SoftWare Repository for Container

User Guide

Issue	04
Date	2022-03-15

HUAWEI TECHNOLOGIES CO., LTD.

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process*. For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

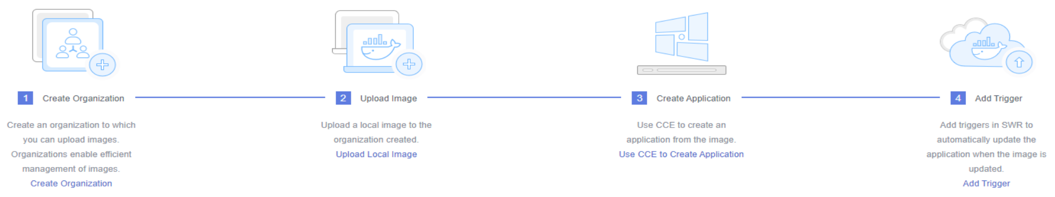
1 Service Overview

1.1 Introduction

SoftWare Repository for Container (SWR) allows you to easily manage the full lifecycle of container images and facilitates secure deployment of images for your applications.

SWR can either work with CCE or be used as an independent container image repository.

Figure 1-1 How SWR works



1 Create Organization

Create an organization to which you can upload images. Organizations enable efficient management of images.
Create Organization

2 Upload Image

Upload a local image to the organization created.
Upload Local Image

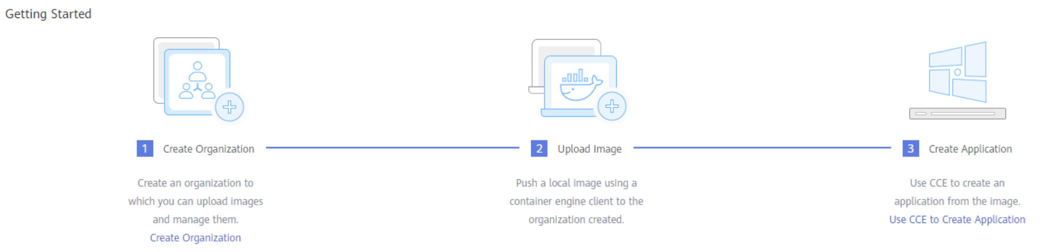
3 Create Application

Use CCE to create an application from the image.
Use CCE to Create Application

4 Add Trigger

Add triggers in SWR to automatically update the application when the image is updated.
Add Trigger

Figure 1-2 How SWR works



Getting Started

1 Create Organization

Create an organization to which you can upload images and manage them.
Create Organization

2 Upload Image

Push a local image using a container engine client to the organization created.

3 Create Application

Use CCE to create an application from the image.
Use CCE to Create Application

Features

- **Full lifecycle management of images**

  SWR manages the whole lifecycle of your container images, including push, pull, and deletion.
- **Private image repository and access control**

  Private image repository and fine-grained permission management allow you to grant different access permissions, namely, read, write, and edit, to different users.
- **Large scale image distribution acceleration**

  SWR uses the image pull acceleration technology to ensure faster image pull for CCE clusters in high concurrency scenarios.
- **Automatic deployment update through triggers**

  Image deployment can be triggered automatically upon image update. Simply set a trigger to the desired image. Every time the image is updated, the application deployed with this image will be automatically updated.

Accessing SWR

The cloud platform provides a web-based management console and HTTPS-based APIs through which you can access the SWR service.

- Using APIs

  If you want to integrate SWR into a third-party system for secondary development, use APIs to access SWR. For details, see *SWR API Reference*.
- Using the management console

  Use this mode if you do not want to integrate SWR into a third-party system.

1.2 Advantages

Ease of Use

- You can directly push and pull container images without platform build or O&M.
- SWR provides an easy-to-use management console for full lifecycle management over container images.

Security and Reliability

- SWR uses HTTPS to secure image transmission, and provides multiple isolation mechanisms between and inside accounts to control access to images.
- Based on professional storage services, SWR provides highly reliable storage service for your container images.

Image Acceleration

SWR uses the image pull acceleration technology to ensure faster image pull for CCE clusters in high concurrency scenarios.

1.3 Application Scenarios

Image Lifecycle Management

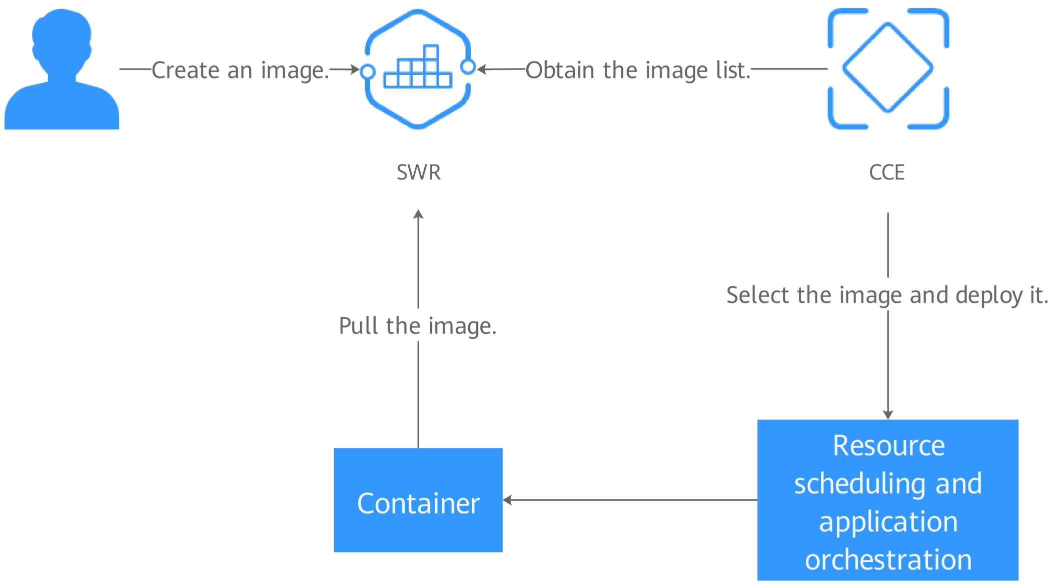
You can use SWR to build, push, pull, synchronize, and delete container images.

Advantages

- Pull acceleration ensures faster image pull for CCE clusters.
- Up to 99.999999999% image storage reliability is achieved by working with Object Storage Service (OBS).
- Fine-grained authorization allows you to control access to specific images and images in specific organizations.

Related service: Cloud Container Engine (CCE)

Figure 1-3 SWR working with CCE


Create an image.
Obtain the image list.
SWR
CCE
Pull the image.
Select the image and deploy it.
Container
Resource scheduling and application orchestration

1.4 Basic Concepts

Image

Images are like templates that include everything needed to run applications. When deploying containerized applications, you can use images from the Docker image center and your private image registries. For example, an image can contain a complete Ubuntu operating system, in which only the required programs and dependencies are installed. Docker images are used to create Docker containers. Docker provides an easy way to create and update your own images. You can also pull images created by other users.

Container

A container is a running instance of a Docker image. Multiple containers can run on one node. Containers are actually software processes. Unlike traditional software processes, containers have separate namespaces and do not run directly on a host.

Images become containers at runtime, that is, containers are created from images. Containers can be created, started, stopped, deleted, and suspended.

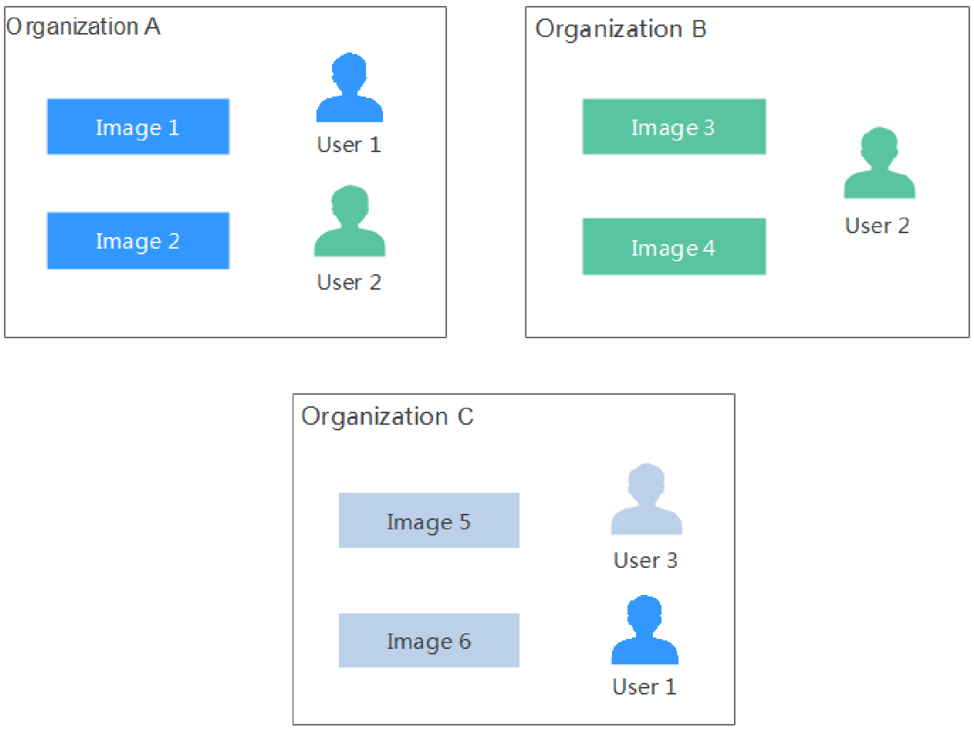
Repository

Image repositories are used for storing Docker images. An image repository hosts different versions of a specific containerized application.

Organization

Organizations are used to isolate image repositories. With each organization being limited to one company or department, images can be managed in a centralized and efficient manner. A user can access different organizations as long as the user has corresponding permissions. Different permissions, namely read, write, and manage, can be assigned to different users in the same account.

Figure 1-4 Organization



1.5 Notes and Constraints

Quotas

Quotas are imposed on the number of organizations a user can create. **Table 1-1** lists the quotas imposed by SWR.

Table 1-1 SWR resource quotas

Resource Type	Quota
Organization	5

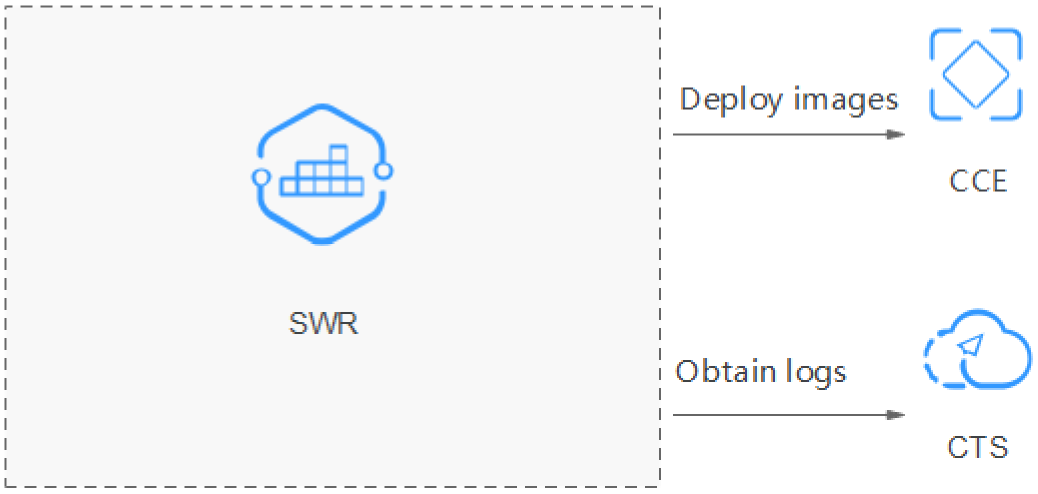
Requirements on Images to Upload

- If you use the container engine client to push images to SWR, each image layer cannot exceed 10 GB.
- If you use the SWR console to upload images, a maximum of 10 files can be uploaded at a time. The size of a single file (including the decompressed files) cannot exceed 2 GB.

1.6 Related Services

SWR works with other cloud services and requires permissions to access them. For details, see **Figure 1-5**.

Figure 1-5 Relationship between SWR and other services



- Cloud Container Engine (CCE)

  CCE is a high-performance, high-reliability service through which enterprises can manage containerized applications. CCE supports native Kubernetes applications and tools, allowing you to easily set up a container runtime environment on the cloud.

  SWR works seamlessly with CCE to allow you to deploy your images held by SWR on CCE clusters.

- Cloud Trace Service (CTS)

  CTS generates traces to enable you to get a history of operations performed on cloud service resources. The content of a trace includes operation requests sent using the management console or open APIs as well as the operation results. You can view all generated traces to query, audit, and backtrack performed operations.

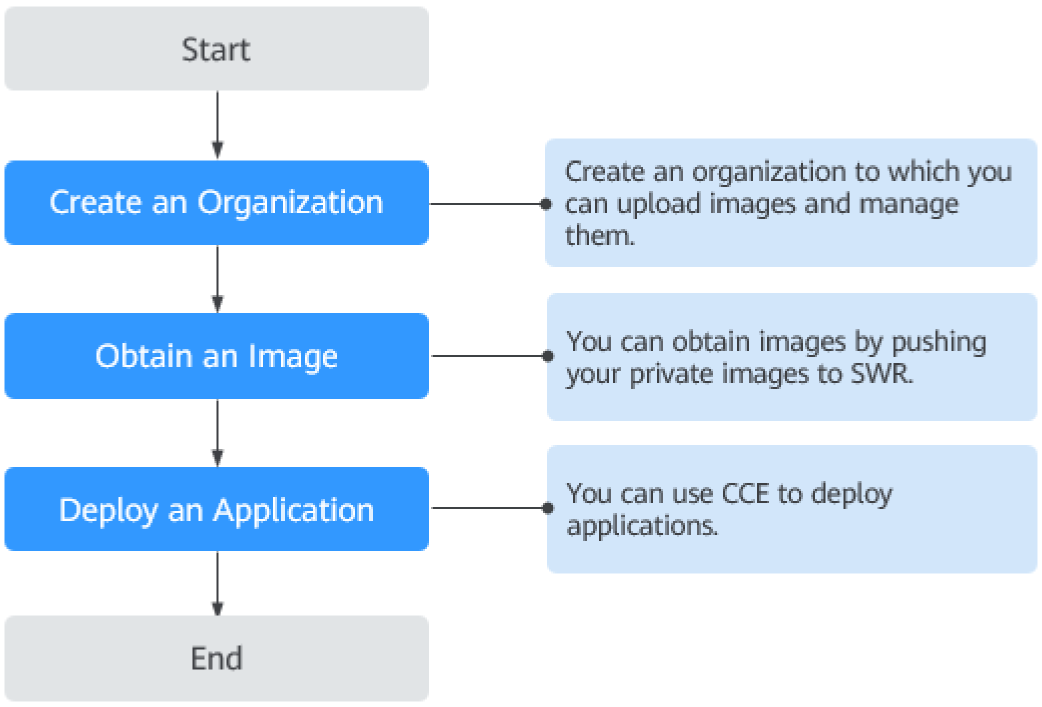
  With CTS, you can record operations associated with SWR for future query, audit, and backtrack operations.

2 Overview

SoftWare Repository for Container (SWR) allows you to easily manage the full lifecycle of container images and facilitates secure deployment of images for your applications.

SWR provides private image repositories and fine-grained permission management, allowing you to grant different access permissions, namely, read, write, and edit, to different users. You can use triggers to automatically update applications when images are updated.

Figure 2-1 How SWR works


Start
Create an Organization
Create an organization to which you can upload images and manage them.
Obtain an Image
You can obtain images by pushing your private images to SWR.
Deploy an Application
You can use CCE to deploy applications.
End

3 Basics of the Container Engine

The container engine, namely, Docker, is an open-source engine which allows you to create a lightweight, portable, and self-sufficient container for any application. SWR is compatible with Docker, allowing you to use Docker CLI and APIs to manage your images.

Installing Docker

Before installing Docker, get a basic understanding of what Docker is and how it works. For more information, see **Docker Documentation**.

Docker is compatible with almost all operating systems. Select a Docker version that best suits your needs. If you are not sure which Docker community edition to use, see **https://docs.docker.com/engine/install/**.

NOTE

- Before using SWR to store container images, ensure that the Docker client used to push images to SWR is updated to 1.11.2 or later.
- Bind an elastic IP address first if your server runs in a private network as the installation requires Internet connection.

On a device running Linux, run the following commands to quickly install Docker:

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
sudo systemctl daemon-reload
sudo systemctl restart docker
```

Building a Container Image

This section walks you through the steps of using a Dockerfile to build a container image for a simple web application. Dockerfile is a text file that contains all the instructions a user can call on the command line to build an image. A container image is a stack consisting of multiple layers. Each instruction creates a layer.

When using a browser to access a containerized application built from a Nginx image, you will see the default Nginx welcome page. In this section, you will build a new image based on the Nginx image to change the welcome message to **Hello, SWR!**

Step 1 Log in to the device running Docker as a root user.

Step 2 Run the following commands to create an empty file named **Dockerfile**:

mkdir mynginx

cd mynginx

touch Dockerfile

Step 3 Edit Dockerfile.

vim Dockerfile

Add the following instructions to the Dockerfile:

```
FROM nginx
RUN echo '<h1>Hello,SWR!</h1>' > /usr/share/nginx/html/index.html
```

In the preceding instructions:

- **FROM**: creates a layer from the base image. A valid Dockerfile must start with a **FROM** instruction. In this example, the **Nginx** image is used as the base image.
- **RUN**: executes a command to create a new layer. One of its syntax forms is RUN <command>. In this example, the **echo** command is executed to display **Hello, SWR!**

Save the changes and exit.

Step 4 Run **docker build** [*option*] <*context path*> to build an image.

docker build -t nginx:v1 .

- **-t nginx:v1**: specifies the image name and tag.
- **.**: indicates the path where the Dockerfile is located. All contents in this path are packed and sent to the Docker to build an image.

Step 5 Run the following command to check the created image. The command output shows that the nginx image has been created with a tag of v1.

docker images

----End

Creating an Image Package

This section describes how to compress a container image into a .tar or .tar.gz package.

Step 1 Log in to the device running Docker as a root user.

Step 2 Run the following command to list images.

docker images

Check the name and tag of the image to be compressed.

Step 3 Run the following command to compress the image into a package.

docker save [OPTIONS] IMAGE [IMAGE...]

NOTE

OPTIONS: You can set this to **--output** or **-o**, indicating that the image is exported to a file. The file should be in either .tar or .tar.gz.

Sample:

```
$ docker save nginx:latest > nginx.tar
$ ls -sh nginx.tar
108M nginx.tar

$ docker save php:5-apache > php.tar.gz
$ ls -sh php.tar.gz
372M php.tar.gz

$ docker save --output nginx.tar nginx
$ ls -sh nginx.tar
108M nginx.tar

$ docker save -o nginx-all.tar nginx
$ docker save -o nginx-latest.tar nginx:latest
```

----End

Importing an Image File

This section describes how to import an image package as an image using the **docker load** command.

There are two modes:

docker load < ***Path/File name.tar***

docker load --input ***Path/File name.tar*** or **docker load -i** ***Path/File name.tar***

Sample:

```
$ docker load --input fedora.tar
```

4 Image Management

4.1 Uploading an Image Through the Client

Scenario

This section walks you through the steps of uploading an image to SWR through the client by taking the **nginx:v1** image built in **3 Basics of the Container Engine** as an example. Uploading an image through the client is to run Docker commands on the client where Docker is installed to push the image to an image repository of SWR.

Notes and Constraints

- Each image layer uploaded through the client cannot exceed 10 GB.
- The Docker client version must be 1.11.2 or later.

Prerequisites

You have created an organization in SWR. For details, see **Creating an Organization**.

Procedure

Step 1 Access SWR.

1. Log in to the SWR console and the VM running the container engine as the **root** user.
2. In the navigation pane on the left, choose **Dashboard** and click **Generate Login Command** in the upper right corner. On the displayed page, click ⧉ to copy the login command.

   NOTE

   – A temporary login command is valid for 24 hours. For details about how to obtain a login command that will remain valid for a long term, see **4.2 Obtaining a Long-Term Valid Login Command**. After you obtain a long-term valid login command, your temporary login commands will still be valid as long as they are in their validity periods.

   – The domain name at the end of the login command is the image repository address. Record the address for later use.
3. Run the **docker login** command on your Docker client (a device that has Docker installed).

   The message "Login Succeeded" will be displayed upon a successful login.

Step 2 Run the following command on the device where Docker is installed to label the **nginx** image:

docker tag *[Image name 1*:*tag 1] [Image repository address]*/*[Organization name]*/*[Image name 2*:*tag 2]*

In the preceding command:

- [Image name 1:tag 1]: Replace it with the actual name and tag of the image to upload.
- [Image repository address]: You can query the address on the SWR console, that is, the domain name at the end of the login command in **Step 1.2**.
- [Organization name]: Replace it with the name of the organization created.
- [Image name 2: tag 2]: Replace it with the desired image name and tag.

Example:

docker tag nginx:v1 swr.ae-ad-1.myhuaweicloud.com/group/nginx:v1

Step 3 Push the image to the image repository by running the following command:

docker push *[Image repository address]*/*[Organization name]*/*[Image name* 2:*tag 2]*

Example:

docker push swr.ae-ad-1.myhuaweicloud.com/group/nginx:v1

The following information will be returned upon a successful push:

```
6d6b9812c8ae: Pushed
695da0025de6: Pushed
fe4c16cbf7a4: Pushed
v1: digest: sha256:eb7e3bbd8e3040efa71d9c2cacfa12a8e39c6b2ccd15eac12bdc49e0b66cee63 size: 948
```

To view the pushed image, refresh the **My Images** page.

----End

4.2 Obtaining a Long-Term Valid Login Command

Scenario

This section describes how to obtain a login command that is permanently valid.

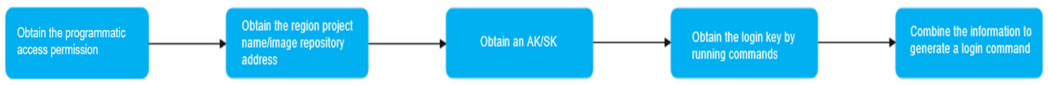
NOTE

For security purposes, it is advised to obtain the login command in the development environment.

Process

You can obtain a long-term valid login command as the following process:

Figure 4-1 Process


Obtain the programmatic access permission
Obtain the region project name/image repository address
Obtain an AK/SK
Obtain the login key by running commands
Combine the information to generate a login command

Procedure

Step 1 **Obtain the programmatic access permission. (If the current user has the permission, skip this step.)**

1. Log in to the management console as an administrator.
2. Click in the upper left corner and select a region and a project.
3. Click in the navigation pane on the left and choose **Management & Deployment** > **Identity and Access Management**.
4. Enter the name of the user to whom you want to grant the programmatic access permission in the search box on the **Users** page.
5. Click the user to go to its details page.
6. Click next to **Access Type**.
7. Select **Programmatic access**. (You can select only programmatic access or both access types.)

Step 2 Obtain the region, project name, and image repository address.

1. Log in to the management console, click your username in the upper right corner, and click **My Credentials**.
2. On the **Projects** tab page, search for the project corresponding to the current region.
3. Obtain the image repository address by referring to **Step 1.2**. The domain name at the end of the login command is the image repository address.

Step 3 Obtain an AK/SK.

NOTE

The access key ID (AK) and secret access key (SK) are a pair of access keys used together to authenticate users who wish to make API requests. The AK/AS pair provides functions similar to a password. If you already have an AK/SK, skip this step.

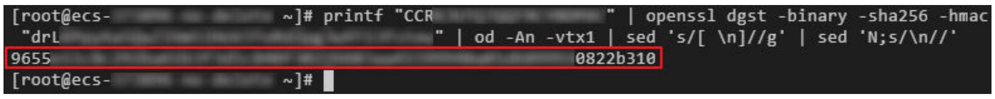
For details, see **How Do I Obtain an Access Key (AK/SK) in the ME-Abu Dhabi-OP5 Region?**

Step 4 Log in to a Linux PC and run the following command to obtain the login key:

printf "*$AK*" | openssl dgst -binary -sha256 -hmac "*$SK*" | od -An -vtx1 | sed 's/[\n]//g' | sed 'N;s/\n//'

In the command, **$AK** and **$SK** indicate the AK and SK obtained in **Step 3** respectively.

Figure 4-2 Sample command output



```
[root@ecs-[redacted] ~]# printf "CCR[redacted]" | openssl dgst -binary -sha256 -hmac
 "drL[redacted]" | od -An -vtx1 | sed 's/[ \n]//g' | sed 'N;s/\n//'
9655[redacted]0822b310
[root@ecs-[redacted] ~]#
```

Step 5 Put the information you obtained in the following format to generate a long-term valid login command:

docker login -u [*Regional project name*]@[*AK*] **-p** [*Login key*] [*Image repository address*]

In the command, the regional project name and image repository address are obtained in **Step 2**, the AK in **Step 3**, and the login key in **Step 4**.

NOTE

The login key is encrypted and cannot be decrypted. Therefore, other users cannot obtain the SK from -p.

The login command can be used on other devices.

Step 6 Run the **history -c** command to clear the operation records.

----End

4.3 Uploading an Image Through the SWR Console

Scenario

This section walks you through the steps of uploading an image to SWR through the SWR console.

Notes and Constraints

- A maximum of 10 files can be uploaded at a time. The size of a single file (including the decompressed files) cannot exceed 2 GB.
- The image package is created using container engine 1.11.2 or later.

Prerequisite

- You have created an organization in SWR. For details, see **Creating an Organization**.
- The image has been saved as a **.tar** or **.tar.gz** file. For details, see **Creating an Image Package**.

Procedure

Step 1 Log in to the SWR console.

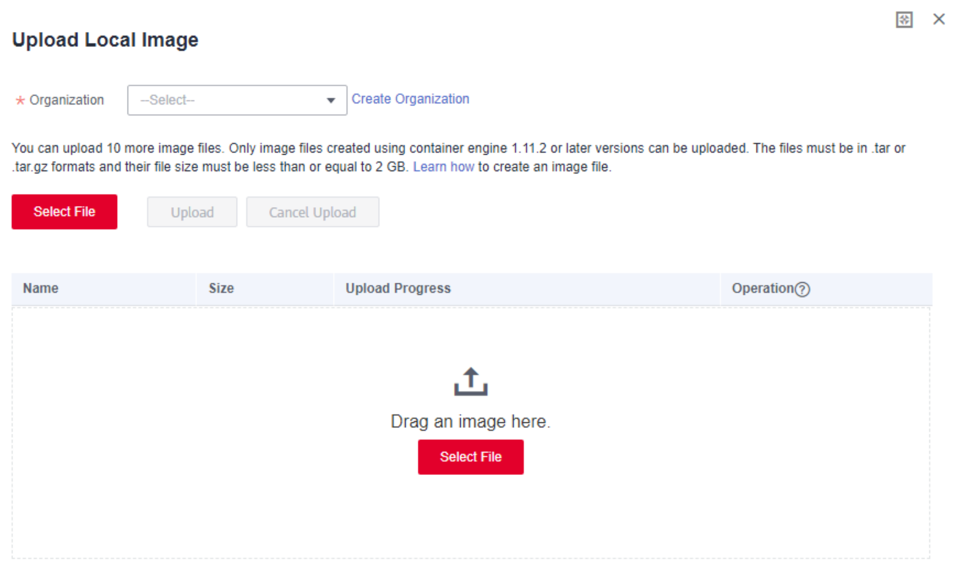
Step 2 In the navigation pane, choose **My Images**. Then click **Upload Through SWR**.

Step 3 On the page displayed, select an organization. Then, click **Select File** to upload the desired image file.

NOTE

If you select multiple images to upload, the system uploads them one by one. Concurrent upload is not supported.

Figure 4-3 Uploading an image



You can upload 10 more image files. Only image files created using container engine 1.11.2 or later versions can be uploaded. The files must be in .tar or .tar.gz formats and their file size must be less than or equal to 2 GB. Learn how to create an image file.

Step 4 Click **Start Upload**.

When the upload progress is complete, the image is successfully uploaded.

----End

4.4 Pulling an Image

Scenario

You can run the **docker pull** command to pull images from SWR.

Procedure

Step 1 Log in to the VM running the container engine as the **root** user.

Step 2 Obtain a login command by referring to **Step 1** and access SWR.

Step 3 Log in to the SWR console.

Step 4 In the navigation pane, choose **My Images** and click the target image.

Step 5 On the **Image Tags** tab page, in the same row as the target image tag, click ⧉ in the **Image Pull Command** column to copy the command.

Figure 4-4 Obtaining the image pull command

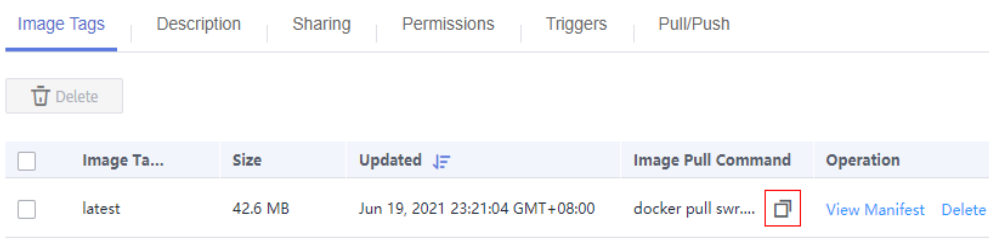


	Image Ta...	Size	Updated	Image Pull Command	Operation
☐	latest	42.6 MB	Jun 19, 2021 23:21:04 GMT+08:00	docker pull swr.... ⧉	View Manifest Delete

Step 6 Run the **image pull** command obtained in **Step 5** on the VM.

Run the **docker images** command to check whether the images are successfully pulled.

```
# docker images
REPOSITORY                        TAG      IMAGE ID       CREATED        SIZE
xxx/group/nginx                   v2.0.0   22f2bf2e2b4f   5 hours ago    22.8MB
```

Step 7 (Optional) Run the following command to save the image as an archive file:

docker save [*Image name:tag name*] **>** [*Archive file name*]

----End

4.5 Setting Image Attributes

Scenario

After uploading an image, you can set image attributes, including its type (public or private), category, and description.

Public images can be pulled by all users; whereas the access to private images requires corresponding permissions. You can add permissions, namely, read, write, and manage, to allow users to access your private images. For details, see **Granting Permissions for a Specific Image**.

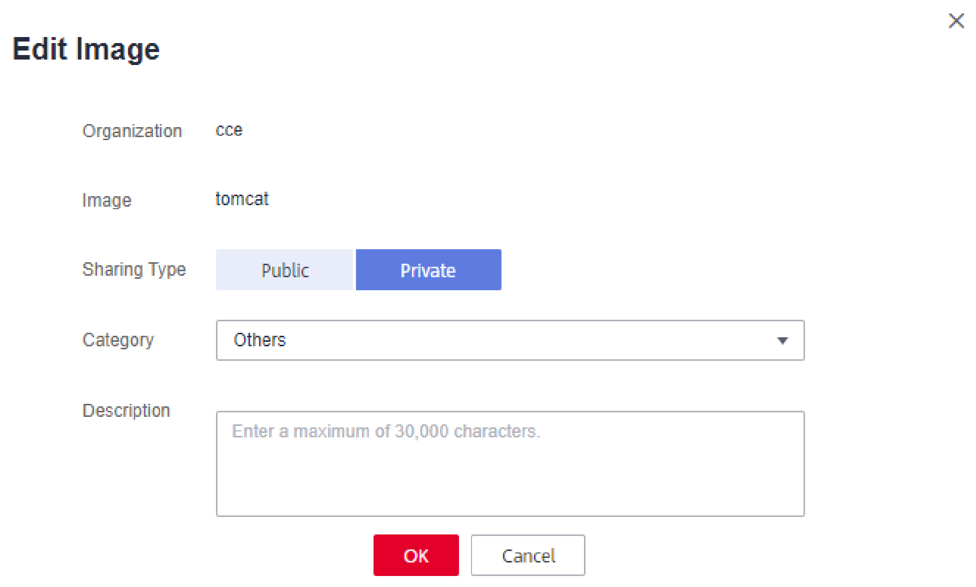
Procedure

Step 1 Log in to the SWR console.

Step 2 In the navigation pane, choose **My Images** and click the desired image.

Step 3 On the details page, click **Edit** in the upper right corner. On the page displayed, set **Sharing Type** (**Public** or **Private**), **Category**, and **Description**, and click **OK**.

Figure 4-5 Setting image attributes



Edit Image

Organization cce

Image tomcat

Sharing Type Public [✓ Private]

Category Others

Description Enter a maximum of 30,000 characters.

Table 4-1 Editing an image

Parameter	Description
Organization	The organization to which the image belongs
Image	Image name
Sharing Type	The following options are available: ● Public ● Private **NOTE** Public images can be pulled and used by all users. ● If your machine and the image repository are in the same region, you can access the image repository over private networks. ● If your machine and the image repository are in different regions, the node must have access to public networks to pull images from the image repository.

Parameter	Description
Category	The following options are available: ● Application server ● Linux ● Windows ● Arm ● Framework & Application ● Database ● Language ● Others
Descriptio n	Image description. Enter a maximum of 30,000 characters.

----End

4.6 Sharing Private Images

Scenario

You can share your **private images** with other accounts and grant the accounts permissions to pull the images.

A user under the account with which you shared the image can then log in to the SWR console to view the image by clicking **My Images** > **Shared Images**. On the tab page, the user can click the target image to check its detailed information, including the image tag and image pull command.

Notes and Constraints

- Only private images can be shared. Public images cannot be shared.
- Only users authorized to manage the private images can share images. The users with whom you share your images only have the read-only permission, which only allows them to pull the images.
- You can share images only with accounts in the same region. Cross-region image sharing is not supported.

Procedure

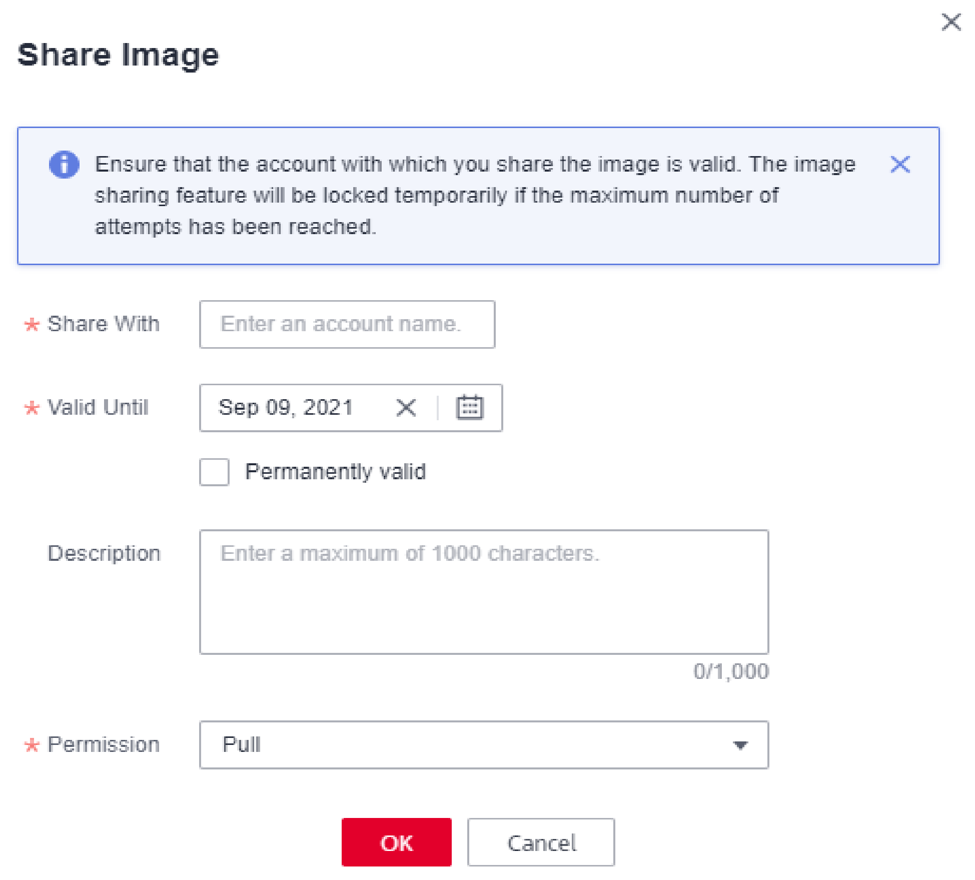
Step 1 Log in to the SWR console.

Step 2 In the navigation pane, choose **My Images** and click the target image.

Step 3 On the details page, click the **Sharing** tab.

Step 4 Click **Share Image**. Set parameters based on **Table 4-2**, and click **OK**.

Figure 4-6 Sharing an image



Share Image

Ensure that the account with which you share the image is valid. The image sharing feature will be locked temporarily if the maximum number of attempts has been reached.

* Share With	Enter an account name.
* Valid Until	Sep 09, 2021
	☐ Permanently valid
Description	Enter a maximum of 1000 characters. 0/1,000
* Permission	Pull

Table 4-2 Sharing an image

Parameter	Description
Share With	Enter an account name with which you want to share the image.
Valid Until	Set a validity period. If you want the image to be permanently accessible to the account, select **Permanently valid**.
Description	Enter a maximum of 1,000 characters.
Permission	Only the **Pull** permission is supported currently.

Step 5 To view all the images that you have shared, choose **My Images** in the navigation pane, click the **Private Images** tab, and select **Display only shared images**.

----End

4.7 Adding a Trigger

Scenario

SWR works with Cloud Container Engine (CCE) to achieve automatic application update. When images are updated, these new images can be automatically deployed to update the applications that use these images. You only need to add a trigger to the desired images. Every time these images are updated, they can trigger automatic updates of the applications that use them.

Prerequisite

A containerized application has been created on CCE by using an image from SWR.

To create an application, log in to the CCE console and create a workload.

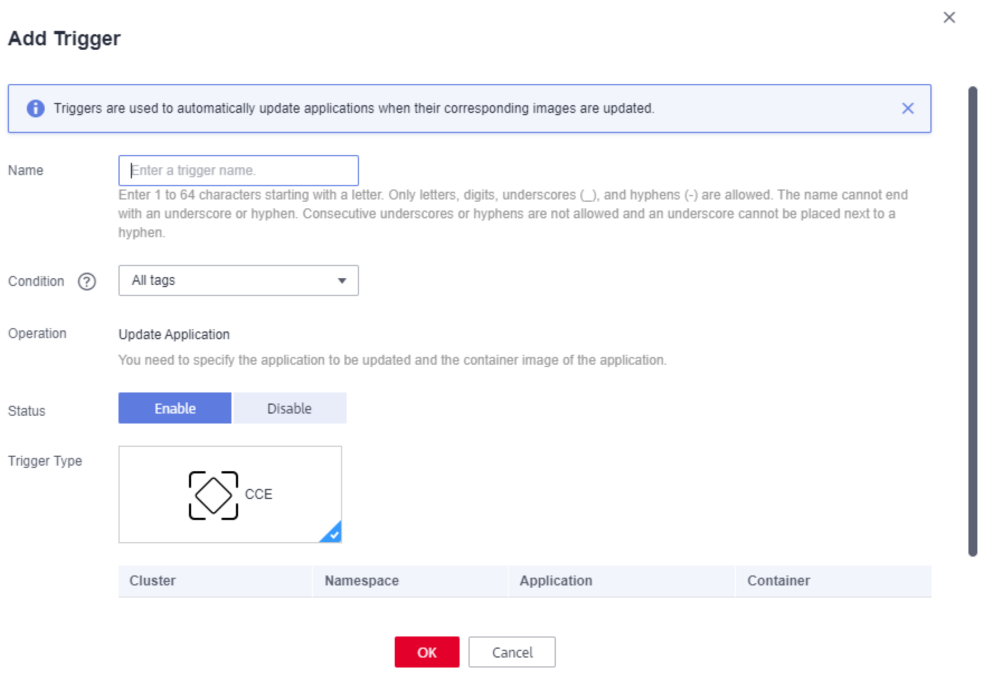
Procedure

Step 1 Log in to the SWR console.

Step 2 In the navigation pane on the left, choose **My Images**, and click the target image.

Step 3 Click the **Triggers** tab, then click **Add Trigger**. On the page displayed, configure the following parameters according to **Table 4-3** and click **OK**.

Figure 4-7 Adding a trigger



Add Trigger

Triggers are used to automatically update applications when their corresponding images are updated.

Name: Enter a trigger name.
Enter 1 to 64 characters starting with a letter. Only letters, digits, underscores (_), and hyphens (-) are allowed. The name cannot end with an underscore or hyphen. Consecutive underscores or hyphens are not allowed and an underscore cannot be placed next to a hyphen.

Condition: All tags

Operation: Update Application
You need to specify the application to be updated and the container image of the application.

Status: [✓ Enable] Disable

Trigger Type: [✓ CCE]

Cluster	Namespace	Application	Container

Table 4-3 Trigger

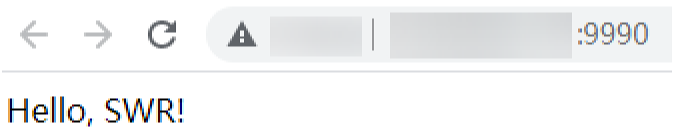
Parameter	Description
Name	The name can contain 1 to 64 characters, and must start with a letter. Only letters, digits, underscores (_), and hyphens (-) are allowed. The name cannot end with an underscore or hyphen. Consecutive underscores or hyphens are not allowed and an underscore cannot be placed next to a hyphen.
Condition	The following trigger conditions are supported: ● All tags: Trigger when any image tags are generated or updated. ● Specific tag: Trigger when a specific image tag is generated or updated. ● Tags matching regular expression: Trigger when an image tag that matches the specified regular expression is generated or updated. The regular expression rules are as follows: – *****: matches any field that does not contain the path separator **/**. – ******: matches any field that contains the path separator **/**. – **?**: Matches any single character except **/**. – **{option 1, option 2, ...}**: matches multiple options.
Operation	Operation that will be triggered when the conditions you set are met. Currently, only application update is supported. You need to specify the application to be updated and the container image of the application.
Status	Select **Enable**.
Trigger Type	Select **CCE**.
Application	Select the container whose image you want to update.

----End

Example

A Deployment named **nginx** is created using the Nginx v1 image. The Deployment provides service to external systems with a welcome page displaying **Hello, SWR!**

Figure 4-8 Nginx deployment



Hello, SWR!

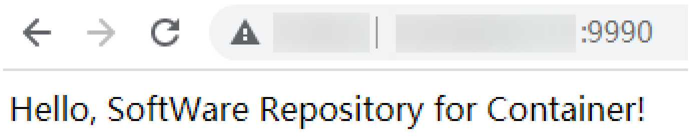
1. Add a trigger to the Nginx image.

   Set **Name** to **All_tags**, **Condition** to **All tags**, and select the application and all its containers that use the Nginx image.
2. Check the image tags. The Nginx image is pushed to SWR. The welcome page of the Deployment created using this new image should display **Hello, SoftWare Repository for Container!**
3. Check whether the deployment is triggered successfully.

   On the **Triggers** tab page, click ˅ and the trigger is successful.

   The welcome page of the Deployment displays **Hello, SoftWare Repository for Container!**

Figure 4-9 Updated Nginx


:9990

Hello, SoftWare Repository for Container!

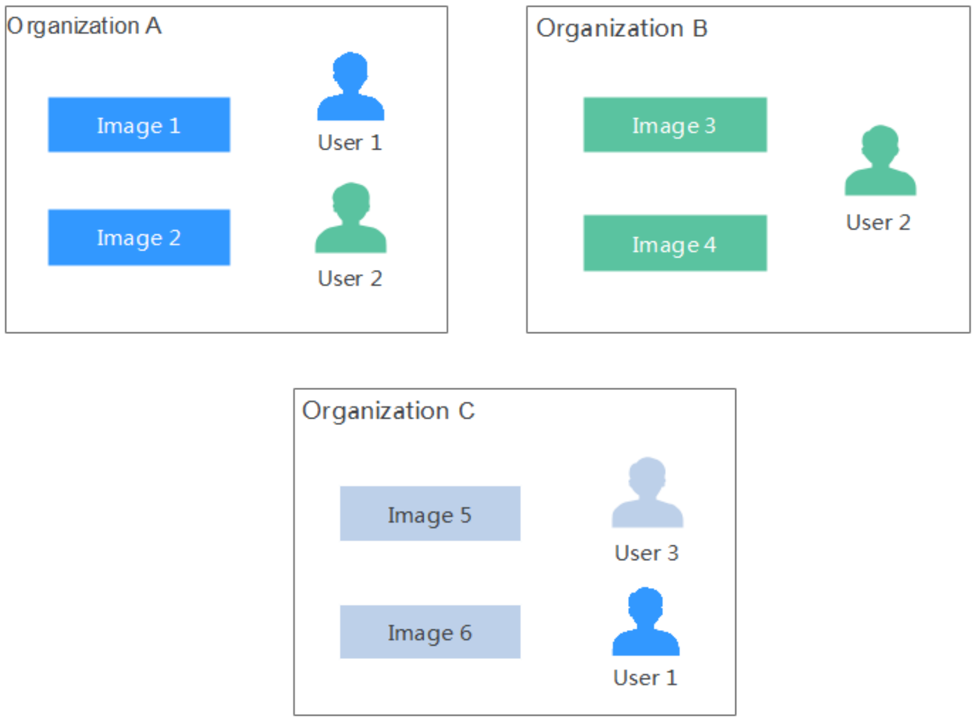
5 Organization Management

Scenario

Organizations enable efficient management of images. Organizations are used to isolate image repositories. With each organization being limited to one company or department, images can be managed in a centralized and efficient manner. An image name needs to be unique within an organization. The same user can access different organizations as long as the user has sufficient permissions, as shown in **Figure 5-1**.

You can grant different permissions, namely, read, write, and manage, to users created by the same account. For details, see **6 User Permissions**.

Figure 5-1 Organization



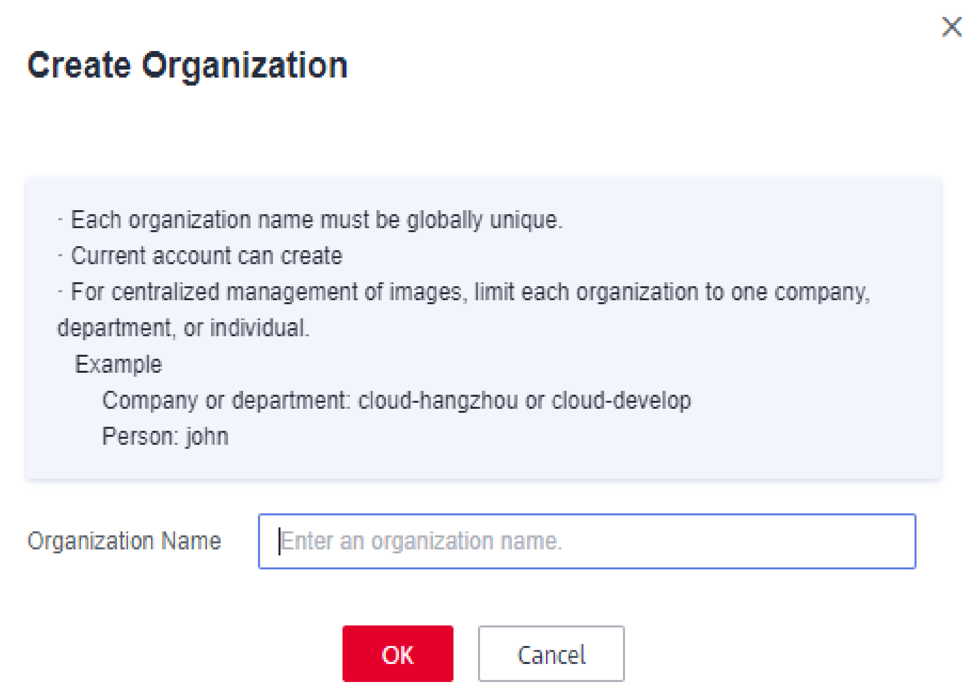
Creating an Organization

You can create organizations based on the organizational structure of your enterprise to facilitate image resource management. Create an organization before you push an image.

Step 1 Log in to the SWR console.

Step 2 In the navigation pane on the left, choose **Organization Management** and click **Create Organization**. On the page displayed, specify **Organization Name** and click **OK**.

Figure 5-2 Creating an organization



Create Organization

· Each organization name must be globally unique.
· Current account can create
· For centralized management of images, limit each organization to one company, department, or individual.
Example
Company or department: cloud-hangzhou or cloud-develop
Person: john

Organization Name: Enter an organization name.

NOTE

- The organization name must be globally unique. If a message is displayed indicating that the organization already exists, the organization name may have been used by another user. Use another organization name.
- After a tenant is deleted, residual organization resources may exist. In this case, the message indicating that the organization already exists could also be displayed when you create an organization. Use another organization name.

----End

Viewing the Images of an Organization

After you create an organization and push images to it, you can view the image list of the organization.

Step 1 Log in to the SWR console.

Step 2 In the navigation pane, choose **Organization Management**. On the page displayed, click the desired organization name in the list.

Step 3 To view the images of this organization, click the **Images** tab.

Figure 5-3 Viewing the images of an organization

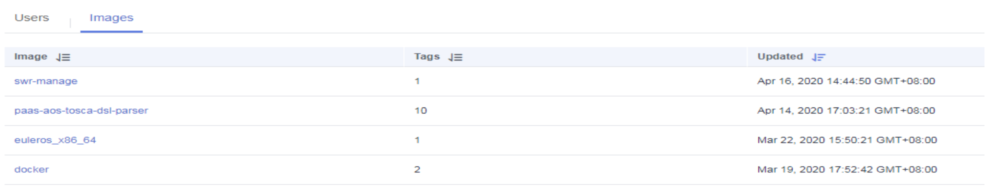


Image	Tags	Updated
swr-manage	1	Apr 16, 2020 14:44:50 GMT+08:00
paas-aos-tosca-dsl-parser	10	Apr 14, 2020 17:03:21 GMT+08:00
euleros_x86_64	1	Mar 22, 2020 15:50:21 GMT+08:00
docker	2	Mar 19, 2020 17:52:42 GMT+08:00

----End

Deleting an Organization

Before deleting an organization, delete all the images in the organization.

Step 1 Log in to the SWR console.

Step 2 In the navigation pane, choose **Organization Management**. On the page displayed, click the desired organization name in the list.

Step 3 Click **Delete** in the upper right corner. In the displayed dialog box, enter **DELETE** as prompted and click **Yes**.

----End

NOTICE

Before you delete a tenant, delete its organizations first; otherwise, residual organization resources may exist. When you create an organization that has the same name with the residual organization, a message is displayed indicating that the organization already exists.

6 User Permissions

Scenario

To manage SWR permissions, you can use Identity and Access Management (IAM). If you have the SWR Administrator or Tenant Administrator permission, you become an admin user of SWR accounts. You can grant permissions to other IAM users in SWR.

If you are not an SWR account admin user, you can request an SWR account admin user to grant you permissions to read, write, or manage a specific image or images in a specific organization.

NOTE

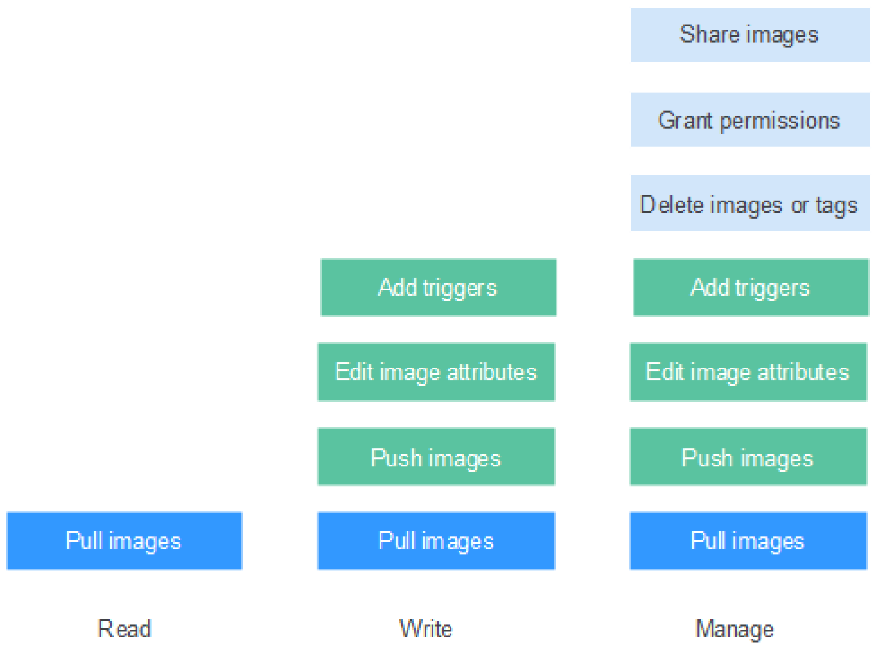
- An SWR account admin user is granted image management permission of all organizations by default, even if the user is not in the authorized user list of the organizations.

Authorization Method

You can grant permissions to users in SWR by using either of the following methods:

- **Grant permissions on an image details page** to allow users to read, write, and manage a specific image.
- **Grant permissions on an organization details page** to allow users to read, write, and manage all the images in an organization.

Figure 6-1 User permissions



You can grant the following three types of permissions to users:

- Read: Users can only pull images.
- Write: Users can pull and push images, add triggers, and edit image attributes.
- Manage: Users can pull and push images, delete images or tags, edit image attributes, grant permissions, add triggers, and share images with other users.

NOTE

To upload images to an organization, you require the write or manage permission for the organization to which images are uploaded. Write and manage permissions added on the image details pages will not be sufficient to upload images.

Granting Permissions for a Specific Image

To allow users to read, write, and manage a specific image, grant corresponding permissions to them on the details page of this image.

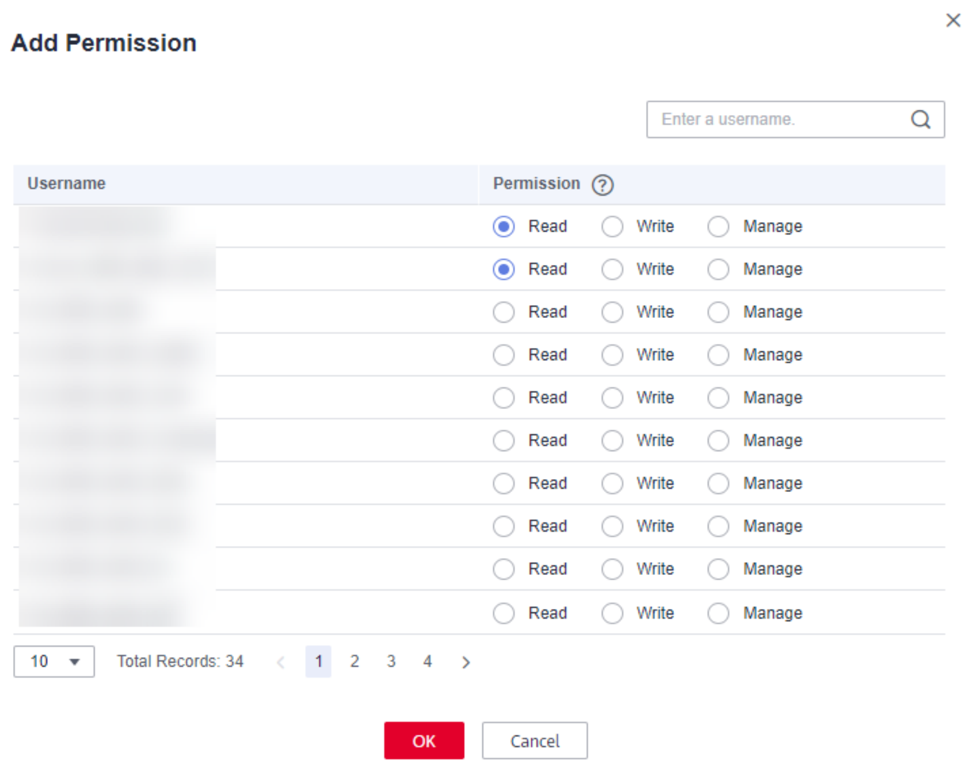
Step 1 Log in to the SWR console.

Step 2 In the navigation pane, choose **My Images** and click the desired image.

Step 3 On the image details page, click the **Permissions** tab.

Step 4 Click **Add Permission**. On the page displayed, click **Read**, **Write**, or **Manage** in the row of the desired username. Click **OK** to confirm.

Figure 6-2 Granting permissions for a specific image



----End

Modifying or Deleting Permissions for a Specific Image

You can also modify or delete user permissions on the image details page.

- To modify permissions, click **Modify** in the row of the desired username on the **Permissions** tab page. Select a permission in the **Permission** drop-down list, and click **Save** in the **Operation** column.
- To delete permissions, click **Delete** in the row of the desired username on the **Permissions** tab page, enter **DELETE** in the dialog box displayed, and then click **Yes**.

Granting Permissions for an Organization

To allow users to read, write, and manage all the images in an organization, grant corresponding permissions to them on the details page of this organization.

Only users with the **Manage** permission can grant permissions for other users.

Step 1 Log in to the SWR console.

Step 2 In the navigation pane, choose **Organization Management**. Then click **Details** in the row of the desired organization.

Step 3 On the **Users** tab page, click **Add Permission**. In the dialog box displayed, select permissions for users and click **OK**.

----End

Modifying or Deleting Permissions for an Organization

You can also modify and delete user permissions of an organization.

- To modify permissions, click **Modify** in the row of the desired username on the **Users** tab page. Select a permission in the **Permission** drop-down list, and click **Save** in the **Operation** column.
- To delete permissions, click **Delete** in the row of the desired username on the **users** tab page, enter **DELETE** in the dialog box displayed, and then click **Yes**.

7 Auditing

7.1 SWR Operations Supported by CTS

Scenario

CTS records operations on cloud resources in your account. You can use the logs to perform security analysis, track resource changes, audit, and locate faults.

With CTS, you can record operations associated with SWR for future query, audit, and backtrack operations.

Key Operations Recorded by CTS

Table 7-1 SWR operations that can be recorded by CTS

Operation	Resource Type	Trace Name
Granting permissions for an organization	usernamespaceauth	createUserNamespaceAuth
Modifying permissions for an organization	usernamespaceauth	updateUserNamespaceAuth
Deleting permissions for an organization	usernamespaceauth	deleteUserNamespaceAuth
Creating a software package	package	createPackage
Modifying a software package	package	updatePackage

Operation	Resource Type	Trace Name
Deleting a software package	package	deletePackage
Creating a repository	repository	createRepository
Modifying a repository	repository	updateRepository
Deleting a repository	repository	deleteRepository
Creating a version	version	createVersion
Modify a version	version	updateVersion
Deleting a version	version	deleteVersion
Uploading an image package	image	uploadImagePackage
Uploading a file	file	uploadFile
Downloading a file	file	downloadFile
Deleting a file	file	deleteFile
Creating an organization	usernamespace	createUserNamespace
Deleting an organization	usernamespace	deleteUserNamesapce
Creating a trigger	trigger	createTrigger
Modifying a trigger	trigger	updateTrigger
Deleting a trigger	trigger	deleteTrigger
Granting permissions for an image repository	userrepositoryauth	createUserRepositoryAuth
Modifying permissions for an image repository	userrepositoryauth	updateUserRepositoryAuth
Deleting permissions for an image repository	userrepositoryauth	deleteUserRepositoryAuth
Creating an image repository	imagerepository	createImageRepository
Modifying an image repository	imagerepository	updateImageRepository
Deleting an image repository	imagerepository	deleteImageRepository

Operation	Resource Type	Trace Name
Deleting an image tag	imagetag	deleteImageTag
Generating a login command	dockerlogincmd	createDockerConfig
Creating a shared image	imagerepositoryaccess-domain	createImageRepositoryAccess-Domain
Modifying a shared image	imagerepositoryaccess-domain	updateImageRepositoryAc-cessDomain
Deleting a shared image	imagerepositoryaccess-domain	deleteImageRepositoryAccess-Domain

7.2 Viewing Logs in CTS

Scenarios

After you enable CTS, the system starts recording operations performed on SWR resources. CTS stores operation records generated within a week.

This section describes how to view the records on the CTS console.

Procedure

Step 1 Log in to the CTS console.

Step 2 In the navigation pane, choose **Trace List**.

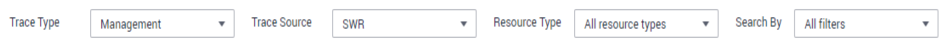
Step 3 Set the filter criteria and click **Query**.

The following filters are available:

- **Trace Type**, **Trace Source**, **Resource Type**, and **Search By**

  Select the desired filter criteria from the drop-down lists. Select **Management** for **Trace Type** and **SWR** for **Trace Source**.

  Figure 7-1 Setting filter criteria

  Trace Type: Management
  Trace Source: SWR
  Resource Type: All resource types
  Search By: All filters

  Among them,

  – If you select **Resource ID** for **Search By**, you need to enter a resource ID. Only whole word match is supported.

  – If you select **Resource name** for **Search By**, you need to select or enter a specific resource name.

- **Operator**: Select a specific operator from the drop-down list.
- **Trace Status**: Select **All trace statuses**, **Normal**, **Warning**, or **Incident**.

- Time range: You can select **Last 1 hour**, **Last 1 day**, **Last 1 week**, or **Customize**.

Step 4 On the left of the target record, click ⌄ to view details.

Step 5 Click **View Trace** in the upper right corner of the trace details area.

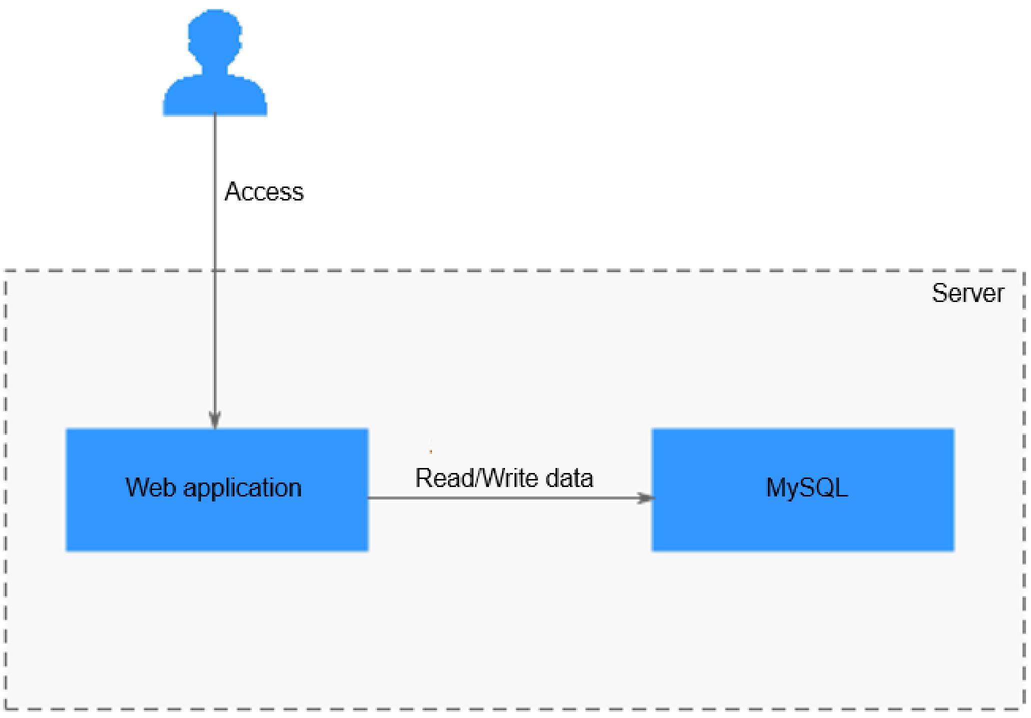
----End

8 Best Practices

8.1 Writing a Quality Dockerfile

This document walks you through how to compile an efficient Dockerfile, using the containerization of an application as an example. Based on the practices of SWR, this file exemplifies how to create images of fewer layers and smaller size to speed up image build process.

The following figure shows a common architecture of an enterprise portal website. This website consists of a web server that provides web services, and a database that stores user data. Normally, the website is deployed on a single server.



To containerize the application, a Dockerfile may be written as follows:

```
FROM ubuntu

ADD . /app

RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y nodejs ssh mysql
RUN cd /app && npm install

# this should start three processes, mysql and ssh
# in the background and node app in foreground
# isn't it beautifully terrible? <3
CMD mysql & sshd & npm start
```

However, the preceding Dockerfile, including the **CMD** command, is problematic.

To rectify and optimize the Dockerfile, here are some tips:

- Run Only One Process in Each Container
- Do Not Upgrade the Tag During Image Build
- Merge RUN Commands that Are of Similar Update Frequency
- Specify an Image Label
- Delete Unnecessary Files
- Select a Suitable Base Image
- Set WORKDIR and CMD
- (Optional) Use ENTRYPOINT
- Run the exec Command in ENTRYPOINT
- Use the COPY Command Preferentially
- Adjust the Order of COPY and RUN Commands
- Set Default Environment Variables, Mapping Ports, and Data Volumes
- Use the EXPOSE Command to Specify Listening Ports
- Use the VOLUME Command to Manage Data Volumes
- Use Labels to Configure Image Metadata
- Add the HEALTHCHECK Command
- Compile the .dockerignore File

Run Only One Process in Each Container

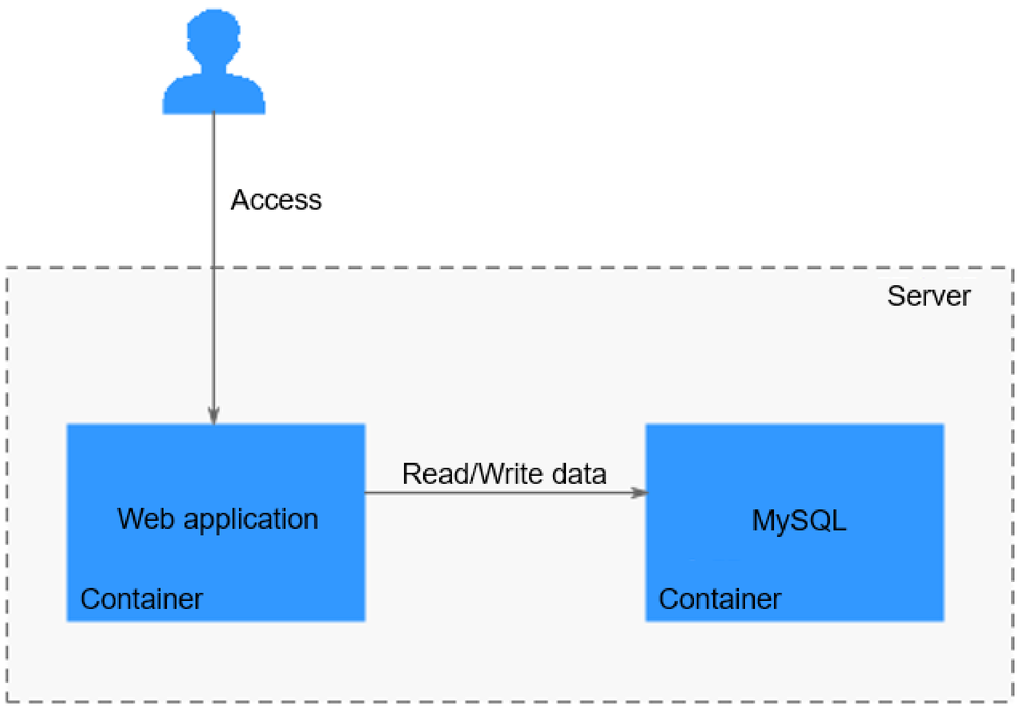
Technically, multiple processes, including database, frontend, backend, and SSH, can run on the same Docker container. However, this is not what containers are built for. Stuffing all the processes into one container not only makes the image extremely large in size, but also prolongs the container building time and wastes resources when you perform horizontal scaling. This is because the whole container has to be rebuilt every time you make small adjustments and the number of containers for each application can only be equally added during scaling in.

Therefore, usually, an application is split into microservices before containerization. You will benefit a lot from the microservice architecture:

- **Independent scaling**: After an application is split into independent microservices, you can adjust the number of pods for each microservice separately.

- **Faster development**: Since microservices are decoupled, they can be coded independently from each other.
- **Security assurance through isolation**: For an overall application, if a security vulnerability exists, attackers can use this vulnerability to obtain the permission for all functions of the application. However, in a microservice architecture, if a service is attacked, attackers can only obtain the access permission for this service, but cannot intrude other services.
- **Stabler service**: If one microservice breaks down, other microservices can still run properly.

To optimize the preceding sample Dockerfile, run the web application and MySQL in different containers.



You can modify them separately. As shown in the following example, MySQL is deleted from the sample Dockerfile. Only Node.js is installed.

```
FROM ubuntu

ADD . /app

RUN apt-get update
RUN apt-get upgrade -y

RUN apt-get install -y nodejs
RUN cd /app && npm install

CMD npm start
```

Do Not Upgrade the Tag During Image Build

To reduce image complexity, dependency, size, and build time, do not install any unnecessary packages in your images. For example, do not include a text editor in a database image.

Contact the package maintenance personnel if a package in the base image is out of date but you do not know which package it is. To upgrade a specific package automatically, for example, **foo**, run the **apt-get install -y foo** command.

apt-get upgrade brings great uncertainty to image build. Inconsistency between images might occur as you are not sure what packages have been installed by **apt-get upgrade** during image build. Therefore, **apt-get upgrade** is usually deleted.

The following is the sample Dockerfile without **apt-get upgrade**:

```
FROM ubuntu

ADD . /app

RUN apt-get update

RUN apt-get install -y nodejs
RUN cd /app && npm install

CMD npm start
```

Merge RUN Commands that Are of Similar Update Frequency

Like an onion, a Docker image consists of many layers. To modify an inner layer, you need to delete all outer layers. Docker images have the following features:

- Each command in a Dockerfile creates an image layer.
- Image layers are cached and reused.
- Cached image layers expire when the files they copy or variables specified in image build change.
- When a cached image layer expires, its subsequent cached image layers expire accordingly.
- Image layers are immutable. If a file is added into a layer and then deleted in the next layer, the file still exists in the image. The file just turns unavailable in the Docker container.

Therefore, merge multiple commands that are of similar updating probability to avoid unnecessary costs. In the sample Dockerfile, **Node.js** and **npm** are installed together. That means **Node.js** is reinstalled each time the source code is modified, which is time and resource consuming.

```
FROM ubuntu

ADD . /app

RUN apt-get update \
    && apt-get install -y nodejs \
    && cd /app \
    && npm install

CMD npm start
```

It would be better to write the Dockerfile as follows:

```
FROM ubuntu

RUN apt-get update && apt-get install -y nodejs
ADD . /app
RUN cd /app && npm install

CMD npm start
```

Specify an Image Label

When no label is specified for an image, the **latest** label is automatically used. Therefore, the **FROM ubuntu** command is the same as **FROM ubuntu:latest**. During an image update, when the **latest** tag points to a new tag, image build may fail.

To specify a label for the **ubuntu** image in the sample Dockerfile, label the image with **16.04** as follows:

```
FROM ubuntu:16.04

RUN apt-get update && apt-get install -y nodejs
ADD . /app
RUN cd /app && npm install

CMD npm start
```

Delete Unnecessary Files

Assume that you have updated the **apt-get** sources, installed some software packages, and saved them in the **/var/lib/apt/lists/** directory.

However, these files are not required in application running. To make the Docker image more lightweight, it is advised to delete these unnecessary files.

Therefore, in the sample Dockerfile, the files in the **/var/lib/apt/lists/** directory are deleted.

```
FROM ubuntu:16.04

RUN apt-get update \
    && apt-get install -y nodejs \
    && rm -rf /var/lib/apt/lists/*

ADD . /app
RUN cd /app && npm install

CMD npm start
```

Select a Suitable Base Image

In our sample Dockerfile, **ubuntu** is selected as the base image. However, as you only need to run the node program, there is no need to use a general-purpose base image. A node image would be a better choice.

The node image of the Alpine version is recommended. It is a mini-Linux operating system with a size of only 4 MB.

```
FROM node:7-alpine

ADD . /app
RUN cd /app && npm install

CMD npm start
```

Set WORKDIR and CMD

The **WORKDIR** command can be used to set a default directory where **RUN**, **CMD**, and **ENTRYPOINT** commands will be run.

The **CMD** command provides default commands for container creation. Write the command in an array. The elements of the array correspond to the words of the command one by one.

```
FROM node:7-alpine

WORKDIR /app
ADD . /app
RUN npm install

CMD ["npm", "start"]
```

(Optional) Use ENTRYPOINT

The **ENTRYPOINT** command is optional because it increases complexity. **ENTRYPOINT** is a script that is executed by default. It uses the specified commands as its parameters. It is usually used to create executable container images.

```
FROM node:7-alpine

WORKDIR /app
ADD . /app
RUN npm install

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

Run the exec Command in ENTRYPOINT

In the preceding **ENTRYPOINT** script, the **exec** command is used to run the node application. If the **exec** command is not used, the container cannot be successfully closed since the **SIGTERM** signal will be swallowed by the **bash** script process. The process started by running the **exec** command can replace the script process. In this way, all signals work normally.

Use the COPY Command Preferentially

The **COPY** command is simple. It is only used to copy files to images. Compared with it, the **ADD** command is more complex. **ADD** can be used to download remote files and decompress compressed packages.

```
FROM node:7-alpine

WORKDIR /app

COPY . /app
RUN npm install

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

Adjust the Order of COPY and RUN Commands

Place the parts that are not to be changed frequently at the front of your Dockerfile to make the most out of the image cache.

In the example Dockerfile, its source code changes frequently. Every time the image is built, the NPM is reinstalled. For example, copy **package.json** first, then install NPM, at last copy the rest of the source code. In this way, changes of the source code will not result in repetitive installation of NPM.

```
FROM node:7-alpine

WORKDIR /app

COPY package.json /app
RUN npm install
COPY . /app

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

Set Default Environment Variables, Mapping Ports, and Data Volumes

Environment variables may be required when running a Docker container. Setting default environment variables in Dockerfile is a good choice. In addition, you can set mapping ports and data volumes in the Dockerfile. Example:

```
FROM node:7-alpine

ENV PROJECT_DIR=/app

WORKDIR $PROJECT_DIR

COPY package.json $PROJECT_DIR
RUN npm install
COPY . $PROJECT_DIR

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

The environment variable specified by the **ENV** command can be used in a container. If you only need to specify the variables used in image build, you can use the **ARG** command.

Use the EXPOSE Command to Specify Listening Ports

The **EXPOSE** command is used to specify listening ports to containers. Select common ports for your applications. For example, for the image that provides the Apache web service, use **EXPOSE 80**; while for the image that provides the MongoDB service, use **EXPOSE 27017**.

For external access, use a flag to indicate how to map a specified port to the selected port during the execution of the **docker run** command.

```
FROM node:7-alpine

ENV PROJECT_DIR=/app

WORKDIR $PROJECT_DIR

COPY package.json $PROJECT_DIR
RUN npm install
COPY . $PROJECT_DIR

ENV APP_PORT=3000
EXPOSE $APP_PORT

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

Use the VOLUME Command to Manage Data Volumes

The **VOLUME** command is used to access database storage files, configuration files, or files and directories of created containers. It is strongly recommended that the **VOLUME** command be used to manage the image modules that can change or the modules modifiable for users.

In the example Dockerfile, a media directory is entered.

```
FROM node:7-alpine

ENV PROJECT_DIR=/app

WORKDIR $PROJECT_DIR

COPY package.json $PROJECT_DIR
RUN npm install
COPY . $PROJECT_DIR

ENV MEDIA_DIR=/media \
    APP_PORT=3000

VOLUME $MEDIA_DIR
EXPOSE $APP_PORT

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

Use Labels to Configure Image Metadata

Add labels to images to help organize images, record permissions, and automate image build. Starting with **LABEL**, add one or more labels with each label occupying one line.

NOTICE

If your string contains spaces, put the string in quotation marks ("") or convert it into escape sequence. If the string itself contains quotation marks, convert the quotation marks.

```
FROM node:7-alpine
LABEL com.example.version="0.0.1-beta"
```

Add the HEALTHCHECK Command

When running a container, you can enable the **--restart always** option. In this case, the Docker daemon restarts the container when the container crashes. This option is useful for containers that need to run for a long time. What if a container is running but unavailable? The **HEALTHCHECK** command enables Docker to periodically check the health status of containers. You only need to specify a command. If the containers are normal, **0** is returned. Otherwise, **1** is returned. When the request fails and the **curl --fail** command is run, a non-zero state is returned. Example:

```
FROM node:7-alpine
LABEL com.example.version="0.0.1-beta"

ENV PROJECT_DIR=/app
WORKDIR $PROJECT_DIR

COPY package.json $PROJECT_DIR
RUN npm install
COPY . $PROJECT_DIR
```

```
ENV MEDIA_DIR=/media \
    APP_PORT=3000

VOLUME $MEDIA_DIR
EXPOSE $APP_PORT
HEALTHCHECK CMD curl --fail http://localhost:$APP_PORT || exit 1

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

Compile the .dockerignore File

The functions and syntax of the **.dockerignore** file are similar to those of the **.gitignore** file. You can ignore unnecessary files to accelerate image build and reduce image size.

Before image build, Docker needs to prepare the context by collecting all required files to the process. By default, the context contains all files in the Dockerfile directory. However, files in such as those in the **.git** directory are unnecessary.

Example:

```
.git/
```

9 FAQs

9.1 General FAQs

9.1.1 What Is SWR?

SoftWare Repository for Container (SWR) allows users to easily manage the full lifecycle of container images and facilitates secure deployment of images for your applications.

9.1.2 About SWR

How Many Images Can Be Stored in SWR?

SWR has no limit on the number of images. You can upload any number of images.

What Is the Bandwidth of SWR?

The bandwidth of SWR dynamically changes based on actual usage.

Is SWR Charged?

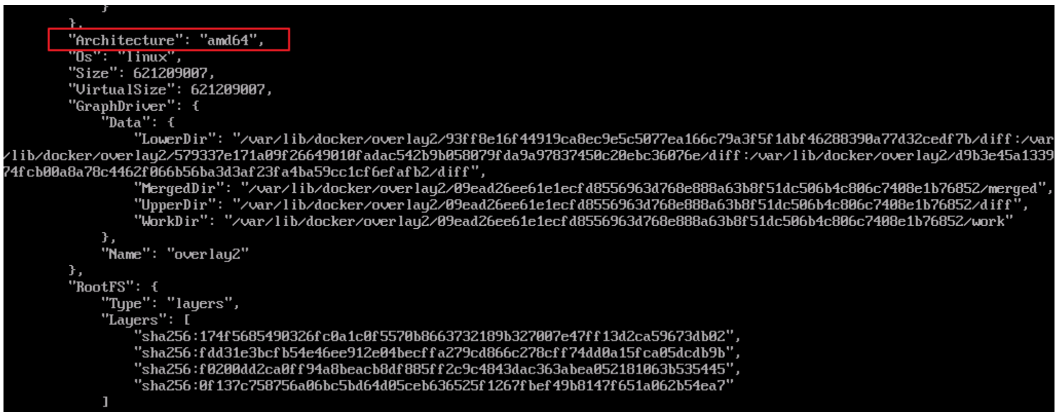
The billing items of SWR include storage space and traffic. Currently, it is free of charge.

Does SWR Support Querying the CPU Architecture (x86 or Arm) of an Image?

- For a public image, you can log in to the SWR console, go to the image center, search for the target image, and view its details, including the architectures supported by the image.
- For a private image, you can Run **docker inspect [*Image name*:*Version name*]** to query the image architecture.

Example: ***docker inspect openjdk:7***.

Figure 9-1 Example



```
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 621209007,
        "VirtualSize": 621209007,
        "GraphDriver": {
            "Data": {
                "LowerDir": "/var/lib/docker/overlay2/93ff8e16f44919ca8ec9e5c5077ea166c79a3f5f1dbf46288390a77d32cedf7b/diff:/var
/lib/docker/overlay2/579337e171a09f26649010fadac542b9b058079fda9a97837450c20ebc36076e/diff:/var/lib/docker/overlay2/d9b3e45a1339
74fcb00a8a78c4462f066b56ba3d3af23fa4ba59cc1cf6efafb2/diff",
                "MergedDir": "/var/lib/docker/overlay2/09ead26ee61e1ecfd8556963d768e888a63b8f51dc506b4c806c7408e1b76852/merged",
                "UpperDir": "/var/lib/docker/overlay2/09ead26ee61e1ecfd8556963d768e888a63b8f51dc506b4c806c7408e1b76852/diff",
                "WorkDir": "/var/lib/docker/overlay2/09ead26ee61e1ecfd8556963d768e888a63b8f51dc506b4c806c7408e1b76852/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:174f5685490326fc0a1c0f5570b8663732189b327007e47ff13d2ca59673db02",
                "sha256:fdd31e3bcfb54e46ee912e04becffa279cd866c278cff74dd0a15fca05dcdb9b",
                "sha256:f0200dd2ca0ff94a8beacb8df885ff2c9c4843dac363abea052181063b535445",
                "sha256:0f137c758756a06bc5bd64d05ceb636525f1267fbef49b8147f651a062b54ea7"
            ]
```

9.1.3 How Do I Create a Container Image?

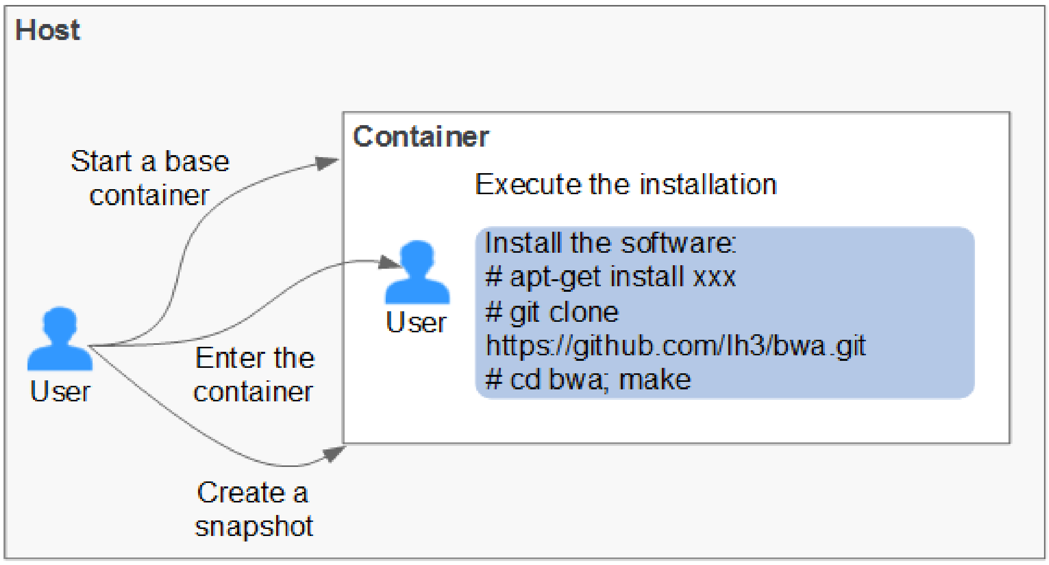
The following two approaches are for you to consider. Approach 1 is for images that will only be updated occasionally whereas approach 2 is for images that will be frequently updated.

- Approach 1: creating a snapshot. This approach involves three key steps: (1) Start a base container by running a base image (for example, Ubuntu image); (2) install the container engine software inside the base container; (3) create a snapshot of the container.
- Approach 2: creating a Dockerfile to build an image. This approach involves two key steps: (1) Write software installation instructions into a Dockerfile; (2) run the **docker build** command to build an image from the Dockerfile.

Approach 1: Creating a Snapshot

This approach is suitable for images that will only be updated occasionally.

Figure 9-2 Creating a snapshot



Procedure:

1. Install the container engine software on a host.
2. Start an empty base container in the interactive mode.

   For example, start a CentOS container in the interactive mode.

   docker run -it centos
3. Run the following commands to install the target software:

   yum install XXX

   git clone https://github.com/lh3/bwa.git

   cd bwa;make

   NOTE

   Install Git in advance and check whether an SSH key is set on the local host.
4. Run the **exit** command to exit the container.
5. Create a snapshot.

   docker commit -m "xx" -a "test" container-id test/image:tag

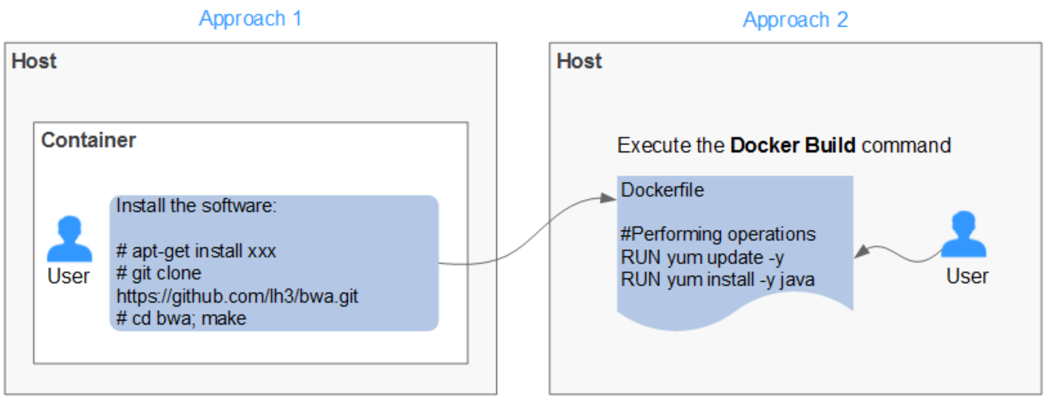
   – **-a**: indicates the author of the base image.
   – **container-id**: indicates the ID of the container you have started in step 2. You can run the **docker ps -a** command to query the container ID.
   – **-m**: indicates the commit message.
   – **test/image:tag**: indicates the repository name/image name:tag name.
6. Run the **docker images** command to list the built container image.

Approach 2: Creating a Dockerfile to Build an Image

This approach is suitable for images that will be frequently updated. In Approach 1, you create a snapshot of the whole container. This could be demanding if you need to frequently update your images. In this case, Approach 2 is put forward to automate the image build process.

The idea behind **Approach 2** is to write the process of **Approach 1** into a Dockerfile and then run the **docker build -t test/image:tag.** command to automatically build an image from the Dockerfile. In the preceding command, **.** indicates the path to the Dockerfile.

Figure 9-3 Creating a Dockerfile to build an image



```
Install the software:

# apt-get install xxx
# git clone
https://github.com/lh3/bwa.git
# cd bwa; make
```

```
#Performing operations
RUN yum update -y
RUN yum install -y java
```

Example Dockerfile:

NOTE

If an external network is required, ensure that network connectivity is available.

```
#Version 1.0.1
FROM centos:latest

# Setting the root user as the executor of subsequent commands
USER root

# Performing operations
RUN yum update -y
RUN yum install -y java

# Using && to concatenate commands
RUN touch test.txt && echo "abc" >>abc.txt

# Setting an externally exposed port
EXPOSE 80 8080 1038

# Adding a network file
ADD https://www.baidu.com/img/bd_logo1.png /opt/

# Setting an environment variable
ENV WEBAPP_PORT=9090

# Setting a work directory
WORKDIR /opt/

# Setting a start command
ENTRYPOINT ["ls"]

# Setting start parameters
CMD ["-a", "-l"]

# Setting a volume
VOLUME ["/data", "/var/www"]

# Setting the trigger operation for a sub-image
ONBUILD ADD . /app/src
ONBUILD RUN echo "on build excuted" >> onbuild.txt
```

Basic Syntax of Dockerfile

- FROM:

  It is used to specify the parent image (base image) from which you are building a new image. Except annotations, a Dockerfile must start with a FROM instruction. Subsequent instructions run in this parent image environment until the next FROM instruction appears. You can create multiple images in the same Dockerfile by adding multiple FROM instructions.

- MAINTAINER:

  It is used to specify the information about the author who creates an image, including the username and email address. This parameter is optional.

- RUN:

  It is used to modify an image. Generally, RUN commands are executed to install libraries, and install and configure programs. After a RUN command is executed, an image layer will be created on the current image. The next command will be executed on the new image. The RUN statement can be in one of the following formats:

  – **RUN yum update**: Command that is executed in the **/bin/sh** directory.
  – **RUN ["yum", "update"]**: Directly invoke **exec**.
  – **RUN yum update && yum install nginx**: Use **&&** to connect multiple commands to a RUN statement.

- EXPOSE:

  It is used to specify one or more network ports that will be exposed on a container. If there are multiple ports, separate them by spaces.

  When running a container, you can set **-P** (uppercase) to map the ports specified in EXPOSE to random ports on a host. Other containers or hosts can communicate with the container through the ports on the host.

  You can also use **-p** (lowercase) to expose the ports that are not listed in EXPOSE.

- ADD:

  It is used to add a file to a new image. The file can be a host file, a network file, or a folder.

  – First parameter: source file (folder)
    ▪ If a relative path is used, this path must correspond to the directory where the Dockerfile is located.
    ▪ If a URL is used, the file needs to be downloaded first and then added to the image.
  – Second parameter: target path
    ▪ If the source file is in the .zip or .tar file, the container engine decompresses the file and then adds it to the specified location of the image.
    ▪ If the source file is a compressed network file specified by a URL, the file will not be decompressed.

- VOLUME:

It is used to create a mount point for a specified path (file or folder) in the image. Multiple containers can share data through the same mount point. Even if one of the containers is stopped, the mount point can still be accessed.

- WORKDIR:

  It is used to specify a new work directory for the next command. The directory can be an absolute or a relative directory. WORKDIR can be specified multiple times as required. When a container is started, the directory specified by the last WORKDIR command is used as the current work directory of the container.

- ENV:

  It is used to set an environment variable for running the container. When running the container, you can set **-e** to modify the environment variable or add other environment variables.

  Example:

  docker run -e WEBAPP_PORT=8000 -e WEBAPP_HOST=www.example.com ...

- CMD:

  It is used to specify the default command for starting a container.

- ENTRYPOINT:

  It is used to specify the default command for starting a container. Difference: For ENTRYPOINT, parameters added to the image during container running will be spliced. For CMD, these parameters will be overwritten.

  - If the Dockerfile specifies that the default command for starting a container is **ls -l**, the default command **ls -l** will be run accordingly. For example:

    - **ENTRYPOINT ["ls", "-l"]**: The program and parameter for starting a container are set to be **ls** and **-l** respectively.

    - **docker run centos**: The **docker run centos ls -l** command is run by default for starting a CentOS container.

    - **docker run centos -a**: When the **-a** parameter is added for starting a CentOS container, the **docker run centos ls -l -a** command is run by default.

  - If the Dockerfile specifies that the default command for starting a container is **--entrypoint** but you need to replace the default command, you can add **--entrypoint** parameters to replace the configuration specified in Dockerfile. Example:

    docker run gutianlangyu/test --entrypoint echo "hello world"

- USER:

  It is used to specify the user or UID for running the container, and running the RUN, CMD, or ENTRYPOINT command.

- ONBUILD:

  Trigger command. During image build, the image builder of the container engine saves all commands specified by the ONBUILD command to the image metadata. These commands will not be executed in the process of building the current image. These commands will be executed only when a new image uses the FROM instruction to specify the parent image as the current image.

Using the FROM instruction to build a child image based on the parent image created by the Dockerfile:

ONBUILD ADD. /app/src: The **ADD. /app/src** command is automatically executed.

9.1.4 How Do I Create an Image Package?

Run the **docker save** command to compress the container image into a .tar or .tar.gz package. The command format is as follows:

docker save [OPTIONS] IMAGE [IMAGE...]

[OPTIONS] can be set to **--output** or **-o**, indicating that the image is exported to a file.

Example:

```
$ docker save nginx:latest > nginx.tar
$ ls -sh nginx.tar
108M nginx.tar

$ docker save php:5-apache > php.tar.gz
$ ls -sh php.tar.gz
372M php.tar.gz

$ docker save --output nginx.tar nginx
$ ls -sh nginx.tar
108M nginx.tar

$ docker save -o nginx-all.tar nginx
$ docker save -o nginx-latest.tar nginx:latest
```

9.1.5 Are There Quotas for SWR Resources?

No quotas are imposed on SWR images. You can push as many images as you need.

Quotas are imposed on the number of organizations a user can create, as shown in **Table 9-1**.

Table 9-1 SWR resource quotas

Resource Type	Quota
Organization	5

9.1.6 Why Does Organization Creation Fail?

Symptom: The creation of an organization fails, and a message is displayed indicating that the organization already exists. However, the organization is not found on the **Organizations** page.

Solution: Change the organization name to one which is globally unique in the Region.

If a message is displayed indicating that the organization already exists, the organization name may have been used by another user. Use another organization name.

9.2 Image Management FAQs

9.2.1 Image Push and Pull

How Do I Push an Image to SWR by Calling APIs?

Currently, SWR does not provide APIs for image push. You can push images using the **docker push** command on a client or using the SWR console.

How Do I Pull an Image from SWR by Calling APIs?

Currently, SWR does not provide APIs for image pull. You can pull images using the **docker pull** command on a client.

Can I Push Arm-based Container Images to SWR?

SWR has no restriction on the kernel architecture of images. There is no difference between pushing an Arm-based image and an x86-based image to SWR.

What Protocol Is Used to Push Images to SWR When I Run the docker push Command?

HTTPS is used.

Will an Image Be Overwritten If I Push an Image That Have the Same Name and Tag with it?

Yes, the original image will be overwritten.

Where Are the Images Pulled by Running the docker pull Command Stored?

Images pulled by running the **docker pull** command are stored on your local hosts. You can run the **docker save** command to save images into TAR archive files.

What Is the Maximum Size of an SWR Layer?

If you use the container engine client to push images to SWR, each image layer cannot exceed 10 GB.

Can SWR Be Accessed over Private Networks? Will I Be Charged for Pushing and Pulling Images over Private Networks?

If your machine and the image repository are in the same region, you can access the image repository through private networks. No additional fees are charged for private network access because you have paid for your servers and EIPs.

If your machine and the image repository are in different regions, the node must have access to public networks to pull images from the image repository.

9.2.2 How Many Tenants Can I Share an SWR Private Image with?

500

9.2.3 What Are the Differences Between Long-Term Valid Login Commands and Temporary Login Commands?

- Temporary login commands will expire after 24 hours.
- Long-term valid login commands are permanently valid.

After you obtain a long-term valid login command, your temporary login commands will still be valid as long as they are in their validity periods.

The long-term valid and temporary login commands can be used by multiple users to log in to the system at the same time.

9.3 Troubleshooting

9.3.1 Why Does the Login Command Fail to Be Executed?

Possible causes are as follows:

1. The container engine is not properly installed, in which case the following error is reported:

   docker: command not found

   Solution: Reinstall the container engine.

   – It is advised to install container engine 1.11.2 or later because earlier versions do not support image push to SWR.

   – If the container engine client is in a private network, bind an elastic IP address (EIP) to the client. This EIP will allow the client to download installation packages from the website.

2. The temporary login command has expired, or the regional project name, access key (AK), or login key in the command is incorrect, in which case the following error is reported:

   unauthorized: authentication required

   Solution: Log in to the SWR console. In the navigation pane on the left, choose **My Images**. On the page displayed, click **Upload Through Client**. Then you can find the information on how to obtain a login command.

   a. To obtain a temporary login command, click **Generate a temporary login command** and then click ❐ to copy the command.

   b. To obtain a long-term valid login command, click **learn how to obtain a login command that has long-term validity** and follow the instructions.

3. The image repository address in the login command is incorrect, in which case the following error is reported:

Error llgging in to v2 endpoint, trying next endpoint: Get https://{{endpoint}}/v2/: dial tcp: lookup {{endpoint}} on xxx.xxx.xxx.xxx:53 : no such host

Solutions:

a. Change the image repository address in the login command.

b. Generate a temporary login command. For detailed instructions, see 2.

4. **x509: certificate has expired or is not yet valid**

   The preceding error is reported when the AK/SK in the login command with long-term validity is deleted. In this case, use a valid AK/SK to generate a login command.

5. **x509: certificate signed by unknown authority**

   Possible Causes:

   The container engine client communicates with SWR through HTTPS. The client verifies the server certificate. If the server certificate is not issued by an authoritative organization, the following error message is displayed: "x509: certificate signed by unknown authority"

   Solutions:

   If you trust the server and skip certificate authentication, manually configure Docker startup parameters as follows:

   – CentOS:

     Modify the **/etc/docker/daemon.json** file. If the file does not exist, manually create it. Add the following content to the file:

```
{
  "insecure-registries": ["{Image repository address}"]
}
```

   – Ubuntu:

     Modify the **/etc/default/docker** file and add the following content to **DOCKER_OPTS**:

```
DOCKER_OPTS="--insecure-registry {image repository address}"
```

   – EulerOS:

     Modify the **/etc/sysconfig/docker** file and add the following content to **INSECURE_REGISTRY**:

```
INSECURE_REGISTRY='--insecure-registry {image repository address}'
```

   NOTE

   The image repository address can be a domain name or an IP address.

   - To obtain the image repository address in domain name format, obtain a temporary login command by referring to 2. The domain name at the end of the command is the image repository address.
   - To obtain the image repository address in IP address format, ping the image repository address in the domain name format.

   After the configuration, run the **systemctl restart docker** command to restart the container engine.

9.3.2 Why Does an Image Fail to Be Uploaded Through a Container Engine Client?

denied: you do not have the permission

Problem: When you push an image to SWR through your container engine client, the operation fails with the following information returned.

denied: you do not have the permission

Possible Causes:

- The organization name you specified has already been used by another user or the maximum number of organizations that you are allowed to create has been reached.
- The **docker login** command you used to log in to SWR is generated using the AK and SK of an IAM user who does not have the permission of the target organization.

Solutions:

- If the organization name has been registered by another user, create an organization and then upload the image.
- If the number of SWR organizations reaches the quota (5 per user), upload the image to an existing organization.
- If the IAM user does not have the permission of the target organization, log in using the cloud account and grant the permission to the IAM user. Then, try again.

Message "tag does not exist: xxxxxx" or "An image does not exist locally with the tag: xxxxxx" Displayed

Problem: When you push an image to SWR through your container engine client, the operation fails with the following information returned.

tag does not exist: xxxxxx

Or

An image does not exist locally with the tag: xxxxxx

Possible cause: The image or image tag to be pushed does not exist.

Solution: Run the **docker images** command to view all the local images. Check the target image name and tag, and push the image again.

name invalid: 'repository' is invalid

Problem: When you push an image to SWR through your container engine client, the operation fails with the following information returned.

name invalid: 'repository' is invalid

Possible cause: The organization name or image name does not comply with the naming rules.

Solution: The regular expressions of the organization (namespace) name and image (repository) name are as follows:

namespace: The value contains a maximum of 64 characters and must meet regular expression **^([a-z]+(?:(?:(?:_|__|[-]*)[a-z0-9]+)+)?)$**.

repository: The value contains a maximum of 128 characters and must meet regular expression **^([a-z0-9]+(?:(?:(?:_|__|[-]*)[a-z0-9]+)+)?)$**.

Specify a valid organization name or image name, and push the image again.

9.3.3 Why Does an Image Fail to Be Uploaded Through SWR Console?

SWR has strict requirements on image name and address format. Invalid image names or addresses could lead to upload failures.

Invalid Image Format

Problem: When you upload an image to SWR through the SWR console, an error message is displayed, indicating that the image format is invalid.

Possible cause: Invalid image address leads to upload failure.

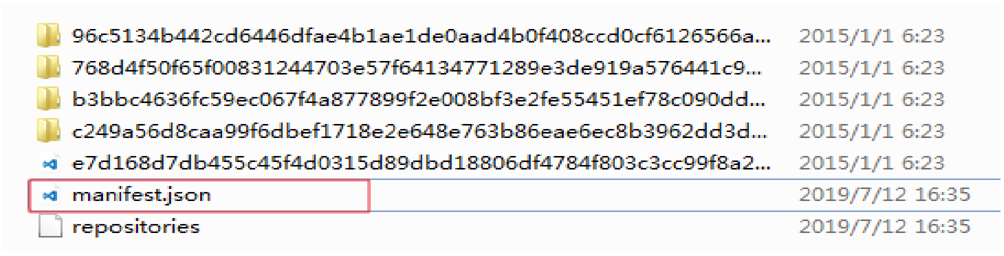
The image tag, which is at the end of an image address, can be omitted. When it is omitted, the latest version of the image will be pushed. Other parts of the image address cannot be omitted. Ensure that they are all correctly configured.

Example: registry.example.com/repo_namespace/repo_name:tag

- **registry.example.com** is an example address of the SWR image repository. Replace it with the actual address.
- **repo_namespace**: organization name. It contains a maximum of 64 characters and must meet regular expression **^([a-z]+(?:(?:(?:_|__|[-]*)[a-z0-9]+)+)?)$**.
- **repo_name:tag**: image name and tag. The image name contains a maximum of 128 characters and must meet regular expression **^([a-z0-9]+(?:(?:(?:_|__|[-]*)[a-z0-9]+)+)?)$**.

To view the image address, decompress the image. Open the **manifest.json** file, and check the value of **RepoTags**.

Figure 9-4 Files in the decompressed package



```
96c5134b442cd6446dfae4b1ae1de0aad4b0f408ccd0cf6126566a...   2015/1/1 6:23
768d4f50f65f00831244703e57f64134771289e3de919a576441c9...   2015/1/1 6:23
b3bbc4636fc59ec067f4a877899f2e008bf3e2fe55451ef78c090dd...   2015/1/1 6:23
c249a56d8caa99f6dbef1718e2e648e763b86eae6ec8b3962dd3d...    2015/1/1 6:23
e7d168d7db455c45f4d0315d89dbd18806df4784f803c3cc99f8a2...   2015/1/1 6:23
manifest.json                                               2019/7/12 16:35
repositories                                                2019/7/12 16:35
```

Solution: Tag the image again according to the naming rules. Run the **docker save** command, and upload the image on the SWR console.

Stuck at the Upload Page Until It Times Out

Problem: When you upload an image to SWR through the SWR console, the upload progress is stuck and the upload task times out at the end.

Possible cause: Invalid image name leads to upload failures.

Solution: Modify the image name according to the naming rules, and try uploading the image again.

NOTICE

It is the image name in the **repositories** and **manifest.json** files that should be checked and modified rather than the name of the image file you select and upload on the SWR console.

9.3.4 Why Does the docker pull Command Fail to Be Executed?

x509: certificate sigined by unknown authority

Problem: When you run the **docker pull** command to pull an image from SWR, error message "x509: certificate signed by unknown certificates" is displayed.

Possible Causes:

- A container engine client and SWR communicate with each other using HTTPS. When the client verifies the server certificate and finds that the root certificate installed on the client is incomplete, the error message "x509: certificate signed by unknown certificates" is displayed.
- A proxy is configured on the container engine client.

Solution:

- If you trust the server and skip certificate authentication, manually configure the startup parameters for the container engine using either of the following methods (use the actual image repository address):
  - Add the following configuration to the **/etc/docker/daemon.json** file. If the file does not exist, manually create it. Ensure that two-space indents are used in the configuration.
    ```
    {
      "insecure-registries":["Image repository address"]
    }
    ```
  - /etc/sysconfig/docker:
    ```
    INSECURE_REGISTRY='--insecure-registry=Image repository address'
    ```

  After configuration, run the **systemctl restart docker** or **service docker start** command to restart the container engine.
- Run the **docker info** command to check whether the proxy is correctly configured. If not, modify the configuration.

Error: remote trust data does not exist

Problem: When you run the **docker pull** command to pull an image from SWR, message "Error: remote trust data does not exist" is displayed.

Possible cause: The image signature verification is enabled on the client. However, the image to be pulled does not contain a signature layer.

Solution: Check whether the environment variable **DOCKER_CONTENT_TRUST** is set to **1**. If yes, delete **DOCKER_CONTENT_TRUST=1** from the **/etc/profile** file and run the **source /etc/profile** command to make the modification take effect.

9.4 Other FAQs

9.4.1 Why Does a CCE Workload Cannot Pull an Image from SWR and the Message Indicating "Not Logged In" Is Displayed?

If a CCE workload cannot pull an SWR image and the message indicating "Not logged in" is displayed, check whether the YAML file of the workload contains the **imagePullSecrets** field and whether the value of **name** is fixed to **default-secret**.

Example:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

A Change History

Release Date	Description
2020-11-05	This issue is the first official release.