.1.0": {
            "revision": "+Oz8skWbwY3k8M6SrNIqNj6bVH/ax+CxvkJx+No5j8I=",
            "status": "Archived"
        }
    },
    "failedVersions": {}
}
```

此示例使用 npm 程序包，但该命令也适用于其他格式。使用单个命令可以将多个版本切换为相同的目标状态，请参见以下示例。

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format npm --package chalk
--versions 4.1.0 4.1.1 --target-status Archived
```

示例输出：

```
{
    "successfulVersions": {
        "4.1.0": {
            "revision": "25/UjBleHs1DZewk+zozoeqH/R80Rc9gL1P8vbzVMJ4=",
            "status": "Archived"
        },
        "4.1.1": {
            "revision": "+Oz8skWbwY3k8M6SrNIqNj6bVH/ax+CxvkJx+No5j8I=",
            "status": "Archived"
        }
    },
    "failedVersions": {}
}
```

请注意，发布程序包版本之后，该版本就无法切换回 Unfinished 状态，因此不允许将此状态作为 `--target-status` 参数的值。要将程序包版本切换至 Disposed 状态，请改用 `dispose-package-versions` 命令，如下所述。

更新程序包版本状态所需的 IAM 权限

要为程序包调用 `update-package-versions-status`，您必须拥有程序包资源的 `codeartifact:UpdatePackageVersionsStatus` 权限。这意味着您可以针对各个程序包授予调用 `update-package-versions-status` 的权限。例如，如果在 IAM 策略中授予在 npm 程序包 *chalk* 上调用 `update-package-versions-status` 的权限，则该策略将包括如下语句。

```
{
  "Action": [
    "codeartifact:UpdatePackageVersionsStatus"
  ],
  "Effect": "Allow",
  "Resource": "arn:aws:codeartifact:us-east-1:111122223333:package/my_domain/my_repo/
npm//chalk"
}
```

更新限定范围的 npm 程序包的状态

要更新具有作用域的 npm 程序包版本的程序包版本状态，请使用 `--namespace` 参数。例如，要取消列出 `@nestjs/core` 的版本 8.0.0，请使用以下命令。

```
aws codeartifact update-package-versions-status --domain my_domain
```

```
 --domain-owner 111122223333 --repository my_repo --format npm --namespace nestjs
--package core --versions 8.0.0 --target-status Unlisted
```

更新 Maven 程序包的状态

Maven 软件包始终有一个组 ID，在中将其称为命名空间。 CodeArtifact调用 `update-package-versions-status` 时使用 `--namespace` 参数来指定 Maven 组 ID。例如，要存档 Maven 程序包 `org.apache.logging.log4j:log4j` 的版本 2.13.1，请使用以下命令。

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format maven
--namespace org.apache.logging.log4j --package log4j
--versions 2.13.1 --target-status Archived
```

指定程序包版本修订

程序包版本修订是一个字符串，它为程序包版本指定一组特定的资产和元数据。您可以指定程序包版本修订来更新处于特定状态的程序包版本的状态。要指定程序包版本修订，请使用 `--version-revisions` 参数来传递一个或多个逗号分隔的程序包版本和程序包版本修订对。仅当程序包版本的当前修订与指定值相匹配时，才会更新程序包版本的状态。

> **Note**
>
> 使用 `--version-revisions` 参数时还必须定义 `—-versions` 参数。

```
aws codeartifact update-package-versions-status --domain my_domain
   --domain-owner 111122223333 --repository my_repo --format npm --package chalk
   --version-revisions "4.1.0=25/UjBleHs1DZewk+zozoeqH/R80Rc9gL1P8bzVMJ4="
   --versions 4.1.0 --target-status Archived
```

要使用单个命令更新多个版本，请将以逗号分隔的版本和版本修订对列表传递给 `--version-revisions` 选项。以下示例命令定义了两个不同的程序包版本和程序包版本修订对。

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format npm
 --package chalk
 --version-revisions "4.1.0=25/UjBleHs1DZewk+zozoeqH/
R80Rc9gL1P8vbzVMJ4=,4.0.0=E3lhBp0RObRTut4pkjV5c1AQGkgSA7Oxtil6hMMzelc="
```

```
 --versions 4.1.0 4.0.0 --target-status Published
```

示例输出：

```
{
    "successfulVersions": {
        "4.0.0": {
            "revision": "E3lhBp0RObRTut4pkjV5c1AQGkgSA7Oxtil6hMMzelc=",
            "status": "Published"
        },
        "4.1.0": {
            "revision": "25/UjBleHs1DZewk+zozoeqH/R80Rc9gL1P8vbzVMJ4=",
            "status": "Published"
        }
    },
    "failedVersions": {}
}
```

更新多个程序包版本时，传递到 --version-revisions 的版本必须与传递到 --versions 的版本相同。如果错误地指定了修订，则该版本的状态将不会更新。

使用预期的状态参数

update-package-versions-status 命令提供的 --expected-status 参数支持指定程序包版本的预期当前状态。如果当前状态与传递给 --expected-status 的值不匹配，则不会更新该程序包版本的状态。

例如，在 *my_repo* 中，npm 程序包 chalk 的版本 4.0.0 和 4.1.0 目前的状态为 Published。由于状态不匹配，调用 update-package-versions-status（指定了 Unlisted 预期状态）将无法更新两个程序包版本。

```
aws codeartifact update-package-versions-status --domain my_domain
--domain-owner 111122223333 --repository my_repo --format npm --package chalk
--versions 4.1.0 4.0.0 --target-status Archived --expected-status Unlisted
```

示例输出：

```
{
    "successfulVersions": {},
    "failedVersions": {
```

```
        "4.0.0": {
            "errorCode": "MISMATCHED_STATUS",
            "errorMessage": "current status: Published, expected status: Unlisted"
        },
        "4.1.0": {
            "errorCode": "MISMATCHED_STATUS",
            "errorMessage": "current status: Published, expected status: Unlisted"
        }
    }
}
```

个别程序包版本的错误

调用 update-package-versions-status 时无法更新程序包版本的状态有多种原因。例如，可能错误地指定了程序包版本修订，或者预期状态与当前状态不匹配。在这些情况下，在 API 响应的 failedVersions 映射中将包括该版本。如果一个版本失败，则可能会跳过在 update-package-versions-status 的同一次调用中指定的其他版本，并且不会更新其状态。这些版本也将包括在 failedVersions 映射中，且其 errorCode 为 SKIPPED。

在 update-package-versions-status 的当前实施中，如果一个或多个版本的状态无法更改，则将跳过所有其他版本。也就是说，要么成功更新所有版本，要么不更新任何版本。在 API 合同中并不能保证这种行为；将来，在对 update-package-versions-status 的单次调用中，有些版本可能会成功，而其他版本会失败。

以下示例命令包括由于程序包版本修订不匹配而导致的版本状态更新失败。该更新失败会导致系统跳过另一个版本状态更新调用。

```
aws codeartifact update-package-versions-status --domain my_domain
  --domain-owner 111122223333 --repository my_repo
  --format npm --package chalk
  --version-revisions "4.1.0=25/UjBleHs1DZewk+zozoeqH/
R80Rc9gL1P8vbzVMJ=,4.0.0=E3lhBp0RObRTut4pkjV5c1AQGkgSA7Oxtil6hMMzelc="
  --versions 4.1.0 4.0.0 --target-status Archived
```

示例输出：

```
{
    "successfulVersions": {},
    "failedVersions": {
        "4.0.0": {
            "errorCode": "SKIPPED",
```

```
            "errorMessage": "version 4.0.0 is skipped"
        },
        "4.1.0": {
            "errorCode": "MISMATCHED_REVISION",
            "errorMessage": "current revision: 25/UjBleHs1DZewk+zozoeqH/
R80Rc9gL1P8vbzVMJ4=, expected revision: 25/UjBleHs1DZewk+zozoeqH/R80Rc9gL1P8vbzVMJ="
        }
    }
}
```

处置程序包版本

Disposed软件包状态的行为与类似Archived，唯一的不同是包裹资产将被永久删除， CodeArtifact 这样域名所有者的账户就不再需要支付资产存储费用。有关每种程序包版本状态的更多信息，请参阅程序包版本状态。要将程序包版本的状态更改为 Disposed，请使用 dispose-package-versions 命令。此功能与 update-package-versions-status 分开，因为处置程序包版本是不可逆转的。由于会删除程序包资源，因此无法将版本的状态更改回 Archived、Unlisted 或 Published。可以对已处置的程序包版本采取的唯一操作是使用 delete-package-versions 命令将其删除。

要成功调用 dispose-package-versions，发出调用的 IAM 主体必须拥有程序包资源的 codeartifact:DisposePackageVersions 权限。

dispose-package-versions 命令的行为与 update-package-versions-status 类似，包括版本修订和预期状态部分中描述的 --version-revisions 和 --expected-status 选项的行为。例如，以下命令尝试处置程序包版本，但由于预期状态不匹配而失败。

```
aws codeartifact dispose-package-versions –domain my_domain --domain-
owner 111122223333
--repository my_repo --format npm --package chalk --versions 4.0.0
--expected-status Unlisted
```

示例输出：

```
{
    "successfulVersions": {},
    "failedVersions": {
        "4.0.0": {
            "errorCode": "MISMATCHED_STATUS",
            "errorMessage": "current status: Published, expected status: Unlisted"
        }
    }
```

```
}
```

如果在 --expected-status 为 Published 时再次运行相同的命令，则处置会成功。

```
aws codeartifact dispose-package-versions –domain my_domain --domain-
owner 111122223333
--repository my_repo --format npm --package chalk --versions 4.0.0
--expected-status Published
```

示例输出：

```
{
    "successfulVersions": {
        "4.0.0": {
            "revision": "E3lhBp0RObRTut4pkjV5c1AQGkgSA7Oxtil6hMMzelc=",
            "status": "Disposed"
        }
    },
    "failedVersions": {}
}
```

编辑程序包来源控制

在中 AWS CodeArtifact，可以通过直接发布软件包版本、从上游存储库中提取包版本或从外部公共存储库中提取包版本来将其添加到存储库中。如果允许通过直接发布和从公有存储库摄取来添加程序包的程序包版本，会使您容易受到依赖项替换攻击。有关更多信息，请参阅 依赖项替换攻击。为了保护自己免受依赖项替换攻击，可以对存储库中的程序包配置程序包来源控制，从而限制将该程序包的版本添加到存储库的方式。

任何想要允许不同程序包的新版本同时来自内部来源（例如直接发布）和外部来源（例如公有存储库）的团队都应考虑配置程序包来源控制。默认情况下，根据程序包的第一个版本添加到存储库的方式来配置程序包来源控制。有关程序包来源控制设置及其默认值的信息，请参阅程序包来源控制设置。

要在使用 `put-package-origin-configuration` API 操作后删除程序包记录，请使用 `delete-package`（请参阅删除包）。

常见的程序包访问控制场景

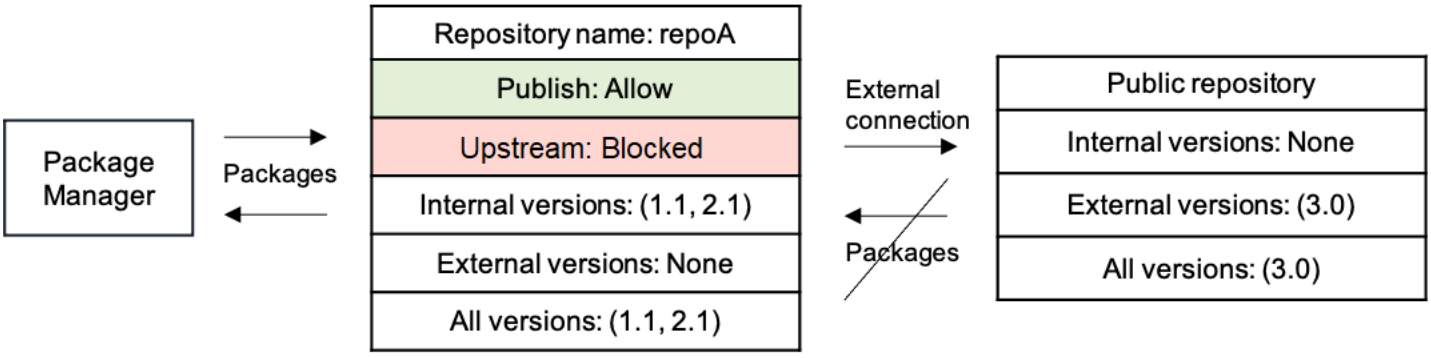
本节包括将软件包版本添加到 CodeArtifact 存储库时的一些常见场景。根据第一个程序包版本的添加方式为新程序包设置程序包来源控制设置。

在以下场景中，内部程序包是直接从程序包管理器发布到存储库的程序包，例如由您或您的团队创作和维护的程序包。外部程序包是存在于公有存储库中的程序包，可以通过外部连接将其摄取到您的存储库中。

为现有内部程序包发布了外部程序包版本

在此场景中，考虑一个内部程序包 packageA。您的团队将 PackageA 的第一个软件包版本发布到存储库。 CodeArtifact 由于这是该程序包的第一个程序包版本，因此程序包来源控制设置会自动设置为发布：允许和上游：阻止。当软件包存在于您的存储库中后，会将同名的包发布到与您的存储库连接的公共 CodeArtifact 存储库中。这可能是针对内部程序包的企图依赖项替换攻击，也可能只是巧合。无论如何，配置程序包来源控制来阻止摄取新的外部版本，从而保护自己免受潜在的攻击。

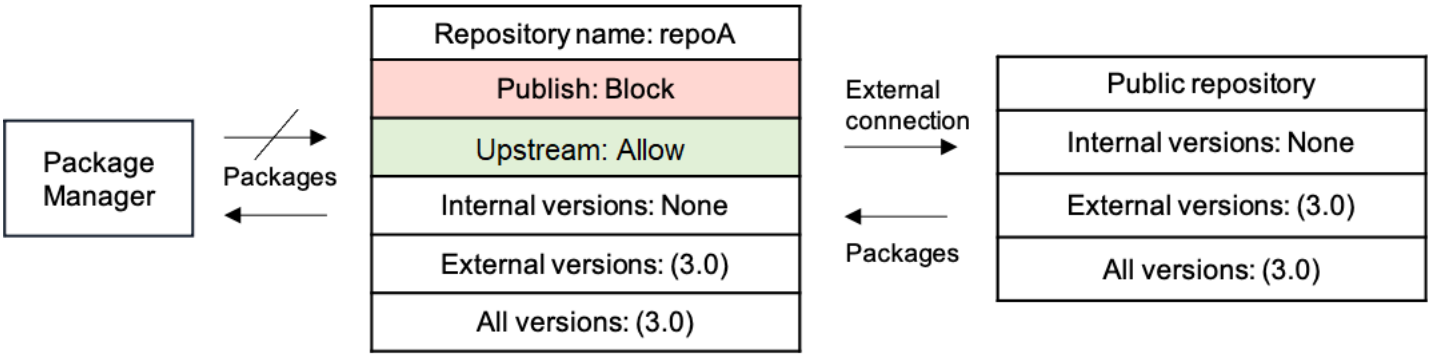
在下图中，RepoA 是您的 CodeArtifact 存储库，它与公共存储库有外部连接。您的存储库包含 packageA 的版本 1.1 和 2.1，但版本 3.0 已发布到公有存储库。通常，repoA 会在程序包管理器请求程序包后摄取版本 3.0。由于软件包提取设置为 “阻止”，因此版本 3.0 不会提取到您的 CodeArtifact 存储库中，也无法供与其连接的包管理员使用。



已为现有外部程序包发布内部程序包版本

在此场景中，在外部的公有存储库中有一个名为 packageB 的程序包，已将该公有存储库连接到您的存储库。当连接到您的存储库的程序包管理器请求 packageB 时，从公有存储库中将程序包版本摄取到您的存储库中。由于这是 packageB 添加到存储库中的第一个程序包版本，因此程序包来源设置配置为发布：阻止和上游：允许。稍后，您尝试将具有相同程序包名称的版本发布到存储库。要么你不知道这个公共包并试图用同名发布一个不相关的包，要么你正在尝试发布一个经过补丁的版本，要么你正在尝试直接发布外部已经存在的确切包版本。 CodeArtifact 将拒绝您尝试发布的版本，但允许您显式覆盖拒绝并在必要时发布该版本。

在下图中，RepoA 是您的 CodeArtifact 存储库，它与公共存储库有外部连接。您的存储库包含从公有存储库中摄取的版本 3.0。您想将版本 1.1 发布到您的存储库。通常，您可以将版本 1.2 发布到 repoA，但是由于发布设置为阻止，因此无法发布版本 1.2。



发布现有外部程序包的已修补程序包版本

在此场景中，在外部的公有存储库中有一个名为 packageB 的程序包，已将该公有存储库连接到您的存储库。当连接到您的存储库的程序包管理器请求 packageB 时，从公有存储库中将程序包版本摄取到您的存储库中。由于这是 packageB 添加到存储库中的第一个程序包版本，因此程序包来源设置配置为发布：阻止和上游：允许。您的团队确定需要将此程序包的已修补程序包版本发布到存储库。为了能够直接发布程序包版本，您的团队将程序包来源控制设置更改为发布：允许和上游：阻止。此程序包的版本

现在可以直接发布到您的存储库并从公有存储库中摄取。在您的团队发布已修补的程序包版本后，您的团队会将程序包来源设置恢复为发布：阻止和上游：允许。

程序包来源控制设置

使用程序包来源控制，您可以配置将程序包版本添加到存储库的方式。以下列表包括可用的程序包来源控制设置和值。

Publish

此设置配置了是否可以使用程序包管理器或类似工具将程序包版本直接发布到存储库。

- 允许：可以直接发布程序包版本。
- 阻止：不可以直接发布程序包版本。

上游

此设置配置了在程序包管理器发出请求时，是从外部的公有存储库中摄取程序包版本，还是在上游存储库中保留程序包版本。

- 允许：任何软件包版本都可以从配置为上游 CodeArtifact 存储库的其他存储库中保留，也可以通过外部连接从公共来源获取。
- BLOCK：Package 版本不能从配置为上游存储 CodeArtifact 库的其他存储库中保留，也不能从具有外部连接的公共来源获取。

默认程序包来源控制设置

程序包的默认来源控制基于该程序包的第一个版本添加到存储库中的方式。

> **Note**
>
> 2022 年 5 月左右之前存在于 CodeArtifact 存储库中的软件包的默认源控制为 “发布：允许” 和 “上游：允许”。必须手动为此类程序包设置程序包来源控制。从那时起，新的程序包开始采用当前的默认值，并于 2022 年 7 月 14 日推出该功能时开始强制执行。有关设置程序包来源控制的更多信息，请参阅编辑程序包来源控制。

- 如果第一个程序包版本由程序包管理器直接发布，则设置将为发布：允许和上游：阻止。

- 如果第一个程序包版本是从公有来源摄取，则设置将为发布：阻止和上游：允许。

编辑程序包来源控制

根据程序包的第一个程序包版本添加到存储库的方式来自动配置程序包来源控制，有关更多信息，请参阅默认程序包来源控制设置。要为 CodeArtifact 存储库中的软件包添加或编辑包源控件，请执行以下过程中的步骤。

添加或编辑程序包来源控制（控制台）

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择存储库，然后选择包含了待编辑程序包的存储库。
3. 在程序包表中，搜索并选择要编辑的程序包。
4. 在程序包摘要页面的来源控制中，选择编辑。
5. 在编辑来源控制中，选择要为此程序包设置的程序包来源控制。必须同时设置两个程序包来源控制设置（“发布”和“上游”）。
   - 要允许直接发布程序包版本，请在发布中选择允许。要阻止发布程序包版本，请选择阻止。
   - 要允许从外部存储库摄取程序包和从上游存储库提取程序包，请在上游来源中选择允许。要阻止所有从外部存储库和上游存储库进行的程序包版本摄取和提取，请选择阻止。

添加或编辑程序包来源控制 (AWS CLI)

1. 如果还没有，请 AWS CLI 按照中的步骤进行配置设置 AWS CodeArtifact。
2. 使用 `put-package-origin-configuration` 命令来添加或编辑程序包来源控制。替换以下字段：
   - 将 *my_dom* ain 替换为包含要更新的软件包的 CodeArtifact 域名。
   - 将 *my_repo* 替换为包含要更新的软件包的 CodeArtifact 存储库。
   - 将 *npm* 替换为要更新的程序包的程序包格式。
   - 将 *my_package* 替换为要更新的程序包的名称。
   - 将 *ALLOW* 和 *BLOCK* 替换为期望的程序包来源控制设置。

```
aws codeartifact put-package-origin-configuration --domain my_domain \
```

```
--repository my_repo --format npm --package my_package \
--restrictions publish=ALLOW,upstream=BLOCK
```

发布和上游存储库

CodeArtifact 不允许发布存在于可访问的上游存储库或公共存储库中的软件包版本。例如，假设您要将 Maven 程序包 `com.mycompany.mypackage:1.0` 发布到存储库 `myrepo`，并且 `myrepo` 有一个与 Maven Central 进行外部连接的上游存储库。考虑以下场景。

1. `com.mycompany.mypackage` 上的程序包来源控制设置为发布：允许和上游：允许。如果存在`com.mycompany.mypackage:1.0`于上游存储库或 Maven Central 中，则 CodeArtifact 会拒绝任何向其发布的尝试，并出现 409 冲突错误。`myrepo`您仍然可以发布另一个版本，例如 `com.mycompany.mypackage:1.1`。
2. `com.mycompany.mypackage` 上的程序包来源控制设置为发布：允许和上游：阻止。您可以将 `com.mycompany.mypackage` 的任何版本发布到由于无法访问程序包版本而尚不存在该版本的存储库中。
3. `com.mycompany.mypackage` 上的程序包来源控制设置为发布：阻止和上游：允许。您不能直接将任何程序包版本发布到您的存储库。

使用中的域名 CodeArtifact

CodeArtifact 域可以更轻松地管理组织中的多个存储库。您可以使用域在不同 AWS 账户拥有的许多存储库中应用权限。即使在多个存储库中提供了某个资产，该资产在域中也仅存储一次。

尽管您可以有多个域，但我们建议使用一个生产域来包含所有已发布的构件，以便您的开发团队可以找到和共享程序包。您可以使用第二个预生产域来测试生产域配置的更改。

这些主题介绍如何使用 CodeArtifact 控制台 AWS CLI、和 AWS CloudFormation 来创建或配置 CodeArtifact 域。

主题

域概述

当你使用时 CodeArtifact，域名可用于以下用途：

- 存储去重：即使在 1 个或 1000 个存储库中提供了某个资产，该资产也只需要在域中存储一次。这意味着您只需支付一次存储费用。
- 快速复制：当您将包从上游 CodeArtifact 存储库拉入下游存储库或使用 CopyPackageVersions API 时，只需要更新元数据记录。不复制任何资产。这样就可以快速设置用于暂存或测试的新存储库。有关更多信息，请参阅 在 CodeArtifact 中使用上游存储库。
- 在存储库和团队之间轻松共享：使用单个 AWS KMS key （KMS 密钥）对域中的所有资产和元数据进行加密。您不需要为每个存储库管理一个密钥，也不需要为单个密钥授予多个账户的访问权限。
- 在多个存储库中应用策略：域管理员可以在整个域应用策略。这包括限制哪些账户有权访问域中的存储库，以及谁可以配置与公有存储库的连接，以便将公有存储库作为程序包的来源。有关更多信息，请参阅域策略。
- 唯一的存储库名称：域为存储库提供命名空间。存储库名称只需要在域中保持唯一。您应该使用易于理解和有意义的名称。

域名在账户中必须唯一。

创建存储库时不能不指定所属的域。使用 CreateRepositoryAPI 创建存储库时，必须指定域名。不能将存储库从一个域移动到另一个域。

仓库可以归拥有该域的同一个 AWS 账户所有，也可以由另一个账户拥有。如果拥有域的账户不同，则必须向拥有存储库的账户授予域资源的 `CreateRepository` 权限。为此，您可以使用PutDomainPermissionsPolicy命令向域中添加资源策略。

尽管一个组织可以有多个域，但建议使用一个生产域来包含所有已发布构件，以便开发团队可以找到和共享整个组织中的程序包。第二个预生产域可用于测试对生产域配置的更改。

跨账户域

域名只需要在一个账户中保持唯一，这意味着一个区域内可能有多个域具有相同名称。因此，如果您要访问由您未通过身份验证的账户所拥有的域，则必须同时在 CLI 和控制台中提供域所有者 ID 和域名。请见以下 CLI 示例。

访问您已通过身份验证的账户所拥有的域：

访问您已通过身份验证的账户内的域时，只需要指定域名即可。以下示例列出了您的账户拥有的 *my_domain* 域中的 *my_repo* 存储库中的程序包。

```
aws codeartifact list-packages --domain my_domain --repository my_repo
```

访问您未通过身份验证的账户所拥有的域：

在访问由您未通过身份验证的账户所拥有的域时，您需要指定域所有者和域名。以下示例列出了 *other-domain* 域的 *other-repo* 存储库中的程序包，这些程序包由您未通过身份验证的账户所拥有。请注意，添加了 `--domain-owner` 参数。

```
aws codeartifact list-packages --domain other-domain --domain-owner 111122223333 --
repository other-repo
```

中支持的 AWS KMS 密钥类型 CodeArtifact

CodeArtifact 仅支持对称 KMS 密钥。您不能使用非对称 KMS 密钥来加密您的 CodeArtifact 域名。有关更多信息，请参阅识别对称和非对称 KMS 密钥。要了解如何创建新的客户管理密钥，请参阅《AWS Key Management Service 开发人员指南》的创建对称加密 KMS 密钥。

CodeArtifact 支持 AWS KMS 外部密钥存储 (XKS)。您应对使用 XKS 密钥进行关键操作的可用性、持久性和延迟负责，这可能会影响可用性、持久性和延迟。 CodeArtifact使用 XKS 密钥的效果的一些示例： CodeArtifact

- 由于所请求程序包的每项资产及其所有依赖项都受到解密延迟的影响，因此 XKS 操作延迟的增加可能会大大增加构建延迟。
- 由于所有资产都已加密 CodeArtifact，因此 XKS 密钥材料丢失将导致使用 XKS 密钥与该域相关的所有资产丢失。

有关 XKS 密钥的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的外部密钥存储。

创建域

您可以使用 CodeArtifact 控制台、 AWS Command Line Interface (AWS CLI) 或创建域 AWS CloudFormation。创建域时，域中不包含任何存储库。有关更多信息，请参阅 创建存储库。有关使用管理 CodeArtifact 域的更多信息 CloudFormation，请参阅使用 AWS CloudFormation 创建 CodeArtifact 资源。

主题

- 创建域（控制台）
- 创建域 (AWS CLI)

创建域（控制台）

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择域，然后选择创建域。
3. 在名称中，输入域的名称。
4. 展开其他配置。
5. 使用 AWS KMS key （KMS 密钥）加密您域中的所有资产。您可以使用 AWS 托管的 KMS 密钥或自己管理的 KMS 密钥。有关中支持的 KMS 密钥类型的更多信息 CodeArtifact，请参阅中支持的 AWS KMS 密钥类型 CodeArtifact。
   - 如果您想使用默认 AWS 托管式密钥，请选择 AWS 托管式密钥。

- 如果您想使用自己管理的 KMS 密钥，请选择客户管理的密钥。要使用自己管理的 KMS 密钥，请在客户管理的密钥 ARN 中搜索并选择 KMS 密钥。

有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS 托管式密钥和客户管理的密钥。

6. 选择创建域。

创建域 (AWS CLI)

要使用创建域 AWS CLI，请使用`create-domain`命令。您必须使用 AWS KMS key （KMS 密钥）来加密您域中的所有资产。您可以使用 AWS 托管 KMS 密钥或您管理的 KMS 密钥。如果您使用 AWS 托管 KMS 密钥，请不要使用`--encryption-key`参数。

有关中支持的 KMS 密钥类型的更多信息 CodeArtifact，请参阅中支持的 AWS KMS 密钥类型 CodeArtifact。有关 KMS 密钥的更多信息，请参阅 AWS 托管式密钥和《AWS Key Management Service 开发人员指南》中的客户管理的密钥。

```
aws codeartifact create-domain --domain my_domain
```

输出中会显示 JSON 格式的数据，并包含有关新域的详细信息。

```
{
    "domain": {
        "name": "my_domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my_domain",
        "status": "Active",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0,
        "createdTime": "2020-10-12T16:51:18.039000-04:00"
    }
}
```

如果您使用自己管理的 KMS 密钥，请在 `--encryption-key` 参数中添加其 Amazon 资源名称 (ARN)。

```
aws codeartifact create-domain --domain my_domain --encryption-key arn:aws:kms:us-
west-2:111122223333:key/your-kms-key
```

输出中会显示 JSON 格式的数据，并包含有关新域的详细信息。

```
{
    "domain": {
        "name": "my_domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my_domain",
        "status": "Active",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0,
        "createdTime": "2020-10-12T16:51:18.039000-04:00"
    }
}
```

创建带标签的域

要创建带标签的域，请在 `create-domain` 命令中添加 `--tags` 参数。

```
aws codeartifact create-domain --domain my_domain --tags key=k1,value=v1
 key=k2,value=v2
```

删除域

您可以使用 CodeArtifact 控制台或 AWS Command Line Interface (AWS CLI) 删除域。

主题

- 有关域删除的限制
- 删除域（控制台）
- 删除域 (AWS CLI)

有关域删除的限制

通常，无法删除包含存储库的域。在删除域之前，必须先删除其存储库。有关更多信息，请参阅 删除存储库。

但是，如果 CodeArtifact 无法再访问该域的 KMS 密钥，即使该域仍包含存储库，也可以将其删除。如果您删除域的 KMS 密钥或撤销 CodeArtifact 用于访问该密钥的 KMS 授权，就会出现这种情况。在这种状态下，无法访问域中的存储库或存储在其中的程序包。当 CodeArtifact 无法访问域名的 KMS 密钥时，也无法列出和删除存储库。因此，当无法访问域的 KMS 密钥时，删除域不会检查该域是否包含存储库。

> **Note**
>
> 当仍包含存储库的域名被删除时， CodeArtifact 将在 15 分钟内异步删除存储库。删除域后，存储库仍将在 CodeArtifact 控制台和`list-repositories`命令输出中可见，直到自动清理存储库为止。

删除域（控制台）

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在导航窗格中，选择域，然后选择要删除的域。
3. 选择 Delete （删除)。

删除域 (AWS CLI)

使用 delete-domain 命令来删除域。

```
aws codeartifact delete-domain --domain my_domain --domain-owner 111122223333
```

输出中会显示 JSON 格式的数据，并包含有关已删除域的详细信息。

```
{
    "domain": {
        "name": "my_domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my_domain",
        "status": "Active",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0,
        "createdTime": "2020-10-12T16:51:18.039000-04:00"
```

```
    }
}
```

域策略

CodeArtifact 支持使用基于资源的权限来控制访问权限。基于资源的权限让您可以指定能够访问资源的用户，以及这些用户可以对该资源执行的操作。默认情况下，只有拥有域的 AWS 账户才能创建和访问域中的存储库。您可以对域应用策略文档，让其他 IAM 主体可以访问该域。

有关更多信息，请参阅策略和权限以及基于身份的策略和基于资源的策略。

主题

- 启用对域的跨账户访问
- 域策略示例
- 域名策略示例 AWS Organizations
- 设置域策略
- 读取域策略
- 删除域策略

启用对域的跨账户访问

资源策略是 JSON 格式的文本文件。该文件必须指定主体 (actor)、一个或多个操作以及效果（`Allow` 或 `Deny`）。要在由另一个账户拥有的域中创建存储库，必须向主体授予域资源的 `CreateRepository` 权限。

例如，以下资源策略向账户 `123456789012` 授予在域中创建存储库的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:CreateRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:root"
            },
```

```
            "Resource": "*"
        }
    ]
}
```

要允许创建带有标签的存储库，必须包括 codeartifact:TagResource 权限。这也将让账户可以向域及其中的所有存储库添加标签。

由于策略仅针对策略所附加到的域的操作进行评估，因此无需指定资源。由于资源是隐含的，因此可以将 Resource 设置为 *。

要访问另一个账户拥有的域中的程序包，必须向主体授予域资源的 GetAuthorizationToken 权限。授予权限后，域所有者可以控制哪些账户可以读取域中存储库的内容。

例如，以下资源策略向账户 123456789012 授予为域中任何存储库检索身份验证令牌的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:GetAuthorizationToken"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 除了域的 GetAuthorizationToken 权限外，还必须向想要从存储库端点提取程序包的主体授予存储库资源的 ReadFromRepository 权限。同样，除了 GetAuthorizationToken 权限之外，还必须向想要将程序包发布到存储库端点的主体授予 PublishPackageVersion 权限。
>
> 有关 ReadFromRepository 和 PublishPackageVersion 权限的更多信息，请参阅存储库策略。

域策略示例

当多个账户使用一个域时，应向这些账户授予一组基本权限，让他们可以充分使用该域。以下资源策略列出了一组允许充分使用域的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "BasicDomainPolicy",
            "Action": [
                "codeartifact:GetDomainPermissionsPolicy",
                "codeartifact:ListRepositoriesInDomain",
                "codeartifact:GetAuthorizationToken",
                "codeartifact:DescribeDomain",
                "codeartifact:CreateRepository"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            }
        }
    ]
}
```

> **Note**
>
> 如果一个域及其所有存储库归一个账户所有，并且只需要从该账户中使用，则无需创建域策略。

域名策略示例 AWS Organizations

您可以使用`aws:PrincipalOrgID`条件密钥向组织中的所有账户授予对 CodeArtifact 域的访问权限，如下所示。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "DomainPolicyForOrganization",
```

```
        "Effect": "Allow",
        "Principal": "*",
        "Action": [
             "codeartifact:GetDomainPermissionsPolicy",
             "codeartifact:ListRepositoriesInDomain",
             "codeartifact:GetAuthorizationToken",
             "codeartifact:DescribeDomain",
             "codeartifact:CreateRepository"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": { "aws:PrincipalOrgID":["o-xxxxxxxxxxx"]}
        }
    }
}
```

有关使用 aws:PrincipalOrgID 条件键的更多信息，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。

设置域策略

您可以使用 put-domain-permissions-policy 命令将策略附加到域。

```
aws codeartifact put-domain-permissions-policy --domain my_domain --domain-
owner 111122223333 \
 --policy-document file://</PATH/TO/policy.json>
```

如果调用 put-domains-permissions-policy，则在评估权限时会忽略域中的资源策略。这样可以确保域的所有者不会将自己锁定在域之外，因而使他们无法更新资源策略。

> **Note**
>
> 您不能向其他 AWS 账户授予使用资源策略更新域上资源策略的权限，因为调用时会忽略资源策略 put-domain-permissions-policy。

示例输出：

```
{
    "policy": {
```

```
        "resourceArn": "arn:aws:codeartifact:region-id:111122223333:domain/my_domain",
        "document": "{ ...policy document content...}",
        "revision": "MQlyyTQRASRU3HB58gBtSDHXG7Q3hvxxxxxxx="
    }
}
```

命令输出包含域资源的 Amazon 资源名称 (ARN)、策略文档的完整内容以及修订标识符。可以使用 `--policy-revision` 选项将修订标识符传递给 `put-domain-permissions-policy`。这样可以确保覆盖文档的已知修订版，而不是由另一个作者设置的较新版本。

读取域策略

要读取策略文档的现有版本，请使用 `get-domain-permissions-policy` 命令。要格式化输出来提高可读性，请将 `--output` 和 `--query policy.document` 与 Python `json.tool` 模块一起使用，如下所示。

```
aws codeartifact get-domain-permissions-policy --domain my_domain --domain-
owner 111122223333 \
   --output text --query policy.document | python -m json.tool
```

示例输出：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "BasicDomainPolicy",
            "Action": [
                "codeartifact:GetDomainPermissionsPolicy",
                "codeartifact:ListRepositoriesInDomain",
                "codeartifact:GetAuthorizationToken",
                "codeartifact:CreateRepository"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            }
        }
    ]
}
```

删除域策略

使用 `delete-domain-permissions-policy` 命令从域中删除策略。

```
aws codeartifact delete-domain-permissions-policy --domain my_domain --domain-
owner 111122223333
```

输出的格式与 `get-domain-permissions-policy` 和 `delete-domain-permissions-policy` 命令的输出格式相同。

将域名标记为 CodeArtifact

标签是与 AWS 资源关联的键/值对。您可以在中将标签应用于您的域名 CodeArtifact。有关 CodeArtifact 资源标记、用例、标签键和值限制以及支持的资源类型的信息，请参阅标记资源。

您可以使用 CLI 在创建域时指定标签。您可以使用控制台或 CLI 来添加或删除标签，以及更新域中标签的值。您最多可以为每个域添加 50 个标签。

主题

- 标记域 (CLI)
- 标记域（控制台）

标记域 (CLI)

您可以使用 CLI 来管理域标签。

主题

- 向域添加标签 (CLI)
- 查看域的标签 (CLI)
- 编辑域的标签 (CLI)
- 从域中删除标签 (CLI)

向域添加标签 (CLI)

您可以使用控制台或 AWS CLI 来标记域名。

要在创建域时为其添加标签，请参阅创建存储库。

在这些步骤中，我们假设您已安装最新版本的 AWS CLI 或已更新到当前版本。有关更多信息，请参阅安装 AWS Command Line Interface。

在终端或命令行运行 tag-resource 命令，指定要为其添加标签的域的 Amazon 资源名称 (ARN)，以及要添加的标签的键/值。

> **Note**
>
> 要获取域的 ARN，请运行 `describe-domain` 命令：
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

您可以为域添加多个标签。例如，要在名为 *my_domain* 的域中添加两个标签，一个标签键名为 *key1*，标签值为 *value1*，另一个标签键名为 *key2*，标签值为 *value2*，请执行以下命令：

```
aws codeartifact tag-resource --resource-arn arn:aws:codeartifact:us-
west-2:123456789012:domain/my_domain --tags key=key1,value=value1 key=key2,value=value2
```

如果成功，此命令没有输出。

查看域的标签 (CLI)

请按照以下步骤使用 AWS CLI 来查看域的 AWS 标签。如果尚未添加标签，则返回的列表为空。

在终端或命令行中，使用域的 Amazon 资源名称 (ARN) 运行 list-tags-for-resource 命令。

> **Note**
>
> 要获取域的 ARN，请运行 `describe-domain` 命令：
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

例如，要查看名为 *my_domain* 且 ARN 值为 `arn:aws:codeartifact:us-west-2:123456789012:domain/my_domain` 的域的标签键和标签值列表，请运行以下命令：

```
aws codeartifact list-tags-for-resource --resource-arn arn:aws:codeartifact:us-
west-2:123456789012:domain/my_domain
```

如果成功，该命令返回类似以下内容的信息：

```
{
    "tags": {
        "key1": "value1",
        "key2": "value2"
    }
}
```

编辑域的标签 (CLI)

按照以下步骤 AWS CLI 使用编辑域的标签。您可以更改现有键的值或添加另一个键。您还可以从域中删除标签，如下一节所示。

在终端或命令行运行 tag-resource 命令，指定要为其更新标签的域的 ARN 并指定标签键和标签值：

> **Note**
>
> 要获取域的 ARN，请运行 `describe-domain` 命令：
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

```
aws codeartifact tag-resource --resource-arn arn:aws:codeartifact:us-
west-2:123456789012:domain/my_domain --tags key=key1,value=newvalue1
```

如果成功，此命令没有输出。

从域中删除标签 (CLI)

按照以下步骤使用从 AWS CLI 网域中移除标签。

> **Note**
>
> 如果您删除某个域，则会从已删除域中移除所有标签关联。您无需在删除域之前移除标签。

在终端或命令行运行 untag-resource 命令，指定要从中删除标签的域的 ARN 以及要删除的标签的标签键。

> **Note**
>
> 要获取域的 ARN，请运行 `describe-domain` 命令：
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

例如，要在名为 *mydomain* 且标签键为 *key1* 和 *key2* 的域中删除多个标签，请运行以下命令：

```
aws codeartifact untag-resource --resource-arn arn:aws:codeartifact:us-
west-2:123456789012:domain/my_domain --tag-keys key1 key2
```

如果成功，此命令没有输出。移除标签后，您可以使用 `list-tags-for-resource` 命令查看存储库中剩余的标签。

标记域（控制台）

您可以使用控制台或 CLI 来标记资源。

主题

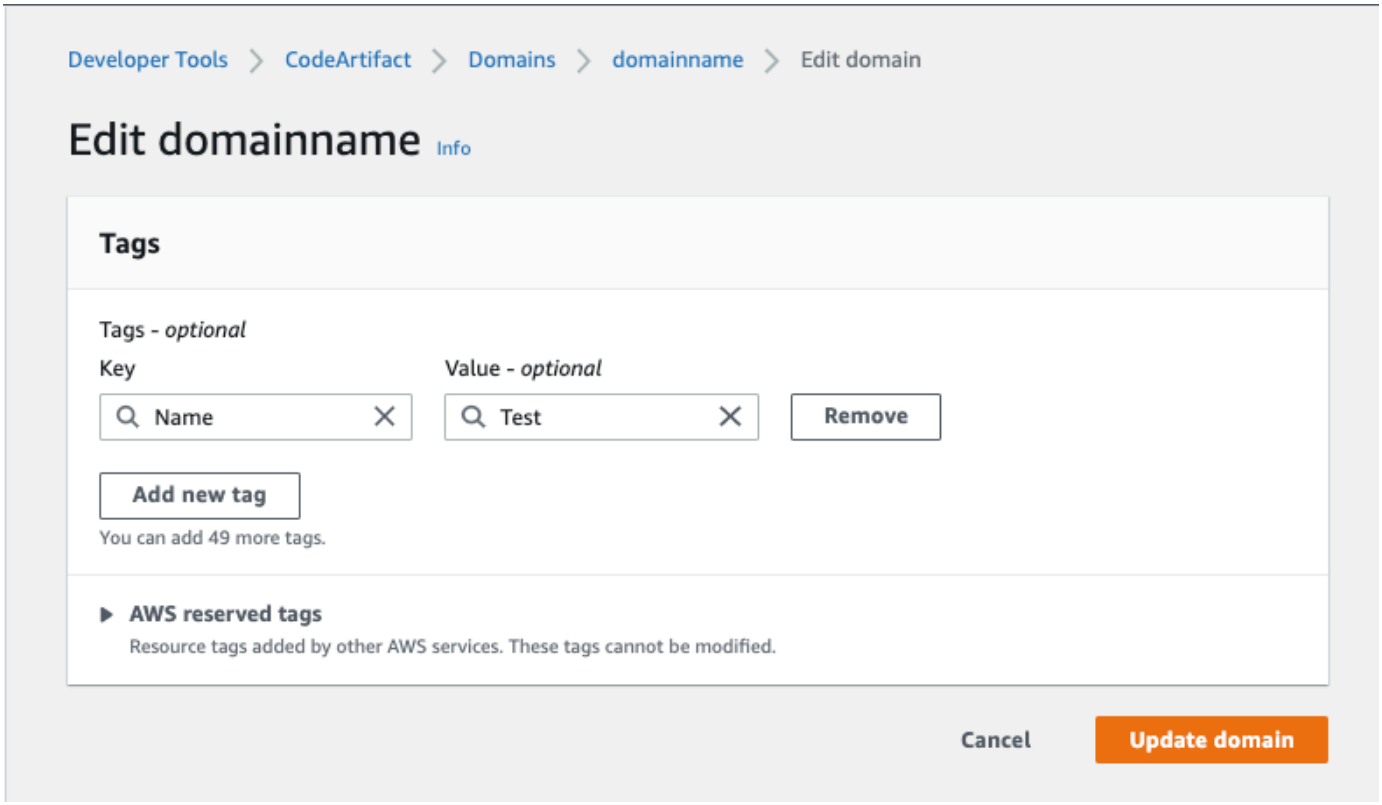
- 为域添加标签（控制台）
- 查看域的标签（控制台）
- 编辑域的标签（控制台）
- 从域中删除标签（控制台）

为域添加标签（控制台）

您可以使用控制台向现有域添加标签。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在域页面上，选择要添加标签的域。
3. 展开详细信息部分。
4. 在域标签下面，如果域没有标签，请选择添加域标签，如果有标签，则选择查看和编辑域标签。
5. 选择添加新标签。

6. 在键和值字段中，输入要添加的每个标签的文本。（值字段为可选项。）例如，在键中，输入 **Name**。在值中，输入 **Test**。



7. （可选）选择添加标签以添加多行并输入多个标签。
8. 选择更新域。

查看域的标签（控制台）

您可以使用控制台列出现有域的标签。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在域页面上，选择要查看标签的域。
3. 展开详细信息部分。
4. 在域标签下面，选择查看和编辑域标签。

> **Note**
>
> 如果未向此域添加任何标签，则控制台会显示添加域标签。

编辑域的标签（控制台）

您可以使用控制台来编辑已添加到域的标签。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在域页面上，选择要更新标签的域。
3. 展开详细信息部分。
4. 在域标签下面，选择查看和编辑域标签。

> **Note**
>
> 如果未向此域添加任何标签，则控制台会显示添加域标签。

5. 在键和值字段中，根据需要更新每个字段的值。例如，对于 **Name** 键，在值中，将 **Test** 更改为 **Prod**。
6. 选择更新域。

从域中删除标签（控制台）

您可以使用控制台从域中删除标签。

1. 打开 AWS CodeArtifact 控制台，网址为 https://console.aws.amazon.com/codesuite/codeartifact/home。
2. 在域页面上，选择要删除标签的域。
3. 展开详细信息部分。
4. 在域标签下面，选择查看和编辑域标签。

> **Note**
>
> 如果未向此域添加任何标签，则控制台会显示添加域标签。

5. 接下来，对于您要删除的每个标签的键和值，选择删除。
6. 选择更新域。

将 CodeArtifact 与 npm 结合使用

这些主题说明了如何在 CodeArtifact 中使用 npm（Node.js 程序包管理器）。

> **Note**
>
> CodeArtifact 支持 `node v4.9.1` 和更高版本以及 `npm v5.0.0` 和更高版本。

主题

在 CodeArtifact 中配置和使用 npm

在 CodeArtifact 中创建存储库后，您可以使用 npm 客户端来安装和发布程序包。使用存储库端点和授权令牌配置 npm 的推荐方法是使用 `aws codeartifact login` 命令。也可以手动配置 npm。

目录

使用 login 命令配置 npm

使用 `aws codeartifact login` 命令提取用于 npm 的凭证。

> **Note**
>
> 如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。

> **Important**
>
> 如果您使用的是 npm 10.x 或更高版本，则必须使用 AWS CLI 版本 2.9.5 或更高版本才能成功运行 `aws codeartifact login` 命令。

```
aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

此命令对您的 ~/.npmrc 文件进行以下更改：

- 使用 AWS 凭证从 CodeArtifact 获取授权令牌后添加授权令牌。
- 将 npm 注册表设置为通过 `--repository` 选项指定的存储库。
- 对于 npm 6 及更低版本：添加 `"always-auth=true"`，为每个 npm 命令发送授权令牌。

调用 `login` 后的默认授权期为 12 小时，且必须调用 `login` 来定期刷新令牌。有关使用 `login` 命令创建的授权令牌的更多信息，请参阅使用 `login` 命令创建的令牌。

不使用 login 命令配置 npm

您可以手动更新 npm 配置，而不使用 `aws codeartifact login` 命令来对 CodeArtifact 存储库配置 npm。

不使用 login 命令配置 npm

1. 在命令行中，提取 CodeArtifact 授权令牌并将其存储在环境变量中。npm 将使用此令牌向您的 CodeArtifact 存储库进行身份验证。

> **Note**
>
> 以下命令适用于 macOS 或 Linux 计算机。有关在 Windows 计算机上配置环境变量的信息，请参阅使用环境变量传递身份验证令牌。

```
CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text`
```

2. 通过运行以下命令来获取 CodeArtifact 存储库的端点。您的存储库端点用于将 npm 指向您的存储库来安装或发布程序包。

   - 将 *my_domain* 替换为您的 CodeArtifact 域名。
   - 将 *111122223333* 替换为域所有者的 AWS 账户 ID。如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。
   - 将 *my_repo* 替换为您的 CodeArtifact 存储库名称。

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-
owner 111122223333 --repository my_repo --format npm
```

以下 URL 是一个示例存储库端点。

```
https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my_repo/
```

> **Important**
>
> 注册 URL 必须以正斜杠 (/) 结尾。否则，您无法连接到存储库。

3. 使用 `npm config set` 命令将注册表设置为您的 CodeArtifact 存储库。将 URL 替换为上一步中的存储库端点 URL。

```
npm config set
 registry=https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
npm/my_repo/
```

4. 使用 npm config set 命令将您的授权令牌添加到 npm 配置。

```
npm config set //my_domain-111122223333.d.codeartifact.region.amazonaws.com/
npm/my_repo/:_authToken=$CODEARTIFACT_AUTH_TOKEN
```

对于 npm 6 或更低版本：要让 npm 始终将身份验证令牌传递给 CodeArtifact（甚至是 GET 请求），请使用 npm config set 来设置 always-auth 配置变量。

```
npm config set //my_domain-111122223333.d.codeartifact.region.amazonaws.com/
npm/my_repo/:always-auth=true
```

示例 npm 配置文件 (**.npmrc**)

以下是按照上述说明设置 CodeArtifact 注册表端点、添加身份验证令牌和配置 always-auth 后的示例 .npmrc 文件。

```
registry=https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my-
cli-repo/
//my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/
my_repo/:_authToken=eyJ2ZX...
//my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my_repo/:always-
auth=true
```

运行 npm 命令

配置 npm 客户端后，您可以运行 npm 命令。假设您的存储库或其中一个上游存储库中存在程序包，则可以使用 npm install 来安装。例如，使用以下命令来安装 lodash 程序包。

```
npm install lodash
```

使用以下命令将新的 npm 程序包发布到 CodeArtifact 存储库。

```
npm publish
```

有关如何创建 npm 程序包的信息，请参阅 npm 文档网站上的创建 Node.js 模块。有关 CodeArtifact 支持的 npm 命令列表，请参阅 npm 命令支持。

验证 npm 身份验证和授权

调用 npm ping 命令是验证以下项的一种方式：

- 您已正确配置凭证，所以可以向 CodeArtifact 存储库进行身份验证。
- 授权配置为您授予 ReadFromRepository 权限。

成功调用 npm ping 后的输出如下所示。

```
$ npm -d ping
npm info it worked if it ends with ok
npm info using npm@6.4.1
npm info using node@v9.5.0
npm info attempt registry request try #1 at 4:30:59 PM
npm http request GET https://<domain>.d.codeartifact.us-west-2.amazonaws.com/npm/
shared/-/ping?write=true
npm http 200 https:///npm/shared/-/ping?write=true
Ping success: {}
npm timing npm Completed in 716ms
npm info ok
```

-d 选项会让 npm 输出额外的调试信息，包括存储库 URL。通过这些信息，可以轻松确认 npm 已配置为使用您期望的存储库。

改回默认 npm 注册表

使用 CodeArtifact 配置 npm 会将 npm 注册表设置为指定的 CodeArtifact 存储库。当您完成与 CodeArtifact 的连接后，可以运行以下命令将 npm 注册表设置回其默认注册表。

```
npm config set registry https://registry.npmjs.com/
```

解决 npm 8.x 或更高版本中安装缓慢的问题

在 npm 版本 8.x 及更高版本中存在一个已知问题，也即，如果向程序包存储库发出请求，并且存储库将客户端重定向到 Amazon S3 而不是直接流式传输资产，则 npm 客户端可能对于每个依赖项会挂起几分钟。

由于 CodeArtifact 存储库旨在始终将请求重定向到 Amazon S3，因此有时会出现此问题，这会导致因 npm 安装时间长而致使构建时间变长。这种行为的实例将以进度条的形式出现，显示达几分钟。

要避免此问题，请在 npm cli 命令中使用 `--no-progress` 或 `progress=false` 标志，如以下示例所示。

```
npm install lodash --no-progress
```

在 CodeArtifact 中配置和使用 Yarn

创建存储库后，您可以使用 Yarn 客户端来管理 npm 程序包。

> **Note**
>
> `Yarn 1.X` 会读取和使用 npm 配置文件 (.npmrc) 中的信息，但 `Yarn 2.X` 不会。`Yarn 2.X` 的配置必须在.yarnrc.yml 文件中定义。

目录

使用 `aws codeartifact login` 命令配置 Yarn 1.X

对于 `Yarn 1.X`，您可以使用 `aws codeartifact login` 命令在 CodeArtifact 中配置 Yarn。`login` 命令将使用您的 CodeArtifact 存储库端点信息和凭证配置 ~/.npmrc 文件。在 `Yarn 1.X` 中，`yarn` 命令使用 ~/.npmrc 文件中的配置信息。

使用 login 命令配置 **`Yarn 1.X`**

1. 如果尚未配置要与 AWS CLI 一起使用的 AWS 凭证，请先配置凭证，如 开始使用 CodeArtifact中所述。
2. 要成功运行 `aws codeartifact login` 命令，必须安装 npm。有关安装说明，请参阅 npm 文档中的下载和安装 Node.js 和 npm。
3. 使用 `aws codeartifact login` 命令提取 CodeArtifact 凭证并配置您的 ~/.npmrc 文件。

   - 将 *`my_domain`* 替换为您的 CodeArtifact 域名。
   - 将 *`111122223333`* 替换为域所有者的 AWS 账户 ID。如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。
   - 将 *`my_repo`* 替换为您的 CodeArtifact 存储库名称。

```
aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

login 命令对 ~/.npmrc 文件进行以下更改：

- 使用 AWS 凭证从 CodeArtifact 获取授权令牌后添加授权令牌。
- 将 npm 注册表设置为通过 --repository 选项指定的存储库。
- 对于 npm 6 及更低版本：添加 "always-auth=true"，为每个 npm 命令发送授权令牌。

调用 login 后的默认授权期为 12 小时，且必须调用 login 来定期刷新令牌。有关使用 login 命令创建的授权令牌的更多信息，请参阅使用 login 命令创建的令牌。

4. 对于 npm 7.X 和 8.X，必须将 always-auth=true 添加到 ~/.npmrc 文件中才能使用 Yarn。

   - 在文本编辑器中打开 ~/.npmrc 文件并在新行中添加 always-auth=true。

您可以使用 yarn config list 命令来检查 Yarn 是否使用了正确的配置。运行命令后，请检查 info npm config 部分中的值。其内容看起来类似于以下代码段。

```
info npm config
{
  registry: 'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/
my_repo/',
  '//my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/
my_repo/:_authToken': 'eyJ2ZXI...',
  'always-auth': true
}
```

使用 **yarn config set** 命令配置 Yarn 2.X

以下程序详细说明了如何使用 yarn config set 命令从命令行更新 .yarnrc.yml 配置，从而配置 Yarn 2.X。

从命令行更新 **yarnrc.yml** 配置

1. 如果尚未配置要与 AWS CLI 一起使用的 AWS 凭证，请先配置凭证，如 开始使用 CodeArtifact中所述。

2. 使用 `aws codeartifact get-repository-endpoint` 命令来获取 CodeArtifact 存储库的端点。

   - 将 *my_domain* 替换为您的 CodeArtifact 域名。
   - 将 *111122223333* 替换为域所有者的 AWS 账户 ID。如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。
   - 将 *my_repo* 替换为您的 CodeArtifact 存储库名称。

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-
owner 111122223333 --repository my_repo --format npm
```

3. 使用存储库端点更新 .yarnrc.yml 文件中的 `npmRegistryServer` 值。

```
yarn config set npmRegistryServer
 "https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/npm/my_repo/"
```

4. 提取 CodeArtifact 授权令牌并将其存储在环境变量中。

> **Note**
>
> 以下命令适用于 macOS 或 Linux 计算机。有关在 Windows 计算机上配置环境变量的信息，请参阅使用环境变量传递身份验证令牌。

   - 将 *my_domain* 替换为您的 CodeArtifact 域名。
   - 将 *111122223333* 替换为域所有者的 AWS 账户 ID。如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。
   - 将 *my_repo* 替换为您的 CodeArtifact 存储库名称。

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text`
```

5. 使用 `yarn config set` 命令将 CodeArtifact 身份验证令牌添加到 .yarnrc.yml 文件中。将以下命令中的 URL 替换为步骤 2 中的存储库端点 URL。

```
yarn config set
 'npmRegistries["https://my_domain-
111122223333.d.codeartifact.region.amazonaws.com/npm/my_repo/"].npmAuthToken'
 "${CODEARTIFACT_AUTH_TOKEN}"
```

6. 使用 `yarn config set` 命令将 npmAlwaysAuth 的值设置为 `true`。将以下命令中的 URL 替换为步骤 2 中的存储库端点 URL。

```
yarn config set
 'npmRegistries["https://my_domain-
111122223333.d.codeartifact.region.amazonaws.com/npm/my_repo/"].npmAlwaysAuth'
 "true"
```

配置完成后，.yarnrc.yml 配置文件的内容应与以下代码段类似。

```
npmRegistries:
  "https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my_repo/":
    npmAlwaysAuth: true
    npmAuthToken: eyJ2ZXI...

npmRegistryServer: "https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/npm/my_repo/"
```

您也可以使用 `yarn config` 命令来检查 npmRegistries 和 npmRegistryServer 的值。

npm 命令支持

以下各节总结了 CodeArtifact 存储库支持的 npm 命令以及不支持的特定命令。

目录

与存储库进行交互的受支持命令

本节列出了 npm 命令，其中 npm 客户端向其配置的注册表发出一个或多个请求（例如，使用 `npm config set registry`）。这些命令已经过验证，在针对 CodeArtifact 存储库调用时可以正常运行。

命令	描述
bugs	尝试猜测程序包的错误跟踪器 URL 的位置，然后尝试打开它。
ci	从零开始安装一个项目。
deprecate	弃用程序包的某个版本。
dist-tag	修改程序包分发标签。
docs	尝试猜测程序包的文档 URL 的位置，然后尝试使用 `--browser` 配置参数打开它。
doctor	运行一组检查来确保您的 npm 安装具有管理 JavaScript 程序包所需的内容。
install	安装程序包。
install-ci-test	从零开始安装一个项目并运行测试。别名：`npm cit`。此命令运行 `npm ci`，然后立即运行 `npm test`。
install-test	安装程序包并运行测试。运行 `npm install`，然后立即运行 `npm test`。
outdated	检查已配置的注册表，查看当前是否有任何已安装的程序包已过时。
ping	ping 已配置或给定的 npm 注册表并验证身份验证。
publish	将程序包版本发布到注册表。

命令	描述
update	猜测程序包存储库 URL 的位置，然后尝试使用 `--browser` 配置参数来打开它。
view	显示程序包元数据。可用于输出元数据属性。

支持的客户端命令

这些命令不需要与存储库进行任何直接交互，因此 CodeArtifact 无需执行任何操作即可支持它们。

命令	描述
build	构建程序包。
cache	操作程序包缓存。
completion	在所有 npm 命令中启用制表符自动完成功能。
config	更新用户和全局 `npmrc` 文件的内容。
dedupe	搜索本地程序包树，并尝试通过将依赖项进一步向上移动来简化结构，这样多个依赖程序包就可以更有效地共享依赖项。
edit	编辑已安装的程序包。在当前工作目录中选择一个依赖项，然后在默认编辑器中打开程序包文件夹。
explore	浏览已安装的程序包。在指定的已安装程序包目录中创建一个子 Shell。如果指定了命令，则该命令将在该子 Shell 中运行，然后立即终止。
help	获取有关 npm 的帮助。
help-search	搜索 npm 帮助文档。
init	创建 `package.json` 文件。

命令	描述
link	创建指向程序包文件夹的符号链接。
ls	列出已安装的程序包。
pack	将程序包打包成 tarball。
prefix	显示前缀。除非也指定了 `-g`，否则这是包含 `package.json` 文件的最接近父目录。
prune	删除未在父程序包依赖项列表中列出的程序包。
rebuild	对匹配的文件夹运行 `npm build` 命令。
restart	运行程序包的停止、重启和启动脚本以及相关的前置和后置脚本。
根	将有效的 `node_modules` 文件夹输出到标准输出。
run-script	运行任意程序包脚本。
shrinkwrap	锁定依赖项版本以供发布。
uninstall	卸载程序包。

不受支持的命令

CodeArtifact 存储库不支持这些 npm 命令。

命令	描述	注意
access	设置已发布程序包的访问级别。	CodeArtifact 使用的权限模型与公有 npmjs 存储库不同。
adduser	添加注册表用户账户	CodeArtifact 使用与公有 npmjs 存储库不同的用户模型。

命令	描述	注意
audit	运行安全审核。	CodeArtifact 目前不提供安全漏洞数据。
hook	管理 npm 钩子，包括添加、删除、列出和更新。	CodeArtifact 目前不支持任何类型的变更通知机制。
login	对用户进行身份验证。这是 `npm adduser` 的一个别名。	CodeArtifact 使用与公有 npmjs 存储库不同的身份验证模型。有关信息，请参阅使用 npm 进行身份验证。
logout	注销注册表。	CodeArtifact 使用与公有 npmjs 存储库不同的身份验证模型。无法从 CodeArtifact 存储库中注销，但是身份验证令牌会在其可配置的到期时间后过期。默认令牌持续时间为 12 小时。
owner	管理程序包所有者。	CodeArtifact 使用的权限模型与公有 npmjs 存储库不同。
profile	更改注册表配置文件的设置。	CodeArtifact 使用与公有 npmjs 存储库不同的用户模型。
搜索	在注册表中搜索与搜索词匹配的程序包。	CodeArtifact 通过 list-packages 命令支持有限的搜索功能。
star	标记您喜欢的程序包。	CodeArtifact 目前不支持任何类型的收藏机制。
stars	查看已标记为收藏的程序包。	CodeArtifact 目前不支持任何类型的收藏机制。

命令	描述	注意
team	管理组织团队和团队成员资格。	CodeArtifact 使用与公有 npmjs 存储库不同的用户和组成员资格模型。有关信息，请参阅《IAM 用户指南》中的身份（用户和角色）。
token	管理您的身份验证令牌。	CodeArtifact 使用不同的模型来获取身份验证令牌。有关信息，请参阅使用 npm 进行身份验证。
unpublish	从注册表中删除程序包。	CodeArtifact 不支持使用 npm 客户端从存储库中删除程序包版本。可以使用 delete-package-version 命令。
whoami	显示 npm 用户名。	CodeArtifact 使用与公有 npmjs 存储库不同的用户模型。

npm 标签处理

npm 注册表支持标签，这些标签是程序包版本的字符串别名。您可以使用标签来提供别名而不是提供版本号。例如，您可能有一个包含多个开发流的项目，并且为每个流使用不同的标签（例如 `stable`、`beta`、`dev`、`canary`）。有关更多信息，请参阅 npm 网站上的 dist-tag。

默认情况下，npm 使用 `latest` 标签来标识程序包的当前版本。`npm install` *`pkg`*（不带 `@`*`version`* 或 `@`*`tag`* 说明符）会安装最新的标签。通常，项目仅对稳定版本使用最新标签。对于不稳定版本或预发行版本使用其他标签。

使用 npm 客户端编辑标签

三个 `npm dist-tag` 命令（`add`、`rm` 和 `ls`）在 CodeArtifact 存储库中的功能与在默认 npm 注册表中的功能相同。

npm 标签和 CopyPackageVersions API

当您使用 CopyPackageVersions API 来复制 npm 程序包版本时，所有作为该版本别名的标签都会复制到目标存储库。如果要复制的版本的标签也存在于目标存储库中，则复制操作会将目标存储库中的标签值设置为与源存储库中的值相匹配。

例如，假设存储库 S 和存储库 D 都包含 web-helper 程序包的一个版本，其最新标签设置如下表所示。

存储库	程序包名称	程序包标签
S	web-helper	最新（版本 1.0.1 的别名）
D	web-helper	最新（版本 1.0.0 的别名）

调用 CopyPackageVersions 来将 web-helper 1.0.1 从 S 复制到 D。操作完成后，存储库 D 中 web-helper 的 latest 标签别名为 1.0.1，而不是 1.0.0。

如果在复制后需要更改标签，请使用 npm dist-tag 命令直接在目标存储库中修改标签。有关 CopyPackageVersions API 的更多信息，请参阅在存储库之间复制程序包。

npm 标签和上游存储库

当 npm 请求程序包的标签且上游存储库也存在该程序包的版本时，CodeArtifact 会合并标签，然后再将标签返回给客户端。例如，名为 R 的存储库有一个名为 U 的上游存储库。下表显示了两个存储库中都存在的名为 web-helper 的程序包的标签。

存储库	程序包名称	程序包标签
R	web-helper	最新（版本 1.0.0 的别名）
U	web-helper	alpha（版本 1.0.1 的别名）

在这种情况下，当 npm 客户端从存储库 R 获取 web-helper 程序包的标签时，它会同时收到最新标签和 alpha 标签。标签指向的版本不会改变。

当上游和下游存储库中的同一个程序包存在相同的标签时，CodeArtifact 会使用上游存储库中存在的标签。例如，假设 webhelper 上的标签已修改为如下所示。

存储库	程序包名称	程序包标签
R	web-helper	最新（版本 1.0.0 的别名）
U	web-helper	最新（版本 1.0.1 的别名）

在这种情况下，当 npm 客户端从存储库 R 提取程序包 web-helper 的标签时，因为上游存储库中存在最新标签，所以该标签将作为版本 1.0.1 的别名。这样就可以运行 `npm update`，在上游存储库中更轻松地使用尚不存在于下游存储库中的新程序包版本。

在下游存储库中发布程序包的新版本时，在上游存储库中使用该标签会出现问题。例如，假设程序包 web-helper 的最新标签在 R 和 U 中相同。

存储库	程序包名称	程序包标签
R	web-helper	最新（版本 1.0.1 的别名）
U	web-helper	最新（版本 1.0.1 的别名）

当版本 1.0.2 发布到 R 时，npm 会将最新标签更新为 1.0.2。

存储库	程序包名称	程序包标签
R	web-helper	最新（版本 1.0.2 的别名）
U	web-helper	最新（版本 1.0.1 的别名）

但是，因为 U 中的最新值是 1.0.1，所以 npm 客户端永远看不到这个标签值。发布 1.0.2 后立即对存储库 R 运行 `npm install` 会安装 1.0.1，而不是安装刚刚发布的版本。要安装最新发布的版本，必须指定确切的程序包版本，如下所示。

```
npm install web-helper@1.0.2
```

支持兼容 npm 的程序包管理器

这些其他程序包管理器与 CodeArtifact 兼容，可以处理 npm 程序包格式和 npm 通信协议：

- pnpm 程序包管理器。确认可与 CodeArtifact 配合使用的最新版本是 3.3.4，已于 2019 年 5 月 18 日发布。
- Yarn 程序包管理器。确认可与 CodeArtifact 配合使用的最新版本是 1.21.1，已于 2019 年 12 月 11 日发布。

> **Note**
>
> 我们建议将 Yarn 2.x 与 CodeArtifact 配合使用。Yarn 1.x 没有 HTTP 重试功能，这意味着该版本更容易出现间歇性服务故障，进而导致出现 500 级状态代码或错误。无法为 Yarn 1.x 配置不同的重试策略，但已在 Yarn 2.x 中添加了重试策略。您可以使用 Yarn 1.x，但可能需要在构建脚本中添加更高级别的重试机制。例如，在一个循环中多次执行 Yarn 命令，以便在下载程序包失败时会进行重试。

将 CodeArtifact 与 Python 结合使用

这些主题说明了如何在 CodeArtifact 中使用 `pip`（Python 程序包管理器）和 `twine`（Python 程序包发布实用程序）。

主题

在 CodeArtifact 中配置和使用 pip

pip 是 Python 程序包的程序包安装程序。要使用 pip 从 CodeArtifact 存储库中安装 Python 程序包，您必须先使用 CodeArtifact 存储库信息和凭证来配置 pip 客户端。

pip 只能用于安装 Python 程序包。要发布 Python 程序包，您可以使用 twine。有关更多信息，请参阅在 CodeArtifact 中配置和使用 twine。

使用 `login` 命令配置 pip

首先，配置要与 AWS CLI 一起使用的 AWS 凭证，如 开始使用 CodeArtifact中所述。然后，使用 CodeArtifact `login` 命令来提取凭证并使用这些凭证来配置 `pip`。

> **Note**
>
> 如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。

要配置 pip，请运行以下命令。

```
aws codeartifact login --tool pip --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

`login` 使用您的 AWS 凭证从 CodeArtifact 提取授权令牌。`login` 命令会编辑 `~/.config/pip/pip.conf`，将 `index-url` 设置为 `--repository` 选项指定的存储库，从而将 `pip` 配置为与 CodeArtifact 一起使用。

调用 `login` 后的默认授权期为 12 小时，且必须调用 `login` 来定期刷新令牌。有关使用 `login` 命令创建的授权令牌的更多信息，请参阅使用 `login` 命令创建的令牌。

不使用 login 命令配置 pip

如果您无法使用 `login` 命令来配置 `pip`，则可以使用 `pip config`。

1. 使用 AWS CLI 来提取新的授权令牌。

   > **Note**
   >
   > 如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。

   ```
   CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
   domain my_domain --domain-owner 111122223333 --query authorizationToken --output
    text`
   ```

2. 使用 `pip config` 来设置 CodeArtifact 注册表 URL 和凭证。以下命令将仅更新当前的环境配置文件。要更新系统范围的配置文件，请将 `site` 替换为 `global`。

   ```
   pip config set site.index-url https://aws:
   $CODEARTIFACT_AUTH_TOKEN@my_domain-
   111122223333.d.codeartifact.region.amazonaws.com/pypi/my_repo/simple/
   ```

> **Important**
>
> 注册 URL 必须以正斜杠 (/) 结尾。否则，您无法连接到存储库。

示例 pip 配置文件

以下是设置 CodeArtifact 注册表 URL 和凭证后的 `pip.conf` 文件示例。

```
[global]
index-url = https://aws:eyJ2ZX...@my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/pypi/my_repo/simple/
```

运行 pip

要运行 pip 命令，必须使用 CodeArtifact 配置 pip。有关更多信息，请参阅以下文档。

1. 按照 设置 AWS CodeArtifact 部分中的步骤来配置您的 AWS 账户、工具和权限。
2. 按照 在 CodeArtifact 中配置和使用 twine 中的步骤配置 twine。

假设您的存储库或其中一个上游存储库中存在程序包，则可以使用 pip install 来安装。例如，使用以下命令来安装 requests 程序包。

```
pip install requests
```

使用 -i 选项暂时恢复为从 https://pypi.org，而不是从 CodeArtifact 存储库安装程序包。

```
pip install -i https://pypi.org/simple requests
```

在 CodeArtifact 中配置和使用 twine

twine 是一款适用于 Python 程序包的程序包发布实用程序。要使用 twine 将 Python 程序包发布到 CodeArtifact 存储库，您必须先使用 CodeArtifact 存储库信息和凭证来配置 twine。

twine 只能用于发布 Python 程序包。要安装 Python 程序包，您可以使用 pip。有关更多信息，请参阅在 CodeArtifact 中配置和使用 pip。

使用 **login** 命令配置 twine

首先，配置要与 AWS CLI 一起使用的 AWS 凭证，如 开始使用 CodeArtifact中所述。然后，使用 CodeArtifact login 命令来提取凭证并使用这些凭证来配置 twine。

> **Note**
>
> 如果您要访问您拥有的域中的存储库，则无需包括 --domain-owner。有关更多信息，请参阅跨账户域。

要配置 twine，请运行以下命令。

```
aws codeartifact login --tool twine --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

login 使用您的 AWS 凭证从 CodeArtifact 提取授权令牌。login 命令会编辑 ~/.pypirc，添加 --repository 选项指定的存储库和凭证，从而将 twine 配置为与 CodeArtifact 一起使用。

调用 login 后的默认授权期为 12 小时，且必须调用 login 来定期刷新令牌。有关使用 login 命令创建的授权令牌的更多信息，请参阅使用 login 命令创建的令牌。

不使用 login 命令配置 twine

如果无法使用 login 命令来配置 twine，则可以使用 ~/.pypirc 文件或环境变量。要使用 ~/.pypirc 文件，请在文件添加以下条目。密码必须是 get-authorization-token API 获取的身份验证令牌。

```
[distutils]
index-servers =
 codeartifact
[codeartifact]
repository = https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
pypi/my_repo/
password = auth-token
username = aws
```

要使用环境变量，请执行以下操作：

> **Note**
>
> 如果您要访问您拥有的域中的存储库，则无需包括 --domain-owner。有关更多信息，请参阅跨账户域。

```
export TWINE_USERNAME=aws
export TWINE_PASSWORD=`aws codeartifact get-authorization-token --domain my_domain --
domain-owner 111122223333 --query authorizationToken --output text`
export TWINE_REPOSITORY_URL=`aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format pypi --query
 repositoryEndpoint --output text`
```

运行 twine

在使用 twine 来发布 Python 程序包资产之前，必须先配置 CodeArtifact 权限和资源。

1. 按照 设置 AWS CodeArtifact 部分中的步骤来配置您的 AWS 账户、工具和权限。
2. 按照 使用 `login` 命令配置 twine 或 不使用 `login` 命令配置 twine 中的步骤配置 twine。

配置 twine 之后，就可以运行 `twine` 命令。使用以下命令来发布 Python 程序包资产。

```
twine upload --repository codeartifact mypackage-1.0.tgz
```

有关如何构建和打包 Python 应用程序的信息，请参阅 Python 打包权威机构网站上的生成分发存档。

Python 程序包名称规范化

在存储程序包之前，CodeArtifact 会对程序包名称进行规范化，这意味着 CodeArtifact 中的程序包名称可能与发布程序包时提供的名称不同。

对于 Python 程序包，在执行规范化时，程序包名称转为小写，所有 `.`、`-` 和 `_` 字符都替换为单个 `-` 字符。因此，程序包名称 `pigeon_cli` 和 `pigeon.cli` 会规范化并存储为 `pigeon-cli`。pip 和 twine 可以使用非规范化名称，但在 CodeArtifact CLI 或 API 请求（例如 `list-package-versions`）和 ARN 中必须使用规范化名称。有关 Python 程序包名称规范化的更多信息，请参阅 Python 文档中的 PEP 503。

Python 兼容性

CodeArtifact 不支持 PyPI 的 `XML-RPC` 或 `JSON` API。

CodeArtifact 支持 PyPI 的 `Legacy` API，但 `simple` API 除外。虽然 CodeArtifact 不支持 `/simple/` API 端点，但支持 `/simple/<project>/` 端点。

有关更多信息，请参阅 Python 打包权威机构的 GitHub 存储库中的以下内容。

- XML-RPC API
- JSON API
- Legacy API

pip 命令支持

以下各节总结了 CodeArtifact 存储库支持的 pip 命令以及不支持的特定命令。

主题

与存储库进行交互的受支持命令

本节列出了 `pip` 命令，其中 `pip` 客户端向其配置的注册表发出一个或多个请求。这些命令已经过验证，在针对 CodeArtifact 存储库调用时可以正常运行。

命令	描述
install	安装程序包。
download	下载程序包。

CodeArtifact 不会实施 `pip search`。如果您已经对 CodeArtifact 存储库配置了 `pip`，则运行 `pip search` 时会搜索并显示来自 PyPI 的程序包。

支持的客户端命令

这些命令不需要与存储库进行任何直接交互，因此 CodeArtifact 无需执行任何操作即可支持它们。

命令	描述
uninstall	卸载程序包。
freeze	按要求格式输出已安装的程序包。
list	列出已安装程序包。
show	显示有关已安装程序包的信息。
check	验证已安装的程序包是否具有兼容的依赖项。

命令	描述
config	管理本地和全局配置。
wheel	根据您的要求构建 wheel。
hash	计算程序包存档的哈希值。
completion	协助完成命令。
debug	显示对调试有用的信息。
help	显示命令的帮助。

从上游和外部连接请求 Python 程序包

从 pypi.org 导入 Python 程序包版本时，CodeArtifact 会导入该程序包版本中的所有资产。虽然大多数 Python 程序包都只包含少量资产，但有些程序包包含 100 多个资产，通常用于支持多种硬件架构和 Python 解释器。

对于现有程序包版本，将新资产发布到 pypi.org 是一种常见的做法。例如，在发布新版本的 Python 时，一些项目会发布新资产。使用 `pip install` 从 CodeArtifact 安装 Python 程序包时，保留在 CodeArtifact 存储库中的程序包版本会更新，以反映来自 pypi.org 的最新资产集。

同样，如果上游 CodeArtifact 存储库中的某个程序包版本有新的可用资产，而这些资产不存在于当前 CodeArtifact 存储库中，则它们将在运行 `pip install` 时保留在当前存储库中。

已撤销的程序包版本

pypi.org 中的某些程序包版本标记为已撤销，这会向程序包安装程序（例如 pip）表明，除非该版本是唯一与版本说明符匹配的版本（使用 == 或 ===），否则不应安装该版本。有关更多信息，请参阅 PEP_592。

如果 CodeArtifact 中的程序包版本最初是从外部连接提取到 pypi.org，那么当您从 CodeArtifact 存储库安装程序包版本时，CodeArtifact 会确保从 pypi.org 提取已更新的程序包版本撤销元数据。

如何知道某个程序包版本是否已撤销

要检查某个程序包版本是否已在 CodeArtifact 中撤销，您可以尝试使用 `pip install packageName===packageVersion` 来安装。如果程序包版本已撤销，您会收到一条与以下内容类似的警告消息：

```
WARNING: The candidate selected for download or install is a yanked version
```

要查看程序包版本是否在 pypi.org 中撤销，您可以访问该程序包版本的 pypi.org 列表，网址为 `https://pypi.org/project/packageName/packageVersion/`。

对私有程序包设置撤销状态

CodeArtifact 不支持为直接发布到 CodeArtifact 存储库的程序包设置撤销元数据。

为什么 CodeArtifact 未提取程序包版本的最新撤销元数据或资产？

通常，CodeArtifact 会确保在从 CodeArtifact 存储库提取 Python 程序包版本时，撤销元数据是最新的，在 pypi.org 上具有最新值。此外，程序包版本中的资产列表还会根据 pypi.org 和任何上游 CodeArtifact 存储库中的最新设置进行更新。无论您是第一次安装程序包版本且 CodeArtifact 将其从 pypi.org 导入到您的 CodeArtifact 存储库，还是之前已安装该程序包，都是如此。但在某些情况下，程序包管理器客户端（例如 pip）不会从 pypi.org 或上游存储库提取最新的撤销元数据。相反，CodeArtifact 将返回已存储在存储库中的数据。本节说明了发生这种情况的三种方式：

上游配置：如果使用 disassociate-external-connection 从存储库或其上游移除与 pypi.org 的外部连接，则不再从 pypi.org 刷新撤销元数据。同样，如果您移除上游存储库，则已移除的存储库中和已移除存储库上游的资产将不再可供当前存储库使用。如果您使用 CodeArtifact 程序包来源控制来阻止提取特定程序包的新版本，则情况也是如此，设置 `upstream=BLOCK` 会阻止刷新撤销元数据。

程序包版本状态：如果您将程序包版本的状态设置为除 `Published` 或 `Unlisted` 之外的任何其它状态，则不会刷新程序包版本的撤销元数据和资产。同样，如果您正在提取特定的程序包版本（例如 `torch 2.0.1`），而上游存储库中存在相同的程序包版本，但状态不是 `Published` 或 `Unlisted`，这也将阻止撤销元数据和资产从上游存储库传播到当前存储库。这是因为其它程序包版本状态表明这些版本不打算再在任何存储库中使用。

直接发布：如果您将特定的程序包版本直接发布到 CodeArtifact 存储库中，这将阻止从其上游存储库和 pypi.org 刷新程序包版本的撤销元数据和资产。例如，假设您使用 Web 浏览器从程序包版本 `torch 2.0.1` 下载资产（例如 `torch-2.0.1-cp311-none-macosx_11_0_arm64.whl`），然后使用 twine 以名称 `torch 2.0.1` 将其发布到您的 CodeArtifact 存储库。CodeArtifact 跟踪到程序包版本

是通过直接发布到存储库，而不是通过与 pypi.org 或上游存储库的外部连接来进入域。在这种情况下，CodeArtifact 不会使撤销元数据与上游存储库或 pypi.org 保持同步。如果您将 `torch 2.0.1` 发布到上游存储库，情况也是如此，因为存在程序包版本，所以会阻止将撤销元数据和资产传播到上游图更下方的存储库。

将 CodeArtifact 与 Maven 结合使用

许多不同的语言（包括 Java、Kotlin、Scala 和 Clojure）都使用 Maven 存储库格式。许多不同的构建工具（包括 Maven、Gradle、Scala SBT、Apache Ivy 和 Leiningen）都支持 CodeArtifact。

我们已经测试并确认了以下版本与 CodeArtifact 兼容：

- 最新的 Maven 版本：3.6.3。
- 最新的 Gradle 版本：6.4.1。我们还测试了 5.5.1。
- 最新的 Clojure 版本：1.11.1。我们还测试了其他版本。

主题

在 Gradle 中使用 CodeArtifact

按照使用环境变量传递身份验证令牌中所述，将 CodeArtifact 身份验证令牌存入环境变量后，请按照以下说明使用来自 CodeArtifact 存储库的 Maven 程序包，以及将新程序包发布到 CodeArtifact 存储库。

主题

提取依赖项

要在 Gradle 构建中提取来自 CodeArtifact 的依赖项，请遵循以下程序。

在 Gradle 构建中提取来自 CodeArtifact 的依赖项

1. 如果还没有身份验证令牌，请按照使用环境变量传递身份验证令牌中的步骤创建 CodeArtifact 身份验证令牌并将其存储在环境变量中。
2. 在项目 `build.gradle` 文件中的 `repositories` 部分添加一个 `maven` 部分。

```
maven {
         url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
         credentials {
             username "aws"
             password System.env.CODEARTIFACT_AUTH_TOKEN
         }
}
```

前面示例中的 `url` 是 CodeArtifact 存储库的端点。Gradle 使用端点来连接到您的存储库。在示例中，`my_domain` 是域的名称，`111122223333` 是域所有者的 ID，`my_repo` 是存储库的名称。您可以使用 `get-repository-endpoint` AWS CLI 命令检索存储库的端点。

例如，如果在名为 *my_domain* 的域中有一个名为 *my_repo* 的存储库，则命令如下所示：

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-
owner 111122223333 --repository my_repo --format maven
```

`get-repository-endpoint` 命令会返回存储库端点：

```
url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
```

前面示例中的 `credentials` 对象包括您在步骤 1 中创建的 CodeArtifact 身份验证令牌，Gradle 使用该令牌对 CodeArtifact 进行身份验证。

3. （可选）- 要使用 CodeArtifact 存储库作为项目依赖项的唯一来源，请从 `build.gradle` 的 `repositories` 中移除任何其他部分。如果您有多个存储库，Gradle 会按照列出的顺序在每个存储库中搜索依赖项。
4. 配置存储库后，您可以使用标准 Gradle 语法在 `dependencies` 部分添加项目依赖项。

```
dependencies {
    implementation 'com.google.guava:guava:27.1-jre'
    implementation 'commons-cli:commons-cli:1.4'
    testImplementation 'org.testng:testng:6.14.3'
}
```

提取插件

默认情况下，Gradle 会解析来自公有 Gradle 插件门户的插件。要从 CodeArtifact 存储库提取插件，请按以下步骤操作。

从 CodeArtifact 存储库提取插件

1. 如果还没有身份验证令牌，请按照使用环境变量传递身份验证令牌中的步骤创建 CodeArtifact 身份验证令牌并将其存储在环境变量中。
2. 在 `settings.gradle` 文件中添加一个 `pluginManagement` 块。`pluginManagement` 块必须出现在 `settings.gradle` 中的任何其他语句之前，请参阅以下代码段：

```
pluginManagement {
    repositories {
        maven {
            name 'my_repo'
            url
 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
                password System.env.CODEARTIFACT_AUTH_TOKEN
            }
        }
    }
}
```

这将确保 Gradle 会解析来自指定存储库的插件。存储库必须有一个与 Gradle 插件门户网站（例如 `gradle-plugins-store`）有外部连接的上游存储库，以便构建版本可以使用常用的 Gradle 插件。有关更多信息，请参阅 Gradle 文档。

发布构件

本节介绍如何将使用 Gradle 构建的 Java 库发布到 CodeArtifact 存储库。

首先，将 maven-publish 插件添加到项目 build.gradle 文件的 plugins 部分。

```
plugins {
    id 'java-library'
    id 'maven-publish'
}
```

接下来，在项目 build.gradle 文件中添加一个 publishing 部分。

```
publishing {
    publications {
        mavenJava(MavenPublication) {
            groupId = 'group-id'
            artifactId = 'artifact-id'
            version = 'version'
            from components.java
        }
    }
    repositories {
        maven {
            url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username "aws"
                password System.env.CODEARTIFACT_AUTH_TOKEN
            }
        }
    }
}
```

maven-publish 插件根据 publishing 部分中指定的 groupId、artifactId 和 version 生成 POM 文件。

对 build.gradle 作出的这些更改完成后，运行以下命令来构建项目并将其上传到存储库。

```
./gradlew publish
```

使用 list-package-versions 来确认程序包已成功发布。

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333
 --repository my_repo --format maven\
  --namespace com.company.framework --package my-package-name
```

示例输出：

```
{
    "format": "maven",
    "namespace": "com.company.framework",
    "package": "example",
    "versions": [
        {
            "version": "1.0",
            "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
            "status": "Published"
        }
    ]
}
```

有关更多信息，请参阅 Gradle 网站上的以下主题：

- 构建 Java 库
- 将项目作为模块发布

在 IntelliJ IDEA 中运行 Gradle 构建

您可以在 IntelliJ IDEA 中运行 Gradle 构建，从 CodeArtifact 提取依赖项。要使用 CodeArtifact 进行身份验证，您必须向 Gradle 提供 CodeArtifact 授权令牌。提供身份验证令牌的方法有三种。

- 方法 1：将身份验证令牌存储在 `gradle.properties` 中。如果您能够覆盖 `gradle.properties` 文件或添加内容，请使用此方法。
- 方法 2：将身份验证令牌存储在单独的文件中。如果您不想修改 `gradle.properties` 文件，请使用此方法。
- 方法 3：通过在 `build.gradle` 中以内联脚本形式运行 `aws`，为每次运行生成新的身份验证令牌。如果您希望 Gradle 脚本在每次运行时都提取一个新令牌，请使用此方法。令牌不会存储在文件系统中。

Token stored in gradle.properties

方法 1：将身份验证令牌存储在 **gradle.properties** 中

> **Note**
>
> 该示例显示了位于 GRADLE_USER_HOME 中的 gradle.properties 文件。

1. 使用以下代码段来更新 build.gradle 文件：

```
repositories {
    maven {
            url
 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username "aws"
                password "$codeartifactToken"
            }
    }
}
```

2. 要从 CodeArtifact 提取插件，请在 settings.gradle 文件中添加一个 pluginManagement 块。pluginManagement 块必须出现在 settings.gradle 中的任何其他语句之前。

```
pluginManagement {
    repositories {
        maven {
            name 'my_repo'
            url
 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
                password "$codeartifactToken"
            }
        }
    }
}
```

3. 提取 CodeArtifact 身份验证令牌：

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text --profile profile-name`
```

4. 将身份验证令牌写入 gradle.properties 文件中：

```
echo "codeartifactToken=$CODEARTIFACT_AUTH_TOKEN" > ~/.gradle/gradle.properties
```

Token stored in separate file

方法 2：将身份验证令牌存储在单独的文件中

1. 使用以下代码段来更新 build.gradle 文件：

```
def props = new Properties()
file("file").withInputStream { props.load(it) }

repositories {

    maven {
             url
 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
             credentials {
                 username "aws"
                 password props.getProperty("codeartifactToken")
             }
    }
}
```

2. 要从 CodeArtifact 提取插件，请在 settings.gradle 文件中添加一个 pluginManagement 块。pluginManagement 块必须出现在 settings.gradle 中的任何其他语句之前。

```
pluginManagement {
    def props = new Properties()
    file("file").withInputStream { props.load(it) }
    repositories {
        maven {
            name 'my_repo'
```

```
            url
 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
                password props.getProperty("codeartifactToken")
            }
        }
    }
}
```

3. 提取 CodeArtifact 身份验证令牌：

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text --profile profile-name`
```

4. 将身份验证令牌写入在 build.gradle 文件中指定的文件：

```
echo "codeartifactToken=$CODEARTIFACT_AUTH_TOKEN" > file
```

Token generated for each run in build.gradle

方法 3：通过在 **build.gradle** 中以内联脚本形式运行 **aws**，为每次运行生成新的身份验证令牌

1. 使用以下代码段来更新 build.gradle 文件：

```
def codeartifactToken = "aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text --profile profile-name".execute().text
    repositories {
        maven {
            url
 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username "aws"
                password codeartifactToken
            }
        }
    }
```

2. 要从 CodeArtifact 提取插件，请在 `settings.gradle` 文件中添加一个 `pluginManagement` 块。`pluginManagement` 块必须出现在 `settings.gradle` 中的任何其他语句之前。

```
pluginManagement {
    def codeartifactToken = "aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text --profile profile-name".execute().text
    repositories {
        maven {
            name 'my_repo'
            url
 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
                password codeartifactToken
            }
        }
    }
}
```

在 mvn 中使用 CodeArtifact

您可以使用 mvn 命令来执行 Maven 构建。本节说明如何配置 mvn 来使用 CodeArtifact 存储库。

主题

提取依赖项

要将 mvn 配置为从 CodeArtifact 存储库提取依赖项，必须编辑 Maven 配置文件 `settings.xml`，并选择性地编辑项目 POM。

1. 如果还没有这样做，请按使用环境变量传递身份验证令牌中所述在环境变量中创建并存储 CodeArtifact 身份验证令牌，以设置对 CodeArtifact 存储库的身份验证。
2. 在 settings.xml 中（该文件通常位于 ~/.m2/settings.xml），添加一个引用 CODEARTIFACT_AUTH_TOKEN 环境变量的 <servers> 部分，以便 Maven 可在 HTTP 请求中传递令牌。

```
<settings>
...
    <servers>
        <server>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
...
</settings>
```

3. 在 <repository> 元素中添加 CodeArtifact 存储库的 URL 端点。您可以在 settings.xml 或项目的 POM 文件中执行此操作。

   您可以使用 get-repository-endpoint AWS CLI 命令检索存储库的端点。

   例如，如果在名为 *my_domain* 的域中有一个名为 *my_repo* 的存储库，则命令如下所示：

```
aws codeartifact get-repository-endpoint --domain my_domain --repository my_repo --
format maven
```

   get-repository-endpoint 命令会返回存储库端点：

```
url 'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_repo/'
```

   按如下方式将存储库端点添加到 settings.xml。

```
<settings>
...
    <profiles>
        <profile>
            <id>default</id>
```

```
            <repositories>
                <repository>
                    <id>codeartifact</id>
                    <url>https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/</url>
                </repository>
            </repositories>
        </profile>
    </profiles>
    <activeProfiles>
        <activeProfile>default</activeProfile>
    </activeProfiles>
    ...
</settings>
```

或者，您可以将 <repositories> 部分添加到项目 POM 文件中，以便仅对该项目使用 CodeArtifact。

```
<project>
...
    <repositories>
        <repository>
            <id>codeartifact</id>
            <name>codeartifact</name>
            <url>https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/</url>
        </repository>
    </repositories>
...
</project>
```

Important

可以在 <id> 元素中使用任何值，但必须与 <server> 和 <repository> 元素中的值相同。这样就可以在发送到 CodeArtifact 的请求中包括指定的凭证。

更改这些配置后，就可以构建项目了。

```
mvn compile
```

Maven 会记录下载到控制台的所有依赖项的完整 URL。

```
[INFO] ------------------< com.example.example:myapp >------------------
[INFO] Building myapp 1.0
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.pom
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.pom (11 kB at 3.9 kB/s)
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom (68 kB at 123
 kB/s)
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.jar
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.jar (54 kB at 134 kB/s)
```

发布构件

要使用 mvn 将 Maven 构件发布到 CodeArtifact 存储库，还必须编辑 `~/.m2/settings.xml` 和项目 POM。

1. 如果还没有这样做，请按使用环境变量传递身份验证令牌中所述在环境变量中创建并存储 CodeArtifact 身份验证令牌，以设置对 CodeArtifact 存储库的身份验证。
2. 在 `settings.xml` 中添加一个引用 `CODEARTIFACT_AUTH_TOKEN` 环境变量的 `<servers>` 部分，以便 Maven 可在 HTTP 请求中传递令牌。

```
<settings>
...
    <servers>
        <server>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
...
```

```
</settings>
```

3. 将 <distributionManagement> 部分添加到项目的 pom.xml。

```
<project>
...
     <distributionManagement>
         <repository>
             <id>codeartifact</id>
             <name>codeartifact</name>
             <url>https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/</url>
         </repository>
     </distributionManagement>
...
</project>
```

更改这些配置后，就可以开始构建项目并将项目发布到指定的存储库。

```
mvn deploy
```

使用 list-package-versions 来确认程序包已成功发布。

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333
 --repository my_repo --format maven \
  --namespace com.company.framework --package my-package-name
```

示例输出：

```
{
    "defaultDisplayVersion": null,
    "format": "maven",
    "namespace": "com.company.framework",
    "package": "my-package-name",
    "versions": [
        {
            "version": "1.0",
            "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
            "status": "Published"
        }
    ]
```

```
}
```

发布第三方构件

您可以使用 `mvn deploy:deploy-file` 将第三方 Maven 构件发布到 CodeArtifact 存储库。这对于想要发布构件且只有 JAR 文件且无权访问程序包源代码或 POM 文件的用户很有用。

`mvn deploy:deploy-file` 命令会根据命令行中传递的信息生成 POM 文件。

发布第三方 Maven 构件

1. 如果还没有这样做，请按使用环境变量传递身份验证令牌中所述在环境变量中创建并存储 CodeArtifact 身份验证令牌，以设置对 CodeArtifact 存储库的身份验证。
2. 创建 `~/.m2/settings.xml` 文件并输入以下内容：

```
<settings>
    <servers>
        <server>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
</settings>
```

3. 运行 `mvn deploy:deploy-file` 命令：

```
mvn deploy:deploy-file -DgroupId=commons-cli          \
-DartifactId=commons-cli       \
-Dversion=1.4                  \
-Dfile=./commons-cli-1.4.jar   \
-Dpackaging=jar                \
-DrepositoryId=codeartifact    \
-Durl=https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/repo-name/
```

> **Note**
>
> 上面的示例会发布 commons-cli 1.4。修改 groupID、artifactID、version 和 file 参数来发布另一个 JAR。

这些说明基于 Apache Maven 文档的将第三方 JAR 部署到远程存储库的指导中的示例。

限制为仅从 CodeArtifact 存储库下载 Maven 依赖项

如果无法从已配置的存储库提取程序包，则默认情况下，mvn 命令会从 Maven Central 提取程序包。将 mirrors 元素添加到 settings.xml，使 mvn 始终使用您的 CodeArtifact 存储库。

```
<settings>
  ...
    <mirrors>
      <mirror>
        <id>central-mirror</id>
        <name>CodeArtifact Maven Central mirror</name>
        <url>https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_repo/</url>
        <mirrorOf>central</mirrorOf>
      </mirror>
    </mirrors>
  ...
</settings>
```

如果添加 mirrors 元素，则在 settings.xml 或 pom.xml 中还必须有一个 pluginRepository 元素。以下示例从 CodeArtifact 存储库提取应用程序依赖项和 Maven 插件。

```
<settings>
...
  <profiles>
    <profile>
      <pluginRepositories>
        <pluginRepository>
          <id>codeartifact</id>
          <name>CodeArtifact Plugins</name>
          <url>https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_repo/</url>
```

```
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
...
</settings>
```

以下示例从 CodeArtifact 存储库提取应用程序依赖项，从 Maven Central 提取 Maven 插件。

```
<profiles>
   <profile>
     <id>default</id>
     ...
     <pluginRepositories>
       <pluginRepository>
         <id>central-plugins</id>
         <name>Central Plugins</name>
         <url>https://repo.maven.apache.org/maven2/</url>
         <releases>
             <enabled>true</enabled>
         </releases>
         <snapshots>
             <enabled>true</enabled>
         </snapshots>
       </pluginRepository>
     </pluginRepositories>
   ....
   </profile>
 </profiles>
```

Apache Maven 项目信息

有关 Maven 的更多信息，请参阅 Apache Maven Project 网站上的以下主题：

- 设置多个存储库
- 设置参考

- 分配管理
- 配置文件

通过 deps.edn 来使用 CodeArtifact

您可以使用包含 `clj` 命令的 `deps.edn` 来管理 Clojure 项目的依赖项。本节说明如何配置 `deps.edn` 来使用 CodeArtifact 存储库。

主题

- 提取依赖项
- 发布构件

提取依赖项

要将 `Clojure` 配置为从 CodeArtifact 存储库提取依赖项，必须编辑 Maven 配置文件 `settings.xml`。

1. 在 `settings.xml` 中，添加一个引用 `CODEARTIFACT_AUTH_TOKEN` 环境变量的 `<servers>` 部分，以便 Clojure 可在 HTTP 请求中传递令牌。

> **Note**
>
> Clojure 预计 settings.xml 文件位于 `~/.m2/settings.xml`。如果该文件在其他地方，则在此位置创建该文件。

```
<settings>
...
    <servers>
        <server>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
...
</settings>
```

2. 如果您的项目还没有 POM xml 文件，请使用 `clj -Spom` 生成该文件。
3. 在您的 `deps.edn` 配置文件中，添加与 Maven `settings.xml` 中的服务器 ID 相匹配的存储库。

```
:mvn/repos {
  "clojars" nil
  "central" nil
  "codeartifact" {:url "https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/"}
}
```

> **Note**
>
> - `tools.deps` 保证首先检查 `central` 和 `clojars` 存储库中是否有 Maven 库。之后，系统会检查 `deps.edn` 中列出的其他存储库。
> - 为了防止直接从 Clojars 和 Maven Central 下载，`central` 和 `clojars` 需要设置为 `nil`。

确保环境变量中有 CodeArtifact 身份验证令牌（请参见使用环境变量传递身份验证令牌）。在发生这些更改之后构建程序包时，`deps.edn` 将从 CodeArtifact 中提取依赖项。

发布构件

1. 更新 Maven 设置和 `deps.edn`，将 CodeArtifact 添加为 Maven 认可的服务器（请参见提取依赖项）。您可以使用诸如 deps-deploy 之类的工具将构件上传到 CodeArtifact。
2. 在 `build.clj` 中，添加一个 `deploy` 任务，将所需的构件上传到先前设置的 `codeartifact` 存储库。

```
(ns build
(:require [deps-deploy.deps-deploy :as dd]))

(defn deploy [_]
  (dd/deploy {:installer :remote
              :artifact "PATH_TO_JAR_FILE.jar"
              :pom-file "pom.xml" ;; pom containing artifact coordinates
              :repository "codeartifact"}))
```

3. 通过运行以下命令来发布构件：`clj -T:build deploy`

有关修改默认存储库的更多信息，请参阅《Clojure Deps 和 CLI 参考依据》中的修改默认存储库。

使用 curl 进行发布

本节介绍如何使用 HTTP 客户端 `curl` 将 Maven 构件发布到 CodeArtifact 存储库。如果您的环境中没有 Maven 客户端或想要安装 Maven 客户端，则使用 `curl` 发布构件会很有用。

使用 **`curl`** 发布 Maven 构件

1. 按照 使用环境变量传递身份验证令牌 中的步骤提取 CodeArtifact 授权令牌，然后返回到这些步骤。
2. 使用以下 `curl` 命令将 JAR 发布到 CodeArtifact 存储库：

   在此程序中的每个 `curl` 命令中，替换以下占位符：

   - 将 *`my_domain`* 替换为您的 CodeArtifact 域名。
   - 将 *`111122223333`* 替换为您的 CodeArtifact 域所有者的 ID。
   - 将 *`us-west-2`* 替换为您的 CodeArtifact 域所在的区域。
   - 将 *`my_repo`* 替换为您的 CodeArtifact 存储库名称。

   ```
   curl --request PUT https://my_domain-111122223333.d.codeartifact.us-
   west-2.amazonaws.com/maven/my_repo/com/mycompany/app/my-app/1.0/my-app-1.0.jar \
        --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/
   octet-stream" \
        --data-binary @my-app-1.0.jar
   ```

   > **Important**
   >
   > 必须在 `--data-binary` 参数的值前面加上一个 `@` 字符。将值放在引号中时，`@` 必须包含在引号内。

3. 使用以下 `curl` 命令将 POM 发布到 CodeArtifact 存储库：

   ```
   curl --request PUT https://my_domain-111122223333.d.codeartifact.us-
   west-2.amazonaws.com/maven/my_repo/com/mycompany/app/my-app/1.0/my-app-1.0.pom \
        --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/
   octet-stream" \
   ```

```
    --data-binary @my-app-1.0.pom
```

4. 此时，Maven 构件将位于您的 CodeArtifact 存储库中，状态为 Unfinished。为了能够使用程序包，程序包必须处于 Published 状态。您可以通过向程序包上传 maven-metadata.xml 文件来将程序包从 Unfinished 切换为 Published，或者调用 UpdatePackageVersionsStatus API 来更改状态。

   a. 选项 1：使用以下 curl 命令将 maven-metadata.xml 文件添加到您的程序包中：

   ```
   curl --request PUT
    https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
   maven/my_repo/com/mycompany/app/my-app/maven-metadata.xml \
        --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/
   octet-stream" \
        --data-binary @maven-metadata.xml
   ```

   以下是 maven-metadata.xml 文件的内容示例：

   ```
   <metadata modelVersion="1.1.0">
       <groupId>com.mycompany.app</groupId>
       <artifactId>my-app</artifactId>
       <versioning>
           <latest>1.0</latest>
           <release>1.0</release>
           <versions>
               <version>1.0</version>
           </versions>
           <lastUpdated>20200731090423</lastUpdated>
       </versioning>
   </metadata>
   ```

   b. 选项 2：使用 UpdatePackageVersionsStatus API 将程序包状态更新为 Published。

   ```
   aws codeartifact update-package-versions-status \
       --domain my_domain \
       --domain-owner 111122223333 \
       --repository my_repo \
       --format maven \
       --namespace com.mycompany.app \
       --package my-app \
       --versions 1.0 \
   ```

```
    --target-status Published
```

如果您只有构件的 JAR 文件，则可以使用 mvn 将可使用的程序包版本发布到 CodeArtifact 存储库。如果您无法访问构件的源代码或 POM，此方法会很有用。有关详细信息，请参阅发布第三方构件。

使用 Maven 校验和

将 Maven 构件发布到 AWS CodeArtifact 存储库时，使用与程序包中每个资产或文件关联的校验和来验证上传。资产的例子包括 jar、pom 和 war 文件。对于每个资产，Maven 构件都包含多个校验和文件，这些文件使用带有附加扩展名（例如 md5 或 sha1）的资产名称。例如，名为 my-maven-package.jar 的文件的校验和文件可能是 my-maven-package.jar.md5 和 my-maven-package.jar.sha1。

> **Note**
>
> Maven 使用术语“artifact”。在本指南中，Maven 程序包与 Maven 构件相同。有关更多信息，请参阅 AWS CodeArtifact 程序包。

校验和存储

CodeArtifact 不会将 Maven 校验和存储为资产。这意味着校验和不会作为单个资产出现在 ListPackageVersionAssets API 的输出中。相反，由 CodeArtifact 计算的校验和适用于采用所有受支持校验和类型的每种资产。例如，对 Maven 程序包版本 commons-lang:commons-lang 2.1 调用 ListPackageVersionAssets 的部分响应如下：

```
{
    "name": "commons-lang-2.1.jar",
    "size": 207723,
    "hashes": {
        "MD5": "51591549f1662a64543f08a1d4a0cf87",
        "SHA-1": "4763ecc9d78781c915c07eb03e90572c7ff04205",
        "SHA-256": "2ded7343dc8e57decd5e6302337139be020fdd885a2935925e8d575975e480b9",
        "SHA-512":
 "a312a5e33b17835f2e82e74ab52ab81f0dec01a7e72a2ba58bb76b6a197ffcd2bb410e341ef7b3720f3b595ce49fd
    }
},
{
    "name": "commons-lang-2.1.pom",
```

```
    "size": 9928,
    "hashes": {
        "MD5": "8e41bacdd69de9373c20326d231c8a5d",
        "SHA-1": "a34d992202615804c534953aba402de55d8ee47c",
        "SHA-256": "f1a709cd489f23498a0b6b3dfbfc0d21d4f15904791446dec7f8a58a7da5bd6a",
        "SHA-512":
 "1631ce8fe4101b6cde857f5b1db9b29b937f98ba445a60e76cc2b8f2a732ff24d19b91821a052c1b56b73325104e9
    }
},
        {
    "name": "maven-metadata.xml",
    "size": 121,
    "hashes": {
        "MD5": "11bb3d48d984f2f49cea1e150b6fa371",
        "SHA-1": "7ef872be17357751ce65cb907834b6c5769998db",
        "SHA-256": "d04d140362ea8989a824a518439246e7194e719557e8d701831b7f5a8228411c",
        "SHA-512":
 "001813a0333ce4b2a47cf44900470bc2265ae65123a8c6b5ac5f2859184608596baa4d8ee0696d0a497755dade0f6
    }
}
```

尽管校验和未存储为资产，但 Maven 客户端仍然可以在预期的位置发布和下载校验和。例如，如果 `commons-lang:commons-lang 2.1` 位于名为 `maven-repo` 的存储库中，则 JAR 文件的 SHA-256 校验和的 URL 路径将为：

```
/maven/maven-repo/commons-lang/commons-lang/2.1/commons-lang-2.1.jar.sha256
```

如果您使用通用 HTTP 客户端（例如 `curl`）将现有 Maven 程序包（例如，之前存储在 Amazon S3 中的程序包）上传到 CodeArtifact，则无需上传校验和。CodeArtifact 会将自动生成校验和。如果要确认已正确上传资产，可以使用 ListPackageVersionAssets API 操作将响应中的校验和与每项资产的原始校验和值进行比较。

发布期间校验和不匹配

除了资产和校验和，Maven 构件还包含一个 `maven-metadata.xml` 文件。Maven 程序包的正常发布顺序是先上传所有资产和校验和，然后再上传 `maven-metadata.xml`。例如，假设客户端配置为发布 SHA-256 校验和文件，则前面所述的 Maven 程序包版本 `commons-lang 2.1` 的发布顺序为：

```
PUT commons-lang-2.1.jar
PUT commons-lang-2.1.jar.sha256
PUT commons-lang-2.1.pom
```

```
PUT commons-lang-2.1.pom.sha256
PUT maven-metadata.xml
PUT maven-metadata.xml.sha256
```

上传资产的校验和文件（例如 JAR 文件）时，如果上传的校验和值与 CodeArtifact 计算的校验和值不匹配，则校验和上传请求会失败，并显示 400（错误请求）响应。如果相应的资产不存在，则请求将失败，并显示 404（未找到）响应。为避免出现此错误，您必须先上传资产，然后再上传校验和。

上传 `maven-metadata.xml` 后，CodeArtifact 通常会将 Maven 程序包版本的状态从 `Unfinished` 更改为 `Published`。如果检测到任何资产的校验和不匹配，CodeArtifact 会返回一个 400（错误请求）来响应 `maven-metadata.xml` 发布请求。此错误可能会导致客户端停止上传该程序包版本的文件。如果出现这种情况，并且 `maven-metadata.xml` 文件未上传，则无法下载已上传的程序包版本的任何资产。这是因为程序包版本的状态未设置为 `Published`，而是保持为 `Unfinished`。

即使 `maven-metadata.xml` 已上传且程序包版本状态已设置为 `Published`，CodeArtifact 仍允许向 Maven 程序包版本添加更多资产。在此状态下，上传不匹配的校验和文件的请求也会失败，并显示 400（错误请求）响应。但是，由于程序包版本状态已设置为 `Published`，因此您可以从程序包中下载任何资产，包括校验和文件上传失败的资产。在为校验和文件上传失败的资产下载校验和时，客户端收到的校验和值将是 CodeArtifact 根据上传的资产数据计算出的校验和值。

CodeArtifact 校验和比较区分大小写，CodeArtifact 计算的校验和采用小写格式。因此，如果上传校验和 `909FA780F76DA393E992A3D2D495F468`，则因为 CodeArtifact 不会将其视为等于 `909fa780f76da393e992a3d2d495f468`，所以校验和不匹配，上传失败。

在校验和不匹配情况下恢复

如果由于校验和不匹配导致校验和上传失败，请尝试以下方法之一进行恢复：

- 再次运行发布 Maven 构件的命令。如果因网络问题导致校验和文件损坏，此方法可能会起作用。如果这样可以解决网络问题，则校验和匹配且下载成功。
- 删除程序包版本，然后重新发布。有关更多信息，请参阅《AWS CodeArtifact API 参考》中的 DeletePackageVersions。

使用 Maven 快照

Maven 快照是 Maven 程序包的特殊版本，该版本引用了最新的生产分支代码。它是先于最终发布版本的开发版本。您可以通过附加到程序包版本的后缀 `SNAPSHOT` 来识别 Maven 程序包的快照版本。例如，版本 `1.1` 的快照是 `1.1-SNAPSHOT`。有关更多信息，请参阅 Apache Maven Project 网站上的什么是快照版本？。

AWS CodeArtifact 支持发布和使用 Maven 快照。使用基于时间的版本号的唯一快照是唯一受支持的快照。CodeArtifact 不支持由 Maven 2 客户端生成的非唯一快照。您可以将支持的 Maven 快照发布到任何 CodeArtifact 存储库。

主题

CodeArtifact 中的快照发布

AWS CodeArtifact 支持客户端（例如 mvn）在发布快照时使用的请求模式。因此，您无需详细了解 Maven 快照的发布方式，即可按照构建工具或程序包管理器的文档进行操作。如果您要进行更复杂的操作，本节详细介绍了 CodeArtifact 如何处理快照。

将 Maven 快照发布到 CodeArtifact 存储库时，其先前版本将保留在名为“构建”的新版本中。每次发布 Maven 快照时，都会创建一个新的构建版本。快照的所有先前版本都保留在其构建版本中。发布 Maven 快照时，其程序包版本状态将设置为 `Published`，包含先前版本的版本的构建的状态设置为 `Unlisted`。此行为仅适用于程序包版本使用 `-SNAPSHOT` 作为后缀的 Maven 程序包版本。

例如，名为 `com.mycompany.myapp:pkg-1` 的 maven 程序包的快照版本会上传到名为 `my-maven-repo` 的 CodeArtifact 存储库。快照版本是 `1.0-SNAPSHOT`。到目前为止，尚未发布 `com.mycompany.myapp:pkg-1` 的任何版本。首先，在以下路径发布初始构建的资产：

```
PUT maven/my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210728.194552-1.jar
PUT maven/my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210728.194552-1.pom
```

请注意，由发布快照版本的客户端生成时间戳 `20210728.194552-1`。

上传 .pom 和 .jar 文件后，存储库中唯一存在的 `com.mycompany.myapp:pkg-1` 版本是 `1.0-20210728.194552-1`。即使在前面的路径中指定的版本是 `1.0-SNAPSHOT`，也会发生这种情况。此时的程序包版本状态为 `Unfinished`。

```
aws codeartifact list-package-versions --domain my-domain --repository \
  my-maven-repo --package pkg-1 --namespace com.mycompany.myapp --format maven
{
    "versions": [
        {
            "version": "1.0-20210728.194552-1",
            "revision": "GipMW+599JmwTcTLaXo9YvDsVQ2bcrrk/02rWJhoKUU=",
            "status": "Unfinished"
        }
    ],
    "defaultDisplayVersion": null,
    "format": "maven",
    "package": "pkg-1",
    "namespace": "com.mycompany.myapp"
}
```

接下来，客户端上传程序包版本的 maven-metadata.xml 文件：

```
PUT my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/maven-metadata.xml
```

成功上传 maven-metadata.xml 文件后，CodeArtifact 会创建 1.0-SNAPSHOT 程序包版本并将 1.0-20210728.194552-1 版本设置为 Unlisted。

```
aws codeartifact list-package-versions --domain my-domain --repository \
  my-maven-repo --package pkg-1 --namespace com.mycompany.myapp --format maven
{
    "versions": [
        {
            "version": "1.0-20210728.194552-1",
            "revision": "GipMW+599JmwTcTLaXo9YvDsVQ2bcrrk/02rWJhoKUU=",
            "status": "Unlisted"
        },
        {
            "version": "1.0-SNAPSHOT",
            "revision": "tWu8n3IX5HR82vzVZQAxlwcvvA4U/+S80edWNAkil24=",
            "status": "Published"
        }
    ],
    "defaultDisplayVersion": "1.0-SNAPSHOT",
    "format": "maven",
    "package": "pkg-1",
    "namespace": "com.mycompany.myapp"
```

```
}
```

此时，可以在构建中使用快照版本 1.0-SNAPSHOT。虽然存储库 my-maven-repo 中有 com.mycompany.myapp:pkg-1 的两个版本，但它们都包含相同的资产。

```
aws codeartifact list-package-version-assets --domain my-domain --repository \
  my-maven-repo --format maven --namespace com.mycompany.myapp \
 --package pkg-1 --package-version 1.0-SNAPSHOT--query 'assets[*].name'
[
     "pkg-1-1.0-20210728.194552-1.jar",
     "pkg-1-1.0-20210728.194552-1.pom"
]
```

将 --package-version 参数更改为 1.0-20210728.194552-1，运行如前所示的相同 list-package-version-assets 命令会得到相同的输出。

在向存储库中添加 1.0-SNAPSHOT 的额外构建时，会为每个新构建创建一个新的 Unlisted 程序包版本。每次都会更新版本 1.0-SNAPSHOT 的资产，因此版本始终引用该版本的最新构建。通过上传新构建的 maven-metadata.xml 文件，开始将 1.0-SNAPSHOT 更新为包含最新资产。

使用快照版本

如果您请求快照，则会返回状态为 Published 的版本。这始终是 Maven 快照的最新版本。您也可以在 URL 路径中使用构建版本号（例如 1.0-20210728.194552-1）而不是快照版本（例如 1.0-SNAPSHOT）来请求快照的特定构建。要查看 Maven 快照的构建版本，请使用《CodeArtifact API 指南》中的 ListPackageVersions API，并将状态参数设置为 Unlisted。

删除快照版本

要删除 Maven 快照的所有构建版本，请使用 DeletePackageVersions API，指定要删除的版本。

使用 curl 进行快照发布

如果 Amazon Simple Storage Service (Amazon S3) 或另一个构件存储库产品中已经存储了您的快照版本，则可能需要将其重新发布到 AWS CodeArtifact。由于 CodeArtifact 采用特殊方式来支持 Maven 快照（请参见CodeArtifact 中的快照发布），因此使用通用 HTTP 客户端（例如 curl）来发布快照比按照使用 curl 进行发布中所述来发布 Maven 版本要复杂得多。请注意，如果您使用 mvn 或 gradle 等 Maven 客户端来构建和部署快照版本，则本节不相关。您需要遵循该客户端的文档进行操作。

发布快照版本涉及发布快照版本的一个或多个构建。在 CodeArtifact 中，如果一个快照版本有 n 个构建，则将会有 n + 1 个 CodeArtifact 版本：n 个构建版本的状态均为 Unlisted，还有一个快照版本

（最新发布的构建）的状态为 Published。快照版本（即版本字符串包含“-SNAPSHOT”的版本）包含一组与最新发布的构建相同的资产。使用 curl 创建此结构的最简单方法如下：

1. 使用 curl 发布所有构建的所有资产。
2. 使用 curl 发布最后一个构建（即带有最新日期时间戳的构建）的 maven-metadata.xml 文件。这将创建一个版本字符串中带有“-SNAPSHOT”且具有正确资产集的版本。
3. 使用 UpdatePackageVersionsStatus API 将所有非最新构建版本的状态设置为 Unlisted。

使用以下 curl 命令来发布程序包 com.mycompany.app:pkg-1 的快照版本 1.0-SNAPSHOT 的快照资产（例如 .jar 和 .pom 文件）：

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210729.171330-2.jar \
     --data-binary @pkg-1-1.0-20210728.194552-1.jar
```

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210729.171330-2.pom \
     --data-binary @pkg-1-1.0-20210728.194552-1.pom
```

使用这些示例时：

- 将 *my_domain* 替换为您的 CodeArtifact 域名。
- 将 *111122223333* 替换为您的 CodeArtifact 域所有者的 AWS 账户 ID。
- 将 *us-west-2* 替换为您的 CodeArtifact 域所在的 AWS 区域。
- 将 *my_maven_repo* 替换为您的 CodeArtifact 存储库名称。

> **Important**
>
> 必须在 --data-binary 参数的值前面加上 @ 字符。将值放在引号中时，@ 必须包含在引号内。

每个构建可能有两个以上的资产需要上传。例如，除了主 JAR 和 pom.xml 之外，可能还有 Javadoc 和源 JAR 文件。因为 CodeArtifact 会自动为每个上传的资产生成校验和，所以不需要为程序包版本资产发布校验和文件。要验证资产是否已正确上传，请使用 list-package-version-assets 命令提取生成的校验和，并将其与原始校验和进行比较。有关 CodeArtifact 如何处理 Maven 校验和的更多信息，请参阅 使用 Maven 校验和。

使用以下 curl 命令来发布最新构建版本的 maven-metadata.xml 文件：

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/maven-metadata.xml \
     --data-binary @maven-metadata.xml
```

maven-metadata.xml 文件必须引用 <snapshotVersions> 元素中最新构建版本的至少一个资产。此外，必须存在 <timestamp> 值，且该值必须与资产文件名中的时间戳相匹配。例如，对于之前发布的 20210729.171330-2 构建，maven-metadata.xml 的内容将为：

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <groupId>com.mycompany.app</groupId>
  <artifactId>pkg-1</artifactId>
  <version>1.0-SNAPSHOT</version>
  <versioning>
    <snapshot>
      <timestamp>20210729.171330</timestamp>
      <buildNumber>2</buildNumber>
    </snapshot>
    <lastUpdated>20210729171330</lastUpdated>
    <snapshotVersions>
      <snapshotVersion>
        <extension>jar</extension>
        <value>1.0-20210729.171330-2</value>
        <updated>20210729171330</updated>
      </snapshotVersion>
      <snapshotVersion>
        <extension>pom</extension>
        <value>1.0-20210729.171330-2</value>
        <updated>20210729171330</updated>
      </snapshotVersion>
    </snapshotVersions>
  </versioning>
```

```
</metadata>
```

发布 maven-metadata.xml 后，最后一步是将所有其他构建版本（即除最新构建之外的所有构建版本）的程序包版本状态设置为 Unlisted。例如，如果 1.0-SNAPSHOT 版本有两个构建，第一个构建是 20210728.194552-1，则将该构建设置为 Unlisted 的命令是：

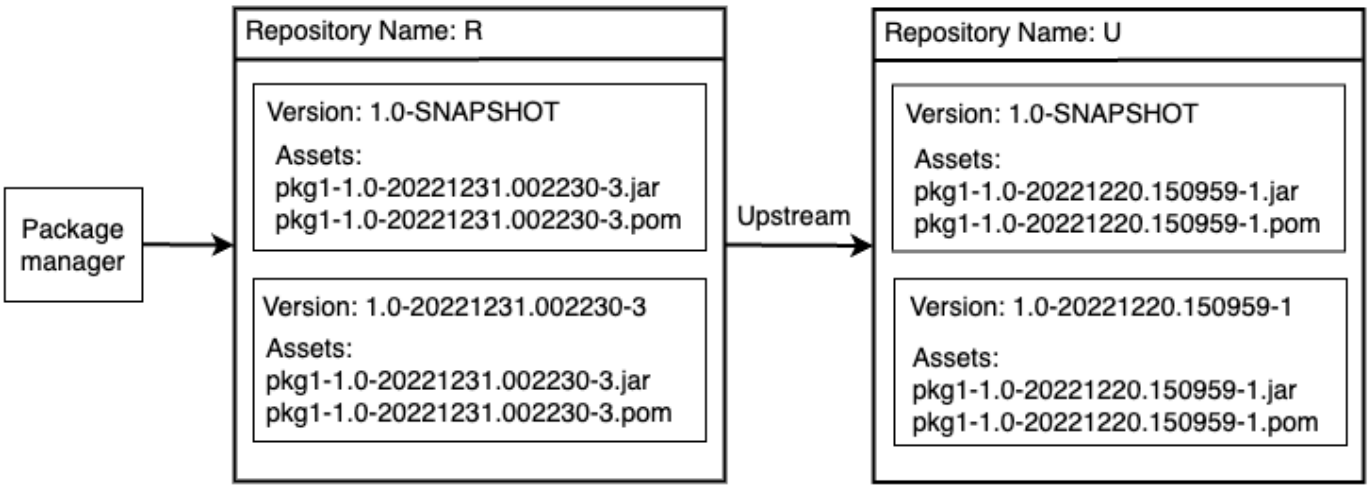
```
aws codeartifact update-package-versions-status --domain my-domain --domain-owner
 111122223333 \
   --repository my-maven-repo --format maven --namespace com.mycompany.app --package
 pkg-1 \
   --versions 1.0-20210728.194552-1 --target-status Unlisted
```

快照和外部连接

无法通过外部连接从 Maven 公有存储库提取 Maven 快照。AWSCodeArtifact 仅支持导入 Maven 发布版本。

快照和上游存储库

通常，与上游存储库一起使用时，Maven 快照的工作方式与 Maven 发布版本相同。例如，假设一个 AWS CodeArtifact 域中有两个存储库（R 和 U），其中 U 是 R 的上游存储库。在这种情况下，您可以自由地将给定程序包的快照构建版本（例如 com.mycompany.app:pkg-1 的 1.0-SNAPSHOT）发布到 R 和 U。但在使用来自 R（下游存储库）的快照构建版本时，需要了解一些重要的行为。



1. 如果 R 中存在 1.0-SNAPSHOT，则只能通过配置为从 R 提取程序包的程序包管理器来检索 R 中的 1.0-SNAPSHOT 资产。您无法通过 R 在 U 中检索 1.0-SNAPSHOT 的资产。这是因为 U 中的快照版本被 R 中的版本所遮蔽。此行为与 Maven 发布版本和其他

程序包格式的行为相同。在图中，/maven/R/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/pkg-1-1.0-20221231.002230-3.jar 的 GET 将返回 200（正常）HTTP 响应代码，而 /maven/R/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/pkg-1-1.0-20221220.150959-1.jar 的 GET 将返回 404（未找到）HTTP 响应代码。

2. 如果 U 中存在 1.0-SNAPSHOT，但在 R 中不存在，则可以从 R 提取 1.0-SNAPSHOT 的资产。这会导致在 R 中保留 1.0-SNAPSHOT，就像发布版本一样。
3. 在 R 中保留 1.0-SNAPSHOT 之后，您可以在 U 中发布 1.0-SNAPSHOT 的其他构建版本。但是，由于出现第 (1) 点中描述的行为，无法从 R 访问这些构建版本。这意味着使用快照版本的标准理由（即通过特定的快照版本使用依赖项的最新版本）在上游关系中不成立。即使向 U 发布了 1.0-SNAPSHOT 的新构建，使用者也无法从 R 访问 1.0-SNAPSHOT 的最新构建。要解决这个问题，要么定期删除 R 中的版本 1.0-SNAPSHOT，要么将客户端配置为从 U 提取 1.0-SNAPSHOT 的版本。
4. Unlisted 快照构建版本可从下游存储库访问。在图中，/maven/R/com/mycompany/myapp/pkg-1/1.0-20221220.150959-1/pkg-1-1.0-20221220.150959-1.jar 的 GET 将返回 200（正常）响应代码。尽管这会请求上游存储库中存在的资产，但由于该版本是使用构建版本字符串 (1.0-20221220.150959-1) 来访问，因此可以通过下游存储库提取资产。这个 GET 也会导致版本 1.0-20221220.150959-1 保留在 R 中，且程序包版本状态为 Unlisted。

从上游和外部连接请求 Maven 程序包

导入标准资产名称

从公有存储库（例如 Maven Central）导入 Maven 程序包版本时，AWS CodeArtifact 会尝试导入该程序包版本中的所有资产。如 请求包含上游存储库的程序包版本 中所述，在以下时候会发生导入：

- 客户端从 CodeArtifact 存储库中请求 Maven 资产。
- 程序包版本尚未出现在存储库或其上游中。
- 存在与公有 Maven 存储库的可访问外部连接。

尽管客户端可能只请求了一个资产，但 CodeArtifact 会尝试导入它可为该程序包版本找到的所有资产。CodeArtifact 如何发现哪些资产可用于 Maven 程序包版本取决于特定的公有存储库。一些公有 Maven 存储库支持请求资产列表，但其他存储库则不支持。对于不提供资产列出方式的存储库，CodeArtifact 会生成一组可能存在的资产名称。例如，当请求 Maven 程序包版本 junit 4.13.2 的任何资产时，CodeArtifact 会尝试导入以下资产：

- junit-4.13.2.pom

- junit-4.13.2.jar
- junit-4.13.2-javadoc.jar
- junit-4.13.2-sources.jar

导入非标准资产名称

当 Maven 客户端请求的资产与上述模式之一不匹配时，CodeArtifact 会检查公有存储库中是否存在该资产。如果存在资产，则会将其导入并添加到现有的程序包版本记录（如果存在）中。例如，Maven 程序包版本 com.android.tools.build:aapt2 7.3.1-8691043 包含以下资产：

- aapt2-7.3.1-8691043.pom
- aapt2-7.3.1-8691043-windows.jar
- aapt2-7.3.1-8691043-osx.jar
- aapt2-7.3.1-8691043-linux.jar

当客户端请求 POM 文件时，如果 CodeArtifact 无法列出程序包版本的资产，则 POM 将是唯一导入的资产。这是因为其他资产都不符合标准资产名称模式。但是，当客户端请求其中一个 JAR 资产时，该资产将会导入并添加到 CodeArtifact 中存储的现有程序包版本中。最下游存储库（客户端向其发出请求的存储库）和具有外部连接的存储库中的程序包版本都将更新为包含新资产，如上游存储库的程序包保留中所述。

通常，在 CodeArtifact 存储库中保留程序包版本之后，它就不会受到上游存储库中更改的影响。有关更多信息，请参阅上游存储库的程序包保留。但是，对于具有前面所述的非标准名称的 Maven 资产，其行为与此规则不同。虽然如果客户端没有请求额外的资产，下游程序包版本就不会更改，但在这种情况下，保留的程序包版本在最初保留后经过修改，因此不是一成不变的。由于具有非标准名称的 Maven 资产在没有特殊处理的情况下无法通过 CodeArtifact 访问，所以需要这种特殊的行为。如果在 CodeArtifact 存储库中保留程序包版本之后，对公有存储库的 Maven 程序包版本添加这种行为，则该行为也会起作用。

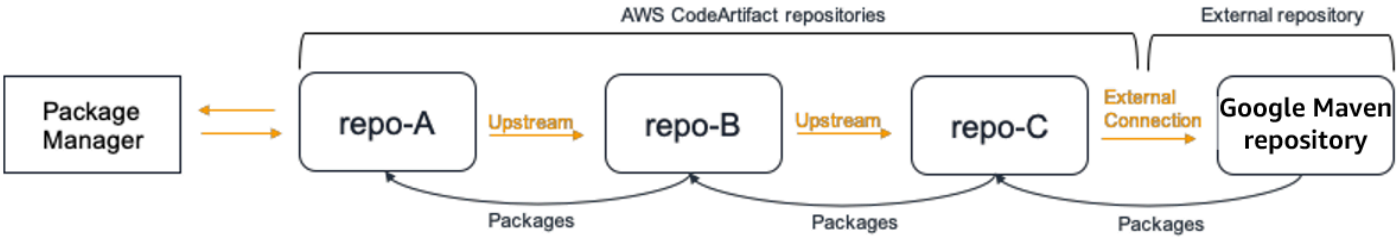
检查资产来源

将新资产添加到以前保留的 Maven 程序包版本时，CodeArtifact 会确认保留的程序包版本的来源是否与新资产的来源相同。这可以防止创建“混合”程序包版本，其中不同的资产来自不同的公有存储库。如果不进行此检查，则在将 Maven 程序包版本发布到多个公有存储库，且这些存储库是 CodeArtifact 存储库上游图的一部分时，可能会发生资产混合。

在上游存储库中导入新资产和程序包版本状态

上游存储库中程序包版本的程序包版本状态会阻止 CodeArtifact 将这些版本保留在下游存储库中。

例如，假设一个域有三个存储库：repo-A、repo-B 和 repo-C，其中 repo-B 是 repo-A 的上游存储库，repo-C 是 repo-B 的下游存储库。



repo-B 中存在 Maven 程序包 com.android.tools.build:aapt2 的程序包版本 7.3.1，其状态为 Published。它不存在于 repo-A 中。如果客户端从 repo-A 请求此程序包版本的资产，则响应将为 200（正常），且 Maven 程序包版本 7.3.1 将保留在 repo-A 中。但是，如果 repo-B 中的程序包版本 7.3.1 的状态为 Archived 或 Disposed，则响应将为 404（未找到），因为处于这两种状态的程序包版本的资产不可下载。

请注意，为 repo-A、repo-B 和 repo-C 中的 com.android.tools.build:aapt2 将程序包来源控制设置为 upstream=BLOCK，则无论程序包的版本状态是什么，都可防止从 repo-A 为该程序包的所有版本提取新资产。

Maven 故障排除

以下信息可协助您处理 Maven 和 CodeArtifact 中的常见问题。

禁用并行写入来修复错误 429：请求过多

从版本 3.9.0 开始，Maven 会并行上传程序包构件（一次最多 5 个文件）。这会导致 CodeArtifact 偶尔使用错误响应代码 429（请求过多）进行响应。如果遇到此错误，可以禁用并行写入来修复错误。

要禁用并行写入，请在 settings.xml 文件的配置文件中将 aether.connector.basic.parallelPut 属性设置为 false，如以下示例所示：

```
<settings>
    <profiles>
        <profile>
            <id>default</id>
            <properties>
```

```
                <aether.connector.basic.parallelPut>false</
aether.connector.basic.parallelPut>
            </properties>
        </profile>
    </profiles>
<settings>
```

有关更多信息，请参阅 Maven 文档中的构件解析程序配置选项。

将 CodeArtifact 与 NuGet 结合使用

以下主题介绍如何使用 CodeArtifact 来使用和发布 NuGet 程序包。

> **Note**
>
> AWS CodeArtifact 支持 NuGet.exe 4.8 及更高版本。

主题

将 CodeArtifact 与 Visual Studio 结合使用

借助 CodeArtifact 凭证提供程序，您可以在 Visual Studio 中直接使用 CodeArtifact 程序包。凭证提供程序简化了在 Visual Studio 中设置和验证 CodeArtifact 存储库的过程，凭证提供程序包含在 AWS Toolkit for Visual Studio 中。

> **Note**
>
> Visual Studio for Mac 未提供 AWS Toolkit for Visual Studio。

要通过 CLI 工具配置和使用 NuGet，请参阅将 CodeArtifact 与 nuget 或 dotnet CLI 配合使用。

主题

在 Visual Studio 中配置 CodeArtifact 凭证提供程序

CodeArtifact 凭证提供程序简化了 CodeArtifact 和 Visual Studio 之间的设置和持续身份验证。CodeArtifact 身份验证令牌的有效期最长为 12 小时。为了避免在 Visual Studio 中使用时手动刷新令牌，凭证提供程序会在当前令牌到期之前定期提取新令牌。

> **Important**
>
> 要使用凭证提供程序，请确保已从之前可能手动添加或通过运行 `aws codeartifact login` 来配置 NuGet 而添加的 `nuget.config` 文件中清除任何现有 AWS CodeArtifact 凭证。

借助 AWS Toolkit for Visual Studio 在 Visual Studio 中使用 CodeArtifact

1. 按照以下步骤安装 AWS Toolkit for Visual Studio。按照这些步骤，该工具包可与 Visual Studio 2017 和 2019 兼容。AWSCodeArtifact 不支持 Visual Studio 2015 及更早版本。
    1. 适用于 Visual Studio 2017 和 Visual Studio 2019 的 Toolkit for Visual Studio 已在 Visual Studio Marketplace 中分发。您也可以在 Visual Studio 中使用工具 ≫ 扩展和更新 (Visual Studio 2017) 或扩展 ≫ 管理扩展 (Visual Studio 2019) 来安装和更新该工具包。
    2. 工具包安装完成后，从视图菜单中选择 AWS Explorer 将其打开。
2. 按照《AWS Toolkit for Visual Studio 用户指南》的提供 AWS 凭证中的步骤，使用您的 AWS 凭证配置 Toolkit for Visual Studio。
3. （可选）设置要与 CodeArtifact 一起使用的 AWS 配置文件。如果未设置配置文件，CodeArtifact 将使用默认配置文件。要设置配置文件，请转至工具 > NuGet 程序包管理器 > 选择 CodeArtifact AWS 配置文件。
4. 在 Visual Studio 中将您的 CodeArtifact 存储库添加为程序包来源。
    1. 在 AWS Explorer 窗口中导航到您的存储库，右键单击并选择 `Copy NuGet Source Endpoint`。
    2. 使用工具 > 选项 命令并滚动到 NuGet 程序包管理器。
    3. 选择程序包来源节点。
    4. 选择 +，编辑名称，然后将在步骤 3a 中复制的存储库 URL 端点粘贴到来源框中，然后选择更新。
    5. 选中您新添加的程序包来源的复选框来启用它。

> **Note**
>
> 我们建议在 CodeArtifact 存储库中添加指向 NuGet.org 的外部连接，并在 Visual Studio 中禁用 nuget.org 程序包来源。使用外部连接时，从 Nuget.org 提取的所有程序包都将存储在您的 CodeArtifact 存储库中。如果 Nuget.org 变得不可用，您的应用程序依赖项仍可用于 CI 构建和本地开发。有关外部连接的更多信息，请参阅将 CodeArtifact 存储库连接到公有存储库。

5. 重新启动 Visual Studio 以使更改生效。

配置完成后，Visual Studio 可以使用 CodeArtifact 存储库及其任何上游存储库中的程序包，如果您添加了外部连接，则还可以使用来自 Nuget.org 的程序包。有关在 Visual Studio 中浏览和安装 NuGet 程序包的更多信息，请参阅 NuGet 文档中的使用 NuGet 程序包管理器在 Visual Studio 中安装和管理程序包。

使用 Visual Studio Package Manager 控制台

Visual Studio Package Manager 控制台将不会使用 Visual Studio 版本的 CodeArtifact 凭证提供程序。要使用它，您必须配置命令行凭证提供程序。参阅 将 CodeArtifact 与 nuget 或 dotnet CLI 配合使用 了解更多信息。

将 CodeArtifact 与 nuget 或 dotnet CLI 配合使用

您可以使用 `nuget` 和 `dotnet` 等 CLI 工具来发布和使用 CodeArtifact 中的程序包。本文档提供有关配置 CLI 工具以及使用这些工具来发布或使用程序包的信息。

主题

- 配置 nuget 或 dotnet CLI
- 使用来自 CodeArtifact 的 NuGet 程序包
- 将 NuGet 程序包发布到 CodeArtifact
- CodeArtifact NuGet 凭证提供程序参考
- CodeArtifact NuGet 凭证提供程序版本

配置 nuget 或 dotnet CLI

您可以使用 CodeArtifact NuGet 凭证提供程序、使用 AWS CLI 或手动配置 nuget 或 dotnet CLI。为了简化设置并持续进行身份验证，强烈建议使用凭证提供程序来配置 NuGet。

方法 1：使用 CodeArtifact NuGet 凭证提供程序进行配置

CodeArtifact NuGet 凭证提供程序使用 NuGet CLI 工具来简化 CodeArtifact 的身份验证和配置。CodeArtifact 身份验证令牌的有效期最长为 12 小时。为了避免在使用 nuget 或 dotnet CLI 时手动刷新令牌，凭证提供程序会在当前令牌到期之前定期提取新令牌。

> **Important**
>
> 要使用凭证提供程序，请确保已从之前可能手动添加或通过运行 `aws codeartifact login` 来配置 NuGet 而添加的 `nuget.config` 文件中清除任何现有 AWS CodeArtifact 凭证。

使用 CodeArtifact NuGet 凭证提供程序来安装和配置

dotnet

1. 使用以下 `dotnet` 命令下载 Nuget.org 的 AWS.CodeArtifact.NuGet.CredentialProvider 工具的最新版本。

   ```
   dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
   ```

2. 使用 `codeartifact-creds install` 命令将凭证提供程序复制到 NuGet 插件文件夹。

   ```
   dotnet codeartifact-creds install
   ```

3. （可选）：设置要与凭证提供程序一起使用的 AWS 配置文件。如果未设置，则凭证提供程序将使用默认配置文件。有关 AWS CLI 配置文件的更多信息，请参阅命名配置文件。

   ```
   dotnet codeartifact-creds configure set profile profile_name
   ```

nuget

执行以下步骤，使用 NuGet CLI 从 Amazon S3 存储桶安装 CodeArtifact NuGet 凭证提供程序并对其进行配置。凭证提供程序将使用默认 AWS CLI 配置文件，有关配置文件的更多信息，请参阅命名配置文件。

1. 从 Amazon S3 存储桶下载最新版本的 CodeArtifact NuGet 凭证提供程序 (codeartifact-nuget-credentialprovider.zip)。

   要查看和下载早期版本，请参阅CodeArtifact NuGet 凭证提供程序版本。

2. 解压缩该文件。
3. 将 AWS.CodeArtifact.NuGetCredentialProvider 文件夹从 netfx 文件夹复制到 Windows 上的 `%user_profile%/.nuget/plugins/netfx/`，或复制到 Linux 或 MacOS 上的 `~/.nuget/plugins/netfx`。
4. 将 AWS.CodeArtifact.NuGetCredentialProvider 文件夹从 netcore 文件夹复制到 Windows 上的 `%user_profile%/.nuget/plugins/netcore/`，或复制到 Linux 或 MacOS 上的 `~/.nuget/plugins/netcore`。

创建存储库并配置凭证提供程序后，您可以使用 `nuget` 或 `dotnet` CLI 工具来安装和发布程序包。有关更多信息，请参阅 使用来自 CodeArtifact 的 NuGet 程序包 和 将 NuGet 程序包发布到 CodeArtifact：

方法 2：使用 login 命令配置 nuget 或 dotnet

AWS CLI 中的 `codeartifact login` 命令将存储库端点和授权令牌添加到您的 NuGet 配置文件中，使 nuget 或 dotnet 能够连接到您的 CodeArtifact 存储库。这将修改用户级 NuGet 配置，该配置位于 Windows 上的 `%appdata%\NuGet\NuGet.Config`，或位于 Mac/Linux 上的 `~/.config/NuGet/NuGet.Config` 或 `~/.nuget/NuGet/NuGet.Config`。有关 NuGet 配置的更多信息，请参阅常用 NuGet 配置。

使用 **`login`** 命令配置 nuget 或 dotnet

1. 配置要与 AWS CLI 一起使用的 AWS 凭证，如 开始使用 CodeArtifact中所述。
2. 确保已正确安装和配置 NuGet CLI 工具（`nuget` 或 `dotnet`）。有关说明，请参阅 nuget 或 dotnet 文档。
3. 使用 CodeArtifact `login` 命令来提取与 NuGet 一起使用的凭证。

> **Note**
>
> 如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。

dotnet

> **Important**
>
> Linux 和 MacOS 用户：由于在非 Windows 平台上不支持加密，因此提取的凭证将以纯文本形式存储在配置文件中。

```
aws codeartifact login --tool dotnet --domain my_domain --domain-
owner 111122223333 --repository my_repo
```

nuget

```
aws codeartifact login --tool nuget --domain my_domain --domain-
owner 111122223333 --repository my_repo
```

login 命令将会：

- 使用您的 AWS 凭证从 CodeArtifact 提取授权令牌。
- 使用 NuGet 程序包源的新条目来更新用户级 NuGet 配置。指向 CodeArtifact 存储库端点的源将称为 *domain_name/repo_name*。

调用 `login` 后的默认授权期为 12 小时，且必须调用 `login` 来定期刷新令牌。有关使用 `login` 命令创建的授权令牌的更多信息，请参阅使用 `login` 命令创建的令牌。

创建存储库并配置身份验证后，可以使用 `nuget`、`dotnet` 或 `msbuild` CLI 客户端来安装和发布程序包。有关更多信息，请参阅 使用来自 CodeArtifact 的 NuGet 程序包 和 将 NuGet 程序包发布到 CodeArtifact：

方法 3：不使用 login 命令来配置 nuget 或 dotnet

要进行手动配置，必须将存储库端点和授权令牌添加到您的 NuGet 配置文件中，使 nuget 或 dotnet 能够连接到您的 CodeArtifact 存储库。

手动配置 nuget 或 dotnet 来连接到您的 CodeArtifact 存储库。

1. 使用 `get-repository-endpoint` AWS CLI 命令确定您的 CodeArtifact 存储库端点。

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-
owner 111122223333 --repository my_repo --format nuget
```

输出示例：

```
{
   "repositoryEndpoint": "https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/nuget/my_repo/"
}
```

2. 使用 `get-authorization-token` AWS CLI 命令从程序包管理器获取用于连接到存储库的授权令牌。

```
aws codeartifact get-authorization-token --domain my_domain
```

输出示例：

```
{
   "authorizationToken": "eyJ2I...viOw",
   "expiration": 1601616533.0
}
```

3. 通过将 `/v3/index.json` 附加到 `get-repository-endpoint` 在步骤 3 中返回的 URL 来创建完整的存储库端点 URL。
4. 将 nuget 或 dotnet 配置为使用步骤 1 中的存储库端点和步骤 2 中的授权令牌。

> **Note**
>
> 对于 nuget 或 dotnet，源网址必须以 `/v3/index.json` 结尾才能成功连接到 CodeArtifact 存储库。

dotnet

Linux 和 MacOS 用户：由于在非 Windows 平台上不支持加密，因此您必须在以下命令中添加 `--store-password-in-clear-text` 标志。请注意，这会将您的密码以纯文本形式存储在配置文件中。

```
dotnet nuget add source https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/nuget/my_repo/v3/index.json --name packageSourceName --
password eyJ2I...vi0w --username aws
```

> **Note**
>
> 要更新现有源，请使用 `dotnet nuget update source` 命令。

nuget

```
nuget sources add -name domain_name/repo_name -Source
 https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
nuget/my_repo/v3/index.json -password eyJ2I...vi0w -username aws
```

输出示例：

```
Package source with Name: domain_name/repo_name added successfully.
```

使用来自 CodeArtifact 的 NuGet 程序包

使用 CodeArtifact 配置 NuGet 后，您可以使用存储在 CodeArtifact 存储库或其中一个上游存储库中的 NuGet 程序包。

要通过 nuget 或 dotnet 使用来自 CodeArtifact 存储库或其中一个上游存储库的程序包版本，请运行以下命令，将 *packageName* 替换为您要使用的程序包的名称，将 *packageSourceName* 替换为您的 CodeArtifact 存储库在 NuGet 配置文件中的源名称。如果您使用 login 命令来配置 NuGet 配置，则源名称为 *domain_name/repo_name*。

> **Note**
>
> 当请求程序包时，NuGet 客户端会缓存该程序包中包含的版本。由于存在这种行为，对于在所需版本变得可用之前请求的程序包，安装可能会失败。为了避免出现这种失败并成功安装现有的程序包，可以在使用 `nuget locals all --clear` 或 `dotnet nuget locals all --clear` 安装之前清除 NuGet 缓存，也可以通过为 nuget 提供 `-NoCache` 选项或为 dotnet 提供 `--no-cache` 选项，避免在执行 `install` 和 `restore` 命令期间使用缓存。

dotnet

```
dotnet add package packageName --source packageSourceName
```

nuget

```
nuget install packageName -Source packageSourceName
```

安装程序包的特定版本

dotnet

```
dotnet add package packageName --version 1.0.0 --source packageSourceName
```

nuget

```
nuget install packageName -Version 1.0.0 -Source packageSourceName
```

有关更多信息，请参阅 Microsoft 文档中的使用 nuget.exe CLI 管理程序包或使用 dotnet CLI 安装和管理程序包。

使用来自 NuGet.org 的 NuGet 程序包

您可以通过 CodeArtifact 存储库使用来自 Nuget.org 的 NuGet 程序包，方法是为存储库配置与 NuGet.org 的外部连接。从 Nuget.org 使用的程序包会摄取并存储在您的 CodeArtifact 存储库中。有关添加外部连接的更多信息，请参阅将 CodeArtifact 存储库连接到公有存储库。

将 NuGet 程序包发布到 CodeArtifact

使用 CodeArtifact 配置 NuGet 后，您可以使用 `nuget` 或 `dotnet` 将程序包版本发布到 CodeArtifact 存储库。

要将程序包版本推送到 CodeArtifact 存储库，请运行以下命令，使用 `.nupkg` 文件的完整路径和 NuGet 配置文件中 CodeArtifact 存储库的源名称。如果您使用 `login` 命令来配置 NuGet 配置，则源名称为 `domain_name/repo_name`。

> **Note**
>
> 如果您没有要发布的 NuGet 程序包，则可以创建一个。有关更多信息，请参阅 Microsoft 文档中的 程序包创建工作流程。

dotnet

```
dotnet nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg --source packageSourceName
```

nuget

```
nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg -Source packageSourceName
```

CodeArtifact NuGet 凭证提供程序参考

借助 CodeArtifact NuGet 凭证提供程序，可以轻松地配置 NuGet 并向 CodeArtifact 存储库进行身份验证。

CodeArtifact NuGet 凭证提供程序命令

本节包括 CodeArtifact NuGet 凭证提供程序的命令列表。这些命令必须加上前缀 `dotnet codeartifact-creds`，如以下示例所示。

```
dotnet codeartifact-creds command
```

- `configure set profile profile`：将凭证提供者配置为使用提供的 AWS 配置文件。
- `configure unset profile`：删除已配置的配置文件（如果已设置）。
- `install`：将凭证提供程序复制到 `plugins` 文件夹。
- `install --profile profile`：将凭证提供程序复制到 `plugins` 文件夹，并将它配置为使用提供的 AWS 配置文件。
- `uninstall`：卸载凭证提供程序。这不会删除对配置文件的更改。
- `uninstall --delete-configuration`：卸载凭证提供程序并删除对配置文件的所有更改。

CodeArtifact NuGet 凭证提供程序日志

要为 CodeArtifact NuGet 凭据提供程序启用日志记录，必须在环境中设置日志文件。凭证提供程序日志包含有用的调试信息，例如：

- 用于建立连接的 AWS 配置文件
- 任何身份验证错误
- 如果提供的端点不是 CodeArtifact URL

设置 CodeArtifact NuGet 凭证提供程序日志文件

```
export AWS_CODEARTIFACT_NUGET_LOGFILE=/path/to/file
```

设置日志文件后，任何 `codeartifact-creds` 命令都会将其日志输出附加到该文件的内容中。

CodeArtifact NuGet 凭证提供程序版本

以下表格包含 CodeArtifact NuGet 凭证提供程序的版本历史记录信息和下载链接。

版本	更改	发布日期	下载链接 (S3)
1.0.1（最新版本）	增加了对 net5、net6 和 SSO 配置文件的支持	2022 年 3 月 5 日	下载 v1.0.1

版本	更改	发布日期	下载链接 (S3)
1.0.0	CodeArtifact NuGet 凭证提供程序的初始版本	2020 年 11 月 20 日	下载 v1.0.0

NuGet 程序包名称、版本和资产名称规范化

在存储程序包名称、资产名称和程序包版本之前，CodeArtifact 会对程序包名称、资产名称和程序包版本进行规范化，这意味着 CodeArtifact 中的名称或版本可能与发布程序包或资产时提供的名称或版本不同。

程序包名称规范化：CodeArtifact 通过将所有字母转换为小写来使 NuGet 程序包名称规范化。

程序包版本规范化：CodeArtifact 使用与 NuGet 相同的模式对 NuGet 程序包版本进行规范化。以下信息来自 NuGet 文档中的规范化版本号。

- 从版本号中删除前导零：
  - `1.00` 视为 `1.0`
  - `1.01.1` 视为 `1.1.1`
  - `1.00.0.1` 视为 `1.0.0.1`
- 版本号第四部分中的零会省略掉：
  - `1.0.0.0` 视为 `1.0.0`
  - `1.0.01.0` 视为 `1.0.1`
- 删除 SemVer 2.0.0 构建元数据：
  - `1.0.7+r3456` 视为 `1.0.7`

程序包资产名称规范化：CodeArtifact 根据规范化的程序包名称和程序包版本来构造 NuGet 程序包资产名称。

API 和 CLI 请求中可使用非规范化程序包名称和版本名称，因为 CodeArtifact 会对这些请求的程序包名称和版本输入进行规范化。例如，`--package Newtonsoft.JSON` 和 `--version 12.0.03.0` 的输入会规范化，并返回一个使用规范化程序包名称 `newtonsoft.json` 和版本 `12.0.3` 的程序包。

必须在 API 和 CLI 请求中使用规范化程序包资产名称，因为 CodeArtifact 不会对 `--asset` 输入执行规范化。

必须在 ARN 中使用规范化名称和版本。

要查找程序包的规范化名称，请使用 `aws codeartifact list-packages` 命令。有关更多信息，请参阅列出程序包名称。

要查找程序包的非规范化名称，请使用 `aws codeartifact describe-package-version` 命令。`displayName` 字段中返回程序包的非规范化名称。有关更多信息，请参阅查看和更新程序包版本详细信息和依赖项。

NuGet 兼容性

本指南说明 CodeArtifact 与不同 NuGet 工具和版本的兼容性。

主题

NuGet 通用兼容性

AWS CodeArtifact 支持 NuGet 4.8 及更高版本。

AWS CodeArtifact 仅支持 NuGet HTTP 协议的 V3 版本。这意味着不支持某些依赖协议 V2 版本的 CLI 命令。有关更多信息，请参阅nuget.exe 命令支持部分。

AWS CodeArtifact 不支持 PowerShellGet 2.x。

NuGet 命令行支持

AWS CodeArtifact 支持 NuGet (`nuget.exe`) 和 .NET Core (`dotnet`) CLI 工具。

nuget.exe 命令支持

由于 CodeArtifact 仅支持 NuGet 的 HTTP 协议的 V3 版本，因此以下命令在用于 CodeArtifact 资源时将不起作用：

- `list`：`nuget list` 命令显示来自给定来源的程序包列表。要获取 CodeArtifact 存储库中的程序包列表，您可以使用 AWS CLI 中的 列出程序包名称 命令。

CodeArtifact 与 Swift 配

这些主题描述了如何使用 Swift Package Manager CodeArtifact 来安装和发布 Swift 软件包。

> **Note**
>
> CodeArtifact 支持 Swift 5.8 及更高版本以及 Xcode 14.3 及更高版本。
> CodeArtifact 推荐 Swift 5.9 及更高版本以及 Xcode 15 及更高版本。

主题

配置 Swift Package Manager CodeArtifact

要使用 Swift Package Manager 向仓库发布软件包或使用其中的软件包 AWS CodeArtifact，你首先需要设置访问 CodeArtifact 仓库的凭证。使用您的 CodeArtifact 凭据和存储库端点配置 Swift Package Manager CLI 的推荐方法是使用`aws codeartifact login`命令。您也可以手动配置 Swift 程序包管理器。

使用 login 命令配置 Swift

使用`aws codeartifact login`命令配置 Swift Package Manager CodeArtifact。

> **Note**
>
> 要使用 login 命令，需要使用 Swift 5.8 或更高版本，建议使用 Swift 5.9 或更高版本。

`aws codeartifact login` 命令将执行以下操作：

1. 从中获取身份验证令牌 CodeArtifact 并将其存储在您的环境中。凭证的存储方式取决于环境的操作系统：
   a. macOS：在 macOS 钥匙串应用程序中创建一个条目。

b. Linux 和 Windows：在 ~/.netrc 文件中创建一个条目。

在所有操作系统中，如果存在凭证条目，则此命令会用新令牌替换该条目。

2. 获取存储 CodeArtifact 库端点 URL 并将其添加到 Swift 配置文件中。该命令将存储库端点 URL 添加到位于 /path/to/project/.swiftpm/configuration/registries.json 的项目级配置文件中。

> **Note**
>
> aws codeartifact login 命令会调用 swift package-registry 命令，后面的命令必须从包含 Package.swift 文件的目录中运行。因此，aws codeartifact login 命令必须在 Swift 项目中运行。

使用 login 命令配置 Swift

1. 导航到包含项目 Package.swift 文件的 Swift 项目目录。
2. 运行以下 aws codeartifact login 命令：

   如果您要访问您拥有的域中的存储库，则无需包括 --domain-owner。有关更多信息，请参阅跨账户域。

```
aws codeartifact login --tool swift --domain my_domain \
--domain-owner 111122223333 --repository my_repo \
[--namespace my_namespace]
```

该--namespace选项将应用程序配置为仅使用存储 CodeArtifact 库中位于指定命名空间中的软件包。CodeArtifact 命名空间是作用域的同义词，用于将代码组织成逻辑组，并防止在代码库包含多个库时可能发生的名称冲突。

调用 login 后的默认授权期为 12 小时，且必须调用 login 来定期刷新令牌。有关使用 login 命令创建的授权令牌的更多信息，请参阅使用 login 命令创建的令牌。

不使用 login 命令配置 Swift

虽然建议您使用 aws codeartifact login 命令来配置 Swift，但您也可以通过手动更新 Swift 程序包管理器配置来配置 Swift 程序包管理器，而不使用 login 命令。

在以下步骤中，您将使用 AWS CLI 来执行以下操作：

1. 从中获取身份验证令牌 CodeArtifact 并将其存储在您的环境中。凭证的存储方式取决于环境的操作系统：
   a. macOS：在 macOS 钥匙串应用程序中创建一个条目。
   b. Linux 和 Windows：在 `~/.netrc` 文件中创建一个条目。
2. 获取您的 CodeArtifact 存储库终端节点 URL。
3. 在 `~/.swiftpm/configuration/registries.json` 配置文件中，添加一个包含存储库端点 URL 和身份验证类型的条目。

不使用 login 命令配置 Swift

1. 在命令行中，使用以下命令获取 CodeArtifact 授权令牌并将其存储在环境变量中。

   - 将 *my_dom* ain 替换为您的 CodeArtifact 域名。
   - 将 *111122223333* 替换为域名所有者的 AWS 账户 ID。如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。

   macOS and Linux

   ```
   export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
   domain my_domain --domain-owner 111122223333 --query authorizationToken --output
    text`
   ```

   Windows

   - Windows（使用默认命令 shell）：

     ```
     for /f %i in ('aws codeartifact get-authorization-token --domain my_domain --
     domain-owner 111122223333 --query authorizationToken --output text') do set
      CODEARTIFACT_AUTH_TOKEN=%i
     ```

   - 窗户 PowerShell：

     ```
     $env:CODEARTIFACT_AUTH_TOKEN = aws codeartifact get-authorization-token --
     domain my_domain --domain-owner 111122223333 --query authorizationToken --
     output text
     ```

2. 运行以下命令获取 CodeArtifact 仓库的终端节点。您的存储库端点用于将 Swift 程序包管理器指向您的存储库来使用或发布程序包。

   - 将 *my_dom* ain 替换为您的 CodeArtifact 域名。
   - 将 *111122223333* 替换为域名所有者的 AWS 账户 ID。如果您要访问您拥有的域中的存储库，则无需包括 `--domain-owner`。有关更多信息，请参阅跨账户域。
   - 将 *my_repo* 替换为你的 CodeArtifact 存储库名称。

   macOS and Linux

   ```
   export CODEARTIFACT_REPO=`aws codeartifact get-repository-endpoint --
   domain my_domain --domain-owner 111122223333 --repository my_repo --format swift
    --query repositoryEndpoint --output text`
   ```

   Windows

   - Windows（使用默认命令 shell）：

     ```
     for /f %i in ('aws codeartifact get-repository-endpoint --domain my_domain
      --domain-owner 111122223333 --repository my_repo --format swift --query
      repositoryEndpoint --output text') do set CODEARTIFACT_REPO=%i
     ```

   - 窗户 PowerShell：

     ```
     $env:CODEARTIFACT_REPO = aws codeartifact get-repository-endpoint --
     domain my_domain --domain-owner 111122223333 --repository my_repo --format
      swift --query repositoryEndpoint --output text
     ```

   以下 URL 是一个示例存储库端点。

   ```
   https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
   swift/my_repo/
   ```

   > **Important**
   >
   > 当用于配置 Swift 程序包管理器时，必须将 `login` 附加到存储库 URL 端点的末尾。在此程序的命令中为您完成附加。

3. 将这两个值存储在环境变量中之后，使用 `swift package-registry login` 命令将它们传递给 Swift，如下所示：

   macOS and Linux

   ```
   swift package-registry login ${CODEARTIFACT_REPO}login --token
    ${CODEARTIFACT_AUTH_TOKEN}
   ```

   Windows

   - Windows（使用默认命令 shell）：

     ```
     swift package-registry login %CODEARTIFACT_REPO%login --token
      %CODEARTIFACT_AUTH_TOKEN%
     ```

   - 窗户 PowerShell：

     ```
     swift package-registry login $Env:CODEARTIFACT_REPO+"login" --token
      $Env:CODEARTIFACT_AUTH_TOKEN
     ```

4. 接下来，更新应用程序使用的软件包注册表，以便从存储 CodeArtifact 库中提取任何依赖项。必须在您尝试解析程序包依赖项所在的项目目录中运行此命令：

   macOS and Linux

   ```
   $ swift package-registry set ${CODEARTIFACT_REPO} [--scope my_scope]
   ```

   Windows

   - Windows（使用默认命令 shell）：

     ```
     $ swift package-registry set %CODEARTIFACT_REPO% [--scope my_scope]
     ```

   - 窗户 PowerShell：

     ```
     $ swift package-registry set $Env:CODEARTIFACT_REPO [--scope my_scope]
     ```

该--scope选项将应用程序配置为仅在指定范围内使用存储 CodeArtifact 库中的软件包。作用域是CodeArtifact 命名空间的同义词，用于将代码组织成逻辑组，并防止代码库包含多个库时可能发生的名称冲突。

5. 通过在项目目录中运行以下命令，可以查看项目级 `.swiftpm/configuration/registries.json` 文件的内容，从而确认已正确设置配置：

```
$ cat .swiftpm/configuration/registries.json
{
  "authentication" : {

  },
  "registries" : {
    "[default]" : {
      "url" : "https://my-domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/swift/my-repo/"
    }
  },
  "version" : 1
}
```

现在，你已经为 CodeArtifact 仓库配置了 Swift Package Manager，你可以用它来发布和使用 Swift 软件包和从中使用 Swift 包。有关更多信息，请参阅使用和发布 Swift 程序包。

使用和发布 Swift 程序包

使用来自的 Swift 包 CodeArtifact

使用以下过程使用 AWS CodeArtifact 存储库中的 Swift 软件包。

使用 CodeArtifact 存储库中的 Swift 软件包

1. 如果还没有，请按照中的配置 Swift Package Manager CodeArtifact步骤将 Swift Package Manager 配置为使用具有正确凭据的 CodeArtifact 存储库。

> **Note**
>
> 生成的授权令牌有效期为 12 小时。如果自创建令牌以来已过去 12 小时，则需要创建一个新的令牌。

2. 编辑应用程序项目文件夹中的 `Package.swift` 文件来更新项目要使用的程序包依赖项。

   a. 如果 `Package.swift` 文件不包含任何 `dependencies` 部分，请添加这个部分。

   b. 在 `Package.swift` 文件的 `dependencies` 部分，通过添加程序包标识符来添加要使用的程序包。程序包标识符包含作用域和程序包名称，中间用句点分隔。有关示例，请参阅后续步骤之后的代码段。

   > **Tip**
   >
   > 要查找软件包标识符，您可以使用 CodeArtifact 控制台。找到要使用的特定程序包版本，并参考程序包版本页面上的安装快捷方式说明。

   c. 如果 `Package.swift` 文件不包含任何 `targets` 部分，请添加这个部分。

   d. 在 `targets` 部分中，添加需要使用依赖项的目标。

   以下代码段是一个示例代码段，显示了 `Package.swift` 文件中已配置的 `dependencies` 和 `targets` 部分：

```
...
    ],
    dependencies: [
        .package(id: "my_scope.package_name", from: "1.0.0")
    ],
    targets: [
      .target(
         name: "MyApp",
         dependencies: ["package_name"]
      ),...
    ],
...
```

3. 现在，所有内容都已配置完毕，请使用以下命令从中下载软件包依赖关系 CodeArtifact。

```
swift package resolve
```

使用 Xcode CodeArtifact 中的 Swift 软件包

使用以下过程在 Xcode 中使用 CodeArtifact 存储库中的 Swift 软件包。

在 Xcode 中使用 CodeArtifact 存储库中的 Swift 软件包

1. 如果还没有，请按照中的配置 Swift Package Manager CodeArtifact步骤将 Swift Package Manager 配置为使用具有正确凭据的 CodeArtifact 存储库。

   > **Note**
   >
   > 生成的授权令牌有效期为 12 小时。如果自创建令牌以来已过去 12 小时，则需要创建一个新的令牌。

2. 在 Xcode 中将这些程序包作为依赖项添加到项目中。

   a. 选择文件 > 添加程序包。

   b. 使用搜索栏来搜索程序包。搜索必须采用 `package_scope.package_name` 形式。

   c. 找到程序包后，选择程序包并选择添加程序包。

   d. 验证程序包后，选择要添加为依赖项的程序包产品，然后选择添加程序包。

如果您在 Xcode 中使用存储 CodeArtifact 库时遇到问题，Swift 故障排除请参阅，了解常见问题和可能的修复方法。

将 Swift 包发布到 CodeArtifact

CodeArtifact 推荐 Swift 5.9 或更高版本，并使用`swift package-registry publish`命令发布 Swift 软件包。如果您使用的是早期版本，则必须使用 Curl 命令将 Swift 软件包发布到 CodeArtifact。

使用**`swift package-registry publish`**命令发布 CodeArtifact 软件包

在 Swift 5.9 或更高版本中使用以下过程使用 Swift Package Manager 将 Swift 软件包发布到 CodeArtifact 存储库。

1. 如果还没有，请按照中的配置 Swift Package Manager CodeArtifact步骤将 Swift Package Manager 配置为使用具有正确凭据的 CodeArtifact 存储库。

> **Note**
>
> 生成的授权令牌有效期为 12 小时。如果自创建令牌以来已过去 12 小时，则需要创建一个新的令牌。

2. 导航到包含程序包的 `Package.swift` 文件的 Swift 项目目录。
3. 运行以下 `swift package-registry publish` 命令来发布程序包。该命令创建软件包源存档并将其发布到您的 CodeArtifact 存储库。

```
swift package-registry publish packageScope.packageName packageVersion
```

例如：

```
swift package-registry publish myScope.myPackage 1.0.0
```

4. 您可以通过在控制台中检查或按如下所示使用 `aws codeartifact list-packages` 命令来确认已发布程序包并存在于存储库中：

```
aws codeartifact list-packages --domain my_domain --repository my_repo
```

您可以按如下所示使用 `aws codeartifact list-package-versions` 命令来列出程序包的单个版本：

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

使用 Cur CodeArtifact l 发布软件包

虽然建议使用 Swift 5.9 或更高版本附带的`swift package-registry publish`命令，但你也可以使用 Curl 将 Swift 软件包发布到。 CodeArtifact

使用以下过程使用 Curl 将 Swift 软件包发布到 AWS CodeArtifact 存储库。

1. 如果还没有创建和更新环境变量，请按照配置 Swift Package Manager CodeArtifact中的步骤创建和更新 `CODEARTIFACT_AUTH_TOKEN` 和 `CODEARTIFACT_REPO` 环境变量。

> **Note**
>
> 授权令牌有效期为 12 小时。如果 CODEARTIFACT_AUTH_TOKEN 环境变量自创建以来已过去 12 小时，则需要使用新的凭证刷新环境变量。

2. 首先，如果您没有创建 Swift 程序包，则可以通过运行以下命令来创建：

```
mkdir testDir && cd testDir
swift package init
git init .
swift package archive-source
```

3. 使用以下 Curl 命令将你的 Swift 软件包发布到 CodeArtifact：

   macOS and Linux

```
curl -X PUT  --user "aws:$CODEARTIFACT_AUTH_TOKEN" \
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@test_dir_package_name.zip" \
"${CODEARTIFACT_REPO}my_scope/package_name/packageVersion"
```

   Windows

```
curl -X PUT  --user "aws:%CODEARTIFACT_AUTH_TOKEN%" \
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@test_dir_package_name.zip" \
"%CODEARTIFACT_REPO%my_scope/package_name/packageVersion"
```

4. 您可以通过在控制台中检查或按如下所示使用 `aws codeartifact list-packages` 命令来确认已发布程序包并存在于存储库中：

```
aws codeartifact list-packages --domain my_domain --repository my_repo
```

   您可以按如下所示使用 `aws codeartifact list-package-versions` 命令来列出程序包的单个版本：

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

从中获取 Swift 软件包 GitHub 并重新发布到 CodeArtifact

使用以下过程从中获取 Swift 软件包 GitHub 并将其重新发布到 CodeArtifact 存储库。

从中获取 Swift 软件包 GitHub 并将其重新发布到 CodeArtifact

1. 如果还没有，请按照中的配置 Swift Package Manager CodeArtifact步骤将 Swift Package Manager 配置为使用具有正确凭据的 CodeArtifact 存储库。

   > **Note**
   >
   > 生成的授权令牌有效期为 12 小时。如果自创建令牌以来已过去 12 小时，则需要创建一个新的令牌。

2. 使用以下 `git clone` 命令克隆要提取并重新发布的 Swift 程序包的 Git 存储库。有关克隆 GitHub 仓库的信息，请参阅 GitHub 文档中的克隆仓库。

   ```
   git clone repoURL
   ```

3. 导航到您刚刚克隆的存储库：

   ```
   cd repoName
   ```

4. 创建包并将其发布到 CodeArtifact。

   a. 推荐：如果您使用的是 Swift 5.9 或更高版本，则可以使用以下`swift package-registry publish`命令创建软件包并将其发布到您配置的 CodeArtifact 存储库中。

      ```
      swift package-registry publish packageScope.packageName versionNumber
      ```

      例如：

      ```
      swift package-registry publish myScope.myPackage 1.0.0
      ```

   b. 如果您使用的 Swift 版本早于 5.9，则必须使用 `swift archive-source` 命令创建程序包，然后使用 Curl 命令来发布程序包。

      i. 如果您尚未配置 `CODEARTIFACT_AUTH_TOKEN` 和 `CODEARTIFACT_REPO` 环境变量，或者自配置以来已过去 12 小时，请按照不使用 login 命令配置 Swift中的步骤进行操作。

      ii. 使用 `swift package archive-source` 命令创建 Swift 程序包：

```
swift package archive-source
```

如果成功，您会在命令行中看到 `Created` *package_name*`.zip`。

iii. 使用以下 Curl 命令将 Swift 软件包发布到 CodeArtifact：

macOS and Linux

```
curl -X PUT  --user "aws:$CODEARTIFACT_AUTH_TOKEN" \
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@package_name.zip" \
"${CODEARTIFACT_REPO}my_scope/package_name/packageVersion"
```

Windows

```
curl -X PUT  --user "aws:%CODEARTIFACT_AUTH_TOKEN%" \
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@package_name.zip" \
"%CODEARTIFACT_REPO%my_scope/package_name/packageVersion"
```

5. 您可以通过在控制台中检查或按如下所示使用 `aws codeartifact list-packages` 命令来确认已发布程序包并存在于存储库中：

```
aws codeartifact list-packages --domain my_domain --repository my_repo
```

您可以按如下所示使用 `aws codeartifact list-package-versions` 命令来列出程序包的单个版本：

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

Swift 程序包名称和命名空间规范化

CodeArtifact 在存储软件包名称和命名空间之前对其进行标准化，这意味着中的名称 CodeArtifact 可能与发布软件包时提供的名称不同。

Pac@@ kage 名称和命名空间标准化：通过将所有字母转换为小写来 CodeArtifact 规范化 Swift 软件包名称和命名空间。

Package 版本标准化： CodeArtifact 不对 Swift 包版本进行标准化。请注意， CodeArtifact 仅支持语义版本控制 2.0 版本模式，有关语义版本控制的更多信息，请参阅语义版本控制 2.0.0。

非标准化包名和命名空间可用于 API 和 CLI 请求，因为 CodeArtifact 会对这些请求的输入进行标准化。例如，`--package myPackage` 和 `--namespace myScope` 的输入会规范化，并返回一个使用规范化程序包名称 `mypackage` 和命名空间 `myscope` 的程序包。

您必须在 ARN 中（例如在 IAM 策略中）使用规范化名称。

要查找程序包的规范化名称，请使用 `aws codeartifact list-packages` 命令。有关更多信息，请参阅列出程序包名称。

Swift 故障排除

以下信息可能有助于你解决 Swift 和的常见问题 CodeArtifact。

即使在配置了 Swift 程序包管理器之后，我在 Xcode 中还是出现了 401 错误

问题：当你尝试将 CodeArtifact 仓库中的软件包作为依赖项添加到 Xcode 中的 Swift 项目时，即使你按照连接到 Swift 的说明进行操作，也会收到 CodeArtifact 401 未经授权的错误。

可能的修复方法：这可能是由存储您的 CodeArtifact 凭据的 macOS 钥匙串应用程序存在问题引起的。要解决这个问题，我们建议打开 Keychain 应用程序并删除所有 CodeArtifact 条目，然后按照中的说明再次使用 CodeArtifact 存储库配置 Swift Package Manager。配置 Swift Package Manager CodeArtifact

由于钥匙串提示输入密码，Xcode 在 CI 计算机上挂起

问题：当你尝试从 CodeArtifact 持续集成 (CI) 服务器上的 Xcode 版本中提取 Swift 包时，例如使用 GitHub Actions，身份验证 CodeArtifact 可能会挂起并最终失败，并显示类似于以下内容的错误消息：

```
Failed to save credentials for
\'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com\'
to keychain: status -60008
```

可能的修复：这是由于未将凭据保存到 CI 计算机上的钥匙串中，并且 Xcode 仅支持保存在钥匙串中的凭据。要解决这个问题，我们建议使用以下步骤手动创建钥匙串条目：

1. 准备好钥匙串。

```
KEYCHAIN_PASSWORD=$(openssl rand -base64 20)
KEYCHAIN_NAME=login.keychain
SYSTEM_KEYCHAIN=/Library/Keychains/System.keychain

if [ -f $HOME/Library/Keychains/"${KEYCHAIN_NAME}"-db ]; then
    echo "Deleting old ${KEYCHAIN_NAME} keychain"
    security delete-keychain "${KEYCHAIN_NAME}"
fi
echo "Create Keychain"
security create-keychain -p "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"

EXISTING_KEYCHAINS=( $( security list-keychains | sed -e 's/ *//' | tr '\n' ' ' |
 tr -d '"') )
sudo security list-keychains -s "${KEYCHAIN_NAME}" "${EXISTING_KEYCHAINS[@]}"

echo "New keychain search list :"
security list-keychain

echo "Configure keychain : remove lock timeout"
security unlock-keychain -p "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"
security set-keychain-settings "${KEYCHAIN_NAME}"
```

2. 获取 CodeArtifact 身份验证令牌和您的存储库端点。

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token \
                                    --region us-west-2 \
                                    --domain my_domain   \
                                    --domain-owner 111122223333   \
                                    --query authorizationToken  \
                                    --output text`

export CODEARTIFACT_REPO=`aws codeartifact get-repository-endpoint  \
                                --region us-west-2   \
                                --domain my_domain   \
                                --domain-owner 111122223333   \
                                --format swift     \
                                --repository my_repo    \
                                --query repositoryEndpoint   \
                                --output text`
```

3. 手动创建钥匙串条目。

```
SERVER=$(echo $CODEARTIFACT_REPO | sed  's/https:\/\///g' | sed 's/.com.*$/.com/g')
AUTHORIZATION=(-T /usr/bin/security -T /usr/bin/codesign -T /usr/bin/xcodebuild -
T /usr/bin/swift \
               -T /Applications/Xcode-15.2.app/Contents/Developer/usr/bin/
xcodebuild)

security delete-internet-password -a token -s $SERVER -r htps "${KEYCHAIN_NAME}"

security add-internet-password -a token \
                               -s $SERVER \
                               -w $CODEARTIFACT_AUTH_TOKEN \
                               -r htps \
                               -U \
                               "${AUTHORIZATION[@]}" \
                               "${KEYCHAIN_NAME}"

security set-internet-password-partition-list \
             -a token \
             -s $SERVER \
             -S "com.apple.swift-
package,com.apple.security,com.apple.dt.Xcode,apple-tool:,apple:,codesign" \
             -k "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"

security find-internet-password   "${KEYCHAIN_NAME}"
```

有关此错误和解决方案的更多信息，请参阅 https://github.com/apple/ swift-package-manager / issues/7236。

在 CodeArtifact 中使用通用程序包

这些主题介绍如何使用 AWS CodeArtifact 来使用和发布通用程序包。

主题

通用程序包概述

您可以使用 `generic` 程序包格式上传任何类型的文件，从而在 CodeArtifact 存储库中创建程序包。通用程序包与任何特定的编程语言、文件类型或程序包管理生态系统无关。这对于存储任意构建构件（例如应用程序安装程序、机器学习模型、配置文件等）和进行版本控制非常有用。

通用程序包由程序包名称、命名空间、版本和一个或多个资产（或文件）组成。在单个 CodeArtifact 存储库中，通用程序包可以与其他格式的程序包共存。

您可以使用 AWS CLI 或 SDK 来处理通用程序包。有关适用于通用程序包的 AWS CLI 命令的完整列表，请参阅通用程序包支持的命令。

通用程序包限制

- 永远不会从上游存储库中提取通用程序包。只能从通用程序包发布到的存储库中提取通用程序包。
- 通用程序包不能声明要从 ListPackageVersionDependencies 返回或在 AWS Management Console 中显示的依赖项。
- 通用程序包可以存储自述文件和许可证文件，但 CodeArtifact 无法解读这些文件。这些文件中的信息不会从 GetPackageVersionReadme 或 DescribePackageVersion 返回，也不会出现在AWS Management Console中。
- 与 CodeArtifact 中的所有程序包一样，资产大小和每个程序包的资产数量受到限制。有关 CodeArtifact 限制和配额的更多信息，请参阅AWS CodeArtifact 中的配额。
- 程序包包含的资产名称必须遵循以下规则：
  - 资产名称可以使用 Unicode 字母和数字。具体而言，允许使用以下 Unicode 字符类别：小写字母 (`Ll`)、修饰符字母 (`Lm`)、其他字母 (`Lo`)、首字母大写字母 (`Lt`)、大写字母 (`Lu`)、字母数字 (`Nl`) 和十进制数字 (`Nd`)。

- 允许使用以下特殊字符：~!@^&()-_+[]{};,.
- 资产无法命名为 . 或 ..
- 空格是唯一允许的空白字符。资产名称不能以空格字符开头或结尾，也不能包括连续空格。

通用程序包支持的命令

您可以使用 AWS CLI 或 SDK 来处理通用程序包。以下 CodeArtifact 命令适用于通用程序包：

- copy-package-versions（请参阅在存储库之间复制程序包）
- delete-package（请参阅删除软件包 (AWS CLI)）
- delete-package-versions（请参阅删除程序包版本 (AWS CLI)）
- describe-package
- describe-package-version（请参阅查看和更新程序包版本详细信息和依赖项）
- dispose-package-versions（请参阅处置程序包版本）
- get-package-version-asset（请参阅下载程序包版本资源）
- list-package-version-assets（请参阅列出程序包版本资产）
- list-package-versions（请参阅列出程序包版本）
- list-packages（请参阅列出程序包名称）
- publish-package-version（请参阅发布通用程序包）
- put-package-origin-configuration（请参阅编辑程序包来源控制）

> **Note**
>
> 您可以使用 publish 来源控制设置来允许或阻止在存储库中发布通用程序包名称。但是，因为无法从上游存储库提取通常程序包，所以 upstream 设置不适用于通用程序包。

- update-package-versions-status（请参阅更新程序包版本状态）

发布和使用通用程序包

要发布通用程序包版本及其相关资产，请使用 publish-package-version 命令。您可以使用 list-package-version-asset 命令列出通用程序包的资产，然后使用 get-package-version-asset 下载资产。以下主题包含使用这些命令发布通用程序包或下载通用程序包资产的分步说明。

发布通用程序包

通用程序包由程序包名称、命名空间、版本和一个或多个资产（或文件）组成。本主题演示如何发布名为 my-package、命名空间为 my-ns、版本为 1.0.0 且包含一个名为 asset.tar.gz 的资产的程序包。

先决条件：

- 使用 CodeArtifact 设置和配置 AWS Command Line Interface（请参阅设置 AWS CodeArtifact）
- 拥有 CodeArtifact 域和存储库（请参阅开始使用 AWS CLI）

发布通用程序包

1. 使用以下命令为要上传到程序包版本的每个文件生成 SHA256 哈希值，并将该值放入环境变量中。此值用作完整性检查，用来确认在最初发送后未更改文件内容。

   Linux

   ```
   export ASSET_SHA256=$(sha256sum asset.tar.gz | awk '{print $1;}')
   ```

   macOS

   ```
   export ASSET_SHA256=$(shasum -a 256 asset.tar.gz | awk '{print $1;}')
   ```

   Windows

   ```
   for /f "tokens=*" %G IN ('certUtil -hashfile asset.tar.gz SHA256 ^| findstr /v
    "hash"') DO SET "ASSET_SHA256=%G"
   ```

2. 调用 publish-package-version 来上传资产并创建新的程序包版本。

   > **Note**
   >
   > 如果您的程序包中包含多个资产，则可以为要上传的每个资产调用一次 publish-package-version。每次调用 publish-package-version 都包括 --unfinished 参数，上传最终资产时除外。省略 --unfinished 会将程序包版本的状态设置为 Published，并阻止将额外资产上传到该版本。

或者，在每次调用 publish-package-version 时都包括 --unfinished，然后使用 update-package-versions-status 命令将程序包版本的状态设置为 Published。

Linux/macOS

```
aws codeartifact publish-package-version --domain my_domain --repository my_repo
 \
      --format generic --namespace my-ns --package my-package --package-
version 1.0.0 \
      --asset-content asset.tar.gz --asset-name asset.tar.gz \
      --asset-sha256 $ASSET_SHA256
```

Windows

```
aws codeartifact publish-package-version --domain my_domain --repository my_repo
 ^
      --format generic --namespace my-ns --package my-package --package-
version 1.0.0 ^
      --asset-content asset.tar.gz --asset-name asset.tar.gz ^
      --asset-sha256 %ASSET_SHA256%
```

下面显示了输出。

```
{
    "format": "generic",
    "namespace": "my-ns",
    "package": "my-package",
    "version": "1.0.0",
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
    "status": "Published",
    "asset": {
        "name": "asset.tar.gz",
        "size": 11,
        "hashes": {
            "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
            "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
            "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
```

```
            "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95
SHA-512"
        }
    }
}
```

列出通用程序包资产

要列出通用程序包中包含的资产，请使用 `list-package-version-assets` 命令。有关更多信息，请参阅列出程序包版本资产。

以下示例列出程序包 `my-package` 的版本 `1.0.0` 的资产。

列出程序包版本资产

- 调用 `list-package-version-assets` 来列出通用程序包中包含的资产。

  Linux/macOS

  ```
  aws codeartifact list-package-version-assets --domain my_domain \
    --repository my_repo --format generic --namespace my-ns \
    --package my-package --package-version 1.0.0
  ```

  Windows

  ```
  aws codeartifact list-package-version-assets --domain my_domain ^
    --repository my_repo --format generic --namespace my-ns ^
    --package my-package --package-version 1.0.0
  ```

  下面显示了输出。

```
{
    "assets": [
        {
            "name": "asset.tar.gz",
            "size": 11,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
```

```
                "SHA-256":
 "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95
SHA-512"
            }
        }
    ],
    "package": "my-package",
    "format": "generic",
    "namespace": "my-ns",
    "version": "1.0.0",
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"
}
```

下载通用程序包资产

要从通用程序包下载资产，请使用 `get-package-version-asset` 命令。有关更多信息，请参阅[下载程序包版本资源](#)。

以下示例将资产 `asset.tar.gz` 从程序包 `my-package` 的版本 `1.0.0` 下载到当前工作目录中也命名为 `asset.tar.gz` 的文件中。

下载程序包版本资产

- 调用 `get-package-version-asset` 从通用程序包下载资产。

  Linux/macOS

  ```
  aws codeartifact get-package-version-asset --domain my_domain \
    --repository my_repo --format generic --namespace my-ns --package my-package \
    --package-version 1.0.0 --asset asset.tar.gz \
    asset.tar.gz
  ```

  Windows

  ```
  aws codeartifact get-package-version-asset --domain my_domain ^
    --repository my_repo --format generic --namespace my-ns --package my-package ^
    --package-version 1.0.0 --asset asset.tar.gz ^
    asset.tar.gz
  ```

下面显示了输出。

```
{
    "assetName": "asset.tar.gz",
    "packageVersion": "1.0.0",
    "packageVersionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"
}
```

将 CodeArtifact 与 CodeBuild 结合使用

这些主题介绍如何在 AWS CodeBuild 构建项目中使用 CodeArtifact 存储库中的程序包。

主题

在 CodeBuild 中使用 npm 程序包

以下步骤已使用 CodeBuild 提供的 Docker 映像中列出的操作系统进行了测试。

使用 IAM 角色设置权限

在 CodeBuild 中使用来自 CodeArtifact 的 npm 程序包时，需要执行这些步骤。

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。在角色页面上，编辑 CodeBuild 构建项目所使用的角色。此角色必须具有以下权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
                      "codeartifact:ReadFromRepository"
                      ],
          "Resource": "*"
      },
      {
          "Effect": "Allow",
          "Action": "sts:GetServiceBearerToken",
```

```
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

> **Important**
>
> 如果您还想使用 CodeBuild 来发布程序包，请添加 **`codeartifact:PublishPackageVersion`** 权限。

有关信息，请参阅《IAM 用户指南》中的修改角色。

登录并使用 npm

要使用 CodeBuild 中的 npm 程序包，请运行项目 `buildspec.yaml` 的 `pre-build` 部分的 `login` 命令来配置 npm，以便从 CodeArtifact 提取程序包。有关更多信息，请参阅 npm 身份验证。

成功运行 `login` 后，您可以运行 `build` 部分中的 npm 命令来安装 npm 程序包。

Linux

> **Note**
>
> 仅当您使用的是较旧的 CodeBuild 映像时，才需要使用 `pip3 install awscli --upgrade --user` 来升级 AWS CLI。如果您使用的是最新映像版本，则可以删除该行。

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333
 --repository my_repo
build:
```

```
  commands:
    - npm install
```

Windows

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool npm --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - npm install
```

在 CodeBuild 中使用 Python 程序包

以下步骤已使用 CodeBuild 提供的 Docker 映像中列出的操作系统进行了测试。

使用 IAM 角色设置权限

在 CodeBuild 中使用来自 CodeArtifact 的 Python 程序包时，需要执行这些步骤。

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。在角色页面上，编辑 CodeBuild 构建项目所使用的角色。此角色必须具有以下权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
```

```
                    "codeartifact:ReadFromRepository"
                    ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

> **Important**
>
> 如果您还想使用 CodeBuild 来发布程序包，请添加 **`codeartifact:PublishPackageVersion`** 权限。

有关信息，请参阅《IAM 用户指南》中的修改角色。

登录并使用 pip 或 twine

要使用 CodeBuild 中的 Python 程序包，请运行项目 `buildspec.yaml` 文件的 `pre-build` 部分的 `login` 命令来配置 `pip`，以便从 CodeArtifact 提取程序包。有关更多信息，请参阅将 CodeArtifact 与 Python 结合使用。

成功运行 `login` 后，您可以运行 `build` 部分中的 `pip` 命令来安装或发布 Python 程序包。

Linux

> **Note**
>
> 仅当您使用的是较旧的 CodeBuild 映像时，才需要使用 `pip3 install awscli --upgrade --user` 来升级 AWS CLI。如果您使用的是最新映像版本，则可以删除该行。

要使用 pip 安装 Python 程序包，请执行以下操作：

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool pip --domain my_domain --domain-owner 111122223333
 --repository my_repo
build:
  commands:
    - pip install requests
```

要使用 twine 发布 Python 程序包，请执行以下操作：

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool twine --domain my_domain --domain-
owner 111122223333 --repository my_repo
build:
  commands:
    - twine upload --repository codeartifact mypackage
```

Windows

要使用 pip 安装 Python 程序包，请执行以下操作：

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool pip --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - pip install requests
```

要使用 twine 发布 Python 程序包，请执行以下操作：

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool twine --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - twine upload --repository codeartifact mypackage
```

在 CodeBuild 中使用 Maven 程序包

使用 IAM 角色设置权限

在 CodeBuild 中使用来自 CodeArtifact 的 Maven 程序包时，需要执行这些步骤。

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。在角色页面上，编辑 CodeBuild 构建项目所使用的角色。此角色必须具有以下权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
                      "codeartifact:ReadFromRepository"
                      ],
          "Resource": "*"
      },
      {
          "Effect": "Allow",
          "Action": "sts:GetServiceBearerToken",
          "Resource": "*",
```

```
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

> **Important**
>
> 如果您还想使用 CodeBuild 发布程序包，请添加 **codeartifact:PublishPackageVersion** 和 **codeartifact:PutPackageMetadata** 权限。

有关信息，请参阅《IAM 用户指南》中的修改角色。

使用 gradle 或 mvn

要将 Maven 程序包与 `gradle` 或 `mvn` 一起使用，请将 CodeArtifact 身份验证令牌存储在环境变量中，如在环境变量中传递身份验证令牌中所述。以下是示例。

> **Note**
>
> 仅当您使用的是较旧的 CodeBuild 映像时，才需要使用 `pip3 install awscli --upgrade --user` 来升级 AWS CLI。如果您使用的是最新映像版本，则可以删除该行。

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output text`
```

要使用 Gradle，请执行以下操作：

如果在 Gradle `build.gradle` 文件中引用了 `CODEARTIFACT_AUTH_TOKEN` 变量，如将 CodeArtifact 与 Gradle 结合使用中所述，则可以从 `buildspec.yaml` `build` 部分调用 Gradle 构建。

```
build:
  commands:
    - gradle build
```

要使用 mvn，请执行以下操作：

您必须按照将 CodeArtifact 与 mvn 结合使用中的说明来配置 Maven 配置文件（`settings.xml` 和 `pom.xml`）。

在 CodeBuild 中使用 NuGet 程序包

以下步骤已使用 CodeBuild 提供的 Docker 映像中列出的操作系统进行了测试。

主题

- 使用 IAM 角色设置权限
- 使用 NuGet 程序包
- 使用 NuGet 程序包进行构建
- 发布 NuGet 程序包

使用 IAM 角色设置权限

在 CodeBuild 中使用来自 CodeArtifact 的 NuGet 程序包时，需要执行这些步骤。

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。在角色页面上，编辑 CodeBuild 构建项目所使用的角色。此角色必须具有以下权限。

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

```
            "Effect": "Allow",
            "Action": [ "codeartifact:GetAuthorizationToken",
                        "codeartifact:GetRepositoryEndpoint",
                        "codeartifact:ReadFromRepository"
                        ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

> **Important**
>
> 如果您还想使用 CodeBuild 来发布程序包，请添加 **`codeartifact:PublishPackageVersion`** 权限。

有关信息，请参阅《IAM 用户指南》中的修改角色。

使用 NuGet 程序包

要使用 CodeBuild 中的 NuGet 程序包，请在项目的 `buildspec.yaml` 文件中包括以下内容。

1. 在 `install` 部分中，安装 CodeArtifact 凭证提供程序来配置命令行工具（例如 `msbuild` 和 `dotnet`），以便生成程序包并将其发布到 CodeArtifact。
2. 在 `pre-build` 部分中，将 CodeArtifact 存储库添加到 NuGet 配置中。

请见以下 `buildspec.yaml` 示例。有关更多信息，请参阅将 CodeArtifact 与 NuGet 结合使用。

安装凭证提供程序并添加存储库源后，您可以从 `build` 部分运行 NuGet CLI 工具命令来使用 NuGet 程序包。

Linux

要通过 dotnet 使用 NuGet 程序包，请执行以下操作：

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      -  dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - dotnet add package <packageName> --source codeartifact
```

Windows

要通过 dotnet 使用 NuGet 程序包，请执行以下操作：

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - dotnet add package <packageName> --source codeartifact
```

使用 NuGet 程序包进行构建

要使用 CodeBuild 中的 NuGet 程序包进行构建，请在项目的 `buildspec.yaml` 文件中包括以下内容。

1. 在 `install` 部分中，安装 CodeArtifact 凭证提供程序来配置命令行工具（例如 `msbuild` 和 `dotnet`），以便生成程序包并将其发布到 CodeArtifact。
2. 在 `pre-build` 部分中，将 CodeArtifact 存储库添加到 NuGet 配置中。

请见以下 `buildspec.yaml` 示例。有关更多信息，请参阅将 CodeArtifact 与 NuGet 结合使用。

安装凭证提供程序并添加存储库源后，您可以从 `build` 部分运行 NuGet CLI 工具命令，例如 `dotnet build`。

Linux

要使用 `dotnet` 构建 NuGet 程序包，请执行以下操作：

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      -  dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - dotnet build
```

要使用 `msbuild` 构建 NuGet 程序包，请执行以下操作：

```
version: 0.2
```

```
phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - msbuild -t:Rebuild -p:Configuration=Release
```

Windows

要使用 `dotnet` 构建 NuGet 程序包，请执行以下操作：

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - dotnet build
```

要使用 `msbuild` 构建 NuGet 程序包，请执行以下操作：

```
version: 0.2

phases:
```

```
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - msbuild -t:Rebuild -p:Configuration=Release
```

发布 NuGet 程序包

要发布 CodeBuild 中的 NuGet 程序包，请在项目的 `buildspec.yaml` 文件中包括以下内容。

1. 在 `install` 部分中，安装 CodeArtifact 凭证提供程序来配置命令行工具（例如 `msbuild` 和 `dotnet`），以便生成程序包并将其发布到 CodeArtifact。
2. 在 `pre-build` 部分中，将 CodeArtifact 存储库添加到 NuGet 配置中。

请见以下 `buildspec.yaml` 示例。有关更多信息，请参阅将 CodeArtifact 与 NuGet 结合使用。

安装凭证提供程序并添加存储库源后，您可以从 `build` 部分运行 NuGet CLI 工具命令并发布 NuGet 程序包。

Linux

要使用 `dotnet` 发布 NuGet 程序包，请执行以下操作：

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
```

```
    commands:
      - dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - dotnet pack -o .
      - dotnet nuget push *.nupkg -s codeartifact
```

Windows

要使用 dotnet 发布 NuGet 程序包，请执行以下操作：

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - dotnet pack -o .
      - dotnet nuget push *.nupkg -s codeartifact
```

依赖项缓存

您可以在 CodeBuild 中启用本地缓存，从而减少每次构建时需要从 CodeArtifact 提取的依赖项数量。有关信息，请参阅《AWS CodeBuild 用户指南》中的在 AWS CodeBuild 中构建缓存。启用自定义本地缓存后，将缓存目录添加到项目的 buildspec.yaml 文件中。

例如，如果您使用的是 mvn，请使用以下内容。

```
cache:
  paths:
    - '/root/.m2/**/*'
```

对于其他工具，请使用下表中所示的缓存文件夹。

工具	缓存目录
mvn	/root/.m2/**/*
gradle	/root/.gradle/caches/**/*
pip	/root/.cache/pip/**/*
npm	/root/.npm/**/*
nuget	/root/.nuget/**/*
yarn (classic)	/root/.cache/yarn/**/*

监控 CodeArtifact

监控是保持 CodeArtifact 及您的其它 AWS 解决方案的可靠性、可用性和性能的重要方面。AWS 提供了以下一些监控工具来监控 CodeArtifact、在出现错误时进行报告并适时自动采取措施。

- 您可以使用 Amazon EventBridge 自动执行您的 AWS 服务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。AWS 服务中的事件将近乎实时传输到 EventBridge。您可以编写简单的规则来指示您关注的事件，并指示要在事件匹配规则时执行的自动化操作。有关更多信息，请参阅 Amazon EventBridge 用户指南和CodeArtifact 事件格式和示例。
- 您可以使用 Amazon CloudWatch 指标按操作查看 CodeArtifact 的使用情况。CloudWatch 指标包括向 CodeArtifact 发出的所有请求，并且按账户显示请求。您可以通过导航至使用情况/按 AWS 资源 AWS 命名空间在 CloudWatch 指标中查看这些指标。有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的使用 Amazon CloudWatch 指标。

主题

监控 CodeArtifact 事件

CodeArtifact 可与 Amazon EventBridge 集成，后者是一项可自动执行和响应事件（包括 CodeArtifact 存储库中的更改）的服务。您可以为事件创建规则并配置在事件匹配规则时会发生什么情况。EventBridge 以前称为 CloudWatch Events。

事件会触发以下操作：

- 调用 AWS Lambda 函数。
- 激活 AWS Step Functions 状态机。
- 通知 Amazon SNS 主题或 Amazon SQS 队列。
- 在 AWS CodePipeline 中启动管道。

当创建、修改或删除程序包版本时，CodeArtifact 会创建事件。以下是 CodeArtifact 事件的示例。

- 发布新的程序包版本（例如，通过运行 `npm publish`）。

- 向现有程序包版本添加新资产（例如，通过将新的 JAR 文件推送到现有 Maven 程序包）。
- 使用 `copy-package-versions` 将程序包版本从一个存储库复制到另一个存储库。有关更多信息，请参阅在存储库之间复制程序包。
- 使用 `delete-package-versions` 删除程序包版本。有关更多信息，请参阅删除包。
- 使用 `delete-package` 删除程序包版本。为已删除程序包的每个版本发布一个事件。有关更多信息，请参阅删除包。
- 如果程序包版本是从上游存储库获取，则在下游存储库中保留该版本。有关更多信息，请参阅在 CodeArtifact 中使用上游存储库。
- 将程序包版本从外部存储库摄取到 CodeArtifact 存储库。有关更多信息，请参阅将 CodeArtifact 存储库连接到公有存储库。

事件会同时发送给拥有该域的账户和管理存储库的账户。例如，假设账户 `111111111111` 拥有域 `my_domain`。账户 `222222222222` 在 `my_domain` 中创建名为 `repo2` 的存储库。当新的程序包版本发布到 `repo2` 时，两个账户都会收到 EventBridge 事件。拥有域的账户 (111111111111) 会收到域中所有存储库的事件。如果一个账户同时拥有域及其中的存储库，则只会传送一个事件。

以下主题说明 CodeArtifact 事件格式。这些主题介绍了如何配置 CodeArtifact 事件，以及如何将事件与其他 AWS 服务一起使用。有关更多信息，请参阅 Amazon EventBridge 用户指南中的 Amazon EventBridge 入门。

CodeArtifact 事件格式和示例

以下是事件字段和描述以及 CodeArtifact 事件的示例。

CodeArtifact 事件格式

所有 CodeArtifact 事件都包括下列字段。

事件字段	描述
version	事件格式的版本。目前只有一个版本 `0`。
id	事件的唯一标识符。
detail-type	事件类型。这决定了 `detail` 对象中的字段。目前支持的一个 `detail-type` 是 `CodeArtifact Package Version State Change`。

事件字段	描述
源	事件的源。对于 CodeArtifact 来说，源是 `aws.codeartifact` 。
account	接收事件的账户的 AWS 账户 ID。
time	触发事件的确切时间。
region	触发了事件的区域。
资源	一个列表，其中包含了已更改程序包的 ARN。该列表包含一个条目。有关程序包 ARN 格式的信息，请参阅授予对程序包的写入权限。
domainName	包含程序包的存储库所在的域。
domainOwner	域所有者的 AWS 账户 ID。
repositoryName	包含程序包的存储库。
repositoryAdministrator	存储库管理员的 AWS 账户 ID。
packageFormat	触发了事件的程序包的格式。
packageNamespace	触发了事件的程序包的命名空间。
packageName	触发了事件的程序包的名称。
packageVersion	触发了事件的程序包的版本。
packageVersionState	触发事件时程序包版本的状态。可能的值为 `Unfinished` 、`Published` 、`Unlisted`、`Archived` 和 `Disposed`。
packageVersionRevision	一个值，用于唯一地标识在触发事件时程序包版本的资产和元数据的状态。如果修改了程序包版本（例如，向 Maven 程序包添加另一个 JAR 文件），则 `packageVersionRevision` 会发生变化。

事件字段	描述
changes.assetsAdded	添加到触发了事件的程序包中的资产数量。资产的示例包括 Maven JAR 文件或 Python Wheel。
changes.assetsRemoved	从触发了事件的程序包中删除的资产数量。
changes.assetsUpdated	触发了事件的程序包中已修改的资产数量。
changes.metadataUpdated	一个布尔值，如果事件包括已修改的程序包级元数据，则设置为 `true`。例如，一个事件可能会修改 Maven `pom.xml` 文件。
changes.statusChanged	一个布尔值，如果修改了事件的 `packageVersionStatus`（例如，如果 `packageVersionStatus` 从 `Unfinished` 更改为 `Published`），则设置为 `true`。
operationType	描述程序包版本更改的概要类型。可能的值为 `Created`、`Updated` 和 `Deleted`。
sequenceNumber	一个整数，指定了程序包的事件编号。程序包中的每个事件都会使 `sequenceNumber` 递增，因此可以按顺序排列事件。事件可以使 `sequenceNumber` 按任意整数递增。 **Note** EventBridge 事件可能以乱序接收。`sequenceNumber` 可以用来确定它们的实际顺序。
eventDeduplicationId	一个 ID，用于区分重复 EventBridge 事件。在极少数情况下，EventBridge 可能会对单个事件或在预定的时间多次触发同一个规则。或者，EventBridge 可能会为给定的触发规则多次调用同一个目标。

CodeArtifact 事件示例

下面是发布 npm 程序包时可能会触发 CodeArtifact 事件的一个示例。

```
{
      "version":"0",
      "id":"73f03fec-a137-971e-6ac6-07c8ffffffff",
      "detail-type":"CodeArtifact Package Version State Change",
      "source":"aws.codeartifact",
      "account":"123456789012",
      "time":"2019-11-21T23:19:54Z",
      "region":"us-west-2",
      "resources":["arn:aws:codeartifact:us-west-2:111122223333:package/my_domain/
myrepo/npm//mypackage"],
      "detail":{
        "domainName":"my_domain",
        "domainOwner":"111122223333",
        "repositoryName":"myrepo",
        "repositoryAdministrator":"123456789012",
        "packageFormat":"npm",
        "packageNamespace":null,
        "packageName":"mypackage",
        "packageVersion":"1.0.0",
        "packageVersionState":"Published",
        "packageVersionRevision":"0E5DE26A4CD79FDF3EBC4924FFFFFFFF",
        "changes":{
          "assetsAdded":1,
          "assetsRemoved":0,
          "metadataUpdated":true,
          "assetsUpdated":0,
          "statusChanged":true
        },
        "operationType":"Created",
        "sequenceNumber":1,
        "eventDeduplicationId":"2mEO0A2Ke07rWUTBXk3CAiQhdTXF4N94LNaT/ffffff="
      }
    }
```

使用事件来启动 CodePipeline 执行

此示例演示如何配置 Amazon EventBridge 规则，以便在发布、修改或删除 CodeArtifact 存储库中的程序包版本时，AWS CodePipeline 执行会启动。

主题

配置 EventBridge 权限

您必须为 EventBridge 添加权限来使用 CodePipeline 调用您创建的规则。要使用 AWS Command Line Interface (AWS CLI) 添加这些权限，请按照《AWS CodePipeline 用户指南》的为 CodeCommit 源创建 CloudWatch 事件规则 (CLI) 中的步骤 1 进行操作。

创建 EventBridge 规则

要创建规则，请使用带有 `--name` 和 `--event-pattern` 参数的 `put-rule` 命令。事件模式指定与每个事件的内容相匹配的值。如果模式与事件匹配，则会触发目标。例如，以下模式匹配来自 `my_domain` 域中的 `myrepo` 存储库的 CodeArtifact 事件。

```
aws events put-rule --name MyCodeArtifactRepoRule --event-pattern \
    '{"source":["aws.codeartifact"],"detail-type":["CodeArtifact Package Version State
 Change"],
    "detail":{"domainName":["my_domain"],"domainOwner":
["111122223333"],"repositoryName":["myrepo"]}}'
```

创建 EventBridge 规则目标

以下命令将目标添加到规则中，这样当事件与规则匹配时，就会触发 CodePipeline 执行。对于 `RoleArn` 参数，请指定之前在本主题中创建的角色的 Amazon 资源名称 (ARN)。

```
aws events put-targets --rule MyCodeArtifactRepoRule --targets \
  'Id=1,Arn=arn:aws:codepipeline:us-west-2:111122223333:pipeline-name,
  RoleArn=arn:aws:iam::123456789012:role/MyRole'
```

使用事件来运行 Lambda 函数

此示例演示如何配置 EventBridge 规则，以便在发布、修改或删除 CodeArtifact 存储库中的程序包版本时，启动一个 AWS Lambda 函数。

有关更多信息，请参阅《Amazon EventBridge 用户指南》中的教程：使用 EventBridge 安排 AWS Lambda 函数。

主题

创建 EventBridge 规则

要创建启动 Lambda 函数的规则，请使用带有 `--name` 和 `--event-pattern` 选项的 `put-rule` 命令。以下模式在 `my_domain` 域中的任何存储库中的 `@types` 作用域内指定 npm 程序包。

```
aws events put-rule --name "MyCodeArtifactRepoRule" --event-pattern \
  '{"source":["aws.codeartifact"],"detail-type":["CodeArtifact Package Version State
 Change"],
  "detail":{"domainName":["my_domain"],"domainOwner":
["111122223333"],"packageNamespace":["types"],"packageFormat":["npm"]}}'
```

创建 EventBridge 规则目标

以下命令将目标添加到规则中，以便在事件与规则匹配时，运行 Lambda 函数。对于 `arn` 参数，请指定 Lambda 函数的 Amazon 资源名称 (ARN)。

```
aws events put-targets --rule MyCodeArtifactRepoRule --targets \
  Id=1,Arn=arn:aws:lambda:us-west-2:111122223333:function:MyLambdaFunction
```

配置 EventBridge 权限

使用 `add-permission` 命令向规则授予调用 Lambda 函数的权限。对于 `--source-arn` 参数，请指定您在本示例前面部分创建的规则的 ARN。

```
aws lambda add-permission --function-name MyLambdaFunction \\
  --statement-id my-statement-id --action 'lambda:InvokeFunction' \\
  --principal events.amazonaws.com \\
  --source-arn arn:aws:events:us-west-2:111122223333:rule/MyCodeArtifactRepoRule
```

CodeArtifact 中的安全性

AWS 十分重视云安全性。为了满足对安全性最敏感的组织的需求，我们打造了具有超高安全性的数据中心和网络架构。作为 AWS 客户，您也将从这些数据中心和网络架构受益。

安全性是 AWS 和您的共同责任。责任共担模型将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS 负责保护在 AWS 云中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 AWS Compliance Programs 的一部分。要了解适用于 CodeArtifact 的合规性计划，请参阅合规性计划范围内的 AWS 服务。
- 云中的安全性 - 您的责任由您使用的 AWS 服务决定。您还需要对其它因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 CodeArtifact 时应用责任共担模式。以下主题说明如何配置 CodeArtifact 来实现您的安全性和合规性目标。您还将了解如何使用其他 AWS 服务来协助您监控和保护 CodeArtifact 资源。

主题

AWS CodeArtifact 中的数据保护

AWS 责任共担模式适用于 AWS CodeArtifact 中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础设施。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 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。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的博客文章 AWS Shared Responsibility Model and GDPR。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。我们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 AWS 资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务 中的所有默认安全控制。
- 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。
- 如果您在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 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 140-2 版》。

我们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）。这包括处理 CodeArtifact 或其他 AWS 服务 时使用控制台、API、AWS CLI 或 AWS SDK。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，我们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。

数据加密

加密是 CodeArtifact 安全性的一个重要组成部分。某些加密（例如，针对传输中的数据的加密）是默认提供的，无需您执行任何操作。其他加密（例如，针对静态数据的加密）可在创建项目或构建时进行配置。

- 静态数据加密 - 存储在 CodeArtifact 中的所有资产均使用 AWS KMS keys（KMS 密钥）进行加密。这包括所有存储库的所有程序包中的所有资产。每个域使用一个 KMS 密钥来加密其所有资产。默认情况下，使用 AWS 托管 KMS 密钥，因此您无需创建 KMS 密钥。如果需要，可以使用自己创建和配置的客户管理的 KMS 密钥。有关更多信息，请参阅《AWS Key Management Service 用户指南》中的创建密钥和 AWS 密钥管理服务概念。在创建域时可以指定客户管理的 KMS 密钥。有关更多信息，请参阅使用中的域名 CodeArtifact。
- 传输中数据加密 - 客户与 CodeArtifact 之间以及 CodeArtifact 与其下游依赖项之间的所有通信均使用 TLS 加密进行保护。

流量隐私

您可以通过将 CodeArtifact 配置为使用接口 Virtual Private Cloud (VPC) 端点来提高 CodeArtifact 域及其所含资产的安全性。为此，您无需互联网网关、NAT 设备或虚拟私有网关。有关更多信息，请参阅使用 Amazon VPC 端点。有关 AWS PrivateLink 和 VPC 端点的更多信息，请参阅 AWS PrivateLink 和通过 PrivateLink 访问 AWS 服务。

监控 CodeArtifact

为了使 AWS CodeArtifact 和您的 AWS 解决方案保持可靠性、可用性和性能，监控是一个重要方面。您应该从 AWS 解决方案的各个部分收集监控数据，以便您可以更轻松地调试多点故障（如果发生）。AWS 提供了以下工具来监控您的 CodeArtifact 资源并对潜在事件作出响应：

主题

- 使用 AWS CloudTrail 记录 CodeArtifact API 调用

使用 AWS CloudTrail 记录 CodeArtifact API 调用

CodeArtifact 与 AWS CloudTrail 集成，后者提供某个用户、角色或 AWS 服务在 CodeArtifact 中所执行操作的记录。CloudTrail 可以将 CodeArtifact 的所有 API 调用作为事件进行记录，包括来自程序包管理器客户端的调用。

如果您创建跟踪记录，则可以使 CloudTrail 事件持续传送到 Amazon Simple Storage Service (Amazon S3) 存储桶（包括 CodeArtifact 事件）。如果您不配置跟踪，则仍可在 CloudTrail 控制台中的 Event history（事件历史记录）中查看最新事件。使用 CloudTrail 收集的信息，您可以确定向 CodeArtifact 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。

要了解有关 CloudTrail 的更多信息，请参阅《AWS CloudTrail 用户指南》。

CloudTrail 中的 CodeArtifact 信息

在您创建 AWS 账户时，将在该账户上启用 CloudTrail。当 CodeArtifact 中发生活动时，该活动将记录在 CloudTrail 事件中，并与其他 AWS 服务事件一同保存在事件历史记录中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅使用 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 CodeArtifact 的事件），请创建跟踪记录。通过跟踪，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有亚马逊云科技区域。此跟踪在 AWS 分区中记录所有区域中的事件，并将日志文

件传送至您指定的 Amazon S3 存储桶。您还可以配置其他 AWS 服务，以便进一步分析在 CloudTrail 日志中收集的事件数据并采取措施。有关更多信息，请参阅以下主题：

- 为您的 AWS 账户创建跟踪记录
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知

在您的 AWS 账户中启用 CloudTrail 日志记录时，对 CodeArtifact 操作的 API 调用在 CloudTrail 日志文件中跟踪，它们随其他 AWS 服务记录一起写入到这些文件中。CloudTrail 基于时间段和文件大小来确定何时创建并写入新文件。

CloudTrail 会记录所有 CodeArtifact 操作。例如，除了程序包管理器客户端命令外，对 `ListRepositories`（在 AWS CLI 中，输入 `aws codeartifact list-repositories`）、`CreateRepository`（`aws codeartifact create-repository`）和 `ListPackages`（`aws codeartifact list-packages`）操作的调用也会在 CloudTrail 日志文件中生成条目。程序包管理器客户端命令通常会向服务器发出多个 HTTP 请求。每个请求都会生成单独的 CloudTrail 日志事件。

跨账户传输 CloudTrail 日志

对于单个 API 调用，最多有三个不同的账户会接收 CloudTrail 日志：

- 发出请求的账户，例如调用 `GetAuthorizationToken` 的账户。
- 存储库管理员账户，例如调用 `ListPackages` 的存储库的管理员账户。
- 域所有者的账户，例如拥有域（包含调用 API 的存储库）的账户。

对于像 `ListRepositoriesInDomain` 这样针对域而不是针对特定存储库执行操作的 API，只有调用账户和域所有者的账户才会收到 CloudTrail 日志。对于像 `ListRepositories` 这样未获得任何资源授权的 API，只有调用方账户才会收到 CloudTrail 日志。

了解 CodeArtifact 日志文件条目

CloudTrail 日志文件可以包含一个或多个日志条目。每个条目列出了多个 JSON 格式的事件。一个日志事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。日志条目不是公用 API 调用的有序堆栈跟踪，因此它们不会以任何特定顺序显示。

主题

- 示例：调用 GetAuthorizationToken API 的日志条目

- 示例：获取 npm 程序包版本的日志条目

示例：调用 GetAuthorizationToken API 的日志条目

由 GetAuthorizationToken 创建的日志条目包括 requestParameters 字段中的域名。


```
{
  "eventVersion": "1.05",
  "userIdentity": {
      "type": "AssumedRole",
      "principalId": "AIDACKCEVSQ6C2EXAMPLE",
      "arn": "arn:aws:sts::123456789012:assumed-role/Console/example",
      "accountId": "123456789012",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "sessionContext": {
          "attributes": {
              "mfaAuthenticated": "false",
              "creationDate": "2018-12-11T13:31:37Z"
          },
          "sessionIssuer": {
              "type": "Role",
              "principalId": "AIDACKCEVSQ6C2EXAMPLE",
              "arn": "arn:aws:iam::123456789012:role/Console",
              "accountId": "123456789012",
              "userName": "Console"
          }
      }
  },
  "eventTime": "2018-12-11T13:31:37Z",
  "eventSource": "codeartifact.amazonaws.com",
  "eventName": "GetAuthorizationToken",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "205.251.233.50",
  "userAgent": "aws-cli/1.16.37 Python/2.7.10 Darwin/16.7.0 botocore/1.12.27",
  "requestParameters": {
      "domainName": "example-domain"
      "domainOwner": "123456789012"
  },
  "responseElements": {
      "sessionToken": "HIDDEN_DUE_TO_SECURITY_REASONS"
  },
  "requestID": "6b342fc0-5bc8-402b-a7f1-ffffffffffff",
  "eventID": "100fde01-32b8-4c2b-8379-ffffffffffff",
```

```
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

示例：获取 npm 程序包版本的日志条目

所有程序包管理器客户端（包括 **npm** 客户端）发出的请求都会记录额外数据，包括 `requestParameters` 字段中的域名、存储库名称和程序包名称。URL 路径和 HTTP 方法记录在 `additionalEventData` 字段中。

```
{
   "eventVersion": "1.05",
   "userIdentity": {
       "type": "AssumedRole",
       "principalId": "AIDACKCEVSQ6C2EXAMPLE",
       "arn": "arn:aws:sts::123456789012:assumed-role/Console/example",
       "accountId": "123456789012",
       "accessKeyId": "ASIAIJIOBJIBSREXAMPLE",
       "sessionContext": {
           "attributes": {
               "mfaAuthenticated": "false",
               "creationDate": "2018-12-17T02:05:16Z"
           },
           "sessionIssuer": {
               "type": "Role",
               "principalId": "AIDACKCEVSQ6C2EXAMPLE",
               "arn": "arn:aws:iam::123456789012:role/Console",
               "accountId": "123456789012",
               "userName": "Console"
           }
       }
   },
   "eventTime": "2018-12-17T02:05:46Z",
   "eventSource": "codeartifact.amazonaws.com",
   "eventName": "ReadFromRepository",
   "awsRegion": "us-west-2",
   "sourceIPAddress": "205.251.233.50",
   "userAgent": "npm/6.14.15 node/v12.22.9 linux x64 ci/custom",
   "requestParameters": {
       "domainName": "example-domain",
       "domainOwner": "123456789012",
       "repositoryName": "example-repo",
```

```
        "packageName": "lodash",
        "packageFormat": "npm",
        "packageVersion": "4.17.20"
    },
    "responseElements": null,
    "additionalEventData": {
        "httpMethod": "GET",
        "requestUri": "/npm/lodash/-/lodash-4.17.20.tgz"
    },
    "requestID": "9f74b4f5-3607-4bb4-9229-ffffffffffff",
    "eventID": "c74e40dd-8847-4058-a14d-ffffffffffff",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


AWS CodeArtifact 的合规性验证

要了解某个 AWS 服务 是否在特定合规性计划范围内，请参阅合规性计划范围内的 AWS 服务，然后选择您感兴趣的合规性计划。有关常规信息，请参阅 AWS 合规性计划。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅在 AWS Artifact 中下载报告、。

您使用 AWS 服务 的合规性责任取决于数据的敏感度、贵公司的合规性目标以及适用的法律法规。AWS 提供以下资源来帮助满足合规性：

- 安全性与合规性快速入门指南——这些部署指南讨论了架构注意事项，并提供了在 AWS 上部署以安全性和合规性为重点的基准环境的步骤。
- Amazon Web Services 上的 HIPAA 安全性和合规性架构设计 – 该白皮书介绍了公司如何使用 AWS 创建符合 HIPAA 标准的应用程序。

> **Note**
>
> 并非所有 AWS 服务 都符合 HIPAA 要求。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS合规性资源 – 此业务手册和指南集合可能适用于您的行业和位置。

- AWS 客户合规指南：从合规角度了解责任共担模式。这些指南总结了保护 AWS 服务 的最佳实践，并将指南映射到跨多个框架的安全控制，包括美国国家标准与技术研究院（NIST）、支付卡行业安全标准委员会（PCI）和国际标准化组织（ISO）。
- 《AWS Config 开发人员指南》中的使用规则评估资源 – 此 AWS Config 服务评测您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub——此 AWS 服务 向您提供 AWS 中安全状态的全面视图。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实操。有关受支持服务及控制的列表，请参阅 Security Hub 控制参考。
- AWS Audit Manager – 此 AWS 服务 可帮助您持续审计您的 AWS 使用情况，以简化管理风险以及与相关法规和行业标准的合规性的方式。

AWS CodeArtifact 身份验证和令牌

CodeArtifact 要求用户通过服务进行身份验证才能发布或使用软件包版本。您必须使用您的 AWS 证书创建授权令牌，从而向 CodeArtifact 服务进行身份验证。要创建授权令牌，必须具有适当的权限。有关创建授权令牌所需的权限，请参阅 AWS CodeArtifact 权限参考中的 `GetAuthorizationToken` 条目。有关 CodeArtifact 权限的更多一般信息，请参阅如何 AWS CodeArtifact 与 IAM 配合使用。

要从中获取授权令牌 CodeArtifact，必须调用 GetAuthorizationToken API。使用 AWS CLI，您可以使用`login`或`get-authorization-token`命令`GetAuthorizationToken`进行调用。

> **Note**
>
> Root 用户无法调用 `GetAuthorizationToken`。

- `aws codeartifact login`：此命令可以轻松配置常用包管理器，以便 CodeArtifact 在单个步骤中使用。调用`login`会使用获取令牌，`GetAuthorizationToken`并使用令牌和正确的 CodeArtifact 存储库端点配置您的包管理器。支持包管理器如下：
  - dotnet
  - npm
  - nuget
  - pip
  - 迅速
  - 缠绕

- `aws codeartifact get-authorization-token`：对于 `login` 不支持的程序包管理器，您可以直接调用 `get-authorization-token`，然后根据需要使用令牌配置程序包管理器，例如，将其添加到配置文件或将其存储为环境变量。

CodeArtifact 授权令牌的默认有效期为 12 小时。令牌的有效期限可以配置为 15 分钟至 12 小时。当有效期限到期时，您必须获取另一个令牌。令牌的有效期限从 `login` 之后或调用 `get-authorization-token` 之后开始。

如果在代入角色时调用 `login` 或 `get-authorization-token`，则可以通过将 `--duration-seconds` 的值设置为 `0`，将令牌的有效期限配置为与角色会话持续时间中的剩余时间相等。否则，令牌有效期限与角色的最长会话持续时间无关。例如，假设您调用`sts assume-role`并指定会话持续时间为 15 分钟，然后调`login`用获取 CodeArtifact 授权令牌。在这种情况下，令牌在整整 12 小时内有效，即使此时间比 15 分钟的会话持续时间更长。有关控制会话持续时间的信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

使用 `login` 命令创建的令牌

该`aws codeartifact login`命令将使用获取令牌，`GetAuthorizationToken`并使用令牌和正确的 CodeArtifact 存储库端点配置您的包管理器。

下表说明了 `login` 命令的参数。

参数	必需	描述
`--tool`	支持	要进行身份验证的程序包管理器。可能的值为`dotnet`、`npm`、`nuget`、`pip`、`swift`和`twin`
`--domain`	支持	存储库所属域的名称。
`--domain-owner`	不支持	域所有者的 ID。如果访问的域名归您未通过身份验证的 AWS 账户所有，则必须使用此参数。有关更多信息，请参阅 跨账户域。
`--repository`	支持	要进行身份验证的存储库的名称。
`--duration-seconds`	不支持	登录信息的有效时间，以秒为单位。最小值为 900*，最大值为 43200。

参数	必需	描述
--namespace	不支持	将命名空间与您的存储库工具相关联。
--dry-run	不支持	仅输出为了将工具与存储库连接而执行的命令，而不对配置进行任何更改。

*在代入角色时如果调用 login，则值为 0 也同样有效。使用 --duration-seconds 0 调用 login 会创建一个令牌，该令牌的有效期限与代入角色的会话持续时间中的剩余时间相等。

以下示例说明了如何使用 login 命令提取授权令牌。

```
aws codeartifact login \
    --tool dotnet | npm | nuget | pip | swift | twine \
    --domain my_domain \
    --domain-owner 111122223333 \
    --repository my_repo
```

有关如何在 npm 中使用 login 命令的具体指导，请参阅在 CodeArtifact 中配置和使用 npm。对于 Python，请参阅将 CodeArtifact 与 Python 结合使用。

使用 GetAuthorizationToken API 创建的令牌

您可以get-authorization-token致电获取授权令牌 CodeArtifact。

```
aws codeartifact get-authorization-token \
    --domain my_domain \
    --domain-owner 111122223333 \
    --query authorizationToken \
    --output text
```

您可以使用 --duration-seconds 参数来更改令牌的有效期限。最小值为 900，最大值为 43200。以下示例创建一个持续 1 小时（3600 秒）的令牌。

```
aws codeartifact get-authorization-token \
    --domain my_domain \
```

```
    --domain-owner 111122223333 \
    --query authorizationToken \
    --output text \
    --duration-seconds 3600
```

如果在代入角色时调用 get-authorization-token，则令牌有效期限与角色的最长会话持续时间无关。通过将 --duration-seconds 设置为 0，可以将令牌配置为在代入角色的会话持续时间到期时过期。

```
aws codeartifact get-authorization-token \
    --domain my_domain \
    --domain-owner 111122223333 \
    --query authorizationToken \
    --output text \
    --duration-seconds 0
```

有关更多信息，请参阅以下文档：

- 有关令牌和环境变量的指导，请参阅使用环境变量传递身份验证令牌。
- 对于 Python 用户，请参阅不使用 login 命令配置 pip或在 CodeArtifact 中配置和使用 twine。
- 对于 Maven 用户，请参阅在 Gradle 中使用 CodeArtifact或在 mvn 中使用 CodeArtifact。
- 对于 npm 用户，请参阅不使用 login 命令配置 npm。

使用环境变量传递身份验证令牌

AWS CodeArtifact 使用 GetAuthorizationToken API 提供的授权令牌对来自 Maven 和 Gradle 等构建工具的请求进行身份验证和授权。有关这些身份验证令牌的更多信息，请参阅使用 GetAuthorizationToken API 创建的令牌。

您可以将这些身份验证令牌存储在环境变量中，构建工具可以读取该变量，以获取从存储库中获取包或向 CodeArtifact 仓库发布软件包所需的令牌。

出于安全考虑，与将令牌存储在可能被其他用户或进程读取或意外提交到源代码控制的文件中相比，这种方法更好。

1. 按照中所述配置您的 AWS 证书安装或升级 AWS CLI，然后对其进行配置。
2. 设置 CODEARTIFACT_AUTH_TOKEN 环境变量：

> **Note**
>
> 在某些情况下，不需要包括 `--domain-owner` 参数。有关更多信息，请参阅 跨账户域。

- macOS 或 Linux：

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text`
```

- Windows（使用默认命令 shell）：

```
for /f %i in ('aws codeartifact get-authorization-token --domain my_domain --
domain-owner 111122223333 --query authorizationToken --output text') do set
 CODEARTIFACT_AUTH_TOKEN=%i
```

- 窗户 PowerShell：

```
$env:CODEARTIFACT_AUTH_TOKEN = aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text
```

撤销 CodeArtifact 授权令牌

当经过身份验证的用户创建 CodeArtifact 用于访问资源的令牌时，该令牌将持续到其可自定义的访问期结束为止。默认访问期为 12 小时。在某些情况下，您可能想要在访问期到期之前撤销对令牌的访问权限。您可以按照以下说明撤消对 CodeArtifact 资源的访问权限。

如果您使用临时安全凭证（例如代入角色或联合用户访问权限）创建访问令牌，则可以通过更新 IAM 策略来拒绝访问，从而撤销访问权限。有关信息，请参阅《IAM 用户指南》中的禁用临时安全凭证的权限。

如果您使用长期 IAM 用户凭证来创建访问令牌，则必须修改用户的策略来拒绝访问或删除 IAM 用户。有关更多信息，请参阅更改 IAM 用户的权限或删除 IAM 用户。

AWS CodeArtifact 中的故障恢复能力

AWS全球基础设施围绕AWS区域和可用区构建。AWS区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。AWS CodeArtifact 在多个可用区运行，并将构件和元数据存储在 Amazon S3 和 Amazon DynamoDB 中。您的数据以冗余方式存储在多个设施中，以及各个设施内的多个设备上。

有关 AWS 区域和可用区的更多信息，请参阅 AWS 全球基础设施。

AWS CodeArtifact 中的基础设施安全性

作为一项托管式服务，AWS CodeArtifact 受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础架构的信息，请参阅 AWS 云安全。要按照基础设施安全最佳实践设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的 基础设施保护。

您可以使用 AWS 发布的 API 调用通过网络访问 CodeArtifact。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（Ephemeral Diffie-Hellman）或 ECDHE（Elliptic Curve Ephemeral 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 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service (AWS STS) 生成临时安全凭证来对请求进行签名。

依赖项替换攻击

程序包管理器简化了打包和共享可重用代码的过程。这些程序包可能是组织为了在其应用程序中使用而开发的私有程序包，也可能是公有程序包（通常是在组织外部开发并由公有程序包存储库分发的开源程序包）。在请求程序包时，开发人员依靠他们的程序包管理器来提取其依赖项的新版本。依赖项替换攻击也称为依赖项混淆攻击，该攻击利用了这样一个事实，即程序包管理器通常无法区分程序包的合法版本和恶意版本。

依赖项替换攻击是被称为软件供应链攻击的黑客攻击的其中一种形式。软件供应链攻击是一种利用软件供应链中任何地方的漏洞进行的攻击。

依赖项替换攻击可以针对任何人，包括使用内部开发的程序包和从公有存储库提取的程序包的用户。攻击者识别内部程序包名称，然后策略性地将同名的恶意代码放置在公有程序包存储库中。通常，恶意代码在版本号较高的程序包中发布。因为程序包管理器认为恶意程序包是程序包的最新版本，所以从这些

公有源中提取恶意代码。这种行为会导致期望程序包和恶意程序包之间发生“混淆”或“替换”，进而导致代码被篡改。

为了防止依赖项替换攻击，AWS CodeArtifact 提供了程序包来源控制。程序包来源控制是用于控制如何将程序包添加到存储库的设置。将新程序包的第一个程序包版本添加到 CodeArtifact 存储库时，会自动配置控制设置。这些控制设置可以确保程序包版本不能既直接发布到您的存储库中，又从公有来源摄取版本，从而保护您免受依赖项替换攻击的威胁。有关程序包来源控制以及如何更改该设置的更多信息，请参阅编辑程序包来源控制。

适用于 Identity and Access 管理 AWS CodeArtifact

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（拥有权限）使用 CodeArtifact 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您所做的工作 CodeArtifact。

服务用户-如果您使用 CodeArtifact 服务完成工作，则管理员会为您提供所需的凭证和权限。当你使用更多 CodeArtifact 功能来完成工作时，你可能需要额外的权限。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。如果您无法访问中的功能 CodeArtifact，请参阅对 AWS CodeArtifact 身份和访问进行故障排除。

服务管理员-如果您负责公司的 CodeArtifact 资源，则可能拥有完全访问权限 CodeArtifact。您的工作是确定您的服务用户应访问哪些 CodeArtifact 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。要详细了解您的公司如何使用 IAM CodeArtifact，请参阅如何 AWS CodeArtifact 与 IAM 配合使用。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理访问权限 CodeArtifact。要查看您可以在 IAM 中使用的 CodeArtifact 基于身份的策略示例，请参阅。基于身份的策略示例 AWS CodeArtifact

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （ IAM Identity Center ）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 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 AWS 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 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 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《AWS IAM Identity Center 用户指南》中的 多重身份验证 和《IAM 用户指南》中的 在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 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 AWS 服务 和资源。此身份被称为 AWS 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。有关要求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。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，我们建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户和应用程序中使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？。

IAM 用户和群组

I AM 用户是您 AWS 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，我们建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅 IAM 用户指南中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的 为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。

- 跨账户存取 – 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 AWS 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 角色与基于资源的策略有何不同。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 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 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 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个操作的权限。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 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 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 AWS 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 EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅 IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 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 AWS 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 AWS 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 policy。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

IAM policy 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 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅 IAM 用户指南中的创建 IAM policy。

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 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Simple Storage Service（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 AWS 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 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅 Amazon Simple Storage Service 开发人员指南 中的访问控制列表（ACL）概览。

其它策略类型

AWS 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 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有特征，则可对任意或全部账户应用服务控制策略（SCP）。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅 AWS Organizations 用户指南中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合身份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 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

如何 AWS CodeArtifact 与 IAM 配合使用

在使用 IAM 管理访问权限之前 CodeArtifact，请先了解有哪些 IAM 功能可供使用 CodeArtifact。

您可以搭配使用的 IAM 功能 AWS CodeArtifact

IAM 功能	CodeArtifact 支持
基于身份的策略	是
基于资源的策略	支持
策略操作	是
策略资源	支持

IAM 功能	CodeArtifact 支持
策略条件键（特定于服务）	不支持
ACL	否
ABAC（策略中的标签）	部分
临时凭证	是
主体权限	支持
服务角色	否
服务相关角色	不支持

要全面了解 CodeArtifact 以及其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

基于身份的策略 CodeArtifact

支持基于身份的策略	是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅 IAM 用户指南中的创建 IAM policy。

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。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

基于身份的策略示例 CodeArtifact

要查看 CodeArtifact 基于身份的策略的示例，请参阅。基于身份的策略示例 AWS CodeArtifact

内部基于资源的政策 CodeArtifact

支持基于资源的策略	支持

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 Simple Storage Service（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 AWS 服务。

要启用跨账户存取，您可以将整个账户或其它账户中的 IAM 实体指定为基于资源的策略中的主体。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。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们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个账户中的主体授予访问权限，则不需要额外的基于身份的策略。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。

的政策行动 CodeArtifact

支持策略操作	支持

管理员可以使用 AWS 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 AWS 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。

要查看 CodeArtifact 操作列表，请参阅《服务授权参考》 AWS CodeArtifact中定义的操作。

正在执行的策略操作在操作前 CodeArtifact 使用以下前缀：

```
codeartifact
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "codeartifact:action1",
      "codeartifact: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": "codeartifact:Describe*"
```

要查看 CodeArtifact 基于身份的策略的示例，请参阅。基于身份的策略示例 AWS CodeArtifact

的政策资源 CodeArtifact

支持策略资源	支持

管理员可以使用 AWS 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": "*"
```

要查看 CodeArtifact 资源类型及其 ARN 的列表，请参阅《服务授权参考》 AWS CodeArtifact中定义的资源。要了解您可以使用哪些操作来指定每种资源的 ARN，请参阅由定义的操作。 AWS CodeArtifact要查看在策略中指定 CodeArtifact 资源 ARN 的示例，请参阅AWS CodeArtifact 资源和操作。

的策略条件密钥 CodeArtifact

支持特定于服务的策略条件密钥	不支持

管理员可以使用 AWS 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 AWS 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 AWS 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 policy 元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

> **Note**
>
> AWS CodeArtifact 不支持以下 AWS 全局条件上下文密钥：
>
> - 引用
> - UserAgent

要查看 CodeArtifact 条件键列表，请参阅《服务授权参考》 AWS CodeArtifact中的条件密钥。要了解可以使用条件键的操作和资源，请参阅由定义的操作 AWS CodeArtifact。

要查看 CodeArtifact 基于身份的策略的示例，请参阅。基于身份的策略示例 AWS CodeArtifact

输入的 ACL CodeArtifact

支持 ACL	否

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

ABAC with CodeArtifact

支持 ABAC（策略中的标签）	部分

基于属性的访问权限控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以向 IAM 实体（用户或角色）和许多 AWS 资源附加标签。标记实体和资源是

ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为 Yes（是）。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为 Partial（部分）。

有关 ABAC 的更多信息,请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

有关为 CodeArtifact资源添加标签的更多信息，包括基于身份的策略示例，用于根据资源上的标签限制对资源的访问权限，请参阅。使用标签控制对 CodeArtifact 资源的访问

将临时证书与 CodeArtifact

支持临时凭证	支持

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IAM 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的 切换到角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

的跨服务主体权限 CodeArtifact

支持转发访问会话 (FAS)	支持

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下

游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 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个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

有两个 CodeArtifact API 操作需要调用方拥有其他服务的权限：

1. `GetAuthorizationToken` 需要 `sts:GetServiceBearerToken` 和 `codeartifact:GetAuthorizationToken` 权限。
2. `CreateDomain`，在提供非默认加密密钥时，需要 KMS 密钥的 `kms:DescribeKey` 和 `kms:CreateGrant` 权限，以及 `codeartifact:CreateDomain` 权限。

有关中操作所需权限和资源的更多信息 CodeArtifact，请参阅AWS CodeArtifact 权限参考。

CodeArtifact 的服务角色

支持服务角色	否

服务角色是由一项服务担任、代表您执行操作的 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 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会中断 CodeArtifact 功能。只有在 CodeArtifact 提供操作指导时才编辑服务角色。

的服务相关角色 CodeArtifact

支持服务相关角色	不支持

服务相关角色是一种链接到的服务角色。 AWS 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 AWS 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。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找服务相关角色列中包含 `Yes` 的表。选择是链接以查看该服务的服务相关角色文档。

基于身份的策略示例 AWS CodeArtifact

默认情况下，用户和角色无权创建或修改 CodeArtifact资源。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS 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 policy。管理员随后可以向角色添加 IAM policy，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 policy。

有关由定义的操作和资源类型的详细信息 CodeArtifact，包括每种资源类型的 ARN 格式，请参阅《服务授权参考》 AWS CodeArtifact中的操作、资源和条件密钥。

主题

策略最佳实践

基于身份的策略决定了某人是否可以在您的账户中创建、访问或删除 CodeArtifact 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 开始使用 AWS 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AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 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 AWS 托管式策略 或 工作职能的AWS 托管式策略。
- 应用最低权限 – 在使用 IAM policy 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 policy 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 AWS 服务的（例如）使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 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 policy，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 policy 语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证 (MFA)-如果 AWS 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 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

使用 CodeArtifact控制台

要访问 AWS CodeArtifact 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 CodeArtifact 资源的详细信息 AWS 账户。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。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用 CodeArtifact 控制台，还需要将`AWSCodeArtifactAdminAccess`或`AWSCodeArtifactReadOnlyAccess` AWS 托管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

AWS 托管（预定义）策略 AWS CodeArtifact

AWS 通过提供由创建和管理的独立 IAM 策略来解决许多常见用例 AWS。这些 AWS 托管策略为常见用例授予必要的权限，因此您可以不必调查需要哪些权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略。

以下 AWS 托管策略是特定的，您可以将其附加到账户中的用户 AWS CodeArtifact。

- `AWSCodeArtifactAdminAccess`— 提供对域名的完全访问权限， CodeArtifact 包括管理 CodeArtifact 域的权限。

```
{
```

```
   "Version": "2012-10-17",
   "Statement": [
      {
         "Action": [
            "codeartifact:*"
         ],
         "Effect": "Allow",
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sts:AWSServiceName": "codeartifact.amazonaws.com"
            }
         }
      }
   ]
}
```

- `AWSCodeArtifactReadOnlyAccess`— 提供对的只读访问权限 CodeArtifact。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Action": [
            "codeartifact:Describe*",
            "codeartifact:Get*",
            "codeartifact:List*",
            "codeartifact:ReadFromRepository"
         ],
         "Effect": "Allow",
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sts:AWSServiceName": "codeartifact.amazonaws.com"
```

```
            }
         }
      }
   ]
}
```

要创建和管理 CodeArtifact 服务角色，还必须附加名为的 AWS 托管策略`IAMFullAccess`。

您也可以创建自己的自定义 IAM 策略以授予 CodeArtifact操作和资源的权限。您可以将这些自定义策略附加到需要这些权限的 IAM 用户或组。

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 AWS CLI 或 AWS 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": "*"
        }
    ]
}
```

允许用户获取有关存储库和域的信息

以下策略允许 IAM 用户或角色列出和描述任何类型的 CodeArtifact 资源，包括域、存储库、包和资产。该策略还包括codeArtifact:ReadFromRepository权限，允许委托人从 CodeArtifact 存储库中获取软件包。该策略不允许创建新域或存储库，也不允许发布新的程序包。

调用 GetAuthorizationToken API 需要 codeartifact:GetAuthorizationToken 和 sts:GetServiceBearerToken 权限。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codeartifact:List*",
            "codeartifact:Describe*",
            "codeartifact:Get*",
            "codeartifact:Read*"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sts:AWSServiceName": "codeartifact.amazonaws.com"
            }
         }
      }
   ]
}
```

允许用户获取有关特定域的信息

以下是一个权限策略示例，仅允许用户列出位于账户 123456789012 的 us-east-2 区域中名称以 my 开头的任何域。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "codeartifact:ListDomains",
         "Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/my*"
      }
   ]
}
```

允许用户获取有关特定存储库的信息

以下是一个权限策略的示例，允许用户获取有关以 test 结尾的存储库的信息，包括有关存储库中的程序包的信息。用户将无法发布、创建或删除资源。

调用 GetAuthorizationToken API 需要 codeartifact:GetAuthorizationToken 和 sts:GetServiceBearerToken 权限。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codeartifact:List*",
            "codeartifact:Describe*",
            "codeartifact:Get*",
            "codeartifact:Read*"
         ],
         "Resource": "arn:aws:codeartifact:*:*:repository/*/*test"
      },
      {
         "Effect": "Allow",
         "Action": [
            "codeartifact:List*",
            "codeartifact:Describe*"
```

```
            ],
            "Resource": "arn:aws:codeartifact:*:*:package/*/*test/*/*/*"
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "codeartifact:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```

限制授权令牌持续时间

用户必须使用授权令牌进行身份验证才能 CodeArtifact 发布或使用软件包版本。授权令牌仅在其配置的有效期限内有效。令牌的默认有效期限为 12 小时。有关授权令牌的更多信息，请参阅AWS CodeArtifact 身份验证和令牌。

提取令牌时，用户可以配置令牌的有效期限。授权令牌有效期限的有效值为 `0`，以及介于 `900`（15 分钟）和 `43200`（12 小时）之间的任意数字。如果值为 `0`，则会创建一个持续时间与用户角色的临时凭证相等的令牌。

管理员可以使用附加到用户或组的权限策略中的 `sts:DurationSeconds` 条件键来限制授权令牌有效期限的有效值。如果用户尝试创建有效期限超出有效值的授权令牌，则令牌创建会失败。

以下示例策略限制了 CodeArtifact 用户创建的授权令牌可能的持续时间。

示例策略：将令牌有效期限限制为正好 12 小时（43200 秒）

使用此策略，用户只能创建有效期限为 12 小时的授权令牌。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codeartifact:*",
            "Resource": "*"
        },
        {
            "Sid": "sts",
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "NumericEquals": {
                    "sts:DurationSeconds": 43200
                },
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

策略示例：将令牌的有效期限制在 15 分钟至 1 小时之间，或者等于用户的临时凭证期限

通过此策略，用户将能够创建有效期为 15 分钟至 1 小时的令牌。用户还可以通过将 --durationSeconds 指定为 0，从而创建持续时间与其角色的临时凭证持续时间相等的令牌。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codeartifact:*",
            "Resource": "*"
        },
        {
            "Sid": "sts",
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
```

```
                "NumericLessThanEquals": {
                    "sts:DurationSeconds": 3600
                },
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

使用标签控制对 CodeArtifact 资源的访问

IAM 用户策略语句中的条件包含在您在指定 CodeArtifact 操作所需资源的权限时所用的语法中。使用条件中的标签是控制对资源和请求的访问的一种方法。有关为 CodeArtifact 资源添加标签的信息，请参阅标记资源。本主题讨论了基于标签的访问控制。

在设计 IAM 策略时，您可以通过授予对特定资源的访问权限来设置精细权限。但随着您管理的资源数量的增加，此任务会变得日益复杂。标记资源并在策略声明条件中使用标签可以简化这一任务。您可以向具有特定标签的任何资源批量授予访问权限。然后，在创建期间或之后，您可以将此标签反复应用到相关资源。

标签可以附加到资源，也可以从请求传入支持标签的服务。在 CodeArtifact 中，资源可以具有标签，而且某些操作可以包括标签。在创建 IAM 策略时，您可以使用标签条件键来控制：

- 基于资源已有的标签，哪些用户可以对域或存储库资源执行操作。
- 哪些标签可以在操作的请求中传递。
- 是否特定标签键可在请求中使用。

有关标签条件键的完整请求和语义，请参阅《IAM 用户指南》中的使用标签控制访问。

> **Important**
>
> 对资源使用标签来限制操作时，标签必须位于对其执行了操作的资源上。例如，要使用标签拒绝 `DescribeRepository` 权限，标签必须位于每个存储库上，而不是域上。有关 CodeArtifact 中的操作列表以及对哪些资源进行操作，请参阅AWS CodeArtifact 权限参考。

基于标签的访问控制示例

以下示例演示了如何为 CodeArtifact 用户指定策略中的标签条件。

Example 1：基于请求中的标签限制操作

AWSCodeArtifactAdminAccess 托管式用户策略可以为用户赋予不受限的权限，允许其对所有资源执行任何 CodeArtifact 操作。

除非请求包含某些标签，否则以下策略会限制此权限并拒绝未经授权的用户创建存储库的权限。为此，如果请求未指定一个名为 costcenter、值为 1 或 2 的标签，则拒绝 CreateRepository 操作。除托管用户策略外，客户的管理员还必须将此 IAM 策略附加到未经授权的 IAM 用户。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "codeartifact:CreateRepository",
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:RequestTag/costcenter": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "codeartifact:CreateRepository",
      "Resource": "*",
      "Condition": {
          "ForAnyValue:StringNotEquals": {
              "aws:RequestTag/costcenter": [
                  "1",
                  "2"
              ]
          }
      }
  }
  ]
}
```

Example 2：基于资源标签限制操作

AWSCodeArtifactAdminAccess 托管式用户策略可以为用户赋予不受限的权限，允许其对所有资源执行任何 CodeArtifact 操作。

以下策略限制此权限并拒绝未经授权的用户对指定域中的存储库执行操作的权限。为此，如果资源具有名为 Key1 的带有值 Value1 或 Value2 之一的标签，那么它会拒绝某些操作。（使用 aws:ResourceTag 条件键，基于资源上的标签控制对这些资源的访问。）除托管用户策略外，客户的管理员还必须将此 IAM 策略附加到未经授权的 IAM 用户。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codeartifact:TagResource",
        "codeartifact:UntagResource",
        "codeartifact:DescribeDomain",
        "codeartifact:DescribeRepository",
        "codeartifact:PutDomainPermissionsPolicy",
        "codeartifact:PutRepositoryPermissionsPolicy",
        "codeartifact:ListRepositoriesInDomain",
        "codeartifact:UpdateRepository",
        "codeartifact:ReadFromRepository",
        "codeartifact:ListPackages",
        "codeartifact:ListTagsForResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Key1": ["Value1", "Value2"]
        }
      }
    }
  ]
}
```

Example 3：基于资源标签允许操作

以下策略授予用户对 CodeArtifact 中的存储库和程序包执行操作以及获取其相关信息的权限。

为此，如果存储库具有名为 Key1、值为 Value1 的标签，则允许特定操作。（aws:RequestTag 条件键用于控制可以通过 IAM 请求传递哪些标签。）aws:TagKeys 条件确保标签键区分大小写。对于未附加 AWSCodeArtifactAdminAccess 托管用户策略的 IAM 用户，此策略非常有用。托管式策略可以为用户赋予不受限的权限，允许其对所有资源执行任何 CodeArtifact 操作。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeartifact:UpdateRepository",
        "codeartifact:DeleteRepository",
        "codeartifact:ListPackages"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Key1": "Value1"
        }
      }
    }
  ]
}
```

Example 4：基于请求中的标签允许操作

以下策略授予用户在 CodeArtifact 中的指定域创建存储库的权限。

为此，如果请求中的创建资源 API 指定一个名为 Key1、值为 Value1 的标签，则允许 CreateRepository 和 TagResource 操作。（aws:RequestTag 条件键用于控制可以通过 IAM 请求传递哪些标签。）aws:TagKeys 条件确保标签键区分大小写。对于未附加 AWSCodeArtifactAdminAccess 托管用户策略的 IAM 用户，此策略非常有用。托管式策略可以为用户赋予不受限的权限，允许其对所有资源执行任何 CodeArtifact 操作。

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

```
        "codeartifact:CreateRepository",
        "codeartifact:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Key1": "Value1"
        }
      }
    }
  ]
}
```

AWS CodeArtifact 权限参考

AWS CodeArtifact 资源和操作

在 AWS CodeArtifact 中，主要资源是域。在策略中，您可以使用 Amazon 资源名称 (ARN) 标识策略应用到的资源。存储库也是资源，且具有相关联的 ARN。有关更多信息，请参阅《Amazon Web Services 一般参考》中的 Amazon 资源名称 (ARN)。

资源类型	ARN 格式
Domain	arn:aws:codeartifact: *region-ID* :*account-ID* :domain/*my_domain*
存储库	arn:aws:codeartifact: *region-ID* :*account-ID* :repository/ *my_domain* /*my_repo*
带有命名空间的程序包	arn:aws:codeartifact: *region-ID* :*account-ID* :package/ *my_domain* /*my_repo*/*package-format* /*namespace* /*package-name*
不带命名空间的程序包	arn:aws:codeartifact: *region-ID* :*account-ID* :package/ *my_domain* /*my_repo*/*package-format* //*package-name*
所有 CodeArtifact 资源	arn:aws:codeartifact:*

资源类型	ARN 格式
指定账户在指定 AWS 区域中拥有的所有 CodeArtifact 资源	arn:aws:codeartifact: *region-ID* :*account-ID* :*

您指定哪个资源 ARN 取决于您要控制访问权限的一个或多个操作。

您可以使用其 ARN 在语句中指定特定域 (*myDomain*)，如下所示。

```
"Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain"
```

您可以使用特定存储库 (*myRepo*) 的 ARN 在语句中指定该存储库，如下所示。

```
"Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain/myRepo"
```

要在单个语句中指定多个资源，请使用逗号分隔其 ARN。以下语句适用于特定域中的所有程序包和存储库。

```
"Resource": [
  "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain",
  "arn:aws:codeartifact:us-east-2:123456789012:repository/myDomain/*",
  "arn:aws:codeartifact:us-east-2:123456789012:package/myDomain/*"
]
```

> **Note**
>
> 许多 AWS 服务将 ARN 中的冒号 (:) 或正斜杠 (/) 视为同一个字符。不过，CodeArtifact 在资源模式和规则中使用精确匹配。请务必在创建事件模式时使用正确的字符，以使其与资源中的 ARN 语法匹配。

AWS CodeArtifact API 操作和权限

在设置访问控制以及编写可附加到 IAM 身份的权限策略（基于身份的策略）时，您可以将下表作为参考。

您可以在 AWS CodeArtifact 策略中使用 AWS 范围的条件键来表示条件。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素参考。

请在策略的 Action 字段中指定这些操作。要指定操作，请在 API 操作名称之前使用 codeartifact: 前缀（例如，codeartifact:CreateDomain 和 codeartifact:AssociateExternalConnection）。要在单个语句中指定多项操作，请使用逗号将它们隔开（例如，"Action": ["codeartifact:CreateDomain", "codeartifact:AssociateExternalConnection"]）。

使用通配符

您可以在策略的 Resource 字段中指定带或不带通配符 (*) 的 ARN 作为资源值。您可以使用通配符指定多个操作或资源。例如，codeartifact:* 指定所有 CodeArtifact 操作，codeartifact:Describe* 指定以单词 Describe 开头的所有 CodeArtifact 操作。

对 AWS CodeArtifact 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用 CodeArtifact 和 IAM 时可能遇到的常见问题。

主题

- 我无权在以下位置执行操作 CodeArtifact
- 我想允许我以外的人 AWS 账户 访问我的 CodeArtifact 资源

我无权在以下位置执行操作 CodeArtifact

如果您收到错误提示，表明您无权执行某个操作，则您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 codeartifact:*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:
 codeartifact: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 codeartifact:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 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 AWS 账户 访问我的 CodeArtifact 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，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解是否 CodeArtifact 支持这些功能，请参阅如何 AWS CodeArtifact 与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 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 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅 IAM 用户指南中的为经过外部身份验证的用户（联合身份验证）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 角色与基于资源的策略有何不同。

使用 Amazon VPC 端点

您可以配置 CodeArtifact 为使用接口虚拟私有云 (VPC) 终端节点来提高 VPC 的安全性。

VPC 终端节点使用的 AWS PrivateLink服务使您可以通过私有 IP 地址访问 CodeArtifact API。 AWS PrivateLink 限制您的 VPC 之间与 AWS 网络之间 CodeArtifact 的所有网络流量。使用接口 VPC 端点时，无需互联网网关、NAT 设备或虚拟私有网关。有关更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的 VPC 端点。

Important

- VPC 终端节点不支持跨AWS 区域请求。请确保在计划向其发出 API 调用的同一 AWS 区域创建终端节点 CodeArtifact。
- VPC 端点仅通过 Amazon Route 53 支持 Amazon 提供的 DNS。如果您希望使用自己的 DNS，可以使用条件 DNS 转发。有关更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的 DHCP 选项集。
- 附加到 VPC 端点的安全组必须允许端口 443 上来自 VPC 的私有子网的传入连接。

主题

为创建 VPC 终端节点 CodeArtifact

要为创建虚拟私有云 (VPC) 终端节点 CodeArtifact，请使用 Amazon EC2 `create-vpc-endpoint` AWS CLI 命令。有关更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的接口 VPC 端点 (AWS PrivateLink)。

需要两个 VPC 终端节点，以便所有请求 CodeArtifact 都在 AWS 网络中。第一个端点用于调用 CodeArtifact API（例如`GetAuthorizationToken`和`CreateRepository`）。

```
com.amazonaws.region.codeartifact.api
```

第二个端点用于使用包管理器和构建工具（例如 npm 和 Gradle）访问 CodeArtifact 存储库。

```
com.amazonaws.region.codeartifact.repositories
```

以下命令创建访问 CodeArtifact 存储库的终端节点。

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region.codeartifact.api --subnet-ids subnetid \
  --security-group-ids groupid --no-private-dns-enabled
```

以下命令创建一个端点来访问程序包管理器和构建工具。

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region.codeartifact.repositories --subnet-ids subnetid \
  --security-group-ids groupid --private-dns-enabled
```

> **Note**
>
> 在创建 `codeartifact.repositories` 端点时，必须使用 `--private-dns-enabled` 选项创建私有 DNS 主机名。但是，由于 `codeartifact.api` 和 `codeartifact.repositories` 端点目前不支持多个私有 DNS 主机名，因此请为 `codeartifact.api` 使用 `--no-private-dns-enabled` 选项。如果您在创建`codeartifact.repositories`终端节点时无法或不想创建私有 DNS 主机名，则必须按照额外的配置步骤在 VPC CodeArtifact 中使用包管理器。请参阅使用不带私有 DNS 的 `codeartifact.repositories` 端点了解更多信息。

创建 VPC 终端节点后，您可能需要对安全组规则进行更多配置才能使用终端节点 CodeArtifact。有关 Amazon VPC 中的安全组的更多信息，请参阅安全组。

如果您在连接时遇到问题 CodeArtifact，可以使用 VPC Reachability Analyzer 工具来调试问题。有关更多信息，请参阅什么是 VPC Reachability Analyzer？

创建 Amazon S3 网关端点

CodeArtifact 使用亚马逊简单存储服务 (Amazon S3) Simple Service 来存储包裹资产。要从中提取包裹 CodeArtifact，您必须为 Amazon S3 创建网关终端节点。当您的构建或部署过程从中下载包时 CodeArtifact，它必须访问 CodeArtifact 才能获取包元数据，Amazon S3 才能下载包资产（例如 Maven `.jar` 文件）。

> **Note**
>
> 使用 Python 或 Swift 包格式时，不需要亚马逊 S3 终端节点。

要为创建 Amazon S3 网关终端节点 CodeArtifact，请使用 Amazon EC2 `create-vpc-endpoint` AWS CLI 命令。在创建端点时，必须为 VPC 选择路由表。有关更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的网关 VPC 端点。

以下示例创建 Amazon S3 端点。

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --service-name com.amazonaws.region.s3 \
  --route-table-ids routetableid
```

Amazon S3 存储桶的最低权限 AWS CodeArtifact

Amazon S3 网关端点使用 IAM 策略文档来限制对服务的访问。要仅允许最低的 Amazon S3 存储桶权限 CodeArtifact，请限制访问您在为终端节点创建 IAM 策略文档时 CodeArtifact 使用的 Amazon S3 存储桶。

下表描述了您应在策略中引用的 Amazon S3 存储桶，以允许 CodeArtifact 在每个区域进行访问。

区域	Amazon S3 存储桶 ARN
us-east-1	arn:aws:s3:::assets-193858265520-us-east-1
us-east-2	arn:aws:s3:::assets-250872398865-us-east-2
us-west-2	arn:aws:s3:::assets-787052242323-us-west-2
eu-west-1	arn:aws:s3:::assets-438097961670-eu-west-1
eu-west-2	arn:aws:s3:::assets-247805302724-eu-west-2
eu-west-3	arn:aws:s3:::assets-762466490029-eu-west-3
eu-north-1	arn:aws:s3:::assets-611884512288-eu-north-1
eu-south-1	arn:aws:s3:::assets-484130244270-eu-south-1

区域	Amazon S3 存储桶 ARN
eu-central-1	arn:aws:s3:::assets-769407342218-eu-central-1
ap-northeast-1	arn:aws:s3:::assets-660291247815-ap-northeast-1
ap-southeast-1	arn:aws:s3:::assets-421485864821-ap-southeast-1
ap-southeast-2	arn:aws:s3:::assets-860415559748-ap-southeast-2
ap-south-1	arn:aws:s3:::assets-681137435769-ap-south-1

您可以使用`aws codeartifact describe-domain`命令来获取 CodeArtifact 域使用的 Amazon S3 存储桶。

```
aws codeartifact describe-domain --domain mydomain
```

```
{
  "domain": {
    "name": "mydomain",
    "owner": "111122223333",
    "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/mydomain",
    "status": "Active",
    "createdTime": 1583075193.861,
    "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/a73que8sq-ba...",
    "repositoryCount": 13,
    "assetSizeBytes": 513830295,
    "s3BucketArn": "arn:aws:s3:::assets-787052242323-us-west-2"
  }
}
```

示例

以下示例说明如何提供us-east-1对该地区 CodeArtifact 操作所需的 Amazon S3 存储桶的访问权限。对于其他区域，请根据上表使用您所在区域的正确权限 ARN 来更新 Resource 条目。

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::assets-193858265520-us-east-1/*"]
    }
  ]
}
```

CodeArtifact 从 VPC 中使用

要使用 VPC 终端节点使用 AWS CLI 或 SDK 调用 CodeArtifact API，您必须覆盖使用的默认终端节点 CodeArtifact。按照配置 AWS CLI 为使用codeartifact.api终端节点中的说明获取 VPC 端点主机名并使用该主机名配置 CLI。

如果您无法或不想在中创建的 com.amazonaws.*region*.codeartifact.repositories VPC 终端节点上启用私有 DNS为创建 VPC 终端节点 CodeArtifact，则必须使用与 VPC 不同的存储库终端节点配置。 CodeArtifact 按照中的使用不带私有 DNS 的 codeartifact.repositories 端点说明配置 CodeArtifact com.amazonaws.*region*.codeartifact.repositories终端节点是否未启用私有 DNS。

配置 AWS CLI 为使用**codeartifact.api**终端节点

按照以下说明使用 com.amazonaws.*region*.codeartifact.api VPC 终端节点使用的主机名覆盖默认 CodeArtifact 主机名。

1. 运行以下命令来查找用于覆盖主机名的 VPC 端点。

```
$ aws ec2 describe-vpc-endpoints --filters Name=service-
name,Values=com.amazonaws.region.codeartifact.api \
```

```
  --query 'VpcEndpoints[*].DnsEntries[*].DnsName'
```

输出如下所示。

```
[
  [
    "vpce-0743fe535b883ffff-76ddffff.api.codeartifact.us-
west-2.vpce.amazonaws.com",
    "vpce-0743fe535b883ffff-76edffff-us-west-2a.api.codeartifact.us-
west-2.vpce.amazonaws.com"
  ]
]
```

在此示例中，您可以使用任一主机名来覆盖 com.amazonaws.*region*.codeartifact.api 端点。

2. 如果您使用 CodeArtifact AWS CLI，则使用`codeartifact login`命令通过将终端节点传递到`--endpoint-url`参数中来替换 Amazon VPC 终端节点的默认 CodeArtifact 主机名。请参阅以下示例。

> **Warning**
>
> `login` 命令不支持 Maven 或 Gradle。要配置这些程序包管理器，请参阅[将 CodeArtifact 与 Maven 结合使用]。

```
aws codeartifact login --tool npm --domain mydomain --domain-owner 111122223333 --
repository myrepo --endpoint-url VPC_endpoint
```

将 *VPC_endpoint* 替换为前缀为 `https://` 的 Amazon VPC 端点。请参阅以下示例端点。

```
https://vpce-0743fe535b883ffff-76ddffff.api.codeartifact.us-
west-2.vpce.amazonaws.com
```

如果您使用 SDK，请查阅 SDK 文档，了解如何覆盖主机名。如何执行此操作因您使用的语言而异。

使用不带私有 DNS 的 codeartifact.repositories 端点

如果您无法或不想在中创建的 com.amazonaws.*region*.codeartifact.repositories VPC 终端节点上启用私有 DNS为创建 VPC 终端节点 CodeArtifact，则必须按照以下说明使用正确的 CodeArtifact URL 配置包管理器。

1. 运行以下命令来查找用于覆盖主机名的 VPC 端点。

```
$ aws ec2 describe-vpc-endpoints --filters Name=service-
name,Values=com.amazonaws.region.codeartifact.repositories \
  --query 'VpcEndpoints[*].DnsEntries[*].DnsName'
```

输出如下所示。

```
[
  [
    "vpce-0743fe535b883ffff-76ddffff.d.codeartifact.us-west-2.vpce.amazonaws.com"
  ]
]
```

2. 更新 VPC 终端节点路径以包含包格式、您的 CodeArtifact 域名和 CodeArtifact 存储库名称。请参阅以下示例。

```
https://vpce-0743fe535b883ffff-76ddffff.d.codeartifact.us-
west-2.vpce.amazonaws.com/format/d/domain_name-domain_owner/repo_name
```

替换示例端点中的以下字段。

- *##*：替换为有效的 CodeArtifact 软件包格式，例如npm或pypi。
- *domain_name*：替换为包含托管软件包的 CodeArtifact 存储库的 CodeArtifact 域名。
- *domain_own* er：替换为 CodeArtifact 域名所有者的 ID，例如。111122223333
- *repo_name*：替换为托管 CodeArtifact 软件包的存储库。

以下 URL 是一个示例 npm 存储库端点。

```
https://vpce-0dc4daf7fca331ed6-et36qa1d.d.codeartifact.us-
west-2.vpce.amazonaws.com/npm/d/domainName-111122223333/repoName
```

3. 将您的程序包管理器配置为使用上一步中更新的 VPC 端点。您必须在不使用 CodeArtifact `login`命令的情况下配置软件包管理器。有关每个程序包格式的配置说明，请参阅以下文档。

   - npm：不使用 login 命令配置 npm
   - nuget：不使用 login 命令来配置 nuget 或 dotnet
   - pip：不使用 login 命令配置 pip
   - twine：在 CodeArtifact 中配置和使用 twine
   - Gradle：在 Gradle 中使用 CodeArtifact
   - mvn：在 mvn 中使用 CodeArtifact

为 CodeArtifact 创建 VPC 终端节点策略

要为创建 VPC 终端节点策略 CodeArtifact，请指定以下内容：

- 可执行操作的主体。
- 可执行的操作。
- 可用于执行操作的资源。

以下示例策略指定账户 123456789012 中的委托人可以调用 `GetAuthorizationToken` API 并从存储库中获取软件包。 CodeArtifact

```
{
  "Statement": [
    {
      "Action": [
        "codeartifact:GetAuthorizationToken",
        "codeartifact:GetRepositoryEndpoint",
        "codeartifact:ReadFromRepository",
        "sts:GetServiceBearerToken"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Principal": {
         "AWS": "arn:aws:iam::123456789012:root"
       }
    }
  ]
```

```
}
```

使用 AWS CloudFormation 创建 CodeArtifact 资源

CodeArtifact 与 AWS CloudFormation 集成，后者是一项服务，有助于对 AWS 资源进行建模和设置，这样您只需花较少的时间来创建和管理资源与基础设施。您创建一个描述您所需的所有 AWS 资源（例如，域或存储库）的模板，AWS CloudFormation 将负责为您预置和配置这些资源。

使用 AWS CloudFormation 时，您可以重复使用您的模板来不断地重复设置您的 CodeArtifact 资源。仅描述您的资源一次，然后在多个账户和 AWS 区域中反复配置相同的资源。

CodeArtifact 和 AWS CloudFormation 模板

要为 CodeArtifact 和相关服务预置和配置资源，您必须了解 AWS CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板可描述您要在 AWS CloudFormation 堆栈中调配的资源。如果您不熟悉 JSON 或 YAML，可以在 AWS CloudFormation Designer 的帮助下开始使用 AWS CloudFormation 模板。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的什么是 AWS CloudFormation Designer？

CodeArtifact 支持在 AWS CloudFormation 中创建域和存储库。有关更多信息（包括域和存储库的 JSON 和 YAML 模板示例），请参阅 AWS::CodeArtifact::Domain 和 AWS::CodeArtifact::Repository。

防止删除 CodeArtifact 资源

CodeArtifact 存储库包含关键的应用程序依赖关系，如果丢失，可能不容易重新创建。为了在使用 CloudFormation 管理 CodeArtifact 资源时保护 CodeArtifact 资源免遭意外删除，请在所有域和存储库中添加值为 `Retain` 的 `DeletionPolicy` 和 `UpdateRetainPolicy` 属性。如果从堆栈模板中移除资源或整个堆栈被意外删除，这种方法可防止删除。以下 YAML 片段显示了具有这些属性的基本域和存储库：

```
Resources:
    MyCodeArtifactDomain:
        Type: 'AWS::CodeArtifact::Domain'
        DeletionPolicy: Retain
        UpdateReplacePolicy: Retain
        Properties:
            DomainName: "my-domain"

    MyCodeArtifactRepository:
```

```
        Type: 'AWS::CodeArtifact::Repository'
        DeletionPolicy: Retain
        UpdateReplacePolicy: Retain
        Properties:
            RepositoryName: "my-repo"
            DomainName: !GetAtt MyCodeArtifactDomain.Name
```

有关这些属性的更多信息，请参阅《AWS CloudFormation 用户指南》中的 DeletionPolicy 和 UpdateReplacePolicy。

了解有关 AWS CloudFormation 的更多信息

要了解有关 AWS CloudFormation 的更多信息，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- AWS CloudFormation 命令行界面用户指南

AWS CodeArtifact 故障排除

以下信息可帮助您处理 CodeArtifact 中的常见问题。

有关特定于格式的故障排除信息，请参阅以下主题：

- Maven 故障排除
- Swift 故障排除

我无法查看通知

问题：当您在开发工具控制台中选择 Settings (设置) 下的 Notifications (通知)时，您会看到一个权限错误。

可能的解决方法：虽然通知是开发人员工具控制台的一项功能，但 CodeArtifact 目前不支持通知。CodeArtifact 的所有托管策略均不包括允许用户查看或管理通知的权限。如果您在开发人员工具控制台中使用其他服务，并且这些服务支持通知，则这些服务的托管策略包括查看和管理这些服务的通知所需的权限。

标记资源

标签是您或 AWS 分配给 AWS 资源的自定义属性标签。每个 AWS 标签具有两个部分：

- 标签键 （例如，CostCenter、Environment、Project 或 Secret）。标签键区分大小写。
- 一个称为标签值的可选字段（例如，111122223333、Production 或团队名称）。省略标签值与使用空字符串效果相同。与标签键一样，标签值区分大小写。

这些被统称为键-值对。

标签有助于您标识和组织 AWS 资源。许多 AWS 服务支持标记，因此，您可以将同一标签分配给来不同服务的资源，以指示这些资源是相关的。例如，您可以将相同的标签分配给为 AWS CodeBuild 项目分配的存储库。

有关使用标签的提示和最佳实践，请参阅标记 AWS 资源的最佳实践白皮书。

您可以在 CodeArtifact 中标记以下资源类型：

- 在中标记存储库 CodeArtifact
- 将域名标记为 CodeArtifact

您可以使用控制台、AWS CLI、CodeArtifact API 或 AWS SDK 来：

- 创建域或存储库时，向其添加标签*。
- 为域或存储库添加、管理和移除标签。

* 在控制台中创建域或存储库时，无法向其添加标签。

除了通过标签标识、组织和跟踪资源之外，您可以在 IAM 策略中使用标签，帮助控制哪些人可以查看您的资源并与之交互。有关基于标签的访问策略示例，请参阅使用标签控制对 CodeArtifact 资源的访问。

使用标签进行 CodeArtifact 成本分配

您可以在 CodeArtifact 中使用标签来分配存储成本和请求成本。

在 CodeArtifact 中分配数据存储成本

数据存储成本与域相关联，因此，要分配 CodeArtifact 存储成本，您可以使用任何应用于域的标签。有关向域添加标签的信息，请参阅将域名标记为 CodeArtifact。

在 CodeArtifact 中分配请求成本

大多数请求的使用与存储库相关联，因此，要分配 CodeArtifact 请求成本，您可以使用任何应用于存储库的标签。有关向存储库添加标签的信息，请参阅在中标记存储库 CodeArtifact。

有些请求类型与域而不是与存储库相关联，因此请求使用情况和与请求相关的成本将分配给域上的标签。确定请求类型是与域关联还是与存储库关联的最佳方法是使用《服务授权参考》中的 AWS CodeArtifact 定义的操作表。在操作列中找到请求类型，然后查看相应资源类型列中的值。如果资源类型为域，则该类型的请求将计入域的费用。如果资源类型为存储库或程序包，则该类型的请求将计入存储库的费用。有些操作会同时显示两种资源类型，对于这些操作，计费资源取决于请求中传递的值。

AWS CodeArtifact 中的配额

下表介绍 CodeArtifact 中的资源配额。要查看 CodeArtifact 的资源配额和服务端点列表，请参阅《Amazon Web Services 一般参考》中的 AWS 服务配额。

您可以为以下 CodeArtifact 资源配额申请增加服务配额。有关请求增加服务配额的更多信息，请参阅 AWS 服务配额。

名称	默认值	可调整	描述
资产文件大小	每个支持的区域：5 GB	是	每个资产的最大文件大小。
每个程序包版本的资产	每个支持的区域：150 个	否	每个程序包版本的资产的最大数量。
CopyPackageVersions 每秒请求数	每个支持的区域：5 个	是	每秒可对 CopyPackageVersions 进行的调用的最大数量。
每个存储库的直接上游	每个支持的区域：10 个	否	每个存储库的直接上游存储库的最大数量。
每个 AWS 账户的域	每个支持的区域：10 个	是	每个 AWS 账户可以创建的域的最大数量。
GetAuthorizationToken 每秒请求数	每个支持的区域：40 个	是	每秒检索的授权令牌的最大数量。
GetPackageVersionAsset 每秒请求数	每个支持的区域：50 个	是	每秒可对 GetPackageVersionAsset 进行的调用的最大数量。
ListPackageVersionAssets 每秒请求数	每个支持的区域：200 个	是	每秒可对 ListPackageVersionAssets 进行的调用的最大数量。

名称	默认值	可调整	描述
ListPackageVersions 每秒请求数	每个支持的区域：200 个	是	每秒可对 ListPackageVersions 进行的调用的最大数量。
ListPackages 每秒请求数	每个支持的区域：200 个	是	每秒可对 ListPackages 进行的调用的最大数量。
PublishPackageVersion 每秒请求数	每个支持的区域：10 个	是	每秒可对 PublishPackageVersion 进行的调用的最大数量。
每秒来自单个 AWS 账户的读取请求的数量。	每个支持的区域：800 个	是	每秒来自一个 AWS 账户的读取请求的最大数量。
每个域的存储库	每个支持的区域：1000 个	是	每个域可以创建的存储库的最大数量。
使用单个身份验证令牌的每秒请求数	每个支持的区域：1200 个	否	每秒使用单个身份验证令牌的请求的最大数量。
每个 IP 地址没有身份验证令牌的请求数	每个支持的区域：600 个	否	每秒不使用来自单个 IP 地址的身份验证令牌的请求的最大数量。
已搜索的上游存储库	每个支持的区域：25 个	否	解析程序包时搜索的上游存储库的最大数量。
每秒来自单个 AWS 账户的写入请求的数量。	每个支持的区域：100 个	是	每秒来自一个 AWS 账户的写入请求的最大数量。

AWS CodeArtifact 用户指南文档历史记录

下表描述了对的文档所做的重要更改 CodeArtifact。

变更	说明	日期
在有关该aws codeartifact login 命令的文档中添加了其他有效的软件包管理器。	在aws codeartifact login命令中使用的有效软件包管理器列表中添加dotnet了nuget、和swift。有关更多信息，请参阅 AWS CodeArtifact 身份验证和令牌。	2024年2月18日
在 Swift 疑难解答文档中添加了有关 Xcode 挂在 CI 机器上的条目	添加了有关可能因钥匙串提示输入密码而导致 Xcode 在 CI 计算机上挂起问题的信息，包括解决方案。有关更多信息，请参阅 由于钥匙串提示输入密码，Xcode 在 CI 计算机上挂起。	2024年2月6日
添加了有关解决 npm 8.x 或更高版本中 npm 程序包安装时间缓慢问题的信息	添加了有关如何解决 npm 软件包安装时间缓慢的信息 CodeArtifact，这可能会导致构建时间变慢。有关更多信息，请参阅 解决 npm 8.x 或更高版本中安装缓慢的问题。	2023 年 12 月 29 日
更新了有关 Python 包资源和元数据行为的信息 CodeArtifact	更新了有关 CodeArtifact 存储库如何保留和刷新 Python 包版本资产和元数据的信息。有关更多信息，请参阅 从上游和外部连接请求 Python 程序包。	2023 年 12 月 14 日
重新整理了有关监控的文档 CodeArtifact	重新整理了有关监控 CodeArtifact 事件的信息，并添加了有	2023 年 12 月 14 日

	关使用亚马逊 CloudWatch 指标查看 CodeArtifact 请求的信息。有关更多信息，请参阅 监控 CodeArtifact。	
添加了有关使用管理 CodeArtifact 资源的更多信息 AWS CloudFormation	添加了有关使用管理 CodeArtifact 资源的文档的参考文献和链接 CloudFormation，包括关于防止删除使用管理的 CodeArtifact 资源的部分 CloudFormation。有关更多信息，请参阅 防止删除 CodeArtifact 资源。	2023 年 12 月 7 日
添加了详细说明 CodeArtifact 对 AWS KMS 外部密钥存储 (XKS) 支持的文档	添加了一个部分，其中包含有关 CodeArtifact KMS 密钥支持的信息，包括将 XKS 密钥与一起 CodeArtifact使用。有关更多信息，请参阅 中支持的 AWS KMS 密钥类型 CodeArtifact。	2023 年 10 月 31 日
更新了现有的故障排除文档并添加了新的故障排除文档	添加了 Maven 故障排除主题，并在一般故障排除主题中包含指向 Swift 和 Maven 故障排除文档的链接。有关更多信息，请参阅 AWS CodeArtifact 故障排除。	2023 年 9 月 28 日
更新了文档，加入了 Swift 程序包管理器发布命令	Swift 5.9 引入了一个用于创建 Swift 程序包并将其发布到程序包存储库的 `swift package-registry publish` 命令。更新了 Swift 文档，加入了使用该命令的说明。有关更多信息，请参阅 CodeArtifact 与 Swift 配。	2023 年 9 月 25 日

添加了有关使用 Swift CodeArtifact 进行配置的文档	CodeArtifact 现在支持 Swift 软件包。添加了文档，其中包含有关配置 Swift 以使用 CodeArtifact 存储库的指南。有关更多信息，请参阅 CodeArtifact 与 Swift 配。	2023 年 9 月 20 日
增加了有关如何 CodeArtifact 处理已取消的 Python 包版本的指南	添加了文档，其中包含有关如何判断 Python 包版本是否被撤销、如何 CodeArtifact 处理已取消的包版本以及常见问题的答案的信息。有关更多信息，请参阅 已撤销的程序包版本。	2023 年 8 月 2 日
修复了 Yarn 文档中错误的命令行命令	修复了 Yarn 文档中获取 CodeArtifact 授权令牌并将其存储在环境变量中的错误命令行命令。	2023 年 7 月 20 日
对 Python 文档进行了一些小的补充和修复了一些小错误	在各自的文档中添加了 pip 和 twine 信息，并更正了在 twine 中使用 `codeartifact login` 命令时会发生的情况。有关更多信息，请参阅 在 CodeArtifact 中配置和使用 pip 和 在 CodeArtifact 中配置和使用 twine。	2023 年 7 月 14 日
修复了文档中不正确的 dotnet 命令 CodeBuild	更正了 在 CodeBuild 中使用 NuGet 程序包 文档中的 `dotnet add package` 命令。	2023 年 7 月 13 日

更新 AWS CodeArtifact 和 AWS Identity and Access Management 文档	彻底修改了 CodeArtifact 文档中的 IAM，以提高与其他 AWS 服务的文档的清晰度和一致性。请参阅 适用于 Identity and Access 管理 AWS CodeArtifact。	2023 年 5 月 24 日
添加了有关已撤销的 Python 程序包版本的信息	添加了有关如何 CodeArtifact 保留已取消的 Python 包版本元数据的信息。有关更多信息，请参阅已撤销的程序包版本。	2023 年 4 月 11 日
添加了有关 Clojure 支持的信息	添加了有关 Clojure 支持（包括管理 Clojure 项目的依赖项）的信息。有关更多信息，请参阅 通过 deps.edn 来使用 CodeArtifact。	2023 年 3 月 21 日
添加了有关通用程序包发布的信息	添加了有关通用程序包以及如何使用 AWS CLI发布和下载程序包内容的信息。有关更多信息，请参阅 在 CodeArtifact 中使用通用程序包、发布和使用通用程序包 和 通用程序包支持的命令。	2023 年 3 月 10 日
添加了有关用于发布的资产大小限制的信息	在程序包发布中添加了一个章节，说明了发布的资产大小限制。	2022 年 6 月 21 日

重构了外部连接文档	移动了外部连接文档并对其进行了重组，重点关注用户的最终目标，即将其 CodeArtifact 存储库连接到公共软件包存储库。还添加了有关实现该目标的不同方法的更多指导和信息。有关更多信息，请参阅 将 CodeArtifact 存储库连接到公有存储库。	2022 年 5 月 9 日
更新了 Amazon CloudWatch 活动的 CodeArtifact 活动信息	向 `account` 字段添加了更多信息并添加了 `repositoryAdministrator` 字段。有关更多信息，请参阅 CodeArtifact 事件格式和示例。	2022 年 3 月 7 日
添加了在没有私有 DNS CodeArtifact 的 VPC 中使用的配置说明	如果您无法或不想在您的 `codeartifact.repositories` VPC 终端节点上启用私有 DNS，则必须使用与 VPC 不同的存储库终端节点配置。 CodeArtifact 请参阅使用不带私有 DNS 的 `codeartifact.repositories` 端点了解更多信息。	2022 年 2 月 8 日
添加了用于更新程序包版本状态的深入文档	将更新程序包版本状态文档展开到自己的主题中。添加了更新软件包版本状态的文档，包括所需的 IAM 权限、各种场景的示例 AWS CLI 命令以及可能的错误。请参阅更新程序包版本状态了解更多信息。	2021 年 9 月 1 日

更新了复制程序包版本文档，提供了更深入的权限信息	在中添加了有关调用`aws codeartifact copy-package-versions` 命令将软件包版本从一个存储库复制到同一域中的 CodeArtifact 另一个存储库所必需的 IAM 和基于资源的策略权限的更多信息。除了提供更多信息外，现在还有源存储库和目标存储库所需的基于资源的策略示例。请参阅复制程序包所需的 IAM 权限了解更多信息。	2021 年 8 月 25 日
更新了在 IntelliJ IDEA 中运行 Gradle 构建的文档	更新了在 IntelliJ IDEA 中运行 Gradle 版本的文档，其中包含配置 Gradle 以从中获取插件的步骤。 CodeArtifact还添加了一个选项，用于通过内联调用为每次新运行创建新的 CodeArtifact 授权令牌`aws codeartifact get-authorization-token` 。请参阅在 IntelliJ IDEA 中运行 Gradle 构建了解更多信息。	2021 年 8 月 23 日
添加了有关配置和使用 Yarn 的文档 AWS CodeArtifact	添加了有关配置和使用 Yarn 1.X 和 Yarn 2.X 来管理 npm 包的文档。 CodeArtifact请参阅在 CodeArtifact 中配置和使用 Yarn了解更多信息。	2021 年 7 月 30 日

AWS CodeArtifact 现在支持 NuGet 软件包	CodeArtifact 用户现在可以发布和使用软件 NuGet 包。添加了有关配置和使用 Visual Studio 和 NuGet 命令行工具（如`nuget` CodeArtifact 存储库）`dotnet`的文档。请参阅将 CodeArtifact 与 NuGet 结合使用了解更多信息。	2020 年 11 月 19 日
在中标记资源 AWS CodeArtifact	在中添加了有关为存储库和域添加标签的 AWS CodeArtifact 文档。请参阅 标记资源。	2020 年 10 月 30 日
CodeArtifact 现在支持 AWS CloudFormation	CodeArtifact 用户现在可以使用 AWS CloudFormation 模板来创建 CodeArtifact 存储库和域。要获得更多信息并开始使用，请参阅使用 AWS CloudFormation 创建 CodeArtifact 资源。	2020 年 10 月 8 日
添加有关创建用于亚马逊 VPC 的 Amazon S3 网关终端节点的信息 CodeArtifact	添加了有关使用 Amazon EC2 AWS CLI 命令创建 Amazon S3 网关终端节点的信息。本文档还包含有关 CodeArtifact 需要在 Amazon VPC 环境中使用的特定权限的信息。请参阅 创建 Amazon S3 网关端点。	2020 年 8 月 12 日
使用 curl 命令发布 Maven 构件和发布第三方 Maven 构件	添加了有关使用 curl 进行发布和发布第三方构件的指导。	2020 年 8 月 10 日
公开发行（GA）版本	《CodeArtifact 用户指南》的初始版本。	2020 年 6 月 10 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。