CodeArtifact User Guide

CodeArtifact

CodeArtifact: CodeArtifact User Guide

Table of Contents

What is AWS CodeArtifact?

AWS CodeArtifact is a secure, highly scalable, managed artifact repository service that helps organizations to store and share software packages for application development. You can use CodeArtifact with popular build tools and package managers such as the NuGet CLI, Maven, Gradle, npm, yarn, pip, and twine. CodeArtifact helps reduce the need for you to manage your own artifact storage system or worry about scaling its infrastructure. There are no limits on the number or total size of the packages that you can store in a CodeArtifact repository.

You can create a connection between your private CodeArtifact repository and an external, public repository, such as npmjs.com or Maven Central. CodeArtifact will then fetch and store packages on demand from the public repository when they're requested by a package manager. This makes it more convenient to consume open-source dependencies used by your application and helps ensure they're always available for builds and development. You can also publish private packages to a CodeArtifact repository. This helps you share proprietary software components between multiple applications and development teams in your organization.

For more information, see AWS CodeArtifact.

How does CodeArtifact work?

CodeArtifact stores software packages in repositories. Repositories are polyglot—a single repository can contain packages of any supported type. 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, you might use each repository 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 the 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 Connect a CodeArtifact repository to a public repository.

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.

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. Packages in CodeArtifact repositories cannot be made publicly available. For more information about authentication and access in CodeArtifact, see AWS CodeArtifact authentication and tokens.

AWS CodeArtifact concepts

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

Topics

Asset

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

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.

Using a domain, you can apply an organizational policy across multiple repositories. With this approach, you determine which accounts can access repositories in the domain, and which public repositories can be used as the sources of packages.

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

Repository

A CodeArtifact *repository* contains a set of package versions, each of which maps to a set of assets. 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 1,000 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 such as `@types` in `@types/node`, a set of package versions, and package-level metadata such as npm tags.

AWS CodeArtifact supports Cargo, generic, Maven, npm, NuGet, PyPI, Ruby, Swift package formats.

Package group

Package groups can be used to apply configuration to multiple packages that match a defined pattern using package format, package namespace, and package name. You can use package groups to more conveniently configure package origin controls for multiple packages. Package origin controls are used to block or allow ingestion or publishing of new package versions, which protects users from malicious actions known as dependency substitution attacks.

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. For more information, see the npm scopes documentation. 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`. For generic packages, a namespace is required. 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.

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. This approach effectively merges the contents of the two repositories from the point of view of a client. Using CodeArtifact, you can create an upstream relationship between two repositories.

How do I get started with CodeArtifact?

We recommend that you complete the following steps:

1. **Learn** more about CodeArtifact by reading AWS CodeArtifact concepts.
2. **Set up** your AWS account, the AWS CLI, and an IAM user by following the steps in Setting up with AWS CodeArtifact.

3. **Use** CodeArtifact by following the instructions in Getting started with CodeArtifact.

Setting up with AWS CodeArtifact

If you've already signed up for Amazon Web Services (AWS), you can start using AWS 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

When you sign up for Amazon Web Services (AWS), you are charged only for the services and resources that you use, including AWS CodeArtifact.

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

To create an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
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.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

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.

   ```
   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 AWS 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.

```
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.

Provision an IAM user

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

To provision anIAM 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. For more information and example policies, see Identity-based policy examples for AWS CodeArtifact.
    - **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*.

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

```
         "Effect": "Allow",
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
            "Condition": {
               "StringEquals": {
                  "sts:AWSServiceName": "codeartifact.amazonaws.com"
               }
            }
      }
    ]
}
```

Note

The `sts:GetServiceBearerToken` permission must be added to the IAM user or role policy. While it can be added to a CodeArtifact domain or repository resource policy, the permission will have no effect in resource policies.

The `sts:GetServiceBearerToken` permission is required to call the CodeArtifact `GetAuthorizationToken` API. This API returns a token that must be used when using a package manager such as `npm` or `pip` with CodeArtifact. To use a package manager with a CodeArtifact repository, your IAM user or role must allow `sts:GetServiceBearerToken` as shown in the preceding policy example.

If you haven't installed the package manager or build tool that you plan to use with CodeArtifact, see Install your package manager or build tool.

Install your package manager or build tool

To publish or consume packages from CodeArtifact, you must use a package manager. There are different package managers for each package type. The following list contains some package managers that you can use with CodeArtifact. If you haven't already, install the package managers for the package type you want to use.

- For npm, use the npm CLI or pnpm.
- For Maven, use either Apache Maven (`mvn`) or Gradle.
- For Python, use pip to install packages and twine to publish packages.
- For NuGet, use the Toolkit for Visual Studio in Visual Studio or the nuget or dotnet CLIs.
- For generic packages, use the AWS CLI or SDK to publish and download package contents.

Next steps

Your next steps will depend on which package type or types you are using with CodeArtifact, and the state of your CodeArtifact resources.

If you are getting started with CodeArtifact for the first time for yourself, your team, or organization, see the following documentation for general getting started information and help creating the resources you will need.

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

If your resources have already been created and you are ready to configure your package manager to push packages to or install packages from a CodeArtifact repository, see the documentation that corresponds to your package type or package manager.

- Using CodeArtifact with npm
- Using CodeArtifact with Python
- Using CodeArtifact with Maven
- Using CodeArtifact with NuGet
- Using CodeArtifact with generic packages

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.

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

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.
- 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.
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**.
12. In **Review and create**, review what CodeArtifact is creating for you.

    - **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**.

    ```
    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
    ```

    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

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

    ```
    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. 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.

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

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

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

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

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.

```
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.

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

3. 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",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/npm-store",
        "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",
           "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
   domain/npm-store",
           "upstreams": [],
           "externalConnections": [
               {
                   "externalConnectionName": "public:npmjs",
                   "packageFormat": "npm",
                   "status": "AVAILABLE"
               }
           ]
       }
   }
   ```

For more information, see Connect a CodeArtifact repository to a public repository.

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.

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

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

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

```
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.

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

```
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 the `npm install` install command completes and when the package is visible in your repository. For details on typical latency when fetching packages from public repositories, see External connection latency.

```
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.

```
{
    "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.

```
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.

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

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

```
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.

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

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

Working with repositories in CodeArtifact

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

Topics

Create a repository

Because all packages in CodeArtifact are stored in repositories, to use CodeArtifact, you must create one. You can create a repository using the CodeArtifact console, the AWS Command Line Interface (AWS CLI), or AWS CloudFormation. Each repository is associated with the AWS account that you use when you create it. You can have multiple repositories, and they are created and grouped in domains. When you create a repository, it does not contain any packages. Repositories are polyglot, which means that a single repository can contain packages of any supported type.

For information about CodeArtifact service limits, such as the maximum number of allowed repositories in a single domain, see Quotas in AWS CodeArtifact. If you hit the maximum number of allowed repositories, 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.

For more information about managing CodeArtifact repositories with CloudFormation, see Creating CodeArtifact resources with AWS CloudFormation.

Note

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

Topics

- Create a repository (console)
- Create a repository (AWS CLI)
- Create a repository with an upstream repository

Create a repository (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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:

> **Important**
>
> CodeArtifact only supports symmetric KMS keys. You cannot use an asymmetric KMS key to encrypt your CodeArtifact domains. For help determining whether a KMS key is symmetric or asymmetric, see Identifying symmetric and asymmetric KMS keys.

- 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.

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

Example output:

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

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.

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",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo",
        "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 or remove upstream repositories (console) and Add or remove upstream repositories (AWS CLI).

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.
- 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 List repositories.
- Use a repository in a different AWS account. For more information, see Repository policies.

Use a package manager client

After you know which repository you want to use, see one of the following topics.

- Using CodeArtifact with Maven
- Using CodeArtifact with npm
- Using CodeArtifact with NuGet

- Using CodeArtifact with Python

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.

> ⚠ **Important**
>
> Deleting a repository cannot be undone. After you delete a repository, you are no longer able to recover it and it cannot be restored.

Topics

- Delete a repository (console)
- Delete a repository (AWS CLI)
- Protect repositories from being deleted

Delete a repository (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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.

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

Example output:

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

Protect repositories from being deleted

You can prevent a repository from being accidentally deleted by including a domain policy similar to the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyRepositoryDeletion",
            "Action": [
                "codeartifact:DeleteRepository"
            ],
            "Effect": "Deny",
            "Resource": "*",
            "Principal": *
        }
    ]
}
```

This policy prevents all principals from deleting the repository, but if you decide later that you need to delete the repository, you can do so by following these steps:

1. In the domain policy, update the policy to the following:

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

```
    "Statement": [
        {
            "Sid": "DenyRepositoryDeletion",
            "Action": [
                "codeartifact:DeleteRepository"
            ],
            "Effect": "Deny",
            "NotResource": "repository-arn",
            "Principal": *
        }
    ]
}
```

Replace *`repository-arn`* with the ARN of the repository that you would like to delete.

2. In the AWS CodeArtifact console, choose **Repositories** and delete your chosen repository.
3. After you've deleted the repository, you can change the policy back to prevent acccidental delections.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyRepositoryDeletion",
            "Action": [
                "codeartifact:DeleteRepository"
            ],
            "Effect": "Deny",
            "Resource": "*",
            "Principal": *
        }
    ]
}
```

Alternatively, you can include the same deny statement in a repository policy. This allow you to have more flexibility to protect high-value repositories from deletion.

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",
            "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain2/repo3",
            "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": [
        {
            "name": "repo1",
            "administratorAccount": "123456789012",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo1",
            "description": "Description of repo1"
        },
        {
            "name": "repo2",
            "administratorAccount": "444455556666",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo2",
            "description": "Description of repo2"
        },
        {
            "name": "repo3",
            "administratorAccount": "444455556666",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo3",
            "description": "Description of repo3"
        }
    ]
}
```

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
```

For more information about the `--query` parameter, see ListRepositories in the *CodeArtifact API Reference*.

View or modify a repository configuration

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

View or modify a repository configuration (console)

You can view details about and update your repository using the CodeArtifact console.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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 Connect a CodeArtifact repository to a public repository.
   - 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.

> **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.
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.

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:

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

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:

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

Note

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 []
```

Example output:

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

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 `*`. In order for a package manager to download a package from this repository, a domain policy for cross-account access will also need to be created. The domain policy must grant at least `codeartifact:GetAuthorizationToken` permission to the principal. For an example of a full domain policy for granting cross-account access, see this [Domain policy example](#).

> **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 actions on them (for example, listing package names and versions). Typically, you grant permissions in the following policy in addition to `ReadFromRepository` because a user who downloads packages from a repository needs to interact with it in other ways too.

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

Set a policy

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

```
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:

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

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 `put-repository-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 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 policy.document` together with the Python `json.tool` module.

```
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:

```
{
    "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.

```
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`.

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

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

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

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

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 `my_repo` repository in domain `my_domain`.

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

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

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

> **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
> 2. Maven: `codeartifact:PutPackageMetadata`

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 `my_repo` repository.

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

Interaction between repository and domain policies

CodeArtifact supports resource policies on domains and repositories. Resource policies are optional. Each domain may have one policy and each repository in the domain may have its own repository policy. If both a domain policy and a repository policy are present, then both are evaluated when determining whether a request to a CodeArtifact repository is allowed or denied. Domain and repository policies are evaluating using the following rules:

- No resource policies are evaluated when performing account-level operations such as ListDomains or ListRepositories.
- No repository policies are evaluated when performing domain-level operations such as DescribeDomain or ListRepositoriesInDomain.
- The domain policy is not evaluated when performing PutDomainPermissionsPolicy. Note that this rule prevents lock-outs.
- The domain policy is evaluated when performing PutRepositoryPermissionsPolicy, but the repository policy is not evaluated.
- An explicit deny in any policy overrides an allow in another policy.
- An explicit allow is only required in one resource policy. Omitting an action from a repository policy will not result in an implicit deny if the domain policy allows the action.
- When no resource policy allows an action, the result is an implicit deny unless the calling principal's account is the domain owner or repository administrator account and an identity-based policy allows the action.

Resource policies are optional when used to grant access in a single account scenario, where the caller account used to access a repository is the same as the domain owner and repository administrator account. Resource policies are required to grant access in a cross-account scenario where the caller's account is not the same as the domain owner or repository administrator account. Cross-account access in CodeArtifact follows the general IAM rules for cross-account access as described in Determining whether a cross-account request is allowed in the *IAM User Guide*.

- A principal in the domain owner account may be granted access to any repository in the domain through an identity-based policy. Note that in this case, no explicit allow is required in a domain or repository policy.

- A principal in the domain owner account may be granted access to any repository through a domain or repository policy. Note that in this case, no explicit allow is required in an identity-based policy.
- A principal in the repository administrator account may be granted access to the repository through an identity-based policy. Note that in this case, no explicit allow is required in a domain or repository policy.
- A principal in another account is only granted access when allowed by at least one resource policy and at least one identity-based policy, with no policy explicitly denying the action.

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.

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)

You can use the CLI to manage repository tags.

Topics

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.

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:aws:codeartifact:us-west-2:111122223333:repository/my_domain/my_repo` ARN value:

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

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)

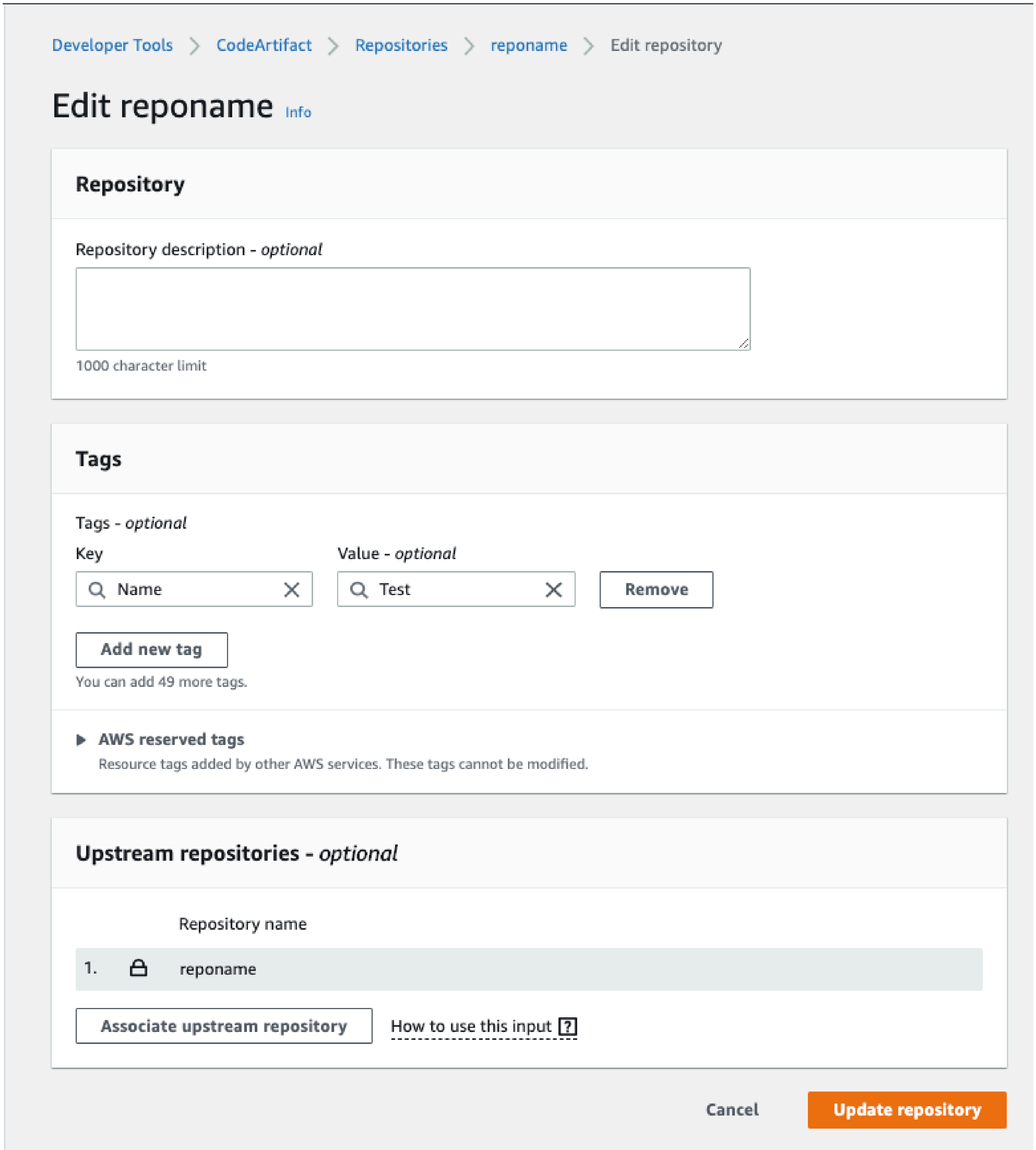
You can use the console or the CLI to tag resources.

Topics

Add tags to a repository (console)

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

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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`.



Developer Tools > CodeArtifact > Repositories > reponame > Edit repository
Info
Remove
Add new tag
Associate upstream repository
How to use this input
Cancel
Update repository

Edit reponame

Repository

Repository description - *optional*

1000 character limit

Tags

Tags - *optional*

Key	Value - *optional*
Name	Test

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
1.	reponame

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.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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 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 and Add or remove upstream repositories.

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. For more information about requesting packages from upstream repositories or external connections, see Requesting a package version with upstream repositories or Requesting packages from external connections.

Topics

What's the difference between upstream repositories and external connections?

In CodeArtifact, upstream repositories and external connections behave mostly the same, but there are a few important differences.

1. You can add up to 10 upstream repositories to a CodeArtifact repository. You can only add one external connection.
2. There are separate API calls to add an upstream repository or an external connection.
3. The package retention behavior is slightly different, as packages requested from upstream repositories are retained in those repositories. For more information, see Package retention in intermediate repositories.

Add or remove upstream repositories

Follow the steps in the following sections to add or remove upstream repositories to or from an CodeArtifact repository. For more information about upstream repositories, see Working with upstream repositories in CodeArtifact.

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, Nuget Gallery, Maven Central, or PyPI, see Add an external connection.

Add or remove upstream repositories (console)

Perform the steps in the following procedure to add a repository as an upstream repository using the CodeArtifact console. For information about adding an upstream repository with the AWS CLI, see Add or remove upstream repositories (AWS CLI).

To add an upstream repository using the CodeArtifact console

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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**, choose **Associate upstream repository** and add the repository you want to add as an upstream repository. You can only add repositories in the same domain as upstream repositories.
6. Choose **Update repository**.

To remove an upstream repository using the CodeArtifact console

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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**, find the list entry of the upstream repository you want to remove and choose **Disassociate**.

   > ⚠ **Important**
   >
   > Once you remove an upstream repository from a CodeArtifact repository, package managers will not have access to packages in the upstream repository or any of its upstream repositories.

6. Choose **Update repository**.

Add or remove upstream repositories (AWS CLI)

You can add 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.

You can only add repositories in the same domain as upstream repositories.

To add upstream repositories (AWS CLI)

1. If you haven't, follow the steps in Setting up with AWS CodeArtifact to set up and configure the AWS CLI with CodeArtifact.
2. Use the `aws codeartifact update-repository` command with the `--upstreams` flag to add upstream repositories.

   > ⓘ **Note**
   >
   > Calling the `update-repository` command replaces the existing configured upstream repositories with the list of repositories provided with the `--upstreams` flag. If you

> want to add upstream repositories and keep the existing ones, you must include the existing upstream repositories in the call.

The following example 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.

For information about connecting to public, external repositories such as npmjs.com or Maven Central, see Connect a CodeArtifact repository to a public repository.

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

To remove an upstream repository (AWS CLI)

1. If you haven't, follow the steps in Setting up with AWS CodeArtifact to set up and configure the AWS CLI with CodeArtifact.
2. To remove upstream repositories from a CodeArtifact repository, use the `update-repository` command with the `--upstreams` flag. The list of repositories provided to the command will be the new set of upstream repositories for the CodeArtifact repository. Include existing upstream repositories that you want to keep, and omit the upstream repositories you want to remove.

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

Connect a CodeArtifact repository to a public repository

You can add a external 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.

In CodeArtifact, the intended way to use external connections is to have one repository per domain with an external connection to a given public repository. For example, if you want to connect to npmjs.com, configure one repository in your domain with an external connection to npmjs.com and configure all the other repositories with an upstream to it. This way, all the repositories can make use of the packages that have already been fetched from npmjs.com, rather than fetching and storing them again.

Topics

Connect to an external repository (console)

When you use the console to add a connection to an external repository, the following will occur:

1. A `-store` repository for the external repository will be created in your CodeArtifact domain if one does not exist already. These `-store` repositories behave as intermediate repositories between your repository and the external repository and allow you to connect to more than one external repository.
2. The appropriate `-store` repository is added as an upstream to your repository.

The following list contains each `-store` repository in CodeArtifact and the respective external repository they connect to.

1. `cargo-store` is connected to crates.io.
2. `clojars-store` is connected to Clojars Repository.

3. `commonsware-store` is connected to CommonsWare Android Repository.
4. `google-android-store` is connected to Google Android.
5. `gradle-plugins-store` is connected to Gradle plugins.
6. `maven-central-store` is connected to Maven Central Repository.
7. `npm-store` is connected to npmjs.com.
8. `nuget-store` is connected to nuget.org.
9. `pypi-store` is connected to the Python Packaging Authority.
10. `rubygems-store` is connected to RubyGems.org.

To connect to an external repository (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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**, choose **Associate upstream repository** and add the appropriate `-store` repository that is connected as an upstream.
6. Choose **Update repository**.

After the `-store` repository is added as an upstream repository, package managers connected to your CodeArtifact repository can fetch packages from the respective external repository.

Connect to an external repository (CLI)

You can use the AWS CLI to connect your CodeArtifact repository to an external repository by adding an external connection directly to the repository. This will allow users connected to the CodeArtifact repository, or any of its downstream repositories, to fetch packages from the configured external repository. Each CodeArtifact repository can only have one external connection.

It is recommended to have one repository per domain with an external connection to a given public repository. To connect other repositories to the public repository, add the repository with the external connection as an upstream to them. If you or someone else in your domain has already configured external connections in the console, your domain likely already has a `-store` repository with an external connection to the public repository you want to connect to. For more information about `-store` repositories and connecting with the console, see Connect to an external repository (console).

To add an external connection to a CodeArtifact repository (CLI)

- Use `associate-external-connection` to add an external connection. The following example connects a repository to the npm public registry, npmjs.com. For a list of supported external repositories, see Supported external connection repositories.

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

  Example output:

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

After adding an external connection, see Requesting packages from external connections for information about requesting packages from an external repository with an external connection.

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.

Repository type	Description	Name
Maven	Clojars repository	`public:maven-clojars`
Maven	CommonsWare Android repository	`public:maven-commonsware`
Maven	Google Android repository	`public:maven-googleandroid`
Maven	Gradle plugins repository	`public:maven-gradleplugins`
Maven	Maven Central	`public:maven-central`
npm	npm public registry	`public:npmjs`
NuGet	NuGet Gallery	`public:nuget-org`
Python	Python Package Index	`public:pypi`
Ruby	RubyGems.org	`public:ruby-gems-org`
Rust	Crates.io	`public:crates-io`

Remove an external connection (CLI)

To remove an external connection that was added by using the `associate-external-connection` command in the AWS CLI, use `disassociate-external-connection`.

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

Example output:

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

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.

The maximum number of direct upstream repositories allowed for one repository is 10. Because direct upstream repositories can also have direct upstream repositories of their own, CodeArtifact can search more than 10 repositories for package versions. The maximum number of repositories CodeArtifact looks in when a package version is requested is 25.

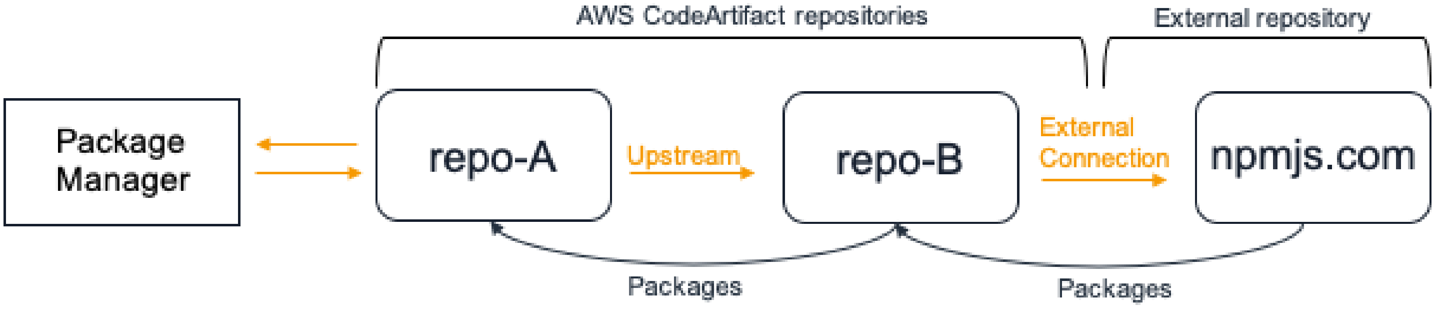
Package retention from upstream repositories

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`.

```
$ npm config get registry
https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my-
downstream-repo/
```

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

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`.

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

Example output:

```
{
    "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.

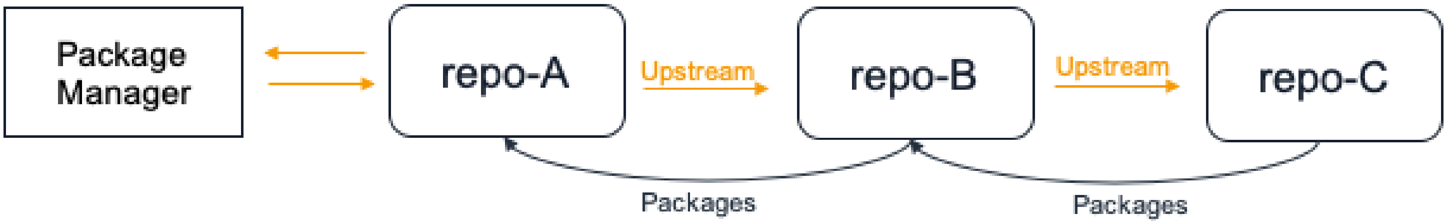
```
aws codeartifact list-package-versions --repository repo-B --domain my_domain \
            --domain-owner 111122223333 --format npm --package lodash --max-results 5
```

Example output:

```
{
    "package": "lodash",
    "format": "npm",
    "versions": [
        {
            "version": "0.10.0",
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.2",
            "revision": "REVISION-2-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.0",
            "revision": "REVISION-3-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.1",
            "revision": "REVISION-4-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.1.0",
            "revision": "REVISION-5-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    ],
    "nextToken": "eyJsaXN0UGFja2FnZVZlcnNpb25zVG9rZW4iOiIwLjIuMiJ9"
}
```

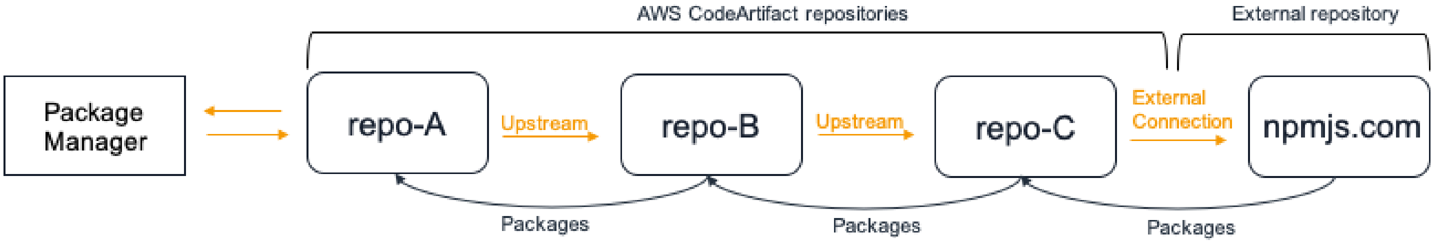
Package retention in intermediate repositories

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 followng 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.

Requesting packages from external connections

The following sections describe how to request a package from an external connection and expected CodeArtifact behavior when requesting a package.

Topics

- Fetch packages from an external connection
- External connection latency
- CodeArtifact behavior when an external repository is not available
- Availability of new package versions
- Importing package versions with more than one asset

Fetch packages from an external connection

To fetch packages from an external connection once you've added it to your CodeArtifact repository as described in Connect a CodeArtifact repository to a public repository, configure your package manager to use your repository and install the packages.

> **Note**
>
> The following instructions use npm, to view configuration and usage instructions for other package types, see Using CodeArtifact with Maven, Using CodeArtifact with NuGet, or Using CodeArtifact with Python.

To fetch packages from an external connection

1. Configure and authenticate your package manager with your CodeArtifact repository. For npm, use the following `aws codeartifact login` command.

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

2. Request the package from the public repository. For npm, use the following `npm install` command, replacing *lodash* with the package you want to install.

```
npm install lodash
```

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

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

Example output:

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

The `list-package-versions` command lists all versions of the package copied into your CodeArtifact repository.

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

Example output:

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

External connection latency

When fetching a package from a public repository using an external connection, there is a delay from when the package is fetched from the public repository and when it is stored in your CodeArtifact repository. For example, say you have installed version 1.2.5 of the npm package "lodash" as described in Fetch packages from an external connection. Although the `npm install lodash` lodash command completed successfully, the package version might not appear in your CodeArtifact repository yet. It typically takes around 3 minutes for the package version to appear in your repository, although occasionally it can take longer.

Because of this latency, you might have successfully retrieved a package version, but might not yet be able to see the version in your repository in the CodeArtifact console or when calling the ListPackages and ListPackageVersions API operations. Once CodeArtifact has asynchronously persisted the package version, it will be visible in the console and via API requests.

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. For more information, see Packages overview. However, in rare cases, an external repository outage might mean that CodeArtifact does not have up-to-date information on which packages and package versions are present in an external repository. In this case, CodeArtifact might permit a package version to be published that it would normally deny.

Availability of new package versions

For a package version in a public repository such as npmjs.com 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, Python, and Nuget packages, there may be a delay of up to 30 minutes from when a new package version is published to npmjs.com, pypi.org, or nuget.org and when it is available for installation from a CodeArtifact repository. CodeArtifact automatically synchronizes metadata from these two repositories to ensure that the cache is up to date.

For 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.

For Maven packages 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 be added to the cache and will be available for download.

Importing package versions with more than one asset

Both Maven and Python packages can have multiple assets per package version. This makes importing packages of these formats more complex than npm and NuGet packages, which only have one asset per package version. For descriptions of which assets are imported for these package types and how newly-added assets are made available, see Requesting Python packages from upstreams and external connections and Requesting Maven packages from upstreams and external connections.

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 or remove upstream repositories (AWS CLI).

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

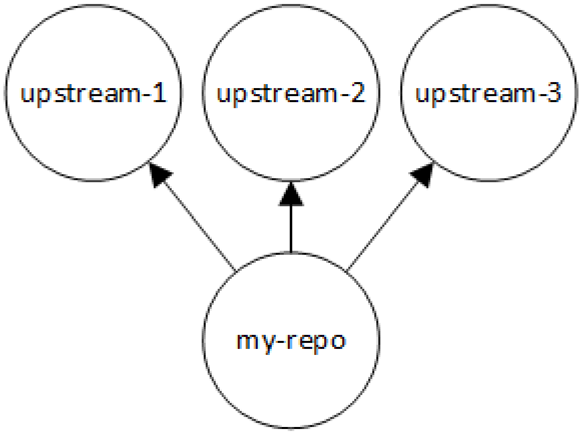
```
aws codeartifact describe-repository --repository my_repo --domain my_domain --domain-
owner 111122223333
```

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

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

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`

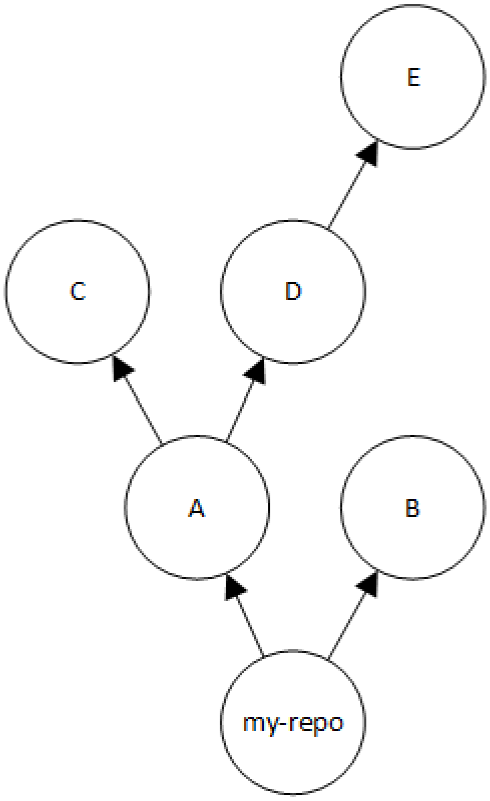
If the package version is found, the search stops, even if it didn't look in all upstream repositories. For example, if the package version is found in `upstream-1`, the search stops and CodeArtifact doesn't look in `upstream-2` or `upstream-3`.

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:

```
{
    "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"
    }
```

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

```
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.

Working with packages in CodeArtifact

The following topics show you how to perform actions on packages using the CodeArtifact CLI and API.

Topics

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 such as `@types` in `@types/node`, a set of package versions, and package-level metadata such as npm tags.

Contents

Supported package formats

AWS CodeArtifact supports Cargo, generic, Maven, npm, NuGet, PyPI, Ruby, Swift package formats.

Package publishing

You can publish new versions of any supported package format 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:

- **Cargo:** `codeartifact:PublishPackageVersion`
- **generic:** `codeartifact:PublishPackageVersion`
- **Maven:** `codeartifact:PublishPackageVersion` and `codeartifact:PutPackageMetadata`
- **npm:** `codeartifact:PublishPackageVersion`
- **NuGet:** `codeartifact:PublishPackageVersion` and `codeartifact:ReadFromRepository`
- **Python:** `codeartifact:PublishPackageVersion`
- **Ruby:** `codeartifact:PublishPackageVersion`
- **Swift:** `codeartifact:PublishPackageVersion`

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

For more information about permissions in CodeArtifact, see AWS CodeArtifact permissions reference.

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 only publish that asset again if 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 (generic, PyPI and Maven), you can add new assets with different names to an existing package version, assuming that you have the required permissions. For generic packages, you can add new assets as long as the package version is in the `Unfinished` state. 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.

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.

Asset size limits for publishing

The maximum size of a package asset that can be published is limited by the **Asset file size maximum** quota shown in Quotas in AWS CodeArtifact. For example, you cannot publish a Maven JAR or Python wheel larger than your current asset file size maximum quota. If you need to store larger assets in CodeArtifact, request a quota increase.

In addition to the asset file size maximum quota, the maximum size of a publishing request for npm packages is 2 GB. This limit is independent of the asset file size maximum quota and cannot be raised with a quota increase. In an npm publishing request (HTTP PUT), package metadata and the content of the npm package tar archive are bundled together. Because of this, the actual maximum size of an npm package that can be published varies and depends on the size of the included metadata.

Note

Published npm packages are limited to a maximum size less than 2 GB.

Publishing latency

Package versions published to a CodeArtifact repository are often available for download in less than one second. For example, if you publish an npm package version to CodeArtifact with `npm publish`, that version should be available to an `npm install` command in less than one second. However, publishing can be inconsistent and can sometimes take longer. If you must use a package version immediately after publishing, use retries to make sure that the download is reliable. For example, after publishing the package version, repeat the download up to three times if the just-published package version is not initially available on the first download attempt.

Note

Importing a package version from a public repository typically takes longer than publishing. For more information, see External connection latency.

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 in the AWS CLI and SDK. For more information, see Update package version status.

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 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 client has uploaded one or more assets for a package version, but has not finalized it by moving it into the `Published` state. Currently only generic and Maven package versions can have a status of `Unfinished`. For Maven packages, 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 Maven package version is **Unfinished**, it will not be included in version lists returned to clients such `mvn` or `gradle`, so it cannot be used as part of a build. Generic packages can be deliberately kept in the `Unfinished` state by providing the `unfinished` flag when calling the PublishPackageVersion API. A generic package can be changed to the `Published` state by omitting the `unfinished` flag, or by calling the UpdatePackageVersionsStatus API.
- **Unlisted** – The package version's assets are available for download from the repository, but the package version is 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 be included in the list of versions returned to package managers. Because 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. After a package version has been marked as **Disposed**, you will no longer be billed for storage of the package assets.

Package versions of all statuses will be returned by default when calling list-package-versions with no `--status` parameter.

Apart from the states listed previously, a package version can also be deleted with the DeletePackageVersions API. After being 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. After a package version has been deleted, you will no longer be billed for storage of that package version's assets.

Package name, package version, and asset name normalization

CodeArtifact normalizes package names, package versions, and asset names before storing them, which means the names or versions in CodeArtifact may be different than the name or version provided when the package was published. For more information about how names and versions are normalized in CodeArtifact for each package type, see the following documentation:

- Python package name normalization
- NuGet package name, version, and asset name normalization

CodeArtifact does not perform normalization on other package formats.

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.

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

Sample output:

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

List npm package names

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

```
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`*.

Note

The `--namespace` option filters by namespace prefix. Any npm package with a scope that starts with the value passed to the `--namespace` option will be returned in the `list-packages` response.

```
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 ID in the `--namespace` option.

> **Note**
>
> The `--namespace` option filters by namespace prefix. Any npm package with a scope that starts with the value passed to the `--namespace` option will be returned in the `list-packages` response.

```
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"

        },
        {
            "package": "commons-compress",
            "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. Searching for all npm packages with a scope that starts with `@types` 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.

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

Sample output:

```
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.

```
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.

```
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 AWS CodeArtifact to get a list of all of the versions of a package name in a repository.

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

Sample output:

```
{
  "defaultDisplayVersion": "1.0.1",
  "format": "npm",
  "package": "kind-of",
  "versions": [
      {
          "version": "1.0.1",
          "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      },
      {
          "version": "1.0.0",
          "revision": "REVISION-SAMPLE-2-C752BEEF6D2CFC",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      },
      {
          "version": "0.1.2",
          "revision": "REVISION-SAMPLE-3-654S65A5C5E1FC",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      },
      {
          "version": "0.1.1",
          "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      },
      {
          "version": "0.1.0",
          "revision": "REVISION-SAMPLE-4-AF669139B772FC",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      }
  ]
}
```

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

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.

List npm package versions

To list all the package versions for an npm package, set the value of the `--format` option to `npm`.

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

To list npm package versions in a specific namespace (npm *scope*), use the `--namespace` option. 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-package-versions --package my_package --domain my_domain \
 --domain-owner 111122223333 --repository my_repo --format npm \
 --namespace types
```

List Maven package versions

To list all the package versions for a Maven package, set the value of the `--format` option to `maven`. You must also specify the Maven group ID in the `--namespace` option.

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

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 `PUBLISHED_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 PUBLISHED_TIME
```

Sample output:

```
{

  "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 generic, 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 the `--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
3.0.0
```

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

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`.

```
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.

```
{
    "format": "pypi",
    "package": "flatten-json",
    "version": "0.1.7",
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
    "assets": [
        {
            "name": "flatten_json-0.1.7-py3-none-any.whl",
            "size": 31520,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f086
SHA-512"
            }
        },
        {
            "name": "flatten_json-0.1.7.tar.gz",
            "size": 2865,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f086
SHA-512"
            }
        }
    ]
}
```

List assets of an npm package

An npm package always has a single asset with a name of `package.tgz`. To list the assets of a scoped npm package, include the scope in the `--namespace` option.

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

List assets of a Maven package

To list the assets of a Maven package, include the package namespace in the `--namespace` option. To list the assets of the Maven package `commons-cli:commons-cli`:

```
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
```

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. 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 111122223333 --repository my_repo \
    --format maven --namespace com.google.guava --package guava --package-version 27.1-
jre \
    --asset guava-27.1-jre.jar \
    guava-27.1-jre.jar
```

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

The output of the command will be:

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

> **Note**
>
> To download assets from a scoped npm package, include the scope in the `--namespace` option. The `@` symbol must be omitted when using `--namespace`. For example, if the scope is `@types`, use `--namespace types`.

Downloading assets using `get-package-version-asset` requires `codeartifact:GetPackageVersionAsset` permission on the package resource. For more information about resource-based permission policies, see Resource-based policies in the *AWS Identity and Access Management User Guide*.

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.

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. If a policy does exist, you may want to add these permissions to it instead of replacing it.

Example source repository permissions policy

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

Example destination repository permissions policy

```
{
    "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.

```
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.

```
{
   "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](#).

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`.

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

Copy Maven package versions

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

```
 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.

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

```
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.

```
{
    "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.

```
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.

```
{
    "successfulVersions": {
        "0.2.0": {
            "revision": "Yad+B1QcBq2kdEVrx1E1vSfHJVh8Pr61hBUkoWPGWX0=",
            "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.

```
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.

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

Version 0.2.1 is marked as `SKIPPED` because it was not copied to the destination repository. Is was not copied because the copy of version 0.3.2 failed because it was already present in the destination repository, but not identical in the source and destination repositories.

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 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.

Delete a package or package version

You can delete one or more package versions at a time using the `delete-package-versions` command. To remove a package from a repository completely, including all associated versions and configuration, use the `delete-package` command. A package can exist in a repository without any package versions. This can happen when all versions are deleted using the `delete-package-versions` command, or if the package was created without any versions using the `put-package-origin-configuration` API operation (see Editing package origin controls).

Topics

- Deleting a package (AWS CLI)

- Deleting a package (console)
- Deleting a package version (AWS CLI)
- Deleting a package version (console)
- Deleting an npm package or package version
- Deleting a Maven package or package version
- Best practices for deleting packages or package versions

Deleting a package (AWS CLI)

You can delete a package, including all of its package versions and configuration, using the `delete-package` command. The following example deletes the PyPI package named `my-package` in the repo `my_repo` in the `my_domain` domain:

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

Sample output:

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

You can confirm that the package was deleted by running `describe-package` for the same package name:

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

Deleting a package (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
2. In the navigation pane, choose **Repositories**.
3. Choose the **Repository** from which you want to delete a package.
4. Choose the **Package** you want to delete.
5. Choose **Delete Package**.

Deleting a package version (AWS CLI)

You can delete one or more package versions at a time using the `delete-package-versions` command. The following example deletes versions `4.0.0`, `4.0.1`, and `5.0.0` of the PyPI 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 pypi \
--package my-package --versions 4.0.0 4.0.1 5.0.0
```

Sample output:

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

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

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

Deleting a package version (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
2. In the navigation pane, choose **Repositories**.
3. Choose the **Repository** from which you want to delete package versions.
4. Choose the **Package** from which you want to delete versions.
5. Select the **Package Version** that you want to delete.
6. Choose **Delete**.

> **Note**
>
> In the console, you can only delete one package version at a time. To delete more than one at a time, use the CLI.

Deleting an npm package or package version

To delete an npm package or individual package versions, set the `--format` option to npm. To delete a package version in a scoped npm package, use the `--namespace` option to specify the scope. For example, to delete the package `@types/react`, use `--namespace types`. Omit the `@` symbol when using `--namespace`.

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

To delete the package `@types/react`, including all of its versions:

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

Deleting a Maven package or package version

To delete a Maven package or individual package versions, set the `--format` option to `maven` and specify the package to delete by passing the Maven group ID with the `--namespace` option and the Maven artifactID with the `--name` option. For example, the following shows how to delete a single version of `com.google.guava:guava`:

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

The following example shows how to delete the package `com.google.guava:guava`, including all of its versions:

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

Best practices for deleting packages or package versions

If you do need to a delete a package version, as a best practice it's recommended that you create a repository to store a backup copy of the package version you'd like to delete. You can do this by first calling `copy-package-versions` to the backup repository:

```
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
```

Once you've copied the package version, you can then call `delete-package-versions` on package or package version you'd like to delete.

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

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.

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.

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. For more information, see View npm package version details or View Maven package version details.

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 --package pyhamcrest --package-version 1.9.0
```

The output might look like the following.

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

Note

CodeArtifact fetches package version details such as package home page or package license information from the metadata provided by the package author. If any of this information exceeds 400 KB, which is the DynamoDB item size limit, CodeArtifact won't be able to process such data and you may not see this information on the console or from the response of `describe-package-version`. For example, a python package such as https://pypi.org/project/rapyd-sdk/ has a very large license field, so this information wouldn't be processed by CodeArtifact.

View npm package version details

To view details about an npm package version, set the value of the `--format` option to **npm**. Optionally, include the package version namespace (npm *scope*) in the `--namespace` option. The value for the `--namespace` option should not include the leading `@`. To search for the namespace `@types`, set the value to *`types`*.

The following returns details about version `4.41.5` of an npm package named `webpack` in the `@types` scope.

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

The output might look like the following.

```
{
  "format": "npm",
  "namespace": "types",
  "package": "webpack",
  "displayName": "webpack",
  "version": "4.41.5",
  "summary": "Packs CommonJs/AMD modules for the browser. Allows ... further output
 omitted for brevity",
  "homePage": "https://github.com/webpack/webpack",
  "sourceCodeRepository": "https://github.com/webpack/webpack.git",
  "publishedTime": 1577481261.09,
  "licenses": [
    {
      "id": "license-id",
      "name": "license-name"
    }
  ],
  "revision": "REVISION-SAMPLE-55C752BEE9B772FC",
  "status": "Published",
  "origin": {
            "domainEntryPoint": {
                "externalConnectionName": "public:npmjs"
            },
            "originType": "EXTERNAL"
  }
}
```

View Maven package version details

To view details about a Maven package version, set the value of the `--format` option to `maven` and include the package version namespace in the `--namespace` option.

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"
}
```

> **Note**
>
> CodeArtifact does not extract package version detail information from parent POM files. The metadata for a given package version will only include information in the POM for that exact package version, not for the parent POM or any other POM referenced transitively using the POM `parent` tag. This means that the output of `describe-package-version` will omit metadata (such as license information) for Maven package versions that rely on a `parent` reference to contain this metadata.

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"
}
```

For the range of supported values for the dependencyType field, see the PackageDependency data type in the *CodeArtifact API*.

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.

Note

CodeArtifact does not support displaying readme files from generic or Maven packages.

```
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
```

The output might look like the following.

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

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.

> **Note**
>
> For more information on package version status, including a list of the available statuses, see Package version status.

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.

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. 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+No5j8I=",
            "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/UjBleHs1DZewk+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",
  "Resource": "arn:aws:codeartifact:us-east-1:111122223333:package/my_domain/my_repo/
npm//chalk"
}
```

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 group ID, which is referred to as a namespace in CodeArtifact. Use the `--namespace` parameter to specify the Maven group ID 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
 --domain-owner 111122223333 --repository my_repo --format maven
--namespace org.apache.logging.log4j --package log4j
--versions 2.13.1 --target-status Archived
```

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.

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

To update multiple versions with a single command, pass a comma-separated list of version and version revision pairs to the `--version-revisions` options. The following example command defines two different package version and package version revision pairs.

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

Sample output:

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

```
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:

```
{
    "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.

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

Sample output:

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

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. For more information about each package version status, see Package version status. 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 and expected status 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:

```
{
    "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:

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

Editing package origin controls

In AWS CodeArtifact, package versions can be added to a repository by directly publishing them, pulling them down from an upstream repository, or ingesting them from an external, public repository. Allowing package versions of a package to be added both by direct publishing and ingesting from public repositories makes you vulnerable to a dependency substitution attack. For more information, see Dependency substitution attacks. To protect yourself against a dependency substitution attack, you can configure package origin controls on a package in a repository to limit how versions of that package can be added to the repository.

Configuring package origin controls should be considered by any team that wants to allow new versions of different packages to come from both internal sources, such as direct publishing, and external sources, such as public repositories. By default, package origin controls will be configured based on how the first version of a package is added to the repository. For information about the package origin control settings and their default values, see Package origin control settings.

To remove the package record after using the `put-package-origin-configuration` API operation, use `delete-package` (see Delete a package or package version).

Common package access control scenarios

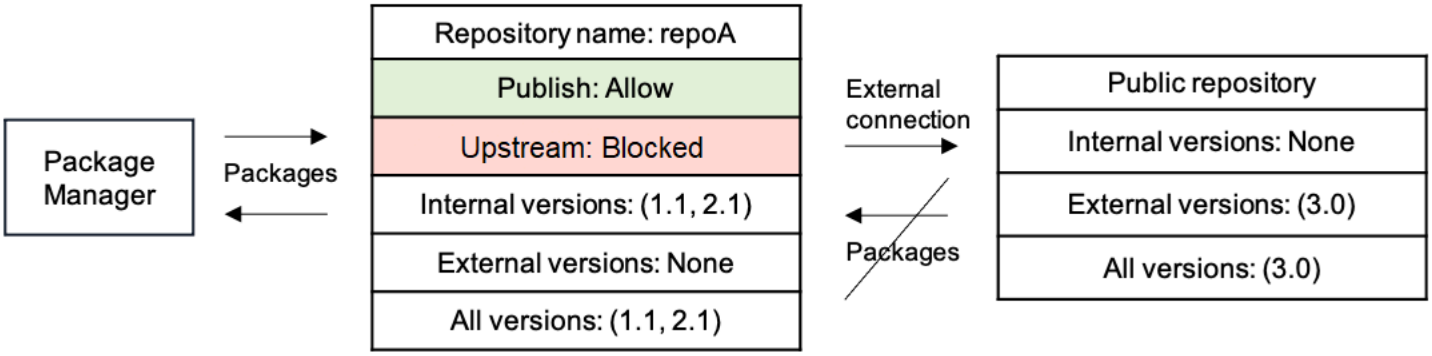
This section includes some common scenarios when a package version is added to a CodeArtifact repository. Package origin control settings will be set for new packages depending on how the first package version is added.

In the following scenarios, an *internal package* is a package that is published directly from a package manager to your repository, such as a package that you or your team authors and maintains. An *external package* is a package that exists in a public repository that can be ingested into your repository with an external connection.

An external package version is published for an existing internal package

In this scenario, consider an internal package, *packageA*. Your team publishes the first package version for *packageA* to a CodeArtifact repository. Because this is the first package version for that package, the package origin control settings are automatically set to **Publish: Allow** and **Upstream: Block**. After the package exists in your repository, a package with the same name is published to a public repository that is connected to your CodeArtifact repository. This could be an attempted dependency substitution attack against the internal package, or it could just be a coincidence. Regardless, package origin controls are configured to block the ingestion of the new external version to protect themselves against a potential attack.

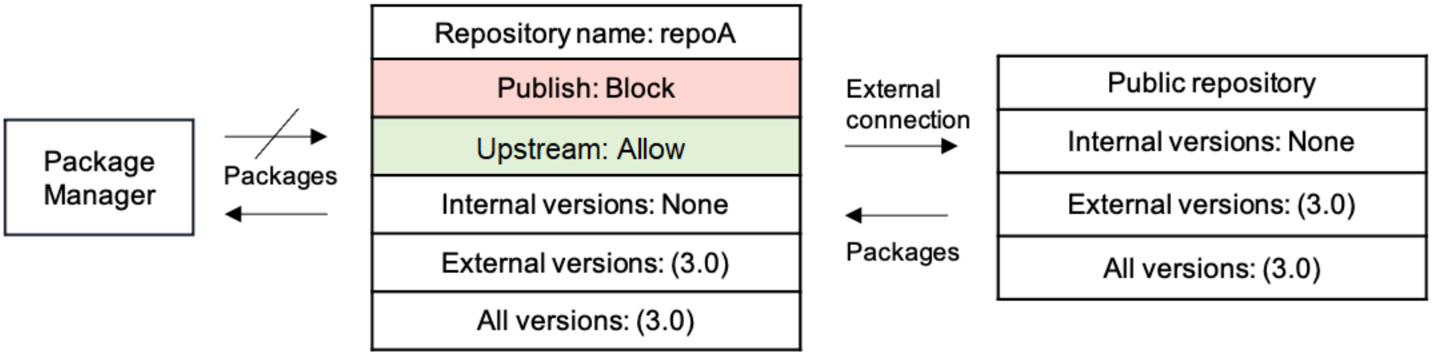
In the following image, *repoA* is your CodeArtifact repository with an external connection to a public repository. Your repository contains versions 1.1 and 2.1 of *packageA*, but version 3.0 is published to the public repository. Normally, *repoA* would ingest version 3.0 after the package was requested by a package manager. Because package ingestion is set to **Block**, version 3.0 is not ingested into your CodeArtifact repository and is not available to package managers connected to it.



An internal package version is published for an existing external package

In this scenario, a package, *packageB* exists externally in a public repository that you have connected to your repository. When a package manager connected to your repository requests *packageB*, the package version is ingested into your repository from the public repository. Because this is the first package version of *packageB* added to your repository, the package origin settings are configured to **Publish: BLOCK** and **Upstream: ALLOW**. Later, you try to publish a version with the same package name to the repository. Either you are not aware of the public package and trying to publish an unrelated package under the same name, or you are trying to publish a patched version, or you are trying to directly publish the exact package version that already exists externally. CodeArtifact will reject the version you are trying to publish, but allow you to explicitly override the rejection and publish the version if necessary.

In the following image, *repoA* is your CodeArtifact repository with an external connection to a public repository. Your repository contains version 3.0 that it ingested from the public repository. You want to publish version 1.1 to your repository. Normally, you could publish version 1.2 to *repoA*, but because publishing is set to **Block**, version 1.2 cannot be published.



Publishing a patched package version of an existing external package

In this scenario, a package, *packageB* exists externally in a public repository that you have connected to your repository. When a package manager connected to your repository requests *packageB*, the package version is ingested into your repository from the public repository. Because this is the first package version of *packageB* added to your repository, the package origin settings are configured to **Publish: BLOCK** and **Upstream: ALLOW**. Your team decides that it needs to publish patched package versions of this package to the repository. To be able to publish package versions directly, your team changes the package origin control settings to **Publish: ALLOW** and **Upstream: BLOCK**. Versions of this package can now be published directly to your repository and ingested from public repositories. After your team publishes the patched package versions, your team reverts the package origin settings to **Publish: BLOCK** and **Upstream: ALLOW**.

Package origin control settings

With package origin controls, you can configure how package versions can be added to a repository. The following lists include the available package origin control settings and values.

> **Note**
>
> The available settings and values are different when configuring origin controls on package groups. For more information, see Package group origin controls.

Publish

This setting configures whether package versions can be published directly to the repository using package managers or similar tools.

- **ALLOW**: Package versions can be published directly.

- **BLOCK**: Package versions cannot be published directly.

Upstream

This setting configures whether package versions can be ingested from external, public repositories, or retained from upstream repositories when requested by a package manager.

- **ALLOW**: Any package version can be retained from other CodeArtifact repositories configured as upstream repositories or ingested from a public source with an external connection.
- **BLOCK**: Package versions cannot be retained from other CodeArtifact repositories configured as upstream repositories or ingested from a public source with an external connection.

Default package origin control settings

The default package origin control settings are configured based on the package's associated package group's origin control settings. For more information about package groups and package group origin controls, see Working with package groups in CodeArtifact and Package group origin controls.

If a package is associated with a package group with restriction settings of `ALLOW` for every restriction type, the default package origin controls for a package will be based on how the first version of that package is added to the repository.

- If the first package version is published direcly by a package manager, the settings will be **Publish: ALLOW** and **Upstream: BLOCK**.
- If the first package version is ingested from a public source, the settings will be **Publish: BLOCK** and **Upstream: ALLOW**.

> **Note**
>
> Packages that existed in CodeArtifact repositories prior to around May 2022 will have a default package origin controls of **Publish: ALLOW** and **Upstream: ALLOW**. Package origin controls must be set manually for such packages. The current default values have been set on new packages since that time, and started being enforced when the feature launched on July 14, 2022. For more information about setting package origin controls, see Editing package origin controls.

Otherwise, if a package is associated with a package group that has at least one restriction setting of BLOCK or ALLOW_SPECIFIC_REPOSITORIES, then the default origin control settings for that package will be set to **Publish: ALLOW** and **Upstream: ALLOW**.

How package origin controls interact with package group origin controls

Because packages have origin control settings, and their associated package groups have origin control settings, it's important to understand how those two different settings interact with one another.

The interaction between the two settings is that a setting of BLOCK always wins over a setting of ALLOW. The following table lists some example configurations and their effective origin control settings.

Package origin control setting	Package group origin control setting	Effective origin control setting
PUBLISH: ALLOW UPSTREAM: ALLOW	PUBLISH: ALLOW UPSTREAM: ALLOW	PUBLISH: ALLOW UPSTREAM: ALLOW
PUBLISH: BLOCK UPSTREAM: ALLOW	PUBLISH: ALLOW UPSTREAM: ALLOW	PUBLISH: BLOCK UPSTREAM: ALLOW
PUBLISH: ALLOW UPSTREAM: ALLOW	PUBLISH: ALLOW UPSTREAM: BLOCK	PUBLISH: ALLOW UPSTREAM: BLOCK

What this means is that a package with origin settings of **Publish: ALLOW** and **Upstream: ALLOW**, then it is effectively deferring to the associated package group's origin control settings.

Editing package origin controls

Package origin controls are configured automatically based on how the first package version of a package is added to the repository, for more information see Default package origin control settings. To add or edit package origin controls for a package in a CodeArtifact repository, perform the steps in the following procedure.

To add or edit package origin controls (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
2. In the navigation pane, choose **Repositories**, and choose the repository that contains the package you want to edit.
3. In the **Packages** table, search for and select the package you want to edit.
4. From the package summary page, in **Origin controls**, choose **Edit**.
5. In **Edit origin controls**, choose the package origin controls you want to set for this package. Both package origin control settings, Publish and Upstream, must be set at the same time.
   - To allow publishing package versions directly, in **Publish**, choose **Allow**. To block publishing of package versions, choose **Block**.
   - To allow ingestion of packages from external repositories and pulling packages from upstream repositories, in **Upstream sources**, choose **Allow**. To block all ingestion and pulling of package versions from external and upstream repositories, choose **Block**.

To add or edit package origin controls (AWS CLI)

1. If you haven't, configure the AWS CLI by following the steps in [Setting up with AWS CodeArtifact](#).
2. Use the `put-package-origin-configuration` command to add or edit package origin controls. Replace the following fields:
   - Replace *my_domain* with the CodeArtifact domain that contains the package you want to update.
   - Replace *my_repo* with the CodeArtifact repository that contains the package you want to update.
   - Replace *npm* with the package format of the package you want to update.
   - Replace *my_package* with the name of the package you want to update.
   - Replace *ALLOW* and *BLOCK* with your desired package origin control settings.

```
aws codeartifact put-package-origin-configuration --domain my_domain \
--repository my_repo --format npm --package my_package \
--restrictions publish=ALLOW,upstream=BLOCK
```

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. Consider the following scenarios.

1. The package origin control settings on `com.mycompany.mypackage` are **Publish: ALLOW** and **Upstream: ALLOW**. 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. You could still publish a different version, such as `com.mycompany.mypackage:1.1`.
2. The package origin control settings on `com.mycompany.mypackage` are **Publish: ALLOW** and **Upstream: BLOCK**. You can publish any version of `com.mycompany.mypackage` to your repository that do not already exist because package versions are not reachable.
3. The package origin control settings on `com.mycompany.mypackage` are **Publish: BLOCK** and **Upstream: ALLOW**. You cannot publish any package versions directly to your repository.

Working with package groups in CodeArtifact

Package groups can be used to apply configuration to multiple packages that match a defined pattern using package format, package namespace, and package name. You can use package groups to more conveniently configure package origin controls for multiple packages. Package origin controls are used to block or allow ingestion or publishing of new package versions, which protects users from malicious actions known as dependency substitution attacks.

Every domain in CodeArtifact automatically contains a root package group. This root package group, `/*`, contains all packages, and allows package versions to enter repositories in the domain from all origin types by default. The root package group can be modified, but cannot be deleted.

The Package Group Configuration feature operates in an eventually consistent manner when creating a new package group or deleting an existing package group. This means that upon creating or deleting a package group, the origin controls will be applied to the expected associated packages, but with some delay due to the eventual consistent behavior. The time to reach eventual consistency depends on the number of package groups in the domain as well as the number of packages in the domain. There may be a brief period where the origin controls are not immediately reflected on the associated packages after a package group creation or deletion.

Additionally, updates to package group origin controls are effective almost immediately. Unlike the creation or deletion of package groups, changes to the origin controls of an existing package group are reflected on the associated packages without the same delay.

These topics contain information about package groups in AWS CodeArtifact.

Topics

Create a package group

You can create a package group using the CodeArtifact console, the AWS Command Line Interface (AWS CLI), or AWS CloudFormation. For more information about managing CodeArtifact package groups with CloudFormation, see Creating CodeArtifact resources with AWS CloudFormation.

Create a package group (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
2. In the navigation pane, choose **Domains**, and then choose the domain in which you want to create a package group.
3. Choose **Package groups**, and choose **Create package group**.
4. In **Package group definition**, enter the package group definition for your package group. The package group definition determines which packages are associated with the group. You can enter the package group definition manually with text, or you can use the visual mode to make selections and the package group definition will be created automatically.
5. To use the visual mode to create the package group definition:

   a. Choose **Visual** to switch to the visual mode..

   b. In **Package format**, choose the format of the packages to be associated with this group.

   c. In **Namespace (Scope)**, choose the namespace criteria to match on.

      - **Equals**: Match the specified namespace exactly. If chosen, enter the namespace to match on.
      - **Blank**: Match packages with no namespace.
      - **Starts with word**: Match namespaces that begin with a specified word. If chosen, enter the prefix word to match on. For more information about words and word boundaries, see Words, word boundaries, and prefix matching.
      - **All**: Match packages in all namespaces.

   d. If **Equals**, **Blank**, or **Starts with word** is selected, in **Package name**, choose the package name criteria to match on.

      - **Exactly equals**: Match the specified package name exactly. If chosen, enter the package name to match on.
      - **Starts with prefix**: Match packages that start with the specified prefix.

        - **Starts with word**: Match packages that begin with a specified word. If chosen, enter the prefix word to match on. For more information about words and word boundaries, see Words, word boundaries, and prefix matching.
        - **All**: Match all packages.

    e. Choose **Next** to review the definition.

6. To enter the package group definition with text:

    a. Choose **Text** to switch to the text mode.

    b. In **Package group definition**, enter the package group definition. For more information about package group definition syntax, see Package group definition syntax and matching behavior.

    c. Choose **Next** to review the definition.

7. In **Review definition**, review the packages that will be included in the new package group based on the definition provided previously. After reviewing, choose **Next**.
8. In **Package group information**, optionally add a description and contact email for the package group. Choose **Next**.
9. In **Package origin controls**, configure the origin controls to be applied to the packages in the group. For more information about package group origin controls, see Package group origin controls.
10. Choose **Create package group**.

Create a package group (AWS CLI)

Use the `create-package-group` command to create a package group in your domain. For the `--package-group` option, enter the package group definition that determines which packages are associated with the group. For more information about package group definition syntax, see Package group definition syntax and matching behavior.

If you haven't, configure the AWS CLI by following the steps in Setting up with AWS CodeArtifact.

```
aws codeartifact create-package-group \
         --domain my_domain \
         --package-group '/nuget/*' \
         --domain-owner 111122223333 \
         --contact-info contact@email.com \
         --description "a new package group" \
        --tags key=key1,value=value1
```

View or edit a package group

You can view a list of all package groups, view details of a specific package group, or edit a package group's details or configuration using the CodeArtifact console or the AWS Command Line Interface (AWS CLI).

View or edit a package group (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
2. In the navigation pane, choose **Domains**, and then choose the domain that contains the package group you want to view or edit.
3. Choose **Package groups**, and choose the package group you want to view or edit.
4. In **Details**, view information about the package group including its parent group, description, ARN, contact email, and package origin controls.
5. In **Subgroups**, view a list of package groups that have this group as a parent group. The package groups in this list can inherit settings from this package group. For more information, see Package group hierarchy and pattern specificity.
6. In **Packages**, view the packages that belong to this package group based on the package group definition. In the **Strength** column, you can see the strength of the package association. For more information, see Package group hierarchy and pattern specificity.
7. To edit package group information, choose **Edit package group**.

    a. In **Information**, update the package group's description or contact information. You cannot edit a package group's definition.

    b. In **Package group origin controls**, update the package group's origin control settings, which determine how associated packages can enter repositories in the domain. For more information, see Package group origin controls.

View or edit a package group (AWS CLI)

Use the following commands to view or edit package groups with the AWS CLI. If you haven't, configure the AWS CLI by following the steps in Setting up with AWS CodeArtifact.

To view all package groups in a domain, use the `list-package-groups` command.

```
aws codeartifact list-package-groups \
         --domain my_domain \
         --domain-owner 111122223333
```

To view details about a package group, use the `describe-package-group` command. For more information about package group definitions, see Package group definition syntax and examples.

```
aws codeartifact describe-package-group \
         --domain my_domain \
         --domain-owner 111122223333 \
         --package-group '/nuget/*'
```

To view the child package groups of a package group, use the `list-sub-package-groups` command.

```
aws codeartifact list-sub-package-groups \
         --domain my_domain \
         --domain-owner 111122223333 \
         --package-group '/nuget/*' \
```

To view the package group that is associated to a package, use the `get-associated-package-group` command. You must use the normalized package name and namespace for the NuGet, Python, and Swift package formats. For more information about how the package names and namespaces are normalized, see the NuGet, Python, and Swift name normalization documentation.

```
aws codeartifact get-associated-package-group \
         --domain my_domain \
         --domain-owner 111122223333 \
         --format npm \
         --package packageName \
         --namespace scope
```

To edit a package group, use the `update-package-group` command. This command is used to update a package group's contact information or description. For information about package group origin control settings, and adding or editing them, see Package group origin controls. For more information about package group definitions, see Package group definition syntax and examples

```
aws codeartifact update-package-group \
         --domain my_domain \
         --package-group '/nuget/*' \
         --domain-owner 111122223333 \
         --contact-info contact@email.com \
         --description "updated package group description"
```

Delete a package group

You can delete a package group using the CodeArtifact console or the AWS Command Line Interface (AWS CLI).

Note the following behavior when deleting package groups:

- The root package group, `/*`, cannot be deleted.
- The packages and package versions that are associated with that package group are not deleted.
- When a package group is deleted, the direct child package groups will become children of the package group's direct parent package group. Therefore, if any of the child groups are inheriting any settings from the parent, those settings could change.

Delete a package group (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
2. In the navigation pane, choose **Domains**, and then choose the domain that contains the package group you want to view or edit.
3. Choose **Package groups**.
4. Choose the package group you want to delete and choose **Delete**.
5. Enter delete in the field and choose **Delete**.

Delete a package group (AWS CLI)

To delete a package group, use the `delete-package-group` command.

```
aws codeartifact delete-package-group \
         --domain my_domain \
         --domain-owner 111122223333 \
          --package-group '/nuget/*'
```

Package group origin controls

Package origin controls are used to configure how package versions can enter a domain. You can set up origin controls on a package group to configure how versions of every package associated with the package group can enter specified repositories in the domain.

Package group origin control settings consist of the following:

- Restriction settings: These settings define if packages can enter a repository in CodeArtifact from publishing, internal upstreams, or external, public repositories.
- Allowed repository lists: Each restriction setting can be set to allow specific repositories. If a restriction setting is set to allow specific repositories, that restriction will have a corresponding allowed repository list.

> **Note**
>
> Origin control settings for package groups are slightly different than the origin control settings for individual packages. For more information about origin control settings for packages, see Package origin control settings.

Restriction settings

The restriction settings of a package group's origin control settings determine how the packages associated with that group can enter repositories in the domain.

PUBLISH

The `PUBLISH` setting configures whether package versions can be published directly to any repository in the domain using package managers or similar tools.

- **ALLOW**: Package versions can be published directly to all repositories.
- **BLOCK**: Package versions cannot be published directly to any repository.
- **ALLOW_SPECIFIC_REPOSITORIES**: Package versions can only be published directly to repositories specified in the allowed repository list for publishing.

- **INHERIT**: The PUBLISH setting is inherited from the first parent package group with a setting that is not INHERIT.

EXTERNAL_UPSTREAM

The EXTERNAL_UPSTREAM setting configures whether package versions can be ingested from external, public repositories when requested by a package manager. For a list of supported external repositories, see Supported external connection repositories.

- **ALLOW**: Any package version can be ingested into all repositories from a public source with an external connection.
- **BLOCK**: Package versions cannot be ingested into any repository from a public source with an external connection.
- **ALLOW_SPECIFIC_REPOSITORIES**: Package versions can only be ingested from a public source into repositories specified in the allowed repository list for external upstreams.
- **INHERIT**: The EXTERNAL_UPSTREAM setting is inherited from the first parent package group with a setting that is not INHERIT.

INTERNAL_UPSTREAM

The INTERNAL_UPSTREAM setting configures whether package versions can be retained from internal upstream repositories in the same CodeArtifact domain when requested by a package manager.

- **ALLOW**: Any package version can be retained from other CodeArtifact repositories configured as upstream repositories.
- **BLOCK**: Package versions cannot be retained from other CodeArtifact repositories configured as upstream repositories.
- **ALLOW_SPECIFIC_REPOSITORIES**: Package versions can only be retained from other CodeArtifact respositories configured as upstream repositories into repositories specified in the allowed repository list for internal upstreams.
- **INHERIT**: The INTERNAL_UPSTREAM setting is inherited from the first parent package group with a setting that is not INHERIT.

Allowed repository lists

When a restriction setting is configured as `ALLOW_SPECIFIC_REPOSITORIES`, the package group contains an accompanying allowed repositories list which contains a list of repositories allowed for that restriction setting. Therefore, a package group contains anywhere from 0 to 3 allowed repository lists, one for each setting configured as `ALLOW_SPECIFIC_REPOSITORIES`.

When you add a repository to a package group's allowed repository list, you must specify which allowed repository list to add it to.

The possible allowed repository lists are as follows:

- `EXTERNAL_UPSTREAM`: Allow or block ingestion of package versions from external repositories in the added repository.
- `INTERNAL_UPSTREAM`: Allow or block pulling package versions from another CodeArtifact repository in the added repository.
- `PUBLISH`: Allow or block direct publishing of package versions from package managers to the added repository.

Editing package group origin control settings

To add or edit origin controls for a package group, perform the steps in the following procedure. For information about the package group origin control settings, see Restriction settings and Allowed repository lists.

To add or edit package group origin controls (CLI)

1. If you haven't, configure the AWS CLI by following the steps in Setting up with AWS CodeArtifact.
2. Use the `update-package-group-origin-configuration` command to add or edit package origin controls.

   - For `--domain`, enter the CodeArtifact domain that contains the package group you want to update.
   - For `--domain-owner`, enter the account number of the owner of the domain.
   - For `--package-group`, enter the package group you want to update.
   - For `--restrictions`, enter key-value pairs that represent the origin control restrictions.

- For `--add-allowed-repositories`, enter a JSON object containing the restriction type and repository name to add to the corresponding allowed repositories list for the restriction.
- For `--remove-allowed-repositories`, enter a JSON object containing the restriction type and repository name to remove from the corresponding allowed repositories list for the restriction.

```
aws codeartifact update-package-group-origin-configuration \
   --domain my_domain \
   --domain-owner 111122223333 \
   --package-group '/nuget/*' \
   --restrictions INTERNAL_UPSTREAM=ALLOW_SPECIFIC_REPOSITORIES \
   --add-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo \
   --remove-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo2
```

The following example adds multiple restrictions, and multiple repositories in one command.

```
aws codeartifact update-package-group-origin-configuration \
   --domain my_domain \
   --domain-owner 111122223333 \
   --package-group '/nuget/*' \
   --
restrictions PUBLISH=BLOCK,EXTERNAL_UPSTREAM=ALLOW_SPECIFIC_REPOSITORIES,INTERNAL_UPSTREAM=
 \
   --add-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo2 \
   --remove-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo2
```

Package group origin control configuration examples

The following examples show package origin control configurations for common package management scenarios.

Allowing packages with private names to be published, but not ingested

This scenario is likely a common scenario in package management:

- Allow packages with private names to be published to repositories in your domain from package managers, and block them from being ingested to repositories in your domain from external, public repositories.
- Allow all other packages to be ingested to repositories in your domain from external, public repositories, and block them from being published to repositories in your domain from package managers.

To achieve this, you should configure a package group with a pattern that includes the private name(s), and origin settings of **PUBLISH: ALLOW**, **EXTERNAL_UPSTREAM: BLOCK**, and **INTERNAL_UPSTREAM: ALLOW**. This will ensure packages with private names can be published directly, but cannot be ingested from external repositories.

The following AWS CLI commands create and configure a package group with origin restriction settings that match the desired behavior:

To create the package group:

```
aws codeartifact create-package-group \
   --domain my_domain \
   --package-group /npm/space/anycompany~ \
   --domain-owner 111122223333 \
   --contact-info contact@email.com | URL \
   --description "my package group"
```

To update the package group's origin configuration:

```
aws codeartifact update-package-group-origin-configuration \
   --domain my_domain \
   --domain-owner 111122223333 \
   --package-group '/npm/space/anycompany~' \
   --restrictions PUBLISH=ALLOW,EXTERNAL_UPSTREAM=BLOCK,INTERNAL_UPSTREAM=ALLOW
```

Allowing ingestion from external repositories through one repository

In this scenario, your domain has multiple repositories. Of those repositories, `repoA` has an upstream connection to `repoB`, which has an external connection to the public repository, `npmjs.com`, as shown:

```
repoA --> repoB --> npmjs.com
```

You want to allow ingestion of packages from a specific package group, `/npm/space/anycompany~` from `npmjs.com` into `repoA`, but only through `repoB`. You also want to block ingestion of packages associated with the package group into any other repositories in your domain, and block direct publishing of packages with package managers. To achieve this, you create and configure the package group as follows:

Origin restriction settings of **PUBLISH: BLOCK**, and **EXTERNAL_UPSTREAM: ALLOW_SPECIFIC_REPOSITORIES**, and **INTERNAL_UPSTREAM: ALLOW_SPECIFIC_REPOSITORIES**.

`repoA` and `repoB` added to the appropriate allowed repository list:

- `repoA` should be added to the `INTERNAL_UPSTREAM` list, as it will get packages from its internal upstream, `repoB`.
- `repoB` should be added to the `EXTERNAL_UPSTREAM` list, as it will get packages from the external repository, `npmjs.com`.

The following AWS CLI commands create and configure a package group with origin restriction settings that match the desired behavior:

To create the package group:

```
aws codeartifact create-package-group \
   --domain my_domain \
   --package-group /npm/space/anycompany~ \
   --domain-owner 111122223333 \
   --contact-info contact@email.com | URL \
   --description "my package group"
```

To update the package group's origin configuration:

```
aws codeartifact update-package-group-origin-configuration \
   --domain my_domain \
   --domain-owner 111122223333 \
   --package-group /npm/space/anycompany~ \
   --
restrictions PUBLISH=BLOCK,EXTERNAL_UPSTREAM=ALLOW_SPECIFIC_REPOSITORIES,INTERNAL_UPSTREAM=ALLO
 \
   --add-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=repoA
 originRestrictionType=EXTERNAL_UPSTREAM,repositoryName=repoB
```

How package group origin control settings interact with package origin control settings

Because packages have origin control settings, and their associated package groups have origin control settings, it's important to understand how those two different settings interact with one another. For information about the interaction between the settings, see How package origin controls interact with package group origin controls.

Package group definition syntax and matching behavior

This topic contains information about defining package groups, pattern matching behavior, package association strength, and package group hierarchy.

Contents

Package group definition syntax and examples

The pattern syntax for defining package groups closely follows the formatting of package paths. A package path is created from a package's coordinate components (format, namespace, and name) by adding a forward slash to the start and separating each of the components with a forward slash. For example, the package path for the npm package named *anycompany-ui-components* in the namespace *space* is */npm/space/anycompany-ui-components*.

A package group pattern follows the same structure as a package path, except components that are not specified as part of the group definition are omitted, and the pattern is terminated with a suffix. The suffix that is included determines the matching behavior of the pattern, as follows:

- A $ suffix will match the full package coordinate.
- A ~ suffix will match a prefix.
- A * suffix will match all values of the previously defined component.

Here are example patterns for each of the allowed combinations:

1. All package formats: `/*`
2. A specific package format: `/npm/*`
3. Package format and namespace prefix: `/maven/com.anycompany~`
4. Package format and namespace: `/npm/space/*`
5. Package format, namespace, and name prefix: `/npm/space/anycompany-ui~`
6. Package format, namespace, and name: `/maven/org.apache.logging.log4j/log4j-core$`

As shown in the examples above, the ~ suffix is added to the end of a namespace or name to represent a prefix match and `*` comes after a forward slash when used to match all values for the next component in the path (either all formats, all namespaces, or all names).

Package group definition and normalization

CodeArtifact normalizes NuGet, Python, and Swift package names, and normalizes Swift package namespaces before storing them. CodeArtifact uses these normalized names when matching packages with package group definitions. Therefore, package groups that contain a namespace or name in these formats must use the normalized namespace and name. For more information about how the package names and namespaces are normalized, see the NuGet, Python, and Swift name normalization documentation.

Namespaces in package group definitions

For packages or package formats without a namespace (Python and NuGet), package groups must not contain a namespace. The package group definition for these package groups contain a blank namespace section. For example, the path for the Python package named *requests* is */python//requests*.

For packages or package formats with a namespace (Maven, generic, and Swift), the namespace must be included if the package name is included. For the Swift package format, the normalized package namespace will be used. For more information about how Swift package namespaces are normalized, see Swift package name and namespace normalization.

Package group hierarchy and pattern specificity

The packages that are "in" or "associated with" a package group are packages with a package path that matches the group's pattern but do not match a more specific group's pattern. For example, given the package groups `/npm/*` and `/npm/space/*`, the package path */npm//react* is associated with the first group (`/npm/*`) while */npm/space/aui.components* and */npm/space/amplify-ui-core* are associated with the second group (`/npm/space/*`). Even though a package may match multiple groups, each package is only associated with a single group, the most specific match, and only that one group's configuration applies to the package.

When a package path matches multiple patterns, the "more specific" pattern can be thought of as the longest matching pattern. Alternatively, the more specific pattern is the one that matches a proper subset of the packages that match the less specific pattern. From our earlier example, every package that matches `/npm/space/*` also matches `/npm/*`, but the reverse is not true, which makes `/npm/space/*` the more specific pattern because it is a proper subset of `/npm/*`. Because one group is a subset of another group, it creates a hierarchy, in which `/npm/space/*` is a subgroup of the parent group, `/npm/*`.

Though only the most specific package group's configuration applies to a package, that group may be configured to inherit from its parent group's configuration.

Words, word boundaries, and prefix matching

Before discussing prefix matching, let's define some key terms:

- A *word* a letter or number followed by zero or more letters, numbers, or mark characters (such as accents, umlauts, etc.).
- A *word boundary* is at the end of a word, when a non-word character is reached. Non-word characters are punctuation characters such as `.`, `-`, and `_`.

Specifically, the regex pattern for a word is `[\p{L}\p{N}][\p{L}\p{N}\p{M}]*`, which can be broken down as follows:

- `\p{L}` represents any letter.
- `\p{N}` represents any number.
- `\p{M}` represents any mark character, such as accents, umlauts, etc.

Therefore, `[\p{L}\p{N}]` represents a number or letter, and `[\p{L}\p{N}\p{M}]*` represents zero or more letters, numbers, or mark characters and a word boundary is at the end of each match of this regex pattern.

> **Note**
>
> Word boundary matching is based on this definition of a "word". It is not based on words defined in a dictionary, or CameCase. For example, there is no word boundary in `oneword` or `OneWord`.

Now that word and word boundary are defined, we can use them to describe prefix matching in CodeArtifact. To indicate a prefix match on a word boundary, a match character (~) is used after a word character. For example, the pattern `/npm/space/foo~` matches the package paths `/npm/space/foo` and `/npm/space/foo-bar`, but not `/npm/space/food` or `/npm/space/foot`.

A wildcard (*) is required to be used instead of ~ when following a non-word character, such as in the pattern `/npm/*`.

Case sensitivity

Package group definitions are case sensitive, which means that patterns that differ only by case can exist as separate package groups. For example, a user can create separate package groups with the patterns `/npm//AsyncStorage$`, `/npm//asyncStorage$`, and `/npm//asyncstorage$` for the three separate packages that exist on the npm Public Registry: *AsyncStorage*, *asyncStorage*, *asyncstorage* that differ only by case.

While case matters, CodeArtifact still associates packages to a package group if the package has a variation of the pattern that differs by case. If a user creates the `/npm//AsyncStorage$` package group without creating the other two groups shown above, then all case variations of the name *AsyncStorage*, including *asyncStorage* and *asyncstorage*, will be associated with the package group.

But, as described in the next section, Strong and weak match, these variations will be handled differently than *AsyncStorage*, which exactly matches the pattern.

Strong and weak match

The information in the previous section, Case sensitivity, states that package groups are case sensitive, and then goes on to explain they are case insensitive. This is because package group definitions in CodeArtifact have a concept of strong match (or exact match) and a weak match (or variation match). A strong match is when the package matches the pattern exactly, without any variation. A weak match is when the package matches a variation of the pattern, such as different letter case. Weak match behavior prevents packages that are variations of a package group's pattern from rolling up to a more general package group. When a package is a variation (weak match) of the most specific matching group's pattern, then the package is associated with the group but the package is blocked instead of applying the group's origin control configuration, preventing any new versions of the package from being pulled from upstreams or published. This behavior reduces the risk of supply chain attacks resulting from dependency confusion of packages with nearly identical names.

To illustrate weak match behavior, suppose package group `/npm/*` allows ingestion and blocks publishing. A more specific package group, `/npm//anycompany-spicy-client$`, is configured to block ingestion and allow publish. The package named *anycompany-spicy-client* is a strong match of the package group, which allows package versions to be published and blocks ingestion of package versions. The only casing of the package name that is allowed to be published is *anycompany-spicy-client*, since it is a strong match for the package definition pattern. A different case variation, such as *AnyCompany-spicy-client* is blocked from publishing because it is a weak match. More importantly, the package group blocks ingestion of all case variations, not just the lowercase name used in the pattern, reducing the risk of a dependency confusion attack.

Additional variations

In addition to case differences, weak matching also ignores differences in sequences of dash `-`, dot `.`, underscore `_`, and confusable characters (such as similar looking characters from separate alphabets). During normalization used for weak matching, CodeArtifact performs casefolding (similar to converting to lowercase), replaces sequences of dash, dot, and underscore characters with a single dot, and normalizes confusable characters.

Weak matching treats dashes, dots, and underscores as equivalent but does not completely ignore them. This means that *foo-bar*, *foo.bar*, *foo..bar*, and *foo_bar* are all weak match equivalents, but *foobar* is not. Although several public repositories implement steps to prevent these types of varations, the protection provided by public repositories does not make this feature of package groups unnecessary. For example, public repositories such as the npm Public Registry registry will only prevent new variations of the package named *my-package* if *my-package* is already published to it. If *my-package* is an internal package and package group `/npm//my-package$` is created that allows publish and blocks ingestion, you likely don't want to publish *my-package* to the npm Public Registry in order to prevent a variant such as *my.package* from being allowed.

While some package formats such as Maven treat these characters differently (Maven treats `.` as a namespace hierarchy separator but not `-` or `_`), something like *com.act-on* could still be confused with *com.act.on*.

> **Note**
>
> Note that whenever multiple variations are associated with a package group, an administrator may create a new package group for a specific variation to configure different behavior for that variation.

Tag a package group in CodeArtifact

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

You can use the CLI to specify tags when you create a package group or add, remove, or update the value of tags of an existing package group.

Tag package groups (CLI)

You can use the CLI to manage package group tags.

If you haven't, configure the AWS CLI by following the steps in Setting up with AWS CodeArtifact.

> **Tip**
>
> To add tags, you must provide the Amazon Resource Name (ARN) of the package group. To get the ARN of the package group, run the `describe-package-group` command:

```
aws codeartifact describe-package-group \
   --domain my_domain \
   --package-group /npm/scope/anycompany~ \
   --query packageGroup.arn
```

Topics

- Add tags to a package group (CLI)
- View tags for a package group (CLI)
- Edit tags for a package group (CLI)
- Remove tags from a package group (CLI)

Add tags to a package group (CLI)

You can add tags to package groups when they are created, or to an existing package group. For information about adding tags to a package group when you create it, see Create a package group.

To add a tag to an existing package group with the AWS CLI, at the terminal or command line, run the **tag-resource** command, specifying the Amazon Resource Name (ARN) of the package group where you want to add tags and the key and value of the tag you want to add. For information about package group ARNs, see Package group ARNs.

You can add more than one tag to a package group. For example, to tag a package group, *`/npm/scope/anycompany~`* 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-2:123456789012:package-
group/my_domain/npm/scope/anycompany~ \
   --tags key=key1,value=value1 key=key2,value=value2
```

If successful, this command has no output.

View tags for a package group (CLI)

Follow these steps to use the AWS CLI to view the AWS tags for a package group. 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 package group. For information about package group ARNs, see Package group ARNs.

For example, to view a list of tag keys and tag values for a package group, */npm/scope/anycompany~* named with an ARN value of arn:aws:codeartifact:*us-west-2*:*123456789012*:package-group/*my_domain*/*npm/scope/anycompany~*

```
aws codeartifact list-tags-for-resource \
   --resource-arn arn:aws:codeartifact:us-west-2:123456789012:package-
group/my_domain/npm/scope/anycompany~
```

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

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

Edit tags for a package group (CLI)

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

At the terminal or command line, run the **tag-resource** command, specifying the ARN of the package group where you want to update a tag and specify the tag key and tag value. For information about package group ARNs, see Package group ARNs.

```
aws codeartifact tag-resource \
   --resource-arn arn:aws:codeartifact:us-west-2:123456789012:package-
group/my_domain/npm/scope/anycompany~ \
   --tags key=key1,value=newvalue1
```

If successful, this command has no output.

Remove tags from a package group (CLI)

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

Note

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

At the terminal or command line, run the **untag-resource** command, specifying the ARN of the package group where you want to remove tags and the tag key of the tag you want to remove. For information about package group ARNs, see Package group ARNs.

For example, to remove multiple tags on a package group, *`/npm/scope/anycompany~`*, with the tag keys *`key1`* and *`key2`*:

```
aws codeartifact untag-resource \
   --resource-arn arn:aws:codeartifact:us-west-2:123456789012:package-
group/my_domain/npm/scope/anycompany~ \
   --tag-keys key1 key2
```

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

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 CodeArtifact console, the AWS CLI, and AWS CloudFormation to create or configure CodeArtifact domains.

Topics

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 1 or 1,000 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, 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.
- **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.

- **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 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.

```
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.

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

Types of AWS KMS keys supported in CodeArtifact

CodeArtifact supports only symmetric KMS keys. You can't use an asymmetric KMS key to encrypt your CodeArtifact domains. For more information, see Identifying symmetric and asymmetric KMS keys. To learn how to create a new customer managed key, see Creating symmetric encryption KMS keys in the *AWS Key Management Service Developer Guide*.

CodeArtifact supports AWS KMS External Key Stores (XKS). You are responsible for the availability, durability, and latency of key operations with XKS keys, which can affect availability, durability, and latency with CodeArtifact. Some examples of effects of using XKS keys with CodeArtifact:

- Because every asset of a requested package and all of its dependencies is subject to decryption latency, build latency can be increased substantially with an increase in XKS operation latency.
- Because all assets are encrypted in CodeArtifact, a loss of XKS key materials will result in a loss of all assets associated with the domain using the XKS key.

For more information about XKS keys, see External key stores in the *AWS Key Management Service Developer Guide*.

Create a domain

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

Topics

- Example AWS KMS key policy

Create a domain (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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. 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. For more information about the supported types of KMS keys in CodeArtifact, see Types of AWS KMS keys supported in CodeArtifact.

   - 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*.
6. Choose **Create domain**.

Create a domain (AWS CLI)

To create a domain with the AWS CLI, use the `create-domain` command. 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.

For more information about the supported types of KMS keys in CodeArtifact, see Types of AWS KMS keys supported in CodeArtifact. For more information about KMS keys, see AWS managed key and Customer managed key in the *AWS Key Management Service Developer Guide*.

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

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

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

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

```
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.

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

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
```

Example AWS KMS key policy

When you create a domain in CodeArtifact, you use a KMS key to encrypt all assets in the domain. You can choose an AWS managed KMS key, or a customer managed key that you manage. For more information about KMS keys, see the AWS Key Management Service Developer Guide.

To use a customer managed key, your KMS key must have a key policy that grants access to CodeArtifact. A key policy is a resource policy for an AWS KMS key and are the primary way to control access to KMS keys. Every KMS key must have exactly one key policy. The statements in the key policy determine who has permission to use the KMS key and how they can use it.

The following example key policy statement allows AWS CodeArtifact to create grants and view key details on behalf of authorized users. This policy statement limits the permission to CodeArtifact acting on the specified account ID’s behalf by using the `kms:ViaService` and `kms:CallerAccount` condition keys. It also grants all AWS KMS permissions to the IAM root user, so the key can be managed after it is created.

```
{
    "Version": "2012-10-17",
    "Id": "key-consolepolicy-3",
    "Statement": [
        {
            "Sid": "Allow access through AWS CodeArtifact for all principals in the
 account that are authorized to use CodeArtifact",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "kms:CreateGrant",
                "kms:DescribeKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:CallerAccount": "111122223333",
                    "kms:ViaService": "codeartifact.us-west-2.amazonaws.com"
                }
            }
        },
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        }
    ]
}
```

Delete a domain

You can delete a domain using the CodeArtifact console or the AWS Command Line Interface (AWS CLI).

Topics

- Restrictions on domain deletion
- Delete a domain (console)
- Delete a domain (AWS CLI)

Restrictions on domain deletion

Normally, 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.

However, if CodeArtifact no longer has access to the domain's KMS key, you can delete the domain even if it still contains repositories. This situation will occur if you delete the domain's KMS key or revoke the KMS grant that CodeArtifact uses to access the key. In this state, you cannot access the repositories in the domain or the packages stored in them. Listing and deleting of repositories is also not possible when CodeArtifact cannot access the domain's KMS key. For this reason, domain deletion doesn't check whether the domain contains repositories when the domain's KMS key is inaccessible.

> **Note**
>
> When a domain that still contains repositories is deleted, CodeArtifact will asynchronously delete the repositories within 15 minutes. After the domain is deleted, the repositories will still be visible in the CodeArtifact console and in the output of the `list-repositories` command until the automatic repository cleanup occurs.

Delete a domain (console)

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my_domain",
        "status": "Active",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0,
        "createdTime": "2020-10-12T16:51:18.039000-04:00"
    }
}
```

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

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.

The domain policy is evaluated for all operations against the domain and all resources within the domain. This means the domain policy may be used to apply permissions to repositories and packages in the domain. When the `Resource` element is set to `*`, then the statement applies to all resources in the domain. For example, if the policy above also included `codeartifact:DescribeRepository` in the list of allowed IAM actions, then the policy would allow calling `DescribeRepository` on every repository in the domain. A domain policy may be used to apply permissions to specific resources in the domain by using specific resource ARNs in the `Resource` element.

> **Note**
>
> Both domain and repository policies may be used to configure permissions. When both policies are present, then both policies will be evaluated and an action is allowed if allowed by either policy. For more information, see [Interaction between repository and domain policies](#).

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": "*"
        }
    ]
}
```

Note

A principal who wants to fetch packages from a repository endpoint must be granted the `ReadFromRepository` permission on the repository resource in addition to the `GetAuthorizationToken` permission on the domain. Similarly, a principal who wants to publish packages to a repository endpoint must be granted the `PublishPackageVersion` permission in addition to `GetAuthorizationToken`.

For more information about the `ReadFromRepository` and `PublishPackageVersion` permissions, see Repository Policies.

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.

```
{
    "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.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "DomainPolicyForOrganization",
        "Effect": "Allow",
        "Principal": "*",
        "Action": [
             "codeartifact:GetDomainPermissionsPolicy",
             "codeartifact:ListRepositoriesInDomain",
             "codeartifact:GetAuthorizationToken",
             "codeartifact:DescribeDomain",
             "codeartifact:CreateRepository"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": { "aws:PrincipalOrgID":["o-xxxxxxxxxxx"]}
        }
    }
}
```

For more information about using the `aws:PrincipalOrgID` condition key, see AWS Global Condition Context Keys in the *IAM User Guide*.

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

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.

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

Sample output:

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

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.

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)
- Tag domains (console)

Tag domains (CLI)

You can use the CLI to manage domain tags.

Topics

- Add tags to a domain (CLI)
- View tags for a domain (CLI)
- Edit tags for a domain (CLI)
- Remove tags from a domain (CLI)

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.

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:
>
> ```
> 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`*:

```
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:
>
> ```
> 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:

```
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:

```
{
    "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 domain using the `list-tags-for-resource` command.

Tag domains (console)

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

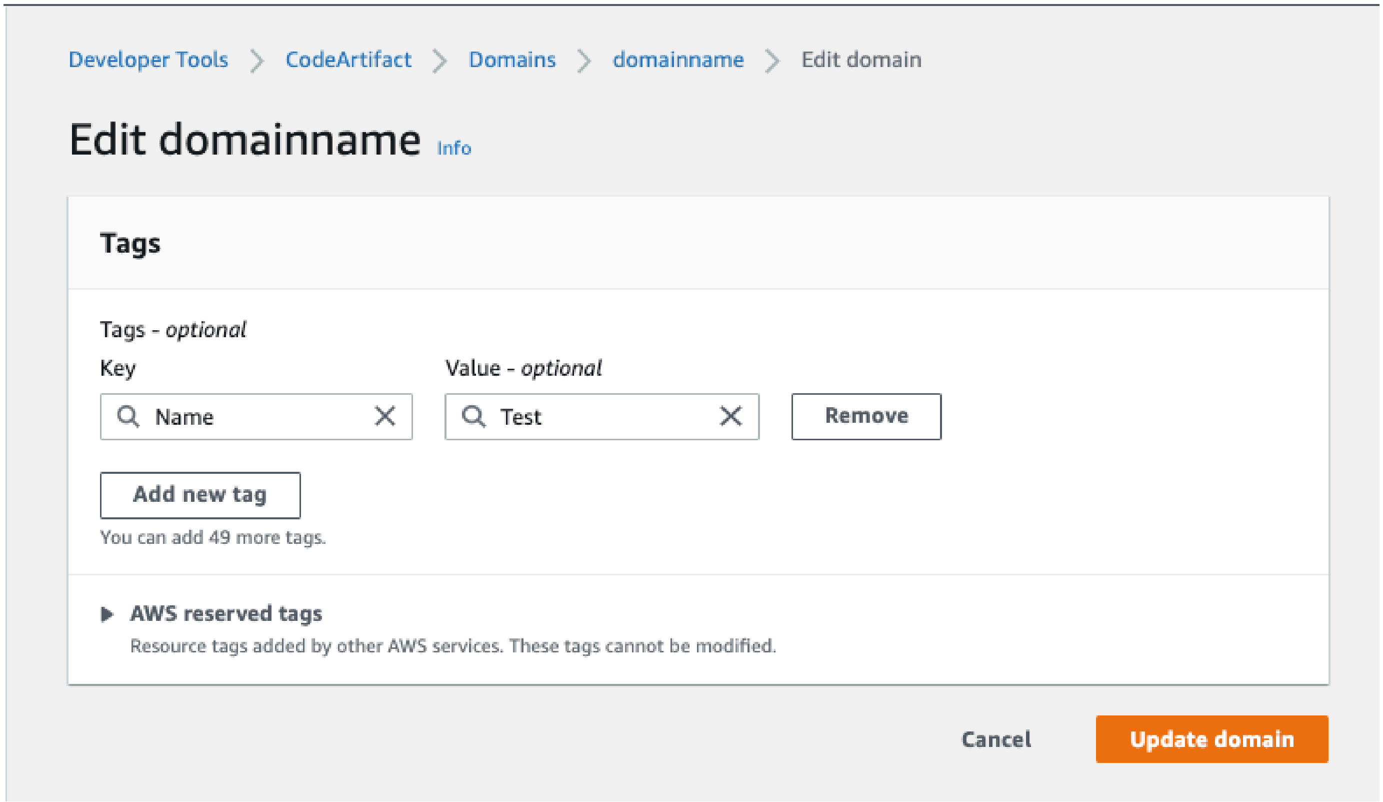
Topics

Add tags to a domain (console)

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

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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`**.


Developer Tools > CodeArtifact > Domains > domainname > Edit domain
Edit domainname Info
Tags
Tags - optional
Key
Value - optional
Name
Test
Remove
Add new tag
You can add 49 more tags.
AWS reserved tags
Resource tags added by other AWS services. These tags cannot be modified.
Cancel
Update domain

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.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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 domain.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.
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 Cargo

These topics describe how to use Cargo, the Rust package manager, with CodeArtifact.

> **Note**
>
> CodeArtifact only supports Cargo 1.74.0 and higher. Cargo 1.74.0 is the earliest version that supports authentication on a CodeArtifact repository.

Topics

Configure and use Cargo with CodeArtifact

You can use Cargo to publish and download crates from CodeArtifact repositories or to fetch crates from crates.io, the Rust community's crate registry. This topic describes how to configure Cargo to authenticate with and use a CodeArtifact repository.

Configure Cargo with CodeArtifact

To use Cargo to install and publish crates from AWS CodeArtifact, you'll first need to configure them with your CodeArtifact repository information. Follow the steps in one of the following procedure to configure Cargo with your CodeArtifact repository endpoint information and credentials.

Configure Cargo using the console instructions

You can use configuration instructions in the console to connect Cargo to your CodeArtifact repository. The console instructions provide a Cargo configuration customized for your CodeArtifact repository. You can use this custom configuration to set up Cargo without needing to find and fill in your CodeArtifact information.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.

2. In the navigation pane, choose **Repositories**, and then choose a repository to connect to Cargo.
3. Choose **View connection instructions**.
4. Choose your operating system.
5. Choose **Cargo**.
6. Follow the generated instructions to connect Cargo to your CodeArtifact repository.

Configure Cargo manually

If you cannot or do not want to use the configuration instructions from the console, you can use the following instructions to connect Cargo to your CodeArtifact repository manually.

macOS and Linux

In order to configure Cargo with CodeArtifact, you need to define your CodeArtifact repository as a registry in the Cargo configuration and provide credentials.

- Replace *`my_registry`* with your registry name.
- 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.
- Replace *`my_repo`* with your CodeArtifact repository name.

Copy the configuration to publish and download Cargo packages to your repository and save it in the `~/.cargo/config.toml` file for a system-level configuration or `.cargo/config.toml` for a project-level configuration:

```
[registries.my_registry]
index = "sparse+https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/cargo/my_repo/"
credential-provider = "cargo:token-from-stdout aws codeartifact get-authorization-token --domain my_domain --domain-owner 111122223333 --region us-west-2 --query authorizationToken --output text"

[registry]
default = "my_registry"
```

```
[source.crates-io]
replace-with = "my_registry"
```

Windows: Download packages only

In order to configure Cargo with CodeArtifact, you need to define your CodeArtifact repository as a registry in the Cargo configuration and provide credentials.

- Replace *my_registry* with your registry name.
- 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](#).
- Replace *my_repo* with your CodeArtifact repository name.

Copy the configuration to only download Cargo packages from your repository and save it in the `%USERPROFILE%\.cargo\config.toml` file for a system-level configuration or `.cargo\config.toml` for a project-level configuration:

```
[registries.my_registry]
index = "sparse+https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/cargo/my_repo/"
credential-provider = "cargo:token-from-stdout aws codeartifact get-authorization-
token --domain my_domain --domain-owner 111122223333 --region us-west-2 --query
 authorizationToken --output text"

[registry]
default = "my_registry"

[source.crates-io]
replace-with = "my_registry"
```

Windows: Publish and download packages

1. In order to configure Cargo with CodeArtifact, you need to define your CodeArtifact repository as a registry in the Cargo configuration and provide credentials.

    - Replace *my_registry* with your registry name.
    - 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.
- Replace *my_repo* with your CodeArtifact repository name.

Copy the configuration to publish and download Cargo packages to your repository and save it in the `%USERPROFILE%\.cargo\config.toml` file for a system-level configuration or `.cargo\config.toml` for a project-level configuration.

It is recommended that you use the credential provider `cargo:token`, which uses the credentials stored in your `~/.cargo/credentials.toml` file. You may run into an error during `cargo publish` if you use `cargo:token-from-stdout` because the Cargo client doesn't trim the authorization token properly during `cargo publish`.

```
[registries.my_registry]
index = "sparse+https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/cargo/my_repo/"
credential-provider = "cargo:token"

[registry]
default = "my_registry"

[source.crates-io]
replace-with = "my_registry"
```

2. To publish Cargo packages to your repository with Windows, you must use the CodeArtifact `get-authorization-token` command and Cargo `login` command to fetch an authorization token and your credentials.

   - Replace *my_registry* with your registry name as defined in `[registries.my_registry]`.
   - 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.

```
aws codeartifact get-authorization-token --domain my_domain --domain-
owner 111122223333 --region us-west-2 --query authorizationToken --output text |
 cargo login --registry my_registry
```

> **Note**
>
> The authorization token generated is valid for 12 hours. You will need to create a new one if 12 hours have passed since a token was created.

The `[registries.my_registry]` section in the preceding example defines a registry with *`my_registry`* and provides `index` and `credential-provider` information.

- `index` specifies the URL of the index for your registry, which is the CodeArtifact repository endpoint that ends with a `/`. The `sparse+` prefix is required for registries that are not Git repositories.

> **Note**
>
> To use a dualstack endpoint, use the `codeartifact.region.on.aws` endpoint.

- `credential-provider` specifies the credential provider for the given registry. If `credential-provider` isn't set, the providers in `registry.global-credential-providers` will be used. By setting `credential-provider` to `cargo:token-from-stdout`, the Cargo client will fetch new authorization token automatically when publishing or downloading from your CodeArtifact repository, therefore you don't need to manually refresh the authorization token every 12 hours.

The `[registry]` section defines the default registry used.

- `default` specifies the name of the registry defined in `[registries.my_registry]`, to use by default when publishing or downloading from your CodeArtifact repository.

The `[source.crates-io]` section defines the default registry used when one isn't specified.

- `replace-with = "my_registry"` replaces the public registry, crates.io with your CodeArtifact repository defined in `[registries.my_registry]`. This configuration is recommended if you need to request packages from the external connection such as crates.io.

  To get all of the benefits of CodeArtifact, such as the package origin control that prevents dependency confusion attacks, it is recommended that you use source replacement. With the source replacement, CodeArtifact proxies all requests to the external connection and copies the package from the external connection to your repository. Without the source replacement, the Cargo client will directly retrieve the package based on the configuration in your `Cargo.toml` file in your project. If a dependency is not marked with `registry=my_registry`, the Cargo client will retrieve it directly from crates.io without communicating with your CodeArtifact repository.

  > **Note**
  >
  > If you start using source replacement and then update your configuration file to not use source replacement, you may encounter errors. The opposite scenario may also lead to errors. Therefore, it is recommended that you avoid changing the configuration for your project.

Installing Cargo crates

Use the following procedures to install Cargo crates from a CodeArtifact repository or from crates.io.

Install Cargo crates from CodeArtifact

You can use the Cargo (`cargo`) CLI to quickly install a specific version of a Cargo crate from your CodeArtifact repository.

To install Cargo crates from a CodeArtifact repository with `cargo`

1. If you haven't, follow the steps in Configure and use Cargo with CodeArtifact to configure the `cargo` CLI to use your CodeArtifact repository with proper credentials.
2. Use the following command to install Cargo crates from CodeArtifact:

   ```
   cargo add my_cargo_package@1.0.0
   ```

For more information, see cargo add in *The Cargo Book*.

Publishing Cargo crates to CodeArtifact

Use the following procedure to publish Cargo crates to a CodeArtifact repository using the `cargo` CLI.

1. If you haven't, follow the steps in Configure and use Cargo with CodeArtifact to configure the `cargo` CLI to use your CodeArtifact repository with proper credentials.
2. Use the following command to publish Cargo crates to a CodeArtifact repository:

   ```
   cargo publish
   ```

   For more information, see cargo publish in *The Cargo Book*.

Cargo command support

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

Contents

Supported commands that require accessing the registry

This section lists Cargo commands where the Cargo client requires access to the registry it's been configured with. These commands have been verified to function correctly when invoked against a CodeArtifact repository.

Command	Description
build	Builds local packages and their dependencies.

Command	Description
check	Checks local packages and their dependencies for errors.
fetch	Fetches the dependencies of a package.
publish	Publishes a package to the registry.

Unsupported commands

These Cargo commands are not supported by CodeArtifact repositories.

Command	Description	
owner	Manages the owners of the crate on the registry.	
search	Searches for packages in the registry.	

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.
- Latest **Clojure** version: 1.11.1 has also been tested.

Topics

- Use CodeArtifact with Gradle
- Use CodeArtifact with mvn
- Use CodeArtifact with deps.edn
- Publishing with curl
- Use Maven checksums
- Use Maven snapshots
- Requesting Maven packages from upstreams and external connections
- Maven troubleshooting

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, follow these instructions to consume Maven packages from, and publish new packages to, a CodeArtifact repository.

Topics

- Fetch dependencies
- Fetch plugins
- Publish artifacts
- Run a Gradle build in IntelliJ IDEA

Fetch dependencies

To fetch dependencies from CodeArtifact in a Gradle build, use the following procedure.

To fetch dependencies from CodeArtifact in a Gradle build

1. If you haven't, create and store a CodeArtifact auth token in an environment variable by following the procedure in Pass an auth token using an environment variable.
2. Add a `maven` section to the `repositories` section in the project `build.gradle` file.

   ```
   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 preceding example 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:

   ```
   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:

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

   The `credentials` object in the preceding example includes the CodeArtifact auth token you created in Step 1 that Gradle uses to authenticate to CodeArtifact.

Note

To use a dualstack endpoint, use the `codeartifact.region.on.aws` endpoint.

3. (Optional) - 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.
4. After you configure the repository, you can add project dependencies to the `dependencies` section with standard Gradle syntax.

```
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, use the following procedure.

To pull plugins from a CodeArtifact repository

1. If you haven't, create and store a CodeArtifact auth token in an environment variable by following the procedure in Pass an auth token using an environment variable.
2. Add a `pluginManagement` block to your `settings.gradle` file. The `pluginManagement` block must appear before any other statements in `settings.gradle`, see the following snippet:

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

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.

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.

```
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-111122223333.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 111122223333
 --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:

```
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`.

```
pluginManagement {
    repositories {
        maven {
            name 'my_repo'
            url
 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
                password "$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 `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:

```
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`.

```
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:

```
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`.

```
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.

Topics

- Fetch dependencies
- Publish artifacts
- Publish third-party artifacts
- Restrict Maven dependency downloads to a CodeArtifact repository
- Apache Maven Project information

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. If you haven't, create and store a CodeArtifact auth token in an environment variable as described in Pass an auth token using an environment variable to set up authentication to your CodeArtifact repository.
2. 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.

   ```
   <settings>
   ...
       <servers>
           <server>
               <id>codeartifact</id>
               <username>aws</username>
               <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
           </server>
       </servers>
   ...
   ```

```
</settings>
```

3. 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:

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

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

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

   > **Note**
   >
   > To use a dualstack endpoint, use the `codeartifact.`*`region`*`.on.aws` endpoint.

   Add the repository endpoint to `settings.xml` as follows.

   ```
   <settings>
   ...
       <profiles>
           <profile>
               <id>default</id>
               <repositories>
                   <repository>
                       <id>codeartifact</id>
                       <url>https://my_domain-111122223333.d.codeartifact.us-
   west-2.amazonaws.com/maven/my_repo/</url>
                   </repository>
               </repositories>
           </profile>
       </profiles>
   ```

```
    <activeProfiles>
        <activeProfile>default</activeProfile>
    </activeProfiles>
    ...
</settings>
```

Or, you can add the <repositories> section to a project POM file to use CodeArtifact for that project only.

```
<project>
...
    <repositories>
        <repository>
            <id>codeartifact</id>
            <name>codeartifact</name>
            <url>https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/</url>
        </repository>
    </repositories>
...
</project>
```

Important

You can use any value in the <id> element, but it must be the same in both the <server> and <repository> elements. This enables the specified credentials to be included in requests to CodeArtifact.

After you make these configuration changes, you can build the project.

```
mvn compile
```

Maven logs the full URL of all the dependencies it downloads to the console.

```
[INFO] ------------------< com.example.example:myapp >-------------------
[INFO] Building myapp 1.0
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.pom
```

```
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.pom (11 kB at 3.9 kB/s)
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom (68 kB at 123
 kB/s)
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.jar
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.jar (54 kB at 134 kB/s)
```

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. If you haven't, create and store a CodeArtifact auth token in an environment variable as described in Pass an auth token using an environment variable to set up authentication to your CodeArtifact repository.
2. 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>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
...
</settings>
```

3. Add a `<distributionManagement>` section to your project's `pom.xml`.

```
<project>
```

```
...
     <distributionManagement>
         <repository>
             <id>codeartifact</id>
             <name>codeartifact</name>
             <url>https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/</url>
         </repository>
     </distributionManagement>
...
</project>
```

After you make these configuration changes, you can build the project and publish it to the specified repository.

```
mvn deploy
```

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

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333
 --repository my_repo --format maven \
  --namespace com.company.framework --package my-package-name
```

Sample output:

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

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. If you haven't, create and store a CodeArtifact auth token in an environment variable as described in Pass an auth token using an environment variable to set up authentication to your CodeArtifact repository.
2. Create a `~/.m2/settings.xml` file with the following contents:

```
<settings>
    <servers>
        <server>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
</settings>
```

3. Run the `mvn deploy:deploy-file` command:

```
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*.

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 `mirrors` element to `settings.xml` to make `mvn` always use your CodeArtifact repository.

```
<settings>
  ...
    <mirrors>
      <mirror>
        <id>central-mirror</id>
        <name>CodeArtifact Maven Central mirror</name>
        <url>https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_repo/</url>
        <mirrorOf>central</mirrorOf>
      </mirror>
    </mirrors>
  ...
</settings>
```

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.

```
<settings>
...
  <profiles>
    <profile>
      <pluginRepositories>
        <pluginRepository>
          <id>codeartifact</id>
          <name>CodeArtifact Plugins</name>
          <url>https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_repo/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
...
</settings>
```

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

```
<profiles>
   <profile>
     <id>default</id>
     ...
     <pluginRepositories>
       <pluginRepository>
         <id>central-plugins</id>
         <name>Central Plugins</name>
         <url>https://repo.maven.apache.org/maven2/</url>
         <releases>
             <enabled>true</enabled>
         </releases>
         <snapshots>
             <enabled>true</enabled>
         </snapshots>
       </pluginRepository>
     </pluginRepositories>
   ....
   </profile>
 </profiles>
```

Apache Maven Project information

For more information about Maven, see these topics on the Apache Maven Project website:

- Setting up Multiple Repositories
- Settings Reference
- Distribution Management
- Profiles

Use CodeArtifact with deps.edn

You use `deps.edn` with `clj` to manage dependencies for Clojure projects. This section shows how to configure `deps.edn` to use a CodeArtifact repository.

Topics

- Fetch dependencies
- Publish artifacts

Fetch dependencies

To configure `Clojure` to fetch dependencies from a CodeArtifact repository, you must edit the Maven configuration file, `settings.xml`.

1. In `settings.xml`, add a `<servers>` section with a reference to the `CODEARTIFACT_AUTH_TOKEN` environment variable so that Clojure passes the token in HTTP requests.

   > **Note**
   >
   > Clojure expects the settings.xml file to be located at `~/.m2/settings.xml`. If elsewhere, create the file in this location.

   ```
   <settings>
   ...
       <servers>
           <server>
               <id>codeartifact</id>
               <username>aws</username>
               <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
           </server>
   ```

```
    </servers>
...
</settings>
```

2. If you do not have one already, generate a POM xml for your project using `clj -Spom`.
3. In your `deps.edn` configuration file, add a repository matching the server id from Maven `settings.xml`.

```
:mvn/repos {
  "clojars" nil
  "central" nil
  "codeartifact" {:url "https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/"}
}
```

> **Note**
>
> - `tools.deps` guarantees that the `central` and `clojars` repositories will be checked first for Maven libraries. Afterward, the other repositories listed in `deps.edn` will be checked.
> - To prevent downloading from Clojars and Maven Central directly, `central` and `clojars` need to be set to `nil`.

Make sure you have the CodeArtifact Auth token in an environment variable (see [Pass an auth token using an environment variable](#)). When building the package after these changes, dependencies in `deps.edn` will be fetched from CodeArtifact.

> **Note**
>
> To use a dualstack endpoint, use the `codeartifact.region.on.aws` endpoint.

Publish artifacts

1. Update your Maven settings and `deps.edn` to include CodeArtifact as a maven-recognized server (see [Fetch dependencies](#)). You can use a tool such as [deps-deploy](#) to upload artifacts to CodeArtifact.

2. In your `build.clj`, add a `deploy` task to upload required artifacts to the previously setup `codeartifact` repository.

   ```
   (ns build
   (:require [deps-deploy.deps-deploy :as dd]))

   (defn deploy [_]
     (dd/deploy {:installer :remote
            :artifact "PATH_TO_JAR_FILE.jar"
            :pom-file "pom.xml" ;; pom containing artifact coordinates
            :repository "codeartifact"}))
   ```

3. Publish the artifact by running the command: `clj -T:build deploy`

For more information on modifying default repositories, see Modifying the default repositories in the *Clojure Deps and CLI Reference Rationale*.

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 and return to these steps.
2. Use the following `curl` command to publish the JAR to a CodeArtifact repository:

   In each of the `curl` commands in this procedure, replace the following placeholders:

   - Replace *`my_domain`* with your CodeArtifact domain name.
   - Replace *`111122223333`* with the ID of the owner of your CodeArtifact domain.
   - Replace *`us-west-2`* with the region in which your CodeArtifact domain is located.
   - Replace *`my_repo`* with your CodeArtifact repository name.

   ```
   curl --request PUT https://my_domain-111122223333.d.codeartifact.us-
   west-2.amazonaws.com/maven/my_repo/com/mycompany/app/my-app/1.0/my-app-1.0.jar \
   ```

```
     --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/
octet-stream" \
     --data-binary @my-app-1.0.jar
```

> **Important**
>
> You must prefix the value of the `--data-binary` parameter with a `@` character. When putting the value in quotation marks, the `@` must be included inside the quotation marks.

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 @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:

```
curl --request PUT
 https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/com/mycompany/app/my-app/maven-metadata.xml \
     --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/
octet-stream" \
     --data-binary @maven-metadata.xml
```

   The following is an example of the contents of a `maven-metadata.xml` file:

```
<metadata modelVersion="1.1.0">
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <versioning>
```

```
        <latest>1.0</latest>
        <release>1.0</release>
        <versions>
            <version>1.0</version>
        </versions>
        <lastUpdated>20200731090423</lastUpdated>
    </versioning>
</metadata>
```

b. 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 \
    --format maven \
    --namespace com.mycompany.app \
    --package my-app \
    --versions 1.0 \
    --target-status Published
```

If you only have an artifact's JAR file, you can publish a consumable package version to a CodeArtifact repository using `mvn`. This can be useful if you do not have access to the artifact's source code or POM. See Publish third-party artifacts for details.

Use 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.

Checksum storage

CodeArtifact does not store Maven checksums as assets. This means that checksums do not appear as individual assets in the output of the ListPackageVersionAssets API. Instead, checksums computed by CodeArtifact are available for each asset in all supported checksum types. For example, part of the response of calling ListPackageVersionAssets on the Maven package version `commons-lang:commons-lang 2.1` is:

```
{
    "name": "commons-lang-2.1.jar",
    "size": 207723,
    "hashes": {
        "MD5": "51591549f1662a64543f08a1d4a0cf87",
        "SHA-1": "4763ecc9d78781c915c07eb03e90572c7ff04205",
        "SHA-256": "2ded7343dc8e57decd5e6302337139be020fdd885a2935925e8d575975e480b9",
        "SHA-512":
 "a312a5e33b17835f2e82e74ab52ab81f0dec01a7e72a2ba58bb76b6a197ffcd2bb410e341ef7b3720f3b595ce49fd
    }
},
{
    "name": "commons-lang-2.1.pom",
    "size": 9928,
    "hashes": {
        "MD5": "8e41bacdd69de9373c20326d231c8a5d",
        "SHA-1": "a34d992202615804c534953aba402de55d8ee47c",
        "SHA-256": "f1a709cd489f23498a0b6b3dfbfc0d21d4f15904791446dec7f8a58a7da5bd6a",
        "SHA-512":
 "1631ce8fe4101b6cde857f5b1db9b29b937f98ba445a60e76cc2b8f2a732ff24d19b91821a052c1b56b73325104e9
    }
},
        {
    "name": "maven-metadata.xml",
    "size": 121,
    "hashes": {
        "MD5": "11bb3d48d984f2f49cea1e150b6fa371",
        "SHA-1": "7ef872be17357751ce65cb907834b6c5769998db",
        "SHA-256": "d04d140362ea8989a824a518439246e7194e719557e8d701831b7f5a8228411c",
        "SHA-512":
 "001813a0333ce4b2a47cf44900470bc2265ae65123a8c6b5ac5f2859184608596baa4d8ee0696d0a497755dade0f6
    }
}
```

Even though checksums are not stored as assets, Maven clients can still publish and download checksums at the expected locations. For example, if `commons-lang:commons-lang 2.1` was in a repository called `maven-repo`, the URL path for the SHA-256 checksum of the JAR file would be:

```
/maven/maven-repo/commons-lang/commons-lang/2.1/commons-lang-2.1.jar.sha256
```

If you're uploading existing Maven packages (for example, packages previously stored in Amazon S3) to CodeArtifact using a generic HTTP client such as `curl`, it's not necessary to upload the checksums. CodeArtifact will generate them automatically. If you want to verify that the assets have been uploaded correctly, you can use the ListPackageVersionAssets API operation to compare the checksums in the response to the original checksum values for each asset.

Checksum mismatches during publishing

Apart from assets and checksums, Maven artifacts also contain a `maven-metadata.xml` file. The normal publishing sequence for a Maven package is for all assets and checksums to be uploaded first, followed by `maven-metadata.xml`. For example, the publishing sequence for the Maven package version `commons-lang 2.1` described previously, assuming the client was configured to publish SHA-256 checksum files, would be:

```
PUT commons-lang-2.1.jar
PUT commons-lang-2.1.jar.sha256
PUT commons-lang-2.1.pom
PUT commons-lang-2.1.pom.sha256
PUT maven-metadata.xml
PUT maven-metadata.xml.sha256
```

When uploading the checksum file for an asset, such as a JAR file, the checksum upload request will fail with a **400 (Bad Request)** response if there's a mismatch between the uploaded checksum value and the checksum value calculated by CodeArtifact. If the corresponding asset doesn't exist, the request will fail with a **404 (Not Found)** response. To avoid this error, you must first upload the asset, and then upload the checksum.

When `maven-metadata.xml` is uploaded, CodeArtifact normally changes the status of the Maven package version from `Unfinished` to `Published`. If a checksum mismatch is detected for any asset, CodeArtifact will return a **400 (Bad Request)** in response to the `maven-metadata.xml` publishing request. This error may cause the client to stop uploading files for that package version. If this occurs, and the `maven-metadata.xml` file is not uploaded, any assets of the package version already uploaded cannot be downloaded. This is because the package version’s status is not set to Published and remains Unfinished.

CodeArtifact allows adding further assets to a Maven package version even after maven-metadata.xml has been uploaded and the package version status has been set to Published. In this status, a request to upload a mismatched checksum file will also fail with a **400 (Bad Request)** response. However, because the package version status has already been set to Published, you can download any asset from the package, including those for which the checksum file upload failed. When downloading a checksum for an asset where the checksum file upload failed, the checksum value that the client receives will be the checksum value calculated by CodeArtifact based on the uploaded asset data.

CodeArtifact checksum comparisons are case sensitive, and the checksums calculated by CodeArtifact are formatted in lowercase. Therefore, if the checksum 909FA780F76DA393E992A3D2D495F468 is uploaded, it will fail with a checksum mismatch because CodeArtifact does not treat it as equal to 909fa780f76da393e992a3d2d495f468.

Recovering from checksum mismatches

If a checksum upload fails as a result of a checksum mismatch, try one of the following to recover:

- Run the command that publishes the Maven artifact again. This might work if a network issue corrupted the checksum file. If this resolves the network issue, the checksum matches and the download is successful.
- Delete the package version and then republish it. For more information, see DeletePackageVersions in the *AWS CodeArtifact API Reference*.

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's 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. Unique snapshots that use a time-based version number are the only snapshots that are supported. CodeArtifact doesn't support non-unique snapshots that are generated by Maven 2 clients. You can publish a supported Maven snapshot to any CodeArtifact repository.

Topics

- Snapshot publishing in CodeArtifact
- Consuming snapshot versions
- Deleting snapshot versions
- Snapshot publishing with curl
- Snapshots and external connections
- Snapshots and upstream repositories

Snapshot publishing in CodeArtifact

AWS CodeArtifact supports the request patterns that clients, such as `mvn`, use when publishing snapshots. Because of this, you can follow the documentation for your build tool or package manager without having a detailed understanding of how Maven snapshots are published. If you're doing something more complex, this section describes in detail how CodeArtifact handles snapshots.

When a Maven snapshot is published to a CodeArtifact repository, 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 package version status is set to `Published` and the status of the build that contains the previous version is set to `Unlisted`. This behavior applies only to Maven package versions where the package version has `-SNAPSHOT` as a suffix.

For example, snapshot versions of a maven package called `com.mycompany.myapp:pkg-1` are uploaded to a CodeArtifact repository called `my-maven-repo`. The snapshot version is `1.0-SNAPSHOT`. So far, no versions of `com.mycompany.myapp:pkg-1` have been published. First, the assets of the initial build are published at these paths:

```
PUT maven/my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210728.194552-1.jar
PUT maven/my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210728.194552-1.pom
```

Note that the timestamp `20210728.194552-1` is generated by the client publishing the snapshot builds.

After the .pom and .jar files are uploaded, the only version of `com.mycompany.myapp:pkg-1` that exists in the repository is `1.0-20210728.194552-1`. This happens even though the version specified in the preceding path is `1.0-SNAPSHOT`. The package version status at this point is `Unfinished`.

```
aws codeartifact list-package-versions --domain my-domain --repository \
  my-maven-repo --package pkg-1 --namespace com.mycompany.myapp --format maven
{
    "versions": [
        {
            "version": "1.0-20210728.194552-1",
            "revision": "GipMW+599JmwTcTLaXo9YvDsVQ2bcrrk/02rWJhoKUU=",
            "status": "Unfinished"
        }
    ],
    "defaultDisplayVersion": null,
    "format": "maven",
    "package": "pkg-1",
    "namespace": "com.mycompany.myapp"
}
```

Next, the client uploads the `maven-metadata.xml` file for the package version:

```
PUT my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/maven-metadata.xml
```

When the maven-metadata.xml file is uploaded successfully, CodeArtifact creates the `1.0-SNAPSHOT` package version and sets the `1.0-20210728.194552-1` version to `Unlisted`.

```
aws codeartifact list-package-versions --domain my-domain --repository \
  my-maven-repo --package pkg-1 --namespace com.mycompany.myapp --format maven
{
    "versions": [
        {
            "version": "1.0-20210728.194552-1",
            "revision": "GipMW+599JmwTcTLaXo9YvDsVQ2bcrrk/02rWJhoKUU=",
            "status": "Unlisted"
        },
        {
            "version": "1.0-SNAPSHOT",
            "revision": "tWu8n3IX5HR82vzVZQAxlwcvvA4U/+S80edWNAkil24=",
            "status": "Published"
        }
    ],
    "defaultDisplayVersion": "1.0-SNAPSHOT",
    "format": "maven",
    "package": "pkg-1",
    "namespace": "com.mycompany.myapp"
}
```

At this point, the snapshot version `1.0-SNAPSHOT` can be consumed in a build. While there are two versions of `com.mycompany.myapp:pkg-1` in the repository `my-maven-repo`, they both contain the same assets.

```
aws codeartifact list-package-version-assets --domain my-domain --repository \
  my-maven-repo --format maven --namespace com.mycompany.myapp \
 --package pkg-1 --package-version 1.0-SNAPSHOT--query 'assets[*].name'
[
     "pkg-1-1.0-20210728.194552-1.jar",
     "pkg-1-1.0-20210728.194552-1.pom"
]
```

Running the same `list-package-version-assets` command as shown previously with the `--package-version` parameter changed to `1.0-20210728.194552-1` results in an identical output.

As additional builds of `1.0-SNAPSHOT` are added to the repository, a new `Unlisted` package version is created for each new build. The assets of the version `1.0-SNAPSHOT` are updated each time so that the version always refers to the latest build for that version. Updating the `1.0-SNAPSHOT` with the latest assets is initiated by uploading the `maven-metadata.xml` file for the new build.

Consuming snapshot versions

If you request a snapshot, the version with the 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 using the build version number (for example, `1.0-20210728.194552-1`) instead of the snapshot version (for example, `1.0-SNAPSHOT`) in the URL path. To see the build versions of a Maven snapshot, use the ListPackageVersions API in the *CodeArtifact API Guide* and set the status parameter to `Unlisted`.

Deleting snapshot versions

To delete all build versions of a Maven snapshot, use the DeletePackageVersions API, specifying the versions that you want to delete.

Snapshot publishing with curl

If you have existing snapshot versions stored in Amazon Simple Storage Service (Amazon S3) or another artifact repository product, you may want to republish them to AWS CodeArtifact. Because of how CodeArtifact supports Maven snapshots (see Snapshot publishing in CodeArtifact), publishing snapshots with a generic HTTP client such as `curl` is more complex than publishing Maven release versions as described in Publishing with curl. Note that this section isn't relevant if you're building and deploying snapshot versions with a Maven client such as `mvn` or `gradle`. You need to follow the documentation for that client.

Publishing a snapshot version involves publishing one or more builds of a snapshot version. In CodeArtifact, if there are *n* builds of a snapshot version, there will be *n + 1* CodeArtifact versions: *n* build versions all with a status of `Unlisted`, and one snapshot version (the latest published build) with a status of `Published`. The snapshot version (that is, the version with a version string that contains "-SNAPSHOT") contains an identical set of assets to the latest published build. The simplest way to create this structure using `curl` is as follows:

1. Publish all assets of all builds using `curl`.
2. Publish the `maven-metadata.xml` file of the last build (that is, the build with the most-recent date-time stamp) with `curl`. This will create a version with "`-SNAPSHOT`" in the version string and with the correct set of assets.
3. Use the UpdatePackageVersionsStatus API to set the status of all the non-latest build versions to `Unlisted`.

Use the following `curl` commands to publish snapshot assets (such as .jar and .pom files) for the snapshot version `1.0-SNAPSHOT` of a package `com.mycompany.app:pkg-1`:

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210729.171330-2.jar \
     --data-binary @pkg-1-1.0-20210728.194552-1.jar
```

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210729.171330-2.pom \
     --data-binary @pkg-1-1.0-20210728.194552-1.pom
```

When using these examples:

- Replace *my_domain* with your CodeArtifact domain name.
- Replace *111122223333* with the AWS account ID of the owner of your CodeArtifact domain.
- Replace *us-west-2* with the AWS Region in which your CodeArtifact domain is located.
- Replace *my_maven_repo* with your CodeArtifact repository name.

Important

You must prefix the value of the `--data-binary` parameter with the `@` character. When putting the value in quotation marks, the `@` must be included inside the quotation marks.

You may have more than two assets to upload for each build. For example, there might be Javadoc and source JAR files in addition to the main JAR and `pom.xml`. It is not necessary to publish checksum files for the package version assets because CodeArtifact automatically generates checksums for each uploaded asset. To verify the assets were uploaded correctly, fetch the generated checksums using the `list-package-version-assets` command and compare those to the original checksums. For more information about how CodeArtifact handles Maven checksums, see Use Maven checksums.

Use the following curl command to publish the `maven-metadata.xml` file for the latest build version:

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/maven-metadata.xml \
     --data-binary @maven-metadata.xml
```

The maven-metadata.xml file must reference at least one of the assets in the latest build version in the <snapshotVersions> element. In addition, the <timestamp> value must be present and must match the timestamp in the asset file names. For example, for the 20210729.171330-2 build published previously, the contents of maven-metadata.xml would be:

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <groupId>com.mycompany.app</groupId>
  <artifactId>pkg-1</artifactId>
  <version>1.0-SNAPSHOT</version>
  <versioning>
    <snapshot>
      <timestamp>20210729.171330</timestamp>
      <buildNumber>2</buildNumber>
    </snapshot>
    <lastUpdated>20210729171330</lastUpdated>
    <snapshotVersions>
      <snapshotVersion>
        <extension>jar</extension>
        <value>1.0-20210729.171330-2</value>
        <updated>20210729171330</updated>
      </snapshotVersion>
      <snapshotVersion>
        <extension>pom</extension>
        <value>1.0-20210729.171330-2</value>
        <updated>20210729171330</updated>
      </snapshotVersion>
    </snapshotVersions>
  </versioning>
</metadata>
```

After maven-metadata.xml has been published, the last step is to set all the other build versions (that is, all the build versions apart from the latest build) to have a package version status of Unlisted. For example, if the 1.0-SNAPSHOT version has two builds, with the first build being 20210728.194552-1, the command to set that build to Unlisted is:

```
aws codeartifact update-package-versions-status --domain my-domain --domain-owner
 111122223333 \
   --repository my-maven-repo --format maven --namespace com.mycompany.app --package
 pkg-1 \
   --versions 1.0-20210728.194552-1 --target-status Unlisted
```

Snapshots and external connections

Maven snapshots cannot be fetched from a Maven public repository through an external connection. AWS CodeArtifact only supports importing Maven release versions.

Snapshots and upstream repositories

In general, Maven snapshots work in the same way as Maven release versions when used with upstream repositories, but there is a limitation if you plan on publishing snapshots of the same package version to two repositories which have an upstream relationship. For example, say that there are two repositories in an AWS CodeArtifact domain, `R` and `U`, where `U` is an upstream of `R`. If you publish a new build in `R`, when a Maven client requests the latest build of that snapshot version, CodeArtifact returns the latest version from `U`. This can be unexpected since the latest version is now in `R`, not `U`. There are two ways to avoid this:

1. Don't publish builds of a snapshot version such as `1.0-SNAPSHOT` in `R`, if `1.0-SNAPSHOT` exists in `U`.
2. Use CodeArtifact package origin controls to disable upstreams on that package in `R`. The latter will allow you to publish builds of`1.0-SNAPSHOT` in `R`, but it will also prevent `R` from getting any other versions of that package from `U` that aren't already retained.

Requesting Maven packages from upstreams and external connections

Importing standard asset names

When importing a Maven package version from a public repository, such as Maven Central, AWS CodeArtifact attempts to import all the assets in that package version. As described in Requesting a package version with upstream repositories, importing occurs when:

- A client requests a Maven asset from a CodeArtifact repository.
- The package version is not already present in the repository or its upstreams.
- There is a reachable external connection to a public Maven repository.

Even though the client may have only requested one asset, CodeArtifact attempts to import all the assets it can find for that package version. How CodeArtifact discovers which assets are available for a Maven package version depends on the particular public repository. Some public Maven repositories support requesting a list of assets, but others do not. For repositories that do not provide a way to list assets, CodeArtifact generates a set of asset names that are likely to exist. For example, when any asset of the Maven package version `junit 4.13.2` is requested, CodeArtifact will attempt to import the following assets:

- `junit-4.13.2.pom`
- `junit-4.13.2.jar`
- `junit-4.13.2-javadoc.jar`
- `junit-4.13.2-sources.jar`

Importing non-standard asset names

When a Maven client requests an asset that doesn't match one of the patterns described above, CodeArtifact checks to see if that asset is present in the public repository. If the asset is present, it will be imported and added to the existing package version record, if one exists. For example, the Maven package version `com.android.tools.build:aapt2 7.3.1-8691043` contains the following assets:

- `aapt2-7.3.1-8691043.pom`
- `aapt2-7.3.1-8691043-windows.jar`
- `aapt2-7.3.1-8691043-osx.jar`
- `aapt2-7.3.1-8691043-linux.jar`

When a client requests the POM file, if CodeArtifact is unable to list the package version's assets, the POM will be the only asset imported. This is because none of the other assets match the standard asset name patterns. However, when the client requests one of the JAR assets, that asset will be imported and added to the existing package version stored in CodeArtifact. The package versions in both the most-downstream repository (the repository the client made the request against) and the repository with the external connection attached will be updated to contain the new asset, as described in Package retention from upstream repositories.

Normally, once a package version is retained in a CodeArtifact repository, it is not affected by changes in upstream repositories. For more information, see Package retention from upstream repositories. However, the behavior for Maven assets with non-standard names described earlier is an exception to this rule. While the downstream package version won't change without an additional asset being requested by a client, in this situation, the retained package version is modified after initially being retained and so is not immutable. This behavior is necessary because Maven assets with non-standard names would otherwise not be accessible through CodeArtifact. The behavior also enables if they are added to a Maven package version on a public repository after the package version was retained in a CodeArtifact repository.

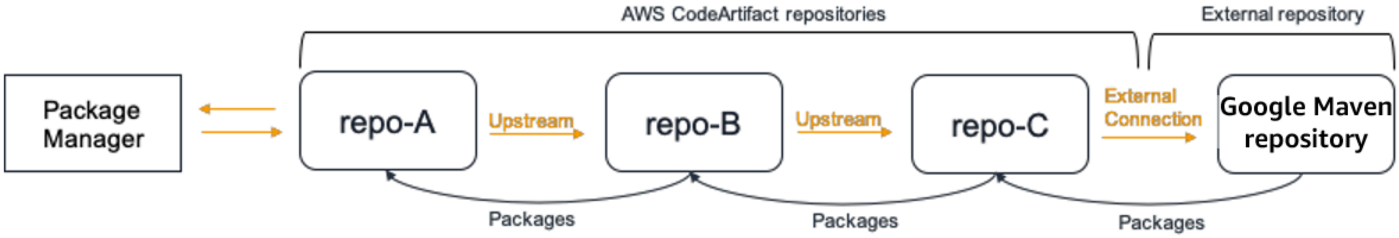
Checking asset origins

When adding a new asset to a previously retained Maven package version, CodeArtifact confirms the origin of the retained package version is the same as origin of the new asset. This prevents creating a “mixed” package version where different assets originate from different public repositories. Without this check, asset mixing could occur if a Maven package version is published to more than one public repository and those repositories are part of a CodeArtifact repository’s upstream graph.

Importing new assets and package version status in upstream repositories

The package version status of package versions in upstream repositories can prevent CodeArtifact from retaining those versions in downstream repositories.

For example, let's say a domain has three repositories: `repo-A`, `repo-B`, and `repo-C`, where `repo-B` is an upsteam of `repo-A` and `repo-C` is upstream of `repo-B`.



Package version `7.3.1` of Maven package `com.android.tools.build:aapt2` is present in `repo-B` and has a status of `Published`. It is not present in `repo-A`. If a client requests an asset of this package version from `repo-A`, the response will be a 200 (OK) and Maven package version `7.3.1` will be retained in `repo-A`. However, if the status of package version `7.3.1` in `repo-B` is `Archived` or `Disposed`, the response will be 404 (Not Found) because the assets of package versions in those two statuses are not downloadable.

Note that setting the package origin control to `upstream=BLOCK` for `com.android.tools.build:aapt2` in `repo-A`, `repo-B`, and `repo-C` will prevent new assets from being fetched for all versions of that package from `repo-A`, regardless of the package version status.

Maven troubleshooting

The following information might help you troubleshoot common issues with Maven and CodeArtifact.

Disable parallel puts to fix error 429: Too Many Requests

Starting with version 3.9.0, Maven uploads package artifacts in parallel (up to 5 files at a time). This can cause CodeArtifact to occassionally respond with an error response code 429 (Too Many Requests). If you encounter this error, you can disable parallel puts to fix it.

To disable parallel puts, set the `aether.connector.basic.parallelPut` property to `false` in your profile in your `settings.xml` file as shown by the following example:

```
<settings>
    <profiles>
        <profile>
            <id>default</id>
            <properties>
                <aether.connector.basic.parallelPut>false</
aether.connector.basic.parallelPut>
            </properties>
        </profile>
    </profiles>
<settings>
```

For more information, see Artifact Resolver Configuration Options in the Maven documentation.

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

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

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.

Important

If you are using npm 10.x or newer, you must use AWS CLI version 2.9.5 or newer to successfully run the `aws codeartifact login` command.

```
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.

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.

```
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.
   - Replace *`my_repo`* with your CodeArtifact repository name.

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

   The following URL is an example repository endpoint.

   ```
   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.

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

> **Note**
>
> To use a dualstack endpoint, use the `codeartifact.region.on.aws` endpoint.

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

```
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`.

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

Example npm configuration file (`.npmrc`)

The following is an example `.npmrc` file after following the preceding instructions to set the CodeArtifact registry endpoint, add an authentication token, and configure `always-auth`.

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

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.

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 request GET https://<domain>.d.codeartifact.us-west-2.amazonaws.com/npm/
shared/-/ping?write=true
npm http 200 https:///npm/shared/-/ping?write=true
Ping success: {}
npm timing npm Completed in 716ms
npm info ok
```

The `-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/
```

Troubleshooting slow installs with npm 8.x or higher

There is a known issue in npm versions 8.x and greater where if a request is made to a package repository, and the repository redirects the client to Amazon S3 instead of streaming the assets directly, the npm client can hang for several minutes per dependency.

Because CodeArtifact repositories are designed to always redirect the request to Amazon S3, sometimes this issue occurs, which causes long build times due to long npm install times. Instances of this behavior will present themselves as a progress bar showing for several minutes.

To avoid this issue, use either the `--no-progress` or `progress=false` flags with `npm` cli commands, as shown in the following example.

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

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

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.
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.
   - 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:

   - 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 refresh the token periodically. For more information about the authorization token created with the `login` command, see Tokens created with the `login` command.

4. **For npm 7.X and 8.X**, you must add `always-auth=true` to your ~/.npmrc file to use Yarn.

   - Open your ~/.npmrc file in a text editor and add `always-auth=true` on a new line.

You can use the `yarn config list` command to check that Yarn is using the correct configuration. After running the command, check the values in the `info npm config` section. The contents should look similar to the following snippet.

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

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.
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.
   - Replace *`my_repo`* with your CodeArtifact repository name.

   ```
   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.

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

4. Fetch a CodeArtifact authorization token and store it in an environment variable.

> **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.

- 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.
- Replace *my_repo* with your CodeArtifact repository name.

```
export 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.

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

```
yarn config set
 'npmRegistries["https://my_domain-
111122223333.d.codeartifact.region.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...

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

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

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.

Command	Description
bugs	Tries to guess the location of a package’s bug tracker URL, and then tries to open it.

Command	Description
ci	Installs a project with a clean slate.
deprecate	Deprecates a version of a package.
dist-tag	Modifies package distribution tags.
docs	Tries to guess the location of a package's documentation URL, and then tries to open it using the `--browser` config parameter.
doctor	Runs a set of checks to ensure that your npm installation has what it needs to manage your JavaScript packages.
install	Installs a package.
install-ci-test	Installs a project with a clean slate and runs tests. Alias: `npm cit`. This command runs an `npm ci` followed immediately by an `npm test`.
install-test	Installs package and runs tests. Runs an `npm install` followed immediately by an `npm test`.
outdated	Checks the configured registry to see if any installed packages are currently outdated.
ping	Pings the configured or given npm registry and verifies authentication.
publish	Publishes a package version to the registry.
update	Guesses the location of a package's repositor y URL, and then tries to open it using the `--browser` config parameter.

Command	Description
view	Displays package metadata. Can be used to print metadata properties.

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.

Command	Description
build	Builds a package.
cache	Manipulates the packages cache.
completion	Enables tab completion in all npm commands.
config	Updates the contents of the user and global `npmrc` files.
dedupe	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.
edit	Edits an installed package. Selects a dependency in the current working directory and opens the package folder in the default editor.
explore	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.

Command	Description
help	Gets help on npm.
help-search	Searches npm help documentation.
init	Creates a `package.json` file.
link	Symlinks a package folder.
ls	Lists installed packages.
pack	Creates a tarball from a package.
prefix	Displays prefix. This is the closest parent directory to contain a `package.json` file unless `-g` is also specified.
prune	Removes packages that are not listed on the parent package's dependencies list.
rebuild	Runs the `npm build` command on the matched folders.
restart	Runs a package's stop, restart, and start scripts and associated pre- and post- scripts.
root	Prints the effective `node_modules` folder to standard out.
run-script	Runs arbitrary package scripts.
shrinkwrap	Locks down dependency versions for publicati on.
uninstall	Uninstalls a package.

Unsupported commands

These npm commands are not supported by CodeArtifact repositories.

Command	Description	Notes
access	Sets the access level on published packages.	CodeArtifact uses a permission model that is different from the public npmjs repository.
adduser	Adds a registry user account	CodeArtifact uses a user model that is different from the public npmjs repository.
audit	Runs a security audit.	CodeArtifact does not currently vend security vulnerability data.
hook	Manages npm hooks, including adding, removing, listing, and updating.	CodeArtifact does not currently support any kind of change notification mechanism.
login	Authenticates a user. This is an alias for `npm adduser`.	CodeArtifact uses an authentication model that is different from the public npmjs repository. For information, see Authentication with npm.
logout	Signs out of the registry.	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.

Command	Description	Notes
owner	Manages package owners.	CodeArtifact uses a permissions model that is different from the public npmjs repository.
profile	Changes settings on your registry profile.	CodeArtifact uses a user model that is different from the public npmjs repository.
search	Searches the registry for packages matching the search terms.	CodeArtifact supports limited search functionality with the list-packages command.
star	Marks your favorite packages.	CodeArtifact currently does not support any kind of favorites mechanism.
stars	Views packages marked as favorites.	CodeArtifact currently does not support any kind of favorites mechanism.
team	Manages organization teams and team memberships.	CodeArtifact uses a user and group membership model that is different from the public npmjs repository. For information, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.
token	Manages your authentication tokens.	CodeArtifact uses a different model for getting authentication tokens. For information, see Authentication with npm.

Command	Description	Notes
unpublish	Removes a package from the registry.	CodeArtifact does not support removing a package version from a repository using the npm client. You can use the delete-package-version command.
whoami	Displays the npm user name.	CodeArtifact uses a user model that is different from the public npmjs repository.

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.

Repository	Package name	Package tags
S	`web-helper`	*latest* (alias for version 1.0.1)
D	`web-helper`	*latest* (alias for version 1.0.0)

`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.

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.

Repository	Package name	Package tags
R	`web-helper`	*latest* (alias for version 1.0.0)
U	`web-helper`	*alpha* (alias for version 1.0.1)

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.

Repository	Package name	Package tags
R	`web-helper`	*latest* (alias for version 1.0.0)

Repository	Package name	Package tags
U	`web-helper`	*latest* (alias for version 1.0.1)

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.

Repository	Package name	Package tags
R	`web-helper`	*latest* (alias for version 1.0.1)
U	`web-helper`	*latest* (alias for version 1.0.1)

When version 1.0.2 is published to R, npm updates the *latest* tag to 1.0.2.

Repository	Package name	Package tags
R	`web-helper`	*latest* (alias for version 1.0.2)
U	`web-helper`	*latest* (alias for version 1.0.1)

However, the npm client never sees this tag value because the value of *latest* in U is 1.0.1. Running `npm install` against repository R immediately after publishing 1.0.2 installs 1.0.1 instead of the version that was just published. To install the most recently published version, you must specify the exact package version, as follows.

```
npm install web-helper@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.

Note

We recommend using Yarn 2.x with CodeArtifact. Yarn 1.x does not have HTTP retries, which means it is more susceptible to intermittent service faults which result in 500-level status codes or errors. There is no way to configure a different retry strategy for Yarn 1.x, but this has been added in Yarn 2.x. You can use Yarn 1.x, but you may need to add higher-level retries in build scripts. For example, running your yarn command in a loop so that it will retry if downloading packages fails.

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
- Use CodeArtifact with the nuget or dotnet CLI
- NuGet package name, version, and asset name normalization
- NuGet compatibility

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.

Topics

- Configure Visual Studio with the CodeArtifact Credential Provider
- Use the Visual Studio Package Manager console

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.

   1. The Toolkit for Visual Studio for Visual Studio 2017 and Visual Studio 2019 is distributed in the Visual Studio Marketplace. You can also install and update the toolkit within Visual Studio by using **Tools** ≫ **Extensions and Updates** (Visual Studio 2017) or **Extensions** ≫ **Manage Extensions** (Visual Studio 2019).
   2. After the toolkit has been installed, open it by choosing **AWS Explorer** from the **View** menu.

2. Configure the Toolkit for Visual Studio with your AWS credentials by following the steps in Providing AWS Credentials in the *AWS Toolkit for Visual Studio User Guide*.
3. (Optional) Set the AWS profile you want to use with CodeArtifact. If not set, CodeArtifact will use the default profile. To set the profile, go to **Tools > NuGet Package Manager > Select CodeArtifact AWS Profile**.
4. Add your CodeArtifact repository as a package source in Visual Studio.

   1. Navigate to your repository in the **AWS Explorer** window, right click and select `Copy NuGet Source Endpoint`.
   2. Use the **Tools > Options** command and scroll to **NuGet Package Manager**.
   3. Select the **Package Sources** node.

4. Select **+**, edit the name, and paste the repository URL endpoint copied in Step 3a in the **Source** box, and select **Update**.
5. Select the checkbox for your newly added package source to enable it.

> **Note**
>
> We recommend adding an external connection to **NuGet.org** to your CodeArtifact repository and disabling the **nuget.org** package source in Visual Studio. When using an external connection, all of the packages fetched from **NuGet.org** will be stored in your CodeArtifact repository. If **NuGet.org** becomes unavailable, your application dependencies will still be available for CI builds and local development. For more information about external connections, see Connect a CodeArtifact repository to a public repository.

5. Restart Visual Studio for the changes to take effect.

After configuration, Visual Studio can consume packages from your CodeArtifact repository, any of its upstream repositories, or from NuGet.org if you have added an external connection. For more information about browsing and installing NuGet packages in Visual Studio, see Install and manage packages in Visual Studio using the NuGet Package Manager in the *NuGet documentation*.

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 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

- Publish NuGet packages to CodeArtifact
- CodeArtifact NuGet Credential Provider reference
- CodeArtifact NuGet Credential Provider versions

Configure the nuget or dotnet CLI

You can configure the nuget or dotnet CLI with the CodeArtifact NuGet 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 NuGet Credential Provider

The CodeArtifact NuGet 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 NuGet Credential Provider

dotnet

1. Download the latest version of the AWS.CodeArtifact.NuGet.CredentialProvider tool from NuGet.org with the following `dotnet` command.

   ```
   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.

   ```
   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.

   ```
   dotnet codeartifact-creds configure set profile profile_name
   ```

nuget

Perform the following steps to use the NuGet CLI to install the CodeArtifact NuGet 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.

1. Download the latest version of the CodeArtifact NuGet Credential Provider (codeartifact-nuget-credentialprovider.zip) from an Amazon S3 bucket.

   To view and download earlier versions, see CodeArtifact NuGet Credential Provider versions.
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 and Publish NuGet packages to CodeArtifact.

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.
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.

   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.

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 and Publish NuGet packages to CodeArtifact.

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.

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

   Example output:

   ```
   {
      "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.

   ```
   aws codeartifact get-authorization-token --domain my_domain
   ```

   Example output:

   ```
   {
      "authorizationToken": "eyJ2I...viOw",
      "expiration": 1601616533.0
   }
   ```

3. Create the full repository endpoint URL by appending `/v3/index.json` to the URL returned by `get-repository-endpoint` in step 3.
4. Configure nuget or dotnet to use the repository endpoint from Step 1 and authorization token from Step 2.

> **Note**
>
> The source URL must end in `/v3/index.json` for nuget or dotnet to successfully connect to a CodeArtifact repository.

dotnet

Linux and MacOS users: Because encryption is not supported on non-Windows platforms, you must add the `--store-password-in-clear-text` flag to the following command. Note that this will store your password as plain text in your configuration file.

```
dotnet nuget add source https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/nuget/my_repo/v3/index.json --name packageSourceName --
password eyJ2I...viOw --username aws
```

> **Note**
>
> To update an existing source, use the `dotnet nuget update source` command.

nuget

```
nuget sources add -name domain_name/repo_name -Source
 https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
nuget/my_repo/v3/index.json -password eyJ2I...viOw -username aws
```

Example output:

```
Package source with Name: domain_name/repo_name added successfully.
```

Note

To use a dualstack endpoint, use the `codeartifact.region.on.aws` endpoint.

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*.

Note

When a package is requested, the NuGet client caches which versions of that package exists. Because of this behavior, an install may fail for a package that was previously requested before the desired version became available. To avoid this failure and successfully install a package that exists, you can either clear the NuGet cache ahead of an install with `nuget locals all --clear` or `dotnet nuget locals all --clear`, or avoid using the cache during `install` and `restore` commands by providing the `-NoCache` option for `nuget` or the `--no-cache` option for `dotnet`.

dotnet

```
dotnet add package packageName --source packageSourceName
```

nuget

```
nuget install packageName -Source packageSourceName
```

To install a specific version of a package

dotnet

```
dotnet add package packageName --version 1.0.0 --source packageSourceName
```

nuget

```
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 Connect a CodeArtifact repository to a public repository.

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`.

> **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

```
dotnet nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg --source packageSourceName
```

nuget

```
nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg -Source packageSourceName
```

CodeArtifact NuGet Credential Provider reference

The CodeArtifact NuGet Credential Provider makes it easy to configure and authenticate NuGet with your CodeArtifact repositories.

CodeArtifact NuGet Credential Provider commands

This section includes the list of commands for the CodeArtifact NuGet Credential Provider. These commands must be prefixed with `dotnet codeartifact-creds` like the following example.

```
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 NuGet Credential Provider logs

To enable logging for the CodeArtifact NuGet 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 NuGet Credential Provider log file

```
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.

CodeArtifact NuGet Credential Provider versions

The following table contains version history information and download links for the CodeArtifact NuGet Credential Provider.

Version	Changes	Date published	Download link (S3)
1.0.2 (latest)	Upgraded dependencies	06/26/2024	Download v1.0.2
1.0.1	Added support for net5, net6, and SSO profiles	03/05/2022	Download v1.0.1
1.0.0	Initial CodeArtifact NuGet Credential Provider release	11/20/2020	Download v1.0.0

NuGet package name, version, and asset name normalization

CodeArtifact normalizes package and asset names and package versions before storing them, which means the names or versions in CodeArtifact may be different than the ones provided when the package or asset was published.

Package name normalization: CodeArtifact normalizes NuGet package names by converting all letters to lowercase.

Package version normalization: CodeArtifact normalizes NuGet package versions using the same pattern as NuGet. The following information is from Normalized version numbers from the NuGet documentation.

- Leading zeroes are removed from version numbers:
  - `1.00` is treated as `1.0`
  - `1.01.1` is treated as `1.1.1`
  - `1.00.0.1` is treated as `1.0.0.1`
- A zero in the fourth part of the version number will be omitted:
  - `1.0.0.0` is treated as `1.0.0`
  - `1.0.01.0` is treated as `1.0.1`
- SemVer 2.0.0 build metadata is removed:
  - `1.0.7+r3456` is treated as `1.0.7`

Package asset name normalization: CodeArtifact constructs the NuGet package asset name from the normalized package name and package version.

The non-normalized package name and version name can be used with API and CLI requests because CodeArtifact performs normalization on the package name and version inputs for those requests. For example, inputs of `--package Newtonsoft.JSON` and `--version 12.0.03.0` would be normalized and return a package that has a normalized package name of `newtonsoft.json` and version of `12.0.3`.

You must use the normalized package asset name in API and CLI requests as CodeArtifact does not perform normalization on the `--asset` input.

You must use normalized names and versions in ARNs.

To find the normalized name of a package, use the `aws codeartifact list-packages` command. For more information, see List package names.

To find the non-normalized name of a package, use the `aws codeartifact describe-package-version` command. The non-normalized name of the package is returned in the `displayName` field. For more information, see View and update package version details and dependencies.

NuGet compatibility

This guide contains information about CodeArtifact's compatibility with different NuGet tools and versions.

Topics

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 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 command from the AWS CLI.

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 and use pip with CodeArtifact

pip is the package installer for Python packages. To use pip to install Python packages from your CodeArtifact repository, you must first configure the pip client with your CodeArtifact repository information and credentials.

pip can only be used to install Python packages. To publish Python packages, you can use twine. For more information, see Configure and use twine with CodeArtifact.

Configure pip with the `login` command

First, configure your AWS credentials for use with the AWS CLI, as described in Getting started with CodeArtifact. Then, use the CodeArtifact `login` command to fetch credentials and configure `pip` with them.

> **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.

To configure pip, run the following command.

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

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

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.

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.

   ```
   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. The following command will update the current environment configuration file only. To update the system-wide configuration file, replace `site` with `global`.

   ```
   pip config set site.index-url https://aws:
   $CODEARTIFACT_AUTH_TOKEN@my_domain-
   111122223333.d.codeartifact.region.amazonaws.com/pypi/my_repo/simple/
   ```

   > **Note**
   >
   > To use a dualstack endpoint, use the `codeartifact.region.on.aws` endpoint.

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]
index-url = https://aws:eyJ2ZX...@my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/pypi/my_repo/simple/
```

Run pip

To run `pip` commands, you must configure `pip` with CodeArtifact. For more information, see the following documentation.

1. Follow the steps in the Setting up with AWS CodeArtifact section to configure your AWS account, tools, and permissions.
2. Configure `twine` by following the steps in Configure and use twine with CodeArtifact.

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.

```
pip install requests
```

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

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

Configure and use twine with CodeArtifact

twine is a package publishing utility for Python packages. To use twine to publish Python packages to your CodeArtifact repository, you must first configure twine with your CodeArtifact repository information and credentials.

twine can only be used to publish Python packages. To install Python packages, you can use pip. For more information, see Configure and use pip with CodeArtifact.

Configure twine with the login command

First, configure your AWS credentials for use with the AWS CLI, as described in Getting started with CodeArtifact. Then, use the CodeArtifact `login` command to fetch credentials and configure twine with them.

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.

To configure twine, run the following command.

```
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. The `login` command configures twine for use with CodeArtifact by editing `~/.pypirc` to add the repository specified by the `--repository` option with credentials.

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.

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
```

Note

To use a dualstack endpoint, use the `codeartifact.region.on.aws` endpoint.

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.

```
export TWINE_USERNAME=aws
export TWINE_PASSWORD=`aws codeartifact get-authorization-token --domain my_domain --
domain-owner 111122223333 --query authorizationToken --output text`
export TWINE_REPOSITORY_URL=`aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format pypi --query
 repositoryEndpoint --output text`
```

Run twine

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 section to configure your AWS account, tools, and permissions.
2. Configure twine by following the steps in Configure twine with the `login` command or Configure twine without the `login` command.

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

```
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.

Python package name normalization

CodeArtifact normalizes package names before storing them, which means the package names in CodeArtifact may be different than the name provided when the package was published.

For Python packages, when performing normalization the package name is lowercased and all instances of the characters `.`, `-`, and `_` are replaced with a single `-` character. So the package names `pigeon_cli` and `pigeon.cli` are normalized and stored as `pigeon-cli`. The non-normalized name can be used by pip and twine but the normalized name must be used in CodeArtifact CLI or API requests (such as `list-package-versions`) and in ARNs. For more information about Python package name normalization, see PEP 503 in the Python documentation.

Python compatibility

CodeArtifact does not support PyPI's `XML-RPC` or `JSON` APIs.

CodeArtifact supports PyPI's `Legacy` APIs, except the `simple` API. While CodeArtifact does not support the `/simple/` API endpoint, it does support the `/simple/<project>/` endpoint.

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

- XML-RPC API
- JSON API
- Legacy 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
- Supported client-side commands

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.

Command	Description
install	Install packages.
download	Download packages.

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.

Command	Description
uninstall	Uninstall packages.
freeze	Output installed packages in requirements format.
list	List installed packages.
show	Show information about installed packages.
check	Verify installed packages have compatible dependencies.

Command	Description
config	Manage local and global configuration.
wheel	Build wheels from your requirements.
hash	Compute hashes of package archives.
completion	Helps with command completion.
debug	Show information useful for debugging.
help	Show help for commands.

Requesting Python packages from upstreams and external connections

When importing a Python package version from pypi.org, CodeArtifact will import all the assets in that package version. While most Python packages contain a small number of assets, some contain over 100 assets, typically to support multiple hardware architectures and Python interpreters.

It's common for new assets to be published to pypi.org for an existing package version. For example, some projects publish new assets when new versions of Python are released. When a Python package is installed from CodeArtifact with `pip install`, package versions retained in the CodeArtifact repository are updated to reflect the latest set of assets from pypi.org.

Similarly, if new assets are available for a package version in an upstream CodeArtifact repository that are not present in the current CodeArtifact repository, they will be retained in the current repository when `pip install` is run.

Yanked package versions

Some package versions in pypi.org are marked as *yanked*, which communicates to the package installer (such as pip) that the version should not be installed unless it is the only one that matches a version specifier (using either == or ===). See PEP_592 for more information.

If a package version in CodeArtifact was originally fetched from an external connection to pypi.org, when you install the package version from a CodeArtifact repository, CodeArtifact ensures that the updated yanked metadata of the package version is fetched from pypi.org.

How to know if a package version is yanked

To check if a package version is yanked in CodeArtifact, you can attempt to install it with `pip install packageName===packageVersion`. If the package version is yanked, you will receive a warning message similar to the following:

```
WARNING: The candidate selected for download or install is a yanked version
```

To check if a package version is yanked in pypi.org, you can visit the package version's pypi.org listing at `https://pypi.org/project/packageName/packageVersion/`.

Setting yanked status on private packages

CodeArtifact does not support setting yanked metadata for packages published directly to CodeArtifact repositories.

Why is CodeArtifact not fetching the latest yanked metadata or assets for a package version?

Normally, CodeArtifact ensures that when a Python package version is fetched from a CodeArtifact repository, the yanked metadata is up-to-date with the latest value on pypi.org. Additionally, the list of assets in the package version are also kept updated with the latest set on pypi.org and any upstream CodeArtifact repositories. This is true whether you're installing the package version for the first time and CodeArtifact imports it from pypi.org into your CodeArtifact repository, or if you've installed the package before. However, there are cases when the package manager client, such as pip, won't pull the latest yanked metadata from pypi.org or upstream repositories. Instead, CodeArtifact will return the data that is already stored in your repository. This section describes the three ways this can occur:

Upstream configuration: If the external connection to pypi.org is removed from the repository or its upstreams using disassociate-external-connection, yanked metadata will no longer be refreshed from pypi.org. Similarly, if you remove an upstream repository, assets from the removed repository and the removed repository's upstreams will no longer be available to the current repository. The same is true if you use CodeArtifact package origin controls to prevent new versions of a specific package from being pulled— setting `upstream=BLOCK` will block yanked metadata from being refreshed.

Package version status: If you set the status of a package version to anything except `Published` or `Unlisted`, yanked metadata and assets of the package version will not be refreshed. Similarly, if you are fetching a specific package version (say `torch 2.0.1`) and the same package version is present in an upstream repository with a status that isn't `Published` or `Unlisted`, this will also block yanked metadata and asset propagation from the upstream repository to the current repository. This is because other package version statuses are an indication that the versions are not meant to be consumed anymore in any repository.

Direct publishing: If you publish a specific package version directly into a CodeArtifact repository, this will prevent yanked metadata and asset refresh for the package version from its upstream repositories and pypi.org. For example, say you download an asset from the package version `torch 2.0.1`, such as `torch-2.0.1-cp311-none-macosx_11_0_arm64.whl`, using a web browser and then publish this to your CodeArtifact repository using twine as `torch 2.0.1`. CodeArtifact tracks that the package version entered the domain by direct publishing to your repository, not from an external connection to pypi.org or an upstream repository. In this case, CodeArtifact does not keep the yanked metadata in sync with upstream repositories or pypi.org. The same is true if you publish `torch 2.0.1` into an upstream repository— the presence of the package version will block propagation of yanked metadata and assets to repositories further down the upstream graph.

Using CodeArtifact with Ruby

These topics describe how to use the RubyGems and Bundler tools with CodeArtifact to install and publish Ruby gems.

> **Note**
>
> CodeArtifact recommends Ruby 3.3 or later and does not work with Ruby 2.6 or older.

Topics

Configure and use RubyGems and Bundler with CodeArtifact

After you create a repository in CodeArtifact, you can use RubyGems (`gem`) and Bundler (`bundle`) to install and publish gems. This topic describes how to configure the package managers to authenticate with and use a CodeArtifact repository.

Configure RubyGems (gem) and Bundler (bundle) with CodeArtifact

To use RubyGems (`gem`) or Bundler (`bundle`) to publish gems to or consume gems from AWS CodeArtifact, you'll first need to configure them with your CodeArtifact repository information, including credentials to access it. Follow the steps in one of the following procedure to configure the `gem` and `bundle` CLI tools with your CodeArtifact repository endpoint information and credentials.

Configure RubyGems and Bundler using the console instructions

You can use configuration instructions in the console to connect your Ruby package managers to your CodeArtifact repository. The console instructions provide custom commands that you can run to set up your package managers without needing to find and fill in your CodeArtifact information.

1. Open the AWS CodeArtifact console at https://console.aws.amazon.com/codesuite/codeartifact/home.

2. In the navigation pane, choose **Repositories**, and then choose the repository that you want to use for installing or pushing Ruby gems.
3. Choose **View connection instructions**.
4. Choose your operating system.
5. Choose the Ruby package manager client that you want to configure with your CodeArtifact repository.
6. Follow the generated instructions to configure the package manager client to install Ruby gems from or publish Ruby gems to the repository.

Configure RubyGems and Bundler manually

If you cannot or do not want to use the configuration instructions from the console, you can use the following instructions to connect to your Ruby package managers to your CodeArtifact repository manually.

1. In a command line, use the following command to fetch a CodeArtifact authorization token and store it in an environment variable.

   - 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](#).

   macOS and Linux

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

   Windows

   - 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
  ```

2. To publish Ruby gems to your repository, use the following command to fetch your CodeArtifact repository's endpoint and storing it in the `RUBYGEMS_HOST` environment variable. The `gem` CLI uses this environment variable to determine where gems are published.

   > **Note**
   >
   > Alternatively, instead of using the `RUBYGEMS_HOST` environment variable, you can provide the repository endpoint with the `--host` option when using the `gem push` command.

   - 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.
   - Replace *`my_repo`* with your CodeArtifact repository name.

   macOS and Linux

   ```
   export RUBYGEMS_HOST=`aws codeartifact get-repository-endpoint --
   domain my_domain --domain-owner 111122223333 --repository my_repo --format ruby
    --query repositoryEndpoint --output text | sed 's:/*$::'`
   ```

   Windows

   The following commands retrieve the repository endpoint, trim the trailing `/`, then store them in an environment variable.

   - Windows (using default command shell):

```
for /f %i in ('aws codeartifact get-repository-endpoint --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format ruby --query
 repositoryEndpoint --output text') do set RUBYGEMS_HOST=%i

set RUBYGEMS_HOST=%RUBYGEMS_HOST:~0,-1%
```

- Windows PowerShell:

```
$env:RUBYGEMS_HOST = (aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format
 ruby --query repositoryEndpoint --output text).TrimEnd("/")
```

The following URL is an example repository endpoint:

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

> **Note**
>
> To use a dualstack endpoint, use the `codeartifact.`*`region`*`.on.aws` endpoint.

3. To publish Ruby gems to your repository, you must authenticate to CodeArtifact with RubyGems by editing your `~/.gem/credentials` file to include your auth token. Create a `~/.gem/` directory and a `~/.gem/credentials` file if the directory or file doesn't exist.

   macOS and Linux

   ```
   echo ":codeartifact_api_key: Bearer $CODEARTIFACT_AUTH_TOKEN" >> ~/.gem/
   credentials
   ```

   Windows

   - Windows (using default command shell):

   ```
   echo :codeartifact_api_key: Bearer %CODEARTIFACT_AUTH_TOKEN% >> %USERPROFILE
   %/.gem/credentials
   ```

   - Windows PowerShell:

```
echo ":codeartifact_api_key: Bearer $env:CODEARTIFACT_AUTH_TOKEN" | Add-
Content ~/.gem/credentials
```

4. To use `gem` to install Ruby gems from your repository, you must add the repository endpoint information and auth token to your `.gemrc` file. You can add it to the global file (`~/.gemrc`) or your project `.gemrc` file. The CodeArtifact information you must add to the `.gemrc` is a combination of the repository endpoint and auth token. It is formatted as follows:

   ```
   https://aws:${CODEARTIFACT_AUTH_TOKEN}@my_domain-111122223333.d.codeartifact.us-
   west-2.amazonaws.com/ruby/my_repo/
   ```

   - For the authentication token, you can use the `CODEARTIFACT_AUTH_TOKEN` environment variable that was set in an earlier step.
   - To fetch the repository endpoint, you can read the value of the `RUBYGEMS_HOST` environment variable that was set earlier, or you can use the following `get-repository-endpoint` command, replacing the values as necessary:

     ```
     aws codeartifact get-repository-endpoint --domain my_domain --domain-
     owner 111122223333 --repository my_repo --format ruby --query repositoryEndpoint
      --output text
     ```

   After you have the endpoint, use a text editor to add `aws:${CODEARTIFACT_AUTH_TOKEN}@` in the appropriate position. Once you have the repository endpoint and auth token string created, add it to the `:sources:` section of your `.gemrc` file with the `echo` command as follows:

   > ⚠ **Warning**
   >
   > CodeArtifact does not support adding repositories as sources using the `gem sources -add` command. You must add the source directly to the file.

   macOS and Linux

   ```
   echo ":sources:
   ```

```
    - https://aws:
${CODEARTIFACT_AUTH_TOKEN}@my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/ruby/my_repo/" > ~/.gemrc
```

Windows

- Windows (using default command shell):

```
echo ":sources:
    - https://aws:%CODEARTIFACT_AUTH_TOKEN
%@my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/ruby/my_repo/"
 > "%USERPROFILE%\.gemrc"
```

- Windows PowerShell:

```
echo ":sources:
    - https://aws:
$env:CODEARTIFACT_AUTH_TOKEN@my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/ruby/my_repo/" | Add-Content ~/.gemrc
```

5. To use Bundler, you must configure Bundler with your repository endpoint URL and authentication token by running the following `bundle config` command:

   macOS and Linux

```
bundle config $RUBYGEMS_HOST aws:$CODEARTIFACT_AUTH_TOKEN
```

   Windows

   - Windows (using default command shell):

```
bundle config %RUBYGEMS_HOST% aws:%CODEARTIFACT_AUTH_TOKEN%
```

   - Windows PowerShell:

```
bundle config $Env:RUBYGEMS_HOST aws:$Env:CODEARTIFACT_AUTH_TOKEN
```

Now that you've configured RubyGems (`gem`) and Bundler (`bundle`) with your CodeArtifact repository, you can use them to publish and consume Ruby gems to and from it.

Installing Ruby gems from CodeArtifact

Use the following procedures to install Ruby gems from an CodeArtifact repository with the `gem` or `bundle` CLI tools.

Install Ruby gems with `gem`

You can use the RubyGems (`gem`) CLI to quickly install a specific version of a Ruby gem from your CodeArtifact repository.

To install Ruby gems from a CodeArtifact repository with `gem`

1. If you haven't, follow the steps in Configure RubyGems (`gem`) and Bundler (`bundle`) with CodeArtifact to configure the `gem` CLI to use your CodeArtifact repository with proper credentials.

   > **Note**
   >
   > The authorization token generated is valid for 12 hours. You will need to create a new one if 12 hours have passed since a token was created.

2. Use the following command to install Ruby gems from CodeArtifact:

   ```
   gem install my_ruby_gem --version 1.0.0
   ```

Install Ruby gems with `bundle`

You can use the Bundler (`bundle`) CLI to install the Ruby gems that are configured in your `Gemfile`.

To install Ruby gems from a CodeArtifact repository with `bundle`

1. If you haven't, follow the steps in Configure RubyGems (`gem`) and Bundler (`bundle`) with CodeArtifact to configure the `bundle` CLI to use your CodeArtifact repository with proper credentials.

> **Note**
>
> The authorization token generated is valid for 12 hours. You will need to create a new one if 12 hours have passed since a token was created.

2. Add your CodeArtifact repository endpoint URL to your `Gemfile` as a `source` to install configured Ruby gems from your CodeArtifact repository and its upstreams.

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

gem 'my_ruby_gem'
```

3. Use the following command to install the Ruby gems as specified in your `Gemfile`:

```
bundle install
```

Publishing Ruby gems to CodeArtifact

Use the following procedure to publish Ruby gems to a CodeArtifact repository using the `gem` CLI.

1. If you haven't, follow the steps in Configure RubyGems (gem) and Bundler (bundle) with CodeArtifact to configure the `gem` CLI to use your CodeArtifact repository with proper credentials.

> **Note**
>
> The authorization token generated is valid for 12 hours. You will need to create a new one if 12 hours have passed since a token was created.

2. Use the following command to publish Ruby gems to a CodeArtifact repository. Note that if you did not set the `RUBYGEMS_HOST` environment variable, you must provide your CodeArtifact repository endpoint in the `--host` option.

```
gem push --key codeartifact_api_key my_ruby_gem-0.0.1.gem
```

RubyGems command support

CodeArtifact supports the `gem install` and `gem push` commands. CodeArtifact does not support the following `gem` commands:

- `gem fetch`
- `gem info --remote`
- `gem list --remote`
- `gem mirror`
- `gem outdated`
- `gem owner`
- `gem query`
- `gem search`
- `gem signin`
- `gem signout`
- `gem sources --add`
- `gem sources --update`
- `gem specification --remote`
- `gem update`
- `gem yank`

Bundler compatibility

This guide contains information about CodeArtifact's compatibility with Bundler.

Bundler compatibility

AWS CodeArtifact recommends Bundler 2.4.11 or higher. If you encounter issues with installation, update the Bundler CLI to the latest version.

Bundler version support

In Bundler versions lower than 2.4.11, there is a limit of 500 dependencies that can be defined in the Gemfile before Bundler decides to query the full index, `specs.4.8.gz`. Since CodeArtifact does not support the full index, specifying more than 500 dependencies will not work with CodeArtifact when using Bundler versions lower than 2.4.11.

To define more than 500 dependencies in your Gemfile with CodeArtifact, update Bundler to version 2.4.11 or higher.

Bundler operations support

CodeArtifact's support for RubyGems does not include the Bundler Compact Index APIs (the `/versions` API is not supported). CodeArtifact only supports the Dependencies API.

Additionally, CodeArtifact does not support the various spec APIs, such as `specs.4.8.gz`.

Using CodeArtifact with Swift

These topics describe how to use the Swift Package Manager with CodeArtifact to install and publish Swift packages.

Note

CodeArtifact supports Swift 5.8 and later and Xcode 14.3 and later. CodeArtifact recommends Swift 5.9 and later and Xcode 15 and later.

Topics

- Configure the Swift Package Manager with CodeArtifact
- Consuming and publishing Swift packages
- Swift package name and namespace normalization
- Swift troubleshooting

Configure the Swift Package Manager with CodeArtifact

To use the Swift Package Manager to publish packages to or consume packages from AWS CodeArtifact, you'll first need to set up credentials to access your CodeArtifact repository. The recommended method for configuring the Swift Package Manager CLI with your CodeArtifact credentials and repository endpoint is by using the `aws codeartifact login` command. You can also configure the Swift Package Manager manually.

Configure Swift with the login command

Use the `aws codeartifact login` command to configure the Swift Package Manager with CodeArtifact.

Note

To use the login command, Swift 5.8 or later is required and Swift 5.9 or later is recommended.

The `aws codeartifact login` command will do the following:

1. Fetch an authentication token from CodeArtifact and store it in your environment. How the credentials are stored depends on the operating system of the environment:
   a. **macOS:** An entry is created in the macOS Keychain application.
   b. **Linux and Windows:** An entry is created in the `~/.netrc` file.

   In all operating systems, if a credentials entry exists, this command replaces that entry with a new token.
2. Fetch your CodeArtifact repository endpoint URL and add it to your Swift configuration file. The command adds the repository endpoint URL to the project level configuration file located at `/path/to/project/.swiftpm/configuration/registries.json`.

Note

The `aws codeartifact login` command calls `swift package-registry` commands that must be run from the directory that contains the `Package.swift` file. Because of this, `aws codeartifact login` command must be run from within the Swift project.

To configure Swift with the login command

1. Navigate to the Swift project directory that contains your project's `Package.swift` file.
2. Run the following `aws codeartifact login` command.

   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.

```
aws codeartifact login --tool swift --domain my_domain \
--domain-owner 111122223333 --repository my_repo \
[--namespace my_namespace]
```

The `--namespace` option configures the application to only consume packages from your CodeArtifact repository if they're in the designated namespace. CodeArtifact namespaces are synonymous with scopes, and are used to organize code into logical groups and to prevent name collisions that can occur when your code base includes multiple libraries.

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.

Configure Swift without the login command

While it is recommended that you configure Swift with the `aws codeartifact login` command, you can also configure the Swift Package Manager without the login command by manually updating the Swift Package Manager configuration.

In the following procedure, you will use the AWS CLI to do the following:

1. Fetch an authentication token from CodeArtifact and store it in your environment. How the credentials are stored depends on the operating system of the environment:
   a. **macOS:** An entry is created in the macOS Keychain application.
   b. **Linux and Windows:** An entry is created in the `~/.netrc` file.
2. Fetch your CodeArtifact repository endpoint URL.
3. In the `~/.swiftpm/configuration/registries.json` configuration file, add an entry with your repository endpoint URL and authentication type.

To configure the Swift without the login command

1. In a command line, use the following command to fetch a CodeArtifact authorization token and store it in an environment variable.

   - 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.

   macOS and Linux

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

Windows

- 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
```

2. Get your CodeArtifact repository's endpoint by running the following command. Your repository endpoint is used to point the Swift Package Manager to your repository to consume 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.
   - Replace *my_repo* with your CodeArtifact repository name.

   macOS and Linux

```
export CODEARTIFACT_REPO=`aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format swift
 --query repositoryEndpoint --output text`
```

   Windows

   - Windows (using default command shell):

```
for /f %i in ('aws codeartifact get-repository-endpoint --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format swift --query
 repositoryEndpoint --output text') do set CODEARTIFACT_REPO=%i
```

   - Windows PowerShell:

```
$env:CODEARTIFACT_REPO = aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format
 swift --query repositoryEndpoint --output text
```

The following URL is an example repository endpoint.

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

Note

To use a dualstack endpoint, use the `codeartifact.region.on.aws` endpoint.

Important

You must append `login` onto the end of the repository URL endpoint when used to configure the Swift Package Manager. This is done for you in the commands of this procedure.

3. With these two values stored in environment variables, pass them to Swift using the `swift package-registry login` command as follows:

   macOS and Linux

   ```
   swift package-registry login ${CODEARTIFACT_REPO}login --token
    ${CODEARTIFACT_AUTH_TOKEN}
   ```

   Windows

   - Windows (using default command shell):

     ```
     swift package-registry login %CODEARTIFACT_REPO%login --token
      %CODEARTIFACT_AUTH_TOKEN%
     ```

   - Windows PowerShell:

```
swift package-registry login $Env:CODEARTIFACT_REPO+"login" --token
 $Env:CODEARTIFACT_AUTH_TOKEN
```

4. Next, update the package registry used by your application so that any dependency will be pulled from your CodeArtifact repository. This command must be run in the project directory where you are trying to resolve the package dependency:

   macOS and Linux

   ```
   $ swift package-registry set ${CODEARTIFACT_REPO} [--scope my_scope]
   ```

   Windows

   - Windows (using default command shell):

     ```
     $ swift package-registry set %CODEARTIFACT_REPO% [--scope my_scope]
     ```

   - Windows PowerShell:

     ```
     $ swift package-registry set $Env:CODEARTIFACT_REPO [--scope my_scope]
     ```

   The `--scope` option configures the application to only consume packages from your CodeArtifact repository if they're in the designated scope. Scopes are synonymous with CodeArtifact namespaces, and are used to organize code into logical groups and to prevent name collisions that can occur when your code base includes multiple libraries.

5. You can confirm the configuration has been set up correctly by viewing the contents of the project level `.swiftpm/configuration/registries.json` file by running the following command in your project directory:

```
$ cat .swiftpm/configuration/registries.json
{
  "authentication" : {

  },
  "registries" : {
    "[default]" : {
      "url" : "https://my-domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/swift/my-repo/"
```

```
    }
  },
  "version" : 1
}
```

Now that you've configured the Swift Package Manager with your CodeArtifact repository, you can use it to publish and consume Swift packages to and from it. For more information, see Consuming and publishing Swift packages.

Consuming and publishing Swift packages

Consuming Swift packages from CodeArtifact

Use the following procedure to consume Swift packages from an AWS CodeArtifact repository.

To consume Swift packages from a CodeArtifact repository

1. If you haven't, follow the steps in Configure the Swift Package Manager with CodeArtifact to configure the Swift Package Manager to use your CodeArtifact repository with proper credentials.

   > **Note**
   >
   > The authorization token generated is valid for 12 hours. You will need to create a new one if 12 hours have passed since a token was created.

2. Edit the `Package.swift` file in your application project folder to update the package dependencies to be used by your project.

   a. If the `Package.swift` file does not contain a `dependencies` section, add one.

   b. In the `dependencies` section of the `Package.swift` file, add the package you want to use by adding its package identifier. The package identifier consists of the scope and package name separated by a period. See the code snippet following a later step for an example.

> **Tip**
>
> To find the package identifier, you can use the CodeArtifact console. Find the specific package version you want to use and reference the **Install shortcut** instructions on the package version page.

c. If the `Package.swift` file does not contain a `targets` section, add one.

d. In the `targets` section, add the targets that will need to use the dependency.

   The following snippet is an example snippet showing configured `dependencies` and `targets` sections in a `Package.swift` file:

```
...
    ],
    dependencies: [
        .package(id: "my_scope.package_name", from: "1.0.0")
    ],
    targets: [
      .target(
         name: "MyApp",
         dependencies: ["package_name"]
      ),...
    ],
...
```

3. Now that everything is configured, use the following command to download the package dependencies from CodeArtifact.

```
swift package resolve
```

Consuming Swift packages from CodeArtifact in Xcode

Use the following procedure to consume Swift packages from a CodeArtifact repository in Xcode.

To consume Swift packages from a CodeArtifact repository in Xcode

1. If you haven't, follow the steps in Configure the Swift Package Manager with CodeArtifact to configure the Swift Package Manager to use your CodeArtifact repository with proper credentials.

> **Note**
>
> The authorization token generated is valid for 12 hours. You will need to create a new one if 12 hours have passed since a token was created.

2. Add the packages as a dependency in your project in Xcode.

   a. Choose **File > Add Packages**.

   b. Search for your package using the search bar. Your search must be in the form `package_scope.package_name`.

   c. Once found, choose the package and choose **Add Package**.

   d. Once the package is verified, choose the package products you want to add as a dependency, and choose **Add Package**.

If you run into problems using your CodeArtifact repository with Xcode, see Swift troubleshooting for common issues and possible fixes.

Publishing Swift packages to CodeArtifact

CodeArtifact recommends Swift 5.9 or later and using the `swift package-registry publish` command to publish Swift packages. If you are using an earlier version, you must use a Curl command to publish Swift packages to CodeArtifact.

Publishing CodeArtifact packages with the `swift package-registry publish` command

Use the following procedure with Swift 5.9 or later to publish Swift packages to a CodeArtifact repository with the Swift Package Manager.

1. If you haven't, follow the steps in Configure the Swift Package Manager with CodeArtifact to configure the Swift Package Manager to use your CodeArtifact repository with proper credentials.

> **Note**
>
> The authorization token generated is valid for 12 hours. You will need to create a new one if 12 hours have passed since it was created.

2. Navigate to the Swift project directory that contains the `Package.swift` file for your package.
3. Run the following `swift package-registry publish` command to publish the package. The command creates a package source archive and publishes it to your CodeArtifact repository.

   ```
   swift package-registry publish packageScope.packageName packageVersion
   ```

   For example:

   ```
   swift package-registry publish myScope.myPackage 1.0.0
   ```

4. You can confirm that the package was published and exists in the repository by checking in the console or using the `aws codeartifact list-packages` command as follows:

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

   You can list the single version of the package using the `aws codeartifact list-package-versions` command as follows:

   ```
   aws codeartifact list-package-versions --domain my_domain --repository my_repo \
   --format swift --namespace my_scope --package package_name
   ```

Publishing CodeArtifact packages with Curl

While it is recommended to use the `swift package-registry publish` command that comes with Swift 5.9 or later, you can also use Curl to publish Swift packages to CodeArtifact.

Use the following procedure to publish Swift packages to an AWS CodeArtifact repository with Curl.

1. If you haven't, create and update the CODEARTIFACT_AUTH_TOKEN and CODEARTIFACT_REPO environment variables by following the steps in Configure the Swift Package Manager with CodeArtifact.

   Note

   The authorization token is valid for 12 hours. You will need to refresh your CODEARTIFACT_AUTH_TOKEN environment variable with new credentials if 12 hours have passed since it was created.

2. First, if you do not have a Swift package created, you can do so by running the following commands:

   ```
   mkdir testDir && cd testDir
   swift package init
   git init .
   swift package archive-source
   ```

3. Use the following Curl command to publish your Swift package to CodeArtifact:

   macOS and Linux

   ```
   curl -X PUT  --user "aws:$CODEARTIFACT_AUTH_TOKEN" \
   -H "Accept: application/vnd.swift.registry.v1+json" \
   -F source-archive="@test_dir_package_name.zip" \
   "${CODEARTIFACT_REPO}my_scope/package_name/packageVersion"
   ```

   Windows

   ```
   curl -X PUT  --user "aws:%CODEARTIFACT_AUTH_TOKEN%" \
   -H "Accept: application/vnd.swift.registry.v1+json" \
   -F source-archive="@test_dir_package_name.zip" \
   "%CODEARTIFACT_REPO%my_scope/package_name/packageVersion"
   ```

4. You can confirm that the package was published and exists in the repository by checking in the console or using the `aws codeartifact list-packages` command as follows:

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

You can list the single version of the package using the `aws codeartifact list-package-versions` command as follows:

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

Fetching Swift packages from GitHub and republishing to CodeArtifact

Use the following procedure to fetch a Swift package from GitHub and republish it to a CodeArtifact repository.

To fetch a Swift package from GitHub and republish it to CodeArtifact

1. If you haven't, follow the steps in Configure the Swift Package Manager with CodeArtifact to configure the Swift Package Manager to use your CodeArtifact repository with proper credentials.

   > **Note**
   >
   > The authorization token generated is valid for 12 hours. You will need to create a new one if 12 hours have passed since a token was created.

2. Clone the git repository of the Swift package you want to fetch and republish by using the following `git clone` command. For information about cloning GitHub repositories, see Cloning a repository in the GitHub Docs.

   ```
   git clone repoURL
   ```

3. Navigate to the repository that you just cloned:

   ```
   cd repoName
   ```

4. Create the package and publish it to CodeArtifact.

   a. **Recommended:** If you are using Swift 5.9 or later, you can use the following `swift package-registry publish` command to create the package and publish it to your configured CodeArtifact repository.

```
swift package-registry publish packageScope.packageName versionNumber
```

For example:

```
swift package-registry publish myScope.myPackage 1.0.0
```

b. If you're using a Swift version that is older than 5.9, you must use the `swift archive-source` command to create the package and then use a Curl command to publish it.

   i. If you haven't configured the `CODEARTIFACT_AUTH_TOKEN` and `CODEARTIFACT_REPO` environment variables, or it's been over 12 hours since you have, follow the steps in [Configure Swift without the login command](#).

   ii. Create the Swift package by using the `swift package archive-source` command:

```
swift package archive-source
```

   If successful, you will see `Created package_name.zip` in the command line.

   iii. Use the following Curl command to publish the Swift package to CodeArtifact:

   macOS and Linux

```
curl -X PUT  --user "aws:$CODEARTIFACT_AUTH_TOKEN" \
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@package_name.zip" \
"${CODEARTIFACT_REPO}my_scope/package_name/packageVersion"
```

   Windows

```
curl -X PUT  --user "aws:%CODEARTIFACT_AUTH_TOKEN%" \
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@package_name.zip" \
"%CODEARTIFACT_REPO%my_scope/package_name/packageVersion"
```

5. You can confirm that the package was published and exists in the repository by checking in the console or using the `aws codeartifact list-packages` command as follows:

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

You can list the single version of the package using the `aws codeartifact list-package-versions` command as follows:

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

Swift package name and namespace normalization

CodeArtifact normalizes package names and namespaces before storing them, which means the names in CodeArtifact may be different than the ones provided when the package was published.

Package name and namespace normalization: CodeArtifact normalizes Swift package names and namespaces by converting all letters to lowercase.

Package version normalization: CodeArtifact does not normalize Swift package versions. Note that CodeArtifact only supports Semantic Versioning 2.0 version patterns, for more information about Semantic Versioning, see Semantic Versioning 2.0.0.

The non-normalized package name and namespace can be used with API and CLI requests because CodeArtifact performs normalization on the inputs for those requests. For example, inputs of `--package myPackage` and `--namespace myScope` would be normalized and return a package that has a normalized package name of `mypackage` and namespace of `myscope`.

You must use normalized names in ARNs, such as in IAM policies.

To find the normalized name of a package, use the `aws codeartifact list-packages` command. For more information, see List package names.

Swift troubleshooting

The following information might help you troubleshoot common issues with Swift and CodeArtifact.

I'm getting a 401 error in Xcode even after configuring the Swift Package Manager

Problem: When you are trying to add a package from your CodeArtifact repository as a dependency to your Swift project in Xcode, you are getting a 401 unauthorized error even after you have followed the instructions for connecting Swift to CodeArtifact.

Possible fixes: This can be caused by an issue with the macOS Keychain application, where your CodeArtifact credentials are stored. To fix this, we recommend opening the Keychain application and deleting all of the CodeArtifact entries and configuring the Swift Package Manager with your CodeArtifact repository again by following the instructions in Configure the Swift Package Manager with CodeArtifact.

Xcode hangs on CI machine due to keychain prompt for password

Problem: When you are trying to pull Swift packages from CodeArtifact as part of an Xcode build on a continuous integration (CI) server, such as with GitHub Actions, authentication with CodeArtifact can hang and eventually fail with an error message similar to the following:

```
Failed to save credentials for
\'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com\'
to keychain: status -60008
```

Possible fixes: This is caused by credentials not being saved to the keychain on CI machines, and Xcode only supporting credentials saved in Keychain. To fix this, we recommend creating the keychain entry manually using the following steps:

1. Prepare the keychain.

```
KEYCHAIN_PASSWORD=$(openssl rand -base64 20)
KEYCHAIN_NAME=login.keychain
SYSTEM_KEYCHAIN=/Library/Keychains/System.keychain

if [ -f $HOME/Library/Keychains/"${KEYCHAIN_NAME}"-db ]; then
    echo "Deleting old ${KEYCHAIN_NAME} keychain"
    security delete-keychain "${KEYCHAIN_NAME}"
fi
echo "Create Keychain"
security create-keychain -p "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"
```

```
EXISTING_KEYCHAINS=( $( security list-keychains | sed -e 's/ *//' | tr '\n' ' ' |
 tr -d '"') )
sudo security list-keychains -s "${KEYCHAIN_NAME}" "${EXISTING_KEYCHAINS[@]}"

echo "New keychain search list :"
security list-keychain

echo "Configure keychain : remove lock timeout"
security unlock-keychain -p "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"
security set-keychain-settings "${KEYCHAIN_NAME}"
```

2. Get a CodeArtifact authentication token and your repository endpoint.

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token \
                                    --region us-west-2 \
                                    --domain my_domain   \
                                    --domain-owner 111122223333   \
                                    --query authorizationToken  \
                                    --output text`

export CODEARTIFACT_REPO=`aws codeartifact get-repository-endpoint  \
                              --region us-west-2   \
                              --domain my_domain   \
                              --domain-owner 111122223333   \
                              --format swift     \
                              --repository my_repo    \
                              --query repositoryEndpoint   \
                              --output text`
```

3. Manually create the Keychain entry.

```
SERVER=$(echo $CODEARTIFACT_REPO | sed  's/https:\/\///g' | sed 's/.com.*$/.com/g')
AUTHORIZATION=(-T /usr/bin/security -T /usr/bin/codesign -T /usr/bin/xcodebuild -
T /usr/bin/swift \
               -T /Applications/Xcode-15.2.app/Contents/Developer/usr/bin/
xcodebuild)

security delete-internet-password -a token -s $SERVER -r htps "${KEYCHAIN_NAME}"

security add-internet-password -a token \
                               -s $SERVER \
                               -w $CODEARTIFACT_AUTH_TOKEN \
                               -r htps \
                               -U \
                               "${AUTHORIZATION[@]}" \
                               "${KEYCHAIN_NAME}"

security set-internet-password-partition-list \
             -a token \
             -s $SERVER \
             -S "com.apple.swift-
package,com.apple.security,com.apple.dt.Xcode,apple-tool:,apple:,codesign" \
             -k "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"

security find-internet-password   "${KEYCHAIN_NAME}"
```

For more information about this error and the solution, see https://github.com/apple/swift-package-manager/issues/7236.

Using CodeArtifact with generic packages

These topics show you how to consume and publish for generic packages using AWS CodeArtifact.

Topics

Generic packages overview

Using the `generic` package format, you can upload any type of file to create a package in a CodeArtifact repository. Generic packages aren't associated with any specific programming language, file type, or package management ecosystem. This can be useful for storing and versioning arbitrary build artifacts, such as application installers, machine learning models, configuration files, and others.

A generic package consists of a package name, namespace, version, and one or more assets (or files). Generic packages can exist alongside packages of other formats in a single CodeArtifact repository.

You can use the AWS CLI or SDK to work with generic packages. For a full list of AWS CLI commands that work with generic packages, see Supported commands for generic packages.

Generic package constraints

- They are never fetched from upstream repositories. They can only be obtained from the repository to which they were published.
- They cannot declare dependencies to be returned from ListPackageVersionDependencies or displayed in the AWS Management Console .
- They can store README and LICENSE files, but they're not interpreted by CodeArtifact. Information in these files is not returned from GetPackageVersionReadme or DescribePackageVersion, and doesn't appear in the AWS Management Console.
- Like all packages in CodeArtifact, there are limits to asset size and the number of assets per package. For more information about limits and quotas in CodeArtifact, see Quotas in AWS CodeArtifact.

- The asset names that they contain must follow these rules:
  - Asset names can use Unicode letters and numbers. Specifically, these Unicode character categories are allowed: Lowercase Letter (`Ll`), Modifier Letter (`Lm`), Other Letter (`Lo`), Titlecase Letter (`Lt`), Uppercase Letter (`Lu`), Letter Number (`Nl`), and Decimal Number (`Nd`).
  - The following special characters are allowed: `~!@^&()-_+[]{};,.`
  - Assets cannot be named `.` or `..`
  - Spaces are the only allowed whitespace character. Asset names cannot start or end with a space character, or include consecutive spaces.

Supported commands for generic packages

You can use the AWS CLI or SDK to work with generic packages. The following CodeArtifact commands work with generic packages:

- copy-package-versions (see Copy packages between repositories)
- delete-package (see Deleting a package (AWS CLI))
- delete-package-versions (see Deleting a package version (AWS CLI))
- describe-package
- describe-package-version (see View and update package version details and dependencies)
- dispose-package-versions (see Disposing of package versions)
- get-package-version-asset (see Download package version assets)
- list-package-version-assets (see List package version assets)
- list-package-versions (see List package versions)
- list-packages (see List package names)
- publish-package-version (see Publishing a generic package)
- put-package-origin-configuration (see Editing package origin controls)

> **Note**
>
> You can use the `publish` origin control setting to allow or block publishing of a generic package name in a repository. However, the `upstream` setting does not apply to generic packages because they cannot be fetched from an upstream repository.

- update-package-versions-status (see Updating package version status)

Publishing and consuming generic packages

To publish a generic package version and its related assets, use the `publish-package-version` command. You can list a generic package's assets using the `list-package-version-asset` command and download them using `get-package-version-asset`. The following topic contains step-by-step instructions to publish generic packages or download generic package assets using these commands.

Publishing a generic package

A generic package consists of a package name, namespace, version, and one or more assets (or files). This topic demonstrates how to publish a package named `my-package`, with the namespace `my-ns`, version `1.0.0`, and containing one asset named `asset.tar.gz`.

Prerequisites:

- Set up and configure the AWS Command Line Interface with CodeArtifact (see Setting up with AWS CodeArtifact)
- Have a CodeArtifact domain and repository (see Getting started using the AWS CLI)

To publish a generic package

1. Use the following command to generate the SHA256 hash for each file you want to upload to a package version, and place the value in an environment variable. This value is used as an integrity check to verify that the file contents have not changed after they were originally sent.

   Linux

   ```
   export ASSET_SHA256=$(sha256sum asset.tar.gz | awk '{print $1;}')
   ```

   macOS

   ```
   export ASSET_SHA256=$(shasum -a 256 asset.tar.gz | awk '{print $1;}')
   ```

   Windows

   ```
   for /f "tokens=*" %G IN ('certUtil -hashfile asset.tar.gz SHA256 ^| findstr /v
    "hash"') DO SET "ASSET_SHA256=%G"
   ```

2. Call `publish-package-version` to upload the asset and create a new package version.

> **Note**
>
> If your package contains more than one asset, you can call `publish-package-version` once for each asset to upload. Include the `--unfinished` argument for each call to `publish-package-version`, except for when uploading the final asset. Omitting `--unfinished` will set the package version's status to `Published`, and prevent additional assets from being uploaded to it.
> Alternatively, include `--unfinished` for every call to `publish-package-version`, then set the package version's status to `Published` using the `update-package-versions-status` command.

Linux/macOS

```
aws codeartifact publish-package-version --domain my_domain --repository my_repo
 \
      --format generic --namespace my-ns --package my-package --package-
version 1.0.0 \
      --asset-content asset.tar.gz --asset-name asset.tar.gz \
      --asset-sha256 $ASSET_SHA256
```

Windows

```
aws codeartifact publish-package-version --domain my_domain --repository my_repo
 ^
      --format generic --namespace my-ns --package my-package --package-
version 1.0.0 ^
      --asset-content asset.tar.gz --asset-name asset.tar.gz ^
      --asset-sha256 %ASSET_SHA256%
```

The following shows the output.

```
{
    "format": "generic",
    "namespace": "my-ns",
    "package": "my-package",
    "version": "1.0.0",
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
    "status": "Published",
    "asset": {
        "name": "asset.tar.gz",
        "size": 11,
        "hashes": {
            "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
            "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
            "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
            "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95
SHA-512"
        }
    }
}
```

Listing generic package assets

To list the assets contained in a generic package, use the `list-package-version-assets` command. For more information, see List package version assets.

The following example lists the assets of version `1.0.0` of package `my-package`.

To list package version assets

- Call `list-package-version-assets` to list the assets contained in a generic package.

  Linux/macOS

  ```
  aws codeartifact list-package-version-assets --domain my_domain \
    --repository my_repo --format generic --namespace my-ns \
    --package my-package --package-version 1.0.0
  ```

  Windows

  ```
  aws codeartifact list-package-version-assets --domain my_domain ^
    --repository my_repo --format generic --namespace my-ns ^
    --package my-package --package-version 1.0.0
  ```

The following shows the output.

```
{
    "assets": [
        {
            "name": "asset.tar.gz",
            "size": 11,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256":
 "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95
SHA-512"
            }
        }
    ],
    "package": "my-package",
    "format": "generic",
    "namespace": "my-ns",
    "version": "1.0.0",
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"
}
```

Downloading generic package assets

To download the assets from a generic package, use the `get-package-version-asset` command. For more information, see Download package version assets.

The following example downloads the asset `asset.tar.gz` from version `1.0.0` of the package `my-package` to the current working directory into a file also named `asset.tar.gz`.

To download package version assets

- Call `get-package-version-asset` to download assets from a generic package.

  Linux/macOS

  ```
  aws codeartifact get-package-version-asset --domain my_domain \
  ```

```
  --repository my_repo --format generic --namespace my-ns --package my-package \
  --package-version 1.0.0 --asset asset.tar.gz \
  asset.tar.gz
```

Windows

```
aws codeartifact get-package-version-asset --domain my_domain ^
  --repository my_repo --format generic --namespace my-ns --package my-package ^
  --package-version 1.0.0 --asset asset.tar.gz ^
  asset.tar.gz
```

The following shows the output.

```
{
    "assetName": "asset.tar.gz",
    "packageVersion": "1.0.0",
    "packageVersionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"
}
```

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
- Using Python packages in CodeBuild
- Using Maven packages in CodeBuild
- Using NuGet packages in CodeBuild
- Dependency caching

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.

For information, see Modifying a Role in the *IAM User Guide*.

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.

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.

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333
 --repository my_repo
build:
  commands:
    - npm install
```

Windows

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool npm --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - npm install
```

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.

   ```
   {
   ```

```
  "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.

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`:

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool pip --domain my_domain --domain-owner 111122223333
 --repository my_repo
build:
  commands:
    - pip install requests
```

To publish Python packages using `twine`:

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool twine --domain my_domain --domain-
owner 111122223333 --repository my_repo
build:
  commands:
    - twine upload --repository codeartifact mypackage
```

Windows

To install Python packages using `pip`:

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
```

```
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool pip --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - pip install requests
```

To publish Python packages using `twine`:

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool twine --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - twine upload --repository codeartifact mypackage
```

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.

   ```
   {
     "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. 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 111122223333 --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, you can invoke your Gradle build from the `buildspec.yaml` `build` section.

```
build:
  commands:
    - gradle build
```

To use mvn:

You must configure your Maven configuration files (`settings.xml` and `pom.xml`) following the instructions in Using CodeArtifact with mvn.

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

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.

```
{
  "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.

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`:

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      -  dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - dotnet add package <packageName> --source codeartifact
```

Windows

To consume NuGet packages using `dotnet`:

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - dotnet add package <packageName> --source codeartifact
```

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.

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`:

```
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`:

```
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`:

```
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`:

```
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.

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`:

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - dotnet pack -o .
      - dotnet nuget push *.nupkg -s codeartifact
```

Windows

To publish NuGet packages using `dotnet`:

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
```

```
  build:
    commands:
      - dotnet pack -o .
      - dotnet nuget push *.nupkg -s codeartifact
```

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.

```
cache:
  paths:
    - '/root/.m2/**/*'
```

For other tools, use the cache folders shown in this table.

Tool	Cache directory
`mvn`	`/root/.m2/**/*`
`gradle`	`/root/.gradle/caches/**/*`
`pip`	`/root/.cache/pip/**/*`
`npm`	`/root/.npm/**/*`
`nuget`	`/root/.nuget/**/*`
`yarn (classic)`	`/root/.cache/yarn/**/*`

Monitoring CodeArtifact

Monitoring is an important part of maintaining the reliability, availability, and performance of CodeArtifact and your other AWS solutions. AWS provides the following monitoring tools to watch CodeArtifact, report when something is wrong, and take automatic actions when appropriate:

- You can use Amazon EventBridge to automate your AWS services and respond automatically to system events, such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you and which automated actions to take when an event matches a rule. For more information, see Amazon EventBridge User Guide and CodeArtifact event format and example.
- You can use Amazon CloudWatch metrics to view CodeArtifact usage by operation. CloudWatch metrics includes all requests made to CodeArtifact, and requests are shown by account. You can view these metrics in CloudWatch metrics by navigating to the **Usage/By AWS Resource** AWS namespace. For more information, see Use Amazon CloudWatch metrics in the *Amazon CloudWatch User Guide*.

Topics

Monitoring 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.
- Deleting package versions using `delete-package-versions`. For more information, see Delete a package or package version.
- Deleting a package versions using `delete-package`. One event will be published for each version of the deleted package. For more information, see Delete a package or package version.
- 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.
- Ingesting a package version from an external repository into a CodeArtifact repository. For more information, see Connect a CodeArtifact repository to a public repository.

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*.

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.

Event field	Description
version	The version of the event format. There is currently only a single version, `0`.
id	A unique identifier for the event.
detail-type	The type of event. This determines the fields in the `detail` object. The one `detail-type` currently supported is `CodeArtifact Package Version State Change`.
source	The source of the event. For CodeArtifact, it will be `aws.codeartifact` .
account	The AWS account ID of the account that receives the event.
time	The exact time the event was triggered.
region	The region where the event was triggered.
resources	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.
domainName	The domain that contains the repository that contains the package.
domainOwner	The AWS account ID of the owner of the domain.
repositoryName	The repository that contains the package.

Event field	Description
repositoryAdministrator	The AWS account ID of the administrator of the repository.
packageFormat	The format of the package that triggered the event.
packageNamespace	The namespace of the package that triggered the event.
packageName	The name of the package that triggered the event.
packageVersion	The version of the package that triggered the event.
packageVersionState	The state of the package version when the event was triggered. Possible values are `Unfinished` , `Published` , `Unlisted`, `Archived`, and `Disposed`.
packageVersionRevision	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 another JAR file to a Maven package), the `packageVersionRevision` changes.
changes.assetsAdded	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.
changes.assetsRemoved	The number of assets removed from a package that triggered an event.
changes.assetsUpdated	The number of assets modified in the package that triggered the event.

Event field	Description
changes.metadataUpdated	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.
changes.statusChanged	A boolean value that is set to `true` if the event's `packageVersionStatus` is modified(for example, if `packageVersionStatus` changes from `Unfinished` to `Published`).
operationType	Describes the high-level type of the package version change. The possible values are `Created`, `Updated`, and `Deleted`.
sequenceNumber	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. **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.

```
{
      "version":"0",
      "id":"73f03fec-a137-971e-6ac6-07c8ffffffff",
      "detail-type":"CodeArtifact Package Version State Change",
      "source":"aws.codeartifact",
      "account":"123456789012",
      "time":"2019-11-21T23:19:54Z",
      "region":"us-west-2",
      "resources":["arn:aws:codeartifact:us-west-2:111122223333:package/my_domain/
myrepo/npm//mypackage"],
      "detail":{
        "domainName":"my_domain",
        "domainOwner":"111122223333",
        "repositoryName":"myrepo",
        "repositoryAdministrator":"123456789012",
        "packageFormat":"npm",
        "packageNamespace":null,
        "packageName":"mypackage",
        "packageVersion":"1.0.0",
        "packageVersionState":"Published",
        "packageVersionRevision":"0E5DE26A4CD79FDF3EBC4924FFFFFFFF",
        "changes":{
          "assetsAdded":1,
          "assetsRemoved":0,
          "metadataUpdated":true,
          "assetsUpdated":0,
          "statusChanged":true
        },
        "operationType":"Created",
        "sequenceNumber":1,
        "eventDeduplicationId":"2mEO0A2Ke07rWUTBXk3CAiQhdTXF4N94LNaT/ffffff="
      }
    }
```

Use an event to start a CodePipeline execution

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
- Create the EventBridge rule
- Create the EventBridge rule target

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.

```
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
- Create the EventBridge rule target
- Configure EventBridge permissions

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.

```
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.

```
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.

```
aws lambda add-permission --function-name MyLambdaFunction \\
  --statement-id my-statement-id --action 'lambda:InvokeFunction' \\
  --principal events.amazonaws.com \\
  --source-arn arn:aws:events:us-west-2:111122223333:rule/MyCodeArtifactRepoRule
```

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

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. You are also responsible for 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 users with AWS IAM Identity Center or 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 require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- 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 sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 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) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text 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 text 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.
- **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. For more information about AWS PrivateLink and VPC endpoints, see AWS PrivateLink and Accessing AWS Services Through PrivateLink.

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

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 package manager client commands. Package manager 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

A log entry created by GetAuthorizationToken includes the domain name in the requestParameters field.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
      "type": "AssumedRole",
      "principalId": "AIDACKCEVSQ6C2EXAMPLE",
      "arn": "arn:aws:sts::123456789012:assumed-role/Console/example",
      "accountId": "123456789012",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "sessionContext": {
          "attributes": {
              "mfaAuthenticated": "false",
              "creationDate": "2018-12-11T13:31:37Z"
          },
          "sessionIssuer": {
              "type": "Role",
              "principalId": "AIDACKCEVSQ6C2EXAMPLE",
              "arn": "arn:aws:iam::123456789012:role/Console",
              "accountId": "123456789012",
              "userName": "Console"
          }
      }
  },
  "eventTime": "2018-12-11T13:31:37Z",
  "eventSource": "codeartifact.amazonaws.com",
  "eventName": "GetAuthorizationToken",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "205.251.233.50",
  "userAgent": "aws-cli/1.16.37 Python/2.7.10 Darwin/16.7.0 botocore/1.12.27",
  "requestParameters": {
      "domainName": "example-domain"
      "domainOwner": "123456789012"
  },
  "responseElements": {
      "sessionToken": "HIDDEN_DUE_TO_SECURITY_REASONS"
  },
  "requestID": "6b342fc0-5bc8-402b-a7f1-ffffffffffff",
  "eventID": "100fde01-32b8-4c2b-8379-ffffffffffff",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Example: A log entry for fetching an npm package version

Requests made by all package manager clients, including 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.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumed-role/Console/example",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIJIOBJIBSREXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-12-17T02:05:16Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Console",
                "accountId": "123456789012",
                "userName": "Console"
            }
        }
    },
    "eventTime": "2018-12-17T02:05:46Z",
    "eventSource": "codeartifact.amazonaws.com",
    "eventName": "ReadFromRepository",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.50",
    "userAgent": "npm/6.14.15 node/v12.22.9 linux x64 ci/custom",
    "requestParameters": {
        "domainName": "example-domain",
        "domainOwner": "123456789012",
        "repositoryName": "example-repo",
        "packageName": "lodash",
        "packageFormat": "npm",
        "packageVersion": "4.17.20"
    },
    "responseElements": null,
    "additionalEventData": {
        "httpMethod": "GET",
        "requestUri": "/npm/lodash/-/lodash-4.17.20.tgz"
    },
    "requestID": "9f74b4f5-3607-4bb4-9229-ffffffffffff",
    "eventID": "c74e40dd-8847-4058-a14d-ffffffffffff",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Compliance validation for AWS CodeArtifact

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. 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.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config 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. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

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 the permissions needed to create an authorization token, see the `GetAuthorizationToken` entry in the AWS CodeArtifact permissions reference. For more general information on CodeArtifact permissions, see How AWS CodeArtifact works with IAM.

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.

> **Note**
>
> Root users cannot call `GetAuthorizationToken`.

- `aws codeartifact login`: 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. The support package managers are as follows:

  - dotnet
  - npm
  - nuget
  - pip
  - swift
  - twine
- `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 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.

Parameter	Required	Description
`--tool`	Yes	The package manager to authenticate to. Possible values are `dotnet`, `npm`, `nuget`, `pip`, `swift` and `twine`.
`--domain`	Yes	The domain name that the repository belongs to.
`--domain-owner`	No	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]().
`--repository`	Yes	The name of the repository to authenticate to.
`--duration-seconds`	No	The time, in seconds, that the login information is valid. The minimum value is 900* and maximum value is 43200.
`--namespace`	No	Associates a namespace with your repository tool.
`--dry-run`	No	Only print the commands that would be executed to connect your tool with your repository without making any changes to your configuration.

*A value of 0 is also valid when calling `login` while assuming a role. Calling `login` with `--duration-seconds 0` creates a token with a lifetime equal to the remaining time in the session duration of an assumed role.

The following example shows how to fetch an authorization token with the `login` command.

```
aws codeartifact login \
    --tool dotnet | npm | nuget | pip | swift | twine \
    --domain my_domain \
    --domain-owner 111122223333 \
    --repository my_repo
```

For specific guidance on how to use the `login` command with npm, see Configure and use npm with CodeArtifact. For Python, see Using CodeArtifact with Python.

Permissions required to call the `GetAuthorizationToken` API

Both the `sts:GetServiceBearerToken` and the `codeartifact:GetAuthorizationToken` permissions are required to call the CodeArtifact `GetAuthorizationToken` API.

To use a package manager with a CodeArtifact repository, your IAM user or role must allow `sts:GetServiceBearerToken`. While `sts:GetServiceBearerToken` can be added to a CodeArtifact domain resource policy, the permission will have no effect in that policy.

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.
- For Python users, see Configure pip without the login command or Configure and use twine with CodeArtifact.
- For Maven users, see Use CodeArtifact with Gradle or Use CodeArtifact with mvn.
- For npm users, see Configuring npm without using the login command.

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.

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.
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.

- 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.

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 AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well‐Architected Framework*.

You use AWS published API calls to access CodeArtifact through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, 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.

Dependency substitution attacks

Package managers simplify the process of packaging and sharing reusable code. These packages may be private packages developed by an organization for use in their applications, or they may be public, typically open-source packages that are developed outside an organization and distributed by public package repositories. When requesting packages, developers rely on their package manager to fetch new versions of their dependencies. Dependency substitution attacks, also known as dependency confusion attacks, exploit the fact that a package manager typically has no way to distinguish legitimate versions of a package from malicious versions.

Dependency substitution attacks belong to a subset of hacks known as software supply chain attacks. A software supply chain attack is an attack that takes advantage of vulnerabilities anywhere in the software supply chain.

A dependency substitution attack can target anyone who uses both internally developed packages and packages fetched from public repositories. The attackers identify internal package names and then strategically place malicious code with the same name in public package repositories. Typically, the malicious code is published in a package with a high version number. Package managers fetch the malicious code from these public feeds because they believe that the malicious packages are the latest versions of the package. This causes a "confusion" or "substitution" between the desired package and the malicious package, leading to the code being compromised.

To prevent dependency substitution attacks, AWS CodeArtifact provides package origin controls. Package origin controls are settings that control how packages can be added to your repositories. The controls can be used to ensure package versions cannot be both published directly to your repository and ingested from public sources, protecting you from dependency substitution attacks. Origin controls can be set on individual packages and multiple packages by setting origin controls on package groups. For more information about package origin controls and how to change them, see Editing package origin controls and Package group origin controls.

Identity and Access Management for AWS CodeArtifact

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use CodeArtifact resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Using tags to control access to CodeArtifact resources
- AWS CodeArtifact permissions reference
- Troubleshooting AWS CodeArtifact identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in CodeArtifact.

Service user – If you use the CodeArtifact service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more CodeArtifact features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in CodeArtifact, see Troubleshooting AWS CodeArtifact identity and access.

Service administrator – If you're in charge of CodeArtifact resources at your company, you probably have full access to CodeArtifact. It's your job to determine which CodeArtifact features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with CodeArtifact, see How AWS CodeArtifact works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to CodeArtifact. To view example CodeArtifact identity-based policies that you can use in IAM, see Identity-based policy examples for AWS CodeArtifact.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries –** A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs) –** SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs) –** RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached

to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS CodeArtifact works with IAM

Before you use IAM to manage access to CodeArtifact, learn what IAM features are available to use with CodeArtifact.

IAM features you can use with AWS CodeArtifact

IAM feature	CodeArtifact support
Identity-based policies	Yes
Resource-based policies	Yes
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	No
ACLs	No
ABAC (tags in policies)	Partial

IAM feature	CodeArtifact support
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	No

To get a high-level view of how CodeArtifact and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for CodeArtifact

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for CodeArtifact

To view examples of CodeArtifact identity-based policies, see Identity-based policy examples for AWS CodeArtifact.

Resource-based policies within CodeArtifact

Supports resource-based policies: Yes

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for CodeArtifact

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of CodeArtifact actions, see Actions defined by AWS CodeArtifact in the *Service Authorization Reference*.

Policy actions in CodeArtifact use the following prefix before the action:

```
codeartifact
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "codeartifact:action1",
      "codeartifact:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "codeartifact:Describe*"
```

To view examples of CodeArtifact identity-based policies, see Identity-based policy examples for AWS CodeArtifact.

Policy resources for CodeArtifact

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of CodeArtifact resource types and their ARNs, see Resources defined by AWS CodeArtifact in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS CodeArtifact. To see examples of specifying CodeArtifact resource ARNs in policies, see AWS CodeArtifact resources and operations.

Policy condition keys for CodeArtifact

Supports service-specific policy condition keys: No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

> **Note**
>
> AWS CodeArtifact does not support the following AWS Global Condition Context Keys:
>
> - Referer
> - UserAgent

To see a list of CodeArtifact condition keys, see Condition keys for AWS CodeArtifact in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS CodeArtifact.

To view examples of CodeArtifact identity-based policies, see Identity-based policy examples for AWS CodeArtifact.

ACLs in CodeArtifact

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with CodeArtifact

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging CodeArtifact resources, including example identity-based policies for limiting access to a resource based on the tags on that resource, see Using tags to control access to CodeArtifact resources.

Using temporary credentials with CodeArtifact

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for CodeArtifact

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

There are two CodeArtifact API actions that require the calling principal to have permissions in other services:

1. `GetAuthorizationToken` requires `sts:GetServiceBearerToken` along with `codeartifact:GetAuthorizationToken`.
2. `CreateDomain`, when providing a non-default encryption key, requires both `kms:DescribeKey` and `kms:CreateGrant` on the KMS key along with `codeartifact:CreateDomain`.

For more information about required permissions and resources for actions in CodeArtifact, see AWS CodeArtifact permissions reference.

Service roles for CodeArtifact

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

⚠ **Warning**

Changing the permissions for a service role might break CodeArtifact functionality. Edit service roles only when CodeArtifact provides guidance to do so.

Service-linked roles for CodeArtifact

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS CodeArtifact

By default, users and roles don't have permission to create or modify CodeArtifact resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by CodeArtifact, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS CodeArtifact in the *Service Authorization Reference*.

Topics

- Allow a user to get information about repositories and domains
- Allow a user to get information about specific domains
- Allow a user to get information about specific repositories
- Limit authorization token duration

Policy best practices

Identity-based policies determine whether someone can create, access, or delete CodeArtifact resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API

operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the CodeArtifact console

To access the AWS CodeArtifact console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the CodeArtifact resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the CodeArtifact console, also attach the `AWSCodeArtifactAdminAccess` or `AWSCodeArtifactReadOnlyAccess` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

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.

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

- `AWSCodeArtifactReadOnlyAccess` – Provides read-only access to CodeArtifact.

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

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.

Allow a user to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

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",
         "Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/my*"
      }
   ]
}
```

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.

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": "arn:aws:codeartifact:*:*:repository/*/*test"
      },
      {
         "Effect": "Allow",
         "Action": [
            "codeartifact:List*",
            "codeartifact:Describe*"
         ],
         "Resource": "arn:aws:codeartifact:*:*:package/*/*test/*/*/*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sts:AWSServiceName": "codeartifact.amazonaws.com"
            }
         }
      },
      {
         "Effect": "Allow",
         "Action": "codeartifact:GetAuthorizationToken",
         "Resource": "*"
      }
   ]
}
```

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.

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.

```
{
    "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`.

```
{
    "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. 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*.

Important

When using tags on resources to limit actions, the tags must be on the resource in which the action operates on. For example, to deny `DescribeRepository` permissions with tags, the tags must be on each repository and not the domain. See AWS CodeArtifact permissions reference for a list of actions in CodeArtifact and which resources they operate on.

Tag-based access control examples

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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "codeartifact:CreateRepository",
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:RequestTag/costcenter": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "codeartifact:CreateRepository",
      "Resource": "*",
      "Condition": {
          "ForAnyValue:StringNotEquals": {
              "aws:RequestTag/costcenter": [
                  "1",
                  "2"
              ]
          }
      }
  }
  ]
}
```

Example 2: 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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codeartifact:TagResource",
        "codeartifact:UntagResource",
        "codeartifact:DescribeDomain",
        "codeartifact:DescribeRepository",
        "codeartifact:PutDomainPermissionsPolicy",
        "codeartifact:PutRepositoryPermissionsPolicy",
        "codeartifact:ListRepositoriesInDomain",
        "codeartifact:UpdateRepository",
        "codeartifact:ReadFromRepository",
        "codeartifact:ListPackages",
        "codeartifact:ListTagsForResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Key1": ["Value1", "Value2"]
        }
      }
    }
  ]
}
```

Example 3: 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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeartifact:UpdateRepository",
        "codeartifact:DeleteRepository",
        "codeartifact:ListPackages"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Key1": "Value1"
        }
      }
    }
  ]
}
```

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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeartifact:CreateRepository",
        "codeartifact:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Key1": "Value1"
        }
      }
    }
  ]
}
```

AWS CodeArtifact permissions reference

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*.

Resource type	ARN format
Domain	`arn:aws:codeartifact: region-ID :account-ID :domain/my_domain`
Repository	`arn:aws:codeartifact: region-ID :account-ID :repository/ my_domain /my_repo`
Package group	`arn:aws:codeartifact: region-ID :account-ID :package-group/ my_domain /encoded_package_group_pattern`
Package with a namespace	`arn:aws:codeartifact: region-ID :account-ID :package/ my_domain /my_repo/package-format /namespace /my_package`
Package without a namespace	`arn:aws:codeartifact: region-ID :account-ID :package/ my_domain /my_repo/package-format //my_package`
All CodeArtifact resources	`arn:aws:codeartifact:*`
All CodeArtifact resources owned by the specified account in the specified AWS Region	`arn:aws:codeartifact: region-ID :account-ID :*`

Which resource ARN you specify depends on which action or actions you want to control access to.

You can indicate a specific domain (*myDomain*) in your statement using its ARN as follows.

```
"Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain"
```

You can indicate a specific repository (*myRepo*) in your statement using its ARN as follows.

```
"Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain/myRepo"
```

To specify multiple resources in a single statement, separate their ARNs with commas. The following statement applies to all packages and repositories in a specific domain.

```
"Resource": [
  "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain",
  "arn:aws:codeartifact:us-east-2:123456789012:repository/myDomain/*",
  "arn:aws:codeartifact:us-east-2:123456789012:package/myDomain/*"
]
```

> **Note**
>
> Many 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.

AWS CodeArtifact API operations and permission

You can use the following table as a reference when you are setting up access control 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`.

Package group ARNs

> **Note**
>
> This section about how package group ARNs and pattern encoding is informational. It is recommended to copy ARNs from the console, or fetch ARNs using the `DescribePackageGroup` API instead of encoding patterns and constructing ARNs.

IAM policies use the wildcard character, `*`, to match multiple IAM actions or multiple resources. Package group patterns also use the `*` character. In order to more easily write IAM policies that match a single package group, the package group ARN format uses an encoded version of the package group pattern.

Specifically, the package group ARN format is as follows:

```
arn:aws:codeartifact:region:account-ID:package-
group/my_domain/encoded_package_group_pattern
```

Where the encoded package group pattern is the package group pattern, with certain special characters replaced with their percent-encoded values. The following list contains the characters and their corresponding percent-encoded values:

- `*` : `%2a`
- `$` : `%24`
- `%` : `%25`

For example, the ARN for a root package group of a domain, (`/*`), would be:

```
arn:aws:codeartifact:us-east-1:111122223333:package-group/my_domain/%2a
```

Note that characters not included in the list can not be encoded, and ARNs are case-sensitive, so `*` must be encoded as `%2a` and not `%2A`.

Troubleshooting AWS CodeArtifact identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with CodeArtifact and IAM.

Topics

I am not authorized to perform an action in CodeArtifact

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `codeartifact:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 codeartifact:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `codeartifact:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my CodeArtifact resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether CodeArtifact supports these features, see How AWS CodeArtifact works with IAM.

- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

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
- Create the Amazon S3 gateway endpoint
- Use CodeArtifact from a VPC
- Create a VPC endpoint policy for CodeArtifact

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.

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region.codeartifact.api --subnet-ids subnetid \
  --security-group-ids groupid --private-dns-enabled
```

The following command creates an endpoint to access package managers and build tools.

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region.codeartifact.repositories --subnet-ids subnetid \
  --security-group-ids groupid --private-dns-enabled
```

Note

When you create a `codeartifact.repositories` endpoint, you must create a private DNS hostname using the `--private-dns-enabled` option. If you can't or do not want to create a private DNS hostname when you create the `codeartifact.repositories` endpoint, you must follow an extra configuration step to use your package manager with CodeArtifact from a VPC. See Use the `codeartifact.repositories` endpoint without private DNS for more information.

After creating VPC endpoints, you may need to do more configuration with security group rules to use the endpoints with CodeArtifact. For more information about security groups in Amazon VPC, see Security groups.

If you are having issues connecting to CodeArtifact, you can use the VPC Reachability Analyzer tool to debug the issue. For more information, see What is VPC Reachability Analyzer?

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).

> **Note**
>
> An Amazon S3 endpoint is not needed when using Python or Swift package formats.

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.

```
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.

Region	Amazon S3 Bucket ARN
us-east-1	arn:aws:s3:::assets-193858265520-us-east-1
us-east-2	arn:aws:s3:::assets-250872398865-us-east-2
us-west-2	arn:aws:s3:::assets-787052242323-us-west-2

Region	Amazon S3 Bucket ARN
eu-west-1	arn:aws:s3:::assets-438097961670-eu-west-1
eu-west-2	arn:aws:s3:::assets-247805302724-eu-west-2
eu-west-3	arn:aws:s3:::assets-762466490029-eu-west-3
eu-north-1	arn:aws:s3:::assets-611884512288-eu-north-1
eu-south-1	arn:aws:s3:::assets-484130244270-eu-south-1
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.

```
aws codeartifact describe-domain --domain mydomain
```

```

{
  "domain": {
    "name": "mydomain",
    "owner": "111122223333",
    "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/mydomain",
    "status": "Active",
    "createdTime": 1583075193.861,
    "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/a73que8sq-ba...",
    "repositoryCount": 13,
    "assetSizeBytes": 513830295,
    "s3BucketArn": "arn:aws:s3:::assets-787052242323-us-west-2"
  }
}
```

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.

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::assets-193858265520-us-east-1/*"]
    }
  ]
}
```

Use CodeArtifact from a VPC

If you cannot or do not want to enable private DNS on your `com.amazonaws.`*`region`*`.codeartifact.repositories` VPC endpoint that you created in Create VPC endpoints for CodeArtifact, you must use a different configuration for the repositories endpoint to use CodeArtifact from a VPC. Follow the instructions in Use the `codeartifact.repositories` endpoint without private DNS to configure CodeArtifact if the `com.amazonaws.`*`region`*`.codeartifact.repositories` endpoint does not have private DNS enabled.

Use the `codeartifact.repositories` endpoint without private DNS

If you cannot or do not want to enable private DNS on your `com.amazonaws.`*`region`*`.codeartifact.repositories` VPC endpoint that you created in Create VPC endpoints for CodeArtifact, you must follow these instructions to configure your package manager with the correct CodeArtifact URL.

1. Run the following command to find a VPC endpoint to use to override the hostname.

```
$ aws ec2 describe-vpc-endpoints --filters Name=service-
name,Values=com.amazonaws.region.codeartifact.repositories \
  --query 'VpcEndpoints[*].DnsEntries[*].DnsName'
```

   The output looks like the following.

```
[
  [
    "vpce-0743fe535b883ffff-76ddffff.d.codeartifact.us-west-2.vpce.amazonaws.com"
  ]
]
```

2. Update the VPC endpoint path to include the package format, your CodeArtifact domain name, and CodeArtifact repository name. See the following example.

```
https://vpce-0743fe535b883ffff-76ddffff.d.codeartifact.us-
west-2.vpce.amazonaws.com/format/d/domain_name-domain_owner/repo_name
```

   Replace the following fields from the example endpoint.

   - *`format`*: Replace with a valid CodeArtifact package format, for example, `npm` or `pypi`.
   - *`domain_name`*: Replace with the CodeArtifact domain that contains the CodeArtifact repository that hosts your packages.
   - *`domain_owner`*: Replace with the ID of the owner of the CodeArtifact domain, for example, `111122223333`.
   - *`repo_name`*: Replace with the CodeArtifact repository that hosts your packages.

   The following URL is an example npm repository endpoint.

```
https://vpce-0dc4daf7fca331ed6-et36qa1d.d.codeartifact.us-
west-2.vpce.amazonaws.com/npm/d/domainName-111122223333/repoName
```

3. Configure your package manager to use the updated VPC endpoint from the previous step. You must configure the package manager without using the CodeArtifact `login` command. For configuration instructions for each package format, see the following documentation.

   - npm: Configuring npm without using the login command
   - nuget: Configure nuget or dotnet without the login command
   - pip: Configure pip without the login command
   - twine: Configure and use twine with CodeArtifact
   - Gradle: Use CodeArtifact with Gradle
   - mvn: Use CodeArtifact with mvn

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.

```
{
  "Statement": [
    {
      "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, 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, repositories, and package groups in AWS CloudFormation. For more information, including examples of JSON and YAML templates, see the following topics in the *AWS CloudFormation User Guide*:

- AWS::CodeArtifact::Domain
- AWS::CodeArtifact::Repository
- AWS::CodeArtifact::PackageGroup

Preventing deletion of CodeArtifact resources

CodeArtifact repositories contain critical aplication dependencies that may not be easy to recreate if lost. To protect CodeArtifact resources against accidential deletion when managing CodeArtifact resources with CloudFormation, include the `DeletionPolicy` and `UpdateRetainPolicy` attributes with a value of `Retain` on all domains and respositories. This will prevent deletion if the resource is removed from the stack template, or the entire stack is accidentially deleted. The following YAML snippet shows a basic domain and repository with these attributes:

```
Resources:
    MyCodeArtifactDomain:
        Type: 'AWS::CodeArtifact::Domain'
        DeletionPolicy: Retain
        UpdateReplacePolicy: Retain
        Properties:
            DomainName: "my-domain"

    MyCodeArtifactRepository:
        Type: 'AWS::CodeArtifact::Repository'
        DeletionPolicy: Retain
        UpdateReplacePolicy: Retain
        Properties:
            RepositoryName: "my-repo"
            DomainName: !GetAtt MyCodeArtifactDomain.Name
```

For more information about these attributes, see DeletionPolicy and UpdateReplacePolicy in the *AWS CloudFormation User Guide*.

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 AWS CodeArtifact

The following information might help you troubleshoot common issues with CodeArtifact.

For information about troubleshooting format-specific issues, see the following topics:

- Maven troubleshooting
- Swift troubleshooting

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 and best practices for using tags, see the Best Practices for Tagging AWS Resources Whitepaper.

You can tag the following resource types in CodeArtifact:

- Tag a repository in CodeArtifact
- Tag a domain in CodeArtifact

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.

CodeArtifact cost allocation with tags

You can use tags to allocate both storage and request costs in CodeArtifact.

Allocating data storage costs in CodeArtifact

Data storage costs are tied to domains, therefore to allocate your CodeArtifact storage costs, you can use any tags that are applied to your domains. For information about adding tags to domains, see Tag a domain in CodeArtifact.

Allocating request costs in CodeArtifact

Most request usage is tied to repositories, therefore to allocate your CodeArtifact requests costs, you can use any tags that are applied to your repositories. For information about adding tags to repositories, see Tag a repository in CodeArtifact.

Some request types are associated with domains rather than repositories, so the request usage and costs related to the requests will be allocated to the tags on the domain. The best way to determine if a request type is associated with a domain or a repository is to use the Actions defined by AWS CodeArtifact table in the *Service Authorization Reference*. Find the request type in the **Actions** column, and look at the value in the corresponding **Resources types** column. If the resource type is **domain**, requests of that type will be billed to the domain. If the resource type is **repository** or **package**, requests of that type will be billed to the repository. Some actions show both resource types, for those actions the billed resource depends on what value is passed in the request.

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 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.

Name	Default	Adjustable	Description
Asset file size	Each supported Region: 5 Gigabytes	Yes	The maximum file size per asset.
Assets per package version	Each supported Region: 150	No	The maximum number of assets per package version.
CopyPackageVersions requests per second	Each supported Region: 5	Yes	The maximum number of calls that can be made to CopyPackageVersions per second.
Direct upstreams per repository	Each supported Region: 10	No	The maximum number of direct upstream repositories per repository.
Domains per AWS account	Each supported Region: 10	Yes	The maximum number of domains that can be created per AWS account.
GetAuthorizationToken requests per second	Each supported Region: 40	Yes	The maximum number of authorization tokens retrieved per second.

Name	Default	Adjuse	Description
GetPackageVersionAsset requests per second	Each supported Region: 50	Yes	The maximum number of calls that can be made to GetPackageVersionAsset per second.
ListPackageVersionAssets requests per second	Each supported Region: 200	Yes	The maximum number of calls that can be made to ListPackageVersionAssets per second.
ListPackageVersions requests per second	Each supported Region: 200	Yes	The maximum number of calls that can be made to ListPackageVersions per second.
ListPackages requests per second	Each supported Region: 200	Yes	The maximum number of calls that can be made to ListPackages per second.
PublishPackageVersion requests per second	Each supported Region: 10	Yes	The maximum number of calls that can be made to PublishPackageVersion per second.
Read requests per second from a single AWS account	Each supported Region: 800	Yes	The maximum number of read requests from one AWS account per second.
Repositories per domain	Each supported Region: 1,000	Yes	The maximum number of repositories that can be created per domain.

Name	Default	Adjustable	Description
Requests per second using a single authentication token	Each supported Region: 1,200	No	The maximum number of requests per second using a single authentication token.
Requests without authentication token per IP address	Each supported Region: 600	No	The maximum number of requests per second without an authentication token from a single IP address.
Upstream repositories searched	Each supported Region: 25	No	The maximum number of upstream repositories searched when resolving a package.
Write requests per second from a single AWS account	Each supported Region: 100	Yes	The maximum number of write requests from one AWS account per second.

Note

In general, each read request made to CodeArtifact counts as one request counted against a quota. However, for the Ruby package format, a single read request to the `/api/v1/dependencies` operation can request data about multiple packages.
For example, the request can look like `https://${CODEARTIFACT_REPO_ENDPOINT}/api/v1/dependencies?gems=gem1,gem2.gem3`. In this example, the request counts as three requests against the quota.
Note that the multiple requests only applies to service quotas, not billing. In the example, you will be billed only for one request, although it counts as three requests towards the service quota.

AWS CodeArtifact user guide document history

The following table describes important changes to the documentation for CodeArtifact.

Change	Description	Date
Added documentation for configuring and using Cargo with CodeArtifact	CodeArtifact now supports Cargo crates. Added documentation with guidance on configuring Cargo to use CodeArtifact repositories. For more information, see Using CodeArtifact with Cargo.	June 20, 2024
Added documentation for configuring and using Ruby with CodeArtifact	CodeArtifact now supports Ruby gems. Added documentation with guidance on configuring Ruby package managers to use CodeArtif act repositories. For more information, see Using CodeArtifact with Ruby.	April 30, 2024
Added an example key policy for creating domains with a customer managed AWS KMS key	Added an example key policy that can be used to create a customer managed KMS key for encrypting assets in CodeArtifact domains. For more information, see Example AWS KMS key policy.	April 18, 2024
Added documentation to support the launch of package groups.	Added documentation about managing and using package groups in CodeArtifact. For more information, see Working with package groups in CodeArtifact.	March 21, 2024

Added additional valid package managers to documentation about the *aws codeartifact login* command.	Added `dotnet`, `nuget`, and `swift` to the list of valid package managers to use with the `aws codeartifact login` command. For more information, see AWS CodeArtifact authentication and tokens.	February 18, 2024
Added an entry to the Swift troubleshooting documentation about Xcode hanging on CI machines	Added information, including a solution, about an issue that can cause Xcode to hang on CI machines due to keychain prompt for password. For more information, see Xcode hangs on CI machine due to keychain prompt for password.	February 6, 2024
Added information about troubleshooting slow npm package install times with npm 8.x or higher	Added information about working around slow npm package install times from CodeArtifact, which could cause slow build times. For more information, see Troubleshooting slow installs with npm 8.x or higher.	December 29, 2023
Updated information about Python package asset and metadata behavior in CodeArtifact	Updated information about how CodeArtifact repositories retain and refresh Python package version assets and metadata. For more information, see Requesting Python packages from upstreams and external connections.	December 14, 2023

Reorganized documentation about monitoring CodeArtifact	Reorganized information about monitoring CodeArtifact events, and added information about viewing CodeArtifact requests with Amazon CloudWatch metrics. For more information, see Monitoring CodeArtifact.	December 14, 2023
Added more information about managing CodeArtifact resources with AWS CloudFormation	Added references and links to documentation about managing CodeArtifact resources with CloudFormation, including a section about preventing deletion of CodeArtifact resources managed with CloudFormation. For more information, see Preventing deletion of CodeArtifact resources.	December 7, 2023
Added documentation detailing CodeArtifact's support of AWS KMS External Key Stores (XKS)	Added a section with information about CodeArtifact's support of KMS keys, including using XKS keys with CodeArtifact. For more information, see Types of AWS KMS keys supported in CodeArtifact.	October 31, 2023

Updated existing and added new troubleshooting documentation	Added a Maven troubleshooting topic and included links to Swift and Maven troubleshooting documentation in the general troubleshooting topic. For more information, see Troubleshooting AWS CodeArtifact.	September 28, 2023
Updated documentation to include the Swift Package Manager publish command	Swift 5.9 introduced a `swift package-registry publish` command to create and publish a Swift package to a package repository. Updated the Swift documentation to include instructions for using that command. For more information, see Using CodeArtifact with Swift.	September 25, 2023
Added documentation for configuring CodeArtifact with Swift	CodeArtifact now supports Swift packages. Added documentation with guidance on configuring Swift to use CodeArtifact repositories. For more information, see Using CodeArtifact with Swift.	September 20, 2023
Added guidance on how CodeArtifact handles yanked Python package versions	Added documentation with information about how to tell if a Python package version is yanked, how CodeArtifact handles yanked package versions, and answers to common questions. For more information, see Yanked package versions.	August 2, 2023

Fixed incorrect command line command in Yarn documentation	Fixed an incorrect command line command that fetches a CodeArtifact authorization token and stores it in an environment variable in the Yarn documentation.	July 20, 2023
Minor additions and small bug fix to Python documentation	Added pip and twine information in their respective documentation and corrected what happens when using the `codeartifact login` command with twine. For more information, see Configure and use pip with CodeArtifact and Configure and use twine with CodeArtifact.	July 14, 2023
Fixed incorrect dotnet commands in CodeBuild documentation	Corrected the `dotnet add package` commands in the Using NuGet packages in CodeBuild documentation.	July 13, 2023
Updated AWS CodeArtifact and AWS Identity and Access Management documentation	Overhauled the IAM in CodeArtifact documentation to add clarity and consistency with documentation for other AWS services. See Identity and Access Management for AWS CodeArtifact.	May 24, 2023

Added information about yanked Python package versions	Added information about how CodeArtifact retains yanked Python package version metadata, For more information, see Yanked package versions.	April 11, 2023
Added information on Clojure support	Added information about Clojure support, including managing dependencies for Clojure projects. For more information, see Use CodeArtifact with deps.edn.	March 21, 2023
Added information on generic package publishing	Added information about generic packages and how to publish and download package contents with the AWS CLI. For more information, see Using CodeArtifact with generic packages, Publishing and consuming generic packages, and Supported commands for generic packages.	March 10, 2023
Added information on asset size limits for publishing	Added a section to Package publishing to explain the asset size limits for publishing.	June 21, 2022

Refactored the external connection documentation	Moved the external connection documentation and reorganized it to focus on the end goal of the user, which is to connect their CodeArtifact repository to public package repositories. Also added more guidance and information around the different methods for achieving that goal. For more information, see Connect a CodeArtifact repository to a public repository.	May 9, 2022
Updated the CodeArtifact event information for Amazon CloudWatch Events	Added more information to the `account` field and added the `repositoryAdministrator` field. For more information, see CodeArtifact event format and example.	March 7, 2022
Added configuration instructions for using CodeArtifact from a VPC without private DNS	If you cannot or do not want to enable private DNS on your `codeartifact.repositories` VPC endpoint, you must use a different configuration for the repositories endpoint to use CodeArtifact from a VPC. See Use the `codeartifact.repositories` endpoint without private DNS for more information.	February 8, 2022

Added in-depth documentation for updating the status of package versions	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 for more information.	September 1, 2021
Updated the copy package versions documentation with more in-depth permissions information	Added more information about the required IAM and resource-based policy permissions for calling the `aws codeartifact copy-package-versions` 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 for more information.	August 25, 2021

Updated documentation for running a Gradle build in IntelliJ IDEA	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 `aws codeartifact get-authorization-token` . See Run a Gradle build in IntelliJ IDEA for more information.	August 23, 2021
Added documentation for configuring and using Yarn with AWS CodeArtifact	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 for more information.	July 30, 2021
AWS CodeArtifact now supports NuGet packages	CodeArtifact users can now publish and consume NuGet packages. Added documentation for configuring and using both Visual Studio and NuGet command line tools like `nuget` and `dotnet` with CodeArtifact repositories. See Using CodeArtifact with NuGet for more information.	November 19, 2020

Tagging resources in AWS CodeArtifact	Added documentation about tagging repositories and domains in AWS CodeArtifact. See Tagging resources.	October 30, 2020
CodeArtifact now supports AWS CloudFormation	CodeArtifact users can now use AWS CloudFormation templates to create CodeArtifact repositories and domains. See Creating CodeArtifact resources with AWS CloudFormation for more information and to get started.	October 8, 2020
Add information about creating Amazon S3 gateway endpoints to use CodeArtifact with Amazon VPC	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.	August 12, 2020
Publishing Maven artifacts with curl and publishing third-party Maven artifacts	Added guidance for Publishing with curl and Publish third-party artifacts.	August 10, 2020
General Availability (GA) release	Initial version of the CodeArtifact User Guide.	June 10, 2020