Amazon ECR

ユーザーガイド

API バージョン 2015-09-21

Amazon ECR: ユーザーガイド

Table of Contents

Amazon Elastic Container Registry とは

Amazon Elastic Container Registry (Amazon ECR) は、安全性と信頼性に優れたスケーラブルなマネージド AWS Docker レジストリサービスです。Amazon ECR では、AWS IAM を使用してプライベート Docker リポジトリにリソースベースのアクセス権限が付与されるため、特定のユーザーまたは Amazon EC2 インスタンスからリポジトリとイメージにアクセスできるようになります。開発者は、Docker CLI を使用してイメージをプッシュ、プル、および管理できます。

Amazon ECR のコンポーネント

Amazon ECR には、次のコンポーネントが含まれています。

レジストリ

Amazon ECR レジストリは、AWS アカウントごとに用意されています。レジストリ内にイメージリポジトリを作成して、これらのリポジトリにイメージを保存できます。詳細については、「Amazon ECR レジストリ (p. 12)」を参照してください。

認証トークン

Docker クライアントがイメージをプッシュおよびプルするには、AWS ユーザーとして Amazon ECR レジストリに対して認証する必要があります。AWS CLI get-login コマンドは、Docker に渡すための認証情報を提供します。詳細については、「レジストリの認証 (p. 12)」を参照してください。

リポジトリ

Amazon ECR イメージリポジトリには、Docker イメージが含まれています。詳細については、「Amazon ECR リポジトリ (p. 15)」を参照してください。

リポジトリポリシー

リポジトリポリシーを使用して、リポジトリとリポジトリ内のイメージへのアクセス権を制御できます。詳細については、「Amazon ECR のリポジトリポリシー (p. 17)」を参照してください。

イメージ

リポジトリには、Docker イメージをプッシュおよびプルできます。開発システムでこれらのイメージをローカルに使用したり、Amazon ECS タスク定義で使用したりすることができます。詳細については、「Amazon ECR での Amazon ECS イメージの使用 (p. 31)」を参照してください。

Amazon ECR の使用を開始する方法

Amazon ECR を使用するには、AWS Command Line Interface と Docker をインストールするようにセットアップする必要があります。詳細については、「Amazon ECR でのセットアップ (p. 2)」および「Amazon ECR における Docker の基本 (p. 5)」を参照してください。

セットアップが終了したら、「Amazon ECR の使用開始 (p. 10)」のチュートリアルを完了する準備が整います。

Amazon ECR でのセットアップ

AWS にサインアップ済みで、Amazon Elastic Container Service (Amazon ECS) を使用している場合は、すぐに Amazon ECR を使用し始めることができます。Amazon ECR は Amazon ECS の拡張機能であるため、2 つのサービスのセットアッププロセスはよく似ています。AWS CLI と Amazon ECR をともに使用するには、最新の Amazon ECR 機能をサポートしているバージョンの AWS CLI を使用する必要があります。AWS CLI で Amazon ECR 機能のサポートが表示されない場合は、最新バージョンにアップグレードする必要があります。詳細については、http://aws.amazon.com/cli/ を参照してください。

Amazon ECR のセットアップを行うには、以下のタスクを完了します。これらのいずれかの手順をすでに完了している場合、それらをスキップして、カスタム AWS CLI のインストールに進むことができます。

1. AWS へのサインアップ (p. 2)
2. IAM ユーザーを作成する (p. 2)
3. AWS CLI をインストールする (p. 4)

AWS へのサインアップ

AWS にサインアップすると、Amazon ECR などすべてのサービスに対して AWS アカウントが自動的にサインアップされます。料金が発生するのは、実際に使用したサービスの分のみです。

既に AWS アカウントをお持ちの場合は、次のタスクに進んでください。AWS アカウントをお持ちでない場合は、次に説明する手順にしたがってアカウントを作成してください。

AWS アカウントを作成するには

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話のキーパッドを用いて確認コードを入力することが求められます。

次のタスクで AWS アカウント番号が必要となるので、メモしておいてください。

IAM ユーザーを作成する

AWS のサービス (Amazon ECR など) の場合は、サービスにアクセスする際に認証情報を提供する必要があります。このため、サービスのリソースにアクセスする権限があるかどうかがサービスによって判定されます。コンソールを使用するにはパスワードが必要です。AWS アカウントのアクセスキーを作成して、コマンドラインインターフェイスまたは API にアクセスすることができます。ただし、AWS アカウントの認証情報を使って AWS にアクセスすることはお勧めしません。代わりに AWS Identity and Access Management (IAM) を使用することをお勧めします。IAM ユーザーを作成して、管理権限を使ってこのユーザーを IAM グループに追加するか、管理権限を付与します。これで、特殊な URL と IAM ユーザーの認証情報を使って、AWS にアクセスできます。

AWS にサインアップしたけれど、自身の IAM ユーザーをまだ作成していない場合は、IAM コンソールを使用して作成できます。

自分用の管理者ユーザーを作成し、そのユーザーを管理者グループに追加するには (コンソール)

1. AWS アカウント E メールアドレスとパスワードを使用して https://console.aws.amazon.com/iam/ で AWS アカウントのルートユーザー として IAM コンソールにサインインします。

   Note

   以下の**Administrator**IAMユーザーの使用に関するベストプラクティスに従い、ルートユーザー認証情報を安全な場所に保管しておくことを強くお勧めします。ルートユーザーとしてサインインして、少数のアカウントおよびサービス管理タスクのみを実行します。
2. ナビゲーションペインで [Users]、[Add user] の順に選択します。
3. [ユーザー名] に「**Administrator**」と入力します。
4. [AWS マネジメントコンソール access (アクセス)] の横にあるチェックボックスをオンにします。[Custom password (カスタムパスワード)] を選択し、その後テキストボックスに新しいパスワードを入力します。
5. (オプション) AWS では、デフォルトで、新しいユーザーに対して初回のサインイン時に新しいパスワードを作成することが必要です。必要に応じて [User must create a new password at next sign-in (ユーザーは次回のサインイン時に新しいパスワードを作成する必要がある)] のチェックボックスをオフにして、新しいユーザーがサインインしてからパスワードをリセットできるようにできます。
6. [Next: Permissions (次へ: アクセス許可)] を選択します。
7. [Set permissions (アクセス許可の設定)] で、[Add user to group (ユーザーをグループに追加)] を選択します。
8. [Create group] を選択します。
9. [グループの作成] ダイアログボックスで、[グループ名] に「**Administrators**」と入力します。
10. [Filter policies (フィルタポリシー)] を選択し、その後 [AWS managed -job function (AWS 管理ジョブの機能] を選択してテーブルのコンテンツをフィルタリングします。
11. ポリシーリストで、[AdministratorAccess] のチェックボックスをオンにします。次に、[Create group] を選択します。

    Note

    `AdministratorAccess` アクセス許可を使用して、AWS Billing and Cost Management コンソールを使用する前に、IAM ユーザーおよびロールの請求へのアクセスをアクティブ化する必要があります。これを行うには、請求コンソールへのアクセスの委任に関するチュートリアルのステップ 1 の手順に従ってください。
12. グループのリストに戻り、新しいグループのチェックボックスをオンにします。必要に応じて [Refresh] を選択し、リスト内のグループを表示します。
13. [次へ: タグ] を選択します。
14. (オプション) タグをキー - 値のペアとしてアタッチして、メタデータをユーザーに追加します。IAM でのタグの使用の詳細については、『IAM ユーザーガイド』の「IAM エンティティのタグ付け」を参照してください。
15. [Next: Review] を選択して、新しいユーザーに追加するグループメンバーシップのリストを表示します。続行する準備ができたら、[Create user] を選択します。

この同じプロセスを繰り返して新しいグループとユーザーを作成し、AWS アカウントのリソースへのアクセス権をユーザーに付与できます。ポリシーを使用して特定の AWS リソースに対するユーザーのアクセス許可を制限する方法については、「アクセス管理」と「ポリシーの例」を参照してください。

新規の IAM ユーザーとしてサインインするには、AWS コンソールからサインアウトし、次の URL を使用します。このとき、your_aws_account_id はハイフンを除いた AWS アカウント番号です (たとえば AWS アカウント番号が `1234-5678-9012` であれば、AWS アカウント ID は `123456789012` となります)。

```
https://your_aws_account_id.signin.aws.amazon.com/console/
```

作成した IAM ユーザー名とパスワードを入力します。サインインすると、ナビゲーションバーに「your_user_name @ your_aws_account_id」が表示されます。

サインページの URL に AWS アカウント ID を含めない場合は、アカウントのエイリアスを作成します。IAM ダッシュボードから [アカウントの別名を作成] を選択し、エイリアス (会社名など) を入力します。アカウントエイリアスを作成した後、サインインするには、次の URL を使用します。

```
https://your_account_alias.signin.aws.amazon.com/console/
```

アカウントの IAM ユーザーのサインインリンクを確認するには、IAM コンソールを開き、ダッシュボードの [IAM users sign-in link] の下を確認します。

IAM の詳細については、「AWS Identity and Access Management ユーザーガイド」を参照してください。

AWS CLI をインストールする

AWS コマンドラインツールを使用して、システムのコマンドラインでコマンドを発行することで、Amazon ECS および AWS タスクを実行できます。これは、コンソールを使用するよりも高速でより便利になります。コマンドラインツールは、AWS のタスクを実行するスクリプトの作成にも便利です。

AWS CLI を Amazon ECR とともに使用するには、最新の AWS CLI バージョンをインストールします (Amazon ECR の機能は、バージョン 1.9.15 から AWS CLI で利用可能)。AWS CLI のバージョンは、aws --version コマンドで確認できます。AWS CLI のインストールまたは最新バージョンへのアップグレードについては、AWS Command Line Interface ユーザーガイド の「AWS コマンドラインインターフェイスのインストール」を参照してください。

Docker のインストール

Docker CLI を Amazon ECR とともに使用するは、最初にシステムに Docker をインストールする必要があります。Docker のインストールの詳細と、ツールに精通するための方法については、「Amazon ECR における Docker の基本 (p. 5)」を参照してください。

Amazon ECR における Docker の基本

Docker は、Linux コンテナに基づいた分散アプリケーションの構築、実行、テスト、デプロイを可能にするテクノロジーです。Amazon ECR はマネージド型 AWS Docker レジストリサービスです。お客様は一般的な Docker CLI を使用してイメージをプッシュ、プル、管理できます。Amazon ECR 製品の詳細、主なお客様導入事例、よくある質問については、Amazon Elastic Container Registry 製品の詳細ページを参照してください。

このガイドのドキュメントは、読者が Docker の概念と機能を基本的に理解していることを前提としています。Docker の詳細については、「Docker とは」、「Docker の概要」を参照してください。

トピック

Docker のインストール

> **Note**
>
> Docker をインストール済みの場合は、この手順をスキップして「Docker イメージの作成 (p. 6)」に進んでください。

Docker は、Ubuntu のような最新の Linux ディストリビューションから Mac OSX や Windows まで、さまざまなオペレーティングシステムで使用できます。特定のオペレーティングシステムに Docker をインストールする方法の詳細については、Docker インストールガイド を参照してください。

Docker を使用するには、ローカルの開発システムさえも必要ありません。Amazon EC2 をすでに使用している場合は、インスタンスを起動し、Docker をインストールして使用し始めることができます。

Amazon EC2 インスタンスに Docker をインストールするには

1. Amazon Linux 2 AMI でインスタンスを起動します。詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイド の「インスタンスの起動」を参照してください。
2. インスタンスに接続します。詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイド の「Linux インスタンスへの接続」を参照してください。
3. インスタンスでインストールされているパッケージとパッケージキャッシュを更新します。

```
sudo yum update -y
```

4. 最新の Docker Community Edition パッケージをインストールします。

```
sudo amazon-linux-extras install docker
```

5. Docker サービスを開始します。

```
sudo service docker start
```

6. ec2-user を docker グループに追加すると、sudo を使用せずに Docker コマンドを実行できます。

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

7. ログアウトし、再びログインして、新しい docker グループアクセス権限を取得します。これは、現在の SSH ターミナルウィンドウを閉じて、新しいウィンドウでインスタンスに再接続することで達成できます。新しい SSH セッションには適切な docker グループ権限があります。
8. ec2-user が sudo を使用せずに Docker コマンドを実行できることを確認します。

```
docker info
```

Note

場合によっては、Docker デーモンにアクセスするための ec2-user に対するアクセス権限を提供するため、インスタンスを再起動する必要があります。次のエラーが表示された場合は、インスタンスを再起動してください。

```
Cannot connect to the Docker daemon. Is the docker daemon running on this host?
```

Docker イメージの作成

このセクションでは、シンプルなウェブアプリケーションの Docker イメージを作成し、ローカルシステムまたは EC2 インスタンスでテストしてから、コンテナレジストリ (Amazon ECR や Docker Hub など) にプッシュして、ECS タスク定義で使用できるようにします。

シンプルなウェブアプリケーションの Docker イメージを作成するには

1. Dockerfile という名前のファイルを作成します。Dockerfile は、Docker イメージに使用する基本イメージと、そのイメージにインストールして実行するものを記述するマニフェストです。Dockerfile の詳細については、「Dockerfile リファレンス」を参照してください。

```
touch Dockerfile
```

2. 前の手順で作成した Dockerfile を編集し、以下のコンテンツを追加します。

```
FROM ubuntu:16.04

# Install dependencies
RUN apt-get update
RUN apt-get -y install apache2

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

# Configure apache
RUN echo '. /etc/apache2/envvars' > /root/run_apache.sh
RUN echo 'mkdir -p /var/run/apache2' >> /root/run_apache.sh
RUN echo 'mkdir -p /var/lock/apache2' >> /root/run_apache.sh
RUN echo '/usr/sbin/apache2 -D FOREGROUND' >> /root/run_apache.sh
RUN chmod 755 /root/run_apache.sh
```

```
EXPOSE 80

CMD /root/run_apache.sh
```

この Dockerfile は Ubuntu 16.04 イメージを使用します。RUN の手順により、パッケージキャッシュが更新され、ウェブサーバー用のいくつかのソフトウェアパッケージがインストールされてから、「Hello World!」のコンテンツがウェブサーバーのドキュメントルートに書き込まれます。EXPOSE 命令はコンテナ上のポート 80 を公開し、CMD 命令はウェブサーバーを起動します。

3. Dockerfile から Docker イメージを作成します。

   Note

   Docker の一部のバージョンでは、下に示す相対パスの代わりに、次のコマンドで Dockerfile への完全パスが必要になる場合があります。

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

4. docker images を実行して、イメージが正しく作成されたことを確認します。

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

出力:

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

5. 新しく構築されたイメージを実行します。-p 80:80 オプションは、コンテナ上の公開されたポート 80 をホストシステム上のポート 80 にマッピングします。docker run の詳細については、「Docker run reference」を参照してください。

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

   Note

   Apache ウェブサーバーからの出力はターミナルウィンドウに表示されます。"Could not reliably determine the server's fully qualified domain name" メッセージは無視できます。

6. ブラウザーを開き、Docker を実行している、コンテナのホストサーバーを参照します。

   - EC2 インスタンスを使用している場合、これはサーバーの [Public DNS] 値であり、SSH でインスタンスに接続するときに使用するアドレスと同じです。インスタンスのセキュリティグループでポート 80 上の受信トラフィックを許可していることを確認します。
   - Docker をローカルに実行している場合は、ブラウザで http://localhost/ を参照します。
   - Windows または Mac コンピューターで docker-machine を使用している場合は、docker-machine ip コマンドを使用して Docker のホスト VirtualBox VM の IP アドレスを見つけ、*machine-name* を、使用中の Docker マシンの名前に置き換えます。

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

   ウェブページに「Hello, World!」が表示されます。

7. [Ctrl + C] キーを押して、Docker コンテナを停止します。

(オプション) イメージの Amazon Elastic Container Registry へのプッシュ

Amazon ECR はマネージド型 AWS Docker レジストリサービスです。お客様は一般的な Docker CLI を使用してイメージをプッシュ、プル、管理できます。Amazon ECR 製品の詳細、主なお客様導入事例、FAQ については、Amazon Elastic Container Registry 製品の詳細ページを参照してください。

このセクションでは以下が必要になります。

- AWS CLI がインストールされ、設定されている。AWS CLI がシステムにインストールされていない場合は、『AWS Command Line Interface ユーザーガイド』の「AWS Command Line Interface のインストール」を参照してください。
- ユーザーには Amazon ECR サービスにアクセスするために必要な IAM アクセス許可があります。詳細については、「Amazon ECR マネージドポリシー」を参照してください。

イメージにタグを付け、Amazon ECR にプッシュするには

1. `hello-world` イメージを保存する Amazon ECR リポジトリを作成します。出力の `repositoryUri` に注目してください。

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

出力:

```
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "hello-repository",
        "repositoryArn": "arn:aws:ecr:region:aws_account_id:repository/hello-
repository",
        "createdAt": 1505337806.0,
        "repositoryUri": "aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository"
    }
}
```

2. 前のステップの `repositoryUri` の値で `hello-world` イメージにタグを付けます。

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

3. aws ecr get-login --no-include-email コマンドを実行して、レジストリ用の docker login 認証コマンド文字列を取得します。

   Note

   get-login コマンドは AWS CLI のバージョン 1.9.15 以降で利用できます。ただし、Docker の最新バージョン (17.06 以降) を使用している場合は、バージョン 1.11.91 以降をお勧めします。AWS CLI のバージョンは、aws --version コマンドで確認できます。Docker のバージョン 17.06 以降を使用している場合は、`get-login` の後に `--no-include-email` オプションを含めます。`Unknown options: --no-include-email` エラーが返された場合は、AWS CLI の最新バージョンをインストールします。詳細については、AWS Command Line Interface ユーザーガイド の「AWS コマンドラインインターフェイスのインストール」を参照してください。

```
aws ecr get-login --no-include-email --region region
```

4. 前のステップで返された docker login コマンドを実行します。このコマンドは、12 時間有効な認証トークンを提供します。

   Important

   docker login コマンドを実行すると、システムの他のユーザーに対して、プロセスリスト (ps -e) 表示でコマンド文字列が表示されます。docker login コマンドには認証情報が含まれているため、システムの他のユーザーがこのようにして認証情報を表示するリスクがあります。また、その認証情報を使用して、リポジトリへのプッシュアクセスおよびプルアクセスを取得する可能性があります。安全なシステムを使用していない場合は、このリスクを考慮してください。`-p` *`password`* オプションを省略してインタラクティブにログインし、求められたときにパスワードを入力することを検討してください。
5. 前のステップの値 `repositoryUri` を使用して、Amazon ECR にイメージをプッシュします。

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

(オプション) クリーンアップする

Amazon ECR イメージの試用を終了したら、レポジトリを削除し、イメージストレージに対して料金が発生しないようにします。

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

Amazon ECR の使用開始

Amazon ECR コンソールでリポジトリを作成することで、Amazon Elastic Container Registry (Amazon ECR) の使用を開始します。まず、Amazon ECR は最初のレポジトリを作成するプロセスをガイドするウィザードを実行します。

> **Important**
>
> 開始する前に、必ず「Amazon ECR でのセットアップ (p. 2)」の手順を完了してください。

リポジトリの設定

リポジトリは Amazon ECR で Docker イメージを保存する場所です。Amazon ECR からイメージをプッシュまたはプルするたびに、Docker にイメージをプッシュまたはプルするレジストリおよびリポジトリの場所を指定します。

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. [Get Started] を選択します。
3. [レポジトリの設定] に、リポジトリの一意の名前を入力し、[レポジトリの作成] を選択します。

Docker イメージの構築、タグ付け、プッシュ

ウィザードのこのセクションでは、Docker CLI を使用して既存のローカルイメージ (Dockerfile から構築したもの、または Docker Hub などの別のレジストリから取得したもの) にタグを付け、そのタグ付きイメージを Amazon ECR レジストリにプッシュします。

1. コンソールからターミナルウィンドウに aws ecr get-login コマンドを貼り付けて、Docker クライアントをレジストリに対して認証するために使用できる docker login コマンドを取得します。

   > **Note**
   >
   > get-login コマンドは AWS CLI のバージョン 1.9.15 以降で利用できます。ただし、Docker の最新バージョン (17.06 以降) を使用している場合は、バージョン 1.11.91 以降をお勧めします。AWS CLI のバージョンは、aws --version コマンドで確認できます。Docker のバージョン 17.06 以降を使用している場合は、`get-login` の後に `--no-include-email` オプションを含めます。`Unknown options: --no-include-email` エラーが返された場合は、AWS CLI の最新バージョンをインストールします。詳細については、AWS Command Line Interface ユーザーガイド の「AWS コマンドラインインターフェイスのインストール」を参照してください。

2. 前のステップで返された docker login コマンドを実行します。このコマンドは、12 時間有効な認証トークンを提供します。

   > **Important**
   >
   > docker login コマンドを実行すると、システムの他のユーザーに対して、プロセスリスト (ps -e) 表示でコマンド文字列が表示されます。docker login コマンドには認証情報が含まれているため、システムの他のユーザーがこのようにして認証情報を表示するリスクがあります。また、その認証情報を使用して、リポジトリへのプッシュアクセスおよびプルアクセスを取得する可能性があります。安全なシステムを使用していない場合は、このリスクを考慮してください。`-p` *`password`* オプションを省略してインタラクティブにログインし、求められたときにパスワードを入力することを検討してください。

3. (オプション) プッシュするイメージの Dockerfile がある場合は、イメージを構築し、新しいレポジトリ用にタグを付けます。コンソールからターミナルウィンドウに docker build コマンドを貼り付けます。Dockerfile と同じディレクトリであることを確認します。

4. コンソールからターミナルウィンドウに docker tag コマンドを貼り付けて、ECR レジストリと新しいレポジトリ用のイメージにタグを付けます。コンソールコマンドでは、前のステップの Dockerfile からイメージが構築されたことを前提とします。Dockerfile からイメージを構築していない場合は、*repository*:latest の最初のインスタンスを、プッシュするローカルイメージのイメージ ID またはイメージ名と置き換えます。
5. docker push コマンドをターミナルウィンドウに貼り付けて、新しくタグ付けされたイメージを ECR レポジトリにプッシュします。
6. [Close] を選択します。

Amazon ECR レジストリ

Amazon ECR レジストリを使用すると、高度な可用性を備えたスケーラブルなアーキテクチャでイメージをホストできるため、アプリケーションのコンテナを確実にデプロイできます。リポジトリを使用すると、イメージリポジトリと Docker イメージを管理できます。AWS アカウントごとに、(デフォルト) Amazon ECR レジストリが 1 つずつ提供されます。

レジストリの概念

- デフォルトレジストリの URL は `https://aws_account_id.dkr.ecr.region.amazonaws.com` です。
- デフォルトでは、アカウントにはデフォルトリポジトリ内のリポジトリへの読み取りおよび書き込みアクセス権があります。ただし、IAM ユーザーには、Amazon ECR API への呼び出しと、リポジトリへのイメージのプッシュまたはプルを行うアクセス許可が必要です。Amazon ECR には、さまざまなレベルでユーザーアクセスをコントロールするマネージド型ポリシーがいくつか用意されています。詳細については、「Amazon ECR マネージドポリシー (p. 50)」を参照してください。
- docker push コマンドと docker pull コマンドを使用してレジストリ内のリポジトリに対してイメージをプッシュおよびプルできるように、そのレジストリに対して Docker クライアントを認証する必要があります。詳細については、「レジストリの認証 (p. 12)」を参照してください。
- リポジトリは、IAM のユーザーアクセスポリシーとリポジトリポリシーによって制御できます。

レジストリの認証

AWS マネジメントコンソール、AWS CLI、または AWS SDK を使用して、レポジトリを作成して管理することができます。また、これらの方法を使用して、イメージの一覧表示や削除などのいくつかのアクションをイメージで実行できます。これらのクライアントは、標準の AWS 認証方法を使用します。技術的には Amazon ECR API を使用してイメージをプッシュおよびプルできますが、ほとんどの場合 Docker CLI (または言語固有の Docker ライブラリ) を使用します。

Docker CLI では標準の AWS 認証方法がサポートされていないため、Docker クライアントを別の方法で認証する必要があります。これにより、イメージのプッシュまたはプルをリクエストしているユーザーを Amazon ECR が認識することができます。Docker CLI を使用している場合は、docker login コマンドを使用して Amazon ECR レジストリに対して認証します。Amazon ECR で提供された認証トークンは 12 時間有効です。GetAuthorizationToken API オペレーションは、ユーザー名 (`AWS`) とパスワードを含む base64 でエンコードされた認証トークンを提供します。この認証トークンは、docker login コマンドでデコードして使用できます。ただし、よりシンプルな get-login コマンド (トークンを取得してデコードし、自動的に docker login コマンドに変換) を AWS CLI で使用することができます。

To authenticate Docker to an Amazon ECR registry with get-login

> Note
>
> The get-login command is available in the AWS CLI starting with version 1.9.15; however, we recommend version 1.11.91 or later for recent versions of Docker (17.06 or later). You can check your AWS CLI version with the aws --version command.

1. Run the aws ecr get-login command. The example below is for the default registry associated with the account making the request. To access other account registries, use the `--registry-ids aws_account_id` option. For more information, see get-login in the AWS CLI Command Reference.

```
aws ecr get-login --region region --no-include-email
```

Output:

```
docker login -u AWS -p password https://aws_account_id.dkr.ecr.us-east-1.amazonaws.com
```

Important

If you receive an `Unknown options: --no-include-email` error, install the latest version of the AWS CLI. For more information, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.

The resulting output is a docker login command that you use to authenticate your Docker client to your Amazon ECR registry.

2. Copy and paste the docker login command into a terminal to authenticate your Docker CLI to the registry. This command provides an authorization token that is valid for the specified registry for 12 hours.

Note

If you are using Windows PowerShell, copying and pasting long strings like this does not work. Use the following command instead.

```
Invoke-Expression -Command (aws ecr get-login --no-include-email)
```

Important

When you execute this docker login command, the command string can be visible to other users on your system in a process list (ps -e) display. Because the docker login command contains authentication credentials, there is a risk that other users on your system could view them this way. They could use the credentials to gain push and pull access to your repositories. If you are not on a secure system, you should consider this risk and log in interactively by omitting the `-p password` option, and then entering the password when prompted.

HTTP API 認証

Amazon ECR は Docker Registry HTTP API をサポートしています。ただし、Amazon ECR はプライベートレジストリであるため、すべての HTTP リクエストで認可トークンを提供する必要があります。curl の `-H` オプションを使用して HTTP 認証ヘッダーを追加し、get-authorization-token AWS CLI コマンドで提供される認証トークンを渡します。

Amazon ECR HTTP API を使用して認証するには

1. AWS CLI を使用して認証トークンを取得し、そのトークンを環境変数に設定します。

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

2. API に対して認証するには、`$TOKEN` 変数を curl の `-H` オプションに渡します。たとえば、次のコマンドでは、Amazon ECR レポジトリでイメージタグを一覧表示します。詳細については、Docker レジストリ HTTP API リファレンスドキュメントを参照してください。

```
curl -i -H "Authorization: Basic $TOKEN" https://012345678910.dkr.ecr.us-
east-1.amazonaws.com/v2/amazonlinux/tags/list
```

出力:

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

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

Amazon ECR リポジトリ

Amazon Elastic Container Registry (Amazon ECR) には、イメージリポジトリの作成、モニタリング、削除と、リポジトリにアクセスできるユーザーを制御するアクセス権限を設定する API オペレーションが用意されています。Amazon ECR コンソールの [リポジトリ] セクションでは、同じアクションを実行できます。Amazon ECR には、Docker CLI も統合されており、開発環境とリポジトリの間でイメージをプッシュおよびプルできます。

トピック

リポジトリの概念

- デフォルトでは、アカウントにはデフォルトリポジトリ (*aws_account_id*.dkr.ecr.*region*.amazonaws.com) 内のリポジトリへの読み取りおよび書き込みアクセス権があります。ただし、IAM ユーザーには、Amazon ECR API への呼び出しと、リポジトリへのイメージのプッシュまたはプルを行うアクセス権限が必要です。Amazon ECR には、さまざまなレベルでユーザーアクセスを制御するいくつかのマネージド型ポリシーが用意されています。詳細については、「Amazon ECR 管理ポリシー (p. 50)」を参照してください。
- リポジトリは、IAM のユーザーアクセスポリシーとリポジトリポリシーによって制御できます。詳細については、「Amazon ECR のリポジトリポリシー (p. 17)」を参照してください。
- リポジトリ名では、似たリポジトリをグループ化するのに使用できる名前空間をサポートできます。たとえば、同じレジストリを使用するチームが複数ある場合、チーム A が `team-a` 名前空間を使用しながらチーム B が `team-b` 名前を使用できます。各チームには、`web-app` と呼ばれる独自のイメージを指定できますが、それぞれの名前の先頭にはチームの名前空間が付くため、2 つのイメージは干渉せずに同時に使用できます。チーム A のイメージは `team-a/web-app` と呼ばれますが、チーム B のイメージは `team-b/web-app` と呼ばれます。

リポジトリの作成

Docker イメージを Amazon ECR にプッシュするには、保存先のリポジトリを作成する必要があります。AWS マネジメントコンソール、または AWS CLI と AWS SDK を使用して Amazon ECR リポジトリを作成できます。

リポジトリを作成するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/repositories) を開きます。
2. ナビゲーションバーから、リポジトリを作成するリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [Repositories] ページで、[Create repository] を選択します。
5. [レポジトリの設定] に、リポジトリの一意の名前を入力し、[レポジトリの作成] を選択します。
6. (オプション) 作成したレポジトリを選択し、[プッシュコマンドの表示] を選択して、イメージを新しいリポジトリにプッシュするステップを表示します。

a. コンソールからターミナルウィンドウに aws ecr get-login コマンドを貼り付けて、Docker クライアントをレジストリに対して認証するために使用できる docker login コマンドを取得します。

   Note

   get-login コマンドは AWS CLI のバージョン 1.9.15 以降で利用できます。ただし、Docker の最新バージョン (17.06 以降) を使用している場合は、バージョン 1.11.91 以降をお勧めします。AWS CLI のバージョンは、aws --version コマンドで確認できます。Docker のバージョン 17.06 以降を使用している場合は、`get-login` の後に `--no-include-email` オプションを含めます。`Unknown options: --no-include-email` エラーが返された場合は、AWS CLI の最新バージョンをインストールします。詳細については、AWS Command Line Interface ユーザーガイド の「AWS コマンドラインインターフェイスのインストール」を参照してください。

b. 前のステップで返された docker login コマンドを実行します。このコマンドは、12 時間有効な認証トークンを提供します。

   Important

   docker login コマンドを実行すると、システムの他のユーザーに対して、プロセスリスト (ps -e) 表示でコマンド文字列が表示されます。docker login コマンドには認証情報が含まれているため、システムの他のユーザーがこのようにして認証情報を表示するリスクがあります。また、その認証情報を使用して、リポジトリへのプッシュアクセスおよびプルアクセスを取得する可能性があります。安全なシステムを使用していない場合は、このリスクを考慮してください。`-p` *`password`* オプションを省略してインタラクティブにログインし、求められたときにパスワードを入力することを検討してください。

c. (オプション) プッシュするイメージの Dockerfile がある場合は、イメージを構築し、新しいレポジトリ用にタグを付けます。コンソールからターミナルウィンドウに docker build コマンドを貼り付けます。Dockerfile と同じディレクトリであることを確認します。

d. コンソールからターミナルウィンドウに docker tag コマンドを貼り付けて、ECR レジストリと新しいレポジトリ用のイメージにタグを付けます。コンソールコマンドでは、前のステップの Dockerfile からイメージが構築されたことを前提とします。Dockerfile からイメージを構築していない場合は、*`repository`*`:latest` の最初のインスタンスを、プッシュするローカルイメージのイメージ ID またはイメージ名と置き換えます。

e. docker push コマンドをターミナルウィンドウに貼り付けて、新しくタグ付けされたイメージを ECR レポジトリにプッシュします。

f. [Close] を選択します。

リポジトリ情報の表示

リポジトリを作成したら、その情報を AWS マネジメントコンソール で表示できます。

- リポジトリに保存されているイメージ
- イメージにタグ付けされているかどうか
- イメージのタグ
- イメージの SHA ダイジェスト
- イメージのサイズ (MiB)
- イメージがいつリポジトリにプッシュされたか

Note

Docker バージョン 1.9 以降、Docker クライアントは V2 Docker レジストリにプッシュする前にイメージレイヤーを圧縮します。docker images コマンドの出力には、非圧縮のイメージサイズ

が表示されるため、AWS マネジメントコンソール に表示されるイメージサイズよりも大きいイメージサイズが返されることがあります。

リポジトリ情報を表示するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/repositories) を開きます。
2. ナビゲーションバーから、表示するリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [Repositories] ページで、表示するリポジトリを選択します。
5. [Repositories: ***repository_name***] ページで、ナビゲーションバーを使用してイメージに関する情報を表示します。
   - [イメージ] を選択して、リポジトリ内のイメージに関する情報を表示します。削除するイメージのうちタグが付いていないものがある場合、削除するリポジトリの左側のボックスを選択し、[Delete] を選択できます。詳細については、「イメージの削除 (p. 32)」を参照してください。
   - [Permissions (アクセス許可)] を選択し、リポジトリに適用されるリポジトリポリシーを表示します。詳細については、「Amazon ECR のリポジトリポリシー (p. 17)」を参照してください。
   - [ライフサイクルポリシー] を選択し、リポジトリに適用されるライフサイクルポリシールールを表示します。ライフサイクルイベント履歴もここに表示されます。詳細については、「Amazon ECR ライフサイクルポリシー (p. 34)」を参照してください。
   - [タグ] を選択して、リポジトリに適用されているメタデータタグを表示します。

リポジトリの削除

リポジトリを使用し終わったら、削除できます。AWS マネジメントコンソール でリポジトリを削除すると、リポジトリに含まれているすべてのイメージも削除されます。これを元に戻すことはできません。

リポジトリを削除するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/repositories) を開きます。
2. ナビゲーションバーから、削除するリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [レポジトリ] ページで、削除するレポジトリを選択して [削除] を選択します。
5. [***repository_name*** の削除] ウィンドウで、選択されたリポジトリを削除することを確認し、[削除] を選択します。

   Important

   選択されたリポジトリ内のイメージもすべて削除されます。

Amazon ECR のリポジトリポリシー

Amazon ECR では、リソースベースのアクセス権限を使用してリポジトリへのアクセスを制御します。リソースベースのアクセス権限により、どの IAM ユーザーあるいはロールがリポジトリにアクセスでき、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。リポジトリへの追加のアクセス権限を許可するポリシードキュメントを適用できます。

レポジトリポリシーと IAM ポリシー

Amazon ECR レポジトリポリシーは、個別の Amazon ECR リポジトリへのアクセスを制御することを対象として具体的に使用される IAM ポリシーのサブセットです。IAM ポリシーは一般的に Amazon ECR

サービス全体にアクセス権限を適用するために使用されますが、特定のリソースへのアクセスを制限するために使用することもできます。

Amazon ECR レポジトリと IAM ポリシーはどちらも、特定の IAM ユーザーあるいはロールがレポジトリで実行できるアクションを決定するために使用されます。ユーザーあるいはロールがレポジトリから 1 つのアクションの実行を許可されていながら、IAM ポリシーからはアクセス権限を拒否される場合 (またはその逆の場合)、そのアクションは拒否されます。ユーザーあるいはロールは、レポジトリポリシーまたは IAM ポリシーのいずれかのみでアクションへのアクセスが許可されていることを必要とし、その両方でこのアクションが許可されている必要はありません。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリに対して認証されるか、Amazon ECR レポジトリとの間でイメージをプッシュまたはプルするには、事前に IAM ポリシーを通じて `ecr:GetAuthorizationToken` API へのアクセス権限がユーザーに許可されている必要があります。Amazon ECR は、さまざまなレベルで複数の管理 IAM ポリシーを提供します。詳細については、「Amazon ECR 管理ポリシー (p. 50)」を参照してください。

これらのいずれかのポリシータイプを使用して、次の例に示すようにレポジトリへのアクセスを制御します。

この例は、指定する IAM ユーザーがレポジトリ内でレポジトリとイメージを説明することを許可する Amazon ECR レポジトリポリシーを示しています。

```
{
  "Version": "2008-10-17",
  "Statement": [{
    "Sid": "ECR Repository Policy",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::123456789012:user/MyUsername"
    },
    "Action": [
       "ecr:DescribeImages",
       "ecr:DescribeRepositories"
    ]
  }]
}
```

この例は、リソースパラメータを使用してポリシーで 1 つのレポジトリを対象とすることで、上記と同じ目的を達成する IAM ポリシーを示しています。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::123456789012:user/MyUsername"
    },
    "Action": [
      "ecr:DescribeImages",
      "ecr:DescribeRepositories"
    ],
    "Resource": [
      "arn:aws:ecr:region:123456789012:repository/MyRepository"
    ]
    }]
}
```

トピック

- リポジトリポリシーステートメントの設定 (p. 19)

リポジトリポリシーステートメントの設定

次の手順に従って、AWS マネジメントコンソール でレポジトリにアクセスポリシーステートメントを追加できます。リポジトリごとに複数のポリシーステートメントを追加できます。エンドポイントポリシーの例については、「Amazon ECR リポジトリポリシーの例 (p. 20)」を参照してください。

> Important
>
> Amazon ECR では、ユーザーがレジストリに対して認証されるか、Amazon ECR レポジトリとの間でイメージをプッシュまたはプルするには、事前に IAM ポリシーを通じて `ecr:GetAuthorizationToken` API へのアクセス権限がユーザーに許可されている必要があります。Amazon ECR は、さまざまなレベルで複数の管理 IAM ポリシーを提供します。詳細については、「Amazon ECR 管理ポリシー (p. 50)」を参照してください。

リポジトリポリシーステートメントを設定するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/repositories) を開きます。
2. ナビゲーションバーから、ポリシーステートメントをオンに設定するリポジトリが含まれるリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [Repositories] ページで、ポリシーステートメントをオンに設定するリポジトリを選択します。
5. ナビゲーションペインの [アクセス許可]、[編集] を選択します。
6. [アクセス権限の編集] ページの [ステートメントを追加] を選択します。
7. [ステートメント名] に、ステートメントの名前を入力します。
8. [Effect] で、ポリシーステートメントがアクセスを許可するか拒否するかを選択します。
9. [Principal] で、ポリシーステートメントを適用するユーザーの範囲を選択します。
   - [全員 (*)] チェックボックスをオンにすることにより、すべての認証済み AWS ユーザーにステートメントを適用できます。
   - [AWS アカウント番号] フィールドにアカウント番号を列挙することにより (例: 111122223333)、特定の AWS アカウント下のすべてのユーザーにステートメントを適用できます。
   - [IAMエンティティ] リストでロールまたはユーザーをオンにし、[>> 追加] を選択して [選択した IAM エンティティ] リストに移動することにより、AWS アカウント下のロールまたはユーザーにステートメントを適用できます。

     > Note
     >
     > AWS マネジメントコンソール で現在サポートされていない、より複雑なリポジトリポリシーは、set-repository-policy AWS CLI コマンドを使用して適用できます。
10. [アクション] で、個別の API オペレーションのリストとの間でポリシーステートメントを適用する Amazon ECR API オペレーションの範囲を選択します。
11. 完了したら、[Save] を選択してポリシーを設定します。
12. 追加する各レポジトリポリシーに対して、前のステップを繰り返します。

リポジトリポリシーステートメントの削除

既存のリポジトリポリシーステートメントをリポジトリに適用する必要がなくなった場合、削除できます。

リポジトリポリシーステートメントを削除するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/repositories) を開きます。
2. ナビゲーションバーから、ポリシーステートメントを削除するリポジトリが含まれるリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [Repositories] ページで、ポリシーステートメントを削除するリポジトリを選択します。
5. ナビゲーションペインの [アクセス許可]、[編集] を選択します。
6. [アクセス権限の編集] ページの [削除] を選択します。

Amazon ECR リポジトリポリシーの例

以下の例では、Amazon ECR リポジトリに対してユーザーが所有するアクセス権限を制御するために使用できるポリシーステートメントを示しています。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリに対して認証されるか、Amazon ECR レポジトリとの間でイメージをプッシュまたはプルするには、事前に IAM ポリシーを通じて `ecr:GetAuthorizationToken` API へのアクセス権限がユーザーに許可されている必要があります。Amazon ECR は、さまざまなレベルで複数の管理 IAM ポリシーを提供します。詳細については、「Amazon ECR 管理ポリシー (p. 50)」を参照してください。

トピック

例: アカウント内での IAM ユーザーの許可

次のリポジトリポリシーでは、アカウント内の IAM ユーザーにイメージのプッシュとプルが許可されます。

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "AllowPushPull",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/push-pull-user-1",
          "arn:aws:iam::123456789012:user/push-pull-user-2"
        ]
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability",
        "ecr:PutImage",
        "ecr:InitiateLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:CompleteLayerUpload"
      ]
    }
```

```
  ]
}
```

例: 他のアカウントの許可

次のリポジトリポリシーでは、特定のアカウントにイメージのプッシュが許可されます。

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "AllowCrossAccountPush",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchCheckLayerAvailability",
        "ecr:PutImage",
        "ecr:InitiateLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:CompleteLayerUpload"
      ]
    }
  ]
}
```

次のリポジトリポリシーでは、すべての AWS アカウントにイメージのプルが許可されます。

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "AllowPull",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ]
    }
  ]
}
```

次のリポジトリポリシーでは、一部の IAM ユーザーにイメージのプルが許可されますが (*pull-user-1* および *pull-user-2*)、別のユーザーにはフルアクセスが許可されます (*admin-user*)。

> **Note**
>
> AWS マネジメントコンソール で現在サポートされていない、より複雑なリポジトリポリシーは、set-repository-policy AWS CLI コマンドを使用して適用できます。

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "AllowPull",
      "Effect": "Allow",
```

```
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/pull-user-1",
          "arn:aws:iam::123456789012:user/pull-user-2"
        ]
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ]
    },
    {
      "Sid": "AllowAll",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/admin-user"
      },
      "Action": [
        "ecr:*"
      ]
    }
  ]
}
```

例: すべて拒否

次のリポジトリポリシーでは、すべてのユーザーに対してすべてのイメージのプルが拒否されます。

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "DenyPull",
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ]
    }
  ]
}
```

Amazon ECR リポジトリのタグ付け

Amazon ECR リポジトリを管理しやすいように、必要に応じて、独自のメタデータをタグ形式で各リポジトリに割り当てることができます。ここでは、タグとその作成方法について説明します。

コンテンツ

タグの基本

タグとは、AWS リソースに付けるラベルです。タグはそれぞれ、1 つのキーとオプションの 1 つの値で構成されており、どちらもお客様側が定義します。

タグを使用すると、AWS リソースを用途、所有者、環境などのさまざまな方法で分類できます。同じ型のリソースが多い場合に役立ちます — 割り当てたタグに基づいて特定のリソースをすばやく識別できます。たとえば、各リポジトリの所有者を追跡しやすくするため、アカウントの Amazon ECR リポジトリに対して一連のタグを定義できます。

ニーズを満たす一連のタグキーを考案されることをお勧めします。一貫性のあるタグキーセットを使用することで、リソースの管理が容易になります。追加したタグに基づいてリソースを検索およびフィルタリングできます。

タグには、Amazon ECR に関連する意味はなく、完全に文字列として解釈されます。また、タグは自動的にリソースに割り当てられます。タグのキーと値は編集でき、タグはリソースからいつでも削除できます。タグの値を空の文字列に設定することはできますが、タグの値を null に設定することはできません。特定のリソースについて既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によって上書きされます。リソースを削除すると、リソースのタグも削除されます。

タグは、AWS マネジメントコンソール、AWS CLI、および Amazon ECR API を使用して操作できます。

AWS Identity and Access Management (IAM) を使用している場合は、AWS アカウント内のどのユーザーがタグを作成、編集、削除するアクセス許可を持つかを制御できます。

リソースにタグを付ける

新規または既存の Amazon ECR リポジトリにタグ付けできます。

Amazon ECR コンソールを使用している場合、新規リソースには作成時にタグを適用でき、既存のリソースには、ナビゲーションペインの [Tags (タグ)] オプションを使用していつでもタグを適用できます。

Amazon ECR API、AWS CLI、または AWS SDK を使用している場合、CreateRepository API アクションの `tags` パラメータを使用して新規リポジトリにタグを適用するか、`TagResource` API アクションを使用して既存のリソースにタグを適用することができます。詳細については、「TagResource」を参照してください。

また、リポジトリの作成時にタグを適用できない場合は、リポジトリ作成プロセスがロールバックされます。これにより、リポジトリがタグ付きで作成されるか、まったく作成されないようになるため、タグ付けされていないリポジトリが存在することがなくなります。作成時にリポジトリにタグ付けすることで、リポジトリ作成後にカスタムタグ付けスクリプトを実行する必要がなくなります。

タグの制限

タグには以下のような基本制限があります。

- リポジトリあたりのタグの最大数 – 50
- タグキーは、リポジトリごとにそれぞれ一意である必要があります。また、各タグキーに設定できる値は 1 つのみです。
- キーの最大長 – 128 文字 (Unicode) (UTF-8)
- 値の最大長 – 256 文字 (Unicode) (UTF-8)
- 複数のサービス間およびリソース間でタグ付けスキーマを使用する場合、他のサービスでも許可される文字に制限が適用されることがあるのでご注意ください。一般的に使用が許可される文字は、UTF-8 で表現できる文字、数字、スペース、および +、-、=、.、_、:、/、@。
- タグのキーと値は大文字と小文字が区別されます。
- キーと値のどちらにも `aws:` プレフィックスは使用しないでください。このプレフィックスは AWS で使用するために予約されています。このプレフィックスが含まれるタグのキーや値を編集したり削除す

ることはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限時には計算されません。

請求用のリソースにタグを付ける

Amazon ECR リポジトリに追加するタグは、コスト配分を有効にした後にコストと使用状況レポートでコスト配分を確認するときに便利です。詳細については、「Amazon ECR 使用状況レポート (p. 66)」を参照してください。

リソースを組み合わせたコストを確認するには、同じタグキー値を持つリソースに基づいて、請求情報を整理します。たとえば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することで、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。タグによるコスト配分レポートの設定の詳細については、AWS Billing and Cost Management ユーザーガイドの「毎月のコスト配分レポート」を参照してください。

Note

レポートを有効にすると、約 24 時間後に、今月のデータを表示できるようになります。

コンソールでのタグの処理

新規または既存のリポジトリに関連付けられたタグは、Amazon ECR コンソールを使用して管理することができます。

Amazon ECR コンソールで特定のリポジトリを選択する場合、ナビゲーションペインの [タグ] を選択してタグを表示することができます。

リポジトリにタグを追加するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [Repositories] ページで、表示するリポジトリを選択します。
5. [Repositories: ***repository_name***] ページで、ナビゲーションペインの [タグ] を選択します。
6. [タグ] ページで、[タグの追加]、[タグの追加] の順に選択します。
7. [Edit Tags (タグの編集)] ページで、各タグのキーと値を指定してから [Save (保存)] を選択します。

個々のリソースからタグを削除するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/) を開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. [Repositories] ページで、表示するリポジトリを選択します。
4. [Repositories: ***repository_name***] ページで、ナビゲーションペインの [タグ] を選択します。
5. [タグ] ページで、[編集] を選択します。
6. [Edit Tags (タグの編集)] ページで、削除するタグごとに [Remove (削除)] を選択してから [Save (保存)] を選択します。

AWS CLI または API でのタグの操作

リソースのタグの追加、更新、リスト表示、および削除には、次を使用します。対応するドキュメントに例が記載されています。

Amazon ECR リソースへのタグ付けのサポート

タスク	AWS CLI	API アクション
1 つ以上のタグを追加、または上書きします。	tag-resource	TagResource
1 つ以上のタグを削除します。	untag-resource	UntagResource

AWS CLI を使用してタグを管理する方法を以下の例に示します。

例 1: 既存のリポジトリにタグ付けする

次のコマンドでは、既存のリポジトリにタグ付けします。

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

例 2: 既存のリポジトリに複数のタグをタグ付けする

次のコマンドでは、既存のリポジトリにタグ付けします。

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

例 3: 既存のリポジトリからタグを削除する

次のコマンドでは、既存のリポジトリからタグを削除します。

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

例 4: リポジトリのタグを一覧表示する

次のコマンドでは、既存のリポジトリに関連付けられているタグを一覧表示します。

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

例 5: リポジトリを作成してタグを適用する

次のコマンドでは、`test-repo` という名前のリポジトリを作成し、キーが `team` で値が `devs` のタグを追加します。

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

イメージ

Amazon Elastic Container Registry (Amazon ECR) では、イメージリポジトリに Docker イメージが保存されます。Docker CLI を使用して、リポジトリからイメージをプッシュおよびプルできます。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリに対して認証されるか、Amazon ECR レポジトリとの間でイメージをプッシュまたはプルするには、事前に IAM ポリシーを通じて `ecr:GetAuthorizationToken` API へのアクセス権限がユーザーに許可されている必要があります。Amazon ECR は、さまざまなレベルで複数の管理 IAM ポリシーを提供します。詳細については、「Amazon ECR 管理ポリシー (p. 50)」を参照してください。

トピック

イメージのプッシュ

開発環境に利用可能な Docker イメージがある場合、docker push コマンドを使用して Amazon ECR リポジトリにプッシュできます。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリに対して認証されるか、Amazon ECR レポジトリとの間でイメージをプッシュまたはプルするには、事前に IAM ポリシーを通じて `ecr:GetAuthorizationToken` API へのアクセス権限がユーザーに許可されている必要があります。Amazon ECR は、さまざまなレベルで複数の管理 IAM ポリシーを提供します。詳細については、「Amazon ECR 管理ポリシー (p. 50)」を参照してください。

Docker イメージを Amazon ECR リポジトリにプッシュするには

1. イメージのプッシュ先となる Amazon ECR レジストリに対して Docker クライアントを認証します。認証トークンは、使用するレジストリごとに取得する必要があり、トークンは 12 時間有効です。詳細については、「レジストリの認証 (p. 12)」を参照してください。
2. プッシュ先となるリポジトリにイメージリポジトリが存在しない場合は、作成します。詳細については、「リポジトリの作成 (p. 15)」を参照してください。
3. プッシュするイメージを識別します。docker images コマンドを実行し、システム上のイメージを一覧表示します。

```
docker images
```

イメージは、結果のコマンド出力で *repository:tag* の値またはイメージ ID によって識別できます。

4. 使用する Amazon ECR レジストリ、リポジトリ、オプションのイメージタグ名の組み合わせによってイメージをタグ付けします。レジストリ形式は *aws_account_id*.dkr.ecr.*region*.amazonaws.com です。リポジトリ名は、イメージ用に作成したリポジトリと一致する必要があります。イメージタグを省略した場合、タグは `latest` と見なされます。

   次の例では、イメージが *aws_account_id*.dkr.ecr.*region*.amazonaws.com/my-web-app として ID *e9ae3c220b23* でタグ付けされます。

   ```
   docker tag e9ae3c220b23 aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app
   ```

5. docker push コマンドを使用してイメージをプッシュします。

   ```
   docker push aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app
   ```

6. (オプション) 任意の追加のタグをイメージに適用し、「Step 4 (p. 27)」および「Step 5 (p. 27)」を繰り返して、それらのタグを Amazon ECR にプッシュします。Amazon ECR では、イメージあたり最大 100 個のタグを適用できます。

AWS CLI を使用してイメージにもう一度タグを付ける

Docker Image Manifest V2 Schema 2 のイメージでは、put-image コマンドの `--image-tag` オプションを使用して、既存のイメージにもう一度タグを付けることができます。Docker でイメージをプルまたはプッシュしなくても、もう一度タグを付けることができます。大きなイメージの場合、このプロセスにより、イメージにもう一度タグを付けるために必要なネットワーク帯域幅と時間がかなり節約されます。

> Note
>
> この手順は、AWS CLI の出力テキストがシェルによって解釈される方法が原因で、Windows クライアントでは機能しません。Windows クライアントのイメージにもう一度タグを付ける方法については、「AWS Tools for Windows PowerShell を使用してイメージにもう一度タグを付ける (p. 28)」を参照してください。

AWS CLI を使用してイメージにもう一度タグを付けるには

1. batch-get-image コマンドを使用して、イメージを再タグ付けして環境変数に書き込むためのイメージマニフェストを取得します。この例では、レポジトリ *amazonlinux* でタグ *latest* を付けたイメージのマニフェストが、環境変数 *MANIFEST* に書き込まれます。

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

2. put-image コマンドの `--image-tag` オプションを使用して、イメージマニフェストを新しいタグで Amazon ECR に配置します。この例では、イメージには *2017.03* というタグが付きます。

   > Note
   >
   > 使用している AWS CLI のバージョンで `--image-tag` オプションが使用できない場合は、最新バージョンにアップグレードします。詳細については、『AWS Command Line Interface ユーザーガイド』の「AWS コマンドラインインターフェイスのインストール」を参照してください。

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

3. 新しいイメージタグがイメージにアタッチされていることを確認します。出力ウィンドウで、イメージにはタグ `latest` および `2017.03` が付いています。

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

出力:

```
{
    "imageDetails": [
        {
            "imageSizeInBytes": 98755613,
            "imageDigest":
 "sha256:8d00af8f076eb15a33019c2a3e7f1f655375681c4e5be157a2685dfe6f247227",
            "imageTags": [
                "latest",
                "2017.03"
            ],
            "registryId": "aws_account_id",
            "repositoryName": "amazonlinux",
            "imagePushedAt": 1499287667.0
        }
    ]
}
```

AWS Tools for Windows PowerShell を使用してイメージにもう一度タグを付ける

Docker Image Manifest V2 Schema 2 のイメージでは、AWS Tools for Windows PowerShell Get-ECRImage コマンドレットの `-ImageTag` オプションを使用して、既存のイメージにもう一度タグを付けることができます。Docker でイメージをプルまたはプッシュしなくても、もう一度タグを付けることができます。大きなイメージの場合、このプロセスにより、イメージにもう一度タグを付けるために必要なネットワーク帯域幅と時間がかなり節約されます。

AWS Tools for Windows PowerShell を使用してイメージにもう一度タグを付けるには

1. Get-ECRImageBatch コマンドレットを使用して、もう一度タグを付けるイメージの説明を取得し、環境変数にそれを書き込みます。この例では、レポジトリ *amazonlinux* でタグ *latest* を付けたイメージが、環境変数 *$Image* に書き込まれます。

   Note

   システムで Get-ECRImageBatch コマンドレットを使用できない場合は、AWS Tools for Windows PowerShell ユーザーガイド の「AWS Tools for Windows PowerShell のセットアップ」を参照してください。

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

2. *$Manifest* 環境変数にイメージのマニフェストを書き込みます。

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

3. Write-ECRImage コマンドレットの `-ImageTag` オプションを使用して、イメージマニフェストを新しいタグで Amazon ECR に配置します。この例では、イメージには *2017.09* というタグが付きます。

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

4. 新しいイメージタグがイメージにアタッチされていることを確認します。出力ウィンドウで、イメージにはタグ `latest` および `2017.09` が付いています。

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

出力:

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

イメージのプル

Amazon ECR で利用可能な Docker イメージを実行する場合、docker pull コマンドを使用してローカル環境にプルします。これはデフォルトのレジストリまたは他の AWS アカウントに関連付けられたレジストリから行うことができます。Amazon ECS タスク定義で Amazon ECR イメージを使用する場合は、「Amazon ECR での Amazon ECS イメージの使用 (p. 31)」を参照してください。

> **Important**
>
> Amazon ECR では、ユーザーがレジストリに対して認証されるか、Amazon ECR レポジトリとの間でイメージをプッシュまたはプルするには、事前に IAM ポリシーを通じて `ecr:GetAuthorizationToken` API へのアクセス権限がユーザーに許可されている必要があります。Amazon ECR は、さまざまなレベルで複数の管理 IAM ポリシーを提供します。詳細については、「Amazon ECR 管理ポリシー (p. 50)」を参照してください。

Amazon ECR リポジトリから Docker イメージをプルするには

1. イメージのプル元になる Amazon ECR レジストリに対して Docker クライアントを認証します。認証トークンは、使用するレジストリごとに取得する必要があり、トークンは 12 時間有効です。詳細については、「レジストリの認証 (p. 12)」を参照してください。
2. (オプション) プルするイメージを識別します。
   - レジストリ内のリポジトリは、aws ecr describe-repositories コマンドを使用してリストできます。

     ```
     aws ecr describe-repositories
     ```

     上のサンプルレジストリには、`amazonlinux` というリポジトリがあります。
   - リポジトリ内のイメージは、aws ecr describe-images コマンドで記述することができます。

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

     上記のリポジトリ例には、`latest` および `2016.09` というタグが付けられたイメージが、イメージダイジェスト `sha256:f1d4ae3f7261a72e98c6ebefe9985cf10a0ea5bd762585a43e0700ed99863807` とともにあります。

3. docker pull コマンドを使用してイメージをプルします。イメージ名の形式は、タグを使用してプルする場合は *registry*/*repository*[:*tag*]、ダイジェストを使用してプルする場合は *registry*/*repository*[@*digest*] とします。

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

Important

repository-url not found: does not exist or no pull access エラーが表示された場合は、Amazon ECR で Docker クライアントを認証する必要があります。詳細については、「レジストリの認証 (p. 12)」を参照してください。

コンテナイメージマニフェストの形式

Amazon ECR は次のコンテナイメージマニフェスト形式をサポートします。

- Docker Image Manifest V2 Schema 1 (Docker バージョン 1.9 以前で使用)
- Docker Image Manifest V2 Schema 2 (Docker バージョン 1.10 以降で使用)
- Open Container Initiative (OCI) 仕様 (v1.0 以降)

Docker Image Manifest V2 Schema 2 のサポートでは、次の機能が提供されます。

- イメージごとに複数のタグを使用する機能。
- Windows コンテナイメージを保存するためのサポート。詳細については、Amazon Elastic Container Service Developer Guide の「Amazon ECR に Windows イメージをプッシュする」を参照してください。

Amazon ECR イメージマニフェストの変換

Amazon ECR との間でイメージをプッシュまたはプルする場合、コンテナエンジンクライアント (Docker など) はレジストリと通信して、クライアントとレジストリで理解される、イメージで使用するマニフェスト形式について合意します。

Docker バージョン 1.9 以前で Amazon ECR にイメージをプッシュする場合、イメージマニフェストは Docker Image Manifest V2 Schema 1 形式で保存されます。Docker バージョン 1.10 以降で Amazon ECR にイメージをプッシュする場合、イメージマニフェストは Docker Image Manifest V2 Schema 2 形式で保存されます。

タグを使用して Amazon ECR からイメージをプルすると、Amazon ECR はリポジトリに保存されているイメージマニフェスト形式を返します。その形式が返されるのは、その形式がクライアントによって理解される場合のみです。保存されているイメージマニフェスト形式がクライアントによって理解されない場合、Amazon ECR はイメージマニフェストを、クライアントによって理解される形式に変換します。たとえば、Docker 1.9 クライアントが Docker Image Manifest V2 Schema 2 で保存されているイメージマニフェストをリクエストすると、Amazon ECR は Docker Image Manifest V2 Schema 1 形式でマニフェストを返します。以下の表は、イメージがタグを使用してプルされた場合に Amazon ECR でサポートされる、利用可能な変換を示しています。

クライアントによってリクエストされたスキーマ	V2、スキーマ 1 形式で ECR にプッシュされる	V2、スキーマ 2 形式で ECR にプッシュされる	OCI 形式で ECR にプッシュされる
V2、スキーマ 1	変換は必要ありません	V2、スキーマ 1 に変換される	V2、スキーマ 1 に変換される

クライアントによってリクエストされたスキーマ	V2、スキーマ 1 形式で ECR にプッシュされる	V2、スキーマ 2 形式で ECR にプッシュされる	OCI 形式で ECR にプッシュされる
V2、スキーマ 2	利用可能な変換はなく、クライアントは V2、スキーマ 1 にフォールバックする	変換は必要ありません	V2、スキーマ 2 に変換される
OCI	変換は利用できません	OCI に変換される	変換は必要ありません

Important

ダイジェストを使用してイメージをプルする場合、利用できる変換はありません。クライアントは、Amazon ECR に保存されているイメージマニフェスト形式を理解する必要があります。Docker 1.9 以前のクライアントで、Docker Image Manifest V2 Schema 2 イメージをダイジェストを使用してリクエストする場合、イメージのプルは失敗します。詳細については、Docker ドキュメントの「Registry compatibility」を参照してください。
この例では、タグを使用して同じイメージをリクエストした場合、Amazon ECR はクライアントが理解できる形式にイメージマニフェストを変換します。イメージのプルが成功します。

Amazon ECR での Amazon ECS イメージの使用

Amazon ECS で ECR イメージを使用できます。ただし、いくつかの前提条件を満たす必要があります。

- コンテナインスタンスでは、バージョン 1.7.0 以上の Amazon ECS コンテナエージェントを使用する必要があります。最新バージョンの Amazon ECS 最適化 AMI では、タスク定義の ECR イメージがサポートされています。最新の Amazon ECS 最適化 AMI の ID などの詳細については、Amazon Elastic Container Service Developer Guide の「Amazon ECS コンテナエージェントのバージョン」を参照してください。
- コンテナインスタンスで使用する Amazon ECS コンテナインスタンスロール (`ecsInstanceRole`) には、Amazon ECR に対する以下の IAM ポリシーアクセス権限が必要です。

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

コンテナインスタンスに `AmazonEC2ContainerServiceforEC2Role` 管理ポリシーを使用すると、ロールに適切なアクセス権限が付与されます。ロールで Amazon ECR がサポートされていることを確認するには、Amazon Elastic Container Service Developer Guide の「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。

- ECS タスク定義で、ECR イメージに完全な `registry/repository:tag` 名前付けを使用していることを確認してください。たとえば、*`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`:`*`latest`* と指定します。

イメージの削除

イメージの使用が終わったら、リポジトリから削除できます。イメージは、AWS マネジメントコンソールまたは AWS CLI を使用して削除できます。

> Note
>
> リポジトリを使い終えた場合、リポジトリ全体とその中のすべてのイメージを削除できます。詳細については、「リポジトリの削除 (p. 17)」を参照してください。

AWS マネジメントコンソール を使用してイメージを削除するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/repositories) を開きます。
2. ナビゲーションバーから、削除するイメージを含むリージョンを選択します。
3. ナビゲーションペインで、[Repositories] を選択します。
4. [Repositories] ページで、削除するイメージを含むリポジトリを選択します。
5. [Repositories: ***repository_name***] ページで、削除するイメージの左側にあるボックスを選択し、[削除] を選択します。
6. [Delete image(s)] ダイアログボックスで、選択したイメージを削除することを確認し、[Delete] を選択します。

AWS CLI を使用してイメージを削除するには

1. リポジトリ内のイメージを一覧表示し、イメージタグまたはダイジェストを使用して識別できるようにします。

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

2. (オプション) 削除するイメージのタグを指定して、イメージの不要なタグを削除します。

> Note
>
> イメージの最後のタグを削除すると、イメージは削除されます。

```
aws ecr batch-delete-image --repository-name my-repo --image-ids imageTag=latest
```

3. 削除するイメージのダイジェストを指定することで、イメージを削除します。

> Note
>
> ダイジェストを参照してイメージを削除する場合、イメージとそのすべてのタグは削除されます。

```
aws ecr batch-delete-image --repository-name my-repo --image-ids
 imageDigest=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304c7c2c1a9d6fa3e9de6bf552d
```

Amazon Linux コンテナイメージ

Amazon Linux コンテナイメージは、Amazon Linux AMI に含まれているのと同じソフトウェアコンポーネントから構築されます。これは、Docker ワークロードのベースイメージとして任意の環境で使用できます。Amazon EC2 のアプリケーション用にすでに Amazon Linux AMI を使用している場合、Amazon Linux コンテナイメージで簡単にアプリケーションをコンテナ化できます。

ローカル開発環境で Amazon Linux コンテナイメージを使用し、Amazon ECS を使ってアプリケーションを AWS クラウドにプッシュできます。詳細については、「Amazon ECR での Amazon ECS イメージの使用 (p. 31)」を参照してください。

Amazon Linux コンテナイメージは Amazon ECR の および Docker Hub で使用できます。Amazon Linux コンテナイメージのサポートは、AWS 開発者フォーラムにアクセスして見つけることができます。

Amazon ECR から Amazon Linux コンテナイメージをプルするには

1. Amazon Linux コンテナイメージの Amazon ECR レジストリに対して Docker クライアントを認証します。認証トークンは 12 時間有効です。詳細については、「レジストリの認証 (p. 12)」を参照してください。イメージのプル元になるリージョンを指定します (不明な場合は、次のコマンドで使用されている `us-west-2` リージョンを指定できます)。次のコマンドで `us-west-2` リージョンを使用しない場合は、後続のコマンドおよびイメージタグで必ずリージョンを変更してください。

   Note

   get-login コマンドは AWS CLI のバージョン 1.9.15 以降で利用できます。ただし、Docker の最新バージョン (17.06 以降) を使用している場合は、バージョン 1.11.91 以降をお勧めします。詳細については、AWS Command Line Interface ユーザーガイド の「AWS コマンドラインインターフェイスのインストール」を参照してください。

   ```
   aws ecr get-login --region us-west-2 --registry-ids 137112412989 --no-include-email
   ```

   出力例:

   ```
   docker login -u AWS -p password https://137112412989.dkr.ecr.us-west-2.amazonaws.com
   ```

   Important

   `Unknown options: --no-include-email` エラーが返された場合は、AWS CLI の最新バージョンをインストールします。詳細については、AWS Command Line Interface ユーザーガイド の「AWS コマンドラインインターフェイスのインストール」を参照してください。

   結果として得られる出力は、Amazon Linux コンテナイメージ Amazon ECR レジストリに対して Docker クライアントを認証するために使用する、docker login コマンドとなります。
2. Docker CLI をレジストリに対して認証するには、docker login コマンドをコピーしてターミナルに貼り付けます。

   Important

   docker login コマンドを実行すると、システムの他のユーザーに対して、プロセスリスト (ps -e) 表示でコマンド文字列が表示されます。docker login コマンドには認証情報が含まれているため、システムの他のユーザーがこのようにして認証情報を表示するリスクがあります。また、その認証情報を使用して、リポジトリへのプッシュアクセスおよびプルアクセスを取得する可能性があります。安全なシステムを使用していない場合は、このリスクを考慮してください。`-p password` オプションを省略してインタラクティブにログインし、求められたときにパスワードを入力することを検討してください。
3. (オプション) Amazon Linux リポジトリ内のイメージは、aws ecr list-images コマンドでリストすることができます。`latest` タグは、使用可能な最新の Amazon Linux コンテナイメージに常に対応します。

   ```
   aws ecr list-images --region us-west-2 --registry-id 137112412989 --repository-name
    amazonlinux
   ```
4. docker pull コマンドを使用して Amazon Linux コンテナイメージをプルします。

```
docker pull 137112412989.dkr.ecr.us-west-2.amazonaws.com/amazonlinux:latest
```

5. (オプション) コンテナをローカルに実行します。

```
docker run -it 137112412989.dkr.ecr.us-west-2.amazonaws.com/amazonlinux:latest /bin/
bash
```

Docker Hub から Amazon Linux コンテナイメージをプルするには

1. docker pull コマンドを使用して Amazon Linux コンテナイメージをプルします。

```
docker pull amazonlinux
```

2. (オプション) コンテナをローカルに実行します。

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

Amazon ECR ライフサイクルポリシー

Amazon ECR ライフサイクルポリシーで、リポジトリ内のイメージのライフサイクル管理を指定することができます。ライフサイクルポリシーは 1 つまたは複数のルールのセットであり、各ルールでは Amazon ECR へのアクションが定義されています。アクションは、指定された文字列がプレフィックスとして付いているタグを含むイメージに適用されます。これにより、未使用のイメージ、たとえば、経過時間またはカウントに基づく有効期限切れイメージなどのクリーンアップを自動化できます。ライフサイクルポリシーの作成後、影響を受けるイメージは 24 時間以内に有効期限切れになります。

トピック

ライフサイクルポリシーのテンプレート

ライフサイクルポリシーのコンテンツは、リポジトリに関連付けられる前に評価されます。以下に示しているのは、ライフサイクルポリシーの JSON 構文テンプレートです。ライフサイクルポリシーの例については、「ライフサイクルポリシーの例 (p. 39)」を参照してください。

```
{
    "rules": [
        {
            "rulePriority": integer,
            "description": "string",
            "selection": {
                "tagStatus": "tagged"|"untagged"|"any",
                "tagPrefixList": list<string>,
                "countType": "imageCountMoreThan"|"sinceImagePushed",
```

```
                "countUnit": "string",
                "countNumber": integer
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

Note

tagPrefixList パラメータは、tagStatus が tagged の場合のみ使用されます。countUnit パラメータは、countType が sinceImagePushed の場合のみ使用されます。countNumber パラメータは、countType が imageCountMoreThan に設定されている場合のみ使用されます。

ライフサイクルポリシーのパラメータ

ライフサイクルポリシーは、次の部分に分けられます。

トピック

ルールの優先順位

`rulePriority`

タイプ: 整数

必須: はい

ルールが評価される順序を低いものから高いものへと設定します。ライフサイクルポリシーにルールを追加するときは、それぞれに rulePriority の一意の値を付ける必要があります。ポリシー内のルール間で値が連続している必要はありません。any の tagStatus を持つルールは、rulePriority の最大値を持ち、最後に評価される必要があります。

説明

`description`

タイプ: 文字列

必須: いいえ

(オプション) ライフサイクルポリシー内のルールの目的について説明します。

タグステータス

`tagStatus`

タイプ: 文字列

必須: はい

追加するライフサイクルポリシーのルールがイメージのタグを指定するかどうかを決定します。使用できるオプションは、`tagged`、`untagged`、または `any` です。`any` を指定する場合は、すべてのイメージにルールが適用されます。`tagged` を指定する場合は、`tagPrefixList` 値も指定する必要があります。`untagged` を指定する場合は、`tagPrefixList` を省略する必要があります。

タグプレフィックスリスト

`tagPrefixList`

タイプ: list[string]

必須: `tagStatus` が [tagged] に設定されている場合のみ、必須

`"tagStatus": "tagged"` を指定した場合にのみ使用されます。ライフサイクルポリシーでアクションを実行するための、カンマ区切りのイメージタグプレフィックスのリストを指定する必要があります。たとえば、イメージに `prod`、`prod1`、`prod2` というようにタグが付いている場合、すべてを指定するためにタグプレフィックス `prod` を使用します。複数のタグを指定する場合、指定されたすべてのタグが付いているイメージのみが選択されます。

カウントタイプ

`countType`

タイプ: 文字列

必須: はい

イメージに適用するカウントタイプを指定します。`countType` が `imageCountMoreThan` に設定してある場合は、`countNumber` も指定して、リポジトリに存在するイメージ数の制限を設定するルールを作成します。`countType` が `sinceImagePushed` に設定してある場合は、`countUnit` および `countNumber` も指定して、リポジトリに存在するイメージの時間制限を指定します。

カウント単位

`countUnit`

タイプ: 文字列

必須: `countType` が `sinceImagePushed` に設定されている場合のみ、必須

日数を表す `countNumber` に加えて、カウント単位の `days` も時間単位として指定します。これを指定するのは `countType` が `sinceImagePushed` である場合に限ります。`countType` が他の値である場合にカウント単位を指定すると、エラーが発生します。

カウント数

`countNumber`

タイプ: 整数

必須: はい

カウント数を指定します。使用している `countType` が `imageCountMoreThan` である場合、この値はリポジトリに維持するイメージの最大数です。使用している `countType` が `sinceImagePushed` である場合、この値はイメージの最大期限です。

アクション

`type`

タイプ: 文字列

必須: はい

アクションタイプを指定します。サポートされる値は `expire` です。

ライフサイクルポリシーの評価ルール

ライフサイクルポリシーの評価者は、プレーンテキスト JSON を解析し、指定したリポジトリ内のイメージへ適用します。ライフサイクルのポリシーを作成する際は次のルールに注意が必要です。

- イメージは 1 または ゼロのルールで正確に期限切れとなります。
- ルールのタグ付け要件に一致するイメージは、優先度がより低いルールによって期限切れにはなりません。
- 優先度がより高いルールにマークされたイメージをルールがマークすることはありませんが、期限切れになっていないかのように識別できます。
- 一連のルールには、一意の一連のタグプレフィックスを含める必要があります。
- タグが付いていないイメージを選択できるのは 1 つのルールのみです。
- 期限切れは常に `pushed_at_time` の順に並べられ、より古いイメージが新しいものよりも先に期限切れとなります。
- `tagPrefixList` を使用する場合、`tagPrefixList` 値のすべてのタグがイメージのタグのいずれかと一致すると、イメージは正常に一致します。
- `countType = imageCountMoreThan` では、イメージは期間の新しいものから始めて最も古いものへと `pushed_at_time` に基づいて順に並べられた後、指定したカウントより大きいイメージはすべて期限切れとなります。
- `countType = sinceImagePushed` では、`countNumber` に基づき、`pushed_at_time` が指定された日数より古いすべてのイメージは期限切れとなります。

ライフサイクルポリシーのプレビュー

ライフサイクルポリシーのプレビューでは、実行する前にイメージリポジトリのライフサイクルポリシーの影響を確認できます。以下の手順では、ライフサイクルポリシーのプレビューを作成する方法を示します。

コンソールを使用してライフサイクルポリシーのプレビューを作成するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/repositories) を開きます。

2. ナビゲーションバーから、ライフサイクルポリシーのプレビューを実行するリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで [Repositories] を選択し、リポジトリを選択します。
4. [Repositories: ***repository_name***] ページで、ナビゲーションペインの [ライフサイクルポリシー] を選択します。
5. [Repositories: ***repository_name***: ライフサイクルポリシー] ページで、[テストルールの編集]、[ルールの作成] の順に選択します。
6. ライフサイクルポリシーのルールに以下の詳細を入力します。

   a. [ルールの優先順位] で、ルールの優先順位を数字で入力します。
   b. [ルールの説明] で、ライフサイクルポリシーのルールの説明を入力します。
   c. [Image status (イメージステータス)] で、[Tagged (タグ付け)]、[Untagged (タグ付けなし)]、または [Any (すべて)] を選択します。
   d. [イメージのステータス] で `Tagged` を指定した場合は、[タグのプレフィックス] で、ライフサイクルポリシーでアクションを実行するイメージタグのリストをオプションで指定できます。[`Untagged`] を指定した場合は、このフィールドは空にする必要があります。
   e. [一致条件] では、[イメージをプッシュしてから] または [次の数値を超えるイメージ数] の値を選択します (該当する場合)。
7. [Save] を選択します。
8. ステップ 5 ~ 7 を繰り返すことにより、追加のライフサイクルポリシーのルールを作成します。
9. ライフサイクルポリシーのプレビューを実行するには、[テストの保存と実行] を選択します。
10. [イメージがテストライフサイクルルールに一致しました] で、ライフサイクルポリシーのプレビューの影響を確認します。
11. プレビューの結果に満足したら、[Apply as lifecycle policy] を選択して指定したルールでライフサイクルポリシーを作成します。

    Note

    ライフサイクルポリシーの作成後、影響を受けるイメージは 24 時間以内に有効期限切れになります。

ライフサイクルポリシーの作成

ライフサイクルポリシーで、未使用のリポジトリイメージを期限切れにする一連のルールを作成できます。以下の手順では、ライフサイクルポリシーを作成する方法を示します。ライフサイクルポリシーの作成後、影響を受けるイメージは 24 時間以内に有効期限切れになります。

AWS CLI を使用してライフサイクルポリシーを作成するには

1. ライフサイクルポリシーを作成するためのリポジトリの ID を取得します。

```
aws ecr describe-repositories
```

2. ライフサイクルポリシーを作成します。

```
aws ecr put-lifecycle-policy [--registry-id <string>] --repository-name <string> --
policy-text <string>
```

コンソールを使用してライフサイクルポリシーを作成するには

1. Amazon ECR コンソール (https://console.aws.amazon.com/ecr/repositories) を開きます。

2. ナビゲーションバーから、ライフサイクルポリシーを作成するリポジトリを含むリージョンを選択します。
3. ナビゲーションペインで [Repositories] を選択し、リポジトリを選択します。
4. [Repositories: ***repository_name***] ページで、ナビゲーションペインの [ライフサイクルポリシー] を選択します。
5. [Repositories: ***repository_name***: ライフサイクルポリシー] ページで、[ルールの作成] を選択します。
6. ライフサイクルポリシーのルールに以下の詳細を入力します。
   a. [ルールの優先順位] で、ルールの優先順位を数字で入力します。
   b. [ルールの説明] で、ライフサイクルポリシーのルールの説明を入力します。
   c. [Image status (イメージステータス)] で、[Tagged (タグ付け)]、[Untagged (タグ付けなし)]、または [Any (すべて)] を選択します。
   d. [イメージのステータス] で `Tagged` を指定した場合は、[タグのプレフィックス] で、ライフサイクルポリシーでアクションを実行するイメージタグのリストをオプションで指定できます。[`Untagged`] を指定した場合は、このフィールドは空にする必要があります。
   e. [一致条件] では、[イメージをプッシュしてから] または [次の数値を超えるイメージ数] の値を選択します (該当する場合)。
7. [Save] を選択します。

ライフサイクルポリシーの例

次に、ライフサイクルポリシーの例を構文で示します。

トピック

イメージの経過日数によるフィルタリング

次の例で、タグ付けされていない `14` 日以上経ったイメージを期限切れにするポリシーのライフサイクルポリシーの構文を示します。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Expire images older than 14 days",
            "selection": {
                "tagStatus": "untagged",
                "countType": "sinceImagePushed",
                "countUnit": "days",
                "countNumber": 14
            },
            "action": {
                "type": "expire"
            }
        }
    ]
```

```
}
```

イメージ数によるフィルタリング

次の例で、タグ付けされていないイメージを 1 つだけ保持して残りはすべて期限切れにするポリシーのライフサイクルポリシーの構文を示します。

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Keep only one untagged image, expire all others",
            "selection": {
                "tagStatus": "untagged",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

複数のルールでのフィルタリング

以下は、ライフサイクルポリシーで複数のルールを使用する例です。サンプルのリポジトリとライフサイクルポリシーが結果の説明とともに示されています。

例 A

リポジトリのコンテンツ

- Image A, Taglist: ["beta-1", "prod-1"], Pushed: 10 days ago
- Image B, Taglist: ["beta-2", "prod-2"], Pushed: 9 days ago
- Image C, Taglist: ["beta-3"], Pushed: 8 days ago

ライフサイクルポリシーのテキスト

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPrefixList": ["prod"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
                "tagStatus": "tagged",
```

```
                "tagPrefixList": ["beta"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、prod というプレフィックスでタグ付けされたイメージを特定します。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。イメージ A が期限切れとしてマークされます。
- ルール 2 は、beta というプレフィックスでタグ付けされたイメージを特定します。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。イメージ A とイメージ B の両方が期限切れとしてマークされます。ただし、イメージ A はルール 1 ですでに確認されていて、もしイメージ B が期限切れであれば、ルール 1 に違反してしまうので、スキップされます。
- 結果: イメージ A は期限切れです。

例 B

これは前の例と同じリポジトリですが、結果を説明するためにルールの優先順位が変更されています。

リポジトリのコンテンツ

- Image A, Taglist: ["beta-1", "prod-1"], Pushed: 10 days ago
- Image B, Taglist: ["beta-2", "prod-2"], Pushed: 9 days ago
- Image C, Taglist: ["beta-3"], Pushed: 8 days ago

ライフサイクルポリシーのテキスト

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPrefixList": ["beta"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
                "tagStatus": "tagged",
                "tagPrefixList": ["prod"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
```

```
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、beta でタグ付けされたイメージを特定します。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。3 つのすべてのイメージが確認され、イメージ A とイメージ B が期限切れとしてマークされます。
- ルール 2 は、prod でタグ付けされたイメージを特定します。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。今回は、使用可能なイメージはすべてルール 1 で確認済みのため、確認できるイメージがありません。そのため、追加でイメージをマークすることはありません。
- 結果: イメージ A と B は期限切れです。

単一のルールで複数のタグをフィルタリングする

次の例で、単一のルールでの複数のタグプレフィックスのライフサイクルポリシーの構文を指定します。サンプルのリポジトリとライフサイクルポリシーが結果の説明とともに示されています。

例 A

単一のルールで複数のタグプレフィックスが指定されたときは、イメージはすべてのリストされたタグプレフィックスに一致する必要があります。

リポジトリのコンテンツ

- Image A, Taglist: ["alpha-1"], Pushed: 12 days ago
- Image B, Taglist: ["beta-1"], Pushed: 11 days ago
- Image C, Taglist: ["alpha-2", "beta-2"], Pushed: 10 days ago
- Image D, Taglist: ["alpha-3"], Pushed: 4 days ago
- Image E, Taglist: ["beta-3"], Pushed: 3 days ago
- Image F, Taglist: ["alpha-4", "beta-4"], Pushed: 2 days ago

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPrefixList": ["alpha", "beta"],
                "countType": "sinceImagePushed",
                "countNumber": 5,
                "countUnit": "days"
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、`alpha` および `beta` でタグ付けされたイメージを特定します。イメージ C と F が確認されます。5 日より古いイメージをマークするので、イメージ C がマークされます。
- 結果: イメージ C は期限切れです。

例 B

次の例では、タグは排他的ではないことを説明します。

リポジトリのコンテンツ

- Image A, Taglist: ["alpha-1", "beta-1", "gamma-1"], Pushed: 10 days ago
- Image B, Taglist: ["alpha-2", "beta-2"], Pushed: 9 days ago
- Image C, Taglist: ["alpha-3", "beta-3", "gamma-2"], Pushed: 8 days ago

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPrefixList": ["alpha", "beta"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、`alpha` および `beta` でタグ付けされたイメージを特定します。すべてのイメージが確認されます。最も古いイメージから始めて、一致するイメージが 1 つか数個になるまでマークし続けます。イメージ A と B が期限切れとしてマークされます。
- 結果: イメージ A と B は期限切れです。

すべてのイメージでのフィルタリング

次のライフサイクルポリシーの例では、異なるフィルタですべてのイメージを指定します。サンプルのリポジトリとライフサイクルポリシーが結果の説明とともに示されています。

例 A

次に、すべてのルールを適用する一方、イメージを 1 つだけ保持して残りはすべて期限切れにするポリシーのライフサイクルポリシーの構文を示します。

リポジトリのコンテンツ

- Image A, Taglist: ["alpha-1"], Pushed: 4 days ago
- Image B, Taglist: ["beta-1"], Pushed: 3 days ago
- Image C, Taglist: [], Pushed: 2 days ago
- Image D, Taglist: ["alpha-2"], Pushed: 1 day ago

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、すべてのイメージを特定します。イメージ A、B、C、D が確認されます。最も新しいもの以外のすべてのイメージが期限切れとされます。イメージ A、B、C が期限切れとしてマークされます。
- 結果: イメージ A、B、C は期限切れです。

例 B

以下の例は、単一のポリシーのすべてのルールのタイプを組み合わせるライフサイクルポリシーを示しています。

リポジトリのコンテンツ

- Image A, Taglist: ["alpha-", "beta-1", "-1"], Pushed: 4 days ago
- Image B, Taglist: [], Pushed: 3 days ago
- Image C, Taglist: ["alpha-2"], Pushed: 2 days ago
- Image D, Taglist: ["git hash"], Pushed: 1 day ago
- Image E, Taglist: [], Pushed: 1 day ago

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Rule 1",
            "selection": {
                "tagStatus": "tagged",
                "tagPrefixList": ["alpha"],
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 2,
            "description": "Rule 2",
            "selection": {
                "tagStatus": "untagged",
                "countType": "sinceImagePushed",
                "countUnit": "day",
```

```
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 3,
            "description": "Rule 3",
            "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        }
    ]
}
```

このライフサイクルポリシーのロジックは次のようになります。

- ルール 1 は、alpha でタグ付けされたイメージを特定します。イメージ A と B を識別します。最も新しいイメージを保持し、残りは期限切れとしてマークされます。イメージ A が期限切れとしてマークされます。
- ルール 2 は、タグ付けされていないイメージを特定します。イメージ B と E を識別します。有効期限が 1 日より古いすべてのイメージがマークされます。イメージ B が期限切れとしてマークされます。
- ルール 3 は、すべてのイメージを特定します。イメージ A、B、C、D、E を識別します。最も新しいイメージを保持し、残りは期限切れとしてマークされます。ただし、優先度がより高いルールにより識別されるため、イメージ A、B、C をマークすることはできません。イメージ D が期限切れとしてマークされます。
- 結果: イメージ A、B、D は期限切れです。

Amazon ECR IAM のポリシーおよびロール

デフォルトでは、IAM ユーザーには Amazon Elastic Container Registry (Amazon ECR) リソースを作成または変更、または Amazon ECR API を使用するタスクを実行する権限がありません。(つまり、Amazon ECR コンソールまたは AWS CLI を使用して実行することもできません)。 IAM ユーザーがリソースを作成または変更、およびタスクを実行できるようにするには、IAM ポリシーを作成する必要があります。これによって、必要な特定のリソースおよび API オペレーションを使用するためのアクセス許可を IAM ユーザーに付与する必要があります。続いて、それらのアクセス権限が必要な IAM ユーザーまたはグループにそのポリシーをアタッチします。

ポリシーをユーザーまたはユーザーのグループにアタッチする場合、ポリシーによって特定リソースの特定タスクを実行するユーザーの権限が許可または拒否されます。詳細については、IAM ユーザーガイド の「アクセス権限とポリシー」と「IAM ポリシーの管理」を参照してください。

同様に、Amazon ECS コンテナインスタンスはユーザーの代わりに Amazon ECR API を呼び出して、Amazon ECS のタスク定義で使用される Docker イメージを取得する必要があります。ユーザーの認証情報を使用して認証する必要があります。この認証は、コンテナインスタンスの IAM ロールを作成し、コンテナインスタンスの起動時にそのロールをコンテナインスタンスに関連付けることにより行われます。詳細については、Amazon Elastic Container Service Developer Guide の「Amazon ECS コンテナインスタンスの IAM ロール」を参照してください。IAM ロールの詳細については、IAM ユーザーガイドの「IAM ロール」を参照してください。

はじめに

IAM ポリシーは、1 つ以上の Amazon ECR オペレーションを使用するアクセス権限を付与または拒否する必要があります。さらに、このオペレーションで使用できるリソース (すべてのリソースか、場合によっては特定のリソース) も指定する必要があります。このポリシーには、リソースに適用する条件も含めることができます。

Amazon ECR では、リソースレベルのアクセス許可が部分的にサポートされます。これは、一部の Amazon ECS API オペレーションでは、ユーザーがそのオペレーションに使用できるリソースを指定できないことを意味します。代わりに、ユーザーがそのオペレーションにすべてのリソースを使用することを許可する必要があります。

トピック

ポリシーの構造

次のトピックでは、IAM ポリシーの簡単な構造について説明します。

トピック

- ユーザーが必要なアクセス許可を持っているかどうかを確認する (p. 49)

ポリシー構文

IAM ポリシーは 1 つ以上のステートメントで構成される JSON ドキュメントです。各ステートメントは次のように構成されます。

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

ステートメントはさまざまな工レメントで構成されます。

- `Effect`:effect は `Allow` または `Deny` にすることができます。デフォルトでは、IAM ユーザーはリソースおよび API オペレーションを使用するアクセス権限がないため、リクエストはすべて拒否されます。明示的な許可はデフォルトに優先します。明示的な拒否はすべての許可に優先します。
- [`Action`]: アクセス許可を付与または拒否する対象とする、特定の API オペレーションです。詳細については、「Amazon ECR のアクション (p. 47)」を参照してください。
- [`Resource`]: アクションによって影響を受けるリソースです。Amazon ECR API オペレーションの中には、オペレーションによって作成/変更できるリソースをポリシー内で特定できるものもあります。ステートメント内でリソースを指定するには、Amazon リソースネーム (ARN) を使用する必要があります。`arn` 値の指定については、「Amazon ECR 用の Amazon リソースネーム (p. 48)」を参照してください。どの API オペレーションがどの ARN をサポートするかについては、「Amazon ECR API アクションでサポートされるリソースレベルのアクセス許可 (p. 51)」を参照してください。API オペレーションが ARN をサポートしていない場合は、* (アスタリスク) ワイルドカードを使用すると、オペレーションがすべてのリソースに影響するように指定できます。
- [`Condition`]: Condition はオプションであり、ポリシーが有効になるタイミングを制御できます。Amazon ECR の条件を指定する方法については、Amazon ECR の条件キー (p. 49) を参照してください。

Amazon ECR のアクション

IAM ポリシーステートメントで、IAM をサポートするすべてのサービスから任意の API オペレーションを指定できます。Amazon ECR の場合、API オペレーションの名前にプレフィックス `ecr:` を付けます。例: `ecr:CreateRepository` および `ecr:DeleteRepository`。

単一のステートメントに複数のオペレーションを指定するには、次のようにコンマで区切ります。

```
"Action": ["ecr:action1", "ecr:action2"]
```

ワイルドカードを使用して複数のオペレーションを指定することもできます。たとえば、以下のように「Delete」という単語で始まる名前のすべてのオペレーションを指定できます。

```
"Action": "ecr:Delete*"
```

すべての Amazon ECR API オペレーションを指定するには、次のように * (アスタリスク) ワイルドカードを使用します。

```
"Action": "ecr:*"
```

Amazon ECR オペレーションの一覧については、Amazon Elastic Container Registry API Reference のアクションに進んでください。

Amazon ECR 用の Amazon リソースネーム

各 IAM ポリシーステートメントは、ARN を使用して指定したリソースに適用されます。

> **Important**
>
> 現時点では、すべての API アクションが個々の ARN をサポートしているわけではありません。今後、さらに多くの API アクションのサポートと、さらに多くの Amazon ECR リソースの ARN が追加される予定です。どの Amazon ECR API アクションでどの ARN を使用できるかについては、「Amazon ECR API アクションでサポートされるリソースレベルのアクセス許可 (p. 51)」を参照してください。

ARN には以下の一般的な構文があります。

```
arn:aws:[service]:[region]:[account]:resourceType/resourcePath
```

`service`

サービス (例: `ecr`)。

`region`

リソースのリージョン (例: `us-east-1`)。

`account`

ハイフンなしの AWS アカウント ID (例: `123456789012`)。

`resourceType`

リソースの種類 (例: `instance`)。

`resourcePath`

リソースを識別するパス。パスにワイルドカードの * (アスタリスク) が使用できます。

たとえば、以下の要領で ARN を使用して、ステートメント内で特定のリポジトリ (`my-repo`) を指定することができます。

```
"Resource": "arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
```

また、特定のアカウントに属するすべてのリポジトリを指定するには、以下の要領で * ワイルドカードを使用します。

```
"Resource": "arn:aws:ecr:us-east-1:123456789012:repository/*"
```

すべてのリソースを指定する場合、または特定の API オペレーションが ARN をサポートしていない場合は、以下の要領で、`Resource` エレメント内で * ワイルドカードを使用します。

```
"Resource": "*"
```

以下の表では、Amazon ECR API オペレーションによって使用される各リソースの種類の ARN を説明しています。

リソースタイプ	ARN
すべての Amazon ECR リソース	`arn:aws:ecr:*`
特定リージョンの特定アカウントが所有するすべての Amazon ECR リソース	`arn:aws:ecr:region:account:*`
リポジトリ	`arn:aws:ecr:region:account:repository/repository-name`

多くの Amazon ECR API オペレーションは、複数のリソースを受け入れます。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、以下のようにコンマで ARN を区切ります。

```
"Resource": ["arn1", "arn2"]
```

詳細については、アマゾン ウェブ サービス全般のリファレンス の「Amazon リソースネーム (ARN) と AWS サービスの名前空間」を参照してください。

Amazon ECR の条件キー

ポリシーステートメントでは、オプションで有効になるタイミングを制御する条件を指定できます。各条件には 1 つ以上のキーと値のペアが含まれます。条件キーは大文字小文字を区別しません。私たちは AWS 範囲の条件キーに加え、追加のサービス固有の条件キーを定義しました。

複数の条件、または単一の条件に複数のキーを指定する場合、論理 AND 演算を使用してそれらを評価します。1 つのキーに複数の値を使用して単一の条件を指定する場合、論理 OR 演算を使用して条件を評価します。アクセス許可が付与されるには、すべての条件を満たしている必要があります。

条件を指定する際にプレースホルダーも使用できます。詳細については、IAM ユーザーガイド の「ポリシー変数」を参照してください。

Amazon ECR は AWS グローバル条件コンテキストキーを実装します。詳細については、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。

Amazon ECR のリポジトリポリシーステートメントの例については、「Amazon ECR のリポジトリポリシー (p. 17)」を参照してください。

ユーザーが必要なアクセス許可を持っているかどうかを確認する

IAM ポリシーを作成したら、そのポリシーがユーザーに特定の API オペレーションおよび必要なリソースを使用するアクセス権限を付与しているかどうかを確認することをお勧めします。ポリシーを本稼働環境に配置する前に確認します。

まずテスト目的の IAM ユーザーを作成し、作成した IAM ポリシーをテストユーザーにアタッチします。次に、テストユーザーとしてリクエストを作成します。テストリクエストは、コンソールまたは AWS CLI を使用して行うことができます。

> Note
>
> IAM ポリシーシミュレーターを使用してポリシーをテストすることもできます。ポリシーシミュレーターの詳細については、IAM ユーザーガイド の「IAM ポリシーシミュレーターの使用」を参照してください。

テストしているアクションがリソースを作成または変更する場合、DryRun パラメータを使用してリクエストを作成する (または、--dry-run オプションで AWS CLI コマンドを実行する) 必要があります。この場合、発信者は認証チェックを行いますが、操作は完了しません。たとえば、実際に終了させることなく、ユーザーが特定のインスタンスを終了できるかどうかを確認できます。テストユーザーに必要なアクセス許可がある場合、リクエストで DryRunOperation が返されます。必要なアクセス許可がない場合は UnauthorizedOperation が返されます。

ポリシーが想定したアクセス許可をユーザーに付与していない場合、または過度に許可している場合、必要に応じてポリシーを調整できます。必要な結果を得るまで再テストします。

> **Important**
>
> ポリシーの変更が反映され、有効になるには数分間かかります。したがって、ポリシーの更新をテストするには 5 分かかると見ておいてください。

認証チェックが失敗した場合、リクエストでは診断情報でエンコードされたメッセージが返されます。DecodeAuthorizationMessage アクションを使用してメッセージをデコードできます。詳細については、AWS Security Token Service API リファレンスの「DecodeAuthorizationMessage」および AWS CLI Command Referenceの「decode-authorization-message」を参照してください。

Amazon ECR 管理ポリシー

Amazon ECR では、IAM ユーザーや EC2 インスタンスにアタッチして、Amazon ECR リソースや API オペレーションで異なる制御レベルを使用できる複数のマネージドポリシーを提供しています。これらのポリシーを直接適用することも、独自のポリシーを作成する開始点として使用することもできます。これらのポリシーに記載された各 API オペレーションの詳細については、Amazon Elastic Container Registry API Reference の「アクション」を参照してください。

トピック

AmazonEC2ContainerRegistryFullAccess

この管理ポリシーでは、管理者権限による Amazon ECR へのフルアクセスを許可します。

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

AmazonEC2ContainerRegistryPowerUser

この管理ポリシーでは、パワーユーザーによる Amazon ECR へのアクセスを許可して、リポジトリへの読み書きアクセスを許可しますが、リポジトリの削除、適用されるポリシードキュメントの変更は許可しません。

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": [
   "ecr:GetAuthorizationToken",
   "ecr:BatchCheckLayerAvailability",
   "ecr:GetDownloadUrlForLayer",
   "ecr:GetRepositoryPolicy",
   "ecr:DescribeRepositories",
   "ecr:ListImages",
   "ecr:DescribeImages",
   "ecr:BatchGetImage",
   "ecr:InitiateLayerUpload",
   "ecr:UploadLayerPart",
   "ecr:CompleteLayerUpload",
   "ecr:PutImage"
  ],
  "Resource": "*"
 }]
}
```

AmazonEC2ContainerRegistryReadOnly

この管理ポリシーでは、リポジトリとリポジトリ内のイメージの一覧表示や、Docker CLI を使用した Amazon ECR からのイメージのプルなど、Amazon ECR への読み取り専用アクセスを許可します。

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

Amazon ECR API アクションでサポートされるリソースレベルのアクセス許可

リソースレベルのアクセス許可とは、ユーザーがアクションを実行可能なリソースを指定できることを意味します。Amazon ECR では、リソースレベルのアクセス許可が部分的にサポートされます。つまり、特定の Amazon ECR オペレーションについて、それらのオペレーションをいつユーザーが使用できるかを制御することができます。ユーザーのアクセスは、満たす必要がある条件、または特定のリソースに基づいて制御できます。

次の表では、現在リソースレベルのアクセス権限をサポートしている Amazon ECR API オペレーションと、サポートされるリソースと各リソースのリソース ARN について説明しています。

Amazon ECR オペレーションの一覧については、Amazon Elastic Container Registry API Reference のアクションに進んでください。

Important

リソースレベルのアクセス許可がサポートされていない Amazon ECR API オペレーションでは、そのオペレーションを使用するアクセス許可をユーザーに付与できます。ポリシーステートメントのリソース要素に、* (アスタリスク) ワイルドカードを指定する必要があります。

API アクション	リソースタイプ	条件キー
BatchCheckLayerAvailability	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
BatchDeleteImage	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
BatchGetImage	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
CompleteLayerUpload	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
CreateRepository	該当なし	aws:RequestTag
DeleteLifecyclePolicy	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
DeleteRepository	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
DeleteRepositoryPolicy	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
DescribeImages	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
DescribeRepositories	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
GetAuthorizationToken	該当なし	
GetDownloadUrlForLayer	リポジトリ	aws:ResourceTag

API アクション	リソースタイプ	条件キー
	arn:aws:ecr:*region*:*account*:repository/*my-repo*	ecr:ResourceTag
GetLifecyclePolicy	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
GetLifecyclePolicyPreview	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
GetRepositoryPolicy	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
InitiateLayerUpload	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
ListImages	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
ListTagsForResource	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
PutImage	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
PutLifecyclePolicy	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
SetRepositoryPolicy	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
StartLifecyclePolicyPreview	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:ResourceTag ecr:ResourceTag
TagResource	リポジトリ arn:aws:ecr:*region*:*account*:repository/*my-repo*	aws:RequestTag aws:ResourceTag ecr:ResourceTag

API アクション	リソースタイプ	条件キー
`UntagResource`	リポジトリ `arn:aws:ecr:region:account:repository/my-repo`	aws:ResourceTag ecr:ResourceTag
`UploadLayerPart`	リポジトリ `arn:aws:ecr:region:account:repository/my-repo`	aws:ResourceTag ecr:ResourceTag

タグベースのアクセスコントロールを使用する

Amazon ECR CreateRepository API アクションでは、リポジトリ作成時にタグを指定することができます。詳細については、「Amazon ECR リポジトリのタグ付け (p. 22)」を参照してください。

作成時にユーザーがリポジトリにタグ付けするには、そのリソースを作成するアクションを使用するためのアクセス許可が必要です (例: `ecr:CreateRepository`)。タグがリソース作成アクションで指定されている場合、Amazon は `ecr:CreateRepository` アクションで追加の承認を実行してユーザーがタグを作成するアクセス権限を持っているかどうかを確認します。

タグベースのアクセスコントロールは、IAM ポリシーで使用できます。次に例を示します。

次のポリシーでは、IAM ユーザーは、`key=environment,value=dev` としてリポジトリを作成またはタグ付けすることのみ、許可されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateTaggedRepository",
            "Effect": "Allow",
            "Action": [
                "ecr:CreateRepository"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/environment": "dev"
                }
            }
        },
        {
            "Sid": "AllowTagRepository",
            "Effect": "Allow",
            "Action": [
                "ecr:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/environment": "dev"
                }
            }
        }
    ]
}
```

次のポリシーでは、IAM ユーザーは、key=environment,value=prod としてタグ付けされていない限り、すべてのリポジトリにアクセスすることができます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ecr:*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "ecr:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecr:ResourceTag/environment": "prod"
                }
            }
        }
    ]
}
```

Amazon ECR IAM ポリシーの作成

お客様のアカウントのユーザーがアクセスを許可される呼び出しとリソースを制限する IAM ポリシーを作成し、IAM ユーザーにそれらのポリシーをアタッチできます。

ポリシーをユーザーまたはユーザーのグループにアタッチする場合、ポリシーによって特定リソースの特定タスクを実行するユーザーの権限が許可または拒否されます。IAM ポリシーの一般的な情報については、IAM ユーザーガイド の「Permissions and Policies」を参照してください。カスタム IAM ポリシーの管理と作成の詳細については、「IAM ポリシーの管理」を参照してください。

ユーザーの IAM ポリシーを作成するには

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
2. ナビゲーションペインで、[Policies]、[Create Policy] の順に選択します。
3. [ポリシーの作成] セクションで、[独自のポリシーを作成] の横にある [選択] を選択します。
4. [ポリシー名] に、一意の名前 (*AmazonECRUserPolicy* など) を入力します。
5. [Policy Document] に、ユーザーに適用するポリシーを貼り付けます。管理ポリシー (p. 50)を開始点として使用して、より制限が強い、または弱い独自の IAM ポリシーを作成し、Amazon ECR で使用できます。
6. [Create Policy] を選択します。

IAM ポリシーをユーザーにアタッチするには

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
2. ナビゲーションペインで、[ユーザー] を選択してから、ポリシーをアタッチするユーザーを選択します。
3. [Permissions]、[Add permissions] の順に選択します。
4. [Grant permissions] セクションで、[Attach existing policies directly] を選択します。
5. 前の手順で作成したカスタムポリシーを選択し、[Next: Review] を選択します。
6. 詳細を確認し、[アクセス権限の追加] を選択して完了します。

Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink)

インターフェイス VPC エンドポイントを使用するように Amazon ECR を設定することで、VPC のセキュリティ体制を強化できます。インターフェイスエンドポイントは、プライベート IP アドレスを使用して Amazon ECR API にプライベートにアクセスできるテクノロジーである AWS PrivateLink を使用しています。PrivateLink は、VPC および Amazon ECR 間のすべてのネットワークトラフィックを Amazon ネットワークに限定します。また、インターネットゲートウェイ、NAT デバイスあるいは仮想プライベートゲートウェイの必要はありません。Fargate 起動タイプを使用する Amazon ECS タスクの場合、これにより、タスクにパブリック IP アドレスを割り当てることなく、Amazon ECR からプライベートイメージをプルしできます。

PrivateLink および VPC エンドポイントの詳細については、「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。

トピック

Amazon ECR VPC エンドポイントに関する考慮事項

Amazon ECR の VPC エンドポイントを設定する前に、以下の考慮事項に注意してください。

- EC2 起動タイプを使用する Amazon ECS タスクで Amazon ECR からプライベートイメージをプルするには、Amazon ECS のインターフェイス VPC エンドポイントも作成してください。詳細については、Amazon Elastic Container Service Developer Guide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。

  Important

  Fargate 起動タイプを使用する Amazon ECS タスクでは、Amazon ECS インターフェイス VPC エンドポイントは不要です。
- Fargate 起動タイプを使用するタスクでは、この機能を活用するために必要なのは com.amazonaws.***region***.ecr.dkr Amazon ECR VPC エンドポイントおよび Amazon S3 ゲートウェイエンドポイントのみです。
- Fargate を使用したタスクでは、VPC にインターネットゲートウェイがなく、タスクで `awslogs` ログドライバーを使用してログ情報を CloudWatch Logs に送信する場合は、CloudWatch Logs 用のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、『Amazon CloudWatch Logs User Guide』の「VPC エンドポイントでの CloudWatch Logs の使用」を参照してください。
- Amazon ECR からコンテナイメージをプルする Fargate 起動タイプを使用するタスクでは、タスク実行ロールに条件キーを追加することで、タスクが使用する特定の VPC およびサービスが使用する VPC エンドポイントへのアクセスを制限することができます。詳細については、『Amazon Elastic Container Service Developer Guide』の「Amazon ECS タスク実行 IAM ロール」を参照してください。

- 現在、VPC エンドポイントはクロスリージョンリクエストをサポートしていません。Amazon ECR に対して API コールを発行するリージョンと同じリージョンにエンドポイントを作成してください。
- VPC エンドポイントは、Amazon Route 53 を介した Amazon 提供の DNS のみをサポートします。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 VPC 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。
- Amazon S3 ゲートウェイエンドポイントを作成する際、コンテナに既に Amazon S3 に対する接続がある場合は、ゲートウェイが追加される間にその接続が一時的に中断される可能性があります。この中断を回避するには、Amazon S3 ゲートウェイエンドポイントを使用する新しい VPC を作成してから、Amazon ECS クラスターとそのコンテナを新しい VPC に移行します。

Amazon ECR 用の VPC エンドポイントの作成

Amazon ECR サービス用の VPC エンドポイントを作成するには、Amazon VPC ユーザーガイドの「インターフェイスエンドポイントの作成」の手順を使用して、以下に説明するエンドポイントを作成します。

Amazon ECS タスクが EC2 起動タイプを使用している場合、両方のエンドポイントが必要です。作成順序は問いません。タスクが Fargate 起動タイプを使用している場合は、com.amazonaws.*region*.ecr.dkr エンドポイントのみが必要です。

com.amazonaws.*region*.ecr.api

> Note
>
> *#####*は、Amazon ECR によってサポートされている AWS リージョンのリージョン ID (例: 米国東部 (オハイオ) リージョン の場合は `us-east-2`) を表します。

`com.amazonaws.region.ecr.api` エンドポイントが作成されたら、プライベート DNS ホスト名を有効にするかどうかは任意です。プライベート DNS ホスト名を有効にするには、VPC エンドポイントの作成時に VPC コンソールで [プライベート DNS 名を有効にする] オプションが選択されていることを確認します。VPC エンドポイントでプライベート DNS ホスト名を有効にする場合は、SDK または AWS CLI を使用する際にエンドポイント URL を指定しなくてもいいように、SDK または AWS CLI を最新バージョンに更新します。

プライベート DNS ホスト名が有効で SDK または AWS CLI の 2019 年 1 月 24 日以前のバージョンを使用している場合は、`--endpoint-url` パラメータを使用してインターフェイスのエンドポイントを指定する必要があります。次の CLI コマンドの例は、エンドポイント URL の形式を示しています。

```
aws ecr create-repository --repository-name name --endpoint-url https://
api.ecr.region.amazonaws.com
```

VPC エンドポイントでプライベート DNS ホスト名を有効にしない場合は、インターフェイスエンドポイントで VPC エンドポイント ID を指定する `--endpoint-url` パラメータを使用する必要があります。形式を次に示します。

```
aws ecr create-repository --repository-name name --endpoint-url
 https://VPC_endpoint_ID.api.ecr.region.vpce.amazonaws.com
```

com.amazonaws.*region*.ecr.dkr

`com.amazonaws.region.ecr.dkr` エンドポイントを作成する際に、プライベート DNS ホスト名を有効にする必要があります。これを行うには、VPC エンドポイントを作成するときに、VPC コンソールで [プライベート DNS 名を有効にする] オプションが選択されていることを確認します。

Amazon S3 ゲートウェイエンドポイントの作成

Amazon S3 のゲートウェイエンドポイントは、すべての Amazon ECS タスクが Amazon ECR からプライベートイメージをプルするために必要です。Amazon ECR は Amazon S3 を使用して Docker イメージレイヤーを保存するためです。コンテナが Amazon ECR から Docker イメージをダウンロードするときに、Amazon ECR にアクセスしてイメージマニフェストを取得し、Amazon S3 にアクセスして実際のイメージレイヤーをダウンロードする必要があります。以下に示しているのは、各 Docker イメージのレイヤーを含む Amazon S3 バケットの Amazon リソースネーム (ARN) です。

```
arn:aws:s3:::prod-region-starport-layer-bucket/*
```

Amazon ECR サービス用の Amazon S3 ゲートウェイエンドポイントを作成するには、Amazon VPC ユーザーガイドの「ゲートウェイエンドポイントの作成」の手順を使用して、以下のエンドポイントを作成します。エンドポイントを作成するときは、VPC のルートテーブルを選択してください。

com.amazonaws.***region***.s3

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。フルアクセスポリシーを選択することもできます。タスクの IAM ロールまたはその他の IAM ユーザーポリシーに設定された制限はこのポリシーに優越して適用されるためです。Amazon S3 バケットへのアクセスを Amazon ECR で必要な最小限のアクセス許可に制限する場合は、「Amazon ECR の最小 Amazon S3 バケットアクセス許可 (p. 58)」を参照してください。

Amazon ECR の最小 Amazon S3 バケットアクセス許可

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。Amazon ECR に必要な最小限の Amazon S3 バケットへのアクセスを許可するには、そのエンドポイントの IAM ポリシードキュメントを作成するときに、アクセスを Amazon ECR が使用する Amazon S3 バケットに制限します。

次の表は、Amazon ECR が必要とする Amazon S3 バケットポリシーのアクセス許可を示しています。

アクセス権限	説明
`arn:aws:s3:::prod-region-starport-layer-bucket/*`	各 Docker イメージのレイヤーを含む Amazon S3 バケットへのアクセスが提供されます。Amazon ECR でサポートされる AWS リージョンのリージョン識別子を表します (例: 米国東部 (オハイオ) リージョン の場合は `us-east-2`)。

例

以下の例では、Amazon ECR オペレーションに必要な 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。

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
```

```
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::prod-region-starport-layer-bucket/*"]
    }
  ]
}
```

Amazon ECR で AWS CLI を使用する

以下のステップは、AWS Command Line Interface をインストールし、Amazon ECR にログインするために役立ちます。そこから、イメージリポジトリを作成して、そのリポジトリにイメージをプッシュできます。また、Amazon ECR でその他の一般的なシナリオを実行できます。

AWS CLI は、AWS サービスを管理するための統合ツールです。ダウンロードおよび設定用の単一のツールのみを使用して、コマンドラインから複数の AWS サービスを制御し、スクリプトを使用してこれらを自動化することができます。詳細については、「AWS Command Line Interface ユーザーガイド」を参照してください。

さまざまな AWS SDK、IDE ツールキット、および Windows PowerShell コマンドラインツールを含む、AWS リソースを管理するためのその他のツールの詳細については、http://aws.amazon.com/tools/ を参照してください。

トピック

ステップ 1: デフォルトレジストリに対して Docker を認証する

AWS CLI のインストールと設定が完了したら、デフォルトレジストリに対して Docker CLI を認証します。これにより、docker コマンドは Amazon ECR を使用してイメージをプッシュおよびプルできます。AWS CLI には、認証プロセスを簡素化するための get-login コマンドが用意されています。

To authenticate Docker to an Amazon ECR registry with get-login

> Note
>
> The get-login command is available in the AWS CLI starting with version 1.9.15; however, we recommend version 1.11.91 or later for recent versions of Docker (17.06 or later). You can check your AWS CLI version with the aws --version command.

1. Run the aws ecr get-login command. The example below is for the default registry associated with the account making the request. To access other account registries, use the `--registry-ids` *`aws_account_id`* option. For more information, see get-login in the AWS CLI Command Reference.

```
aws ecr get-login --region region --no-include-email
```

Output:

```
```

```
docker login -u AWS -p password https://aws_account_id.dkr.ecr.us-east-1.amazonaws.com
```

Important

If you receive an `Unknown options: --no-include-email` error, install the latest version of the AWS CLI. For more information, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.

The resulting output is a docker login command that you use to authenticate your Docker client to your Amazon ECR registry.

2. Copy and paste the docker login command into a terminal to authenticate your Docker CLI to the registry. This command provides an authorization token that is valid for the specified registry for 12 hours.

Note

If you are using Windows PowerShell, copying and pasting long strings like this does not work. Use the following command instead.

```
Invoke-Expression -Command (aws ecr get-login --no-include-email)
```

Important

When you execute this docker login command, the command string can be visible to other users on your system in a process list (ps -e) display. Because the docker login command contains authentication credentials, there is a risk that other users on your system could view them this way. They could use the credentials to gain push and pull access to your repositories. If you are not on a secure system, you should consider this risk and log in interactively by omitting the `-p password` option, and then entering the password when prompted.

ステップ 2: Docker イメージを取得する

イメージを Amazon ECR にプッシュする前に、プッシュするイメージが必要です。使用するイメージがない場合は、「Amazon ECR における Docker の基本 (p. 5)」の手順を実行することで作成できます。または、Amazon ECR レジストリで必要な Docker Hub からイメージをプルします。Docker Hub から ローカルシステムに `ubuntu:trusty` イメージをプルするには、次のコマンドを実行します。

```
$ docker pull ubuntu:trusty
trusty: Pulling from library/ubuntu
0a85502c06c9: Pull complete
0998bf8fb9e9: Pull complete
a6785352b25c: Pull complete
e9ae3c220b23: Pull complete
Digest: sha256:3cb273da02362a6e667b54f6cf907edd5255c706f9de279c97cfccc7c6988124
Status: Downloaded newer image for ubuntu:trusty
```

ステップ 3: レポジトリを作成する

これで Amazon ECR にプッシュするイメージが用意できたので、それを保持するレポジトリを作成する必要があります。この例では、`ubuntu:trusty` イメージを後でプッシュする、`ubuntu` と呼ばれるレポジトリを作成します。レポジトリを作成するには、次のコマンドを実行します。

```
$ aws ecr create-repository --repository-name ubuntu
```

```
{
    "repository": {
        "registryId": "111122223333",
        "repositoryName": "ubuntu",
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/ubuntu"
    }
}
```

ステップ 4: イメージを Amazon ECR にプッシュする

ここで、前のセクションで作成した Amazon ECR リポジトリにイメージをプッシュできます。docker CLI を使ってイメージをプッシュしますが、この作業を正しく完了するために満たす必要がある前提条件がいくつかあります。

- docker の最小バージョン 1.7 がインストールされている。
- Amazon ECR 認証トークンが docker login で設定されている。
- Amazon ECR リポジトリが存在し、リポジトリにプッシュするアクセス権がユーザーにある。

これらの前提条件が満たされたら、アカウントのデフォルトレジストリで新しく作成されたレポジトリにイメージをプッシュできます。

イメージにタグを付け、Amazon ECR にプッシュするには

1. ローカルに保存したイメージをリストし、タグを付けてプッシュするイメージを識別します。

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL
 SIZE
ubuntu              trusty              e9ae3c220b23        3 weeks ago         187.9
 MB
```

2. リポジトリにプッシュするイメージにタグを付けます。

```
$ docker tag ubuntu:trusty aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu:trusty
```

3. イメージをプッシュします。

```
$ docker push aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu:trusty
The push refers to a repository [aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu]
 (len: 1)
e9ae3c220b23: Pushed
a6785352b25c: Pushed
0998bf8fb9e9: Pushed
0a85502c06c9: Pushed
trusty: digest: sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636b95aed25f52c89b
 size: 6774
```

ステップ 5: Amazon ECR からイメージをプルする

イメージが Amazon ECR リポジトリにプッシュされたら、他の場所からプルできます。docker CLI を使ってイメージをプルしますが、この作業を正しく完了するために満たす必要がある前提条件がいくつかあります。

- docker の最小バージョン 1.7 がインストールされている。
- Amazon ECR 認証トークンが docker login で設定されている。
- Amazon ECR リポジトリが存在し、リポジトリからプルするアクセス権がユーザーにある。

これらの前提条件が満たされたら、イメージをプルできます。Amazon ECR からイメージの例をプルするには、次のコマンドを実行します。

```
$ docker pull aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu:trusty
trusty: Pulling from ubuntu
0a85502c06c9: Pull complete
0998bf8fb9e9: Pull complete
a6785352b25c: Pull complete
e9ae3c220b23: Pull complete
Digest: sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636b95aed25f52c89b
Status: Downloaded newer image for aws_account_id.dkr.ecr.us-east-1.amazonaws.com/
ubuntu:trusty
```

ステップ 6: イメージを削除する

いずれかのリポジトリで、イメージが必要なくなったと判断した場合、batch-delete-image コマンドで削除できます。イメージを削除するには、イメージがあるレポジトリ、およびイメージの `imageTag` または `imageDigest` の値を指定する必要があります。次の例では、イメージタグが `ubuntu` で、`trusty` リポジトリにあるイメージを削除します。

```
$ aws ecr batch-delete-image --repository-name ubuntu --image-ids imageTag=trusty
{
    "failures": [],
    "imageIds": [
        {
            "imageTag": "trusty",
            "imageDigest":
 "sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636b95aed25f52c89b"
        }
    ]
}
```

ステップ 7: リポジトリを削除する

イメージのリポジトリ全体が必要ないと判断した場合は、リポジトリを削除できます。デフォルトでは、イメージを含むリポジトリを削除することはできません。ただし、`--force` フラグを使用すると、この操作を行うことができます。イメージ (およびその中に含まれるすべてのイメージ) を含むリポジトリを削除するには、次のコマンドを実行します。

```
 $ aws ecr delete-repository --repository-name ubuntu --force
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "ubuntu",
        "repositoryArn": "arn:aws:ecr:us-east-1:aws_account_id:repository/ubuntu",
        "createdAt": 1457671643.0,
        "repositoryUri": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu"
    }
}
```

Amazon ECR サービス制限

次の表は、変更できる AWS アカウントの Amazon Elastic Container Registry (Amazon ECR) のデフォルト制限を示しています。詳細については、アマゾン ウェブ サービス全般のリファレンスの「AWS サービスの制限」を参照してください。

リソース	デフォルトの制限
リージョンあたりのレポジトリの最大数	1,000
リポジトリあたりのイメージの最大数	1,000
1 アカウント、1 リージョン、1 秒あたりの `GetAuthorizationToken` API トランザクション数	20 が維持され、200 までのバーストが可能 *
1 アカウント、1 リージョン、1 秒あたりのリポジトリに対する `docker pull` トランザクション数	200 が維持され、400 までのバーストが可能 *
1 アカウント、1 リージョン、1 秒あたりのリポジトリに対する `docker pull layer` トランザクション数	200 が維持され、400 までのバーストが可能 *
1 アカウント、1 リージョン、1 秒あたりのリポジトリに対する `docker push` トランザクション数	10 が維持され、40 までのバーストが可能 *

* 各リージョンで、各アカウントには、トランザクションに応じて、特定数までのクレジットを保存できるバケットが割り当てられます。1 秒あたり一定の割合で補充されます。たとえば、`GetAuthorizationToken` API トランザクションでは、バケットは最大 200 クレジットを保存できます。1 秒あたり 200 `GetAuthorizationToken` API トランザクションに達すると、1 秒あたり 20 トランザクションが無期限に維持されます。

次の表は、変更できない Amazon ECR および Docker イメージの他の制限を示しています。

> **Note**
>
> 以下の表に示されているレイヤーパート情報は、Amazon ECR API アクションを直接呼び出し、イメージプッシュオペレーションのマルチパートアップロードを開始しているお客様にのみ適用されます。これはまれであると想定されます。
> docker CLI を使用してイメージをプル、タグ付け、プッシュすることをお勧めします。

リソース	デフォルトの制限
イメージごとのレイヤーの最大数	127 (これは、現在の Docker の制限です)
イメージごとのタグの最大数	100
レイヤーの最大サイズ **	10,000 MiB
レイヤーパートの最大サイズ	10 MiB
レイヤーパートの最小サイズ	5 MiB (アップロードの最終レイヤーパートを除く)
レイヤーパートの最大数	1,000

リソース	デフォルトの制限
ライフサイクルポリシーのルールの最大数	50
ライフサイクルポリシーの最大長	30720

** ここに示されている最大レイヤーサイズは、レイヤーパートの最大サイズ (10 MiB) をレイヤーパートの最大数 (1,000) で乗算することにより計算されます。

Amazon ECR 使用状況レポート

AWS は、Amazon ECR リソースのコストおよび使用量を分析できる、Cost Explorer と呼ばれる無料のレポートツールを提供します。

Cost Explorer は、使用量とコストの表を表示するために使用できる無料のツールです。過去 13 か月からデータを表示でき、また次の 3 か月間にどのくらい使用する可能性があるかを予測します。時間の経過に伴う AWS リソースの使用量パターンを確認して、さらに照会が必要な分野を識別し、コストの把握に役立つ傾向を確認するには、Cost Explorer を使用します。データの時間範囲を指定したり、時間データを日または月ごとに表示することもできます。

コストと使用状況レポートの計測データには、すべての Amazon ECR リポジトリの使用量が示されます。計測データには、各リポジトリの が含まれます。リポジトリでは、コストとそれらを関連付けていませんが、。リポジトリを識別するには、メタデータタグを使用します。詳細については、「請求用のリソースにタグを付ける (p. 24)」を参照してください。

AWS のコストと使用状況レポートの作成の詳細については、AWS Billing and Cost Management ユーザーガイド の「AWS のコストと使用状況レポート」を参照してください。

AWS CloudTrail を使用した Amazon ECR API コールのログ作成

Amazon ECR は AWS CloudTrail と統合されています。このサービスは、Amazon ECR 内でユーザーやロール、または AWS のサービスによって実行されたアクションを記録するサービスです。CloudTrail は、Amazon ECR コンソールからのコールや、Amazon ECR API へのコード呼び出しを含む、Amazon ECR 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。証跡を作成する場合は、Amazon ECR のイベントなど、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。証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)] で最新のイベントを表示できます。この情報を使用して、Amazon ECR に対して実行されたリクエスト、リクエスト元の IP アドレス、リクエストの実行者、実行日時などの詳細を確認できます。

詳細については、AWS CloudTrail User Guideを参照してください。

CloudTrail 内の Amazon ECR 情報

CloudTrail は、アカウント作成時に AWS アカウントで有効になります。Amazon ECR でアクティビティが発生すると、そのアクティビティは [Event history] の AWS の他のサービスのイベントとともに CloudTrail イベントに記録されます。最近のイベントは、AWS アカウントで表示、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。

AWS のイベントなど、Amazon ECR 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。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべてのリージョンに適用されます。証跡では、AWS パーティションのすべてのリージョンからのイベントがログに記録され、指定した Amazon S3 バケットにログファイルが配信されます。さらに、より詳細な分析と CloudTrail ログで収集されたデータに基づいた行動のためにその他の AWS サービスを設定できます。詳細については、以下のトピックを参照してください。

- 証跡を作成するための概要
- CloudTrail でサポートされるサービスと統合
- CloudTrail の Amazon SNS 通知の設定
- 「複数のリージョンから CloudTrail ログファイルを受け取る」と「複数のアカウントから CloudTrail ログファイルを受け取る」

Amazon ECR アクションはすべて CloudTrail によって記録されます。また、これらのアクションは Amazon Elastic Container Registry API Referenceで説明されています。一般的なタスクを実行すると、タスクの一部であった API アクションごとに、CloudTrail ログファイルにセクションが生成されます。たとえば、レポジトリを作成すると、CloudTrail ログファイルに `GetAuthorizationToken`、`CreateRepository`、および `SetRepositoryPolicy` セクションが生成されます。レポジトリにイメージをプッシュすると、`InitiateLayerUpload`、`UploadLayerPart`、`CompleteLayerUpload`、および `PutImage` セクションが生成されます。イメージをプルすると、`GetDownloadUrlForLayer` および `GetBatchImage` セクションが生成されます。これらの一般的なタスクの例については、「CloudTrail ログエントリの例 (p. 68)」を参照してください。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。この ID 情報は以下のことを確認するのに役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか.
- リクエストが、別の AWS サービスによって送信されたかどうか。

詳細については、「CloudTrail `userIdentity` 要素」を参照してください。

Amazon ECR ログファイルエントリの概要

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できる設定です。CloudTrail ログファイルには、1 つ以上のログエントリが含まれます。イベントは任意の送信元からの単一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などに関する情報が含まれます。CloudTrail ログファイルは、パブリック API 呼び出し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。

CloudTrail ログエントリの例

いくつかの一般的な Amazon ECR タスクの CloudTrail ログエントリの例を次に示します。

> Note
>
> これらの例は、読みやすくするために書式設定されています。CloudTrail ログファイルでは、すべてのエントリとイベントが 1 行に連結されます。さらに、この例は 1 つの Amazon ECR エントリに限定しています。実際の CloudTrail ログファイルには、複数の AWS サービスからのエントリとイベントが記録されます。

次の例では、`CreateRepository` アクションを説明する CloudTrail ログ エントリを示します。

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-07-11T21:54:07Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            }
        }
    },
    "eventTime": "2018-07-11T22:17:43Z",
    "eventSource": "ecr.amazonaws.com",
    "eventName": "CreateRepository",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "repositoryName": "testrepo"
```

```
    },
    "responseElements": {
        "repository": {
            "repositoryArn": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
            "repositoryName": "testrepo",
            "repositoryUri": "123456789012.dkr.ecr.us-east-2.amazonaws.com/testrepo",
            "createdAt": "Jul 11, 2018 10:17:44 PM",
            "registryId": "123456789012"
        }
    },
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "resources": [
        {
            "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
            "accountId": "123456789012"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

以下の例は、PutImage アクションを使用するイメージのプッシュの CloudTrail ログエントリを示しています。

Note

イメージをプッシュすると、InitiateLayerUpload、UploadLayerPart、および CompleteLayerUpload の参照も CloudTrail ログに表示されます。

```
{
    "eventVersion": "1.04",
    "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
    "arn": "arn:aws:sts::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major",
  "sessionContext": {
   "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2019-04-15T16:42:14Z"
   }
  }
 },
 "eventTime": "2019-04-15T16:45:00Z",
 "eventSource": "ecr.amazonaws.com",
 "eventName": "PutImage",
 "awsRegion": "us-east-2",
 "sourceIPAddress": "203.0.113.12",
 "userAgent": "console.amazonaws.com",
 "requestParameters": {
  "repositoryName": "testrepo",
  "imageTag": "latest",
  "registryId": "123456789012",
  "imageManifest": "{\n   \"schemaVersion\": 2,\n   \"mediaType\": \"application/
vnd.docker.distribution.manifest.v2+json\",\n   \"config\": {\n      \"mediaType\":
 \"application/vnd.docker.container.image.v1+json\",\n      \"size\": 5543,\n
 \"digest\": \"sha256:000b9b805af1cdb60628898c9f411996301a1c13afd3dbef1d8a16ac6dbf503a
\"\n   },\n   \"layers\": [\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 43252507,\n
 \"digest\": \"sha256:3b37166ec61459e76e33282dda08f2a9cd698ca7e3d6bc44e6a6e7580cdeff8e
\"\n      },\n      {\n         \"mediaType\": \"application/
```

```
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 846,\n         \"digest
\": \"sha256:504facff238fde83f1ca8f9f54520b4219c5b8f80be9616ddc52d31448a044bd
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 615,\n         \"digest
\": \"sha256:ebbcacd28e101968415b0c812b2d2dc60f969e36b0b08c073bf796e12b1bb449\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 850,\n         \"digest
\": \"sha256:c7fb3351ecad291a88b92b600037e2435c84a347683d540042086fe72c902b8a
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 168,\n         \"digest
\": \"sha256:2e3debadcbf7e542e2aefbce1b64a358b1931fb403b3e4aeca27cb4d809d56c2\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 37720774,\n         \"digest
\": \"sha256:f8c9f51ad524d8ae9bf4db69cd3e720ba92373ec265f5c390ffb21bb0c277941\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 30432107,\n
 \"digest\": \"sha256:813a50b13f61cf1f8d25f19fa96ad3aa5b552896c83e86ce413b48b091d7f01b
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 197,\n         \"digest
\": \"sha256:7ab043301a6187ea3293d80b30ba06c7bf1a0c3cd4c43d10353b31bc0cecfe7d
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 154,\n         \"digest
\": \"sha256:67012cca8f31dc3b8ee2305e7762fee20c250513effdedb38a1c37784a5a2e71\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 176,\n         \"digest
\": \"sha256:3bc892145603fffc9b1c97c94e2985b4cb19ca508750b15845a5d97becbd1a0e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 183,\n         \"digest
\": \"sha256:6f1c79518f18251d35977e7e46bfa6c6b9cf50df2a79d4194941d95c54258d18\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest
\": \"sha256:b7bcfbc2e2888afebede4dd1cd5eebf029bb6315feeaf0b56e425e11a50afe42\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest\":
 \"sha256:2b220f8b0f32b7c2ed8eaafe1c802633bbd94849b9ab73926f0ba46cdae91629\"\n      }\n
  ]\n}"
 },
 "responseElements": {
  "image": {
   "repositoryName": "testrepo",
   "imageManifest": "{\n   \"schemaVersion\": 2,\n   \"mediaType\": \"application/
vnd.docker.distribution.manifest.v2+json\",\n   \"config\": {\n      \"mediaType\":
 \"application/vnd.docker.container.image.v1+json\",\n      \"size\": 5543,\n
 \"digest\": \"sha256:000b9b805af1cdb60628898c9f411996301a1c13afd3dbef1d8a16ac6dbf503a
\"\n   },\n   \"layers\": [\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 43252507,\n
 \"digest\": \"sha256:3b37166ec61459e76e33282dda08f2a9cd698ca7e3d6bc44e6a6e7580cdeff8e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 846,\n         \"digest
\": \"sha256:504facff238fde83f1ca8f9f54520b4219c5b8f80be9616ddc52d31448a044bd
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 615,\n         \"digest
\": \"sha256:ebbcacd28e101968415b0c812b2d2dc60f969e36b0b08c073bf796e12b1bb449\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 850,\n         \"digest
\": \"sha256:c7fb3351ecad291a88b92b600037e2435c84a347683d540042086fe72c902b8a
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 168,\n         \"digest
\": \"sha256:2e3debadcbf7e542e2aefbce1b64a358b1931fb403b3e4aeca27cb4d809d56c2\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 37720774,\n         \"digest
\": \"sha256:f8c9f51ad524d8ae9bf4db69cd3e720ba92373ec265f5c390ffb21bb0c277941\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 30432107,\n
 \"digest\": \"sha256:813a50b13f61cf1f8d25f19fa96ad3aa5b552896c83e86ce413b48b091d7f01b
\"\n      },\n      {\n         \"mediaType\": \"application/
```

```
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 197,\n         \"digest
\": \"sha256:7ab043301a6187ea3293d80b30ba06c7bf1a0c3cd4c43d10353b31bc0cecfe7d
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 154,\n         \"digest
\": \"sha256:67012cca8f31dc3b8ee2305e7762fee20c250513effdedb38a1c37784a5a2e71\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 176,\n         \"digest
\": \"sha256:3bc892145603fffc9b1c97c94e2985b4cb19ca508750b15845a5d97becbd1a0e
\"\n      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 183,\n         \"digest
\": \"sha256:6f1c79518f18251d35977e7e46bfa6c6b9cf50df2a79d4194941d95c54258d18\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest
\": \"sha256:b7bcfbc2e2888afebede4dd1cd5eebf029bb6315feeaf0b56e425e11a50afe42\"\n
      },\n      {\n         \"mediaType\": \"application/
vnd.docker.image.rootfs.diff.tar.gzip\",\n         \"size\": 212,\n         \"digest\":
 \"sha256:2b220f8b0f32b7c2ed8eaafe1c802633bbd94849b9ab73926f0ba46cdae91629\"\n      }\n
  ]\n}",
   "registryId": "123456789012",
   "imageId": {
    "imageDigest":
 "sha256:98c8b060c21d9adbb6b8c41b916e95e6307102786973ab93a41e8b86d1fc6d3e",
    "imageTag": "latest"
   }
  }
 },
 "requestID": "cf044b7d-5f9d-11e9-9b2a-95983139cc57",
 "eventID": "2bfd4ee2-2178-4a82-a27d-b12939923f0f",
 "resources": [{
  "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
  "accountId": "123456789012"
 }],
 "eventType": "AwsApiCall",
 "recipientAccountId": "123456789012"
}
```

以下の例は、BatchGetImage アクションを使用するイメージのプルの CloudTrail ログエントリを示しています。

> **Note**
>
> イメージをプルすると、すでにイメージがローカルにない場合、GetDownloadUrlForLayer の参照も CloudTrail ログに表示されます。

```
{
    "eventVersion": "1.04",
    "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
    "arn": "arn:aws:sts::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major",
  "sessionContext": {
   "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2019-04-15T16:42:14Z"
   }
  }
 },
 "eventTime": "2019-04-15T17:23:20Z",
 "eventSource": "ecr.amazonaws.com",
 "eventName": "BatchGetImage",
 "awsRegion": "us-east-2",
```

```
 "sourceIPAddress": "203.0.113.12",
 "userAgent": "console.amazonaws.com",
 "requestParameters": {
  "imageIds": [{
   "imageTag": "latest"
  }],
  "acceptedMediaTypes": [
   "application/json",
   "application/vnd.oci.image.manifest.v1+json",
   "application/vnd.oci.image.index.v1+json",
   "application/vnd.docker.distribution.manifest.v2+json",
   "application/vnd.docker.distribution.manifest.list.v2+json",
   "application/vnd.docker.distribution.manifest.v1+prettyjws"
  ],
  "repositoryName": "testrepo",
  "registryId": "123456789012"
 },
 "responseElements": null,
 "requestID": "2a1b97ee-5fa3-11e9-a8cd-cd2391aeda93",
 "eventID": "c84f5880-c2f9-4585-9757-28fa5c1065df",
 "resources": [{
  "ARN": "arn:aws:ecr:us-east-2:123456789012:repository/testrepo",
  "accountId": "123456789012"
 }],
 "eventType": "AwsApiCall",
 "recipientAccountId": "123456789012"
}
```

Amazon ECR トラブルシューティング

この章では、Amazon Elastic Container Registry (Amazon ECR) の診断情報と、一般的な問題とエラーメッセージのトラブルシューティングステップを示します。

トピック

Docker デバッグ出力の有効化

Docker 関連の問題をデバッグを開始するには、最初にホストインスタンスで実行している Docker デーモンで Docker デバッグ出力を有効にします。Amazon ECS コンテナインスタンスの Amazon ECR からプルしたイメージを使用している場合の Docker デバッグの有効化の詳細については、Amazon Elastic Container Service Developer Guide の「Docker デバッグ出力の有効化」を参照してください。

AWS CloudTrail を有効にしています

Amazon ECR によって返されるエラーに関する追加情報は、AWS CloudTrail を有効にすることで検出できます。これは、AWS アカウントの AWS コールを記録するサービスです。CloudTrail は、Amazon S3 バケットにログファイルを配信します。CloudTrail が収集した情報を使用して、成功した AWS サービスリクエスト、そのリクエストの送信者、そのリクエストの送信時間などを確認できます。CloudTrail の詳細 (有効にする方法、ログファイルを検索する方法を含む) については、AWS CloudTrail User Guide を参照してください。CloudTrail と Amazon ECR の使用の詳細については、「AWS CloudTrail を使用した Amazon ECR API コールのログ作成 (p. 67)」を参照してください。

Amazon ECR に対するパフォーマンスの最適化

次のセクションでは、Amazon ECR の使用時にパフォーマンスを最適化するために使用できる設定と戦略の推奨事項を示します。

レイヤーの同時アップロードを利用するには、Docker 1.10 以降を使用する

Docker イメージは、イメージの中間ビルドステージであるレイヤーで構成されます。Dockerfile の各行により、新しいレイヤーが作成されます。Docker 1.10 以降を使用する場合、Docker はデフォルトで可能な限り多くのレイヤーを Amazon ECR への同時アップロードとしてプッシュし、それによりアップロード時間が高速になります。

小さなベースイメージを使用する

Docker Hub を通じて利用できるデフォルトイメージには、アプリケーションが要求しない多くの依存関係が含まれている場合があります。Docker コミュニティで他のユーザーによって作成、管理される、より小さいイメージを使用するか、Docker の最小スクラッチイメージを使用して独自のベースイメージを構築することを検討します。詳細については、Docker のドキュメントの「ベースイメージの作成」を参照してください。

Dockerfile で、最も変更の少ない依存関係を前に配置する

Docker はレイヤーをキャッシュし、それによりビルド時間を高速化します。最後のビルド以降にレイヤーで何も変更されていない場合、Docker はレイヤーを再構築する代わりにキャッシュしたバージョンを使用します。ただし、各レイヤーは、それ以前のレイヤーに依存しています。レイヤーが変更された場合、Docker はそのレイヤーだけでなく、そのレイヤーの後のレイヤーも同様に再コンパイルします。

Docker ファイルの再構築と、レイヤーの再アップロードに必要な時間を最小化するため、最も変更を行わない依存関係を Dockerfile で前に配置します。頻繁に変更を行う依存関係 (アプリケーションのソースコードなど) はスタックで後に配置します。

コマンドをチェーン処理して不要なファイルの保存を避ける

レイヤーで作成された中間ファイルは、それ以降のレイヤーで削除された場合でも、そのレイヤーの一部として残ります。次の例を考えます。

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

この例では、最初と 2 番目の RUN コマンドによって作成されたレイヤーには、元の .tar.gz ファイルと解凍されていないそのすべてのコンテンツが含まれます。ただし、.tar.gz ファイルは 4 番目の RUN コマンドによって削除されます。これらのコマンドを 1 つの RUN ステートメントにチェーン処理して、不要なファイルが最終的な Docker イメージの一部とはならないようにできます。

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

最も近いリージョンのエンドポイントを使用する

アプリケーションが実行されている場所に最も近いリージョンのエンドポイントを使用することにより、Amazon ECR からのイメージのプルのレイテンシーを減らすことができます。アプリケーションが Amazon EC2 インスタンスで実行されている場合、次のシェルコードを使用して、インスタンスのアベイラビリティーゾーンからリージョンを取得できます。

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

リージョンは --region パラメーターを使用して AWS CLI コマンドに渡すか、aws configure コマンドを使用してプロファイルのデフォルトリージョンとして設定できます。また、AWS SDK を使用して呼び出しを行うときに、リージョンを設定することもできます。詳細については、ご使用の特定のプログラミング言語の SDK のドキュメントを参照してください。

Amazon ECR 使用時の Docker コマンドのエラーのトラブルシューティング

トピック

- Amazon ECR レポジトリからイメージをプルするときのエラー: "ファイルシステムの検証に失敗しました" または "404: イメージが見つかりません" (p. 75)

場合によっては、Amazon ECR に対して Docker コマンドを実行すると、エラーメッセージが表示されることがあります。一般的なエラーメッセージと考えられる解決策を以下に説明します。

Amazon ECR レポジトリからイメージをプルするときのエラー: "ファイルシステムの検証に失敗しました" または "404: イメージが見つかりません"

docker pull コマンドを使用すると、Docker 1.9 以降で Amazon ECR レポジトリからイメージをプルするときはエラー `Filesystem verification failed` が表示されます。Docker 1.9 より前のバージョンを使用するときはエラー `404: Image not found` が表示される場合があります。

考えられる理由とその説明を以下に示します。

ローカルディスクがいっぱいである

docker pull を実行しているローカルディスクがいっぱいである場合、ローカルファイルで計算された SHA-1 ハッシュは、Amazon ECR で計算されたものとは異なる可能性があります。ローカルディスクに、プルしている Docker イメージを保存するのに十分な空き容量があることを確認します。新しいイメージの容量を確保するために、古いイメージを削除することもできます。docker images コマンドを使用して、ローカルにダウンロードしたすべての Docker イメージのリストとそのサイズを表示します。

ネットワークエラーにより、クライアントがリモートリポジトリに接続できない

Amazon ECR レポジトリへの呼び出しでは、インターネットへの機能している接続が必要です。ネットワーク設定を確認し、他のツールやアプリケーションがインターネット上のリソースにアクセスできることを確認します。プライベートサブネットの Amazon EC2 インスタンスで docker pull を実行している場合は、そのサブネットにインターネットへのルートがあることを確認します。ネットワークアドレス変換 (NAT) サーバーまたはマネージド NAT ゲートウェイを使用します。

現時点では、Amazon ECR リポジトリへの呼び出しでは、会社のファイアウォールから Amazon Simple Storage Service (Amazon S3) へのネットワークアクセスも必要です。組織で、サービスエンドポイントを許可するファイアウォールソフトウェアまたは NAT デバイスを使用している場合、現在のリージョンの Amazon S3 サービスエンドポイントが許可されていることを確認します。

HTTP プロキシの背後で Docker を使用している場合は、適切なプロキシ設定を使用して Docker を設定できます。詳細については、Docker ドキュメントの「HTTP プロキシ」を参照してください。

エラー: Amazon ECR からイメージをプルする際の、"ファイルシステムレイヤーの検証に失敗しました"

docker pull コマンドを使用してイメージをプルするときに、エラー `image image-name not found` が表示される場合があります。Docker のログを調べると、次のようなエラーが見つかる場合があります。

```
filesystem layer verification failed for digest sha256:2b96f...
```

このエラーは、イメージの 1 つ以上のレイヤーがダウンロードに失敗したことを示します。考えられる理由とその説明を以下に示します。

古いバージョンの Docker 使用している

このエラーは、Docker 1.10 より前のバージョンを使用している場合に、まれに発生することがあります。Docker クライアントを 1.10 以降にアップグレードします。

クライアントでネットワークエラーまたはディスクエラーが発生した

`Filesystem verification failed` メッセージについて前に説明したように、ディスクがいっぱいであるか、ネットワークの問題により、1 つ以上のレイヤーをダウンロードできない可能性があります。上記の推奨事項に従って、ファイルシステムがいっぱいでないこと、およびネットワーク内から Amazon S3 へのアクセスを有効にしたことを確認します。

レポジトリへのプッシュの際に、HTTP 403 エラー、または "基本的な認証情報がありません" というエラーが表示される

aws ecr get-login コマンドを使用して Docker に対して正常に認証された場合であっても、HTTP 403 (禁止) エラーが発生したり、docker push コマンドからエラーメッセージ `no basic auth credentials` を受け取る場合があります。この問題の既知の原因をいくつか次に示します。

別のリージョンに対して認証されている

認証リクエストは特定のリージョンに関連付けられていて、リージョン間では使用できません。たとえば、米国西部 (オレゴン) から認証トークンを取得する場合、これを使用して、米国東部 (バージニア北部) のリポジトリに対して認証を行うことはできません。この問題を解決するには、認証と docker push コマンド呼び出しの両方に対して同じリージョンを使用していることを確認します。

間違った AWS アカウントのリポジトリへのプッシュを認証しています

ある AWS アカウントの IAM ユーザーを使用しているが、別のアカウントによってホストされているリポジトリにプッシュする場合、aws ecr get-login を呼び出すときに --registry-ids パラメータを明示的に指定する必要があります。これを行わない場合、デフォルトで取得される Docker ログインコマンドは、IAM ユーザーをホストする同じアカウントのリポジトリへのプッシュのみを許可します。リポジトリをホストするアカウントにプッシュする権限はありません。aws ecr get-login からのレスポンスのリポジトリ URL が、プッシュ先のリポジトリ URL (URL のアカウント ID 部分を含む) と一致することを常に確認してください。

トークンの有効期限が切れた。

`GetAuthorizationToken` オペレーションを使用して取得したトークンのデフォルトのトークン有効期限は 12 時間です。ただし、一時的なセキュリティ認証情報メカニズムを使用して認証を行ってトークンを受け取る場合、トークンの有効期間は、一時的なセキュリティ認証情報の長さと同じです。一時的なセキュリティ認証情報のメカニズムには、多要素認証 (MFA) や AWS Security Token Service があります。たとえば、ロールを引き受けて aws ecr get-login コマンドを呼び出す場合、認証トークンは 15 分～1 時間で有効期限が切れます。これは、aws sts assume-role コマンドを呼び出すときに使用する設定によって異なります。

`wincred` 認証情報マネージャーのバグ

一部のバージョンの Docker for Windows では、`wincred` という認証情報マネージャーを使用します。この認証情報マネージャーは、aws ecr get-login によって生成された Docker ログインコマンドを正しく処理しません (詳細については、https://github.com/docker/docker/issues/22910 を参照)。出力される Docker ログインコマンドは実行できますが、イメージをプッシュまたはプルしようとすると、コマンドは失敗します。これに対処するには、aws ecr get-login から出力される Docker ログインコマンドのレジストリ引数から `https://` スキームを削除します。HTTPS スキームのない Docker ログインコマンドの例を次に示します。

```
docker login -u AWS -p <password> <aws_account_id>.dkr.ecr.<region>.amazonaws.com
```

Amazon ECR エラーメッセージのトラブルシューティング

トピック

Amazon ECS コンソールまたは AWS CLI を通じてトリガーされた API コールがエラーメッセージで終了することがあります。一般的なエラーメッセージと考えられる解決策を以下に説明します。

aws ecr get-login を実行する際のエラー: "デーモンからのエラー応答: 無効なレジストリエンドポイントです"

aws ecr get-login コマンドを実行して Amazon ECR リポジトリのログイン認証情報を取得するときに、次のエラーが表示される場合があります。

```
Error response from daemon: invalid registry endpoint
    https://xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/v0/: unable to ping registry
 endpoint
    https://xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/v0/
v2 ping attempt failed with error: Get https://xxxxxxxxxxxx.dkr.ecr.us-
east-1.amazonaws.com/v2/:
    dial tcp: lookup xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com on 172.20.10.1:53:
    read udp 172.20.10.1:53: i/o timeout
```

このエラーは、Docker Toolbox、Docker for Windows、または Docker for Mac を実行している MacOS X および Windows システムで発生することがあります。これは、他のアプリケーションがローカルゲートウェイ (192.168.0.1) を介してルートを変更するときに、よく発生します。仮想マシンはこのゲートウェイを介して Amazon ECR サービスにアクセスする必要があります。Docker Toolbox の使用中にこのエラーが発生した場合は、Docker Machine 環境またはローカルクライアントのオペレーティングシステムを再起動することで、しばしば解決できます。これによって問題が解決しない場合は、docker-machine ssh コマンドを使用してコンテナインスタンスにログインします。外部ホストで DNS ルックアップを行って、ローカルホストと同じ結果となることを確認します。結果が異なる場合は、Docker Toolbox のドキュメントを参照して、Docker Machine 環境が適切に設定されていることを確認します。

HTTP 429: リクエストが多すぎる、または ThrottleException

1 つ以上の Amazon ECR コマンドまたは API コールから、`429: Too Many Requests` エラーまたは `ThrottleException` エラーを受け取ることがあります。Docker ツールを Amazon ECR とともに使用している場合、Docker バージョン 1.12.0 以降では、エラーメッセージ `TOOMANYREQUESTS: Rate`

exceeded が表示されることがあります。Docker バージョン 1.12.0 以前では、エラー Unknown: Rate exceeded が表示されることがあります。

これは、短い間隔で Amazon ECR の単一のエンドポイントを繰り返し呼び出していて、リクエストがスロットリングされていることを示します。スロットリングは、1 人のユーザーから単一のエンドポイントへの呼び出しが、期間中に特定のしきい値を超えたときに発生します。

Amazon ECR のさまざまな API オペレーションでは、異なるスロットリングが行われます。

たとえば、GetAuthorizationToken アクションのスロットリングは 20 TPS (トランザクション/秒) で、最大 200 TPS のバーストが可能です。各リージョンで、各アカウントには、最大 200 GetAuthorizationToken クレジットを保存できるバケットが割り当てられ、1 秒あたり 20 クレジットの割合で補充されます。バケットに 200 クレジットがある場合、1 秒あたり 200 GetAuthorizationToken API トランザクションに達すると、1 秒あたり 20 トランザクションが無期限に維持されます。

スロットリングエラーを処理するには、増分バックオフをコードに含めて再試行関数を実装します。詳細については、アマゾン ウェブ サービス全般のリファレンスの「AWS でのエラーの再試行とエクスポネンシャルバックオフ」を参照してください。

HTTP 403: "ユーザー [arn] は [operation] の実行を許可されていません"

Amazon ECR を使用してアクションを実行しようとすると、次のエラーを受け取ることがあります。

```
$ aws ecr get-login
A client error (AccessDeniedException) occurred when calling the GetAuthorizationToken
 operation:
    User: arn:aws:iam::account-number:user/username is not authorized to perform:
    ecr:GetAuthorizationToken on resource: *
```

これは、Amazon ECR を使用するアクセス権限がユーザーに付与されていないか、それらのアクセス権限が正しく設定されていないことを示します。特に、Amazon ECR レポジトリに対してアクションを実行している場合は、そのリポジトリにアクセスする権限がユーザーに付与されていることを確認します。Amazon ECR のアクセス権限の作成と検証の詳細については、「Amazon ECR IAM のポリシーおよびロール (p. 46)」を参照してください。

HTTP 404: "レポジトリは存在しません" エラー

現在存在しない Docker Hub リポジトリを指定すると、Docker Hub はそのレポジトリを自動的に作成します。Amazon ECR では、レポジトリを使用するには、その前に新しいリポジトリを明示的に作成する必要があります。これにより、(タイプミスなどの原因で) 新しいレポジトリが間違って作成されるのを防止し、適切なセキュリティアクセスポリシーが明示的に新しいレポジトリに割り当てられます。レポジトリの作成方法の詳細については、「Amazon ECR リポジトリ (p. 15)」を参照してください。

ドキュメント履歴

次の表に、Amazon ECR の前回のリリース以後に行われた、文書の重要な変更を示します。また、お客様からいただいたフィードバックに対応するために、ドキュメントを頻繁に更新しています。

- 文書の最終更新: 2017 年 11 月 21 日

変更	説明	改訂日
インターフェイス VPC エンドポイント (AWS PrivateLink)	AWS PrivateLink を使用したインターフェイス VPC エンドポイントの設定のサポートが追加されました。これにより、インターネット、NAT インスタンス、VPN 接続、または AWS Direct Connect を経由せずに、VPC と Amazon ECR とをプライベートに接続できます。 詳細については、「Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink) (p. 56)」を参照してください。	2019 年 1 月 25 日
リソースへのタグ付け	メタデータタグをリポジトリに追加するためのサポートが Amazon ECR に追加されました。 詳細については、「Amazon ECR リポジトリのタグ付け (p. 22)」を参照してください。	2018 年 12 月 18 日
Amazon ECR の名前変更	Amazon Elastic Container Registry の名前が変更されました (旧 Amazon EC2 コンテナレジストリ)。	2017 年 11 月 21 日
ライフサイクルポリシー	Amazon ECR ライフサイクルポリシーで、リポジトリ内のイメージのライフサイクル管理を指定することができます。 詳細については、「Amazon ECR ライフサイクルポリシー (p. 34)」を参照してください。	2017 年 10 月 11 日
Amazon ECR が Docker Image Manifest 2、Schema 2 をサポート	Amazon ECR が Docker Image Manifest V2 Schema 2 (Docker バージョン 1.10 以降で使用) をサポートするようになりました。 詳細については、「コンテナイメージマニフェストの形式 (p. 30)」を参照してください。	2017 年 1 月 27 日
Amazon ECR の一般提供	Amazon Elastic Container Registry (Amazon ECR) は、安全性と信頼性に優れたスケーラブルなマネージド AWS Docker レジストリサービスです。	2015 年 12 月 21 日

AWS の用語集

最新の AWS の用語については、『AWS General Reference』の「AWS の用語集」を参照してください。