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

Note
Amazon ECR supports public container image repositories as well. For more information, see What is Amazon ECR Public in the *Amazon ECR Public User Guide*.

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 private registry is provided to each AWS account; you can create one or more repositories in your registry and store images in them. For more information, see Amazon ECR private registry (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 Private registry authentication (p. 13).

Repository

An Amazon ECR repository contains your Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts. For more information, see Amazon ECR private repositories (p. 21).

Repository policy

You can control access to your repositories and the images within them with repository policies. For more information, see Private repository policies (p. 24).

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. 79) and Using Amazon ECR Images with Amazon EKS (p. 81).

Features of Amazon ECR

Amazon ECR provides the following features:

- Lifecycle policies help with managing the lifecycle of the images in your repositories. You define rules that result in the cleaning up of unused images. You can test rules before applying them to your repository. For more information, see Lifecycle policies (p. 55).
- Image scanning helps in identifying software vulnerabilities in your container images. Each repository can be configured to **scan on push**. This 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. 67).
- Cross-Region and cross-account replication makes it easier for you to have your images where you need them. This is configured as a registry setting and is on a per-Region basis. For more information, see Private registry settings (p. 15).
- Pull through cache rules provide a way to cache repositories in remote public registries in your private Amazon ECR registry. Using a pull through cache rule, Amazon ECR will periodically reach out to the remote registry to ensure the cached image in your Amazon ECR private registry is up to date. For more information, see Using pull through cache rules (p. 42).

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 Using Amazon ECR with 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 those two services is similar, as Amazon ECR is an extension of both services. When using the AWS CLI with Amazon ECR, we recommend that you 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 an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

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

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

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

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

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

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create an administrative user

- For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center.

  For instructions, see Getting started in the *AWS IAM Identity Center User Guide*.

Sign in as the administrative user

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In 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 **Visibility settings**, choose **Private**.
4. For **Repository name**, specify a name for the repository.
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. 66).
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.

   Important
   Configuring image scanning at the repository level has been deprecated in favor of configuring it at the registry level. For more information, see Image scanning (p. 67).
7. For **KMS encryption**, choose whether to enable server-side encryption using AWS KMS keys stored in the AWS Key Management Service service. For more information about this feature, see Encryption at rest (p. 108).
8. 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 Using Amazon ECR with 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 using the command from the console in 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. Using the **docker build** command from the console in 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 using the **docker push** command in a terminal window.
6. Choose **Close**.

Using Amazon ECR with the AWS CLI

The following steps walk you through the steps needed to push a container image to a private Amazon ECR repository 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 Command Line Interface 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 macOS 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 Amazon 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 Amazon 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 public.ecr.aws/docker/library/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 Amazon 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 macOS 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.

The `get-login-password` is the preferred method for authenticating to an Amazon ECR private registry when using the AWS CLI. Ensure that you have configured your AWS CLI to interact with AWS. For more information, see AWS CLI configuration basics in the *AWS Command Line Interface User Guide*.

When passing the Amazon ECR authorization 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 or the get-login-password command is unavailable, ensure you are using the latest version of the AWS CLI. For more information on installing or upgrading to the latest version of the AWS CLI, 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-repository` 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-repository \
    --image-scanning-configuration scanOnPush=true \
    --region region
```

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.region.amazonaws.com/hello-
repository
```

3. Push the image.

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

Output:

```
The push refers to a repository [aws_account_id.dkr.ecr.region.amazonaws.com/hello-
repository] (len: 1)
e9ae3c220b23: Pushed
a6785352b25c: Pushed
0998bf8fb9e9: Pushed
0a85502c06c9: Pushed
latest: digest: sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636EXAMPLE 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.region.amazonaws.com/hello-repository:latest
```

Output:

```
latest: Pulling from hello-repository
0a85502c06c9: Pull complete
0998bf8fb9e9: Pull complete
a6785352b25c: Pull complete
e9ae3c220b23: Pull complete
Digest: sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636EXAMPLE
Status: Downloaded newer image for aws_account_id.dkr.region.amazonaws.com/hello-
repository: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-repository` repository with the image tag `latest`.

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

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-repository \
      --force \
      --region region
```

Amazon ECR private registry

An Amazon ECR private registry hosts your container images in a highly available and scalable architecture. You can use your private registry to manage private image repositories consisting of Docker and Open Container Initiative (OCI) images and artifacts. Each AWS account is provided with a default private Amazon ECR registry. For more information about Amazon ECR public registries, see Public registries in the *Amazon Elastic Container Registry Public User Guide*.

Private registry concepts

- The URL for your default private registry is `https://aws_account_id.dkr.ecr.us-west-2.amazonaws.com`.
- By default, your account has read and write access to the repositories in your private registry. However,users require permissions to make calls to the Amazon ECR APIs and to push or pull images to and from your private 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. 102).
- You must authenticate your Docker client to your private 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 Private registry authentication (p. 13).
- Private repositories can be controlled with both user access policies and repository policies. For more information about repository policies, see Private repository policies (p. 24).
- The repositories in your private registry can be replicated across Regions in your own private registry and across separate accounts by configuring replication for your private registry. For more information, see Private image replication (p. 49).

Private registry authentication

You can use the AWS Management Console, the AWS CLI, or the AWS SDKs to create and manage private 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. Even though you can use the Amazon ECR API to push and pull images, you're more likely to use the Docker CLI or a language-specific Docker library.

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

The registry authentication methods that are detailed in the following sections 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.

> **Note**
> The Amazon ECR Docker credential helper doesn't support multi-factor authentication (MFA) currently.

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 private registry with the CLI

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
```

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
   ```

   The output is as follows:

```
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"]}
```

Private registry settings

Amazon ECR uses private registry settings to configure features at the registry level. The private registry settings are configured separately for each Region. You can use private registry settings to configure the following features.

- **Registry permissions**—You can use your registry permissions policy to grant permissions to an AWS principal to the replication and pull through cache features. For more information, see Private registry permissions (p. 15).
- **Pull through cache rules**—You can create pull through cache rules to cache images from an external public registry in your Amazon ECR private registry. For more information, see Using pull through cache rules (p. 42).
- **Replication**—You can configure repositories for either cross-Region or cross-account replication. For more information, see Private image replication (p. 49)
- **Scanning configuration**—By default, your registry is enabled for basic scanning. You may enable enhanced scanning which provides an automated, continuous scanning mode that scans for both operating system and programming language package vulnerabilities. For more information, see Image scanning (p. 67).

Private registry permissions

Amazon ECR uses a **registry policy** to grant permissions to an AWS principal at the private registry level. These permissions are used to scope access to the replication and pull through cache features.

Amazon ECR only enforces the following permissions at the private registry level. If any additional actions are added to the registry policy, an error will occur.

- `ecr:ReplicateImage` – Grants permission to another account, referred to as the source registry, to replicate its images to your registry. This is only used for cross-account replication.
- `ecr:BatchImportUpstreamImage` – Grants permission to retrieve the external image and import it to your private registry.
- `ecr:CreateRepository` – Grants permission to create a repository in a private registry. This permission is required if the repository storing either the replicated or cached images doesn't already exist in the private registry.

> **Note**
> While it is possible to add the `ecr:*` action to a private registry permissions policy, it is considered best practice to only add the specific actions required based on the feature you're using rather than use a wildcard.

Topics

- Setting a private registry permission statement (p. 16)

Setting a private registry permission statement

You can add or update the permissions policy for your registry by using the following steps. You can add multiple policy statements per registry. For example policies, see Private registry policy examples (p. 18).

Topics

Private registry permissions for replication

The cross account policy type is used to grant permissions to an AWS principal, allowing the replication of the repositories from a source registry to your registry. By default, you have permission to configure cross-Region replication within your own registry. You only need to configure the registry policy if you're granting another account permission to replicate contents to your registry.

A registry policy must grant permission for the `ecr:ReplicateImage` API action. This API is an internal Amazon ECR API that can replicate images between Regions or accounts. You can also grant permission for the `ecr:CreateRepository` permission, which allows Amazon ECR to create repositories in your registry if they don't exist already. If the `ecr:CreateRepository` permission isn't provided, a repository with the same name as the source repository must be created manually in your registry. If neither is done, replication fails. Any failed CreateRepository or ReplicateImage API actions show up in CloudTrail.

To configure a permissions policy for replication (AWS Management Console)

To configure a replication permissions policy for a private registry (AWS Management Console)

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, choose the Region to configure your registry policy in.
3. In the navigation pane, choose **Private registry**, **Registry permissions**.
4. On the **Registry permissions** page, choose **Generate statement**.
5. Complete the following steps to define your policy statement using the policy generator.
   a. For **Policy type**, choose **Cross account policy**.
   b. For **Statement ID**, enter a unique statement ID. This field is used as the `Sid` on the registry policy.
   c. For **Accounts**, enter the account IDs for each account you want to grant permissions to. When specifying multiple account IDs, separate them with a comma.
6. Expand the **Preview policy statement** section to review the registry permissions policy statement.
7. After the policy statement is confirmed, choose **Add to policy** to save the policy to your registry.

To configure a permissions policy for replication (AWS CLI)

To configure a permissions policy for a private registry (AWS CLI)

1. Create a file named `registry_policy.json` and populate it with a registry policy.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/*"
            ]
        }
    ]
}
```

2. Create the registry policy using the policy file.

```
aws ecr put-registry-policy \
      --policy-text file://registry_policy.json \
      --region us-west-2
```

3. Retrieve the policy for your registry to confirm.

```
aws ecr get-registry-policy \
      --region us-west-2
```

Private registry permissions for pull through cache

Amazon ECR private registry permissions may be used to scope the permissions of individual IAM entities to use pull through cache. If an IAM entity has more permissions granted by an IAM policy than the registry permissions policy is granting, the IAM policy takes precedence.

To create a private registry permissions policy (AWS Management Console)

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, choose the Region to configure your private registry permissions statement in.
3. In the navigation pane, choose **Private registry**, **Registry permissions**.
4. On the **Registry permissions** page, choose **Generate statement**.
5. For each pull through cache permissions policy statement you want to create, do the following.
    a. For **Policy type**, choose **Pull through cache policy**.
    b. For **Statement id**, provide a name for the pull through cache statement policy.
    c. For **IAM entities**, specify the users, groups, or roles to include in the policy.
    d. For **Repository namespace**, select the pull through cache rule to associate the policy with.
    e. For **Repository names**, specify the repository base name to apply the rule for. For example, if you want to specify the Amazon Linux repository on Amazon ECR Public, the repository name would be `amazonlinux`.

Deleting a private registry permission statement

You can delete all permissions policy statements for your registry by using the following steps.

To delete a permissions policy for a private registry (AWS Management Console)

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, choose the Region to configure your registry permissions policy in.
3. In the navigation pane, choose **Registries**.
4. On the **Registries** page, select your **Private** registry and choose **Permissions**.
5. On the **Private registry permissions** page, choose **Delete**.
6. On the **Delete registry policy** confirmation screen, choose **Delete policy**.

To delete a permissions policy for a private registry (AWS CLI)

1. Delete the registry policy.

```
aws ecr delete-registry-policy \
     --region us-west-2
```

2. Retrieve the policy for your registry to confirm.

```
aws ecr get-registry-policy \
     --region us-west-2
```

Private registry policy examples

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

> **Note**
> In each example, if the `ecr:CreateRepository` action is removed from your registry permission statement, replication can still occur. However, for successful replication, you need to create repositories with the same name within your account.

Example: Allow the root user of a source account to replicate all repositories

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/*"
            ]
```

```
        }
    ]
}
```

Example: Allow multiple accounts

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/*"
            ]
        },
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/*"
            ]
        }
    ]
}
```

Example: Allow the root user of a source account to replicate all repositories with prefix `prod-`.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ReplicationAccessCrossAccount",
            "Effect":"Allow",
            "Principal":{
                "AWS":"arn:aws:iam::source_account_id:root"
            },
            "Action":[
                "ecr:CreateRepository",
                "ecr:ReplicateImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-west-2:your_account_id:repository/prod-*"
            ]
        }
    ]
```

```
}
```

Amazon ECR private 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, so that you push and pull images from your development environments to your repositories.

Topics

Private 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, users require permissions to make calls to the Amazon ECR APIs and to push or pull images to and 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. 102).
- Repositories can be controlled with both user access policies and individual repository policies. For more information, see Private repository policies (p. 24).
- 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 can use the `team-a` namespace, and Team B can use the `team-b` namespace. By doing this, each team has their own image called `web-app` with each image prefaced with the team namespace. This configuration allows these images on each team to be used simultaneously without interference. Team A's image is `team-a/web-app`, and Team B's image is `team-b/web-app`.
- Your images can be replicated to other repositories across Regions in your own registry and across accounts. You can do this by specifying a replication configuration in your registry settings. For more information, see Private registry settings (p. 15).

Creating a private repository

Your container images are stored in Amazon ECR repositories. Use the following steps to create a private repository using the AWS Management Console. For steps to create a repository using the AWS CLI, see Step 3: Create a repository (p. 10).

To create a repository (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 **Private** tab, and then choose **Create repository**.
5. For **Visibility settings**, verify that **Private** is selected.

6. For **Repository name**, enter a unique name for your repository. The repository name can be specified on its own (for example `nginx-web-app`). Alternatively, it can be prepended with a namespace to group the repository into a category (for example `project-a/nginx-web-app`).

   Note
   The repository name may container a maximum of 256 characters. The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes. Using a double hyphen, double underscore, or double forward slash isn't supported.

7. For **Tag immutability**, choose the tag mutability setting for the repository. Repositories configured with immutable tags prevent image tags from being overwritten. For more information, see Image tag mutability (p. 66).
8. For **Scan on push**, while you can specify the scan settings at the repository level for basic scanning, it is best practice to specify the scan configuration at the private registry level. Specify the scanning settings at the private registry allow you to enable either enhanced scanning or basic scanning as well as define filters to specify which repositories are scanned. For more information, see Image scanning (p. 67).
9. 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 key (KMS key) with the alias `aws/ecr`. This key 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. 108).
10. When KMS encryption is enabled, select **Customer encryption settings (advanced)** to choose your own KMS key. The KMS key must be 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.
11. Choose **Create repository**.
12. (Optional) Select the repository that you created and choose **View push commands** to view the steps to push an image to your new repository. For more information about pushing an image to your repository, see Pushing an image (p. 33).

Viewing private repository details

After you created a repository, you can view details about the repository in the AWS Management Console:

- Which images are stored in a repository
- Details about each image stored in the repository, including the size and SHA digest for each image
- The scan frequency specified for the contents of the repository
- Whether the repository has an active pull through cache rule associated with it
- The encryption setting for the repository

Note
Starting 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. Therefore, keep in mind that Docker might return a larger image than the image shown in the AWS Management Console.

To view repository information (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 repository to view.
3. In the navigation pane, choose **Repositories**.

4. On the **Repositories** page, choose the **Private** tab and then the repository to view.
5. On the repository detail page, the console defaults to the **Images** view. Use the navigation menu to view other information about the repository.
   - Choose **Summary** to view the repository details and pull count data for the repository.
   - Choose **Images** to view information about the image tags in the repository. To view more information about the image, select the image tag. For more information, see Viewing image details (p. 41).

     If there are untagged images that you want 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. 47).
   - Choose **Permissions** to view the repository policies that are applied to the repository. For more information, see Private repository policies (p. 24).
   - 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. 55).
   - Choose **Tags** to view the metadata tags that are applied to the repository.

Editing a private repository

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

To edit a repository (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 repository to edit.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the **Private** tab and then 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 prevent image tags from being overwritten. For more information, see Image tag mutability (p. 66).
6. For **Image scan settings**, while you can specify the scan settings at the repository level for basic scanning, it is best practice to specify the scan configuration at the private registry level. Specify the scanning settings at the private registry allow you to enable either enhanced scanning or basic scanning as well as define filters to specify which repositories are scanned. For more information, see Image scanning (p. 67).
7. For **Encryption settings**, this is a view only field as the encryption settings for a repository can't be changed once the repository is created.
8. Choose **Save** to update the repository settings.

Deleting a private repository

If you're finished 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 (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 repository to delete.

3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the **Private** tab and then 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.

Private repository policies

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

Topics

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 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. 102).

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 user to describe the repository and the images within the repository.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ECRRepositoryPolicy",
            "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. 90).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDescribeRepoImage",
            "Effect": "Allow",
            "Action": [
                "ecr:DescribeImages",
                "ecr:DescribeRepositories"
            ],
            "Resource": ["arn:aws:ecr:region:account-id:repository/repository-name"]
        }
    ]
}
```

Setting a private 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 Private repository policy examples (p. 26).

> **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. 102).

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 to view the contents of the repository.
5. From the repository image list view, in the navigation pane, choose **Permissions**, **Edit**.

   > **Note**
   > If you don't see the **Permissions** option in the navigation pane, ensure that you are in the repository image list view.
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.

  Important
  The account you are granting permissions to must have the Region you are creating the repository policy in enabled, otherwise an error will occur.
- 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 private 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**.

Private repository policy examples

The following examples show policy statements that you could use to control the permissions that authenticated 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. 102).

Example: Allow one or more users

The following repository policy allows one or more users to push and pull images to and from a repository.

```
{
    "Version": "2012-10-17",
    "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:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "ecr:CompleteLayerUpload",
                "ecr:GetDownloadUrlForLayer",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ]
        }
    ]
}
```

Example: Allow another account

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

Important
The account you are granting permissions to must have the Region you are creating the repository policy in enabled, otherwise an error will occur.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCrossAccountPush",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:root"
            },
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:CompleteLayerUpload",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ]
        }
    ]
}
```

The following repository policy allows some 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": "2012-10-17",
    "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:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ]
        },
        {
            "Sid": "AllowAll",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:user/admin-user"
            },
            "Action": [
                "ecr:*"
            ]
        }
    ]
}
```

Example: Deny all

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyPull",
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ]
        }
    ]
}
```

Example: Restricting access to specific IP addresses

The following example denies permissions to any user to perform any Amazon ECR operations when applied to a repository from a specific range of addresses.

The condition in this statement identifies the `54.240.143.*` range of allowed Internet Protocol version 4 (IPv4) IP addresses.

The `Condition` block uses the `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": "Deny",
            "Principal": "*",
            "Action": "ecr:*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "54.240.143.0/24"
                }
            }
        }
    ]
}
```

Example: Allow an AWS service

The following repository policy allows AWS CodeBuild access to the Amazon ECR API actions necessary for integration with that service. When using the following example, you should use the `aws:SourceArn` and `aws:SourceAccount` condition keys to scope which resources can assume these permissions. 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"
         ],
         "Condition":{
            "ArnLike":{
               "aws:SourceArn":"arn:aws:codebuild:region:123456789012:project/project-name"
            },
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            }
         }
      }
   ]
}
```

Tagging a private repository

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

Tag basics

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and a *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 the owner of each repository.

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. 122).

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 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 (AWS Management Console)

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, select the check box next to the repository you want to tag.
5. From the **Action** menu, select **Repository tags**.
6. On the **Repository tags** page, select **Add tags**, **Add tag**.
7. On the **Edit repository tags** page, specify the key and value for each tag, and then choose **Save**.

To delete a tag from an individual resource (AWS Management Console)

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, select the check box next to the repository you want to remove a tag from.
4. From the **Action** menu, select **Repository tags**.
5. On the **Repository tags** page, select **Edit**.
6. On the **Edit repository 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
```

Private images

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

Topics

Pushing an image

You can push your Docker images, manifest lists, and Open Container Initiative (OCI) images and compatible artifacts to your private repositories. The following pages describe these in more detail.

Amazon ECR also provides a way to replicate your images to other repositories, across Regions in your own registry and across different accounts, by specifying a replication configuration in your private registry settings. For more information, see Private registry settings (p. 15).

Topics

Required IAM permissions for pushing an image

Amazon ECR requires that users have the following permissions to push images. Following the best practice of granting least privilege, you can scope these permissions down to a specific repository or you can grant the permissions for all repositories. A user must authenticate to each Amazon ECR registry they want to push images to by requesting an authorization token. 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. 102).

The following IAM policy grants the required permissions for pushing an image without scoping to a specific repository.

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

The following IAM policy grants the required permissions for pushing an image and scopes to a specific repository. The repository must be specified as a full Amazon Resource Name (ARN).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:CompleteLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:InitiateLayerUpload",
                "ecr:BatchCheckLayerAvailability",
                "ecr:PutImage"
            ],
            "Resource": "arn:aws:ecr:region:111122223333:repository/repository-name"
        },
        {
            "Effect": "Allow",
            "Action": "ecr:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```

Pushing a Docker image

You can push your container images to an Amazon ECR repository with the **docker push** command. 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. 35).

To push a Docker image to an Amazon ECR repository

The Amazon ECR repository must exist before you push the image. For more information, see the section called “Creating a repository” (p. 21).

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 Private registry authentication (p. 13).

To authenticate Docker to an Amazon ECR registry, 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*.

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

2. If your image repository doesn't exist in the registry you intend to push to yet, create it. For more information, see Creating a private repository (p. 21).
3. Identify the local image to push. Run the **docker images** command to list the container 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.*us-west-2*.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 a local image with the ID *e9ae3c220b23* as *aws_account_id*.dkr.ecr.*us-west-2*.amazonaws.com/my-repository:tag.

```
docker tag e9ae3c220b23 aws_account_id.dkr.ecr.us-west-2.amazonaws.com/my-repository:tag
```

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

```
docker push aws_account_id.dkr.ecr.us-west-2.amazonaws.com/my-repository:tag
```

6. (Optional) Apply any additional tags to your image and push those tags to Amazon ECR by repeating Step 4 (p. 35) and Step 5 (p. 35).

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. In most cases, the manifest list is created from images that serve the same function but for different operating systems or architectures. The manifest list isn't required. For more information, see docker manifest.

Important
Your Docker CLI must have experimental features turned on 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 about how to push an image, see Pushing a Docker image (p. 34).

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

The Amazon ECR repository must exist before you push the image. For more information, see the section called “Creating a repository” (p. 21).

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 Private registry authentication (p. 13).

   To authenticate Docker to an Amazon ECR registry, 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*.

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

2. List the images in your repository, confirming the image tags.

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

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.us-west-2.amazonaws.com/my-repository aws_account_id.dkr.ecr.us-west-2.amazonaws.com/my-repository:image_one_tag aws_account_id.dkr.ecr.us-west-2.amazonaws.com/my-repository: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.us-west-2.amazonaws.com/my-repository
   ```

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

   ```
   docker manifest push aws_account_id.dkr.ecr.us-west-2.amazonaws.com/my-repository
   ```

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. 82).

To push a Helm chart to an Amazon ECR repository

1. Install the latest version of the Helm client. These steps were written using Helm version `3.8.2`. For more information, see Installing Helm.
2. Use the following steps to create a test Helm chart. For more information, see Helm Docs - Getting Started.

   a. Create a Helm chart named `helm-test-chart` and clear the contents of the `templates` directory.

   ```
   helm create helm-test-chart
   rm -rf ./helm-test-chart/templates/*
   ```

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

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

3. Package the chart. The output will contain the filename of the packaged chart which you use when pushing the Helm chart.

   ```
   cd ../..
   helm package helm-test-chart
   ```

   Output

   ```
   Successfully packaged chart and saved it to: /Users/username/helm-test-chart-0.1.0.tgz
   ```

4. Create a repository to store your Helm chart. The name of your repository should match the name you used when creating the Helm chart in step 2. For more information, see Creating a private repository (p. 21).

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

5. 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 Private 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.us-west-2.amazonaws.com
   ```

6. Push the Helm chart using the **helm push** command. The output should include the Amazon ECR repository URI and SHA digest.

   ```
   helm push helm-test-chart-0.1.0.tgz oci://aws_account_id.dkr.ecr.us-
   west-2.amazonaws.com/
   ```

7. Describe your Helm chart.

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

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

```
{
    "imageDetails": [
        {
            "registryId": "aws_account_id",
            "repositoryName": "helm-test-chart",
            "imageDigest":
 "sha256:dd8aebdda7df991a0ffe0b3d6c0cf315fd582cd26f9755a347a52adEXAMPLE",
            "imageTags": [
                "0.1.0"
            ],
            "imageSizeInBytes": 1620,
            "imagePushedAt": "2021-09-23T11:39:30-05:00",
            "imageManifestMediaType": "application/vnd.oci.image.manifest.v1+json",
            "artifactMediaType": "application/vnd.cncf.helm.config.v1+json"
        }
    ]
}
```

8. (Optional) For additional steps, install the Helm configmap and get started with Amazon EKS. For more information, see Installing a Helm chart hosted on Amazon ECR with Amazon EKS (p. 82).

Signing an image

Amazon ECR integrates with AWS Signer to provide a way for you to sign your container images. You can store both your container images and the signatures in your private repositories.

Considerations

The following should be considered when using Amazon ECR image signing.

- Signatures stored in your repository count against the service quota for the maximum number of images per repository. For more information, see Amazon ECR service quotas (p. 135).
- When using Amazon ECR lifecycle policies, any action by a rule to expire or delete an OCI image index will result in Amazon ECR deleting any signatures referenced by that image index within 24 hours.

Prerequisites

Before you begin, The following prerequisites must be met.

- Install and configure the latest version of the AWS CLI. For more information, see Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.
- Install the Notation CLI and the AWS Signer plugin for Notation. For more information, see Prerequisites for signing container images in the *AWS Signer Developer Guide*.
- Have a container image stored in an Amazon ECR private repository to sign. For more information, see Pushing an image (p. 33).

Configure authentication for the Notary client

Before you can create a signature using the Notation CLI, you must configure the client so it can authenticate to Amazon ECR. If you have Docker installed on the same host where you install the Notation client, then Notation will reuse the same authentication method you use for the Docker client. The Docker `login` and `logout` commands will allow the Notation `sign` and `verify` commands to use those same credentials, and you don't have to separately authenticate Notation. For more information on configuring your Notation client for authentication, see Authenticate with OCI-compliant registries in the Notary Project documentation

If you are not using Docker or another tool that uses Docker credentials, then we recommend using the Amazon ECR Docker Credential Helper as your credential store. For more information on how to install and configure the Amazon ECR Credential Helper, see Amazon ECR Docker Credential Helper.

Signing an image

The following steps can be used to create the resources necessary to sign a container image and store the signature in an Amazon ECR private repository. Notation signs images using the digest.

To sign an image

1. Create an AWS Signer signing profile using the `Notation-OCI-SHA384-ECDSA` signing platform. You can optionally specify a signature validity period using the `--signature-validity-period` parameter. This value may be specified using `DAYS`, `MONTHS`, or `YEARS`. If no validity period is specified, the default value of 135 months is used.

   ```
   aws signer put-signing-profile --profile-name ecr_signing_profile --platform-id
    Notation-OCI-SHA384-ECDSA
   ```

   Note
   The signing profile name only supports alphanumeric characters and the underscore (_).

2. Authenticate the Notation client to your default registry. The following example uses the AWS CLI to authenticate the Notation CLI to an Amazon ECR private registry.

   ```
   aws ecr get-login-password --region region | notation login --username AWS --password-
   stdin 111122223333.dkr.ecr.region.amazonaws.com
   ```

3. Use the Notation CLI to sign the image, specifying the image using the repository name and the SHA digest. This creates the signature and pushes it to the same Amazon ECR private repository that the image being signed is in.

   In the following example, we are signing an image in the `curl` repository with SHA digest `sha256:ca78e5f730f9a789ef8c63bb55275ac12dfb9e8099e6EXAMPLE`.

   ```
   notation
    sign 111122223333.dkr.ecr.region.amazonaws.com/
   curl@sha256:ca78e5f730f9a789ef8c63bb55275ac12dfb9e8099e6EXAMPLE --plugin
    "com.amazonaws.signer.notation.plugin" --id "arn:aws:signer:region:111122223333:/
   signing-profiles/ecrSigningProfileName"
   ```

Verify an image locally after signing

After you sign a container image using AWS Signer and Notation, you or an authorized member of your team can verify the origin and integrity of the image by cryptographic means.

Complete the following steps to verify that an image is valid with Notation.

To verify an image

1. A trust store is required for verification. If you used the installer for the AWS Signer plugin and Notation, a trust store was set up automatically and provisioned with a root certificate.
2. Set up a trust policy similar to the one below, modifying as needed the names of the signing profiles you are using to verify images.

```
{
   "version":"1.0",
   "trustPolicies":[
      {
         "name":"aws-signer-tp",
         "registryScopes":[
            "*"
         ],
         "signatureVerification":{
            "level":"strict"
         },
         "trustStores":[
            "signingAuthority:aws-signer-ts"
         ],
         "trustedIdentities":[
            "arn:aws:signer:region:111122223333:/signing-profiles/ecr_signing_profile",
            "arn:aws:signer:region:111122223333:/signing-profiles/ecr_signing_profile2"
         ]
      }
   ]
}
```

3. Import the policy into Notation.

```
$ notation policy import mypolicy.json
```

Output:

```
Existing trust policy configuration found, do you
want to overwrite it? [y/N] y

Trust policy configuration imported successfully.
```

4. Verify the signature, specifying the signature using the repository name and the SHA digest.

```
$ notation verify 111122223333.dkr.ecr.region.amazonaws.com/curl@SHA256_digest
```

Output:

```
Successfully verified signature for 111122223333.dkr.ecr.us-
west-2.amazonaws.com/curl@SHA256_digest
```

Deleting a signature

When you create and push a signature using the Notation CLI, an OCI image index is created in your Amazon ECR repository as well. The Amazon ECR API doesn't support deleting artifacts or images referred to by an OCI image index, so the following are the available options to clean up these artifacts.

- (Recommended) You can use the ORAS CLI to delete the artifact and ORAS will handle updating or deleting the image index.
- You can use the Amazon ECR API or console to delete the OCI image index first and then the referenced artifact such as the signature.

When using the ORAS client to delete signatures and other reference type artifacts, ORAS manages the OCI image index. ORAS will first remove the reference to the artifact from the index, and then will delete the manifest. The `oras manifest delete` command can be used, referencing the index of the signature artifact. For more information on installing and configuring the ORAS client, see ORAS CLI in the ORAS documentation.

The following example command can be used to delete a signature.

```
oras manifest
 delete 111122223333.dkr.ecr.region.amazonaws.com/
repository_name@sha256:ca78e5f730f9a789ef8c63bb55275ac12dfb9e8099e6EXAMPLE
```

Viewing image details

After you have pushed an image to your repository, you can view its information in the AWS Management Console. The details included are as follows:

- Image URI
- Image tags
- Artifact media type
- Image manifest type
- Scanning status
- The size of the image in MB
- When the image was pushed to the repository
- The replication status

To view image details (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 repository containing your image.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the repository to view.
5. On the **Repositories : *repository_name*** page, choose the image to view the details of.

Pulling an image

If you want 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. 79).

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. 102).

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 Private 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 might need to authenticate your Docker client with Amazon ECR. For more information, see Private registry authentication (p. 13).

Using pull through cache rules

Amazon ECR supports caching repositories in remote public registries in your private Amazon ECR registry. Amazon ECR currently supports creating pull through cache rules for Amazon ECR Public, Quay, and the Kubernetes container image registry. Once a pull through cache rule is created for an external public registry, simply pull an image from that external public registry using your Amazon ECR private registry URI and then Amazon ECR creates a repository and caches that image. When a cached image is pulled using the Amazon ECR private registry URI, Amazon ECR checks the remote registry to see if there is a new version of the image and will update your private registry up to one time every 24 hours.

Considerations for using pull through cache

The following should be considered when using Amazon ECR pull through cache.

- Creating pull through cache rules isn't supported in the following Regions:
  - China (Beijing) (`cn-north-1`)
  - China (Ningxia) (`cn-northwest-1`)
  - AWS GovCloud (US-East) (`us-gov-east-1`)

  - AWS GovCloud (US-West) (`us-gov-west-1`)
- When pulling images using pull through cache, the Amazon ECR FIPS service endpoints aren't supported the first time an image is pulled. Using the Amazon ECR FIPS service endpoints work on subsequent pulls though.
- You can create a maximum of 10 pull through cache rules for your private registry.
- When cached images are pulled through the Amazon ECR private registry URI, the image pulls are initiated by AWS IP addresses. This ensures that the image pull doesn't count against any pull rate quotas the public registry has.
- When a cached image is pulled through the Amazon ECR private registry URI, Amazon ECR checks the remote repository up to once per 24 hours to verify whether the cached image is the latest version. This timer is based off the last pull of the cached image.
- When a multi-architecture image is pulled using a pull through cache rule, the manifest list and each image referenced in the manifest list are pulled to the Amazon ECR repository. If you only want to pull a specific architecture, you can pull the image using the image digest or tag associated with the architecture rather than the tag associated with the manifest list.
- Amazon ECR uses a service-linked IAM role, which provides the permissions needed for Amazon ECR to create the repository for and push the cached image on your behalf. The service-linked IAM role is created automatically when a pull through cache rule is created. For more information, see Amazon ECR service-linked role for pull through cache (p. 99).
- By default, the user,, group, or role pulling the cached image has the permissions granted to them through their IAM policy. You may use the Amazon ECR private registry permissions policy to further scope the permissions of an IAM entity. For more information, see Using registry permissions (p. 44).
- Amazon ECR repositories created using the pull through cache workflow are treated like any other Amazon ECR repository. All repository features, such as replication and image scanning are supported.
- When a new repository is created using a pull through cache rule, tag immutability is disabled by default. If you manually turn on tag immutability on the repository, Amazon ECR may not be able to update the cached images.
- When a new repository is created using a pull through cache rule, AWS KMS encryption is disabled by default. If you want to use AWS KMS encryption, you can create the repository manually prior to the first image pull.
- When an image is pulled using a pull through cache rule for the first time, if you've configured Amazon ECR to use an interface VPC endpoint using AWS PrivateLink then you need to create a public subnet in the same VPC, with a NAT gateway, and then route all outbound traffic to the internet from their private subnet to the NAT gateway in order for the pull to work. Subsequent image pulls don't require this. For more information, see Scenario: Access the internet from a private subnet in the *Amazon Virtual Private Cloud User Guide*.

Required IAM permissions

In addition to the Amazon ECR API permissions needed to authenticate to a private registry and to push and pull images, the following additional permissions are needed to use pull through cache rules.

- `ecr:CreatePullThroughCacheRule` – Grants permission to create a pull through cache rule. This permission must be granted via an identity-based IAM policy.
- `ecr:BatchImportUpstreamImage` – Grants permission to retrieve the external image and import it to your private registry. This permission can be granted by using the private registry permissions policy, an identity-based IAM policy, or by using the resource-based repository permissions policy. For more information about using repository permissions, see Private repository policies (p. 24).
- `ecr:CreateRepository` – Grants permission to create a repository in a private registry. This permission is required if the repository storing the cached images doesn't already exist. This

permission can be granted by either an identity-based IAM policy or the private registry permissions policy.

Using registry permissions

Amazon ECR private registry permissions may be used to scope the permissions of individual IAM entities to use pull through cache. If an IAM entity has more permissions granted by an IAM policy than the registry permissions policy is granting, the IAM policy takes precedence. For example, if user has `ecr:*` permissions granted, no additional permissions are needed at the registry level.

To create a private registry permissions policy (AWS Management Console)

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, choose the Region to configure your private registry permissions statement in.
3. In the navigation pane, choose **Private registry**, **Registry permissions**.
4. On the **Registry permissions** page, choose **Generate statement**.
5. For each pull through cache permissions policy statement you want to create, do the following.
   a. For **Policy type**, choose **Pull through cache policy**.
   b. For **Statement id**, provide a name for the pull through cache statement policy.
   c. For **IAM entities**, specify the users, groups, or roles to include in the policy.
   d. For **Repository namespace**, select the pull through cache rule to associate the policy with.
   e. For **Repository names**, specify the repository base name to apply the rule for. For example, if you want to specify the Amazon Linux repository on Amazon ECR Public, the repository name would be `amazonlinux`.

To create a private registry permissions policy (AWS CLI)

Use the following AWS CLI command to specify the private registry permissions using the AWS CLI.

1. Create a local file named `ptc-registry-policy.json` with the contents of your registry policy. The following example grants the `ecr-pull-through-cache-user` permission to create a repository and pull an image from Amazon ECR Public, which is the upstream source associated with the previously created pull through cache rule.

```
{
  "Sid": "PullThroughCacheFromReadOnlyRole",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ecr-pull-through-cache-user"
  },
  "Action": [
    "ecr:CreateRepository",
    "ecr:BatchImportUpstreamImage"
  ],
  "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/ecr-public/*"
}
```

Important
The `ecr-CreateRepository` permission is only required if the repository storing the cached images doesn't already exist. For example, if the repository creation action and the image pull actions are being done by separate IAM principals such as an administrator and a developer.

2. Use the put-registry-policy command to set the registry policy.

```
aws ecr put-registry-policy \
     --policy-text file://ptc-registry.policy.json
```

Creating a pull through cache rule

You create a pull through cache rule for each external public registry containing images you want to cache in your Amazon ECR private registry.

To create a pull through cache rule (AWS Management Console)

To create a pull through cache rule (AWS Management Console)

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, choose the Region to configure your private registry settings in.
3. In the navigation pane, choose **Private registry**, **Pull through cache**.
4. On the **Pull through cache configuration** page, choose **Add rule**.
5. On the **Create pull through cache rule** page, do the following.
   a. For **Public registry**, choose one of the preconfigured public registries.
   b. For **Amazon ECR repository namespace**, specify the repository namespace to use when caching images pulled from the source public registry. By default, a namespace is populated but a custom namespace can be specified as well.
   c. Choose **Save** to save the pull through cache rule to your registry settings.
6. Repeat the previous step for each pull through cache you want to create. The pull through cache rules are created separately for each Region.

To create a pull through cache rule (AWS CLI)

Use the following AWS CLI commands to create a pull through cache rule for private registry using the AWS CLI.

- create-pull-through-cache-rule (AWS CLI)

  The following example creates a pull through cache rule for the Amazon ECR Public registry. It specifies a repository prefix of `ecr-public`, which results in each repository created using the pull through cache rule to have the naming scheme of `ecr-public/`*`upstream-repository-name`*.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix ecr-public \
     --upstream-registry-url public.ecr.aws \
     --region us-east-2
```

  The following example creates a pull through cache rule for the Quay public registry. It specifies a repository prefix of `quay`, which results in each repository created using the pull through cache rule to have the naming scheme of `quay/`*`upstream-repository-name`*.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix quay \
     --upstream-registry-url quay.io \
     --region us-east-2
```

The following example creates a pull through cache rule for the Kubernetes public registry. It specifies a repository prefix of `kubernetes`, which results in each repository created using the pull through cache rule to have the naming scheme of `kubernetes/`*`upstream-repository-name`*.

```
aws ecr create-pull-through-cache-rule \
     --ecr-repository-prefix kubernetes \
     --upstream-registry-url registry.k8s.io \
     --region us-east-2
```

Working with pull through cache images

After a pull through cache rule is created for an external public registry, simply pull the remote images using your Amazon ECR repository URI and the images are cached locally. The following are the formats for the supported public registries. If you receive an error pulling an upstream image using a pull through cache rule, see Troubleshooting pull through cache issues (p. 145) for the most common errors and how to resolve them.

Note
The following examples use the default Amazon ECR repository namespace values that the AWS Management Console uses. Ensure that you use the Amazon ECR private repository URI that you've configured.

Amazon ECR Public

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/ecr-public/repository_name/
image_name:tag
```

Quay

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/quay/repository_name/image_name:tag
```

Kubernetes

```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/kubernetes/repository_name/
image_name:tag
```

Deleting a pull through cache rule

You can delete a pull through cache rule to stop the caching behavior. Deleting a pull through cache rule doesn't have any effect on the repositories or images that were cached, it only stops future caching behavior.

To delete a pull through cache rule (AWS Management Console)

To delete a pull through cache rule (AWS Management Console)

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, choose the Region to configure your private registry settings in.
3. In the navigation pane, choose **Private registry**, **Pull through cache**.

4. On the **Pull through cache configuration** page, select the pull through cache rules to delete, and then choose **Delete rule**.
5. In the navigation pane, choose **Private registry**, **Registry permissions**.
6. (Optional) On the **Registry permissions** page, review the existing registry permissions policy statements. You may delete any registry permissions policy statements associated with the repository namespace for the deleted pull through cache rule.

To delete a pull through cache rule (AWS CLI)

Use the following AWS CLI commands to delete a pull through cache rule using the AWS CLI.

- delete-pull-through-cache-rule (AWS CLI)

  The following example deletes a pull through cache rule that uses the `ecr-public` repository prefix..

```
aws ecr delete-pull-through-cache-rule \
     --ecr-repository-prefix ecr-public \
     --region us-east-2
```

Deleting an image

If you're finished using an image, you can delete it from your repository. If you're finished with a repository, you can delete the entire repository and all of the images within it. For more information, see Deleting a private repository (p. 23).

As an alternative to deleting images manually, you can create repository lifecycle policies which provide more control over the lifecycle management of images in your repositories. Lifecycle policies automate this process for you. For more information, see Lifecycle policies (p. 55).

To delete an image (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 (AWS CLI)

1. List the images in your repository. Tagged images will have both an image digest as well as a list of associated tags. Untagged images will only have an image digest.

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

2. (Optional) Delete any unwanted tags for the image by specifying the tag associated with the image you want to delete. When the last tag is deleted from an image, the image is also deleted.

```
aws ecr batch-delete-image \
```

```
    --repository-name my-repo \
    --image-ids imageTag=tag1 imageTag=tag2
```

3. Delete a tagged or untagged image by specifying the image digest. 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=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304EXAMPLE
```

To delete multiple images, you can specify multiple image tags or image digests in the request.

```
aws ecr batch-delete-image \
    --repository-name my-repo \
    --image-ids imageDigest=sha256:4f70ef7a4d29e8c0c302b13e25962d8f7a0bd304EXAMPLE
 imageDigest=sha256:f5t0e245ssffc302b13e25962d8f7a0bd304EXAMPLE
```

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 a file. In this example, the manifest for an image with the tag, *`latest`*, in the repository, *`amazonlinux`*, is written to an environment variable named *`MANIFEST`*.

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

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 isn't 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 following output, the image has the tags `latest` and `2017.03`.

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

The output is as follows:

```
{
    "imageDetails": [
```

```
        {
            "imageSizeInBytes": 98755613,
            "imageDigest":
 "sha256:8d00af8f076eb15a33019c2a3e7f1f655375681c4e5be157a26EXAMPLE",
            "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 obtain 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 following output, the image has the tags `latest` and `2017.09`.

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

   The output is as follows:

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

Private image replication

You can configure your Amazon ECR private registry to support the replication of your repositories. Amazon ECR supports both cross-Region and cross-account replication. For cross-account replication to

occur, the destination account must configure a registry permissions policy to allow replication from the source registry to occur. For more information, see Private registry permissions (p. 15).

Topics

Considerations for private image replication

The following should be considered when using private image replication.

- Only repository content pushed to a repository after replication is configured is replicated. Any preexisting content in a repository isn't replicated. Once replication is configured for a repository, Amazon ECR keeps the destination and source synchronized.
- The first time you configure your private registry for replication, Amazon ECR creates a service-linked IAM role on your behalf. The service-linked IAM role grants the Amazon ECR replication service the permission it needs to create repositories and replicate images in your registry. For more information, see Using service-linked roles for Amazon ECR (p. 97).
- For cross-account replication to occur, the private registry destination must grant permission to allow the source registry to replicate its images. This is done by setting a private registry permissions policy. For more information, see Private registry permissions (p. 15).
- If the permission policy for a private registry are changed to remove a permission, any in-progress replications previously granted may complete.
- For cross-Region replication to occur, both the source and destination accounts must be opted-in to the Region prior to any replication actions occurring within or to that Region. For more information, see Managing AWS Regions in the *Amazon Web Services General Reference*.
- Cross-Region replication is not supported between AWS partitions. For example, a repository in `us-west-2` can't be replicated to `cn-north-1`. For more information about AWS partitions, see ARN format in the *AWS General Reference*.
- The replication configuration for a private registry may contain up to 25 unique destinations across all rules, with a maximum of 10 rules total. Each rule may contain up to 100 filters. This allows for specifying separate rules for repositories containing images used for production and testing, for example.
- The replication configuration supports filtering which repositories in a private registry are replicated by specifying a repository prefix. For an example, see Example: Configuring cross-Region replication using a repository filter (p. 53).
- A replication action only occurs once per image push. For example, if you configured cross-Region replication from `us-west-2` to `us-east-1` and from `us-east-1` to `us-east-2`, an image pushed to `us-west-2` replicates to only `us-east-1`, it doesn't replicate again to `us-east-2`. This behavior applies to both cross-Region and cross-account replication.
- The majority of images replicate in less than 30 minutes, but in rare cases the replication might take longer.
- Registry replication doesn't perform any delete actions. Replicated images and repositories can be manually deleted when they are no longer being used.
- Repository policies, including IAM policies, and lifecycle policies aren't replicated and don't have any effect other than on the repository they are defined for.
- Repository settings aren't replicated. The tag immutability, image scanning, and KMS encryption settings are disabled by default on all repositories created because of a replication action. The tag

immutability and image scanning setting can be changed after the repository is created. However, the setting only applies to images pushed after the setting has changed.

- If tag immutability is enabled on a repository and an image is replicated that uses the same tag as an existing image, the image is replicated but won't contain the duplicated tag. This might result in the image being untagged.

Configuring private image replication

Replication settings are configured separately for each Region. Use the following steps to configure replication for your private registry.

To configure registry replication settings (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 configure your registry replication settings for.
3. In the navigation pane, choose **Private registry**.
4. On the **Private registry** page, on the **Replication** section, choose **Edit**.
5. On the **Replication** page, choose **Add replication rule**.
6. On the **Destination types** page, choose whether to enable cross-Region replication, cross-account replication, or both and then choose **Next**.
7. If cross-Region replication is enabled, then for **Configure destination regions**, choose one or more **Destination regions** and then choose **Next**.
8. If cross-account replication is enabled, then for **Cross-account replication**, choose the cross-account replication setting for the registry. For **Destination account**, enter the account ID for the destination account and one or more **Destination regions** to replicate to. Choose **Destination account +** to configure additional accounts as replication destinations.

   Important
   For cross-account replication to occur, the destination account must configure a registry permissions policy to allow replication to occur. For more information, see Private registry permissions (p. 15).
9. (Optional) On the **Add filters** page, specify one or more filters for the replication rule and then choose **Add**. Repeat this step for each filter you want to associate with the replication action. A filter must be specified as a repository name prefix. If no filters are added, the contents of all repositories are replicated. Choose **Next** once all filters have been added.
10. On the **Review and submit** page, review the replication rule configuration and then choose **Submit rule**.

To configure registry replication settings (AWS CLI)

1. Create a JSON file containing the replication rules to define for your registry. A replication configuration may contain up to 10 rules, with up to 25 unique destinations across all rules and 100 filters per each rule. To configure cross-Region replication within your own account, you specify your own account ID. For more examples, see Private image replication examples (p. 52).

```
{
 "rules": [{
  "destinations": [{
   "region": "destination_region",
   "registryId": "destination_accountId"
  }],
  "repositoryFilters": [{
   "filter": "repository_prefix_name",
```

```
    "filterType": "PREFIX_MATCH"
  }]
 }]
}
```

2. Create a replication configuration for your registry.

```
aws ecr put-replication-configuration \
     --replication-configuration file://replication-settings.json \
     --region us-west-2
```

3. Confirm your registry settings.

```
aws ecr describe-registry \
     --region us-west-2
```

Viewing replication status

The replication status of an individual container image can be viewed by querying using either the image tag or image digest.

Checking replication status (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 is the source of your replicated registry.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, choose the repository to check the replication status of.
5. On the repository details page, choose the **Image tag** to check the replication status of.
6. For **Image replication status**, verify the replication status. You can view the replication status based on the image tag or image digest.

Checking replication status (AWS CLI)

- The replication status of the contents of a repository can be viewed based on the image tag using the following command.

```
aws ecr describe-image-replication-status \
     --repository-name repository_name \
     --image-id imageTag=image_tag \
     --region us-west-2
```

- The replication status of the contents of a repository can be viewed based on the image digest using the following command.

```
aws ecr describe-image-replication-status \
     --repository-name repository_name \
     --image-id imageDigest=image_digest \
     --region us-west-2
```

Private image replication examples

The following examples show how private image replication can be used.

Example: Configuring cross-Region replication to a single destination Region

The following shows an example for configuring cross-Region replication within a single registry. This example assumes that your account ID is 111122223333 and that you're specifying this replication configuration in a Region other than `us-west-2`.

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-2",
                    "registryId": "111122223333"
                }
            ]
        }
    ]
}
```

Example: Configuring cross-Region replication using a repository filter

The following shows an example for configuring cross-Region replication for repositories that match a prefix name value. This example assumes your account ID is 111122223333 and that you're specifying this replication configuration in a Region other than `us-west-1` and have repositories with a prefix of `prod`.

```
{
 "rules": [{
  "destinations": [{
   "region": "us-west-1",
   "registryId": "111122223333"
  }],
  "repositoryFilters": [{
   "filter": "prod",
   "filterType": "PREFIX_MATCH"
  }]
 }]
}
```

Example: Configuring cross-Region replication to multiple destination Regions

The following shows an example for configuring cross-Region replication within a single registry. This example assumes your account ID is 111122223333 and that you're specifying this replication configuration in a Region other than `us-west-1` or `us-west-2`.

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-1",
                    "registryId": "111122223333"
                },
                {
```

```
                    "region": "us-west-2",
                    "registryId": "111122223333"
                }
            ]
        }
    ]
}
```

Example: Configuring cross-account replication

The following shows an example for configuring cross-account replication for your registry. This example configures replication to the 444455556666 account and to the us-west-2 Region.

Important
For cross-account replication to occur, the destination account must configure a registry permissions policy to allow replication to occur. For more information, see Private registry permissions (p. 15).

```
{
    "rules": [
        {
            "destinations": [
                {
                    "region": "us-west-2",
                    "registryId": "444455556666"
                }
            ]
        }
    ]
}
```

Example: Specifying multiple rules in a configuration

The following shows an example for configuring multiple replication rules for your registry. This example configures replication for the *111122223333* account with one rule that replicates repositories with a prefix of prod to the us-west-2 Region and repositories with a prefix of test to the us-east-2 Region. A replication configuration may contain up to 10 rules, with each rule specifying up to 25 destinations.

```
{
 "rules": [{
   "destinations": [{
    "region": "us-west-2",
    "registryId": "111122223333"
   }],
   "repositoryFilters": [{
    "filter": "prod",
    "filterType": "PREFIX_MATCH"
   }]
  },
  {
   "destinations": [{
    "region": "us-east-2",
    "registryId": "111122223333"
   }],
   "repositoryFilters": [{
    "filter": "test",
    "filterType": "PREFIX_MATCH"
   }]
  }
 ]
```

```
}
```

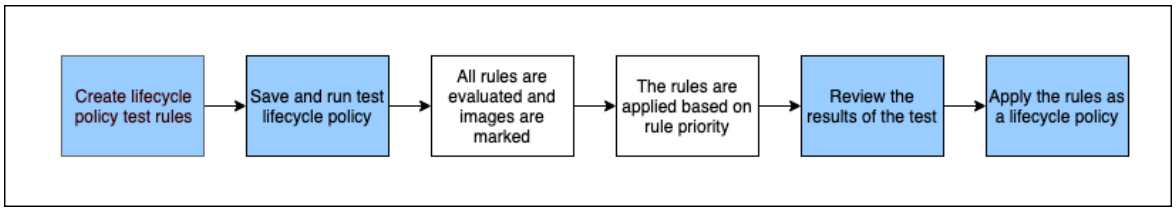
Lifecycle policies

Amazon ECR lifecycle policies provide more control over the lifecycle management of images in a private repository. A lifecycle policy contains one or more rules, where each rule defines an action for Amazon ECR. This provides a way to automate the cleaning up of your container images by expiring images based on age or count. You should expect that images become expired within 24 hours after they meet the expiration criteria per your lifecycle policy. When Amazon ECR performs an action based on a lifecycle policy, this is captured as an event in AWS CloudTrail. For more information, see Logging Amazon ECR actions with AWS CloudTrail (p. 126).

How lifecycle policies work

A lifecycle policy consists of one or more rules that determine which images in a repository should be expired. When considering the use of lifecycle policies, it's important to use the lifecycle policy preview to confirm which images the lifecycle policy expires before applying it to a repository. Once a lifecycle policy is applied to a repository, you should expect that images become expired within 24 hours after they meet the expiration criteria. When Amazon ECR performs an action based on a lifecycle policy, this is captured as an event in AWS CloudTrail. For more information, see Logging Amazon ECR actions with AWS CloudTrail (p. 126).

The following diagram shows the lifecycle policy workflow.



1. Create one or more test rules.
2. Save the test rules and run the preview.
3. The lifecycle policy evaluator goes through all of the rules and marks the images that each rule affects.
4. The lifecycle policy evaluator then applies the rules, based on rule priority, and displays which images in the repository are set to be expired.
5. Review the results of the test, ensuring that the images that are marked to be expired are what you intended.
6. Apply the test rules as the lifecycle policy for the repository.
7. Once the lifecycle policy is created, you should expect that images become expired within 24 hours after they meet the expiration criteria.

Lifecycle policy evaluation rules

The lifecycle policy evaluator is responsible for parsing the plaintext JSON of the lifecycle policy, evaluating all rules, and then applying those rules based on rule priority to the images in the repository. The following explains the logic of the lifecycle policy evaluator in more detail. For examples, see Examples of lifecycle policies (p. 60).

- All rules are evaluated at the same time, regardless of rule priority. After all rules are evaluated, they are then applied based on rule priority.

- 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.
- If an image is referenced by a manifest list, it cannot be expired without the manifest list being deleted first.
- 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.

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. 60).

```
{
    "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`.

Lifecycle policy parameters

Lifecycle policies are split into the following parts:

Topics

- Rule priority (p. 57)

Rule priority

`rulePriority`

Type: integer

Required: yes

Sets the order in which rules are applied, lowest to highest. A lifecycle policy rule with a priority of 1 will be applied 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 evaluated against 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`.

Creating a lifecycle policy preview

A lifecycle policy preview provides a way see the impact of a lifecycle policy on an image repository before you apply it. It is considered best practice to do a preview before applying a lifecycle policy to a repository. The following procedure shows you how to create a lifecycle policy preview.

To create a lifecycle policy preview (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 repository on which to perform a lifecycle policy preview.
3. In the navigation pane, choose **Repositories**.
4. On the **Repositories** page, on the **Private** tab, select a repository to view the repository image list.
5. On the repository image list view, in the left navigation pane, choose **Lifecycle Policy**.

   Note
   If you don't see the **Lifecycle Policy** option in the navigation pane, ensure that you are in the repository image list view.
6. On the repository lifecycle policy page, choose **Edit test rules**, **Create rule**.
7. Enter the following details for each test 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).
   f. Choose **Save**.
8. Create additional test 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. You should expect that after applying a lifecycle policy, the affected images are expired within 24 hours.
12. If you aren't satisfied with the preview results, you may delete one or more test lifecycle rules and create one or more rules to replace them and then repeat the test. If you don't apply the test lifecycle rules as a lifecycle policy, the test rules will persist in the console.

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.

Important
It is considered best practice to create a lifecycle policy preview to ensure that the images affected by your lifecycle policy rules are what you intend. For more information, see Creating a lifecycle policy preview (p. 59).

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**.
4. On the **Repositories** page, on the **Private** tab, select a repository to view the repository image list.
5. On the repository image list view, in the left navigation pane, choose **Lifecycle Policy**.

   Note
   If you don't see the **Lifecycle Policy** option in the navigation pane, ensure that you are in the repository image list view.
6. On the repository lifecycle policy page, choose **Create rule**.
7. 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).
   f. Choose **Save**.
8. Create additional lifecycle policy rules by repeating steps 5–7.

To create a lifecycle policy (AWS CLI)

To create a lifecycle policy using the AWS CLI

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

```
aws ecr describe-repositories
```

2. Create a local file named `policy.json` with the contents of the lifecycle policy. For lifecycle policy examples, see Examples of lifecycle policies (p. 60).
3. Create a lifecycle policy by specifying the repository name and reference the lifecycle policy JSON file you created.

```
aws ecr put-lifecycle-policy \
      --repository-name repository-name \
      --lifecycle-policy-text file://policy.json
```

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 turn on tag immutability to prevent image tags from being overwritten. After the repository is configured for immutable tags, an `ImageTagAlreadyExistsException` error is returned if you attempt to push an image with a tag that

is already in the repository. When tag immutability is turned on for a repository, this affects all tags and you cannot make some tags immutable while others aren't.

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 private repository (p. 21) and Editing a private repository (p. 23).

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. The following scanning types are offered.

- **Enhanced scanning**—Amazon ECR integrates with Amazon Inspector to provide automated, continuous scanning of your repositories. Your container images are scanned for both operating systems and programing language package vulnerabilities. As new vulnerabilities appear, the scan results are updated and Amazon Inspector emits an event to EventBridge to notify you.
- **Basic scanning**—Amazon ECR uses the Common Vulnerabilities and Exposures (CVEs) database from the open-source Clair project. With basic scanning, you configure your repositories to scan on push or you can perform manual scans and Amazon ECR provides a list of scan findings.

Using filters

When an image scanning is configured for your private registry, you may specify that all repositories be scanned or you can specify filters to scope which repositories are scanned.

When **basic** scanning is used, you may specify scan on push filters to specify which repositories are set to do an image scan when new images are pushed. Any repositories not matching a basic scanning scan on push filter will be set to the **manual** scan frequency which means to perform a scan, you must manually trigger the scan.

When **enhanced** scanning is used, you may specify separate filters for scan on push and continuous scanning. Any repositories not matching an enhanced scanning filter will have scanning disabled. If you are using enhanced scanning and specify separate filters for scan on push and continuous scanning where multiple filters match the same repository, then Amazon ECR enforces the continuous scanning filter over the scan on push filter for that repository.

When a filter is specified, a filter with no wildcard will match all repository names that contain the filter. A filter with a wildcard (*) matches on any repository name where the wildcard replaces zero or more characters in the repository name. The following table provides examples where repository names are expressed on the horizontal axis and example filters are specified on the vertical axis.

	prod	**repo-prod**	**prod-repo**	**repo-prod-repo**	**prodrepo**
prod	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
*prod	✓ Yes	✓ Yes	✗ No	✗ No	✗ No
prod*	✓ Yes	✗ No	✓ Yes	✗ No	✓ Yes
prod	✓ Yes	✓ Yes	✓ Yes	✓ Yes	✓ Yes
prod*repo	✗ No	✗ No	✓ Yes	✗ No	✓ Yes

Topics

Enhanced scanning

Amazon ECR enhanced scanning is an integration with Amazon Inspector which provides vulnerability scanning for your container images. Your container images are scanned for both operating systems and programing language package vulnerabilities. You can view the scan findings with both Amazon ECR and with Amazon Inspector directly. For more information about Amazon Inspector, see Scanning container images with Amazon Inspector in the *Amazon Inspector User Guide*.

With enhanced scanning, you can choose which repositories are configured for automatic, continuous scanning and which are configured for scan on push. This is done by setting scan filters.

Considerations for enhanced scanning

The following should be considered when enabling Amazon ECR enhanced scanning.

- There is no additional cost from Amazon ECR to use this feature, however there is a cost from Amazon Inspector to scan your images. For more information, see Amazon Inspector pricing.
- Enhanced scanning isn't supported in the following Regions:
  - Middle East (UAE) (`me-central-1`)

  - Asia Pacific (Hyderabad) (`ap-south-2`)
  - Israel (Tel Aviv) (`il-central-1`)
  - Asia Pacific (Melbourne) (`ap-southeast-4`)
  - Europe (Spain) (`eu-south-2`)
- Amazon Inspector supports scanning for specific operating systems. For a full list, see Supported operating systems - Amazon ECR scanning in the *Amazon Inspector User Guide*.
- Amazon Inspector uses a service-linked IAM role, which provides the permissions needed to provide enhanced scanning for your repositories. The service-linked IAM role is created automatically by Amazon Inspector when enhanced scanning is turned on for your private registry. For more information, see Using service-linked roles for Amazon Inspector in the *Amazon Inspector User Guide*.
- When you initially turn on enhanced scanning for your private registry, Amazon Inspector only recognizes images pushed to Amazon ECR in the last 30 days, based on the image push timestamp. Older images will have the `SCAN_ELIGIBILITY_EXPIRED` scan status. If you'd like these images to be scanned by Amazon Inspector you should push them again to your repository.
- All images pushed to Amazon ECR after enhanced scanning is turned on are continually scanned for the configured duration. By default, the duration is **Lifetime**. This setting can be configured using the Amazon Inspector console. For more information, see Changing the enhanced scanning duration (p. 71).
- When enhanced scanning is turned on for your Amazon ECR private registry, repositories matching the scan filters are scanned using enhanced scanning only. Any repositories that don't match a filter will have an `Off` scan frequency and won't be scanned. Manual scans using enhanced scanning aren't supported. For more information, see Using filters (p. 67).
- If you specify separate filters for scan on push and continuous scanning where multiple filters match the same repository, then Amazon ECR enforces the continuous scanning filter over the scan on push filter for that repository.
- When enhanced scanning is turned on, Amazon ECR sends an event to EventBridge when the scan frequency for a repository is changed. Amazon Inspector emits events to EventBridge when an initial scan is completed and when an image scan finding is created, updated, or closed.

Required IAM permissions

Amazon ECR enhanced scanning requires an Amazon Inspector service-linked IAM role and that the IAM principal enabling and using enhanced scanning has permissions to call the Amazon Inspector APIs needed for scanning. The Amazon Inspector service-linked IAM role is created automatically by Amazon Inspector when enhanced scanning is turned on for your private registry. For more information, see Using service-linked roles for Amazon Inspector in the *Amazon Inspector User Guide*.

The following IAM policy grants the required permissions for enabling and using enhanced scanning. It includes the permission needed for Amazon Inspector to create the service-linked IAM role as well as the Amazon Inspector API permissions needed to turned on and off enhanced scanning and retrieve the scan findings.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "inspector2:Enable",
                "inspector2:Disable",
                "inspector2:ListFindings",
                "inspector2:ListAccountPermissions",
                "inspector2:ListCoverage"
            ],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "inspector2.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Using enhanced scanning

To turn enhanced scanning (AWS Management Console)

To turn on enhanced scanning for your private registry (AWS Management Console)

The scanning configuration is defined at the private registry level on a per-Region basis.

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region to set the scanning configuration for.
3. In the navigation pane, choose **Private registry**, **Scanning**.
4. On the **Scanning configuration** page, for **Scan type** choose **Enhanced scanning**.
5. (Optional) By default, when **Enhanced scanning** is selected, all of your repositories are set for **Continuous scanning**. You can change the default scanning configuration by deselecting the **Continuously scan all repositories** box. You can then configure all repositories for scan on push or you can specify separate scan filters for continuous and scan on push. When scan filters are set, you can choose **Preview repository matches** to verify which repositories in your registry match the defined filters.

   Important
   Filters with no wildcard will match all repository names that contain the filter. Filters with wildcards (*) match on a repository name where the wildcard replaces zero or more characters in the repository name.
6. Choose **Save**.
7. Repeat these steps in each Region in which you want to turn on enhanced scanning.

To turn on enhanced scanning (AWS CLI)

Use the following AWS CLI command to turn on enhanced scanning for your private registry using the AWS CLI. You can specify scan filters using the `rules` object.

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

  The following example turns on enhanced scanning for your private registry. By default, when no `rules` are specified, Amazon ECR sets the scanning configuration to continuous scanning for all repositories.

```
aws ecr put-registry-scanning-configuration \
     --scan-type ENHANCED \
     --region us-east-2
```

The following example turns on enhanced scanning for your private registry and specifies a scan filter. The scan filter in the example turns on continuous scanning for all repositories with `prod` in its name.

```
aws ecr put-registry-scanning-configuration \
     --scan-type ENHANCED \
     --rules '[{"repositoryFilters" : [{"filter":"prod","filterType" :
 "WILDCARD"}],"scanFrequency" : "CONTINUOUS_SCAN"}]' \
     --region us-east-2
```

The following example turns on enhanced scanning for your private registry and specifies multiple scan filters. The scan filters in the example turns on continuous scanning for all repositories with `prod` in its name and turns on scan on push only for all other repositories.

```
aws ecr put-registry-scanning-configuration \
     --scan-type ENHANCED \
     --rules '[{"repositoryFilters" : [{"filter":"prod","filterType" :
 "WILDCARD"}],"scanFrequency" : "CONTINUOUS_SCAN"},{"repositoryFilters" :
 [{"filter":"*","filterType" : "WILDCARD"}],"scanFrequency" : "SCAN_ON_PUSH"}]' \
     --region us-west-2
```

Changing the enhanced scanning duration

Amazon Inspector supports configuring the duration that your private repositories are continuously monitored for. By default, when enhanced scanning is turned on for your Amazon ECR private registry, the Amazon Inspector service continually monitors your repositories until either the image is deleted or enhanced scanning is disabled. The duration that Amazon Inspector scans your images can be changed using the Amazon Inspector settings. The available scan durations are **Lifetime (default)**, **180 days**, and **30 days**. When the scan duration for a repository elapses, the scan status of `SCAN_ELIGIBILITY_EXPIRED` is displayed when listing your scan vulnerabilities. For more information, see Changing the Amazon ECR automated re-scan duration in the *Amazon Inspector User Guide*.

To change the enhanced scanning duration setting

1. Open the Amazon Inspector console at https://console.aws.amazon.com/inspector/v2/home.
2. In the left navigation, expand **Settings** and then choose **General**.
3. On the **Settings** page, under **ECR re-scan duration** choose a setting, then choose **Save**.

EventBridge events

When enhanced scanning is turned on, Amazon ECR sends an event to EventBridge when the scan frequency for a repository is changed. Amazon Inspector emits events to EventBridge when an initial scan is completed and when an image scan finding is created, updated, or closed.

Event for a repository scan frequency change

When enhanced scanning is turned on for your registry, the following event is sent by Amazon ECR when there is a change with a resource that has enhanced scanning turned on. This includes new repositories being created, the scan frequency for a repository being changed, or when images are created or deleted in repositories with enhanced scanning turned on. For more information, see Image scanning (p. 67).

```
{
 "version": "0",
 "id": "0c18352a-a4d4-6853-ef53-0abEXAMPLE",
 "detail-type": "ECR Scan Resource Change",
```

```
 "source": "aws.ecr",
 "account": "123456789012",
 "time": "2021-10-14T20:53:46Z",
 "region": "us-east-1",
 "resources": [],
 "detail": {
  "action-type": "SCAN_FREQUENCY_CHANGE",
  "repositories": [{
    "repository-name": "repository-1",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-1",
    "scan-frequency": "SCAN_ON_PUSH",
    "previous-scan-frequency": "MANUAL"
   },
   {
    "repository-name": "repository-2",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-2",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   },
   {
    "repository-name": "repository-3",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-3",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   }
  ],
  "resource-type": "REPOSITORY",
  "scan-type": "ENHANCED"
 }
}
```

Event for an initial image scan (enhanced scanning)

When enhanced scanning is turned on for your registry, the following event is sent by Amazon Inspector when the initial 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 Enhanced scanning (p. 68).

Event pattern:

```
{
  "source": ["aws.inspector2"],
  "detail-type": ["Inspector2 Scan"]
}
```

Example output:

```
{
    "version": "0",
    "id": "739c0d3c-4f02-85c7-5a88-94a9EXAMPLE",
    "detail-type": "Inspector2 Scan",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2021-12-03T18:03:16Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-sample"
    ],
    "detail": {
        "scan-status": "INITIAL_SCAN_COMPLETE",
        "repository-name": "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-
ecs-sample",
```

```
        "finding-severity-counts": {
            "CRITICAL": 7,
            "HIGH": 61,
            "MEDIUM": 62,
            "TOTAL": 158
        },
        "image-digest":
 "sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77e5EXAMPLE",
        "image-tags": [
            "latest"
        ]
    }
}
```

Event for an image scan finding update (enhanced scanning)

When enhanced scanning is turned on for your registry, the following event is sent by Amazon Inspector when the image scan finding is created, updated, or closed. For more information, see Enhanced scanning (p. 68).

Event pattern:

```
{
  "source": ["aws.inspector2"],
  "detail-type": ["Inspector2 Finding"]
}
```

Example output:

```
{
    "version": "0",
    "id": "42dbea55-45ad-b2b4-87a8-afaEXAMPLE",
    "detail-type": "Inspector2 Finding",
    "source": "aws.inspector2",
    "account": "123456789012",
    "time": "2021-12-03T18:02:30Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-sample/
sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77eEXAMPLE"
    ],
    "detail": {
        "awsAccountId": "123456789012",
        "description": "In libssh2 v1.9.0 and earlier versions, the SSH_MSG_DISCONNECT
 logic in packet.c has an integer overflow in a bounds check, enabling an attacker to
 specify an arbitrary (out-of-bounds) offset for a subsequent memory read. A crafted
 SSH server may be able to disclose sensitive information or cause a denial of service
 condition on the client system when a user connects to the server.",
        "findingArn": "arn:aws:inspector2:us-east-2:123456789012:finding/
be674aaddd0f75ac632055EXAMPLE",
        "firstObservedAt": "Dec 3, 2021, 6:02:30 PM",
        "inspectorScore": 6.5,
        "inspectorScoreDetails": {
            "adjustedCvss": {
                "adjustments": [],
                "cvssSource": "REDHAT_CVE",
                "score": 6.5,
                "scoreSource": "REDHAT_CVE",
                "scoringVector": "CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:N",
                "version": "3.0"
            }
        },
        "lastObservedAt": "Dec 3, 2021, 6:02:30 PM",
```

```
        "packageVulnerabilityDetails": {
            "cvss": [
                {
                    "baseScore": 6.5,
                    "scoringVector": "CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:N",
                    "source": "REDHAT_CVE",
                    "version": "3.0"
                },
                {
                    "baseScore": 5.8,
                    "scoringVector": "AV:N/AC:M/Au:N/C:P/I:N/A:P",
                    "source": "NVD",
                    "version": "2.0"
                },
                {
                    "baseScore": 8.1,
                    "scoringVector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:H",
                    "source": "NVD",
                    "version": "3.1"
                }
            ],
            "referenceUrls": [
                "https://access.redhat.com/errata/RHSA-2020:3915"
            ],
            "source": "REDHAT_CVE",
            "sourceUrl": "https://access.redhat.com/security/cve/CVE-2019-17498",
            "vendorCreatedAt": "Oct 16, 2019, 12:00:00 AM",
            "vendorSeverity": "Moderate",
            "vulnerabilityId": "CVE-2019-17498",
            "vulnerablePackages": [
                {
                    "arch": "X86_64",
                    "epoch": 0,
                    "name": "libssh2",
                    "packageManager": "OS",
                    "release": "12.amzn2.2",
                    "sourceLayerHash":
 "sha256:72d97abdfae3b3c933ff41e39779cc72853d7bd9dc1e4800c5294dEXAMPLE",
                    "version": "1.4.3"
                }
            ]
        },
        "remediation": {
            "recommendation": {
                "text": "Update all packages in the vulnerable packages section to their
 latest versions."
            }
        },
        "resources": [
            {
                "details": {
                    "awsEcrContainerImage": {
                        "architecture": "amd64",
                        "imageHash":
 "sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77e5EXAMPLE",
                        "imageTags": [
                            "latest"
                        ],
                        "platform": "AMAZON_LINUX_2",
                        "pushedAt": "Dec 3, 2021, 6:02:13 PM",
                        "registry": "123456789012",
                        "repositoryName": "amazon/amazon-ecs-sample"
                    }
                },
                "id": "arn:aws:ecr:us-east-2:123456789012:repository/amazon/amazon-ecs-
 sample/sha256:36c7b282abd0186e01419f2e58743e1bf635808231049bbc9d77EXAMPLE",
```

```
                "partition": "N/A",
                "region": "N/A",
                "type": "AWS_ECR_CONTAINER_IMAGE"
            }
        ],
        "severity": "MEDIUM",
        "status": "ACTIVE",
        "title": "CVE-2019-17498 - libssh2",
        "type": "PACKAGE_VULNERABILITY",
        "updatedAt": "Dec 3, 2021, 6:02:30 PM"
    }
}
```


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. 145).

To retrieve image scan findings (AWS Management 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 where your repository exists.
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 **See findings** for the image to retrieve the scan findings for.
6. When viewing the **Findings**, the vulnerability name in the **Name** column is a link to the Amazon Inspector console where you can view more details.

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
```

Basic scanning

Amazon ECR provides basic scanning type which uses the Common Vulnerabilities and Exposures (CVEs) database from the open-source Clair project. With basic scanning enabled on your private registry, you can configure repository filters to specify which repositories are set to scan on push or you can perform manual scans. Amazon ECR provides a list of scan findings. Each container image may be scanned once per 24 hours. Amazon ECR uses the Common Vulnerabilities and Exposures (CVEs) database from the open-source Clair project and provides 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.

When basic scanning is used, you may specify scan on push filters to specify which repositories are set to do an image scan when new images are pushed. Any repositories not matching a scan on push filter will be set to the **manual** scan frequency which means to perform a scan, you must manually trigger the scan. 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. 124).

For troubleshooting details for some common issues when scanning images, see Troubleshooting image scanning issues (p. 145).

Using basic scanning

By default, Amazon ECR enables basic scanning on all private registries. As a result, unless you've changed the scanning settings on your private registry there should be no need to enable basic scanning. You may use the following steps to verify that basic scanning is enabled and define one or more scan on push filters.

To turn on basic scanning for your private registry (AWS Management Console)

The scanning configuration is defined at the private registry level on a per-Region basis.

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region to set the scanning configuration for.
3. In the navigation pane, choose **Private registry**, **Scanning**.
4. On the **Scanning configuration** page, For **Scan type** choose **Basic scanning**.
5. By default all of your repositories are set for **Manual** scanning. You can optionally configure scan on push by specifying **Scan on push filters**. You can set scan on push for all repositories or individual repositories. For more information, see Using filters (p. 67).

Manually scanning an image

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

For troubleshooting details for some common issues when scanning images, see Troubleshooting image scanning issues (p. 145).

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. 145).

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 for a singular 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 earlier, 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 later, 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 isn't understood by the client, Amazon ECR converts the image manifest into a format that is understood. 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 following table 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 Amazon ECR private repositories to host container images and artifacts that your Amazon ECS tasks may pull from. For this to work, the Amazon ECS, or Fargate, container agent

must have permissions to make the `ecr:BatchGetImage`, `ecr:GetDownloadUrlForLayer`, and `ecr:GetAuthorizationToken` APIs.

Required IAM permissions

The following table shows the IAM role to use, for each launch type, that provides the required permissions for your tasks to pull from an Amazon ECR private repository. Amazon ECS provides managed IAM policies that include the required permissions.

Launch type	IAM role	AWS managed IAM policy
Amazon ECS on Amazon EC2 instances	Use the container instance IAM role, which is associated with the Amazon EC2 instance registered to your Amazon ECS cluster. For more information, see Container instance IAM role in the *Amazon Elastic Container Service Developer Guide*.	`AmazonEC2ContainerServiceforEC2Role` For more information, see AmazonEC2ContainerServiceforEC2Role in the *Amazon Elastic Container Service Developer Guide*
Amazon ECS on Fargate	Use the task execution IAM role that you reference in your Amazon ECS task definition. For more information, see Task execution IAM role in the *Amazon Elastic Container Service Developer Guide*.	`AmazonECSTaskExecutionRolePolicy` For more information, see AmazonECSTaskExecutionRolePolicy in the *Amazon Elastic Container Service Developer Guide*.
Amazon ECS on external instances	Use the container instance IAM role, which is associated with the on-premises server or virtual machine (VM) registered to your Amazon ECS cluster. For more information, see Container instance Amazon ECS role in the *Amazon Elastic Container Service Developer Guide*.	`AmazonEC2ContainerServiceforEC2Role` For more information, see AmazonEC2ContainerServiceforEC2Role in the *Amazon Elastic Container Service Developer Guide*.

Important
The AWS managed IAM policies contain additional permissions that you may not require for your use. In this case, these are the minimum required permissions to pull from an Amazon ECR private repository.

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

Specifying an Amazon ECR image in an Amazon ECS task definition

When creating an Amazon ECS task definition, you can specify a container image hosted in an Amazon ECR private repository. In the task definition, ensure that you use the full `registry/repository:tag` naming for your Amazon ECR images. For example, *aws_account_id*`.dkr.ecr.`*region*`.amazonaws.com/`*my-repository*`:`*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-repository: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.

- For Amazon EKS workloads hosted on managed or self-managed nodes, the Amazon EKS worker node IAM role (`NodeInstanceRole`) is required. The Amazon EKS worker node IAM role 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": "*"
          }
      ]
  }
  ```

  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.
- For Amazon EKS workloads hosted on AWS Fargate, you must use the Fargate pod execution role, which provides your pods permission to pull images from private Amazon ECR repositories. For more information, see Create a Fargate pod execution role.

- 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-repository`*`:`*`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 latest version of the Helm client. These steps were written using Helm version `3.9.0`. For more information, see Installing Helm.
- You have at least version `1.23.9` or `2.6.3` of the AWS CLI installed on your computer. For more information, see Installing or updating the latest version of the AWS CLI.
- You have pushed a Helm chart to your Amazon ECR repository. For more information, see Pushing a Helm chart (p. 36).
- 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. 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 Private 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
   ```

2. Install the chart. Replace *`helm-test-chart`* with your repository and *`0.1.0`* with your Helm chart's tag.

   ```
   helm install ecr-chart-demo oci://aws_account_id.dkr.ecr.region.amazonaws.com/helm-test-chart --version 0.1.0
   ```

   The output should look similar to this:

   ```
   NAME: ecr-chart-demo
   LAST DEPLOYED: Tue May 31 17:38:56 2022
   NAMESPACE: default
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   ```

3. Verify the chart installation.

```
helm list -n default
```

Example output:

```
NAME            NAMESPACE       REVISION        UPDATED
 STATUS          CHART                   APP VERSION
ecr-chart-demo  default         1               2022-06-01 15:56:40.128669157 +0000 UTC
 deployed        helm-test-chart-0.1.0   1.16.0
```

4. (Optional) See the installed Helm chart `ConfigMap`.

```
kubectl describe configmap helm-test-chart-configmap
```

5. 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's available for use in any environment as a base image for Docker workloads. If you're using the Amazon Linux AMI for applications in Amazon EC2, you can 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 AWS using Amazon ECS. For more information, see Using Amazon ECR images with Amazon ECS (p. 79).

The Amazon Linux container image is available on Amazon ECR Public 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 Public

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

   Note
   The **ecr-public** commands are available in the AWS CLI starting with version `1.18.1.187`, however we recommend using 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*.

```
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --
password-stdin public.ecr.aws
```

   The output is as follows:

```
Login succeeded
```

2. Pull the Amazon Linux container image using the **docker pull** command. To view the Amazon Linux container image on the Amazon ECR Public Gallery, see Amazon ECR Public Gallery - amazonlinux.

```
docker pull public.ecr.aws/amazonlinux/amazonlinux:latest
```

3. (Optional) Run the container locally.

```
docker run -it public.ecr.aws/amazonlinux/amazonlinux /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

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that 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. 106).

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 service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon ECR, see How Amazon Elastic Container Registry works with IAM (p. 89).

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. 102).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

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

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

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

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see 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:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects

in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Elastic Container Registry in the *Service Authorization Reference*.
- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see 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 or IAM users, 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 AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see 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. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

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

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see 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

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

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

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

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

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

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

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

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

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

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

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. 105).

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. 102).

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 Private repository policies (p. 24).

> **Note**
> In an Amazon ECR repository policy, the policy element `Sid` supports additional characters and spacing not supported in IAM policies.

Examples

To view examples of Amazon ECR resource-based policies, see Private repository policy examples (p. 26),

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 a private repository (p. 29).

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. 105).

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 supports service-linked roles. For more information, see Using service-linked roles for Amazon ECR (p. 97).

AWS managed policies for Amazon Elastic Container Registry

Amazon ECR provides several managed policies that you can attach to IAM users or Amazon EC2 instances. These policies allow differing levels of control over access to Amazon ECR resources and API operations. You can apply these policies directly or 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

You can attach the `AmazonEC2ContainerRegistryFullAccess` policy to your IAM identities.

You can use this managed policy as a starting point to create your own IAM policy based on your specific requirements. For example, you can create a policy specifically for providing a user or role with full administrator access to manage the use of Amazon ECR. With the Amazon ECR Lifecycle Policies feature, you can specify the lifecycle management of images in a repository. Lifecycle policy events are reported as CloudTrail events. Amazon ECR is integrated with AWS CloudTrail so it can display your lifecycle policy events directly in the Amazon ECR console. The `AmazonEC2ContainerRegistryFullAccess` managed IAM policy includes the `cloudtrail:LookupEvents` permission to facilitate this behavior.

Permissions details

This policy includes the following permissions:

- `ecr` – Allows principals full access to all Amazon ECR APIs.
- `cloudtrail` – Allows principals to looks up management events or AWS CloudTrail Insights events that are captured by CloudTrail.

The AmazonEC2ContainerRegistryFullAccess policy is as follows.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:*",
                "cloudtrail:LookupEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "replication.ecr.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

AmazonEC2ContainerRegistryPowerUser

You can attach the AmazonEC2ContainerRegistryPowerUser policy to your IAM identities.

This policy grants administrative permissions that allow IAM users to read and write to repositories, but doesn't allow them to delete repositories or change the policy documents that are applied to them.

Permissions details

This policy includes the following permissions:

- ecr – Allows principals to read and write to respositores, as well as read lifecycle policies. Principals aren't granted permission to delete repositories or change the lifecycle policies that are applied to them.

The AmazonEC2ContainerRegistryPowerUser policy is as follows.

```
{
    "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

You can attach the `AmazonEC2ContainerRegistryReadOnly` policy to your IAM identities.

This policy grants read-only permissions to Amazon ECR. This includes the ability to list repositories and images within the repositories. It also includes the ability to pull images from Amazon ECR with the Docker CLI.

Permissions details

This policy includes the following permissions:

- `ecr` – Allows principals to read repositories and their respective lifecycle policies.

The `AmazonEC2ContainerRegistryReadOnly` policy is as follows.

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

AWSECRPullThroughCache_ServiceRolePolicy

You can't attach the `AWSECRPullThroughCache_ServiceRolePolicy` managed IAM policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon ECR to push images to

your repositories through the pull through cache workflow. For more information, see Using service-linked roles for Amazon ECR (p. 97).

ECRReplicationServiceRolePolicy

You can't attach the ECRReplicationServiceRolePolicy managed IAM policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon ECR to perform actions on your behalf. For more information, see Using service-linked roles for Amazon ECR (p. 97).

Amazon ECR updates to AWS managed policies

View details about updates to AWS managed policies for Amazon ECR since the time that this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon ECR Document history page.

Change	Description	Date
AWSECRPullThroughCache_ServiceRolePolicy (p. 95) – New policy	Amazon ECR added a new policy. This policy is associated with the AWSServiceRoleForECRPullThroughCache service-linked role for the pull through cache feature.	November 29, 2021
ECRReplicationServiceRolePolicy (p. 96) – New policy	Amazon ECR added a new policy. This policy is associated with the AWSServiceRoleForECRReplication service-linked role for the replication feature.	December 4, 2020
AmazonEC2ContainerRegistryFullAccess (p. 95) – Update to an existing policy	Amazon ECR added new permissions to the AmazonEC2ContainerRegistryFullAccess policy. These permissions allow principals to create the Amazon ECR service-linked role.	December 4, 2020
AmazonEC2ContainerRegistryReadOnly (p. 96) – Update to an existing policy	Amazon ECR added new permissions to the AmazonEC2ContainerRegistryReadOnly policy which allow principals to read lifecycle policies, list tags, and describe the scan findings for images.	December 10, 2019
AmazonEC2ContainerRegistryPowerUser (p. 96) – Update to an existing policy	Amazon ECR added new permissions to the AmazonEC2ContainerRegistryPowerUser policy. They allow principals to read lifecycle policies, list tags, and describe the scan findings for images.	December 10, 2019
AmazonEC2ContainerRegistryFullAccess (p. 95) – Update to an existing policy	Amazon ECR added new permissions to the AmazonEC2ContainerRegistryFullAccess policy. They allow principals to look up management events or	November 10, 2017

Change	Description	Date
	AWS CloudTrail Insights events that are captured by CloudTrail.	
AmazonEC2ContainerRegistryReadOnly (p. 96) – Update to an existing policy	Amazon ECR added new permissions to the `AmazonEC2ContainerRegistryReadOnly` policy. They allow principals to describe Amazon ECR images.	October 11, 2016
AmazonEC2ContainerRegistryPowerUser (p. 96) – Update to an existing policy	Amazon ECR added new permissions to the `AmazonEC2ContainerRegistryPowerUser` policy. They allow principals to describe Amazon ECR images.	October 11, 2016
AmazonEC2ContainerRegistryReadOnly (p. 96) – New policy	Amazon ECR added a new policy which grants read-only permissions to Amazon ECR. These permissions include the ability to list repositories and images within the repositories. They also include the ability to pull images from Amazon ECR with the Docker CLI.	December 21, 2015
AmazonEC2ContainerRegistryPowerUser (p. 96) – New policy	Amazon ECR added a new policy which grants administrative permissions that allow usrs to read and write to repositories but doesn't allow them to delete repositories or change the policy documents that are applied to them.	December 21, 2015
AmazonEC2ContainerRegistryFullAccess (p. 96) – New policy	Amazon ECR added a new policy. This policy grants full access to Amazon ECR.	December 21, 2015
Amazon ECR started tracking changes	Amazon ECR started tracking changes for AWS managed policies.	June 24, 2021

Using service-linked roles for Amazon ECR

Amazon Elastic Container Registry (Amazon ECR) uses AWS Identity and Access Management (IAM) service-linked roles to provide the permissions necessary to use the replication and pull through cache features. A service-linked role is a unique type of IAM role that is linked directly to Amazon ECR. The service-linked role is predefined by Amazon ECR. It includes all of the permissions that the service requires to support the replication and pull through cache features for your private registry. After you configure replication or pull through cache for your registry, a service-linked role is created automatically on your behalf. For more information, see Private registry settings (p. 15).

A service-linked role makes setting up replication and pull through cache with Amazon ECR easier. This is because, by using it, you don't have to manually add all the necessary permissions. Amazon ECR defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon ECR can assume

its roles. The defined permissions include the trust policy and the permissions policy. The permissions policy can't be attached to any other IAM entity.

You can delete the corresponding service-linked role only after disabling either replication or pull through cache on your registry. This ensures that you don't inadvertently remove the permissions Amazon ECR requires for these features.

For information about other services that support service-linked roles, see AWS services that work with IAM. On this linked-to page, look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the relevant service-linked role documentation for that service.

Topics

Supported Regions for Amazon ECR service-linked roles

Amazon ECR supports using service-linked roles in all of the Regions where the Amazon ECR service is available. For more information about Amazon ECR Region availability, see AWS Regions and Endpoints.

Amazon ECR service-linked role for replication

Service-linked role permissions for Amazon ECR

Amazon ECR uses service-linked roles named **AWSServiceRoleForECRReplication** – Allows Amazon ECR to replicate images across multiple accounts..

The AWSServiceRoleForECRReplication service-linked role trusts the following services to assume the role:

- `replication.ecr.amazonaws.com`

The following `ECRReplicationServiceRolePolicy` role permissions policy allows Amazon ECR to use the following actions on resources:

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

Note
The `ReplicateImage` is an internal API that Amazon ECR uses for replication and can't be called directly.

You must configure permissions to allow an IAM entity (for example a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon ECR

You don't need to manually create the Amazon ECR service-linked role. When you configure replication settings for your registry in the AWS Management Console, the AWS CLI, or the AWS API, Amazon ECR creates the service-linked role for you.

If you delete this service-linked role and need to create it again, you can use the same process to recreate the role in your account. When you configure replication settings for your registry, Amazon ECR creates the service-linked role for you again.

Editing a service-linked role for Amazon ECR

Amazon ECR doesn't allow manually editing the AWSServiceRoleForECRReplication service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting the service-linked role for Amazon ECR

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way, you don't have an unused entity that isn't actively monitored or maintained. However, you must remove the replication configuration for your registry in every Region before you can manually delete the service-linked role.

> **Note**
> If you try to delete resources while the Amazon ECR service is still using the roles, your delete action might fail. If that happens, wait for a few minutes and try again.

To delete Amazon ECR resources used by the AWSServiceRoleForECRReplication

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, choose the Region your replication configuration is set on.
3. In the navigation pane, choose **Private registry**.
4. On the **Private registry** page, on the **Replication configuration** section, choose **Edit**.
5. To delete all of your replication rules, choose **Delete all**. This step requires confirmation.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the **AWSServiceRoleForECRReplication** service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Amazon ECR service-linked role for pull through cache

Amazon ECR uses a service-linked role named **AWSServiceRoleForECRPullThroughCache** which gives permission for Amazon ECR to push images to your repositories through the pull through cache workflow.

Service-linked role permissions for Amazon ECR

The **AWSServiceRoleForECRPullThroughCache** service-linked role trusts the following service to assume the role.

- `pullthroughcache.ecr.amazonaws.com`

The following `AWSECRPullThroughCache_ServiceRolePolicy` permissions policy is attached to the service-linked role and allows Amazon ECR to use the following actions.

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

You must configure permissions to allow an IAM entity (for example a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon ECR

You don't need to manually create the Amazon ECR service-linked role for pull through cache. When you create a pull through cache rule for your private registry in the AWS Management Console, the AWS CLI, or the AWS API, Amazon ECR creates the service-linked role for you.

If you delete this service-linked role and need to create it again, you can use the same process to recreate the role in your account. When you create a pull through cache rule for your private registry, Amazon ECR creates the service-linked role for you again if it doesn't already exist.

Editing a service-linked role for Amazon ECR

Amazon ECR doesn't allow manually editing the **AWSServiceRoleForECRPullThroughCache** service-linked role. After the service-linked role is created, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting the service-linked role for Amazon ECR

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way, you don’t have an unused entity that isn't actively monitored or maintained. However, you must delete the pull through cache rules for your registry in every Region before you can manually delete the service-linked role.

> **Note**
> If you try to delete resources while the Amazon ECR service is still using the role, your delete action might fail. If that happens, wait for a few minutes and try again.

To delete Amazon ECR resources used by the AWSServiceRoleForECRPullThroughCache service-linked role

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/.
2. From the navigation bar, choose the Region where your pull through cache rules are created.
3. In the navigation pane, choose **Private registry**.
4. On the **Private registry** page, on the **Pull through cache configuration** section, choose **Edit**.
5. For each pull through cache rule you have created, select the rule and then choose **Delete rule**.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the **AWSServiceRoleForECRPullThroughCache** service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` or `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that Amazon ECR gives another service to the resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:`*`servicename`*`:`*`region`*`:`*`123456789012`*`:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions.

The value of `aws:SourceArn` must be ResourceDescription.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in an Amazon ECR repository policy to allow AWS CodeBuild access to the Amazon ECR API actions necessary for integration with that service while also preventing the confused deputy problem.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CodeBuildAccess",
         "Effect":"Allow",
         "Principal":{
            "Service":"codebuild.amazonaws.com"
         },
         "Action":[
            "ecr:BatchGetImage",
            "ecr:GetDownloadUrlForLayer"
         ],
         "Condition":{
            "ArnLike":{
               "aws:SourceArn":"arn:aws:codebuild:region:123456789012:project/project-name"
            },
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            }
         }
      }
```

```
    ]
}
```

Amazon Elastic Container Registry Identity-based policy examples

By default, users and roles don't have permission to create or modify Amazon ECR resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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

For details about actions and resource types defined by Amazon ECR, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon Elastic Container Registry in the *Service Authorization Reference*.

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 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM

policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the 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 (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 a 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 a private repository (p. 29).

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 a 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 a 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 you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `ecr:`*`GetWidget`* permissions.

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

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `ecr:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated 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 that are 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's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my 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. 89).
- 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

The AWS shared responsibility model applies to data protection in Amazon Elastic Container Service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

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

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.

- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon ECS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

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 User Guide*.

For more control over the encryption for your Amazon ECR repositories, you can use server-side encryption with KMS keys 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 key, which is managed by Amazon ECR, or specify your own KMS key (referred to as a customer managed key). For more information, see Protecting Data Using Server-Side Encryption with KMS keys Stored in AWS KMS (SSE-KMS) in the *Amazon Simple Storage Service User 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 private repository (p. 21).

When a repository is created with AWS KMS encryption enabled, a KMS key is used to encrypt the contents of the repository. Moreover, Amazon ECR adds an AWS KMS grant to the KMS key 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 KMS key specified in the encryption configuration.
2. Amazon ECR sends two CreateGrant requests to AWS KMS to create grants on the KMS key 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 KMS key to use for encrypting the image layer and manifest.
4. AWS KMS generates a new data key, encrypts it under the specified KMS key, 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 is 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 KMS key, Amazon ECR uses an AWS managed key with the alias `aws/ecr` by default. This KMS key 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 KMS key, the key must exist in the same Region as your repository.
- 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 KMS key you are using.

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

By default, when AWS KMS encryption is enabled for an Amazon ECR repository but no KMS key is specified, the AWS managed key for Amazon ECR is used. When the AWS-managed KMS key 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 KMS key used to encrypt the repository can be specified using the resource parameter.

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

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

When creating your own KMS key, you can either use the default key policy AWS KMS creates or you can specify your own. To ensure that the customer managed key remains manageable by the account owner, the key policy for the KMS key 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 KMS key. To allow the KMS key 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 KMS key full access to the KMS key. 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": "EnableIAMUserPermissions",
            "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 KMS key 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":
 "AllowAccessToCreateAndRetireTheGrantsAssociatedWithTheKeyAsWellAsDescribeTheKey",
            "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 KMS keys 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 KMS keys in the Region when enabling encryption for the repository. The following IAM policy example shows the permissions needed to list your KMS keys 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 KMS key 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 keys and customer managed KMS keys 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

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

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

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

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

  Note
  Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.
- 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. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Infrastructure Security in Amazon Elastic Container Registry

As a managed service, Amazon Elastic Container Registry is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon ECR through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

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

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. 114).

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 hosted on Amazon EC2 instances 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 hosted on Fargate don't require the Amazon ECS interface VPC endpoints.
- Amazon ECS tasks hosted on Fargate using Linux 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 hosted on Fargate using Linux 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 hosted on Fargate using Windows platform version `1.0.0` 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 hosted on Fargate 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 hosted on Fargate 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. 118).
- 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 currently don't support Amazon ECR Public repositories. Consider using a pull through cache rule to host the public image in a private repository in the same Region as the VPC endpoint. For more information, see Using pull through cache rules (p. 42).
- VPC endpoints only support AWS 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.
- When an image is pulled using a pull through cache rule for the first time, if you've configured Amazon ECR to use an interface VPC endpoint using AWS PrivateLink then you need to create a public subnet in the same VPC, with a NAT gateway, and then route all outbound traffic to the internet from their private subnet to the NAT gateway in order for the pull to work. Subsequent image pulls don't require this. For more information, see Scenario: Access the internet from a private subnet in the *Amazon Virtual Private Cloud User Guide*.

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 behavior 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 hosted on Amazon EC2 instances require both Amazon ECR endpoints and the Amazon S3 gateway endpoint.

Amazon ECS tasks hosted on Fargate using platform version `1.4.0` or later require both Amazon ECR VPC endpoints and the Amazon S3 gateway endpoints.

Amazon ECS tasks hosted on Fargate that use 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.

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 this endpoint, you must enable a private DNS hostname. To do this, ensure that the **Enable Private DNS Name** option is selected in the Amazon 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 `CreateRepository` go to this endpoint.

When this 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. 117).

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 Using CloudWatch Logs with interface VPC endpoints 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 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",
                   "ecr:GetAuthorizationToken"
  ],
  "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",
                          "ecr:GetAuthorizationToken"
   ],
   "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. 116).
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.

Shared subnets

You can't create, describe, modify, or delete VPC endpoints in subnets that are shared with you. However, you can use the VPC endpoints in subnets that are shared with you.

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. 135).

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.
`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. 31).

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

Amazon ECR repository metrics

Amazon ECR sends repository pull count metrics to Amazon CloudWatch. Amazon ECR metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the Amazon CloudWatch User Guide.

Topics

Enabling CloudWatch metrics

Amazon ECR sends repository metrics automatically for all repositories. There is no need to take any manual steps.

Available metrics and dimensions

The following sections list the metrics and dimensions that Amazon ECR sends to Amazon CloudWatch.

Amazon ECR metrics

Amazon ECR provides metrics for you to monitor your repositories. You can measure the pull count.

The `AWS/ECR` namespace includes the following metrics.

`RepositoryPullCount`

The total number of pulls for the images in the repository.

Valid dimensions: `RepositoryName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Sum.

Unit: Integer.

Dimensions for Amazon ECR metrics

Amazon ECR metrics use the `AWS/ECR` namespace and provide metrics for the following dimensions.

`RepositoryName`

This dimension filters the data that you request for all container images in a specified repository.

Viewing Amazon ECR metrics

You can view Amazon ECR repository metrics on the CloudWatch console. The CloudWatch console provides a fine-grained and customizable display of your resources.

Viewing Amazon ECR metrics using the CloudWatch console

Amazon ECR repository metrics can be viewed on the CloudWatch console. The console provides the most detailed view of Amazon ECR metrics, and you can tailor the views to suit your needs. For more information, see the Amazon CloudWatch User Guide.

To view metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation, choose **Metrics**, **All metrics**.
3. On the **Browse** tab, under **AWS Namespaces**, choose **ECR**.
4. Choose the metrics to view. Repository metrics are scoped as **ECR > Repository Metrics**.

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 Amazon SQS 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. Events are emitted on a best effort basis.

Event for a completed image push

The following event is sent when each image push is completed. For more information, see Pushing a Docker image (p. 34).

```
{
    "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-repository-name",
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "action-type": "PUSH",
        "image-tag": "latest"
    }
}
```

Event for a completed image scan (basic scanning)

When basic scanning is enabled for your registry, 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 Basic scanning (p. 76).

> **Note**
> For details about events that Amazon Inspector emits when enhanced scanning is enabled, see EventBridge events (p. 71).

```
{
    "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-repository-name"
    ],
    "detail": {
        "scan-status": "COMPLETE",
        "repository-name": "my-repository-name",
        "finding-severity-counts": {
        "CRITICAL": 10,
        "MEDIUM": 9
      },
        "image-digest":
 "sha256:7f5b2640fe6fb4f46592dfd3410c4a79dac4f89e4782432e0378abcd1234",
        "image-tags": []
    }
}
```

Event for a change notification on a resource with enhanced scanning enabled (enhanced scanning)

When enhanced scanning is enabled for your registry, the following event is sent by Amazon ECR when there is a change with a resource that has enhanced scanning enabled. This includes new repositories being created, the scan frequency for a repository being changed, or when images are created or deleted in repositories with enhanced scanning enabled. For more information, see Image scanning (p. 67).

```
{
 "version": "0",
 "id": "0c18352a-a4d4-6853-ef53-0ab8638973bf",
 "detail-type": "ECR Scan Resource Change",
 "source": "aws.ecr",
 "account": "123456789012",
 "time": "2021-10-14T20:53:46Z",
 "region": "us-east-1",
 "resources": [],
 "detail": {
  "action-type": "SCAN_FREQUENCY_CHANGE",
  "repositories": [{
    "repository-name": "repository-1",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-1",
    "scan-frequency": "SCAN_ON_PUSH",
    "previous-scan-frequency": "MANUAL"
   },
   {
    "repository-name": "repository-2",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-2",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   },
   {
    "repository-name": "repository-3",
    "repository-arn": "arn:aws:ecr:us-east-1:123456789012:repository/repository-3",
    "scan-frequency": "CONTINUOUS_SCAN",
    "previous-scan-frequency": "SCAN_ON_PUSH"
   }
  ],
  "resource-type": "REPOSITORY",
  "scan-type": "ENHANCED"
```

```
 }
}
```

Event for an image deletion

The following event is sent when an image is deleted. For more information, see Deleting an image (p. 47).

```
{
    "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-repository-name",
        "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

> **Important**
> Due to the size limitations of individual CloudTrail events, for lifecycle policy actions where 10 or more images are expired Amazon ECR sends multiple events to CloudTrail. Additionally, Amazon ECR includes a maximum of 100 tags per image.

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. 127).

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 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: Image pull action (p. 132)
- Example: Image lifecycle policy action (p. 133)

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.

Important
Due to the size limitations of individual CloudTrail events, for lifecycle policy actions where 10 or more images are expired Amazon ECR sends multiple events to CloudTrail. Additionally, Amazon ECR includes a maximum of 100 tags per image.

```
{
    "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).

Name	Default	Adjust	Description
Filters per rule in a replication configuration	Each supported Region: 100	No	The maximum number of filters per rule in a replication configuration.
Images per repository	Each supported Region: 10,000	Yes	The maximum number of images per repository.
Layer parts	Each supported Region: 4,200	No	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.
Lifecycle policy length	Each supported Region: 30,720	No	The maximum number of characters in a lifecycle policy.
Maximum layer part size	Each supported Region: 10	No	The maximum 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.
Maximum layer size	Each supported Region: 52,000	No	The maximum size (MiB) of a layer.
Minimum layer part size	Each supported Region: 5	No	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.
Pull through cache rules per registry	Each supported Region: 10	No	The maximum number of pull-through cache rules.
Rate of BatchCheckLayerAvailability requests	Each supported Region: 1,000 per second	Yes	The maximum number of BatchCheckLayerAvailability requests that you can make per second in the current Region. When an image is pushed to a repository, each image layer is checked to verify if it has been

Name	Default	Adjust	Description
			uploaded before. If it has been uploaded, then the image layer is skipped.
Rate of BatchGetImage requests	Each supported Region: 2,000 per second	Yes	The maximum number of BatchGetImage requests that you can make per second in the current Region. When an image is pulled, the BatchGetImage API is called once to retrieve the image manifest. If you request a quota increase for this API, review your GetDownloadUrlForLayer usage as well.
Rate of CompleteLayerUpload requests	Each supported Region: 100 per second	Yes	The maximum number of CompleteLayerUpload requests that you can make per second in the current Region. When an image is pushed, the CompleteLayerUpload API is called once per each new image layer to verify that the upload has completed.
Rate of GetAuthorizationToken requests	Each supported Region: 500 per second	Yes	The maximum number of GetAuthorizationToken requests that you can make per second in the current Region.
Rate of GetDownloadUrlForLayer requests	Each supported Region: 3,000 per second	Yes	The maximum number of GetDownloadUrlForLayer requests that you can make per second in the current Region. When an image is pulled, the GetDownloadUrlForLayer API is called once per image layer that is not already cached. If you request a quota increase for this API, review your BatchGetImage usage as well.

Name	Default	Adjust	Description
Rate of InitiateLayerUpload requests	Each supported Region: 100 per second	Yes	The maximum number of InitiateLayerUpload requests that you can make per second in the current Region. When an image is pushed, the InitiateLayerUpload API is called once per image layer that has not already been uploaded. Whether or not an image layer has been uploaded is determined by the BatchCheckLayerAvailability API action.
Rate of PutImage requests	Each supported Region: 10 per second	Yes	The maximum number of PutImage requests that you can make per second in the current 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.
Rate of UploadLayerPart requests	Each supported Region: 500 per second	Yes	The maximum number of UploadLayerPart requests that you can make per second in the current Region. When an image is pushed, each new image layer is uploaded in parts and the UploadLayerPart API is called once per each new image layer part.
Rate of image scans	Each supported Region: 1	No	The maximum number of image scans per image, per 24 hours.
Registered repositories	Each supported Region: 10,000	Yes	The maximum number of repositories that you can create in this account in the current Region.
Rules per lifecycle policy	Each supported Region: 50	No	The maximum number of rules in a lifecycle policy
Rules per replication configuration	Each supported Region: 10	No	The maximum number of rules in a replication configuration.

Name	Default	Adjust	Description
Tags per image	Each supported Region: 1,000	No	The maximum number of tags per image.
Unique destinations across all rules in a replication configuration	Each supported Region: 25	No	The maximum number of unique destinations across all rules in a replication configuration.

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. 121).

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. 126).

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

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.

Topics

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. For more information, see the section called “Registry authentication” (p. 13).

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 Private repository policies (p. 24).

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-password** (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-password**. 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.

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-password
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. 85).

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 private repositories (p. 21).

Error: Cannot perform an interactive login from a non TTY device

If you receive the error `Cannot perform an interactive login from a non TTY device`, the following troubleshooting steps should help.

- Verify that you're using AWS CLI version 2 and that you don't have a conflicting version of AWS CLI version 1 on your system. For more information, see Installing or updating the latest version of the AWS CLI.
- Verify that you've configured your AWS CLI with valid credentials. For more information, see Installing or updating the latest version of the AWS CLI.

- Verify that the syntax of your AWS CLI command is correct.

Troubleshooting pull through cache issues

When pulling an upstream image using a pull through cache rule, the following are the most common errors you may receive.

Repository does not exist

An error indicating that the repository doesn't exist is most often caused by either the repository not existing in your Amazon ECR private registry or the `ecr:CreateRepository` permission not being granted to the IAM principal pulling the upstream image. To resolve this error, you should verify that the repository URI in your pull command is correct, the required IAM permissions are granted to the IAM principal pulling the upstream image, or that the repository for the upstream image to be pushed to is created in your Amazon ECR private registry before doing the upstream image pull. For more information about the required IAM permissions, see Required IAM permissions (p. 43)

The following is an example of this error.

```
Error response from daemon: repository 111122223333.dkr.ecr.us-east-1.amazonaws.com/
ecr-public/amazonlinux/amazonlinux not found: name unknown: The repository with
 name 'ecr-public/amazonlinux/amazonlinux' does not exist in the registry with id
 '111122223333'
```

Requested image not found

An error indicating that the image can't be found is most often caused by either the image not existing in the upstream registry or the `ecr:BatchImportUpstreamImage` permission not being granted to the IAM principal pulling the upstream image but the repository already being created in your Amazon ECR private registry. To resolve this error, you should verify the upstream image and image tag name is correct and that it exists and the required IAM permissions are granted to the IAM principal pulling the upstream image. For more information about the required IAM permissions, see Required IAM permissions (p. 43).

The following is an example of this error.

```
Error response from daemon: manifest for 111122223333.dkr.ecr.us-east-1.amazonaws.com/
ecr-public/amazonlinux/amazonlinux:latest not found: manifest unknown: Requested image
 not found
```

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 perform a basic scan on an image that was built using an operating system that Amazon ECR doesn't support basic 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.

Important
When using enhanced scanning, Amazon Inspector supports scanning for specific operating systems and media types. For a full list, see Supported operating systems and media types in the *Amazon Inspector User Guide*.

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.

Understanding scan status SCAN_ELIGIBILITY_EXPIRED

When enhanced scanning using Amazon Inspector is enabled for your private registry and you are viewing your scan vulnerabilities, you may see a scan status of SCAN_ELIGIBILITY_EXPIRED. The following are the most common causes of this.

- When you initially turn on enhanced scanning for your private registry, Amazon Inspector only recognizes images pushed to Amazon ECR in the last 30 days, based on the image push timestamp. Older images will have the SCAN_ELIGIBILITY_EXPIRED scan status. If you'd like these images to be scanned by Amazon Inspector you should push them again to your repository.
- If the **ECR re-scan duration** is changed in the Amazon Inspector console and that time elapses, the scan status of the image is changed to `inactive` with a reason code of `expired`, and all associated findings for the image are scheduled to be closed. This results in the Amazon ECR console listing the scan status as SCAN_ELIGIBILITY_EXPIRED.

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
Amazon ECR image signing	Amazon ECR and AWS Signer added support for creating and pushing container image signatures using the Notary client. For more information, see Signing an image (p. 38).	6 June 2023
Added Kubernetes container registry to pull through cache rules	Amazon ECR added support for creating pull through cache rules for the Kubernetes container registry. For more information, see Using pull through cache rules (p. 42).	1 June 2023
Amazon ECR enhanced scanning duration support	Amazon Inspector added support for setting the duration that your repositories are monitored for when enhanced scanning is enabled. For more information, see Changing the enhanced scanning duration (p. 71).	28 June 2022
Amazon ECR sends repository pull count metrics to Amazon CloudWatch	Amazon ECR sends repository pull count metrics to Amazon CloudWatch. For more information, see Amazon ECR repository metrics (p. 122).	6 January 2022
Expanded replication support	Amazon ECR added support for filtering which repositories are replicated. For more information, see Private image replication (p. 49).	21 September 2021
AWS managed policies for Amazon ECR	Amazon ECR added documentation of AWS managed policies. For more information, see AWS managed policies for Amazon Elastic Container Registry (p. 93).	24 June 2021
Cross-Region and cross-account replication	Amazon ECR added support for configuring replication settings for your private registry. For more information, see Private registry settings (p. 15).	8 December 2020
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. 36).	24 August 2020
Encryption at rest	Amazon ECR added support for configuring encryption for your repositories using server-side encryption with customer managed keys stored in AWS Key Management Service (AWS KMS). For more information, see Encryption at rest (p. 108).	29 July 2020
Multi-architecture images	Amazon ECR added support for creating and pushing Docker manifest lists which are used for multi-architecture images.	28 April 2020

Change	Description	Date
	For more information, see Pushing a multi-architecture image (p. 35).	
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. 121).	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. 135).	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. 14).	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 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. 67).	24 Oct 2019
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. 118).	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. 66).	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. 114).	25 Jan 2019

Change	Description	Date
Resource tagging	Amazon ECR added support for adding metadata tags to your repositories. For more information, see Tagging a private repository (p. 29).	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. 55).	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. 78).	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 Glossary Reference*.