CodeArtifact ユーザーガイド

CodeArtifact

CodeArtifact: CodeArtifact ユーザーガイド

Table of Contents

AWS CodeArtifact とは

AWS CodeArtifact は、組織がアプリケーション開発用のソフトウェアパッケージを保存および共有するのに役立つ、安全でスケーラブルなマネージドアーティファクトリポジトリサービスです。CodeArtifact は、NuGet CLI、Maven、Gradle、npm、yarn、pip、twine などの一般的なビルドツールやパッケージマネージャーで使用できます。CodeArtifact によって、インフラストラクチャのスケーリングに関する懸念や、独自のアーティファクトストレージシステムを管理する必要がなくなります。CodeArtifact リポジトリに保存できるパッケージの数や合計サイズに制限はありません。

ユーザーは、プライベート CodeArtifact リポジトリと npmjs.com や Maven Central などの外部のパブリックリポジトリの間に接続を作成することができます。その後、CodeArtifact は、パッケージマネージャーからリクエストを受けた場合に、パブリックリポジトリからパッケージをオンデマンドで取得して保存します。これにより、アプリケーションで使用するオープンソースの依存関係を簡単に使用でき、構築や開発で依存関係をいつでも利用できるようになります。また、プライベートパッケージを CodeArtifact リポジトリに公開することもできます。これにより、組織内の複数のアプリケーションや開発チームに固有のソフトウェアコンポーネントを共有できるようになります。

詳細については、AWS CodeArtifactを参照してください。

CodeArtifactの仕組みとは?

CodeArtifactはソフトウェアパッケージをリポジトリに格納します。リポジトリはポリグロットです-単一のリポジトリには、サポートされている任意のタイプのパッケージを含めることができます。すべての CodeArtifactリポジトリは、単一のCodeArtifactドメインのメンバーです。1 つまたは複数のリポジトリを持つ組織では、1 つのプロダクションドメインを使用することをお勧めします。例えば、各リポジトリを別の開発チーム用に使用することがあります。リポジトリ内のパッケージは、複数開発チーム間で検出して共有できます。

リポジトリにパッケージを追加するには、npm や Maven などのパッケージマネージャーがリポジトリのエンドポイント (URL) を使用するように設定します。その後、パッケージマネージャーを使用して、パッケージをリポジトリに公開できます。また、npmjs、NuGet Gallery、Maven Central、PyPI などのパブリックリポジトリへの外部接続を設定することで、オープンソースパッケージをリポジトリにインポートすることもできます。詳細については、「CodeArtifact リポジトリをパブリックリポジトリに接続する」を参照してください。

あるリポジトリ内のパッケージを、同じドメイン内の別のリポジトリで使用できるようにすることができます。これを行うには、一方のリポジトリをもう一方のリポジトリのアップストリームとして設

定します。アップストリームリポジトリで使用可能なすべてのパッケージバージョンは、ダウンストリームリポジトリでも使用できます。さらに、パブリックリポジトリへの外部接続を介してアップストリームリポジトリで使用できるすべてのパッケージは、ダウンストリームリポジトリでも使用できます。詳細については、「CodeArtifact でアップストリームリポジトリを操作する」を参照してください。

CodeArtifactでは、パッケージバージョンを公開したり消費したりするために、ユーザーがサービスで認証を受ける必要があります。CodeArtifactサービスを利用するには、自分の AWS 資格情報を使って認可トークンを作成し、認証を受ける必要があります。CodeArtifact リポジトリのパッケージは公開できません。CodeArtifact の認証とアクセスに関する詳細については、「AWSCodeArtifact認証とトークン」を参照してください。

AWS CodeArtifact の概念

CodeArtifactを使用するときに知っておくべき概念と用語をいくつか紹介します。

トピック

- アセット
- ドメイン
- Repository
- パッケージ
- パッケージグループ
- パッケージ名前空間
- パッケージバージョン
- パッケージバージョンリビジョン
- アップストリームリポジトリ

アセット

アセットとは、CodeArtifact に格納されている、npm `.tgz` ファイルや Maven POM、JAR ファイルなど、パッケージのバージョンに関連付けられた個々のファイルのことです。

ドメイン

リポジトリは、ドメインと呼ばれる上位レベルのエンティティに集約されます。すべてのパッケージアセットとメタデータはドメインに格納されますが、リポジトリを通じて消費されます。Maven

JARファイルなどの特定のパッケージアセットは、存在するリポジトリの数に関係なく、ドメインごとに 1 回保存されます。ドメイン内のすべてのアセットとメタデータは、 AWS Key Management Service () に保存されているのと同じ AWS KMS key (KMS キー) で暗号化されますAWS KMS。

各リポジトリは単一のドメインのメンバーであり、別のドメインに移動することはできません。

ドメインを使用すると、複数のリポジトリに組織ポリシーを適用できます。このアプローチを使用して、どのアカウントがドメイン内のリポジトリにアクセスできるか、どのパブリックリポジトリをパッケージのソースとして使用できるかを決定します。

1 つの組織が複数のドメインを使用することもできますが、公開されたアーティファクトをすべて含む 1 つの本番ドメインを使用することをお勧めします。そうすることで、チームは組織全体でパッケージを見つけて共有できます。

Repository

CodeArtifactリポジトリには、それぞれがアセットのセットにマップされているパッケージバージョンのセットが含まれている。リポジトリはポリグロットです。単一のリポジトリには、サポートされているあらゆるタイプのパッケージを格納することができます。各リポジトリは、nuget CLI、npm CLI、Maven CLI (mvn)、pip などのツールを使用してパッケージを取得および公開するためのエンドポイントを公開します。ドメインごとに最大 1,000 個のリポジトリを作成できます。

パッケージ

[パッケージ] とは、依存関係の解決とソフトウェアのインストールに必要なソフトウェアとメタデータのバンドルです。CodeArtifact には、パッケージはパッケージ名、`@types/node`に`@types`などの[ネームスペース] (オプション) など、パッケージバージョンのセット、およびパッケージレベルのメタデータ (npm タグなど) が含まれます。

AWS CodeArtifact は、Cargo、ジェネリック、Maven、npm、NuGet、PyPI、Ruby、Swift パッケージ形式をサポートしています。

パッケージグループ

パッケージグループを使用すると、パッケージ形式、パッケージ名前空間、パッケージ名を使用して、定義したパターンに一致する複数のパッケージに設定を適用できます。パッケージグループを使用すると、複数のパッケージのパッケージオリジンコントロールをより簡単に設定できます。パッケージオリジンコントロールは、新しいパッケージバージョンの取り込みや公開をブロックまたは許

可するために使用します。これにより、依存関係置換攻撃と呼ばれる悪意のあるアクションからユーザーを保護できます。

パッケージ名前空間

パッケージ形式によっては、パッケージを論理グループに整理し、階層的なパッケージ名をサポートし名前の衝突を回避します。例えば、npm はスコープをサポートします。詳細については、「npm scopes documentation」を参照してください。npm パッケージ`@types/node`はスコープが`@types`、名前が`node`です。他にも多くのパッケージ名が`@types`スコープにあります。CodeArtifactでは、スコープ (“タイプ”) はパッケージ名前空間と呼ばれ、名前 (“ノード”) はパッケージ名と呼ばれます。Maven パッケージの場合、パッケージ名前空間はMaven GroupIDに対応します。Mavenパッケージ`org.apache.logging.log4j:log4j` は、groupID (パッケージの名前空間)が`org.apache.logging.log4j`、artifactID (パッケージ名) が`log4j`です。ジェネリックパッケージには名前空間が必要です。PyPI などの一部のパッケージ形式では、npmスコープやMaven GroupIDのような概念を持つ階層名をサポートしていません。パッケージ名をグループ化する方法がないと、名前の衝突を回避するのが難しくなります。

パッケージバージョン

パッケージバージョンは、`@types/node 12.6.9`のようにパッケージの特定のバージョンを識別します。バージョン番号の形式とセマンティクスは、パッケージ形式によって異なります。例えば、npmパッケージのバージョンはセマンティックバージョニングの仕様 に準拠する必要があります。CodeArtifactでは、パッケージバージョンは、バージョン識別子、パッケージバージョンレベルのメタデータ、およびアセットセットで構成されます。

パッケージバージョンリビジョン

パッケージバージョンリビジョンは、パッケージバージョンの特定のアセットとメタデータのセットを識別する文字列です。パッケージバージョンが更新されるたびに、新しいパッケージバージョンリビジョンが作成されます。例えば、あるバージョンの Python パッケージのソース配布用アーカイブ (sdist) を公開し、後で同じバージョンにコンパイルされたコードを含む Python ホイールを追加することができます。ホイールを公開すると、新しいパッケージバージョンリビジョンが作成されます。

アップストリームリポジトリ

ダウンストリームリポジトリのリポジトリエンドポイントからあるリポジトリ内のパッケージバージョンにアクセスできる場合、このリポジトリはアップストリームリポジトリになります。このアプローチは、クライアントの観点から見ると、2 つのリポジトリのコンテンツを効果的に統合しま

す。CodeArtifact を使用すると、2 つのリポジトリ間にアップストリームリレーションシップを作成できます。

CodeArtifactの使用を開始するにはどうしたらいいですか?

次の手順を実行することをお勧めします。

1. 「AWS CodeArtifact の概念」を参照して CodeArtifact の詳細について学ぶ。
2. の手順に従って AWS アカウント、 AWS CLI、、および IAM ユーザーを設定しますAWS CodeArtifact でのセットアップ。
3. 「CodeArtifact の開始方法」の指示に従って、CodeArtifact を使用する。

AWS CodeArtifact でのセットアップ

Amazon Web Services (AWS) に既にサインアップしている場合は、 AWS CodeArtifact の使用をすぐに開始できます。CodeArtifact コンソールを開き、ドメインとリポジトリの作成 を選択し、Launch Wizard (起動ウィザード) のステップに従って、最初のドメインとリポジトリを作成します。

AWS にまだサインアップしていない場合、または最初のドメインとリポジトリの作成にサポートが必要な場合は、次のタスクを実行して CodeArtifact を使用するようにセットアップします。

トピック

にサインアップする AWS アカウント

の使用を開始するには AWS、 が必要です AWS アカウント。の作成の詳細については AWS アカウント、「 AWS アカウント管理 リファレンスガイド」の「 の開始方法 AWS アカウント」を参照してください。

をインストールまたはアップグレードしてから設定する AWS CLI

ローカル開発マシンで AWS Command Line Interface (AWS CLI) から CodeArtifact コマンドを呼び出すには、 をインストールする必要があります AWS CLI。

古いバージョンの AWS CLI がインストールされている場合は、CodeArtifact コマンドが使用可能になるようにアップグレードする必要があります。CodeArtifact コマンドは、次の AWS CLI バージョンで使用できます。

1. AWS CLI 1: 1.18.77 以降
2. AWS CLI 2: 2.0.21 以降

バージョンを確認するには、`aws --version` コマンドを使用します。

をインストールして設定するには AWS CLI

1. 「 のインストール」の手順 AWS CLI で をインストールまたはアップグレードします。 AWS Command Line Interface
2. 次のように AWS CLI、configure コマンドを使用して を設定します。

   ```
   aws configure
   ```

   プロンプトが表示されたら、CodeArtifact で使用する IAM ユーザーの AWS アクセスキーと AWS シークレットアクセスキーを指定します。デフォルトの AWS リージョン 名の入力を求められたら、パイプラインを作成するリージョン (us-east-2 など) を指定します。デフォルトの出力形式の入力を求められたら、json を指定します。

   > **Important**
   >
   > を設定すると AWS CLI、 を指定するように求められます AWS リージョン。「AWS 全般のリファレンス」の「Region and Endpoints」に記載されているサポートされているリージョンから一つを選びます。

   詳細については、「Configuring the AWS Command Line Interface」および「Managing access keys for IAM users」を参照してください。
3. インストールまたはアップグレードを確認するには、 AWS CLIから次のコマンドを呼び出します。

   ```
   aws codeartifact help
   ```

   成功すると、このコマンドにより使用できる CodeArtifact コマンドのリストが表示されます。

IAM ユーザーを作成して、そのユーザーに CodeArtifact へのアクセス許可を付与できます。詳細については、「IAM ユーザーのプロビジョニング」を参照してください。

IAM ユーザーのプロビジョニング

以下の手順に従って、CodeArtifact を使用するための IAM ユーザーを準備します。

IAM ユーザーをプロビジョニングするには

1. IAM ユーザーを作成するか、 AWS アカウントに関連付けられた既存のユーザーを使用します。詳細については、IAM ユーザーガイドの「IAM ユーザーの作成」およびAWS 「IAM ポリシーの概要」を参照してください。
2. IAM ユーザーに CodeArtifact へのアクセス許可を付与します。

   - オプション 1:カスタム IAM ポリシーを作成します。カスタム IAM ポリシーでは、最低限必要なアクセス許可を提供し、認証トークンの持続時間を変更できます。詳細とポリシー例については、「AWS CodeArtifact のアイデンティティベースのポリシーの例」を参照してください。
   - オプション 2: `AWSCodeArtifactAdminAccess` AWS 管理ポリシーを使用します。次のスニペットは、このポリシーの内容を示しています。

     > **Important**
     >
     > このポリシーにより、CodeArtifact API へのフルアクセスが付与されます。常に必要最小限のアクセス許可を使用してタスクを達成してください。詳細については、「IAM ユーザーガイド」の「IAM ベストプラクティス」を参照してください。

     JSON

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

```
                }
            }
        }
    ]
}
```

> **Note**
>
> IAM ユーザーまたはロールポリシーに `sts:GetServiceBearerToken` アクセス許可を追加する必要があります。アクセス許可は、CodeArtifact ドメインまたはリポジトリリソースポリシーに追加できますが、リソースポリシーには効果がありません。

CodeArtifact `GetAuthorizationToken` API を呼び出すには、`sts:GetServiceBearerToken` アクセス許可が必要です。この API は、CodeArtifact で npm または `pip` のようなパッケージマネージャーを使用するときに、使用する必要があるトークンを返します。CodeArtifact リポジトリでパッケージマネージャーを使用するには、IAM ユーザーまたはロールが、上記のポリシー例に示すように `sts:GetServiceBearerToken` を許可する必要があります。

CodeArtifact で使用する予定のパッケージマネージャーまたはビルドツールをインストールしていない場合は、パッケージマネージャーまたはビルドツールをインストールする を参照してください。

パッケージマネージャーまたはビルドツールをインストールする

CodeArtifact のパッケージを公開または使用するには、パッケージマネージャーを使用する必要があります。パッケージタイプごとに異なるパッケージマネージャーがあります。CodeArtifact で使用できるパッケージマネージャーの一部を次のリストに示します。インストールしていない場合は、使用する予定のパッケージタイプのパッケージマネージャーをインストールします。

- npm の場合は、npm CLI または pnpm を使用します。
- Maven の場合は、Apache Maven (`mvn`) または Gradle のいずれかを使用します。
- Python の場合は、pip を使用してパッケージをインストールし、twine を使用してパッケージを公開します。
- NuGet の場合は、Visual Studio で Visual Studio 用のツールキット を使用するか、CLI で nuget または dotnet を使用します。
- ジェネリックパッケージの場合は、AWS CLI または SDK を使用してパッケージコンテンツを公開およびダウンロードします。

次の手順

次のステップは、CodeArtifact で使用している 1 つまたは複数のパッケージタイプ、および CodeArtifact リソースの状態によって異なります。

あなた自身、チーム、または組織で初めて CodeArtifact を使い始める場合は、以下のドキュメントで一般的な入門情報や必要なリソースの作成について確認してください。

- コンソールを使用した開始方法
- AWS CLI を使用した開始方法

リソースがすでに作成されていて、CodeArtifact リポジトリにパッケージをプッシュしたり、CodeArtifact リポジトリからパッケージをインストールしたりするようにパッケージマネージャーを設定する準備ができている場合は、パッケージタイプまたはパッケージマネージャーに対応するドキュメントを参照してください。

- CodeArtifactをnpmで使用する
- PythonでCodeArtifactを使う
- MavenでCodeArtifactを使う
- NuGetでCodeArtifactを使う
- CodeArtifact でのジェネリックパッケージの使用

CodeArtifact の開始方法

この入門チュートリアルでは、CodeArtifact を使用して以下を作成します。

- `my-domain` というドメイン。
- `my-domain` に含まれる `my-repo` というリポジトリ。
- `my-domain` に含まれる `npm-store` というリポジトリ。`npm-store` には npm パブリックリポジトリへの外部接続があります。この接続は、npm パッケージを `my-repo` リポジトリに取り込むために使用されます。

このチュートリアルを開始する前に、CodeArtifact AWS CodeArtifact の概念 をご一読ください。

> **Note**
>
> このチュートリアルでは、AWS アカウントに課金される可能性のあるリソースを作成する必要があります。詳細については、[CodeArtifact の料金] を参照してください。

トピック

前提条件

AWS マネジメントコンソール または AWS Command Line Interface(AWS CLI) を使って、このチュートリアルを完了できます。チュートリアルに従うには、まず次の前提条件を完了する必要があります。

- 「AWS CodeArtifact でのセットアップ」の各ステップを完了します。
- npm CLI をインストールします。詳細については、npm ドキュメントの [Node.js と npm のダウンロードとインストール] を参照してください。

コンソールを使用した開始方法

AWS マネジメントコンソールを使用して CodeArtifact の使用を開始するには、以下のステップを実行します。このガイドでは、npm パッケージマネージャーを使用します。別のパッケージマネージャーを使用している場合は、次の手順の一部を変更する必要があります。

1. AWS マネジメントコンソール にサインインして、https://console.aws.amazon.com/codesuite/codeartifact/start で AWS CodeArtifact コンソールを開きます。詳細については、「AWS CodeArtifact でのセットアップ」を参照してください。
2. [リポジトリの作成]を選択します。
3. [リポジトリ名] に「**my-repo**」と入力します。
4. (オプション) [Repository Description] (リポジトリの説明)で、リポジトリの説明を入力します。
5. [パブリックアップストリームリポジトリ] で [npm-store] を選択し、`my-repo` リポジトリの上流にある npmjs に接続されたリポジトリを作成します。

   CodeArtifact は、このリポジトリに名前 `npm-store` を割り当てます。アップストリームリポジトリで利用可能なすべてのパッケージ `npm-store` は、その下流のリポジトリ `my-repo` でも使用可能です。
6. [Next] (次へ) を選択します。
7. [AWS アカウント] で、[この AWS アカウント] を選択します。
8. [ドメイン名] で **my-domain** と入力します。
9. [Additional configuration (追加設定)] を展開します。
10. ドメイン内のすべてのアセットを暗号化するためには、AWS KMS key (KMS キー) を使用する必要があります。AWS マネージドキーまたは、管理する KMS キーを使用できます。

    - デフォルトAWS マネージドキーを使用するには、[AWS マネージドキー] を選択してください。
    - 管理している KMS キーを使用する場合、[カスタマーマネージドキー] を選択してください。[カスタマーマネージドキー ARN] で管理している KMS キーを使用するには、KMS キーを検索して選択します。

    詳細については、[AWS Key Management Service デベロッパーガイド] の AWS マネージドキーおよび [カスタマーマネージドキー] を参照してください。
11. [Next] (次へ) を選択します。
12. [確認と作成] で、CodeArtifact が作成しているものを確認します。

- [パッケージフロー] は、my-domain、my-repo および npm-store の関連性を示します。
- [ステップ 1: リポジトリを作成する] は、my-repo および npm-store の詳細を表示します。。
- [ステップ 2: ドメインを選択する] は、my-domain の詳細を表示します。

準備が完了したら、[リポジトリの作成] を選択します。

13. [my-repo] ページで [接続手順の表示] を選択してから、[npm] を選択します。
14. AWS CLI を使用して、[この AWS CLI CodeArtifact コマンドを使用して npm クライアントを設定する] に示されている login コマンドを実行します。

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

ログインが成功したことを確認する出力が表示されます。

```
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
```

エラー Could not connect to the endpoint URL が発生した場合は、AWS CLI が設定されており、[デフォルトリージョン名] がリポジトリを作成したのと同じリージョンに設定されていることを確認してください。「Configuring the AWS Command Line Interface」を参照してください。

詳細については、「CodeArtifact で npm を設定して使用する」を参照してください。

15. npm CLI を使用して npm パッケージをインストールします。例えば、一般的な npm パッケージ lodash をインストールするには、以下のコマンドを使用します。

```
npm install lodash
```

16. CodeArtifact コンソールに戻ります。[my-repo] リポジトリが開いている場合は、ページを更新します。それ以外の場合は、ナビゲーションペインで [リポジトリ] 選択してから、[my-repo] を選択します。

[パッケージ]で、インストールした npm ライブラリまたはパッケージが表示されます。パッケージの名前を選択して、そのバージョンとステータスを表示できます。最新バージョンを選択して、依存関係、アセットなどのパッケージの詳細を表示できます。

> **Note**
>
> パッケージをインストールしてからリポジトリに取り込むまでの間に遅延が生じる場合があります。

17. 更なるAWS の課金を回避するには、このチュートリアルで使用したリソースを削除します。

> **Note**
>
> リポジトリを含むドメインは削除できないため、`my-domain` を削除する前に `my-repo` および`npm-store` を削除する必要があります。

a. ナビゲーションペインで、[リポジトリ] を選択します。
b. [npm ストア] を選択して [削除] を選択し、ステップに従ってリポジトリを削除します。
c. [my-repo] を選択して [削除] を選択し、ステップに従ってリポジトリを削除します。
d. ナビゲーションペインで、[ドメイン] を選択します。
e. [my-dain] を選択して [削除] を選択し、ステップに従ってドメインを削除します。

AWS CLI を使用した開始方法

AWS Command Line Interface (AWS CLI) を使用して CodeArtifactを開始するには、以下のステップを実行します。詳細については、「をインストールまたはアップグレードしてから設定する AWS CLI」を参照してください。このガイドでは、npm パッケージマネージャーを使用します。別のパッケージマネージャーを使用している場合は、次のステップの一部を変更する必要があります。

1. AWS CLI を使用して create-domain コマンドを実行します。

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

JSON 形式のデータが、新しいドメインの詳細とともに出力に表示されます。

```
{
    "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
    }
}
```

エラー Could not connect to the endpoint URL が発生した場合は、AWS CLI が設定されており、[デフォルトリージョン名] がリポジトリを作成したのと同じリージョンに設定されていることを確認してください。「Configuring the AWS Command Line Interface」を参照してください。

2. create-repositoryコマンドを使用して、ドメインにリポジトリを作成します。

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

JSON 形式のデータが、新しいリポジトリの詳細とともに出力に表示されます。

```
{
    "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. create-repository コマンドを使用して、my-repo リポジトリのアップストリームリポジトリを作成します。

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

JSON 形式のデータが、新しいリポジトリの詳細とともに出力に表示されます。

```
{
    "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. associate-external-connection コマンドを使用して、npm 公開リポジトリへの外部接続をnpm-store リポジトリに追加します。

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

JSON 形式のデータが、リポジトリとその新しい外部接続についての詳細とともに出力に表示されます。

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

詳細については、「CodeArtifact リポジトリをパブリックリポジトリに接続する」を参照してください。

5. update-repository コマンドを使用して、npm-store リポジトリを my-repo リポジトリへのアップストリームリポジトリとして関連付けます。

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

JSON 形式のデータが、新しいアップストリームリポジトリを含む、更新されたリポジトリの詳細とともに出力に表示されます。

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

詳細については、「アップストリームリポジトリを追加または削除する (AWS CLI)」を参照してください。

6. login コマンドを使用して、my-repo リポジトリに npm パッケージマネージャーを設定します。

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

ログインが成功したことを確認する出力が表示されます。

```
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
```

詳細については、「CodeArtifact で npm を設定して使用する」を参照してください。

7. npm CLI を使用して npm パッケージをインストールします。例えば、一般的な npm パッケージ lodash をインストールするには、以下のコマンドを使用します。

```
npm install lodash
```

8. list-packages コマンドを使用して、my-repo リポジトリでインストールしたばかりのパッケージを表示します。

> **Note**
>
> npm install インストールコマンドが完了してからリポジトリに表示するまでの間に遅延が生じる場合があります。パブリックリポジトリからパッケージを取得するときの一般的なレイテンシーの詳細については、「外部接続のレイテンシー」を参照してください。

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

JSON 形式のデータが、インストールしたパッケージの形式と名前とともに出力に表示されます。

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

```
    ]
}
```

これで、次の三つの CodeArtifact リソースが作成されました。

- ドメインmy-domain。
- my-domain に含まれるリポジトリ my-repo。このリポジトリには、利用可能な npm パッケージがあります。
- my-domain に含まれるリポジトリ npm-store。このリポジトリは、パブリック npm リポジトリへの外部接続を持ち、アップストリームリポジトリとして my-repo リポジトリに関連付けられています。

9. 更なる AWS の課金を回避するには、このチュートリアルで使用したリソースを削除します。

> **Note**
>
> リポジトリを含むドメインは削除できないため、my-domain を削除する前に my-repo および npm-store を削除する必要があります。

a. npm-store リポジトリを削除するには、delete-repository コマンドを使用します。

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

JSON 形式のデータが、削除されたリポジトリの詳細とともに出力に表示されます。

```
{
    "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. npm-store リポジトリを削除するには、 delete-repository コマンドを使用します。

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

JSON 形式のデータが、削除されたリポジトリの詳細とともに出力に表示されます。

```
{
    "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. my-domain リポジトリを削除するには、 delete-domain コマンドを使用します。

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

JSON 形式のデータが、削除されたドメインの詳細とともに出力に表示されます。

```
{
    "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
    }
}
```

CodeArtifact でのリポジトリの操作

これらのトピックでは、CodeArtifact コンソール AWS CLI、および CodeArtifact APIs を使用してリポジトリを作成、一覧表示、更新、削除する方法について説明します。

トピック

リポジトリの作成

CodeArtifact のすべてのパッケージはリポジトリに保存されるため、CodeArtifact を使用するには、リポジトリを作成する必要があります。CodeArtifact コンソール、 AWS Command Line Interface (AWS CLI)、または を使用してリポジトリを作成できます CloudFormation。各リポジトリは、作成時に使用する AWS アカウントに関連付けられます。ユーザーは複数のリポジトリを作成でき、リポジトリはドメインで作成およびグループ化されます。リポジトリを作成する際、パッケージは含まれていません。リポジトリはポリグロットで、単一のリポジトリには、サポートされている任意のタイプのパッケージを含めることができます。

単一のドメインで許可されるリポジトリの最大数など、CodeArtifact サービスの制限については、「AWS CodeArtifact のクォータ」を参照してください。許可されているリポジトリの最大数に達した場合は、リポジトリを削除して空き容量を増やすことができます。

リポジトリにはひとつ以上の CodeArtifact リポジトリをアップストリームリポジトリとして関連付けることができます。これにより、パッケージマネージャークライアントは、単一の URL エンドポイントを使用して、複数のリポジトリに含まれるパッケージにアクセスできます。詳細については、「CodeArtifact でアップストリームリポジトリを操作する」を参照してください。

CloudFormation を使用して CodeArtifact リポジトリを管理する方法の詳細については、「AWS CloudFormation での CodeArtifact リソースの作成」を参照してください。

> **Note**
>
> リポジトリを作成したら、名前、関連付けられた AWS アカウント、またはドメインを変更することはできません。

トピック

リポジトリを作成する (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで [リポジトリ] を選択し、[リポジトリの作成] を選択します。
3. [リポジトリ名] に、リポジトリの名前を入力します。
4. (オプション) [リポジトリの説明] で、このリポジトリの説明を任意で入力することができます。
5. (オプション) [アップストリームリポジトリの公開] で、Maven Central や npmjs.com などのパッケージ権限にリポジトリを接続する中間リポジトリを追加します。
6. [Next (次へ)] を選択します。
7. [AWS カウント] で、ドメインを所有するアカウントにサインインしている場合は、[この AWS アカウント] を選択します。別の AWS アカウントがドメインを所有している場合、[異なる AWS アカウント] を選択してください。
8. [ドメイン] で、リポジトリを作成するドメインを選択します。

   アカウントにドメインがない場合は、作成する必要があります。新しいドメインの名前を [ドメイン名] に入力します。

   [Additional configuration (追加設定)] を展開します。

   (AWS KMS key KMS キー) を使用して、ドメイン内のすべてのアセットを暗号化する必要があります。 AWS マネージドキー または管理する KMS キーを使用できます。

> **Important**
>
> CodeArtifact は、対称 KMS キーのみをサポートしています。非対称 KMS キー を使用して CodeArtifact ドメインを暗号化することはできません。KMS キーが対称か非対称かを判別するには、「対称および非対称 KMS キーを識別する」を参照してください。

- デフォルト AWS マネージドキーを使用するには、[AWS マネージドキー] を選択してください。
- 管理している KMS キーを使用する場合、[カスタマーマネージドキー] を選択してください。[カスタマーマネージドキー ARN] で管理している KMS キーを使用するには、KMS キーを検索して選択します。

詳細については、[AWS Key Management Service デベロッパーガイド] の AWS マネージドキー と [カスタマーマネージドキー] を参照してください。

9. [Next] (次へ) を選択します。
10. [レビューと作成] で、CodeArtifact が何を作成しているのかをレビューします。
    - [パッケージフロー] は、ドメインとリポジトリがどのように接続されているかを示しています。
    - [ステップ 1: リポジトリを作成する] に、作成されるリポジトリとオプションのアップストリームリポジトリの詳細が示されます。
    - [ステップ 2: ドメインの選択] は、`my_domain`に関する詳細を示します。

    準備が完了したら、 [リポジトリの作成] を選択します。

リポジトリを作成する (AWS CLI)

`create-repository`コマンドを使用して、ドメインにリポジトリを作成します。

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

出力の例:

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

新しいリポジトリにはパッケージは含まれていません。各リポジトリは、リポジトリの作成時に認証される AWS アカウントと関連付けられています。

タグ付きのリポジトリの作成

タグを使用してリポジトリを作成するには、--tagsパラメータをcreate-domainコマンドに追加してください。

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

アップストリームのリポジトリと一緒にリポジトリを作成

リポジトリを作成するときに、アップストリームリポジトリをひとつ以上指定できます。

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

出力の例:

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
```

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

> **Note**
>
> アップストリームリポジトリと一緒にリポジトリを作成するには、AssociateWithDownstreamRepositoryアップストリームリポジトリでのアクションの権限が必要です。

リポジトリ作成後に、アップストリームリポジトリを追加するには、アップストリームリポジトリを追加または削除する (コンソール)およびアップストリームリポジトリを追加または削除する (AWS CLI)を参照してください。

リポジトリへの接続

AWS アカウントに対する認証を行うようにプロファイルと認証情報を設定した後、CodeArtifact で使用するリポジトリを決定します。次のオプションがあります。

- リポジトリを作成します。詳細については、 [リポジトリの作成] を参照してください。
- アカウントにすでに存在するリポジトリを使用します。list-repositoriesコマンドを使用して、 AWS アカウントで作成したリポジトリを検索します。詳細については、「リポジトリを一覧表示させる」を参照してください。
- 別の AWS アカウントのリポジトリを使用します。詳細については、[リポジトリポリシー] を参照してください。

パッケージマネージャークライアントを使用する

使用するリポジトリがわかった後、次のいずれかのトピックを参照してください。

- Maven で CodeArtifact を使う
- npm で CodeArtifact を使う
- NuGetで CodeArtifact を使う
- Pythonで CodeArtifact を使う

リポジトリを削除する

CodeArtifact コンソールまたは AWS CLIを使用してリポジトリを削除できます。リポジトリを削除すると、そのリポジトリにパッケージをプッシュしたり、そこからパッケージをプルしたりすることはできなくなります。リポジトリ内のすべてのパッケージは永続的に使用できなくなり、復元できなくなります。同じ名前のリポジトリを作成できますが、その内容は空になります。

> **Important**
>
> リポジトリを削除したら、元に戻すことはできません。リポジトリの削除後は、そのリポジトリを復元できなくなります。

トピック

- リポジトリを削除する (コンソール)
- リポジトリを削除する (AWS CLI)
- リポジトリが削除されないように保護する

リポジトリを削除する (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで [リポジトリ] を選択し、その後、削除するリポジトリを選択します。
3. [削除] を選択し、次に、ステップに従ってドメインを削除します。

リポジトリを削除する (AWS CLI)

`delete-repository` コマンドを使用してリポジトリを削除します。

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

出力の例:

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

リポジトリが削除されないように保護する

次のようなドメインポリシーを含めることで、リポジトリが誤って削除されるのを防げます。

JSON

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

```
    ]
}
```

このポリシーにより、すべてのプリンシパルがリポジトリを削除できなくなりますが、後からリポジトリを削除する必要があると判断した場合は、以下の手順で削除できます。

1. ドメインポリシーで、次のようにポリシーを更新します。

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Sid": "DenyRepositoryDeletion",
               "Action": [
                   "codeartifact:DeleteRepository"
               ],
               "Effect": "Deny",
               "NotResource": "arn:aws:iam::*:role/Service*",
               "Principal": "*"
           }
       ]
   }
   ```

   repository-arn は、削除するリポジトリの ARN に置き換えてください。

2. AWS CodeArtifact コンソールで、リポジトリを選択し、選択したリポジトリを削除します。
3. リポジトリを削除したら、誤削除を防ぐためにポリシーを元に戻せます。

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Sid": "DenyRepositoryDeletion",
               "Action": [
                   "codeartifact:DeleteRepository"
   ```

```
            ],
            "Effect": "Deny",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

または、同じ拒否ステートメントをリポジトリポリシーに含めることができます。これにより、価値の高いリポジトリを削除から保護するために、より柔軟に対応できるようになります。

リポジトリを一覧表示させる

このトピックのコマンドを使用して、 AWS アカウントまたはドメインのリポジトリを一覧表示します。

AWS アカウントのリポジトリを一覧表示する

このコマンドを使用して、 AWS アカウント内のすべてのリポジトリを一覧表示します。

```
aws codeartifact list-repositories
```

サンプル出力:

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

list-repositoriesおよび--max-resultsパラメータを使用し、--next-tokenからの応答をページ分割できます。--max-resultsの場合、1 ~ 1000 の整数を指定して、単一ページに返される結果の数を指定できます。デフォルトは 50 に設定されています。後続ページを返すには、list-repositoriesをもう一度実行し、前のコマンド出力で受信したnextTokenの値を--next-tokenにパスします。--next-tokenオプションが使用されなければ、結果の最初のページが常に表示されます。

ドメインのリポジトリを一覧表示する

list-repositories-in-domainを使って、ドメイン内のすべてのリポジトリのリストを取得します。

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

出力結果を見ると、いくつかのリポジトリが異なる AWS アカウントで管理されていることがわかります。

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

--max-resultsおよび--next-tokenパラメータを使用し、list-repositories-in-domainの応答をページ分割できます。--max-resultsの場合、1 ~ 1000 の整数を指定して、単一ページに返される結果の数を指定できます。デフォルトは 50 に設定されています。後続ページを返すには、list-repositories-in-domainをもう一度実行し、前のコマンド出力で受信したnextTokenの値を--next-tokenにパスします。--next-tokenのオプションが使用されなければ、結果の最初のページが常に表示されます。

リポジトリ名をよりコンパクトなリストとして出力するには、次のコマンドを試してください。

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

サンプル出力:

```
repo1
```

```
repo2
repo3
```

次の例では、リポジトリ名に加えてアカウント ID を出力します。

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

サンプル出力:

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

--queryパラメータの詳細については、[CodeArtifact API リファレンス] の [リポジトリの一覧表示] を参照してください。

リポジトリの設定を表示または変更する

CodeArtifact のコンソールまたは AWS Command Line Interface (AWS CLI) を使用して、リポジトリを表示し、その詳細を更新することができます。

> **Note**
>
> リポジトリを作成したら、名前、関連付けられた AWS アカウント、またはドメインを変更することはできません。

トピック

- リポジトリの設定 (コンソール) を表示または変更する
- リポジトリ設定を表示または変更する (AWS CLI)

リポジトリの設定 (コンソール) を表示または変更する

CodeArtifact のコンソールを使用して、リポジトリの詳細を表示し、更新することができます。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで、[リポジトリ] をクリックし、表示または編集したいリポジトリの名前を選択します。
3. [詳細] を展開すると、以下のように表示されます。

   - リポジトリのドメイン。詳細を確認するには、ドメイン名を選択してください。
   - リポジトリのリソースポリシー。[リポジトリポリシーを適用する] をクリックして、ひとつ追加します。
   - リポジトリの Amazon リソースネーム (ARN)。
   - リポジトリに外部接続がある場合は、接続をクリックして詳細を確認できます。リポジトリに設定できる外部接続はひとつのみです。詳細については、「CodeArtifact リポジトリをパブリックリポジトリに接続する」を参照してください。
   - リポジトリにアップストリームリポジトリがある場合は、いずれかをクリックして、その詳細を参照することができます。リポジトリには、最大 10 個の直接のアップストリームリポジトリを設定できます。詳細については、「CodeArtifact でアップストリームリポジトリを操作する」を参照してください。

   > **Note**
   >
   > リポジトリは、外部接続またはアップストリームリポジトリを設定できますが、両方設定することはできません。

4. [パッケージ] をクリックすると、このリポジトリで使用可能なパッケージがすべて表示されます。パッケージをクリックして、詳細を確認してください。
5. [接続手順の表示] をクリックし、パッケージマネージャーを選択して、 CodeArtifact で設定する方法を学びます。
6. [リポジトリポリシーの適用] をクリックして、リソースポリシーをリポジトリに更新または追加します。詳細については、「リポジトリポリシー」を参照してください。
7. [編集] をクリックして、以下を追加または更新します。

   - リポジトリの説明。
   - リポジトリに関連付けられたタグ。
   - リポジトリに外部接続がある場合は、接続先の公開リポジトリを変更できます。あるいは、ひとつもしくはそれ以上の既存のリポジトリをアップストリームリポジトリとして追加できま

す。パッケージが要求されたときにCodeArtifactで優先させたい順番に並べます。詳細については、「アップストリームリポジトリの優先順位」を参照してください。

リポジトリ設定を表示または変更する (AWS CLI)

CodeArtifact でリポジトリの現在の設定を表示するには、describe-repositoryコマンドを使用します。

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

出力の例:

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

リポジトリのアップストリーム設定を変更する

アップストリームリポジトリを使用すると、パッケージマネージャークライアントは、単一の URL エンドポイントを使用して、複数のリポジトリに含まれるパッケージにアクセスすることができます。リポジトリのアップストリームの関係を追加または変更するには、update-repositoryコマンドを使用してください。

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

出力の例:

```
{
```

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

> **Note**
>
> アップストリームリポジトリを追加するには、アップストリームリポジトリで のAssociateWithDownstreamRepositoryアクションの権限が必要です。

リポジトリのアップストリームの関係を削除するには、空のリストを--upstreamsオプションの引数として使用します。

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

出力の例:

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

```
}
```

リポジトリポリシー

CodeArtifact は、リソースベースの権限を使用してアクセスをコントロールします。リソースベースの権限により、リポジトリにだれがアクセスでき、どのようなアクションを実行できるかを指定できます。デフォルトでは、リポジトリの所有者のみリポジトリにアクセスできます。他の IAM プリンシパルがリポジトリにアクセスできるようにするポリシードキュメントを適用することができます。

詳細については、[リソースベースのポリシー] および [アイデンティティベースおよびリソースベースのポリシー] を参照してください。

読み取りアクセスを許可するリソースポリシーを作成する

リソースポリシーは、JSON 形式のテキストファイルです。ファイルには、プリンシパル (アクター)、ひとつ以上のアクション、および エフェクト (`Allow`または`Deny`) を指定しなければいけません。例えば、次のリソースポリシーは、アカウントに、リポジトリからパッケージをダウンロードする`123456789012`許可を付与します。

JSON

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

ポリシーは、それが添付されているリポジトリに対する操作についてのみ評価されるため、リソースを指定する必要はありません。リソースが暗示されているので、`Resource`を`*`に設定できます。

パッケージマネージャーがこのリポジトリからパッケージをダウンロードするには、クロスアカウントアクセス用のドメインポリシーも作成する必要があります。ドメインポリシーは、少なくともcodeartifact:GetAuthorizationToken 権限をプリンシパルに付与する必要があります。クロスアカウントアクセス用のフルドメインポリシーの例については、「ドメインポリシーの例」を参照してください。

> **Note**
>
> codeartifact:ReadFromRepositoryアクションは、リポジトリリソースでのみ使用できます。パッケージの Amazon リソースネーム (ARN) を、リポジトリ内のパッケージのサブセットへの読み取りアクセスを許可するcodeartifact:ReadFromRepositoryアクションとするリソースとして指定することはできません。特定のプリンシパルは、リポジトリ内のすべてのパッケージを読み取れるか、あるいは、全く読み取れません。

リポジトリで指定されるアクションはReadFromRepositoryのみであるため、アカウント1234567890のユーザーとロールは、リポジトリからパッケージをダウンロードできます。ただし、他のアクション (パッケージ名やバージョンの一覧表示など) を実行することはできません。通常、ReadFromRepositoryに追加して以下のポリシーに権限を付与します。これは、リポジトリからパッケージをダウンロードするユーザーが、他の方法でもリポジトリと関わる必要があるためです。

JSON

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

ポリシーの設定

ポリシードキュメントを作成したら、put-repository-permissions-policyコマンドでリポジトリにアタッチします。

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

put-repository-permissions-policyをコールすると、権限を評価するときに、リポジトリのリソースポリシーは無視されます。これにより、ドメインの所有者がリポジトリから自分自身をロックアウトすることができなくなり、リソースポリシーを更新することを防ぐことができます。

> **Note**
>
> put-repository-permissions-policy を呼び出すときにリソースポリシーが無視されるため、リソースポリシーを使用してリポジトリのリソースポリシーを更新するアクセス許可を別の AWS アカウントに付与することはできません。

サンプル出力:

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

コマンドの出力には、リポジトリリソースの Amazon リソースネーム (ARN)、ポリシードキュメントの完全な内容、リビジョン識別子が含まれます。--policy-revisionオプションを使用して、リビジョン識別子をput-repository-permissions-policyに渡すことができます。これにより、別のライターによって設定された新しいバージョンではなく、ドキュメントの既知のリビジョンが確実に上書きされることが保証されます。

ポリシーを読み込む

get-repository-permissions-policyコマンドを使用して、ポリシードキュメントの既存のバージョンを読み込みます。読みやすいように出力をフォーマットするには、Pythonjson.toolモジュールと共に--outputおよび--query policy.documentを使用してください。

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

サンプル出力:

JSON

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

```
        }
    ]
}
```

ポリシーの削除

delete-repository-permissions-policyコマンドを使用して、リポジトリからポリシーを削除します。

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

出力のフォーマットは、get-repository-permissions-policyコマンドのフォーマットと同じです。

プリンシパルに読み取りアクセスを許可する

ポリシードキュメントでアカウントのルートユーザーをプリンシパルとして指定すると、そのアカウントのすべてのユーザーとロールへのアクセス権が付与されます。選択したユーザーまたはロールへのアクセスを制限するには、その ARN をPrincipalポリシーのセクションで使用してください。例えば、アカウント 123456789012 内の IAM ユーザー bob に読み取りアクセスを付与するには、以下を使用します。

JSON

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

```
    ]
}
```

パッケージへの書き込みアクセスを許可する

codeartifact:PublishPackageVersionアクションは、パッケージの新しいバージョンを公開するための権限をコントロールするために使用されます。このアクションで使用されるリソースは、パッケージである必要があります。CodeArtifact パッケージ ARN の形式は次のとおりです。

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

次の例は、ドメインmy_domainのmy_repoリポジトリでスコープ@parityと名前uiを持つ npm パッケージの ARN を示しています。

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

スコープを持たない npm パッケージの ARN には、ネームスペースフィールドに空の文字列になっています。例えば、スコープがないパッケージで、ドメインmy_domainのmy_repoリポジトリに名前reactのある ARN は以下のようになります。

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

以下のポリシーは、my_repoリポジトリ中の@parity/uiのバージョンを公開する権限をアカウント123456789012に付与します。

JSON

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

```
            },
            "Resource": "arn:aws:codeartifact:us-
east-1:111122223333:package/my_domain/my_repo/npm/parity/ui"
        }
    ]
}
```

> **Important**
>
> Maven および NuGet パッケージのバージョンを公開する権限を付与するには、次の権限を codeartifact:PublishPackageVersion に加えて追加します。
>
> 1. NuGet:codeartifact:ReadFromRepositoryおよびリポジトリリソースを指定します。
> 2. Mavencodeartifact:PutPackageMetadata

このポリシーでは、リソースの一部としてドメインとリポジトリを指定するため、そのリポジトリに添付されている場合にのみ公開が許可されます。

リポジトリへの書き込み権限の付与

ワイルドカードを使用して、リポジトリ内のすべてのパッケージに対して書き込む許可を付与できます。例えば、次のポリシーを使用して、my_repoリポジトリのすべてのパッケージに書き込む許可をアカウントに付与します。

JSON

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

```
            "Resource": "arn:aws:codeartifact:us-
east-1:111122223333:package/my_domain/my_repo/*"
        }
    ]
}
```

リポジトリポリシーとドメインポリシーの相互作用

CodeArtifact では、ドメインおよびリポジトリに対してリソースポリシーを設定できます。リソースポリシーはオプションです。各ドメインには 1 つのポリシーを設定し、ドメイン内の各リポジトリには独自のリポジトリポリシーを設定できます。ドメインポリシーとリポジトリポリシーの両方が存在する場合、CodeArtifact リポジトリへのリクエストが許可されるか拒否されるかを判断するときに、両方が評価されます。ドメインポリシーとリポジトリポリシーは、次のルールを使用して評価されます。

- ListDomains や ListRepositories などのアカウント単位のオペレーションを実行する場合、リソースポリシーは評価されません。
- DescribeDomain や ListRepositoriesInDomain などのドメイン単位のオペレーションを実行する場合、リポジトリポリシーは評価されません。
- PutDomainPermissionsPolicy を実行する場合、ドメインポリシーは評価されません。なお、このルールはロックアウトを防ぐためのものです。
- PutRepositoryPermissionsPolicy を実行する場合、ドメインポリシーは評価されますが、リポジトリポリシーは評価されません。
- いずれかのポリシーに明示的な Deny がある場合、他のポリシーで Allow が指定されていても、それより優先されます。
- 明示的な Allow は、いずれか 1 つのリソースポリシーにあれば十分です。リポジトリポリシーでアクションを省略していても、ドメインポリシーでそのアクションが許可されていれば、暗黙的な Deny にはなりません。
- リソースポリシーのどれもそのアクションを許可していない場合、結果は暗黙的な Deny となります。ただし、呼び出し元プリンシパルのアカウントがドメイン所有者またはリポジトリ管理者アカウントであり、さらにアイデンティティベースポリシーでそのアクションが許可されている場合を除きます。

リソースポリシーは、単一アカウントのシナリオ、つまりリポジトリへアクセスするために使用する呼び出し元アカウントが、ドメイン所有者であり、かつリポジトリ管理者アカウントと同じである

場合、アクセス許可を付与するために必須ではありません。呼び出し元アカウントがドメイン所有者またはリポジトリ管理者アカウントと同じではないクロスアカウントの場合、アクセスを付与するためにリソースポリシーが必須となります。CodeArtifact のクロスアカウントアクセスは、「IAM ユーザーガイド」の「クロスアカウントリクエストが許可されているかどうかを確認」に記載のクロスアカウントアクセスの一般的な IAM ルールに従います。

- ドメイン所有者アカウントのプリンシパルは、アイデンティティベースポリシーを通じてドメイン内の任意のリポジトリへのアクセスを許可される場合があります。なお、この場合、ドメインポリシーまたはリポジトリポリシーに明示的な Allow を記載する必要はありません。
- ドメイン所有者アカウントのプリンシパルは、ドメインポリシーまたはリポジトリポリシーを通じて任意のリポジトリへのアクセスを許可される場合があります。なお、この場合、アイデンティティベースポリシーに明示的な Allow を記載する必要はありません。
- リポジトリ管理者アカウントのプリンシパルは、アイデンティティベースポリシーを通じてリポジトリへのアクセスを許可される場合があります。なお、この場合、ドメインポリシーまたはリポジトリポリシーに明示的な Allow を記載する必要はありません。
- 別アカウントに属するプリンシパルがアクセスを許可されるのは、少なくとも 1 つのリソースポリシーおよび少なくとも 1 つのアイデンティティベースポリシーでそのアクションが許可されていて、さらにいずれのポリシーにも明示的な Deny が存在しない場合に限られます。

CodeArtifact でリポジトリをタグ付けする

タグは、AWS リソースに関連付けられるキーと値のペアです。CodeArtifact では、リポジトリにタグを適用することができます。CodeArtifact リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプの詳細については、リソースのタグ付けを参照してください。

リポジトリを作成するときに CLI を使用してタグを指定できます。コンソールまたは CLI を使用してタグを追加または削除し、リポジトリのタグの値を更新できます。リソースごとに最大 50 個のタグを追加できます。

トピック

- タグリポジトリ (CLI)
- タグリポジトリ (コンソール)

タグリポジトリ (CLI)

CLI を使用して、リポジトリタグを管理できます。

トピック

リポジトリにタグを追加する (CLI)

コンソールまたは AWS CLI を使用して、リポジトリにタグを付けることができます。

リポジトリを作成するときにタグを追加するには、「リポジトリの作成」を参照してください。

以下のステップでは、 AWS CLI の最新版をすでにインストールしているか、最新版に更新しているものとします。詳細については、「AWS Command Line Interfaceのインストール」を参照してください。

ターミナルまたはコマンドラインで、タグを追加するリポジトリの Amazon リソースネーム (ARN) および追加するタグのキーと値を指定して、tag-resource コマンドを実行します。

> **Note**
>
> リポジトリの ARN を取得するには、`describe-repository` コマンドを実行します。
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

リポジトリには複数のタグを追加できます。例えば、*my_domain* というドメイン内の *my_repo* という名前のリポジトリに 2 つのタグを付けます。*value1* のタグ値がある*key1* という名前のタグキーと、*value2* のタグ値がある*key2* という名前のタグキーです。

```
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
```

成功した場合は、コマンドの出力はありません。

リポジトリのタグを表示する (CLI)

を使用してリポジトリの AWS タグ AWS CLI を表示するには、次の手順に従います。タグが追加されていない場合、返されるリストは空になります。

ターミナルまたはコマンドラインで、list-tags-for-resource コマンドを実行します。

> **Note**
>
> リポジトリの ARN を取得するには、`describe-repository` コマンドを実行します :
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

例えば、*[my_domain]* というドメインの *[my_repo]* という名前のリポジトリのタグキーとタグ値のリストを`arn:aws:codeartifact:`*us-west-2*`:`*111122223333*`:repository/`*my_domain*`/`*my_repo* ARN 値で表示するには:

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

成功した場合、このコマンドは次のような情報を返します。

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

リポジトリ (CLI) のタグを編集する

を使用してリポジトリのタグ AWS CLI を編集するには、次の手順に従います。既存のキーの値を変更したり、別のキーを追加できます。

ターミナルまたはコマンドラインで、tag-resource コマンドを実行して、タグを更新するリポジトリの ARN を指定し、タグキーとタグ値を指定します。

> **Note**
>
> リポジトリの ARN を取得するには、describe-repository コマンドを実行します。
>
> ```
> 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
```

成功した場合は、コマンドの出力はありません。

リポジトリ (CLI) からタグを削除する

を使用してリポジトリからタグ AWS CLI を削除するには、次の手順に従います。

> **Note**
>
> リポジトリを削除すると、関連付けられたすべてのタグが削除されたリポジトリから解除されます。リポジトリを削除する前にタグを削除する必要はありません。

ターミナルまたはコマンドラインで、untag-resource コマンドを実行して、削除するタグのリポジトリの ARN と、削除するタグのタグキーを指定します。

> **Note**
>
> リポジトリの ARN を取得するには、describe-repository コマンドを実行します。
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

例えば、*key1* および *key2* という名前のタグキーのある、*my_domain* という名前のドメインの *my_repo* という名前のリポジトリで複数のタグを削除するには、次を行います。

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

成功した場合は、コマンドの出力はありません。タグを削除した後、リポジトリの残りのタグは、`list-tags-for-resource`コマンドを使用して表示することができます。

タグリポジトリ (コンソール)

コンソールまたは CLI を使用して、リソースのタグ付けをします。

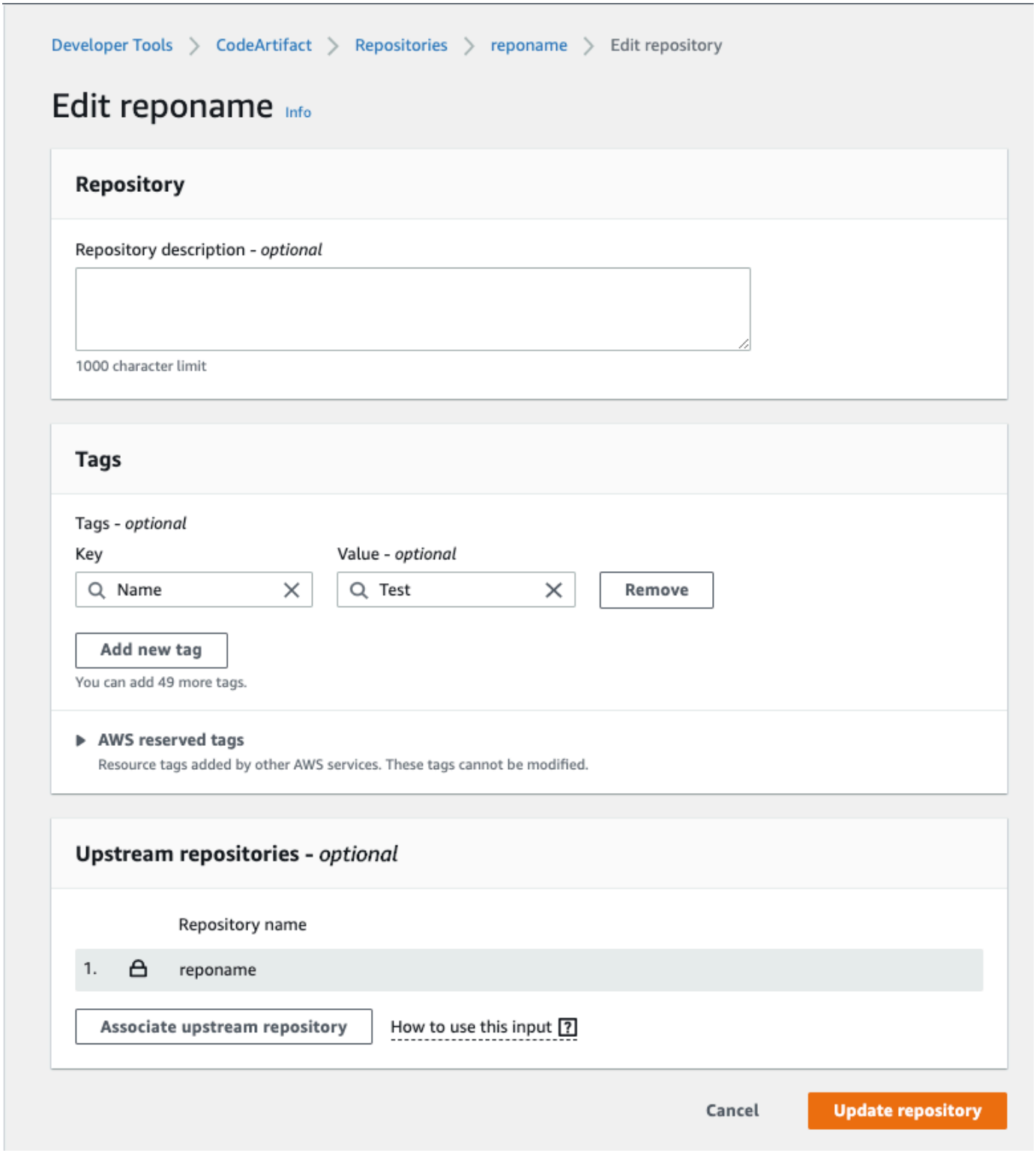
トピック

- リポジトリにタグを追加する (コンソール)
- リポジトリのタグを表示する (コンソール)
- リポジトリのタグを編集する (コンソール)
- リポジトリからタグを削除する (コンソール)

リポジトリにタグを追加する (コンソール)

コンソールを使用して既存のリポジトリにタグを追加します。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. [リポジトリ] ページで、タグを追加するリポジトリをクリックします。
3. [詳細] セクションを展開します。
4. [リポジトリタグ] で、リポジトリにタグがない場合は、[リポジトリタグの追加] をクリックします。リポジトリにタグがある場合は、[リポジトリタグの表示と編集] をクリックします。
5. [新しいタグを追加] をクリックします。
6. [キー] フィールドと [値] フィールドに、追加するタグごとにテキストを入力します。([値] フィールドはオプションです。) 例えば、[キー] では、「**Name**」と入力します。[値] には「**Test**」と入力します。



7. (オプション) [タグを追加] をクリックして行を追加し、さらにタグを入力します。
8. [リポジトリを更新]をクリックします。

リポジトリのタグを表示する (コンソール)

コンソールを使用して既存のパイプラインのタグを一覧表示します。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. [Repositories] (リポジトリ) で、タグを表示するリポジトリの名前をクリックします。
3. [詳細] のセクションを展開します。
4. [リポジトリタグ] で、[リポジトリタグの表示と編集] をクリックします。

> **Note**
>
> このリポジトリにタグが追加されていない場合、コンソールは [リポジトリタグの追加] を読み取ります。

リポジトリのタグを編集する (コンソール)

コンソールを使用してリポジトリに追加されたタグを編集します。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. [Repositories] (リポジトリ) のページで、タグを更新するリポジトリの名前をクリックします。
3. [詳細] セクションを展開します。
4. [リポジトリタグ] で、[リポジトリタグの表示と編集] をクリックします。

> **Note**
>
> このリポジトリにタグが追加されていない場合、コンソールは [リポジトリタグの追加] を読み取ります。

5. [キー] フィールドと [値] フィールドに、必要に応じて各フィールドの値を更新します。例えば、**Name**キーの場合は、[値] で、**Test** を **Prod** に変更します。
6. [リポジトリを更新]をクリックします。

リポジトリからタグを削除する (コンソール)

コンソールを使用してリポジトリからタグを削除できます。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. [Repositories] (リポジトリ) で、タグを削除するリポジトリの名前を選択します。
3. [詳細] セクションを展開します。
4. [リポジトリタグ] で、[リポジトリタグの表示と編集] をクリックします。

> **Note**
>
> このリポジトリにタグが追加されていない場合、コンソールは [リポジトリタグの追加] を読み取ります。

5. 削除する各タグのキーと値の横にある [Remove tag] をクリックします。
6. [リポジトリを更新]をクリックします。

CodeArtifact でアップストリームリポジトリを操作する

リポジトリは、アップストリームリポジトリとして他の AWS CodeArtifact リポジトリを持つことができます。これにより、パッケージマネージャークライアントは、単一のリポジトリエンドポイントを使用して、複数のリポジトリに含まれるパッケージにアクセスできます。

AWS マネジメントコンソール、、または SDK を使用して AWS CLI、1 つ以上のアップストリームリポジトリを AWS CodeArtifact リポジトリに追加できます。リポジトリをアップストリームリポジトリに関連付けるには、アップストリームリポジトリへの`AssociateWithDownstreamRepository`アクションの許可が必要です。詳細については、「アップストリームのリポジトリと一緒にリポジトリを作成」および「アップストリームリポジトリを追加または削除する」を参照してください。

アップストリームリポジトリにパブリックリポジトリへの外部接続がある場合、そこから下流にあるリポジトリは、そのパブリックリポジトリからパッケージを取得することができます。例えば、リポジトリ`my_repo`が `upstream`という名のアップストリームリポジトリを持ち、`upstream`がパブリック npm リポジトリへの外部接続を持つとします。この場合、`my_repo` に接続しているパッケージマネージャーは、npm パブリックリポジトリからパッケージを取得することができます。アップストリームリポジトリまたは外部接続からのパッケージのリクエストの詳細については、「アップストリームリポジトリを持つパッケージバージョンのリクエスト」または「外部接続からのパッケージのリクエスト」を参照してください。

トピック

- アップストリームリポジトリと外部接続の違いは何ですか。
- アップストリームリポジトリを追加または削除する
- CodeArtifact リポジトリをパブリックリポジトリに接続する
- アップストリームリポジトリを持つパッケージバージョンのリクエスト
- 外部接続からのパッケージのリクエスト
- アップストリームリポジトリの優先順位
- アップストリームリポジトリでの API 動作

アップストリームリポジトリと外部接続の違いは何ですか。

CodeArtifact では、アップストリームリポジトリと外部接続はほとんど同じように動作しますが、いくつかの重要な違いがあります。

1. CodeArtifact リポジトリには最大 10 個のアップストリームリポジトリを追加できます。追加できる外部接続は 1 つだけです。
2. アップストリームリポジトリまたは外部接続を追加するための API コールは別のものです。
3. アップストリームリポジトリからリクエストされたパッケージはそれらのリポジトリに保持されるため、パッケージの保持動作は少し異なります。詳細については、「中間リポジトリでのパッケージの保持」を参照してください。

アップストリームリポジトリを追加または削除する

CodeArtifact リポジトリにアップストリームリポジトリを追加または削除するには、以下のセクションの手順に従います。アップストリームリポジトリの作成方法の詳細については、「CodeArtifact でアップストリームリポジトリを操作する」を参照してください。

このガイドでは、その他のCodeArtifact リポジトリをアップストリームリポジトリとして設定する方法について説明します。npmjs.com、Nuget Gallery、Maven Central、PyPI などのパブリックリポジトリへの外部接続の設定については、「Add an external connection」を参照してください。

アップストリームリポジトリを追加または削除する (コンソール)

CodeArtifact コンソールを使用してリポジトリをアップストリームリポジトリとして追加するには、次の手順に従います。でアップストリームリポジトリを追加する方法については AWS CLI、「」を参照してくださいアップストリームリポジトリを追加または削除する (AWS CLI)。

CodeArtifact コンソールを使用してアップストリームリポジトリを追加するには

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで、Domains(ドメイン) をクリックし、リポジトリを含むドメイン名を選択します。
3. リポジトリの名前を選択します。
4. [Edit] を選択します。
5. [アップストリームリポジトリ] で、[アップストリームリポジトリの関連付け] を選択し、アップストリームリポジトリとして加えるリポジトリを追加します。リポジトリはアップストリームリポジトリと同じドメインにのみ追加できます。
6. [リポジトリを更新]をクリックします。

CodeArtifact コンソールを使用してアップストリームリポジトリを削除するには

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで、Domains(ドメイン) をクリックし、リポジトリを含むドメイン名を選択します。
3. リポジトリの名前を選択します。
4. [Edit] を選択します。
5. [アップストリームリポジトリ] で、削除するアップストリームリポジトリのリストエントリを探し、[関連付けの解除] を選択します。

> **Important**
>
> CodeArtifact リポジトリからアップストリームリポジトリを削除すると、パッケージマネージャーはアップストリームリポジトリのパッケージ、またはアップストリームリポジトリのいずれにもアクセスできなくなります。

6. [リポジトリを更新]をクリックします。

アップストリームリポジトリを追加または削除する (AWS CLI)

AWS Command Line Interface (AWS CLI) を使用して、CodeArtifact リポジトリのアップストリームリポジトリを追加または削除できます。これを行うには、`update-repository`コマンドを使用します。そして`--upstreams`パラメータを使用して、アップストリームリポジトリを指定します。

リポジトリはアップストリームリポジトリと同じドメインにのみ追加できます。

アップストリームリポジトリを追加するには (AWS CLI)

1. まだ設定していない場合は、「」の手順に従って CodeArtifact AWS CLI で をAWS CodeArtifact でのセットアップセットアップおよび設定します。
2. `--upstreams` フラグを含む `aws codeartifact update-repository` コマンドを使用して、アップストリームリポジトリを追加します。

> **Note**
>
> `update-repository` コマンドを呼び出すと、設定済みの既存のアップストリームリポジトリが `--upstreams` フラグを含むリポジトリのリストに置き換えられます。アップ

ストリームリポジトリを追加し既存のリポジトリも維持する場合は、既存のアップストリームリポジトリを呼び出しに含める必要があります。

次の例のコマンドは、my_domain という名前のドメインにある my_repo という名前のリポジトリに 2 つのアップストリームリポジトリを追加します。--upstreamsパラメータ内のアップストリームリポジトリの順序は、CodeArtifactがmy_repoリポジトリからパッケージをリクエストする際の検索優先順位を決定します。詳細については、「アップストリームリポジトリの優先順位」を参照してください。

npmjs.com や Maven Central などの外部パブリックリポジトリへの接続については、「CodeArtifact リポジトリをパブリックリポジトリに接続する」を参照してください。

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

アウトプットには、次のようなアップストリームリポジトリが含まれます。

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

アップストリームリポジトリを削除するには (AWS CLI)

1. まだ設定していない場合は、「」の手順に従って CodeArtifact AWS CLI で をAWS CodeArtifact でのセットアップセットアップおよび設定します。
2. CodeArtifact リポジトリからアップストリームリポジトリを削除するには、--upstreams フラグを含む update-repository コマンドを使用します。コマンドに提供されるリポジトリのリストは、CodeArtifact リポジトリの新しいアップストリームリポジトリセットになります。削除しない既存のアップストリームリポジトリを含め、削除するアップストリームリポジトリは省略します。

   あるリポジトリからすべてのアップストリームリポジトリを削除するには、update-repositoryコマンドを使用し、引数なしで--upstreamsを含めます。以下は、my_domainという名前のドメインに含まれるmy_repoという名前のリポジトリからアップストリームリポジトリを削除します。

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

   アウトプットは、upstreamsのリストが空であることを示しています。

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

CodeArtifact リポジトリをパブリックリポジトリに接続する

CodeArtifact リポジトリと、https://npmjs.com または [Maven Central リポジトリ] のような外部のパブリックリポジトリ間の外部接続を追加できます。その後、リポジトリにまだ存在しない CodeArtifact リポジトリからパッケージをリクエストすると、そのパッケージを外部接続から取得できます。これにより、アプリケーションで使用されるオープンソースの依存関係を使用できるようになります。

CodeArtifact で想定されている外部接続の使用方法は、特定のパブリックリポジトリへの外部接続を持つドメインごとに 1 つのリポジトリを持つことです。例えば、npmjs.com に接続する場合、ドメイン内の 1 つのリポジトリを npmjs.com への外部接続で設定し、他のすべてのリポジトリをそのアップストリームに設定します。こうすることで、npmjs.com から取得済みのパッケージを再度取得し保存することなく、すべてのリポジトリがパッケージを利用できます。

トピック

外部リポジトリに接続する (コンソール)

コンソールを使用して外部リポジトリへの接続を追加すると、以下のことが起こります。

1. 外部リポジトリの `-store` リポジトリが存在しない場合は、CodeArtifact ドメインにリポジトリが作成されます。これらの `-store` リポジトリは、リポジトリと外部リポジトリの中間リポジトリとして機能し、複数の外部リポジトリに接続できます。
2. 適切な `-store` リポジトリがリポジトリのアップストリームとして追加されます。

次のリストには、CodeArtifact 内の各 `-store` リポジトリと、それらが接続する外部リポジトリが含まれています。

1. `cargo-store` は、crates.io に接続されます。
2. `clojars-store` は、Clojars リポジトリに接続されます。
3. `commonsware-store` は、CommonsWare Android リポジトリに接続されます。
4. `google-android-store` は、Google Android に接続されます。

5. `gradle-plugins-store` は、Gradle プラグインに接続されます。
6. `maven-central-store` は、Maven Central リポジトリに接続されます。
7. `npm-store` は、npmjs.com に接続されます。
8. `nuget-store` は、nuget.org に接続されます。
9. `pypi-store` は、Python Packaging Authority に接続されます。
10. `rubygems-store` は、RubyGems.org に接続されます。

外部リポジトリに接続するには (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで、Domains(ドメイン) をクリックし、リポジトリを含むドメイン名を選択します。
3. リポジトリの名前を選択します。
4. [Edit] を選択します。
5. [アップストリームリポジトリ] で [アップストリームリポジトリを関連付け] を選択し、アップストリームとして接続されている適切な `-store` リポジトリを追加します。
6. [リポジトリを更新]をクリックします。

`-store` リポジトリをアップストリームリポジトリとして追加すると、CodeArtifact リポジトリに接続されたパッケージマネージャーは、それぞれの外部リポジトリからパッケージを取得できます。

外部リポジトリに接続する (CLI)

を使用して AWS CLI 、外部接続をリポジトリに直接追加することで、CodeArtifact リポジトリを外部リポジトリに接続できます。これにより、CodeArtifact リポジトリまたはそのダウンストリームリポジトリに接続しているユーザーは、設定された外部リポジトリからパッケージを取得できます。各 CodeArtifact リポジトリに設定できる外部接続は 1 つのみです。

特定のパブリックリポジトリへの外部接続については、ドメインごとに 1 つのリポジトリを用意することをお勧めします。他のリポジトリをパブリックリポジトリに接続するには、外部接続のリポジトリをアップストリームとして追加します。あなたやドメイン内の他のユーザーがコンソールですでに外部接続を設定している場合、そのドメインには、接続するパブリックリポジトリへの外部接続を備えた `-store` リポジトリがすでにある可能性があります。`-store` リポジトリとコンソールでの接続の詳細については、「外部リポジトリに接続する (コンソール)」を参照してください。

CodeArtifact リポジトリに外部接続を追加するには (CLI)

- `associate-external-connection` を使用して、外部接続を追加します。次の例では、リポジトリを npm パブリックレジストリ (npmjs.com) に接続します。サポートされている外部リポジトリのリストについては、「サポートされている外部接続リポジトリ」を参照してください。

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

出力例:

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

外部接続を追加した後に、外部接続を使用して外部リポジトリにパッケージをリクエストする方法については、「外部接続からのパッケージのリクエスト」を参照してください。

サポートされている外部接続リポジトリ

CodeArtifact は、次の公開リポジトリへの外部接続をサポートしています。CodeArtifact CLI を使用して外部接続を指定するには、`associate-external-connection`コマンドを実行するときの`--external-connection`パラメータの [名前] の列の値を使用します。

リポジトリタイプ	説明	名前
Maven	Clojars リポジトリ	`public:maven-clojars`
Maven	CommonsWare Android リポジトリ	`public:maven-commonsware`
Maven	Google Android リポジトリ	`public:maven-googleandroid`
Maven (メイヴン)	Gradle プラグインリポジトリ	`public:maven-gradleplugins`
Maven (メイヴン)	Maven Central	`public:maven-central`
npm	npm 公開レジストリ	`public:npmjs`
NuGet	NuGet ギャラリー	`public:nuget-org`
Python (パイソン)	Python パッケージインデックス	`public:pypi`
Ruby	RubyGems.org	`public:ruby-gems-org`
Rust	Crates.io	`public:crates-io`

外部接続を削除する (CLI)

で `associate-external-connection` コマンドを使用して追加された外部接続を削除するには AWS CLI、 を使用しますdisassociate-external-connection。

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

出力例:

```
{
```

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

アップストリームリポジトリを持つパッケージバージョンのリクエスト

クライアント (例: npm) が、複数のアップストリームリポジトリを持つ my_repo という名前の CodeArtifact リポジトリからパッケージバージョンをリクエストすると、以下のことが起こります。

- リクエストされたパッケージバージョンがmy_repoに含まれる場合、クライアントにリターンされます。
- リクエストされたパッケージバージョンがmy_repoに含まれない場合、CodeArtifactはmy_repoのアップストリームリポジトリでそれを検索します。パッケージバージョンが見つかると、そのバージョンへのリファレンスがmy_repoにコピーされます、そしてパッケージのバージョンがクライアントに返されます。
- my_repoにもそのアップストリームリポジトリにもパッケージバージョンがない場合、HTTP404Not Foundレスポンスがクライアントには返されます。

create-repositoryまたはupdate-repositoryコマンドを使ってアップストリームリポジトリを追加した場合、--upstreamsパラメータに渡された順番によって、パッケージバージョンがリクエストされた時の優先順位が決まります。パッケージバージョンがリクエストされた時、CodeArtifact に使用させたい順番で--upstreamsを使ってアップストリームリポジトリを指定します。詳細については、「アップストリームリポジトリの優先順位」を参照してください。

1 つのリポジトリに許可される直接アップストリームリポジトリの最大数は 10 です。直接アップストリームリポジトリは独自の直接アップストリームリポジトリを持つこともできるため、CodeArtifact は 10 を超えるリポジトリでパッケージバージョンを検索できます。パッケージバージョンがリクエストされた時にCodeArtifactが検索するリポジトリの最大数は25です。

アップストリームリポジトリからのパッケージの保持

リクエストされたパッケージバージョンが、アップストリームリポジトリで見つかった場合、そのバージョンのリファレンスは保持され、ダウンストリームリポジトリから常時利用できます。保持されたパッケージバージョンは、次のいずれの影響も受けません:

- アップストリームリポジトリの削除。
- アップストリームリポジトリのダウンストリームリポジトリからの切断。
- アップストリームリポジトリからのパッケージバージョンの削除。
- アップストリームリポジトリのパッケージバージョンの編集 (例えば、新しいアセットを追加するなど) 。

アップストリームの関係を通じてパッケージを取得する

CodeArtifactリポジトリに外部接続を持つリポジトリとのアップストリーム関係がある場合、アップストリームリポジトリにないパッケージのリクエストは外部リポジトリからコピーされます。例えば、次の設定を考えてみます。repo-Aという名前のリポジトリが、repo-Bという名のアップストリームリポジトリを持っています。repo-Bは https://npmjs.comに外部接続しています。

もしnpmがrepo-Aリポジトリを使用するよう設定されていた場合、npm installの実行によりhttps://npmjs.comからrepo-Bへのパッケージのコピーが開始されます。インストールされているバージョンもrepo-Aプルされます。次の例では、lodashがインストールされます。

```
$ 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
```

npm installの実行後、repo-A には最新バージョン (lodash 4.17.20)のみが含まれますが、その理由はそのバージョンがrepo-Aからnpmにより取得されたためです。

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

出力例:

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

repo-Bはhttps://npmjs.comに外部接続するため、https://npmjs.comからインポートされるすべてのパッケージバージョンはrepo-Bに保存されます。これらのパッケージバージョンは、repo-Bとのアップストリーム関係を持つ任意のダウンストリームリポジトリによって取得されている可能性があります。

repo-Bのコンテンツは、https://npmjs.comからインポートされたすべてのパッケージとパッケージバージョンを段階的に確認する方法を紹介します。例えば、段階的にインポートされたlodashパッケージのすべてのバージョンを見るためには、以下のようにlist-package-versionsを使用します。

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

出力例:

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

```
        {
            "version": "0.2.0",
            "revision": "REVISION-3-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.1",
            "revision": "REVISION-4-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.1.0",
            "revision": "REVISION-5-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    ],
    "nextToken": "eyJsaXN0UGFja2FnZVZlcnNpb25zVG9rZW4iOiIwLjIuMiJ9"
}
```

中間リポジトリでのパッケージの保持

CodeArtifactは、アップストリームリポジトリを連結できます。例えば、repo-Aはrepo-Bを、repo-Bはrepo-Cをアップストリームとして持つことができます。この設定により、repo-Bとrepo-Cにあるパッケージバージョンがrepo-Aから入手可能になります。

パッケージマネージャーがリポジトリrepo-Aに接続し、リポジトリrepo-Cからパッケージバージョンを取得する場合、そのパッケージバージョンはリポジトリrepo-Bに保持されません。パッケージバージョンは、最も下流のリポジトリにのみ保持されます、この例では、repo-Aにのみ保持されます。中間リポジトリには保持されません。これは、長いチェーンにも当てはまります。例えば、repo-A、repo-B、repo-C および repo-D という4 つのリポジトリがあり、repo-A に接続しているパッケージマネージャーが、repo-D からパッケージバージョンを取得した場合、そのパッケージバージョンは repo-A に保持されますが、repo-B または repo-C には保持されません。

パッケージ保持に関する動作は、外部リポジトリからパッケージバージョンをプルする場合と同様ですが、パッケージバージョンは常に外部接続を持つリポジトリに保持されます。例えば、repo-A のアップストリームとして repo-B があり、repo-B のアップストリームとして repo-C があり、repo-C は npmjs.com を外部接続として持っています。次の図を参照してください。

repo-A に接続しているパッケージマネージャーが、例えば lodash 4.17.20 というパッケージバージョンをリクエストし、そのパッケージバージョンが 3 つのリポジトリのいずれにも存在しない場合は、npmjs.com から取得されます。lodash 4.17.20 が取得されると、最も下流のリポジトリである repo-A と、npmjs.com への外部接続がある repo-C に保持されます。lodash 4.17.20 は中間リポジトリである repo-B には保持されません。

外部接続からのパッケージのリクエスト

以下のセクションでは、外部接続からパッケージをリクエストする方法と、パッケージをリクエストする際に期待される CodeArtifact の動作について説明します。

トピック

外部接続からパッケージを取得する

「CodeArtifact リポジトリをパブリックリポジトリに接続する」で説明されているように CodeArtifact リポジトリに追加した外部接続からパッケージを取得するには、リポジトリを使用するようにパッケージマネージャーを設定し、パッケージをインストールします。

> **Note**
>
> 以下の説明では npm を使用します。他のパッケージタイプの設定と使用方法を確認するには、「MavenでCodeArtifactを使う」、「NuGetでCodeArtifactを使う」、「PythonでCodeArtifactを使う」を参照してください。

外部接続からパッケージを取得するには

1. CodeArtifact リポジトリを使用してパッケージマネージャーの設定と認証を行います。npm で、次の `aws codeartifact login` コマンドを使用します。

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

2. パブリックリポジトリのパッケージをリクエストします。npm で、次の npm install コマンドを使用し、*lodash* をインストールするパッケージで置換します。

```
npm install lodash
```

3. パッケージが CodeArtifact リポジトリにコピーされたら、list-packagesおよびlist-package-versionsコマンドで表示します。

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

出力例:

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

list-package-versionsコマンドは、CodeArtifact リポジトリにコピーされたパッケージのすべてのバージョンを一覧表示します。

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

出力例:

```
{
    "defaultDisplayVersion: "1.2.5"
    "format": "npm",
    "package": "lodash",
    "namespace": null,
    "versions": [
```

```
        {
            "version": "1.2.5",
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    ]
}
```

外部接続のレイテンシー

外部接続を使用してパブリックリポジトリからパッケージを取得する場合、パッケージがパブリックリポジトリから取得されてから CodeArtifact リポジトリに保存されるまでに遅延が発生します。例えば、「外部接続からパッケージを取得する」で説明されているように npm パッケージ「lodash」のバージョン 1.2.5 をインストールしたとします。`npm install lodash` lodash コマンドは正常に完了したものの、パッケージバージョンが CodeArtifact リポジトリに表示されない場合があります。通常は、パッケージバージョンがリポジトリに表示されるまでに約 3 分かかりますが、それ以上かかる場合もあります。

このレイテンシーが原因で、パッケージバージョンを正常に取得できたとしても、CodeArtifact コンソール、または ListPackages API オペレーションおよび ListPackageVersions API オペレーションを呼び出す際に、リポジトリ内のバージョンがまだ表示されていないことがあります。CodeArtifact がパッケージバージョンを非同期的に永続化すると、コンソールと API リクエストを介して表示されます。

外部リポジトリが利用できない場合の CodeArtifact 動作

場合によっては、外部リポジトリが停止することがあります。つまり、CodeArtifact が外部リポジトリからパッケージを取得できないか、パッケージの取得が通常よりはるかに遅くなっているということです。これが発生すると、パッケージバージョンはすでに外部リポジトリからプルされています (例: npmjs.com)、CodeArtifact リポジトリに保存されているものは、CodeArtifact から引き続きダウンロードできます。ただし、CodeArtifact にまだ保存されていないパッケージは、そのリポジトリへの外部接続が設定されている場合でも、利用できない場合があります。例えば、CodeArtifact リポジトリには、これまでアプリケーションで使用していた npm パッケージのバージョン `lodash 4.17.19` が含まれているかも知れません。`4.17.20`にアップグレードする場合、通常 CodeArtifact はその新しいバージョンを npmjs.com から取得して、 CodeArtifact リポジトリに保存します。ただし、npmjs.com 停止が発生していると、この新しいバージョンは利用できません。唯一の回避策は、npmjs.com が回復してから後でもう一度やり直すことです。

外部リポジトリの停止は、CodeArtifact への新しいパッケージバージョンの公開にも影響する可能性があります。外部接続が設定されたリポジトリでは、CodeArtifact は外部リポジトリにすでに存在するパッケージバージョンの公開を許可しません。詳細については、「パッケージの概要」を参照してください。ただし、外部リポジトリの停止により、CodeArtifact が外部リポジトリに存在するパッケージとパッケージのバージョンに関する最新情報を持っていないことがまれに発生します。この場合、CodeArtifact は、通常拒否するパッケージバージョンの公開を許可することがあります。

新しいパッケージバージョンの入手可能性

npmjs.com などのパブリックリポジトリ内のパッケージバージョンを CodeArtifact リポジトリから使用できるようにするには、まずこれをリージョンパッケージのメタデータキャッシュに追加する必要があります。このキャッシュは各 AWS リージョンの CodeArtifact によって維持され、サポートされているパブリックリポジトリの内容を記述するメタデータが含まれています。このキャッシュのために、新しいパッケージバージョンがパブリックリポジトリに公開されてから、CodeArtifact から利用可能になるまでの間に、遅延が生じます。この遅延は、パッケージの種類によって異なります。

npm、Python、および Nuget のパッケージの場合、新しいパッケージバージョンが npmjs.com、pypi.org、または nuget.org に公開されてから、CodeArtifact リポジトリからインストールできるようになるまで、最大 30 分の遅延が発生することがあります。CodeArtifact は、キャッシュが最新であることを確認するために、これらの 2 つのリポジトリからのメタデータを自動的に同期します。

Maven パッケージの場合、新しいパッケージバージョンがパブリックリポジトリに公開されてから、CodeArtifact リポジトリからインストールできるようになるまで、最大 3 時間の遅延が発生することがあります。CodeArtifact は、最大で 3 時間に一回、パッケージの新しいバージョンをチェックします。3 時間のキャッシュライフタイムが終了した後に、指定されたパッケージ名に対する最初のリクエストは、そのパッケージのすべての新しいバージョンをリージョンキャッシュにインポートすることです。

一般的に使用されている Maven パッケージでは、通常 3 時間ごとに新しいバージョンがインポートされます。リクエストの頻度が高いため、キャッシュの有効期間が終了するとすぐにキャッシュが更新されることがよくあります。使用頻度の低いパッケージの場合、CodeArtifact リポジトリからパッケージのバージョンが要求されるまで、キャッシュは最新バージョンを持ちません。最初のリクエストでは、CodeArtifact から以前にインポートされたバージョンのみが使用可能になりますが、このリクエストによってキャッシュが更新されます。その後のリクエストにより、パッケージの新しいバージョンがキャッシュに追加され、ダウンロードできるようになります。

複数のアセットを含むパッケージバージョンのインポート

Maven パッケージと Python パッケージはどちらも、パッケージバージョンごとに複数のアセットを持つことができます。そのため、これらの形式のパッケージのインポートは、パッケージバージョンごとにアセットが 1 つしかない npm や NuGet パッケージよりも複雑になります。これらのパッケージタイプでどのアセットがインポートされるのか、また新しく追加されたアセットがどのように利用できるようになるのかについては、「アップストリームと外部接続からの Python パッケージのリクエスト」および「アップストリームと外部接続からの Maven パッケージのリクエスト」を参照してください。

アップストリームリポジトリの優先順位

1つ以上のアップストリームリポジトリを持つリポジトリからパッケージバージョンをリクエストする場合、その優先順位は、`create-repository`または`update-repository`コマンドを実行する際にリストアップされた順番に対応します。要求されたパッケージバージョンが見つかると、すべてのアップストリームリポジトリを検索する前に検索は停止します。詳細については、「アップストリームリポジトリを追加または削除する (AWS CLI)」を参照してください。

優先順位を表示するには `describe-repository`コマンドを実行します。

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

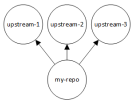
結果は次のとおりになります。これは、アップストリームリポジトリの優先順位が、まず`upstream-1`、次が `upstream-2`、最後が`upstream-3`であることを示しています。

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

簡単な優先順位の例

次の図では、my_repoリポジトリには 3 つのアップストリームリポジトリがあります。アップストリームリポジトリの優先順位は、upstream-1、upstream-2、upstream-3の順です。



my_repoでのパッケージバージョンのリクエストでは、それが見つかるか、またはHTTP404 Not Foundレスポンスがクライアントに返されるまで、次の順序でリポジトリを検索します。

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

パッケージバージョンが見つかると、すべてのアップストリームリポジトリでの検索が終了していなくても、検索は停止します。例えば、パッケージバージョンが upstream-1 で見つかった場合、検索は停止し、CodeArtifact が upstream-2 または upstream-3 を検索することはありません。

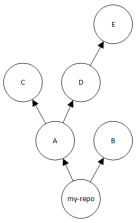
AWS CLI コマンドを使用して でパッケージバージョンlist-package-versionsを一覧表示する場合my_repo、 でのみ検索されますmy_repo。アップストリームリポジトリのパッケージバージョンはリストアップされません。

複雑な優先順位の例

アップストリームリポジトリに独自のアップストリームリポジトリがある場合、次のアップストリームリポジトリに移動する前に、同じロジックを使用してパッケージバージョンを検索します。例えば、my_repoリポジトリにはA とBの2つのアップストリームリポジトリがあるとします。リポジト

リAにアップストリームリポジトリがある場合、my_repoにあるパッケージバージョンのリクエストは、最初にmy_repo、次にAを検索し、それからAのアップストリームリポジトリを検索する、というように続きます。

次の図では、my_repoリポジトリにアップストリームリポジトリが含まれています。アップストリームリポジトリAには、アップストリームリポジトリが2つあり、Dにはアップストリームリポジトリが1つあります。図内の同じレベルにあるアップストリームリポジトリは、優先順位が左から右に表示されます。(リポジトリAはリポジトリBより優先順位が高く、リポジトリCはリポジトリDより優先順位が高い)。



この例では、my_repo でのパッケージバージョンのリクエストで、それが見つかるか、または、パッケージマネージャーが HTTP404 Not Found レスポンスをクライアントに返すまで、次の順序でリポジトリを検索します。

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

アップストリームリポジトリでの API 動作

アップストリームリポジトリに接続されているリポジトリで特定のCodeArtifact APIを呼び出すと、パッケージまたはパッケージバージョンがターゲットリポジトリかアップストリームリポジトリに格納されているかどうかによって、動作が異なる場合があります。これらのAPIの動作については、こちらを参照してください。

CodeArtifact APIの詳細については、CodeArtifact APIリファレンスを参照してください。

パッケージやパッケージバージョンを参照するほとんどのAPIは、指定されたパッケージバージョンがターゲットリポジトリに存在しない場合、ResourceNotFoundエラーを返します。これは、パッ

ケージまたはパッケージバージョンがアップストリームリポジトリに存在する場合にも当てはまります。実際には、これらのAPIを呼び出すとき、アップストリームリポジトリは無視されます。それらのAPIは次のとおりです:

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

この動作を確認するには、target-repoとupstream-repoの2つのリポジトリがあります:target-repoは空で、アップストリームリポジトリとして設定されているupstream-repoを持っています。upstream-repoにはnpmパッケージlodashが含まれます。

lodashパッケージを含むupstream-repo上にある DescribePackageVersionAPIを呼び出すと、次の結果が得られます。


```
{
    "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"
```

```
    }
```

空でありながらアップストリームとして設定されている upstream-repo を持つ target-repo 上の同じ API を呼び出すと、以下の結果が得られます。

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

CopyPackageVersionsAPIの動作は異なります。デフォルトでは、CopyPackageVersionsAPIは、ターゲットリポジトリに格納されているパッケージバージョンのみをコピーします。パッケージバージョンがアップストリームリポジトリに格納されているが、ターゲットリポジトリには保存されていない場合、コピーされません。アップストリームリポジトリにのみ格納されているパッケージのパッケージバージョンを含めるには、APIリクエストのincludeFromUpstreamの値をtrueにセットします。

CopyPackageVersionsAPIの詳細については、リポジトリ間でのパッケージのコピー を参照してください。

CodeArtifact でのパッケージの操作

以下のトピックでは、CodeArtifact CLI と API を使用してパッケージに対するアクションを実行する方法について説明します。

トピック

パッケージの概要

[パッケージ] とは、依存関係の解決とソフトウェアのインストールに必要なソフトウェアとメタデータのバンドルです。CodeArtifact には、パッケージはパッケージ名、`@types/node`に`@types`などの [ネームスペース] (オプション) など、パッケージバージョンのセット、およびパッケージレベルのメタデータ (npm タグなど) が含まれます。

目次

  - 公開時のレイテンシー
- パッケージバージョンのステータス
- パッケージ名、パッケージバージョン、アセット名の正規化

サポートされるパッケージ形式

AWS CodeArtifact は、Cargo、generic、Maven、npm、NuGet、PyPI、Ruby、Swift パッケージ形式をサポートしています。

メッセージの公開

サポートされるパッケージ形式はどれも新しいバージョンが、npm、twine、Maven、Gradle、nuget、およびdotnetなどのツールを使い、CodeArtifact リポジトリで公開することができます。

公開許可

AWS Identity and Access Management (IAM) ユーザーまたはロールには、送信先リポジトリに発行するアクセス許可が必要です。パッケージを公開するには、以下の権限が必要です。

- Cargo: `codeartifact:PublishPackageVersion`
- ジェネリック: `codeartifact:PublishPackageVersion`
- Maven: `codeartifact:PublishPackageVersion`そして`codeartifact:PutPackageMetadata`
- npm`codeartifact:PublishPackageVersion`
- NuGet: `codeartifact:PublishPackageVersion`そして`codeartifact:ReadFromRepository`
- Python: `codeartifact:PublishPackageVersion`
- Ruby: `codeartifact:PublishPackageVersion`
- Swift: `codeartifact:PublishPackageVersion`

上記の権限リストでは、IAM ポリシーで `codeartifact:PublishPackageVersion` 権限と `codeartifact:PutPackageMetadata` 権限の `package` リソースを指定する必要があります。また、`codeartifact:ReadFromRepository` 権限の `repository` リソースも指定する必要があります。

CodeArtifact の権限の詳細については、「AWS CodeArtifact アクセス許可リファレンス」を参照してください。

パッケージアセットの上書き

別のコンテンツで既に存在するパッケージアセットを再公開することはできません。例えば、JAR アセット `mypackage-1.0.jar`を持つ Maven パッケージをすでに公開したとします。古いアセットのチェックサムと新しいアセットのチェックサムが同じである場合のみ、そのアセットを再度公開できます。新しいコンテンツで同じアセットを再公開するには、最初にdelete-package-versionsコマンドを使ってパッケージバージョンを削除してください。異なるコンテンツで同じアセット名を再公開しようとすると、HTTP 409 の競合エラーが発生します。

複数のアセット (PyPI と Maven) をサポートするパッケージ形式の場合、必要な権限を持っていれば、いつでも既存のパッケージバージョンに異なる名前の新しいアセットを追加できます。ジェネリックパッケージの場合、パッケージバージョンが `Unfinished` 状態にある限り、新しいアセットを追加できます。npm はパッケージバージョンごとにひとつのアセットしかサポートしないため、公開されたパッケージバージョンを何らかの方法で変更するには、まず、delete-package-versionsを使用してそれを削除する必要があります。

すでに存在するアセットを再公開しようとした場合 (例えば、`mypackage-1.0.jar`)、公開されたアセットと新規アセットの内容が同じである場合、操作が冪等であるため、この操作は成功します。

プライベートパッケージと公開リポジトリ

CodeArtifact は、CodeArtifact リポジトリに保存されているパッケージを npmjs.com や Maven Central などの公開リポジトリに公開しません。CodeArtifact は、公開リポジトリから CodeArtifact リポジトリにパッケージをインポートしますが、パッケージを別の方向に移動することはありません。CodeArtifact リポジトリに発行するパッケージはプライベートのままで、アクセスを許可した AWS アカウント、ロール、およびユーザーのみが使用できます。

パッチが適用されたパッケージバージョンの公開

場合によっては、公開リポジトリで利用可能な変更パッケージバージョンを公開したい場合があります。例えば、`mydep 1.1`という重要なアプリケーション依存関係にバグが見つかった場合、パッケージベンダーがその変更をレビューして承認できるよりも早く修正する必要があるとしましょう。前述のように、CodeArtifact リポジトリで、公開リポジトリがアップストリームのリポジトリと外部接続を介して CodeArtifact リポジトリから到達可能な場合、CodeArtifact は`mydep 1.1`の公開を防ぎます。

この問題を回避するには、公開リポジトリに到達できない別の CodeArtifact リポジトリにパッケージバージョンを公開します。次に、`copy-package-versions` API を使用し、パッチが適用された `mydep 1.1` のバージョンを、使用先の CodeArtifact リポジトリにコピーします。

公開時のアセットサイズ制限

公開できるパッケージアセットの最大サイズは、「AWS CodeArtifact のクォータ」に示されているアセットファイルサイズの最大クォータによって制限されます。例えば、現在のアセットファイルサイズの最大クォータを超える Maven JAR または Python ホイールを公開することはできません。より大きなアセットを CodeArtifact に保存する場合は、クォータの引き上げをリクエストしてください。

アセットファイルサイズの最大クォータに加えて、npm パッケージの公開リクエストの最大サイズは 2 GB の制限があります。この制限はアセットファイルサイズの最大クォータとは関係なく、クォータを引き上げることで増やすことはできません。npm 公開リクエスト (HTTP PUT) では、パッケージメタデータと npm パッケージ tar アーカイブのコンテンツが一緒にバンドルされます。このため、公開できる npm パッケージの実際の最大サイズは、含まれるメタデータのサイズによって異なります。

> **Note**
>
> 公開される npm パッケージの最大サイズは 2 GB 未満に制限されています。

公開時のレイテンシー

CodeArtifact リポジトリに公開されたパッケージバージョンは、多くの場合 1 秒未満でダウンロードできます。例えば、`npm publish` で npm パッケージバージョンを CodeArtifact に公開した場合、そのバージョンは 1 秒以内に `npm install` コマンドで使用できるようになります。ただし、公開には一貫性がなく、時間がかかることがあります。公開後すぐにパッケージバージョンを使用する必要がある場合は、再試行を行ってダウンロードの信頼性を確保してください。例えば、パッケージバージョンを公開した後、公開したばかりのパッケージバージョンが最初にダウンロードできなかった場合は、ダウンロードを最大 3 回繰り返します。

> **Note**
>
> 通常、パブリックリポジトリからパッケージバージョンをインポートすると、公開よりも時間がかかります。詳細については、「外部接続のレイテンシー」を参照してください。

パッケージバージョンのステータス

CodeArtifact のすべてのパッケージバージョンには、パッケージバージョンの現在の状態と使用可用性を示すステータスがあります。 AWS CLI と SDK で、パッケージバージョンのステータスを変更することができます。詳細については、「パッケージバージョンのステータスの更新」を参照してください。

パッケージバージョンのステータスに設定できる値は以下の通りです。

- [公開] - パッケージバージョンは正常に公開され、パッケージマネージャーを使用してリクエストできます。パッケージバージョンは、`npm view <package-name> versions` の出力など、パッケージマネージャーによって返されるパッケージバージョンリストに含まれます。パッケージバージョンのすべてのアセットは、リポジトリから入手できます。
- [未完了] - クライアントはパッケージバージョンの 1 つ以上のアセットをアップロードしましたが、それを `Published` 状態に移行してアップロードを完了することができませんでした。現在 `Unfinished` のステータスになり得るのは、ジェネリックパッケージバージョンと Maven パッケージバージョンのみです。これは、クライアントがパッケージバージョンの 1 つ以上のアセットをアップロードしたものの、そのバージョンを含むパッケージの `maven-metadata.xml` ファイルを公開しなかった場合に発生します。Maven パッケージバージョンが [未完了] の場合、`mvn` や `gradle` などのクライアントに返されるバージョンリストには含まれないため、ビルドの一部として使用することはできません。PublishPackageVersion API を呼び出すときに `unfinished` フラグを指定することで、ジェネリックパッケージを意図的に `Unfinished` 状態に保つことができます。ジェネリックパッケージを `Published` 状態に変更するには、`unfinished` フラグを省略するか、UpdatePackageVersionsStatus API を呼び出します。
- [一覧表示されていない] - パッケージバージョンのアセットはリポジトリからダウンロードできますが、パッケージマネージャーによって返されるパッケージバージョンのリストには含まれません。例えば、npm パッケージの場合、`npm view <package-name> versions` の出力にパッケージバージョンは含まれません。つまり、npm の依存関係解決論理は、使用可能なバージョンのリストに表示されないため、パッケージのバージョンを選択しません。ただし、[一覧表示されていない] パッケージバージョンがすでにすべての`npm package-lock.json`ファイルで参照されていれば、`npm ci`の実行時などに、ダウンロードしてインストールできます。
- [アーカイブ済み] - パッケージバージョンのアセットはこれ以降ダウンロードできません。パッケージバージョンは、パッケージマネージャーによって返されるバージョンのリストには含まれません。アセットが使用できないため、クライアントによるパッケージバージョンの使用はブロックされます。アプリケーションのビルドが、[アーカイブ済み] に更新されたバージョンに依存している場合、パッケージのバージョンがローカルにキャッシュされていないと仮定されるため、構築は中断されます。[アーカイブ済み] パッケージバージョンがまだリポジトリに存在するため、パッ

ケージマネージャーまたはビルドツールを使用して再度公開することはできません。しかし、パッケージバージョンのステータスを [パッケージのバージョンステータスのアップロード API] で [一覧表示されていない] または [公開] にすることは可能です。

- [開放済み] - パッケージバージョンはリストに表示されず、アセットをリポジトリからダウンロードできません。[開放済み] と [アーカイブ済み] ステータスの主な違いは、[開放済み] のステータス場合、パッケージバージョンのアセットは CodeArtifact によって完全に削除されることです。このため、パッケージバージョンを[開放済み] から[アーカイブ済み]、[一覧表示されていない]、または [公開] に移動することはできません。アセットが削除されているため、パッケージバージョンはこれ以降使用できません。パッケージバージョンが [開放済み] としてマークされた後は、パッケージアセットの保存に関する費用は、これ以降請求されなくなります。

`--status` パラメータなしで list-package-versions を呼び出すと、すべてのステータスのパッケージバージョンがデフォルトで返されます。

上記の状態とは別に、パッケージバージョンを DeletePackageVersions API で削除することもできます。削除後、パッケージバージョンはリポジトリ内に存在しなくなり、パッケージマネージャーまたはビルドツールを使用して、そのパッケージバージョンを自由に再公開できます。パッケージバージョンが [開放済み] としてマークされた後は、パッケージアセットの保存に関する費用は、これ以降請求されなくなります。

パッケージ名、パッケージバージョン、アセット名の正規化

CodeArtifact は、パッケージ名、パッケージバージョン、およびアセット名を保存する前に正規化します。つまり、CodeArtifact の名前またはバージョンは、パッケージが公開されたときに提供された名前やバージョンとは異なる場合があります。CodeArtifact でパッケージタイプごとに名前とバージョンを正規化する方法については、次のドキュメントを参照してください。

- Python パッケージ名の正規化
- NuGet パッケージ名、バージョン、アセット名の正規化

CodeArtifact は、他のパッケージ形式では正規化を行いません。

パッケージ名を一覧表示する

CodeArtifact で`list-packages`コマンドを使用し、リポジトリ内のすべてのパッケージ名のリストを取得します。このコマンドは、パッケージ名のみを返し、バージョンは返しません。

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

サンプル出力:

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

npm パッケージ名を一覧表示する

npm パッケージの名前のみを一覧表示するには、`--format`オプションの値を`npm`に設定します。

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

ネームスペース (npm [範囲]) 内の npm パッケージを一覧表示するには、`--namespace`および`--format`オプションを使用してください。

> **Important**
>
> `--namespace`オプションの値には`@`の先頭を含めないでください。ネームスペース`@types`を検索するには、値を *`[###]`* に設定します。

> **Note**
>
> --namespace オプションは名前空間のプレフィックスでフィルタリングします。--namespace オプションに渡された値で始まるスコープの npm パッケージは、すべて list-packages レスポンスで返されます。

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

サンプル出力:

```
{
    "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"

        }
    ]
}
```

Maven パッケージ名を一覧表示する

Maven パッケージの名前のみを一覧表示するには、--format オプションの値を maven に変更します。また、--namespaceオプションで Maven グループ ID を指定する必要があります。

> **Note**
>
> `--namespace` オプションは名前空間のプレフィックスでフィルタリングします。`--namespace` オプションに渡された値で始まるスコープの npm パッケージは、すべて `list-packages` レスポンスで返されます。

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

サンプル出力:

```
{
    "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"

        }
    ]
}
```

Python パッケージ名を一覧表示する

Python パッケージの名前のみを一覧表示するには、`--format`オプションの値を`pypi`に設定します。

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

パッケージ名のプレフィックスによるフィルタリング

指定した文字列で始まるパッケージを返すには、--package-prefixオプションを使用できます。

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

サンプル出力:

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

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

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

        }
    ]
}
```

サポートされている検索オプションの組み合わせ

--format、--namespace、および--package-prefixの任意の組み合わせのオプション (ただし、--namespace単独では使用できません) が使用できます。スコープが @types で始まるすべての npm パッケージを検索するには、--format オプションを指定する必要があります。--namespaceのみを使用すると、エラーが発生します。

三つのオプションのいずれを使用しないことも、list-packagesによってもサポートされていて、そうすると、リポジトリ内に存在するすべてのフォーマットのパッケージを表示します。

出力形式

すべての AWS CLI コマンドで使用できるパラメータを使用して、list-packagesレスポンスをコンパクトで読みやすくすることができます。--queryパラメータを使用して、返される各パッケージバージョンの形式を指定します。--outputパラメータを使用して、レスポンスをプレーンテキストとしてフォーマットします。

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

サンプル出力:

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

詳細については、[AWS Command Line Interface ユーザーガイド]の [AWS CLIからのコマンド出力のコントロール] を参照してください。

デフォルトおよびその他のオプション

デフォルトでは、list-packagesによって返される結果の最大数は100に設定されています。この結果制限は、--max-resultsオプションを使って変更できます。

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

--max-results の許容される最大値は 1,000 です。1,000 を超えるパッケージを持つリポジトリ内のパッケージを一覧表示できるように許可するために、list-packagesがレスポンスのnextTokenフィールドを使ってページ割りのサポートをします。リポジトリ内のパッケージ数

が--max-resultsの値より大きい場合は、nextTokenの値をlist-packagesの別の呼び出しに渡して、結果の次のページを取得できます。

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo \
  --next-token r00ABXNyAEdjb...
```

パッケージバージョンを一覧表示する

in AWS CodeArtifact list-package-versions コマンドを使用して、リポジトリ内のパッケージ名のすべてのバージョンのリストを取得します。

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

サンプル出力:

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


--statusパラメータをlist-package-versionsコールに追加して、パッケージバージョンのステータスに基づいて結果をフィルタリングすることができます。パッケージバージョンのステータスの詳細については、「パッケージバージョンのステータス」を参照してください。

list-package-versionsおよび--max-resultsパラメータを使用し、--next-tokenからの応答をページ分割できます。--max-resultsの場合、1 ~ 1000 の整数を指定して、単一ページに返される結果の数を指定できます。デフォルトは 50 に設定されています。後続ページを返すには、list-package-versionsをもう一度実行し、前のコマンド出力で受信したnextTokenの値を--next-tokenにパスします。--next-tokenオプションが使用されないと、常に結果の最初のページが返されます。

list-package-versionsコマンドはアップストリームリポジトリのパッケージバージョンを一覧表示しません。ただし、パッケージバージョンのリクエスト中に、リポジトリにコピーされたアップストリームリポジトリ内のパッケージバージョンへの参照が一覧表示されます。詳細については、「CodeArtifact でアップストリームリポジトリを操作する」を参照してください。

npm パッケージバージョンを一覧表示する

npm パッケージのすべてのパッケージバージョンを一覧表示するには、--format オプションの値を npm に設定します。

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

特定の名前空間 (npm scope) の npm パッケージバージョンを一覧表示するには、--namespace オプションを使用します。--namespaceオプションの値には@の先頭を含めないでください。ネームスペース@typesを検索するには、値を *[###]* に設定します。

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

Maven パッケージバージョンを一覧表示する

Maven パッケージのすべてのパッケージバージョンを一覧表示するには、--format オプションの値を maven に設定します。また、--namespaceオプションで Maven グループ ID を指定する必要があります。

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

バージョンを並べ替える

list-package-versionsは、公開時間に基づいて降順にソートされたバージョンを出力できます(最近公開されたバージョンが最初に一覧表示されます)。次のように、PUBLISHED_TIMEの値の--sort-byパラメータを指定します。

```
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
```

サンプル出力:

```
{

  "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...."
}
```

デフォルトの表示バージョン

パッケージ形式によって defaultDisplayVersion の戻り値が異なります。

- ジェネリック、Maven、PyPI パッケージの場合、これは最近公開されたパッケージバージョンです。
- npm パッケージの場合、これはlatestタグによって参照されるバージョンです。そのlatestタグが設定されていない場合は、最近公開されたパッケージバージョンとなります。

出力形式

すべての AWS CLI コマンドで使用できるパラメータを使用して、list-package-versionsレスポンスをコンパクトで読みやすくすることができます。--queryパラメータを使用して、返される各パッケージバージョンの形式を指定します。--output パラメータを使用して、レスポンスをプレーンテキストとしてフォーマットします。

```
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]'
```

サンプル出力:

```
0.1.1
0.1.2
0.1.0
3.0.0
```

詳細については、[AWS Command Line Interface ユーザーガイド] の [AWS CLIからのコマンド出力のコントロール] を参照してください。

パッケージバージョンのアセットを一覧表示する

[アセット] とは、 npm.tgzファイルや Maven POM 、または JAR ファイルのように CodeArtifact に保存されていて、パッケージバージョンに関連付けられている個々のファイルのことを指しま

す。list-package-version-assetsコマンドを使用して、各パッケージバージョンのアセットを一覧表示します。

list-package-version-assets コマンドを実行して、 AWS アカウントと現在の AWS リージョンの各アセットに関する次の情報を返します。

- 名前。
- サイズ (バイト単位)。
- チェックサムの検証に使用されるハッシュ値のセット。

例えば、Python パッケージflatten-json、バージョン0.1.7のアセットを一覧表示するには、次のコマンドを使用します。

```
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
```

出力は以下のようになります。

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

npm パッケージのアセットを一覧表示する

npm パッケージには、常にpackage.tgzという名前を持つ単一のアセットがあります。。スコープ指定された npm パッケージのアセットを一覧表示するには、--namespace オプションにスコープを含めます。

```
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
```

Maven パッケージのアセットを一覧表示する

Maven パッケージのアセットを一覧表示するには、--namespace オプションにパッケージ名前空間を含めます。Maven パッケージのアセットを一覧表示するには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
```

パッケージバージョンアセットのダウンロード

[アセット] とは、 npm.tgzファイルや Maven POM 、または JAR ファイルのように CodeArtifact に保存されていて、パッケージバージョンに関連付けられている個々のファイルのことを指します。パッケージアセットは、get-package-version-assets commandを使用してダウンロードする

ことができます。これにより、npmまたはpipのようなパッケージマネージャークライアントを使用せずにアセットを取得することができます。アセットをダウンロードするには、list-package-version-assets コマンドを使用して入手できるアセットの名前を提供する必要があります。詳細については、パッケージバージョンのアセットを一覧表示するを参照してください。アセットは、指定したファイル名でローカルストレージにダウンロードされます。

次の例では、Maven パッケージの *[com.google.guava:guava]* のバージョン *<27.1-JRE>* から *<guava-27.1-jre.jar>*のアセットをダウンロードします。

```
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
```

この例では、ファイル名は上記のコマンドの最後の引数によって、*guava-27.1-jre.jar* と指定され、ダウンロードしたアセットは *guava-27.1-jre.jar* と名前が付けられます。

コマンドの出力は次のようになります。

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

> **Note**
>
> スコープ指定された npm パッケージからアセットをダウンロードするには、--namespace オプションにスコープを含めます。--namespace を使用するときは、@ 記号を省略する必要があります。例えば、スコープが @types の場合は、--namespace types を使用します。

get-package-version-assetを使用してアセットをダウンロードするには、パッケージリソースに対するcodeartifact:GetPackageVersionAsset許可が必要となります。リソースベースの権限ポリシーの詳細については、[AWS Identity and Access Management ユーザーガイド] の [リソースベースのポリシー] を参照してください。

リポジトリ間でのパッケージのコピー

CodeArtifact で、あるリポジトリから別のリポジトリにパッケージバージョンをコピーできます。これは、パッケージプロモーションワークフローや、チームやプロジェクト間でパッケージバージョンを共有するなどのシナリオに役立ちます。パッケージバージョンをコピーするには、ソースリポジトリと送信先リポジトリが同じドメインにある必要があります。

パッケージをコピーするのに必要な IAM 権限

CodeArtifact でパッケージバージョンをコピーするには、呼び出し元のユーザーが必要な IAM 権限を持ち、ソースリポジトリと送信先のリポジトリに添付されたリソースベースのポリシーが必要な権限を持っていることが必要です。リソースベースの権限ポリシーと CodeArtifact リポジトリの詳細については、リポジトリポリシーを参照してください。

`copy-package-versions`を呼び出しているユーザーには、ソースリポジトリに関する`ReadFromRepository`許可および`CopyPackageVersions`送信先リポジトリに関する許可が必要です。

ソースリポジトリには、`ReadFromRepository`許可が必要で、送信先リポジトリには IAM アカウントまたはユーザーによるパッケージのコピーに割り当てられた`CopyPackageVersions`許可が必要です。次のポリシーは、`put-repository-permissions-policy`コマンドでソースリポジトリまたは送信先リポジトリに追加されるリポジトリポリシーの例です。*`[111122223333]`* をコール元のアカウントの ID `copy-package-versions`に置換する。

> **Note**
>
> 現在のリポジトリポリシーが存在する場合、`put-repository-permissions-policy`をコールすると、そのポリシーは置換されます。`get-repository-permissions-policy`コマンドを使用して、ポリシーが存在するかどうかを確認することができます。詳細については、ポリシーを読み込むを参照してください。ポリシーが存在する場合は、そのポリシーを置換する代わりにこれらの権限をポリシーに追加することをお勧めします。

ソースリポジトリの権限ポリシーの例

JSON

```
{
```

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

送信先リポジトリの権限ポリシーの例

JSON

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

パッケージバージョンをコピーする

CodeArtifact のcopy-package-versionsコマンドを使用して、ひとつまたはそれ以上のパッケージバージョンをソースリポジトリから同じドメイン内のコピー先リポジトリにコピーすることが

できます。次の例では、my-packageという名前の npm パッケージのバージョン 6.0.2 と 4.0.0 をmy_repoリポジトリからrepo-2リポジトリへコピーします。

```
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
```

一度のオペレーションで、同じパッケージ名の複数のバージョンをコピーできます。異なるパッケージ名のバージョンをコピーするには、それぞれについてcopy-package-versionsを呼び出す必要があります。

前のコマンドでは、両方のバージョンが正常にコピーできると仮定して、次の出力が生成されます。

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

アップストリームリポジトリからパッケージをコピーする

通常、copy-package-versionsはコピーするバージョンの--source-repositoryのオプションで指定されたリポジトリ内だけを検索します。ただ、--include-from-upstreamオプションを使用してソースリポジトリとそのアップストリームリポジトリの両方からバージョンをコピーすることはできます。CodeArtifact SDK を使用する場合は、CopyPackageVersionsAPIとincludeFromUpstreamパラメータを true に設定します。詳細については、「CodeArtifact でアップストリームリポジトリを操作する」を参照してください。

スコープ指定された npm パッケージをコピーする

スコープ内の npm パッケージバージョンをコピーするには、--namespaceオプションを使用して、スコープを指定します。例えば、パッケージ@types/reactをコピーするには、--namespace typesを使用します。--namespaceを使用するときは、@記号を省略する必要があります。

```
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
```

Maven パッケージのバージョンをコピーする

リポジトリ間で Maven パッケージのバージョンをコピーするには、Maven グループ ID を --namespace オプションで、また Maven ArtifactID を --name オプションで渡して、コピーするパッケージを指定します。例えば、com.google.guava:guava の単一のバージョンをコピーするには:

```
 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
```

パッケージのバージョンが正常にコピーされると、出力は以下のようになります。

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

ソースリポジトリに存在しないバージョン

ソースリポジトリに存在しないバージョンを指定すると、コピーは失敗します。ソースリポジトリにいくつかのバージョンが存在したり、存在しないバージョンがある場合、すべてのバージョンのコ

ピーが失敗します。次の例では、array-unique npm パッケージのバージョン 0.2.0 はソースリポジトリに存在しますが、バージョン 5.6.7 は存在しません。

```
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
```

出力は以下のようになります。

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

SKIPPEDエラーコードは、別のバージョンをコピーできなかったために、このバージョンがコピー先のリポジトリにコピーされなかったことを示すために使用されます。

送信先リポジトリにすでに存在するバージョン

パッケージバージョンがすでに存在するリポジトリにコピーされると、CodeArtifact は二つのリポジトリ内のパッケージアセットとパッケージバージョンレベルのメタデータを比較します。

パッケージバージョンのアセットとメタデータがソースリポジトリと送信先リポジトリで同一である場合、コピーは実行されませんが、オペレーションは成功したと見なされます。つまり、copy-package-versions は冪等性です。この場合、ソースリポジトリと送信先のリポジトリの両方にすでに存在していたバージョンは、copy-package-versionsの出力には表示されません。

次の例では、npm パッケージの二つのバージョンを示します。array-uniqueはソースリポジトリrepo-1に存在します。バージョン 0.2.1 は送信先のリポジトリにも存在しますdest-repoバージョン 0.2.0 はそうではありません。

```
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
```

出力は以下のようになります。

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

バージョン 0.2.0 がsuccessfulVersionsに一覧表示されているのは、ソースから送信先リポジトリに正常にコピーされたためです。バージョン 0.2.1 は、宛先リポジトリにすでに存在していたため、出力には表示されません。

パッケージバージョンのアセットまたはメタデータがソースリポジトリと送信先のリポジトリで異なる場合、コピー操作は失敗します。--allow-overwriteパラメータを使用して、強制的に上書きすることができます。

ソースリポジトリにいくつかのバージョンが存在したり、存在しないバージョンがある場合、すべてのバージョンのコピーが失敗します。次の例では、array-uniquenpm package パッケージのバージョン 0.3.2 は、ソースと送信先のリポジトリの両方に存在しますが、パッケージバージョンの内容は異なります。バージョン 0.2.1 はソースリポジトリに存在しますが、送信先には存在しません。

```
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
```

出力は以下のようになります。

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

バージョン 0.2.1 がSKIPPEDとしてマークされている理由は、コピー先のリポジトリにコピーされていないからです。バージョン 0.3.2 のコピーが送信先リポジトリにすでに存在していたが、ソースリポジトリと送信先のリポジトリでは同一ではないため、コピーは実行されませんでした。

パッケージバージョンリビジョンの指定

パッケージバージョンリビジョンは、パッケージバージョンの特定のアセットとメタデータのセットを指定する文字列です。パッケージバージョンリビジョンを指定して、特定の状態にあるパッケージバージョンをコピーできます。パッケージバージョンのリビジョンを指定するには、--version-revisionsパラメータを使用して、1 つ以上のカンマ区切りパッケージバージョンとパッケージバージョンリビジョンのペアをcopy-package-versionsコマンドに渡します。

> **Note**
>
> --versions または --version-revisions パラメータをcopy-package-versionsで指定する必要があります。両方を指定することはできません。

次の例では、パッケージのバージョン 0.3.2 がパッケージバージョンリビジョンREVISION-1-SAMPLE-6C81EFF7DA55CCのソースリポジトリに存在する場合、そのバージョン 0.3.2 my-package のみをコピーします。

```
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
```

次の例では、パッケージの二つのバージョンをコピーしています :my-package、0.3.2と0.3.13。このコピーは、my-packageのソースリポジトリバージョン 0.3.2 にリビジョンREVISION-1-

SAMPLE-6C81EFF7DA55CCがあり、バージョン0.3.13にはリビジョンREVISION-2-SAMPLE-55C752BEE772FCがある場合にのみ成功します。

```
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
```

パッケージバージョンのリビジョンを見つけるには、describe-package-versionまたはlist-package-versionsコマンドを使用してください。

詳細については、パッケージバージョンリビジョンおよび [CodeArtifact API リファレンス] の [パッケージバージョンのコピー] を参照してください。

npm パッケージをコピーする

npm パッケージでのcopy-package-versions動作の詳細については、[npm タグ と CopyPackageVersions API] を参照してください。

パッケージまたはパッケージバージョンを削除する

ひとつ以上のパッケージバージョンを一度に削除するには、delete-package-versionsコマンドを使用できます。関連するすべてのバージョンと設定を含め、リポジトリからパッケージを完全に削除するには、delete-package コマンドを使用します。パッケージは、パッケージバージョンがなくてもリポジトリに存在できます。これは、delete-package-versions コマンドを使用してすべてのバージョンを削除した場合や、put-package-origin-configuration API オペレーションを使用してバージョンなしでパッケージを作成した場合に発生する可能性があります (「」を参照パッケージオリジンコントロールの編集)。

トピック

- パッケージの削除 (AWS CLI)
- パッケージの削除 (コンソール)
- パッケージバージョンの削除 (AWS CLI)
- パッケージバージョンの削除 (コンソール)
- npm パッケージまたはパッケージバージョンの削除
- Maven パッケージまたはパッケージバージョンの削除

- パッケージまたはパッケージバージョンの削除におけるベストプラクティス

パッケージの削除 (AWS CLI)

delete-package コマンドを使用すると、パッケージとそのすべてのパッケージバージョンと設定を含むパッケージを削除できます。次の例では、my_domain ドメインの my-package リポジトリにある my_repo という名前の PyPI パッケージを削除します。

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

サンプル出力:

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

パッケージが削除されたことを確認するには、同じパッケージ名に対して describe-package を実行します。

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

パッケージの削除 (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで、[Repositories] を選択します。

3. パッケージを削除する [リポジトリ] を選択します。
4. 削除する [パッケージ] を選択します。
5. [パッケージを削除] を選択します。

パッケージバージョンの削除 (AWS CLI)

ひとつ以上のパッケージバージョンを一度に削除するには、delete-package-versionsコマンドを使用できます。以下の例では、my_domain ドメインの my_repo にある my-package という名前の PyPI パッケージバージョン 4.0.0、4.0.1、および 5.0.0 を削除します。

```
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
```

サンプル出力:

```
{
   "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": {}
}
```

バージョンが削除されたことを確認するには、list-package-versionsを同じパッケージ名で実行してください:

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

パッケージバージョンの削除 (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで、[Repositories] を選択します。
3. パッケージバージョンを削除する [リポジトリ] を選択します。
4. バージョンを削除する [パッケージ] を選択します。
5. 削除する [パッケージバージョン] を選択します。
6. [削除] を選択します。

> **Note**
>
> コンソールで一度に削除できるパッケージバージョンは 1 つのみです。一度に複数を削除するには、CLI を使用します。

npm パッケージまたはパッケージバージョンの削除

npm パッケージ、または個別のパッケージバージョンを削除するには、`--format` オプションを npm に設定します。スコープ内の npm パッケージを削除するには、`--namespace` オプションを使用して、スコープを指定します。例えば、`@types/react` パッケージを削除するには、`--namespace types` を使用します。`--namespace` を使用する際は、`@` 記号を省略します。

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

`@types/react` パッケージをそのすべてのバージョンも含めて削除するには:

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

Maven パッケージまたはパッケージバージョンの削除

Maven パッケージ、または個別のパッケージバージョンを削除するには、--format オプションを maven に設定し、--namespace オプションで Maven グループ ID を、--name オプションで Maven artifactID を渡して、削除するパッケージを指定します。次の例は、com.google.guava:guava の単一のバージョンを削除する方法を示しています。

```
 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
```

次の例は、com.google.guava:guava パッケージとそのバージョンをすべて削除する方法を示しています。

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

パッケージまたはパッケージバージョンの削除におけるベストプラクティス

パッケージバージョンを削除する必要がある場合は、ベストプラクティスとして、リポジトリを作成し、削除するパッケージバージョンのバックアップコピーを保存することをお勧めします。これを行うには、まず copy-package-versions をバックアップリポジトリに呼び出します。

```
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
```

パッケージバージョンをコピーした後に、削除するパッケージまたはパッケージバージョンに対して delete-package-versions を呼び出すことができます。

```
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
```

パッケージのバージョンの詳細と依存関係の表示および更新

CodeArtifact では、依存関係を含むパッケージのバージョンに関する情報を表示できます。パッケージバージョンのステータスを更新することもできます。パッケージのバージョンのステータスの詳細については、パッケージバージョンのステータスを参照してください。

パッケージバージョンの詳細を表示

`describe-package-version`コマンドを使用して、パッケージのバージョンの詳細を表示します。パッケージバージョンの詳細は、パッケージが CodeArtifact に公開されるときにパッケージから抽出されます。異なるパッケージの詳細は異なり、形式や作成者が追加した情報の量によって異なります。

`describe-package-version`コマンドの出力にあるほとんどの情報は、パッケージ形式に応じて異なります。例えば、`describe-package-version`は`package.json`ファイルから npm パッケージの情報を抽出します。リビジョンは CodeArtifact によって作成されます。詳細については、「パッケージバージョンリビジョンの指定」を参照してください。

同じ名前を持つ二つのパッケージバージョンは、それぞれ異なるネームスペースに存在する場合、同じリポジトリに配置できます。オプションの`--namespace`パラメータを使用して、ネームスペースを指定します。詳細については、npm パッケージバージョンの詳細の表示またはMaven パッケージバージョンの詳細の表示を参照してください。

次の例では、`my_repo`リポジトリ内の`pyhamcrest`という名前の Python パッケージのバージョン`1.9.0`の詳細を返します。

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

出力は次のようになります。

```
{
  "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 は、パッケージのホームページやパッケージライセンス情報などのパッケージバージョンの詳細を、パッケージ作成者が提供するメタデータから取得します。この情報のいずれかが DynamoDB 項目の上限サイズである 400 KB を超える場合、CodeArtifact はそのデータを処理できず、コンソールまたは `describe-package-version` のレスポンスにこの情報が表示されない可能性があります。例えば、https://pypi.org/project/rapyd-sdk/ などの Python パッケージのライセンスフィールドは非常に大きいため、この情報は CodeArtifact では処理されません。

npm パッケージバージョンの詳細の表示

npm パッケージバージョンの詳細を表示するには、`--format` オプションの値を **`npm`** に設定します。`--namespace` オプションで、パッケージバージョンの名前空間 (npm scope) をオプションに含めます。`--namespace`オプションの値には`@`の先頭を含めないでください。ネームスペース`@types`を検索するには、値を *`[###]`* に設定します。

以下は、`@types` スコープ内の `webpack` という名前の npm パッケージバージョン `4.41.5` の詳細を返します。

```
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
```

出力は次のようになります。

```
{
  "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"
  }
}
```

Maven パッケージバージョンの詳細の表示

Maven パッケージバージョンの詳細を表示するには、--format オプションの値を maven に設定し、パッケージバージョンの名前空間を --namespace オプションに含めます。

次の例では、org.apache.commonsネームスペースとmy_repoリポジトリに存在するcommons-rng-client-apiという名前の Maven パッケージのバージョン1.2の詳細を返します。

```
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
```

出力は次のようになります。

```
{
  "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 は、親 POM ファイルからパッケージバージョンの詳細情報を抽出しません。特定のパッケージバージョンのメタデータには、正確なパッケージバージョンの POM 内の情報のみが含まれ、POM parent タグを使用して推移的に参照される親 POM やその他の POM の情報は含まれません。つまり、describe-package-version の出力では、このメタデータを含む parent 参照に依存している Maven パッケージバージョンのメタデータ (ライセンス情報など) は省略されます。

パッケージバージョンの依存関係を表示する

list-package-version-dependenciesコマンドを使用すると、パッケージバージョンの依存関係のリストを取得できます。次のコマンドは、my_domainドメインのmy_repoリポジトリのバージョン4.41.5のmy-packageという名前の npm パッケージの依存関係を一覧表示します。

```
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
```

出力は次のようになります。

```
{
  "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"
}
```

dependencyType フィールドでサポートされている値の範囲については、CodeArtifact API の PackageDependency データ型を参照してください。

パッケージバージョンの readme ファイルの表示

npm などの一部のパッケージ形式には、READMEファイルが含まれます。get-package-version-readmeを使用してパッケージバージョンのREADMEファイルを取得します。次のコマンドは、my_domainドメインのmy_repoリポジトリにあるバージョン4.41.5のmy-packageという名前の npm パッケージのREADMEファイルを返します。

> **Note**
>
> CodeArtifact は、ジェネリックパッケージまたは Maven パッケージの readme ファイルの表示をサポートしていません。

```
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
```

出力は次のようになります。

```
{
  "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"
}
```

パッケージバージョンのステータスの更新

CodeArtifact のすべてのパッケージバージョンには、パッケージバージョンの現在の状態と使用可用性を示すステータスがあります。パッケージバージョンのステータスは、 AWS CLI と コンソールの両方を使用して変更できます。

> **Note**
>
> 使用可能なステータスのリストを含む、パッケージバージョンのステータスの詳細については、パッケージバージョンのステータスを参照してください。

パッケージバージョンのステータスの更新

パッケージバージョンのステータスを設定することで、そのパッケージバージョンをリポジトリから完全に削除することなく、使用方法をコントロールできます。例えば、パッケージバージョンのステータスがUnlistedの場合、通常どおりダウンロードすることはできますが、npm viewのようなコマンドに戻されたパッケージバージョンリストには表示されません。[パッケージのバージョンステータスのアップロード API] では、単一の API コールで、同じパッケージの複数のバージョンのパッケージバージョンステータスを設定することができます。さまざまなステータスの説明については、パッケージの概要を参照してください。

update-package-versions-statusコマンドを使用して、パッケージバージョンのステータスをPublished、Unlisted、またはArchivedに変更します コマンドを使用するために必要な IAM 権限を要求するには、パッケージバージョンのステータスを更新するために必要な IAM 権限を参照してください。次の例では、npm パッケージchalkのバージョン 4.1.0 のステータスをArchivedに設定します。

```
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
```

サンプル出力:

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

この例では、npm パッケージを使用しますが、このコマンドは他の形式でも同じように機能します。ひとつのコマンドを使用して、複数のバージョンを同じターゲットステータスに移動できます。次の例を参照してください。

```
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
```

サンプル出力:

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

いったん公開されると、パッケージバージョンをUnfinishedステータスに戻せないので、このステータスは--target-statusパラメータの値として許可されません。パッケージバージョンをDisposedの状態に動かすためには、代わりに以下のようにdispose-package-versionsコマンドを使用します。

パッケージバージョンのステータスを更新するために必要な IAM 権限

パッケージに update-package-versions-status を呼び出す場合は、パッケージリソースに対する codeartifact:UpdatePackageVersionsStatus 権限が必要です。つまり、パッケージごとに update-package-versions-status を呼び出す権限を付与することができます。例えば、npm パッケージ *[####]* の update-package-versions-status を呼び出す許可を付与する IAM ポリシーには、次のようなステートメントが含まれます。

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

スコープ指定された npm パッケージのステータスの更新

スコープを使用して npm パッケージバージョンのパッケージバージョンステータスを更新するには、--namespaceパラメータを使用してください。例えば、@nestjs/coreのバージョン 8.0.0 を一覧表示しないようにするためには、次のようなコマンドを使用します。

```
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
```

Maven パッケージのステータスの更新

Maven パッケージには常にグループ ID があり、CodeArtifact では名前空間と呼ばれます。update-package-versions-status を呼び出す際は、--namespace パラメータで Maven グループ ID を指定します。例えば、Maven パッケージ org.apache.logging.log4j:log4j のバージョン 2.13.1 をアーカイブするには、以下のコマンドを使用します。

```
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
```

パッケージバージョンリビジョンの指定

パッケージバージョンリビジョンは、パッケージバージョンの特定のアセットとメタデータのセットを指定する文字列です。パッケージバージョンリビジョンを指定して、特定の状態にあるパッケージバージョンを更新できます。パッケージバージョンのリビジョンを指定するには、--version-revisions パラメータを使用して、ひとつ以上のカンマ区切りパッケージバージョンとパッケージバージョンリビジョンのペアを渡します。パッケージバージョンのステータスは、パッケージバージョンの現在のリビジョンが指定された値と一致する場合にのみ更新されます。

> **Note**
>
> --version-revisionsパラメータを使用する場合、–-versionsパラメータも定義する必要があります。

```
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
```

ひとつのコマンドで複数のバージョンを更新するには、バージョンのカンマ区切りリストとバージョンリビジョンのペアを--version-revisionsオプションに渡します。次のコマンドの例では、二つの異なるパッケージバージョンとパッケージバージョンのリビジョンのペアを定義します。

```
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
```

サンプル出力:

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

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

複数のパッケージバージョンを更新する場合、--version-revisionsに渡されますバージョンが--versionsに渡されたバージョンと同じである必要があります。リビジョンが正しく指定されていない場合、そのバージョンのステータスは更新されません。

期待されるステータスパラメータの使用

update-package-versions-statusコマンドは、パッケージバージョンの予想される現在のステータスの指定をサポートする--expected-statusパラメータを提供します。現在のステータスが--expected-statusに渡された値と一致しない場合、そのパッケージバージョンのステータスは更新されません。

例えば、*[my_repo]* の、npm パッケージchalkのバージョン 4.0.0 および 4.1.0 には、現在Publishedのステータスがあります。Unlistedの期待されるステータスを指定するupdate-package-versions-statusへの呼び出しは、ステータスが一致しないため、両方のパッケージバージョンの更新ができません。

```
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
```

サンプル出力:

```
{
    "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"
        }
```

```
    }
}
```

個々のパッケージバージョンでのエラー

update-package-versions-statusを呼び出すときにパッケージバージョンのステータスが更新されない理由は複数あります。例えば、パッケージバージョンのリビジョンが正しく指定されていなかったり、予期されるステータスが現在のステータスと一致しない可能性があります。このような場合、そのバージョンは API レスポンスのfailedVersionsマップに含まれます。ひとつのバージョンが失敗した場合、update-package-versions-statusへの同じ呼び出しで指定された他のバージョンがスキップされ、ステータスが更新されない可能性があります。このようなバージョンは、SKIPPEDのerrorCodeでfailedVersionsマップにも含まれます。

update-package-versions-statusの現在の実装では、ひとつ以上のバージョンがステータスを変更できない場合、他のすべてのバージョンはスキップされます。つまり、すべてのバージョンが正常に更新されるか、すべてのバージョンが更新されないかです。この動作は API コントラクトでは保証されません。将来的には、一部のバージョンは成功し、他のバージョンはupdate-package-versions-statusへの 一度の呼び出しで失敗する可能性があります。

次のコマンドの例には、パッケージバージョンのリビジョンの不一致によるバージョンステータス更新の失敗が含まれます。この更新に障害があると、別のバージョンステータス更新の呼び出しがスキップされます。

```
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
```

サンプル出力:

```
{
    "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="
        }
    }
}
```

パッケージバージョンの廃棄

Disposedパッケージのステータスは、Archivedと同様の動作をします。ただし、パッケージアセットは CodeArtifact によって完全に削除されるため、ドメイン所有者のアカウントがアセットストレージに対してこれ以降請求されなくなります。各パッケージバージョンのステータスの詳細については、「パッケージバージョンのステータス」を参照してください。パッケージバージョンのステータスをDisposedに変更するには、dispose-package-versionsコマンドを使用してください。この機能はupdate-package-versions-statusとは別個のものです。それは、パッケージバージョンの破棄が可逆的なものでないからです。パッケージアセットが削除されるため、バージョンのステータスをArchived、Unlisted、またはPublishedに戻すことはできません。廃棄されたパッケージバージョンに対して実行できる唯一のアクションは、delete-package-versionsコマンドを使用して削除することだけです。

dispose-package-versions の呼び出しに成功するには、呼び出し IAM プリンシパルがパッケージリソースに対する codeartifact:DisposePackageVersions 権限を持っている必要があります。

dispose-package-versions コマンドの動作は、update-package-versions-status と同様であり、バージョンリビジョンおよび予期されるステータスのセクションで説明されている --version-revisions および --expected-status のオプションの動作も含まれます。例えば、次のコマンドはパッケージバージョンを破棄しようとしますが、予期されるステータスが一致しないために失敗します。

```
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
```

サンプル出力:

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

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

同じコマンドをPublishedの--expected-statusで再び実行すれば、破棄は成功します。

```
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
```

サンプル出力:

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

パッケージオリジンコントロールの編集

In AWS CodeArtifact では、パッケージバージョンを直接公開するか、アップストリームリポジトリからプルダウンするか、外部パブリックリポジトリから取り込むことで、パッケージバージョンをリポジトリに追加できます。直接公開とパブリックリポジトリからの取り込みの両方の方法でパッケージバージョンの追加を許可すると、依存関係置換攻撃に対して脆弱になります。詳細については、「依存関係置換攻撃」を参照してください。依存関係置換攻撃から保護するには、リポジトリ内のパッケージにパッケージオリジンコントロールを設定して、パッケージバージョンをリポジトリに追加する方法を制限します。

さまざまなパッケージの新しいバージョンを、直接公開などの内部ソースとパブリックリポジトリなどの外部ソースの両方から入手できるようにしたいと考えているチームは、パッケージオリジンコントロールの設定を検討する必要があります。デフォルトでは、パッケージオリジンコントロールは、パッケージの最初のバージョンがリポジトリに追加された方法に基づいて設定されます。パッケージオリジンコントロール設定とそのデフォルト値については、「パッケージオリジンコントロール設定」を参照してください。

`put-package-origin-configuration` API オペレーションを使用した後にパッケージレコードを削除するには、「`delete-package`」を使用します (「パッケージまたはパッケージバージョンを削除する」を参照)。

一般的なパッケージアクセスコントロールシナリオ

このセクションでは、パッケージバージョンが CodeArtifact リポジトリに追加される際のいくつかの一般的なシナリオについて説明します。パッケージオリジンコントロール設定は、最初のパッケージバージョンが追加された方法に基づいて、新しいパッケージに設定されます。

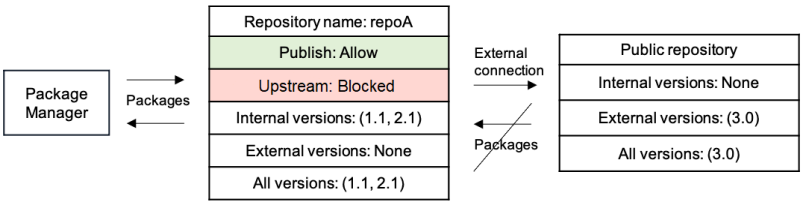
以下のシナリオの内部パッケージは、パッケージマネージャーからリポジトリに直接公開されるパッケージ (ユーザーまたはチームが作成、管理するパッケージなど) を指します。外部パッケージは、パブリックリポジトリに存在するパッケージで、外部接続でリポジトリに取り込むことができます。

外部パッケージバージョンが既存の内部パッケージに公開される

このシナリオでは、内部パッケージ「packageA」について考えてみます。チームは packageA の最初のパッケージバージョンを CodeArtifact リポジトリに公開します。これはパッケージの最初のパッケージバージョンであるため、パッケージオリジンコントロール設定は自動的に [公開: 許可] および [アップストリーム: ブロック] に設定されます。パッケージがリポジトリに保存されると、同じ名前のパッケージが CodeArtifact リポジトリに接続されているパブリックリポジトリに公開されます。これは、内部パッケージに対して試みられた依存関係置換攻撃である場合も、単なる偶然である

場合もあり得ます。いずれの場合でも、パッケージオリジンコントロールは、潜在的な攻撃からパッケージバージョンを保護するために、新しい外部バージョンの取り込みをブロックするように設定されています。

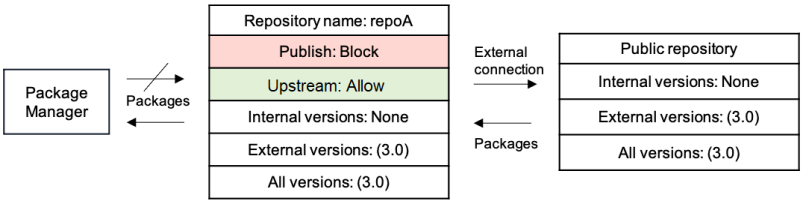
次の画像では、repoA はパブリックリポジトリへの外部接続を持つ CodeArtifact リポジトリを表します。リポジトリには packageA のバージョン 1.1 と 2.1 が含まれていますが、バージョン 3.0 はパブリックリポジトリに公開されています。通常、repoA はパッケージマネージャーからパッケージがリクエストされた後にバージョン 3.0 を取り込みます。パッケージの取り込みが [ブロック] に設定されているため、バージョン 3.0 は CodeArtifact リポジトリに取り込まれず、接続しているパッケージマネージャーからは使用できません。



内部パッケージバージョンが既存の外部パッケージに公開される

このシナリオでは、packageB という名前のパッケージは、リポジトリに接続したパブリックリポジトリの外部に存在します。リポジトリに接続しているパッケージマネージャーが packageB をリクエストすると、パッケージバージョンはパブリックリポジトリからリポジトリに取り込まれます。これは packageB の最初のパッケージバージョンであるため、パッケージオリジンコントロール設定は自動的に [公開: ブロック] および [アップストリーム: 許可] に設定されます。その後、ユーザーは同じパッケージ名のバージョンをリポジトリに公開しようとします。ここでは、ユーザーが公開パッケージを知らず、関連のないパッケージを同じ名前で公開しようとしているか、パッチが適用されたバージョンを公開しようとしているか、外部にすでに存在するパッケージバージョンとまったく同じものを直接公開しようとしている場合が考えられます。CodeArtifact は、公開しようとしているバージョンを拒否しますが、拒否を明示的にオーバーライドして、必要に応じてバージョンを公開することができます。

次の画像では、repoA はパブリックリポジトリへの外部接続を持つ CodeArtifact リポジトリを表します。リポジトリには、パブリックリポジトリから取り込んだバージョン 3.0 が含まれています。ユーザーは、バージョン 1.1 をリポジトリに公開したいと考えています。通常、バージョン 1.2 を repoA に公開できますが、公開が [ブロック] に設定されているため、バージョン 1.2 は公開できません。



既存の外部パッケージにパッチを適用したパッケージバージョンを公開する

このシナリオでは、packageB という名前のパッケージは、リポジトリに接続したパブリックリポジトリの外部に存在します。リポジトリに接続しているパッケージマネージャーが packageB をリクエストすると、パッケージバージョンはパブリックリポジトリからリポジトリに取り込まれます。これは packageB の最初のパッケージバージョンであるため、パッケージオリジンコントロール設定は自動的に [公開: ブロック] および [アップストリーム: 許可] に設定されます。チームは、このパッケージのパッチが適用されたパッケージバージョンをリポジトリに公開することを決めました。パッケージバージョンを直接公開するために、チームはパッケージのオリジンコントロール設定を [公開: 許可] および アップストリーム: ブロック] に変更します。これで、このパッケージのバージョンをリポジトリに直接公開し、パブリックリポジトリから取り込むことができます。チームがパッチを適用したパッケージバージョンを公開した後、チームはパッケージオリジンの設定を [公開: ブロック] および [アップストリーム: 許可] に戻します。

パッケージオリジンコントロール設定

パッケージオリジンコントロールでは、パッケージバージョンをリポジトリに追加する方法を設定できます。以下のリストには、使用可能なパッケージオリジンコントロールの設定と値が含まれています。

> **Note**
>
> パッケージグループでオリジンコントロールを設定する場合、使用できる設定項目や値は異なります。詳細については、「パッケージグループのオリジンコントロール」を参照してください。

公開

この設定は、パッケージマネージャーや類似のツールを使用してパッケージのバージョンをリポジトリに直接公開できるかどうかを設定します。

- 許可: パッケージバージョンを直接公開できます。
- ブロック: パッケージバージョンは直接公開できません。

アップストリーム

この設定は、パッケージマネージャーからのリクエストに応じて、パッケージバージョンを外部のパブリックリポジトリから取り込むことができるか、アップストリームリポジトリから保持できるかを設定します。

- 許可: すべてのパッケージバージョンは、アップストリームリポジトリとして設定された他の CodeArtifact リポジトリから保持することも、外部接続を使用してパブリックソースから取り込むこともできます。
- ブロック: すべてのパッケージバージョンは、アップストリームリポジトリとして設定された他の CodeArtifact リポジトリから保持することも、外部接続を使用してパブリックソースから取り込むこともできません。

パッケージオリジンコントロールのデフォルト設定

デフォルトのパッケージオリジンコントロール設定は、パッケージに関連付けられたパッケージグループのオリジンコントロール設定に基づいて設定されます。パッケージグループとパッケージグループのオリジンコントロールの詳細については、「CodeArtifact でのパッケージグループの操作」および「パッケージグループのオリジンコントロール」を参照してください。

パッケージが、すべての制限タイプについて `ALLOW` の制限設定を持つパッケージグループに関連付けられている場合、パッケージのデフォルトのパッケージオリジンコントロールは、そのパッケージの最初のバージョンがどのようにリポジトリに追加されたかに基づいて決まります。

- 最初のパッケージバージョンがパッケージマネージャーによって直接公開された場合、設定は [公開: 許可] と [アップストリーム: ブロック] になります。
- 最初のパッケージバージョンがパブリックソースから取り込まれた場合、設定は [公開: ブロック] と [アップストリーム: 許可] になります。

> **Note**
>
> 2022 年 5 月以前に CodeArtifact リポジトリに存在していたパッケージでは、デフォルトのパッケージオリジンコントロールは [公開: 許可] と [アップストリーム: 許可] になります。このようなパッケージでは、パッケージオリジンコントロールを手動で設定する必要があります。それ以降は、新しいパッケージには現在のデフォルト値が設定され、2022 年 7 月 14 日にこの機能がリリースされた時点でデフォルト値が強制されるようになりました。パッケージオリジンコントロールの設定の詳細については、「パッケージオリジンコントロールの編集」を参照してください。

パッケージが、少なくとも 1 つ以上の BLOCK または ALLOW_SPECIFIC_REPOSITORIES の制限設定を持つパッケージグループに関連付けられている場合、そのパッケージのデフォルトのオリジンコントロール設定は公開: ALLOW およびアップストリーム: ALLOW に設定されます。

パッケージオリジンコントロールとパッケージグループオリジンコントロールの相互作用

パッケージ、および関連するパッケージグループはそれぞれオリジンコントロール設定を備えているため、これらの 2 つの異なる設定がどのように相互作用するかを理解することが重要です。

両設定間の相互作用において、BLOCK の設定は常に ALLOW の設定よりも優先されます。次の表に、いくつかの設定例とその有効なオリジンコントロール設定を示します。

パッケージオリジンコントロール設定	パッケージグループオリジンコントロール設定	有効なオリジンコントロール設定
公開: ALLOW アップストリーム: ALLOW	公開: ALLOW アップストリーム: ALLOW	公開: ALLOW アップストリーム: ALLOW
公開: BLOCK アップストリーム: ALLOW	公開: ALLOW アップストリーム: ALLOW	公開: BLOCK アップストリーム: ALLOW
公開: ALLOW アップストリーム: ALLOW	公開: ALLOW アップストリーム: BLOCK	公開: ALLOW アップストリーム: BLOCK

つまり、パッケージのオリジン設定が 公開: ALLOW およびアップストリーム: ALLOW になっている場合、そのパッケージは実質的に、関連付けられているパッケージグループのオリジンコントロール設定に従うことを意味します。

パッケージオリジンコントロールの編集

パッケージオリジンコントロールは、パッケージの最初のパッケージバージョンがリポジトリに追加された方法に基づいて自動的に設定されます。詳細については、「パッケージオリジンコントロールのデフォルト設定」を参照してください。CodeArtifact リポジトリ内のパッケージのパッケージオリジンコントロールを追加または編集するには、次の手順に従います。

パッケージオリジンコントロールを追加または編集するには (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで [リポジトリ] を選択し、編集するパッケージを含むリポジトリを選択します。
3. [パッケージ] の表で、編集するパッケージを検索して選択します。
4. パッケージの概要ページの [オリジンコントロール] で、[編集] を選択します。
5. [オリジンコントロールの編集] で、パッケージに設定するパッケージオリジンコントロールを選択します。[公開] と [アップストリーム] の両方のパッケージオリジンコントロール設定を同時に設定する必要があります。

   - パッケージバージョンを直接公開できるようにするには、[公開] で [許可] を選択します。パッケージバージョンの公開を禁止するには、[ブロック] を選択します。
   - 外部リポジトリからのパッケージの取り込みとアップストリームリポジトリからのパッケージの取得を許可するには、[アップストリームソース] で [許可] を選択します。外部リポジトリおよびアップストリームリポジトリからのパッケージバージョンの取り込みとプルをすべてブロックするには、[ブロック] を選択します。

パッケージオリジンコントロールを追加または編集するには (AWS CLI)

1. まだ設定していない場合は、 AWS CLI 「」の手順に従って を設定しますAWS CodeArtifact でのセットアップ。
2. パッケージオリジンコントロールを追加または編集するには `put-package-origin-configuration` を使用します。次のフィールドを変更します。

   - *`my_domain`* を、更新するパッケージを含む CodeArtifact ドメインで置換します。
   - *`my_repo`* を、更新するパッケージを含む CodeArtifact リポジトリで置換します。
   - *`npm`* を、更新するパッケージのパッケージの形式で置換します。
   - *`my_package`* を、更新するパッケージのパッケージの名前で置換します。
   - *`[##]`* と *`[####]`* を使用するパッケージオリジンコントロール設定で置換します。

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

公開リポジトリとアップストリームリポジトリ

CodeArtifact では、到達可能なアップストリームリポジトリまたは公開リポジトリに存在するパッケージバージョンを公開することはできません。例えば、Maven パッケージcom.mycompany.mypackage:1.0をリポジトリmyrepoに公開するとし、myrepoが Maven Central への外部接続を持つアップストリームリポジトリを持つとします。次のシナリオを考えてみます。

1. com.mycompany.mypackage 上のパッケージオリジンコントロール設定は、[公開: 許可] と [アップストリーム: 許可] です。もし com.mycompany.mypackage:1.0 がアップストリームのリポジトリまたは Maven Central に存在する場合、CodeArtifact はそのリポジトリへの公開の試みを、myrepo 409 の競合エラーを通してすべて拒否します。com.mycompany.mypackage:1.1 などの別のバージョンを公開することもできます。
2. com.mycompany.mypackage 上のパッケージオリジンコントロール設定は、[公開: 許可] と [アップストリーム: ブロック] です。ユーザーはパッケージバージョンにはアクセスできないため、まだ存在していないすべてのバージョンの com.mycompany.mypackage をリポジトリに公開できます。
3. com.mycompany.mypackage 上のパッケージオリジンコントロール設定は、[公開: ブロック] と [アップストリーム: 許可] です。この場合、どのパッケージバージョンもリポジトリに直接公開することはできません。

CodeArtifact でのパッケージグループの操作

パッケージグループを使用すると、パッケージ形式、パッケージ名前空間、パッケージ名を使用して、定義したパターンに一致する複数のパッケージに設定を適用できます。パッケージグループを使用すると、複数のパッケージのパッケージオリジンコントロールをより簡単に設定できます。パッケージオリジンコントロールは、新しいパッケージバージョンの取り込みや公開をブロックまたは許可するために使用します。これにより、依存関係置換攻撃と呼ばれる悪意のあるアクションからユーザーを保護できます。

CodeArtifact のすべてのドメインには、ルートパッケージグループが自動で含まれています。このルートパッケージグループ (/*) には、すべてのパッケージが含まれています。これにより、パッケージバージョンは、デフォルトですべてのオリジンタイプからドメイン内のリポジトリに入れます。ルートパッケージグループは変更できますが、削除はできません。

新しいパッケージグループを作成する場合、または既存のパッケージグループを削除する場合、パッケージグループ設定機能は、結果整合性のある形で動作します。つまり、パッケージグループを作成または削除すると、オリジンコントロールは想定される関連パッケージに適用されますが、結果整合性による遅延が生じるため、反映までに時間がかかるが場合があります。結果整合性に達するまでの時間は、ドメイン内パッケージグループの数と、ドメイン内パッケージの数によって異なります。パッケージグループの作成または削除後、関連パッケージへのオリジンコントロールの反映にわずかな遅れが生じる場合があります。

一方で、パッケージグループのオリジンコントロールを更新した場合は、ほぼ即時に反映されます。パッケージグループの作成または削除の場合とは異なり、既存のパッケージグループのオリジンコントロールの変更は、同様の遅延なく関連パッケージに反映されます。

これらのトピックには、AWS CodeArtifact のパッケージグループに関する情報が含まれます。

トピック

パッケージグループを作成する

CodeArtifact コンソール、AWS Command Line Interface (AWS CLI)、または CloudFormation を使用して、パッケージグループを作成できます。CloudFormation を使用して CodeArtifact パッケージグループを管理する方法の詳細については、「AWS CloudFormation での CodeArtifact リソースの作成」を参照してください。

パッケージグループを作成する (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで [ドメイン] を選択し、パッケージグループを作成するドメインを選択します。
3. [パッケージグループ]、[パッケージグループを作成] を順に選択します。
4. [パッケージグループの定義] に、作成するパッケージグループの定義を入力します。パッケージグループの定義によって、グループに関連付けられるパッケージが決まります。パッケージグループの定義は、テキストとして手動入力できます。または、ビジュアルモードを使用して選択するとパッケージグループの定義が自動で作成されます。
5. ビジュアルモードを使用してパッケージグループの定義を作成する方法:

   a. [ビジュアル] を選択してビジュアルモードに切り替えます。
   b. [パッケージの形式] で、このグループに関連付けるパッケージの形式を選択します。
   c. [名前空間 (スコープ)] で、一致させたい名前空間条件を選択します。

      - [次と等しい:]: 指定された名前空間と完全に一致します。選択した場合、一致させたい名前空間を入力します。
      - [空白]: 名前空間のないパッケージと一致します。
      - [次で始まる:]: 指定された単語で始まる名前空間と一致します。選択した場合、一致させるプレフィックスの単語を入力します。単語および単語境界の詳細については、「単語、単語境界、プレフィックスマッチング」を参照してください。
      - [すべて]: すべての名前空間のパッケージと一致します。

   d. [次と等しい:]、[空白]、または [次で始まる:] が選択されている場合は、[パッケージ名] で、一致させたいパッケージ名条件を選択します。

      - [完全に等しい:]: 指定されたパッケージ名と完全に一致します。選択した場合、一致させるパッケージ名を入力します。

- [次のプレフィックスで始まる:]: 指定されたプレフィックスで始まるパッケージと一致します。
- [次で始まる:]: 指定された単語で始まるパッケージと一致します。選択した場合、一致させるプレフィックスの単語を入力します。単語および単語境界の詳細については、「単語、単語境界、プレフィックスマッチング」を参照してください。
- [すべて]: すべてのパッケージと一致します。

e. [次へ] を選択して、定義を確認します。

6. テキストでパッケージグループ定義を入力する方法:

a. [テキスト] を選択してテキストモードに切り替えます。

b. [パッケージグループの定義] に、パッケージグループの定義を入力します。パッケージグループ定義の構文の詳細については、「パッケージグループ定義の構文と一致動作」を参照してください。

c. [次へ] を選択して、定義を確認します。

7. [レビューの定義] で、前述の定義に基づき、新しいパッケージグループに含まれるパッケージを確認します。パッケージを確認したら、[次へ] を選択します。

8. [パッケージグループの情報] に、パッケージグループの説明と連絡先メールアドレスを任意で追加します。[次へ] を選択します。

9. [パッケージオリジンコントロール] で、グループ内のパッケージに適用するオリジンコントロールを設定します。パッケージグループオリジンコントロールの設定の詳細については、「パッケージグループのオリジンコントロール」を参照してください。

10. [パッケージグループを作成] を選択します。

パッケージグループを作成する (AWS CLI)

`create-package-group` コマンドを使用して、ドメインにパッケージグループを作成します。`--package-group` オプションで、グループに関連付けられるパッケージを決定するパッケージグループの定義を入力します。パッケージグループ定義の構文の詳細については、「パッケージグループ定義の構文と一致動作」を参照してください。

まだ設定していない場合は、「AWS CodeArtifact でのセットアップ」の手順に従って AWS CLI を設定してください。

```
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
```

パッケージグループを表示または編集する

CodeArtifact コンソールまたは AWS Command Line Interface (AWS CLI) を使用して、すべてのパッケージグループを一覧表示したり、特定のパッケージグループの詳細を表示したり、パッケージグループの詳細や設定を編集したりできます。

パッケージグループを表示または編集する (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで [ドメイン] を選択し、表示または編集するパッケージグループを含むドメインを選択します。
3. [パッケージグループ] を選択し、表示または編集するパッケージグループを選択します。
4. [詳細] で、親グループ、説明、ARN、連絡先 E メール、パッケージオリジンコントロールなど、パッケージグループに関する情報を表示します。
5. [サブグループ] では、このグループを親グループとするパッケージグループを一覧表示できます。このリスト内のパッケージグループは、このパッケージグループから設定を継承できます。詳細については、「パッケージグループの階層とパターンの特異度」を参照してください。
6. [パッケージ] で、パッケージグループの定義に基づいて、このパッケージグループに属するパッケージを表示します。[強度] 列には、パッケージの関連付けの強度が表示されます。詳細については、「パッケージグループの階層とパターンの特異度」を参照してください。
7. パッケージグループ情報を編集するには、[パッケージグループを編集] を選択します。

    a. [情報] で、パッケージグループの説明または連絡先情報を更新します。パッケージグループの定義を編集することはできません。

    b. [パッケージグループのオリジンコントロール] で、パッケージグループのオリジンコントロール設定を更新します。これにより、関連パッケージがドメイン内のリポジトリに入れられる方法が決まります。詳細については、「パッケージグループのオリジンコントロール」を参照してください。

パッケージグループを表示または編集する (AWS CLI)

次のコマンドを使用して、AWS CLI でパッケージグループを表示または編集します。まだ設定していない場合は、「AWS CodeArtifact でのセットアップ」の手順に従って AWS CLI を設定してください。

ドメイン内のすべてのパッケージグループを表示するには、`list-package-groups` コマンドを使用します。

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

パッケージグループの詳細を表示するには、`describe-package-group` コマンドを使用します。パッケージグループ定義の詳細については、「パッケージグループ定義の構文と例」を参照してください。

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

パッケージグループの子パッケージグループを表示するには、`list-sub-package-groups` コマンドを使用します。

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

パッケージに関連付けられるパッケージグループを表示するには、`get-associated-package-group` コマンドを使用します。NuGet、Python、Swift パッケージ形式では、正規化されたパッケージ名と名前空間を使用しなければなりません。パッケージ名と名前空間を正規化する方法の詳細については、NuGet、Python、Swift の名前の正規化ドキュメントを参照してください。

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

```
        --namespace scope
```

パッケージグループを編集するには、update-package-group コマンドを使用します。このコマンドは、パッケージグループの連絡先情報または説明の更新に使用されます。パッケージグループのオリジンコントロール設定とその追加または編集については、「パッケージグループのオリジンコントロール」を参照してください。パッケージグループ定義の詳細については、「パッケージグループ定義の構文と例」を参照してください。

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

パッケージグループを削除する

CodeArtifact コンソールまたは AWS Command Line Interface (AWS CLI) を使用してパッケージグループを削除できます。

パッケージグループを削除する際は、次の動作に注意してください。

- ルートパッケージグループ (/*) は削除できません。
- 当該パッケージグループに関連付けられているパッケージとパッケージバージョンは削除されません。
- パッケージグループを削除すると、直接の子パッケージグループは、パッケージグループの直上の親パッケージグループの子になります。したがって、いずれかの子グループが親の設定を継承している場合、それらの設定は変更される可能性があります。

パッケージグループを削除する (コンソール)

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで [ドメイン] を選択し、表示または編集するパッケージグループを含むドメインを選択します。
3. [パッケージグループ] を選択します。
4. 削除するパッケージグループを選択し、[削除] を選択します。

5. フィールドに「delete」と入力し、[削除] を選択します。

パッケージグループを削除する (AWS CLI)

パッケージグループを削除するには、delete-package-group コマンドを使用します。

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

パッケージグループのオリジンコントロール

パッケージオリジンコントロールを使用して、パッケージバージョンをドメインに入れる方法を設定します。パッケージグループでオリジンコントロールを設定することで、パッケージグループに関連付けられたすべてのパッケージのバージョンを、特定のドメイン内リポジトリに入れる方法を設定できます。

パッケージグループのオリジンコントロール設定には、以下の項目が含まれます。

- 制限設定: この設定は、パッケージを、CodeArtifact のリポジトリに、公開、内部アップストリーム、外部パブリックリポジトリのいずれから入れることが出来るかを定義します。
- 許可されたリポジトリリスト: 各制限項目ごとに、どのリポジトリを許可するかを個別に指定できます。特定のリポジトリを許可するように制限が設定されている場合、その制限には対応する許可されたリポジトリのリストが含まれます。

> **Note**
>
> パッケージグループのオリジンコントロール設定は、個別パッケージのオリジンコントロール設定と若干異なります。パッケージオリジンコントロール設定の詳細については、「パッケージオリジンコントロール設定」を参照してください。

制限設定

パッケージグループのオリジンコントロール設定の制限設定によって、そのグループに関連付けられたパッケージがドメイン内のリポジトリに入る方法が決まります。

発行

PUBLISH 設定は、パッケージマネージャーや類似のツールを使用してパッケージバージョンをあらゆるドメイン内リポジトリに直接公開できるかどうかを設定します。

- [ALLOW]: すべてのパッケージバージョンを直接公開できます。
- [BLOCK]: パッケージバージョンはどのリポジトリにも直接公開できません。
- [ALLOW_SPECIFIC_REPOSITORIES]: パッケージバージョンは、公開を許可されたリポジトリリストで指定されたリポジトリにのみ、直接公開できます。
- [INHERIT]: PUBLISH 設定は、INHERIT ではない設定を持つ最初の親パッケージグループから継承されます。

EXTERNAL_UPSTREAM

EXTERNAL_UPSTREAM 設定は、パッケージマネージャーからのリクエストに応じて、パッケージバージョンを外部のパブリックリポジトリから取り込むことができるかどうかを設定します。サポートされている外部リポジトリのリストについては、「サポートされている外部接続リポジトリ」を参照してください。

- [ALLOW]: すべてのパッケージバージョンは、外部接続を持つパブリックソースからすべてのリポジトリに取り込むことができます。
- [BLOCK]: パッケージバージョンは、外部接続を持つパブリックソースからリポジトリに取り込むことはできません。
- [ALLOW_SPECIFIC_REPOSITORIES]: パッケージバージョンは、パブリックソースから、外部アップストリームの許可されたリポジトリリスト内の指定されたリポジトリにのみ取り込むことができます。
- [INHERIT]: EXTERNAL_UPSTREAM 設定は、INHERIT ではない設定を持つ最初の親パッケージグループから継承されます。

INTERNAL_UPSTREAM

INTERNAL_UPSTREAM 設定は、パッケージマネージャーからのリクエストに応じて、パッケージバージョンを同じ CodeArtifact ドメインの内部アップストリームリポジトリから保持できるかどうかを設定します。

- [ALLOW]: アップストリームリポジトリとして設定された他の CodeArtifact リポジトリからは、任意のパッケージバージョンを保持することができます。
- [BLOCK]: アップストリームリポジトリとして設定された他の CodeArtifact リポジトリからは、パッケージバージョンを保持することはできません。
- [ALLOW_SPECIFIC_REPOSITORIES]: パッケージバージョンは、アップストリームリポジトリとして設定された他の CodeArtifact リポジトリから、内部アップストリームの許可されたリポジトリリストで指定されたリポジトリに対してのみ保持できます。
- [INHERIT]: INTERNAL_UPSTREAM 設定は、INHERIT 以外の設定を持つ最初の親パッケージグループから継承されます。

許可されたリポジトリリスト

制限設定が ALLOW_SPECIFIC_REPOSITORIES として設定されている場合、パッケージグループには、その制限設定で許可されるリポジトリリストを含む、付随する許可されたリポジトリリストが含まれます。したがって、パッケージグループには、ALLOW_SPECIFIC_REPOSITORIES として設定された設定ごとに 1 つずつ、0 ~ 3 個の許可されたリポジトリリストが含まれます。

パッケージグループの許可されたリポジトリリストにリポジトリを追加するときは、追加先の許可されたリポジトリリストを指定しなければなりません。

許可されるリポジトリリストは次のとおりです。

- EXTERNAL_UPSTREAM: 追加するリポジトリ内の外部リポジトリからのパッケージバージョンの取り込みを許可またはブロックします。
- INTERNAL_UPSTREAM: 追加するリポジトリ内の別の CodeArtifact リポジトリからのパッケージバージョンのプルを許可またはブロックします。
- PUBLISH: パッケージマネージャーから追加するリポジトリへのパッケージバージョンの直接公開を許可またはブロックします。

パッケージグループオリジンコントロール設定の編集

パッケージグループのオリジンコントロールを追加または編集するには、次の手順に従います。パッケージグループオリジンコントロール設定については、「制限設定」および「許可されたリポジトリリスト」を参照してください。

パッケージグループオリジンコントロールを追加または編集する方法 (CLI)

1. まだ設定していない場合は、「AWS CodeArtifact でのセットアップ」の手順に従って AWS CLI を設定してください。
2. パッケージオリジンコントロールを追加または編集するには update-package-group-origin-configuration を使用します。

   - --domain には、更新するパッケージグループを含む CodeArtifact ドメインを入力します。
   - --domain-owner には、ドメインの所有者のアカウント番号を入力します。
   - --package-group には、更新するパッケージグループを入力します。
   - --restrictions には、オリジンコントロールの制限を表すキーと値のペアを入力します。
   - --add-allowed-repositories には、制限タイプとリポジトリ名を含む JSON オブジェクトを入力します。これは、制限に対応する許可されたリポジトリリストに追加されます。
   - --remove-allowed-repositories には、制限タイプとリポジトリ名を含む JSON オブジェクトを入力します。これは、制限に対応する許可されたリポジトリリストから削除されます。

```
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
```

次の例では、複数の制限と複数のリポジトリを 1 つのコマンドに追加します。

```
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
```

パッケージグループのオリジンコントロール設定の例

次の例は、一般的なパッケージ管理シナリオにおけるパッケージオリジンのコントロール設定です。

プライベート名を持つパッケージの公開の許可 (取り込みは許可しない)

このシナリオは、パッケージ管理における一般的なシナリオであると考えられます。

- プライベート名を持つパッケージについて、パッケージマネージャーからドメイン内リポジトリに公開されることを許可し、外部パブリックリポジトリからドメイン内リポジトリに取り込まれることをブロックします。
- 他のすべてのパッケージについて、外部パブリックリポジトリからドメイン内リポジトリに取り込まれることを許可し、パッケージマネージャーからドメイン内リポジトリに公開されるることをブロックします。

これを実現するには、プライベート名 (複数可)、および PUBLISH: ALLOW、EXTERNAL_UPSTREAM: BLOCK、および INTERNAL_UPSTREAM: ALLOW のオリジン設定を含むパターンでパッケージグループを設定する必要があります。これにより、プライベート名を持つパッケージは直接公開されるようになりますが、外部リポジトリから取り込むことはできなくなります。

次の AWS CLI コマンドは、目的の動作に一致するオリジン制限設定を使用してパッケージグループを作成して設定します。

パッケージグループを作成する方法:

```
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"
```

パッケージグループのオリジン設定を更新する方法:

```
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
```

特定のリポジトリを介した外部リポジトリからの取り込みの許可

このシナリオでは、ドメインに複数のリポジトリがあります。これらのリポジトリのうち、以下に示すように、repoA は repoB にアップストリーム接続し、さらに、公開リポジトリ (npmjs.com) に外部接続しています。

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

特定のパッケージグループからのパッケージの取り込み (npmjs.com から repoA への /npm/space/anycompany~ の取り込み) を、repoB からのみ許可します。また、パッケージグループに関連付けられたパッケージがドメイン内の他のいかなるリポジトリにも取り込まれないように、またパッケージマネージャーからパッケージが直接公開されないようにブロックしたい場合があります。これを実現するには、パッケージグループを次のように作成、設定します。

PUBLISH: BLOCK、EXTERNAL_UPSTREAM: ALLOW_SPECIFIC_REPOSITORIES、INTERNAL_UPSTREAM: ALLOW_SPECIFIC_REPOSITORIES のオリジン制限設定。

repoA と repoB を適切な許可されたリポジトリリストに追加します。

- repoA は、内部アップストリーム (repoB) からパッケージを取得するため、INTERNAL_UPSTREAM リストに追加する必要があります。
- repoB は、外部リポジトリ (npmjs.com) からパッケージを取得するため、EXTERNAL_UPSTREAM リストに追加する必要があります。

次の AWS CLI コマンドは、目的の動作に一致するオリジン制限設定を使用してパッケージグループを作成して設定します。

パッケージグループを作成する方法:

```
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"
```

パッケージグループのオリジン設定を更新する方法:

```
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
```

パッケージグループのオリジンコントロール設定とパッケージオリジンコントロール設定の相互作用

パッケージ、および関連するパッケージグループはそれぞれオリジンコントロール設定を備えているため、これらの 2 つの異なる設定がどのように相互作用するかを理解することが重要です。設定間の相互作用については、「パッケージオリジンコントロールとパッケージグループオリジンコントロールの相互作用」を参照してください。

パッケージグループ定義の構文と一致動作

このトピックには、パッケージグループの定義、パターンマッチング動作、パッケージの関連付け強度、パッケージグループ階層に関する情報が含まれます。

目次

- 強い一致と弱い一致
- その他のバリエーション

パッケージグループ定義の構文と例

パッケージグループを定義するためのパターン構文は、パッケージパスの形式に厳密に従います。パッケージパスは、パッケージの座標コンポーネント (形式、名前空間、名前) に基づき作成されます。先頭にスラッシュを追加し、各コンポーネントをスラッシュで区切ります。例えば、space という名前空間の anycompany-ui-components という npm パッケージのパッケージパスは、/npm/space/anycompany-ui-components です。

パッケージグループパターンは、パッケージパスと同じ構造に従います。ただし、グループ定義の一部として指定されていないコンポーネントは省略され、パターンの末尾にはサフィックスが付きます。付加されるサフィックスによって、パターンの一致動作が次のように決まります。

- サフィックスが $ の場合、完全なパッケージ座標が一致します。
- サフィックスが ~ の場合、プレフィックスが一致します。
- サフィックスが * の場合、以前に定義したコンポーネントのすべての値が一致します。

使用できる各組み合わせのパターン例を以下に示します。

1. すべてのパッケージ形式: /*
2. 特定のパッケージ形式: /npm/*
3. パッケージ形式と名前空間プレフィックス: /maven/com.anycompany~
4. パッケージ形式と名前空間: /npm/space/*
5. パッケージ形式、名前空間、名前プレフィックス: /npm/space/anycompany-ui~
6. パッケージ形式、名前空間、名前: /maven/org.apache.logging.log4j/log4j-core$

上記の例に示すように、~ サフィックスは、プレフィックスの一致を表すために名前空間または名前の末尾に付加されます。パス内の次のコンポーネントのすべての値 (すべての形式、すべての名前空間、またはすべての名前) を一致させるために使用する場合、スラッシュの後に * が付加されます。

パッケージグループの定義と正規化

CodeArtifact は、NuGet、Python、Swift パッケージ名を正規化し、保存する前に Swift パッケージ名前空間を正規化します。CodeArtifact は、パッケージグループ定義でパッケージを照合するときに、

これらの正規化された名前を使用します。したがって、これらの形式の名前空間または名前を含むパッケージグループは、正規化された名前空間と名前を使用しなければなりません。パッケージ名と名前空間を正規化する方法の詳細については、NuGet、Python、Swift の名前の正規化ドキュメントを参照してください。

パッケージグループ定義の名前空間

名前空間を含まないパッケージまたはパッケージ形式の場合 (Python および NuGet)、パッケージグループに名前空間を含めることはできません。これらのパッケージグループのパッケージグループ定義には、空白の名前空間セクションが含まれています。例えば、requests という名前の Python パッケージのパスは /python//requests です。

名前空間を含むパッケージまたはパッケージ形式の場合 (Maven、汎用、Swift)、パッケージ名が含まれていれば名前空間を含める必要があります。Swift パッケージ形式の場合は、正規化されたパッケージ名前空間が使用されます。Swift パッケージ名前空間の正規化方法の詳細については、「Swift パッケージ名と名前空間の正規化」を参照してください。

パッケージグループの階層とパターンの特異度

パッケージグループに「含まれる」または「関連付けられる」パッケージとは、そのグループのパターンには一致するが、より詳細なパターンを持つ別のグループには一致しないパッケージです。例えば、パッケージグループ /npm/*と /npm/space/* の場合、パッケージパス /npm//react は最初のグループ (/npm/*) に関連付けられ、/npm/space/aui.components と /npm/space/amplify-ui-core は 2 番目のグループ (/npm/space/*) に関連付けられます。パッケージが複数のグループに一致する場合でも、そのパッケージが実際に関連付けられるのは 1 つのグループだけであり、それは最も特異的に一致するグループです。

パッケージパスが複数のパターンに一致する場合、より「特異的」なパターンとは、最長一致するパターンと考えることができます。別の言い方をすると、より特異的なパターンとは、より一般的なパターンに一致するパッケージのうち、真部分集合にだけ一致するパターンを指します。前の例では、/npm/space/* に一致するすべてのパッケージが /npm/* とも一致しますが、その逆は成り立ちません。つまり、/npm/space/* は、/npm/* の真部分集合であるため、より特異的なパターンになります。一方のグループがもう一方のグループの部分集合である場合、そこには階層が生じ、/npm/space/* は親グループである /npm/* のサブグループとなります。

パッケージには最も特異的なパッケージグループの設定のみが適用されますが、そのグループが親グループの設定を継承するように構成されている場合もあります。

単語、単語境界、プレフィックスマッチング

プレフィックスマッチングについて説明する前に、いくつかの重要な用語を定義します。

- 単語は、文字または数字であり、0 個以上の文字、数字、またはマーク文字 (アクセント、ウムラウトなど) が続きます。
- 単語境界は、単語以外の文字に達したときの単語の末尾にあります。単語以外の文字は、.、-、_ などの句読点文字です。

具体的には、単語の正規表現パターンは `[\p{L}\p{N}][\p{L}\p{N}\p{M}]*` で、次のように分解できます。

- `\p{L}` は任意の文字を表しています。
- `\p{N}` は任意の数字を表しています。
- `\p{M}` は、アクセント、ウムラウトなどのマーク文字を表しています。

したがって、`[\p{L}\p{N}]` は数字または文字を表し、`[\p{L}\p{N}\p{M}]*` は 0 個以上の文字、数字、またはマーク文字を表します。この正規表現パターンの各一致の末尾には単語境界があります。

> **Note**
>
> 単語境界の一致は、この「単語」の定義に基づいています。これは、辞書で定義される単語や CamelCase を基準にしたものではありません。例えば、`oneword` または `OneWord` に単語境界はありません。

以上のように単語と単語境界を定義したので、CodeArtifact のプレフィックスマッチングについて説明できます。単語境界のプレフィックスの一致を示すために、単語文字の後に一致文字 (~) が使用されます。例えば、パターン `/npm/space/foo~` は、パッケージパス `/npm/space/foo` と `/npm/space/foo-bar` に一致しますが、`/npm/space/food` と `/npm/space/foot` には一致しません。

パターン `/npm/*` のように、単語以外の文字に続く場合は ~ の代わりにワイルドカード (*) を使用する必要があります。

大文字と小文字の区別

パッケージグループ定義では大文字と小文字が区別されます。つまり、大文字と小文字だけが異なるパターンでも、個別のパッケージグループとして存在できます。例えば、ユーザーは、npm Public Registry に存在する、大文字と小文字のみが異なる AsyncStorage、asyncStorage、asyncstorage という 3 つの個別のパッケージに対して、パターン `/npm//AsyncStorage$`、`/npm//asyncStorage$`、および `/npm//asyncstorage$` を備える個別のパッケージグループを作成できます。

大文字と小文字は区別されますが、パッケージに大文字と小文字が異なるパターンのバリエーションがあっても、CodeArtifact はパッケージをパッケージグループに関連付けます。ユーザーが、上記のうち `/npm//AsyncStorage$` のパッケージグループのみを作成し、他の 2 つのグループを作成しない場合、AsyncStorage の大文字と小文字のすべてのバリエーション (asyncStorage と asyncstorage など) がパッケージグループに関連付けられます。ただし、次のセクション (強い一致と弱い一致) で説明するように、これらのバリエーションはパターンと完全に一致する AsyncStorage とは異なる方法で処理されます。

強い一致と弱い一致

前のセクション (大文字と小文字の区別) では、パッケージグループでは大文字と小文字が区別されると述べたうえで、大文字と小文字は同じように扱われると説明しました。これは、CodeArtifact のパッケージグループ定義には、強い一致 (完全一致) と弱い一致 (バリエーション一致) の概念があるためです。強い一致は、パッケージがパターンと完全に一致し、バリエーションがない場合です。弱い一致は、パッケージがパターンのバリエーション (大文字と小文字が異なるなど) と一致する場合です。弱い一致の動作により、パッケージグループのパターンのバリエーションにあたるパッケージが、より一般的なパッケージグループに割り当てられてしまうことを防ぎます。パッケージが、最も特異的に一致するグループのパターンのバリエーション (弱い一致) である場合、パッケージはグループに関連付けられます。ただし、グループのオリジンコントロール設定が適用されることはなく、パッケージはブロックされるため、パッケージの新しいバージョンがアップストリームからプルされたり公開されたりすることはありません。この動作により、ほぼ同名のパッケージの依存関係の混乱に起因するサプライチェーン攻撃のリスクが軽減されます。

弱い一致の動作を示す例として、パッケージグループ `/npm/*` が取り込みを許可し、公開をブロックするとします。より具体的なパッケージグループ `/npm//anycompany-spicy-client$` は、取り込みをブロックし、公開を許可するように設定されています。anycompany-spicy-client という名前のパッケージは、そのパッケージグループの強い一致であり、パッケージバージョンの公開を許可し、パッケージバージョンの取り込みをブロックします。公開が許可される、大文字と小文字を区別するパッケージ名は、パッケージ定義パターンとの強い一致である anycompany-spicy-client のみで

す。AnyCompany-spicy-client などの大文字と小文字が異なるバリエーションは、弱い一致であるため公開がブロックされます。さらに重要なのは、パッケージグループはパターンで使用される小文字の名前だけでなく、大文字と小文字のすべてのバリエーションの取り込みをブロックします。これにより、依存関係撹乱攻撃のリスクが低減されます。

その他のバリエーション

大文字と小文字の違いに加えて、弱い一致では、ダッシュ (-)、ドット (.)、アンダースコア (_)、および紛らわしい文字 (別のアルファベットに見た目が似ている文字など) の並びの違いも無視されます。弱い一致に使用される正規化の際に、CodeArtifact は大文字小文字の畳み込み (小文字への変換と同様) を実行し、ダッシュ、ドット、アンダースコアの文字の並びを 1 つのドットに置き換え、混同されやすい文字を正規化します。

弱い一致では、ダッシュ、ドット、アンダースコアは同等に扱われますが、完全に無視されることはありません。つまり、foo-bar、foo.bar、foo..bar、および foo_bar はすべて同等の弱い一致ですが、foobar は同じではありません。一部のパブリックリポジトリには、これらのタイプのバリエーションを防ぐためのステップが実装されていますが、パブリックリポジトリによって提供される保護により、パッケージグループのこの機能が不要になることはありません。例えば、npm Public Registry レジストリなどのパブリックリポジトリは、my-package という名前のパッケージが既に公開されている場合にのみ、my-package の新しいバリエーションを防止します。my-package が内部パッケージであり、公開を許可して取り込みをブロックするパッケージグループ `/npm//my-package$` が作成されている場合は、my.package などのバリアントが許可されないように、my-package を npm Public Registry に公開しないことが推奨されます。

Maven などの一部のパッケージ形式はこれらの文字を異なるものとして扱いますが (Maven では、. は名前空間階層の区切り文字として扱われますが、- や _ はそのように扱われません)、com.act-on のようなものはそれでも com.act.on と混同される可能性があります。

> **Note**
>
> なお、複数のバリエーションがパッケージグループに関連付けられるたびに、管理者は特定のバリエーションの新しいパッケージグループを作成して、そのバリエーションに異なる動作を設定する場合があります。

CodeArtifact でパッケージグループにタグを付ける

タグは、AWS リソースに関連付けられるキーと値のペアです。CodeArtifact では、パッケージグループにタグを適用できます。CodeArtifact リソースのタグ付け、ユースケース、タグのキーと値の

制約、サポートされているリソースタイプの詳細については、リソースのタグ付けを参照してください。

パッケージグループを作成するとき、または、既存のパッケージグループのタグ値を追加、削除、更新するときに CLI を使用してタグを指定できます。

パッケージグループにタグを付ける (CLI)

CLI を使用してパッケージグループのタグを管理します。

まだ設定していない場合は、「AWS CodeArtifact でのセットアップ」の手順に従って AWS CLI を設定してください。

> **Tip**
>
> タグを追加するには、パッケージグループの Amazon リソースネーム (ARN) の指定が必要です。パッケージグループの ARN を取得するには、`describe-package-group` コマンドを実行します。
>
> ```
> aws codeartifact describe-package-group \
>    --domain my_domain \
>    --package-group /npm/scope/anycompany~ \
>    --query packageGroup.arn
> ```

トピック

- パッケージグループにタグを追加する (CLI)
- パッケージグループのタグを表示する (CLI)
- パッケージグループのタグを編集する (CLI)
- パッケージグループからタグを削除する (CLI)

パッケージグループにタグを追加する (CLI)

タグは、パッケージグループの作成時に追加したり、既存のパッケージグループに追加したりできます。パッケージグループの作成時にタグを追加する方法については、「パッケージグループを作成する」を参照してください。

AWS CLI を使用して既存のパッケージグループにタグを追加するには、ターミナルまたはコマンドラインで、tag-resource コマンドを実行して、タグを追加するパッケージグループの Amazon リソースネーム (ARN) と、追加するタグのキーおよび値を指定します。パッケージグループの ARN については、「パッケージグループ ARN」を参照してください。

パッケージグループには 1 つ以上のタグを追加できます。例えば、*/npm/scope/anycompany~* という名前のパッケージグループに 2 つのタグを付けます。*value1* のタグ値がある *key1* という名前のタグキーと、*value2* のタグ値がある *key2* という名前のタグキーです。

```
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
```

成功した場合は、コマンドの出力はありません。

パッケージグループのタグを表示する (CLI)

AWS CLI を使用してパッケージグループの AWS タグを表示するには、以下の手順に従います。タグが追加されていない場合、返されるリストは空になります。

ターミナルまたはコマンドラインで、パッケージグループの Amazon リソースネーム (ARN) を指定して list-tags-for-resource コマンドを実行します。パッケージグループの ARN については、「パッケージグループ ARN」を参照してください。

例えば、ARN 値が arn:aws:codeartifact:*us-west-2*:*123456789012*:package-group/*my_domain*/*npm/scope/anycompany~* のパッケージグループ */npm/scope/anycompany~* のタグキーとタグ値のリストを表示する方法は次のとおりです。

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

成功した場合、このコマンドは次のような情報を返します。

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

```
}
```

パッケージグループのタグを編集する (CLI)

AWS CLI を使用してパッケージグループのタグを編集するには、以下の手順に従います。既存のキーの値を変更したり、別のキーを追加できます。次のセクションに示すように、パッケージグループからタグを削除することもできます。

ターミナルまたはコマンドラインで、tag-resource コマンドを実行して、タグを更新するパッケージグループの ARN、およびタグキーとタグ値を指定します。パッケージグループの ARN については、「パッケージグループ ARN」を参照してください。

```
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
```

成功した場合は、コマンドの出力はありません。

パッケージグループからタグを削除する (CLI)

AWS CLI を使用してパッケージグループからタグを削除するには、以下の手順に従います。

> **Note**
>
> パッケージグループを削除すると、削除されたパッケージグループのすべてのタグの関連付けが解除されます。パッケージグループを削除する前にタグを削除する必要はありません。

ターミナルまたはコマンドラインで、untag-resource コマンドを実行して、タグを削除するパッケージグループの ARN と、削除するタグのタグキーを指定します。パッケージグループの ARN については、「パッケージグループ ARN」を参照してください。

例えば、タグキー *key1* と *key2* を使用して、パッケージグループ */npm/scope/anycompany~* の複数のタグを削除する方法は次のとおりです。

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

成功した場合は、コマンドの出力はありません。タグを削除した後、パッケージグループに残っているタグは、`list-tags-for-resource` コマンドを使用して表示できます。

CodeArtifact でドメインを操作する

CodeArtifact の [ドメイン] を使用すると、組織全体で複数のリポジトリを簡単に管理できます。ドメインを使用して、さまざまな AWS アカウントが所有する多くのリポジトリに権限を適用できます。アセットは、複数のリポジトリから利用できる場合でも、ドメインに一度保存されるだけです。

複数のドメインを使用することもできますが、開発チームがパッケージを見つけて共有できるように、公開された公開されたアーティファクトをすべて含むひとつの本稼働ドメインを使用することをお勧めします。二つ目の運用前ドメインを使用して、本稼働ドメインの設定に対する変更をテストできます。

これらのトピックでは、CodeArtifact コンソール、 AWS CLI、 を使用して CodeArtifact ドメイン CloudFormation を作成または設定する方法について説明します。

トピック

ドメインの概要

CodeArtifact で作業している場合、ドメインは次の場合に便利です。

- 重複排除されたストレージ: リポジトリで利用できるアセットが 1 つでも、1,000 個でも、ドメインには一度しか保存する必要がありません。つまり、ストレージ料金は一度しか払わないということです。
- 高速コピー: アップストリームの CodeArtifact リポジトリからダウンストリームにパッケージをプルする場合、または [パッケージバージョンのコピー API] を使う場合、メタデータレコードだけ更新する必要があります。アセットはコピーされません。これにより、ステージングまたはテスト用の新しいリポジトリを迅速にセットアップできます。詳細については、「CodeArtifact でアップストリームリポジトリを操作する」を参照してください。
- リポジトリとチーム間での共有が容易: ドメイン内のすべてのアセットとメタデータは 1 つの AWS KMS key (KMS キー) で暗号化されます。リポジトリごとにキーを管理したり、複数のアカウントにひとつのキーへのアクセス権を付与したりする必要はありません。

- 複数のリポジトリにポリシーを適用する: ドメイン管理者は、ドメイン全体にポリシーを適用できます。これには、ドメイン内のリポジトリにアクセスできるアカウントを制限したり、パッケージのソースとして使用する公開リポジトリへの接続を設定できるアカウントを制限することも含まれます。詳細については、[ドメインポリシー] を参照してください。
- [固有のリポジトリ名]: ドメインはリポジトリのネームスペースを提供します。リポジトリ名は、ドメイン内で固有であれば十分です。わかりやすい意味のある名前を使ってください。

ドメイン名はアカウント内で固有である必要があります。

ドメインがないと、リポジトリを作成することはできません。[リポジトリの作成] API を使用してリポジトリを作成するには、ドメイン名を指定する必要があります。あるドメインから別のドメインにリポジトリを移動することはできません。

リポジトリは、ドメインを所有するのと同じ AWS アカウント、または別のアカウントによって所有できます。所有アカウントが異なる場合は、リポジトリ所有アカウントに`CreateRepository`ドメインリソースに対する権限を与える必要があります。これを行うには、[ドメイン権限ポリシーを加える] コマンドを使用してドメインにリソースポリシーを追加してください。

ひとつの組織が複数のドメインを使用することもできますが、開発チームが組織中でパッケージを見つけて共有できるように、公開されたアーティファクトをすべて含む 1 つの本稼働ドメインを使用することをお勧めします。二つ目の運用前ドメインがあれば、本稼働ドメイン設定の変更をテストするのに便利です。

クロスアカウントドメイン

ドメイン名はアカウント内でのみ固有であれば十分です。つまり、リージョン内に同じ名前のドメインが複数存在することも可能です。このため、認証されていないアカウントが所有するドメインにアクセスする場合は、CLI とコンソールの両方で、ドメイン所有者 ID とドメイン名を提供する必要があります。以下の CLI の例を参照してください。

認証されたアカウントが所有するドメインにアクセス:

認証されたアカウント内のドメインにアクセスする場合は、ドメイン名を指定するだけで済みます。以下の例では、アカウントに所有されている *[my_domain]* というドメインの *[my_repo]* リポジトリ内のパッケージを一覧表示します。

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

認証されていないアカウントが所有するドメインにアクセス:

認証されていないアカウントが所有するドメインにアクセスする場合は、ドメイン所有者とドメイン名を指定する必要があります。以下の例では、認証されていないアカウントが所有する *[other-domain]* ドメインの *[other-repo]* リポジトリ内パッケージを一覧表示します。--domain-owner パラメータが追加されたことに注目してください。

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

CodeArtifact でサポートされている AWS KMS キーのタイプ

CodeArtifact は、対称 KMS キーのみをサポートしています。非対称 KMS キー を使用して CodeArtifact ドメインを暗号化することはできません。詳細については、「Identifying symmetric and asymmetric KMS keys」を参照してください。新しいカスタマーマネージドキーの作成方法については、「AWS Key Management Service Developer Guide」の「Creating symmetric encryption KMS keys」を参照してください。

CodeArtifact は AWS KMS 外部キーストア (XKS) をサポートしています。CodeArtifact の可用性、耐久性、レイテンシーに影響を与える可能性のある XKS キーを使用するキーオペレーションの可用性、耐久性、レイテンシーについては、ユーザーが責任を負います。CodeArtifact で XKS キーを使用する場合の影響のいくつかの例を以下に示します。

- リクエストされたパッケージのすべてのアセットとその依存関係は復号レイテンシーの影響を受けるため、XKS 操作のレイテンシーが増えると構築のレイテンシーが大幅に増加する可能性があります。
- すべてのアセットは CodeArtifact で暗号化されるため、XKS キーマテリアルが失われると、XKS キーを使用するドメインに関連付けられているすべてのアセットが失われます。

XKS キーの詳細については、「AWS Key Management Service Developer Guide」の「External key stores」を参照してください。

ドメインの作成

CodeArtifact コンソール、 AWS Command Line Interface (AWS CLI)、または を使用してドメインを作成できます CloudFormation。ドメインを作成するとき、そのドメインにはリポジトリは含まれていません。詳細については、「リポジトリの作成」を参照してください。CloudFormation を使用して CodeArtifact ドメインを管理する方法の詳細については、「AWS CloudFormation での CodeArtifact リソースの作成」を参照してください。

トピック

ドメイン (コンソール) を作成するには

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで [ドメイン] をクリックし、[Create domain] (ドメインの作成) をクリックします。
3. [名前] にドメイン名を入力します。
4. [Additional configuration] (追加設定) を展開します。
5. (AWS KMS key KMS キー) を使用して、ドメイン内のすべてのアセットを暗号化します。AWS マネージド KMS キーまたは管理する KMS キーを使用することができます。CodeArtifact でサポートされている KMS キーのタイプの詳細については、「CodeArtifact でサポートされている AWS KMS キーのタイプ」を参照してください。

   - デフォルト AWS マネージドキーを使用するには、[AWS マネージドキー] を選択してください。
   - 管理している KMS キーを使用する場合、[カスタマーマネージドキー] を選択してください。[カスタマーマネージドキー ARN] で管理している KMS キーを使用するには、KMS キーを検索して選択します。

   詳細については、[AWS Key Management Service デベロッパーガイド] の AWS マネージドキーと [カスタマーマネージドキー] を参照してください。
6. [ドメインの作成] をクリックします。

ドメイン (AWS CLI) の作成

を使用してドメインを作成するには AWS CLI、 `create-domain` コマンドを使用します。(AWS KMS key KMS キー) を使用して、ドメイン内のすべてのアセットを暗号化する必要があります。AWS マネージド KMS キーまたは管理する KMS キーを使用できます。 AWS マネージド KMS キーを使用する場合は、 `--encryption-key`パラメータを使用しないでください。

CodeArtifact でサポートされている KMS キーのタイプの詳細については、「CodeArtifact でサポートされている AWS KMS キーのタイプ」を参照してください。KMS キーの詳細については、「AWS Key Management Service Developer Guide」の「AWS マネージドキー」および「Customer managed key」を参照してください。

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

JSON 形式のデータが、新しいドメインの詳細とともに出力に表示されます。

```
{
    "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"
    }
}
```

管理する KMS キーを使用する場合は、Amazon リソースネーム (ARN) を`--encryption-key`パラメータに含めてください。

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

JSON 形式のデータが、新しいドメインの詳細とともに出力に表示されます。

```
{
    "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"
    }
```

```
}
```

タグ付きのドメインの作成

タグ付きのドメインを作成するには、`--tags`パラメータを`create-domain`コマンドに追加してください。

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

AWS KMS キーポリシーの例

CodeArtifact でドメインを作成する場合、KMS キーを使用してドメイン内のすべてのアセットを暗号化します。 AWS マネージド KMS キー、またはご自身で管理するカスタマーマネージドキーを選択できます。KMS キーの詳細については、「AWS Key Management Service デベロッパーガイド」を参照してください。

カスタマーマネージドキーを使用するには、CodeArtifact へのアクセスを許可するキーポリシーが KMS キーに必要です。キーポリシーは AWS KMS キーのリソースポリシーであり、KMS キーへのアクセスを制御する主な方法です。すべての KMS キーには、厳密に 1 つのキーポリシーが必要です。キーポリシーのステートメントでは、KMS キーの使用が許可されるユーザーとその使用方法を決定します。

次のキーポリシーステートメントの例では、 AWS CodeArtifact が権限を作成し、承認されたユーザーに代わってキーの詳細を表示することを許可します。このポリシーステートメントは、`kms:ViaService` および `kms:CallerAccount` の条件キーを使用して、指定されたアカウント ID に代わって動作する CodeArtifact に対してのみ許可を限定します。また、IAM ルートユーザーにすべての AWS KMS アクセス許可を付与するため、キーの作成後にキーを管理できます。

JSON

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

ドメインの削除

CodeArtifact コンソールまたは AWS Command Line Interface () を使用してドメインを削除できます AWS CLI。

トピック

- ドメイン削除の制限
- ドメイン (コンソール) を削除するには
- ドメインAWS CLIの削除

ドメイン削除の制限

通常は、リポジトリを含むドメインは削除できません。ドメインを削除する前に、まずリポジトリを削除する必要があります。詳細については、「リポジトリを削除する」を参照してください。

ただし、CodeArtifact がドメインの KMS キーにアクセスできなくなった場合は、リポジトリが含まれている場合でもドメインを削除できます。この状況は、ドメインの KMS キーを削除するか、CodeArtifact がキーにアクセスするために使用する KMS 権限を取り消した場合に発生します。この状態では、ドメイン内のリポジトリやリポジトリに保存されているパッケージにはアクセスできません。CodeArtifact がドメインの KMS キーにアクセスできない場合、リポジトリの一覧表示と削除もできません。このため、ドメインの KMS キーにアクセスできない場合、ドメインの削除ではドメインにリポジトリが含まれているかどうかは確認されません。

> **Note**
>
> リポジトリを含むドメインが削除されると、CodeArtifact は 15 分以内にリポジトリを非同期的に削除します。ドメインが削除されても、リポジトリの自動クリーンアップが実行されるまで、リポジトリは CodeArtifact コンソールと `list-repositories` コマンドの出力に表示されたままになります。

ドメイン (コンソール) を削除するには

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで、[ドメイン] をクリックし、削除するドメインをクリックします。
3. [削除] をクリックします。

ドメインAWS CLIの削除

ドメインを削除するには、delete-domain コマンドを使用します。

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

JSON 形式のデータが、削除されたドメインの詳細とともに出力に表示されます。

```
{
```

```
    "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"
    }
}
```

ドメインポリシー

CodeArtifact は、リソースベースの権限の使用とアクセスのコントロールをサポートしています。リソースベースの権限により、リソースにだれがアクセスでき、そこでどのようなアクションを実行できるかを指定できます。デフォルトでは、ドメインを所有する AWS アカウントのみがドメイン内のリポジトリを作成してアクセスすることができます。ドメインにポリシードキュメントを適用して、他の IAM プリンシパルがそこにアクセスできるように許可を与えることができます。

詳細については、[ポリシーと権限] そして [アイデンティティベースおよびリソースベースのポリシー] を参照してください。

トピック

- ドメインへのクロスアカウントアクセスを有効にする
- ドメインポリシーの例
- を使用したドメインポリシーの例 AWS Organizations
- ドメインポリシーを設定する
- ドメインポリシーを読み取る
- ドメインポリシーを削除する

ドメインへのクロスアカウントアクセスを有効にする

リソースポリシーは、JSON 形式のテキストファイルです。このファイルには、プリンシパル (アクター) とひとつ以上のアクションとエフェクト (`Allow`または`Deny`) を指定する必要があります。別のアカウントが所有するドメインにリポジトリを作成するには、プリンシパルに対して [ドメイン] リソースへの `CreateRepository` 権限を付与する必要があります。

例えば、次のリソースポリシーでは、ドメイン内にリポジトリを作成する 123456789012 権限をアカウントに付与しています。

JSON

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

タグ付きのリポジトリの作成を許可するには、codeartifact:TagResource 権限を含める必要があります。これにより、ドメインとその中のすべてのリポジトリにタグを追加するためのアカウントアクセスも付与されます。

ドメインポリシーは、ドメイン、およびドメイン内のすべてのリソースに対するすべてのオペレーションについて評価されます。つまり、ドメインポリシーを使用して、ドメイン内のリポジトリとパッケージに権限を適用できます。Resource 要素が * に設定されている場合、ステートメントはドメイン内のすべてのリソースに適用されます。例えば、上記のポリシーにおいて、許可された IAM アクションのリストに codeartifact:DescribeRepository も含まれている場合、ポリシーは、ドメイン内のすべてのリポジトリに DescribeRepository の呼び出しを許可します。ドメインポリシーを使用すると、Resource 要素で特定のリソース ARN を使用して、ドメイン内の特定のリソースに権限を適用できます。

> **Note**
>
> 権限の設定には、ドメインポリシーとリポジトリポリシーを両方とも使用できます。両方のポリシーが存在する場合、両方のポリシーが評価され、いずれかのポリシーで許可されて

いればアクションは許可されます。詳細については、「リポジトリポリシーとドメインポリシーの相互作用」を参照してください。

別のアカウントが所有するドメイン内のパッケージにアクセスするには、[ドメインリソース] における GetAuthorizationToken 権限をプリンシパルに付与する必要があります。これにより、ドメイン所有者は、ドメイン内のリポジトリのコンテンツを読み取ることができるアカウントをコントロールできます。

例えば、次のリソースポリシーでは、ドメイン内の任意のリポジトリの認証トークンを取得する123456789012許可をアカウントに付与しています。

JSON

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

> **Note**
>
> リポジトリエンドポイントからパッケージを取得するプリンシパルには、ドメインに対する GetAuthorizationToken 権限に加えてリポジトリリソースに対する ReadFromRepository 権限が付与される必要があります。同様に、リポジトリエンドポイントにパッケージを公開するプリンシパルには、GetAuthorizationTokenに加えてPublishPackageVersionへの権限が必要となります。

ReadFromRepository と PublishPackageVersion への権限の詳細については、[レポジトリーポリシー] を参照してください。

ドメインポリシーの例

複数のアカウントがドメインを使用している場合、ドメインを完全に使用できるようにするには、そのアカウントに基本的な権限セットを付与する必要があります。次のリソースポリシーは、ドメインの完全な使用を許可する一連の権限を一覧表示します。

JSON

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

Note

ドメインとそのリポジトリすべてが単一のアカウントによって所有され、そのアカウントからのみ使用する必要がある場合は、ドメインポリシーを作成する必要はありません。

を使用したドメインポリシーの例 AWS Organizations

aws:PrincipalOrgID条件キーを使って、以下のように、組織内のすべてのアカウントから CodeArtifact ドメインへアクセスする許可を付与することができます。

JSON

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

aws:PrincipalOrgID 条件キーの使用についての詳細は、[IAM ユーザーガイド] の [AWS グローバル条件コンテキストキー] を参照してください。

ドメインポリシーを設定する

put-domain-permissions-policyコマンドを使用して、ドメインにポリシーをアタッチすることできます。

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

put-domains-permissions-policyを呼び出すとき、権限を評価する場合は、ドメインのリソースポリシーは無視されます。これにより、ドメインの所有者がドメインから自分自身をロックアウトできなくなり、リソースポリシーを更新できなくなることを防ぎます。

> **Note**
>
> put-domain-permissions-policy を呼び出すときにリソースポリシーが無視されるため、リソースポリシーを使用してドメインのリソースポリシーを更新するアクセス許可を別の AWS アカウントに付与することはできません。

サンプル出力:

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

コマンドの出力には、ドメインリソースの Amazon リソースネーム (ARN)、ポリシードキュメントの完全な内容、リビジョン識別子が含まれます。リビジョン識別子は--policy-revisionオプションを使用してput-domain-permissions-policyにパスできます。これにより、別のライターが設定した新しいバージョンではなく、既知のリビジョンのドキュメントが確実に上書きされます。

ドメインポリシーを読み取る

ポリシードキュメントの既存のバージョンを読み取るには、get-domain-permissions-policyコマンドを使用します。読みやすいように出力をフォーマットするには、--outputおよび--query policy.documentを Python json.toolモジュールと共に、次の通りに使用してください。

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

サンプル出力:

JSON

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

ドメインポリシーを削除する

delete-domain-permissions-policyコマンドを使用して、ドメインからポリシーを削除します。

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

出力の形式は、get-domain-permissions-policyおよびdelete-domain-permissions-policyコマンドの形式と同様になります。

CodeArtifact でドメインをタグ付けする

タグは、AWS リソースに関連付けられるキーと値のペアです。CodeArtifact では、ドメインにタグを適用することができます。CodeArtifact リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプについては、リソースのタグ付けを参照してください。

ドメインを作成する際、 CLI を使用してタグを指定できます。コンソールまたは CLI を使用して、ドメインにおけるタグの追加または削除、そしてドメインのタグの値を更新することができます。ドメインごとに最大 50 個のタグを追加できます。

トピック

ドメインにタグ付けする (CLI)

CLI を使用して、ドメインのタグを管理できます。

トピック

ドメインにタグを追加する (CLI)

コンソールまたは を使用して、ドメイン AWS CLI にタグを付けることができます。

ドメインの作成時にタグを追加するには、「リポジトリの作成」を参照してください。

以下のステップでは、 AWS CLI の最新版をすでにインストールしているか、最新版に更新しているものとします。詳細については、[AWS Command Line Interfaceのインストール]」を参照してください。

ターミナルまたはコマンドラインで、tag-resource コマンドを実行して、タグを追加するドメインの Amazon リソースネーム (ARN)を指定し、追加するタグキーとタグ値を指定します。

> **Note**
>
> ドメインの ARN を取得するには、`describe-domain` コマンドを実行します。
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

ドメインには複数のタグを追加できます。例えば、*my_domain* という名前のドメインに 2 つのタグを付けます。*value1* のタグ値がある *key1* という名前のタグキーと、*value2* のタグ値がある *key2* という名前のタグキーです。

```
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
```

成功した場合は、コマンドの出力はありません。

ドメインのタグを表示する (CLI)

を使用してドメインの AWS タグ AWS CLI を表示するには、次の手順に従います。タグが追加されていない場合、返されるリストは空になります。

ターミナルまたはコマンドラインで、ドメインの Amazon リソースネーム (ARN)というlist-tags-for-resourceコマンドを実行します。

> **Note**
>
> ドメインの ARN を取得するには、describe-domainコマンドを実行します:
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

例えば、arn:aws:codeartifact:*us-west-2*:*123456789012*:domain/*my_domain* ARN の値を持った *[my_domain]* というドメインのタグキーとタグ値のリストを表示するには、次のように入力します。

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

成功した場合、このコマンドは次のような情報を返します。

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

ドメインのタグを表示する (CLI)

を使用してドメインのタグ AWS CLI を編集するには、次の手順に従います。既存のキーの値を変更したり、別のキーを追加できます。次のセクションに示すように、ドメインからタグを削除することもできます。

ターミナルまたはコマンドラインで、tag-resource コマンドを実行して、タグを更新するドメインの ARN を指定し、タグキーとタグ値を指定します。

> **Note**
>
> ドメインの ARN を取得するには、describe-domainコマンドを実行してください:
>
> ```
> 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
```

成功した場合は、コマンドの出力はありません。

ドメインからタグを削除する (CLI)

を使用してドメインからタグ AWS CLI を削除するには、次の手順に従います。

> **Note**
>
> ドメインを削除すると、削除されたドメインからすべてのタグの関連付けが削除されます。ドメインを削除する前にタグを削除する必要はありません。

ターミナルまたはコマンドラインで、untag-resource コマンドを実行して、タグを削除するドメインの ARN と削除するタグのタグキーを指定します。

> **Note**
>
> ドメインの ARN を取得するには、describe-domainコマンドを実行してください :

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

例えば、*key1* および *key2* という名前のタグキーのある、*mydomain* という名前のドメインで複数のタグを削除するには、次を行います。

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

成功した場合は、コマンドの出力はありません。タグを削除した後、ドメインの残りのタグは、`list-tags-for-resource` コマンドを使用して確認できます。

ドメインにタグ付けする (コンソール)

コンソールまたは CLI を使用して、リソースのタグ付けをします。

トピック

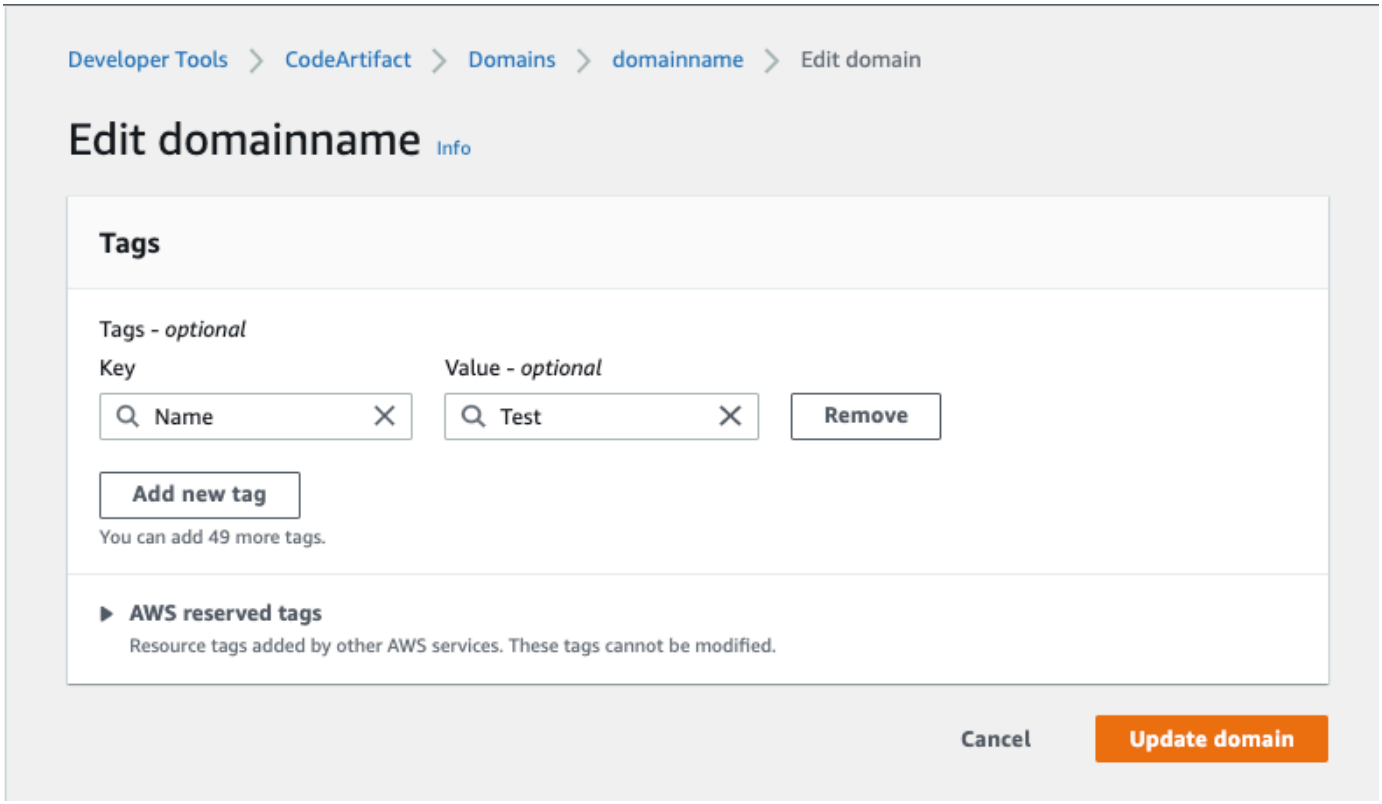
- ドメインにタグを追加する (コンソール)
- ドメインのタグを表示する (コンソール)
- ドメインのタグを表示する (コンソール)
- ドメインからタグを削除する (コンソール)

ドメインにタグを追加する (コンソール)

コンソールを使用して既存のドメインにタグを追加します。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. [ドメイン] ページで、タグを追加するドメインをクリックします。
3. [詳細] セクションを展開します。
4. [ドメインタグ] で、ドメインにタグがない場合は、[ドメインタグを追加する] をクリックするか、もしあれば [ドメインタグの表示と編集] をクリックします。
5. [新しいタグを追加] をクリックします。

6. [キー] フィールドと [値] フィールドに、追加するタグごとにテキストを入力します。([値] フィールドはオプションです。) 例えば、[キー] では、「**Name**」と入力します。[値] には「**Test**」と入力します。



7. (オプション) [タグの追加] をクリックして行を追加し、さらにタグを入力します。
8. [ドメインの更新] をクリックします。

ドメインのタグを表示する (コンソール)

コンソールを使用して既存のドメインのタグを一覧表示します。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. [ドメイン] ページで、タグを表示するドメインをクリックします。
3. [詳細] のセクションを展開します。
4. [ドメインタグ] で、[ドメインタグの表示と編集] をクリックします。

> **Note**
>
> このドメインにタグが追加されていない場合、コンソールは [ドメインタグの追加] を読み取ります。

ドメインのタグを表示する (コンソール)

コンソールを使用してドメインに追加されたタグを編集します。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. [ドメイン] ページで、タグを更新するドメインをクリックします。
3. [詳細] のセクションを展開します。
4. [ドメインタグ] で、[ドメインタグの表示と編集] をクリックします。

> **Note**
>
> このドメインにタグが追加されていない場合、コンソールは [ドメインタグの追加] を読み取ります。

5. [キー] フィールドと [値] フィールドで、必要に応じて各フィールドの値を更新します。例えば、**Name** キーの場合は、[値] で、**Test** を **Prod** に変更します。
6. [ドメインの更新] をクリックします。

ドメインからタグを削除する (コンソール)

コンソールを使用してパイプラインからタグを削除できます。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. [ドメイン] ページで、タグを削除するドメインをクリックします。
3. [詳細] のセクションを展開します。
4. [ドメインタグ] で、[ドメインタグの表示と編集] をクリックします。

> **Note**
>
> このドメインにタグが追加されていない場合、コンソールは [ドメインタグを追加] を読み取ります。

5. 削除する各タグのキーと値の横にある [削除] をクリックします。
6. [ドメインの更新] をクリックします。

Cargo で CodeArtifact を使用する

以下のトピックでは、Rust パッケージマネージャーである Cargo を CodeArtifact で使用する方法について説明します。

> **Note**
>
> CodeArtifact は Cargo 1.74.0 以降のみをサポートしています。Cargo 1.74.0 が、CodeArtifact リポジトリでの認証をサポートする最も古いバージョンです。

トピック

CodeArtifact で Cargo を設定して使用する

Cargo を使用して、CodeArtifact リポジトリからクレートを公開およびダウンロードしたり、Rust コミュニティのクレートレジストリである crates.io からクレートを取得したりできます。このトピックでは、CodeArtifact リポジトリの認証と使用のために Cargo を設定する方法について説明します。

CodeArtifact で Cargo を設定する

Cargo を使用して AWS CodeArtifact からクレートをインストールして公開するには、まず CodeArtifact リポジトリ情報を使用してクレートを設定する必要があります。次のいずれかの手順に沿って、CodeArtifact リポジトリのエンドポイント情報と認証情報を使用して Cargo を設定します。

コンソールの手順を使用して Cargo を設定する

コンソールの設定手順を使用して、Cargo を CodeArtifact リポジトリに接続できます。コンソールの手順では、お使いの CodeArtifact リポジトリ用にカスタマイズされた Cargo 設定を提供します。このカスタム設定を使用すれば、CodeArtifact 情報を検索して入力する必要なく、Cargo をセットアップできます。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。
2. ナビゲーションペインで、[リポジトリ] を選択し、Cargo に接続するリポジトリを選びます。
3. [接続手順の表示] を選択します。
4. オペレーティングシステムを選択します。
5. [Cargo] を選択します。
6. 生成された手順に沿って、Cargo を CodeArtifact リポジトリに接続します。

手動で Cargo を設定する

コンソールの設定手順を使用できない場合、または使用しない場合は、次の手順を使用して、手動で Cargo を CodeArtifact リポジトリに接続できます。

macOS and Linux

CodeArtifact で Cargo を設定するには、CodeArtifact リポジトリを Cargo 設定のレジストリとして定義し、認証情報を指定する必要があります。

- *my_registry* をレジストリ名に置き換えます。
- *my_domain* を CodeArtifact ドメイン名に置き換えます。
- *111122223333* をAWS ドメインの所有者のアカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合は、`--domain-owner` を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
- *my_repo* を CodeArtifact リポジトリ名で置き換えます。

設定をコピーして、Cargo パッケージをリポジトリに公開してダウンロードし、`~/.cargo/config.toml` ファイル (システム単位の設定の場合) または `.cargo/config.toml` (プロジェクト単位の設定の場合) に保存します。

```
[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

CodeArtifact で Cargo を設定するには、CodeArtifact リポジトリを Cargo 設定のレジストリとして定義し、認証情報を指定する必要があります。

- *my_registry* をレジストリ名に置き換えます。
- *my_domain* を CodeArtifact ドメイン名に置き換えます。
- *111122223333* をAWS ドメインの所有者のアカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合は、--domain-owner を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
- *my_repo* を CodeArtifact リポジトリ名で置き換えます。

設定をコピーして、リポジトリから Cargo パッケージのみをダウンロードし、%USERPROFILE%\.cargo\config.toml ファイル (システム単位の設定の場合) または .cargo\config.toml (プロジェクト単位の設定の場合) に保存します。

```
[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. CodeArtifact で Cargo を設定するには、CodeArtifact リポジトリを Cargo 設定のレジストリとして定義し、認証情報を指定する必要があります。

- *my_registry* をレジストリ名に置き換えます。
- *my_domain* を CodeArtifact ドメイン名に置き換えます。
- *111122223333* をAWS ドメインの所有者のアカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合は、--domain-owner を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
- *my_repo* を CodeArtifact リポジトリ名で置き換えます。

設定をコピーして、Cargo パッケージをリポジトリに公開してダウンロードし、%USERPROFILE%\.cargo\config.toml ファイル (システム単位の設定の場合) または .cargo\config.toml (プロジェクト単位の設定の場合) に保存します。

~/.cargo/credentials.toml ファイルに保存されている認証情報を使用する、認証情報プロバイダー cargo:token を使用することをお勧めします。Cargo クライアントが cargo publish 中に認可トークンを適切にトリミングしないため、cargo:token-from-stdout を使用すると cargo publish 中にエラーが発生する可能性があります。

```
[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. Windows でリポジトリに Cargo パッケージを公開するには、CodeArtifact get-authorization-token コマンドと Cargo login コマンドを使用して認可トークンと認証情報を取得する必要があります。

   - *my_registry* を、[registries.*my_registry*] で定義されているレジストリ名に置き換えます。
   - *my_domain* を CodeArtifact ドメイン名に置き換えます。
   - *111122223333* をAWS ドメインの所有者のアカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合は、--domain-owner を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

```
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**
>
> 認可トークンの有効期限は 12 時間です。トークンの作成後 12 時間が経過した場合は、新しいトークンを作成する必要があります。

上記の例の [registries.*my_registry*] セクションでは、*my_registry* を使用してレジストリを定義し、index および credential-provider の情報を提供します。

- index は、レジストリのインデックスの URL を指定します。これは、/ で終わる CodeArtifact リポジトリのエンドポイントです。sparse+ プレフィックスは、Git リポジトリではないレジストリに必要です。

  > **Note**
  >
  > デュアルスタックエンドポイントを使用するには、codeartifact.*region*.on.aws エンドポイントを使用してください。

- credential-provider は、指定レジストリの認証情報プロバイダーを指定します。credential-provider が設定されていない場合は、registry.global-credential-providers のプロバイダーが使用されます。credential-provider を cargo:token-from-stdout に設定することで、Cargo クライアントは、CodeArtifact リポジトリから公開またはダウンロードするときに新しい認可トークンを自動的に取得するため、12 時間ごとに認可トークンを手動で更新する必要はありません。

[registry] セクションは、使用するデフォルトのレジストリを定義します。

- default は、CodeArtifact リポジトリから公開またはダウンロードするときにデフォルトで使用する、[registries.*my_registry*] で定義されたレジストリの名前を指定します。

[source.crates-io] セクションは、レジストリが指定されていない場合に使用する、デフォルトのレジストリを定義します。

- replace-with = "*my_registry*" は、パブリックレジストリ crates.io を、[registries.*my_registry*] で定義された CodeArtifact リポジトリに置き換えます。crates.io などの外部接続からパッケージをリクエストする必要がある場合、この設定が推奨されます。

  依存関係撹乱攻撃を防ぐパッケージオリジンコントロールなど、CodeArtifact の利点を最大限に得るには、ソース置換を使用することが推奨されます。ソースを置換する場合、CodeArtifact は、外部接続へのすべてのリクエストをプロキシし、外部接続からリポジトリにパッケージをコピーします。ソースを置換しない場合、Cargo クライアントはプロジェクトの Cargo.toml ファイルの設定に基づいてパッケージを直接取得します。依存関係が registry=*my_registry* でマークされていない場合、Cargo クライアントは CodeArtifact リポジトリと通信せずに crates.io から直接取得します。

  > **Note**
  >
  > ソース置換の使用を開始してから、ソース置換を使用しないように設定ファイルを更新すると、エラーが発生する可能性があります。また、逆のシナリオでもエラーが発生する可能性があります。そのため、プロジェクトの設定は変更しないことをお勧めします。

Cargo クレートのインストール

次の手順に沿って、CodeArtifact リポジトリまたは crates.io から Cargo クレートをインストールします。

CodeArtifact から Cargo クレートをインストールする

Cargo (cargo) CLI を使用すると、CodeArtifact リポジトリから特定のバージョンの Cargo クレートをすばやくインストールできます。

cargo を使用して CodeArtifact リポジトリから Cargo クレートをインストールする方法

1. まだ cargo CLI を設定していない場合は、「CodeArtifact で Cargo を設定して使用する」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して設定します。
2. 次のコマンドを使用して、CodeArtifact から Cargo クレートをインストールします。

```
cargo add my_cargo_package@1.0.0
```

詳細については、「The Cargo Book」の「cargo add」を参照してください。

CodeArtifact への Cargo クレートの公開

`cargo` CLI を使用して Cargo クレートを CodeArtifact リポジトリに公開するには、次の手順に従います。

1. まだ `cargo` CLI を設定していない場合は、「CodeArtifact で Cargo を設定して使用する」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して設定します。
2. 次のコマンドを使用して、Cargo を CodeArtifact リポジトリに公開します。

```
cargo publish
```

詳細については、「The Cargo Book」の「cargo publish」を参照してください。

Cargo コマンドのサポート

以下のセクションでは、CodeArtifact リポジトリでサポートされている Cargo コマンドと、サポートされていない特定のコマンドについてまとめます。

目次

レジストリへのアクセスを必要とするサポートされたコマンド

このセクションでは、Cargo クライアントが設定されたレジストリへのアクセスを必要とする Cargo コマンドをリストアップします。これらのコマンドは、CodeArtifactリポジトリに対して呼び出されたときに正しく機能することが確認されています。

コマンド	説明
buid (構築)	ローカルパッケージとその依存関係をビルドします。
チェック	ローカルパッケージとその依存関係にエラーがないか確認します。
フェッチ	パッケージの依存関係を取得します。
publish (出力)	パッケージをレジストリに公開します。

サポートされていないコマンド

以下の Cargo コマンドは、CodeArtifact リポジトリではサポートされていません。

コマンド	説明
owner (オーナー)	レジストリ上のクレートの所有者を管理します。
search (検索)	レジストリ内のパッケージを検索します。

MavenでCodeArtifactを使う

Mavenリポジトリ形式は Java、Kotlin、Scala、Clojureなど、さまざまな言語で使用されています。Maven、Gradle、Scala SBT、Apache Ivy、Leiningenなど、さまざまなビルドツールでサポートされています。

以下のバージョンで CodeArtifact との互換性をテストし、確認しました。

- 最新の Maven バージョン: 3.6.3。
- 最新の Gradle バージョン: 6.4.1、 5.5.1 もテスト済み。
- 最新の Clojure バージョン: 1.11.1 もテスト済み。

トピック

Gradle で CodeArtifact を使用する

環境変数に CodeArtifact 認証トークンを取得した後、環境変数を使用して認証トークンを渡す の指示に従って、CodeArtifact リポジトリから Maven パッケージを使用し、新しいパッケージを CodeArtifact リポジトリに公開します。

トピック

依存関係の取得

Gradle ビルドの CodeArtifact から依存関係を取得するには、次の手順に従います。

Gradle ビルドの CodeArtifact から依存関係を取得するには

1. まだ実行していない場合は、「環境変数を使用して認証トークンを渡す」の手順に従って CodeArtifact 認証トークンを作成して環境変数に保存します。
2. プロジェクトファイル build.gradle の repositories セクションに maven セクションを追加します。

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

上記の例の url は、CodeArtifact リポジトリのエンドポイントです。Gradle は、エンドポイントを使用してリポジトリに接続します。サンプルでは、my_domain はドメインの名前、111122223333 はドメインの所有者の ID 、そして my_repo はリポジトリの名前です。get-repository-endpoint AWS CLI コマンドを使用して、リポジトリのエンドポイントを取得できます。

例えば、*my_domain* という名前の ドメイン内の*repo* という名前のリポジトリでは、コマンドは次のとおりです。

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

get-repository-endpoint コマンドはリポジトリエンドポイントを返します。

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

上記の例の credentials オブジェクトには、手順 1 で作成した CodeArtifact 認証トークンが含まれています。Gradle は このトークンを CodeArtifact の認証に使用します。

> **Note**
>
> デュアルスタックエンドポイントを使用するには、codeartifact.*region*.on.aws エンドポイントを使用してください。

3. (オプション) CodeArtifact リポジトリをプロジェクトの依存関係の唯一のソースとして使用するには、build.gradle から repositories 内の他のセクションを削除します。複数のリポジトリがある場合、Gradle はリストされている順序で各リポジトリの依存関係を検索します。
4. リポジトリを構成したら、プロジェクトの依存関係を標準の Gradle 構文で dependencies セクションに追加できます。

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

プラグインの取得

デフォルトでは、Gradle はパブリック Gradle Plugin Portal からプラグインを解決します。CodeArtifact リポジトリからプラグインをプルするには、次の手順に従います。

CodeArtifact リポジトリからプラグインをプルするには

1. まだ実行していない場合は、「環境変数を使用して認証トークンを渡す」の手順に従って CodeArtifact 認証トークンを作成して環境変数に保存します。
2. pluginManagement ブロックを settings.gradle ファイルに追加します。pluginManagement ブロックは、settings.gradle の他のステートメントの前に置く必要があります。次のスニペットを参照してください。

```
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
            }
        }
    }
}
```

これにより、Gradle は指定したリポジトリからプラグインを解決します。一般的に必要な Gradle プラグインをビルドで使用できるように、リポジトリには Gradle Plugin Portal への外部接続を持つ上流リポジトリが必要です (例: `gradle-plugins-store`) 。詳細については、Gradle ドキュメントを参照してください。

アーティファクトの公開

このセクションでは、Gradle でビルドされた Java ライブラリを CodeArtifact リポジトリに公開する方法について説明します。

まず、`maven-publish` プラグインをプロジェクトの `build.gradle` ファイルの `plugins` セクションに追加します。

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

次に、`publishing` セクションをプロジェクト `build.gradle` ファイルに追加します。

```
publishing {
    publications {
        mavenJava(MavenPublication) {
            groupId = 'group-id'
            artifactId = 'artifact-id'
            version = 'version'
            from components.java
        }
    }
    repositories {
        maven {
            url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
```

```
            credentials {
                username "aws"
                password System.env.CODEARTIFACT_AUTH_TOKEN
            }
        }
    }
}
```

maven-publish プラグインは、publishing セクションで指定された groupId、artifactId および version に基づいて POM ファイルを生成します。

これらの build.gradle への変更が完了したら、次のコマンドを実行してプロジェクトをビルドし、それをリポジトリにアップロードします。

```
./gradlew publish
```

list-package-versionsを使用して、パッケージが正常に発行されたことを確認します。

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

サンプル出力:

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

詳細については、Gradle ウェブサイトで以下のトピックを参照してください。

- Java ライブラリの構築

- プロジェクトをモジュールとして公開する

IntelliJ IDEAで Gradle ビルドを実行する

IntelliJ IDEAで、CodeArtifact から依存関係をプルする Gradle ビルドを実行できます。CodeArtifact で認証するには、CodeArtifact 認可トークンを Gradle に提供する必要があります。認証トークンを提供する方法は 3 つあります。

- 方法 1: `gradle.properties` に認証トークンを保存する。この方法は、`gradle.properties` ファイルのコンテンツに上書きまたは追加できる場合に使用します。
- 方法 2: 認証トークンを別のファイルに保存する。この方法は、`gradle.properties` ファイルを修正したくない場合に使用します。
- 方法 3: `aws` を `build.gradle` のインラインスクリプトとして実行して、実行ごとに新しい認証トークンを生成する。この方法は、実行ごとに Gradle スクリプトが新しいトークンを取得するようにしたい場合に使用します。トークンはファイルシステムに保存されません。

Token stored in gradle.properties

方法 1: **`gradle.properties`** に認証トークンを保存する。

> **Note**
>
> この例は `GRADLE_USER_HOME` にある `gradle.properties` ファイルを示します。

1. 次のスニペットを使用して `build.gradle` ファイルを更新する:

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

2. CodeArtifact からプラグインをフェッチするには、settings.gradle ファイルに pluginManagement ブロックを追加します。pluginManagement ブロックは、settings.gradle の他のステートメントの前に置く必要があります。

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

3. CodeArtifact 認証トークンを取得します。

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

4. gradle.properties ファイルに認証トークンを書き込む:

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

Token stored in separate file

方法 2: 認証トークンを別のファイルに保存する

1. 次のスニペットを使用して build.gradle ファイルを更新する:

```
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. CodeArtifact からプラグインをフェッチするには、settings.gradle ファイルに pluginManagement ブロックを追加します。pluginManagement ブロックは、settings.gradle の他のステートメントの前に置く必要があります。

```
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. CodeArtifact 認証トークンを取得します。

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

4. build.gradle ファイルで指定したファイルに認証トークンを書き込みます。

```
echo "codeartifactToken=$CODEARTIFACT_AUTH_TOKEN" > file
```

Token generated for each run in build.gradle

方法 3: **aws** を **build.gradle** のインラインスクリプトとして実行して、実行ごとに新しい認証トークンを生成する。

1. 次のスニペットを使用して build.gradle ファイルを更新する:

```
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. CodeArtifact からプラグインをフェッチするには、settings.gradle ファイルに pluginManagement ブロックを追加します。pluginManagement ブロックは、settings.gradle の他のステートメントの前に置く必要があります。

```
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
            }
        }
    }
```

```
}
```

mvn で CodeArtifact を使用する

Maven ビルドを実行するには、mvn コマンドを使用してください。このセクションでは、CodeArtifact リポジトリを使用して mvn を設定する方法を示します。

トピック

- 依存関係の取得
- アーティファクトの公開
- サードパーティのアーティファクト
- CodeArtifact リポジトリへの Maven 依存関係のダウンロードを制限する
- Apache Maven プロジェクト情報

依存関係の取得

mvnを設定して CodeArtifact リポジトリから依存関係を取得するには、Maven 設定ファイル settings.xml、オプションで、プロジェクトの POM を編集する必要があります。。

1. まだ行っていない場合は、CodeArtifact リポジトリへの認証を設定する「環境変数を使用して認証トークンを渡す」の説明に従って、CodeArtifact 認証トークンを作成して環境変数に保存します。
2. settings.xml (通常、~/.m2/settings.xmlの場所にあります。) で、CODEARTIFACT_AUTH_TOKEN 環境変数を参照して <servers> セクションを追加し、HTTP リクエストで Maven が トークンを渡すようにします。

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

```
</settings>
```

3. <repository> 要素にある CodeArtifact リポジトリの URL エンドポイントを追加します。これは、settings.xmlまたはプロジェクトの POM ファイルで行えます。

   リポジトリのエンドポイントを取得するには、 get-repository-endpoint AWS CLI コマンドを使用します。

   例えば、*my_domain* という名前のドメイン内の *repo* という名前のリポジトリのコマンドは次のとおりです。

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

   get-repository-endpoint コマンドはリポジトリエンドポイントを返します。

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

   > **Note**
   >
   > デュアルスタックエンドポイントを使用するには、codeartifact.*region*.on.aws エンドポイントを使用してください。

   settings.xml へのリポジトリエンドポイントの追加は以下のとおりです。

   ```
   <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>
```

または、<repositories> セクションをプロジェクトの POM ファイルに追加して、そのプロジェクトに対してのみ CodeArtifact を使用できます。

```
<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**
>
> <id> 要素にある任意の値を使用できますが、<server> および <repository> 要素の両方で同じでなければなりません。これにより、指定された認証情報を CodeArtifact へのリクエストに含めることができます。

これらの設定変更を行った後、プロジェクトを構築できます。

```
mvn compile
```

Maven は、ダウンロードするすべての依存関係の完全な URL をコンソールに記録します。

```
[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)
```

アーティファクトの公開

mvn で Maven アーティファクトを CodeArtifact リポジトリに公開するには、~/.m2/settings.xml およびプロジェクト POM を編集する必要もあります。

1. まだ行っていない場合は、CodeArtifact リポジトリへの認証を設定する「環境変数を使用して認証トークンを渡す」の説明に従って、CodeArtifact 認証トークンを作成して環境変数に保存します。
2. CODEARTIFACT_AUTH_TOKEN 環境変数を参照して <servers> セクションを settings.xml に追加し、Maven が HTTP リクエストで トークンを渡すようにします。

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

3. <distributionManagement> セクションをプロジェクトの pom.xml に追加します。

   ```
   <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>
```

これらの設定変更を行った後、プロジェクトを構築して指定したリポジトリに公開できます。

```
mvn deploy
```

list-package-versions を使用して、パッケージが正常に公開されたことを確認します。

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

サンプル出力:

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

サードパーティのアーティファクト

`mvn deploy:deploy-file` を使用して、サードパーティ Maven アーティファクトを CodeArtifact リポジトリに公開できます。これは、アーティファクトを公開し、JAR ファイルのみを持ち、パッケージソースコードや POM ファイルにアクセスできないユーザーに役立ちます。

`mvn deploy:deploy-file` コマンド は、コマンドラインで渡された情報に基づいて POM ファイルを生成します。

サードパーティの Maven アーティファクトを公開する

1. まだ行っていない場合は、CodeArtifact リポジトリへの認証を設定する「環境変数を使用して認証トークンを渡す」の説明に従って、CodeArtifact 認証トークンを作成して環境変数に保存します。
2. 次のコンテンツを含む `~/.m2/settings.xml` ファイルを作成します。

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

3. `mvn deploy:deploy-file` コマンドを実行します。

```
mvn deploy:deploy-file -DgroupId=commons-cli          \
-DartifactId=commons-cli       \
-Dversion=1.4                  \
-Dfile=./commons-cli-1.4.jar   \
-Dpackaging=jar                \
-DrepositoryId=codeartifact    \
-Durl=https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/repo-name/
```

> **Note**
>
> 上記の例は commons-cli 1.4 を公開しています。groupId、artifactID、version、およびファイルの引数を変更して、別の JAR を公開します。

この手順は、Apache Maven ドキュメント の サードパーティの JAR をリモートリポジトリにデプロイするためのガイド の例に基づいています。

CodeArtifact リポジトリへの Maven 依存関係のダウンロードを制限する

設定されたリポジトリからパッケージを取得できない場合、デフォルトでmvnコマンドでMaven Centralからパッケージを取得します。mirrors要素をsettings.xmlに追加して、mvnが常にCodeArtifact リポジトリを使用するようにします。

```
<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>
```

mirrors要素を追加すると、settings.xmlまたはpom.xmlにpluginRepository要素が含まれる必要があります。次の例では、CodeArtifact リポジトリからアプリケーションの依存関係とMaven プラグインを取得します。

```
<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>
```

次の例では、CodeArtifact リポジトリからアプリケーションの依存関係を取得し、Maven CentralからMaven プラグインを取得します。

```
<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 プロジェクト情報

Maven の詳細については、Apache Maven プロジェクトウェブサイトの以下のトピックを参照してください。

- 複数のリポジトリの設定
- 設定リファレンス
- ディストリビューション管理
- プロファイル

CodeArtifact で deps.edn を使用する

`deps.edn` と `clj` を使用して、Clojure プロジェクトの依存関係を管理します。このセクションでは、CodeArtifact リポジトリを使用して `deps.edn` を設定する方法を示します。

トピック

- 依存関係の取得
- アーティファクトの公開

依存関係の取得

`Clojure` を設定して CodeArtifact リポジトリから依存関係を取得するには、Maven 設定ファイル `settings.xml` を編集する必要があります。

1. `CODEARTIFACT_AUTH_TOKEN` 環境変数を参照して `<servers>` セクションを `settings.xml` に追加し、Clojure が HTTP リクエストでトークンを渡すようにします。

> **Note**
>
> Clojure は settings.xml ファイルが `~/.m2/settings.xml` にあることを想定しています。他の場所にある場合は、この場所にファイルを作成してください。

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

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

2. ファイルを作成していない場合は、`clj -Spom` を使用してプロジェクト用の POM xml を生成します。
3. deps.edn 設定ファイルに、Maven `settings.xml` のサーバー ID と一致するリポジトリを追加します。

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

> **Note**
>
> - `clojars` によって、Maven ライブラリーの `tools.deps` と `central` リポジトリが最初にチェックされることが保証されます。その後、`deps.edn` に記載されている他のリポジトリがチェックされます。
> - Clojars と Maven Central から直接ダウンロードされないようにするには、`central` と `clojars` を `nil` に設定する必要があります。

CodeArtifact Auth トークンが環境変数に含まれていることを確認してください (「環境変数を使用して認証トークンを渡す」を参照)。これらの変更後にパッケージを構築する際、`deps.edn` の依存関係が CodeArtifact から取得されます。

> **Note**
>
> デュアルスタックエンドポイントを使用するには、`codeartifact.region.on.aws` エンドポイントを使用してください。

アーティファクトの公開

1. Maven 設定 と deps.edn を更新して、CodeArtifact を Maven で認識されるサーバーとして含めるようにします (「依存関係の取得」を参照)。deps-deploy などのツールを使用して、アーティファクトを CodeArtifact にアップロードできます。
2. build.clj に、以前に設定した codeartifact リポジトリに必要なアーティファクトをアップロードする deploy タスクを追加します。

```
(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. clj -T:build deploy コマンドを実行してアーティファクトを公開します。

デフォルトリポジトリの変更に関する詳細は、「Clojure Deps and CLI Reference Rationale」の「Modifying the default repositories」を参照してください。

curl で公開する

このセクションでは、HTTP クライアント curl を使用して、Maven Artifact (アーティファクト) を CodeArtifact リポジトリに公開する方法を説明します。curl を使用したArtifact (アーティファクト) の公開は、Maven クライアントを環境にインストールしていない、またはインストールしたくない場合に便利です。

curl を使用して Maven Artifact (アーティファクト) を公開する

1. CodeArtifact 認可トークンを取得するには、「環境変数を使用して認証トークンを渡す」のステップに従います。その後、これらのステップに戻ります。
2. 次の curl コマンドを使用して、JAR を CodeArtifact リポジトリに公開します。

   この手順の各 curl コマンドで、次のプレースホルダを置き換えます。

   - *my_domain* を CodeArtifact ドメイン名に置き換えます。
   - *111122223333* を CodeArtifact ドメインの所有者の ID に置き換えます。

- *us-west-2* を CodeArtifact ドメインがあるリージョンに置き換えます。
- *my_repo* を CodeArtifact リポジトリ名で置き換えます。

```
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**
>
> `--data-binary` パラメータの値には `@` 文字をプレフィックスとして付ける必要があります。値を引用符で囲む場合は、`@` を引用符で囲む必要があります。

3. 次の `curl` コマンドを使用して、POM を CodeArtifact リポジトリに公開します。

```
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. この時点で、Maven Artifact (アーティファクト) は `Unfinished` のステータスで CodeArtifact リポジトリにあります。パッケージを消費できるようにするには、パッケージが `Published` のステータスである必要があります。パッケージを `Unfinished` から `Published` に移動するには、`maven-metadata.xml` ファイルをパッケージにアップロードするか、[UpdatePackageVersionsStatus API] を呼び出して、ステータスを変更します。

   a. オプション 1: 次の `curl` コマンドを使用して、`maven-metadata.xml` ファイルをパッケージに追加する:

```
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
```

次に示すのは、maven-metadata.xml ファイルのコンテンツの例です:

```
<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. オプション 2: UpdatePackageVersionsStatus API を使用して、パッケージのステータスを Published に更新する。

```
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
```

Artifact (アーティファクト) の JAR ファイルしかない場合は、mvn を使用して使用可能なパッケージ版を CodeArtifact リポジトリに公開できます。これは、Artifact (アーティファクト) のソースコードまたは POM にアクセスできない場合に便利です。詳細については、「サードパーティのアーティファクト」を参照してください。

Maven チェックサムの使用

Maven アーティファクトが AWS CodeArtifact リポジトリに発行されると、パッケージ内の各アセットまたはファイルに関連付けられたチェックサムを使用してアップロードを検証します。アセットの例は、[jar]、[pom] および [war] ファイルです。各アセットについて、Maven Artifact (アーティファクト) には、md5 または sha1 など、アセット名に追加の拡張子が付いた複数のチェックサムファ

イルが含まれています。例えば、my-maven-package.jar という名前のファイルのチェックサムファイルは my-maven-package.jar.md5 および my-maven-package.jar.sha1 である可能性があります。

> **Note**
>
> Maven はこの条件 artifact を使用します。このガイドでは、Maven パッケージは Maven Artifact (アーティファクト) と同じです。詳細については、[AWS CodeArtifact パッケージ] を参照してください。

チェックサムストレージ

CodeArtifact は Maven チェックサムをアセットとして保存しません。つまり、ListPackageVersionAssets API の出力では、チェックサムは個別のアセットとして表示されません。代わりに、CodeArtifact によって計算されたチェックサムは、サポートされているすべてのチェックサムタイプの各アセットで使用できます。例えば、Maven のパッケージバージョン commons-lang:commons-lang 2.1 で ListPackageVersionAssets を呼び出す場合の応答の一部は次のようになります。

```
{
    "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
    }
}
```

チェックサムはアセットとして保存されませんが、Maven クライアントは期待される場所でチェックサムを公開およびダウンロードできます。例えば、commons-lang:commons-lang 2.1 が maven-repo というリポジトリにある場合、JAR ファイルの SHA-256 チェックサムの URL パスは次のようになります。

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

既存の Maven パッケージ (以前に Amazon S3 に保存されたパッケージなど) を、curl などの汎用 HTTP クライアントを使用して CodeArtifact にアップロードする場合、チェックサムをアップロードする必要はありません。CodeArtifact はそれらを自動的に生成します。アセットが正しくアップロードされたことを確認するには、ListPackageVersionAssets API オペレーションを使用して、レスポンス内のチェックサムを各アセットの元のチェックサム値と比較します。

公開中のチェックサムの不一致

アセットとチェックサムの他に、Maven アーティファクトには maven-metadata.xml ファイルも含まれています。Maven パッケージの通常の公開手順では、すべてのアセットとチェックサムを最初にアップロードし、その後に maven-metadata.xml をアップロードします。例えば、前述の Maven パッケージバージョン commons-lang 2.1 の公開順序は、クライアントが SHA-256 チェックサムファイルを公開するように設定されていると仮定すると、次のようになります。

```
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
```

JAR ファイルなどのアセットのチェックサムファイルをアップロードする際に、アップロードされたチェックサム値と CodeArtifact によって計算されたチェックサム値の間に不一致があると、チェックサムのアップロードリクエストは 400 (Bad Request) レスポンスで失敗します。対応するアセットが存在しない場合、リクエストは 404 (Not Found) レスポンスで失敗します。このエラーを回避するには、まずアセットをアップロードし、次にチェックサムをアップロードする必要があります。

`maven-metadata.xml` をアップロードする際、CodeArtifact は通常 Maven パッケージバージョンのステータスを `Unfinished` から `Published` に変更します。いずれかのアセットでチェックサムの不一致が検出された場合、CodeArtifact は `maven-metadata.xml` 公開リクエストへの応答で 400 (Bad Request) を返します。このエラーにより、クライアントはこのパッケージバージョンのファイルのアップロードを停止することがあります。アップロードが停止し、`maven-metadata.xml` ファイルがアップロードされない場合、アップロード済みのパッケージバージョンのアセットはダウンロードできません。これは、パッケージバージョンのステータスが `Published` に設定されておらず、`Unfinished` のままであるためです。

CodeArtifact では、`maven-metadata.xml` がアップロードされパッケージバージョンのステータスが `Published` に設定された後でも、Maven のパッケージバージョンにさらにアセットを追加できます。このステータスでは、一致しないチェックサムファイルをアップロードするリクエストも 400 (Bad Request) レスポンスで失敗します。ただし、パッケージバージョンのステータスはすでに `Published` に設定されているため、チェックサムファイルのアップロードに失敗したものも含め、パッケージから任意のアセットをダウンロードできます。チェックサムファイルのアップロードに失敗したアセットのチェックサムをダウンロードする場合、クライアントが受け取るチェックサム値は、アップロードされたアセットデータに基づいて CodeArtifact によって計算されたチェックサム値になります。

CodeArtifact のチェックサム比較では大文字と小文字が区別され、CodeArtifact によって計算されたチェックサムは小文字でフォーマットされます。そのため、チェックサム `909FA780F76DA393E992A3D2D495F468` をアップロードすると、CodeArtifact では `909fa780f76da393e992a3d2d495f468` と同じチェックサムとして扱われないため、チェックサムの不一致でアップロードが失敗します。

チェックサムの不一致の解決

チェックサムの不一致が原因でチェックサムのアップロードが失敗した場合は、次のいずれかを試して問題を解決します。

- Maven アーティファクトを公開するコマンドを再度実行します。これは、ダウンロード中にネットワークの問題によってチェックサムファイルが破損した場合に役立つ可能性があります。再試行でネットワークの問題が解決された場合は、チェックサムが一致し、ダウンロードが成功します。
- パッケージバージョンを削除して、再度公開します。詳細については、「AWS CodeArtifact API Reference」の「DeletePackageVersions」を参照してください。

Maven スナップショットを使用する

Maven スナップショット は、最新のプロダクションブランチコードを参照する Maven パッケージの特別なバージョンです。これは最終リリース版に先行する開発版です。Maven パッケージのスナップショットバージョンは、パッケージバージョンに追加されているサフィックス SNAPSHOT で識別できます。例えば、バージョン 1.1 のスナップショットは 1.1-SNAPSHOT です。詳細については、Apache Maven プロジェクトウェブサイト上の スナップショットバージョンとは何ですか？ を参照してください。

AWS CodeArtifact は Maven スナップショットの発行と消費をサポートしています。サポートされているスナップショットは、時間ベースのバージョン番号を使用する一意のスナップショットのみです。CodeArtifact は、Maven 2 クライアントによって生成される一意でないスナップショットをサポートしていません。サポートされている Maven スナップショットは、任意の CodeArtifact リポジトリに公開できます。

トピック

- スナップショットを CodeArtifact で公開する
- スナップショットバージョンを使用する
- スナップショットバージョンを削除する
- curl を使用してスナップショットを公開する
- スナップショットと外部接続
- スナップショットとアップストリームリポジトリ

スナップショットを CodeArtifact で公開する

AWS CodeArtifact は、 などのクライアントがスナップショットを発行するときにmvn使用するリクエストパターンをサポートしています。そのため、Maven スナップショットの公開方法を詳しく理解していなくても、ビルドツールやパッケージマネージャーのドキュメントに従って操作することが

できます。より複雑な操作を実行する場合は、このセクションの CodeArtifact のスナップショットの処理方法を参照してください。

Maven スナップショットが CodeArtifact に公開されると、その前のバージョンはビルドという新しいバージョンに保存されます。Maven スナップショットが公開されるたびに、新しいビルドバージョンが作成されます。スナップショットの以前のバージョンはすべて、ビルドバージョンで保持されます。Maven スナップショットが公開されると、そのパッケージバージョンのステータスは Published に設定され、前のバージョンを含むビルドのステータスは Unlisted に設定されます。この動作は、パッケージバージョンに -SNAPSHOT サフィックスが付いている Maven パッケージバージョンのみに適用されます。

例えば、com.mycompany.myapp:pkg-1 という Maven パッケージのスナップショットバージョンは、my-maven-repo という CodeArtifact リポジトリにアップロードされます。スナップショットバージョンは 1.0-SNAPSHOT です。この時点では、com.mycompany.myapp:pkg-1 のバージョンは公開されていません。まず、初期ビルドのアセットが以下のパスで公開されます。

```
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
```

タイムスタンプ 20210728.194552-1 は、スナップショットビルドを公開するクライアントによって生成されることに注意してください。

.pom ファイルと.jar ファイルがアップロードされると、リポジトリに存在する唯一の com.mycompany.myapp:pkg-1 のバージョンは 1.0-20210728.194552-1 です。これは、前のパスで指定されたバージョンが 1.0-SNAPSHOT である場合でも同様です。この時点でのパッケージバージョンステータスは Unfinished です。

```
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"
}
```

次に、クライアントはパッケージバージョンの maven-metadata.xml ファイルをアップロードします。

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

maven-metadata.xml ファイルが正常にアップロードされると、CodeArtifact は 1.0-SNAPSHOT パッケージバージョンを作成し、1.0-20210728.194552-1 バージョンを Unlisted に設定します。

```
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"
}
```

この時点で、スナップショットバージョン 1.0-SNAPSHOT はビルドで使用できます。リポジトリ my-maven-repo には com.mycompany.myapp:pkg-1 の 2 つのバージョンが存在しますが、どちらも同じアセットを含んでいます。

```
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"
]
```

`--package-version` パラメータを `1.0-20210728.194552-1` に変更して上記と同じ `list-package-version-assets` コマンドを実行すると、同じ出力が得られます。

`1.0-SNAPSHOT` のビルドがリポジトリに追加されると、新しいビルドごとに新しい `Unlisted` パッケージバージョンが作成されます。バージョン `1.0-SNAPSHOT` のアセットは毎回更新されるため、バージョンは常にそのバージョンの最新ビルドになります。`1.0-SNAPSHOT` を最新のアセットで更新するには、新しいビルドの `maven-metadata.xml` ファイルをアップロードします。

スナップショットバージョンを使用する

スナップショットをリクエストすると、ステータス `Published` を持つバージョンが返されます。これは常に Maven スナップショットの最新バージョンです。URL パス内のスナップショットバージョン (`1.0-SNAPSHOT` など) の代わりにビルドバージョン番号 (`1.0-20210728.194552-1` など) を使用して、スナップショットの特定のビルドをリクエストすることもできます。Maven スナップショットのビルドバージョンを確認するには、「CodeArtifact API Guide」の ListPackageVersions API を使用して、ステータスパラメータを `Unlisted` に設定します。

スナップショットバージョンを削除する

Maven スナップショットのすべてのビルドバージョンを削除するには、DeletePackageVersions API を使用して削除するバージョンを指定します。

curl を使用してスナップショットを公開する

Amazon Simple Storage Service (Amazon S3) または別のアーティファクトリポジトリ製品に保存されている既存のスナップショットバージョンがある場合は、 AWS CodeArtifact に再公開できます。CodeArtifact は Maven スナップショットをサポートしているため (「スナップショットを CodeArtifact で公開する」を参照)、`curl` などの汎用 HTTP クライアントを使用してスナップショットを公開することは、「curl で公開する」で説明されている Maven リリースバージョンの公開よりも複雑になります。`mvn` や `gradle` などの Maven クライアントでスナップショットバージョンをビルドしてデプロイする場合、このセクションの記載は適用されないことに注意してください。対象のクライアントのドキュメントに従ってください。

スナップショットバージョンを公開するには、スナップショットバージョンの 1 つ以上のビルドを公開する必要があります。CodeArtifact では、スナップショットバージョンのビルドが n 件ある場合、n+ 1 の CodeArtifact バージョンが存在します。n 個のすべてのビルドバージョンのステータスは Unlisted で、1 つのスナップショットバージョン (最新の公開ビルド) のステータスは Published です。スナップショットバージョン (つまり、「-SNAPSHOT」を含むバージョン文字列を含むバージョン) には、最新の公開ビルドと同じアセットセットが含まれています。curl を使用してこの構造を作成する最も簡単な方法は次のとおりです。

1. curl を使用して、すべてのビルドのすべてのアセットを公開します。
2. curl を使用して、前のビルド (最新の日付/タイムスタンプが付いたビルド) の maven-metadata.xml ファイルを公開します。これにより、バージョン文字列に「-SNAPSHOT」を含み、正しいアセットセットを含むバージョンが作成されます。
3. UpdatePackageVersionsStatus API を使用して、最新でないすべてのビルドバージョンのステータスを Unlisted に設定します。

以下の curl コマンドを使用して、com.mycompany.app:pkg-1 パッケージのスナップショットバージョン 1.0-SNAPSHOT 用のスナップショットアセット (.jar ファイルや.pom ファイルなど) を公開します。

```
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
```

これらの例を使用する場合:

- *my_domain* を CodeArtifact ドメイン名に置き換えます。
- *111122223333* を CodeArtifact ドメインの所有者の AWS アカウント ID に置き換えます。
- *us-west-2* を CodeArtifact ドメインがある AWS リージョン に置き換えます。

- *my_maven_repo* を CodeArtifact リポジトリ名に置き換えます。

> **Important**
>
> `--data-binary` パラメータの値には `@` 文字をプレフィックスとして付ける必要があります。値を引用符で囲む場合は、`@` を引用符で囲む必要があります。

ビルドごとにアップロードするアセットが 3 つ以上ある場合があります。例えば、メインの JAR と `pom.xml` に加えて Javadoc ファイルとソース JAR ファイルがある場合があります。CodeArtifact はアップロードされたアセットごとにチェックサムを自動的に生成するため、パッケージバージョンアセットのチェックサムファイルを公開する必要はありません。アセットが正しくアップロードされたことを確認するには、`list-package-version-assets` コマンドを使用して生成されたチェックサムを取得し、元のチェックサムと比較します。CodeArtifact での Maven チェックサムの処理方法の詳細については、「Maven チェックサムの使用」を参照してください。

次の curl コマンドを使用して、最新のビルドバージョンの `maven-metadata.xml` ファイルを公開します。

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

`maven-metadata.xml` ファイルは、`<snapshotVersions>` エレメントの最新ビルドバージョンのアセットを少なくとも 1 つ参照している必要があります。また `<timestamp>` 値があり、アセットファイル名のタイムスタンプと一致している必要があります。例えば、以前に公開された `20210729.171330-2` ビルドの場合、`maven-metadata.xml` の内容は次のようになります。

```
<?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>
```

maven-metadata.xml を公開した後の最後のステップは、他のすべてのビルドバージョン (最新のビルド以外のすべてのビルドバージョン) のパッケージバージョンステータスを Unlisted に設定することです。例えば、1.0-SNAPSHOT バージョンに 2 つのビルドがあり、最初のビルドが 20210728.194552-1 の場合、そのビルドを設定するコマンドは Unlisted になります。

```
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
```

スナップショットと外部接続

Maven スナップショットは、外部接続を介して Maven パブリックリポジトリから取得することはできません。 AWS CodeArtifact は Maven リリースバージョンのインポートのみをサポートしています。

スナップショットとアップストリームリポジトリ

一般的に、Maven スナップショットは、アップストリームリポジトリで使用する場合 Maven リリースバージョンと同じように動作します。しかし、アップストリーム関係にある 2 つのリポジトリに対して同じパッケージバージョンのスナップショットを公開する場合は、制限があります。たとえば、 AWS CodeArtifact ドメインに 2 つのリポジトリがあり、 R Uが のアップストリームUである があるとしますR。R で新しいビルドを公開する場合、Maven クライアントがそのスナップショット

バージョンの最新のビルドをリクエストすると、CodeArtifact は U から最新バージョンを返します。最新バージョンは U ではなく R にあるため、これは予期されていない可能性があります。この問題を避けるには、以下の 2 つの方法があります。

1. U に 1.0-SNAPSHOT などのスナップショットバージョンのビルドが存在する場合は、1.0-SNAPSHOT を R に公開しない。
2. CodeArtifact パッケージオリジンコントロールを使用して、R で当該パッケージのアップストリームを無効にする。後者の方法では、1.0-SNAPSHOT のビルドを R 公開できるようになりますが、その一方で、U が保持されていない当該パッケージの他のバージョンを R から取得することができなくなります。

アップストリームと外部接続からの Maven パッケージのリクエスト

標準アセット名のインポート

Maven Central などのパブリックリポジトリから Maven パッケージバージョンをインポートする際、AWS CodeArtifact はそのパッケージバージョン内のすべてのアセットのインポートを試みます。「アップストリームリポジトリを持つパッケージバージョンのリクエスト」で説明したように、インポートは次の場合に行われます。

- クライアントが CodeArtifact リポジトリから Maven アセットをリクエストする。
- パッケージバージョンはリポジトリにもアップストリームにも存在しない。
- パブリック Maven リポジトリにアクセス可能な外部接続がある。

クライアントが 1 つのアセットのみをリクエストする場合でも、CodeArtifact はそのパッケージバージョンで見つかったすべてのアセットをインポートしようとします。CodeArtifact が Maven パッケージバージョンで使用できるアセットをどのように検出するかは、パブリックリポジトリによって異なります。一部のパブリック Maven リポジトリは、アセットリストのリクエストをサポートし、別のリポジトリはアセットリストのリクエストをサポートしていません。アセットリストを提供しないリポジトリの場合、CodeArtifact は存在する可能性のあるアセット名のセットを生成します。例えば、Maven パッケージバージョン `junit 4.13.2` のアセットがリクエストされると、CodeArtifact は次のアセットのインポートを試みます。

- `junit-4.13.2.pom`

- junit-4.13.2.jar
- junit-4.13.2-javadoc.jar
- junit-4.13.2-sources.jar

非標準アセット名のインポート

Maven クライアントが上記のパターンのいずれにも一致しないアセットをリクエストすると、CodeArtifact はそのアセットがパブリックリポジトリに存在するかどうかを確認します。アセットが存在する場合、そのアセットはインポートされ、既存のパッケージバージョンレコード (存在する場合) に追加されます。例えば、Maven パッケージバージョン com.android.tools.build:aapt2 7.3.1-8691043 には以下のアセットが含まれています。

- 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

クライアントが POM ファイルをリクエストする際に、CodeArtifact がパッケージバージョンのアセットを一覧表示できない場合、インポートされるアセットは POM のみです。これは、他のどのアセットも標準のアセット名パターンに一致しないためです。ただし、クライアントが JAR アセットの 1 つを要求すると、そのアセットはインポートされ、CodeArtifact に格納されている既存のパッケージバージョンに追加されます。「アップストリームリポジトリからのパッケージの保持」で説明されているように、最も下流のダウンストリームリポジトリ (クライアントがリクエストを行ったリポジトリ) と外部接続がアタッチされているリポジトリの両方のパッケージバージョンが、新しいアセットを含むように更新されます。

通常、パッケージバージョンが CodeArtifact リポジトリに保持されると、アップストリームリポジトリでの変更による影響を受けません。詳細については、「アップストリームリポジトリからのパッケージの保持」を参照してください。ただし、前述した非標準名の Maven アセットの動作は、この規則の例外です。ダウンストリームのパッケージバージョンは、クライアントから追加のアセットをリクエストされない限り変更されませんが、この場合、保持されるパッケージバージョンは最初に保持された後に変更されるため、不変ではありません。非標準名の Maven アセットには CodeArtifact からアクセスできないため、この動作は必要です。この動作は、パッケージバージョンが CodeArtifact リポジトリに保持された後にパブリックリポジトリの Maven パッケージバージョンに追加された場合にも有効になります。

アセットオリジンの確認

以前保持されていた Maven パッケージバージョンに新しいアセットを追加すると、CodeArtifact は保持されているパッケージバージョンのオリジンが新しいアセットの提供元と同じであることを確認します。これにより、異なるパブリックリポジトリから異なるアセットが発行される「混在」パッケージバージョンの作成を防ぐことができます。このチェックを行わない場合、Maven パッケージのバージョンが複数のパブリックリポジトリに公開され、それらのリポジトリが CodeArtifact リポジトリのアップストリームグラフの一部である場合、アセットが混在する可能性があります。

アップストリームリポジトリへの新しいアセットのインポートとパッケージバージョンステータスのインポート

アップストリームリポジトリのパッケージバージョンのパッケージバージョンステータスにより、CodeArtifact がそれらのバージョンをダウンストリームリポジトリに保持できなくなることがあります。

例えば、あるドメインに `repo-A`、`repo-B`、および `repo-C` の 3 つのリポジトリがあるとします。ここで、`repo-B` は `repo-A` のアップストリームに、`repo-C` は `repo-B` のアップストリームにあります。

Maven パッケージ `com.android.tools.build:aapt2` のパッケージバージョン `7.3.1` は `repo-B` にあり、ステータスは `Published` です。`repo-A` には存在しません。クライアントがこのパッケージバージョンのアセットを `repo-A` からリクエストした場合、レスポンスは 200 (OK) になり、Maven パッケージバージョン `7.3.1` は `repo-A` で保持されます。ただし、`repo-B` のパッケージバージョン `7.3.1` のステータスが `Archived` または `Disposed` の場合、これら 2 つのステータスのパッケージバージョンのアセットはダウンロードできないため、応答は 404 (Not Found) になります。

`com.android.tools.build:aapt2` の パッケージオリジンコントロール を `repo-A` で `upstream=BLOCK` に設定すると、`repo-B` と `repo-C` によってパッケージバージョンのステータスにかかわらず、そのパッケージのすべてのバージョンの新しいアセットは `repo-A` から取得されなくなることに注意してください。

Maven のトラブルシューティング

以下の情報は、Maven および CodeArtifact での一般的な問題のトラブルシューティングに役立ちます。

並列 PUT を無効にして 429: Too Many Requests を修正する

バージョン 3.9.0 以降、Maven はパッケージアーティファクトを並列でアップロードします (一度に最大 5 ファイル)。これにより、CodeArtifact がエラーレスポンスコード 429 (Too Many Requests) で応答することがあります。このエラーは、並列 PUT を無効にすることで修正できます。

並列 PUT を無効にするには、次の例に示すように、`settings.xml` ファイルのプロファイルで `aether.connector.basic.parallelPut` プロパティを `false` に設定します。

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

詳細については、Maven ドキュメントの「Artifact Resolver Configuration Options」を参照してください。

CodeArtifactをnpmで使用する

以下のトピックでは、Node.js パッケージマネージャーnpmをCodeArtifactで使用する方法について説明します。

> **Note**
>
> CodeArtifactは、`node v4.9.1`およびそれ以降、`npm v5.0.0`およびそれ以降に対応します。

トピック

CodeArtifact で npm を設定して使用する

CodeArtifact でリポジトリを作成したら、npm クライアントを使用してパッケージをインストールして公開できます。リポジトリエンドポイントと認可トークンで npm を設定するための推奨される方法は、`aws codeartifact login` コマンドの使用です。npm を手動で設定することもできます。

目次

login コマンドを使用した npm の設定

`aws codeartifact login` コマンドを使用して、npm で使用する認証情報を取得します。

> **Note**
>
> 所有しているドメインのリポジトリにアクセスする場合は、`--domain-owner` を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

> **Important**
>
> npm 10.x 以降を使用している場合は、 AWS CLI バージョン 2.9.5 以降を使用して`aws codeartifact login`コマンドを正常に実行する必要があります。

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

このコマンドは、 ~/.npmrc ファイルに次の変更を加えます:

- AWS 認証情報を使用して CodeArtifact から認可トークンを取得し、追加します。
- npm レジストリを、`--repository` オプションで指定されたリポジトリに設定します。
- npm 6 以下の場合: `"always-auth=true"` を追加すると、認可トークンがすべての npm コマンドに対して送信されます。

`login` を呼び出した後のデフォルトの認可時間は 12 時間であり、トークンを定期的に更新するには、`login` を呼び出す必要があります。`login` コマンドで作成される認可トークンの詳細については、「loginコマンドで作成されたトークン」を参照してください。

login コマンドを使用せずに npm を設定する

`aws codeartifact login` コマンドを使用せずに CodeArtifact リポジトリで npm を設定するには、npm 設定を手動で更新します。

login コマンドを使用せずに npm を設定するには

1. コマンドラインで、CodeArtifact 認可トークンを取得し、環境変数に保存します。npm はこのトークンを使用して CodeArtifact リポジトリで認証します。

   > **Note**
   >
   > 次のコマンドは、macOS または Linux 用です。Windows での環境変数の設定については、「環境変数を使用して認証トークンを渡す」を参照してください。

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

2. 次のコマンドを実行して、CodeArtifact リポジトリのエンドポイントを取得します。リポジトリエンドポイントは、パッケージをインストールまたは公開するために npm をリポジトリに指すために使用されます。

   - *my_domain* を CodeArtifact ドメイン名で置き換えます。
   - *111122223333* をドメイン所有者の AWS アカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
   - *my_repo* を CodeArtifact リポジトリ名で置き換えます。

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

   次の URL は、リポジトリエンドポイントの例です。

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

   > **Important**
   >
   > レジストリ URL はスラッシュ (/) で終わる必要があります。そうしないと、リポジトリに接続することはできません。

3. npm config setコマンドを使用して、CodeArtifact リポジトリにレジストリを設定します。URL を、前のステップのリポジトリエンドポイントの URL に置き換えます。

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

> **Note**
>
> デュアルスタックエンドポイントを使用するには、codeartifact.*region*.on.aws エンドポイントを使用してください。

4. npm config set コマンドを使用して、npm 設定に認可トークンを追加します。

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

npm 6 以下の場合: npm に常に認証トークンを CodeArtifact に渡すようにするには、GET リクエストの場合は、npm config set で always-auth 設定変数を設定します。

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

構成ファイルの例 (**.npmrc**)

次に .npmrc ファイルの例を示します。CodeArtifact レジストリエンドポイントを設定するための先の指示に従った後、認証トークンを追加し、always-auth を設定します。

```
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
```

npm コマンドを実行する

npm クライアントを設定したら、npm コマンドを実行できます。パッケージがリポジトリまたはその上流のリポジトリの1つに存在すると仮定すると、パッケージを `npm install` でインストールできます。例えば、以下を使用して `lodash` パッケージをインストールします。

```
npm install lodash
```

次のコマンドを使用して、新しい npm パッケージを CodeArtifact リポジトリに公開します。

```
npm publish
```

npm パッケージを作成する方法については、npm ドキュメントウェブサイト上の Node.js モジュールの作成 を参照してください。CodeArtifact でサポートされる npm コマンドのリストについては、npm コマンドのサポート を参照してください。

npm の認証と認可の検証

`npm ping` コマンドの呼び出しは、次のことを検証する方法です。

- CodeArtifact リポジトリに対して認証できるように、認証情報が正しく設定されました。
- 認可設定により、`ReadFromRepository` アクセス許可が与えられます。

`npm ping` の正常な呼び出しからの出力は次のようになります。

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

-d オプションを指定すると、npm はリポジトリ URL を含む追加のデバッグ情報を出力します。この情報により、npm が期待するリポジトリを使用するように設定されていることを簡単に確認できます。

デフォルトの npm レジストリに戻す

npm を CodeArtifact で設定すると、npm レジストリが指定された CodeArtifact リポジトリに設定されます。次のコマンドを実行すると、CodeArtifact への接続が完了したら、npm レジストリをデフォルトのレジストリに戻すことができます。

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

npm 8.x 以降でのインストールが遅い場合のトラブルシューティング

npm バージョン8.x 以降には既知の問題があります。パッケージリポジトリに対してリクエストが行われ、リポジトリがアセットを直接ストリーミングする代わりにクライアントを Amazon S3 にリダイレクトすると、npm クライアントが依存関係ごとに数分間ハングアップする可能性があります。

CodeArtifact リポジトリはリクエストを常に Amazon S3 にリダイレクトするように設計されているため、この問題が発生することがあり、npm のインストール時間が長くなるため、ビルド時間が長くなります。この動作が発生すると、進行状況バーが数分間表示されます。

この問題を回避するには、次の例のように npm CLI コマンドで `--no-progress` または `progress=false` フラグを使用します。

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

CodeArtifact で Yarn を設定して使用する

リポジトリを作成すると、Yarn クライアントを使用して npm パッケージを管理できます。

> **Note**
>
> `Yarn 1.X` は npm 設定ファイル (.npmrc) から情報を読み取り、使用します。一方、`Yarn 2.X` はこれを行いません。`Yarn 2.X` の設定は .yarnrc.yml ファイルで定義する必要があります。

目次

aws codeartifact login コマンドを使用して Yarn 1.X を設定します。

Yarn 1.X の場合では、aws codeartifact login コマンドを使用して CodeArtifact で Yarn を設定できます。login コマンドは、CodeArtifact リポジトリのエンドポイント情報と認証情報を使用して ~/.npmrc ファイルを設定します。Yarn 1.X と yarn コマンドは、 ~/.npmrc ファイルの設定情報を使用します。

login コマンドを使用して **Yarn 1.X** を設定するには

1. まだ設定していない場合は、CodeArtifact の開始方法 に記述されているように、AWS CLI で使用するための AWS 認証情報を設定します。
2. aws codeartifact login コマンドを正常に実行するには、npm をインストールする必要があります。インストール手順については、npm ドキュメント の Node.js と npm のダウンロードとインストール を参照してください。
3. aws codeartifact login コマンドを使用して CodeArtifact 認証情報を取得し、~/.npmrc ファイルを設定します。

   - *my_domain* を CodeArtifact ドメイン名で置き換えます。
   - *111122223333* をAWS ドメインの所有者のアカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合は、--domain-owner を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
   - *my_repo* を CodeArtifact リポジトリ名で置き換えます。

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

   loginコマンドを使用すると、~/.npmrc ファイルに次の変更が行われます。

   - AWS 認証情報を使用して CodeArtifact から認可トークンを取得し、追加します。
   - npm レジストリを、--repository オプションで指定されたリポジトリに設定します。

- npm 6 以下の場合: "always-auth=true" を追加すると、認可トークンはすべての npm コマンドに対して送信されます。

login を呼び出した後のデフォルトの認可期間は12時間であり、トークンを定期的に更新するには login を呼び出す必要があります。login コマンドを使用して作成された認可トークンの詳細については、「loginコマンドで作成されたトークン」を参照してください。

4. npm 7.X および 8.X の場合は、always-auth=true を~/.npmrc ファイルに追加して Yarn を使用する必要があります。

   - ~/.npmrc ファイルをテキストエディタで開き、always-auth=true を新しい行に追加します。

yarn config list コマンドを使用して、Yarn が正しい設定を使用していることを確認できます。コマンドを実行した後、info npm config セクションの値をチェックします。コンテンツは次のスニペットのようになります。

```
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
}
```

yarn config set コマンドを使用して Yarn 2.X を設定します。

次の手順は、yarn config set コマンドを使用して、コマンドラインから .yarnrc.yml 設定を更新して Yarn 2.X を設定する方法の詳細です。

コマンドラインから **yarnrc.yml** 設定を更新するには

1. まだ設定していない場合は、CodeArtifact の開始方法 に記述されているように、AWS CLI で使用するための AWS 認証情報を設定します。
2. aws codeartifact get-repository-endpoint コマンドを使用して、CodeArtifact リポジトリのエンドポイントを取得します。

- *my_domain* を CodeArtifact ドメイン名で置き換えます。
- *111122223333* をAWS ドメインの所有者のアカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合は、--domain-owner を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
- *my_repo* を CodeArtifact リポジトリ名で置き換えます。

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

3. リポジトリエンドポイントを使用して、.yarnrc.yml ファイルで npmRegistryServer 値を更新します。

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

4. CodeArtifact 認可トークンを取得し、環境変数に保存します。

> **Note**
>
> 次のコードは、Linux と MacOS 用です。Windows での環境変数の設定については、「環境変数を使用して認証トークンを渡す」を参照してください。

- *my_domain* を CodeArtifact ドメイン名で置き換えます。
- *111122223333* をAWS ドメインの所有者のアカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合は、--domain-owner を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
- *my_repo* を CodeArtifact リポジトリ名で置き換えます。

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

5. yarn config set コマンドを使用して、CodeArtifact 認可トークンを .yarnrc.yml ファイルに追加します。次のコマンドの URL を、ステップ 2 のリポジトリエンドポイントの URL に置き換えます。

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

6. `yarn config set` コマンドを使用して、npmAlwaysAuth の値を `true` に設定します。次のコマンドの URL を、ステップ 2 のリポジトリエンドポイントの URL に置き換えます。

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

設定後、.yarnrc.yml 設定ファイルには、次のスニペットに似た内容が含まれている必要があります。

```
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/"
```

また、`yarn config` コマンドを使用して、npmRegistries および npmRegistryServer の値を確認することもできます。

npm コマンドサポート

以下のセクションでは、CodeArtifact リポジトリでサポートされている npm コマンドと、サポートされていない特定のコマンドについてまとめます。

目次

リポジトリと対話するサポートされているコマンド

このセクションでは、npm クライアントが設定されているレジストリに対して 1 つ以上の要求を行う npm コマンドの一覧を示します (例えば、`npm config set registry`)。これらのコマンドは、CodeArtifact リポジトリに対して呼び出されたときに正しく機能することが確認されています。

コマンド	説明
bugs (バグ)	パッケージのバグトラッカー URL の場所を推測し、開こうとします。
ci (クリーンスレートインストール)	クリーンスレートでプロジェクトをインストールします。
deprecate (非推奨)	パッケージのバージョンを非推奨にします。
dist-tag (配布タグ)	パッケージ配布タグを変更します。
docs (ドキュメンテーション)	パッケージのドキュメンテーション URL の場所を推測し、`--browser config` パラメータを使用してそれを開こうとします。
doctor (ドクター)	一連のチェックを実行して、npm インストールに JavaScript パッケージを管理するために必要なものがあることを確認します。
install (インストール)	パッケージをインストールします。
install-ci-test (クリーンスレートインストールのテスト)	クリーンスレートでプロジェクトをインストールし、テストを実行します。エイリアス: `npm cit` このコマンドは、`npm ci` の直後に `npm test` を実行します。
install-test (インストールテスト)	パッケージをインストールしてテストを実行します。`npm install` の直後に `npm test` を実行します。

コマンド	説明
outdated (旧式)	設定されたレジストリをチェックして、インストールされているパッケージが現在古いかどうかを確認します。
ping (旧式)	設定または指定された npm レジストリに ping を実行し、認証を検証します。
publish (出力)	パッケージバージョンをレジストリに出力します。
update (アップデート)	パッケージのリポジトリ URL の場所を推測し、`--browser` config パラメータを使用してそれを開こうとします。
view (表示)	パッケージメタデータの表示 メタデータプロパティを印刷するために使用できます。

サポートされているクライアント側のコマンド

これらのコマンドはリポジトリとの直接的なやりとりを必要としないため、CodeArtifactはそのサポートのために何もする必要はありません。

コマンド	説明
buid (構築)	パッケージを構築します。
cache (キャッシュ)	パッケージキャッシュを操作します。
completion (完成)	すべての npm コマンドでタブ補完を有効にします。
config (設定)	ユーザーとグローバル `npmrc` ファイルのコンテンツを更新します。
depute (代理)	ローカルパッケージツリーを検索し、依存関係をツリー上に移動して構造を単純化しようとし

コマンド	説明
	ます。依存関係は複数の依存パッケージによってより効果的に共有できます。
edit (編集)	インストールされたパッケージを編集します。現在作業中のディレクトリ内の依存関係を選択し、パッケージフォルダをデフォルトエディタで開きます。
explore (調査)	インストールされているパッケージを参照します。指定されたインストール済みパッケージのディレクトリにサブシェルをスポーンします。コマンドが指定されている場合はサブシェルで実行され、すぐに終了します。
help (ヘルプ)	npm に関するヘルプを取得します。
help-search (ヘルプ検索)	npm ヘルプドキュメントを検索します。
init (初期)	`package.json` ファイルを作成します。
link (リンク)	パッケージフォルダをシンボリックリンクします。
ls (リスト)	インストールされているパッケージを一覧表示します。
pack (パッケージ)	パッケージから tarball を作成します。
prefix (プレフィックス)	プレフィックスを表示します。これは、`-g` も指定されていない限り、`package.json` ファイルを格納する最も近い親ディレクトリです。
prune (削除)	親パッケージの依存関係リストに一覧表示されていないパッケージを削除します。
rebuild (再構築)	一致したフォルダに対して `npm build` コマンドを実行します。

コマンド	説明
restart (再起動)	パッケージの停止、再起動、開始スクリプト、および関連する前後のスクリプトを実行します。
root (ルート)	効果的な node_modules フォルダをデフォルト出力に印刷します。
run-script (スクリプト実行)	任意のパッケージスクリプトを実行します。
shrinkwrap (収縮包装)	パブリケーションの依存関係バージョンをロックダウンします。
uninstall (アンインストール)	パッケージをアンインストールします。

サポートされていないコマンド

これらの npm コマンドは、CodeArtifact リポジトリではサポートされていません。

コマンド	説明	注意事項
access (アクセス)	公開パッケージのアクセスレベルを設定します。	CodeArtifact は、パブリック npmjs リポジトリとは異なるアクセス許可モデルを使用します。
adduser	レジストリユーザーアカウントを追加します。	CodeArtifact は、パブリック npmjs リポジトリとは異なるユーザーモデルを使用します。
audit (監査)	セキュリティ監査を実行します。	CodeArtifact は現在、セキュリティ脆弱性データを提供していません。

コマンド	説明	注意事項
hook (フック)	追加、削除、リスト、更新など、npm フックを管理します。	CodeArtifact は現在、いかなる種類の変更通知メカニズムもサポートしていません。
login (ログイン)	ユーザーを認証します。これは `npm adduser` のエイリアスです。	CodeArtifact は、公開 npmjs リポジトリとは異なる認証モデルを使用します。詳細については、npm の認証 を参照してください。
logout (サインアウト)	レジストリからサインアウトします。	CodeArtifact は、パブリック npmjs リポジトリとは異なる認証モデルを使用します。CodeArtifact リポジトリからサインアウトする方法はありませんが、認証トークンは、設定可能な有効期限後に期限切れになります。デフォルトのトークンの期間は 12 時間です。
owner (オーナー)	パッケージの所有者を管理します。	CodeArtifact は、公開 npmjs リポジトリとは異なるアクセス許可モデルを使用します。
profile (プロファイル)	レジストリプロファイルの設定を変更します。	CodeArtifact は、パブリック npmjs リポジトリとは異なるユーザーモデルを使用します。
search (検索)	検索語に一致するパッケージをレジストリで検索します。	CodeArtifact は、list-packages コマンドを使用した限定的な検索機能をサポートしています。

コマンド	説明	注意事項
star (星)	お気に入りのパッケージをマークします。	CodeArtifact は現在、いかなる種類のお気に入りメカニズムもサポートしていません。
stars (星)	お気に入りとしてマークされたパッケージを表示します。	CodeArtifact は現在、いかなる種類のお気に入りメカニズムもサポートしていません。
team (チーム)	組織チームとチームメンバーシップを管理します。	CodeArtifact は、公開 npmjs リポジトリとは異なるユーザーおよびグループのメンバーシップモデルを使用します。詳細については、IAM ユーザーガイド の アイデンティティ (ユーザー、グループ、ロール) を参照してください。
token (トークン)	認証トークンを管理します。	CodeArtifact は、認証トークンを取得するために別のモデルを使用します。詳細については、npm での認証 を参照してください。
unpublished	レジストリからパッケージを削除します。	CodeArtifact は、npm クライアントを使用したリポジトリからの公開バージョンの削除をサポートしていません。delete-package-version コマンドを使用できます。
whoami (私は誰)	npm ユーザー名を表示します。	CodeArtifact は、公開 npmjs リポジトリとは異なるユーザーモデルを使用します。

npm タグ処理

npm レジストリは タグ をサポートしており、これはパッケージバージョンの文字列エイリアスです。タグを使用して、バージョン番号の代わりにエイリアスを指定できます。例えば、複数の開発ストリームを持つプロジェクトがあり、別のタグ (例えば、`stable`、`beta`、`dev`、`canary`) をストリームごとに使用する場合があります。詳細については、npm ウェブサイト上の dist-tag を参照してください。

デフォルトでは、npm は `latest` タグを使用して、パッケージの現在のバージョンを識別します。`npm install` *`pkg`* (`@`*`version`* または `@`*`tag`* 指定子なし) は latest タグをインストールします。通常、プロジェクトは安定版リリースバージョンに対してのみ、latest タグを使用します。他のタグは、不安定版またはプレリリースバージョンに使用されます。

npm クライアントでタグを編集する

3 つの `npm dist-tag` コマンド (`add`、`rm` および `ls`) は、 CodeArtifact リポジトリでは、デフォルト npm レジストリ 内と同様に機能します。

npm タグと CopyPackageVersions API

`CopyPackageVersions` API を使用してnpm パッケージバージョンをコピーすると、そのバージョンをエイリアシングするすべてのタグがコピー先リポジトリにコピーされます。コピーされるバージョンに、コピー先にも存在するタグがある場合、コピー操作によって、コピー先リポジトリ内のタグ値がコピー元リポジトリの値と一致するように設定されます。

例えば、この表に示すように、リポジトリ S とリポジトリ D の両方に、latest タグセットで `web-helper` パッケージのシングルバージョンが含まれます。

リポジトリ	パッケージ名	パッケージタグ
S	`web-helper`	latest (バージョン 1.0.1 のエイリアス)
D	`web-helper`	latest (バージョン 1.0.0 のエイリアス)

S から D へ web-helper 1.0.1 をコピーするために CopyPackageVersions が呼び出されます。操作が完了した後、リポジトリ D の web-helper 上の latest タグは、 1.0.0 ではなく 1.0.1 に別名を付けます。

コピー後にタグを変更する必要がある場合は、npm dist-tag コマンドを実行して、コピー先リポジトリ内のタグを直接変更します。CopyPackageVersions API の詳細については、リポジトリ間でのパッケージのコピー を参照してください。

npm タグと上流リポジトリ

npm がパッケージのタグを要求し、そのパッケージのバージョンが上流リポジトリにも存在する場合、CodeArtifact はタグをマージしてからクライアントに返します。例えば、R というリポジトリには U という上流のリポジトリがあります。次の表に、両方のリポジトリに存在する web-helper という名前のパッケージのタグを示します。

リポジトリ	パッケージ名	パッケージタグ
R	web-helper	latest (バージョン 1.0.0 のエイリアス)
U	web-helper	alpha (バージョン 1.0.1 のエイリアス)

この場合、npm クライアントがリポジトリ R の web-helper パッケージのタグを取得すると、latest および alpha 両方の タグを取得します。タグが指すバージョンは変更されません。

同じタグが上流と下流の両方のリポジトリで同じパッケージに存在する場合、CodeArtifact は、上流リポジトリにあるタグを使用します。例えば、ウェブヘルパー 上のタグが次のように変更したとします。

リポジトリ	パッケージ名	パッケージタグ
R	web-helper	latest (バージョン 1.0.0 のエイリアス)
U	web-helper	latest (バージョン 1.0.1 のエイリアス)

この場合、npm クライアントが、リポジトリ R からパッケージ ウェブヘルパー のタグを取得すると、latest タグはバージョン 1.0.1 に別名を付けます。これが上流のリポジトリにあるからです。これにより、`npm update` を実行して、下流リポジトリにまだ存在していない上流リポジトリで、新しいパッケージバージョンを簡単に使用できるようになります。

上流リポジトリでタグを使用すると、パッケージの新しいバージョンを下流リポジトリに公開するときに問題が発生する可能性があります。例えば、パッケージ ウェブヘルパー の latest タグ は、R と U の両方で同じです。

リポジトリ	パッケージ名	パッケージタグ
R	`web-helper`	latest (バージョン 1.0.1 のエイリアス)
U	`web-helper`	latest (バージョン 1.0.1 のエイリアス)

R にバージョン 1.0.2 が公開されると、npm は latest タグを 1.0.2 に更新します。

リポジトリ	パッケージ名	パッケージタグ
R	`web-helper`	latest (バージョン 1.0.2 のエイリアス)
U	`web-helper`	latest (バージョン 1.0.1 のエイリアス)

ただし、U 内の latest は 1.0.1 であるため、npm クライアントはこのタグの値を認識しません。1.0.2 を公開した直後に、リポジトリ R に対して `npm install` を実行すると、公開されたばかりのバージョンの代わりに 1.0.1 がインストールされます。最後に公開されたバージョンをインストールするには、次のように正確なパッケージバージョンを指定する必要があります。

```
npm install web-helper@1.0.2
```

npm 互換パッケージマネージャーのサポート

これらの他のパッケージマネージャーは CodeArtifact と互換性があり、npm パッケージ形式と npm ワイヤプロトコルで動作します。

- pnpm パッケージマネージャー。CodeArtifact で動作することが確認された最新バージョンは 3.3.4 で、2019 年 5 月 18 日にリリースされたものです。
- Yarn パッケージマネージャー。CodeArtifact で動作することが確認された最新バージョンは 1.21.1 で、2019 年 12 月 11 日にリリースされたものです。

> **Note**
>
> CodeArtifact では Yarn 2.x を使用することをおすすめします。Yarn 1.x は HTTP リトライを行わないため、500 レベルのステータスコードやエラーが発生する断続的なサービス障害の影響を受けやすくなります。Yarn 1.x には別のリトライ戦略を設定する方法はありませんが、Yarn 2.x では追加されています。Yarn 1.x を使用することもできますが、ビルドスクリプトへの高レベルのリトライの追加が必要になる場合があります。例えば、yarn コマンドをループで実行して、パッケージのダウンロードに失敗した場合に再試行するようにします。

NuGetでCodeArtifactを使う

以下のトピックでは、CodeArtifactを使用したNuGetパッケージの消費および公開方法について説明します。

> **Note**
>
> AWSCodeArtifactはNuGet.exe バージョン 4.8 以上のみをサポートします。

トピック

- Visual Studio で CodeArtifact を使用する
- CodeArtifact をnuget または dotnet CLI で使用する
- NuGet パッケージ名、バージョン、アセット名の正規化
- NuGet の互換性

Visual Studio で CodeArtifact を使用する

CodeArtifact 認証情報プロバイダーを使用して、Visual Studio で CodeArtifact からパッケージを直接使用できます。認証情報プロバイダーは、Visual Studio での CodeArtifact リポジトリのセットアップと認証を簡素化し、AWS Toolkit for Visual Studio で使用可能です。

> **Note**
>
> AWS Toolkit for Visual Studio は、Visual Studio for Mac では使用できません。

CLI ツールで NuGet を設定して使用するには、CodeArtifact をnuget または dotnet CLI で使用する を参照してください。

トピック

- CodeArtifact 認証情報プロバイダーを使用して、Visual Studio を設定します。
- Visual Studio のパッケージマネージャーコンソールを使用する

CodeArtifact 認証情報プロバイダーを使用して、Visual Studio を設定します。

CodeArtifact 認証情報プロバイダーは、CodeArtifact と Visual Studio 間のセットアップと継続的な認証を簡素化します。CodeArtifact 認証トークンは、最大 12 時間有効です。Visual Studio での作業中にトークンを手動で更新する必要がないように、資格情報プロバイダーは、現在のトークンの有効期限が切れる前に定期的に新しいトークンを取得します。

> **Important**
>
> 認証情報プロバイダーを使用するには、手動または `aws codeartifact login` を実行して追加された可能性のある `nuget.config` ファイルから既存のAWS CodeArtifactが削除されていることを確認します。

AWS Toolkit for Visual Studio を使用して、Visual Studioで CodeArtifact を使用する

1. 以下の手順を使用して、AWS Toolkit for Visual Studio をインストールします。このツールキットは、次の手順を使用して Visual Studio 2017 および 2019 と互換性があります。AWSCodeArtifact は Visual Studio 2015 以前をサポートしていません。
   1. Visual Studio 2017 および Visual Studio 2019 の Toolkit for Visual Studio は、Visual Studio Marketplace で配信されています。また、Visual Studio 内でツールキットをインストールおよび更新するには、[Tools] ≫ [Extensions and Updates] (Visual Studio 2017) または [Extensions] ≫ [Manage Extensions] (Visual Studio 2019) の順に選択します。
   2. ツールキットをインストールしたら、表示 メニューから AWS エクスプローラ を選択してツールキットを開きます。
2. AWS Toolkit for Visual Studioユーザーガイド の AWS認証情報の提供 にある手順に従って、AWS認証情報でToolkit for Visual Studio を設定します。
3. (オプション) AWS CodeArtifact で使用するプロファイルを設定します。設定されていない場合、CodeArtifact はデフォルトのプロファイルを使用します。プロファイルを設定するには、ツール > NuGet パッケージマネージャー > CodeArtifact AWS プロフィールの選択 に移動します。
4. Visual Studio で、CodeArtifact リポジトリをパッケージソースとして追加します。
   1. AWSエクスプローラ ウィンドウでリポジトリに移動し、右クリックして `Copy NuGet Source Endpoint` を選択します。

2. ツール > オプション コマンドを使用して、NuGet パッケージマネージャー までスクロールします。
3. パッケージソース ノードを選択します。
4. + を選択して名前を編集し、ステップ 3a でコピーしたリポジトリ URL エンドポイントを 送信元 ボックスに貼り付け、更新を選択します。
5. 新しく追加したパッケージソースのチェックボックスを選択して有効にします。

> **Note**
>
> Nuget.org への外部接続を CodeArtifact リポジトリに追加し、Visual Studio で nuget.org パッケージソースを無効にすることをお勧めします。外部接続を使用する場合、Nuget.org から取得されたすべてのパッケージは、 CodeArtifact リポジトリに保存されます。Nuget.org が使用できなくなっても、アプリケーションの依存関係は CI ビルドとローカル開発で引き続き使用できます。外部接続の詳細については、「CodeArtifact リポジトリをパブリックリポジトリに接続する」を参照してください。

5. Visual Studio を再起動して、変更を有効にします。

設定後、Visual Studio は CodeArtifact リポジトリ、その上流のリポジトリー、または外部接続を追加した場合は Nuget.org からパッケージを消費できます。Visual Studio での NuGet パッケージの参照とインストールの詳細については、NuGet ドキュメント の NuGet パッケージマネージャーを使用して Visual Studio でパッケージをインストールして管理する を参照してください。

Visual Studio のパッケージマネージャーコンソールを使用する

Visual Studio のパッケージマネージャーコンソールは、Visual Studio バージョンの CodeArtifact 認証情報プロバイダーを使用しません。これを使用するには、コマンドライン認証情報プロバイダーを設定する必要があります。詳細については「CodeArtifact をnuget または dotnet CLI で使用する」を参照してください。

CodeArtifact をnuget または dotnet CLI で使用する

`nuget` および `dotnet` のような CLI ツールを使用して、CodeArtifact からパッケージを公開して消費できます。このドキュメントでは、CLI ツールの設定と、それらを使用してパッケージを公開または使用する方法について説明します。

トピック

nuget または dotnet CLI を設定する

nuget または dotnet CLI は、CodeArtifact NuGet 認証情報プロバイダーを使用して、 を使用して AWS CLI、または手動で設定できます。セットアップを簡素化し、認証を継続するために、NuGet を認証情報プロバイダーで設定することを強くお勧めします。

方法 1: CodeArtifact NuGet 認証情報プロバイダーを使用して設定する

CodeArtifact NuGet 認証情報プロバイダーは、NuGet CLI ツールを使用した CodeArtifact の認証と設定を簡素化します。CodeArtifact 認証トークンは、最大 12 時間有効です。nuget または dotnet CLI の使用中にトークンを手動で更新する必要がないように、認証情報プロバイダーは現在のトークンの有効期限が切れる前に定期的に新しいトークンを取得します。

> **Important**
>
> 認証情報プロバイダーを使用するには、手動で追加された既存の AWS CodeArtifact 認証情報、または以前に NuGet を設定するために を実行して追加された可能性のある既存の CodeArtifact 認証情報が`nuget.config`ファイルからクリアされていることを確認します。`aws codeartifact login` NuGet

CodeArtifact NuGet 認証情報プロバイダーのインストールと設定

dotnet

1. 以下の `dotnet` コマンドを使用して、NuGet.org の AWS.CodeArtifact.NuGet.CredentialProvider ツールをダウンロードします。

```
dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
```

2. `codeartifact-creds install` コマンドを使用して、認証情報プロバイダーを NuGet プラグインフォルダにコピーします。

```
dotnet codeartifact-creds install
```

3. (オプション): 認証情報プロバイダーで使用する AWS プロファイルを設定します。設定されていない場合は、認証情報プロバイダーはデフォルトのプロファイルを使用します。 AWS CLI プロファイルの詳細については、「名前付きプロファイル」を参照してください。

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

nuget

次の手順を実行して、NuGet CLI を使用して Amazon S3 バケットから CodeArtifact NuGet 認証情報プロバイダーをインストールして設定します。認証情報プロバイダーはデフォルトの AWS CLI プロファイルを使用します。プロファイルの詳細については、「名前付きプロファイル」を参照してください。

1. CodeArtifact NuGet 認証情報プロバイダー (codeartifact-nuget-credentialprovider.zip) の最新バージョンを Amazon S3 バケットからダウンロードします。

   以前のバージョンを表示してダウンロードするには、「CodeArtifact NuGet 認証情報プロバイダーのバージョン」を参照してください。
2. ファイル を解凍します。
3. AWS.CodeArtifact.NuGetCredentialProvider フォルダを、netfx フォルダから、Windows の場合は `%user_profile%/.nuget/plugins/netfx/`、LinuxまたはMacOSの場合は `~/.nuget/plugins/netfx` にコピーします。
4. AWS.CodeArtifact.NuGetCredentialProvider フォルダを、netcore フォルダから、Windows の場合 `%user_profile%/.nuget/plugins/netcore/` は、LinuxまたはMacOS の場合は `~/.nuget/plugins/netcore` にコピーします。

リポジトリを作成して認証情報プロバイダーを設定したら、nuget または dotnet CLI ツールを使用して、パッケージをインストールして公開できます。詳細については、「CodeArtifact から NuGet パッケージを消費する」および「NuGet パッケージを CodeArtifact に公開する」を参照してください。

方法 2: login コマンドで nuget または dotnet を設定する

の `codeartifact login` コマンドは、NuGet 設定ファイルにリポジトリエンドポイントと認可トークン AWS CLI を追加し、nuget または dotnet が CodeArtifact リポジトリに接続できるようにします。これにより、Windows の場合は`%appdata%\NuGet\NuGet.Config`、Mac/Linux の場合は `~/.config/NuGet/NuGet.Config` または `~/.nuget/NuGet/NuGet.Config` にあるユーザーレベルの NuGet 設定が変更されます。NuGet 設定の詳細については、一般的な NuGet の設定 を参照してください。

nuget または dotnet を **`login`** コマンドで設定する

1. の説明に従って AWS CLI、 で使用する AWS 認証情報を設定しますCodeArtifact の開始方法。
2. NuGet CLI ツール (nugetまたはdotnet) が正しくインストールされ、設定されていることを確認します。手順については、nuget または dotnet ドキュメントを参照してください。
3. CodeArtifact `login` コマンドを使用して、NuGet で使用する認証情報を取得する。

> **Note**
>
> 所有しているドメインのリポジトリにアクセスする場合は、`--domain-owner` を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

dotnet

> **Important**
>
> Linux および MacOS のユーザー: 暗号化は Windows 以外のプラットフォームではサポートされていないため、取得した資格情報はプレーンテキストとして設定ファイルに保存されます。

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

nuget

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

login コマンドは次の処理を行います:

- AWS 認証情報を使用して、CodeArtifact から認可トークンを取得します。
- ユーザーレベルの NuGet 設定を、NuGet パッケージソースの新しいエントリで更新します。CodeArtifact リポジトリエンドポイントを指すソースは *domain_name/repo_name* と呼ばれます。

`login` を呼び出した後のデフォルトの認可期間は12時間であり、トークンを定期的に更新するには、`login` を呼び出す必要があります。`login` コマンドを使用して作成された認可トークンの詳細については、「loginコマンドで作成されたトークン」を参照してください。

リポジトリを作成して認証を設定したら、`nuget`、`dotnet` または `msbuild` パッケージを使用して、CLI クライアントをインストールして公開できます。詳細については、「CodeArtifact から NuGet パッケージを消費する」および「NuGet パッケージを CodeArtifact に公開する」を参照してください。

方法 3: login コマンドなしで nuget または dotnet を設定する

マニュアル設定の場合、リポジトリエンドポイントと認可)トークンを NuGet 設定ファイルに追加して、nuget または dotnet が CodeArtifact リポジトリに接続できるようにする必要があります。

CodeArtifact リポジトリに接続するには、nuget または dotnet を手動で設定します。

1. `get-repository-endpoint` AWS CLI コマンドを使用して CodeArtifact リポジトリエンドポイントを決定します。

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

出力例:

```
{
```

```
   "repositoryEndpoint": "https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/nuget/my_repo/"
}
```

2. get-authorization-token AWS CLI コマンドを使用して、パッケージマネージャーからリポジトリに接続するための認可トークンを取得します。

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

出力例:

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

3. ステップ 3 の get-repository-endpoint で返された URL に /v3/index.json を追加して、完全なリポジトリエンドポイント URL を作成します。
4. ステップ 1 のリポジトリエンドポイントとステップ 2 の認可トークンを使用するには、nuget または dotnet を設定します。

> **Note**
>
> nuget または dotnet が CodeArtifact リポジトリに正常に接続できるようにするには、ソース URL の最後が /v3/index.json である必要があります。

dotnet

Linux および MacOS のユーザー: 暗号化は Windows 以外のプラットフォームではサポートされていないため、次のコマンドに --store-password-in-clear-text フラグを追加する必要があります。これにより、パスワードがプレーンテキストとして設定ファイルに保存されることに注意してください。

```
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**
>
> 既存のソースを更新するには、`dotnet nuget update source` コマンドを使用します。

nuget

```
nuget sources add -name domain_name/repo_name -Source
 https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
nuget/my_repo/v3/index.json -password eyJ2I...vi0w -username aws
```

出力例:

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

> **Note**
>
> デュアルスタックエンドポイントを使用するには、`codeartifact.region.on.aws` エンドポイントを使用してください。

CodeArtifact から NuGet パッケージを消費する

いったん CodeArtifact でNuGetを設定 したら、CodeArtifact リポジトリまたはその上流リポジトリの 1 つに保存されている NuGet パッケージを使用できます。

CodeArtifact リポジトリまたはその上流リポジトリの 1 つからパッケージバージョンを `nuget` または `dotnet` で消費するには、次のコマンドを実行し、*packageName* を消費したいパッケージの名前で置き換え、*packageSourceName* をNuGet 設定ファイル内の CodeArtifact リポジトリのソース名で置き換えます。`login` コマンド を使用して NuGet 設定を設定した場合、ソース名は *domain_name/repo_name* です。

> **Note**
>
> パッケージがリクエストされると、NuGet クライアントはパッケージのどのバージョンが存在するかをキャッシュします。この動作のため、目的のバージョンが入手可能になる前にリ

クエストされたパッケージのインストールが失敗することがあります。この問題を回避し、既存のパッケージを正常にインストールするには、`nuget locals all --clear` または `dotnet nuget locals all --clear` を使用してインストール前に NuGet キャッシュをクリアするか、または`nuget` の `-NoCache` オプションか `dotnet` の `--no-cache` オプションを指定して、`install` コマンドおよび `restore` コマンド中にキャッシュしないようにします。

dotnet

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

nuget

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

パッケージの特定バージョンをインストールするには

dotnet

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

nuget

```
nuget install packageName -Version 1.0.0 -Source packageSourceName
```

詳細については、マイクロソフトドキュメント の nuget.exe CLI を使用してパッケージを管理する または dotnet CLI を使用してパッケージをインストールして管理する を参照してください。

NuGet.org から NuGet パッケージを消費する

NuGet パッケージは、Nuget.org への外部接続を設定することで、CodeArtifact リポジトリを介して Nuget.org から消費できます。Nuget.org から消費されたパッケージは、CodeArtifact リポジトリに取り込まれて保存されます。外部接続の追加方法の詳細については、「CodeArtifact リポジトリをパブリックリポジトリに接続する」を参照してください。

NuGet パッケージを CodeArtifact に公開する

いったん CodeArtifact で NuGet を設定 すると、nuget または dotnet を使用して CodeArtifact リポジトリにパッケージバージョンを公開できます。

パッケージバージョンを CodeArtifact リポジトリにプッシュするには、次のコマンドを、NuGet 設定ファイル内の .nupkg ファイルと CodeArtifact リポジトリのソース名にフルパスで実行します。login コマンドを使用して NuGet 設定を設定した場合、ソース名は domain_name/repo_name です。

> **Note**
>
> 公開するパッケージがない場合は、NuGet パッケージを作成できます。詳細については、Microsoft ドキュメント の パッケージ作成ワークフロー を参照してください。

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 認証情報プロバイダーの参照

CodeArtifact NuGet 認証情報プロバイダーを使用すると、CodeArtifact リポジトリを使用して NuGet を簡単に設定および認証できます。

CodeArtifact NuGet 認証情報プロバイダーのコマンド

このセクションには、CodeArtifact NuGet 認証情報プロバイダーのコマンドリストが含まれます。これらのコマンドには、次の例のように、dotnet codeartifact-creds でプレフィックスを付ける必要があります。

```
dotnet codeartifact-creds command
```

- `configure set profile profile`: 提供された AWS プロファイルを使用するように認証情報プロバイダーを設定します。
- `configure unset profile`: 設定済みのプロファイルを削除します。
- `install`: 認証情報プロバイダーを `plugins` フォルダにコピーします。
- `install --profile profile`: 認証情報プロバイダーを `plugins`フォルダにコピーし、提供された AWS プロファイルを使用するように設定します。
- `uninstall`: 認証情報プロバイダーをアンインストールします。これにより、設定ファイルに対する変更は削除されません。
- `uninstall --delete-configuration`: 認証情報プロバイダーをアンインストールして、設定ファイルに対するすべての変更を削除します。

CodeArtifact NuGet 認証情報プロバイダーのログ

CodeArtifact NuGet 認証情報プロバイダーのログを有効にするには、環境でログファイルを設定する必要があります。認証情報プロバイダーのログには、次のような有用なデバッグ情報が含まれています:

- 接続に使用される AWS プロファイル
- すべての認証エラー
- 提供されたエンドポイントがCodeArtifact URLでない場合

CodeArtifact NuGet 認証情報プロバイダーのログファイルを設定する

```
export AWS_CODEARTIFACT_NUGET_LOGFILE=/path/to/file
```

ログファイルの設定後、`codeartifact-creds` コマンドはログ出力をそのファイルの内容に追加します。

CodeArtifact NuGet 認証情報プロバイダーのバージョン

CodeArtifact NuGet 認証情報プロバイダーのバージョン履歴情報とダウンロードリンクを次の表に示します。

バージョン	変更	公開日	ダウンロードリンク (S3)
1.0.2 (最新)	依存関係を更新しました	06/26/2024	v1.0.2 をダウンロードする
1.0.1	net5、net6、および SSO プロファイルのサポートが追加されました。	03/05/2022	v1.0.1 をダウンロード
1.0.0	CodeArtifact NuGet 認証情報プロバイダーの初期リリース	11/20/2020	v1.0.0 をダウンロード

NuGet パッケージ名、バージョン、アセット名の正規化

CodeArtifact は、パッケージ名、アセット名、およびパッケージのバージョンを保存する前に正規化します。つまり、CodeArtifact の名前またはバージョンは、パッケージまたはアセットが公開されたときに提供されたものとは異なる場合があります。

パッケージ名の正規化: CodeArtifact は、すべての文字を小文字に変換することで NuGet パッケージ名を正規化します。

パッケージバージョンの正規化: CodeArtifact は NuGet と同じパターンを使用して NuGet パッケージのバージョンを正規化します。以下の情報は、NuGet ドキュメントの「Normalized version numbers」に記載されているものです。

- 先頭の 0 はバージョン番号から削除されます。
  - `1.00` は `1.0` として扱われます。
  - `1.01.1` は `1.1.1` として扱われます。
  - `1.00.0.1` は `1.0.0.1` として扱われます。
- バージョン番号の 4 番目の部分の 0 は省略されます。
  - `1.0.0.0` は `1.0.0` として扱われます。
  - `1.0.01.0` は `1.0.1` として扱われます。
- SemVer 2.0.0 ビルドメタデータは削除されます。

- `1.0.7+r3456` は `1.0.7` として扱われます。

パッケージアセット名の正規化: CodeArtifact は、正規化されたパッケージ名とパッケージバージョンから NuGet パッケージアセット名を作成します。

CodeArtifact はこれらのリクエストのパッケージ名とバージョンの入力を正規化するため、正規化されていないパッケージ名とバージョン名は API および CLI リクエストで使用できます。例えば、`--package Newtonsoft.JSON` および `--version 12.0.03.0` の入力は正規化され、正規化されたパッケージ名 `newtonsoft.json` とバージョン `12.0.3` が返されます。

CodeArtifact は `--asset` 入力に対して正規化を実行しないため、API および CLI リクエストでは正規化されたパッケージアセット名を使用する必要があります。

ARN では正規化された名前とバージョンを使用する必要があります。

正規化されたパッケージ名を検索するには、`aws codeartifact list-packages` コマンドを使用します。詳細については、「パッケージ名を一覧表示する」を参照してください。

正規化されていないパッケージ名を検索するには、`aws codeartifact describe-package-version` コマンドを使用します。正規化されていないパッケージ名が `displayName` フィールドに返されます。詳細については、「パッケージのバージョンの詳細と依存関係の表示および更新」を参照してください。

NuGet の互換性

このガイドには、CodeArtifact のさまざまな NuGet ツールおよびバージョンとの互換性に関する情報が含まれています。

トピック

- NuGet の一般的な互換性
- NuGet コマンドラインサポート

NuGet の一般的な互換性

AWS CodeArtifact は NuGet 4.8 以降をサポートしています。

AWS CodeArtifact は NuGet HTTP プロトコルの V3 のみをサポートしています。これは、プロトコルの V2 に依存する一部の CLI コマンドはサポートされていないことを意味します。詳細については、「nuget.exe コマンドのサポート」セクションを参照してください。

AWS CodeArtifact は PowerShellGet 2.x をサポートしていません。

NuGet コマンドラインサポート

AWS CodeArtifact は、NuGet (`nuget.exe`) と .NET Core (`dotnet`) CLI ツールをサポートしています。

nuget.exe コマンドのサポート

CodeArtifact は NuGet の HTTP プロトコルの V3 のみをサポートしているため、CodeArtifact リソースに対して次のコマンドを使用すると機能しません。

- `list`: `nuget list` コマンド は、指定したソースからパッケージのリストを表示します。CodeArtifact リポジトリ内のパッケージのリストを取得するには、AWS CLIから パッケージ名を一覧表示する コマンドを使用できます。

PythonでCodeArtifactを使う

以下のトピックでは、pip、Python パッケージマネージャー、そして twine、Python パッケージパブリッシングユーティリティを CodeArtifact で使う方法を解説します。

トピック

CodeArtifact で pip を設定して使用する

pip は Python パッケージのパッケージインストーラーです。pip を使用して CodeArtifact リポジトリから Python パッケージをインストールするには、まず CodeArtifact リポジトリの情報と認証情報を使用して pip クライアントを設定する必要があります。

pip は Python パッケージのインストールにのみ使用できます。Python パッケージを公開するには、twine を使用します。詳細については、「CodeArtifact で twine を設定して使用する」を参照してください。

`login` コマンドで pip を設定します。

まず、「」の説明に従って AWS CLI、 で使用する AWS 認証情報を設定しますCodeArtifact の開始方法。次に、CodeArtifact `login` コマンドを使用して認証情報を取得し、この認証情報を使用して `pip` を設定します。

> **Note**
>
> 所有しているドメインのリポジトリにアクセスする場合、`--domain-owner`を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

pipを設定するには、次のコマンドを実行します。

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

login は、 AWS 認証情報を使用して CodeArtifact から認可トークンを取得します。login コマンドは、~/.config/pip/pip.conf を編集して --repository オプションで指定されたリポジトリに index-url を設定することで、pip を CodeArtifact で使用できるようにします。

login を呼び出した後のデフォルトの認可時間は 12 時間であり、トークンを定期的に更新するには、login を呼び出す必要があります。login コマンドで作成される認可トークンの詳細については、「loginコマンドで作成されたトークン」を参照してください。

ログインコマンドを使用せずにpipを設定する

pipの設定にloginコマンドを使用できない場合、pip configが使用できます。

1. AWS CLI を使用して、新しい認可トークンを取得します。

> **Note**
>
> 所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

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

2. pip configを使用して CodeArtifactレジストリ URL と資格情報を設定します。以下のコマンドは、現在の環境設定ファイルのみを更新します。システム全体の設定ファイルを更新するには、site を global に置き換えます。

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

> **Note**
>
> デュアルスタックエンドポイントを使用するには、codeartifact.*region*.on.aws エンドポイントを使用してください。

> **Important**
>
> レジストリURLは、スラッシュ (/) で終わる必要があります。そうでないと、リポジトリに接続することはできません。

pip 設定ファイルの例

CodeArtifactのレジストリURLと資格情報を設定した後のpip.confファイルの例を以下に示します。

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

pipを実行する

pipコマンドを実行するには、CodeArtifactでpipを設定しなくてはなりません。詳細については、次のドキュメントを参照してください。

1. AWS CodeArtifact でのセットアップ セクションの手順に従って、 AWS アカウント、ツール、アクセス許可を設定します。
2. CodeArtifact で twine を設定して使用するの手順に従って、twineを設定します。

パッケージが、リポジトリまたはそのアップストリームリポジトリの1つに存在する場合、pip install でインストールすることができます。例えば、requestsパッケージをインストールするには、次のコマンドを使用します。

```
pip install requests
```

CodeArtifactリポジトリではなく、https://pypi.org からのパッケージのインストールに一時的に戻すには、-i オプションを使います。

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

CodeArtifact で twine を設定して使用する

twine は Python パッケージのパッケージ公開ユーティリティです。twine を使用して CodeArtifact リポジトリに Python パッケージを公開するには、まず CodeArtifact リポジトリの情報と認証情報を使用して twine を設定する必要があります。

twine は Python パッケージの公開にのみ使用できます。Python パッケージをインストールするには、pip を使用します。詳細については、「CodeArtifact で pip を設定して使用する」を参照してください。

`login` コマンドを使用して twine を設定する

まず、「」の説明に従って AWS CLI、 で使用する AWS 認証情報を設定しますCodeArtifact の開始方法。次に、CodeArtifact `login` コマンドを使用して認証情報を取得し、この認証情報を使用して twine を設定します。

> **Note**
>
> 所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

twineを設定するには、次のコマンドを実行します。

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

`login` は、 AWS 認証情報を使用して CodeArtifact から認可トークンを取得します。`login` コマンドは、`~/.pypirc` を編集し認証情報を含む `--repository` オプションで指定されたリポジトリを追加することで、twine を CodeArtifact で使用できるように設定します。

`login` を呼び出した後のデフォルトの認可時間は 12 時間であり、トークンを定期的に更新するには、`login` を呼び出す必要があります。`login` コマンドで作成される認可トークンの詳細については、「loginコマンドで作成されたトークン」を参照してください。

login コマンドを使用せずに twine を設定する

login コマンドが twine の設定に使用できない場合、~/.pypirc ファイルまたは環境変数を使用することができます。~/.pypircファイル を使用するためには、次のエントリを追加します。パスワードは、get-authorization-tokenAPI によって取得された認証トークンである必要があります。

```
[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**
>
> デュアルスタックエンドポイントを使用するには、codeartifact.*region*.on.aws エンドポイントを使用してください。

環境変数を使用するには、以下の操作を実行します。

> **Note**
>
> 所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません 詳細については、「クロスアカウントドメイン」を参照してください。

```
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`
```

twineを実行する

twine を使って Python パッケージアセットを公開するには、まず CodeArtifact の権限とリソースを設定する必要があります。

1. AWS CodeArtifact でのセットアップ セクションの手順に従って、 AWS アカウント、ツール、アクセス許可を設定します。
2. 「login コマンドを使用して twine を設定する」または「login コマンドを使用せずに twine を設定する」の手順に従って twine を設定します。

twine の設定後、twine コマンドを実行することができます。Pythonパッケージアセットを公開するには、次のコマンドを使用します。

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

Pythonアプリケーションのビルドとパッケージ化の方法については、Pythonパッケージングオーソリティのウェブサイトの配布アーカイブの生成 (Generating Distribution Archives)を参照してください。

Python パッケージ名の正規化

CodeArtifact は、パッケージ名を保存する前に正規化します。つまり、CodeArtifact のパッケージ名は、パッケージが公開されたときに提供されたものとは異なる場合があります。

Python パッケージの場合、パッケージ名を正規化すると、パッケージ名は小文字になり、すべての `.`、`-`、`_` は 1 つの `-` に置き換えられます。そのため、`pigeon_cli` と `pigeon.cli` のパッケージ名は正規化され、`pigeon-cli` として保存されます。pip と twine では正規化されていない名前を使用できますが、CodeArtifact CLI または API リクエスト (`list-package-versions` など) および ARN では、正規化された名前を使用する必要があります。Python パッケージ名の正規化の詳細については、Python のドキュメントの「PEP 503」を参照してください。

Python の互換性

CodeArtifactでは PyPIの`XML-RPC`、または `JSONAPI` はサポートしていません。

CodeArtifactは PyPIの`Legacy`API をサポートしています。ただし`simple`API はその限りではありません。CodeArtifact は `/simple/` API エンドポイントをサポートしていませんが、`/simple/<project>/` エンドポイントはサポートしています。

詳細については、Pythonパッケージングオーソリティの GitHub リポジトリの以下を参照してください。

- XML-RPC API
- JSON API
- Legacy API

pipコマンドサポート

以下のセクションでは、CodeArtifactリポジトリでサポートされているpipコマンドと、サポートされていない特定のコマンドについてまとめます。

トピック

- リポジトリとインタラクトするサポートされたコマンド
- サポートされているクライアント側コマンド

リポジトリとインタラクトするサポートされたコマンド

このセクションでは、pipクライアントが設定されたレジストリに1つかそれ以上のリクエストを行うpipコマンドをリストアップします。これらのコマンドは、CodeArtifactリポジトリに対して呼び出されたときに正しく機能することが確認されています。

コマンド	説明
install (インストール)	パッケージのインストール
download	パッケージのダウンロード

CodeArtifactは`pip search`を実装していません。pipを CodeArtifactのリポジトリで設定した場合、`pip search`を実行するとPyPIからパッケージを検索して表示します。

サポートされているクライアント側コマンド

これらのコマンドはリポジトリとの直接的なやりとりを必要としないため、CodeArtifactはそのサポートのために何もする必要はありません。

コマンド	説明
uninstall (アンインストール)	パッケージをアンインストールする
フリーズ	インストール済みパッケージを要件形式で出力します。
list	インストールされているパッケージを一覧表示します。
show	インストールされたパッケージに関する情報を表示します。
チェック	インストールされているパッケージに互換性のある依存関係があることを確認します。
config	ローカルおよびグローバル設定を管理します。
ホイール	要件からホイールを構築します。
ハッシュ	パッケージアーカイブのハッシュを計算します。
完了	コマンド補完に役立ちます。
debug	デバッグ時に便利な情報を表示します。
help	コマンドのヘルプを表示します。

アップストリームと外部接続からの Python パッケージのリクエスト

Python パッケージバージョンを pypi.org からインポートすると、CodeArtifact はそのパッケージバージョン内のすべてのアセットをインポートします。ほとんどの Python パッケージには少数のアセットが含まれていますが、複数のハードウェアアーキテクチャと Python インタープリタをサポートするために 100 を超えるアセットを含むパッケージもあります。

既存のパッケージバージョン用の新しいアセットは pypi.org で頻繁に公開されています。例えば、Python の新しいバージョンがリリースされる際に、新しいアセットを公開するいくつかのプロジェクトがあります。`pip install` を使用して、Python パッケージを CodeArtifact からインストールすると、CodeArtifact リポジトリに保持されているパッケージバージョンが、pypi.org からの最新のアセットセットを反映するように更新されます。

同様に、現在の CodeArtifact リポジトリに存在しない、アップストリームの CodeArtifact リポジトリにあるパッケージバージョン用の新しいアセットが使用できる場合、それらは `pip install` の実行時に現在のリポジトリに保持されます。

削除されたパッケージバージョン

pypi.org の一部のパッケージバージョンは yanked とマークされています。これは、パッケージインストーラー (pip など) に、バージョン指定子と一致する唯一のバージョンでない限り (== または === を使用して)、そのバージョンをインストールしてはならないことを示します。詳細については、「PEP_592」を参照してください。

CodeArtifact のパッケージバージョンが最初に pypi.org への外部接続から取得された場合、CodeArtifact リポジトリからパッケージバージョンをインストールすると、CodeArtifact はパッケージバージョンの更新された取得済みメタデータが pypi.org から取得されることを保証します。

パッケージバージョンが削除されているかどうかを確認する方法

CodeArtifact でパッケージバージョンが削除されているかどうかを確認するには、`pip install` *`packageName`*`===`*`packageVersion`* を使用してインストールを試みます。パッケージバージョンが削除されている場合、次のような警告メッセージが表示されます。

```
WARNING: The candidate selected for download or install is a yanked version
```

pypi.org でパッケージバージョンが削除されているかどうかを確認するには、`https://pypi.org/project/`*`packageName`*`/`*`packageVersion`*`/` でそのパッケージバージョンの pypi.org リストを確認してください。

プライベートパッケージに yanked ステータスを設定する

CodeArtifact は、CodeArtifact リポジトリに直接公開されたパッケージの削除されたメタデータの設定をサポートしていません。

CodeArtifact がパッケージバージョンの最新の削除済みメタデータまたはアセットを取得しないのはなぜですか?

通常、CodeArtifact は Python パッケージのバージョンが CodeArtifact リポジトリから取得されたときに、削除されたメタデータが pypi.org 上の最新の値で更新されていることを確認します。さらに、パッケージバージョンのアセットのリストも、pypi.org とアップストリームの CodeArtifact リポジトリで最新のセットにより、最新の状態に保たれます。これは、パッケージバージョンを初めてインストールし、CodeArtifact が pypi.org から CodeCodeArtifact リポジトリにインポートした場合にも、以前にパッケージをインストールした場合にも当てはまります。ただし、pip などのパッケージマネージャークライアントが、削除された最新のメタデータを pypi.org またはアップストリームレポジトリから取得しない場合があります。代わりに、CodeArtifact はリポジトリに既に保存されているデータを返します。このセクションでは、これが発生する 3 つのケースについて説明します。

アップストリーム設定: disassociate-external-connection を使用してレポジトリまたはそのアップストリームから pypi.org への外部接続を削除すると、削除されたメタデータは pypi.org から更新されなくなります。同様に、アップストリームのリポジトリを削除すると、削除されたリポジトリと削除されたリポジトリのアップストリームのアセットは、現在のリポジトリでは利用できなくなります。CodeArtifact のパッケージオリジンコントロールを使用して特定のパッケージの新しいバージョンがプルされないようにする場合も同様です。`upstream=BLOCK` 設定は削除されたメタデータの更新をブロックします。

パッケージバージョンのステータス: パッケージバージョンのステータスを `Published` または `Unlisted` 以外に設定した場合、削除されたメタデータおよびパッケージバージョンのアセットは更新されません。同様に、特定のパッケージバージョン (`torch 2.0.1` など) を取得しようとしていて、同じパッケージバージョンがアップストリームリポジトリに存在し、ステータスが `Published` または `Unlisted` でない場合も、削除されたメタデータとアセットは、アップストリームレポジトリから現在のレポジトリに伝播されません。これは、他のパッケージバージョンのステータスは、そのバージョンがどのリポジトリでも使用されないことを示しているためです。

直接公開: 特定のパッケージバージョンを CodeArtifact リポジトリに直接公開すると、削除されたメタデータおよアセットは、アップストリームレポジトリおよび pypi.org からパッケージに対して更新されません。例えば、Web ブラウザーを使用して `torch 2.0.1` などのパッケージバージョン `torch-2.0.1-cp311-none-macosx_11_0_arm64.whl` からアセットをダウンロードし、twine を `torch 2.0.1` として使用して CodeArtifact リポジトリに公開する場合です。CodeArtifact は、pypi.org またはアップストリームレポジトリへの外部接続からではなく、リポジトリへの直接公開によってパッケージバージョンがドメインに入ったことを追跡します。この場合、CodeArtifact は削除されたメタデータをアップストリームのリポジトリまたは pypi.org と同期しません。これは、アップストリームリポジトリに `torch 2.0.1` を公開する場合も同様です。パッケージバージョン

があると、削除されたメタデータとアセットがアップストリームグラフの下位にあるリポジトリに伝播されません。

Ruby で CodeArtifact を使用する

これらのトピックで説明するのは、CodeArtifact で RubyGems と Bundler ツールを使用して、Ruby の gem をインストールし、公開する方法です。

> **Note**
>
> CodeArtifact では Ruby 3.3 以降の使用が推奨されます。Ruby 2.6 以前では動作しません。

トピック

CodeArtifact で RubyGems と Bundler を設定して使用する

CodeArtifact でリポジトリを作成したら、RubyGems (`gem`) と Bundler (`bundle`) を使用して gem をインストールして公開できます。このトピックでは、CodeArtifact リポジトリの認証と使用のためにパッケージマネージャーを設定する方法について説明します。

CodeArtifact で RubyGems (**`gem`**) と Bundler (**`bundle`**) を設定する

RubyGems (`gem`) または Bundler (`bundle`) を使用して、 AWS CodeArtifact に gem を発行したり、CodeArtifact から gem を使用したりするには、まず CodeArtifact リポジトリ情報を使用してそれらを設定する必要があります。これには、アクセスするための認証情報が含まれます。次のいずれかの手順に沿って、CodeArtifact リポジトリのエンドポイント情報と認証情報を使用して `gem` および `bundle` CLI ツールを設定します。

コンソールの手順を使用して RubyGems と Bundler を設定する

コンソールの設定手順を使用して、Ruby パッケージマネージャーを CodeArtifact リポジトリに接続できます。コンソールの手順にはカスタムコマンドが用意されており、CodeArtifact 情報を検索して入力することなく、パッケージマネージャーをセットアップできます。

1. https://console.aws.amazon.com/codesuite/codeartifact/home で AWS CodeArtifact コンソールを開きます。

2. ナビゲーションペインで、[リポジトリ] をクリックし、Ruby の gem のインストールまたはプッシュに使用するリポジトリの名前を選択します。
3. [接続手順の表示] を選択します。
4. オペレーティングシステムを選択します。
5. CodeArtifact リポジトリで設定する Ruby パッケージマネージャークライアントを選択します。
6. 生成された手順に従って、Ruby gem をリポジトリからインストールするか、Ruby gem をリポジトリに公開するようにパッケージマネージャークライアントを設定します。

RubyGems と Bundler を手動で設定する

コンソールの設定手順を使用できない場合、または使用しない場合は、次の手順を使用して、手動で Ruby パッケージマネージャーを CodeArtifact リポジトリに接続できます。

1. コマンドラインで以下のコマンドを使用して、CodeArtifact 認可トークンを取得し、認可トークンを環境変数に保存します。

   - *my_domain* を CodeArtifact ドメイン名に置き換えます。
   - *111122223333* をドメイン所有者の AWS アカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

   macOS and Linux

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

   Windows

   - Windows (デフォルトのコマンドシェルを使用):

     ```
     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. Ruby の gem をリポジトリに公開するには、次のコマンドを使用して CodeArtifact リポジトリのエンドポイントを取得し、RUBYGEMS_HOST 環境変数に保存します。gem CLI はこの環境変数を使用して、gem が公開される場所を決定します。

> **Note**
>
> または、RUBYGEMS_HOST 環境変数を使用する代わりに、gem push コマンドを使用するときにリポジトリエンドポイントに --host オプションを指定することもできます。

- *my_domain* を CodeArtifact ドメイン名に置き換えます。
- *111122223333* をドメイン所有者の AWS アカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
- *my_repo* を CodeArtifact リポジトリ名で置き換えます。

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

次のコマンドは、リポジトリエンドポイントを取得し、末尾の / をトリミングして、環境変数に保存します。

- Windows (デフォルトのコマンドシェルを使用):

```
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("/")
```

次の URL は、リポジトリエンドポイントの例です。

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

> **Note**
>
> デュアルスタックエンドポイントを使用するには、codeartifact.*region*.on.aws エンドポイントを使用してください。

3. Ruby の gem をリポジトリに公開するには、~/.gem/credentials ファイルを編集して認証トークンを含めることで、RubyGems で CodeArtifact を認証する必要があります。~/.gem/ ディレクトリまたは ~/.gem/credentials ファイルが存在しない場合は、ディレクトリとファイルを作成します。

   macOS and Linux

```
echo ":codeartifact_api_key: Bearer $CODEARTIFACT_AUTH_TOKEN" >> ~/.gem/
credentials
```

   Windows

   - Windows (デフォルトのコマンドシェルを使用):

```
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. gem を使用してリポジトリから Ruby gem をインストールするには、リポジトリエンドポイント情報と認証トークンを .gemrc ファイルに追加する必要があります。グローバルファイル (~/.gemrc) またはプロジェクト .gemrc ファイルに追加できます。.gemrc に追加する必要がある CodeArtifact 情報は、リポジトリエンドポイントと認証トークンの組み合わせです。形式は次のとおりです。

```
https://aws:${CODEARTIFACT_AUTH_TOKEN}@my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/ruby/my_repo/
```

- 認証トークンには、前のステップで設定した CODEARTIFACT_AUTH_TOKEN 環境変数を使用できます。
- リポジトリエンドポイントを取得するには、前もって設定した RUBYGEMS_HOST 環境変数の値を読み取るか、次の get-repository-endpoint コマンドを使用し、必要に応じて値を置き換えます。

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-
owner 111122223333 --repository my_repo --format ruby --query repositoryEndpoint
 --output text
```

エンドポイントを取得したら、テキストエディタを使用して適切な位置に aws: ${CODEARTIFACT_AUTH_TOKEN}@ を追加します。リポジトリエンドポイントと認証トークンの文字列を作成したら、次のように echo コマンドを使用して .gemrc ファイルの :sources: セクションに追加します。

> **Warning**
>
> CodeArtifact は、gem sources -add コマンドを使用したソースとしてのリポジトリの追加をサポートしていません。ソースはファイルに直接追加する必要があります。

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 (デフォルトのコマンドシェルを使用):

```
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. Bundler を使用するには、次の `bundle config` コマンドを実行し、リポジトリエンドポイント URL と認証トークンを使用して Bundler を設定する必要があります。

   macOS and Linux

   ```
   bundle config $RUBYGEMS_HOST aws:$CODEARTIFACT_AUTH_TOKEN
   ```

   Windows

   - Windows (デフォルトのコマンドシェルを使用):

     ```
     bundle config %RUBYGEMS_HOST% aws:%CODEARTIFACT_AUTH_TOKEN%
     ```

   - Windows PowerShell：

     ```
     bundle config $Env:RUBYGEMS_HOST aws:$Env:CODEARTIFACT_AUTH_TOKEN
     ```

CodeArtifact リポジトリで設定した RubyGems (`gem`) と Bundler (`bundle`) を使用して Ruby の gem を公開し、そこから gem を使用できます。

CodeArtifact からの Ruby gem のインストール

gem または bundle CLI ツールを使用して CodeArtifact リポジトリから Ruby の gem をインストールするには、次の手順に従います。

gem で Ruby gem をインストールする

RubyGems (gem) CLI を使用すると、CodeArtifact リポジトリから特定のバージョンの Ruby gem をすばやくインストールできます。

gem を使用して CodeArtifact リポジトリから Ruby gem をインストールする方法

1. まだ gem CLI を設定していない場合は、「CodeArtifact で RubyGems (gem) と Bundler (bundle) を設定する」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して設定します。

   > **Note**
   >
   > 認可トークンの有効期限は 12 時間です。トークンの作成後 12 時間が経過した場合は、新しいトークンを作成する必要があります。

2. 次のコマンドを使用して、CodeArtifact から Ruby gem をインストールします。

```
gem install my_ruby_gem --version 1.0.0
```

bundle で Ruby gem をインストールする

Bundler (bundle) CLI を使用して、Gemfile で設定された Ruby gem をインストールできます。

bundle を使用して CodeArtifact リポジトリから Ruby gem をインストールする方法

1. まだ bundle CLI を設定していない場合は、「CodeArtifact で RubyGems (gem) と Bundler (bundle) を設定する」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して設定します。

> **Note**
>
> 認可トークンの有効期限は 12 時間です。トークンの作成後 12 時間が経過した場合は、新しいトークンを作成する必要があります。

2. CodeArtifact リポジトリエンドポイント URL を `source` として `Gemfile` に追加し、CodeArtifact リポジトリとそのアップストリームから設定済みの Ruby gem をインストールします。

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

gem 'my_ruby_gem'
```

3. 次のコマンドを使用して、`Gemfile` で指定された Ruby gem をインストールします。

```
bundle install
```

CodeArtifact への Ruby gem の公開

gem CLI を使用して Ruby の gem を CodeArtifact リポジトリに公開するには、次の手順に従います。

1. まだ gem CLI を設定していない場合は、「CodeArtifact で RubyGems (gem) と Bundler (bundle) を設定する」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して設定します。

> **Note**
>
> 認可トークンの有効期限は 12 時間です。トークンの作成後 12 時間が経過した場合は、新しいトークンを作成する必要があります。

2. 次のコマンドを使用して、Ruby の gem を CodeArtifact リポジトリに公開します。`RUBYGEMS_HOST` 環境変数を設定していない場合は、`--host` オプションで CodeArtifact リポジトリエンドポイントを指定する必要があります。

```
gem push --key codeartifact_api_key my_ruby_gem-0.0.1.gem
```

RubyGems コマンドのサポート

CodeArtifact は、`gem install` コマンドと `gem push` コマンドをサポートしています。

> **Note**
>
> `gem install` コマンドを CodeArtifact で使用するには、RubyGems バージョン 3.5.18 以前が必要です。RubyGems バージョン 3.5.19 以降では、Gem を解決するために Compact Index API が必要ですが、CodeArtifact は現在サポートしていません。RubyGems 3.5.19 以降を使用している場合は、代わりに Bundler を使用して Gem をインストールするか、 を実行して RubyGems をダウングレードしますgem update --system 3.5.18。

CodeArtifact は、次の `gem` コマンドをサポートしていません。

- `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 の互換性

このガイドには、CodeArtifact の Bundler との互換性に関する情報が含まれています。

Bundler の互換性

AWS CodeArtifact は Bundler 2.4.11 以降を推奨しています。インストールで問題が発生した場合は、Bundler CLI を最新バージョンに更新してください。

Bundler バージョンのサポート

2.4.11 より前のバージョンの Bundler では、Gemfile で定義できる依存関係は 500 に制限されており、500 を超えると Bundler がインデックス全体のクエリを決定します (`specs.4.8.gz`)。CodeArtifact ではフルインデックスがサポートされていません。そのため、2.4.11 より前の Bundler バージョンを使用する場合、500 を超える依存関係を指定すると CodeArtifact では機能しません。

CodeArtifact を使用して Gemfile に 500 を超える依存関係を定義するには、Bundler をバージョン 2.4.11 以降に更新してください。

Bundler オペレーションのサポート

CodeArtifact による RubyGems のサポートに Bundler Compact Index API は含まれません (`/versions` API はサポートされていません)。CodeArtifact は Dependencies API のみをサポートします。

コンパクトインデックスはサポートされていないため、Bundler は 1 つのリクエストで複数の Gem 名を送信する Dependencies API (`/api/v1/dependencies`) を使用して Gem を解決します。リクエストの各 Gem 名は、アカウントの 1 秒あたりの読み取りリクエストクォータに対する個別のリクエストとしてカウントされます。たとえば、Bundler が 20 個の gem 名を含む依存関係リクエストを送信する場合、クォータに対して 20 個のリクエストとしてカウントされます。これにより、HTTP リクエスト数が設定された制限を大幅に下回っているように見える場合でも、同時実行性が高い CI/CD 環境でスロットリングが発生する可能性があります。Ruby Gem の解決中にスロットリングが発生した場合は、1 つの AWS アカウントから 1 秒あたりの読み取りリクエストのクォータの引き上げをリクエストします。詳細については、「AWS CodeArtifact のクォータ」を参照してください。

また、CodeArtifact は各種仕様 API (`specs.4.8.gz` など) をサポートしていません。

Swift で CodeArtifact を使う

以下のトピックでは、Swift パッケージマネージャーを CodeArtifact で使用して、Swift パッケージをインストールし公開する方法について説明します。

> **Note**
>
> CodeArtifact は、Swift 5.8 以降、および Xcode 14.3 以降をサポートしています。CodeArtifact では、Swift 5.9 以降、および Xcode 15 以降の使用を推奨します。

トピック

CodeArtifact での Swift パッケージマネージャーの設定

Swift Package Manager を使用して AWS CodeArtifact にパッケージを発行したり、CodeArtifact からパッケージを使用したりするには、まず CodeArtifact リポジトリにアクセスするための認証情報を設定する必要があります。CodeArtifact の認証情報とリポジトリエンドポイントに Swift パッケージマネージャー CLI を設定するには、`aws codeartifact login` コマンドの使用を推奨します。Swift パッケージマネージャーは、手動で設定することもできます。

login コマンドを使用して Swift を設定する

`aws codeartifact login` コマンドを使用して、CodeArtifact で Swift パッケージマネージャーを設定します。

> **Note**
>
> login コマンドを使用するには Swift 5.8 以降が必要で、Swift 5.9 以降の使用を推奨します。

`aws codeartifact login` コマンドは、次の操作を行います。

1. CodeArtifact から認証トークンを取得して、認証トークンを環境変数に保存します。認証情報の保存方法は、環境のオペレーティングシステムによって異なります。

   a. macOS: macOS キーチェーンアプリケーションにエントリが作成されます。

   b. Linux および Windows: ~/.netrc ファイルにエントリが作成されます。

   どちらのオペレーティングシステムの場合でも、認証情報エントリが存在する場合、このコマンドは既存のエントリを新しいトークンに置き換えます。

2. CodeArtifact リポジトリのエンドポイント URL を取得し、Swift 設定ファイルに追加します。このコマンドは、リポジトリのエンドポイント URL を、/path/to/project/.swiftpm/configuration/registries.json にあるプロジェクトレベルの設定ファイルに追加します。

> **Note**
>
> aws codeartifact login コマンドは、実行が必要な swift package-registry コマンドを、Package.swift ファイルを含むディレクトリから呼び出します。そのため、aws codeartifact login コマンドは Swift プロジェクト内から実行する必要があります。

login コマンドを使用して Swift を設定するには

1. プロジェクトの Package.swift ファイルが含まれている Swift プロジェクトディレクトリに移動します。
2. 次の aws codeartifact login コマンドを実行します。

   所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

```
aws codeartifact login --tool swift --domain my_domain \
--domain-owner 111122223333 --repository my_repo \
[--namespace my_namespace]
```

--namespace オプションは、CodeArtifact リポジトリのパッケージが指定された名前空間にある場合にのみ、パッケージを使用するようにアプリケーションを設定します。CodeArtifact の名前空間はスコープと同義であり、コードを論理グループに整理したり、コードベースに複数のライブラリが含まれている場合に発生する可能性のある名前の競合を防ぐために使用されます。

login を呼び出した後のデフォルトの認可時間は 12 時間であり、トークンを定期的に更新するには、login を呼び出す必要があります。login コマンドで作成される認可トークンの詳細については、「loginコマンドで作成されたトークン」を参照してください。

login コマンドを使用せずに Swift を設定する

aws codeartifact login コマンドを使用して Swift を設定することを推奨しますが、Swift パッケージマネージャーの設定を手動で更新することで、login コマンドを使用せずに Swift パッケージマネージャーを設定することもできます。

次の手順では、 AWS CLI を使用して以下を実行します。

1. CodeArtifact から認証トークンを取得して、認証トークンを環境変数に保存します。認証情報の保存方法は、環境のオペレーティングシステムによって異なります。
   a. macOS: macOS キーチェーンアプリケーションにエントリが作成されます。
   b. Linux および Windows: ~/.netrc ファイルにエントリが作成されます。
2. CodeArtifact リポジトリのエンドポイント URL を取得します。
3. ~/.swiftpm/configuration/registries.json 設定ファイルに、リポジトリのエンドポイント URL と認証タイプを含むエントリを追加します。

login コマンドを使用せずに Swift を設定するには

1. コマンドラインで以下のコマンドを使用して、CodeArtifact 認可トークンを取得し、認可トークンを環境変数に保存します。

   - *my_domain* を CodeArtifact ドメイン名に置き換えます。
   - *111122223333* をドメイン所有者の AWS アカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

   macOS and Linux

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

Windows

- Windows (デフォルトのコマンドシェルを使用):

```
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. 次のコマンドを実行して、CodeArtifact リポジトリのエンドポイントを取得します。リポジトリエンドポイントは、パッケージを使用または公開するリポジトリを Swift パッケージマネージャーに示すために使用されます。

- *my_domain* を CodeArtifact ドメイン名に置き換えます。
- *111122223333* をドメイン所有者の AWS アカウント ID に置き換えます。所有しているドメインのリポジトリにアクセスする場合、--domain-ownerを含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。
- *my_repo* を CodeArtifact リポジトリ名で置き換えます。

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 (デフォルトのコマンドシェルを使用):

```
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
```

次の URL は、リポジトリエンドポイントの例です。

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

> **Note**
>
> デュアルスタックエンドポイントを使用するには、codeartifact.*region*.on.aws エンドポイントを使用してください。

> **Important**
>
> Swift パッケージマネージャーの設定に使用する場合は、リポジトリの URL エンドポイントの末尾に login を追加する必要があります。これはこの手順のコマンドで自動的に行われます。

3. これら 2 つの値を環境変数に格納したら、以下のように swift package-registry login コマンドを使用して Swift に渡します。

   macOS and Linux

   ```
   swift package-registry login ${CODEARTIFACT_REPO}login --token
    ${CODEARTIFACT_AUTH_TOKEN}
   ```

   Windows

   - Windows (デフォルトのコマンドシェルを使用):

     ```
     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. 次に、アプリケーションが使用するパッケージレジストリを更新して、依存関係が CodeArtifact リポジトリから取得されるようにします。このコマンドは、パッケージの依存関係を解決しようとしているプロジェクトディレクトリで実行する必要があります。

   macOS and Linux

   ```
   $ swift package-registry set ${CODEARTIFACT_REPO} [--scope my_scope]
   ```

   Windows

   - Windows (デフォルトのコマンドシェルを使用):

     ```
     $ swift package-registry set %CODEARTIFACT_REPO% [--scope my_scope]
     ```

   - Windows PowerShell：

     ```
     $ swift package-registry set $Env:CODEARTIFACT_REPO [--scope my_scope]
     ```

   `--scope` オプションは、CodeArtifact リポジトリのパッケージが指定されたスコープにある場合にのみ、パッケージを使用するようにアプリケーションを設定します。スコープは CodeArtifact の名前空間と同義であり、コードを論理グループに整理したり、コードベースに複数のライブラリが含まれている場合に発生する可能性のある名前の競合を防ぐために使用されます。

5. プロジェクトディレクトリで以下のコマンドを実行しプロジェクトレベルの `.swiftpm/configuration/registries.json` ファイルの内容を表示することで、正しく設定されたことを確認できます。

   ```
   $ 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
}
```

CodeArtifact リポジトリでの Swift パッケージマネージャーの設定は以上です。これで Swift パッケージマネージャーを使用して Swift パッケージをリポジトリに公開したり、リポジトリから Swift パッケージを使用したりできるようになります。詳細については、「Swift パッケージの使用と公開」を参照してください。

Swift パッケージの使用と公開

CodeArtifact で Swift パッケージを使用する

次の手順を使用して、 AWS CodeArtifact リポジトリから Swift パッケージを使用します。

CodeArtifact リポジトリで Swift パッケージを使用するには

1. まだ設定していない場合は、「CodeArtifact での Swift パッケージマネージャーの設定」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して Swift パッケージマネージャーを設定します。

   > **Note**
   >
   > 認可トークンの有効期限は 12 時間です。トークンの作成後 12 時間が経過した場合は、新しいトークンを作成する必要があります。

2. アプリケーションプロジェクトフォルダー内の `Package.swift` ファイルを編集して、プロジェクトで使用されるパッケージの依存関係を更新します。

   a. `Package.swift` ファイルに `dependencies` セクションが含まれていない場合は、セクションを追加してください。

   b. `Package.swift`ファイルの `dependencies` セクションにパッケージ ID を追加して、使用するパッケージを追加します。パッケージ ID は、スコープとパッケージ名をピリオドで区切ったものです。例については、後のステップのコードスニペットを参照してください。

> **Tip**
>
> パッケージ ID は、CodeArtifact コンソールで確認できます。使用する特定のパッケージバージョンを探して、パッケージバージョンページの [インストールショートカット] の指示を参照してください。

c. `Package.swift` ファイルに `targets` セクションが含まれていない場合は、セクションを追加してください。

d. `targets` セクションに、依存関係の使用に必要なターゲットを追加します。

以下のスニペットは、`Package.swift` ファイル内の設定済みの `dependencies` セクションおよび `targets` セクションを示すスニペットの例です。

```
...
    ],
    dependencies: [
        .package(id: "my_scope.package_name", from: "1.0.0")
    ],
    targets: [
      .target(
         name: "MyApp",
         dependencies: ["package_name"]
      ),...
    ],
...
```

3. 設定が完了したので、次のコマンドを使用して CodeArtifact からパッケージの依存関係をダウンロードします。

```
swift package resolve
```

CodeArtifact の Swift パッケージを Xcode で使用する

CodeArtifact リポジトリの Swift パッケージを Xcode で使用するには、次の手順に従います。

CodeArtifact リポジトリの Swift パッケージを Xcode で使用するには

1. まだ設定していない場合は、「CodeArtifact での Swift パッケージマネージャーの設定」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して Swift パッケージマネージャーを設定します。

   > **Note**
   >
   > 認可トークンの有効期限は 12 時間です。トークンの作成後 12 時間が経過した場合は、新しいトークンを作成する必要があります。

2. パッケージを Xcode のプロジェクトに依存関係として追加します。

   a. [ファイル] > [パッケージを追加] と選択します。
   b. 検索バーを使用してパッケージを検索します。検索では `package_scope.package_name` の形式を使用します。
   c. パッケージが見つかったら、パッケージを選択して、[パッケージを追加] を選択します。
   d. パッケージの確認後、依存関係として追加するパッケージを選択し、[パッケージを追加] を選択します。

CodeArtifact リポジトリを Xcode で使用する際に問題が発生した場合、「Swift のトラブルシューティング」を参照して一般的な問題と考えられる修正方法を確認してください。

Swift パッケージを CodeArtifact に公開する

Swift パッケージを公開するには、CodeArtifact で Swift 5.9 以降および `swift package-registry publish` コマンドの使用を推奨します。以前のバージョンを使用している場合、Swift パッケージを CodeArtifact に公開するには、Curl コマンドを使用する必要があります。

`swift package-registry publish` コマンドを使用した CodeArtifact パッケージの公開

Swift 5.9 以降で以下の手順を実行し、Swift パッケージマネージャーを使用して Swift パッケージを CodeArtifact リポジトリに公開します。

1. まだ設定していない場合は、「CodeArtifact での Swift パッケージマネージャーの設定」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して Swift パッケージマネージャーを設定します。

> **Note**
>
> 認可トークンの有効期限は 12 時間です。作成後 12 時間が経過した場合は、新しいトークンを作成する必要があります。

2. パッケージの Package.swift ファイルが含まれている Swift プロジェクトディレクトリに移動します。
3. 次の swift package-registry publish コマンドを実行して、パッケージを公開します。このコマンドはパッケージソースアーカイブを作成して、CodeArtifact リポジトリに公開します。

```
swift package-registry publish packageScope.packageName packageVersion
```

例えば、次のようになります。

```
swift package-registry publish myScope.myPackage 1.0.0
```

4. パッケージが公開され、リポジトリに存在することを確認するには、コンソールにチェックインするか、以下のように aws codeartifact list-packages コマンドを使用します。

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

以下のように aws codeartifact list-package-versions コマンドを使用して、パッケージの単一バージョンを一覧表示できます。

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

Curl を使用した CodeArtifact パッケージの公開

Swift 5.9 以降で使用可能な swift package-registry publish コマンドの使用を推奨しますが、Curl を使用して Swift パッケージを CodeArtifact に公開することもできます。

Curl を使用して Swift パッケージを AWS CodeArtifact リポジトリに発行するには、次の手順に従います。

1. まだ行っていない場合は、「CodeArtifact での Swift パッケージマネージャーの設定」の手順に従って CODEARTIFACT_AUTH_TOKEN 環境変数および CODEARTIFACT_REPO 環境変数を作成して更新します。

   > **Note**
   >
   > 認可トークンは 12 時間有効です。CODEARTIFACT_AUTH_TOKEN 環境変数が作成されてから 12 時間が経過した場合は、新しい認証情報で環境変数を更新する必要があります。

2. Swift パッケージを作成していない場合は、最初に以下のコマンドを実行してパッケージを作成します。

   ```
   mkdir testDir && cd testDir
   swift package init
   git init .
   swift package archive-source
   ```

3. 次の Curl コマンドを使用して、Swift パッケージを CodeArtifact に公開します。

   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. パッケージが公開され、リポジトリに存在することを確認するには、コンソールにチェックインするか、以下のように aws codeartifact list-packages コマンドを使用します。

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

以下のように aws codeartifact list-package-versions コマンドを使用して、パッケージの単一バージョンを一覧表示できます。

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

GitHub から Swift パッケージを取得して CodeArtifact に再度公開する

以下の手順を使用して GitHub から Swift パッケージを取得し、CodeArtifact リポジトリに再度公開します。

GitHub から Swift パッケージを取得して CodeArtifact に再度公開するには

1. まだ設定していない場合は、「CodeArtifact での Swift パッケージマネージャーの設定」の手順に従って、CodeArtifact リポジトリを使用するように適切な認証情報を使用して Swift パッケージマネージャーを設定します。

   > **Note**
   >
   > 認可トークンの有効期限は 12 時間です。トークンの作成後 12 時間が経過した場合は、新しいトークンを作成する必要があります。

2. 次の git clone コマンドを使用して、取得して再公開する Swift パッケージの git リポジトリを複製します。GitHub リポジトリの複製については、GitHub ドキュメントの「Cloning a repository」を参照してください。

   ```
   git clone repoURL
   ```

3. 複製したリポジトリに移動します。

   ```
   cd repoName
   ```

4. パッケージを作成して、CodeArtifact に公開します。

   a. 推奨: Swift 5.9 以降を使用している場合は、次の swift package-registry publish コマンドを使用してパッケージを作成し、設定した CodeArtifact リポジトリに公開できます。

```
swift package-registry publish packageScope.packageName versionNumber
```

例えば、次のようになります。

```
swift package-registry publish myScope.myPackage 1.0.0
```

b. 5.9 より前のバージョンの Swift を使用している場合は、`swift archive-source` コマンドを使用してパッケージを作成し、その後 Curl コマンドを使用してパッケージを公開する必要があります。

i. CODEARTIFACT_AUTH_TOKEN 環境変数および CODEARTIFACT_REPO 環境変数を設定していない場合、または設定してから 12 時間以上経過している場合は、「login コマンドを使用せずに Swift を設定する」の手順に従います。

ii. `swift package archive-source` コマンドを使用して Swift パッケージを作成します。

```
swift package archive-source
```

正常に作成されたら、コマンドラインに `Created package_name.zip` が表示されます。

iii. 次の Curl コマンドを使用して、Swift パッケージを CodeArtifact に公開します。

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. パッケージが公開され、リポジトリに存在することを確認するには、コンソールにチェックインするか、以下のように `aws codeartifact list-packages` コマンドを使用します。

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

以下のように aws codeartifact list-package-versions コマンドを使用して、パッケージの単一バージョンを一覧表示できます。

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

Swift パッケージ名と名前空間の正規化

CodeArtifact は、パッケージ名と名前空間を保存する前に正規化します。つまり、CodeArtifact 内の名前は、パッケージが公開されたときに提供されたものとは異なる場合があります。

パッケージ名と名前空間の正規化: CodeArtifact は、すべての文字を小文字に変換することで Swift パッケージ名と名前空間を正規化します。

パッケージバージョンの正規化: CodeArtifact は Swift パッケージバージョンを正規化しません。CodeArtifact はセマンティックバージョニング 2.0 バージョンパターンのみをサポートしていることに注意してください。セマンティックバージョニングの詳細については、「Semantic Versioning 2.0.0」を参照してください。

CodeArtifact はこれらのリクエストの入力を正規化するため、正規化されていないパッケージ名と名前空間は API および CLI リクエストで使用できます。例えば、--package myPackage および --namespace myScope の入力は正規化され、正規化されたパッケージ名 mypackage と名前空間 myscope が返されます。

IAM ポリシーなどの ARN では正規化された名前を使用する必要があります。

正規化されたパッケージ名を検索するには、aws codeartifact list-packages コマンドを使用します。詳細については、「パッケージ名を一覧表示する」を参照してください。

Swift のトラブルシューティング

以下の情報は、Swift および CodeArtifact での一般的な問題のトラブルシューティングに役立ちます。

Swift パッケージマネージャーを設定した後も Xcode で 401 エラーが表示される

問題: CodeArtifact リポジトリから Xcode の Swift プロジェクトへの依存関係としてパッケージを追加しようとすると、Swift を CodeArtifact に接続するための指示に従った後でも、401 不正エラーが表示される。

考えられる解決方法: CodeArtifact の認証情報が保存されている macOS キーチェーンアプリケーションの問題が原因である可能性があります。この問題を解決するには、キーチェーンアプリケーションを開いて CodeArtifact のエントリをすべて削除し、「CodeArtifact での Swift パッケージマネージャーの設定」の手順に従って Swift パッケージマネージャーを CodeArtifact リポジトリで再度設定することをお勧めします。

パスワードのキーチェーンプロンプトが原因で CI マシンで Xcode が ハングする

問題: GitHub Actions などの継続的インテグレーション (CI) サーバー上の Xcode ビルドの一部として CodeArtifact から Swift パッケージをプルしようとすると、CodeArtifact による認証がハングし、最終的に失敗して次のようなエラーメッセージが表示されることがあります。

```
Failed to save credentials for
\'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com\'
to keychain: status -60008
```

解決方法: これは、認証情報が CI マシンのキーチェーンに保存されておらず、Xcode がキーチェーンに保存されている認証情報のみをサポートしていることが原因です。この問題を修正するには、次の手順を使用してキーチェーンエントリを手動で作成することをお勧めします。

1. キーチェーンを準備します。

```
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. CodeArtifact 認証トークンとリポジトリエンドポイントを取得します。

```
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. Keychain エントリを手動で作成します。

```
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}"
```

このエラーと解決方法の詳細については、https://github.com/apple/swift-package-manager/issues/7236 を参照してください。

CodeArtifact でのジェネリックパッケージの使用

以下のトピックでは、 AWS CodeArtifact を使用して、ジェネリックパッケージを使用および公開する方法について説明します。

トピック

ジェネリックパッケージの概要

`generic` パッケージ形式を使用すると、任意のタイプのファイルをアップロードして CodeArtifact リポジトリにパッケージを作成できます。ジェネリックパッケージは、特定のプログラミング言語、ファイルタイプ、またはパッケージ管理エコシステムには関連付けられていません。これは、アプリケーションインストーラー、機械学習モデル、設定ファイルなど、任意のビルドアーティファクトの保存とバージョン管理に使用できます。

ジェネリックパッケージは、パッケージ名、名前空間、バージョン、および 1 つ以上のアセット (またはファイル) で構成されます。ジェネリックパッケージは、他の形式のパッケージと一緒に単一の CodeArtifact リポジトリに存在できます。

AWS CLI または SDK を使用して汎用パッケージを操作できます。汎用パッケージで動作する AWS CLI コマンドの完全なリストについては、「」を参照してくださいジェネリックパッケージでサポートされるコマンド。

ジェネリックパッケージの制約

- アップストリームリポジトリからは取得できません。ジェネリックパッケージは公開先のリポジトリからのみ取得できます。
- ListPackageVersionDependencies から返される依存関係を宣言したり、 AWS マネジメントコンソール に表示したりすることはできません。
- ジェネリックパッケージには README ファイルと LICENSE ファイルを保存できますが、CodeArtifact はこれらのファイルを読み取ることはできません。これらのファイル内の情報は GetPackageVersionReadme や DescribePackageVersion では返されず、 AWS マネジメントコンソールにも表示されません。

- CodeArtifact のすべてのパッケージと同様に、アセットのサイズとパッケージあたりのアセット数には制限があります。CodeArtifact での制限とクォータの詳細については、「AWS CodeArtifact のクォータ」を参照してください。
- ジェネリックパッケージに含まれるアセット名は、以下の規則に従う必要があります。
  - アセット名には Unicode の文字と数字を使用できます。具体的には、小文字 (Ll)、修飾文字 (Lm)、その他の文字 (Lo)、タイトルケース文字 (Lt)、大文字 (Lu)、文字番号 (Nl)、および 10 進数 (Nd) の Unicode 文字カテゴリを使用できます。
  - 次の特殊文字を使用できます。`~!@^&()-_+[]{};,.`
  - `.` や `..` はアセット名に使用できません。
  - 使用できる空白文字はスペースのみです。アセット名の先頭や末尾にスペースを含めることはできません。また、連続するスペースを含めることはできません。

ジェネリックパッケージでサポートされるコマンド

AWS CLI または SDK を使用して汎用パッケージを操作できます。ジェネリックパッケージでは、以下の CodeArtifact コマンドを使用できます。

- copy-package-versions (「リポジトリ間でのパッケージのコピー」を参照)
- delete-package (「パッケージの削除 (AWS CLI)」を参照)
- delete-package-versions (「パッケージバージョンの削除 (AWS CLI)」を参照)
- describe-package
- describe-package-version (「パッケージのバージョンの詳細と依存関係の表示および更新」を参照)
- dispose-package-versions (「パッケージバージョンの廃棄」を参照)
- get-package-version-asset (「パッケージバージョンアセットのダウンロード」を参照)
- list-package-version-assets (「パッケージバージョンのアセットを一覧表示する」を参照)
- list-package-versions (「パッケージバージョンを一覧表示する」を参照)
- list-packages (「パッケージ名を一覧表示する」を参照)
- publish-package-version (「ジェネリックパッケージの公開」を参照)
- put-package-origin-configuration (「パッケージオリジンコントロールの編集」を参照)

> **Note**
>
> publish オリジンコントロール設定を使用して、リポジトリ内のジェネリックパッケージ名の公開を許可または禁止できます。ただし、ジェネリックパッケージはアップストリームリポジトリから取得できないため、upstream 設定はジェネリックパッケージには適用されません。

- update-package-versions-status (「パッケージバージョンのステータスの更新」を参照)

ジェネリックパッケージの公開と使用

ジェネリックパッケージバージョンとそれに関連するアセットを公開するには、publish-package-version コマンドを使用します。list-package-version-asset コマンドを使用してジェネリックパッケージのアセットを一覧表示し、get-package-version-asset を使用してダウンロードします。以下のトピックでは、これらのコマンドを使用してジェネリックパッケージを公開したり、ジェネリックパッケージアセットをダウンロードしたりする手順について説明します。

ジェネリックパッケージの公開

ジェネリックパッケージは、パッケージ名、名前空間、バージョン、および 1 つ以上のアセット (またはファイル) で構成されます。このトピックでは、my-package という名前のパッケージを、名前空間 my-ns、バージョン 1.0.0、asset.tar.gz という名前の 1 つのアセットを含むパッケージを公開する方法を示します。

前提条件:

- CodeArtifact AWS Command Line Interface を使用して をセットアップおよび設定する (「」を参照AWS CodeArtifact でのセットアップ)
- CodeArtifact ドメインとリポジトリを用意する (「AWS CLI を使用した開始方法」を参照)

ジェネリックパッケージを公開するには

1. 以下のコマンドを使用して、パッケージバージョンにアップロードする各ファイルの SHA256 ハッシュを生成し、その値を環境変数に入力します。この値は、最初に送信された後にファイルの内容が変更されていないことを確認するための整合性チェックに使用されます。

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. publish-package-version を呼び出して、アセットをアップロードし、新しいパッケージバージョンを作成します。

> **Note**
>
> パッケージに複数のアセットが含まれている場合は、アセットごとに publish-package-version を 1 回呼び出してアップロードします。最後のアセットをアップロードする場合を除き、publish-package-version を呼び出すたびに --unfinished 引数を含めます。--unfinished を省略するとパッケージバージョンのステータスが Published に設定され、追加のアセットがアップロードされなくなります。
>
> または、publish-package-version を呼び出すたびに --unfinished を含め、update-package-versions-status コマンドを使用してパッケージバージョンのステータスを Published に設定します。

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%
```

出力は以下のようになります。

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

ジェネリックパッケージアセットの一覧表示

ジェネリックパッケージに含まれるアセットを一覧表示するには、list-package-version-assets コマンドを使用します。詳細については、「パッケージバージョンのアセットを一覧表示する」を参照してください。

以下の例では、パッケージ my-package のバージョン 1.0.0 のアセットを一覧表示しています。

パッケージバージョンのアセットを一覧表示するには

- ジェネリックパッケージに含まれるアセットを一覧表示するには、list-package-version-assets を呼び出します。

  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
  ```

  出力は以下のようになります。

  ```
  {
      "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"
}
```

ジェネリックパッケージアセットのダウンロード

ジェネリックパッケージからアセットをダウンロードするには、get-package-version-asset コマンドを使用します。詳細については、「パッケージバージョンアセットのダウンロード」を参照してください。

次の例では、パッケージ my-package のバージョン 1.0.0 から、asset.tar.gz という名前のファイルとしてアセット asset.tar.gz を現在の作業ディレクトリにダウンロードします。

パッケージバージョンアセットをダウンロードするには

- get-package-version-asset を呼び出してジェネリックパッケージからアセットをダウンロードします。

  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
  ```

  出力は以下のようになります。

  ```
  {
      "assetName": "asset.tar.gz",
      "packageVersion": "1.0.0",
      "packageVersionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"
  }
  ```

CodeBuild で CodeArtiFact を使用する

これらのトピックでは、 AWS CodeBuild ビルドプロジェクトの CodeArtifact リポジトリでパッケージを使用する方法について説明します。

トピック

CodeBuild で npm パッケージを使用する

次のステップは、CodeBuild に用意されている Docker イメージ に記載されたオペレーティングシステムでテストされています。

IAM ロールを使用したアクセス許可の設定

これらのステップは、CodeBuild で CodeArtifact の npm パッケージを使用する場合に必要です。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで Roles (ロール) を選択します。ロールページで、CodeBuild 構築プロジェクトで使用されるロールを編集します。このロールには、以下のアクセス許可が必要です。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
```

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

> **Important**
>
> CodeBuild を使用してパッケージを公開する場合は、**codeartifact:PublishPackageVersion** アクセス許可 を追加します。

詳細については、IAM ユーザーガイド の ロールの変更 を参照してください。

ログインして npm を使う

CodeBuild の npm パッケージを使用するには、プロジェクトbuildspec.yaml の pre-build セクションから login コマンドを実行し、npm を設定して CodeArtifact からパッケージをフェッチします。詳細については、npm を使った認証 を参照してください。

login が正常に実行されたら、build セクションから npm コマンドを実行して npm パッケージをインストールできます。

Linux

> **Note**
>
> 古い CodeBuild イメージpip3 install awscli --upgrade --userを使用している場合にのみ、 AWS CLI で をアップグレードする必要があります。最新のイメージバージョンを使用している場合は、その行を削除できます。

```
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
```

Server

```
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
```

CodeBuild での Python パッケージの使用

次のステップは、CodeBuild に用意されている Docker イメージ に記載されているオペレーティングシステムでテストされています。

IAM ロールを使用したアクセス許可の設定

これらのステップは、CodeBuild で CodeArtifact の Python パッケージを使用する場合に必要です。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで Roles (ロール) を選択します。ロールページで、CodeBuild 構築プロジェクトで使用されるロールを編集します。このロールには、以下のアクセス許可が必要です。

   JSON

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

> **Important**
>
> CodeBuild を使用してパッケージを公開する場合は、**codeartifact:PublishPackageVersion** アクセス許可 を追加します。

詳細については、IAM ユーザーガイド の ロールの変更 を参照してください。

ログインして pip または twine を使う

CodeBuild の Python パッケージを使用するには、プロジェクトの `buildspec.yaml` ファイルの `pre-build` セクションから `login` コマンドを実行し、`pip` を設定し、CodeArtifact からパッケージをフェッチします。詳細については、PythonでCodeArtifactを使う を参照してください。

`login` が正常に実行されたら、`build` セクションから `pip` コマンドを実行して Python パッケージをインストールまたは公開できます。

Linux

> **Note**
>
> 古い CodeBuild イメージ`pip3 install awscli --upgrade --user`を使用している場合にのみ、 AWS CLI で をアップグレードする必要があります。最新のイメージバージョンを使用している場合は、その行を削除できます。

`pip` を使用して Python パッケージをインストールするには:

```
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
```

`twine` を使用して Python パッケージを公開するには:

```
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
```

Server

pip を使用して Python パッケージをインストールするには:

```
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
```

twine を使用して Python パッケージを公開するには:

```
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
```

CodeBuild で Maven パッケージを使用する

IAM ロールを使用したアクセス許可の設定

これらのステップは、CodeBuild で CodeArtifact の Maven パッケージを使用する場合に必要です。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで Roles (ロール) を選択します。ロールページで、CodeBuild 構築プロジェクトで使用されるロールを編集します。このロールには、以下のアクセス許可が必要です。

   JSON

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

> **Important**
>
> CodeBuild を使用してパッケージを公開する場合は、**codeartifact:PublishPackageVersion** および **codeartifact:PutPackageMetadata** アクセス許可を追加します。

詳細については、IAM ユーザーガイド の ロールの変更 を参照してください。

gradle または mvn を使用する

`gradle` または `mvn` で Maven パッケージを使用するには、CodeArtifact 認証トークンを環境変数に保存します (環境変数に認証トークンを渡すを参照)。以下に例を示します。

> **Note**
>
> 古い CodeBuild イメージ`pip3 install awscli --upgrade --user`を使用している場合にのみ、 AWS CLI で をアップグレードする必要があります。最新のイメージバージョンを使用している場合は、その行を削除できます。

```
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`
```

Gradle を使用するには:

Gradle `build.gradle` ファイルで `CODEARTIFACT_AUTH_TOKEN` 変数を参照した場合は、Gradle で CodeArtifact を使用する で説明されているように、Gradle 構築を`buildspec.yaml` `build` セクションから呼び出せます。

```
build:
  commands:
    - gradle build
```

mvn を使用するには:

mvn で CodeArtifact を使用する の手順に従って、Maven 設定ファイル (`settings.xml` および `pom.xml`) を設定する必要があります。

CodeBuild で NuGet パッケージを使用する

次のステップは、CodeBuild に用意されている Docker イメージ に記載されているオペレーティングシステムでテストされています。

トピック

- IAM ロールを使用したアクセス許可の設定
- NuGet パッケージを消費する
- NuGet パッケージで構築する
- NuGet パッケージを公開する

IAM ロールを使用したアクセス許可の設定

これらのステップは、CodeBuild で CodeArtifact の NuGet パッケージを使用する場合に必要です。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで Roles (ロール) を選択します。ロールページで、CodeBuild 構築プロジェクトで使用されるロールを編集します。このロールには、以下のアクセス許可が必要です。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
                      "codeartifact:ReadFromRepository"
                      ],
          "Resource": "*"
```

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

> **Important**
>
> CodeBuild を使用してパッケージを公開する場合は、**codeartifact:PublishPackageVersion** アクセス許可 を追加します。

詳細については、IAM ユーザーガイド の ロールの変更 を参照してください。

NuGet パッケージを消費する

CodeBuild から NuGet パッケージを消費するには、以下をプロジェクトの `buildspec.yaml` ファイルに含めます。

1. `install` セクションで、CodeArtifact 認証情報プロバイダーをインストールして、`msbuild` および`dotnet` のようなコマンドラインツールを設定します。そして、CodeArtifact にパッケージを構築して公開します。
2. `pre-build`セクションで、CodeArtifact リポジトリを NuGet 設定に追加します。

次の`buildspec.yaml` 例を参照してください。詳細については、NuGetでCodeArtifactを使う を参照してください。

認証情報プロバイダーがインストールされ、リポジトリソースが追加されたら、NuGet CLI ツールコマンドを `build` セクションから実行して NuGet パッケージを消費できます。

Linux

dotnet を使用して NuGet パッケージを消費するには:

```
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
```

Server

dotnet を使用して NuGet パッケージを消費するには:

```
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
```

NuGet パッケージで構築する

CodeBuild から NuGet パッケージを使用して構築するには、以下をプロジェクトの `buildspec.yaml` ファイルに含めます。

1. `install`セクションで、CodeArtifact 認証情報プロバイダーをインストールして、`msbuild` および `dotnet` のようなコマンドラインツールを設定します。そして、CodeArtifact にパッケージを構築して公開します。
2. `pre-build`セクションで、CodeArtifact リポジトリを NuGet 設定に追加します。

次の`buildspec.yaml` 例を参照してください。詳細については、NuGetでCodeArtifactを使う を参照してください。

認証情報プロバイダーがインストールされ、リポジトリソースが追加されたら、`build` セクションから `dotnet build` のような NuGet CLI ツールコマンドを実行できます。

Linux

`dotnet` を使用して NuGet パッケージを構築するには:

```
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
```

`msbuild` を使用して NuGet パッケージを構築するには:

```
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
```

Server

dotnet を使用して NuGet パッケージを構築するには:

```
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
```

msbuild を使用して NuGet パッケージを構築するには:

```
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
```

NuGet パッケージを公開する

CodeBuild から NuGet パッケージを公開するには、以下をプロジェクトの `buildspec.yaml` ファイルに含めます。

1. `install` セクションで、CodeArtifact 認証情報プロバイダーをインストールして、`msbuild` および `dotnet` のようなコマンドラインツールを設定します。そして、CodeArtifact にパッケージを構築して公開します。
2. `pre-build`セクションで、CodeArtifact リポジトリを NuGet 設定に追加します。

次の`buildspec.yaml` 例を参照してください。詳細については、NuGetでCodeArtifactを使う を参照してください。

認証情報プロバイダーがインストールされ、リポジトリソースが追加されたら、NuGet CLI ツールコマンドを `build` セクションから実行して、NuGet パッケージを公開できます。

Linux

`dotnet` を使用して NuGet パッケージを公開するには:

```
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
```

Server

dotnet を使用して NuGet パッケージを公開するには:

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - dotnet pack -o .
      - dotnet nuget push *.nupkg -s codeartifact
```

依存関係のキャッシュ

CodeBuild でローカルキャッシュを有効にして、構築ごとに CodeArtifact からフェッチする必要がある依存関係の数を減らすことができます。詳細については、AWS CodeBuild ユーザーガイド の AWS CodeBuildでキャッシュを構築する を参照してください。カスタムローカルキャッシュを有効にしたら、キャッシュディレクトリをプロジェクトの buildspec.yaml ファイルに追加します。

例えば、mvn を使用している場合は、次のようにします。

```
cache:
  paths:
    - '/root/.m2/**/*'
```

その他のツールについては、次の表に示すキャッシュフォルダを使用します。

ツール	キャッシュディレクトリ
mvn	/root/.m2/**/*
gradle	/root/.gradle/caches/**/*
pip	/root/.cache/pip/**/*
npm	/root/.npm/**/*
nuget	/root/.nuget/**/*
yarn (classic)	/root/.cache/yarn/**/*

CodeArtifact をモニタリングする

モニタリングは、CodeArtifact およびその他の AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。 には、CodeArtifact を監視し、問題が発生したときに報告し、必要に応じて自動アクションを実行するための以下のモニタリングツール AWS が用意されています。

- Amazon EventBridge を使用して AWS サービスを自動化し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。 AWS のサービスからのイベントは、ほぼリアルタイムに EventBridge に提供されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。詳細については、Amazon EventBridge ユーザーガイドおよび CodeArtifact イベントの形式と例 を参照してください。
- Amazon CloudWatch メトリクスを使用して、CodeArtifact の使用状況をオペレーション別に表示できます。CloudWatch メトリクスには、CodeArtifact に対して行われたすべてのリクエストが含まれ、リクエストはアカウントごとに表示されます。これらのメトリクスを CloudWatch メトリクスで表示するには、Usage/By AWS Resource AWS 名前空間に移動します。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch メトリクスの使用」を参照してください。

トピック

CodeArtifact イベントをモニタリングする

CodeArtifact は、CodeArtifact リポジトリの変更などのイベントに自動で対応するサービス Amazon EventBridge と統合されています。イベントのルールを作成し、イベントがルールに一致したときの動作を設定できます。EventBridge は、以前は CloudWatch Events と呼ばれていました。

イベントをトリガーとして、以下のアクションを実行できます。

- AWS Lambda 関数の呼び出し。
- AWS Step Functions ステートマシンのアクティブ化。

- Amazon SNS トピックまたは Amazon SQS キューの通知。
- でパイプラインを開始します AWS CodePipeline。

CodeArtifact は、パッケージのバージョンが作成、変更、または削除されたときにイベントを作成します。以下は、CodeArtifact のイベントの例です。

- 新しいパッケージバージョンを公開する (例えば、`npm publish` を実行する)。
- 既存のパッケージバージョンに新しいアセットを追加する (例えば、新しい JAR ファイルを既存の Maven パッケージにプッシュする)。
- `copy-package-versions` を使用して、あるリポジトリから別のリポジトリにパッケージバージョンをコピーする。詳細については、「リポジトリ間でのパッケージのコピー」を参照してください。
- `delete-package-versions` を使用してパッケージバージョンを削除します。詳細については、「パッケージまたはパッケージバージョンを削除する」を参照してください。
- `delete-package` を使用してパッケージバージョンを削除します。削除されたパッケージバージョンごとに 1 つのイベントが公開されます。詳細については、「パッケージまたはパッケージバージョンを削除する」を参照してください。
- アップストリームリポジトリからフェッチされたパッケージバージョンをダウンストリームリポジトリに保持する。詳細については、「CodeArtifact でアップストリームリポジトリを操作する」を参照してください。
- 外部リポジトリからパッケージバージョンを CodeArtifact リポジトリに取り込みます。詳細については、「CodeArtifact リポジトリをパブリックリポジトリに接続する」を参照してください。

イベントは、ドメインを所有するアカウントと、リポジトリを管理するアカウントの両方に配信されます。例えば、そのアカウント `111111111111` がドメイン `my_domain` を所有しているとします。アカウント `222222222222` が `repo2` という `my_domain` でリポジトリを作成します。新しいパッケージのバージョンが `repo2` に公開される際は、両方のアカウントが EventBridge イベントを受信します。ドメイン所有アカウント (111111111111) は、ドメイン内のすべてのリポジトリのイベントを受信します。ひとつのアカウントがドメインとその中のリポジトリの両方を所有している場合、1 つのイベントのみが配信されます。

次のトピックでは、CodeArtifact イベントの形式について説明します。CodeArtifact イベントを設定する方法と、他の AWS サービスでイベントを使用する方法について説明します。詳細については、Amazon EventBridge ユーザーガイドのAmazon EventBridge の開始方法を参照してください 。

CodeArtifact イベントの形式と例

以下は、CodeArtifactイベントの例と、イベントのフィールドとその説明です。

CodeArtifact イベントの形式

すべての CodeArtifact イベントには次のフィールドがあります。

イベントフィールド	説明
version (バージョン)	イベント形式のバージョン。現在は単一のバージョン、「0」しかありません。
ID	イベントに固有の識別子。
detail-type (ディテールタイプ)	イベントのタイプ。これにより、detail オブジェクト内のフィールドが決定されます。現在サポートされているひとつのdetail-type は CodeArtifact Package Version State Change です。
source (ソース)	イベントのソース。CodeArtifact の場合、aws.codeartifact です。
account (アカウント)	イベントを受信する AWS アカウントのアカウント ID。
time (タイム)	イベントがトリガーされた正確な時刻。
region (リージョン)	イベントがトリガーされたリージョン。
resources (リソース)	変更されたパッケージの ARN を含むリスト。リストにはひとつのエントリが含まれています。パッケージ ARN 形式の詳細については、パッケージへの書き込みアクセスを許可するを参照してください。
domainName (ドメインネーム)	パッケージを含むリポジトリを含むドメイン。
domainOwner (ドメインオーナー)	ドメインの所有者の AWS アカウント ID。

イベントフィールド	説明
repositoryName (リポジトリネーム)	パッケージを含むリポジトリ。
repositoryAdministrator	リポジトリの管理者の AWS アカウント ID。
packageFormat (パッケージフォーマット)	イベントをトリガーしたパッケージの形式。
packageNamespace (パッケージネームスペース)	イベントをトリガーしたパッケージの名前スペース。
packageName (パッケージネーム)	イベントをトリガーしたパッケージの名前。
packageVersion (パッケージバージョン)	イベントをトリガーしたパッケージのバージョン。
packageVersionState (パッケージバージョンステート)	イベントがトリガーされたときのパッケージバージョンの状態。使用できる値は `Unfinished` 、`Published` 、`Unlisted`、`Archived` および `Disposed` です。
packageVersionRevision (パッケージバージョンリビジョン)	イベントがトリガーされたときの、パッケージバージョンのアセットとメタデータの状態をユニークに識別する値。パッケージのバージョンが変更された場合 (例えば、別の JAR ファイルを Maven パッケージに追加した場合)、`packageVersionRevision` が変わります。
changes.assetsAdded (チェンジズアセットアディッド)	イベントをトリガーしたパッケージに追加されたアセットの数。アセットの例としては、Maven JAR ファイルやPython ホイールがあります。
changes.assetsRemoved (チェンジズアセットリムーブド)	イベントをトリガーしたパッケージから削除されたアセットの数。

イベントフィールド	説明
Changes.assetsUpdated (チェンジズアセットアップデーテッド)	イベントをトリガーしたパッケージ内で変更されたアセットの数。
changes.metadataUpdated (チェンジズメタデータアップデ–テッド)	変更されたパッケージレベルのメタデータがイベントに含まれている場合、trueに設定されたブール値。例えば、あるイベントが Maven pom.xml ファイルを変更する場合。
changes.statusChanged (チェンジズステータスチェンジド)	イベントの packageVersionStatus が変更されている場合、true に設定されたブール値 (例えば、packageVersionStatus が Unfinished から Published に変更するなど)。
operationType (オペレーションタイプ)	パッケージのバージョン変更の上位タイプについて説明します。指定できる値は、Created、Updated および Deleted です。
sequenceNumber (シーケンスナンバー)	パッケージのイベント番号を指定する整数。パッケージ上の各イベントは、sequenceNumber を増加させるので、イベントを順次配置することができます イベントは、任意の整数によって sequenceNumber を増加させることができます。 **Note** EventBridge イベントは、順不同で受信される場合があります。sequenceNumber は実際の順序を決定するために使用できます。

イベントフィールド	説明
eventDeduplicationId (イベントデュプリケーションID)	重複する EventBridge イベントを区別するために使用される ID。まれに、EventBridge は、単一のイベントまたはスケジュールされた時刻に対して、同じルールを複数回トリガーする場合があります。または、特定のトリガールールに対して、同じターゲットを複数回呼び出す場合があります。

CodeArtifact イベントの例

以下は、npm パッケージの公開時にトリガーされる可能性がある CodeArtifact イベントの例です。

```
{
      "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="
      }
    }
```

イベントを使用して CodePipeline の実行を開始する

この例では、Amazon EventBridge ルールを設定して、CodeArtifact リポジトリ内のパッケージバージョンが公開、変更、または削除されたときに AWS CodePipeline 実行を開始する方法を説明します。

トピック

- EventBridge アクセス許可の設定
- EventBridge ルールを作成するには
- EventBridge ルールターゲットを作成するには

EventBridge アクセス許可の設定

CodePipeline を使用して作成したルールを呼び出すには、EventBridge のアクセス許可を追加する必要があります。 AWS Command Line Interface (AWS CLI) を使用してこれらのアクセス許可を追加するには、「 AWS CodePipeline ユーザーガイド」のCodeCommit ソース (CLI) の CloudWatch Events ルールを作成する」のステップ 1 に従います。

EventBridge ルールを作成するには

ルールを作成するには、`put-rule` コマンドを`--name` および`--event-pattern` パラメータとともに使用します。イベントパターンは、各イベントの内容と一致する値を指定します。パターンがイベントと一致すると、ターゲットがトリガーされます。例えば、次のパターンは、`my_domain` ドメインの`myrepo` リポジトリの CodeArtifact イベントと一致します。

```
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"]}}'
```

EventBridge ルールターゲットを作成するには

次のコマンドは、ルールにターゲットを追加して、イベントがルールに一致したときに CodePipeline の実行がトリガーされるようにするものです。RoleArn パラメータには、このトピックで先に作成したロールの Amazon リソースネーム (ARN) を指定します。

```
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'
```

イベントを使用してLambda 関数を実行するには

この例では、CodeArtifact リポジトリ内のパッケージバージョンが公開、変更または削除された際に、 AWS Lambda 関数を起動する EventBridge ルールを設定する方法について説明します。

詳細については、Amazon EventBridge ユーザーガイド の チュートリアル: EventBridge を使用した AWS Lambda 関数 のスケジュール設定 を参照してください。

トピック

- EventBridge ルールを作成するには
- EventBridge ルールターゲットを作成するには
- EventBridge アクセス許可の設定

EventBridge ルールを作成するには

Lambda 関数を起動するルールを作成するには、put-rule コマンドを --name および --event-pattern オプションとともに使用します。次のパターンは、my_domain ドメインの任意のリポジトリの @types スコープ内にある npm パッケージを指定するものです。

```
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"]}}'
```

EventBridge ルールターゲットを作成するには

次のコマンドは、イベントがルールに一致したときに Lambda 関数を実行するルールにターゲットを追加するものです。arn パラメータについては、Lambda 関数の Amazon リソースネーム (ARN) を指定します。

```
aws events put-targets --rule MyCodeArtifactRepoRule --targets \
  Id=1,Arn=arn:aws:lambda:us-west-2:111122223333:function:MyLambdaFunction
```

EventBridge アクセス許可の設定

add-permission コマンドを使用して、Lambda 関数を呼び出すためのルールに対するアクセス許可を付与します。--source-arn パラメータについては、この例で先に作成したルールの ARN を指定します。

```
aws lambda add-permission --function-name MyLambdaFunction \\
  --statement-id my-statement-id --action 'lambda:InvokeFunction' \\
  --principal events.amazonaws.com \\
  --source-arn arn:aws:events:us-west-2:111122223333:rule/MyCodeArtifactRepoRule
```

CodeArtifactにおけるセキュリティ

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS お客様とお客様の間の責任共有です。責任共有モデルではこれをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 では、安全に使用できるサービスも提供しています。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。CodeArtifactに適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムで対象となる AWS のサービス」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、CodeArtifactを使用する際の責任共有モデルの適用方法についての理解に役立ちます。以下のトピックでは、セキュリティおよびコンプライアンス上の目的を達成するためにCodeArtifactを設定する方法について説明します。また、CodeArtifactリソースのモニタリングや保護に役立つ他のAWS サービスの使用方法についても説明します。

トピック

- AWS CodeArtifact でのデータ保護
- CodeArtifact をモニタリングする
- AWS CodeArtifact のコンプライアンス検証
- AWSCodeArtifact認証とトークン
- AWS CodeArtifact の耐障害性
- インフラストラクチャセキュリティ in AWS CodeArtifact
- 依存関係置換攻撃
- AWS CodeArtifact の Identity and Access Management

AWS CodeArtifact でのデータ保護

責任 AWS 共有モデル、AWS CodeArtifact でのデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「Data Privacy FAQChina」を参照してください。 欧州におけるデータ保護に関する情報については、General Data Protection Regulation (GDPR) Center を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM アイデンティティセンターまたは AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須ですが、TLS 1.3 を推奨します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、または SDK を使用して CodeArtifact AWS CLIまたは他の AWS のサービス を使用する場合も同様です。AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

データ暗号化

暗号化は CodeArtifact のセキュリティの重要な部分です。転送中のデータの暗号化など、一部の暗号化はデフォルトで提供されるため、特に操作は不要です。その他の暗号化 (保管中のデータの暗号化など) については、プロジェクトまたはビルドの作成時に設定できます。

- 保管中のデータの暗号化 - CodeArtifact に保存されているすべてのアセットは、 AWS KMS keys (KMS キー) を使用して暗号化されます。これには、すべてのリポジトリのすべてのパッケージ内のすべてのアセットが含まれます。ドメインごとにひとつのKMSキーを使用して、そのすべてのアセットを暗号化します。デフォルトでは、 AWS マネージド KMS キーが使用されるため、KMS キーを作成する必要はありません。必要に応じて、自分で作成し、設定したカスタマーマネージド KMS キーを使うこともできます。詳細については、AWS Key Management Service ユーザーガイド のKMS キーの作成とAWS 鍵管理サービスの概念を参照してください。ドメインの作成時に、カスタマーマネージドKMS キーを指定できます。詳細については、「CodeArtifact でドメインを操作する」を参照してください。
- 転送時のデータの暗号化 - カスタマーと CodeArtifact との間、および CodeArtifact とそのダウンストリームの依存関係との間のすべての通信は、TLS 暗号化を使用して保護されます。

トラフィックのプライバシー

CodeArtifact でインターフェイス仮想プライベートクラウド (VPC) エンドポイントを使用するように設定することで、CodeArtifact ドメインとその中に含まれるアセットのセキュリティを向上させることができます。これを行う場合、インターネットゲートウェイ、NAT デバイス、または、仮想プライベートゲートウェイは必要ありません。詳細については、「Amazon VPCエンドポイントの使用」を参照してください。 AWS PrivateLink および VPC エンドポイントの詳細については、AWS PrivateLink およびPrivateLink を介した AWS サービスへのアクセスを参照してください。

CodeArtifact をモニタリングする

モニタリングは、 AWS CodeArtifact と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。 は、CodeArtifact リソースをモニタリングし、潜在的なインシデントに対応するために以下 AWS を提供します。

トピック

- を使用した CodeArtifact API コールのログ記録 AWS CloudTrail

を使用した CodeArtifact API コールのログ記録 AWS CloudTrail

CodeArtifact はAWS CloudTrail、CodeArtifact のユーザー、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています。CloudTrailは、パッケージマネージャークライアントからの呼び出しを含む、CodeArtifact のすべての APIコールをイベントとしてキャプチャします。

追跡を作成する場合は、 CodeArtifactへのイベントを含む、Amazon Simple Storage Service (Amazon S3) バケットへの CloudTrailイベントの継続的な配信を有効にすることができます。追跡を設定しない場合でも、CloudTrailコンソールの [Event history (イベント履歴)] でほとんどの最新のイベントを表示できます。CloudTrailで収集された情報を使用して、CodeArtifactに対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

CloudTrailの詳細については、AWS CloudTrail ユーザーガイドを参照してください。

CloudTrailのCodeArtifact情報

CloudTrail は、 AWS アカウントの作成時にアカウントで有効になります。CodeArtifact でアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

CodeArtifact のイベントなど、 AWS アカウントのイベントの継続的な記録については、証跡を作成します。追跡により、CloudTrail はログファイルを Simple Storage Service (Amazon S3) バケットに配信できます。デフォルトでは、コンソールで作成した証跡がすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。その他の AWS サービスを設定して、CloudTrailログに収集されたイベントデータをより詳細に分析し、それに基づく対応を行うこともできます。詳細については、以下の各トピックを参照してください。

- AWS アカウントに対する追跡の作成
- CloudTrail がサポートするサービスと統合
- CloudTrail 用 Amazon SNS 通知の構成

AWS アカウントで CloudTrail ログ記録が有効になっている場合、CodeArtifact アクションに対して行われた API コールは CloudTrail ログファイルで追跡され、他の AWS サービスレコードとともに書き込まれます。CloudTrailは、期間とファイルサイズに基づいて、新しいファイルをいつ作成して書き込むかを決定します。

すべてのCodeArtifactのアクションは、CloudTrailにより口グに記録されます。たとえば、 `ListRepositories` (、`aws codeartifact list-repositories`) AWS CLI、`CreateRepository` (`aws codeartifact create-repository`)、 `ListPackages` (`aws codeartifact list-packages`) の各アクションを呼び出すと、パッケージマネージャーのクライアントコマンドに加えて、CloudTrail ログファイルにエントリが生成されます。パッケージマネージャーのクライアントコマンドは、通常、サーバーに対して複数の HTTP リクエストを行います。各リクエストは、別々の CloudTrail口グイベントを生成します。

CloudTrail口グのクロスアカウント配信

1 回の API コールに対して、最大3つの個別のアカウントが CloudTrail口グを受信します。

- リクエストを行ったアカウント。例えば、`GetAuthorizationToken`を呼び出したアカウント。
- リポジトリ管理者アカウント。例えば、`ListPackages`が呼び出されたリポジトリを管理するアカウント。
- ドメイン所有者のアカウント。例えば、APIが呼び出されたリポジトリがあり、そのリポジトリを含むドメインを所有するアカウント。

`ListRepositoriesInDomain`のように、特定のリポジトリではなくドメインに対するアクションである APIについては、呼び出したアカウントとドメイン所有者のアカウントのみが CloudTrailの口グを受け取ります。どのリソースに対しても認証されていない`ListRepositories`のような API の場合、発信者元のアカウントのみが CloudTrail口グを受け取ります。

CodeArtifact口グファイルエントリについて

CloudTrail ログファイルには、1 つ以上の口グエントリを含むことができます。各エントリには、複数の JSON 形式のイベントがリストされます。口グイベントは任意のソースからの1つのリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメーターなどに関する情報が含まれます。口グエントリは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

トピック

- 例: GetAuthorizationToken API を呼び出すための口グエントリ
- 例: npm パッケージバージョンを取得するための口グエントリ

例: GetAuthorizationToken API を呼び出すためのログエントリ

GetAuthorizationTokenによって作成されるログエントリには、requestParametersフィールドのドメイン名が含まれます。


```
{
  "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"
}
```

例: npm パッケージバージョンを取得するためのログエントリ

npm クライアントを含むすべてのパッケージマネージャークライアントからのリクエストには、ドメイン名、リポジトリ名、パッケージ名などの追加データがrequestParametersフィールドに記録されます。URL パスとHTTPメソッドは、additionalEventData フィールドに記録されます。

```
{
   "eventVersion": "1.05",
   "userIdentity": {
       "type": "AssumedRole",
       "principalId": "AIDACKCEVSQ6C2EXAMPLE",
       "arn": "arn:aws:sts::123456789012:assumed-role/Console/example",
       "accountId": "123456789012",
       "accessKeyId": "ASIAIJIOBJIBSREXAMPLE",
       "sessionContext": {
           "attributes": {
               "mfaAuthenticated": "false",
               "creationDate": "2018-12-17T02:05:16Z"
           },
           "sessionIssuer": {
               "type": "Role",
               "principalId": "AIDACKCEVSQ6C2EXAMPLE",
               "arn": "arn:aws:iam::123456789012:role/Console",
               "accountId": "123456789012",
               "userName": "Console"
           }
       }
   },
   "eventTime": "2018-12-17T02:05:46Z",
   "eventSource": "codeartifact.amazonaws.com",
   "eventName": "ReadFromRepository",
   "awsRegion": "us-west-2",
   "sourceIPAddress": "205.251.233.50",
   "userAgent": "npm/6.14.15 node/v12.22.9 linux x64 ci/custom",
   "requestParameters": {
       "domainName": "example-domain",
       "domainOwner": "123456789012",
       "repositoryName": "example-repo",
       "packageName": "lodash",
       "packageFormat": "npm",
```

```
        "packageVersion": "4.17.20"
    },
    "responseElements": null,
    "additionalEventData": {
        "httpMethod": "GET",
        "requestUri": "/npm/lodash/-/lodash-4.17.20.tgz"
    },
    "requestID": "9f74b4f5-3607-4bb4-9229-ffffffffffff",
    "eventID": "c74e40dd-8847-4058-a14d-ffffffffffff",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

AWS CodeArtifact のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「コンプライアンスAWS のサービス プログラムによるスコープ」の「コンプライアンス」の「」を参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。を使用する際のコンプライアンス責任の詳細については AWS のサービス、AWS 「 セキュリティドキュメント」を参照してください。

AWSCodeArtifact認証とトークン

CodeArtifactでは、パッケージバージョンを公開したり消費したりするために、ユーザーがサービスで認証を受ける必要があります。CodeArtifact サービスを利用するには、自分の AWS 認証情報を使って認可トークンを作成し、認証を受ける必要があります。認可トークンを作成するには、適切なアクセス権限を持っている必要があります。認可トークンの作成に必要な権限については、「AWS CodeArtifact アクセス許可リファレンス」の GetAuthorizationToken エントリを参照してください。CodeArtifact の権限の一般的な情報については、「How AWS CodeArtifact と IAM の連携」を参照してください。

CodeArtifact から認可トークンを取得するには、GetAuthorizationToken API を呼び出す必要があります。AWS CLIを使用することで、loginまたはget-authorization-tokenコマンドでGetAuthorizationTokenを実行することができます。

> **Note**
>
> ルートユーザーは GetAuthorizationToken を呼び出すことができません。

- aws codeartifact login: このコマンドを使用すると、一般的なパッケージマネージャーが CodeArtifact をワンステップで使えるように簡単に設定できます。loginを呼び出すと、GetAuthorizationTokenでトークンを取得し、パッケージマネージャにトークンと正しいCodeArtifactリポジトリのエンドポイントを設定します。次のパッケージマネージャーがサポートされています。
  - dotnet
  - npm
  - nuget
  - pip
  - swift
  - twine
- aws codeartifact get-authorization-token:loginでサポートされていないパッケージマネージャの場合、直接get-authorization-tokenを呼び出すことができます。その後、必要に応じて、設定します例えばファイルにトークンを追加したり、環境変数に格納するなどして、パッケージマネージャにトークンを設定します。

CodeArtifact 認可トークンは、デフォルトで 12 時間有効です。トークンの有効期限は15分から12時間の間で設定できます。有効期限切れになったら、別のトークンを取得する必要があります。トークンの有効期限は、loginまたはget-authorization-tokenが実行されてからカウントダウンされます。

ロールを引き受ける際にloginまたはget-authorization-tokenが呼び出された場合、--duration-secondsの値を0に設定することで、トークンの有効期限がロールのセッション時間の残り時間と同じになるように設定できます。それ以外の場合、トークンの有効期限は、ロールの最大セッション時間とは無関係です。例えば、sts assume-role を呼び出して 15 分間のセッション時間を指定し、その後 login を呼び出して CodeArtifact 認可トークンを取得するとします。この場

合、トークンは15分のセッションよりも長いですが、12時間全体にわたって有効です。セッションの継続時間の制御については、IAM ユーザーガイド の中の IAM ロールの使用 を参照してください。

loginコマンドで作成されたトークン

aws codeartifact loginコマンドは、GetAuthorizationTokenでトークンを取得し、トークンと正しい CodeArtifactリポジトリエンドポイントを使用してパッケージマネージャを設定します。

次の表では、loginコマンドのパラメータについて説明します。

[Parameter] (パラメータ)	必須	説明
--tool	あり	認証するパッケージマネージャ。想定される値は、dotnet、npm、nuget、pip、swift、です。
--domain	あり	リポジトリが属するドメイン名。
--domain-owner	なし	ドメインの所有者のID。このパラメータは、自分が認証されていないAWSアカウントが所有しているドメインにアクセスする場合に必要です。詳細については、「クロスアカウントドメイン」を参照してください。
--repository	あり	認証先のリポジトリの名前。
--duration-seconds	なし	ログイン情報が有効な時間 (秒単位)。最小値は 900*で、最大値は 43200 です。
--namespace	なし	ネームスペースをリポジトリツールに関連付けます。
--dry-run	なし	設定を変更せずに、ツールをリポジトリに接続するために実行されるコマンドのみを出力します。

[Parameter] (パラメータ)	必須	説明
* ロールを担いながら login を呼び出す時にも0の値は有効です。--duration-seconds 0でloginを実行すると、引き受けたロールのセッション時間の残り時間と等しい有効期限を持つトークンを作成します。		

次の例は、login コマンドを使用して認可トークンを取得する方法を示しています。

```
aws codeartifact login \
    --tool dotnet | npm | nuget | pip | swift | twine \
    --domain my_domain \
    --domain-owner 111122223333 \
    --repository my_repo
```

npm でloginコマンドを実行する具体的方法についてはCodeArtifact で npm を設定して使用するを参照してください。Pythonについては、PythonでCodeArtifactを使う を参照してください。

GetAuthorizationToken API を呼び出すために必要な権限

CodeArtifact GetAuthorizationToken API を呼び出すには、sts:GetServiceBearerToken 権限と codeartifact:GetAuthorizationToken 権限の両方が必要です。

CodeArtifact リポジトリでパッケージマネージャーを使用するには、IAM ユーザーまたはロールが sts:GetServiceBearerToken を許可する必要があります。sts:GetServiceBearerToken は、CodeArtifact ドメインリソースポリシーに追加できますが、そのポリシーに影響はしません。

GetAuthorizationTokenAPIで作成されたトークン

CodeArtifact から認可トークンを取得するため、get-authorization-token を呼び出します。

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

--duration-seconds引数を使用して、トークンの有効期限を変更できます。最小値は900で、最大値は43200です。次の例では、1時間(3600 秒)続くトークンを作成します。

```
aws codeartifact get-authorization-token \
    --domain my_domain \
    --domain-owner 111122223333 \
    --query authorizationToken \
    --output text \
    --duration-seconds 3600
```

ロールを引き受けながらget-authorization-tokenを実行する場合、トークンの有効期限は、ロールの最大セッション時間とは無関係です。引き受けたロールのセッション時間が満了したときにトークンが期限切れになるように構成するには、--duration-secondsを0に設定します。

```
aws codeartifact get-authorization-token \
    --domain my_domain \
    --domain-owner 111122223333 \
    --query authorizationToken \
    --output text \
    --duration-seconds 0
```

詳細については、次のドキュメントを参照してください。

- トークンと環境変数のガイダンスについては、環境変数を使用して認証トークンを渡すを参照してください。
- Python ユーザーについては、ログインコマンドを使用せずにpipを設定するまたはCodeArtifact で twine を設定して使用するを参照してください。
- Maven ユーザーについては、Gradle で CodeArtifact を使用するまたはmvn で CodeArtifact を使用するを参照してください。
- npmユーザーについては、login コマンドを使用せずに npm を設定するを参照してください。

環境変数を使用して認証トークンを渡す

AWS CodeArtifact は、Maven や Gradle などのビルドツールからのリクエストを認証し認可するために、GetAuthorizationToken API によって提供された認可トークンを使用します。それらの認証トークンについての詳細は、GetAuthorizationTokenAPIで作成されたトークンを参照してください。

これらの認証トークンは、環境変数に格納してビルドツールで読み取りCodeArtifactリポジトリからパッケージを取得したりパッケージをそのリポジトリに公開したりするために必要なトークンを取得できます。

他ユーザーやプロセスによって読み取られたり、誤ってソース管理にチェックされる危険のあるファイルにトークンを格納するよりも、この方法がセキュリティ上の理由で適しています。

1. をインストールまたはアップグレードしてから設定する AWS CLIの説明に従ってAWS資格情報を設定します。
2. CODEARTIFACT_AUTH_TOKEN 環境変数を設定します:

> **Note**
>
> 一部のシナリオでは、--domain-owner引数を含める必要はありません。詳細については、「クロスアカウントドメイン」を参照してください。

- macOS、Linux :

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

- Windows (デフォルトのコマンドシェルを使用):

```
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
```

CodeArtifact 認可トークンの取り消し

認証されたユーザーが CodeArtifactリソースにアクセスするためのトークンを作成すると、そのトークンは変更可能なアクセス期間が終了するまで有効です。デフォルトのアクセス期間は12時間です。場合によっては、アクセス期間が切れる前にトークンへのアクセスを取り消すことができます。次の手順に従って、CodeArtifactリソースへのアクセスを取り消すことができます。

引き受けたロール またはフェデレーティッドユーザーアクセス のように一時的なセキュリティ資格情報を使用してアクセストークンを作成した場合は、IAM ポリシーを更新してアクセスを拒否することで、アクセスを取り消すことができます。詳細については、IAM ユーザーガイド の一時的なセキュリティ資格情報の許可を無効にするを参照してください。

長期的な IAM ユーザー資格情報を使用してアクセストークンを作成した場合は、アクセスを拒否するようにユーザーズポリシーを変更するか、IAM ユーザーを削除する必要があります。詳細については、IAM ユーザーのアクセス権限の変更またはIAM ユーザーの削除 を参照してください。

AWS CodeArtifact の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、高度に冗長なネットワークで接続された、物理的に分離された複数のアベイラビリティーゾーンを提供します。AWS CodeArtifactは複数のアベイラビリティーゾーン、アーティファクトデータとAmazon S3 および Amazon DynamoDBに格納している メタデータで運用されている。暗号化されたデータは、複数の施設と各施設内の複数のデバイスにまたがって冗長的に保存され、高い可用性と高い耐久性を実現します。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

インフラストラクチャセキュリティ in AWS CodeArtifact

マネージドサービスである AWS CodeArtifact は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して環境を AWS 設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由で CodeArtifact にアクセスします。クライアントは次をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

依存関係置換攻撃

パッケージマネージャーは、再利用可能なコードをパッケージ化して共有するプロセスを簡素化します。これらのパッケージは、ある組織がアプリケーションで使用するために開発したプライベートパッケージの場合もあれば、組織外で開発され、パブリックパッケージリポジトリによって配布されるパブリックパッケージ (通常はオープンソースパッケージ) の場合もあります。パッケージをリクエストする際、開発者はパッケージマネージャーを使用して依存関係の新しいバージョンを取得します。依存関係置換攻撃 (依存関係かく乱攻撃とも呼ばれる) は、通常、パッケージマネージャーでパッケージの正規バージョンと悪意のあるバージョンを区別できない点を悪用するものです。

依存関係置換攻撃は、ソフトウェアサプライチェーン攻撃と呼ばれるハッキングのサブセットに属します。ソフトウェアサプライチェーン攻撃は、ソフトウェアサプライチェーンのあらゆる場所にある脆弱性を利用する攻撃です。

依存関係置換攻撃は、内部で開発されたパッケージとパブリックリポジトリから取得したパッケージの両方を使用するすべてのユーザーを標的にする可能性があります。攻撃者は内部パッケージ名を特定し、同じ名前の悪意のあるコードを公開パッケージリポジトリに戦略的に配置します。通常、悪意のあるコードはバージョン番号の高いパッケージで公開されます。パッケージマネージャーは、悪意のあるパッケージをパッケージの最新バージョンとみなすため、これらの公開フィードから悪意のあるコードを取得します。これにより、目的のパッケージと悪意のあるパッケージが「かく乱」または「置換」され、コードが危険にさらされることになります。

依存関係置換攻撃を防ぐために、AWS CodeArtifact にはパッケージオリジンコントロールが用意されています。パッケージオリジンコントロールは、パッケージをリポジトリに追加する方法を制御する設定です。このコントロールを使用することで、パッケージのバージョンがリポジトリへ直接公開されることと、公開ソースから取り込まれることの両方が同時に発生しないようにでき、依存関係置換攻撃から保護されます。パッケージグループにオリジンコントロールを設定することで、個別パッケージと複数パッケージにオリジンコントロールを設定できます。パッケージオリジンコントロールとその変更方法については、「パッケージオリジンコントロールの編集」と「パッケージグループのオリジンコントロール」を参照してください。

AWS CodeArtifact の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に CodeArtifact リソースの使用を承認する (アクセス許可を付与する) かを管理します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「トラブルシューティング AWS CodeArtifact アイデンティティとアクセス」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「How AWS CodeArtifact と IAM の連携」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「AWS CodeArtifact のアイデンティティベースのポリシーの例」を参照)

アイデンティティを使用した認証

認証は、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント ルートユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーティッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID Directory Service ソースの認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、 AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「IAM アイデンティティセンターとは」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスする必要がある AWS」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すことで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーショ

ンに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられている場合のアクセス許可を定義します。 は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の上限を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。
- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

How AWS CodeArtifact と IAM の連携

IAM を使用して CodeArtifact へのアクセスを管理する前に、CodeArtifact で利用できる IAM の機能について学びます。

AWS CodeArtifact で使用できる IAM 機能

IAM 機能	CodeArtifact のサポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	はい

IAM 機能	CodeArtifact のサポート
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー (サポート固有)	いいえ
ACL	なし
ABAC (ポリシー内のタグ)	部分的
一時認証情報	あり
プリンシパルアクセス権限	あり
サービスロール	いいえ
サービスリンクロール	いいえ

CodeArtifact およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要については、IAM ユーザーガイドのAWS 「IAM と連携する のサービス」を参照してください。

CodeArtifact でのアイデンティティベースのポリシー

アイデンティティベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

CodeArtifact のアイデンティティベースのポリシー例

CodeArtifact アイデンティティベースのポリシーの例については、「AWS CodeArtifact のアイデンティティベースのポリシーの例」を参照してください。

CodeArtifact 内のリソースベースのポリシー

リソースベースのポリシーのサポート: あり

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーで、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。詳細については、IAM ユーザーガイドのIAM でのクロスアカウントリソースアクセスを参照してください。

CodeArtifact のポリシーアクション

ポリシーアクションのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

CodeArtifact アクションのリストを確認するには、「サービス認可リファレンス」のAWS CodeArtifact で定義されるアクション」を参照してください。

CodeArtifact のポリシーアクションは、アクションの前に次のプレフィックスを追加します。

```
codeartifact
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "codeartifact:action1",
      "codeartifact:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "codeartifact:Describe*"
```

CodeArtifact アイデンティティベースのポリシーの例については、「AWS CodeArtifact のアイデンティティベースのポリシーの例」を参照してください。

CodeArtifact のポリシーリソース

ポリシーリソースのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

CodeArtifact リソースタイプとその ARNsAWS CodeArtifact で定義されるリソース」を参照してください。 各リソースの ARN を指定できるアクションについては、AWS CodeArtifact で定義されるアクション」を参照してください。ポリシーで CodeArtifact リソース ARN を指定する例については、「AWS CodeArtifact リソースとオペレーション」を参照してください。

CodeArtifact のポリシー条件キー

サービス固有のポリシー条件キーへのサポート: なし

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「グローバル条件コンテキストキー」を参照してください。

> **Note**
>
> AWS CodeArtifact は、次の AWS グローバル条件コンテキストキーをサポートしていません。
>
> - リファラー
> - UserAgent

CodeArtifact 条件キーのリストを確認するには、「サービス認可リファレンス」のAWS CodeArtifact の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、AWS CodeArtifact で定義されるアクション」を参照してください。

CodeArtifact アイデンティティベースのポリシーの例については、「AWS CodeArtifact のアイデンティティベースのポリシーの例」を参照してください。

CodeArtifact 内の ACL

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

CodeArtifact での ABAC

ABAC (ポリシー内のタグ) のサポート: 一部

属性ベースのアクセスコントロール (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグ

がリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「ABAC 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセスコントロール (ABAC) を使用する」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースのポリシーの例など、CodeArtifact リソースのタグ付けの詳細については、「タグを使用した CodeArtifact リソースへのアクセスのコントロール」を参照してください。

CodeArtifact での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一時的な認証情報は、 AWS リソースへの短期的なアクセスを提供し、フェデレーションまたは切り替えロールを使用する場合に自動的に作成されます。 AWS では、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することをお勧めします。詳細については、「IAM ユーザーガイド」の「IAM の一時的な認証情報」および「AWS のサービス と IAM との連携」を参照してください。

CodeArtifact のクロスサービスプリンシパル許可

転送アクセスセッション (FAS) のサポート: あり

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

呼び出し側のプリンシパルに他のサービスの権限が必要な CodeArtifact API アクションは 2 つあります。

1. `GetAuthorizationToken` では `sts:GetServiceBearerToken` と `codeartifact:GetAuthorizationToken` が必要です。
2. `CreateDomain` がデフォルト以外の暗号化キーを指定する場合は、`codeartifact:CreateDomain` に加えて KMS キーに `kms:DescribeKey` と `kms:CreateGrant` の両方が必要です。

CodeArtifact のアクションで必要な権限とリソースの詳細については、「AWS CodeArtifact アクセス許可リファレンス」を参照してください。

CodeArtifact のサービスロール

サービスロールのサポート: なし

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、IAM ユーザーガイドの AWS のサービスに許可を委任するロールを作成するを参照してください。

> **Warning**
>
> サービスロールの権限を変更すると、CodeArtifact の機能が使用できなくなる場合があります。CodeArtifact で指示されないかぎり、サービスロールを編集しないでください。

CodeArtifact のサービスにリンクされたロール

サービスにリンクされたロールのサポート: なし

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「IAM と提携するAWS のサービス」を参照してください。表の「サービスリンクロール」列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

AWS CodeArtifact のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーおよびロールには、CodeArtifact リソースを作成または変更する権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの JSON ポリシードキュメントを使用して IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーを作成する (コンソール)」を参照してください。

各リソースタイプの ARN の形式など、CodeArtifact で定義されるアクションとリソースタイプの詳細については、「サービス認可リファレンス」のAWS CodeArtifact のアクション、リソース、および条件キー」を参照してください。 ARNs

トピック

- ポリシーに関するベストプラクティス
- CodeArtifact コンソールの使用
- AWS CodeArtifact の AWS 管理 (事前定義) ポリシー
- ユーザーが自分の許可を表示できるようにする
- リポジトリやドメインに関する情報の取得をユーザーに許可する
- 特定のドメインに関する情報の取得をユーザーに許可する
- 特定のリポジトリに関する情報の取得をユーザーに許可する
- 認可トークンの期間の制限

ポリシーに関するベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内の CodeArtifact リソースを誰かが作成、アクセス、または削除できるどうかを決定します。これらのアクションでは、 AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイド の AWS マネージドポリシー または ジョブ機能のAWS マネージドポリシー を参照してください。

- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

CodeArtifact コンソールの使用

AWS CodeArtifact コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の CodeArtifact リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが CodeArtifact コンソールを引き続き使用できるようにするには、エンティティに AWSCodeArtifactAdminAccessまたは AWSCodeArtifactReadOnlyAccess AWS 管理ポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

AWS CodeArtifact の AWS 管理 (事前定義) ポリシー

AWS は、 によって作成および管理されるスタンドアロン IAM ポリシーを提供することで、多くの一般的なユースケースに対処します AWS。これらの AWS 管理ポリシーは、一般的なユースケースに必要なアクセス許可を付与するため、必要なアクセス許可を調査する必要がなくなります。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」 を参照してください。

アカウントのユーザーにアタッチできる次の AWS 管理ポリシーは、 AWS CodeArtifact に固有のものです。

- AWSCodeArtifactAdminAccess – CodeArtifact ドメインを管理するためのアクセス許可を含む CodeArtifact へのフルアクセスを提供します。

  JSON

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

- `AWSCodeArtifactReadOnlyAccess` – CodeArtifact への読み取り専用アクセスを提供します。

  JSON

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

CodeArtifact サービスロールを作成および管理するには、 という名前 AWS の管理ポリシーもアタッチする必要がありますIAMFullAccess。

独自のカスタム IAM ポリシーを作成して、CodeArtifact アクションとリソースのためのアクセス許可を与えることもできます。これらのカスタムポリシーは、それらのアクセス許可が必要な IAM ユーザーまたはグループにアタッチできます。

ユーザーが自分の許可を表示できるようにする

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、

または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

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

リポジトリやドメインに関する情報の取得をユーザーに許可する

以下のポリシーは、IAM ユーザーまたはロールが、ドメイン、リポジトリ、パッケージ、アセットなど、あらゆるタイプの CodeArtifact リソースをリストおよび記述できるようにするものです。ポリシーには、プリンシパルが CodeArtifact リポジトリからパッケージを取得できるようにす

るcodeArtifact:ReadFromRepositoryアクセス許可も含まれます。新しいドメインやリポジトリの作成は許可せず、新しいパッケージの公開も許可しません。

GetAuthorizationToken API を呼び出すには、codeartifact:GetAuthorizationToken と sts:GetServiceBearerToken へのアクセス許可が必要です。

JSON

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

特定のドメインに関する情報の取得をユーザーに許可する

us-east-2リージョンにあり、123456789012 のアカウントで、名前が my で始まるドメインについての情報のみをユーザーがリストアップすることを許可するアクセス許可ポリシーの例を次に示します。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "codeartifact:ListDomains",
         "Resource": "arn:aws:codeartifact:us-east-2:111122223333:domain/my*"

      }
   ]
}
```

特定のリポジトリに関する情報の取得をユーザーに許可する

testで終わるリポジトリおよびそれに含まれるパッケージに関する情報をユーザーが取得できるようにするアクセス許可ポリシーの例を以下に示します。ユーザーは、リソースを発行したり、作成したり、削除することができなくなります。

GetAuthorizationToken API を呼び出すには、codeartifact:GetAuthorizationToken と sts:GetServiceBearerToken へのアクセス許可が必要です。

JSON

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

認可トークンの期間の制限

ユーザーが、パッケージバージョンを公開または使用するには、認可トークンを使用して CodeArtifact を認証する必要があります。認可トークンは、設定された有効期間中にのみ有効です。トークンの有効期間はデフォルトで 12 時間となっています。認可レスポンスの詳細については、「AWSCodeArtifact認証とトークン」を参照してください。

トークンを取得するときに、ユーザーはトークンの有効期間を設定できます。認可トークンの有効期間の有効値は、0、および 900 (15 分) から 43200 (12 時間) までの間の任意の数値です。0の値は、ユーザーのロールの一時的な認証情報に等しい期間を持つトークンを作成します。

管理者は、ユーザーまたはグループに付与された権限ポリシーの sts:DurationSeconds 条件キーを使用して、認可トークンの有効期間中の有効値を制限できます。有効値以外の有効期間を持つ認可トークンをユーザーが作成しようとすると、トークンの作成は失敗します。

次のポリシー例では、CodeArtifact ユーザーによって作成された認可トークンの有効期間を制限しています。

ポリシー例:トークンの有効期間を 12 時間 (43200 秒) ちょうどに制限する

このポリシーを使用すると、ユーザーは有効期間が 12 時間の認可トークンしか作成することができません。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codeartifact:*",
            "Resource": "*"
        },
        {
            "Sid": "sts",
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "NumericEquals": {
                    "sts:DurationSeconds": 43200
                },
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

ポリシー例:トークンの有効期間を 15 分から 1 時間、またはユーザーの一時的な認証情報期間と等しい時間に制限する

このポリシーにより、ユーザーは 15 分から 1 時間の間で有効なトークンを作成することができます。また、ユーザーは、`--durationSeconds`に対して`0`を指定することで、ロールの一時的な認証情報の有効期間を保持するトークンを作成することもできます。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codeartifact:*",
            "Resource": "*"
        },
        {
            "Sid": "sts",
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "NumericLessThanEquals": {
                    "sts:DurationSeconds": 3600
                },
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

タグを使用した CodeArtifact リソースへのアクセスのコントロール

IAM ユーザーポリシーステートメントの条件は、CodeArtifact アクションに必要なリソースへのアクセス許可を指定するために使用する構文の一部です。条件内でタグを使用することは、リソースとリクエストへのアクセスをコントロールするひとつの方法です。CodeArtifact リソースのタグ付けの詳細については、リソースのタグ付け を参照してください。このトピックでは、タグベースのアクセスコントロールについて説明します。

IAM ポリシーの設計時に特定のリソースへのアクセス権を付与することで、詳細なアクセス許可を設定できます。管理するリソースの数が増えるに従って、このタスクはより困難になります。リソー

スにタグ付けしてポリシーステートメント条件でタグを使用することにより、このタスクをより容易にすることができます。特定のタグを使用して任意のリソースへのアクセス権を一括して付与します。次に、作成時や以降の段階で、このタグを関連リソースに繰り返し適用します。

タグは、リソースにアタッチしたり、タグ付けをサポートするサービスへのリクエストに渡したりすることができます。CodeArtifact では、リソースにタグを付けることができ、一部のアクションにタグを含めることができます。IAM ポリシーを作成するときに、タグ条件キーを使用して以下をコントロールできます。

- どのユーザーがドメインやリポジトリのリソースに対してアクションを実行できるか、既に付けられているタグに基づいて表示します。
- どのタグをアクションのリクエストで渡すことができるか。
- リクエストで特定のタグキーを使用できるかどうか。

タグ条件キーの完全な構文と意味については、[IAM ユーザーガイド]の [タグを使用したアクセスコントロール]」を参照してください。

> **Important**
>
> リソースのタグを使用してアクションを制限する場合、タグはアクションが実行されるリソース上にある必要があります。例えば、タグを使用して `DescribeRepository` 権限を拒否するには、ドメインではなく各リポジトリにタグ付けする必要があります。CodeArtifact のアクションとそれらが実行されるリソースのリストについては、「AWS CodeArtifact アクセス許可リファレンス」を参照してください。

タグベースのアクセスコントロールの例

次の例は、CodeArtifact ユーザー用のポリシーでタグ条件を指定する方法を示しています。

Example 1: リクエストのタグに基づいてアクションを制限する

`AWSCodeArtifactAdminAccess`マネージドユーザーポリシーは、すべてのリソースに対して任意の CodeArtifact アクションを実行する無制限のアクセス許可をユーザーに付与します。

次のポリシーでは、リクエストに特定のタグが含まれていない限り、この権限を制限し、未認証のユーザーに対してリポジトリを作成するアクセス許可を拒否します。そのために、1 または 2 の値のいずれかを持つ `costcenter` という名前のタグをこのリクエストが指定していない場

合、CreateRepository アクションを拒否します。お客様の管理者は、未認証の IAM ユーザーには、マネージドユーザーポリシーに加えて、この IAM ポリシーをアタッチする必要があります。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "codeartifact:CreateRepository",
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:RequestTag/costcenter": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "codeartifact:CreateRepository",
      "Resource": "*",
      "Condition": {
          "ForAnyValue:StringNotEquals": {
              "aws:RequestTag/costcenter": [
                  "1",
                  "2"
              ]
          }
      }
  }
  ]
}
```

Example 2: リソースタグに基づいてアクションを制限する

AWSCodeArtifactAdminAccessマネージドユーザーポリシーは、すべてのリソースに対して任意の CodeArtifact アクションを実行する無制限のアクセス許可をユーザーに付与します。

次のポリシーでは、この権限を制限し、未認証のユーザーが指定したリポジトリに対してアクションを実行するアクセス許可を拒否します。そのために、Value1 または Value2 の値のいずれか

を持つ Key1 という名前のタグをこのリソースが持つ場合、一部のアクションを拒否します。(このaws:ResourceTag 条件キーを使用して、それらのリソースのタグに基づいて、このリソースへのアクセスをコントロールします)。お客様の管理者は、未認証の IAM ユーザーには、マネージドユーザーポリシーに加えて、この IAM ポリシーをアタッチする必要があります。

JSON

```
{
  "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: リソースタグに基づいてアクションを許可する

次のポリシーは、CodeArtifact のリポジトリやパッケージについてアクションを実行し、情報を取得するアクセス許可をユーザーに付与します。

そのために、Value1 という値を持つ Key1 という名前のタグをこのリポジトリが持つ場合に、特定のアクションを許可します。(この aws:RequestTag 条件キーを使用して、IAM リクエストで渡すことができるタグをコントロールします)。aws:TagKeys 条件は、タグキーの大文字と小文字を区別します。このポリシーは、AWSCodeArtifactAdminAccess マネージドユーザーポリシーが添付されていない IAM ユーザーに便利です。このマネージドポリシーは、すべてのリソースに対して任意の CodeArtifact アクションを実行する無制限の許可をユーザーに付与します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeartifact:UpdateRepository",
        "codeartifact:DeleteRepository",
        "codeartifact:ListPackages"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Key1": "Value1"
        }
      }
    }
  ]
}
```

Example 4: リクエストのタグに基づいてアクションを許可する

次のポリシーでは、CodeArtifact の指定されたドメインにリポジトリを作成するアクセス許可をユーザーに付与します。

そのために、Value1 という値を持つ Key1 という名前のタグをこのリクエストのリソース作成 API が指定する場合に、CreateRepository アクションと TagResource アクションを許可します。(この aws:RequestTag 条件キーを使用して、IAM リクエストで渡すことができるタグをコントロールします)。aws:TagKeys 条件は、タグキーの大文字と小文字を区別します。このポリシーは、AWSCodeArtifactAdminAccess マネージドユーザーポリシーが添付されていない IAM ユー

ザーに便利です。このマネージドポリシーは、すべてのリソースに対して任意の CodeArtifact アクションを実行する無制限の許可をユーザーに付与します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeartifact:CreateRepository",
        "codeartifact:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Key1": "Value1"
        }
      }
    }
  ]
}
```

AWS CodeArtifact アクセス許可リファレンス

AWS CodeArtifact リソースとオペレーション

In AWS CodeArtifact、プライマリリソースはドメインです。ポリシーで Amazon リソースネーム (ARN) を使用して、ポリシーを適用するリソースを識別します。リポジトリはリソースでもあり、ARN が関連付けられています。詳細については、「Amazon Web Services 全般のリファレンス」の「Amazon Resource Names (ARNs)」を参照してください。

リソースタイプ	ARN 形式
ドメイン	arn:aws:codeartifact: *region-ID* :*account-ID* :domain/*my_domain*

リソースタイプ	ARN 形式
Repository	arn:aws:codeartifact: *region-ID* :*account-ID* :repository/ *my_domain* /*my_repo*
パッケージグループ	arn:aws:codeartifact: *region-ID* :*account-ID* :package-group/ *my_domain* /*encoded_package_group_pattern*
ネームスペースを持つパッケージ	arn:aws:codeartifact: *region-ID* :*account-ID* :package/ *my_domain* /*my_repo*/*package-format* /*namespace* /*my_package*
ネームスペースを持たないパッケージ	arn:aws:codeartifact: *region-ID* :*account-ID* :package/ *my_domain* /*my_repo*/*package-format* //*my_package*
すべての CodeArtifact リソース	arn:aws:codeartifact:*
特定の AWS リージョンの特定アカウントに所有されるすべての CodeArtifact リソース	arn:aws:codeartifact: *region-ID* :*account-ID* :*

どのリソース ARN を指定するかは、アクセスをコントロールする対象のアクションによって異なります。

以下のように ARN を使用して、ステートメント内で特定のドメイン (*myDomain*) を指定できます。

```
"Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain"
```

以下のように ARN を使用して、ステートメント内で特定のリポジトリ (*myRepo*) を指定できます。

```
"Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain/myRepo"
```

単一のステートメントに複数のリソースを指定するには、コンマで ARN を区切ります。次のステートメントは、特定のドメインのすべてのパッケージとリポジトリに適用されます。

```
"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**
>
> 多くの AWS サービスは、ARN でARNs。ただし、CodeArtifact では、完全一致したリソースパターンとルールが使用されます。イベントパターンの作成時に正しい文字を使用して、リソース内の ARN 構文とそれらの文字が一致する必要があります。

AWS CodeArtifact API オペレーションとアクセス許可

アクセスコントロールをセットアップし、IAM アイデンティティ (アイデンティティベースのポリシー) にアタッチできるアクセス権限ポリシーを作成する際に、以下の表をリファレンスとして使用できます。

AWS CodeArtifact ポリシーで AWS全体の条件キーを使用して、条件を表現できます。表の詳細については、[IAM ユーザーガイド]の「IAM JSON ポリシー要素のリファレンス」を参照してください。

アクションは、ポリシーの `Action` フィールドで指定します。アクションを指定するには、API オペレーション名 (例えば、`codeartifact:` や `codeartifact:CreateDomain`) の前に `codeartifact:AssociateExternalConnection` プレフィックスを使用します。単一のステートメントに複数のアクションを指定するには、コンマで区切ります (例えば、`"Action": [ "codeartifact:CreateDomain", "codeartifact:AssociateExternalConnection" ]`)。

ワイルドカード文字の使用

ポリシーの `Resource` フィールドでリソース値として、ワイルドカード文字 (*) を使用して、または使用せずに ARN を指定します。ワイルドカードを使用して複数のアクションまたはリソースを指定することができます。例えば、`codeartifact:*` は、すべての CodeArtifact アクションを指定し、`codeartifact:Describe*` は、`Describe` という単語で始まるすべての CodeArtifact アクションを指定します。

パッケージグループ ARN

> **Note**
>
> このセクションでは、パッケージグループの ARN とパターンエンコーディングがどのように情報を提供するかについて説明します。パターンをエンコードして ARN を構築するのではなく、コンソールから ARN をコピーするか、`DescribePackageGroup` API を使用して ARN を取得することをお勧めします。

IAM ポリシーは、ワイルドカード文字 (*) を使用して、複数の IAM アクションまたは複数のリソースを照合します。パッケージグループのパターンも * 文字を使用します。単一のパッケージグループに一致する IAM ポリシーをより簡単に記述するために、パッケージグループの ARN 形式は、エンコードされたバージョンのパッケージグループパターンを使用します。

具体的なパッケージグループの ARN 形式は次のとおりです。

```
arn:aws:codeartifact:region:account-ID:package-
group/my_domain/encoded_package_group_pattern
```

ここで、エンコードされたパッケージグループパターンは、特定の特殊文字がパーセントエンコーディングされた値に置き換えられたパッケージグループパターンです。文字とそれに対応するパーセントエンコーディングされた値のリストは次のとおりです。

- * : %2a
- $: %24
- % : %25

例えば、ドメインのルートパッケージグループの ARN (/*) は次のようになります。

```
arn:aws:codeartifact:us-east-1:111122223333:package-group/my_domain/%2a
```

リストに含まれていない文字はエンコードできず、ARN では大文字と小文字が区別されるため、* は、%2A ではなく %2a としてエンコードする必要があります。

トラブルシューティング AWS CodeArtifact アイデンティティとアクセス

次の情報は、CodeArtifact と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

CodeArtifact でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な codeartifact:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 codeartifact:GetWidget on resource: my-example-widget
```

この場合、codeartifact:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに CodeArtifact リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- CodeArtifact の今後の対応予定機能の詳細については、「How AWS CodeArtifact と IAM の連携」を参照してください。

- 所有 AWS アカウント する のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント する別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティーが所有する へのアクセスを提供する AWS アカウント」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

Amazon VPCエンドポイントの使用

CodeArtifactがインターフェイスの仮想プライベートクラウド (VPC)エンドポイントを使用するように設定することで、VPCのセキュリティを向上させることができます。

VPC エンドポイントは AWS PrivateLink、プライベート IP アドレスを介して CodeArtifact APIs にアクセスできるようにするサービスである を使用します。 は、VPC と CodeArtifact 間のすべてのネットワークトラフィックを AWS ネットワーク AWS PrivateLink に制限します。インターフェイス VPC エンドポイントを使用する場合、インターネットゲートウェイ、NAT デバイス、仮想プライベートゲートウェイは必要ありません。詳細については、「Amazon Virtual Private Cloud User Guide」の「VPC Endpoints」を参照してください。

> **Important**
>
> - VPC エンドポイントは、クロスAWS リージョンリクエストをサポートしていません。CodeArtifact への API コールを発行する予定のリージョンと同じ AWS リージョンにエンドポイントを作成してください。
> - VPC エンドポイントでは、Amazon Route 53 を介して Amazon 提供の DNS のみがサポートされています。独自のDNSを使用する場合には、条件付きDNS転送を使用できます。詳細については、Amazon Virtual Private Cloud ユーザーガイドの「DHCP オプションセット」を参照してください。
> - VPCエンドポイントにアタッチされたセキュリティグループでは、VPCのプライベートサブネットから、ポート443で着信接続を許可する必要があります。

トピック

- CodeArtifact用の VPC エンドポイントを作成する
- Amazon S3ゲートウェイエンドポイントを作成する
- VPCから CodeArtifactを使用する
- CodeArtifact用の VPCエンドポイントポリシーを作成する

CodeArtifact用の VPC エンドポイントを作成する

CodeArtifact の仮想プライベートクラウド (VPC) エンドポイントを作成するには、Amazon EC2 `create-vpc-endpoint` AWS CLI コマンドを使用します。詳細については、Amazon Virtual

Private Cloud ユーザーガイド のインターフェイス VPC エンドポイント(AWS PrivateLink) を参照してください。

CodeArtifact へのすべてのリクエストが AWS ネットワーク内にあるように、2 つの VPC エンドポイントが必要です。最初のエンドポイントはCodeArtifact API の呼び出しに使用されます(例えば、`GetAuthorizationToken`そして`CreateRepository`)。

```
com.amazonaws.region.codeartifact.api
```

2 番目のエンドポイントは、パッケージマネージャーとビルドツール (例えば、npm や Gradle) を使用して CodeArtifact リポジトリにアクセスするために使用されます。

```
com.amazonaws.region.codeartifact.repositories
```

次のコマンドは、CodeArtifactリポジトリにアクセスするためのエンドポイントを作成します。

```
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
```

次のコマンドは、パッケージマネージャーとビルドツールにアクセスするためのエンドポイントを作成します。

```
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**
>
> `codeartifact.repositories`エンドポイントを作成する際には、`--private-dns-enabled`オプションを使用してプライベートDNSホスト名を作成する必要があります。 `codeartifact.repositories` エンドポイントの作成時にプライベート DNS ホスト名を作成できない、または作成しない場合は、VPC から CodeArtifact でパッケージマネージャーを使用するには、追加の設定手順が必要です。詳細については「プライベート DNS なしで `codeartifact.repositories` エンドポイントを使用する」を参照してください。

VPC エンドポイントを作成した後、CodeArtifact でエンドポイントを使用するには、セキュリティグループルールの追加設定が必要な場合があります。Amazon VPC のセキュリティグループの詳細については、「Security groups」を参照してください。

CodeArtifact への接続に問題がある場合は、VPC Reachability Analyzer ツールを使用して問題をデバッグできます。詳細については、「What is VPC Reachability Analyzer?」を参照してください。

Amazon S3ゲートウェイエンドポイントを作成する

CodeArtifactは、Amazon Simple Storage Service (Amazon S3)を使用してパッケージアセットを保存します。CodeArtifactからパッケージを取得するには、Amazon S3用のゲートウェイエンドポイントを作成する必要があります。ビルドまたはデプロイメントプロセスが CodeArtifactからパッケージをダウンロードする場合、CodeArtifactにアクセスしてパッケージのメタデータを取得し、Amazon S3でパッケージアセット (例えば Maven `.jar` ファイルなど) をダウンロードする必要があります。

> **Note**
>
> Python または Swift パッケージ形式を使用する場合、Amazon S3 エンドポイントは不要です。

CodeArtifact の Amazon S3 ゲートウェイエンドポイントを作成するには、Amazon EC2 `create-vpc-endpoint` AWS CLI コマンドを使用します。エンドポイントを作成するときは、必ず VPC 用のルートテーブルを選択してください。詳細については、Amazon Virtual Private Cloud ユーザーガイド の ゲートウェイ VPCエンドポイント を参照してください。

次のコマンドは、Amazon S3エンドポイントを作成します。

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --service-name com.amazonaws.region.s3 \
  --route-table-ids routetableid
```

AWS CodeArtifact の Amazon S3 バケットの最小アクセス許可

Amazon S3ゲートウェイエンドポイントは IAM ポリシードキュメントを使用してサービスへのアクセスを制限します。 CodeArtifactに最小限のAmazon S3 バケット権限のみを許可するには、エンドポイント用のIAM ポリシードキュメントを作成する際に、CodeArtifactが使用するAmazon S3バケットへのアクセスを制限してください。

次の表では、各リージョンで CodeArtifactへのアクセスを許可するためにポリシーで参照する必要がある Amazon S3バケットについて説明します。

リージョン	Amazon S3 バケットの ARN
us–east–1	arn:aws:s3:::assets-193858265520-us-east-1
us-east-2	arn:aws:s3:::assets-250872398865-us-east-2
us-west-2	arn:aws:s3:::assets-787052242323-us-west-2
eu-west-1	arn:aws:s3:::assets-438097961670-eu-west-1
eu-west-2	arn:aws:s3:::assets-247805302724-eu-west-2
eu-west-3	arn:aws:s3:::assets-762466490029-eu-west-3
eu-north-1	arn:aws:s3:::assets-611884512288-eu-north-1
eu-south-1	arn:aws:s3:::assets-484130244270-eu-south-1
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

`aws codeartifact describe-domain`コマンドを使用すると、CodeArtifactドメインを使用することによりAmazon S3バケットを取得することができます。

```
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"
  }
}
```

例

次の例は、us-east-1リージョン内の CodeArtifactの操作に必要な Amazon S3バケットへのアクセスを提供する方法を示しています。その他のリージョンの場合は、上記の表に基づいて、お住まいの地域の正しいアクセス許可 ARNでResourceエントリを更新してください。

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::assets-193858265520-us-east-1/*"]
    }
  ]
}
```

VPCから CodeArtifactを使用する

「CodeArtifact用の VPC エンドポイントを作成する」で作成した com.amazonaws.*region*.codeartifact.repositories VPC エンドポイントでプ

ライベート DNS を有効にできない、または有効にしない場合は、VPC から CodeArtifact を使用するようにリポジトリエンドポイントに別の設定を使用する必要があります。com.amazonaws.*region*.codeartifact.repositories エンドポイントでプライベート DNS が有効になっていない場合は、「プライベート DNS なしで codeartifact.repositories エンドポイントを使用する」の手順に従って CodeArtifact を設定します。

プライベート DNS なしで codeartifact.repositories エンドポイントを使用する

「CodeArtifact用の VPC エンドポイントを作成する」で作成した com.amazonaws.*region*.codeartifact.repositories VPC エンドポイントでプライベート DNS を有効にできない、または有効にしない場合は、以下の手順に従って、パッケージマネージャーに正しい CodeArtifact URL を設定する必要があります。

1. 次のコマンドを実行して、ホスト名をオーバーライドするために使用する VPC エンドポイントを見つけます。

   ```
   $ aws ec2 describe-vpc-endpoints --filters Name=service-
   name,Values=com.amazonaws.region.codeartifact.repositories \
     --query 'VpcEndpoints[*].DnsEntries[*].DnsName'
   ```

   出力は次のようになります。

   ```
   [
     [
       "vpce-0743fe535b883ffff-76ddffff.d.codeartifact.us-west-2.vpce.amazonaws.com"
     ]
   ]
   ```

2. VPC エンドポイントパスを更新して、パッケージ形式、CodeArtifact のドメイン名、および CodeArtifact リポジトリ名を含めます。次の例を参照してください。

   ```
   https://vpce-0743fe535b883ffff-76ddffff.d.codeartifact.us-
   west-2.vpce.amazonaws.com/format/d/domain_name-domain_owner/repo_name
   ```

   エンドポイントの例の以下のフィールドを置き換えます。

   - *format*: 有効な CodeArtifact パッケージフォーマット (npm または pypi など) に置き換えます。

- *domain_name*: パッケージをホストする CodeArtifact リポジトリを含む CodeArtifact ドメインに置き換えます。
- *domain_owner*: CodeArtifact ドメインの所有者の ID に置き換えます (111122223333 など)。
- *repo_name*: パッケージをホストする CodeArtifact リポジトリに置き換えます。

次の URL は、npm リポジトリエンドポイントの例です。

```
https://vpce-0dc4daf7fca331ed6-et36qa1d.d.codeartifact.us-
west-2.vpce.amazonaws.com/npm/d/domainName-111122223333/repoName
```

3. 前の手順で更新した VPC エンドポイントを使用するようにパッケージマネージャーを設定します。CodeArtifact `login` コマンドを使用せずにパッケージマネージャーを設定する必要があります。各パッケージ形式の設定手順については、以下のドキュメントを参照してください。

   - npm: login コマンドを使用せずに npm を設定する
   - nuget: Configure nuget or dotnet without the login command
   - pip: ログインコマンドを使用せずにpipを設定する
   - twine: CodeArtifact で twine を設定して使用する
   - Gradle: Gradle で CodeArtifact を使用する
   - mvn: mvn で CodeArtifact を使用する

CodeArtifact用の VPCエンドポイントポリシーを作成する

CodeArtifact 用の VPC エンドポイントポリシーを作成するには、以下を指定します:

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- 自身に対してアクションを実行できたリソース。

この例のポリシーでは、アカウント 123456789012のプリンシパル
が`GetAuthorizationToken`API を呼び出して CodeArtifactリポジトリからパッケージを取得できることを指定しています。

```
{
```

```
  "Statement": [
    {
      "Action": [
        "codeartifact:GetAuthorizationToken",
        "codeartifact:GetRepositoryEndpoint",
        "codeartifact:ReadFromRepository",
        "sts:GetServiceBearerToken"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Principal": {
         "AWS": "arn:aws:iam::123456789012:root"
       }
    }
  ]
}
```

AWS CloudFormation での CodeArtifact リソースの作成

CodeArtifact は AWS CloudFormation と統合されています。これは、AWS リソースをモデル化してセットアップするためのサービスで、リソースとインフラストラクチャの作成と管理に費やす時間を短縮できます。必要なすべての AWS リソースを記述するテンプレートを作成すると、これらのリソースが CloudFormation で自動的にプロビジョニングおよび設定されます。

CloudFormation を使用すると、テンプレートを再利用して CodeArtifact リソースを同じように繰り返してセットアップできます。リソースを一度記述するだけで、同じリソースを複数のアカウントと複数の AWS リージョンで何度でも繰り返してプロビジョニングできます。

CodeArtifact および CloudFormation テンプレート

CodeArtifact および関連サービスのリソースをプロビジョニングして設定するには、CloudFormation テンプレート について理解しておく必要があります。テンプレートは、JSON や YAML でフォーマットされたテキストファイルです。これらのテンプレートには、CloudFormation スタックにプロビジョニングしたいリソースを記述します。JSON や YAML に不慣れな方は、CloudFormation デザイナー を使えば、CloudFormation テンプレートを使いこなすことができます。詳細については、AWS CloudFormation ユーザーガイドの「What is AWS CloudFormation Designer? (AWS CloudFormation デザイナーとは)」を参照してください。

CodeArtifact は、CloudFormation でのドメイン、リポジトリ、パッケージグループの作成をサポートしています。これらのリソースタイプの JSON テンプレートと YAML テンプレートの例を含む詳細情報については、「CloudFormation ユーザーガイド」の以下のトピックを参照してください。

- AWS::CodeArtifact::Domain
- AWS::CodeArtifact::Repository
- AWS::CodeArtifact::PackageGroup

CodeArtifact リソースの削除の防止

CodeArtifact リポジトリには、失われた場合に簡単に再作成できない可能性がある、重要なアプリケーション依存関係が含まれています。CloudFormation を使用して CodeArtifact リソースを管理するときに CodeArtifact リソースが誤って削除されないようにするには、すべてのドメインとリポジトリに値 `Retain` を含む `DeletionPolicy` および `UpdateRetainPolicy` 属性を含めます。これにより、スタックテンプレートからリソースが削除されたり、スタック全体が誤って削除されたりし

ても、削除を防ぐことができます。次の YAML スニペットは、これらの属性を持つ基本的なドメインとリポジトリを示しています。

```
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
```

これらの属性の詳細については、「AWS CloudFormation ユーザーガイド」の「DeletionPolicy」と「UpdateReplacePolicy」を参照してください。

CloudFormation の詳細はこちら

CloudFormation の詳細については、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

AWS CodeArtifact のトラブルシューティング

以下の情報は、CodeArtifactでの一般的な問題のトラブルシューティングに役立ちます。

フォーマットに固有の問題のトラブルシューティングについては、以下のトピックを参照してください。

- Maven のトラブルシューティング
- Swift のトラブルシューティング

通知を表示できません

問題: デベロッパーツールコンソールで、[設定] から [通知] を選択すると、アクセス許可エラーが表示されます。

解決方法: 通知はデベロッパーツールコンソールの機能ですが、CodeArtifactは現在通知をサポートしていません。CodeArtifactの管理ポリシーには、ユーザーが通知を表示または管理できるようにする権限は含まれていません。デベロッパーツールコンソールで他のサービスを使用し、それらのサービスが通知をサポートしている場合、それらサービスの管理ポリシーには、サービスの通知を表示および管理するために必要な権限が含まれます。

リソースのタグ付け

タグは、ユーザーまたは AWS が AWS リソースに割り当てるカスタム属性ラベルです。各 AWS タグは 2 つの部分で構成されます:

- タグキー (CostCenter、Environment、Project、Secret など)。タグキーでは、大文字と小文字が区別されます。
- タグ値と呼ばれるオプションのフィールド (111122223333、Production、チーム名など)。タグ値を省略すると、空の文字列を使用した場合と同じになります。タグキーと同様に、タグ値では大文字と小文字が区別されます。

これらを合わせて、キーと値のペアと呼ばれます。

タグは、AWS リソースの識別や整理に役立ちます。多くの AWS のサービスではタグ付けがサポートされるため、さまざまなサービスまでリソースの関連を示すことができリソースに同じタグを割り当てることができます。例えば、AWS CodeBuild プロジェクトに割り当てたものと同じタグをリポジトリに割り当てることができます。

タグを使用する際のヒントやベストプラクティスについては、「AWS リソースのタグ付けのベストプラクティス」ホワイトペーパーを参照してください。

CodeArtifactでは、以下のリソースタイプにタグを付けることができます。

- CodeArtifact でリポジトリをタグ付けする
- CodeArtifact でドメインをタグ付けする

コンソール、AWS CLI、CodeArtifact API、または AWSSDK は以下の目的のため使用できます:

- ドメインまたはリポジトリの作成時にタグを追加できます。*
- ドメインまたはリポジトリのタグを追加、管理、削除します。

*ドメインまたはリポジトリをコンソール内で作成する場合は、ドメインにタグを追加することはできません。

タグを使用してリソースを識別、整理、追跡するだけでなく、IAM ポリシーのタグを使って、リソースを表示および操作できるユーザーを制御することもできます。タグベースのアクセスポリシー

の例については、「タグを使用した CodeArtifact リソースへのアクセスのコントロール」を参照してください。

タグを使用した CodeArtifact のコスト配分

タグを使用して、CodeArtifact のストレージコストとリクエストコストの両方を割り当てることができます。

CodeArtifact でのデータストレージコストの割り当て

データストレージコストはドメインに関連付けられているため、CodeArtifact ストレージコストを割り当てるには、ドメインに適用される任意のタグを使用できます。ドメインへのタグの追加については、「CodeArtifact でドメインをタグ付けする」を参照してください。

CodeArtifact でのリクエストコストの割り当て

ほとんどのリクエスト使用状況はリポジトリに関連付けられているため、CodeArtifact リクエストのコストを割り当てるには、リポジトリに適用される任意のタグを使用できます。リポジトリへのタグの追加については、「CodeArtifact でリポジトリをタグ付けする」を参照してください。

一部のリクエストタイプはリポジトリではなくドメインに関連付けられているため、リクエストの使用状況とリクエストに関連するコストはドメインのタグに割り当てられます。リクエストタイプがドメインとリポジトリのどちらに関連付けられているかを判断する最良の方法は、「サービス認可リファレンス」の「AWS CodeArtifact で定義されるアクション」の表を使用することです。アクション列でリクエストタイプを探し、対応するリソースタイプ列の値を確認します。リソースタイプがドメインの場合、そのタイプのリクエストはドメインに請求されます。リソースタイプがリポジトリまたはパッケージの場合、そのタイプのリクエストはリポジトリに請求されます。アクションの中には両方のリソースタイプが表示されるものもありますが、これらのアクションの場合、課金されるリソースはリクエストで渡される値によって異なります。

AWS CodeArtifact のクォータ

次の表では、CodeArtifactのリソースクォータについて説明します。CodeArtifact のサービスエンドポイントのリストとともにリソースクォータを表示するには、「Amazon Web Services 全般のリファレンス」の「AWS service quotas」を参照してください。

次の CodeArtifactリソースクォータに対して、サービスクォータの引き上げリクエストをすることができます。クォータ引き上げのリクエストの詳細については、AWS サービスクォータ を参照してください。

名前	デフォルト	引き上げ可能	説明
アセットファイルのサイズ	サポートされている各リージョン: 5 ギガバイト	あり	アセットあたりの最大ファイルサイズ。
パッケージバージョンあたりのアセット	サポートされている各リージョン: 150	いいえ	パッケージバージョンあたりの最大アセット数。
1 秒あたりの CopyPackageVersions リクエスト	サポートされている各リージョン : 5	あり	CopyPackageVersions に対して実行できる 1 秒あたりの呼び出しの最大数。
リポジトリあたりのダイレクトアップストリーム	サポートされている各リージョン: 10	いいえ	リポジトリあたりの直接アップストリームリポジトリの最大数。
AWS アカウントあたりのドメイン	サポートされている各リージョン: 10	あり	AWS アカウントごとに作成できるドメインの最大数。

名前	デフォルト	引き上げ可能	説明
1 秒あたりの GetAuthorizationToken リクエスト	サポートされている各リージョン: 40	可能	1 秒あたりの取得される認可トークンの最大数。
1 秒あたりの GetPackageVersionAsset リクエスト	サポートされている各リージョン: 50	可能	GetPackageVersionAsset に対して実行できる 1 秒あたりの呼び出しの最大数。
1 秒あたりの ListPackageVersionAssets リクエスト	サポートされている各リージョン: 200	あり	ListPackageVersion Assets に対して実行できる 1 秒あたりの呼び出しの最大数。
1 秒あたりの ListPackageVersions リクエスト	サポートされている各リージョン: 200	あり	ListPackageVersions に対して実行できる 1 秒あたりの呼び出しの最大数。
1 秒あたりの ListPackages リクエスト	サポートされている各リージョン: 200	あり	ListPackages に対して実行できる 1 秒あたりの呼び出しの最大数。
1 秒あたりの PublishPackageVersion リクエスト数	サポートされている各リージョン: 10	あり	PublishPackageVersion に対して実行できる 1 秒あたりの呼び出しの最大数。
1 つの AWS アカウントからの 1 秒あたりの読み取りリクエスト数	サポートされている各リージョン: 800	あり	1 秒あたり 1 つの AWS アカウントからの読み取りリクエストの最大数。

名前	デフォルト	引き上げ可能	説明
ドメインあたりのリポジトリ	サポートされている各リージョン: 1,000	あり	ドメインごとに作成できるリポジトリの最大数。
1 つの認証トークンを使用した 1 秒あたりのリクエスト数	サポートされている各リージョン: 1,200	いいえ	単一の認証トークンを使用した 1 秒あたりの最大リクエスト数。
IP アドレスあたりの認証トークンがないリクエスト	サポートされている各リージョン: 600	いいえ	1 つの IP アドレスから行われる、認証トークンを使用しない 1 秒あたりのリクエストの最大数。
検索されたアップストリームリポジトリ	サポートされている各リージョン: 25	いいえ	パッケージを解決するときに検索されるアップストリームリポジトリの最大数。
1 つの AWS アカウントからの 1 秒あたりの書き込みリクエスト数	サポートされている各リージョン: 100	可能	1 秒あたり 1 つの AWS アカウントからの書き込みリクエストの最大数。

> **Note**
>
> 一般的に、CodeArtifact に対して行われた各読み取りリクエストは、1 つのクォータに対して 1 つのリクエストとしてカウントされます。ただし、Ruby パッケージ形式の場合は、`/api/v1/dependencies` オペレーションへの 1 回の読み取りリクエストで、複数のパッケージに関するデータをリクエストできます。

例えば、リクエストは `https://${CODEARTIFACT_REPO_ENDPOINT}/api/v1/dependencies?gems=gem1,gem2.gem3` のようになります。このリクエスト例は、クォータに対する 3 つのリクエストとしてカウントされます。
なお、複数のリクエストはサービスクォータにのみ適用されます。請求には適用されません。この例では、1 つのリクエストに対してのみ請求されますが、サービスクォータに対する 3 つのリクエストとしてカウントされます。複数の同時`bundle install`オペレーションを実行する CI/CD 環境の場合、有効なリクエストレートは HTTP リクエスト数よりも大幅に高くなります。Ruby Gem の解決中にスロットリングが発生した場合は、1 つの AWS アカウントから 1 秒あたりの読み取りリクエストのクォータの引き上げをリクエストします。

AWS CodeArtifact ユーザーガイドのドキュメント履歴

以下の表は CodeArtifact ドキュメントの重要な変更点をまとめたものです。

変更	説明	日付
CodeArtifact での Cargo の設定と使用に関するドキュメントを追加しました	CodeArtifact で Cargo クレートがサポートされました。CodeArtifact リポジトリを使用するための Cargo の設定に関するガイダンスを含むドキュメントを追加しました。詳細については、「Cargo で CodeArtifact を使用する」を参照してください。	2024 年 6 月 20 日
CodeArtifact での Ruby の設定と使用に関するドキュメントを追加しました	CodeArtifact で Ruby の gem がサポートされました。CodeArtifact リポジトリを使用するための Ruby パッケージマネージャーの設定に関するガイダンスを含むドキュメントを追加しました。詳細については、「Ruby で CodeArtifact を使用する」を参照してください。	2024 年 4 月 30 日
カスタマーマネージドキーを使用してドメインを作成するための AWS KMS キーポリシーの例を追加しました	CodeArtifact ドメインのアセットを暗号化するためのカスタマーマネージド KMS キーの作成に使用できるキーポリシーの例を追加しました。詳細については、「AWS KMS キーポリシーの例」を参照してください。	2024 年 4 月 18 日

パッケージグループの作成をサポートするドキュメントを追加しました。	CodeArtifact のパッケージグループの管理と使用に関するドキュメントを追加しました。詳細については、「CodeArtifact でのパッケージグループの操作」を参照してください。	2024 年 3 月 21 日
aws codeartifact login コマンドに関するドキュメントに有効なパッケージマネージャーを追加しました。	`aws codeartifact login` コマンドで使用する有効なパッケージマネージャーのリストに `dotnet`、`nuget`、`swift` を追加しました。詳細については、「AWSCodeArtifact認証とトークン」を参照してください。	2024 年 2 月 18 日
Swift トラブルシューティングドキュメントに CI マシンでの Xcode ハングに関するエントリを追加しました	パスワードのキーチェーンプロンプトが原因で、CI マシン上で Xcode がハングする可能性のある問題について、その解決策を含む情報を追加しました。詳細については、「パスワードのキーチェーンプロンプトが原因で CI マシンで Xcode が ハングする」を参照してください。	2024 年 2 月 6 日

npm 8.x 以降で npm パッケージのインストール時間が遅くなる場合のトラブルシューティングに関する情報を追加しました	ビルド時間が遅くなる可能性のある、CodeArtifact からの遅い npm パッケージのインストール時間の回避に関する情報を追加しました。詳細については、「npm 8.x 以降でのインストールが遅い場合のトラブルシューティング」を参照してください。	2023 年 12 月 29 日
CodeArtifact の Python パッケージアセットとメタデータの動作に関する情報を更新しました	CodeArtifact リポジトリが Python パッケージバージョンのアセットとメタデータを保持および更新する方法に関する情報を更新しました。詳細については、「アップストリームと外部接続からの Python パッケージのリクエスト」を参照してください。	2023 年 12 月 14 日
CodeArtifact のモニタリングに関するドキュメントを再編成しました	CodeArtifact イベントのモニタリングに関する情報を再編成し、Amazon CloudWatch メトリクスによる CodeArtifact リクエストの表示に関する情報を追加しました。詳細については、「CodeArtifact をモニタリングする」を参照してください。	2023 年 12 月 14 日

を使用した CodeArtifact リソースの管理に関する詳細情報を追加しました。CloudFormation	CloudFormation で管理されている CodeArtifact リソースの削除防止に関するセクションを含む、CloudFormation による CodeArtifact リソースの管理に関するドキュメントへの参照とリンクを追加しました。詳細については、「CodeArtifact リソースの削除の防止」を参照してください。	2023 年 12 月 7 日
CodeArtifact による AWS KMS 外部キーストア (XKS) のサポートに関する詳細ドキュメントを追加	CodeArtifact による KMS キーのサポートについての情報を記載したセクションを追加しました。これには、CodeArtifact での XKS キーの使用も含まれます。詳細については、「CodeArtifact でサポートされている AWS KMS キーのタイプ」を参照してください。	2023 年 10 月 31 日
既存のトラブルシューティングドキュメントを更新し、新しいトラブルシューティングドキュメントを追加しました	Maven に関するトラブルシューティングトピックを追加し、一般的なトラブルシューティングトピックに Swift と Maven のトラブルシューティングドキュメントへのリンクを追加しました。詳細については、「AWS CodeArtifact のトラブルシューティング」を参照してください。	2023 年 9 月 28 日

Swift パッケージマネージャーの公開コマンドを追加してドキュメントを更新しました	Swift 5.9 では、Swift パッケージを作成してパッケージリポジトリに公開する `swift package-registry publish` コマンドが導入されました。このコマンドの使用方法を追加して、Swift のドキュメントを更新しました。詳細については、「Swift で CodeArtifact を使う」を参照してください。	2023 年 9 月 25 日
CodeArtifact での Swift の設定に関するドキュメントを追加しました	CodeArtifact は Swift パッケージをサポートするようになりました。CodeArtifact リポジトリを使用するための Swift の設定に関するガイダンスを含むドキュメントを追加しました。詳細については、「Swift で CodeArtifact を使う」を参照してください。	2023 年 9 月 20 日
CodeArtifact での削除された Python パッケージバージョンの処理方法に関するガイダンスを追加しました	Python パッケージのバージョンが削除されているかどうかを確認する方法、CodeArtifact での削除されたパッケージバージョンの処理方法、よくある質問への回答などの情報を記載したドキュメントを追加しました。詳細については、「削除されたパッケージバージョン」を参照してください。	2023 年 8 月 2 日

Yarn ドキュメントの誤ったコマンドラインコマンドを修正しました	Yarn に関するドキュメントで、CodeArtifact 認可トークンを取得して環境変数に保存するコマンドラインのコマンドの誤記を修正しました。	2023 年 7 月 20 日
Python ドキュメントへの軽微な追加と誤記の修正	それぞれのドキュメントに pip と twine の情報を追加し、twine で `codeartifact login` コマンドを使用した場合の動作を修正しました。詳細については、「CodeArtifact で pip を設定して使用する」および「CodeArtifact で twine を設定して使用する」を参照してください。	2023 年 7 月 14 日
CodeBuild ドキュメント内の dotnet コマンドに関する誤記を修正しました	CodeBuild で NuGet パッケージを使用する ドキュメントの `dotnet add package` コマンドの記載を修正しました。	2023 年 7 月 13 日
Updated AWS CodeArtifact と AWS Identity and Access Management ドキュメント	CodeArtifact ドキュメントの IAM を見直し、他の AWS サービスのドキュメントとの明確さと一貫性を追加しました。「AWS CodeArtifact の Identity and Access Management」を参照してください。	2023 年 5 月 24 日

削除された Python パッケージバージョンに関する情報を追加しました	CodeArtifact が削除された Python パッケージバージョンのメタデータをどのように保持するかについての情報を追加しました。詳細については、「削除されたパッケージバージョン」を参照してください。	2023 年 4 月 11 日
Clojure のサポートに関する情報を追加しました	Clojure プロジェクトの依存関係の管理など、Clojure のサポートに関する情報を追加しました。詳細については、「CodeArtifact で deps.edn を使用する」を参照してください。	2023 年 3 月 21 日
ジェネリックパッケージの公開に関する情報を追加しました	ジェネリックパッケージに関する情報、およびパッケージコンテンツを AWS CLIで公開およびダウンロードする方法についての情報を追加しました。詳細についてはCodeArtifact でのジェネリックパッケージの使用、ジェネリックパッケージの公開と使用、およびジェネリックパッケージでサポートされるコマンドを参照してください。	2023 年 3 月 10 日
公開時のアセットサイズ制限に関する情報を追加しました	公開時のアセットサイズの制限の説明をパッケージの公開セクションに追加しました。	2022 年 6 月 21 日

外部接続のドキュメントをリファクタリングしました	外部接続のドキュメントを移動し、CodeArtifact リポジトリをパブリックパッケージリポジトリに接続するというユーザーの最終目標に焦点を当てるように再編成しました。さらに、その目標を達成するためのさまざまな方法に関するガイダンスと情報を追加しました。詳細については、「CodeArtifact リポジトリをパブリックリポジトリに接続する」を参照してください。	2022 年 5 月 9 日
Amazon CloudWatch Events の CodeArtifact イベント情報を更新しました	account フィールドに情報を追加し、repositoryAdministrator フィールドを追加しました。詳細については、「CodeArtifact イベントの形式と例」を参照してください。	2022 年 3 月 7 日

プライベート DNS を使用せずに VPC から CodeArtifact を使用するための設定手順を追加しました	`codeartifact.repositories` VPC エンドポイントでプライベート DNS を有効にできない、または有効にしない場合は、VPC から CodeArtifact を使用するようにリポジトリエンドポイントに別の設定を使用する必要があります。詳細については「プライベート DNS なしで `codeartifact.repositories` エンドポイントを使用する」を参照してください。	2022 年 2 月 8 日
パッケージバージョンのステータスを更新するための詳細なドキュメントを追加しました	更新パッケージのバージョンのステータスドキュメントを独自のトピックに拡張しました。必要な IAM アクセス許可、さまざまなシナリオの AWS CLI コマンド例、考えられるエラーなど、パッケージバージョンのステータスを更新するためのドキュメントを追加しました。詳細については「パッケージバージョンのステータスの更新」を参照してください。	2021 年 9 月 1 日

詳細なアクセス許可情報とともに、コピーパッケージバージョンのドキュメントを更新しました。	CodeArtifact の同じドメイン内のあるリポジトリから別のリポジトリにパッケージバージョンをコピーする `aws codeartifact copy-package-versions` コマンドを呼び出すために必要な、IAM およびリソースベースのポリシーアクセス許可に関する詳細を追加しました。詳細情報とともに、送信元および送信先リポジトリに必要なリソースベースのポリシーの例が示されるようになりました。詳細については「パッケージをコピーするのに必要な IAM 権限」を参照してください。	2021 年 8 月 25 日
IntelliJ IDEA で Gradle ビルドを実行するためのドキュメントを更新しました	IntelliJ IDEA で Gradle ビルドを実行するためのドキュメントを更新し、CodeArtifact からプラグインを取得するように Gradle を設定するステップを追加しました。また、`aws codeartifact get-authorization-token` へのインライン呼び出しを使用して、新しい実行ごとに新しい CodeArtifact 認可トークンを作成するオプションを追加しました。詳細については「IntelliJ IDEAで Gradle ビルドを実行する」を参照してください。	2021 年 8 月 23 日

AWS CodeArtifact で Yarn を設定および使用するためのドキュメントを追加	CodeArtifact で npm パッケージを管理するために、 Yarn 1.X と Yarn 2.X を設定して使用するためのドキュメントを追加しました。詳細については「CodeArtifact で Yarn を設定して使用する」を参照してください。	2021 年 7 月 30 日
AWS CodeArtifact が NuGet パッケージをサポートするようになりました	CodeArtifact ユーザーは NuGet パッケージを公開して消費できるようになりました。CodeArtifact リポジトリを使用した `nuget` や `dotnet` のような、Visual Studio と NuGet コマンドラインツールの両方を設定して使用するためのドキュメントを追加しました 詳細については「NuGet でCodeArtifactを使う」を参照してください。	2020 年 11 月 19 日
AWS CodeArtifact でのリソースのタグ付け	AWS CodeArtifact のリポジトリとドメインのタグ付けに関するドキュメントを追加しました。「リソースのタグ付け」を参照してください。	2020 年 10 月 30 日
CodeArtifact が をサポートするようになりました CloudFormation	CodeArtifact ユーザーは CloudFormation テンプレートを使用して CodeArtifact リポジトリとドメインを作成できるようになりました。詳細については、AWS CloudFormation での CodeArtifact リソースの作成 および開始方法を参照してください。	2020 年 10 月 8 日

Amazon VPC で CodeArtifact を使用するための Amazon S3 ゲートウェイエンドポイントの作成に関する情報を追加する	Amazon EC2 AWS CLI コマンドを使用した Amazon S3 ゲートウェイエンドポイントの作成に関する情報を追加しました。このドキュメントには、Amazon VPC 環境で CodeArtifact を使用するために必要な特定のアクセス許可に関する情報も含まれています。「Amazon S3ゲートウェイエンドポイントを作成する」を参照してください。	2020 年 8 月 12 日
curl で Maven アーティファクトを公開し、サードパーティの Maven アーティファクトを公開する	curl で公開する および サードパーティのアーティファクト のガイダンスを追加しました。	2020 年 8 月 10 日
一般提供 (GA)	CodeArtifact ユーザーガイドの初版。	2020 年 6 月 10 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。