Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.
## Table of Contents

### What is AWS CodeArtifact?
- CodeArtifact CodeArtifact User Guide

### How does CodeArtifact work?
- CodeArtifact CodeArtifact User Guide

### Concepts
- Domain
- Repository
- Package
- Package version
- Package version revision
- Upstream repository
- Asset
- Package namespace

### How do I get started with CodeArtifact?
- CodeArtifact CodeArtifact User Guide

### Setting up
- Sign up for AWS
- Install or upgrade and then configure the AWS CLI
- Provision an IAM User
- Install your package manager or build tool

### Getting started
- Prerequisites
- Getting started using the console
- Getting started using the AWS CLI

### Working with repositories
- Create a repository
- Create a repository (console)
- Create a repository (AWS CLI)
- Create a repository with an upstream repository
- Connect to a repository
- Use a package manager client
- Delete a repository
- Delete a repository (console)
- Delete a repository (AWS CLI)

### List repositories
- List repositories in an AWS account
- List repositories in the domain

### View or modify a repository configuration
- View or modify a repository configuration (console)
- View or modify a repository configuration (AWS CLI)

### Repository policies
- Create a resource policy to grant read access
- Set a policy
- Read a policy
- Delete a policy
- Grant read access to principals
- Grant write access to packages
- Grant write access to a repository

### Add an external connection
- Add an external connection to a repository
- Supported external connection repositories
- Remove an external connection
- Fetch npm packages from an external connection
- Fetch Maven packages from an external connection
- npm ingestion behavior
- Maven ingestion behavior
- CodeArtifact behavior when an external repository is not available
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publishing with curl</td>
<td>109</td>
</tr>
<tr>
<td>Using Maven checksums</td>
<td>111</td>
</tr>
<tr>
<td>Use Maven snapshots</td>
<td>111</td>
</tr>
<tr>
<td>Using NuGet</td>
<td>113</td>
</tr>
<tr>
<td>Use CodeArtifact with Visual Studio</td>
<td>113</td>
</tr>
<tr>
<td>Configure Visual Studio with the CodeArtifact Credential Provider</td>
<td>113</td>
</tr>
<tr>
<td>Use the Visual Studio Package Manager console</td>
<td>114</td>
</tr>
<tr>
<td>Use CodeArtifact with nuget or dotnet</td>
<td>114</td>
</tr>
<tr>
<td>Configure the nuget or dotnet CLI</td>
<td>115</td>
</tr>
<tr>
<td>Consume NuGet packages</td>
<td>118</td>
</tr>
<tr>
<td>Publish NuGet packages</td>
<td>118</td>
</tr>
<tr>
<td>CodeArtifact Credential Provider reference</td>
<td>119</td>
</tr>
<tr>
<td>NuGet compatibility</td>
<td>120</td>
</tr>
<tr>
<td>General NuGet compatibility</td>
<td>120</td>
</tr>
<tr>
<td>NuGet command line support</td>
<td>120</td>
</tr>
<tr>
<td>Using CodeArtifact with CodeBuild</td>
<td>121</td>
</tr>
<tr>
<td>Using npm packages in CodeBuild</td>
<td>121</td>
</tr>
<tr>
<td>Set up permissions with IAM roles</td>
<td>121</td>
</tr>
<tr>
<td>Log in and use npm</td>
<td>122</td>
</tr>
<tr>
<td>Using Python packages in CodeBuild</td>
<td>122</td>
</tr>
<tr>
<td>Set up permissions with IAM roles</td>
<td>122</td>
</tr>
<tr>
<td>Log in and use pip or twine</td>
<td>123</td>
</tr>
<tr>
<td>Using Maven packages in CodeBuild</td>
<td>124</td>
</tr>
<tr>
<td>Set up permissions with IAM roles</td>
<td>124</td>
</tr>
<tr>
<td>Use gradle or mvn</td>
<td>125</td>
</tr>
<tr>
<td>Using NuGet packages in CodeBuild</td>
<td>126</td>
</tr>
<tr>
<td>Set up permissions with IAM roles</td>
<td>126</td>
</tr>
<tr>
<td>Consume NuGet packages</td>
<td>126</td>
</tr>
<tr>
<td>Build with NuGet packages</td>
<td>128</td>
</tr>
<tr>
<td>Publish NuGet packages</td>
<td>129</td>
</tr>
<tr>
<td>Dependency caching</td>
<td>130</td>
</tr>
<tr>
<td>Working with CodeArtifact events</td>
<td>132</td>
</tr>
<tr>
<td>CodeArtifact event format and example</td>
<td>132</td>
</tr>
<tr>
<td>CodeArtifact event format</td>
<td>133</td>
</tr>
<tr>
<td>CodeArtifact event example</td>
<td>134</td>
</tr>
<tr>
<td>Use an event to start a CodePipeline execution</td>
<td>135</td>
</tr>
<tr>
<td>Configure EventBridge permissions</td>
<td>135</td>
</tr>
<tr>
<td>Create the EventBridge rule</td>
<td>135</td>
</tr>
<tr>
<td>Create the EventBridge rule target</td>
<td>136</td>
</tr>
<tr>
<td>Use an event to run a Lambda function</td>
<td>136</td>
</tr>
<tr>
<td>Create the EventBridge rule</td>
<td>136</td>
</tr>
<tr>
<td>Create the EventBridge rule target</td>
<td>136</td>
</tr>
<tr>
<td>Configure EventBridge permissions</td>
<td>136</td>
</tr>
<tr>
<td>Security</td>
<td>138</td>
</tr>
<tr>
<td>Data protection</td>
<td>138</td>
</tr>
<tr>
<td>Data encryption</td>
<td>139</td>
</tr>
<tr>
<td>Traffic privacy</td>
<td>139</td>
</tr>
<tr>
<td>Identity and access management</td>
<td>139</td>
</tr>
<tr>
<td>Authentication</td>
<td>139</td>
</tr>
<tr>
<td>Access control</td>
<td>140</td>
</tr>
<tr>
<td>Overview of managing access</td>
<td>141</td>
</tr>
<tr>
<td>Using identity-based policies</td>
<td>144</td>
</tr>
<tr>
<td>Using tags to control access to CodeArtifact resources</td>
<td>149</td>
</tr>
<tr>
<td>AWS CodeArtifact permissions reference</td>
<td>151</td>
</tr>
<tr>
<td>Monitoring</td>
<td>152</td>
</tr>
<tr>
<td>Logging CodeArtifact API calls with AWS CloudTrail</td>
<td>152</td>
</tr>
<tr>
<td>Compliance validation</td>
<td>155</td>
</tr>
</tbody>
</table>
What is AWS CodeArtifact?

CodeArtifact is a fully managed artifact repository service that makes it easy for organizations to securely store and share software packages used for application development. You can use CodeArtifact with popular build tools and package managers such as **NuGet**, **Maven**, **Gradle**, **npm**, **yarn**, **pip**, and **twine**.

CodeArtifact automatically scales when you ingest or publish new packages to your repositories. Because it’s a fully managed service, the setup and operation of its infrastructure is done for you. Integration with AWS Key Management Service (AWS KMS) secures all assets in a domain with one AWS KMS key (KMS key) that either you manage or AWS manages for you.

For more information, see [AWS CodeArtifact](#).

How does CodeArtifact work?

CodeArtifact stores software packages in repositories. Every CodeArtifact repository is a member of a single CodeArtifact domain. We recommend that you use one production domain for your organization with one or more repositories. For example, each repository might be used for a different development team. Packages in your repositories can then be discovered and shared across your development teams.

To add packages to a repository, configure a package manager such as **npm** or **maven** to use the repository endpoint (URL). You can then use the package manager to publish packages to repository. You can also import open-source packages into a repository by configuring it with an external connection to a public repository such as npmjs, NuGet Gallery, Maven Central, or PyPI. For more information, see [Add an external connection](#) (p. 26).

You can make packages in one repository available to another repository in the same domain. To do this, configure one repository as an upstream of the other. All package versions available to the upstream repository are also available to the downstream repository. In addition, all packages that are available to the upstream repository through an external connection to a public repository are available to the downstream repository. For more information, see [Working with upstream repositories in CodeArtifact](#) (p. 38).

AWS CodeArtifact Concepts

Here are some concepts and terms to know when you use CodeArtifact.

**Topics**
- Domain (p. 2)
- Repository (p. 2)
- Package (p. 2)
- Package version (p. 2)
- Package version revision (p. 2)
- Upstream repository (p. 2)
- Asset (p. 3)
- Package namespace (p. 3)
Domain

Repositories are aggregated into a higher-level entity known as a domain. All package assets and metadata are stored in the domain, but they are consumed through repositories. A given package asset, such as a Maven JAR file, is stored once per domain, no matter how many repositories it’s present in. All of the assets and metadata in a domain are encrypted with the same AWS KMS key (KMS key) stored in AWS Key Management Service (AWS KMS).

Each repository is a member of a single domain and can’t be moved to a different domain.

The domain allows organizational policy to be applied across multiple repositories, such as which accounts can access repositories in the domain, and which public repositories can be used as sources of packages.

Although an organization can have multiple domains, we recommend a single production domain that contains all published artifacts so that teams can find and share packages across their organization.

Repository

A CodeArtifact repository contains a set of package versions (p. 2), each of which maps to a set of assets (p. 3). Repositories are polyglot—a single repository can contain packages of any supported type. Each repository exposes endpoints for fetching and publishing packages using tools like the nuget CLI, the npm CLI, the Maven CLI (mvn), and pip. You can create up to 1000 repositories per domain.

Package

A package is a bundle of software and the metadata that is required to resolve dependencies and install the software. In CodeArtifact, a package consists of a package name, an optional namespace (p. 3) such as @types in @types/node, a set of package versions, and package-level metadata such as npm tags.

AWS CodeArtifact supports npm (p. 84), PyPI (p. 96), Maven (p. 100), and NuGet (p. 113) package formats.

Package version

A package version identifies the specific version of a package, such as @types/node 12.6.9. The version number format and semantics vary for different package formats. For example, npm package versions must conform to the Semantic Versioning specification. In CodeArtifact, a package version consists of the version identifier, package version level metadata, and a set of assets.

Package version revision

A package version revision is a string that identifies a specific set of assets and metadata for a package version. Each time a package version is updated, a new package version revision is created. For example, you might publish a source distribution archive (sdist) for a Python package version, and later add a Python wheel that contains compiled code to the same version. When you publish the wheel, a new package version revision is created.

Upstream repository

One repository is upstream of another when the package versions in it can be accessed from the repository endpoint of the downstream repository, effectively merging the contents of the two repositories from the point of view of a client. CodeArtifact allows creating an upstream relationship between two repositories.
Asset

An asset is an individual file stored in CodeArtifact that is associated with a package version, such as an npm .tgz file or Maven POM and JAR files.

Package namespace

Some package formats support hierarchical package names to organize packages into logical groups and help avoid name collisions. For example, npm supports scopes, see the npm scopes documentation for more information. The npm package @types/node has a scope of @types and a name of node. There are many other package names in the @types scope. In CodeArtifact, the scope (“types”) is referred to as the package namespace and the name (“node”) is referred to as the package name. For Maven packages, the package namespace corresponds to the Maven groupID. The Maven package org.apache.logging.log4j:log4j has a groupID (package namespace) of org.apache.logging.log4j and the artifactID (package name) log4j. Some package formats such as PyPI don’t support hierarchical names with a concept similar to npm scope or Maven groupID. Without a way to group package names, it can be more difficult to avoid name collisions.

How do I get started with CodeArtifact?

We recommend that you complete the following steps:

1. Learn more about CodeArtifact by reading the information in AWS CodeArtifact Concepts (p. 1).
2. Set up your AWS account, the AWS CLI, and an IAM user by following the steps in Setting up with AWS CodeArtifact (p. 4).
3. Use CodeArtifact by following the instructions in Getting started with CodeArtifact (p. 7).
Setting up with AWS CodeArtifact

If you've already signed up for Amazon Web Services (AWS), you can start using CodeArtifact immediately. You can open the CodeArtifact console, choose Create a domain and repository, and follow the steps in the launch wizard to create your first domain and repository.

If you haven’t signed up for AWS yet, or need assistance creating your first domain and repository, complete the following tasks to get set up to use CodeArtifact:

Topics
- Sign up for AWS (p. 4)
- Install or upgrade and then configure the AWS CLI (p. 4)
- Provision an IAM User (p. 5)
- Install your package manager or build tool (p. 6)

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including CodeArtifact. You are charged only for the services that you use. With CodeArtifact, you pay only for what you use.

If you already have an AWS account, skip to the next task, Install or upgrade and then configure the AWS CLI (p. 4). If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

2. Follow the online instructions.
   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Install or upgrade and then configure the AWS CLI

To call CodeArtifact commands from the AWS Command Line Interface (AWS CLI) on a local development machine, you must install the AWS CLI.

If you have an older version of the AWS CLI installed, you must upgrade it so the CodeArtifact commands are available. CodeArtifact commands are available in the following AWS CLI versions.

1. AWS CLI 1: 1.18.77 and newer
2. AWS CLI 2: 2.0.21 and newer

To check the version, use the aws --version command.

To install and configure the AWS CLI

1. Install or upgrade the AWS CLI with the instructions in Installing the AWS Command Line Interface.
2. Configure the AWS CLI, with the `configure` command, as follows.

```bash
aws configure
```

When prompted, specify the AWS access key and AWS secret access key of the IAM user that you will use with CodeArtifact. When prompted for the default region name, specify the region where you will create the pipeline, such as `us-east-2`. When prompted for the default output format, specify `json`.

**Important**
When you configure the AWS CLI, you are prompted to specify an AWS Region. Choose one of the supported regions listed in `Region and Endpoints` in the `AWS General Reference`.

For more information, see Configuring the AWS Command Line Interface and Managing Access Keys for IAM Users.

3. To verify the installation or upgrade, call the following command from the AWS CLI.

```bash
aws codeartifact help
```

If successful, this command displays a list of available CodeArtifact commands.

Next, you can create an IAM user and grant that user access to CodeArtifact. For more information, see Provision an IAM User (p. 5).

---

**Provision an IAM User**

Follow these instructions to prepare an IAM user to use CodeArtifact.

**To provision an IAM user**

1. Create an IAM user, or use one that is associated with your AWS account. For more information, see Creating an IAM User and Overview of AWS IAM Policies in the IAM User Guide.

2. Grant the IAM user access to CodeArtifact.

   - **Option 1:** Create a custom IAM policy. With a custom IAM policy, you can provide the minimum required permissions and change how long authentication tokens last. See Using identity-based policies for AWS CodeArtifact (p. 144) for more information and example policies.

   - **Option 2:** Use the `AWSCodeArtifactAdminAccess` AWS managed policy. The following snippet shows the contents of this policy.

     **Important**
     This policy grants access to all CodeArtifact APIs. We recommend that you always use the minimum permissions required to accomplish your task. For more information, see IAM Best Practices in the IAM User Guide.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "codeartifact:*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```
Install your package manager or build tool

If you have not already, install the package manager or build tool that you want to use with CodeArtifact.

- For npm, you can use the npm CLI or pnpm.
- For Maven, you can use either Maven (mvn) or Gradle.
- For Python, you can use pip to install packages and twine to upload packages to CodeArtifact.
- For NuGet, you can use the AWS Toolkit for Visual Studio Code in Visual Studio or the nuget or dotnet CLIs.
Getting started with CodeArtifact

In this getting started tutorial, you use CodeArtifact to create the following:

- A domain called my-domain.
- A repository called my-repo that is contained in my-domain.
- A repository called npm-store that is contained in my-domain. The npm-store has an external connection to the npm public repository. This connection is used to ingest an npm package into the my-repo repository.

Before starting this tutorial, we recommend that you review CodeArtifact AWS CodeArtifact Concepts (p. 1).

**Note**
This tutorial requires you to create resources that might result in charges to your AWS account. For more information, see CodeArtifact pricing.

Topics
- Prerequisites (p. 7)
- Getting started using the console (p. 7)
- Getting started using the AWS CLI (p. 9)

Prerequisites

You can complete this tutorial using the AWS Management Console or the AWS Command Line Interface (AWS CLI). To follow the tutorial, you must first complete the following prerequisites:

- Complete the steps in Setting up with AWS CodeArtifact (p. 4).
- Install the npm CLI. For more information, see Downloading and installing Node.js and npm in the npm documentation.

Getting started using the console

Run the following steps to get started with CodeArtifact using the AWS Management Console. This guide uses the npm package manager, if you are using a different package manager, you will need to modify some of the following steps.

1. Sign in to the AWS Management Console and open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/start. For more information, see Setting up with AWS CodeArtifact (p. 4).
2. Choose Create repository.
3. In Repository name, enter my-repo.
4. (Optional) In Repository Description, enter an optional description for your repository.
5. In Public upstream repositories, select npm-store to create a repository connected to npmjs that is upstream from your my-repo repository.

CodeArtifact assigns the name npm-store to this repository for you. All packages available in the upstream repository npm-store are also available to its downstream repository, my-repo.
6. Choose Next.
7. In AWS account, choose This AWS account.
8. In **Domain name**, enter `my-domain`.
9. Expand **Additional configuration**.
10. You must use an AWS KMS key (KMS key) to encrypt all assets in your domain. You can use an AWS managed key or a KMS key that you manage:
    - Choose **AWS managed key** if you want to use the default AWS managed key.
    - Choose **Customer managed key** if you want to use a KMS key that you manage. To use a KMS key that you manage, in **Customer managed key ARN**, search for and choose the KMS key.

    For more information, see [AWS managed key](#) and [Customer managed key](#) in the [AWS Key Management Service Developer Guide](#).

11. Choose **Next**.

    - **Package flow** shows how `my-domain`, `my-repo`, and `npm-store` are related.
    - **Step 1: Create repository** shows details about `my-repo` and `npm-store`.
    - **Step 2: Select domain** shows details about `my-domain`.

    When you're ready, choose **Create repository**.

13. On the `my-repo` page, choose **View connection instructions**, and then choose `npm`.

14. Use the AWS CLI to run the `login` command shown under **Configure your npm client using this AWS CLI CodeArtifact command**.

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

    You should receive output confirming your login succeeded.

    ```plaintext
    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
    ``

    If you receive the error `Could not connect to the endpoint URL`, make sure that your AWS CLI is configured and that your **Default region name** is set to the same region where you created your repository, see [Configuring the AWS Command Line Interface](#).

    For more information, see [Configure and use npm with CodeArtifact](#) (p. 84)

15. Use the npm CLI to install an npm package. For example, to install the popular npm package lodash, use the following command.

    ```sh
    npm install lodash
    ``

16. Return to the CodeArtifact console. If your `my-repo` repository is open, refresh the page. Otherwise, in the navigation pane, choose **Repositories**, and then choose `my-repo`.

    Under **Packages**, you should see the npm library, or package, that you installed. You can choose the name of the package to view its version and status. You can choose its latest version to view package details such as dependencies, assets, and more.

    **Note**
    There may be a delay between when you install the package and when it is ingested into your repository.

17. To avoid further AWS charges, delete the resources that you used during this tutorial:
Note
You cannot delete a domain that contains repositories, so you must delete my-repo and npm-store before you delete my-domain.

a. From the navigation pane, choose Repositories.
b. Choose npm-store, choose Delete, and then follow the steps to delete the repository.
c. Choose my-repo, choose Delete, and then follow the steps to delete the repository.
d. From the navigation pane, choose Domains.
e. Choose my-domain, choose Delete, and then follow the steps to delete the domain.

Getting started using the AWS CLI

Run the following steps to get started with CodeArtifact using the AWS Command Line Interface (AWS CLI). For more information, see Install or upgrade and then configure the AWS CLI (p. 4). This guide uses the npm package manager, if you are using a different package manager, you will need to modify some of the following steps.

1. Use the AWS CLI to run the create-domain command.

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

JSON-formatted data appears in the output with details about your new domain.

```json
{
    "domain": {
        "name": "my-domain",
        "owner": "111122223333",
        "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
    }
}
```

If you receive the error Could not connect to the endpoint URL, make sure that your AWS CLI is configured and that your Default region name is set to the same region where you created your repository, see Configuring the AWS Command Line Interface.

2. Use the create-repository command to create a repository in your domain.

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

JSON-formatted data appears in the output with details about your new repository.

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

9
3. Use the `create-repository` command to create an upstream repository for your `my-repo` repository.

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

JSON-formatted data appears in the output with details about your new repository.

```
{
  "repository": {
    "name": "npm-store",
    "administratorAccount": "111122223333",
    "domainName": "my-domain",
    "domainOwner": "111122223333",
    "upstreams": [],
    "externalConnections": []
  }
}
```

4. Use the `associate-external-connection` command to add an external connection to the npm public repository to your `npm-store` repository.

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

JSON-formatted data appears in the output with details about the repository and its new external connection.

```
{
  "repository": {
    "name": "npm-store",
    "administratorAccount": "111122223333",
    "domainName": "my-domain",
    "domainOwner": "111122223333",
    "upstreams": [],
    "externalConnections": [
      {
        "externalConnectionName": "public:npmjs",
        "packageFormat": "npm",
        "status": "AVAILABLE"
      }
    ]
  }
}
```

For more information, see Add an external connection (p. 26).

5. Use the `update-repository` command to associate the `npm-store` repository as an upstream repository to the `my-repo` repository.

```
aws codeartifact update-repository --repository my-repo --domain my-domain --domain-owner 111122223333 --upstreams repositoryName=npm-store
```
JSON-formatted data appears in the output with details about your updated repository, including its new upstream repository.

```json
{
  "repository": {
    "name": "my-repo",
    "administratorAccount": "111122223333",
    "domainName": "my-domain",
    "domainOwner": "111122223333",
    "upstreams": [
      {
        "repositoryName": "npm-store"
      }
    ],
    "externalConnections": []
  }
}
```

For more information, see Add, update, or remove upstream repositories (AWS CLI) (p. 39).

6. Use the `login` command to configure your npm package manager with your `my-repo` repository.

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

You should receive output confirming your login succeeded.

```
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
```

For more information, see Configure and use npm with CodeArtifact (p. 84).

7. Use the `npm CLI` to install an npm package. For example, to install the popular npm package lodash, use the following command.

```bash
npm install lodash
```

8. Use the `list-packages` command to view the package you just installed in your `my-repo` repository.

   **Note**
   There may be a delay between when you install the package and when it is ingested into your repository.

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

JSON-formatted data appears in the output with the format and name of the package that you installed.

```json
{
  "packages": [
    {
      "format": "npm",
      "package": "lodash"
    }
  ]
}
```
You now have three CodeArtifact resources:

- The domain `my-domain`.
- The repository `my-repo` that is contained in `my-domain`. This repository has an npm package available to it.
- The repository `npm-store` that is contained in `my-domain`. This repository has an external connection to the public npm repository and is associated as an upstream repository with the `my-repo` repository.

9. To avoid further AWS charges, delete the resources that you used during this tutorial:

   **Note**
   You cannot delete a domain that contains repositories, so you must delete `my-repo` and `npm-store` before you delete `my-domain`.

   a. Use the `delete-repository` command to delete the `npm-store` repository.

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

   JSON-formatted data appears in the output with details about the deleted repository.

   ```json
   
   
   
   ```

   b. Use the `delete-repository` command to delete the `npm-store` repository.

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

   JSON-formatted data appears in the output with details about the deleted repository.

   ```json
   
   
   ```
c. **Use the `delete-domain` command to delete the `my-domain` repository.**

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

JSON-formatted data appears in the output with details about the deleted domain.

```
{
  "domain": {
    "name": "my-domain",
    "owner": "111122223333",
    "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
  }
}
```
Working with repositories in CodeArtifact

These topics show you how to use the CodeArtifact CLI and API to create, list, update, and delete repositories.

Topics
- Create a repository (p. 14)
- Connect to a repository (p. 16)
- Delete a repository (p. 17)
- List repositories (p. 17)
- View or modify a repository configuration (p. 20)
- Repository policies (p. 22)
- Add an external connection (p. 26)
- Tag a repository in CodeArtifact (p. 32)

Create a repository

You can create a repository using the CodeArtifact console or the AWS Command Line Interface (AWS CLI). When you create a repository, it does not contain any packages. Each repository is associated with the AWS account that you use when you create it. An AWS account can have up to 10,000 repositories. For more information on CodeArtifact service limits, see Quotas in AWS CodeArtifact (p. 168). You can delete repositories to make room for more.

A repository can have one or more CodeArtifact repositories associated with it as upstream repositories. This allows a package manager client to access the packages contained in more than one repository using a single URL endpoint. For more information, see Working with upstream repositories in CodeArtifact (p. 38).

Note
After you create a repository, you cannot change its name, associated AWS account, or domain.

Topics
- Create a repository (console) (p. 14)
- Create a repository (AWS CLI) (p. 15)
- Create a repository with an upstream repository (p. 16)

Create a repository (console)

2. On the navigation pane, choose Repositories, and then choose Create repository.
3. For Repository name, enter a name for your repository.
4. (Optional) In Repository description, enter an optional description for your repository.
5. (Optional) In Publish upstream repositories, add intermediate repositories that connect your repositories to package authorities such as Maven Central or npmjs.com.
6. Choose Next.
7. In **AWS account**, choose **This AWS account** if you are signed in to the account that owns the domain. Choose **Different AWS account** if another AWS account owns the domain.

8. In **Domain**, choose the domain that the repository will be created in.
   
   If there are no domains in the account, you must create one. Enter the name for the new domain in **Domain name**.
   
   Expand **Additional configuration**.
   
   You must use an AWS KMS key (KMS key) to encrypt all assets in your domain. You can use an AWS managed key or a KMS key that you manage:
   
   - Choose **AWS managed key** if you want to use the default AWS managed key.
   - Choose **Customer managed key** if you want to use a KMS key that you manage. To use a KMS key that you manage, in **Customer managed key ARN**, search for and choose the KMS key.
   
   For more information, see **AWS managed keys** and **customer managed key** in the **AWS Key Management Service Developer Guide**.

9. Choose **Next**.

10. In **Review and create**, review what CodeArtifact is creating for you.
    
    - **Package flow** shows how your domain and repositories are connected.
    - **Step 1: Create repository** shows details about the repository and optional upstream repositories that will be created.
    - **Step 2: Select domain** shows details about **my_domain**.

    When you're ready, choose **Create repository**.

---

**Create a repository (AWS CLI)**

Use the `create-repository` command to create a repository in your domain.

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

Example output:

```json
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "description": "My new repository",
        "upstreams": "[]",
        "externalConnections": "[]"
    }
}
```

A new repository doesn't contain any packages. Each repository is associated with the AWS account that you're authenticated to when the repository is created. An AWS account can have a maximum of 100 repositories. Repositories that have been deleted with the `delete-repository` command don't count towards this limit.
Create a repository with tags

To create a repository with tags, add the --tags parameter to your create-domain command.

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

Create a repository with an upstream repository

You can specify one or more upstream repositories when you create a repository.

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

Example output:

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

Note

To create a repository with an upstream, you must have permission for the AssociateWithDownstreamRepository action on the upstream repository.

To add an upstream to a repository after it's been created, see Add, update, or remove upstream repositories (console) (p. 38) and Add, update, or remove upstream repositories (AWS CLI) (p. 39).

Connect to a repository

After you have configured your profile and credentials to authenticate to your AWS account, decide which repository to use in CodeArtifact. You have the following options:

- Create a repository. For more information, see Creating a Repository (p. 14).
- Use a repository that already exists in your account. You can use the list-repositories command to find the repositories created in your AWS account. For more information, see ??? (p. 17).
- Use a repository in a different AWS account. For more information, see Repository policies (p. 22).

Use a package manager client

After you know which repository you want to use, see one of the following:
Delete a repository

You can delete a repository using the CodeArtifact console or the AWS CLI. After a repository has been deleted, you can no longer push packages to it or pull packages from it. All packages in the repository become permanently unavailable and cannot be restored. You can create a repository with the same name, but its contents will be empty.

**Topics**
- Delete a repository (console) (p. 17)
- Delete a repository (AWS CLI) (p. 17)

**Delete a repository (console)**

2. On the navigation pane, choose **Repositories**, then choose the repository that you want to delete.
3. Choose **Delete** and then follow the steps to delete the domain.

**Delete a repository (AWS CLI)**

Use the `delete-repository` command to delete a repository.

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

Example output:

```json
{
   "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": []
   }
}
```

**List repositories**

Use the commands in this topic to list repositories in an AWS account or domain.
List repositories in an AWS account

Use this command to list all of the repositories in your AWS account.

```
aws codeartifact list-repositories
```

Sample output:

```
{
  "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",
      "description": "Description of repo3"
    }
  ]
}
```

You can paginate the response from `list-repositories` using the `--max-results` and `--next-token` parameters. For `--max-results`, specify an integer from 1 to 1000 to specify the number of results returned in a single page. Its default is 50. To return subsequent pages, run `list-repositories` again and pass the `nextToken` value received in the previous command output to `--next-token`. When the `--next-token` option is not used, the first page of results is always returned.

List repositories in the domain

Use `list-repositories-in-domain` to get a list of all the repositories in a domain.

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

The output shows that some of the repositories are administered by different AWS accounts.

```
{
  "repositories": [
  
  ]
}
You can paginate the response from `list-repositories-in-domain` using the `--max-results` and `--next-token` parameters. For `--max-results`, specify an integer from 1 to 1000 to specify the number of results returned in a single page. Its default is 50. To return subsequent pages, run `list-repositories-in-domain` again and pass the `nextToken` value received in the previous command output to `--next-token`. When the `--next-token` option is not used, the first page of results is always returned.

To output the repository names in a more compact list, try the following command.

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

Sample output:

```
repo1
repo2
repo3
```

The following example outputs the account ID in addition to the repository name.

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

Sample output:

```
repo1 710221105108
repo2 710221105108
repo3 532996949307
```
You can view and update details about your repository using the CodeArtifact console or the AWS Command Line Interface (AWS CLI).

**Note**
After you create a repository, you cannot change its name, associated AWS account, or domain.

**Topics**
- View or modify a repository configuration (console) (p. 20)
- View or modify a repository configuration (AWS CLI) (p. 21)

**View or modify a repository configuration (console)**
You can view details about and update your repository using the CodeArtifact console.

2. In the navigation pane, choose **Repositories**, and then choose the repository name that you want to view or modify.
3. Expand **Details** to see the following:
   - The repository's domain. Choose the domain name to learn more about it.
   - The repository's resource policy. Choose **Apply a repository policy** to add one.
   - The repository's Amazon Resource Name (ARN).
   - If your repository has an external connection, you can choose the connection to learn more about it. A repository can have only one external connection. For more information, see **Add an external connection** (p. 26).
   - If your repository has upstream repositories, you can choose one to see its details. A repository can have up to 10 direct upstream repositories. For more information, see **Working with upstream repositories in CodeArtifact** (p. 38).

   **Note**
   A repository can have an external connection or upstream repositories, but not both.

4. In **Packages**, you can see any packages that are available to this repository. Choose a package to learn more about it.
5. Choose **View connection instructions**, and then choose a package manager to learn how to configure it with CodeArtifact.
6. Choose **Apply repository policy** to update or add a resource policy to your repository. For more information, see **Repository policies** (p. 22).
7. Choose **Edit** to add or update the following:
   - The repository description.
   - Tags associated with the repository.
   - If your repository has an external connection, you can change which public repository it connects to. Otherwise, you can add one or more existing repositories as upstream repositories. Arrange them in the order you want them prioritized by CodeArtifact when a package is requested. For more information, see **Upstream repository priority order** (p. 42).
View or modify a repository configuration (AWS CLI)

To view a repository's current configuration in CodeArtifact, use the `describe-repository` command.

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

Example output:

```

```

```

```

Modify a repository upstream configuration

An upstream repository allows a package manager client to access the packages contained in more than one repository using a single URL endpoint. To add or change a repository's upstream relationship, use the `update-repository` command.

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

Example output:

```

```

```

```

Note

To add an upstream repository, you must have permission for the AssociateWithDownstreamRepository action on the upstream repository.

To remove a repository's upstream relationship, use an empty list as the argument to the --upstreams option.

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

CodeArtifact uses resource-based permissions to control access. Resource-based permissions let you specify who has access to a repository and what actions they can perform on it. By default, only the repository owner has access to a repository. You can apply a policy document that allows other IAM principals to access your repository.

For more information, see Resource-Based Policies and Identity-Based Policies and Resource-Based Policies.

Create a resource policy to grant read access

A resource policy is a text file in JSON format. The file must specify a principal (actor), one or more actions, and an effect (Allow or Deny). For example, the following resource policy grants the account 123456789012 permission to download packages from the repository.

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

Because the policy is evaluated only for operations against the repository that it's attached to, you don't need to specify a resource. Because the resource is implied, you can set the Resource to *.

**Note**

The `codeartifact:ReadFromRepository` action can only be used on a repository resource. You cannot put a package's Amazon Resource Name (ARN) as a resource with `codeartifact:ReadFromRepository` as the action to allow read access to a subset of packages in a repository. A given principal can either read all the packages in a repository or none of them.

Because the only action specified in the repository is `ReadFromRepository`, users and roles from account 1234567890 can download packages from the repository. However, they can't perform other
Set a policy

After you create a policy document, use the `put-repository-permissions-policy` command to attach it to a repository:

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

When you call `put-repository-permissions-policy`, the resource policy on the repository is ignored when evaluating permissions. This ensures that the owner of a domain cannot lock themselves out of the repository, which would prevent them from being able to update the resource policy.

**Note**

You cannot grant permissions to another AWS account to update the resource policy on a repository using a resource policy, since the resource policy is ignored when calling `put-repository-permissions-policy`. 

Sample output:

```json
{
  "policy": {
    "resourceArn": "arn:aws:codeartifact:region-id:111122223333:repository/my_domain/my_repo",
    "document": "{ ...policy document content... }",
    "revision": "MQlyyTqRASRU3HB58gBtSDNX07Q3hvxxxxxx="
  }
}
```

The output of the command contains the Amazon Resource Name (ARN) of the repository resource, the full contents of the policy document, and a revision identifier. You can pass the revision identifier to
Read a policy

Use the `get-repository-permissions-policy` command to read an existing version of a policy document. To format the output for readability, use the `--output` and `--query` options together with the Python `json.tool` module.

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

Sample output:

```json
{
  "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": "*"
    }
  ]
}
```

Delete a policy

Use the `delete-repository-permissions-policy` command to delete a policy from a repository.

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

The format of the output is the same as that of the `get-repository-permissions-policy` command.

Grant read access to principals

When you specify the root user of an account as the principal in a policy document, you grant access to all of the users and roles in that account. To limit access to selected users or roles, use their ARN in the Principal section of the policy. For example, use the following to grant read access to the IAM user bob in account 123456789012.

```json
{
  "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": "*"
    }
  ]
}
```
Grant write access to packages

The `codeartifact:PublishPackageVersion` action is used to control permission to publish new versions of a package. The resource used with this action must be a package. The format of CodeArtifact package ARNs is as follows.

```
```

The following example shows the ARN for an npm package with scope `@parity` and name `ui` in the example-repo repository in domain `my_domain`.

```
```

The ARN for an npm package without a scope has the empty string for the namespace field. For example, the following is the ARN for a package without a scope and with name `react` in the example-repo repository in domain `my_domain`.

```
```

The following policy grants account `123456789012` permission to publish versions of `@parity/ui` in the example-repo repository.

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

**Important**

To grant permission to publish Maven and NuGet package versions, add the following permissions in addition to `codeartifact:PublishPackageVersion`. 
1. NuGet: `codeartifact:ReadFromRepository` and specify the repository resource

Because this policy specifies a domain and repository as part of the resource, it allows publishing only when attached to that repository.

**Grant write access to a repository**

You can use wildcards to grant write permission for all packages in a repository. For example, use the following policy to grant an account permission to write to all packages in the `example-repo` repository.

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

**Add an external connection**

You can add a connection between a CodeArtifact repository and an external, public repository such as https://npmjs.com or the Maven Central repository. Then, when you request a package from the CodeArtifact repository that's not already present in the repository, the package can be fetched from the external connection. This makes it possible to consume open-source dependencies used by your application.

**Topics**

- Add an external connection to a repository (p. 26)
- Supported external connection repositories (p. 27)
- Remove an external connection (p. 27)
- Fetch npm packages from an external connection (p. 28)
- Fetch Maven packages from an external connection (p. 29)
- npm ingestion behavior (p. 30)
- Maven ingestion behavior (p. 31)
- CodeArtifact behavior when an external repository is not available (p. 31)
- Availability of new package versions (p. 32)

**Add an external connection to a repository**

To add an external connection to a CodeArtifact repository, use `associate-external-connection`.

```
aws codeartifact associate-external-connection --external-connection public:npmjs
```
Example output:

```json
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "description": "A description of my_repo",
        "upstreams": [],
        "externalConnections": [
            {
                "externalConnectionName": "public:npmjs",
                "packageFormat": "npm",
                "status": "AVAILABLE"
            }
        ]
    }
}
```

**Note**
A repository is limited to a single external connection only.

## Supported external connection repositories

CodeArtifact supports an external connection to the following public repositories. To use the CodeArtifact CLI to specify an external connection, use the value in the **Name** column for the `--external-connection` parameter when you run the `associate-external-connection` command.

<table>
<thead>
<tr>
<th>Repository type</th>
<th>Description</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>npm</td>
<td>npm public registry</td>
<td>public:npmjs</td>
</tr>
<tr>
<td>Python</td>
<td>Python Package Index</td>
<td>public:pypi</td>
</tr>
<tr>
<td>Maven</td>
<td>Maven Central</td>
<td>public:maven-central</td>
</tr>
<tr>
<td>Maven</td>
<td>Google Android repository</td>
<td>public:maven-googleandroid</td>
</tr>
<tr>
<td>Maven</td>
<td>Gradle plugins repository</td>
<td>public:maven-gradleplugins</td>
</tr>
<tr>
<td>Maven</td>
<td>CommonsWare Android repository</td>
<td>public:maven-commonware</td>
</tr>
<tr>
<td>NuGet</td>
<td>NuGet Gallery</td>
<td>public:nuget-org</td>
</tr>
</tbody>
</table>

## Remove an external connection

To remove an external connection, use `disassociate-external-connection`.

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

```json
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "description": "A description of my_repo",
        "upstreams": [],
        "externalConnections": []
    }
}
```

**Fetch npm packages from an external connection**

After you add an external connection, configure your package manager to use your CodeArtifact repository. Use the following for **npm**.

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

Then, request the package from the public repository.

```bash
npm install lodash
```

After the package has been copied into your CodeArtifact repository, you can use the list-packages and list-package-versions commands to view it.

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

Example output:

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

The list-package-versions command lists all versions of the package copied into your CodeArtifact repository. In some cases, this is all of the versions of the package in the external repository. In other cases, this is a subset of those versions. For more information, see npm ingestion behavior (p. 30).

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

Example output:

```json
{
    "defaultDisplayVersion": "1.2.5",
    "format": "npm",
    "package": "lodash",
}
```
Fetch Maven packages from an external connection

After you add an external connection, configure your build tool to use your CodeArtifact repository. For more information, see Use CodeArtifact with mvn (p. 105) and Use CodeArtifact with Gradle (p. 100). If you run either tool (for example, `gradle build`), packages are requested from Maven Central and stored in your CodeArtifact repository.

Restrict Maven dependency downloads to a CodeArtifact repository

If a package cannot be fetched from a configured repository, by default, the `mvn` command fetches it from Maven Central. Add the `maven` element to `settings.xml` to make `mvn` always use your CodeArtifact repository.

```xml
<settings>
  ...
  <mirrors>
    <mirror>
      <id>central-mirror</id>
      <name>CodeArtifact Maven Central mirror</name>
      <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>
  ...
</settings>
```

If you add a `mirrors` element, you must also have a `pluginRepository` element in your `settings.xml` or `pom.xml`. The following example fetches application dependencies and Maven plugins from a CodeArtifact repository.

```xml
<settings>
  ...
  <pluginRepository/>
</settings>
```
npm ingestion behavior

The following example fetches application dependencies from a CodeArtifact repository and fetches Maven plugins from Maven Central.

```xml
<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>
```

npm ingestion behavior

When a package is requested from a repository with an external connection to https://npmjs.com, CodeArtifact ingests that package version and up to two versions of its direct and transitive dependencies. This reduces the time to ingest the dependency tree.

Each dependency has a specified version constraint. For example, the npm package version webpack 4.41.2 specifies a dependency on @babel/core with a version constraint of ^7.7.2. The caret (^) specifies that the latest minor or patch version is used (for example, 7.7.4 or 7.8.0). When webpack 4.41.2 is ingested, the most recent published version of @babel/core that satisfies the version constraint is ingested. If the version of @babel/core specified by the latest tag is different, it is also ingested. This logic applies to the direct and transitive dependencies of @babel/core and the direct and transitive dependencies of webpack 4.41.2.

If ingestion of a package is not complete in 40 seconds, a 404 error is returned to the client.

```bash
npm ERR! code E404
```
When this occurs, CodeArtifact is still copying packages from the external repository asynchronously. Retry the same command to complete the ingestion of the entire dependency tree.

**Maven ingestion behavior**

When a Maven package is requested from a repository with an external connection to Maven Central repository, CodeArtifact ingests all assets of the package version that follow the standard Maven asset naming conventions. The dependencies of a package version are not ingested until they are requested by the client (for example, `mvn`).

If ingestion of a required asset is not complete within 40 seconds, a 404 error is returned to the client. A timeout error during a Gradle build might look like the following.

```
> Could not resolve all files for configuration ':compileClasspath'.
  > Could not resolve org.mockito:mockito-core:3.1.0.
  Required by:
    project :
      > Could not resolve org.mockito:mockito-core:3.1.0.
      > Could not get resource 'https://my_domain/codeartifact.aws.a2z.com/maven/my_domain/org/mockito/mockito-core/3.1.0/mockito-core-3.1.0.pom'.
    > Could not GET 'https://my_domain/codeartifact.aws.a2z.com/maven/my_domain/org/mockito/mockito-core/3.1.0/mockito-core-3.1.0.pom'.
```

When this occurs, CodeArtifact is still copying packages from the external repository asynchronously. Retry the same command to complete the ingestion of the entire dependency tree.

**CodeArtifact behavior when an external repository is not available**

Occasionally, an external repository will experience an outage that means CodeArtifact cannot fetch packages from it, or fetching packages is much slower than normal. When this occurs, package versions already pulled from an external repository (e.g. `npmjs.com`) and stored in a CodeArtifact repository will continue to be available for download from CodeArtifact. However, packages that are not already stored in CodeArtifact may not be available, even when an external connection to that repository has been configured. For example, your CodeArtifact repository might contain the npm package version `lodash 4.17.19` because that's what you have been using in your application so far. When you want to upgrade to `4.17.20`, normally CodeArtifact will fetch that new version from `npmjs.com` and store it in your CodeArtifact repository. However, if `npmjs.com` is experiencing an outage this new version will not be available. The only workaround is to try again later once `npmjs.com` has recovered.

External repository outages can also affect publishing new package versions to CodeArtifact. In a repository with an external connection configured, CodeArtifact will not permit publishing a package version that is already present in the external repository, see Packages overview (p. 46) for more information. However, an external repository outage may in rare cases mean that CodeArtifact does not have up-to-date information on which packages and package versions are present in an external repository.
Availability of new package versions

For a package version in a public repository such as npmjs.com or the Python Package Index (PyPI) to be available through a CodeArtifact repository, it must first be added to a regional package metadata cache. This cache is maintained by CodeArtifact in each AWS region and contains metadata that describes the contents of supported public repositories. Because of this cache, there is a delay between when a new package version is published to a public repository and when it is available from CodeArtifact. This delay varies by package type.

For npm and NuGet packages, there may be a delay of up to 10 minutes from when a new package version is published to npmjs.com or nuget.org and when it is available for installation from a CodeArtifact repository. CodeArtifact automatically synchronizes metadata from these two repositories to ensure the cache is up-to-date.

For Python and Maven packages, there may be a delay of up to 3 hours from when a new package version is published to a public repository and when it is available for installation from a CodeArtifact repository. CodeArtifact will check for new versions of a package at most once every 3 hours. The first request for a given package name after the 3-hour cache lifetime has expired will cause all new versions of that package to be imported into the regional cache.

This means that for Python and Maven packages that are in common use, new versions will typically be imported every 3 hours because the high rate of requests means that the cache will often be updated as soon as the cache lifetime has expired. For infrequently used packages, the cache will not have the latest version until a version of the package is requested from a CodeArtifact repository. On the first request, only previously-imported versions will be available from CodeArtifact, but this request will cause the cache to be updated. On subsequent requests, the new versions of the package will have been added to the cache and will be available for download.

Tag a repository in CodeArtifact

Tags are key-value pairs associated with AWS resources. You can apply tags to your repositories in CodeArtifact. For information about CodeArtifact resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging resources (p. 167).

You can use the CLI to specify tags when you create a repository. You can use the console or CLI to add or remove tags, and update the values of tags in a repository. You can add up to 50 tags to each repository.

Topics
- Tag repositories (CLI) (p. 32)
- Tag repositories (console) (p. 34)

Tag repositories (CLI)

You can use the CLI to manage repository tags.

Topics
- Add tags to a repository (CLI) (p. 33)
- View tags for a repository (CLI) (p. 33)
- Edit tags for a repository (CLI) (p. 34)
- Remove tags from a repository (CLI) (p. 34)
Add tags to a repository (CLI)

You can use the console or the AWS CLI to tag repositories.

To add a tag to a repository when you create it, see Create a repository (p. 14).

In these steps, we assume that you have already installed a recent version of the AWS CLI or updated to the current version. For more information, see Installing the AWS Command Line Interface.

At the terminal or command line, run the `tag-resource` command, specifying the Amazon Resource Name (ARN) of the repository where you want to add tags and the key and value of the tag you want to add.

```
Note
To get the ARN of the repository, run the `describe-repository` command:
```
aws codeartifact describe-repository --domain my_domain --repository my_repo --query repository.arn
```

You can add more than one tag to a repository. For example, to tag a repository named `my_repo` in a domain named `my_domain` with two tags, a tag key named `key1` with the tag value of `value1`, and a tag key named `key2` with the tag value of `value2`:

```
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
```

If successful, this command has no output.

View tags for a repository (CLI)

Follow these steps to use the AWS CLI to view the AWS tags for a repository. If no tags have been added, the returned list is empty.

At the terminal or command line, run the `list-tags-for-resource` command.

```
Note
To get the ARN of the repository, run the `describe-repository` command:
```
aws codeartifact describe-repository --domain my_domain --repository my_repo --query repository.arn
```

For example, to view a list of tag keys and tag values for a repository named `my_repo` in a domain named `my_domain` with the ARN value:

```
```

If successful, this command returns information similar to the following:

```
{
  "tags": {
    "key1": "value1",
    "key2": "value2"
  }
}
```
Edit tags for a repository (CLI)

Follow these steps to use the AWS CLI to edit a tag for a repository. You can change the value for an existing key or add another key.

At the terminal or command line, run the `tag-resource` command, specifying the ARN of the repository where you want to update a tag and specify the tag key and tag value.

**Note**
To get the ARN of the repository, run the `describe-repository` command:

```
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
```

If successful, this command has no output.

Remove tags from a repository (CLI)

Follow these steps to use the AWS CLI to remove a tag from a repository.

**Note**
If you delete a repository, all tag associations are removed from the deleted repository. You do not have to remove tags before you delete a repository.

At the terminal or command line, run the `untag-resource` command, specifying the ARN of the repository where you want to remove tags and the tag key of the tag you want to remove.

**Note**
To get the ARN of the repository, run the `describe-repository` command:

```
aws codeartifact describe-repository --domain my_domain --repository my_repo --query repository.arn
```

For example, to remove multiple tags on a repository named `my_repo` in a domain named `my_domain` with the tag keys `key1` and `key2`:

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

If successful, this command has no output. After removing tags, you can view the remaining tags on the repository using the `list-tags-for-resource` command.

Tag repositories (console)

You can use the console or the CLI to tag resources.

**Topics**
- Add tags to a repository (console) (p. 35)
- View tags for a repository (console) (p. 37)
- Edit tags for a repository (console) (p. 37)
- Remove tags from a repository (console) (p. 37)
Add tags to a repository (console)

You can use the console to add tags to an existing repository.

2. On the Repositories page, choose the repository that you want to add tags to.
3. Expand the Details section.
4. Under Repository tags, if there are no tags on the repository, choose Add repository tags. If there are tags on the repository, choose View and edit repository tags.
5. Choose Add new tag.
6. In the Key and Value fields, enter the text for each tag you want to add. (The Value field is optional.) For example, in Key, enter Name. In Value, enter Test.
Tag repositories (console)

**Repository**

Repository description - *optional*

1000 character limit

**Tags**

Tags - *optional*

<table>
<thead>
<tr>
<th>Key</th>
<th>Value - <em>optional</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Test</td>
</tr>
</tbody>
</table>

You can add 49 more tags.

- **AWS reserved tags**
  Resource tags added by other AWS services. These tags cannot be modified.

**Upstream repositories - *optional***

Repository name
7. (Optional) Choose **Add tag** to add more rows and enter more tags.
8. Choose **Update repository**.

**View tags for a repository (console)**

You can use the console to list tags for existing repositories.

2. On the **Repositories** page, choose the repository where you want to view tags.
3. Expand the **Details** section.
4. Under **Repository tags**, choose **View and edit repository tags**.
   
   *Note*
   
   If there are no tags added to this repository, the console will read **Add repository tags**.

**Edit tags for a repository (console)**

You can use the console to edit tags that have been added to repository.

2. On the **Repositories** page, choose the repository where you want to update tags.
3. Expand the **Details** section.
4. Under **Repository tags**, choose **View and edit repository tags**.
   
   *Note*
   
   If there are no tags added to this repository, the console will read **Add repository tags**.

5. In the **Key** and **Value** fields, update the values in each field as needed. For example, for the **Name** key, in **Value**, change **Test** to **Prod**.
6. Choose **Update repository**.

**Remove tags from a repository (console)**

You can use the console to delete tags from repositories.

2. On the **Repositories** page, choose the repository where you want to remove tags.
3. Expand the **Details** section.
4. Under **Repository tags**, choose **View and edit repository tags**.
   
   *Note*
   
   If there are no tags added to this repository, the console will read **Add repository tags**.

5. Next to the key and value for each tag you want to delete, choose **Remove**.
6. Choose **Update repository**.
Working with upstream repositories in CodeArtifact

A repository can have other AWS CodeArtifact repositories as upstream repositories. This enables a package manager client to access the packages that are contained in more than one repository using a single repository endpoint. You can set upstream repositories using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the AWS CodeArtifact API.

If an upstream repository has an external connection to a public repository, the repositories that are downstream from it can pull packages from that public repository. For example, suppose that the repository my_repo has an upstream repository named upstream, and upstream has an external connection to a public npm repository. In this case, a package manager that is connected to my_repo can pull packages from the npm public repository.

You can add one or more upstream repositories to an AWS CodeArtifact repository using the AWS Management Console, AWS CLI, or SDK. To associate a repository with an upstream repository, you must have permission for the AssociateWithDownstreamRepository action on the upstream repository.

For more information, see Create a repository with an upstream repository (p. 16) and Add an external connection (p. 26).

Topics
- Add, update, or remove upstream repositories (console) (p. 38)
- Add, update, or remove upstream repositories (AWS CLI) (p. 39)
- Requesting a package version with upstream repositories (p. 40)
- Upstream repository priority order (p. 42)
- API behavior with upstream repositories (p. 44)

Add, update, or remove upstream repositories (console)

To add an upstream repository using the CodeArtifact console:

2. In the navigation pane, choose Domains, and then choose the domain name that contains your repository.
3. Choose the name of your repository.
4. Choose Edit.
5. In Upstream repositories, click Associate upstream repository and add the repository to be added as an upstream repository.
6. Choose Update repository.

Note
This guide contains information about configuring other CodeArtifact repositories as upstream repositories. For information about configuring an external connection to public repositories like npmjs.com, Maven Central, or PyPI, see Add an external connection.
Add, update, or remove upstream repositories (AWS CLI)

You can add, update, or remove a CodeArtifact repository's upstream repositories using the AWS Command Line Interface (AWS CLI). To do this, use the `update-repository` command, and specify the upstream repositories using the `--upstreams` parameter.

**Note**
This guide contains information about configuring other CodeArtifact repositories as upstream repositories. For information about configuring an external connection to public repositories like npmjs.com, Maven Central, or PyPI, see Add an external connection.

The following command adds two upstream repositories to a repository named `my_repo` that is in a domain named `my_domain`. The order of the upstream repositories in the `--upstreams` parameter determines their search priority when CodeArtifact requests a package from the `my_repo` repository. For more information, see Upstream repository priority order (p. 42).

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

The output contains the upstream repositories, as follows.

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

To remove all upstream repositories from a repository, use the `update-repository` command and include `--upstreams` without an argument. The following removes upstream repositories from a repository named `my_repo` that is contained in a domain named `my_domain`.

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

The output shows that the list of `upstreams` is empty.

```
{
  "repository": {
    "name": "my_repo",
    "administratorAccount": "123456789012",
    "domainName": "my_domain",
    "domainOwner": "111122223333",
  }
}
```
Requesting a package version with upstream repositories

When a client (for example, npm) requests a package version from a CodeArtifact repository named `my_repo` that has multiple upstream repositories, the following can occur:

- If `my_repo` contains the requested package version, it is returned to the client.
- If `my_repo` does not contain the requested package version, CodeArtifact looks for it in `my_repo`'s upstream repositories. If the package version is found, a reference to it is copied to `my_repo`, and the package version is returned to the client.
- If neither `my_repo` nor its upstream repositories contain the package version, an HTTP 404 Not Found response is returned to the client.

When you add upstream repositories using the `create-repository` or `update-repository` command, the order they are passed to the `--upstreams` parameter determines their priority when a package version is requested. Specify upstream repositories with `--upstreams` in the order that you want CodeArtifact to use when a package version is requested. For more information, see Upstream repository priority order (p. 42).

The maximum number of direct upstream repositories allowed for one repository is 10. The maximum number of repositories CodeArtifact looks in when a package version is requested is 25.

Package retention from upstream repositories

If a requested package version is found in an upstream repository, a reference to it is retained and is always available from the downstream repository. The retained package version is not affected by any of the following:

- Deleting the upstream repository.
- Disconnecting the upstream repository from the downstream repository.
- Deleting the package version from the upstream repository.
- Editing the package version in the upstream repository (for example, by adding a new asset to it).

Fetch packages through an upstream relationship

If a CodeArtifact repository has an upstream relationship with a repository that has an external connection, requests for packages not in the upstream repository are copied from the external repository. For example, consider the following configuration: a repository named `repo-A` has an upstream repository named `repo-B`. `repo-B` has an external connection to `https://npmjs.com`.

If `npm` is configured to use the `repo-A` repository, running `npm install` triggers the copying of packages from `https://npmjs.com` into `repo-B`. The versions installed are also pulled into `repo-A`. The following example installs `lodash`. 

```bash
40
```
Fetch packages through an upstream relationship

```bash
$ npm config get registry
$ npm install lodash
+ lodash@4.17.20
added 1 package from 2 contributors in 6.933s
```

After running `npm install`, `repo-A` contains just the latest version (lodash 4.17.20) because that's the version that was fetched by `npm` from `repo-A`.

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

Example output:

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

Because `repo-B` has an external connection to https://npmjs.com, all the package versions that are imported from https://npmjs.com are stored in `repo-B`. These package versions could have been fetched by any downstream repository with an upstream relationship to `repo-B`.

The contents of `repo-B` provide a way to see all the packages and package versions imported from https://npmjs.com over time. For example, to see all the versions of the lodash package imported over time, you can use `list-package-versions`, as follows.

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

Example output:

```json
{
    "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"
        }
    ]
}
```
Package retention in intermediate repositories

CodeArtifact allows chaining upstream repositories. For example, repo-A can have repo-B as an upstream and repo-B can have repo-C as an upstream. This configuration makes the package versions in repo-B and repo-C available from repo-A.

When a package manager connects to repository repo-A and fetches a package version from repository repo-C, the package version will not be retained in repository repo-B. The package version will only be retained in the most-downstream repository, in this example, repo-A. It will not be retained in any intermediate repositories. This is also true for longer chains; for example if there were four repositories repo-A, repo-B, repo-C, and repo-D and a package manager connected to repo-A fetched a package version from repo-D, the package version would be retained in repo-A but not in repo-B or repo-C.

Package retention behavior is similar when pulling a package version from an external repository, except that the package version is always retained in the repository that has the external connection attached. For example, repo-A has repo-B as an upstream. repo-B has repo-C as an upstream, and repo-C also has npmjs.com configured as an external connection; see the following diagram.

If a package manager connected to repo-A requests a package version, lodash 4.17.20 for example, and the package version is not present in any of the three repositories, it will be fetched from npmjs.com. When lodash 4.17.20 is fetched, it will be retained in repo-A as that is the most-downstream repository and repo-C as it has the external connection to npmjs.com attached. lodash 4.17.20 will not be retained in repo-B as that is an intermediate repository.

Upstream repository priority order

When you request a package version from a repository with one or more upstream repositories, their priority corresponds to the order that they were listed when calling the create-repository or update-repository command. When the requested package version is found, the search stops, even if it didn’t search all upstream repositories. For more information, see Add, update, or remove upstream repositories (AWS CLI) (p. 39).

Use the describe-repository command to see the priority order.

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

The result might be the following. It shows that the upstream repository priority is upstream-1 first, upstream-2 second, and upstream-3 third.
Simple priority order example

In the following diagram, the `my_repo` repository has three upstream repositories. The priority order of the upstream repositories is `upstream-1`, `upstream-2`, `upstream-3`.

A request for a package version in `my_repo` searches the repositories in the following order until it is found, or until an HTTP 404 Not Found response is returned to the client:

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

If the package version is found, the search stops, even if it didn't look in all upstream repositories. For example, if the package version is found in `upstream-1`, the search stops and CodeArtifact doesn't look in `upstream-2` or `upstream-3`.

When you use the AWS CLI command `list-package-versions` to list package versions in `my_repo`, it looks only in `my_repo`. It does not list package versions in upstream repositories.

Complex priority order example

If an upstream repository has its own upstream repositories, the same logic is used to find a package version before moving to the next upstream repository. For example, suppose that your `my_repo` repository has two upstream repositories, A and B. If repository A has upstream repositories, a request for a package version in `my_repo` first looks in `my_repo`, second in A, then in the upstream repositories of A, and so on.

In the following diagram, the `my_repo` repository contains upstream repositories. Upstream repository A has two upstream repositories, and D has one upstream repository. Upstream repositories at the same
level in the diagram appear in their priority order, left to right (repository A has a higher priority order than repository B, and repository C has a higher priority order than repository D).

In this example, a request for a package version in my_repo looks in the repositories in the following order until it is found, or until a package manager returns an HTTP 404 Not Found response to the client:

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

**API behavior with upstream repositories**

When you call certain CodeArtifact APIs on repositories that are connected to upstream repositories, the behavior may be different depending on if the packages or package versions are stored in the target repository or the upstream repository. The behavior of these APIs is documented here.

For more information on CodeArtifact APIs, see the CodeArtifact API Reference.

Most APIs that reference a package or package version will return a ResourceNotFound error if the specified package version is not present in the target repository. This is true even if the package or package version is present in an upstream repository. Effectively, upstream repositories are ignored when calling these APIs. These APIs are:

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

To demonstrate this behavior, we have two repositories: target-repo and upstream-repo. target-repo is empty and has upstream-repo configured as an upstream repository. upstream-repo contains the npm package lodash.

When calling the DescribePackageVersion API on upstream-repo, which contains the lodash package, we get the following output:
When calling the same API on `target-repo`, which is empty but has `upstream-repo` configured as an upstream, we get the following output:

```json
{
  "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"
  }
}
```

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

The `CopyPackageVersions` API behaves differently. By default, `CopyPackageVersions` API only copies package versions that are stored in the target repository. If a package version is stored in the upstream repository but not in the target repository, it will not be copied. To include package versions of packages that are stored only in the upstream repository, set the value of `includeFromUpstream` to `true` in your API request.

For more information on the `CopyPackageVersions` API, see Copy packages between repositories (p. 58).
Working with packages in CodeArtifact

These topics show you how to list, copy, delete, and search for packages using the CodeArtifact CLI and API.

Topics
- Packages overview (p. 46)
- List package names (p. 49)
- List package versions (p. 52)
- List package version assets (p. 55)
- Download package version assets (p. 57)
- Copy packages between repositories (p. 58)
- Delete a package version (p. 62)
- View and update package version details and dependencies (p. 63)
- Update package version status (p. 66)

Packages overview

A package is a bundle of software and the metadata that is required to resolve dependencies and install the software. In CodeArtifact, a package consists of a package name, an optional namespace (p. 3) such as @types in @types/node, a set of package versions, and package-level metadata such as npm tags.

Contents
- Supported package formats (p. 46)
- Package publishing (p. 46)
  - Publishing permissions (p. 47)
  - Overwriting package assets (p. 47)
  - Publishing and upstream repositories (p. 47)
  - Private packages and public repositories (p. 47)
  - Publishing patched package versions (p. 48)
- Package version status (p. 48)

Supported package formats

AWS CodeArtifact supports npm (p. 84), PyPI (p. 96), Maven (p. 100), and NuGet (p. 113) package formats.

Package publishing

You can publish new versions of any supported package format (p. 46) to a CodeArtifact repository using tools such as npm, twine, Maven, Gradle, nuget, and dotnet.
Publishing permissions

Your AWS Identity and Access Management (IAM) user or role must have permissions to publish to the destination repository. The following permissions are required to publish packages.

1. **Maven**: `codeartifact:PublishPackageVersion` and `codeartifact:PutPackageMetadata`
2. **npm**: `codeartifact:PublishPackageVersion`
3. **NuGet**: `codeartifact:PublishPackageVersion` and `codeartifact:ReadFromRepository`
4. **Python**: `codeartifact:PublishPackageVersion`

In the list of permissions above, your IAM policy must specify the package resource for the `codeartifact:PublishPackageVersion` and `codeartifact:PutPackageMetadata` permissions and it must specify the repository resource for the `codeartifact:ReadFromRepository` permission.


Overwriting package assets

You can't republish a package asset that already exists with different content. For example, suppose that you already published a Maven package with a JAR asset `mypackage-1.0.jar`. You can't publish that asset again unless the checksum of the old and new assets are identical. To republish the same asset with new content, delete the package version using the `delete-package-versions` command first. Trying to republish the same asset name with different content will result in an HTTP 409 conflict error.

For package formats that support multiple assets (PyPI and Maven), you can add new assets with different names to an existing package version at any time, assuming that you have the required permissions. Because npm only supports a single asset per package version, to modify a published package version in any way, you must first delete it using `delete-package-versions`.

If you try to republish an asset that already exists (for example, `mypackage-1.0.jar`), and the content of the published asset and the new asset are identical, the operation will succeed because the operation is idempotent.

Publishing and upstream repositories

CodeArtifact doesn't allow publishing package versions that are present in reachable upstream repositories or public repositories. For example, suppose that you want to publish a Maven package `com.mycompany.mypackage:1.0` to a repository `myrepo`, and `myrepo` has an upstream repository with an external connection to Maven Central. If `com.mycompany.mypackage:1.0` is present in the upstream repository or in Maven Central, CodeArtifact rejects any attempt to publish to it in `myrepo` with a 409 conflict error. This helps prevent scenarios where you might accidentally publish a package with the same name and version as a package in an upstream repository, which can result in unexpected behavior.

You can still publish different versions of a package name that exists in an upstream repository. For example, if `com.mycompany.mypackage:1.0` is present in an upstream repository, but `com.mycompany.mypackage:1.1` is not, you can publish `com.mycompany.mypackage:1.1` to the downstream repository.

Private packages and public repositories

CodeArtifact does not publish packages stored in CodeArtifact repositories to public repositories such as npmjs.com or Maven Central. CodeArtifact imports packages from public repositories to a CodeArtifact...
repository, but it never moves packages in the other direction. Packages that you publish to CodeArtifact repositories remain private and are only available to the AWS accounts, roles, and users to which you have granted access.

**Publishing patched package versions**

Sometimes you might want to publish a modified package version, potentially one that is available in a public repository. For example, you might have found a bug in a critical application dependency called `mydep 1.1`, and you need to fix it sooner than the package vendor can review and accept the change. As described previously, CodeArtifact prevents you from publishing `mydep 1.1` in your CodeArtifact repository if the public repository is reachable from your CodeArtifact repository via upstream repositories and an external connection.

To work around this, publish the package version to a different CodeArtifact repository where the public repository isn't reachable. Then use the `copy-package-versions` API to copy the patched version of `mydep 1.1` to the CodeArtifact repository where you will consume it from.

**Package version status**

Every package version in CodeArtifact has a status that describes the current state and availability of the package version. You can change the package version status using both the AWS CLI and the console. For more information, see Update package version status (p. 66).

The following are possible values for package version status:

- **Published**: The package version is successfully published and can be requested using a package manager. The package version will be included in the output of the `ListPackageVersions` API and included in package versions lists returned to package managers, for example, in the output of `npm view <package-name> versions`. All assets of the package version are available from the repository.

- **Unfinished**: The last attempt to publish did not complete. Currently only Maven package versions can have a status of **Unfinished**. This can occur when the client uploads one or more assets for a package version but does not publish a `maven-metadata.xml` file for the package that includes that version. When a package version is **Unfinished**, it will not be included in the default output from the `ListPackageVersions` API.

- **Unlisted**: The package version's assets are available for download from the repository, but it is not included in the default output of the `ListPackageVersions` API. The package version is also not included in the list of versions returned to package managers. For example, for an npm package, the output of `npm view <package-name> versions` will not include the package version. This means that npm's dependency resolution logic will not select the package version because the version does not appear in the list of available versions. However, if the **Unlisted** package version is already referenced in an `npm package-lock.json` file, it can still be downloaded and installed, for example, when running `npm ci`.

- **Archived**: The package version's assets can no longer be downloaded. The package version will not appear in the default output of the `ListPackageVersions` API, and will not be included in the list of versions returned to package managers. Because the the assets are not available, consumption of the package version by clients is blocked. If your application build depends on a version that is updated to **Archived**, the build will break, assuming the package version has not been locally cached. You cannot use a package manager or build tool to re-publish an **Archived** package version because it is still present in the repository, but you can change the package version's status back to **Unlisted** or **Published** with the `UpdatePackageVersionsStatus` API.

- **Disposed**: The package version doesn't appear in listings and the assets cannot be downloaded from the repository. They key difference between **Disposed** and **Archived** is that with a status of **Disposed**, the assets of the package version will be permanently deleted by CodeArtifact. For this reason, you cannot move a package version from **Disposed** to **Archived**, **Unlisted**, or **Published**. The package
version can no longer be used because the assets have been deleted. Once a package version has been marked as **Disposed**, you will no longer be billed for storage of the package assets.

Apart from the states above, a package version can also be deleted with the DeletePackageVersions API. Once deleted, a package version is no longer in the repository and you can freely re-publish that package version using a package manager or build tool. Once a package version has been deleted, you will no longer be billed for storage of that package version's assets.

## List package names

Use the `list-packages` command in CodeArtifact to get a list of all the package names in a repository. This command returns only the package names, not the versions.

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

Sample output:

```json
{
  "nextToken": "eyJidWNrZXRJZCI6I...
,  "packages": [
    {  
      "package": "acorn",
      "format": "npm"
    },
    {  
      "package": "acorn-dynamic-import",
      "format": "npm"
    },
    {  
      "package": "ajv",
      "format": "npm"
    },
    {  
      "package": "ajv-keywords",
      "format": "npm"
    },
    {  
      "package": "anymatch",
      "format": "npm"
    },
    {  
      "package": "ast",
      "namespace": "webassemblyjs",
      "format": "npm"
    }
  ]
}
```

## List npm package names

To list only the names of npm packages, set the value of the `--format` option to `npm`.

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

To list npm packages in a namespace (npm scope), use the --namespace and --format options.

**Important**
The value for the --namespace option should not include the leading @. To search for the namespace @types, set the value to types.

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

Sample output:

```
{
   "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"
      }
   ]
}
```

**List Maven package names**

To list only the names of Maven packages, set the value of the --format option to maven. You must also specify the Maven group in the --namespace option.

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

Sample output:

```
{
   "nextToken": "eyJidWNrZXRJZ...",
   "packages": [
      {
         "package": "commons-lang3",
         "namespace": "org.apache.commons",
         "format": "maven"
      },
      {
         "package": "commons-collections4",
         "namespace": "org.apache.commons",
         "format": "maven"
      }
   ]
}
```
List Python package names

To list only the names of Python packages, set the value of the `--format` option to `pypi`.

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

Filter by package name prefix

To return packages that begin with a specified string, you can use the `--package-prefix` option.

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

Sample output:

```
{
  "nextToken": "eyJidWNrZXRJZ...",
  "packages": [
    {
      "package": "path",
      "format": "npm"
    },
    {
      "package": "pat-test",
      "format": "npm"
    },
    {
      "package": "patch-math3",
      "format": "npm"
    }
  ]
}
```

Supported search option combinations

You can use the `--format`, `--namespace`, and `--package-prefix` options in any combination, except that `--namespace` can't be used by itself. For example, searching for all packages in the `@types` scope requires the `--format` option to be specified. Using `--namespace` by itself results in an error.
Using none of the three options is also supported by `list-packages` and will return all packages of all formats present in the repository.

**Format output**

You can use parameters that are available to all AWS CLI commands to make the `list-packages` response compact and more readable. Use the `--query` parameter to specify the format of each returned package version. Use the `--output` parameter to format the response as plaintext.

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

Sample output:

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

For more information, see Controlling command output from the AWS CLI in the AWS Command Line Interface User Guide.

**Defaults and other options**

By default, the maximum number of results returned by `list-packages` is 100. You can change this result limit by using the `--max-results` option.

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

The maximum allowed value of `--max-results` is 1,000. To allow listing packages in repositories with more than 1,000 packages, `list-packages` supports pagination using the `nextToken` field in the response. If the number of packages in the repository is more than the value of `--max-results`, you can pass the value of `nextToken` to another invocation of `list-packages` to get the next page of results.

```bash
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --repository my_repo \ --next-token r00AbXNyAEdjb...
```

**List package versions**

Use the `list-package-versions` command in CodeArtifact to get a list of all of the versions of a package name in a repository.

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

```json
{
    "defaultDisplayVersion": "1.0.1",
    "format": "npm",
    "package": "kind-of",
    "versions": [
        {
            "version": "1.0.1",
            "revision": "REVISION-SAMPLE-1-C7F455E9B772FC",
            "status": "Published"
        },
        {
            "version": "1.0.0",
            "revision": "REVISION-SAMPLE-2-C752BEEF6D2CFC",
            "status": "Published"
        },
        {
            "version": "0.1.2",
            "revision": "REVISION-SAMPLE-3-654565A5C5E1FC",
            "status": "Published"
        },
        {
            "version": "0.1.1",
            "revision": "REVISION-SAMPLE-1-C7F455E9B772FC",
            "status": "Published"
        },
        {
            "version": "0.1.0",
            "revision": "REVISION-SAMPLE-4-AF669139B772FC",
            "status": "Published"
        }
    ]
}
```

You can add the `--status` parameter to the `list-package-versions` call to filter the results based on package version status. For more information on package version status, see Package version status (p. 48).

You can paginate the response from `list-package-versions` using the `--max-results` and `--next-token` parameters. For `--max-results`, specify an integer from 1 to 1000 to specify the number of results returned in a single page. Its default is 50. To return subsequent pages, run `list-package-versions` again and pass the `nextToken` value received in the previous command output to `--next-token`. When the `--next-token` option is not used, the first page of results is always returned.

The `list-package-versions` command does not list package versions in upstream repositories. However, references to package versions in an upstream repository that were copied to your repository during a package version request are listed. For more information, see Working with upstream repositories in CodeArtifact (p. 38).

Sort versions

`list-package-versions` can output versions sorted in descending order based on publish time (the most-recently published versions are listed first). Use the `--sort-by` parameter with a value of `PUBLISH_TIME`, as follows.

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

Sample output:
```json
{
  "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...."
}
```

**Default display version**

The return value for `defaultDisplayVersion` depends on the package format:

- For Maven and PyPI packages, it’s the most recently published package version.
- For npm packages, it’s the version referenced by the `latest` tag. If the `latest` tag is not set, it’s the most recently published package version.

**Format output**

You can use parameters that are available to all AWS CLI commands to make the `list-package-versions` response compact and more readable. Use the `--query` parameter to specify the format of each returned package version. Use `--output` parameter to format the response as plain text.

```
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]' 
```

Sample output:

```
0.1.1
0.1.2
0.1.0
```
List package version assets

An asset is an individual file (for example, an npm .tgz file or Maven POM or JAR file) stored in CodeArtifact that is associated with a package version. You can use the list-package-version-assets command to list the assets in each package version.

Run the list-package-version-assets command to return the following information about each asset in your AWS account and your current AWS Region:

- Its name.
- Its size, in bytes.
- A set of hash values used for checksum validation.

For example, use the following command to list the assets of the Python package flatten-json, version 0.1.7.

```bash
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
```

The following shows the output.

```json
{
   "format": "pypi",
   "package": "flatten-json",
   "version": "0.1.7",
   "versionRevision": "REVISION-SAMPLE-1-C7F455E987272FC",
   "assets": [
      {
         "name": "flatten_json-0.1.7-py3-none-any.whl",
         "size": 31520,
         "hashes": {
            "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
            "SHA-1": "69b215c25dd4cdad997a786ec6EXAMPLE-SHA-1",
            "SHA-256": "43f24850b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-SHA-256",
            "SHA-512": "3947382ac2c180ee3f2aba4f8788241527c8db9defe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f0864e0dec3a
                       SHA-512"
         }
      },
      {
         "name": "flatten_json-0.1.7.tar.gz",
         "size": 2865,
         "hashes": {
            "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
            "SHA-1": "69b215c25dd4cdad997a786ec6EXAMPLE-SHA-1",
            "SHA-256": "43f24850b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-SHA-256",
            "SHA-512": "3947382ac2c180ee3f2aba4f8788241527c8db9defe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f0864e0dec3a
                       SHA-512"
         }
      }
   ]
}
```
To list the assets of the Maven package `commons-cli`:

```bash
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
```

The following shows the output.

```json
{
    "format": "maven",
    "namespace": "commons-cli",
    "package": "commons-cli",
    "version": "1.0",
    "versionRevision": "REVISION-SAMPLE-55C752BEE9B772FC",
    "assets": [
        {
            "name": "commons-cli-1.0.jar",
            "size": 30117,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25d4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256": "43f24850b7b7d79c5fa652418518fbd427e602b1edabe6EXAMPLE-SHA-256",
                "SHA-512": "3947382ac2c180ee3f2aba4f8788241527c8db9de9f4b039abe9fc560aaf5af5afced7bd1e80a0dca9ce320d95f0864e0dec3a4c99978e602b1edabe6EXAMPLE-SHA-512"
            }
        },
        {
            "name": "commons-cli-1.0.pom",
            "size": 2105,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25d4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256": "43f24850b7b7d79c5fa652418518fbd427e602b1edabe6EXAMPLE-SHA-256",
                "SHA-512": "3947382ac2c180ee3f2aba4f8788241527c8db9de9f4b039abe9fc560aaf5af5afced7bd1e80a0dca9ce320d95f0864e0dec3a4c99978e602b1edabe6EXAMPLE-SHA-512"
            }
        },
        {
            "name": "maven-metadata.xml",
            "size": 119,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25d4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256": "43f24850b7b7d79c5fa652418518fbd427e602b1edabe6EXAMPLE-SHA-256",
                "SHA-512": "3947382ac2c180ee3f2aba4f8788241527c8db9de9f4b039abe9fc560aaf5af5afced7bd1e80a0dca9ce320d95f0864e0dec3a4c99978e602b1edabe6EXAMPLE-SHA-512"
            }
        }
    ]
}
```

An npm package always has a single asset with a name of `package.tgz`. 

56
aws codeartifact list-package-version-assets --domain my_domain --domain-owner 1111222333333
  --repository my_repo \  
  --format npm --package webpack --package-version 4.9.2

The following shows the output.

```
{
"format": "npm",
"package": "webpack",
"version": "4.9.2",
"versionRevision": "REVISION-SAMPLE-55C752BEE9B772FC",
"assets": [
  {
    "name": "package.tgz",
    "size": 242930,
    "hashes": {"MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
               "SHA-256": "43f24850b7b7d79c5fa65241851fbd6f427e602b1edabe6EXAMPLE-SHA-256",
               "SHA-512": "3947382ac2c180ee3f2aba4f8f788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f0864e0dec3aEXAMPLE-SHA-512"
  }
]
}
```

Download package version assets

An asset is an individual file (for example, an npm `.tgz` file or Maven POM or JAR file) stored in CodeArtifact that is associated with a package version. You can download package assets using the `get-package-version-assets` command. This allows you to retrieve assets without using a package manager client such as npm or pip. To download an asset you must provide the asset’s name which can be obtained using the `list-package-version-assets` command, for more information see List package version assets (p. 55). The asset will be downloaded to local storage with a file name that you specify.

The following example downloads the `guava-27.1-jre.jar` asset from the Maven package `com.google.guava:guava` with version `27.1-jre`.

```
aws codeartifact get-package-version-asset --domain my_domain --domain-owner 1111222333333
  --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
```

The output of the command will be:

```
{
  "assetName": "guava-27.1-jre.jar",
  "packageVersion": "27.1-jre",
  "packageVersionRevision": "YGp9ck2tmy03P9Sxioc1fYzQ0BfT98zhQ7bERv62I="
}
```

In this example, the file name was specified as `guava-27.1-jre.jar` by the last argument in the command above, so the downloaded asset will be named `guava-27.1-jre.jar`. 

57
Copy packages between repositories

You can copy package versions from one repository to another in CodeArtifact. This can be helpful for scenarios such as package promotion workflows or sharing package versions between teams or projects. The source and destination repositories must be in the same domain to copy package versions.

Required IAM permissions to copy packages

To copy package versions in CodeArtifact, the calling user must have the required IAM permissions and the resource-based policy attached to the source and destination repositories must have the required permissions. For more information about resource-based permissions policies and CodeArtifact repositories, see Repository policies (p. 22).

The user calling copy-package-versions must have the ReadFromRepository permission on the source repository and the CopyPackageVersions permission on the destination repository.

The source repository must have the ReadFromRepository permission and the destination repository must have the CopyPackageVersions permission assigned to the IAM account or user copying packages. The following policies are example repository policies to be added to the source repository or destination repository with the put-repository-permissions-policy command. Replace 111122223333 with the ID of the account calling copy-package-versions.

Note
Calling put-repository-permissions-policy will replace the current repository policy if one exists. You can use the get-repository-permissions-policy command to see if a policy exists, for more information see Read a policy (p. 24). If a policy does exist, you may want to add these permissions to it instead of replacing it.

Example source repository permissions policy

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

Example destination repository permissions policy

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

Use the `copy-package-versions` command in CodeArtifact to copy one or more package versions from a source repository to a destination repository in the same domain. The following example will copy versions 6.0.2 and 4.0.0 of an npm package named `my-package` from the `my_repo` repository to the `repo-2` repository.

```bash
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
```

You can copy multiple versions of the same package name in a single operation. To copy versions of different package names, you must call `copy-package-versions` for each one.

The previous command will produce the following output, assuming both versions could be copied successfully.

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

Copy a package from upstream repositories

Normally, `copy-package-versions` only looks in the repository specified by the `--source-repository` option for versions to copy. However, you can copy versions from both the source repository and its upstream repositories by using the `--include-from-upstream` option. If you use the CodeArtifact SDK, call the `CopyPackageVersions` API with the `includeFromUpstream` parameter set to true. For more information, see Working with upstream repositories in CodeArtifact (p. 38).

Copy a scoped npm package

To copy an npm package version in a scope, use the `--namespace` option to specify the scope. For example, to copy the package `@types/react`, use `--namespace types`. The `@` symbol must be omitted when using `--namespace`.
Copy Maven package versions

To copy Maven package versions between repositories, specify the package to copy by passing the Maven groupId with the `--namespace` option and the Maven artifactId with the `--name` option. For example, to copy a single version of `com.google.guava:guava`:

```bash
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
```

If the package version is copied successfully, the output will be similar to the following.

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

Versions that do not exist in the source repository

If you specify a version that does not exist in the source repository, the copy will fail. If some versions exist in the source repository and some do not, all versions will fail to copy. In the following example, version 0.2.0 of the `array-unique` npm package is present in the source repository, but version 5.6.7 is not:

```bash
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
```

The output in this scenario will be similar to the following.

```json
{
  "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"
    }
  }
}
```

The `SKIPPED` error code is used to indicate that the version was not copied to the destination repository because another version was not able to be copied.
Versions that already exist in the destination repository

When a package version is copied to a repository where it already exists, CodeArtifact compares its package assets and package version level metadata in the two repositories.

If the package version assets and metadata are identical in the source and destination repositories, a copy is not performed but the operation is considered successful. This means that `copy-package-versions` is idempotent. When this occurs, the version that was already present in both the source and destination repositories will not be listed in the output of `copy-package-versions`.

In the following example, two versions of the npm package `array-unique` are present in the source repository `repo-1`. Version 0.2.1 is also present in the destination repository `dest-repo` and version 0.2.0 is not.

```bash
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
```

The output in this scenario will be similar to the following.

```json
{
   "successfulVersions": {
      "0.2.0": {
         "revision": "Yad+B1QcBq2kdEVrx1E1vSfHJvh8Pr61hbUkoWPWXO=",
         "status": "Published"
      }
   },
   "failedVersions": {}
}
```

Version 0.2.0 is listed in `successfulVersions` because it was successfully copied from the source to destination repository. Version 0.2.1 is not shown in the output as it was already present in the destination repository.

If the package version assets or metadata differ in the source and destination repositories, the copy operation will fail. You can use the `--allow-overwrite` parameter to force an overwrite.

If some versions exist in the destination repository and some do not, all versions will fail to copy. In the following example, version 0.3.2 of the `array-unique` npm package is present in both the source and destination repositories, but the contents of the package version are different. Version 0.2.1 is present in the source repository but not the destination.

```bash
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
```

The output in this scenario will be similar to the following.

```json
{
   "successfulVersions": {},
   "failedVersions": {
      "0.2.1": {
         "errorCode": "SKIPPED",
         "errorMessage": "Version 0.2.1 was skipped"
      }
   }
}
```
Specifying a package version revision

A package version revision is a string that specifies a specific set of assets and metadata for a package version. You can specify a package version revision to copy package versions that are in a specific state. To specify a package version revision, use the `--version-revisions` parameter to pass one or more comma-separated package version and the package version revision pairs to the `copy-package-versions` command.

Note
You must specify the `--versions` or the `--version-revisions` parameter with `copy-package-versions`. You cannot specify both.

The following example will only copy version 0.3.2 of the package my-package if it is present in the source repository with package version revision `REVISION-1-SAMPLE-6C81EFF7DA55CC`.

```
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
```

The following example copies two versions of package my-package, 0.3.2 and 0.3.13. The copy will only succeed if in the source repository version 0.3.2 of my-package has revision `REVISION-1-SAMPLE-6C81EFF7DA55CC` and version 0.3.13 has revision `REVISION-2-SAMPLE-55C752BEE772FC`.

```
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
```

To find the revisions of a package version, use the `describe-package-version` or the `list-package-versions` command.

For more information, see Package version revision (p. 2) and CopyPackageVersion in the CodeArtifact API Reference.

Copy npm packages

For more information about `copy-package-versions` behavior with npm packages, see npm tags and the CopyPackageVersions API (p. 93).

Delete a package version

You can delete one or more package versions at a time using the `delete-package-versions` command. The following deletes versions 4.0.0, 4.0.1, and 5.0.0 of the npm package named my-package in the `my_repo` in the `my_domain` domain:
aws codeartifact delete-package-versions --domain my_domain --domain-owner 111122223333 --repository my_repo \  --format npm --package my-package --versions 4.0.0 4.0.1 5.0.0

Sample output:

```json
{
    "successfulVersions": {
        "4.0.0": {
            "revision": "oxwwYC9dDeuBoCt6+PDSw6CMZ7rXeiXY44BM321awo=",
            "status": "Deleted"
        },
        "4.0.1": {
            "revision": "byaaQR748wrsdBaT+PDSw6CMZ7rXeiBKM055laaqWhm0=",
            "status": "Deleted"
        },
        "5.0.0": {
            "revision": "yubm34QWeST345ts+ASeioPI354rXeiSWr734PotwRw=",
            "status": "Deleted"
        }
    },
    "failedVersions": {}
}
```

You can confirm that the versions were deleted by running `list-package-versions` for the same package name:

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

---

**View and update package version details and dependencies**

You can view information about a package version, including dependencies, in CodeArtifact. You can also update the status of a package version. For more information on package version status, see Package version status (p. 48).

### View package version details

Use the `describe-package-version` command to view details about package versions. Package version details are extracted from a package when it is published to CodeArtifact. The details in different packages vary and depend on their formats and how much information their authors added to them.

Most information in the output of the `describe-package-version` command depends on the package format. For example, `describe-package-version` extracts an npm package's information from its `package.json` file. The revision is created by CodeArtifact. For more information, see Specifying a package version revision (p. 62).

Two package versions with the same name can be in the same repository if they each are in different namespaces. Use the optional `--namespace` parameter to specify a namespace. The following returns details about version 4.41.5 of an npm package named webpack.
The following example returns details about version 1.2 of a Maven package named commons-rng-client-api that is in the org.apache.commons namespace and the my_repo repository.

```
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
```

The output might look like the following.

```
{
  "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"
}
```

The following example returns details about version 1.9.0 of a Python package named pyhamcrest that is in the my_repo repository.

```
aws codeartifact describe-package-version --domain my_domain --domain-owner 111122223333 --repository my_repo \
--format pypi --namespace org.apache.commons --package pyhamcrest --package-version 1.9.0
```

The output might look like the following.

```
{
  "format": "pypi",
  "package": "PyHamcrest",
  "summary": ""...
```
View package version dependencies

Use the `list-package-version-dependencies` command to get a list of a package version’s dependencies. The following command lists the dependencies of an npm package named `my-package`, version 4.41.5, in the `my_repo` repository, in the `my_domain` domain.

```
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
```

The output might look like the following.

```
{
    "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"
}
```

View package version readme file

Some package formats, such as npm, include a README file. Use the `get-package-version-readme` to get the README file of a package version. The following command returns the README file of an npm package named `my-package`, version 4.41.5, in the `my_repo` repository, in the `my_domain` domain.

```
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
```
Update package version status

Every package version in CodeArtifact has a status that describes the current state and availability of the package version. You can change the package version status using both the AWS CLI and the console. For more information on package version status, including a list of the available statuses, see Package version status (p. 48).

Updating package version status

Setting the status of a package version allows controlling how a package version can be used without deleting it completely from the repository. For example, when a package version has a status of Unlisted, it can still be downloaded as normal, but it will not appear in package version lists returned to commands such as `npm view`. The `UpdatePackageVersionsStatus API` allows setting the package version status of multiple versions of the same package in a single API call. For a description of the different statuses, see Packages overview (p. 46).

Use the `update-package-versions-status` command to change the status of a package version to Published, Unlisted, or Archived. To see the required IAM permissions to use the command, see Required IAM permissions to update a package version status (p. 67). The following example sets the status of version 4.1.0 of the npm package `chalk` to 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
```

Sample output:

```
{
    "successfulVersions": {
        "4.1.0": {
            "revision": "+Oz8skWbwY3k8M6SrNIqNj6bVH/ax+CXvkJx+N0Sj8I=",
            "status": "Archived"
        }
    },
    "failedVersions": {}
}
```

This example uses an npm package, but the command works identically for other formats. Multiple versions can be moved to the same target status using a single command, see the following example.

```
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
```
Sample output:

```
{
  "successfulVersions": {
    "4.1.0": {
      "revision": "25/UjBleHa1DZewk+zozoeqH/R80Rc9gL1P8vbzVMJ4=",
      "status": "Archived"
    },
    "4.1.1": {
      "revision": "+Oz8skWbwY3k8M6SrNIqNj6bVH/ax+CxvkJx+No5j8I=",
      "status": "Archived"
    }
  },
  "failedVersions": {}
}
```

Note that once published, a package version cannot be moved back to the Unfinished state, so this status is not permitted as a value for the `--target-status` parameter. To move the package version to the Disposed state, use the `dispose-package-versions` command instead as described below.

**Required IAM permissions to update a package version status**

To call `update-package-versions-status` for a package, you must have the `codeartifact:UpdatePackageVersionsStatus` permission on the package resource. This means you can grant permission to call `update-package-versions-status` on a per-package basis. For example, an IAM policy that grants permission to call `update-package-versions-status` on the npm package `chalk` would include a statement like the following.

```
{
  "Action": [
    "codeartifact:UpdatePackageVersionsStatus"
  ],
  "Effect": "Allow",
}
```

**Updating status for a scoped npm package**

To update the package version status of an npm package version with a scope, use the `--namespace` parameter. For example, to unlist version 8.0.0 of `@nestjs/core`, use the following command.

```
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
```

**Updating status for a Maven package**

Maven packages always have a groupID, which is referred to as a namespace in CodeArtifact. Use the `--namespace` parameter to specify the Maven groupID when calling `update-package-versions-status`. For example, to archive version 2.13.1 of the Maven package `org.apache.logging.log4j:log4j`, use the following command.

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

67
Specifying a package version revision

A package version revision is a string that specifies a specific set of assets and metadata for a package version. You can specify a package version revision to update the status of package versions that are in a specific state. To specify a package version revision, use the `--version-revisions` parameter to pass one or more comma-separated package versions and the package version revision pairs. The status of a package version will only be updated if the current revision of the package version matches the value specified.

**Note**

The `--versions` parameter must also be defined when using the `--version-revisions` parameter.

```bash
aws codeartifact update-package-versions-status --domain my_domain
--domain-owner 111122223333 --repository my_repo --format maven
--package log4j
--target-status Archived
```

Sample output:

```
{
  "successfulVersions": {
    "4.0.0": {
      "revision": "E3lhBp0RoBTut4pkjV5c1AQGkgSA7Oxtil6hMMz6c=",
      "status": "Published"
    },
    "4.1.0": {
      "revision": "25/UjBleHs1DZeWk+zozeqH/R80rc9gL1P8bzVMJ4=",
      "status": "Published"
    }
  },
  "failedVersions": {}
}
```

When updating multiple package versions, the versions passed to `--version-revisions` must be the same as the versions passed to `--versions`. If a revision is specified incorrectly, that version will not have its status updated.

**Using the expected status parameter**

The `update-package-versions-status` command provides the `--expected-status` parameter that supports specifying the expected current status of a package version. If the current status does not match the value passed to `--expected-status`, the status of that package version will not be updated.
For example, in *my_repo*, versions 4.0.0 and 4.1.0 of the npm package *chalk* currently have a status of *Published*. A call to `update-package-versions-status` that specifies an expected status of *Unlisted* will fail to update both package versions because of the status mismatch.

```bash
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
```

Sample output:

```json
{
  "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"
    }
  }
}
```

Errors with individual package versions

There are multiple reasons why the status of a package version will not be updated when calling `update-package-versions-status`. For example, the package version revision may have been specified incorrectly, or the expected status does not match the current status. In these cases, the version will be included in the `failedVersions` map in the API response. If one version fails, other versions specified in the same call to `update-package-versions-status` might be skipped and not have their status updated. Such versions will also be included in the `failedVersions` map with an `errorCode` of `SKIPPED`.

In the current implementation of `update-package-versions-status`, if one or more versions cannot have their status changed, all other versions will be skipped. That is, either all versions are updated successfully or no versions are updated. This behavior is not guaranteed in the API contract; in the future, some versions might succeed while other versions fail in a single call to `update-package-versions-status`.

The following example command includes an version status update failure caused by a package version revision mismatch. That update failure causes another version status update call to be skipped.

```bash
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/Uj8leHs1DZewk+zo30eqM/RS0Rg9LIvPzbvKJ7/4.0.0=EO1hBp0ObF7ut4pkjY5c1AOGxGkA70xtil6MMzLc=
--versions 4.1.0 4.0.0 --target-status Archived
```

Sample output:

```json
{
  "successfulVersions": {},
  "failedVersions": {
    "4.0.0": {
      "errorCode": "SKIPPED",
      "errorMessage": "version 4.0.0 is skipped"
    }
  }
}
```
Disposing of package versions

The Disposed package status has similar behavior to Archived, except that the package assets will be permanently deleted by CodeArtifact so that the domain owner’s account will no longer be billed for the asset storage. To change the status of a package version to Disposed, use the dispose-package-versions command. This capability is separate from update-package-versions-status because disposing of a package version is not reversible. Because the package assets will be deleted, the version’s status cannot be changed back to Archived, Unlisted, or Published. The only action that can be taken on a package version that has been disposed is for it to be deleted using the delete-package-versions command.

To call dispose-package-versions successfully, the calling IAM principal must have the codeartifact:DisposePackageVersions permission on the package resource.

The behavior of the dispose-package-versions command is similar to update-package-versions-status, including the behavior of the --version-revisions and --expected-status options that are described in the version revision (p. 68) and expected status (p. 68) sections. For example, the following command attempts to dispose a package version but fails due to a mismatched expected status.

```
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
```

Sample output:

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

If the same command is run again with an --expected-status of Published, the disposal will succeed.

```
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
```

Sample output:

```json
{
  "successfulVersions": {
    "4.0.0": {"70
```
"revision": "E3lhBp0RObRTut4pkjV5c1AQGkkSA7Oxtil6hMz1c=",
"status": "Disposed"
},
"failedVersions": {}
Working with domains in CodeArtifact

CodeArtifact domains make it easier to manage multiple repositories across an organization. You can use a domain to apply permissions across many repositories owned by different AWS accounts. An asset is stored only once in a domain, even if it's available from multiple repositories.

Although you can have multiple domains, we recommend a single production domain that contains all published artifacts so that your development teams can find and share packages. You can use a second preproduction domain to test changes to the production domain configuration.

These topics describe how to use the CLI and API to configure CodeArtifact domains.

Topics
- Domain overview (p. 72)
- Create a domain (p. 73)
- Delete a domain (p. 75)
- Domain policies (p. 75)
- Tag a domain in CodeArtifact (p. 79)

Domain overview

When you're working with CodeArtifact, domains are useful for the following:

- **Deduplicated storage**: An asset only needs to be stored once in a domain, even if it's available in 2 or 2,000 repositories. That means you only pay for storage once.

- **Fast copying**: When you pull packages from an upstream CodeArtifact repository into a downstream or use the CopyPackageVersions API (p. 58), only metadata records must be updated. No assets are copied. This makes it fast to set up a new repository for staging or testing. For more information, see Working with upstream repositories in CodeArtifact (p. 38).

- **Easy sharing across repositories and teams**: All of the assets and metadata in a domain are encrypted with a single AWS KMS key (KMS key). You don't need to manage a key for each repository or grant multiple accounts access to a single key.

- **Apply policy across multiple repositories**: The domain administrator can apply policy across the domain. This includes restricting which accounts have access to repositories in the domain, and who can configure connections to public repositories to use as sources of packages. For more information, see Domain policies (p. 75).

- **Unique repository names**: The domain provides a namespace for repositories. Repository names only need to be unique within the domain. You should use meaningful names that are easy to understand.

Domain names must be unique within an account.

You cannot create a repository without a domain. When you use the CreateRepository (p. 14) API to create a repository, you must specify a domain name. You cannot move a repository from one domain to another.
A repository can be owned by the same AWS account that owns the domain, or a different account. If the owning accounts are different, the repository-owning account must be granted the `CreateRepository` permission on the domain resource. You can do this by adding a resource policy to the domain using the `PutDomainPermissionsPolicy` command.

Although an organization can have multiple domains, the recommendation is to have a single production domain that contains all published artifacts so that development teams can find and share packages across their organization. A second pre-production domain can be useful for testing changes to the production domain configuration.

### Cross-account domains

Domain names only need to be unique within an account, which means there could be multiple domains within a region that have the same name. Because of this, if you want to access a domain that is owned by an account you are not authenticated to, you must provide the domain owner ID along with the domain name in both the CLI and the console. See the following CLI examples.

**Access a domain owned by an account you are authenticated to:**

When accessing a domain within the account you're authenticated to, you only need to specify the domain name. The following example lists packages in the `my_repo` repository in the `my_domain` domain that is owned by your account.

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

**Access a domain owned by an account that you are not authenticated to:**

When accessing a domain that is owned by an account that you're not authenticated to, you need to specify the domain owner as well as the domain name. The following example lists packages in the `other-repo` repository in the `other-domain` domain that is owned by an account that you are not authenticated to. Notice the addition of the `--domain-owner` parameter.

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

### Create a domain

You can create a domain using the CodeArtifact console or the AWS Command Line Interface (AWS CLI). When you create a domain, it does not contain any repositories. For more information, see Create a repository.

**Topics**
- Create a domain (console)
- Create a domain (AWS CLI)

**Create a domain (console)**

2. In the navigation pane, choose **Domains**, and then choose **Create domain**.
3. In **Name**, enter a name for your domain.
4. Expand **Additional configuration**.

5. You must use an AWS KMS key (KMS key) to encrypt all assets in your domain. You can use an AWS managed KMS key or a KMS key that you manage.

   - Choose **AWS managed key** if you want to use the default AWS managed key.
   - Choose **Customer managed key** if you want to use a KMS key that you manage. To use a KMS key that you manage, in **Customer managed key ARN**, search for and choose the KMS key.

   For more information, see [AWS managed key](https://docs.aws.amazon.com/codeartifact/latest/userguide/create-domain-managed-key.html) and [Customer managed key](https://docs.aws.amazon.com/codeartifact/latest/userguide/create-domain-managed-key.html) in the **AWS Key Management Service Developer Guide**.

6. Choose **Create domain**.

---

**Create a domain (AWS CLI)**

Use the `create-domain` command to create domain with the AWS CLI. You must use an AWS KMS key (KMS key) to encrypt all assets in your domain. You can use an AWS managed KMS key or a KMS key that you manage. If you use an AWS managed KMS key, do not use the `--encryption-key` parameter.

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

JSON-formatted data appears in the output with details about your new domain.

```json
{
   "domain": {
      "name": "my_domain",
      "owner": "111122223333",
      "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"
   }
}
```

If you use a KMS key that you manage, include its Amazon Resource Name (ARN) with the `--encryption-key` parameter.

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

JSON-formatted data appears in the output with details about your new domain.

```json
{
   "domain": {
      "name": "my_domain",
      "owner": "111122223333",
      "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"
   }
}
```
Create a domain with tags

To create a domain with tags, add the `--tags` parameter to your `create-domain` command.

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

Delete a domain

You can delete a domain using the CodeArtifact console or the AWS Command Line Interface (AWS CLI). You can't delete a domain that contains repositories. Before you delete the domain, you must first delete its repositories. For more information, see Delete a repository (p. 17).

Topics
- Delete a domain (console) (p. 75)
- Delete a domain (AWS CLI) (p. 75)

Delete a domain (console)

2. In the navigation pane, choose Domains, then choose the domain that you want to delete.
3. Choose Delete.

Delete a domain (AWS CLI)

Use the `delete-domain` command to delete a domain.

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

JSON-formatted data appears in the output with details about the deleted domain.

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

Domain policies

CodeArtifact supports using resource-based permissions to control access. Resource-based permissions let you specify who has access to a resource and which actions they can perform on it. By default, only
the AWS account that owns the domain can create and access repositories in the domain. You can apply a policy document to a domain to allow other IAM principals to access it.

For more information, see Policies and Permissions and Identity-Based Policies and Resource-Based Policies.

Topics
• Enable cross-account access to a domain (p. 76)
• Domain policy example (p. 77)
• Domain policy example with AWS Organizations (p. 77)
• Set a domain policy (p. 78)
• Read a domain policy (p. 78)
• Delete a domain policy (p. 79)

Enable cross-account access to a domain

A resource policy is a text file in JSON format. The file must specify a principal (actor), one or more actions, and an effect (Allow or Deny). To create a repository in a domain owned by another account, the principal must be granted the CreateRepository permission on the domain resource.

For example, the following resource policy grants the account 123456789012 permission to create a repository in the domain.

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

To allow creating repositories with tags, you must include the codeartifact:TagResource permission. This will also give the account access to add tags to the domain and all repositories in it.

Because the policy is evaluated only for operations against the domain it's attached to, you do not need to specify a resource. Because the resource is implied, the Resource can be set to *.

To access packages in a domain owned by another account, a principal must be granted the GetAuthorizationToken permission on the domain resource. This allows the domain owner to exercise control over which accounts can read the contents of repositories in the domain.

For example, the following resource policy grants the account 123456789012 permission to retrieve an auth token for any repository in the domain.

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

When multiple accounts are using a domain, the accounts should be granted a basic set of permissions to allow full use of the domain. The following resource policy lists a set of permissions that allow full use of the domain.

```json
{
    "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**
You don't need to create a domain policy if a domain and all its repositories are owned by a single account and only need to be used from that account.

Domain policy example with AWS Organizations

You can use the `aws:PrincipalOrgID` condition key to grant access to an CodeArtifact domain from all accounts in your organization, as follows.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "BasicDomainPolicy",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            }
        }
    ]
}
```
Set a domain policy

You can use the `put-domain-permissions-policy` command to attach a policy to a domain.

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

When you call `put-domains-permissions-policy`, the resource policy on the domain is ignored when evaluating permissions. This ensures that the owner of a domain cannot lock themselves out of the domain, which would prevent them from being able to update the resource policy.

**Note**

You cannot grant permissions to another AWS account to update the resource policy on a domain using a resource policy, since the resource policy is ignored when calling `put-domain-permissions-policy`.

Sample output:

```
{
  "policy": {
    "document": "{ ...policy document content... }",
    "revision": "MQlyyTqRASRU3HB58gBtSNDX27Q3hvxxxxxxxxx=",
  }
}
```

The output of the command contains the Amazon Resource Name (ARN) of the domain resource, the full contents of the policy document, and a revision identifier. The revision identifier can be passed to `put-domain-permissions-policy` using the `--policy-revision` option. This ensures that a known revision of the document is being overwritten, and not a newer version set by another writer.

Read a domain policy

To read an existing version of a policy document, use the `get-domain-permissions-policy` command. To format the output for readability, use the `--output` and `--query policy.document` together with the Python `json` tool module, as follows.
Delete a domain policy

Use the `delete-domain-permissions-policy` command to delete a policy from a domain.

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

The format of the output is the same as that of the `get-domain-permissions-policy` and `delete-domain-permissions-policy` commands.

Tag a domain in CodeArtifact

Tags are key-value pairs associated with AWS resources. You can apply tags to your domains in CodeArtifact. For information about CodeArtifact resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging resources (p. 167).

You can use the CLI to specify tags when you create a domain. You can use the console or CLI to add or remove tags, and update the values of tags in a domain. You can add up to 50 tags to each domain.

Topics

- Tag domains (CLI) (p. 79)
- Tag domains (console) (p. 81)

Tag domains (CLI)

You can use the CLI to manage domain tags.

Topics
• Add tags to a domain (CLI) (p. 80)
• View tags for a domain (CLI) (p. 80)
• Edit tags for a domain (CLI) (p. 81)
• Remove tags from a domain (CLI) (p. 81)

Add tags to a domain (CLI)

You can use the console or the AWS CLI to tag domains.

To add a tag to a domain when you create it, see Create a repository (p. 14).

In these steps, we assume that you have already installed a recent version of the AWS CLI or updated to the current version. For more information, see Installing the AWS Command Line Interface.

At the terminal or command line, run the `tag-resource` command, specifying the Amazon Resource Name (ARN) of the domain where you want to add tags and the key and value of the tag you want to add.

Note
To get the ARN of the domain, run the `describe-domain` command:

```bash
aws codeartifact describe-domain --domain my_domain --query domain.arn
```

You can add more than one tag to a domain. For example, to tag a domain named `my_domain` with two tags, a tag key named `key1` with the tag value of `value1`, and a tag key named `key2` with the tag value of `value2`:

```bash
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
```

If successful, this command has no output.

View tags for a domain (CLI)

Follow these steps to use the AWS CLI to view the AWS tags for a domain. If no tags have been added, the returned list is empty.

At the terminal or command line, run the `list-tags-for-resource` command with the Amazon Resource Name (ARN) of the domain.

Note
To get the ARN of the domain, run the `describe-domain` command:

```bash
aws codeartifact describe-domain --domain my_domain --query domain.arn
```

For example, to view a list of tag keys and tag values for a domain named `my_domain` with the arn:aws:codeartifact:us-west-2:123456789012:domain/my_domain ARN value:

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

If successful, this command returns information similar to the following:

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

## Edit tags for a domain (CLI)

Follow these steps to use the AWS CLI to edit a tag for a domain. You can change the value for an existing key or add another key. You can also remove tags from a domain, as shown in the next section.

At the terminal or command line, run the `tag-resource` command, specifying the ARN of the domain where you want to update a tag and specify the tag key and tag value:

**Note**
To get the ARN of the domain, run the `describe-domain` command:

```
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
```

If successful, this command has no output.

## Remove tags from a domain (CLI)

Follow these steps to use the AWS CLI to remove a tag from a domain.

**Note**
If you delete a domain, all tag associations are removed from the deleted domain. You do not have to remove tags before you delete a domain.

At the terminal or command line, run the `untag-resource` command, specifying the ARN of the domain where you want to remove tags and the tag key of the tag you want to remove.

**Note**
To get the ARN of the domain, run the `describe-domain` command:

```
aws codeartifact describe-domain --domain my_domain --query domain.arn
```

For example, to remove multiple tags on a domain named `mydomain` with the tag keys `key1` and `key2`:

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

If successful, this command has no output. After removing tags, you can view the remaining tags on the repository using the `list-tags-for-resource` command.

## Tag domains (console)

You can use the console or the CLI to tag resources.

### Topics
- Add tags to a domain (console) (p. 82)
- View tags for a domain (console) (p. 83)
- Edit tags for a domain (console) (p. 83)
- Remove tags from a domain (console) (p. 83)
Add tags to a domain (console)

You can use the console to add tags to an existing domain.

2. On the Domains page, choose the domain that you want to add tags to.
3. Expand the Details section.
4. Under Domain tags, choose Add domain tags if there are no tags on the domain, or choose View and edit domain tags if there are.
5. Choose Add new tag.
6. In the Key and Value fields, enter the text for each tag you want to add. (The Value field is optional.) For example, in Key, enter Name. In Value, enter Test.
7. (Optional) Choose Add tag to add more rows and enter more tags.
8. Choose **Update domain**.

**View tags for a domain (console)**

You can use the console to list tags for existing domains.

2. On the **Domains** page, choose the domain where you want to view tags.
3. Expand the **Details** section.
4. Under **Domain tags**, choose **View and edit domain tags**.
   
   **Note**
   
   If there are no tags added to this domain, the console will read **Add domain tags**.

**Edit tags for a domain (console)**

You can use the console to edit tags that have been added to the domain.

2. On the **Domains** page, choose the domain where you want to update tags.
3. Expand the **Details** section.
4. Under **Domain tags**, choose **View and edit domain tags**.
   
   **Note**
   
   If there are no tags added to this domain, the console will read **Add domain tags**.
5. In the **Key** and **Value** fields, update the values in each field as needed. For example, for the **Name** key, in **Value**, change **Test** to **Prod**.
6. Choose **Update domain**.

**Remove tags from a domain (console)**

You can use the console to delete tags from domains.

2. On the **Domains** page, choose the domain where you want to remove tags.
3. Expand the **Details** section.
4. Under **Domain tags**, choose **View and edit domain tags**.
   
   **Note**
   
   If there are no tags added to this domain, the console will read **Add domain tags**.
5. Next to the key and value for each tag you want to delete, choose **Remove**.
6. Choose **Update domain**.
Using CodeArtifact with npm

These topics describe how to use npm, the Node.js package manager, with CodeArtifact.

Note
CodeArtifact supports node v4.9.1 and later and npm v5.0.0 and later.

Topics
• Configure and use npm with CodeArtifact (p. 84)
• Configure and use Yarn with CodeArtifact (p. 87)
• npm command support (p. 89)
• npm tag handling (p. 93)
• Support for npm-compatible package managers (p. 95)

Configure and use npm with CodeArtifact

After you create a repository in CodeArtifact, you can use the npm client to install and publish packages. The recommended method for configuring npm with your repository endpoint and authorization token is by using the aws codeartifact login command. You can also configure npm manually.

Contents
• Configuring npm with the login command (p. 84)
• Configuring npm without using the login command (p. 85)
• Running npm commands (p. 86)
• Verifying npm authentication and authorization (p. 86)
• Changing back to the default npm registry (p. 87)

Configuring npm with the login command

Use the aws codeartifact login command to fetch credentials for use with npm.

Note
If you are accessing a repository in a domain that you own, you don't need to include --domain-owner. For more information, see Cross-account domains (p. 73).

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

This command makes the following changes to your ~/.npmrc file:

• Adds an authorization token after fetching it from CodeArtifact using your AWS credentials.
• Sets the npm registry to the repository specified by the --repository option.
• For npm 6 and lower: Adds "always-auth=true" so the authorization token is sent for every npm command.
The default authorization period after calling `login` is 12 hours, and `login` must be called to periodically refresh the token. For more information about the authorization token created with the `login` command, see Tokens created with the `login` command (p. 156).

## Configuring npm without using the login command

You can configure npm with your CodeArtifact repository without the `aws codeartifact login` command by manually updating the npm configuration.

### To configure npm without using the login command

1. In a command line, fetch a CodeArtifact authorization token and store it in an environment variable. npm will use this token to authenticate with your CodeArtifact repository.

   **Note**
   The following command is for macOS or Linux machines. For information on configuring environment variables on a Windows machine, see Pass an auth token using an environment variable (p. 157).

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

2. Get your CodeArtifact repository's endpoint by running the following command. Your repository endpoint is used to point npm to your repository to install or publish packages.

   - Replace `my_domain` with your CodeArtifact domain name.
   - Replace `111122223333` with the AWS account ID of the owner of the domain. If you are accessing a repository in a domain that you own, you don't need to include `--domain-owner`. For more information, see Cross-account domains (p. 73).
   - Replace `my_repo` with your CodeArtifact repository name.

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

   The following URL is an example repository endpoint.

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

   **Important**
   The registry URL must end with a forward slash (/). Otherwise, you cannot connect to the repository.

3. Use the `npm config set` command to set the registry to your CodeArtifact repository. Replace the URL with the repository endpoint URL from the previous step.

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

4. Use the `npm config set` command to add your authorization token to your npm configuration.

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

   **For npm 6 or lower:** To make npm always pass the auth token to CodeArtifact, even for GET requests, set the `always-auth` configuration variable with `npm config set`. 

---

85
Running npm commands

After you configure the npm client, you can run npm commands. Assuming that a package is present in your repository or one of its upstream repositories, you can install it with `npm install`. For example, use the following to install the `lodash` package.

```
npm install lodash
```

Use the following command to publish a new npm package to a CodeArtifact repository.

```
npm publish
```

For information about how to create npm packages, see Creating Node.js Modules on the npm documentation website. For a list of npm commands supported by CodeArtifact, see npm Command Support (p. 89).

Verifying npm authentication and authorization

Invoking the `npm ping` command is a way to verify the following:

- You have correctly configured your credentials so that you can authenticate to an CodeArtifact repository.
- The authorization configuration grants you the `ReadFromRepository` permission.

The output from a successful invocation of `npm ping` looks like the following.

```
$ 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 200 https:///npm/shared/-/ping?write=true
Ping success: {}
Ping success: {}
Timing: 93ms
```

86
The 
\( \text{-d} \) option causes npm to print additional debug information, including the repository URL. This information makes it easy to confirm that npm is configured to use the repository you expect.

**Changing back to the default npm registry**

Configuring npm with CodeArtifact sets the npm registry to the specified CodeArtifact repository. You can run the following command to set the npm registry back to its default registry when you're done connecting to CodeArtifact.

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

**Configure and use Yarn with CodeArtifact**

After you create a repository, you can use the Yarn client to manage npm packages.

**Note**

Yarn 1.X reads and uses information from your npm configuration file (.npmrc), while Yarn 2.X does not. The configuration for Yarn 2.X must be defined in the .yarnrc.yml file.

**Contents**

- Configure Yarn 1.X with the aws codeartifact login command (p. 87)
- Configure Yarn 2.X with the yarn config set command (p. 88)

**Configure Yarn 1.X with the aws codeartifact login command**

For Yarn 1.X, you can configure Yarn with CodeArtifact using the `aws codeartifact login` command. The `login` command will configure your `~/.npmrc` file with your CodeArtifact repository endpoint information and credentials. With Yarn 1.X, yarn commands use the configuration information from the `~/.npmrc` file.

**To configure Yarn 1.X with the login command**

1. If you haven't done so already, configure your AWS credentials for use with the AWS CLI, as described in Getting started with CodeArtifact (p. 7).
2. To run the `aws codeartifact login` command successfully, npm must be installed. See Downloading and installing Node.js and npm in the npm documentation for installation instructions.
3. Use the `aws codeartifact login` command to fetch CodeArtifact credentials and configure your `~/.npmrc` file.

   - Replace `my_domain` with your CodeArtifact domain name.
   - Replace `111122223333` with the AWS account ID of the owner of the domain. If you are accessing a repository in a domain that you own, you don't need to include `--domain-owner`. For more information, see Cross-account domains (p. 73).
   - Replace `my_repo` with your CodeArtifact repository name.

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

The `login` command makes the following changes to your `~/.npmrc` file:
Configure Yarn 2.X with the `yarn config set` command

The following procedure details how to configure Yarn 2.X by updating your `.yarnrc.yml` configuration from the command line with the `yarn config set` command.

**To update the `.yarnrc.yml` configuration from the command line**

1. If you haven't done so already, configure your AWS credentials for use with the AWS CLI, as described in Getting started with CodeArtifact (p. 7).
2. Use the `aws codeartifact get-repository-endpoint` command to get your CodeArtifact repository's endpoint.
   - Replace `my_domain` with your CodeArtifact domain name.
   - Replace `111122223333` with the AWS account ID of the owner of the domain. If you are accessing a repository in a domain that you own, you don't need to include `--domain-owner`. For more information, see Cross-account domains (p. 73).
   - Replace `my_repo` with your CodeArtifact repository name.

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

3. Update the `npmRegistryServer` value in your `.yarnrc.yml` file with your repository endpoint.

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

   **Note**
   The following command is for macOS or Linux machines. For information on configuring environment variables on a Windows machine, see Pass an auth token using an environment variable (p. 157).

   - Replace `my_domain` with your CodeArtifact domain name.
   - Replace `111122223333` with the AWS account ID of the owner of the domain. If you are accessing a repository in a domain that you own, you don't need to include `--domain-owner`. For more information, see Cross-account domains (p. 73).
   - Replace `my_repo` with your CodeArtifact repository name.

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

5. Use the `yarn config set` command to add your CodeArtifact authentication token to your `.yarnrc.yml` file. Replace the URL in the following command with your repository endpoint URL from Step 2.

   ```sh
   yarn config set
   'npmRegistries["https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my_repo/"].npmAuthToken' "${CODEARTIFACT_TOKEN}"
   ```

6. Use the `yarn config set` command to set the value of `npmAlwaysAuth` to `true`. Replace the URL in the following command with your repository endpoint URL from Step 2.

   ```sh
   yarn config set
   'npmRegistries["https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my_repo/"].npmAlwaysAuth' "true"
   ```

After configuring, your `.yarnrc.yml` configuration file should have contents similar to the following snippet.

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

```

You can also use the `yarn config` command to check the values of `npmRegistries` and `npmRegistryServer`.

### npm command support

The following sections summarize the npm commands that are supported, by CodeArtifact repositories, in addition to specific commands that are not supported.

**Contents**
- Supported commands that interact with a repository (p. 90)
- Supported client-side commands (p. 90)
- Unsupported commands (p. 92)
Supported commands that interact with a repository

This section lists npm commands where the npm client makes one or more requests to the registry it's been configured with (for example, with `npm config set registry`). These commands have been verified to function correctly when invoked against a CodeArtifact repository.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bugs</td>
<td>Tries to guess the location of a package's bug tracker URL, and then tries to open it.</td>
</tr>
<tr>
<td>ci</td>
<td>Installs a project with a clean slate.</td>
</tr>
<tr>
<td>deprecate</td>
<td>Deprecates a version of a package.</td>
</tr>
<tr>
<td>dist-tag</td>
<td>Modifies package distribution tags.</td>
</tr>
<tr>
<td>docs</td>
<td>Tries to guess the location of a package's documentation URL, and then tries to open it using the <code>--browser</code> config parameter.</td>
</tr>
<tr>
<td>doctor</td>
<td>Runs a set of checks to ensure that your npm installation has what it needs to manage your JavaScript packages.</td>
</tr>
<tr>
<td>install</td>
<td>Installs a package.</td>
</tr>
<tr>
<td>install-ci-test</td>
<td>Installs a project with a clean slate and runs tests.</td>
</tr>
<tr>
<td></td>
<td>Alias: <code>npm ci</code>. This command runs an <code>npm ci</code> followed immediately by an <code>npm test</code>.</td>
</tr>
<tr>
<td>install-test</td>
<td>Installs package and runs tests. Runs an <code>npm install</code> followed immediately by an <code>npm test</code>.</td>
</tr>
<tr>
<td>outdated</td>
<td>Checks the configured registry to see if any installed packages are currently outdated.</td>
</tr>
<tr>
<td>ping</td>
<td>Pings the configured or given npm registry and verifies authentication.</td>
</tr>
<tr>
<td>publish</td>
<td>Publishes a package version to the registry.</td>
</tr>
<tr>
<td>update</td>
<td>Guesses the location of a package's repository URL, and then tries to open it using the <code>--browser</code> config parameter.</td>
</tr>
<tr>
<td>view</td>
<td>Displays package metadata. Can be used to print metadata properties.</td>
</tr>
</tbody>
</table>

Supported client-side commands

These commands don't require any direct interaction with a repository, so CodeArtifact does not need to do anything to support them.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>Displays the npm bin folder.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>build</td>
<td>Builds a package.</td>
</tr>
<tr>
<td>cache</td>
<td>Manipulates the packages cache.</td>
</tr>
<tr>
<td>completion</td>
<td>Enables tab completion in all npm commands.</td>
</tr>
<tr>
<td>config</td>
<td>Updates the contents of the user and global npmrc files.</td>
</tr>
<tr>
<td>dedupe</td>
<td>Searches the local package tree and attempts to simplify the structure by moving dependencies further up the tree, where they can be more effectively shared by multiple dependent packages.</td>
</tr>
<tr>
<td>edit</td>
<td>Edits an installed package. Selects a dependency in the current working directory and opens the package folder in the default editor.</td>
</tr>
<tr>
<td>explore</td>
<td>Browses an installed package. Spawns a subshell in the directory of the installed package specified. If a command is specified, then it is run in the subshell, which then immediately terminates.</td>
</tr>
<tr>
<td>help</td>
<td>Gets help on npm.</td>
</tr>
<tr>
<td>help-search</td>
<td>Searches npm help documentation.</td>
</tr>
<tr>
<td>init</td>
<td>Creates a package.json file.</td>
</tr>
<tr>
<td>link</td>
<td>Symlinks a package folder.</td>
</tr>
<tr>
<td>ls</td>
<td>Lists installed packages.</td>
</tr>
<tr>
<td>pack</td>
<td>Creates a tarball from a package.</td>
</tr>
<tr>
<td>prefix</td>
<td>Displays prefix. This is the closest parent directory to contain a package.json file unless --g is also specified.</td>
</tr>
<tr>
<td>prune</td>
<td>Removes packages that are not listed on the parent package's dependencies list.</td>
</tr>
<tr>
<td>rebuild</td>
<td>Runs the npm build command on the matched folders.</td>
</tr>
<tr>
<td>restart</td>
<td>Runs a package's stop, restart, and start scripts and associated pre- and post- scripts.</td>
</tr>
<tr>
<td>root</td>
<td>Prints the effective node_modules folder to standard out.</td>
</tr>
<tr>
<td>run-script</td>
<td>Runs arbitrary package scripts.</td>
</tr>
<tr>
<td>shrinkwrap</td>
<td>Locks down dependency versions for publication.</td>
</tr>
<tr>
<td>uninstall</td>
<td>Uninstalls a package.</td>
</tr>
</tbody>
</table>
## Unsupported commands

These npm commands are not supported by CodeArtifact repositories.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>Sets the access level on published packages.</td>
<td>CodeArtifact uses a permission model that is different from the public npmjs repository.</td>
</tr>
<tr>
<td>adduser</td>
<td>Adds a registry user account</td>
<td>CodeArtifact uses a user model that is different from the public npmjs repository.</td>
</tr>
<tr>
<td>audit</td>
<td>Runs a security audit.</td>
<td>CodeArtifact does not currently vend security vulnerability data.</td>
</tr>
<tr>
<td>hook</td>
<td>Manages npm hooks, including adding, removing, listing, and updating.</td>
<td>CodeArtifact does not currently support any kind of change notification mechanism.</td>
</tr>
<tr>
<td>login</td>
<td>Authenticates a user. This is an alias for npm adduser.</td>
<td>CodeArtifact uses an authentication model that is different from the public npmjs repository. For information, see Authentication with npm (p. 84).</td>
</tr>
<tr>
<td>logout</td>
<td>Signs out of the registry.</td>
<td>CodeArtifact uses an authentication model that is different from the public npmjs repository. There is no way to sign out from a CodeArtifact repository, but authentication tokens expire after their configurable expiration time. The default token duration is 12 hours.</td>
</tr>
<tr>
<td>owner</td>
<td>Manages package owners.</td>
<td>CodeArtifact uses a permissions model that is different from the public npmjs repository.</td>
</tr>
<tr>
<td>profile</td>
<td>Changes settings on your registry profile.</td>
<td>CodeArtifact uses a user model that is different from the public npmjs repository.</td>
</tr>
<tr>
<td>search</td>
<td>Searches the registry for packages matching the search terms.</td>
<td>CodeArtifact supports limited search functionality with the list-packages (p. 49) command.</td>
</tr>
<tr>
<td>star</td>
<td>Marks your favorite packages.</td>
<td>CodeArtifact currently does not support any kind of favorites mechanism.</td>
</tr>
<tr>
<td>stars</td>
<td>Views packages marked as favorites.</td>
<td>CodeArtifact currently does not support any kind of favorites mechanism.</td>
</tr>
</tbody>
</table>
npm tag handling

npm registries support tags, which are string aliases for package versions. You can use tags to provide an alias instead of version numbers. For example, you might have a project with multiple streams of development and use a different tag (for example, stable, beta, dev, canary) for each stream. For more information, see dist-tag on the npm website.

By default, npm uses the latest tag to identify the current version of a package. `npm install pkg` (without @version or @tag specifier) installs the latest tag. Typically, projects use the latest tag for stable release versions only. Other tags are used for unstable or prerelease versions.

Edit tags with the npm client

The three npm dist-tag commands (add, rm, and ls) function identically in CodeArtifact repositories as they do in the default npm registry.

npm tags and the CopyPackageVersions API

When you use the CopyPackageVersions API to copy an npm package version, all tags aliasing that version are copied to the destination repository. When a version that is being copied has a tag that is also present in the destination, the copy operation sets the tag value in the destination repository to match the value in the source repository.

For example, say both repository S and repository D contain a single version of the web-helper package with the latest tag set as shown in this table.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Package name</th>
<th>Package tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>web-helper</td>
<td>latest (alias for version 1.0.1)</td>
</tr>
</tbody>
</table>
CopyPackageVersions is invoked to copy `web-helper` 1.0.1 from S to D. After the operation is complete, the `latest` tag on `web-helper` in repository D aliases 1.0.1, not 1.0.0.

If you need to change tags after copying, use the `npm dist-tag` command to modify tags directly in the destination repository. For more information about the CopyPackageVersions API, see Copying Packages Between Repositories (p. 58).

### npm tags and upstream repositories

When npm requests the tags for a package and versions of that package are also present in an upstream repository, CodeArtifact merges the tags before returning them to the client. For example, a repository named R has an upstream repository named U. The following table shows the tags for a package named `web-helper` that's present in both repositories.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Package name</th>
<th>Package tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td><code>web-helper</code></td>
<td><code>latest</code> (alias for 1.0.0)</td>
</tr>
<tr>
<td>U</td>
<td><code>web-helper</code></td>
<td><code>alpha</code> (alias for 1.0.1)</td>
</tr>
</tbody>
</table>

In this case, when the npm client fetches the tags for the `web-helper` package from repository R, it receives both the `latest` and `alpha` tags. The versions the tags point to won’t change.

When the same tag is present on the same package in both the upstream and downstream repository, CodeArtifact uses the tag that is present in the upstream repository. For example, suppose that the tags on `webhelper` have been modified to look like the following.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Package name</th>
<th>Package tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td><code>web-helper</code></td>
<td><code>latest</code> (alias for 1.0.0)</td>
</tr>
<tr>
<td>U</td>
<td><code>web-helper</code></td>
<td><code>latest</code> (alias for 1.0.1)</td>
</tr>
</tbody>
</table>

In this case, when the npm client fetches the tags for package `web-helper` from repository R, the `latest` tag will alias the version `1.0.1` because that's what's in the upstream repository. This makes it easy to consume new package versions in an upstream repository that are not yet present in a downstream repository by running `npm update`.

Using the tag in the upstream repository can be problematic when publishing new versions of a package in a downstream repository. For example, say that the latest tag on the package `web-helper` is the same in both R and U.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Package name</th>
<th>Package tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td><code>web-helper</code></td>
<td><code>latest</code> (alias for 1.0.1)</td>
</tr>
<tr>
<td>U</td>
<td><code>web-helper</code></td>
<td><code>latest</code> (alias for 1.0.1)</td>
</tr>
</tbody>
</table>

When version 1.0.2 is published to R, npm updates the `latest` tag to 1.0.2.
Support for npm-compatible package managers

These other package managers are compatible with CodeArtifact and work with the npm package format and npm wire protocol:

- **pnpm package manager.** The latest version confirmed to work with CodeArtifact is 3.3.4, which was released on May 18, 2019.
- **Yarn package manager.** The latest version confirmed to work with CodeArtifact is 1.21.1, which was released on December 11, 2019.
Using CodeArtifact with Python

These topics describe how to use pip, the Python package manager, and twine, the Python package publishing utility, with CodeArtifact.

Topics
- Configure clients with the login command (p. 96)
- Configure pip without the login command (p. 97)
- Configure twine without the login command (p. 97)
- Run pip (p. 98)
- Run twine (p. 98)
- Python compatibility (p. 98)

Configure clients with the login command

After you create a repository, you can use the pip client to install packages and the twine client to publish packages.

First, configure your AWS credentials for use with the AWS CLI, as described in Getting started with CodeArtifact (p. 7). Then, use the CodeArtifact login command to fetch credentials for use with pip or twine.

Note
If you are accessing a repository in a domain that you own, you don’t need to include --domain-owner. For more information, see Cross-account domains (p. 73).

To configure pip, run the following command.

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

To configure twine, run the following command.

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

login fetches an authorization token from CodeArtifact using your AWS credentials.

Depending on the value of the --tool option, the login command will:

- Configure pip for use with CodeArtifact by editing ~/.config/pip/pip.conf to set the index-url to the repository specified by the --repository option.
- Configure twine for use with CodeArtifact by editing ~/.pypirc to create an index-server section for the repository specified by the --repository option.

The default authorization period after calling login is 12 hours, and login must be called to periodically refresh the token. For more information about the authorization token created with the login command, see Tokens created with the login command (p. 156).
Configure pip without the login command

If you cannot use the login command to configure pip, you can use pip config.

1. Use the AWS CLI to fetch a new authorization token.
   
   **Note**
   
   If you are accessing a repository in a domain that you own, you do not need to include the --domain-owner. For more information, see Cross-account domains (p. 73).
   
   ```
   CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --domain my_domain --domain-owner 111122223333 --query authorizationToken --output text`
   ```

2. Use pip config to set the CodeArtifact registry URL and credentials.

   ```
   ```

   **Important**
   
   The registry URL must end with a forward slash (/). Otherwise, you cannot connect to the repository.

   **Example pip configuration file**

   The following is an example of a `pip.conf` file after setting the CodeArtifact registry URL and credentials.

   ```
   [global]
   ```

Configure twine without the login command

If you cannot use the login command to configure twine, you can use the `~/.pypirc` file or environment variables. To use the `~/.pypirc` file, add the following entries to it. The password must be an auth token acquired by the `get-authorization-token` API.

```
[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
```

To use environment variables, do the following.

**Note**

If you are accessing a repository in a domain that you own, you do not need to include the --domain-owner. For more information, see Cross-account domains (p. 73).

```
export TWINE_USERNAME=aws
```
To run pip commands, you must configure pip with CodeArtifact. For more information, see Configure clients with the login command (p. 96).

Assuming that a package is present in your repository or one of its upstream repositories, you can install it with `pip install`. For example, use the following command to install the `requests` package.

```bash	pip install requests
```

Use the `-i` option to temporarily revert to installing packages from `https://pypi.org` instead of your CodeArtifact repository.

```bash	pip install -i https://pypi.org/simple requests
```

Before using `twine` to publish Python package assets, you must first configure CodeArtifact permissions and resources.

1. Follow the steps in the Setting up with AWS CodeArtifact (p. 4) section to configure your AWS account, tools, and permissions.
2. Configure `twine` by following the steps in Configure clients with the login command (p. 96).

After you configure `twine`, you can run `twine` commands. Use the following command to publish Python package assets.

```bash
twine upload --repository codeartifact mypackage-1.0.tgz
```

For information about how to build and package your Python application, see Generating Distribution Archives on the Python Packaging Authority website.

CodeArtifact supports PyPI's Legacy APIs, except the `simple` API. CodeArtifact does not support PyPI's XML-RPC or JSON APIs.

For more information, see the following on the Python Packaging Authority's GitHub repository.

- Legacy API
- XML-RPC API
- JSON API
pip command support

The following sections summarize the pip commands that are supported, by CodeArtifact repositories, in addition to specific commands that are not supported.

Topics
- Supported commands that interact with a repository (p. 99)
- Supported client-side commands (p. 99)

Supported commands that interact with a repository

This section lists pip commands where the pip client makes one or more requests to the registry it’s been configured with. These commands have been verified to function correctly when invoked against a CodeArtifact repository.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>install</td>
<td>Install packages.</td>
</tr>
<tr>
<td>download</td>
<td>Download packages.</td>
</tr>
</tbody>
</table>

CodeArtifact does not implement pip search. If you have configured pip with a CodeArtifact repository, running pip search will search and show packages from PyPI.

Supported client-side commands

These commands don't require any direct interaction with a repository, so CodeArtifact does not need to do anything to support them.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uninstall</td>
<td>Uninstall packages.</td>
</tr>
<tr>
<td>freeze</td>
<td>Output installed packages in requirements format.</td>
</tr>
<tr>
<td>list</td>
<td>List installed packages.</td>
</tr>
<tr>
<td>show</td>
<td>Show information about installed packages.</td>
</tr>
<tr>
<td>check</td>
<td>Verify installed packages have compatible dependencies.</td>
</tr>
<tr>
<td>config</td>
<td>Manage local and global configuration.</td>
</tr>
<tr>
<td>wheel</td>
<td>Build wheels from your requirements.</td>
</tr>
<tr>
<td>hash</td>
<td>Compute hashes of package archives.</td>
</tr>
<tr>
<td>completion</td>
<td>Helps with command completion.</td>
</tr>
<tr>
<td>debug</td>
<td>Show information useful for debugging.</td>
</tr>
<tr>
<td>help</td>
<td>Show help for commands.</td>
</tr>
</tbody>
</table>
Using CodeArtifact with Maven

The Maven repository format is used by many different languages, including Java, Kotlin, Scala, and Clojure. It's supported by many different build tools, including Maven, Gradle, Scala SBT, Apache Ivy, and Leiningen.

We have tested and confirmed compatibility with CodeArtifact for the following versions:

- Latest **Maven** version: 3.6.3.
- Latest **Gradle** version: 6.4.1. 5.5.1 has also been tested.

**Topics**

- Use CodeArtifact with Gradle (p. 100)
- Use CodeArtifact with maven (p. 105)
- Publishing with curl (p. 109)
- Using Maven checksums (p. 111)
- Use Maven snapshots (p. 111)

Use CodeArtifact with Gradle

After you have the CodeArtifact auth token in an environment variable as described in Pass an auth token using an environment variable (p. 157), follow these instructions to consume Maven packages from, and publish new packages to, a CodeArtifact repository.

**Topics**

- Fetch dependencies (p. 100)
- Fetch plugins (p. 101)
- Publish artifacts (p. 101)
- Run a Gradle build in IntelliJ IDEA (p. 103)

Fetch dependencies

To fetch dependencies from CodeArtifact in a Gradle build, add a maven section to the repositories section in the project build.gradle file.

```gradle
maven {
    url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/maven/
    my_repo/'
    credentials {
        username "aws"
        password System.env.CODEARTIFACT_AUTH_TOKEN
    }
}
```

The url in the sample above is your CodeArtifact repository's endpoint. Gradle uses the endpoint to connect to your repository. In the sample, my_domain is the name of your domain, 111122223333 is the ID of the owner of the domain, and my_repo is the name of your repository. You can retrieve a repository's endpoint by using the get-repository-endpoint AWS CLI command.
For example, with a repository named `my_repo` inside a domain named `my_domain`, the command is as follows:

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

The `get-repository-endpoint` command will return the repository endpoint:

```plaintext
url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/maven/my_repo/
```

To use the CodeArtifact repository as the only source for your project dependencies, remove any other sections in `repositories` from `build.gradle`. If you have more than one repository, Gradle searches each repository for dependencies in the order they are listed.

After you configure the repository, you can add project dependencies to the `dependencies` section with standard Gradle syntax.

```gradle
dependencies {
    implementation 'com.google.guava:guava:27.1-jre'
    implementation 'commons-cli:commons-cli:1.4'
    testImplementation 'org.testng:testng:6.14.3'
}
```

### Fetch plugins

By default Gradle will resolve plugins from the public Gradle Plugin Portal. To pull plugins from a CodeArtifact repository, add a `pluginManagement` block to your `settings.gradle` file. The `pluginManagement` block must appear before any other statements in `settings.gradle`:

```gradle
pluginManagement {
    repositories {
        maven {
            name 'my_repo'
            url 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/maven/my_repo/
            credentials {
                username 'aws'
                password System.env.CODEARTIFACT_AUTH_TOKEN
            }
        }
    }
}
```

This will ensure that Gradle resolves plugins from the specified repository. The repository must have an upstream repository with an external connection to the Gradle Plugin Portal (e.g. `gradle-plugins-store`) so that commonly-required Gradle plugins are available to the build. For more information, see the [Gradle documentation](https://docs.gradle.org/).

### Publish artifacts

This section describes how to publish a Java library built with Gradle to a CodeArtifact repository.

First, add the `maven-publish` plugin to the `plugins` section of the project's `build.gradle` file.

```gradle
plugins {
    id 'java-library'
}
```
id 'maven-publish'
}

Next, add a publishing section to the project build.gradle file.

```
publishing {
  publications {
    mavenJava(MavenPublication) {
      groupId = 'group-id'
      artifactId = 'artifact-id'
      version = 'version'
      from components.java
    }
  }
  repositories {
    maven {
      url 'https://my_domain-11122223333.d.codeartifact.region.amazonaws.com/maven/my_repo/'
      credentials {
        username "aws"
        password System.env.CODEARTIFACT_AUTH_TOKEN
      }
    }
  }
}
```

The maven-publish plugin generates a POM file based on the groupId, artifactId, and version specified in the publishing section.

After these changes to build.gradle are complete, run the following command to build the project and upload it to the repository.

```
./gradlew publish
```

Use list-package-versions to check that the package was successfully published.

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 11122223333 --repository my_repo --format maven
--namespace com.company.framework --package my-package-name
```

Sample output:

```
{
  "format": "maven",
  "namespace": "com.company.framework",
  "package": "example",
  "versions": [
    {
      "version": "1.0",
      "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
      "status": "Published"
    }
  ]
}
```

For more information, see these topics on the Gradle website:

- Building Java Libraries
- Publishing a project as a module
Run a Gradle build in IntelliJ IDEA

You can run a Gradle build in IntelliJ IDEA that pulls dependencies from CodeArtifact. To authenticate with CodeArtifact, you must provide Gradle with a CodeArtifact authorization token. There are three methods to provide an auth token.

- **Method 1:** Storing the auth token in `gradle.properties`. Use this method if you are able to overwrite or add to the contents of the `gradle.properties` file.
- **Method 2:** Storing the auth token in a separate file. Use this method if you do not want to modify your `gradle.properties` file.
- **Method 3:** Generating a new auth token for each run by running `aws` as an inline script in `build.gradle`. Use this method if you want the Gradle script to fetch a new token on each run. The token won't be stored on the file system.

**Token stored in gradle.properties**

**Method 1: Storing the auth token in `gradle.properties`**

**Note**
The example shows the `gradle.properties` file located in `GRADLE_USER_HOME`.

1. Update your `build.gradle` file with the following snippet:

   ```gradle
   repositories {
     maven {
       url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/maven/my_repo/'
       credentials {
         username "aws"
         password "$codeartifactToken"
       }
     }
   }
   
   To fetch plugins from CodeArtifact, add a `pluginManagement` block to your `settings.gradle` file. The `pluginManagement` block must appear before any other statements in `settings.gradle`.

   ```gradle
   pluginManagement {
     repositories {
       maven {
         name 'my_repo'
         url 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/maven/my_repo/'
         credentials {
           username 'aws'
           password "$codeartifactToken"
         }
       }
     }
   }
   
   Fetch a CodeArtifact auth token:

   ```bash
   export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --domain my_domain --domain-owner 111122223333 --query authorizationToken --output text --profile profile-name`
   ```
4. Write the auth token into the `gradle.properties` file:

```
> echo "codeartifactToken=$CODEARTIFACT_AUTH_TOKEN" > ~/.gradle/gradle.properties
```

Token stored in separate file

**Method 2: Storing the auth token in a separate file**

1. Update your `build.gradle` file with the following snippet:

```java
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. To fetch plugins from CodeArtifact, add a `pluginManagement` block to your `settings.gradle` file. The `pluginManagement` block must appear before any other statements in `settings.gradle`.

```gradle
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. Fetch a CodeArtifact auth token:

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

4. Write the auth token into the file that was specified in your `build.gradle` file:

```
> echo "codeartifactToken=$CODEARTIFACT_AUTH_TOKEN" > file
```
Token generated for each run in build.gradle

**Method 3: Generating a new auth token for each run by running aws as an inline script in build.gradle**

1. Update your build.gradle file with the following snippet:

   ```groovy
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. To fetch plugins from CodeArtifact, add a pluginManagement block to your settings.gradle file. The pluginManagement block must appear before any other statements in settings.gradle.

   ```groovy
   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
         }
       }
     }
   }
   ```

---

**Use CodeArtifact with mvn**

You use the `mvn` command to execute Maven builds. This section shows how to configure `mvn` to use a CodeArtifact repository.

After you have the CodeArtifact auth token in an environment variable as described in Passing an Auth Token Using an Environment Variable (p. 157), follow these instructions to consume Maven packages from, and publish new packages to, a CodeArtifact repository.

**Topics**

- Fetch dependencies (p. 100)
- Publish artifacts (p. 101)
- Publish third-party artifacts (p. 108)
Fetch dependencies

To configure `mvn` to fetch dependencies from a CodeArtifact repository, you must edit the Maven configuration file, `settings.xml`, and optionally, your project's POM.

1. In `settings.xml` (typically found at `~/.m2/settings.xml`), add a `<servers>` section with a reference to the `CODEARTIFACT_AUTH_TOKEN` environment variable so that Maven passes the token in HTTP requests.

   ```xml
   <settings>
     ...<servers>
       <server>
         <id>codeartifact</id>
         <username>aws</username>
         <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
       </server>
     </servers>
     ...</settings>
   ```

2. Add the URL endpoint for your CodeArtifact repository in a `<repository>` element. You can do this in `settings.xml` or your project's POM file.

   You can retrieve your repository's endpoint by using the `get-repository-endpoint` AWS CLI command.

   For example, with a repository named `my_repo` inside a domain named `my_domain`, the command is as follows:

   ```bash
   aws codeartifact get-repository-endpoint --domain my_domain --repository my_repo --format maven
   ```

   The `get-repository-endpoint` command will return the repository endpoint:

   ```text
   url 'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/maven/my_repo/'
   ````

   Add the repository endpoint to `settings.xml` as follows.

   ```xml
   <settings>
     ...<profiles>
       <profile>
         <id>default</id>
         <repositories>
           <repository>
             <id>codeartifact</id>
           </repository>
         </repositories>
       </profile>
     </profiles>
   </settings>
```
Publish artifacts

To publish a Maven artifact with `mvn` to a CodeArtifact repository, you must also edit `~/.m2/settings.xml` and the project POM.

1. Add a `<servers>` section to `settings.xml` with a reference to the `CODEARTIFACT_AUTH_TOKEN` environment variable so that Maven passes the token in HTTP requests.

```
<settings>
  ...
  <servers>
    <server>
      <!-- CODEARTIFACT_AUTH_TOKEN environment variable -->
    </server>
  </servers>
</settings>
```
Publish third-party artifacts

You can publish third-party Maven artifacts to a CodeArtifact repository with `mvn deploy:deploy-file`. This can be helpful to users that want to publish artifacts and only have JAR files and don't have access to package source code or POM files.

The `mvn deploy:deploy-file` command will generate a POM file based on the information passed in the command line.
Publish third-party Maven artifacts

1. Create a ~/.m2/settings.xml file with the following contents:

```xml
<settings>
  <servers>
    <server>
      <id>codeartifact</id>
      <username>aws</username>
      <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
    </server>
  </servers>
</settings>
```

2. Fetch a CodeArtifact authorization token:

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

3. Run the mvn deploy:deploy-file command:

```bash
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**
The example above publishes commons-cli 1.4. Modify the groupId, artifactId, version, and file arguments to publish a different JAR.

These instructions are based on examples in the Guide to deploying 3rd party JARs to remote repository from the Apache Maven documentation.

For more information, see these topics on the Apache Maven Project website:

- Setting up Multiple Repositories
- Settings Reference
- Distribution Management
- Profiles

Publishing with curl

This section shows how to use the HTTP client curl to publish Maven artifacts to a CodeArtifact repository. Publishing artifacts with curl can be useful if you do not have or want to install the Maven client in your environments.
Publish a Maven artifact with curl

1. Fetch a CodeArtifact authorization token by following the steps in Pass an auth token using an environment variable (p. 157) and return to these steps.

2. Use the following curl command to publish the JAR to a CodeArtifact repository:

   ```
   curl --request PUT https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/maven/my_repo/app/my-app/1.0/my-app-1.0.jar 
   --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/octet-stream" 
   --data-binary @target/my-app-1.0.jar
   ```

   In the sample above, my_domain is the name of your domain, 111122223333 is the ID of AWS account that owns the domain, and my_repo is the name of your repository.

3. Use the following curl command to publish the POM to a CodeArtifact repository:

   ```
   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 @target/my-app-1.0.pom
   ```

4. At this point, the Maven artifact will be in your CodeArtifact repository with a status of Unfinished. To be able to consume the package, it must be in the Published state. You can move the package from Unfinished to Published by either uploading a maven-metadata.xml file to your package, or calling the UpdatePackageVersionsStatus API to change the status.

   a. Option 1: Use the following curl command to add a maven-metadata.xml file to your package:

   ```
   --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/octet-stream" 
   --data-binary @target/maven-metadata.xml
   ```

   The following is an example of the contents of a maven-metadata.xml file:

   ```
   <!DOCTYPE metadata [ ]> 
   <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> 
   </versioning> 
   <lastUpdated>20200731090423</lastUpdated> 
   </metadata>
   ```

   b. Option 2: Update the package status to Published with the UpdatePackageVersionsStatus API.

   ```
   aws codeartifact update-package-versions-status \
   --domain my_domain \
   --domain-owner 111122223333 \ 
   --repository my_repo \
   ```
Using Maven checksums

When a Maven artifact is published to an AWS CodeArtifact repository, the checksum associated with each asset or file in the package is used to validate the upload. Examples of assets are *jar*, *pom*, and *war* files. For each asset, the Maven artifact contains multiple checksum files that use the asset name with an additional extension, such as *md5* or *sha1*. For example, the checksum files for a file named *my-maven-package.jar* might be *my-maven-package.jar.md5* and *my-maven-package.jar.sha1*.

**Note**

Maven uses the term *artifact*. In this guide, a Maven package is the same as a Maven artifact. For more information, see [AWS CodeArtifact package](#). Every Maven artifact also contains a *maven-metadata.xml* file. This file must be uploaded for a publish to succeed. If a checksum mismatch is detected during the upload of any artifact file, the publish stops. This might prevent the *maven-metadata.xml* from being uploaded. When that happens, the status of the Maven artifact is set to *Unfinished*. You cannot download assets that are part of a package with this status.

Keep the following in mind in the event of a checksum mismatch when you publish a Maven package:

- If the checksum mismatch occurs before *maven-metadata.xml* is uploaded, the status of the package is not set to *Unfinished*. The package is not visible and its assets cannot be consumed. When this happens, try one of the following, and then try to download the asset again.
  - Run the command that publishes the Maven artifact again. This might work if a network issue corrupted the checksum file during download. If the network issue is resolved for the retry, the checksum matches and the download is successful.
  - If republishing the Maven artifact doesn’t work, delete the package and then republish it. For more information, see [DeletePackageVersions](#) in the [AWS CodeArtifact API Reference](#).
- If the checksum mismatch occurs after *maven-metadata.xml* is uploaded, the status of the package is set to *Published*. You can consume any asset from the package, including those with checksum mismatches. When you download an asset, the checksum generated by AWS CodeArtifact is downloaded with it. If the downloaded file is associated with a checksum mismatch, its downloaded checksum file might not match the checksum that was uploaded when the package was published.

**Use Maven snapshots**

A Maven *snapshot* is a special version of a Maven package that refers to the latest production branch code. It is a development version that precedes the final release version. You can identify a snapshot version of a Maven package by the suffix *SNAPSHOT* that is appended to the package version. For example, the snapshot of version *1.1* is *1.1-SNAPSHOT*. For more information, see [What is a SNAPSHOT version?](#) on the Apache Maven Project website.

AWS CodeArtifact supports publishing and consuming Maven snapshots. You can publish a Maven snapshot to an AWS CodeArtifact repository or, if you are directly connected, to an upstream...
repository. However, a snapshot version in both a repository and one of its upstream repositories is not supported. For example, if you upload a Maven package with version 1.2-SNAPSHOT to your repository, CodeArtifact does not support uploading a Maven package with the same snapshot version to one of its upstream repositories. This scenario might return unpredictable results.

When a Maven snapshot is published, its previous version is preserved in a new version called a build. Each time a Maven snapshot is published, a new build version is created. All previous versions of a snapshot are maintained in its build versions. When a Maven snapshot is published, its status is set to Published and the status of the build that contains the previous version is set to Unlisted.

If you request a snapshot, the version with status Published is returned. This is always the most recent version of the Maven snapshot. You can also request a particular build of a snapshot. To see the build versions of a Maven snapshot, use ListPackageVersions and set the status parameter to Unlisted.

To delete all build versions of a Maven snapshot, use DeletePackageVersions and set the expectedStatus parameter to Unlisted.
Using CodeArtifact with NuGet

These topics describe how to consume and publish NuGet packages using CodeArtifact.

Note
AWS CodeArtifact only supports NuGet.exe version 4.8 and higher.

Topics
• Use CodeArtifact with Visual Studio (p. 113)
• Use CodeArtifact with the nuget or dotnet CLI (p. 114)
• NuGet compatibility (p. 120)

Use CodeArtifact with Visual Studio

You can consume packages from CodeArtifact directly in Visual Studio with the CodeArtifact Credential Provider. The credential provider simplifies the setup and authentication of your CodeArtifact repositories in Visual Studio and is available in the AWS Toolkit for Visual Studio.

Note
The AWS Toolkit for Visual Studio is not available for Visual Studio for Mac.

To configure and use NuGet with CLI tools, see Use CodeArtifact with the nuget or dotnet CLI (p. 114).

Topics
• Configure Visual Studio with the CodeArtifact Credential Provider (p. 113)
• Use the Visual Studio Package Manager console (p. 114)

Configure Visual Studio with the CodeArtifact Credential Provider

The CodeArtifact Credential Provider simplifies the setup and continued authentication between CodeArtifact and Visual Studio. CodeArtifact authentication tokens are valid for a maximum of 12 hours. To avoid having to manually refresh the token while working in Visual Studio, the credential provider periodically fetches a new token before the current token expires.

Important
To use the credential provider, ensure that any existing AWS CodeArtifact credentials are cleared from your nuget.config file that may have been added manually or by running aws codeartifact login to configure NuGet previously.

Use CodeArtifact in Visual Studio with the AWS Toolkit for Visual Studio

1. Install the AWS Toolkit for Visual Studio using the following steps. The toolkit is compatible with Visual Studio 2017 and 2019 using these steps. AWS CodeArtifact does not support Visual Studio 2015 and earlier.
Use the Visual Studio Package Manager console

The Visual Studio Package Manager console will not use the Visual Studio version of the CodeArtifact Credential Provider. To use it, you will have to configure the command-line credential provider. See Use CodeArtifact with the nuget or dotnet CLI (p. 114) for more information.

Use CodeArtifact with the nuget or dotnet CLI

You can use CLI tools like nuget and dotnet to publish and consume packages from CodeArtifact. This document provides information about configuring the CLI tools and using them to publish or consume packages.

Topics

- Configure the nuget or dotnet CLI (p. 115)
- Consume NuGet packages from CodeArtifact (p. 118)
- Publish NuGet packages to CodeArtifact (p. 118)
Configure the nuget or dotnet CLI

You can configure the nuget or dotnet CLI with a CodeArtifact Credential Provider, with the AWS CLI, or manually. Configuring NuGet with the credential provider is highly recommended for simplified setup and continued authentication.

Method 1: Configure with the CodeArtifact Credential Provider

The CodeArtifact Credential Provider simplifies the authentication and configuration of CodeArtifact with NuGet CLI tools. CodeArtifact authentication tokens are valid for a maximum of 12 hours. To avoid having to manually refresh the token while using the nuget or dotnet CLI, the credential provider periodically fetches a new token before the current token expires.

**Important**

To use the credential provider, ensure that any existing AWS CodeArtifact credentials are cleared from your `nuget.config` file that may have been added manually or by running `aws codeartifact login` to configure NuGet previously.

**Install and configure the CodeArtifact Credential Provider for NuGet**

**dotnet**

1. Download the [AWS.CodeArtifact.NuGet.CredentialProvider tool from NuGet.org](https://nuget.org) with the following `dotnet` command.

   ```bash
   dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
   ```

2. Use the `codeartifact-creds install` command to copy the credential provider to the NuGet plugins folder.

   ```bash
   dotnet codeartifact-creds install
   ```

3. (Optional): Set the AWS profile you want to use with the credential provider. If not set, the credential provider will use the default profile. For more information on AWS CLI profiles, see [Named profiles](#).

   ```bash
   dotnet codeartifact-creds configure set profile profile_name
   ```

**nuget**

Perform the following steps to use the NuGet CLI to install the CodeArtifact Credential Provider from an Amazon S3 bucket and configure it. The credential provider will use the default AWS CLI profile, for more information on profiles, see [Named profiles](#).

2. Unzip the file.
3. Copy the `AWS.CodeArtifact.NuGetCredentialProvider` folder from the `netfx` folder to `%user_profile%/.nuget/plugins/netfx/` on Windows or `~/.nuget/plugins/netfx` on Linux or MacOS.
4. Copy the `AWS.CodeArtifact.NuGetCredentialProvider` folder from the `netcore` folder to `%user_profile%/.nuget/plugins/netcore/` on Windows or `~/.nuget/plugins/netcore` on Linux or MacOS.
After you create a repository and configure the credential provider you can use the `nuget` or `dotnet` CLI tools to install and publish packages. For more information, see Consume NuGet packages from CodeArtifact (p. 118) and Publish NuGet packages to CodeArtifact (p. 118).

**Method 2: Configure nuget or dotnet with the login command**

The `codeartifact login` command in the AWS CLI adds a repository endpoint and authorization token to your NuGet configuration file enabling `nuget` or `dotnet` to connect to your CodeArtifact repository. This will modify the user-level NuGet configuration which is located at `%appdata%\NuGet\NuGet.Config` for Windows and `~/.config/NuGet/NuGet.Config` or `~/.nuget/NuGet/NuGet.Config` for Mac/Linux. For more information about NuGet configurations, see Common NuGet configurations.

Configure nuget or dotnet with the `login` command

1. Configure your AWS credentials for use with the AWS CLI, as described in Getting started with CodeArtifact (p. 7).
2. Ensure that the NuGet CLI tool (`nuget` or `dotnet`) has been properly installed and configured. For instructions, see the nuget or dotnet documentation.
3. Use the CodeArtifact `login` command to fetch credentials for use with NuGet.

   **Note**
   
   If you are accessing a repository in a domain that you own, you don't need to include `--domain-owner`. For more information, see Cross-account domains (p. 73).

   **dotnet**

   **Important**
   
   *Linux and MacOS users*: Because encryption is not supported on non-Windows platforms, your fetched credentials will be stored as plain text in your configuration file.

   ```
   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
   ```

   The login command will:

   - Fetch an authorization token from CodeArtifact using your AWS credentials.
   - Update your user-level NuGet configuration with a new entry for your NuGet package source. The source that points to your CodeArtifact repository endpoint will be called `domain_name/repo_name`.

   The default authorization period after calling `login` is 12 hours, and `login` must be called to periodically refresh the token. For more information about the authorization token created with the `login` command, see Tokens created with the `login` command (p. 156).

   After you create a repository and configure authentication you can use the `nuget`, `dotnet`, or `msbuild` CLI clients to install and publish packages. For more information, see Consume NuGet packages from CodeArtifact (p. 118) and Publish NuGet packages to CodeArtifact (p. 118).
Method 3: Configure nuget or dotnet without the login command

For manual configuration, you must add a repository endpoint and authorization token to your NuGet configuration file to enable nuget or dotnet to connect to your CodeArtifact repository.

**Manually configure nuget or dotnet to connect to your CodeArtifact repository.**

1. Determine your CodeArtifact repository endpoint by using the `get-repository-endpoint` AWS CLI command.

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

   Example output:

   ```json
   {
   "repositoryEndpoint": "https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/nuget/my_repo/"
   }
   ```

2. Get an authorization token to connect to your repository from your package manager by using the `get-authorization-token` AWS CLI command.

   ```bash
   aws codeartifact get-authorization-token --domain my_domain
   ```

   Example output:

   ```json
   {
   "authorizationToken": "eyJ2I...viOw",
   "expiration": 1601616533.0
   }
   ```

3. Configure nuget or dotnet to use the repository endpoint from Step 1 and authorization token from Step 2. Note, you will need to append `/v3/index.json` to your repository endpoint.

   **dotnet**

   ```bash
   ```

   Example output:

   ```json
   Example output:
   ```

   **nuget**

   ```bash
   ```

   Example output:
Consume NuGet packages from CodeArtifact

Once you have configured NuGet with CodeArtifact, you can consume NuGet packages that are stored in your CodeArtifact repository or one of its upstream repositories.

To consume a package version from a CodeArtifact repository or one of its upstream repositories with nuget or dotnet, run the following command replacing `packageName` with the name of the package you want to consume and `packageSourceName` with the source name for your CodeArtifact repository in your NuGet configuration file. If you used the `login` command to configure your NuGet configuration, the source name is `domain_name/repo_name`.

**dotnet**

```bash
dotnet add package packageName --source packageSourceName
```

**nuget**

```bash
nuget install packageName -Source packageSourceName
```

To install a specific version of a package

**dotnet**

```bash
dotnet add package packageName --version 1.0.0 --source packageSourceName
```

**nuget**

```bash
nuget install packageName -Version 1.0.0 -Source packageSourceName
```

See Manage packages using the nuget.exe CLI or Install and manage packages using the dotnet CLI in the Microsoft Documentation for more information.

Consume NuGet packages from NuGet.org

You can consume NuGet packages from NuGet.org through a CodeArtifact repository by configuring the repository with an external connection to NuGet.org. Packages consumed from NuGet.org are ingested and stored in your CodeArtifact repository. For more information about adding external connections, see Add an external connection (p. 26).

Publish NuGet packages to CodeArtifact

Once you have configured NuGet with CodeArtifact, you can use nuget or dotnet to publish package versions to CodeArtifact repositories.

To push a package version to a CodeArtifact repository, run the following command with the full path to your `.nupkg` file and the source name for your CodeArtifact repository in your NuGet configuration file. If you used the `login` command to configure your NuGet configuration, the source name is `domain_name/repo_name`.

```bash
nuget install packageName -Source packageSourceName
```
**Note**
You can create a NuGet package if you do not have one to publish. For more information, see Package creation workflow in the Microsoft documentation.

dotnet

```bash
dotnet nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg --source packageSourceName
```

nuget

```bash
nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg -Source packageSourceName
```

# CodeArtifact Credential Provider reference

The CodeArtifact Credential Provider makes it easy to configure and authenticate NuGet with your CodeArtifact repositories.

## CodeArtifact Credential Provider commands

This section includes the list of commands for the CodeArtifact Credential Provider. These commands must be prefixed with `dotnet codeartifact-creds` like the following example.

```bash
dotnet codeartifact-creds command
```

- `configure set profile profile`: Configures the credential provider to use the provided AWS profile.
- `configure unset profile`: Removes the configured profile if set.
- `install`: Copies the credential provider to the plugins folder.
- `install --profile profile`: Copies the credential provider to the plugins folder and configures it to use the provided AWS profile.
- `uninstall`: Uninstalls the credential provider. This does not remove the changes to the configuration file.
- `uninstall --delete-configuration`: Uninstalls the credential provider and removes all changes to the configuration file.

## CodeArtifact Credential Provider logs

To enable logging for the CodeArtifact Credential Provider, you must set the log file in your environment. The credential provider logs contain helpful debugging information such as:

- The AWS profile used to make connections
- Any authentication errors
- If the endpoint provided is not a CodeArtifact URL

### Set the CodeArtifact Credential Provider log file

```bash
export AWS_CODEARTIFACT_NUGET_LOGFILE=/path/to/file
```

After the log file is set, any `codeartifact-creds` command will append its log output to the contents of that file.
NuGet compatibility

This guide contains information about CodeArtifact's compatibility with different NuGet tools and versions.

Topics
- General NuGet compatibility (p. 120)
- NuGet command line support (p. 120)

General NuGet compatibility

AWS CodeArtifact supports NuGet 4.8 and higher.

AWS CodeArtifact only supports V3 of the NuGet HTTP protocol. This means that some CLI commands that rely V2 of the protocol are not supported. See the nuget.exe command support (p. 120) section for more information.

AWS CodeArtifact does not support PowerShellGet 2.x.

NuGet command line support

AWS CodeArtifact supports the NuGet (nuget.exe) and .NET Core (dotnet) CLI tools.

nuget.exe command support

Because CodeArtifact only supports V3 of NuGet's HTTP protocol, the following commands will not work when used against CodeArtifact resources:

- list: The nuget list command displays a list of packages from a given source. To get a list of packages in a CodeArtifact repository, you can use the List package names (p. 49) command from the AWS CLI.
Using CodeArtifact with CodeBuild

These topics describe how to use packages in a CodeArtifact repository in an AWS CodeBuild build project.

Topics
- Using npm packages in CodeBuild (p. 121)
- Using Python packages in CodeBuild (p. 122)
- Using Maven packages in CodeBuild (p. 124)
- Using NuGet packages in CodeBuild (p. 126)
- Dependency caching (p. 130)

Using npm packages in CodeBuild

The following steps have been tested with the operating systems listed in Docker images provided by CodeBuild.

Set up permissions with IAM roles

These steps are required when using npm packages from CodeArtifact in CodeBuild.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles. On the Roles page, edit the role used by your CodeBuild build project. This role must have the following permissions.

```
{
    "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**

If you also want to use CodeBuild to publish packages, add the `codeartifact:PublishPackageVersion` permission.
Log in and use npm

To use npm packages from CodeBuild, run the `login` command from the `pre-build` section of your project's `buildspec.yaml` to configure npm to fetch packages from CodeArtifact. For more information, see Authentication with npm (p. 84).

After `login` has run successfully, you can run npm commands from the `build` section to install npm packages.

**Linux**

**Note**
It is only necessary to upgrade the AWS CLI with `pip3 install awscli --upgrade --user` if you are using an older CodeBuild image. If you are using the latest image versions, you can remove that line.

```bash
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**

```bash
version: 0.2
phases:
  install:
    commands:
      - 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
```

Using Python packages in CodeBuild

The following steps have been tested with the operating systems listed in the Docker images provided by CodeBuild.

**Set up permissions with IAM roles**

These steps are required when using Python packages from CodeArtifact in CodeBuild.
1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose Roles. On the Roles page, edit the role used by your CodeBuild build project. This role must have the following permissions.

```json
{
   "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**

If you also want to use CodeBuild to publish packages, add the `codeartifact:PublishPackageVersion` permission.

For information, see Modifying a Role in the IAM User Guide.

---

## Log in and use pip or twine

To use Python packages from CodeBuild, run the `login` command from the `pre-build` section of your project's `buildspec.yaml` file to configure `pip` to fetch packages from CodeArtifact. For more information, see Using CodeArtifact with Python (p. 96).

After `login` has run successfully, you can run `pip` commands from the `build` section to install or publish Python packages.

### Linux

**Note**

It is only necessary to upgrade the AWS CLI with `pip3 install awscli --upgrade --user` if you are using an older CodeBuild image. If you are using the latest image versions, you can remove that line.

To install Python packages using `pip`:

```yaml
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool pip --domain my_domain --domain-owner 111122223333 --repository my_repo
build:
```
Using Maven packages in CodeBuild

Set up permissions with IAM roles

These steps are required when using Maven packages from CodeArtifact in CodeBuild.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**. On the **Roles** page, edit the role used by your CodeBuild build project. This role must have the following permissions.

```json
{
    "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**
If you also want to use CodeBuild to publish packages, add the `codeartifact:PublishPackageVersion` and `codeartifact:PutPackageMetadata` permissions.

For information, see Modifying a Role in the IAM User Guide.

### Use gradle or mvn

To use Maven packages with **gradle** or **mvn**, store the CodeArtifact auth token in an environment variable, as described in Pass an auth token in an environment variable (p. 157). The following is an example.

**Note**
It is only necessary to upgrade the AWS CLI with `pip3 install awscli --upgrade --user` if you are using an older CodeBuild image. If you are using the latest image versions, you can remove that line.

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --domain my_domain --domain-owner 11112223333 --query authorizationToken --output text`
```

**To use Gradle:**

If you referenced the `CODEARTIFACT_AUTH_TOKEN` variable in your Gradle `build.gradle` file as described in Using CodeArtifact with Gradle (p. 100), you can invoke your Gradle build from the `buildspec.yaml` build section.

```
build:
  commands:
```
Using NuGet packages in CodeBuild

The following steps have been tested with the operating systems listed in the Docker images provided by CodeBuild.

Topics
- Set up permissions with IAM roles (p. 126)
- Consume NuGet packages (p. 127)
- Build with NuGet packages (p. 128)
- Publish NuGet packages (p. 129)

Set up permissions with IAM roles

These steps are required when using NuGet packages from CodeArtifact in CodeBuild.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles. On the Roles page, edit the role used by your CodeBuild build project. This role must have the following permissions.

```json
{
  "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**
If you also want to use CodeBuild to publish packages, add the `codeartifact:PublishPackageVersion` permission.

For information, see Modifying a Role in the IAM User Guide.
Consume NuGet packages

To consume NuGet packages from CodeBuild, include the following in your project's buildspec.yaml file.

1. In the install section, install the CodeArtifact Credential Provider to configure command line tools such as msbuild and dotnet to build and publish packages to CodeArtifact.
2. In the pre-build section, add your CodeArtifact repository to your NuGet configuration.

See the following buildspec.yaml examples. For more information, see Using CodeArtifact with NuGet (p. 113).

After the credential provider is installed and your repository source is added, you can run NuGet CLI tool commands from the build section to consume NuGet packages.

Linux

To consume NuGet packages using dotnet:

```yaml
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 packageName --source packageSourceName
```

Windows

To consume NuGet packages using dotnet:

```yaml
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 packageName --source packageSourceName
```
Build with NuGet packages

To build with NuGet packages from CodeBuild, include the following in your project’s buildspec.yaml file.

1. In the install section, install the CodeArtifact Credential Provider to configure command line tools such as msbuild and dotnet to build and publish packages to CodeArtifact.
2. In the pre-build section, add your CodeArtifact repository to your NuGet configuration.

See the following buildspec.yaml examples. For more information, see Using CodeArtifact with NuGet (p. 113).

After the credential provider is installed and your repository source is added, you can run NuGet CLI tool commands like dotnet build from the build section.

**Linux**

To build NuGet packages using dotnet:

```yaml
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
```

To build NuGet packages using msbuild:

```yaml
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

To build NuGet packages using dotnet:

```yaml
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
```

To build NuGet packages using msbuild:

```yaml
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
```

Publish NuGet packages

To publish NuGet packages from CodeBuild, include the following in your project's `buildspec.yaml` file.

1. In the `install` section, install the CodeArtifact Credential Provider to configure command line tools such as `msbuild` and `dotnet` to build and publish packages to CodeArtifact.
2. In the `pre-build` section, add your CodeArtifact repository to your NuGet configuration.

See the following `buildspec.yaml` examples. For more information, see Using CodeArtifact with NuGet (p. 113).

After the credential provider is installed and your repository source is added, you can run NuGet CLI tool commands from the `build` section and publish your NuGet packages.

Linux

To publish NuGet packages using dotnet:

```yaml
version: 0.2
```
Dependency caching

You can enable local caching in CodeBuild to reduce the number of dependencies that need to be fetched from CodeArtifact for each build. For information, see Build Caching in AWS CodeBuild in the AWS CodeBuild User Guide. After you enable a custom local cache, add the cache directory to your project's `buildspec.yaml` file.

For example, if you are using `mvn`, use the following.

```yaml
cache:
  paths:
    - './root/.m2/**/*'
```

For other tools, use the cache folders shown in this table.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Cache directory</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mvn</code></td>
<td><code>/root/.m2/**/*</code></td>
</tr>
</tbody>
</table>
## Dependency caching

<table>
<thead>
<tr>
<th>Tool</th>
<th>Cache directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>gradle</td>
<td><code>/root/.gradle/caches/**/*</code></td>
</tr>
<tr>
<td>pip</td>
<td><code>/root/.cache/pip/**/*</code></td>
</tr>
<tr>
<td>npm</td>
<td><code>/root/.npm/**/*</code></td>
</tr>
<tr>
<td>nuget</td>
<td><code>/root/.nuget/**/*</code></td>
</tr>
</tbody>
</table>
Working with CodeArtifact events

CodeArtifact is integrated with Amazon EventBridge, a service that automates and responds to events, including changes in a CodeArtifact repository. You can create rules for events and configure what happens when an event matches a rule. EventBridge was formerly called CloudWatch Events.

The following actions can be triggered by an event:

- Invoking an AWS Lambda function.
- Activating an AWS Step Functions state machine.
- Notifying an Amazon SNS topic or an Amazon SQS queue.
- Starting a pipeline in AWS CodePipeline.

CodeArtifact creates an event when a package version is created, modified, or deleted. The following are examples of CodeArtifact events:

- Publishing a new package version (for example, by running `npm publish`).
- Adding a new asset to an existing package version (for example, by pushing a new JAR file to an existing Maven package).
- Copying a package version from one repository to another using `copy-package-versions`. For more information, see Copy packages between repositories (p. 58).
- Deleting a package version using `delete-package-version`. For more information, see Delete a package version (p. 62).
- Retaining a package version in a downstream repository when it has been fetched from an upstream repository. For more information, see Working with upstream repositories in CodeArtifact (p. 38).
- Ingesting a package version from an external repository into a CodeArtifact repository. For more information, see Add an external connection (p. 26).

Events are delivered to both the account that owns the domain and the account that administers the repository. For example, suppose that account 111111111111 owns the domain `my_domain`. Account 222222222222 creates a repository in `my_domain` called `repo2`. When a new package version is published to `repo2`, both accounts receive the EventBridge events. The domain-owning account (111111111111) receives events for all repositories in the domain. If a single account owns both the domain and the repository within it, only a single event is delivered.

The following topics describe the CodeArtifact event format. They show you how to configure CodeArtifact events, and how to use events with other AWS services. For more information, see Getting Started with Amazon EventBridge in the Amazon EventBridge User Guide.

Topics

- CodeArtifact event format and example (p. 132)
- Use an event to start a CodePipeline execution (p. 135)
- Use an event to run a Lambda function (p. 136)

CodeArtifact event format and example

The following are event fields and descriptions along with an example of a CodeArtifact event.
# CodeArtifact event format

All CodeArtifact events include the following fields.

<table>
<thead>
<tr>
<th>Event field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>The version of the event format. There is currently only a single version, 0.</td>
</tr>
<tr>
<td>id</td>
<td>A unique identifier for the event.</td>
</tr>
<tr>
<td>detail-type</td>
<td>The type of event. This determines the fields in the detail object. The one detail-type currently supported is CodeArtifact Package Version State Change.</td>
</tr>
<tr>
<td>source</td>
<td>The source of the event. For CodeArtifact, it will be aws.codeartifact.</td>
</tr>
<tr>
<td>account</td>
<td>The account that owns the domain and repository that triggered the event.</td>
</tr>
<tr>
<td>time</td>
<td>The exact time the event was triggered.</td>
</tr>
<tr>
<td>region</td>
<td>The region where the event was triggered.</td>
</tr>
<tr>
<td>resources</td>
<td>A list that contains the ARN of the package that changed. The list contains one entry. For information about package ARN format, see Grant write access to packages (p. 25).</td>
</tr>
<tr>
<td>domainName</td>
<td>The domain that contains the repository that contains the package.</td>
</tr>
<tr>
<td>domainOwner</td>
<td>The AWS account ID of the owner of the domain.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The repository that contains the package.</td>
</tr>
<tr>
<td>packageFormat</td>
<td>The format of the package that triggered the event.</td>
</tr>
<tr>
<td>packageNamespace</td>
<td>The namespace of the package that triggered the event.</td>
</tr>
<tr>
<td>packageName</td>
<td>The name of the package that triggered the event.</td>
</tr>
<tr>
<td>packageVersion</td>
<td>The version of the package that triggered the event.</td>
</tr>
<tr>
<td>packageVersionState</td>
<td>The state of the package version when the event was triggered. Possible values are Unfinished, Published, Unlisted, Archived, and Disposed.</td>
</tr>
<tr>
<td>packageVersionRevision</td>
<td>A value that uniquely identifies the state of the assets and metadata of the package version when the event was triggered. If the package version is modified (for example, by adding</td>
</tr>
<tr>
<td>Event field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>changes.assetsAdded</td>
<td>The number of assets added to a package that triggered an event. Examples of an asset are a Maven JAR file or a Python wheel.</td>
</tr>
<tr>
<td>changes.assetsRemoved</td>
<td>The number of assets removed from a package that triggered an event.</td>
</tr>
<tr>
<td>changes.assetsUpdated</td>
<td>The number of assets modified in the package that triggered the event.</td>
</tr>
<tr>
<td>changes.metadataUpdated</td>
<td>A boolean value that is set to true if the event includes modified package-level metadata. For example, an event might modify a Maven pom.xml file.</td>
</tr>
<tr>
<td>changes.statusChanged</td>
<td>A boolean value that is set to true if the event's packageVersionStatus is modified (for example, if packageVersionStatus changes from Unfinished to Published).</td>
</tr>
<tr>
<td>operationType</td>
<td>Describes the high-level type of the package version change. The possible values are Created, Updated, and Deleted.</td>
</tr>
<tr>
<td>sequenceNumber</td>
<td>An integer that specifies an event number for a package. Each event on a package increments the sequenceNumber so events can be arranged sequentially. An event can increment the sequenceNumber by any integer number.</td>
</tr>
</tbody>
</table>

**Note**

EventBridge events might be received out of order. sequenceNumber can be used to determine their actual order.

| eventDeduplicationId       | An ID used to differentiate duplicate EventBridge events. In rare cases, EventBridge might trigger the same rule more than once for a single event or scheduled time. Or, it might invoke the same target more than once for a given triggered rule. |

---

**CodeArtifact event example**

The following is an example of a CodeArtifact event that might be triggered when an npm package is published.

```json
{
  "version":"0",
  "id":"73f03fec-a137-971e-6ac6-07c8fffffff",
  "detail-type":"CodeArtifact Package Version State Change",
  "source":"aws.codeartifact",
  "account":"123456789012",
  "time":"2019-11-21T23:19:54Z",
  "region":"us-west-2",
}
This example demonstrates how to configure an Amazon EventBridge rule so that an AWS CodePipeline execution starts when a package version in a CodeArtifact repository is published, modified, or deleted.

Topics
- Configure EventBridge permissions (p. 135)
- Create the EventBridge rule (p. 135)
- Create the EventBridge rule target (p. 136)

Configure EventBridge permissions

You must add permissions for EventBridge to use CodePipeline to invoke the rule that you create. To add these permissions using the AWS Command Line Interface (AWS CLI), follow step 1 in Create a CloudWatch Events Rule for a CodeCommit Source (CLI) in the AWS CodePipeline User Guide.

Create the EventBridge rule

To create the rule, use the put-rule command with the --name and --event-pattern parameters. The event pattern specifies values that are matched against the contents of each event. The target is triggered if the pattern matches the event. For example, the following pattern matches CodeArtifact events from the myrepo repository in the my_domain domain.

```
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" }}'
```
Create the EventBridge rule target

The following command adds a target to the rule so that when an event matches the rule, a CodePipeline execution is triggered. For the RoleArn parameter, specify the Amazon Resource Name (ARN) of the role created earlier in this topic.

```bash
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
```

Use an event to run a Lambda function

This example shows you how to configure an EventBridge rule that starts an AWS Lambda function when a package version in a CodeArtifact repository is published, modified, or deleted.

For more information, see Tutorial: Schedule AWS Lambda Functions Using EventBridge in the Amazon EventBridge User Guide.

Topics

- Create the EventBridge rule (p. 136)
- Create the EventBridge rule target (p. 136)
- Configure EventBridge permissions (p. 136)

Create the EventBridge rule

To create a rule that starts a Lambda function, use the `put-rule` command with the `--name` and `--event-pattern` options. The following pattern specifies npm packages in the @types scope in any repository in the my_domain domain.

```bash
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"]}'
```

Create the EventBridge rule target

The following command adds a target to the rule that runs the Lambda function when an event matches the rule. For the arn parameter, specify the Amazon Resource Name (ARN) of the Lambda function.

```bash
aws events put-targets --rule MyCodeArtifactRepoRule --targets \
  Id=1,Arn=arn:aws:lambda:us-west-2:111122223333:function:MyLambdaFunction
```

Configure EventBridge permissions

Use the `add-permission` command to grant permissions for the rule to invoke a Lambda function. For the `--source-arn` parameter, specify the ARN of the rule that you created earlier in this example.

```bash
aws lambda add-permission --function-name MyLambdaFunction \ 
  --statement-id my-statement-id --action 'lambda:InvokeFunction'
```
--principal events.amazonaws.com \\
Security in CodeArtifact

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to CodeArtifact, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using CodeArtifact. The following topics show you how to configure CodeArtifact to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CodeArtifact resources.

Topics
- Data protection in AWS CodeArtifact (p. 138)
- Identity and access management in AWS CodeArtifact (p. 139)
- Monitoring CodeArtifact (p. 152)
- Compliance validation for AWS CodeArtifact (p. 155)
- AWS CodeArtifact authentication and tokens (p. 155)
- Resilience in AWS CodeArtifact (p. 158)
- Infrastructure security in AWS CodeArtifact (p. 159)

Data protection in AWS CodeArtifact

The AWS shared responsibility model applies to data protection in AWS CodeArtifact. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
• Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
• If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a Name field. This includes when you work with CodeArtifact or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption
Encryption is an important part of CodeArtifact security. Some encryption, such as for data in transit, is provided by default and does not require you to do anything. Other encryption, such as for data at rest, you can configure when you create your project or build.

• **Encryption of data at rest** - All assets stored in CodeArtifact are encrypted by using AWS KMS keys (KMS keys). This includes all assets in all packages in all repositories. One KMS key is used for each domain to encrypt all its assets. By default, an AWS managed KMS key is used, so you do not need to create a KMS key. If you want, you can use a customer-managed KMS key that you create and configure. For more information, see Creating keys and AWS Key Management Service concepts in the AWS Key Management Service User Guide. You can specify a customer-managed KMS key when you create a domain. For more information, see Working with domains in CodeArtifact (p. 72).
• **Encryption of data in transit** - All communication between customers and CodeArtifact and between CodeArtifact and its downstream dependencies protected using TLS encryption.

Traffic privacy
You can improve the security of your CodeArtifact domains and the assets that they contain by configuring CodeArtifact to use an interface virtual private cloud (VPC) endpoint. To do this, you don't need an internet gateway, NAT device, or virtual private gateway. For more information, see Working with Amazon VPC endpoints (p. 160). For more information about AWS PrivateLink and VPC endpoints, see AWS PrivateLink and Accessing AWS Services Through PrivateLink.

Identity and access management in AWS CodeArtifact
Access to AWS CodeArtifact requires credentials. Those credentials must have permissions to access AWS resources, domains, and their repositories. The following sections describe how you can use AWS Identity and Access Management (IAM) and CodeArtifact to help secure access to your resources.

• Authentication (p. 139)
• Access control (p. 140)

Authentication
You can access AWS as any of the following types of identities:
• **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your *root credentials*, and they provide complete access to all of your AWS resources.

  **Important**
  For security reasons, we recommend that you use the root credentials only to create an administrator user, which is an IAM user with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the IAM User Guide.

• **IAM user** – An IAM user is simply an identity in your AWS account that has custom permissions (for example, permission to create domains and repositories in CodeArtifact). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the AWS SDKs or by using the AWS Command Line Interface (AWS CLI). The AWS SDKs and AWS CLI tools use the access keys to cryptographically sign your request. If you don't use the AWS tools, you must sign the request yourself. CodeArtifact supports Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see the Signature Version 4 Signing Process in the AWS General Reference.

• **IAM role** – An IAM role is similar to an IAM user, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:
  - **Federated user access** – Instead of creating an IAM user, you can use pre-existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.
  - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the IAM User Guide. You can also grant cross-account access to a domain using resource-based policies. For more information, see Enable cross-account access to a domain (p. 76).
  - **AWS service access** – You can use an IAM role in your account to grant permissions to an AWS service to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data stored in the bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.
  - **Applications running on Amazon EC2** – Instead of storing access keys in the Amazon EC2 instance for use by applications running on the instance and making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an Amazon EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the Amazon EC2 instance to get temporary credentials. For more information, see Using Roles for Applications on Amazon EC2 in the IAM User Guide.

**Access control**

You can have valid credentials to authenticate your requests, but unless you have permissions, you cannot create or access AWS CodeArtifact resources. For example, you must have permissions to create, view, or delete repositories, associate a connection to an external public repository to a CodeArtifact repository, and to delete package versions.
The following sections describe how to manage permissions for CodeArtifact. We recommend that you read the overview first.

- Overview of managing access permissions to your AWS CodeArtifact resources (p. 141)
- Using identity-based policies for AWS CodeArtifact (p. 144)
- AWS CodeArtifact permissions reference (p. 151)

## Overview of managing access permissions to your AWS CodeArtifact resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles).

### Note
An account administrator (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the IAM User Guide.

When you grant permissions, you decide who is getting the permissions, the resources they can access, and the actions that can be performed on those resources.

### Topics
- AWS CodeArtifact resources and operations (p. 141)
- Understanding resource ownership (p. 142)
- Managing access to resources (p. 142)
- Specifying policy elements: Actions, effects, and principals (p. 143)
- AWS Global Condition Context Keys (p. 144)

## AWS CodeArtifact resources and operations

In AWS CodeArtifact, the primary resource is a domain. In a policy, you use an Amazon Resource Name (ARN) to identify the resource the policy applies to. Repositories are also resources and have ARNs associated with them. For more information, see Amazon Resource Names (ARNs) in the Amazon Web Services General Reference.

<table>
<thead>
<tr>
<th>Resource type</th>
<th>ARN format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>arn:aws:codeartifact:region-ID:account-ID:domain/my_domain</td>
</tr>
<tr>
<td>Repository</td>
<td>arn:aws:codeartifact:region-ID:account-ID:repository/my_domain/my_repo</td>
</tr>
<tr>
<td>All CodeArtifact resources</td>
<td>arn:aws:codeartifact:*</td>
</tr>
</tbody>
</table>
Overview of managing access

<table>
<thead>
<tr>
<th>Resource type</th>
<th>ARN format</th>
</tr>
</thead>
<tbody>
<tr>
<td>All CodeArtifact resources owned by the specified account in the specified AWS Region</td>
<td>arn:aws:codeartifact:region-ID:account-ID:*</td>
</tr>
</tbody>
</table>

**Note**
Most AWS services treat a colon (:) or a forward slash (/) as the same character in ARNs. However, CodeArtifact uses an exact match in resource patterns and rules. Be sure to use the correct characters when you create event patterns so that they match the ARN syntax in the resource.

For example, you can indicate a specific domain (`myDomain`) in your statement using its ARN as follows.

```json
```

To specify all resources, or if an API action does not support ARNs, use the wildcard character (`*`) in the **Resource** element as follows.

```json
"Resource": "*"
```

To specify multiple resources in a single statement, separate their ARNs with commas, as follows.

```json
]
```

CodeArtifact provides a set of operations to work with the CodeArtifact resources. For a list, see AWS CodeArtifact permissions reference (p. 151).

**Understanding resource ownership**

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create a rule, your AWS account is the owner of the CodeArtifact resource.
- If you create an IAM user in your AWS account and grant permissions to create CodeArtifact resources to that user, the user can create CodeArtifact resources. However, your AWS account, to which the user belongs, owns the CodeArtifact resources.
- If you create an IAM role in your AWS account with permissions to create CodeArtifact resources, anyone who can assume the role can create CodeArtifact resources. Your AWS account, to which the role belongs, owns the CodeArtifact resources.

**Managing access to resources**

A permissions policy describes who has access to which resources.

**Note**
This section discusses the use of IAM in AWS CodeArtifact. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the
Policies attached to an IAM identity are referred to as identity-based policies (IAM policies). Policies attached to a resource are referred to as resource-based policies. CodeArtifact supports identity-based (IAM policies) and resource-based policies.

Identity-based policies

You can attach policies to IAM identities.

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to view domains and other AWS CodeArtifact resources in the AWS CodeArtifact console, you can attach a permissions policy to a user or group that the user belongs to.

- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:

  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. The principal in the trust policy must also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

For more information about using IAM to delegate permissions, see Access Management in the IAM User Guide.

In CodeArtifact, identity-based policies are used to manage permissions to the resources related to artifact management. For example, you can control access to a domain.

You can create IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users. For more information about how to create IAM roles and to explore example IAM policy statements for CodeArtifact, see Overview of managing access permissions to your AWS CodeArtifact resources (p. 141).

Specifying policy elements: Actions, effects, and principals

For each AWS CodeArtifact resource, the service defines a set of API operations. To grant permissions for these API operations, CodeArtifact defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more information, see AWS CodeArtifact resources and operations (p. 141) and AWS CodeArtifact permissions reference (p. 151).

The following are the basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the policy applies to.

- **Action** – You use action keywords to identify resource operations you want to allow or deny. For example, the `codeartifact:DeleteDomain` permission gives the user permissions to perform the `DeleteDomain` operation.

- **Effect** – You specify the effect, either allow or deny, when the user requests the action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource. You might do this to make sure that a user cannot access a resource, even if a different policy grants access.
Using identity-based policies

- **Principal** – In identity-based policies (IAM policies), the user the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions.


For a table showing all of the CodeArtifact API actions and the resources they apply to, see the [AWS CodeArtifact permissions reference](https://docs.aws.amazon.com/codeartifact/latest/userguide/codeartifact-manage-access.html#codeartifact-manage-access-resources) (p. 151).

**AWS Global Condition Context Keys**

AWS CodeArtifact does not support the following AWS Global Condition Context Keys:

- Referer
- UserAgent

For more information on condition keys, see [AWS Global Condition Context Keys](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements.html).

**Using identity-based policies for AWS CodeArtifact**

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on AWS CodeArtifact resources.

**Important**

We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your CodeArtifact resources. For more information, see [Overview of managing access permissions to your AWS CodeArtifact resources](https://docs.aws.amazon.com/codeartifact/latest/userguide/codeartifact-manage-access.html) (p. 141).

**Topics**

- Policy examples (p. 144)
- Permissions required to use the AWS CodeArtifact console (p. 146)
- AWS managed (predefined) policies for AWS CodeArtifact (p. 146)
- Limit authorization token duration (p. 147)

**Policy examples**

The following IAM policy examples provide access to specific CodeArtifact actions and resources. Some policies include the `sts:GetServiceBearerToken` permission, which is required to call the `GetAuthorizationToken` API.

**Allow a user to get information about repositories and domains**

The following policy allows an IAM user or role to list and describe any type of CodeArtifact resource, including domains, repositories, packages, and assets. The policy also includes the `codeArtifact:ReadFromRepository` permission, which allows the principal to fetch packages from a CodeArtifact repository. It does not allow creating new domains or repositories and does not allow publishing new packages.

The `codeArtifact:GetAuthorizationToken` and `sts:GetServiceBearerToken` permissions are required to call the `GetAuthorizationToken` API.

```
"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"
        }
      }
    }
]
}

Allow a user to get information about specific domains

The following shows an example of a permissions policy that allows a user to list domains only in the us-east-2 region for account 123456789012 for any domain that starts with the name my.

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codeartifact:ListDomains",
    }
  ]
}

Allow a user to get information about specific repositories

The following shows an example of a permissions policy that allows a user to get information about repositories that end with test, including information about the packages in them. The user will not be able to publish, create, or delete resources.

{
  "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": [
Permissions required to use the AWS CodeArtifact console

A user who uses the AWS CodeArtifact console must have a minimum set of permissions that allows
the user to describe other AWS resources for the AWS account. You must have permissions from the
following services:

- AWS CodeArtifact
- AWS Key Management Service (AWS KMS)

If you create an IAM policy that is more restrictive than the minimum required permissions, the console
won't function as intended.

AWS managed (predefined) policies for AWS CodeArtifact

AWS addresses many common use cases by providing standalone IAM policies that are created and
administered by AWS. These AWS managed policies grant necessary permissions for common use cases
so you can avoid having to investigate what permissions are needed. For more information, see AWS
Managed Policies in the IAM User Guide.

The following AWS managed policies, which you can attach to users in your account, are specific to AWS
CodeArtifact.

- AWSCodeArtifactAdminAccess – Provides full access to CodeArtifact including permissions to
  administrate CodeArtifact domains.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:*",
                "codeartifact:Describe*"
            ],
            "Resource": "arn:aws:codeartifact:*:*:package/*/test/*//*/*",
            "Effect": "Allow",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "codeartifact:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```
Using identity-based policies

```
"Action": "sts:GetServiceBearerToken",
"Resource": "*",
"Condition": {
  "StringEquals": {
    "sts:AWSServiceName": "codeartifact.amazonaws.com"
  }
}
```

- **AWSCodeArtifactReadOnlyAccess** – Provides read-only access to CodeArtifact.

To create and manage CodeArtifact service roles, you must also attach the AWS managed policy named **IAMFullAccess**.

You can also create your own custom IAM policies to allow permissions for CodeArtifact actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

**Limit authorization token duration**

Users must authenticate to CodeArtifact with authorization tokens to publish or consume package versions. Authorization tokens are valid only during their configured lifetime. Tokens have a default lifetime of 12 hours. For more information on authorization tokens, see AWS CodeArtifact authentication and tokens (p. 155).

When fetching a token, users can configure the lifetime of the token. Valid values for the lifetime of an authorization token are 0, and any number between 900 (15 minutes) and 43200 (12 hours). A value of 0 will create a token with a duration equal to the user's role's temporary credentials.

Administrators can limit the valid values for the lifetime of an authorization token by using the `sts:DurationSeconds` condition key in the permissions policy attached to the user or group. If the user attempts to create an authorization token with a lifetime outside of the valid values, the token creation will fail.
The following example policies limit the possible durations of an authorization token created by CodeArtifact users.

**Example policy: Limit token lifetime to exactly 12 hours (43200 seconds)**

With this policy, users will only be able to create authorization tokens with a lifetime of 12 hours.

```json
{
   "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"
            }
         }
      }
   ]
}
```

**Example policy: Limit token lifetime between 15 minutes and 1 hour, or equal to the user’s temporary credentials period**

With this policy, users will be able to create tokens that are valid between 15 minutes and 1 hour. Users will also be able to create a token that lasts the duration of their role’s temporary credentials by specifying 0 for `--durationSeconds`.

```json
{
   "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"
            }
         }
      }
   ]
}
```
Using tags to control access to CodeArtifact resources

Conditions in IAM user policy statements are part of the syntax that you use to specify permissions to resources required by CodeArtifact actions. Using tags in conditions is one way to control access to resources and requests. For information about tagging CodeArtifact resources, see Tagging resources (p. 167). This topic discusses tag-based access control.

When you design IAM policies, you might be setting granular permissions by granting access to specific resources. As the number of resources that you manage grows, this task becomes more difficult. Tagging resources and using tags in policy statement conditions can make this task easier. You grant access in bulk to any resource with a certain tag. Then you repeatedly apply this tag to relevant resources, during creation or later.

Tags can be attached to the resource or passed in the request to services that support tagging. In CodeArtifact, resources can have tags, and some actions can include tags. When you create an IAM policy, you can use tag condition keys to control:

- Which users can perform actions on a domain or repository resource, based on tags that it already has.
- Which tags can be passed in an action's request.
- Whether specific tag keys can be used in a request.

For the complete syntax and semantics of tag condition keys, see Controlling Access Using Tags in the IAM User Guide.

The following examples demonstrate how to specify tag conditions in policies for CodeArtifact users.

Example 1: Limit actions based on tags in the request

The AWSCodeArtifactAdminAccess managed user policy gives users unlimited permission to perform any CodeArtifact action on any resource.

The following policy limits this power and denies unauthorized users permission to create repositories unless the request contains certain tags. To do that, it denies the CreateRepository action if the request does not specify a tag named costcenter with one of the values 1 or 2. A customer's administrator must attach this IAM policy to unauthorized IAM users, in addition to the managed user policy.

```json
{
    "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
                    ]
                }
            }
        }
    ]
}
```
Using tags to control access to CodeArtifact resources

Example 2: Limit actions based on resource tags

The AWSCodeArtifactAdminAccess managed user policy gives users unlimited permission to perform any CodeArtifact action on any resource.

The following policy limits this power and denies unauthorized users permission to perform actions on repositories in specified domains. To do that, it denies some actions if the resource has a tag named Key1 with one of the values Value1 or Value2. (The aws:ResourceTag condition key is used to control access to the resources based on the tags on those resources.) A customer's administrator must attach this IAM policy to unauthorized IAM users, in addition to the managed user policy.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codeartifact:TagResource",
        "codeartifact:UntagResource",
        "codeartifact:ListPackages",
        "codeartifact:DeletePackageVersions",
        "codeartifact:ListPackageVersions",
        "codeartifact:DescribePackageVersion",
        "codeartifact:ListTagsForResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Key1": ["Value1", "Value2"]
        }
      }
    }
  ]
}
```

Example 3: Allow actions based on resource tags

The following policy grants users permission to perform actions on, and get information about, repositories and packages in CodeArtifact.

To do that, it allows specific actions if the repository has a tag named Key1 with the value Value1. (The aws:RequestTag condition key is used to control which tags can be passed in an IAM request.) The aws:TagKeys condition ensures tag key case sensitivity. This policy is useful for IAM users who don't have the AWSCodeArtifactAdminAccess managed user policy attached. The managed policy gives users unlimited permission to perform any CodeArtifact action on any resource.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeartifact:UpdateRepository",
```
Example 4: Allow actions based on tags in the request

The following policy grants users permission to create repositories in specified domains in CodeArtifact. To do that, it allows the CreateRepository and TagResource actions if the create resource API in the request specifies a tag named Key1 with the value Value1. (The aws:RequestTag condition key is used to control which tags can be passed in an IAM request.) The aws:TagKeys condition ensures tag key case sensitivity. This policy is useful for IAM users who don't have the AWSCodeArtifactAdminAccess managed user policy attached. The managed policy gives users unlimited permission to perform any CodeArtifact action on any resource.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codeartifact:CreateRepository",
                "codeartifact:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Key1": "Value1"
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": ["Key1"]
                }
            }
        }
    ]
}
```

AWS CodeArtifact permissions reference

You can use the following table as a reference when you are setting up Access control (p. 140) and writing permissions policies that you can attach to an IAM identity (identity-based policies).

You can use AWS-wide condition keys in your AWS CodeArtifact policies to express conditions. For a list, see IAM JSON Policy Elements Reference in the IAM User Guide.

You specify the actions in the policy's Action field. To specify an action, use the codeartifact: prefix followed by the API operation name (for example, codeartifact:CreateDomain and codeartifact:AssociateExternalConnection). To specify multiple actions in a single statement,
separate them with commas (for example, "Action": [ "codeartifact:CreateDomain", "codeartifact:AssociateExternalConnection" ]).

Using wildcard characters

You specify an ARN, with or without a wildcard character (*), as the resource value in the policy's Resource field. You can use a wildcard to specify multiple actions or resources. For example, codeartifact:* specifies all CodeArtifact actions and codeartifact:Describe* specifies all CodeArtifact actions that begin with the word Describe.

Monitoring CodeArtifact

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS CodeArtifact and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure, if one occurs. AWS provides the following for monitoring your CodeArtifact resources and for responding to potential incidents:

Topics

- Logging CodeArtifact API calls with AWS CloudTrail (p. 152)

Logging CodeArtifact API calls with AWS CloudTrail

CodeArtifact is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in CodeArtifact. CloudTrail captures all API calls for CodeArtifact as events, including calls from package manager clients.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon Simple Storage Service (Amazon S3) bucket, including events for CodeArtifact. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to CodeArtifact, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

CodeArtifact information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in CodeArtifact, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for CodeArtifact, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. You can also configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following topics:

- Creating a Trail for Your AWS Account
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to CodeArtifact actions are tracked in CloudTrail log files, where they are written with other AWS service records. CloudTrail determines when to create and write to a new file based on a time period and file size.
All CodeArtifact actions are logged by CloudTrail. For example, calls to the `ListRepositories` (in the AWS CLI, `aws codeartifact list-repositories`), `CreateRepository` (`aws codeartifact create-repository`), and `ListPackages` (`aws codeartifact list-packages`) actions generate entries in the CloudTrail log files, in addition to `npm` client commands such as `npm install` and `npm publish`. `npm` client commands typically make more than one HTTP request to the server. Each request generates a separate CloudTrail log event.

**Cross-account delivery of CloudTrail logs**

Up to three separate accounts receive CloudTrail logs for a single API call:

- The account that made the request—for example, the account that called `GetAuthorizationToken`.
- The repository administrator account—for example, the account that administers the repository that `ListPackages` was called on.
- The domain owner's account—for example, the account that owns the domain that contains the repository that an API was called on.

For APIs like `ListRepositoriesInDomain` that are actions against a domain and not a specific repository, only the calling account and the domain owner's account receive the CloudTrail log. For APIs like `ListRepositories` that are not authorized against any resource, only the account of the caller receives the CloudTrail log.

**Understanding CodeArtifact log file entries**

CloudTrail log files can contain one or more log entries. Each entry lists multiple JSON-formatted events. A log event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. Log entries are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

**Topics**

- Example: A log entry for calling the `GetAuthorizationToken` API (p. 153)
- Example: A log entry for fetching an npm package version (p. 154)

**Example: A log entry for calling the `GetAuthorizationToken` API**

A log entry created by `GetAuthorizationToken` includes the domain name in the `requestParameters` field.

```json
{
  "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"
      }
    }
  }
}
```

153
Example: A log entry for fetching an npm package version

Requests made by the npm client have additional data logged including the domain name, repository name, and package name in the requestParameters field. The URL path and HTTP method are logged in the additionalEventData field.

```json
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6CZEXAMPLE",
    "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": "AIDACKCEVSQ6CZEXAMPLE",
        "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": "AWS Internal",
  "requestParameters": {
    "domainName": "example-domain",
    "repositoryName": "example-repo",
    "packageName": "lodash",
    "packageFormat": "npm",
    "packageVersion": "4.17.20"
  }
}
```
Compliance validation for AWS CodeArtifact

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs.

At present, AWS CodeArtifact is not in scope of any specific compliance programs.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact in the AWS Artifact User Guide.

Your compliance responsibility when using CodeArtifact is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. If your use of CodeArtifact is subject to compliance with standards such as HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- **Architecting for HIPAA Security and Compliance Whitepaper** – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
- **AWS Config** – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- **AWS Security Hub** – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

AWS CodeArtifact authentication and tokens

CodeArtifact requires users to authenticate with the service in order to publish or consume package versions. You must authenticate to the CodeArtifact service by creating an authorization token using your AWS credentials. In order to create an authorization token, you must have the correct permissions. For more information on CodeArtifact permissions, see Overview of managing access permissions to your AWS CodeArtifact resources (p. 141).

To fetch an authorization token from CodeArtifact, you must call the `GetAuthorizationToken` API. Using the AWS CLI, you can call `GetAuthorizationToken` with the `login` or `get-authorization-token` command.

- `aws codeartifact login (npm, pip, and twine)`: This command makes it easy to configure common package managers to use CodeArtifact in a single step. Calling `login` fetches a token
with `GetAuthorizationToken` and configures your package manager with the token and correct CodeArtifact repository endpoint.

- `aws codeartifact get-authorization-token`: For package managers not supported by `login`, you can call `get-authorization-token` directly and then configure your package manager with the token as required, for example, by adding it to a configuration file or storing it in an environment variable.

CodeArtifact authorization tokens are valid for a default period of 12 hours. Tokens can be configured with a lifetime between 15 minutes and 12 hours. When the lifetime expires, you must fetch another token. The token lifetime begins after `login` or `get-authorization-token` is called.

If `login` or `get-authorization-token` is called while assuming a role, you can configure the lifetime of the token to be equal to the remaining time in the session duration of the role by setting the value of `--duration-seconds` to 0. Otherwise, the token lifetime is independent of the maximum session duration of the role. For example, suppose that you call `sts assume-role` and specify a session duration of 15 minutes, and then call `login` to fetch a CodeArtifact authorization token. In this case, the token is valid for the full 12-hour period even though this is longer than the 15-minute session duration. For information about controlling session duration, see Using IAM Roles in the IAM User Guide.

**Tokens created with the `login` command**

The `aws codeartifact login` command will fetch a token with `GetAuthorizationToken` and configure your package manager with the token and correct CodeArtifact repository endpoint.

The following table describes the parameters for the `login` command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--tool</code></td>
<td>Yes</td>
<td>The package manager to authenticate to. Possible values are <code>npm</code>, <code>pip</code>, and <code>twine</code>.</td>
</tr>
<tr>
<td><code>--domain</code></td>
<td>Yes</td>
<td>The domain name that the repository belongs to.</td>
</tr>
<tr>
<td><code>--domain-owner</code></td>
<td>No</td>
<td>The ID of the owner of the domain. This parameter is required if accessing a domain that is owned by an AWS account that you are not authenticated to. For more information, see Cross-account domains (p. 73).</td>
</tr>
<tr>
<td><code>--repository</code></td>
<td>Yes</td>
<td>The name of the repository to authenticate to.</td>
</tr>
<tr>
<td><code>--duration-seconds</code></td>
<td>No</td>
<td>The time, in seconds, that the login information is valid. The minimum value is 900* and maximum value is 43200.</td>
</tr>
<tr>
<td><code>--namespace</code></td>
<td>No</td>
<td>Associates a namespace with your repository tool.</td>
</tr>
<tr>
<td><code>--dry-run</code></td>
<td>No</td>
<td>Only print the commands that would be executed to connect your tool with your repository without making any changes to your configuration.</td>
</tr>
</tbody>
</table>
Tokens created with the GetAuthorizationToken API

You can call `get-authorization-token` to fetch an authorization token from CodeArtifact.

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

You can change how long a token is valid using the `--duration-seconds` argument. The minimum value is 900 and the maximum value is 43200. The following example creates a token that will last for 1 hour (3600 seconds).

```
aws codeartifact get-authorization-token --domain my_domain --domain-owner 111122223333 --query authorizationToken --output text --duration-seconds 3600
```

If calling `get-authorization-token` while assuming a role the token lifetime is independent of the maximum session duration of the role. You can configure the token to expire when the assumed role's session duration expires by setting `--duration-seconds` to 0.

```
aws codeartifact get-authorization-token --domain my_domain --domain-owner 111122223333 --query authorizationToken --output text --duration-seconds 0
```

See the following documentation for more information:

- For guidance on tokens and environment variables, see Pass an auth token using an environment variable (p. 157).
- For Python users, see Configure pip without the login command (p. 97) or Configure twine without the login command (p. 97).
- For Maven users, see Use CodeArtifact with Gradle (p. 100) or Use CodeArtifact with mvn (p. 105).
- For npm users, see Configuring npm without using the login command (p. 85).

Pass an auth token using an environment variable

AWS CodeArtifact uses authorization tokens vended by the GetAuthorizationToken API to authenticate and authorize requests from build tools such as Maven and Gradle. For more information on these auth tokens, see Tokens created with the GetAuthorizationToken API (p. 157).
You can store these auth tokens in an environment variable that can be read by a build tool to obtain the token it needs to fetch packages from a CodeArtifact repository or publish packages to it.

For security reasons, this approach is preferable to storing the token in a file where it might be read by other users or processes, or accidentally checked into source control.

1. Configure your AWS credentials as described in Install or upgrade and then configure the AWS CLI (p. 4).
2. Set the `CODEARTIFACT_AUTH_TOKEN` environment variable:

    ```
    Note
    In some scenarios, you don't need to include the `--domain-owner` argument. For more information, see Cross-account domains (p. 73).
    ```

    - macOS or Linux:

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

    - Windows (using default command 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
      ```

    - Windows PowerShell:

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

Revoking CodeArtifact authorization tokens

When an authenticated user creates a token to access CodeArtifact resources, that token lasts until its customizable access period has ended. The default access period is 12 hours. In some circumstances, you might want to revoke access to a token before the access period has expired. You can revoke access to CodeArtifact resources by following these instructions.

If you created the access token using temporary security credentials, such as assumed roles or federated user access, you can revoke access by updating an IAM policy to deny access. For information, see Disabling Permissions for Temporary Security Credentials in the IAM User Guide.

If you used long-term IAM user credentials to create the access token, you must modify the user's policy to deny access, or delete the IAM user. For more information, see Changing Permissions for an IAM User or Deleting an IAM User.

If you used an account root user's credentials to call GetAuthorizationToken, you can't invalidate the authorization token before it expires because the root user doesn't have a permissions policy.

Resilience in AWS CodeArtifact

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. AWS CodeArtifact operates in multiple Availability Zones and stores artifact data and metadata in Amazon S3 and Amazon DynamoDB. Your encrypted
data is redundantly stored across multiple facilities and multiple devices in each facility, making it highly available and highly durable.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in AWS CodeArtifact

As a managed service, AWS CodeArtifact is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access CodeArtifact through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.
Working with Amazon VPC endpoints

You can configure CodeArtifact to use an interface virtual private cloud (VPC) endpoint to improve the security of your VPC.

VPC endpoints use AWS PrivateLink, a service that makes it possible for you to access CodeArtifact APIs through private IP addresses. AWS PrivateLink restricts all network traffic between your VPC and CodeArtifact to the AWS network. When you use an interface VPC endpoint, you don’t need an internet gateway, NAT device, or virtual private gateway. For more information, see VPC Endpoints in the Amazon Virtual Private Cloud User Guide.

Important

- VPC endpoints do not support cross-AWS Region requests. Make sure that you create your endpoint in the same AWS Region where you plan to issue your API calls to CodeArtifact.
- VPC endpoints only support Amazon-provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see DHCP Option Sets in the Amazon Virtual Private Cloud User Guide.
- The security group attached to the VPC endpoint must allow incoming connections on port 443 from the private subnet of the VPC.

Topics

- Create VPC endpoints for CodeArtifact (p. 160)
- Create the Amazon S3 gateway endpoint (p. 161)
- Use CodeArtifact from a VPC (p. 163)
- Create a VPC endpoint policy for CodeArtifact (p. 163)

Create VPC endpoints for CodeArtifact

To create virtual private cloud (VPC) endpoints for CodeArtifact, use the Amazon EC2 create-vpc-endpoint AWS CLI command. For more information, see Interface VPC Endpoints (AWS PrivateLink) in the Amazon Virtual Private Cloud User Guide.

Two VPC endpoints are required so that all requests to CodeArtifact are in the AWS network. The first endpoint is used to call CodeArtifact APIs (for example, GetAuthorizationToken and CreateRepository).

```
com.amazonaws.region.codeartifact.api
```

The second endpoint is used to access CodeArtifact repositories using package managers and build tools (for example, npm and Gradle).

```
com.amazonaws.region.codeartifact.repositories
```

The following command creates an endpoint to access CodeArtifact repositories.
Create the Amazon S3 gateway endpoint

CodeArtifact uses Amazon Simple Storage Service (Amazon S3) to store package assets. To pull packages from CodeArtifact, you must create a gateway endpoint for Amazon S3. When your build or deployment process downloads packages from CodeArtifact, it must access CodeArtifact to get package metadata and Amazon S3 to download package assets (for example, Maven .jar files).

To create the Amazon S3 gateway endpoint for CodeArtifact, use the Amazon EC2 create-vpc-endpoint AWS CLI command. When you create the endpoint, you must select the route tables for your VPC. For more information, see Gateway VPC Endpoints in the Amazon Virtual Private Cloud User Guide.

The following command creates an Amazon S3 endpoint.

```bash
aws ec2 create-vpc-endpoint --vpc-id vpcid --service-name com.amazonaws.region.s3 --route-table-ids routetableid
```

Minimum Amazon S3 bucket permissions for AWS CodeArtifact

The Amazon S3 gateway endpoint uses an IAM policy document to limit access to the service. To allow only the minimum Amazon S3 bucket permissions for CodeArtifact, restrict access to the Amazon S3 bucket that CodeArtifact uses when you create the IAM policy document for the endpoint.

The following table describes the Amazon S3 buckets you should reference in your policies to allow access to CodeArtifact in each region.

<table>
<thead>
<tr>
<th>Region</th>
<th>Amazon S3 Bucket ARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-1</td>
<td>arn:aws:s3:::assets-193858265520-us-east-1</td>
</tr>
<tr>
<td>us-east-2</td>
<td>arn:aws:s3:::assets-250872398865-us-east-2</td>
</tr>
<tr>
<td>us-west-2</td>
<td>arn:aws:s3:::assets-787052242323-us-west-2</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>arn:aws:s3:::assets-438097961670-eu-west-1</td>
</tr>
</tbody>
</table>
### Region | Amazon S3 Bucket ARN
--- | ---
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
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

You can use the `aws codeartifact describe-domain` command to fetch the Amazon S3 bucket used by a CodeArtifact domain.

```bash
aws codeartifact describe-domain --domain mydomain
```

```
{
  "domain": {
    "name": "mydomain",
    "owner": "111122223333",
    "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-78705224323-us-west-2"
  }
}
```

### Example

The following example illustrates how to provide access to the Amazon S3 buckets required for CodeArtifact operations in the `us-east-1` region. For other regions, update the `Resource` entry with the correct permission ARN for your region based on the table above.

```json
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
```
Use CodeArtifact from a VPC

Because the `com.amazonaws.region.codeartifact.api` endpoint uses a hostname different from the default hostname used by CodeArtifact, you must override the hostname when you use the CodeArtifact AWS CLI or SDK.

Run the following command to find a VPC endpoint to use to override the hostname.

```bash
$ aws ec2 describe-vpc-endpoints --filters Name=service-name,Values=com.amazonaws.region.codeartifact.api --query 'VpcEndpoints[*].DnsEntries[*].DnsName'
```

The output looks like the following.

```json
[
  [
    "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",
  ]
]
```

In this example, you can use either hostname to override the `com.amazonaws.region.codeartifact.api` endpoint.

If you use the CodeArtifact AWS CLI, pass the hostname to the `--endpoint-url` parameter, as in the following example.

```bash
aws codeartifact login --tool npm --domain mydomain --domain-owner 111122223333 --repository myrepo --endpoint-url VPC_endpoint
```

If you use the SDK, consult your SDK documentation to learn how to override a hostname. How to do this varies by the language that you use.

Create a VPC endpoint policy for CodeArtifact

To create a VPC endpoint policy for CodeArtifact, specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources that can have actions performed on them.

The following example policy specifies that principals in the account 123456789012 can call the `GetAuthorizationToken` API and fetch packages from a CodeArtifact repository.

```json
{
  "Statement": [
    {
      "Resource": ["arn:aws:s3:::assets-193858265520-us-east-1/*"]
    }
  ]
}
```
{
  "Action": [
    "codeartifact:GetAuthorizationToken",
    "codeartifact:GetRepositoryEndpoint",
    "codeartifact:ReadFromRepository",
    "sts:GetServiceBearerToken"
  ],
  "Effect": "Allow",
  "Resource": "*",
  "Principal": {
    "AWS": "arn:aws:iam::123456789012:root"
  }
}
Creating CodeArtifact resources with AWS CloudFormation

CodeArtifact is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want (like domains or repositories), and AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your CodeArtifact resources consistently and repeatedly. Just describe your resources once and then provision the same resources over and over in multiple accounts and AWS Regions.

CodeArtifact and AWS CloudFormation templates

To provision and configure resources for CodeArtifact and related services, you must understand AWS CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see What is AWS CloudFormation Designer? in the AWS CloudFormation User Guide.

CodeArtifact supports creating domains and repositories in AWS CloudFormation. For more information, including examples of JSON and YAML templates for domains and repositories, see AWS::CodeArtifact::Domain and AWS::CodeArtifact::Repository.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- AWS CloudFormation Command Line Interface User Guide
Troubleshooting

The following information might help you troubleshoot common issues with CodeArtifact.

I cannot view notifications

**Problem:** When you are in the Developer Tools console and choose **Notifications** under **Settings**, you see a permissions error.

**Possible fixes:** While notifications are a feature of the Developer Tools console, CodeArtifact does not currently support notifications. None of the managed policies for CodeArtifact include permissions that allow users to view or manage notifications. If you use other services in the Developer Tools console, and those services support notifications, the managed policies for those services include the permissions required to view and manage notifications for those services.
Tagging resources

A tag is a custom attribute label that you or AWS assigns to an AWS resource. Each AWS tag has two parts:

- A tag key (for example, CostCenter, Environment, Project, or Secret). Tag keys are case sensitive.
- An optional field known as a tag value (for example, 111122223333, Production, or a team name). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Together these are known as key-value pairs.

Tags help you identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to a repository that you assign to an AWS CodeBuild project.

For tips on using tags, see the AWS Tagging Strategies post on the AWS Answers blog.

You can tag the following resource types in CodeArtifact:

- Tag a repository in CodeArtifact (p. 32)
- Tag a domain in CodeArtifact (p. 79)

You can use the console, AWS CLI, CodeArtifact APIs, or AWS SDKs to:

- Add tags to a domain or repository when you create it*.
- Add, manage, and remove tags for a domain or repository.

* You cannot add tags to a domain or repository when you create it in the console.

In addition to identifying, organizing, and tracking your resource with tags, you can use tags in IAM policies to help control who can view and interact with your resource. For examples of tag-based access policies, see Using tags to control access to CodeArtifact resources (p. 149)
# Quotas in AWS CodeArtifact

The following table describes resource quotas in CodeArtifact. To view the resource quotas along with the list of service endpoints for CodeArtifact, see [AWS service quotas](https://docs.aws.amazon.com/AmazonWebServices/latest/Glask REFERENCE/) in the Amazon Web Services General Reference.

You can request a service quota increase for the following CodeArtifact resource quotas. For more information about requesting a service quota increase, see [AWS Service Quotas](https://docs.aws.amazon.com/service-quotas/latest/guide/service-quotas-overview.html).

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset file size maximum</td>
<td>1 Gigabytes</td>
<td>No</td>
</tr>
<tr>
<td>Assets per package version maximum</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>CopyPackageVersions maximum requests per second</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>Direct upstream repository maximum</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>Domains per AWS account maximum</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>GetAuthorizationToken maximum requests per second</td>
<td>40</td>
<td>No</td>
</tr>
<tr>
<td>GetPackageVersionAsset maximum requests per second</td>
<td>50</td>
<td>No</td>
</tr>
<tr>
<td>ListPackageVersionAssets maximum requests per second</td>
<td>20</td>
<td>No</td>
</tr>
<tr>
<td>ListPackageVersions maximum requests per second</td>
<td>200</td>
<td>No</td>
</tr>
<tr>
<td>ListPackages maximum requests per second</td>
<td>200</td>
<td>No</td>
</tr>
<tr>
<td>Maximum requests per second using a single authentication token.</td>
<td>800</td>
<td>No</td>
</tr>
<tr>
<td>Repositories per domain maximum</td>
<td>1,000</td>
<td>No</td>
</tr>
<tr>
<td>Repository maximum read requests per second from a single AWS account</td>
<td>800</td>
<td>No</td>
</tr>
<tr>
<td>Repository maximum read requests per second from multiple AWS accounts</td>
<td>800</td>
<td>No</td>
</tr>
<tr>
<td>Repository maximum write requests per second from a single AWS account</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>Repository maximum write requests per second from multiple AWS accounts</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>Requests without authentication token per IP address maximum</td>
<td>600</td>
<td>No</td>
</tr>
<tr>
<td>Upstream repository search maximum</td>
<td>25</td>
<td>No</td>
</tr>
</tbody>
</table>
## AWS CodeArtifact user guide document history

**Latest documentation update:** August 25, 2021

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added in-depth documentation for updating the status of package versions (p. 169)</td>
<td>Expanded the update package version status documentation into its own topic. Added documentation for updating a package version’s status, including required IAM permissions, example AWS CLI commands for various scenarios, and possible errors. See Update package version status (p. 66) for more information.</td>
<td>September 1, 2021</td>
</tr>
<tr>
<td>Updated the copy package versions documentation with more in-depth permissions information (p. 169)</td>
<td>Added more information about the required IAM and resource-based policy permissions for calling the <code>aws codeartifact copy-package-versions</code> command to copy package versions from one repository to another within the same domain in CodeArtifact. Along with more information, there are now examples of the required resource-based policies for the source and destination repository. See Required IAM permissions to copy packages (p. 58) for more information.</td>
<td>August 25, 2021</td>
</tr>
<tr>
<td>Updated documentation for running a Gradle build in IntelliJ IDEA (p. 169)</td>
<td>Updated the documentation for running a Gradle build in IntelliJ IDEA with steps for configuring Gradle to fetch plugins from CodeArtifact. Also added an option to create a new CodeArtifact authorization token for each new run with an inline call to <code>aws codeartifact get-authorization-token</code>. See Run a Gradle build in IntelliJ IDEA (p. 103) for more information.</td>
<td>August 23, 2021</td>
</tr>
<tr>
<td>Added documentation for configuring and using Yarn with AWS CodeArtifact (p. 169)</td>
<td>Added documentation for configuring and using Yarn 1.X and Yarn 2.X to manage npm packages with CodeArtifact. See Configure and use Yarn with CodeArtifact (p. 87) for more information.</td>
<td>July 30, 2021</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>AWS CodeArtifact now supports NuGet packages (p. 169)</td>
<td>CodeArtifact users can now publish and consume NuGet packages. Added documentation for configuring and using both Visual Studio and NuGet command line tools like <code>nuget</code> and <code>dotnet</code> with CodeArtifact repositories. See Using CodeArtifact with NuGet (p. 113) for more information.</td>
<td>November 19, 2020</td>
</tr>
<tr>
<td>Tagging resources in AWS CodeArtifact (p. 169)</td>
<td>Added documentation about tagging repositories and domains in AWS CodeArtifact. See Tagging resources (p. 167).</td>
<td>October 30, 2020</td>
</tr>
<tr>
<td>CodeArtifact now supports AWS CloudFormation (p. 169)</td>
<td>CodeArtifact users can now use AWS CloudFormation templates to create CodeArtifact repositories and domains. See Creating CodeArtifact resources with AWS CloudFormation (p. 165) for more information and to get started.</td>
<td>October 8, 2020</td>
</tr>
<tr>
<td>Add information about creating Amazon S3 gateway endpoints to use CodeArtifact with Amazon VPC (p. 169)</td>
<td>Added information about creating Amazon S3 gateway endpoints with the Amazon EC2 AWS CLI command. This documentation also contains information about the specific permissions that CodeArtifact requires to be used with Amazon VPC environments. See Create the Amazon S3 gateway endpoint (p. 161).</td>
<td>August 12, 2020</td>
</tr>
<tr>
<td>Publishing Maven artifacts with curl and publishing third-party Maven artifacts (p. 169)</td>
<td>Added guidance for Publishing with curl (p. 109) and Publish third-party artifacts (p. 108).</td>
<td>August 10, 2020</td>
</tr>
</tbody>
</table>