Amazon ECR

User Guide

API Version 2015-09-21

Amazon ECR: User Guide

Table of Contents

What Is Amazon Elastic Container Registry?

Amazon Elastic Container Registry (Amazon ECR) is a managed AWS container image registry service that is secure, scalable, and reliable. Amazon ECR supports private container image repositories with resource-based permissions using AWS IAM so that specific users or Amazon EC2 instances can access repositories and images. Developers can use their preferred CLI to push, pull, and manage Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts.

The AWS container services team maintains a public roadmap on GitHub. It contains information about what the teams are working on and allows all AWS customers the ability to give direct feedback. For more information, see AWS Containers Roadmap.

Components of Amazon ECR

Amazon ECR contains the following components:

Registry

An Amazon ECR registry is provided to each AWS account; you can create image repositories in your registry and store images in them. For more information, see Amazon ECR registries (p. 13).

Authorization token

Your client must authenticate to Amazon ECR registries as an AWS user before it can push and pull images. For more information, see Registry authentication (p. 13).

Repository

An Amazon ECR image repository contains your Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts. For more information, see Amazon ECR repositories (p. 16).

Repository policy

You can control access to your repositories and the images within them with repository policies. For more information, see Repository policies (p. 19).

Image

You can push and pull container images to your repositories. You can use these images locally on your development system, or you can use them in Amazon ECS task definitions and Amazon EKS pod specifications. For more information, see Using Amazon ECR images with Amazon ECS (p. 51) and Using Amazon ECR Images with Amazon EKS (p. 52).

Features of Amazon ECR

Amazon ECR provides the following features:

- Lifecycle policies enable you to manage the lifecycle of the images in your repositories. You define rules which result in the cleaning up of unused images which you can test before applying to your repository. For more information, see Lifecycle policies (p. 35).

- Image scanning helps in identifying software vulnerabilities in your container images. Each repository can be configured to **scan on push**, which ensures that each new image pushed to the repository is scanned. You can then retrieve the results of the image scan. For more information, see Image scanning (p. 47).

How to Get Started with Amazon ECR

To use Amazon ECR, you must be set up to install the AWS Command Line Interface and Docker. For more information, see Setting up with Amazon ECR (p. 3) and Getting started with Amazon ECR using the AWS CLI (p. 7).

Pricing for Amazon ECR

With Amazon ECR, you only pay for the amount of data you store in your repositories and for the data transfer from your image pushes and pulls. For more information, see Amazon ECR pricing.

Setting up with Amazon ECR

If you've signed up for AWS and have been using Amazon Elastic Container Service (Amazon ECS) or Amazon Elastic Kubernetes Service (Amazon EKS), you are close to being able to use Amazon ECR. The setup process for these two services is similar, as Amazon ECR is an extension to these services. To use the AWS CLI with Amazon ECR, you must use a version of the AWS CLI that supports the latest Amazon ECR features. If you do not see support for an Amazon ECR feature in the AWS CLI, you should upgrade to the latest version. For more information, see http://aws.amazon.com/cli/.

Complete the following tasks to get set up to push a container image to Amazon ECR for the first time. If you have already completed any of these steps, you may skip them and move on to the next step.

Sign up for AWS

When you sign up for AWS, your AWS account is automatically signed up for all services, including Amazon ECR. You are charged only for the services that you use.

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

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

Note your AWS account number, because you'll need it for the next task.

Create an IAM user

Services in AWS, such as Amazon ECR, require that you provide credentials when you access them, so that the service can determine whether you have permission to access its resources. The console requires your password. You can create access keys for your AWS account to access the command line interface or API. However, we don't recommend that you access AWS using the credentials for your AWS account; we recommend that you use AWS Identity and Access Management (IAM) instead. Create an IAM user, and then add the user to an IAM group with administrative permissions or grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console.

To create an administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the IAM console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note
We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.

2. In the navigation pane, choose **Users** and then choose **Add user**.
3. For **User name**, enter **`Administrator`**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **`Administrators`**.
10. Choose **Filter policies**, and then select **AWS managed -job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

    Note
    You must activate IAM user and role access to Billing before you can use the `AdministratorAccess` permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.
12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see Access Management and Example Policies.

To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where *your_aws_account_id* is your AWS account number without the hyphens (for example, if your AWS account number is `1234-5678-9012`, your AWS account ID is `123456789012`):

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

Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays "*your_user_name* @ *your_aws_account_id*".

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias. From the IAM dashboard, choose **Customize** and enter an **Account Alias**, such as your company name. For more information, see Your AWS Account ID and Its Alias in the *IAM User Guide*.

To sign in after you create an account alias, use the following URL:

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

To verify the sign-in link for IAM users for your account, open the IAM console and check under **IAM users sign-in link** on the dashboard.

For more information about IAM, see the AWS Identity and Access Management User Guide.

Getting started with Amazon ECR using the AWS Management Console

Get started with Amazon ECR by creating a repository in the Amazon ECR console. The Amazon ECR console guides you through the process to get started creating your first repository.

Before you begin, be sure that you've completed the steps in Setting up with Amazon ECR (p. 3).

To create an image repository

A repository is where you store your Docker or Open Container Initiative (OCI) images in Amazon ECR. Each time you push or pull an image from Amazon ECR, you specify the repository and the registry location which informs where to push the image to or where to pull it from.

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. Choose **Get Started**.
3. For **Tag immutability**, choose the tag mutability setting for the repository. Repositories configured with immutable tags will prevent image tags from being overwritten. For more information, see Image tag mutability (p. 46).
4. For **Scan on push**, choose the image scanning setting for the repository. Repositories configured to scan on push will start an image scan whenever an image is pushed, otherwise image scans need to be started manually. For more information, see Image scanning (p. 47).
5. Choose **Create repository**.

Build, tag, and push a Docker image

In this section of the wizard, you use the Docker CLI to tag an existing local image (that you have built from a Dockerfile or pulled from another registry, such as Docker Hub) and then push the tagged image to your Amazon ECR registry. For more detailed steps on using the Docker CLI, see Getting started with Amazon ECR using the AWS CLI (p. 7).

1. Select the repository you created and choose **View push commands** to view the steps to push an image to your new repository.
2. Run the login command that authenticates your Docker client to your registry by pasting the command from the console into a terminal window. This command provides an authorization token that is valid for 12 hours.
3. (Optional) If you have a Dockerfile for the image to push, build the image and tag it for your new repository. Pasting the **docker build** command from the console into a terminal window. Make sure that you are in the same directory as your Dockerfile.
4. Tag the image with your Amazon ECR registry URI and your new repository by pasting the **docker tag** command from the console into a terminal window. The console command assumes that your image was built from a Dockerfile in the previous step. If you did not build your image from a Dockerfile, replace the first instance of *`repository`*`:latest` with the image ID or image name of your local image to push.
5. Push the newly tagged image to your repository by pasting the **docker push** command into a terminal window.
6. Choose **Close**.

Getting started with Amazon ECR using the AWS CLI

The following steps walk you through the steps needed to push a container image to Amazon ECR for the first time using the Docker CLI and the AWS CLI.

For more information on the other tools available for managing your AWS resources, including the different AWS SDKs, IDE toolkits, and the Windows PowerShell command line tools, see http://aws.amazon.com/tools/.

Prerequisites

Before you begin, be sure that you have completed the steps in Setting up with Amazon ECR (p. 3).

If you do not already have the latest AWS CLI and Docker installed and ready to use, use the following steps to install both of these tools.

Install the AWS CLI

You can use the AWS command line tools to issue commands at your system's command line to perform Amazon ECR and other AWS tasks. This can be faster and more convenient than using the console. The command line tools are also useful for building scripts that perform AWS tasks.

To use the AWS CLI with Amazon ECR, install the latest AWS CLI version (Amazon ECR functionality is available in the AWS CLI starting with version 1.9.15). You can check your AWS CLI version with the **aws --version** command. For information about installing the AWS CLI or upgrading it to the latest version, see Installing the AWS CLI version 2 in the *AWS Command Line Interface User Guide*.

Install Docker

Docker is available on many different operating systems, including most modern Linux distributions, like Ubuntu, and even Mac OSX and Windows. For more information about how to install Docker on your particular operating system, go to the Docker installation guide.

You don't need a local development system to use Docker. If you are using Amazon EC2 already, you can launch an Amazon Linux 2 instance and install Docker to get started.

If you already have Docker installed, skip to Step 1: Create a Docker image (p. 8).

To install Docker on an Amazon EC2 instance

1. Launch an instance with the Amazon Linux 2 AMI. For more information, see Launching an Instance in the *Amazon EC2 User Guide for Linux Instances*.
2. Connect to your instance. For more information, see Connect to Your Linux Instance in the *Amazon EC2 User Guide for Linux Instances*.

3. Update the installed packages and package cache on your instance.

```
sudo yum update -y
```

4. Install the most recent Docker Community Edition package.

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

5. Start the Docker service.

```
sudo service docker start
```

6. Add the `ec2-user` to the `docker` group so you can execute Docker commands without using `sudo`.

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

7. Log out and log back in again to pick up the new `docker` group permissions. You can accomplish this by closing your current SSH terminal window and reconnecting to your instance in a new one. Your new SSH session will have the appropriate `docker` group permissions.
8. Verify that the `ec2-user` can run Docker commands without `sudo`.

```
docker info
```

Note
In some cases, you may need to reboot your instance to provide permissions for the `ec2-user` to access the Docker daemon. Try rebooting your instance if you see the following error:

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

Step 1: Create a Docker image

In this section, you create a Docker image of a simple web application, and test it on your local system or EC2 instance, and then push the image to a container registry (such as Amazon ECR or Docker Hub) so you can use it in an ECS task definition.

To create a Docker image of a simple web application

1. Create a file called `Dockerfile`. A Dockerfile is a manifest that describes the base image to use for your Docker image and what you want installed and running on it. For more information about Dockerfiles, go to the Dockerfile Reference.

```
touch Dockerfile
```

2. Edit the `Dockerfile` you just created and add the following content.

```
FROM ubuntu:18.04

# Install dependencies
RUN apt-get update && \
 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 && \
 echo 'mkdir -p /var/run/apache2' >> /root/run_apache.sh && \
 echo 'mkdir -p /var/lock/apache2' >> /root/run_apache.sh && \
 echo '/usr/sbin/apache2 -D FOREGROUND' >> /root/run_apache.sh && \
 chmod 755 /root/run_apache.sh

EXPOSE 80

CMD /root/run_apache.sh
```

This Dockerfile uses the Ubuntu 18.04 image. The `RUN` instructions update the package caches, install some software packages for the web server, and then write the "Hello World!" content to the web server's document root. The `EXPOSE` instruction exposes port 80 on the container, and the `CMD` instruction starts the web server.

3. Build the Docker image from your Dockerfile.

   Note
   Some versions of Docker may require the full path to your Dockerfile in the following command, instead of the relative path shown below.

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

4. Run **docker images** to verify that the image was created correctly.

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

   Output:

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

5. Run the newly built image. The `-p 80:80` option maps the exposed port 80 on the container to port 80 on the host system. For more information about **docker run**, go to the Docker run reference.

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

   Note
   Output from the Apache web server is displayed in the terminal window. You can ignore the "`Could not reliably determine the server's fully qualified domain name`" message.

6. Open a browser and point to the server that is running Docker and hosting your container.

   - If you are using an EC2 instance, this is the **Public DNS** value for the server, which is the same address you use to connect to the instance with SSH. Make sure that the security group for your instance allows inbound traffic on port 80.
   - If you are running Docker locally, point your browser to http://localhost/.
   - If you are using **docker-machine** on a Windows or Mac computer, find the IP address of the VirtualBox VM that is hosting Docker with the **docker-machine ip** command, substituting *`machine-name`* with the name of the docker machine you are using.

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

   You should see a web page with your "Hello World!" statement.

7. Stop the Docker container by typing **Ctrl + c**.

Step 2: Authenticate to your default registry

After you have installed and configured the AWS CLI, authenticate the Docker CLI to your default registry. That way, the **docker** command can push and pull images with Amazon ECR. The AWS CLI provides a **get-login-password** command to simplify the authentication process.

To authenticate Docker to an Amazon ECR registry with get-login-password, run the **aws ecr get-login-password** command. When passing the authentication token to the **docker login** command, use the value `AWS` for the username and specify the Amazon ECR registry URI you want to authenticate to. If authenticating to multiple registries, you must repeat the command for each registry.

Important
If you receive an error, install or upgrade to 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*.

- get-login-password (AWS CLI)

```
aws ecr get-login-password --region region | docker login --username AWS --password-
stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

- Get-ECRLoginCommand (AWS Tools for Windows PowerShell)

```
(Get-ECRLoginCommand).Password | docker login --username AWS --password-
stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

Step 3: Create a repository

Now that you have an image to push to Amazon ECR, you must create a repository to hold it. In this example, you create a repository called `hello-world` to which you later push the `hello-world:latest` image. To create a repository, run the following command:

```
aws ecr create-repository \
    --repository-name hello-world \
    --image-scanning-configuration scanOnPush=true \
    --region us-east-1
```

Step 4: Push an image to Amazon ECR

Now you can push your image to the Amazon ECR repository you created in the previous section. You use the **docker** CLI to push images, but there are a few prerequisites that must be satisfied for this to work properly:

- The minimum version of **docker** is installed: 1.7
- The Amazon ECR authorization token has been configured with **docker login**.
- The Amazon ECR repository exists and the user has access to push to the repository.

After those prerequisites are met, you can push your image to your newly created repository in the default registry for your account.

To tag and push an image to Amazon ECR

1. List the images you have stored locally to identify the image to tag and push.

```
docker images
```

Output:

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

2. Tag the image to push to your repository.

```
docker tag hello-world:latest aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
```

3. Push the image.

```
docker push aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

Output:

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

Step 5: Pull an image from Amazon ECR

After your image has been pushed to your Amazon ECR repository, you can pull it from other locations. Use the **docker** CLI to pull images, but there are a few prerequisites that must be satisfied for this to work properly:

- The minimum version of **docker** is installed: 1.7
- The Amazon ECR authorization token has been configured with **docker login**.
- The Amazon ECR repository exists and the user has access to pull from the repository.

After those prerequisites are met, you can pull your image. To pull your example image from Amazon ECR, run the following command:

```
docker pull aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

Output:

```
latest: Pulling from hello-world
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/hello-
world:latest
```

Step 6: Delete an image

If you decide that you no longer need or want an image in one of your repositories, you can delete it with the **batch-delete-image** command. To delete an image, you must specify the repository that it is in and either a `imageTag` or `imageDigest` value for the image. The example below deletes an image in the `hello-world` repository with the image tag `latest`.

```
aws ecr batch-delete-image \
      --repository-name hello-world \
      --image-ids imageTag=latest
```

Output:

```
{
    "failures": [],
    "imageIds": [
        {
            "imageTag": "latest",
            "imageDigest":
 "sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636b95aed25f52c89b"
        }
    ]
}
```

Step 7: Delete a repository

If you decide that you no longer need or want an entire repository of images, you can delete the repository. By default, you cannot delete a repository that contains images; however, the `--force` flag allows this. To delete a repository that contains images (and all the images within it), run the following command.

```
aws ecr delete-repository \
      --repository-name hello-world \
      --force
```

Amazon ECR registries

Amazon ECR registries host your container images in a highly available and scalable architecture, allowing you to deploy containers reliably for your applications. You can use your registry to manage image repositories consisting of Docker and Open Container Initiative (OCI) images. Each AWS account is provided with a single (default) Amazon ECR registry.

Registry concepts

- The URL for your default registry is `https://aws_account_id.dkr.ecr.region.amazonaws.com`.
- By default, your account has read and write access to the repositories in your default registry. However, IAM users require permissions to make calls to the Amazon ECR APIs and to push or pull images from your repositories. Amazon ECR provides several managed policies to control user access at varying levels. For more information, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).
- You must authenticate your Docker client to a registry so that you can use the **docker push** and **docker pull** commands to push and pull images to and from the repositories in that registry. For more information, see Registry authentication (p. 13).
- Repositories can be controlled with both IAM user access policies and repository policies. For more information about repository policies, see Repository policies (p. 19).

Registry authentication

You can use the AWS Management Console, the AWS CLI, or the AWS SDKs to create and manage repositories. You can also use those methods to perform some actions on images, such as listing or deleting them. These clients use standard AWS authentication methods. Although technically you can use the Amazon ECR API to push and pull images, you are much more likely to use the Docker CLI or a language-specific Docker library.

The Docker CLI does not support native IAM authentication methods. Additional steps must be taken so that Amazon ECR can authenticate and authorize Docker push and pull requests.

The following registry authentication methods are available.

Using the Amazon ECR credential helper

Amazon ECR provides a Docker credential helper which makes it easier to store and use Docker credentials when pushing and pulling images to Amazon ECR. For installation and configuration steps, see Amazon ECR Docker Credential Helper.

Using an authorization token

An authorization token's permission scope matches that of the IAM principal used to retrieve the authentication token. An authentication token is used to access any Amazon ECR registry that your IAM principal has access to and is valid for 12 hours. To obtain an authorization token, you must use the GetAuthorizationToken API operation to retrieve a base64-encoded authorization token containing the username `AWS` and an encoded password. The AWS CLI `get-login-password` command simplifies

this by retrieving and decoding the authorization token which you can then pipe into a **docker login** command to authenticate.

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

To authenticate Docker to an Amazon ECR registry with get-login-password, run the **aws ecr get-login-password** command. When passing the authentication token to the **docker login** command, use the value `AWS` for the username and specify the Amazon ECR registry URI you want to authenticate to. If authenticating to multiple registries, you must repeat the command for each registry.

> **Important**
> If you receive an error, install or upgrade to 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*.

- get-login-password (AWS CLI)

```
aws ecr get-login-password --region region | docker login --username AWS --password-
stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

- Get-ECRLoginCommand (AWS Tools for Windows PowerShell)

```
(Get-ECRLoginCommand).Password | docker login --username AWS --password-
stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

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

When using AWS CLI versions prior to 1.17.10, the **get-login** command is available to authenticate to your Amazon ECR registry. 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
   ```

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

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

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 (Get-ECRLoginCommand -Region region).Command
   ```

   > **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 use the **ecr get-login-password** command as described above.

Using HTTP API authentication

Amazon ECR supports the Docker Registry HTTP API. However, because Amazon ECR is a private registry, you must provide an authorization token with every HTTP request. You can add an HTTP authorization header using the `-H` option for **curl** and pass the authorization token provided by the **get-authorization-token** AWS CLI command.

To authenticate with the Amazon ECR HTTP API

1. Retrieve an authorization token with the AWS CLI and set it to an environment variable.

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

2. To authenticate to the API, pass the `$TOKEN` variable to the `-H` option of **curl**. For example, the following command lists the image tags in an Amazon ECR repository. For more information, see the Docker Registry HTTP API reference documentation.

```
curl -i -H "Authorization: Basic $TOKEN"
 https://aws_account_id.dkr.ecr.region.amazonaws.com/v2/amazonlinux/tags/list
```

   Output:

```
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 repositories

Amazon Elastic Container Registry (Amazon ECR) provides API operations to create, monitor, and delete image repositories and set permissions that control who can access them. You can perform the same actions in the **Repositories** section of the Amazon ECR console. Amazon ECR also integrates with the Docker CLI allowing you to push and pull images from your development environments to your repositories.

Topics

Repository concepts

- By default, your account has read and write access to the repositories in your default registry (*aws_account_id*.dkr.ecr.*region*.amazonaws.com). However, IAM users require permissions to make calls to the Amazon ECR APIs and to push or pull images from your repositories. Amazon ECR provides several managed policies to control user access at varying levels; for more information, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).
- Repositories can be controlled with both IAM user access policies and repository policies. For more information, see Repository policies (p. 19).
- Repository names can support namespaces, which you can use to group similar repositories. For example if there are several teams using the same registry, Team A could use the `team-a` namespace while Team B uses the `team-b` namespace. Each team could have their own image called `web-app`, but because they are each prefaced with the team namespace, the two images can be used simultaneously without interference. Team A's image would be called `team-a/web-app`, while Team B's image would be called `team-b/web-app`.

Creating a repository

Before you can push your Docker images to Amazon ECR, you must create a repository to store them in. You can create Amazon ECR repositories with the AWS Management Console, or with the AWS CLI and AWS SDKs.

To create a repository

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region to create your repository in.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose **Create repository**.
5. For **Repository name**, enter a unique name for your repository. The repository name may be specified on its own (such as `nginx-web-app`) or it can be prepended with a namespace to group the repository into a category (such as `project-a/nginx-web-app`).

Note
The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, and forward slashes.

6. For **Tag immutability**, choose the tag mutability setting for the repository. Repositories configured with immutable tags will prevent image tags from being overwritten. For more information, see Image tag mutability (p. 46).
7. For **Scan on push**, choose the image scanning setting for the repository. Repositories configured to scan on push will start an image scan whenever an image is pushed, otherwise image scans need to be started manually. For more information, see Image scanning (p. 47).
8. For **KMS encryption**, choose whether to enable encryption of the images in the repository using AWS Key Management Service. By default, when KMS encryption is enabled Amazon ECR uses an AWS managed customer master key (CMK) with alias `aws/ecr`, which is created in your account the first time that you create a repository with KMS encryption enabled. For more information, see Encryption at rest (p. 71).
9. When KMS encryption is enabled, select **Customer encryption settings (advanced)** to choose your own CMK. The CMK must exist in the same Region as the cluster. Choose **Create an AWS KMS key** to navigate to the AWS KMS console to create your own key.
10. Choose **Create repository**.
11. (Optional) Select the repository you created and choose **View push commands** to view the steps to push an image to your new repository.

    a. Run the login command that authenticates your Docker client to your registry by pasting the command from the console into a terminal window. This command provides an authorization token that is valid for 12 hours.
    b. (Optional) If you have a Dockerfile for the image to push, build the image and tag it for your new repository. Pasting the **docker build** command from the console into a terminal window. Make sure that you are in the same directory as your Dockerfile.
    c. Tag the image with your Amazon ECR registry URI and your new repository by pasting the **docker tag** command from the console into a terminal window. The console command assumes that your image was built from a Dockerfile in the previous step. If you did not build your image from a Dockerfile, replace the first instance of *`repository`*`:latest` with the image ID or image name of your local image to push.
    d. Push the newly tagged image to your repository by pasting the **docker push** command into a terminal window.
    e. Choose **Close**.

Viewing repository information

After you have created a repository, you can view its information in the AWS Management Console:

- Which images are stored in a repository
- Whether an image is tagged
- The tags for the image
- The SHA digest for the images
- The size of the images in MiB
- When the image was pushed to the repository

Note
Beginning with Docker version 1.9, the Docker client compresses image layers before pushing them to a V2 Docker registry. The output of the **docker images** command shows the

uncompressed image size, so it may return a larger image size than the image sizes shown in the AWS Management Console.

To view repository information

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region that contains the repository to view.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the repository to view.
5. On the **Repositories : *repository_name*** page, use the navigation bar to view information about an image.
   - Choose **Images** to view information about the images in the repository. If there are untagged images that you would like to delete, you can select the box to the left of the repositories to delete and choose **Delete**. For more information, see Deleting an image (p. 32).
   - Choose **Permissions** to view the repository policies that are applied to the repository. For more information, see Repository policies (p. 19).
   - Choose **Lifecycle Policy** to view the lifecycle policy rules that are applied to the repository. The lifecycle events history is also viewed here. For more information, see Lifecycle policies (p. 35).
   - Choose **Tags** to view the metadata tags that are applied to the repository.

Editing a repository

Existing repositories can be edited to change its image tag mutability and image scanning settings.

To edit a repository

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region that contains the repository to edit.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, select the repository to edit and choose **Edit**.
5. For **Tag immutability**, choose the tag mutability setting for the repository. Repositories configured with immutable tags will prevent image tags from being overwritten. For more information, see Image tag mutability (p. 46).
6. For **Scan on push**, choose the image scanning setting for the repository. Repositories configured to scan on push will start an image scan whenever an image is pushed, otherwise image scans need to be started manually. For more information, see Image scanning (p. 47).
7. Choose **Save** to update the repository settings.

Deleting a repository

If you are done using a repository, you can delete it. When you delete a repository in the AWS Management Console, all of the images contained in the repository are also deleted; this cannot be undone.

To delete a repository

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region that contains the repository to delete.
3. In the navigation pane, choose **Repositories**.

4. On the **Repositories** page, select the repository to delete and choose **Delete**.
5. In the **Delete *repository_name*** window, verify that the selected repositories should be deleted and choose **Delete**.

   Important
   Any images in the selected repositories are also deleted.

Repository policies

Amazon ECR uses resource-based permissions to control access to repositories. Resource-based permissions let you specify which IAM users or roles have access to a repository and what actions they can perform on it. By default, only the repository owner has access to a repository. You can apply a policy document that allow additional permissions to your repository.

Repository policies vs IAM policies

Amazon ECR repository policies are a subset of IAM policies that are scoped for, and specifically used for, controlling access to individual Amazon ECR repositories. IAM policies are generally used to apply permissions for the entire Amazon ECR service but can also be used to control access to specific resources as well.

Both Amazon ECR repository policies and IAM policies are used when determining which actions a specific IAM user or role may perform on a repository. If a user or role is allowed to perform an action through a repository policy but is denied permission through an IAM policy (or vice versa) then the action will be denied. A user or role only needs to be allowed permission for an action through either a repository policy or an IAM policy but not both for the action to be allowed.

Important
Amazon ECR requires that users have permission to make calls to the `ecr:GetAuthorizationToken` API through an IAM policy before they can authenticate to a registry and push or pull any images from any Amazon ECR repository. Amazon ECR provides several managed IAM policies to control user access at varying levels; for more information, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).

You can use either of these policy types to control access to your repositories, as shown in the following examples.

This example shows an Amazon ECR repository policy, which allows for a specific IAM user to describe the repository and the images within the repository.

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

This example shows an IAM policy that achieves the same goal as above, by scoping the policy to a repository (specified by the full ARN of the repository) using the resource parameter. For more information about Amazon Resource Name (ARN) format, see Resources (p. 61).

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "ECR Repository Policy",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::account-id:user/username"
    },
    "Action": [
      "ecr:DescribeImages",
      "ecr:DescribeRepositories"
    ],
    "Resource": [
      "arn:aws:ecr:region:account-id:repository/repository-name"
    ]
    }]
}
```

Topics

Setting a repository policy statement

You can add an access policy statement to a repository in the AWS Management Console by following the steps below. You can add multiple policy statements per repository. For example policies, see Repository policy examples (p. 21).

> **Important**
> Amazon ECR requires that users have permission to make calls to the `ecr:GetAuthorizationToken` API through an IAM policy before they can authenticate to a registry and push or pull any images from any Amazon ECR repository. Amazon ECR provides several managed IAM policies to control user access at varying levels; for more information, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).

To set a repository policy statement

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region that contains the repository to set a policy statement on.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the repository to set a policy statement on.
5. In the navigation pane, choose **Permissions**, **Edit**.
6. On the **Edit permissions** page, choose **Add statement**.
7. For **Statement name**, enter a name for the statement.
8. For **Effect**, choose whether the policy statement will result in an allow or an explicit deny.
9. For **Principal**, choose the scope to apply the policy statement to. For more information, see AWS JSON Policy Elements: Principal in the *IAM User Guide*.
   - You can apply the statement to all authenticated AWS users by selecting the **Everyone (*)** check box.
   - For **Service principal**, specify the service principal name (for example, `ecs.amazonaws.com`) to apply the statement to a specific service.

- For **AWS Account IDs**, specify an AWS account number (for example, `111122223333`) to apply the statement to all users under a specific AWS account. Multiple accounts can be specified by using a comma delimited list.
- For **IAM Entities**, select the roles or users under your AWS account to apply the statement to.

  Note
  For more complicated repository policies that are not currently supported in the AWS Management Console, you can apply the policy with the **set-repository-policy** AWS CLI command.

10. For **Actions**, choose the scope of the Amazon ECR API operations that the policy statement should apply to from the list of individual API operations.
11. When you are finished, choose **Save** to set the policy.
12. Repeat the previous step for each repository policy to add.

Deleting a repository policy statement

If you no longer want an existing repository policy statement to apply to a repository, you can delete it.

To delete a repository policy statement

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region that contains the repository to delete a policy statement from.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the repository to delete a policy statement from.
5. In the navigation pane, choose **Permissions**, **Edit**.
6. On the **Edit permissions** page, choose **Delete**.

Repository policy examples

The following examples show policy statements that you could use to control the permissions that users have to Amazon ECR repositories.

Important
Amazon ECR requires that users have permission to make calls to the `ecr:GetAuthorizationToken` API through an IAM policy before they can authenticate to a registry and push or pull any images from any Amazon ECR repository. Amazon ECR provides several managed IAM policies to control user access at varying levels; for more information, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).

Example: Allow an IAM user within your account

The following repository policy allows IAM users within your account to push and pull images.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "AllowPushPull",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id:user/push-pull-user-1",
```

```
                "arn:aws:iam::account-id:user/push-pull-user-2"
            ]
        },
        "Action": [
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage",
            "ecr:BatchCheckLayerAvailability",
            "ecr:PutImage",
            "ecr:InitiateLayerUpload",
            "ecr:UploadLayerPart",
            "ecr:CompleteLayerUpload"
        ]
    }
  ]
}
```

Example: Allow another account

The following repository policy allows a specific account to push images.

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

The following repository policy allows some IAM users to pull images (*pull-user-1* and *pull-user-2*) while providing full access to another (*admin-user*).

Note
For more complicated repository policies that are not currently supported in the AWS Management Console, you can apply the policy with the **set-repository-policy** AWS CLI command.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "AllowPull",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id:user/pull-user-1",
                    "arn:aws:iam::account-id:user/pull-user-2"
                ]
            },
            "Action": [
```

```
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ]
        },
        {
            "Sid": "AllowAll",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:user/admin-user"
            },
            "Action": [
                "ecr:*"
            ]
        }
    ]
}
```

Example: Allow all AWS accounts to pull images

The following repository policy allows all AWS accounts to pull images.

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

Example: Deny all

The following repository policy denies all users the ability to pull images.

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

Example: Restricting access to specific IP addresses

The following example grants permissions to any user to perform any Amazon ECR operations when applied to a repository. However, the request must originate from the range of IP addresses specified in the condition.

The condition in this statement identifies the `54.240.143.*` range of allowed Internet Protocol version 4 (IPv4) IP addresses, with one exception: `54.240.143.188`.

The `Condition` block uses the `IpAddress` and `NotIpAddress` conditions and the `aws:SourceIp` condition key, which is an AWS-wide condition key. For more information about these condition keys, see AWS Global Condition Context Keys. The`aws:sourceIp` IPv4 values use the standard CIDR notation. For more information, see IP Address Condition Operators in the *IAM User Guide*.

```
{
    "Version": "2012-10-17",
    "Id": "ECRPolicyId1",
    "Statement": [
        {
            "Sid": "IPAllow",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "ecr:*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "54.240.143.188/32"
                },
                "IpAddress": {
                    "aws:SourceIp": "54.240.143.0/24"
                }
            }
        }
    ]
}
```

Example: Service-linked role

The following repository policy allows AWS CodeBuild access to the Amazon ECR API actions necessary for integration with that service. For more information, see Amazon ECR Sample for CodeBuild in the *AWS CodeBuild User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CodeBuildAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "codebuild.amazonaws.com"
            },
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ]
        }
    ]
}
```

Tagging an Amazon ECR repository

To help you manage your Amazon ECR repositories, you can optionally assign your own metadata to each repository in the form of *tags*. This topic describes tags and shows you how to create them.

Contents

Tag basics

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and an optional *value*, both of which you define.

Tags enable you to categorize your AWS resources in different ways, for example, by purpose, owner, or environment. This is useful when you have many resources of the same type—you can quickly identify a specific resource based on the tags you've assigned to it. For example, you could define a set of tags for your account's Amazon ECR repositories that helps you track each repo's owner.

We recommend that you devise a set of tag keys that meets your needs. Using a consistent set of tag keys makes it easier for you to manage your resources. You can search and filter the resources based on the tags you add.

Tags don't have any semantic meaning to Amazon ECR and are interpreted strictly as a string of characters. Also, tags are not automatically assigned to your resources. You can edit tag keys and values, and you can remove tags from a resource at any time. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value. If you delete a resource, any tags for the resource are also deleted.

You can work with tags using the AWS Management Console, the AWS CLI, and the Amazon ECR API.

If you're using AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to create, edit, or delete tags.

Tagging your resources

You can tag new or existing Amazon ECR repositories.

If you're using the Amazon ECR console, you can apply tags to new resources when they are created or existing resources by using the **Tags** option on the navigation pane at any time.

If you're using the Amazon ECR API, the AWS CLI, or an AWS SDK, you can apply tags to new repositories using the `tags` parameter on the CreateRepository API action or use the `TagResource` API action to apply tags to existing resources. For more information, see TagResource.

Additionally, if tags cannot be applied during repository creation, we roll back the repository creation process. This ensures that repositories are either created with tags or not created at all, and that no repositories are left untagged at any time. By tagging repositories at the time of creation, you can eliminate the need to run custom tagging scripts after repository creation.

Tag restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per repository – 50
- For each repository, each tag key must be unique, and each tag key can have only one value.

- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple services and resources, remember that other services may have restrictions on allowed characters. Generally allowed characters are: letters, numbers, and spaces representable in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case-sensitive.
- Don't use the `aws:` prefix for either keys or values; it's reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags per resource limit.

Tagging your resources for billing

The tags you add to your Amazon ECR repositories are helpful when reviewing cost allocation after enabling them in your Cost & Usage Report. For more information, see Amazon ECR usage reports (p. 85).

To see the cost of your combined resources, you can organize your billing information based on resources that have the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information about setting up a cost allocation report with tags, see The Monthly Cost Allocation Report in the *AWS Billing and Cost Management User Guide*.

> **Note**
> If you've just enabled reporting, data for the current month is available for viewing after 24 hours.

Working with tags using the console

Using the Amazon ECR console, you can manage the tags associated with new or existing repositories.

When you select a specific repository in the Amazon ECR console, you can view the tags by selecting **Tags** in the navigation pane.

To add a tag to a repository

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, select the region to use.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the repository to view.
5. On the **Repositories : *repository_name*** page, select **Tags** from the navigation pane.
6. On the **Tags** page, select **Add tags**, **Add tag**.
7. On the **Edit Tags** page, specify the key and value for each tag, and then choose **Save**.

To delete a tag from an individual resource

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, select the region to use.
3. On the **Repositories** page, choose the repository to view.
4. On the **Repositories : *repository_name*** page, select **Tags** from the navigation pane.
5. On the **Tags** page, select **Edit**.
6. On the **Edit Tags** page, select **Remove** for each tag you want to delete, and choose **Save**.

Working with tags using the AWS CLI or API

Use the following to add, update, list, and delete the tags for your resources. The corresponding documentation provides examples.

Tagging Support for Amazon ECR Resources

Task	AWS CLI	API Action
Add or overwrite one or more tags.	tag-resource	TagResource
Delete one or more tags.	untag-resource	UntagResource

The following examples show how to manage tags using the AWS CLI.

Example 1: Tag an existing repository

The following command tags an existing repository.

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

Example 2: Tag an existing repository with multiple tags

The following command tags an existing repository.

```
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
```

Example 3: Untag an existing repository

The following command deletes a tag from an existing repository.

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

Example 4: List tags for a repository

The following command lists the tags associated with an existing repository.

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

Example 5: Create a repository and apply a tag

The following command creates a repository named `test-repo` and adds a tag with key `team` and value `devs`.

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

Images

Amazon Elastic Container Registry (Amazon ECR) stores Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts in repositories. You can use the Docker CLI or your preferred client to push and pull images to and from your repositories.

> **Important**
> Amazon ECR requires that users have permission to make calls to the `ecr:GetAuthorizationToken` API through an IAM policy before they can authenticate to a registry and push or pull any images from any Amazon ECR repository. Amazon ECR provides several managed IAM policies to control user access at varying levels; for more information, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).

Topics

Pushing an image

You can push your Docker images to an Amazon ECR repository with the **docker push** command.

> **Important**
> Amazon ECR requires that users have permission to make calls to the `ecr:GetAuthorizationToken` API through an IAM policy before they can authenticate to a registry and push or pull any images from any Amazon ECR repository. Amazon ECR provides several managed IAM policies to control user access at varying levels; for more information, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).

Amazon ECR also supports creating and pushing Docker manifest lists which are used for multi-architecture images. Each image referenced in a manifest list must already be pushed to your repository. For more information, see Pushing a multi-architecture image (p. 29).

To push a Docker image to an Amazon ECR repository

1. Authenticate your Docker client to the Amazon ECR registry to which you intend to push your image. Authentication tokens must be obtained for each registry used, and the tokens are valid for 12 hours. For more information, see Registry authentication (p. 13).

2. If your image repository does not exist in the registry you intend to push to yet, create it. For more information, see Creating a repository (p. 16).
3. Identify the image to push. Run the **docker images** command to list the images on your system.

```
docker images
```

   You can identify an image with the *repository:tag* value or the image ID in the resulting command output.
4. Tag your image with the Amazon ECR registry, repository, and optional image tag name combination to use. The registry format is *aws_account_id*.dkr.ecr.*region*.amazonaws.com. The repository name should match the repository that you created for your image. If you omit the image tag, we assume that the tag is `latest`.

   The following example tags an image with the ID *e9ae3c220b23* as *aws_account_id*.dkr.ecr.*region*.amazonaws.com/my-web-app

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

5. Push the image using the **docker push** command:

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

6. (Optional) Apply any additional tags to your image and push those tags to Amazon ECR by repeating Step 4 (p. 29) and Step 5 (p. 29). You can apply up to 100 tags per image in Amazon ECR.

Pushing a multi-architecture image

Amazon ECR supports creating and pushing Docker manifest lists which are used for multi-architecture images. A *manifest list* is a list of images that is created by specifying one or more image names. Typically the manifest list is created from images that serve the same function but for different operating systems or architectures, but this is not required. For more information, see docker manifest.

> **Important**
> Your Docker CLI must have experimental features enabled to use this feature. For more information, see Experimental features.

A manifest list can be pulled or referenced in an Amazon ECS task definition or Amazon EKS pod spec like other Amazon ECR images.

The following steps can be used to create and push a Docker manifest list to an Amazon ECR repository. You must already have the images pushed to your repository to reference in the Docker manifest. For information on pushing an image, see Pushing an image (p. 28).

To push a multi-architecture Docker image to an Amazon ECR repository

1. Authenticate your Docker client to the Amazon ECR registry to which you intend to push your image. Authentication tokens must be obtained for each registry used, and the tokens are valid for 12 hours. For more information, see Registry authentication (p. 13).
2. List the images in your repository, confirming the image tags.

```
aws ecr describe-images --repository-name my-web-app
```

3. Create the Docker manifest list. The `manifest create` command verifies that the referenced images are already in your repository and creates the manifest locally.

```
docker manifest create aws_account_id.dkr.ecr.region.amazonaws.com/my-
web-app aws_account_id.dkr.ecr.region.amazonaws.com/my-web-
app:image_one_tag aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:image_two
```

4. (Optional) Inspect the Docker manifest list. This enables you to confirm the size and digest for each image manifest referenced in the manifest list.

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

5. Push the Docker manifest list to your Amazon ECR repository.

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

Pushing a Helm chart

Amazon ECR supports pushing Open Container Initiative (OCI) artifacts to your repositories. To display this functionality, use the following steps to push a Helm chart to Amazon ECR.

For more information about using your Amazon ECR hosted Helm charts with Amazon EKS, see Installing a Helm chart hosted on Amazon ECR with Amazon EKS (p. 53).

To push a Helm chart to an Amazon ECR repository

1. Install the Helm client version 3. For more information, see Installing Helm.
2. Enable OCI support in the Helm 3 client.

```
export HELM_EXPERIMENTAL_OCI=1
```

3. Create a repository to store your Helm chart. For more information, see Creating a repository (p. 16).

```
aws ecr create-repository \
     --repository-name artifact-test \
     --region us-west-2
```

4. Authenticate your Helm client to the Amazon ECR registry to which you intend to push your Helm chart. Authentication tokens must be obtained for each registry used, and the tokens are valid for 12 hours. For more information, see Registry authentication (p. 13).

```
aws ecr get-login-password \
     --region us-west-2 | helm registry login \
     --username AWS \
     --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

5. Use the following steps to create a test Helm chart. For more information, see Helm Docs - Getting Started.

   a. Create a directory named `helm-tutorial` to work in.

```
mkdir helm-tutorial
cd helm-tutorial
```

   b. Create a Helm chart named `mychart` and clear the contents of the `templates` directory.

```
helm create mychart
```

```
rm -rf ./mychart/templates/*
```

c. Create a ConfigMap in the `templates` folder.

```
cd mychart/templates
cat <<EOF > configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mychart-configmap
data:
  myvalue: "Hello World"
EOF
```

6. Save the chart locally and create an alias for the chart with your registry URI.

```
cd ..
helm chart save . mychart
helm chart save . aws_account_id.dkr.ecr.us-west-2.amazonaws.com/artifact-test:mychart
```

7. Identify the Helm chart to push. Run the **helm chart list** command to list the Helm charts on your system.

```
helm chart list
```

The output should look similar to this:

```
REF                                                             NAME    VERSION DIGEST
 SIZE    CREATED
aws_account_id.dkr.ecr.us-west-2.amazonaws.com/artifact-tes..   mychart 0.1.0   30e0a03
 3.6 KiB 14 seconds
mychart                                                         mychart 0.1.0   ba3e62a 3.6
 KiB About a minute
```

8. Push the Helm chart using the **helm chart push** command:

```
helm chart push aws_account_id.dkr.ecr.region.amazonaws.com/artifact-test:mychart
```

9. Describe your Helm chart.

```
aws ecr describe-images \
     --repository-name artifact-test \
     --region us-west-2
```

In the output, verify the `artifactMediaType` parameter indicates the proper artifact type.

```
{
    "imageDetails": [
        {
            "registryId": "aws_account_id",
            "repositoryName": "artifact-test",
            "imageDigest":
 "sha256:f23ab9dc0fda33175e465bd694a5f4cade93eaf62715fa9390d9fEXAMPLE",
            "imageTags": [
                "mychart"
            ],
            "imageSizeInBytes": 3714,
            "imagePushedAt": 1597433021.0,
            "imageManifestMediaType": "application/vnd.oci.image.manifest.v1+json",
            "artifactMediaType": "application/vnd.cncf.helm.config.v1+json"
```

```
        }
    ]
}
```

Pulling an image

If you would like to run a Docker image that is available in Amazon ECR, you can pull it to your local environment with the **docker pull** command. You can do this from either your default registry or from a registry associated with another AWS account. To use an Amazon ECR image in an Amazon ECS task definition, see Using Amazon ECR images with Amazon ECS (p. 51).

Important
Amazon ECR requires that users have permission to make calls to the `ecr:GetAuthorizationToken` API through an IAM policy before they can authenticate to a registry and push or pull any images from any Amazon ECR repository. Amazon ECR provides several managed IAM policies to control user access at varying levels; for more information, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).

To pull a Docker image from an Amazon ECR repository

1. Authenticate your Docker client to the Amazon ECR registry that you intend to pull your image from. Authentication tokens must be obtained for each registry used, and the tokens are valid for 12 hours. For more information, see Registry authentication (p. 13).
2. (Optional) Identify the image to pull.
   - You can list the repositories in a registry with the **aws ecr describe-repositories** command:

     ```
     aws ecr describe-repositories
     ```

     The example registry above has a repository called `amazonlinux`.
   - You can describe the images within a repository with the **aws ecr describe-images** command:

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

     The example repository above has an image tagged as `latest` and `2016.09`, with the image digest `sha256:f1d4ae3f7261a72e98c6ebefe9985cf10a0ea5bd762585a43e0700ed99863807`.
3. Pull the image using the **docker pull** command. The image name format should be *registry*/*repository*[:*tag*] to pull by tag, or *registry*/*repository*[@*digest*] to pull by digest.

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

   Important
   If you receive a *repository-url* `not found: does not exist or no pull access` error, you may need to authenticate your Docker client with Amazon ECR. For more information, see Registry authentication (p. 13).

Deleting an image

If you are done using an image, you can delete it from your repository. You can delete an image using the AWS Management Console, or the AWS CLI.

Note
If you are done with a repository, you can delete the entire repository and all of the images within it. For more information, see Deleting a repository (p. 18).

To delete an image with the AWS Management Console

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region that contains the image to delete.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the repository that contains the image to delete.
5. On the **Repositories:** ***repository_name*** page, select the box to the left of the image to delete and choose **Delete**.
6. In the **Delete image(s)** dialog box, verify that the selected images should be deleted and choose **Delete**.

To delete an image with the AWS CLI

1. List the images in your repository so that you can identify them by image tag or digest.

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

2. (Optional) Delete any unwanted tags for the image by specifying the tag of the image you want to delete.

   Note
   When you delete the last tag for an image, the image is deleted.

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

3. Delete the image by specifying the digest of the image to delete.

   Note
   When you delete an image by referencing its digest, the image and all of its tags are deleted.

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

Retagging an image

With Docker Image Manifest V2 Schema 2 images, you can use the `--image-tag` option of the **put-image** command to retag an existing image. You can retag without pulling or pushing the image with Docker. For larger images, this process saves a considerable amount of network bandwidth and time required to retag an image.

To retag an image (AWS CLI)

To retag an image with the AWS CLI

1. Use the **batch-get-image** command to get the image manifest for the image to retag and write it to an environment variable. In this example, the manifest for an image with the tag, *latest*, in the repository, *amazonlinux*, is written to the environment variable, *MANIFEST*.

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

2. Use the `--image-tag` option of the **put-image** command to put the image manifest to Amazon ECR with a new tag. In this example, the image is tagged as *`2017.03`*.

   Note
   If the `--image-tag` option is not available in your version of the AWS CLI, upgrade to the latest version. For more information, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

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

3. Verify that your new image tag is attached to your image. In the output below, the image has the tags `latest` and `2017.03`.

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

   Output:

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

To retag an image (AWS Tools for Windows PowerShell)

To retag an image with the AWS Tools for Windows PowerShell

1. Use the **Get-ECRImageBatch** cmdlet to get the description of the image to retag and write it to an environment variable. In this example, an image with the tag, *`latest`*, in the repository, *`amazonlinux`*, is written to the environment variable, *`$Image`*.

   Note
   If you don't have the **Get-ECRImageBatch** cmdlet available on your system, see Setting up the AWS Tools for Windows PowerShell in the *AWS Tools for Windows PowerShell User Guide*.

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

2. Write the manifest of the image to the *`$Manifest`* environment variable.

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

3. Use the `-ImageTag` option of the **Write-ECRImage** cmdlet to put the image manifest to Amazon ECR with a new tag. In this example, the image is tagged as *2017.09*.

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

4. Verify that your new image tag is attached to your image. In the output below, the image has the tags `latest` and `2017.09`.

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

Output:

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

Lifecycle policies

Amazon ECR lifecycle policies enable you to specify the lifecycle management of images in a repository. A lifecycle policy is a set of one or more rules, where each rule defines an action for Amazon ECR. The actions apply to images that contain tags prefixed with the given strings. This allows the automation of cleaning up unused images, for example expiring images based on age or count. You should expect that after creating a lifecycle policy the affected images are expired within 24 hours.

Topics

Lifecycle policy template

The contents of your lifecycle policy is evaluated before being associated with a repository. The following is the JSON syntax template for the lifecycle policy. For lifecycle policy examples, see Examples of lifecycle policies (p. 40).

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

```
            }
        }
    ]
}
```

Note
The `tagPrefixList` parameter is only used if `tagStatus` is `tagged`. The `countUnit` parameter is only used if `countType` is `sinceImagePushed`. The `countNumber` parameter is only used if `countType` is set to `imageCountMoreThan`.

Lifecycle policy parameters

Lifecycle policies are split into the following parts:

Topics

Rule priority

`rulePriority`

Type: integer

Required: yes

Sets the order in which rules are evaluated, lowest to highest. A lifecycle policy rule with a priority of `1` will be acted upon first, a rule with priority of `2` will be next, and so on. When you add rules to a lifecycle policy, you must give them each a unique value for `rulePriority`. Values do not need to be sequential across rules in a policy. A rule with a `tagStatus` value of `any` must have the highest value for `rulePriority` and be evaluated last.

Description

`description`

Type: string

Required: no

(Optional) Describes the purpose of a rule within a lifecycle policy.

Tag status

`tagStatus`

Type: string

Required: yes

Determines whether the lifecycle policy rule that you are adding specifies a tag for an image. Acceptable options are `tagged`, `untagged`, or `any`. If you specify `any`, then all images have the rule applied to them. If you specify `tagged`, then you must also specify a `tagPrefixList` value. If you specify `untagged`, then you must omit `tagPrefixList`.

Tag prefix list

`tagPrefixList`

Type: list[string]

Required: yes, only if `tagStatus` is set to tagged

Only used if you specified `"tagStatus": "tagged"`. You must specify a comma-separated list of image tag prefixes on which to take action with your lifecycle policy. For example, if your images are tagged as `prod`, `prod1`, `prod2`, and so on, you would use the tag prefix `prod` to specify all of them. If you specify multiple tags, only the images with all specified tags are selected.

Count type

`countType`

Type: string

Required: yes

Specify a count type to apply to the images.

If `countType` is set to `imageCountMoreThan`, you also specify `countNumber` to create a rule that sets a limit on the number of images that exist in your repository. If `countType` is set to `sinceImagePushed`, you also specify `countUnit` and `countNumber` to specify a time limit on the images that exist in your repository.

Count unit

`countUnit`

Type: string

Required: yes, only if `countType` is set to `sinceImagePushed`

Specify a count unit of `days` to indicate that as the unit of time, in addition to `countNumber`, which is the number of days.

This should only be specified when `countType` is `sinceImagePushed`; an error will occur if you specify a count unit when `countType` is any other value.

Count number

`countNumber`

Type: integer

Required: yes

Specify a count number. Acceptable values are positive integers (`0` is not an accepted value).

If the `countType` used is `imageCountMoreThan`, then the value is the maximum number of images that you want to retain in your repository. If the `countType` used is `sinceImagePushed`, then the value is the maximum age limit for your images.

Action

`type`

Type: string

Required: yes

Specify an action type. The supported value is `expire`.

Lifecycle policy evaluation rules

The lifecycle policy evaluator is responsible for parsing the plaintext JSON and applying it to the images in the specified repository. The following rules should be noted when creating a lifecycle policy:

- An image is expired by exactly one or zero rules.
- An image that matches the tagging requirements of a rule cannot be expired by a rule with a lower priority.
- Rules can never mark images that are marked by higher priority rules, but can still identify them as if they haven't been expired.
- The set of rules must contain a unique set of tag prefixes.
- Only one rule is allowed to select untagged images.
- Expiration is always ordered by `pushed_at_time`, and always expires older images before newer ones.
- When using the `tagPrefixList`, an image is successfully matched if *all* of the tags in the `tagPrefixList` value are matched against any of the image's tags.
- With `countType = imageCountMoreThan`, images are sorted from youngest to oldest based on `pushed_at_time` and then all images greater than the specified count are expired.
- With `countType = sinceImagePushed`, all images whose `pushed_at_time` is older than the specified number of days based on `countNumber` are expired.

Creating a lifecycle policy preview

A lifecycle policy preview allows you to see the impact of a lifecycle policy on an image repository before you execute it. The following procedure shows you how to create a lifecycle policy preview.

To create a lifecycle policy preview using the console

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region that contains the repository on which to perform a lifecycle policy preview.
3. In the navigation pane, choose **Repositories** and select a repository.
4. On the **Repositories: *`repository_name`*** page, in the navigation pane choose **Lifecycle Policy**.
5. On the **Repositories: *`repository_name`*: Lifecycle policy** page, choose **Edit test rules**, **Create rule**.
6. Enter the following details for your lifecycle policy rule:

    a. For **Rule priority**, type a number for the rule priority.

b. For **Rule description**, type a description for the lifecycle policy rule.
c. For **Image status**, choose **Tagged**, **Untagged**, or **Any**.
d. If you specified `Tagged` for **Image status**, then for **Tag prefixes**, you can optionally specify a list of image tags on which to take action with your lifecycle policy. If you specified `Untagged`, this field must be empty.
e. For **Match criteria**, choose values for **Since image pushed** or **Image count more than** (if applicable).

7. Choose **Save**.
8. Create additional lifecycle policy rules by repeating steps 5–7.
9. To run the lifecycle policy preview, choose **Save and run test**.
10. Under **Image matches for test lifecycle rules**, review the impact of your lifecycle policy preview.
11. If you are satisfied with the preview results, choose **Apply as lifecycle policy** to create a lifecycle policy with the specified rules.

Note
You should expect that after creating a lifecycle policy, the affected images are expired within 24 hours.

Creating a lifecycle policy

A lifecycle policy allows you to create a set of rules that expire unused repository images. The following procedure shows you how to create a lifecycle policy. You should expect that after creating a lifecycle policy, the affected images are expired within 24 hours.

To create a lifecycle policy (AWS CLI)

To create a lifecycle policy using the AWS CLI

1. Obtain the ID of the repository for which to create the lifecycle policy:

```
aws ecr describe-repositories
```

2. Create a lifecycle policy:

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

To create a lifecycle policy (AWS Management Console)

To create a lifecycle policy using the console

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region that contains the repository for which to create a lifecycle policy.
3. In the navigation pane, choose **Repositories** and select a repository.
4. On the **Repositories: *repository_name*** page, in the navigation pane choose **Lifecycle Policy**.
5. On the **Repositories: *repository_name*: Lifecycle policy** page, choose **Create rule**.
6. Enter the following details for your lifecycle policy rule:

a. For **Rule priority**, type a number for the rule priority.
b. For **Rule description**, type a description for the lifecycle policy rule.

c. For **Image status**, choose **Tagged**, **Untagged**, or **Any**.

d. If you specified `Tagged` for **Image status**, then for **Tag prefixes**, you can optionally specify a list of image tags on which to take action with your lifecycle policy. If you specified `Untagged`, this field must be empty.

e. For **Match criteria**, choose values for **Since image pushed** or **Image count more than** (if applicable).

7. Choose **Save**.

Examples of lifecycle policies

The following are example lifecycle policies, showing the syntax.

Topics

Filtering on image age

The following example shows the lifecycle policy syntax for a policy that expires untagged images older than `14` days:

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

Filtering on image count

The following example shows the lifecycle policy syntax for a policy that keeps only one untagged image and expires all others:

```
{
    "rules": [
        {
            "rulePriority": 1,
            "description": "Keep only one untagged image, expire all others",
            "selection": {
                "tagStatus": "untagged",
```

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

Filtering on multiple rules

The following examples use multiple rules in a lifecycle policy. An example repository and lifecycle policy are given along with an explanation of the outcome.

Example A

Repository contents:

- 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

Lifecycle policy text:

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

The logic of this lifecycle policy would be:

- Rule 1 identifies images tagged with prefix `prod`. It should mark images, starting with the oldest, until there is one or fewer images remaining that match. It marks Image A for expiration.

- Rule 2 identifies images tagged with prefix `beta`. It should mark images, starting with the oldest, until there is one or fewer images remaining that match. It marks both Image A and Image B for expiration. However, Image A has already been seen by Rule 1 and if Image B were expired it would violate Rule 1 and thus is skipped.
- Result: Image A is expired.

Example B

This is the same repository as the previous example but the rule priority order is changed to illustrate the outcome.

Repository contents:

- 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

Lifecycle policy text:

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

The logic of this lifecycle policy would be:

- Rule 1 identifies images tagged with `beta`. It should mark images, starting with the oldest, until there is one or fewer images remaining that match. It sees all three images and would mark Image A and Image B for expiration.
- Rule 2 identifies images tagged with `prod`. It should mark images, starting with the oldest, until there is one or fewer images remaining that match. It would see no images because all available images were already seen by Rule 1 and thus would mark no additional images.

- Result: Images A and B are expired.

Filtering on multiple tags in a single rule

The following examples specify the lifecycle policy syntax for multiple tag prefixes in a single rule. An example repository and lifecycle policy are given along with an explanation of the outcome.

Example A

When multiple tag prefixes are specified on a single rule, images must match all listed tag prefixes.

Repository contents:

- 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"
            }
        }
    ]
}
```

The logic of this lifecycle policy would be:

- Rule 1 identifies images tagged with `alpha` and `beta`. It sees images C and F. It should mark images that are older than five days, which would be Image C.
- Result: Image C is expired.

Example B

The following example illustrates that tags are not exclusive.

Repository contents:

- 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"
            }
        }
    ]
}
```

The logic of this lifecycle policy would be:

- Rule 1 identifies images tagged with `alpha` and `beta`. It sees all images. It should mark images, starting with the oldest, until there is one or fewer images remaining that match. It marks image A and B for expiration.
- Result: Images A and B are expired.

Filtering on all images

The following lifecycle policy examples specify all images with different filters. An example repository and lifecycle policy are given along with an explanation of the outcome.

Example A

The following shows the lifecycle policy syntax for a policy that applies to all rules but keeps only one image and expires all others.

Repository contents:

- 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"
            }
        }
    ]
```

```
}
```

The logic of this lifecycle policy would be:

- Rule 1 identifies all images. It sees images A, B, C, and D. It should expire all images other than the newest one. It marks images A, B, and C for expiration.
- Result: Images A, B, and C are expired.

Example B

The following example illustrates a lifecycle policy that combines all the rule types in a single policy.

Repository contents:

- Image A, Taglist: ["alpha-1", "beta-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": "days",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
        },
        {
            "rulePriority": 3,
            "description": "Rule 3",
            "selection": {
                "tagStatus": "any",
                "countType": "imageCountMoreThan",
                "countNumber": 1
            },
            "action": {
                "type": "expire"
            }
```

```
        }
    ]
}
```

The logic of this lifecycle policy would be:

- Rule 1 identifies images tagged with `alpha`. It identifies images A and C. It should keep the newest image and mark the rest for expiration. It marks image A for expiration.
- Rule 2 identifies untagged images. It identifies images B and E. It should mark all images older than one day for expiration. It marks image B for expiration.
- Rule 3 identifies all images. It identifies images A, B, C, D, and E. It should keep the newest image and mark the rest for expiration. However, it can't mark images A, B, C, or E because they were identified by higher priority rules. It marks image D for expiration.
- Result: Images A, B, and D are expired.

Image tag mutability

You can configure a repository to be immutable to prevent image tags from being overwritten. Once the repository is configured for immutable tags, an `ImageTagAlreadyExistsException` error will be returned if you attempt to push an image with a tag that already exists in the repository.

You can use the AWS Management Console and AWS CLI tools to set image tag mutability for either a new repository during creation or for an existing repository at any time. For console steps, see Creating a repository (p. 16) and Editing a repository (p. 18).

To create a repository with immutable tags configured

Use one of the following commands to create a new image repository with immutable tags configured.

- create-repository (AWS CLI)

```
aws ecr create-repository --repository-name name --image-tag-mutability IMMUTABLE --
region us-east-2
```

- New-ECRRepository (AWS Tools for Windows PowerShell)

```
New-ECRRepository -RepositoryName name -ImageTagMutability IMMUTABLE -Region us-east-2 -
Force
```

To update the image tag mutability settings for an existing repository

Use one of the following commands to update the image tag mutability settings for an existing repository.

- put-image-tag-mutability (AWS CLI)

```
aws ecr put-image-tag-mutability --repository-name name --image-tag-mutability IMMUTABLE
 --region us-east-2
```

- Write-ECRImageTagMutability (AWS Tools for Windows PowerShell)

```
Write-ECRImageTagMutability -RepositoryName name -ImageTagMutability IMMUTABLE -
Region us-east-2 -Force
```

Image scanning

Amazon ECR image scanning helps in identifying software vulnerabilities in your container images. Amazon ECR uses the Common Vulnerabilities and Exposures (CVEs) database from the open source Clair project and provides you with a list of scan findings. You can review the scan findings for information about the security of the container images that are being deployed. For more information about Clair, see Clair on GitHub.

Amazon ECR uses the severity for a CVE from the upstream distribution source if available, otherwise we use the Common Vulnerability Scoring System (CVSS) score. The CVSS score can be used to obtain the NVD vulnerability severity rating. For more information, see NVD Vulnerability Severity Ratings.

You can manually scan container images stored in Amazon ECR, or you can configure your repositories to scan images when you push them to a repository. The last completed image scan findings can be retrieved for each image. Amazon ECR sends an event to Amazon EventBridge (formerly called CloudWatch Events) when an image scan is completed. For more information, see Amazon ECR events and EventBridge (p. 85).

For troubleshooting details for some common issues when scanning images, see Troubleshooting Image Scanning Issues (p. 106).

Topics

Configuring a repository to scan on push

You can configure the image scan settings either for a new repository during creation or for an existing repository. When **scan on push** is enabled, images are scanned after being pushed to a repository. If **scan on push** is disabled on a repository then you must manually start each image scan to get the scan results.

Topics

Creating a new repository to scan on push

When a new repository is configured to **scan on push**, all new images pushed to the repository will be scanned. Results from the last completed image scan can then be retrieved. For more information, see Retrieving image scan findings (p. 49).

For AWS Management Console steps, see Creating a repository (p. 16).

To create a repository configured for scan on push (AWS CLI)

Use the following command to create a new repository with image **scan on push** configured.

- create-repository (AWS CLI)

```
aws ecr create-repository --repository-name name --image-scanning-configuration
 scanOnPush=true --region us-east-2
```

To create a repository configured for scan on push (AWS Tools for Windows PowerShell)

Use the following command to create a new repository with image **scan on push** configured.

- New-ECRRepository (AWS Tools for Windows PowerShell)

```
New-ECRRepository -RepositoryName name -ImageScanningConfiguration_ScanOnPush true -
Region us-east-2 -Force
```

Configure an existing repository to scan on push

Your existing repositories can be configured to scan images when you push them to a repository. This setting will apply to future image pushes. Results from the last completed image scan can then be retrieved. For more information, see Retrieving image scan findings (p. 49).

For AWS Management Console steps, see Editing a repository (p. 18).

To edit the settings of an existing repository (AWS CLI)

Use the following command to edit the image scanning settings of an existing repository.

- put-image-scanning-configuration (AWS CLI)

```
aws ecr put-image-scanning-configuration --repository-name name --image-scanning-
configuration scanOnPush=true --region us-east-2
```

> **Note**
> To disable image **scan on push** for a repository, specify `scanOnPush=false`.

To edit the settings of an existing repository (AWS Tools for Windows PowerShell)

Use the following command to edit the image scanning settings of an existing repository.

- New-ECRRepository (AWS Tools for Windows PowerShell)

```
Write-ECRImageScanningConfiguration -RepositoryName name -
ImageScanningConfiguration_ScanOnPush true -Region us-east-2 -Force
```

Manually scanning an image

You can start image scans manually when you want to scan images in repositories that are not configured to **scan on push**. An image can only be scanned once per day. This limit includes the initial **scan on push**, if enabled, and any manual scans.

For troubleshooting details for some common issues when scanning images, see Troubleshooting Image Scanning Issues (p. 106).

To start a manual scan of an image (console)

Use the following steps to start a manual image scan using the AWS Management Console.

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region to create your repository in.
3. In the navigation pane, choose **Repositories**.

4. On the **Repositories** page, choose the repository that contains the image to scan.
5. On the **Images** page, select the image to scan and then choose **Scan**.

To start a manual scan of an image (AWS CLI)

Use the following AWS CLI command to start a manual scan of an image. You can specify an image using the `imageTag` or `imageDigest`, both of which can be obtained using the list-images CLI command.

- start-image-scan (AWS CLI)

  The following example uses an image tag.

```
aws ecr start-image-scan --repository-name name --image-id imageTag=tag_name --region us-east-2
```

  The following example uses an image digest.

```
aws ecr start-image-scan --repository-name name --image-id imageDigest=sha256_hash --region us-east-2
```

To start a manual scan of an image (AWS Tools for Windows PowerShell)

Use the following AWS Tools for Windows PowerShell command to start a manual scan of an image. You can specify an image using the `ImageId_ImageTag` or `ImageId_ImageDigest`, both of which can be obtained using the Get-ECRImage CLI command.

- Get-ECRImageScanFinding (AWS Tools for Windows PowerShell)

  The following example uses an image tag.

```
Start-ECRImageScan -RepositoryName name -ImageId_ImageTag tag_name -Region us-east-2 -Force
```

  The following example uses an image digest.

```
Start-ECRImageScan -RepositoryName name -ImageId_ImageDigest sha256_hash -Region us-east-2 -Force
```

Retrieving image scan findings

You can retrieve the scan findings for the last completed image scan. The findings list by severity the software vulnerabilities that were discovered, based on the Common Vulnerabilities and Exposures (CVEs) database.

For troubleshooting details for some common issues when scanning images, see Troubleshooting Image Scanning Issues (p. 106).

To retrieve image scan findings (console)

Use the following steps to retrieve image scan findings using the AWS Management Console.

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region to create your repository in.

3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the repository that contains the image to retrieve the scan findings for.
5. On the **Images** page, under the **Vulnerabilities** column, select **Details** for the image to retrieve the scan findings for.

To retrieve image scan findings (AWS CLI)

Use the following AWS CLI command to retrieve image scan findings using the AWS CLI. You can specify an image using the `imageTag` or `imageDigest`, both of which can be obtained using the list-images CLI command.

- describe-image-scan-findings (AWS CLI)

  The following example uses an image tag.

```
aws ecr describe-image-scan-findings --repository-name name --image-id imageTag=tag_name
 --region us-east-2
```

  The following example uses an image digest.

```
aws ecr describe-image-scan-findings --repository-name name --image-id
 imageDigest=sha256_hash --region us-east-2
```

To retrieve image scan findings (AWS Tools for Windows PowerShell)

Use the following AWS Tools for Windows PowerShell command to retrieve image scan findings. You can specify an image using the `ImageId_ImageTag` or `ImageId_ImageDigest`, both of which can be obtained using the Get-ECRImage CLI command.

- Get-ECRImageScanFinding (AWS Tools for Windows PowerShell)

  The following example uses an image tag.

```
Get-ECRImageScanFinding -RepositoryName name -ImageId_ImageTag tag_name -Region us-east-2
```

  The following example uses an image digest.

```
Get-ECRImageScanFinding -RepositoryName name -ImageId_ImageDigest sha256_hash -Region us-
east-2
```

Container image manifest formats

Amazon ECR supports the following container image manifest formats:

- Docker Image Manifest V2 Schema 1 (used with Docker version 1.9 and older)
- Docker Image Manifest V2 Schema 2 (used with Docker version 1.10 and newer)
- Open Container Initiative (OCI) Specifications (v1.0 and up)

Support for Docker Image Manifest V2 Schema 2 provides the following functionality:

- The ability to use multiple tags per image.
- Support for storing Windows container images. For more information, see Pushing Windows Images to Amazon ECR in the *Amazon Elastic Container Service Developer Guide*.

Amazon ECR image manifest conversion

When you push and pull images to and from Amazon ECR, your container engine client (for example, Docker) communicates with the registry to agree on a manifest format that is understood by the client and the registry to use for the image.

When you push an image to Amazon ECR with Docker version 1.9 or older, the image manifest format is stored as Docker Image Manifest V2 Schema 1. When you push an image to Amazon ECR with Docker version 1.10 or newer, the image manifest format is stored as Docker Image Manifest V2 Schema 2.

When you pull an image from Amazon ECR *by tag*, Amazon ECR returns the image manifest format that is stored in the repository. The format is returned only if that format is understood by the client. If the stored image manifest format is not understood by the client, Amazon ECR converts the image manifest into a format that is understood by the client. For example, if a Docker 1.9 client requests an image manifest that is stored as Docker Image Manifest V2 Schema 2, Amazon ECR returns the manifest in the Docker Image Manifest V2 Schema 1 format. The table below describes the available conversions supported by Amazon ECR when an image is pulled *by tag*:

Schema requested by client	Pushed to ECR as V2, schema 1	Pushed to ECR as V2, schema 2	Pushed to ECR as OCI
V2, schema 1	No translation required	Translated to V2, schema 1	Translated to V2, schema 1
V2, schema 2	No translation available, client falls back to V2, schema 1	No translation required	Translated to V2, schema 2
OCI	No translation available	Translated to OCI	No translation required

Important
If you pull an image *by digest*, there is no translation available; your client must understand the image manifest format that is stored in Amazon ECR. If you request a Docker Image Manifest V2 Schema 2 image by digest on a Docker 1.9 or older client, the image pull fails. For more information, see Registry compatibility in the Docker documentation.
In this example, if you request the same image *by tag*, Amazon ECR translates the image manifest into a format that the client can understand. The image pull succeeds.

Using Amazon ECR images with Amazon ECS

You can use your container images hosted in Amazon ECR in your Amazon ECS task definitions, but you need to satisfy the following prerequisites.

- When using the EC2 launch type for your Amazon ECS tasks, your container instances must be using at least version 1.7.0 of the Amazon ECS container agent. The latest version of the Amazon ECS–optimized AMI supports Amazon ECR images in task definitions. For more information, including the latest Amazon ECS–optimized AMI IDs, see Amazon ECS-optimized AMI versions in the *Amazon Elastic Container Service Developer Guide*.

- The Amazon ECS container instance IAM role (`ecsInstanceRole`) that you use must contain the following IAM policy permissions for Amazon ECR.

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

  If you use the **AmazonEC2ContainerServiceforEC2Role** managed policy, then your container instance IAM role has the proper permissions. To check that your role supports Amazon ECR, see Amazon ECS container instance IAM role in the *Amazon Elastic Container Service Developer Guide*.
- In your Amazon ECS task definitions, make sure that you are using the full `registry/repository:tag` naming for your Amazon ECR images. For example, *aws_account_id*`.dkr.ecr.`*region*`.amazonaws.com/`*my-web-app*`:`*latest*.

  The following task definition snippet shows the syntax you would use to specify a container image hosted in Amazon ECR in your Amazon ECS task definition.

```
{
    "family": "task-definition-name",
    ...
    "containerDefinitions": [
        {
            "name": "container-name",
            "image": "aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest",
            ...
        }
    ],
    ...
}
```

Using Amazon ECR Images with Amazon EKS

You can use your Amazon ECR images with Amazon EKS, but you need to satisfy the following prerequisites.

- The Amazon EKS worker node IAM role (`NodeInstanceRole`) that you use with your worker nodes must possess the following IAM policy permissions for Amazon ECR.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
```

```
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

Note
If you used `eksctl` or the AWS CloudFormation templates in Getting Started with Amazon EKS to create your cluster and worker node groups, these IAM permissions are applied to your worker node IAM role by default.

- When referencing an image from Amazon ECR, you must use the full `registry/repository:tag` naming for the image. For example, *`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`:`*`latest`*.

Installing a Helm chart hosted on Amazon ECR with Amazon EKS

Your Helm charts hosted in Amazon ECR can be installed on your Amazon EKS clusters. The following steps demonstrate this.

Prerequisites

Before you begin, ensure the following steps have been completed.

- Install the Helm client version 3. For more information, see Installing Helm.
- You have pushed a Helm chart to your Amazon ECR repository. For more information, see Pushing a Helm chart (p. 30).
- You have configured `kubectl` to work with Amazon EKS. For more information, see Create a `kubeconfig` for Amazon EKS in the Amazon EKS User Guide. If the following commands succeeds for your cluster, you're properly configured.

```
kubectl get svc
```

Install an Amazon ECR hosted Helm chart to an Amazon EKS cluster

1. Enable OCI support in the Helm 3 client.

   ```
   export HELM_EXPERIMENTAL_OCI=1
   ```

2. Authenticate your Helm client to the Amazon ECR registry that your Helm chart is hosted. Authentication tokens must be obtained for each registry used, and the tokens are valid for 12 hours. For more information, see Registry authentication (p. 13).

   ```
   aws ecr get-login-password \
        --region us-west-2 | helm registry login \
        --username AWS \
        --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
   ```

3. Pull your Helm chart to your local cache.

   ```
   helm chart pull aws_account_id.dkr.ecr.region.amazonaws.com/repository-name:mychart
   ```

4. Export the chart to a local directory. In this example, we use a directory named `charts`.

```
helm chart export aws_account_id.dkr.ecr.region.amazonaws.com/repository-name:mychart
 --destination ./charts
```

5. Install the chart.

```
helm install ecr-chart-demo ./mychart
```

   The output should look similar to this:

```
NAME: ecr-chart-demo
LAST DEPLOYED: Wed Sep  2 14:32:07 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
```

6. Verify the chart installation. The output will be a YAML representation of the Kubernetes resources deployed by the chart.

```
helm get manifest ecr-chart-demo
```

7. (Optional) See your Helm chart running in your Amazon EKS pod.

```
kubectl get pods --all-namespaces
```

8. When you are finished, you can remove the chart release from your cluster.

```
helm uninstall ecr-chart-demo
```

Amazon Linux container image

The Amazon Linux container image is built from the same software components that are included in the Amazon Linux AMI. It is available for use in any environment as a base image for Docker workloads. If you are already using the Amazon Linux AMI for applications in Amazon EC2, then you can easily containerize your applications with the Amazon Linux container image.

You can use the Amazon Linux container image in your local development environment and then push your application to the AWS Cloud using Amazon ECS. For more information, see Using Amazon ECR images with Amazon ECS (p. 51).

The Amazon Linux container image is available in Amazon ECR and on Docker Hub. Support for the Amazon Linux container image can be found by visiting the AWS developer forums.

To pull the Amazon Linux container image from Amazon ECR

1. Authenticate your Docker client to the Amazon Linux container image Amazon ECR registry. Authentication tokens are valid for 12 hours. For more information, see Registry authentication (p. 13).

   Note
   The **get-login-password** command is available in the AWS CLI starting with version `1.17.10`. For more information, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-
stdin 137112412989.dkr.ecr.us-east-1.amazonaws.com
```

Output:

```
Login succeeded
```

Important
If you receive an error, install or upgrade to 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*.

2. (Optional) You can list the images within the Amazon Linux repository with the **aws ecr list-images** command. The `latest` tag always corresponds with the latest Amazon Linux container image that is available.

```
aws ecr list-images --region us-east-1 --registry-id 137112412989 --repository-name
 amazonlinux
```

3. Pull the Amazon Linux container image using the **docker pull** command.

```
docker pull 137112412989.dkr.ecr.us-east-1.amazonaws.com/amazonlinux:latest
```

4. (Optional) Run the container locally.

```
docker run -it 137112412989.dkr.ecr.us-east-1.amazonaws.com/amazonlinux:latest /bin/
bash
```

To pull the Amazon Linux container image from Docker Hub

1. Pull the Amazon Linux container image using the **docker pull** command.

```
docker pull amazonlinux
```

2. (Optional) Run the container locally.

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

Security in Amazon Elastic Container Registry

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon ECR, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon ECR. The following topics show you how to configure Amazon ECR to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon ECR resources.

Topics

Identity and Access Management for Amazon Elastic Container Registry

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon ECR resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Using Tag-Based Access Control (p. 67)
- Troubleshooting Amazon Elastic Container Registry Identity and Access (p. 69)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon ECR.

Service user – If you use the Amazon ECR service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon ECR features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon ECR, see Troubleshooting Amazon Elastic Container Registry Identity and Access (p. 69).

Service administrator – If you're in charge of Amazon ECR resources at your company, you probably have full access to Amazon ECR. It's your job to determine which Amazon ECR features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon ECR, see How Amazon Elastic Container Registry Works with IAM (p. 60).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon ECR. To view example Amazon ECR identity-based policies that you can use in IAM, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don’t use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then

securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2

instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of

entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the *IAM User Guide*.

How Amazon Elastic Container Registry Works with IAM

Before you use IAM to manage access to Amazon ECR, you should understand what IAM features are available to use with Amazon ECR. To get a high-level view of how Amazon ECR and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

Amazon ECR Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon ECR supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon ECR use the following prefix before the action: `ecr:`. For example, to grant someone permission to create an Amazon ECR repository with the Amazon ECR `CreateRepository` API operation, you include the `ecr:CreateRepository` action in their policy. Policy statements must

include either an `Action` or `NotAction` element. Amazon ECR defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

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

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

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

To see a list of Amazon ECR actions, see Actions, Resources, and Condition Keys for Amazon Elastic Container Registry in the *IAM User Guide*.

Resources

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

An Amazon ECR repository resource has the following ARN:

```
arn:${Partition}:ecr:${Region}:${Account}:repository/${Repository-name}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `my-repo` repository in the `us-east-1` Region in your statement, use the following ARN:

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

To specify all repositories that belong to a specific account, use the wildcard (*):

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

To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "resource1",
      "resource2"
```

To see a list of Amazon ECR resource types and their ARNs, see Resources Defined by Amazon Elastic Container Registry in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Elastic Container Registry.

Condition Keys

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM Policy Elements: Variables and Tags in the *IAM User Guide*.

Amazon ECR defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Most Amazon ECR actions support the `aws:ResourceTag` and `ecr:ResourceTag` condition keys. For more information, see Using Tag-Based Access Control (p. 67).

To see a list of Amazon ECR condition keys, see Condition Keys Defined by Amazon Elastic Container Registry in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon Elastic Container Registry.

Examples

To view examples of Amazon ECR identity-based policies, see Amazon Elastic Container Registry Identity-Based Policy Examples (p. 65).

Amazon ECR Resource-Based Policies

Resource-based policies are JSON policy documents that specify what actions a specified principal can perform on an Amazon ECR resource and under what conditions. Amazon ECR supports resource-based permissions policies for Amazon ECR repositories. Resource-based policies let you grant usage permission to other accounts on a per-resource basis. You can also use a resource-based policy to allow an AWS service to access your Amazon ECR repositories.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, you must also grant the principal entity permission to access the resource. Grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

The Amazon ECR service supports only one type of resource-based policy called a *repository policy*, which is attached to a *repository*. This policy defines which principal entities (accounts, users, roles, and federated users) can perform actions on the repository.

To learn how to attach a resource-based policy to a repository, see Repository policies (p. 19).

Examples

To view examples of Amazon ECR resource-based policies, see Repository policy examples (p. 21),

Authorization Based on Amazon ECR Tags

You can attach tags to Amazon ECR resources or pass tags in a request to Amazon ECR. To control access based on tags, you provide tag information in the condition element of a policy using the `ecr:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys. For more information about tagging Amazon ECR resources, see Tagging an Amazon ECR repository (p. 24).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Using Tag-Based Access Control (p. 67).

Amazon ECR IAM Roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon ECR

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon ECR supports using temporary credentials.

Service-Linked Roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon ECR does not support service-linked roles.

Amazon ECR Managed Policies

Amazon ECR provides several managed policies that you can attach to IAM users or EC2 instances that allow differing levels of control over Amazon ECR resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own policies. For more information about each API operation mentioned in these policies, see Actions in the *Amazon Elastic Container Registry API Reference*.

Topics

AmazonEC2ContainerRegistryFullAccess

This managed policy is a starting point for customers who are looking to provide an IAM user or role with full administrator access to manage their use of Amazon ECR. The Amazon ECR Lifecycle Policies feature enables customers to specify the lifecycle management of images in a repository. Lifecycle policy events are reported as CloudTrail events, and Amazon ECR is integrated with AWS CloudTrail to display a customer's lifecycle policy events directly in the Amazon ECR console. The `AmazonEC2ContainerRegistryFullAccess` managed IAM policy includes the `cloudtrail:LookupEvents` permission to facilitate this behavior.

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

```
    ]
}
```

AmazonEC2ContainerRegistryPowerUser

This managed policy allows power user access to Amazon ECR, which allows read and write access to repositories, but does not allow users to delete repositories or change the policy documents applied to them.

```
{
    "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:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage"
            ],
            "Resource": "*"
        }
    ]
}
```

AmazonEC2ContainerRegistryReadOnly

This managed policy allows read-only access to Amazon ECR, such as the ability to list repositories and the images within the repositories, and also to pull images from Amazon ECR with the Docker CLI.

```
{
    "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:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings"
            ],
```

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

Amazon Elastic Container Registry Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Amazon ECR resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon ECR resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Amazon ECR quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Using the Amazon ECR Console

To access the Amazon Elastic Container Registry console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon ECR resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon ECR console, add the `AmazonEC2ContainerRegistryReadOnly` AWS managed policy to the entities. For more information, see Adding Permissions to a User in the *IAM User Guide*:

```
{
    "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:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings"
            ],
            "Resource": "*"
        }
    ]
}
```

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
```

```
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Accessing One Amazon ECR Repository

In this example, you want to grant an IAM user in your AWS account access to one of your Amazon ECR repositories, `my-repo`. You also want to allow the user to push, pull, and list images.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListImagesInRepository",
         "Effect":"Allow",
         "Action":[
            "ecr:ListImages"
         ],
         "Resource":"arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
      },
      {
         "Sid":"GetAuthorizationToken",
         "Effect":"Allow",
         "Action":[
            "ecr:GetAuthorizationToken"
         ],
         "Resource":"*"
      },
      {
         "Sid":"ManageRepositoryContents",
         "Effect":"Allow",
         "Action":[
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage"
         ],
         "Resource":"arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
      }
   ]
}
```

Using Tag-Based Access Control

The Amazon ECR CreateRepository API action enables you to specify tags when you create the repository. For more information, see Tagging an Amazon ECR repository (p. 24).

To enable users to tag repositories on creation, they must have permissions to use the action that creates the resource (for example, `ecr:CreateRepository`). If tags are specified in the resource-creating

action, Amazon performs additional authorization on the `ecr:CreateRepository` action to verify if users have permissions to create tags.

You can used tag-based access control through IAM policies. The following are examples.

The following policy would only allow an IAM user to create or tag a repository as `key=environment,value=dev`.

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

The following policy would allow an IAM user access to all repositories unless they were tagged as `key=environment,value=prod`.

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

Troubleshooting Amazon Elastic Container Registry Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon ECR and IAM.

Topics

I Am Not Authorized to Perform an Action in Amazon ECR

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a repository but does not have `ecr:DescribeRepositories` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 ecr:DescribeRepositories on resource: my-repo
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-repo` resource using the `ecr:DescribeRepositories` action.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon ECR.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon ECR. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important
Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the *IAM User Guide*.

I'm an Administrator and Want to Allow Others to Access Amazon ECR

To allow others to access Amazon ECR, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon ECR.

To get started right away, see Creating Your First IAM Delegated User and Group in the *IAM User Guide*.

I Want to Allow People Outside of My AWS Account to Access My Amazon ECR Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon ECR supports these features, see How Amazon Elastic Container Registry Works with IAM (p. 60).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Data protection in Amazon ECR

Amazon Elastic Container Registry (Amazon ECR) conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon ECR or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon ECR or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

Topics

Encryption at rest

Amazon ECR stores images in Amazon S3 buckets that Amazon ECR manages. By default, Amazon ECR uses server-side encryption with Amazon S3-managed encryption keys which encrypts your data at rest using an AES-256 encryption algorithm. This does not require any action on your part and is offered at no additional charge. For more information, see Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys (SSE-S3) in the *Amazon Simple Storage Service Developer Guide*.

For more control over the encryption for your Amazon ECR repositories, you can use server-side encryption with customer master keys (CMKs) stored in AWS Key Management Service (AWS KMS). When you use AWS KMS to encrypt your data, you can either use the default AWS-managed CMK, which is managed by Amazon ECR, or specify your own CMK (referred to as a customer-managed CMK). For more information, see Protecting Data Using Server-Side Encryption with CMKs Stored in AWS Key Management Service (SSE-KMS) in the *Amazon Simple Storage Service Developer Guide*.

Each Amazon ECR repository has an encryption configuration, which is set when the repository is created. You can use different encryption configurations on each repository. For more information, see Creating a repository (p. 16).

When a repository is created with AWS KMS encryption enabled, a CMK is used to encrypt the contents of the repository. Moreover, Amazon ECR adds an AWS KMS grant to the CMK with the Amazon ECR repository as the grantee principal.

The following provides a high-level understanding of how Amazon ECR is integrated with AWS KMS to encrypt and decrypt your repositories:

1. When creating a repository, Amazon ECR sends a DescribeKey call to AWS KMS to validate and retrieve the Amazon Resource Name (ARN) of the CMK specified in the encryption configuration.
2. Amazon ECR sends two CreateGrant requests to AWS KMS to create grants on the CMK to allow Amazon ECR to encrypt and decrypt data using the data key.

3. When pushing an image, a GenerateDataKey request is made to AWS KMS that specifies the CMK to use for encrypting the image layer and manifest.
4. AWS KMS generates a new data key, encrypts it under the specified CMK, and sends the encrypted data key to be stored with the image layer metadata and the image manifest.
5. When pulling an image, a Decrypt request is made to AWS KMS, specifying the encrypted data key.
6. AWS KMS decrypts the encrypted data key and sends the decrypted data key to Amazon S3.
7. The data key in used to decrypt the image layer before the image layer being pulled.
8. When a repository is deleted, Amazon ECR sends two RetireGrant requests to AWS KMS to retire the grants created for the repository.

Considerations

The following points should be considered when using AWS KMS encryption with Amazon ECR.

- If you create your Amazon ECR repository with KMS encryption and you do not specify a CMK, Amazon ECR uses an AWS-managed CMK with the alias `aws/ecr` by default. This CMK is created in your account the first time that you create a repository with KMS encryption enabled.
- When you use KMS encryption with your own CMK, the key must exist in the same Region as your repository.
- AWS KMS enforces a limit of 500 grants per CMK. As a result, there is a limit of 500 Amazon ECR repositories that can be encrypted per CMK.
- The grants that Amazon ECR creates on your behalf should not be revoked. If you revoke the grant that gives Amazon ECR permission to use the AWS KMS keys in your account, Amazon ECR cannot access this data, encrypt new images pushed to the repository, or decrypt them when they are pulled. When you revoke a grant for Amazon ECR, the change occurs immediately. To revoke access rights, you should delete the repository rather than revoking the grant. When a repository is deleted, Amazon ECR retires the grants on your behalf.
- There is a cost associated with using AWS KMS keys. For more information, see AWS Key Management Service pricing.

Required IAM permissions

When creating or deleting an Amazon ECR repository with server-side encryption using AWS KMS, the permissions required depend on the specific customer master key (CMK) you are using.

Required IAM permissions when using the AWS managed CMK for Amazon ECR

By default, when AWS KMS encryption is enabled for an Amazon ECR repository but no CMK is specified, the AWS-managed CMK for Amazon ECR is used. When the AWS-managed CMK for Amazon ECR is used to encrypt a repository, any principal that has permission to create a repository can also enable AWS KMS encryption on the repository. However, the IAM principal that deletes the repository must have the `kms:RetireGrant` permission. This enables the retirement of the grants that were added to the AWS KMS key when the repository was created.

The following example IAM policy can be added as an inline policy to a user to ensure they have the minimum permissions needed to delete a repository that has encryption enabled. The AWS KMS key used to encrypt the repository can be specified using the resource parameter.

```
{
    "Version": "2012-10-17",
    "Id": "ecr-kms-permissions",
    "Statement": [
        {
```

```
            "Sid": "Allow access to retire the grants associated with the key",
            "Effect": "Allow",
            "Action": [
                "kms:RetireGrant"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/b8d9ae76-080c-4043-92EXAMPLE"
        }
    ]
}
```

Required IAM permissions when using a customer-managed CMK

When creating a repository with AWS KMS encryption enabled using a customer-managed CMK, there are required permissions for both the CMK key policy and the IAM policy for the user or role creating the repository.

When creating your own CMK, you can either use the default key policy AWS KMS creates or you can specify your own. To ensure that the customer-managed CMK remains manageable by the account owner, the key policy for the CMK should allow all AWS KMS actions for the root user of the account. Additional scoped permissions may be added to the key policy but at minimum the root user should be given permissions to manage the CMK. To allow the CMK to be used only for requests that originate in Amazon ECR, you can use the kms:ViaService condition key with the `ecr.<region>.amazonaws.com` value.

The following example key policy gives the AWS account (root user) that owns the CMK full access to the CMK. For more information about this example key policy, see Allows access to the AWS account and enables IAM policies in the *AWS Key Management Service Developer Guide*.

```
{
    "Version": "2012-10-17",
    "Id": "ecr-key-policy",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        }
    ]
}
```

The IAM user, IAM role, or AWS account creating your repositories must have the `kms:CreateGrant`, `kms:RetireGrant`, and `kms:DescribeKey` permission in addition to the necessary Amazon ECR permissions.

> **Note**
> The `kms:RetireGrant` permission must be added to the IAM policy of the user or role creating the repository. The `kms:CreateGrant` and `kms:DescribeKey` permissions can be added to either the key policy for the CMK or the IAM policy of user or role creating the repository. For more information on how AWS KMS permissions work, see AWS KMS API permissions: Actions and resources reference in the *AWS Key Management Service Developer Guide*.

The following example IAM policy can be added as an inline policy to a user to ensure they have the minimum permissions needed to create a repository with encryption enabled and delete the repository when they are finished with it. The AWS KMS key used to encrypt the repository can be specified using the resource parameter.

```
{
    "Version": "2012-10-17",
    "Id": "ecr-kms-permissions",
    "Statement": [
        {
            "Sid": "Allow access to create and retire the grants associated with the key as
 well as describe the key",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:RetireGrant",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/b8d9ae76-080c-4043-92EXAMPLE"
        }
    ]
}
```

Allow a user to list CMKs in the console when creating a repository

When using the Amazon ECR console to create a repository, you can grant permissions to enable a user to list the customer-managed CMKs in the Region when enabling encryption for the repository. The following IAM policy example shows the permissions needed to list your CMKs and aliases when using the console.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:ListKeys",
      "kms:ListAliases",
      "kms:DescribeKey"
    ],
    "Resource": "*"
  }
}
```

Monitoring Amazon ECR interaction with AWS KMS

You can use AWS CloudTrail to track the requests that Amazon ECR sends to AWS KMS on your behalf. The log entries in the CloudTrail log contain an encryption context key to make them more easily identifiable.

Amazon ECR encryption context

An *encryption context* is a set of key–value pairs that contains arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

In its GenerateDataKey and Decrypt requests to AWS KMS, Amazon ECR uses an encryption context with two name–value pairs that identify the repository and Amazon S3 bucket being used. This is shown in the following example. The names do not vary, but combined encryption context values will be different for each value.

```
"encryptionContext": {
    "aws:s3:arn": "arn:aws:s3:::us-west-2-starport-manifest-bucket/EXAMPLE1-90ab-cdef-fedc-
ba987BUCKET1/sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df",
```

```
    "aws:ecr:arn": "arn:aws:ecr:us-west-2:111122223333:repository/repository-name"
}
```

You can use the encryption context to identify these cryptographic operation in audit records and logs, such as AWS CloudTrail and Amazon CloudWatch Logs, and as a condition for authorization in policies and grants.

The Amazon ECR encryption context consists of two name-value pairs.

- **aws:s3:arn** – The first name–value pair identifies the bucket. The key is `aws:s3:arn`. The value is the Amazon Resource Name (ARN) of the Amazon S3 bucket.

  ```
  "aws:s3:arn": "ARN of an Amazon S3 bucket"
  ```

  For example, if the ARN of the bucket is `arn:aws:s3:::us-west-2-starport-manifest-bucket/EXAMPLE1-90ab-cdef-fedc-ba987BUCKET1/sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df`, the encryption context would include the following pair.

  ```
  "arn:aws:s3:::us-west-2-starport-manifest-bucket/EXAMPLE1-90ab-cdef-fedc-ba987BUCKET1/
  sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df"
  ```

- **aws:ecr:arn** – The second name–value pair identifies the Amazon Resource Name (ARN) of the repository. The key is `aws:ecr:arn`. The value is the ARN of the repository.

  ```
  "aws:ecr:arn": "ARN of an Amazon ECR repository"
  ```

  For example, if the ARN of the repository is `arn:aws:ecr:us-west-2:111122223333:repository/repository-name`, the encryption context would include the following pair.

  ```
  "aws:ecr:arn": "arn:aws:ecr:us-west-2:111122223333:repository/repository-name"
  ```

Troubleshooting

When deleting an Amazon ECR repository with the console, if the repository is successfully deleted but Amazon ECR is unable to retire the grants added to your CMK for your repository, you will receive the following error.

```
The repository [{repository-name}] has been deleted successfully but the grants created by
 the kmsKey [{kms_key}] failed to be retired
```

When this occurs, you can retire the AWS KMS grants for the repository yourself.

To retire AWS KMS grants for a repository manually

1. List the grants for the AWS KMS key used for the repository. The `key-id` value is included in the error you receive from the console. You can also use the `list-keys` command to list both the AWS managed CMKs and customer-managed CMKs in a specific Region in your account.

   ```
   aws kms list-grants \
        --key-id b8d9ae76-080c-4043-9237-c815bfc21dfc
        --region us-west-2
   ```

The output include an `EncryptionContextSubset` with the Amazon Resource Name (ARN) of your repository. This can be used to determine which grant added to the key is the one you want to retire. The `GrantId` value will be used when retiring the grant in the next step.

2. Retire each grant for the AWS KMS key added for the repository. Replace the value for *`GrantId`* with the ID of the grant from the output of the previous step.

```
aws kms retire-grant \
     --key-id b8d9ae76-080c-4043-9237-c815bfc21dfc \
     --grant-id GrantId \
     --region us-west-2
```

Compliance Validation for Amazon Elastic Container Registry

Third-party auditors assess the security and compliance of Amazon Elastic Container Registry as part of multiple AWS compliance programs. These include SOC, PCI, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon ECR is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Infrastructure Security in Amazon Elastic Container Registry

As a managed service, Amazon Elastic Container Registry is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon ECR through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support

cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location, but Amazon ECR does support resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon ECR policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon ECR resource from only the specific VPC within the AWS network. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) (p. 77).

Amazon ECR interface VPC endpoints (AWS PrivateLink)

You can improve the security posture of your VPC by configuring Amazon ECR to use an interface VPC endpoint. VPC endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Amazon ECR APIs through private IP addresses. AWS PrivateLink restricts all network traffic between your VPC and Amazon ECR to the Amazon network. You don't need an internet gateway, a NAT device, or a virtual private gateway.

For more information about AWS PrivateLink and VPC endpoints, see VPC Endpoints in the *Amazon VPC User Guide*.

Considerations for Amazon ECR VPC endpoints

Before you configure VPC endpoints for Amazon ECR, be aware of the following considerations.

- To allow your Amazon ECS tasks that use the EC2 launch type to pull private images from Amazon ECR, ensure that you also create the interface VPC endpoints for Amazon ECS. For more information, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon Elastic Container Service Developer Guide*.

  Important
  Amazon ECS tasks that use the Fargate launch type don't require the Amazon ECS interface VPC endpoints.
- Amazon ECS tasks using the Fargate launch type and platform version 1.3.0 or earlier only require the **com.amazonaws.*region*.ecr.dkr** Amazon ECR VPC endpoint and the Amazon S3 gateway endpoint to take advantage of this feature.
- Amazon ECS tasks using the Fargate launch type and platform version 1.4.0 or later require both the **com.amazonaws.*region*.ecr.dkr** and **com.amazonaws.*region*.ecr.api** Amazon ECR VPC endpoints as well as the Amazon S3 gateway endpoint to take advantage of this feature.
- Amazon ECS tasks using the Fargate launch type that pull container images from Amazon ECR can restrict access to the specific VPC their tasks use and to the VPC endpoint the service uses by adding condition keys to the task execution IAM role for the task. For more information, see Optional IAM Permissions for Fargate Tasks Pulling Amazon ECR Images over Interface Endpoints in the *Amazon Elastic Container Service Developer Guide*.
- Amazon ECS tasks using the Fargate launch type that pull container images from Amazon ECR that also use the `awslogs` log driver to send log information to CloudWatch Logs require the CloudWatch Logs VPC endpoint. For more information, see Create the CloudWatch Logs endpoint (p. 80).
- The security group attached to the VPC endpoint must allow incoming connections on port 443 from the private subnet of the VPC.
- VPC endpoints currently don't support cross-Region requests. Ensure that you create your VPC endpoints in the same Region where you plan to issue your API calls to Amazon ECR.

- VPC endpoints only support Amazon provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see DHCP Options Sets in the *Amazon VPC User Guide*.
- If your containers have existing connections to Amazon S3, their connections might be briefly interrupted when you add the Amazon S3 gateway endpoint. If you want to avoid this interruption, create a new VPC that uses the Amazon S3 gateway endpoint and then migrate your Amazon ECS cluster and its containers into the new VPC.

Considerations for Windows images

Images based on the Windows operating system include artifacts that are restricted by license from being distributed. By default, when you push Windows images to an Amazon ECR repository, the layers that include these artifacts are not pushed as they are considered *foreign layers*. When the artifacts are provided by Microsoft, the foreign layers are retrieved from Microsoft Azure infrastructure. For this reason, to enable your containers to pull these foreign layers from Azure additional steps are needed beyond creating the VPC endpoints.

It is possible to override this behaviour when pushing Windows images to Amazon ECR by using the `--allow-nondistributable-artifacts` flag in the Docker daemon. When enabled, this flag will push the licensed layers to Amazon ECR which enables these images to be pulled from Amazon ECR via the VPC endpoint without additional access to Azure being required.

> **Important**
> Using the `--allow-nondistributable-artifacts` flag does not preclude your obligation to comply with the terms of the Windows container base image license; you cannot post Windows content for public or third-party redistribution. Usage within your own environment is allowed.

To enable the use of this flag for your Docker installation, you must modify the Docker daemon configuration file which, depending on your Docker installation, can typically be configured in settings or preferences menu under the **Docker Engine** section or by editing the `C:\ProgramData\docker\config\daemon.json` file directly.

The following is an example of the required configuration. Replace the value with the repository URI you are pushing images to.

```
{
    "allow-nondistributable-artifacts": [
        "111122223333.dkr.ecr.us-west-2.amazonaws.com"
    ]
}
```

After modifying the Docker daemon configuration file, you must restart the Docker daemon before attempting to push your image. Confirm the push worked by verifying that the base layer was pushed to your repository.

> **Note**
> The base layers for Windows images are large. The layer size will result in a longer time to push and additional storage costs in Amazon ECR for these images. For these reasons, we recommend only using this option when it is strictly required to reduce build times and ongoing storage costs. For example, the `mcr.microsoft.com/windows/servercore` image is approximately 1.7 GiB in size when compressed in Amazon ECR.

Create the VPC endpoints for Amazon ECR

To create the VPC endpoints for the Amazon ECR service, use the Creating an Interface Endpoint procedure in the *Amazon VPC User Guide*.

Amazon ECS tasks using the EC2 launch type require both Amazon ECR endpoints and the Amazon S3 gateway endpoint.

Amazon ECS tasks using the Fargate launch type and platform version 1.3.0 or earlier only require the **com.amazonaws.*region*.ecr.dkr** Amazon ECR VPC endpoint and the Amazon S3 gateway endpoints.

Amazon ECS tasks using the Fargate launch type and platform version 1.4.0 or later require both the **com.amazonaws.*region*.ecr.dkr** and **com.amazonaws.*region*.ecr.api** Amazon ECR VPC endpoints and the Amazon S3 gateway endpoints.

Note
The order that the endpoints are created in doesn't matter.

com.amazonaws.*region*.ecr.dkr

This endpoint is used for the Docker Registry APIs. Docker client commands such as `push` and `pull` use this endpoint.

When you create the **com.amazonaws.*region*.ecr.dkr** endpoint, you must enable a private DNS hostname. To do this, ensure that the **Enable Private DNS Name** option is selected in the VPC console when you create the VPC endpoint.

com.amazonaws.*region*.ecr.api

Note
The specified *region* represents the Region identifier for an AWS Region supported by Amazon ECR, such as `us-east-2` for the US East (Ohio) Region.

This endpoint is used for calls to the Amazon ECR API. API actions such as DescribeImages and CreateRepositories go to this endpoint.

When the **com.amazonaws.*region*.ecr.api** endpoint is created, you have the option to enable a private DNS hostname. Enable this setting by selecting **Enable Private DNS Name** in the VPC console when you create the VPC endpoint. If you enable a private DNS hostname for the VPC endpoint, update your SDK or AWS CLI to the latest version so that specifying an endpoint URL when using the SDK or AWS CLI isn't necessary.

If you enable a private DNS hostname and are using an SDK or AWS CLI version released before January 24, 2019, you must use the `--endpoint-url` parameter to specify the interface endpoints. The following example shows the format for the endpoint URL.

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

If you don't enable a private DNS hostname for the VPC endpoint, you must use the `--endpoint-url` parameter specifying the VPC endpoint ID for the interface endpoint. The following example shows the format for the endpoint URL.

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

Create the Amazon S3 gateway endpoint

For your Amazon ECS tasks to pull private images from Amazon ECR, you must create a gateway endpoint for Amazon S3. The gateway endpoint is required because Amazon ECR uses Amazon S3 to store your image layers. When your containers download images from Amazon ECR, they must access Amazon ECR to get the image manifest and then Amazon S3 to download the actual image layers. The following is the Amazon Resource Name (ARN) of the Amazon S3 bucket containing the layers for each Docker image.

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

Use the Creating a gateway endpoint procedure in the *Amazon VPC User Guide* to create the following Amazon S3 gateway endpoint for Amazon ECR. When creating the endpoint, be sure to select the route tables for your VPC.

com.amazonaws.*region*.s3

The Amazon S3 gateway endpoint uses an IAM policy document to limit access to the service. The **Full Access** policy can be used because any restrictions that you have put in your task IAM roles or other IAM user policies still apply on top of this policy. If you want to limit Amazon S3 bucket access to the minimum required permissions for using Amazon ECR, see Minimum Amazon S3 Bucket Permissions for Amazon ECR (p. 80).

Minimum Amazon S3 Bucket Permissions for Amazon ECR

The Amazon S3 gateway endpoint uses an IAM policy document to limit access to the service. To allow only the minimum Amazon S3 bucket permissions for Amazon ECR, restrict access to the Amazon S3 bucket that Amazon ECR uses when you create the IAM policy document for the endpoint.

The following table describes the Amazon S3 bucket policy permissions needed by Amazon ECR.

Permission	Description
`arn:aws:s3:::prod-region-starport-layer-bucket/*`	Provides access to the Amazon S3 bucket containing the layers for each Docker image. Represents the Region identifier for an AWS Region supported by Amazon ECR, such as `us-east-2` for the US East (Ohio) Region.

Example

The following example illustrates how to provide access to the Amazon S3 buckets required for Amazon ECR operations.

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::prod-region-starport-layer-bucket/*"]
    }
  ]
}
```

Create the CloudWatch Logs endpoint

Amazon ECS tasks using the Fargate launch type that use a VPC without an internet gateway that also use the `awslogs` log driver to send log information to CloudWatch Logs require that you create the **com.amazonaws.*region*.logs** interface VPC endpoint for CloudWatch Logs. For more information, see Creating a gateway endpoint in the *Amazon CloudWatch Logs User Guide*.

Create an endpoint policy for your Amazon ECR VPC endpoints

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, AWS attaches a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace IAM user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service. Endpoint policies must be written in JSON format. For more information, see Controlling Access to Services with VPC Endpoints in the *Amazon VPC User Guide*.

We recommend creating a single IAM resource policy and attaching it to both of the Amazon ECR VPC endpoints.

The following is an example of an endpoint policy for Amazon ECR. This policy enables a specific IAM role to pull images from Amazon ECR.

```
{
 "Statement": [{
  "Sid": "AllowPull",
  "Principal": {
   "AWS": "arn:aws:iam::1234567890:role/role_name"
  },
  "Action": [
   "ecr:BatchGetImage",
   "ecr:GetDownloadUrlForLayer"
  ],
  "Effect": "Allow",
  "Resource": "*"
 }]
}
```

The following endpoint policy example prevents a specified repository from being deleted.

```
{
 "Statement": [{
   "Sid": "AllowAll",
   "Principal": "*",
   "Action": "*",
   "Effect": "Allow",
   "Resource": "*"
  },
  {
   "Sid": "PreventDelete",
   "Principal": "*",
   "Action": "ecr:DeleteRepository",
   "Effect": "Deny",
   "Resource": "arn:aws:ecr:region:1234567890:repository/repository_name"
  }
 ]
}
```

The following endpoint policy example combines the two previous examples into a single policy.

```
{
 "Statement": [{
   "Sid": "AllowAll",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "*",
   "Resource": "*"
  },
  {
```

```
   "Sid": "PreventDelete",
   "Effect": "Deny",
   "Principal": "*",
   "Action": "ecr:DeleteRepository",
   "Resource": "arn:aws:ecr:region:1234567890:repository/repository_name"
  },
  {
   "Sid": "AllowPull",
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::1234567890:role/role_name"
   },
   "Action": [
    "ecr:BatchGetImage",
    "ecr:GetDownloadUrlForLayer"
   ],
   "Resource": "*"
  }
 ]
}
```

To modify the VPC endpoint policy for Amazon ECR

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the VPC endpoints for Amazon ECR, see Create the VPC endpoints for Amazon ECR (p. 78).
4. Select the Amazon ECR VPC endpoint to add a policy to, and choose the **Policy** tab in the lower half of the screen.
5. Choose **Edit Policy** and make the changes to the policy.
6. Choose **Save** to save the policy.

Amazon ECR monitoring

You can monitor your Amazon ECR API usage with Amazon CloudWatch, which collects and processes raw data from Amazon ECR into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain perspective on your API usage. Amazon ECR metric data is automatically sent to CloudWatch in one-minute periods. For more information about CloudWatch, see the Amazon CloudWatch User Guide.

Amazon ECR provides metrics based on your API usage for authorization, image push, and image pull actions.

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon ECR and your AWS solutions. We recommend that you collect monitoring data from the resources that make up your AWS solution so that you can more easily debug a multi-point failure if one occurs. Before you start monitoring Amazon ECR, however, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

The next step is to establish a baseline for normal Amazon ECR performance in your environment by measuring performance at various times and under different load conditions. As you monitor Amazon ECR, store historical monitoring data so that you can compare it with new performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

Topics

Visualizing your service quotas and setting alarms

You can use the CloudWatch console to visualize your service quotas and see how your current usage compares to service quotas. You can also set alarms so that you will be notified when you approach a quota.

To visualize a service quota and optionally set an alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. On the **All metrics** tab, choose **Usage**, then choose **By AWS Resource**.

   The list of service quota usage metrics appears.
4. Select the check box next to one of the metrics.

The graph displays your current usage of that AWS resource.

5. To add your service quota to the graph, do the following:

   a. Choose the **Graphed metrics** tab.

   b. Choose **Math expression**, **Start with an empty expression**. Then in the new row, under **Details**, enter **`SERVICE_QUOTA(m1)`**.

      A new line is added to the graph, displaying the service quota for the resource represented in the metric.

6. To see your current usage as a percentage of the quota, add a new expression or change the current **SERVICE_QUOTA** expression. For the new expression, use **`m1/60/SERVICE_QUOTA(m1)*100`**.

7. (Optional) To set an alarm that notifies you if you approach the service quota, do the following:

   a. On the **`m1/60/SERVICE_QUOTA(m1)*100`** row, under **Actions**, choose the alarm icon. It looks like a bell.

      The alarm creation page appears.

   b. Under **Conditions**, ensure that **Threshold type** is **Static** and **Whenever Expression1 is** is set to **Greater**. Under **than**, enter **`80`**. This creates an alarm that goes into ALARM state when your usage exceeds 80 percent of the quota.

   c. Choose **Next**.

   d. On the next page, select an Amazon SNS topic or create a new one. This topic is notified when the alarm goes to ALARM state. Then choose **Next**.

   e. On the next page, enter a name and description for the alarm, and then choose **Next**.

   f. Choose **Create alarm**.

Amazon ECR usage metrics

You can use CloudWatch usage metrics to provide visibility into your account's usage of resources. Use these metrics to visualize your current service usage on CloudWatch graphs and dashboards.

Amazon ECR usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information about Amazon ECR service quotas, see Amazon ECR service quotas (p. 96).

Amazon ECR publishes the following metrics in the `AWS/Usage` namespace.

Metric	Description
`CallCount`	The number of API action calls from your account. The resources are defined by the dimensions associated with the metric. The most useful statistic for this metric is `SUM`, which represents the sum of the values from all contributors during the period defined.

The following dimensions are used to refine the usage metrics that are published by Amazon ECR.

Dimension	Description
`Service`	The name of the AWS service containing the resource. For Amazon ECR usage metrics, the value for this dimension is `ECR`.

Dimension	Description
`Type`	The type of entity that is being reported. Currently, the only valid value for Amazon ECR usage metrics is `API`.
`Resource`	The type of resource that is running. Currently, Amazon ECR returns information on your API usage for the following API actions. • GetAuthorizationToken • BatchCheckLayerAvailability • InitiateLayerUpload • UploadLayerPart • CompleteLayerUpload • PutImage • BatchGetImage • GetDownloadUrlForLayer
`Class`	The class of resource being tracked. Currently, Amazon ECR does not use the class dimension.

Amazon ECR usage reports

AWS provides a free reporting tool called Cost Explorer that enables you to analyze the cost and usage of your Amazon ECR resources.

Use Cost Explorer to view charts of your usage and costs. You can view data from the previous 13 months and forecast how much you are likely to spend for the next three months. You can use Cost Explorer to see patterns in how much you spend on AWS resources over time, identify areas that need further inquiry, and see trends that you can use to understand your costs. You also can specify time ranges for the data and view time data by day or by month.

The metering data in your Cost and Usage Reports shows usage across all of your Amazon ECR repositories. For more information, see Tagging your resources for billing (p. 26).

For more information about creating an AWS Cost and Usage Report, see AWS Cost and Usage Report in the *AWS Billing and Cost Management User Guide*.

Amazon ECR events and EventBridge

Amazon EventBridge enables you to automate your AWS services and to respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you and include automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Adding events to log groups in CloudWatch Logs
- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an AWS SMS queue

For more information, see Getting Started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

Sample events from Amazon ECR

The following are example events from Amazon ECR.

Event for a completed image push

The following event is sent when each image push is completed. For more information, see Pushing an image (p. 28).

```
{
    "version": "0",
    "id": "13cde686-328b-6117-af20-0e5566167482",
    "detail-type": "ECR Image Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-11-16T01:54:34Z",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "result": "SUCCESS",
        "repository-name": "my-repo",
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "action-type": "PUSH",
        "image-tag": "latest"
    }
}
```

Event for a completed image scan

The following event is sent when each image scan is completed. The `finding-severity-counts` parameter will only return a value for a severity level if one exists. For example, if the image contains no findings at `CRITICAL` level, then no critical count is returned. For more information, see Image scanning (p. 47).

```
{
    "version": "0",
    "id": "85fc3613-e913-7fc4-a80c-a3753e4aa9ae",
    "detail-type": "ECR Image Scan",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-10-29T02:36:48Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
    ],
    "detail": {
        "scan-status": "COMPLETE",
        "repository-name": "my-repo",
        "finding-severity-counts": {
        "CRITICAL": 10,
        "MEDIUM”: 9
      },
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "image-tags": []
    }
}
```

Event for an image deletion

The following event is sent when an image is deleted. For more information, see Deleting an image (p. 32).

```
{
    "version": "0",
    "id": "dd3b46cb-2c74-f49e-393b-28286b67279d",
    "detail-type": "ECR Image Action",
    "source": "aws.ecr",
    "account": "123456789012",
    "time": "2019-11-16T02:01:05Z",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "result": "SUCCESS",
        "repository-name": "my-repo",
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "action-type": "DELETE",
        "image-tag": "latest"
    }
}
```

Logging Amazon ECR actions with AWS CloudTrail

Amazon ECR is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, a role, or an AWS service in Amazon ECR. CloudTrail captures the following Amazon ECR actions as events:

- All API calls, including calls from the Amazon ECR console
- All actions taken due to the encryption settings on your repositories
- All actions taken due to lifecycle policy rules, including both successful and unsuccessful actions

When a trail is created, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon ECR. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using this information, you can determine the request that was made to Amazon ECR, the originating IP address, who made the request, when it was made, and additional details.

For more information, see the AWS CloudTrail User Guide.

Amazon ECR information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon ECR, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon ECR, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. When you create a trail in the console, you can apply the trail to a single Region or to all Regions. The trail logs events in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Creating a trail for your AWS account

- AWS Service Integrations With CloudTrail Logs
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon ECR API actions are logged by CloudTrail and are documented in the Amazon Elastic Container Registry API Reference. When you perform common tasks, sections are generated in the CloudTrail log files for each API action that is part of that task. For example, when you create a repository, `GetAuthorizationToken`, `CreateRepository` and `SetRepositoryPolicy` sections are generated in the CloudTrail log files. When you push an image to a repository, `InitiateLayerUpload`, `UploadLayerPart`, `CompleteLayerUpload`, and `PutImage` sections are generated. When you pull an image, `GetDownloadUrlForLayer` and `BatchGetImage` sections are generated. For examples of these common tasks, see CloudTrail log entry examples (p. 88).

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail `userIdentity` Element.

Understanding Amazon ECR log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and other information. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

CloudTrail log entry examples

The following are CloudTrail log entry examples for a few common Amazon ECR tasks.

> **Note**
> These examples have been formatted for improved readability. In a CloudTrail log file, all entries and events are concatenated into a single line. In addition, this example has been limited to a single Amazon ECR entry. In a real CloudTrail log file, you see entries and events from multiple AWS services.

Topics

Example: Create repository action

The following example shows a CloudTrail log entry that demonstrates the `CreateRepository` action.

```
{
```

```
    "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"
}
```


Example: AWS KMS CreateGrant API action when creating an Amazon ECR repository

The following example shows a CloudTrail log entry that demonstrates the AWS KMS CreateGrant action when creating an Amazon ECR repository with KMS encryption enabled. For each repository that is created with KMS encryption is enabled, you should see two CreateGrant log entries in CloudTrail.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAIEP6W46J43IG7LXAQ",
```

```
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "sessionIssuer": {

            },
            "webIdFederationData": {

            },
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-06-10T19:22:10Z"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-06-10T19:22:10Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "keyId": "4b55e5bf-39c8-41ad-b589-18464af7758a",
        "granteePrincipal": "ecr.us-west-2.amazonaws.com",
        "operations": [
            "GenerateDataKey",
            "Decrypt"
        ],
        "retiringPrincipal": "ecr.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecr:arn": "arn:aws:ecr:us-west-2:123456789012:repository/testrepo"
            }
        }
    },
    "responseElements": {
        "grantId": "3636af9adfee1accb67b83941087dcd45e7fadc4e74ff0103bb338422b5055f3"
    },
    "requestID": "047b7dea-b56b-4013-87e9-a089f0f6602b",
    "eventID": "af4c9573-c56a-4886-baca-a77526544469",
    "readOnly": false,
    "resources": [
        {
            "accountId": "123456789012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:123456789012:key/4b55e5bf-39c8-41ad-
b589-18464af7758a"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


Example: Image push action

The following example shows a CloudTrail log entry that demonstrates an image push which uses the `PutImage` action.

Note
When pushing an image, you will also see `InitiateLayerUpload`, `UploadLayerPart`, and `CompleteLayerUpload` references in the CloudTrail logs.

```
{
    "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"
}
```

Example: Image pull action

The following example shows a CloudTrail log entry that demonstrates an image pull which uses the `BatchGetImage` action.

Note
When pulling an image, if you don't already have the image locally, you will also see `GetDownloadUrlForLayer` references in the CloudTrail logs.

```
{
    "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"
}
```

Example: Image lifecycle policy action

The following example shows a CloudTrail log entry that demonstrates when an image is expired due to a lifecycle policy rule. This event type can be located by filtering for `PolicyExecutionEvent` for the event name field.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "accountId": "123456789012",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-03-12T20:22:12Z",
    "eventSource": "ecr.amazonaws.com",
    "eventName": "PolicyExecutionEvent",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "9354dd7f-9aac-4e9d-956d-12561a4923aa",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:ecr:us-west-2:123456789012:repository/testrepo",
            "accountId": "123456789012",
            "type": "AWS::ECR::Repository"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "123456789012",
    "serviceEventDetails": {
        "repositoryName": "testrepo",
        "lifecycleEventPolicy": {
            "lifecycleEventRules": [
                {
                    "rulePriority": 1,
                    "description": "remove all images > 2",
                    "lifecycleEventSelection": {
                        "tagStatus": "Any",
                        "tagPrefixList": [],
                        "countType": "Image count more than",
                        "countNumber": 2
                    },
                    "action": "expire"
                }
            ],
            "lastEvaluatedAt": 0,
            "policyVersion": 1,
            "policyId": "ceb86829-58e7-9498-920c-aa042e33037b"
        },
        "lifecycleEventImageActions": [
            {
                "lifecycleEventImage": {
                    "digest":
 "sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45",
                    "tagStatus": "Tagged",
                    "tagList": [
                        "alpine"
                    ],
                    "pushedAt": 1584042813000
                },
                "rulePriority": 1
            },
```

```
            {
                "lifecycleEventImage": {
                    "digest":
 "sha256:6ab380c5a5acf71c1b6660d645d2cd79cc8ce91b38e0352cbf9561e050427baf",
                    "tagStatus": "Tagged",
                    "tagList": [
                        "centos"
                    ],
                    "pushedAt": 1584042842000
                },
                "rulePriority": 1
            }
        ]
    }
}
```

Amazon ECR service quotas

The following table provides the default service quotas for Amazon Elastic Container Registry (Amazon ECR).

Service quota	Description	Default quota value
Registered repositories	The maximum number of repositories that you can create per Region.	10,000
Image per repository	The maximum number of images per repository.	10,000

The following table provides the default rate quotas for each of the Amazon ECR API actions involved with the image push and image pull actions.

Amazon ECR action	API operation	Description	Default quota value
Authentication	Rate of GetAuthorizationToken requests	The rate of GetAuthorizationToken API requests that you can make per second, per Region.	200
Image push	Rate of BatchCheckLayerAvailability requests	The rate of BatchCheckLayerAvailability API requests that you can make per second, per Region. When an image is pushed to a repository, each image layer is checked to verify if it has been uploaded before. If it has been uploaded, then the image layer is skipped.	200
	Rate of InitiateLayerUpload requests	The rate of InitiateLayerUpload API requests that you can make per second, per Region. When an image is pushed, the InitiateLayerUpload API is called once per image layer that has not already been uploaded. Whether	10

Amazon ECR action	API operation	Description	Default quota value
		or not an image layer has been uploaded is determined by the BatchCheckLayerAvailability API action.	
	Rate of CompleteLayerUpload requests	The rate of CompleteLayerUpload API requests that you can make per second, per Region. When an image is pushed, the CompleteLayerUpload API is called once per each new image layer to verify that the upload has completed.	10
	Rate of UploadLayerPart requests	The rate of UploadLayerPart API requests that you can make per second, per Region. When an image is pushed, each new image layer is uploaded in parts. The maximum size of each image layer part can be 20,971,520 bytes (or about 20MB). The UploadLayerPart API is called once per each new image layer part.	260
	Rate of PutImage requests	The rate of PutImage API requests that you can make per second, per Region. When an image is pushed and all new image layers have been uploaded, the PutImage API is called once to create or update the image manifest and the tags associated with the image.	10

Amazon ECR action	API operation	Description	Default quota value
Image pull	Rate of BatchGetImage requests	The rate of BatchGetImage API requests that you can make per second, per Region. When an image is pulled, the BatchGetImage API is called once to retrieve the image manifest.	1,000
	Rate of GetDownloadUrlForLayer requests	The rate of GetDownloadUrlForLayer API requests that you can make per second, per Region. When an image is pulled, the GetDownloadUrlForLayer API is called once per image layer that is not already cached.	1,500

The following table provides other quotas for Amazon ECR and Docker images that cannot be changed.

Note
The layer part information mentioned in the following table is only applicable if you are calling the Amazon ECR API actions directly to initiate multipart uploads for image push operations. This is a rare action. We recommend that you use the Docker CLI to pull, tag, and push images.

Service quota	Description	Quota value
Layer parts	The maximum number of layer parts. This is only applicable if you are using Amazon ECR API actions directly to initiate multipart uploads for image push operations.	1,000
Maximum layer size	The maximum size (MiB) of a layer. **	10,000
Minimum layer part size	The minimum size (MiB) of a layer part. This is only applicable if you are using Amazon ECR API actions directly to initiate multipart uploads for image push operations.	5
Maximum layer part size	The maximum size (MiB) of a layer part. This is only applicable if you are using Amazon ECR API actions directly to initiate	10

Service quota	Description	Quota value
	multipart uploads for image push operations.	
Tags per image	The maximum number of tags per image.	1000
Lifecycle policy length	The maximum number of characters in a lifecycle policy.	30,720
Rules per lifecycle policy	The maximum number of rules in a lifecycle policy.	50
Rate of image scans	The maximum number of image scans per image, per day.	1

** The maximum layer size listed here is calculated by multiplying the maximum layer part size (10 MiB) by the maximum number of layer parts (1,000).

Managing your Amazon ECR service quotas in the AWS Management Console

Amazon ECR has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see What Is Service Quotas? in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of all Amazon ECR service quotas.

To view Amazon ECR service quotas (AWS Management Console)

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon Elastic Container Registry (Amazon ECR)**.

   In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.
4. To view additional information about a service quota, such as the description, choose the quota name.

To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

Creating a CloudWatch alarm to monitor API usage metrics

Amazon ECR provides CloudWatch usage metrics that correspond to the AWS service quotas for each of the APIs involved with the registry authentication, image push, and image pull actions. In the Service Quotas console, you can visualize your usage on a graph and configure alarms that alert you when your usage approaches a service quota. For more information, see Amazon ECR usage metrics (p. 84).

Use the following steps to create a CloudWatch alarm based on one of the Amazon ECR API usage metrics.

To create an alarm based on your Amazon ECR usage quotas (AWS Management Console)

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon Elastic Container Registry (Amazon ECR)**.
4. In the **Service quotas** list, select the Amazon ECR usage quota you want to create an alarm for.
5. In the Amazon CloudWatch Events alarms section, choose **Create**.
6. For **Alarm threshold**, choose the percentage of your applied quota value that you want to set as the alarm value.
7. For **Alarm name**, enter a name for the alarm and then choose **Create**.

Amazon ECR Troubleshooting

This chapter helps you find diagnostic information for Amazon Elastic Container Registry (Amazon ECR), and provides troubleshooting steps for common issues and error messages.

Topics

Enabling Docker Debug Output

To begin debugging any Docker-related issue, you should start by enabling Docker debugging output on the Docker daemon running on your host instances. For more information about enabling Docker debugging if you are using images pulled from Amazon ECR on Amazon ECS container instances, see Enabling Docker Debug Output in the *Amazon Elastic Container Service Developer Guide*.

Enabling AWS CloudTrail

Additional information about errors returned by Amazon ECR can be discovered by enabling AWS CloudTrail, which is a service that records AWS calls for your AWS account. CloudTrail delivers log files to an Amazon S3 bucket. By using information collected by CloudTrail, you can determine what requests were successfully made to AWS services, who made the request, when it was made, and so on. To learn more about CloudTrail, including how to turn it on and find your log files, see the AWS CloudTrail User Guide. For more information on using CloudTrail with Amazon ECR, see Logging Amazon ECR actions with AWS CloudTrail (p. 87).

Optimizing Performance for Amazon ECR

The following section provides recommendations on settings and strategies that can be used to optimize performance when using Amazon ECR.

Use Docker 1.10 and above to take advantage of simultaneous layer uploads

> Docker images are composed of layers, which are intermediate build stages of the image. Each line in a Dockerfile results in the creation of a new layer. When you use Docker 1.10 and above, Docker defaults to pushing as many layers as possible as simultaneous uploads to Amazon ECR, resulting in faster upload times.

Use a smaller base image

> The default images available through Docker Hub may contain many dependencies that your application doesn't require. Consider using a smaller image created and maintained by others in the Docker community, or build your own base image using Docker's minimal scratch image. For more information, see Create a base image in the Docker documentation.

Place the dependencies that change the least earlier in your Dockerfile

Docker caches layers, and that speeds up build times. If nothing on a layer has changed since the last build, Docker uses the cached version instead of rebuilding the layer. However, each layer is dependent on the layers that came before it. If a layer changes, Docker recompiles not only that layer, but any layers that come after that layer as well.

To minimize the time required to rebuild a Dockerfile and to re-upload layers, consider placing the dependencies that change the least frequently earlier in your Dockerfile. Place rapidly changing dependencies (such as your application's source code) later in the stack.

Chain commands to avoid unnecessary file storage

Intermediate files created on a layer remain a part of that layer even if they are deleted in a subsequent layer. Consider the following example:

```
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
```

In this example, the layers created by the first and second RUN commands contain the original .tar.gz file and all of its unzipped contents. This is even though the .tar.gz file is deleted by the fourth RUN command. These commands can be chained together into a single RUN statement to ensure that these unnecessary files aren't part of the final Docker image:

```
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
```

Use the closest regional endpoint

You can reduce latency in pulling images from Amazon ECR by ensuring that you are using the regional endpoint closest to where your application is running. If your application is running on an Amazon EC2 instance, you can use the following shell code to obtain the region from the Availability Zone of the instance:

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

The region can be passed to AWS CLI commands using the **--region** parameter, or set as the default region for a profile using the **aws configure** command. You can also set the region when making calls using the AWS SDK. For more information, see the documentation for the SDK for your specific programming language.

Troubleshooting Errors with Docker Commands When Using Amazon ECR

Topics

- HTTP 403 Errors or "no basic auth credentials" Error When Pushing to Repository (p. 104)

In some cases, running a Docker command against Amazon ECR may result in an error message. Some common error messages and potential solutions are explained below.

Error: "Filesystem Verification Failed" or "404: Image Not Found" When Pulling an Image From an Amazon ECR Repository

You may receive the error `Filesystem verification failed` when using the **docker pull** command to pull an image from an Amazon ECR repository with Docker 1.9 or above. You may receive the error `404: Image not found` when you are using Docker versions before 1.9.

Some possible reasons and their explanations are given below.

The local disk is full

> If the local disk on which you're running **docker pull** is full, then the SHA-1 hash calculated on the local file may be different than the one calculated by Amazon ECR. Check that your local disk has enough remaining free space to store the Docker image you are pulling. You can also delete old images to make room for new ones. Use the **docker images** command to see a list of all locally downloaded Docker images, along with their sizes.

Client cannot connect to the remote repository due to network error

> Calls to an Amazon ECR repository require a functioning connection to the internet. Verify your network settings, and verify that other tools and applications can access resources on the internet. If you are running **docker pull** on an Amazon EC2 instance in a private subnet, verify that the subnet has a route to the internet. Use a network address translation (NAT) server or a managed NAT gateway.
>
> Currently, calls to an Amazon ECR repository also require network access through your corporate firewall to Amazon Simple Storage Service (Amazon S3). If your organization uses firewall software or a NAT device that allows service endpoints, ensure that the Amazon S3 service endpoints for your current Region are allowed.
>
> If you are using Docker behind an HTTP proxy, you can configure Docker with the appropriate proxy settings. For more information, see HTTP proxy in the Docker documentation.

Error: "Filesystem Layer Verification Failed" When Pulling Images from Amazon ECR

You may receive the error `image image-name not found` when pulling images using the **docker pull** command. If you inspect the Docker logs, you may see an error like the following:

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

This error indicates that one or more of the layers for your image has failed to download. Some possible reasons and their explanations are given below.

You are using an older version of Docker

> This error can occur in a small percentage of cases when using a Docker version less than 1.10. Upgrade your Docker client to 1.10 or greater.

Your client has encountered a network or disk error

A full disk or a network issue may prevent one or more layers from downloading, as discussed earlier about the `Filesystem verification failed` message. Follow the recommendations above to ensure that your filesystem is not full, and that you have enabled access to Amazon S3 from within your network.

HTTP 403 Errors or "no basic auth credentials" Error When Pushing to Repository

There are times when you may receive an `HTTP 403 (Forbidden)` error, or the error message `no basic auth credentials` from the **docker push** or **docker pull** commands, even if you have successfully authenticated to Docker using the **aws ecr get-login-password** command. The following are some known causes of this issue:

You have authenticated to a different region

Authentication requests are tied to specific regions, and cannot be used across regions. For example, if you obtain an authorization token from US West (Oregon), you cannot use it to authenticate against your repositories in US East (N. Virginia). To resolve the issue, ensure that you have retrieved an authentication token from the same Region your repository exists in.

You have authenticated to push to a repository you don't have permissions for

You do not have the necessary permissions to push to the repository. For more information, see Repository policies (p. 19).

Your token has expired

The default authorization token expiration period for tokens obtained using the `GetAuthorizationToken` operation is 12 hours.

Bug in `wincred` credential manager

Some versions of Docker for Windows use a credential manager called `wincred`, which does not properly handle the Docker login command produced by **aws ecr get-login** (for more information, see https://github.com/docker/docker/issues/22910). You can run the Docker login command that is output, but when you try to push or pull images, those commands fail. You can work around this bug by removing the `https://` scheme from the registry argument in the Docker login command that is output from **aws ecr get-login**. An example Docker login command without the HTTPS scheme is shown below.

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

Troubleshooting Amazon ECR Error Messages

In some cases, an API call that you have triggered through the Amazon ECS console or the AWS CLI exits with an error message. Some common error messages and potential solutions are explained below.

Error: "Error Response from Daemon: Invalid Registry Endpoint" When Running aws ecr get-login

You may see the following error when running the **aws ecr get-login** command to obtain the login credentials for your Amazon ECR repository:

```
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
```

This error can occur on MacOS X and Windows systems that are running Docker Toolbox, Docker for Windows, or Docker for Mac. It is often caused when other applications alter the routes through the local gateway (192.168.0.1) through which the virtual machine must call to access the Amazon ECR service. If this error occurs when using Docker Toolbox, then it can often be resolved by restarting the Docker Machine environment, or rebooting the local client operating system. If this does not resolve the issue, use the **docker-machine ssh** command to log in to your container instance. Perform a DNS lookup on an external host to verify that you see the same results as you see on your local host. If the results differ, consult the documentation for Docker Toolbox to ensure that your Docker Machine environment is configured properly.

HTTP 429: Too Many Requests or ThrottleException

You may receive a `429: Too Many Requests` error or a `ThrottleException` error from one or more Amazon ECR commands or API calls. If you are using Docker tools with Amazon ECR, then for Docker versions 1.12.0 and greater, you may see the error message `TOOMANYREQUESTS: Rate exceeded`. For versions of Docker below 1.12.0, you may see the error `Unknown: Rate exceeded`.

This indicates that you are calling a single endpoint in Amazon ECR repeatedly over a short interval, and that your requests are getting throttled. Throttling occurs when calls to a single endpoint from a single user exceed a certain threshold over a period of time.

Various API operations in Amazon ECR have different throttles.

For example, the throttle for the `GetAuthorizationToken` action is 20 transaction per second (TPS), with up to a 200 TPS burst allowed. In each region, each account receives a bucket that can store up to 200 `GetAuthorizationToken` credits. These credits are replenished at a rate of 20 per second. If your bucket has 200 credits, you could achieve 200 `GetAuthorizationToken` API transactions per second for one second, and then sustain 20 transactions per second indefinitely.

To handle throttling errors, implement a retry function with incremental backoff into your code. For more information, see Error Retries and Exponential Backoff in AWS in the Amazon Web Services General Reference.

HTTP 403: "User [arn] is not authorized to perform [operation]"

You may receive the following error when attempting to perform an action with Amazon ECR:

```
$ 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: *
```

This indicates that your user does not have permissions granted to use Amazon ECR, or that those permissions are not set up correctly. In particular, if you are performing actions against an Amazon ECR repository, verify that the user has been granted permissions to access that repository. For

more information about creating and verifying permissions for Amazon ECR, see Identity and Access Management for Amazon Elastic Container Registry (p. 56).

HTTP 404: "Repository Does Not Exist" Error

If you specify a Docker Hub repository that does not currently exist, Docker Hub creates it automatically. With Amazon ECR, new repositories must be explicitly created before they can be used. This prevents new repositories from being created accidentally (for example, due to typos), and it also ensures that an appropriate security access policy is explicitly assigned to any new repositories. For more information about creating repositories, see Amazon ECR repositories (p. 16).

Troubleshooting Image Scanning Issues

The following are common image scan failures. You can view errors like this in the Amazon ECR console by displaying the image details or through the API or AWS CLI by using the `DescribeImageScanFindings` API.

UnsupportedImageError

You may get an `UnsupportedImageError` error when attempting to scan an image that was built using an operating system that Amazon ECR doesn't support image scanning for. Amazon ECR supports package vulnerability scanning for major versions of Amazon Linux, Amazon Linux 2, Debian, Ubuntu, CentOS, Oracle Linux, Alpine, and RHEL Linux distributions. Once a distribution loses support from its vendor, Amazon ECR may no longer support scanning it for vulnerabilities. Amazon ECR does not support scanning images built from the Docker scratch image.

An `UNDEFINED` severity level is returned

You may receive a scan finding that has a severity level of `UNDEFINED`. The following are the common causes for this:

- The vulnerability was not assigned a priority by the CVE source.
- The vulnerability was assigned a priority that Amazon ECR did not recognize.

To determine the severity and description of a vulnerability, you can view the CVE directly from the source.

Document History

The following table describes the important changes to the documentation since the last release of Amazon ECR. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
OCI artifact support	Amazon ECR added support for pushing and pulling Open Container Initiative (OCI) artifacts. A new parameter `artifactMediaType` was added to the `DescribeImages` API response to indicate the type of artifact. For more information, see Pushing a Helm chart (p. 30).	24 August 2020
Encryption at rest	Amazon ECR added support for configuring encryption for your repositories using server-side encryption with customer master keys (CMKs) stored in AWS Key Management Service (AWS KMS). For more information, see Encryption at rest (p. 71).	29 July 2020
Multi-architecture images	Amazon ECR added support for creating and pushing Docker manifest lists which are used for multi-architecture images. For more information, see Pushing a multi-architecture image (p. 29).	28 April 2020
Amazon ECR Usage Metrics	Amazon ECR added CloudWatch usage metrics which provides visiblity into your account's resource usage. You also have the ability to create CloudWatch alarms from both the CloudWatch and Service Quotas consoles to get alerts when your usage approaches your applied service quota. For more information, see Amazon ECR usage metrics (p. 84).	28 Feb 2020
Updated Amazon ECR service quotas	Updated the Amazon ECR service quotas to include per-API quotas. For more information, see Amazon ECR service quotas (p. 96).	19 Feb 2020
Added `get-login-password` command	Added support for **get-login-password**, which provides a simple and secure method for retrieving an authorization token. For more information, see Using an authorization token (p. 13).	4 Feb 2020
Image Scanning	Added support for image scanning, which helps in identifying software vulnerabilities in your container images. Amazon ECR uses the Common Vulnerabilities	24 Oct 2019

Change	Description	Date
	and Exposures (CVEs) database from the open source CoreOS Clair project and provides you with a list of scan findings. For more information, see Image scanning (p. 47).	
VPC Endpoint Policy	Added support for setting an IAM policy on the Amazon ECR interface VPC endpoints. For more information, see Create an endpoint policy for your Amazon ECR VPC endpoints (p. 81).	26 Sept 2019
Image Tag Mutability	Added support for configuring a repository to be immutable to prevent image tags from being overwritten. For more information, see Image tag mutability (p. 46).	25 July 2019
Interface VPC Endpoints (AWS PrivateLink)	Added support for configuring interface VPC endpoints powered by AWS PrivateLink. This allows you to create a private connection between your VPC and Amazon ECR without requiring access over the Internet, through a NAT instance, a VPN connection, or AWS Direct Connect. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) (p. 77).	25 Jan 2019
Resource tagging	Amazon ECR added support for adding metadata tags to your repositories. For more information, see Tagging an Amazon ECR repository (p. 24).	18 Dec 2018
Amazon ECR Name Change	Amazon Elastic Container Registry is renamed (previously Amazon EC2 Container Registry).	21 Nov 2017
Lifecycle Policies	Amazon ECR lifecycle policies enable you to specify the lifecycle management of images in a repository. For more information, see Lifecycle policies (p. 35).	11 Oct 2017
Amazon ECR support for Docker image manifest 2, schema 2	Amazon ECR now supports Docker Image Manifest V2 Schema 2 (used with Docker version 1.10 and newer). For more information, see Container image manifest formats (p. 50).	27 Jan 2017
Amazon ECR General Availability	Amazon Elastic Container Registry (Amazon ECR) is a managed AWS Docker registry service that is secure, scalable, and reliable.	21 Dec 2015

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.