SoftWare Repository for Container

Best Practices

Issue	01
Date	2022-12-05

HUAWEI TECHNOLOGIES CO., LTD.

Security Declaration

Vulnerability

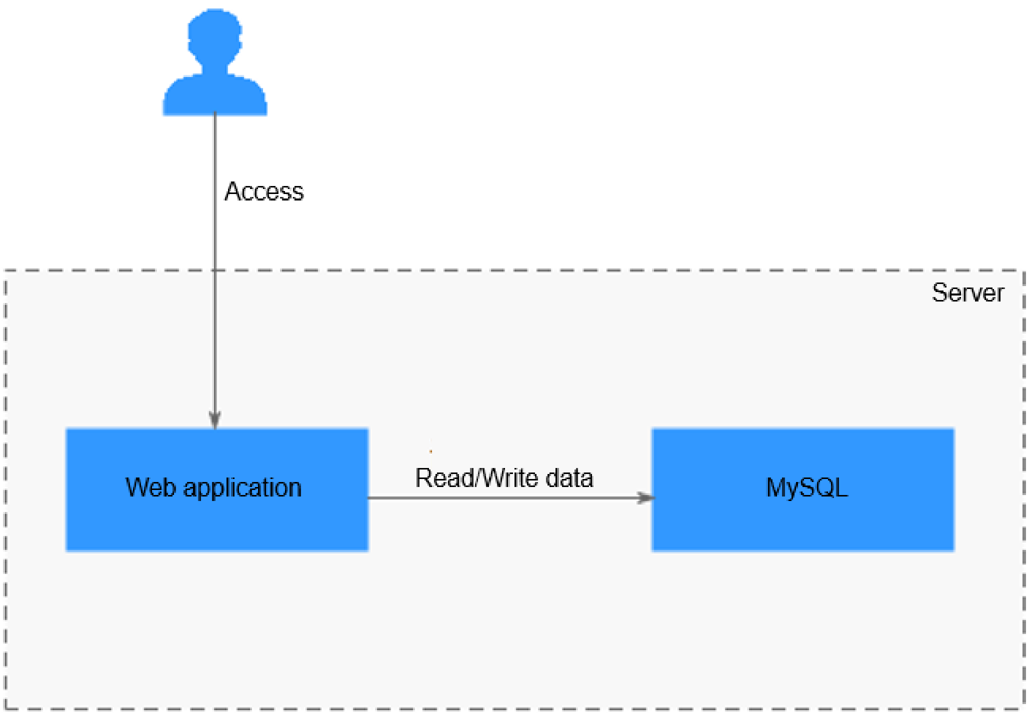
Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process*. For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page: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

Contents

1 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.

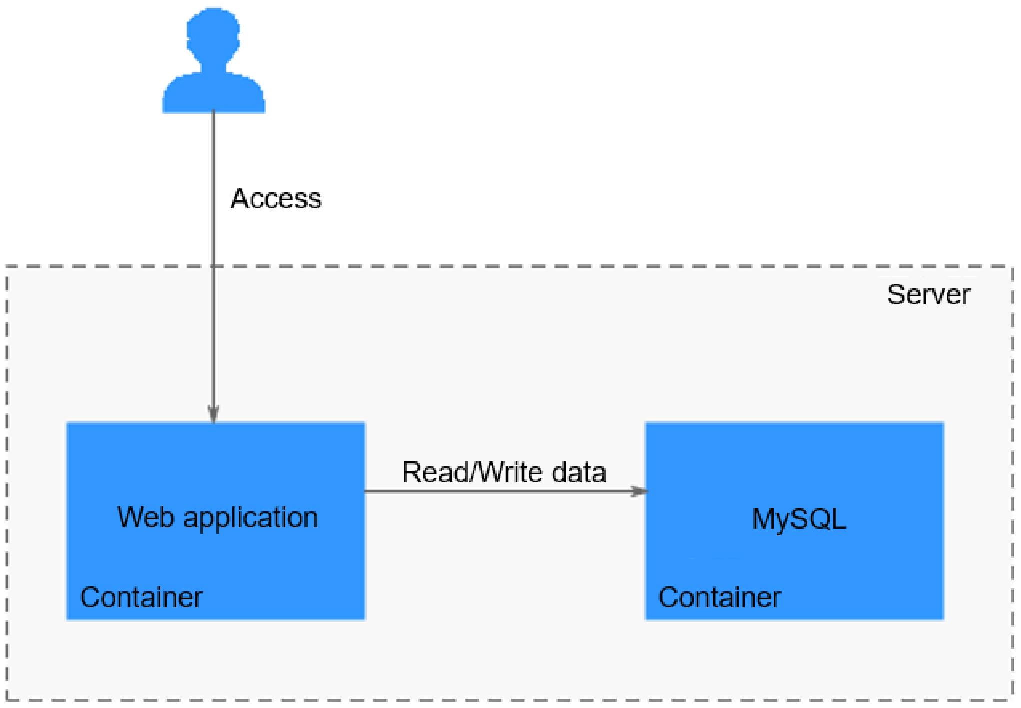
Therefore, usually, an application is split into microservices before containerization. You will benefit a lot from the microservice architecture:

- **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 to run applications. 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 Docker 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 directories such as **.git** are unnecessary.

Example:

```
.git/
```

2 Configuring Access Network

2.1 Overview

When your container engine client is a CCE or ECS node, how can you pull or push an image at an optimal speed? This section provides you with three solutions. You can select one as needed.

2.2 Private Network Access

If your container engine client is a CCE or ECS node and is in the same region as the image repository, you can push or pull an image using a private network.

Private network access does not need any configurations.

CAUTION

If you access OBS through fixed VPC endpoints, you need to configure policies to allow the access to OBS bucket clusters that store SWR container images. Otherwise, the images may fail to be pulled. You can submit a **service ticket** to obtain information about SWR's OBS bucket clusters in different regions.

2.3 Public Network Access

Scenarios

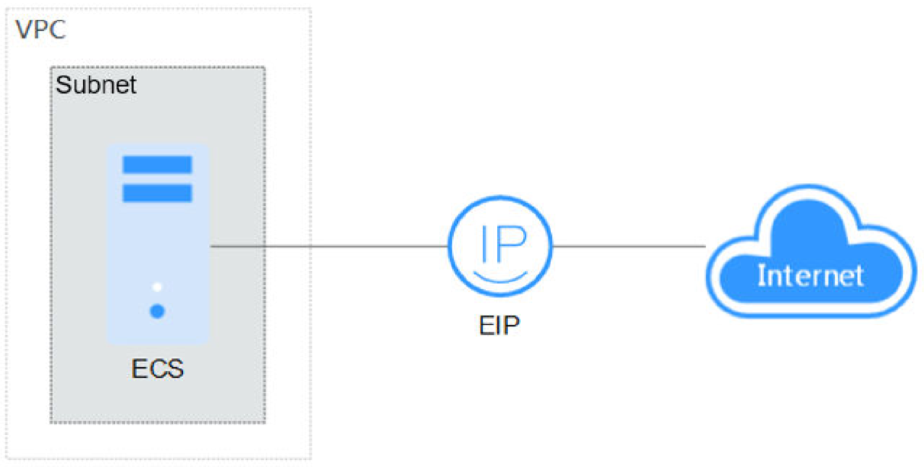
If your container engine client is a CCE or ECS node and is in the **different** region from the image repository, you can push or pull an image using a public network and the client needs to be bound to an EIP. For public network access, there are two scenarios.

- **Public Network Access for Single ECS**
- **Public Network Access for Multiple ECSs**

Public Network Access for Single ECS

If an ECS needs to access a public network, you can bind it to an EIP. Huawei Cloud provides multiple billing modes (such as pay-per-use and pay-per-traffic). You can select one as required and flexibly unbind an unnecessary EIP.

Figure 2-1 Network topology



Step 1 Log in to the management console.

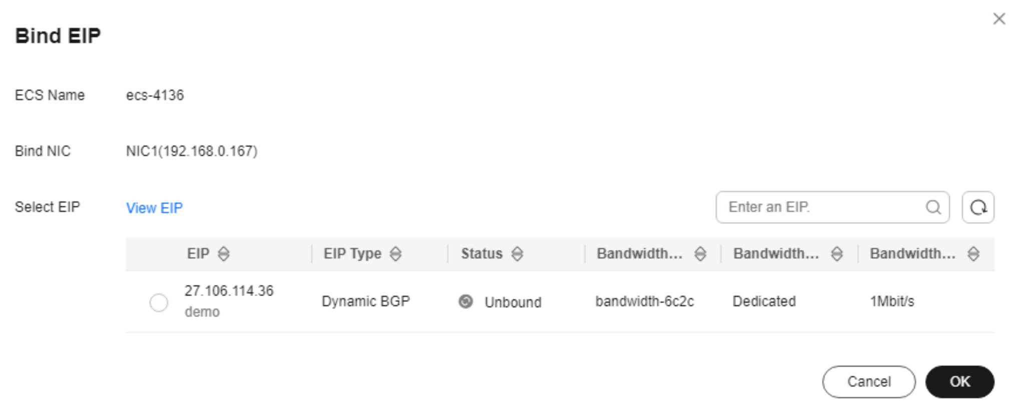
Step 2 Click at the upper left corner and select the desired region and project.

Step 3 Click and choose **Computing** > **Elastic Cloud Server**.

Step 4 In the ECS list, select the ECS to which an EIP is to be bound, and choose **More** > **Manage Network** > **Bind EIP** in the **Operation** column.

Step 5 Select an EIP and click **OK**.

Figure 2-2 Binding an EIP


Bind EIP

ECS Name ecs-4136

Bind NIC NIC1(192.168.0.167)

Select EIP View EIP

EIP	EIP Type	Status	Bandwidth...	Bandwidth...	Bandwidth...
27.106.114.36 demo	Dynamic BGP	Unbound	bandwidth-6c2c	Dedicated	1Mbit/s

Step 6 After the ECS is bound to the EIP, you can view the bound EIP in the ECS list.

NOTE

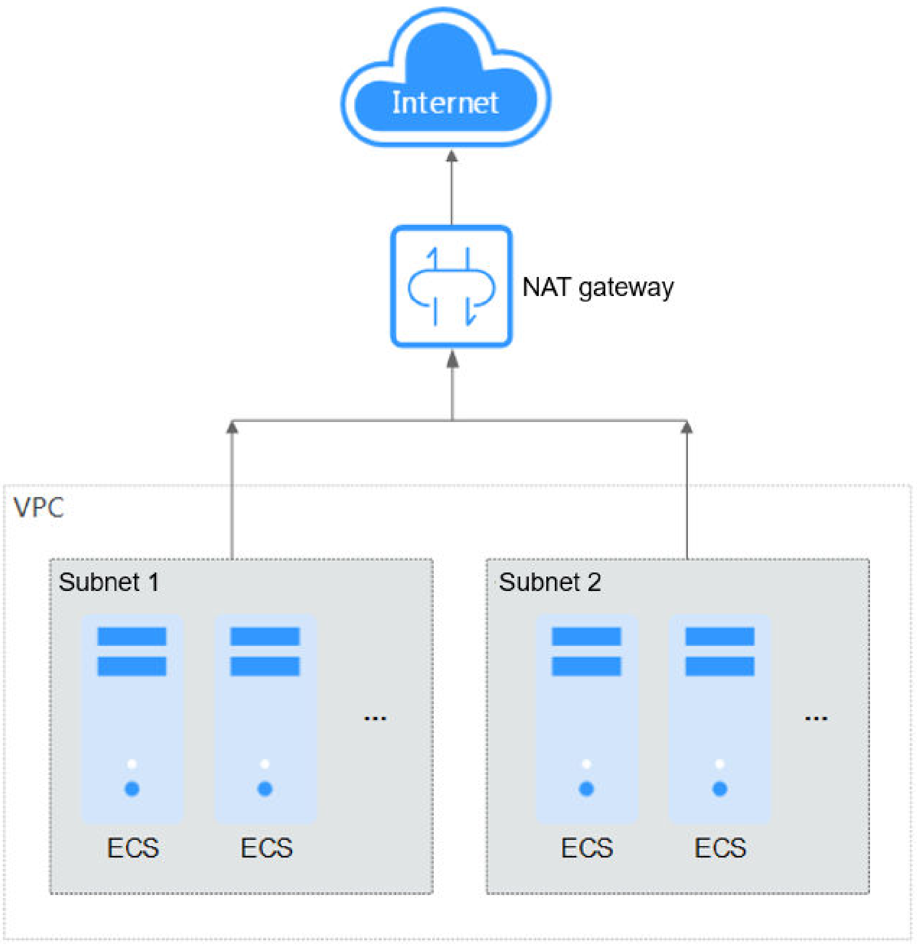
If no EIP is available in the current region, the EIP list is empty. In this case, purchase an EIP and bind it again.

----End

Public Network Access for Multiple ECSs

If all ECSs in your VPC need to access a public network, you can use NAT Gateway and configure SNAT rules by subnet to easily build a public network egress for the VPC. If no SNAT rule is configured, external users cannot directly access the public network IP address of the NAT gateway through a public network, which makes ECS more secure compared with public network access through an EIP.

Figure 2-3 Network topology



Step 1 Bind the ECS to an EIP. For details, see **Public Network Access for Single ECS**.

Step 2 Create a NAT gateway. For details, see **Buying a Public NAT Gateway**.

a. Log in to the management console.

b. Click at the upper left corner and select the desired region and project.

c. Click ☰ at the upper left corner, and choose **Networking** > **NAT Gateway**.

d. On the displayed page, click **Buy Public NAT Gateway**.

e. Configure parameters as prompted.

Step 3 Configure SNAT rules and bind EIPs to subnets. For details, see **Adding an SNAT Rule**.

a. Log in to the management console.

b. Click at the upper left corner and select the desired region and project.

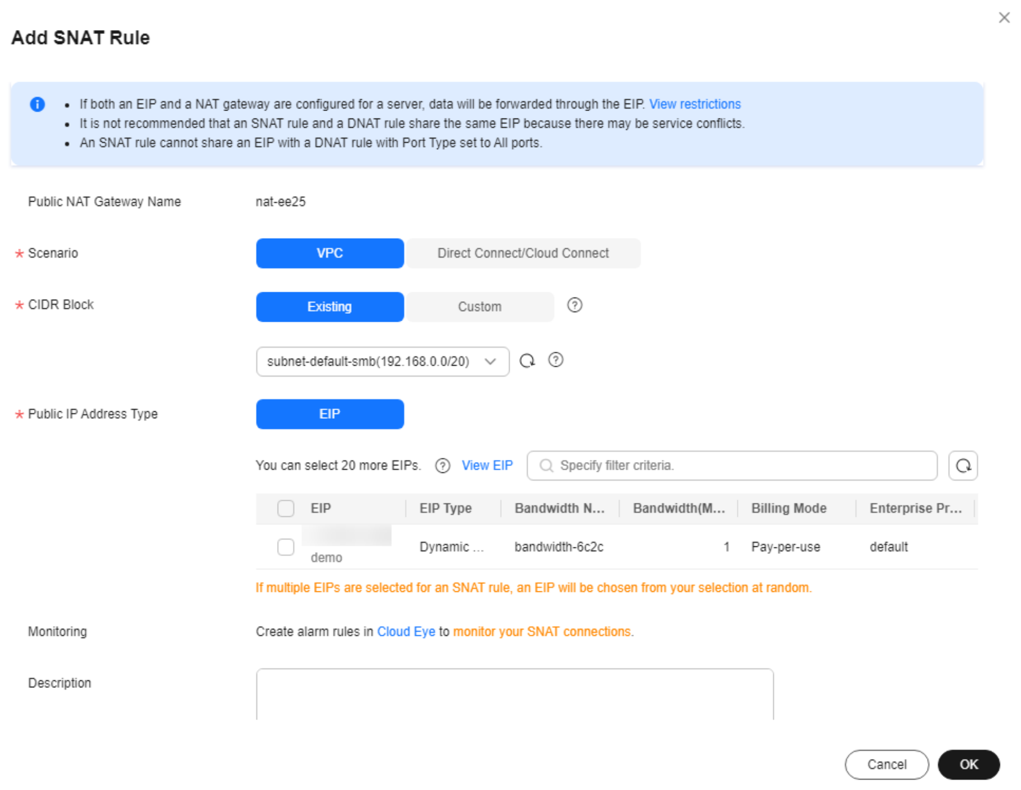
c. Click ☰ at the upper left corner, and choose **Networking** > **NAT Gateway**.

d. On the displayed page, click the name of the NAT gateway that you want to add an SNAT rule for.

e. On the **SNAT Rules** tab, click **Add SNAT Rule**.

f. Configure parameters as prompted.

Figure 2-4 Adding an SNAT rule



Add SNAT Rule

- If both an EIP and a NAT gateway are configured for a server, data will be forwarded through the EIP. View restrictions
- It is not recommended that an SNAT rule and a DNAT rule share the same EIP because there may be service conflicts.
- An SNAT rule cannot share an EIP with a DNAT rule with Port Type set to All ports.

Public NAT Gateway Name nat-ee25

* Scenario: VPC | Direct Connect/Cloud Connect

* CIDR Block: Existing | Custom

subnet-default-smb(192.168.0.0/20)

* Public IP Address Type: EIP

You can select 20 more EIPs. View EIP

	EIP	EIP Type	Bandwidth N...	Bandwidth(M...	Billing Mode	Enterprise Pr...
☐	[redacted] demo	Dynamic ...	bandwidth-6c2c	1	Pay-per-use	default

If multiple EIPs are selected for an SNAT rule, an EIP will be chosen from your selection at random.

Monitoring: Create alarm rules in Cloud Eye to monitor your SNAT connections.

Description

----End

CAUTION

If you access OBS by configuring the local **/etc/hosts** file, add the domain names of the OBS bucket clusters that store container images to this file. Otherwise, the images may fail to be pulled. You can submit a **service ticket** to obtain information about SWR's OBS bucket clusters in different regions.

2.4 VPN/Direct Connect Access

Scenario

If your local data center or private network cannot access SWR through a public network, you can use Direct Connect or VPN to connect to Huawei Cloud VPC and use a VPC endpoint to access SWR.

This solution applies only to pushing images through SWR. To pull images, you also need to **configure a VPC endpoint for accessing OBS using the OBS private address**.

Procedure

Step 1 **Create a VPC.** For details, see **Creating a VPC**.

Step 2 **Create a Direct Connect connection or VPN so that the data center can connect to the VPC through Direct Connect or VPN.**

Step 3 **Buy a VPC endpoint.**

a. Log in to the management console.

b. Click at the upper left corner and select the desired region and project.

c. In the **Service List** at the upper left corner, choose **Networking** > **VPC Endpoint**.

d. On the displayed page, click **Buy VPC Endpoint**.

e. Configure the parameters as prompted.

f. Click **Next**.

g. Confirm the order details and click **Submit**.

Step 4 **Obtain the private network IP address and domain name for accessing the VPC.**

NOTE

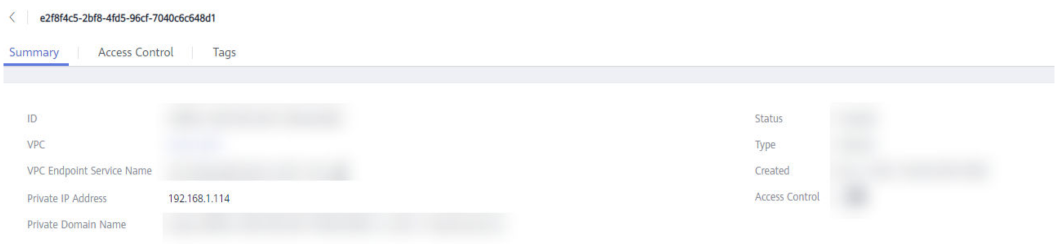
By default, the domain name resolution rule is automatically added to Huawei Cloud VPCs. You only need to configure hosts for non-Huawei Cloud endpoints.

a. Go to the endpoint list.

b. Locate the purchased endpoint and click the endpoint ID to go to the details page.

c. On the page displayed, you can view the IP address and private network domain name of the endpoint.

Figure 2-5 Endpoint details page



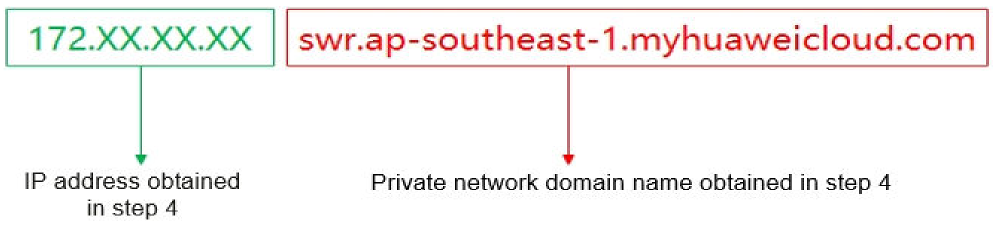
e2f8f4c5-2bf8-4fd5-96cf-7040c6c648d1

Summary | Access Control | Tags

ID	[redacted]	Status	[redacted]
VPC	[redacted]	Type	[redacted]
VPC Endpoint Service Name	[redacted]	Created	[redacted]
Private IP Address	192.168.1.114	Access Control	[redacted]
Private Domain Name	[redacted]		

Step 5 **Configure hosts of the local data center.** The hosts IP address consists of the IP address and private network domain name of the endpoint. Example:

Figure 2-6 Example hosts


172.XX.XX.XX
swr.ap-southeast-1.myhuaweicloud.com
IP address obtained in step 4
Private network domain name obtained in step 4

CAUTION

In this section, **172.xx.xx.xx swr.ap-southeast-1.myhuaweicloud.com** is only an example. Replace it with the actual IP address and private network domain name.

There are two configuration methods:

Configuring Hosts for Linux

Customizing DNS Hosts

- **Configuring Hosts for Linux:**

1. Run the following command to open the **/etc/hosts** file:
```
sudo vim /etc/hosts
```
2. Add a custom domain name in the format of *xx.xx.xx.xx* **swr.***xx* **-** *xx***.myhuaweicloud.com**.

   NOTE

   xx.xx.xx.xx and **swr.***xx* -*xx***.myhuaweicloud.com** indicate the IP address and domain name obtained in **Step 4**, respectively.
3. Run the following command to restart the network.
```
sudo/etc/init.d/networking restart
```
- **Customizing DNS Hosts:**

1. Obtain the IP address of the VPC endpoint by referring to **4**.
2. Configure DNS forwarding rules on the DNS server in the local data center.

   The method of configuring DNS forwarding rules varies depending onOSs. For details, see the operation guide of the corresponding DNS software.

   This step uses the Linux OS and Bind (common DNS software) as an example.

   a. Edit the **/etc/named.conf** file to add a zone.

   ```
   zone " swr.xx-xx.myhuaweicloud.com " IN {
     type master;
     file " /var/named/swr.xx-xx.myhuaweicloud.com.zone";
   };
   ```

   NOTE

   swr.*xx-xx***.myhuaweicloud.com** indicates the private network domain name obtained in **Step 4**.

   b. Configure forward DNS resolution. Create the **/var/named/swr.***xx-xx***.myhuaweicloud.com.zone** file corresponding to the file in **a**.

   ```
   $TTL     604800
   @ IN      SOA       swr.xx-xx.myhuaweicloud.com. root.localhost. (
                              2          ; Serial
                         604800          ; Refresh
                          86400          ; Retry
                        2419200          ; Expire
                         604800 )        ; Negative Cache TTL
   ;
   @    IN    NS    swr.xx-xx.myhuaweicloud.com.
   swr.xx-xx.myhuaweicloud.com.    IN    A      xx.xx.xx.xx
   ```

   c. Restart the service.

   /sbin/service named restart

   NOTE

   – You can query SWR endpoints in different regions in **Regions and Endpoints**.
   – If no DNS server is available in the local data center, add the endpoint IP address for accessing DNS to the **/etc/resolv.conf** file of the local data center.
   – **swr.***xx-xx***.myhuaweicloud.com** indicates the IP address obtained in **Step 4**.

Step 6 **Run the following command to verify the configuration and check the output.**

ping swr.xx -xx.myhuaweicloud.com

Step 7 Use this domain name (**swr.xx -xx.myhuaweicloud.com**) in the later access to SWR.

----End

3 Migrating Container Images

3.1 Overview

Challenges

Containers are growing in popularity. Many enterprises choose to build their own Kubernetes clusters. However, the O&M workload of on-premises clusters is heavy, and O&M personnel need to configure the management systems and monitoring solutions by themselves. For enterprises, managing a large number of images requires high O&M, labor, and management costs, and the efficiency is low.

SoftWare Repository for Container (SWR) manages container images that function on multiple architectures, such as Linux and Arm. Enterprises can migrate their image repositories to SWR to reduce costs.

This section describes four ways to migrate image repositories to SWR smoothly. You can select one as needed.

Migration Solutions

Table 3-1 Comparison of migration solutions and application scenarios

Solution	Application Scenario	Precautions
Migrating images to SWR using Docker commands	Small quantity of images	● Disk storage leads to the timely deletion of local images and time-cost flushing. ● Docker daemon strictly restricts the number of concurrent pull/push operations, so high-concurrency synchronization cannot be performed. ● Scripts are complex because HTTP APIs are needed to perform the operations that cannot be implemented through Docker CLI.
Migrating images to SWR using image-syncer	A large number of images	● Many-to-many image repository synchronization is supported. ● Docker Registry V2-based image repositories (such as Docker Hub, Quay, and Harbor) can be migrated to SWR. ● Memory- and network-dependent synchronization is fast. ● Flushing the Blob information of synchronized images avoids repetition. ● The number of concurrent synchronization tasks can be adjusted in the configuration file. ● Automatically retrying failed synchronization tasks can resolve most network jitter during image synchronization. ● Docker or other programs are not required.

Solution	Application Scenario	Precautions
Migrating images to SWR using image-migrator	A large number of images	Images are migrated from a Docker registry built on Docker Registry v2 to SWR.
Synchronizing images across clouds from Harbor to SWR	A customer deploys services in multiple clouds and uses Harbor as their image repository.	Only Harbor v1.10.5 and later versions are supported.

3.2 Migrating Images to SWR Using Docker Commands

Scenarios

SWR provides easy-to-use image hosting and efficient distribution services. If small quantity of images need to be migrated, enterprises can use the **docker pull/push** command to migrate images to SWR.

Procedure

Step 1 Pull images from the source repository.

Run the **docker pull** command to pull the images.

Example: **docker pull nginx:latest**

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

```
# docker images
REPOSITORY                     TAG       IMAGE ID       CREATED         SIZE
nginx                        latest    22f2bf2e2b4f   5 hours ago     22.8MB
```

Step 2 Push the images pulled in **Step 1** to SWR.

1. Log in to the VM where the target container is located and log in to SWR. For details, see **Uploading an Image Through a Container Engine Client**.
2. Tag the images.

   docker tag [Image name:Tag name] [Image repository address]/[Organization name]/[Image name:Tag name]

   Example:

   docker tag nginx:v1 swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/nginx:v1
3. Run the following command to push the images to the target image repository.

   docker push [Image repository address]/[Organization name]/[Image name:Tag name]

   Example:

docker push swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/nginx:v1

4. Check whether the following information is returned. If yes, the push is successful.

```
fbce26647e70: Pushed
fb04ab8effa8: Pushed
8f736d52032f: Pushed
009f1d338b57: Pushed
678bbd796838: Pushed
d1279c519351: Pushed
f68ef921efae: Pushed
v1: digest: sha256:0cdfc7910db531bfa7726de4c19ec556bc9190aad9bd3de93787e8bce3385f8d size: 
1780
```

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

----End

3.3 Migrating Images to SWR Using image-syncer

Scenarios

If a small quantity of images need to be migrated, you can use Docker commands. However, for thousands of images and several TBs of image repository data, it takes a long time and even data may be lost. In this case, you can use the open source image migration tool **image-syncer**.

Procedure

Step 1 **Download image-syncer**. Then, decompress and run it.

The following uses image-syncer v1.3.1 as an example.

wget https://github.com/AliyunContainerService/image-syncer/releases/download/v1.3.1/image-syncer-v1.3.1-linux-amd64.tar.gz

tar -zvxf image-syncer-v1.3.1-linux-amd64.tar.gz

Step 2 Create **auth.json**, the authentication information file of the image repositories.

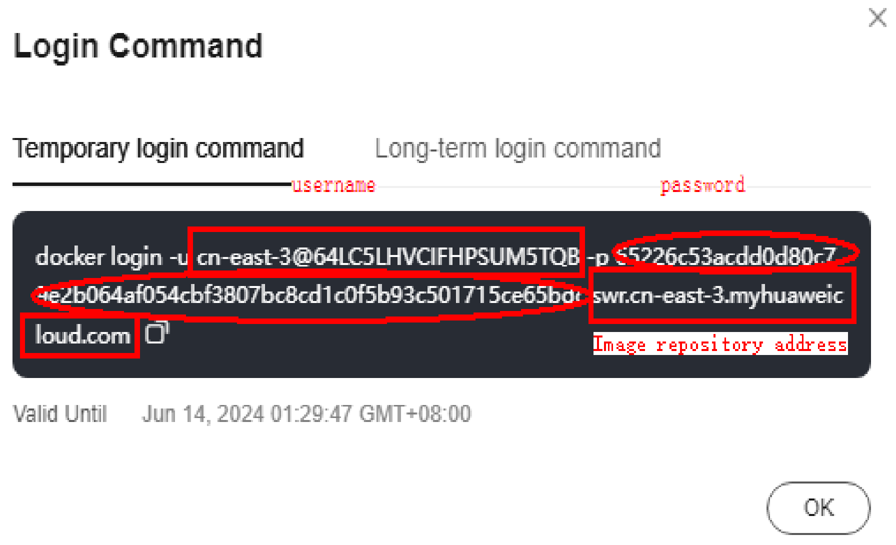
image-syncer can migrate Docker Registry V2-based repositories. Write the authentication information of the source and target repositories. The following is an example:

```
{
   "swr.××××.myhuaweicloud.com": {
      "username": "××××@F1I3Q......",
      "password": "2fd4c869ea0......"
   },
   "swr.××××.myhuaweicloud.com": {
      "username": "××××@4N3FA......",
      "password": "f1c82b57855f9d35......"
   }
```

In the command, **swr.××××.myhuaweicloud.com** indicates the image repository address. You can obtain the username and password from the login command.

Log in to the SWR console, and click **Generate Login Command** in the upper right corner to obtain the login command.

Figure 3-1 Generating a login command



```
docker login -u cn-east-3@64LC5LHVCIFHPSUM5TQB -p 5226c53acdd0d80c74e2b064af054cbf3807bc8cd1c0f5b93c501715ce65bd swr.cn-east-3.myhuaweicloud.com
```

Step 3 Create **images.json**, the image synchronization description file.

In the following example, the source repository address is on the left, and the target repository address is on the right. image-syncer also supports other description modes. For details, see **README.md**.

```
{
"swr.ap-southeast-3.myhuaweicloud.com/org-ss/canary-consumer": "swr.ap-southeast-1.myhuaweicloud.com/dev-container/canary-consumer"
}
```

Step 4 Run the following command to migrate the images to SWR:

./image-syncer --auth=./auth.json --images=./images.json --namespace=dev-container --registry=swr.ap-southeast-1.myhuaweicloud.com --retries=3 --log=./log

Table 3-2 Command parameter description

Parameter	Description
--config	Path of the configuration file. This file needs to be created before you start the synchronization. By default, the configuration file is the **config.json** file in the current directory. (This parameter can be replaced with parameters **--auth** and **--images** which represent authentication information and repository synchronization rules respectively.)
--images	Path of the image rules file. This file needs to be created before you start the synchronization. By default, the rule file is the **images.json** file in the current directory.
--auth	Path of the authentication file. This file needs to be created before you start the synchronization. By default, the authentication file is the **auth.json** file in the current directory.

Parameter	Description
--log	Path of the log file. Logs will be printed to Stderr by default. You need to use **cat** to check logs.
--namespace	default-namespace. default-namespace can also be set by environment variable **DEFAULT_NAMESPACE**. If both the parameter and environment variable are set, the parameter will take precedence. default-namespace will work only if default-registry is not empty.
--proc	Number of goroutines. The default value is **5**. You are advised to use this value and do not change it.
--retries	Number of retries. The default value is **2**. The retries of failed sync tasks will start after all sync tasks are executed once. Retrying sync tasks will resolve most occasional network problems during synchronization.
--registry	default-registry. default-registry can also be set by environment variable **DEFAULT_REGISTRY**. If both the parameter and environment variable are set, the parameter will take precedence. default-registry will work only if default-namespace is not empty.

Step 5 Log in to the target image repository to check the migrated images.

----End

3.4 Migrating Images to SWR using image-migrator

To ensure that container images can be properly pulled after cluster migration and improve container deployment efficiency, you are advised to migrate self-built image repositories to Huawei Cloud SoftWare Repository for Container (SWR).

image-migrator is an image migration tool that can automatically migrate images from a Docker registry built on Docker Registry v2 to SWR.

Preparations

Before the migration, prepare a server with kubectl installed for the connection between the source and destination clusters. The server must have at least 5 GB of local disk space and at least 8 GB of memory so that image-migrator can work properly and can store related data, such as data collected from the source cluster and recommendation data of the destination cluster.

image-migrator can run on Linux (x86 and Arm) and Windows.

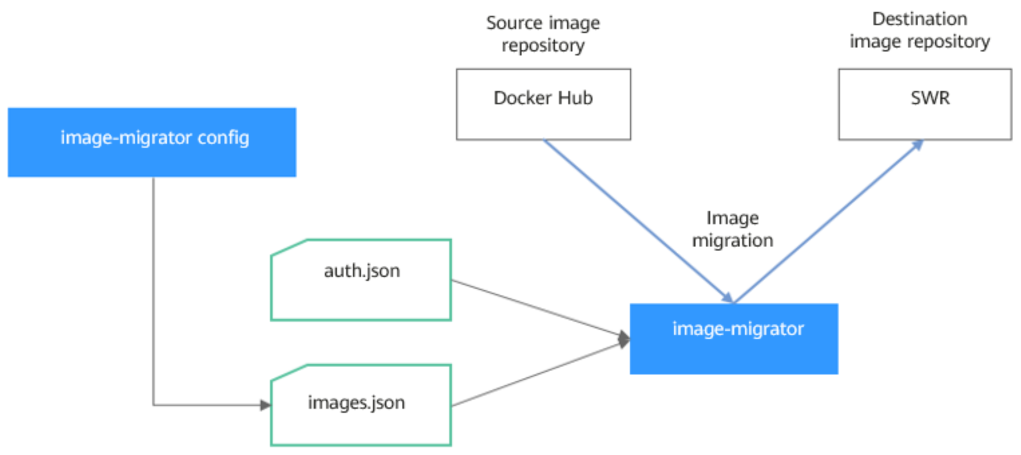
Before using image-migrator in Linux, run the **chmod u+x** *Tool name* command (for example, **chmod u+x image-migrator-linux-amd64**) to grant the execute permission.

Table 3-3 Obtaining the image-migrator package

image-migrator	image-migrator is an image migration tool that can automatically migrate images from a Docker registry built on Docker Registry v2 to SWR, or from a registry on a third-party cloud to SWR.	Linux x86: **https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-linux-amd64** Linux Arm: **https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-linux-arm64** Windows: **https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-windows-amd64.exe**

How image-migrator Works

Figure 3-2 How image-migrator works



When using image-migrator to migrate images to SWR, you need to prepare two files. One is the registry access permission file **auth.json**. The two objects in the file are the accounts and passwords of the source and destination registries. The other is the image list file **images.json**, which consists of multiple image synchronization rules. Each rule contains a source repository (key) and a destination repository (value). Place these two files in the directory where image-migrator is located and run a simple command to migrate the image. The two files are described as follows:

- **auth.json**

  auth.json is the registry access permission file. Each object is the username and password of a registry. Generally, you must have permission to pull images from and access image tags in the source registry and permission to push images to and create repositories in the destination registry. If you access a registry anonymously, you do not need to enter the username and password. Structure of the **auth.json** file:

```
{
    "Source registry address": { },
    "Destination registry address": {
        "username": "xxxxxx",
        "password": "xxxxxx",
        "insecure": true
    }
}
```

To be more specific:

- The *Source registry address* and *Destination registry address* can be in the *registry* or *registry/namespace* format matching that in **images.json**. The matched URL in images uses the corresponding username and password for image synchronization. The *registry/namespace* format is preferred.

  If the destination registry address is in the *registry* format, you can obtain it from the SWR console. On the **Dashboard** page, click **Generate Login Command** in the upper right corner. The domain name at the end of the login command is the SWR registry address, for example, **swr.cn-north-4.myhuaweicloud.com**. Note that the address varies depending on the region. Switch to the corresponding region to obtain the address. If the value is in the *registry/namespace* format, replace *namespace* with the organization name of SWR.
- **username**: (Optional) username. You can set it to a specific value or use a string of the ${env} or $env type to reference an environment variable.
- **password**: (Optional) password. You can set it to a specific value or use a string of the ${env} or $env type to reference an environment variable.
- **insecure**: (Optional) whether *registry* is an HTTP service. If yes, the value of **insecure** is **true**. The default value is **false**.

NOTE

The username of the destination SWR registry is in the following format: *Regional project name*@*AK*. The password is the encrypted login key of the AK and SK. For details, see **Obtaining a Long-Term Valid Login Command**.

Example:

```
{
    "quay.io/coreos": { },
    "swr.cn-north-4.myhuaweicloud.com": {
        "username": "cn-north-4@RVHVMX******",
        "password": "cab4ceab4a1545***************",
        "insecure": true
    }
}
```

- **images.json**

  This file is essentially a list of images to migrate and consists of multiple image synchronization rules. Each rule contains a source repository (key) and a destination repository (value). The specific requirements are as follows:

  a. The largest unit that can be synchronized using one rule is repository. The entire namespace or registry cannot be synchronized using one rule.

  b. The formats of the source and destination repositories are similar to those of the image URL used by the **docker pull/push** command (*registry/namespace/repository:tag*).

  c. Both the source and destination repositories (if the destination registry is not an empty string) contain at least *registry*/*namespace*/*repository*.

d. The source registry field cannot be empty. To synchronize data from a source registry to multiple destination registries, you need to configure multiple rules.

e. The destination repository name can be different from the source repository name. In this case, the synchronization function is similar to docker pull + docker tag + docker push.

f. If the source repository field does not contain tags, all tags of the repository have been synchronized to the destination repository. In this case, the destination repository cannot contain tags.

g. If the source repository field contains tags, only one tag in the source repository has been synchronized to the destination repository. If the destination repository does not contain tags, the source tag is used by default.

h. If the destination repository is an empty string, the source image will be synchronized to the default namespace of the default registry. The repository and tag are the same as those of the source repository. The default registry and namespace can be configured using command line parameters and environment variables.

Example:

```
{
    "quay.io/coreos/etcd:1.0.0": "swr.cn-north-4.myhuaweicloud.com/test/etcd:1.0.0",
    "quay.io/coreos/etcd": "swr.cn-north-4.myhuaweicloud.com/test/etcd",
    "quay.io/coreos/etcd:2.7.3": "swr.cn-north-4.myhuaweicloud.com/test/etcd"
}
```

We provide a config subcommand of the image-migrator tool to automatically obtain the image that is being used by the workload in the cluster. For details, see **Usage of image-migrator config**. After obtaining the **images.json** file, you can modify, add, or delete its content as needed.

How to Use image-migrator

NOTE

image-migrator can run on Linux (x86 and Arm) and Windows. The usage is similar in both environments. This section uses the Linux (x86) environment as an example.

If Linux (Arm) or Windows is used, replace **image-migrator-linux-amd64** in the following command with **image-migrator-linux-arm64** or **image-migrator-windows-amd64.exe**.

Run **./image-migrator-linux-amd64 -h** in the directory where image-migrator is located to learn about its usage.

- **--auth**: specifies the path of **auth.json**. By default, **auth.json** is stored in the directory where image-migrator is located.
- **--images**: specifies the path of **images.json**. By default, **images.json** is stored in the directory where image-migrator is located.
- **--log**: specifies the path for storing logs generated by image-migrator. The default value is **image-migrator.log** in the current directory of image-migrator.
- **--namespace**: specifies the default namespace of the destination repository. That is, if the namespace of the destination repository is not specified in **images.json**, you can specify it when running the migration command.

- **--registry**: specifies the default registry of the destination repository. That is, if the registry of the destination repository is not specified in **images.json**, you can specify it when running the migration command.
- **--retries**: specifies the number of retry times when the migration fails. The default value is **3**.
- **--workers**: specifies the number of concurrent workers for image migration. The default value is **7**.

```
$ ./image-migrator-linux-amd64 -h
A Fast and Flexible docker registry image images tool implement by Go.

Usage:
  image-migrator [flags]

Aliases:
  image-migrator, image-migrator

Flags:
      --auth string        auth file path. This flag need to be pair used with --images. (default "./auth.json")
  -h, --help               help for image-migrator
      --images string      images file path. This flag need to be pair used with --auth (default "./images.json")
      --log string         log file path (default "./image-migrator.log")
      --namespace string   default target namespace when target namespace is not given in the images
config file, can also be set with DEFAULT_NAMESPACE environment value
      --registry string    default target registry url when target registry is not given in the images config file,
can also be set with DEFAULT_REGISTRY environment value
  -r, --retries int        times to retry failed tasks (default 3)
  -w, --workers int        numbers of working goroutines (default 7)

$./image-migrator --workers=5 --auth=./auth.json --images=./images.json --namespace=test \
--registry=swr.cn-north-4.myhuaweicloud.com --retries=2
$ ./image-migrator
Start to generate images tasks, please wait ...
Start to handle images tasks, please wait ...
Images(38) migration finished, 0 images tasks failed, 0 tasks generate failed
```

Example:

./image-migrator --workers=5 --auth=./auth.json --images=./images.json --namespace=test --registry=swr.cn-north-4.myhuaweicloud.com --retries=2

The preceding command is used to migrate images in the **images.json** file to **swr.cn-north-4.myhuaweicloud.com/test**. If the migration fails, you can retry twice. A maximum of five images can be migrated at a time.

Usage of image-migrator config

The config subcommand of image-migrator can be used to obtain images used in cluster applications and generate the **images.json** file in the directory where the tool is located. You can run **./image-migrator-linux-amd64 config -h** to learn how to use the config subcommand.

- **-k, --kubeconfig**: specifies the location of the kubeconfig file of kubectl. The default value is **$HOME/.kube/config**. The kubeconfig file is used to configure access to the Kubernetes cluster. The kubeconfig file contains the authentication credentials and endpoints (access addresses) required for accessing and registering the Kubernetes cluster. For details, see the **Kubernetes documentation**.
- **-n, --namespaces**: specifies the namespace of the image to be obtained. Multiple namespaces are separated by commas (,), for example, ns1,ns2,ns3. The default value is **""**, indicating that images of all namespaces are obtained.

- **-t, --repo**: specifies the destination repository address (*registry*/*namespace*).

```
$ ./image-migrator-linux-amd64 config -h
generate images.json

Usage:
  image-migrator config [flags]

Flags:
  -h, --help                help for config
  -k, --kubeconfig string   The kubeconfig of k8s cluster's. Default is the $HOME/.kube/config. (default "/
root/.kube/config")
  -n, --namespaces string   Specify a namespace for information collection. If multiple namespaces are
specified, separate them with commas (,), such as ns1,ns2. default("") is all namespaces
  -t, --repo string         target repo,such as swr.cn-north-4.myhuaweicloud.com/test
```

Examples:

- Specify a namespace:

  ./image-migrator-linux-amd64 config -n default -t swr.cn-north-4.myhuaweicloud.com/test
- Specify multiple namespaces:

  ./image-migrator-linux-amd64 config -n default,kube-system -t swr.cn-north-4.myhuaweicloud.com/test
- If no namespace is specified, images of all namespaces are obtained:

  ./image-migrator-linux-amd64 config -t swr.cn-north-4.myhuaweicloud.com/test

Procedure

Step 1 Prepare the registry access permission file **auth.json**.

Create an **auth.json** file and modify it based on the format. If the repository is accessed anonymously, you do not need to enter information such as the username and password. Place the file in the directory where image-migrator is located.

Example:

```
{
  "quay.io/coreos": { },
  "swr.cn-north-4.myhuaweicloud.com": {
    "username": "cn-north-4@RVHVMX******",
    "password": "cab4ceab4a1545***************",
    "insecure": true
  }
}
```

For details about the parameters, see the **auth.json file**.

Step 2 Prepare the image list file **images.json**.

1. Connect to the source cluster using kubectl. For details, see **Connecting to a Cluster Using kubectl**.
2. Run the config subcommand for image migration to generate the **images.json** file.

   You can refer to the methods and examples in **Usage of image-migrator config** to obtain the image used in the source cluster application without specifying the namespace, or by specifying one or more namespaces.

3. Modify the **images.json** to make it meet the requirements in **images.json file**.

Step 3 Migrate images.

You can run the default **./image-migrator-linux-amd64** command to migrate images or configure image-migrator parameters as needed.

For example, run the following command:

./image-migrator-linux-amd64 --workers=5 --auth=./auth.json --images=./images.json --namespace=test --registry=swr.cn-north-4.myhuaweicloud.com --retries=2

Example:

```
$ ./image-migrator-linux-amd64
Start to generate images tasks, please wait ...
Start to handle images tasks, please wait ...
Images(38) migration finished, 0 images tasks failed, 0 tasks generate failed
```

Step 4 View the result.

After the preceding command is executed, information similar to the following is displayed:

```
Images(38) migration finished, 0 images tasks failed, 0 tasks generate failed
```

The preceding information indicates that 38 images have been migrated to SWR.

----End

3.5 Synchronizing Images Across Clouds from Harbor to SWR

Scenarios

A customer deploys services in multiple clouds and uses Harbor as their image repository. There are two scenarios for synchronizing images from Harbor to SWR:

1. Harbor accesses SWR through a public network. For details, see **Accessing SWR Through a Public Network**.
2. Harbor accesses SWR through a VPC endpoint by using a private line. For details, see **Accessing SWR Through a VPC Endpoint by Using a Private Line**.

Background

Harbor is an open source enterprise-class Docker Registry server developed by VMware. It extends the Docker Distribution by adding the functionalities such as role-based access control (RBAC), image scanning, and image replication. Harbor has been widely used to store and distribute container images.

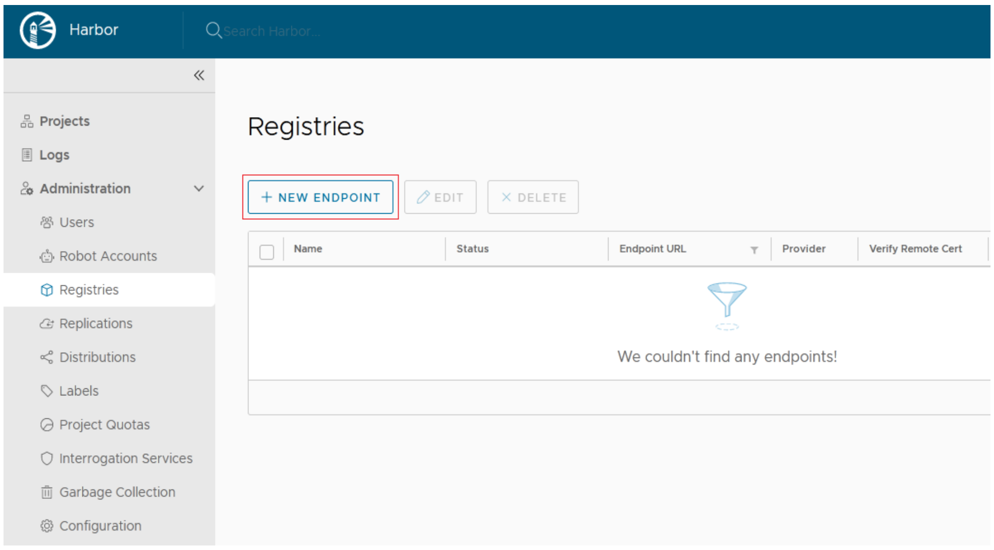
Accessing SWR Through a Public Network

Step 1 Configure a registry endpoint on Harbor.

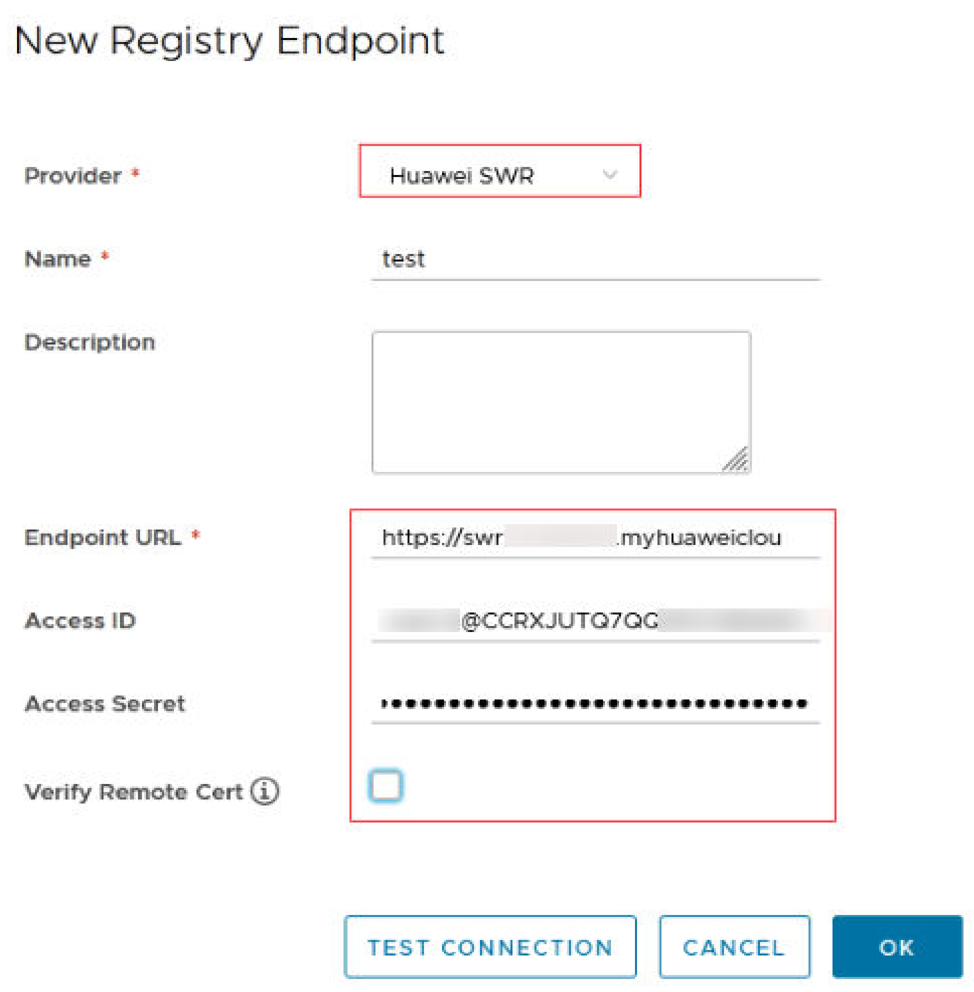
NOTE

Huawei Cloud SWR has integrated with Harbor 1.10.5 and later versions. You only need to set **Provider** to **Huawei SWR** when configuring your endpoint. This document uses Harbor 2.4.1 as an example.

1. Add an endpoint.



2. Configure the following parameters.

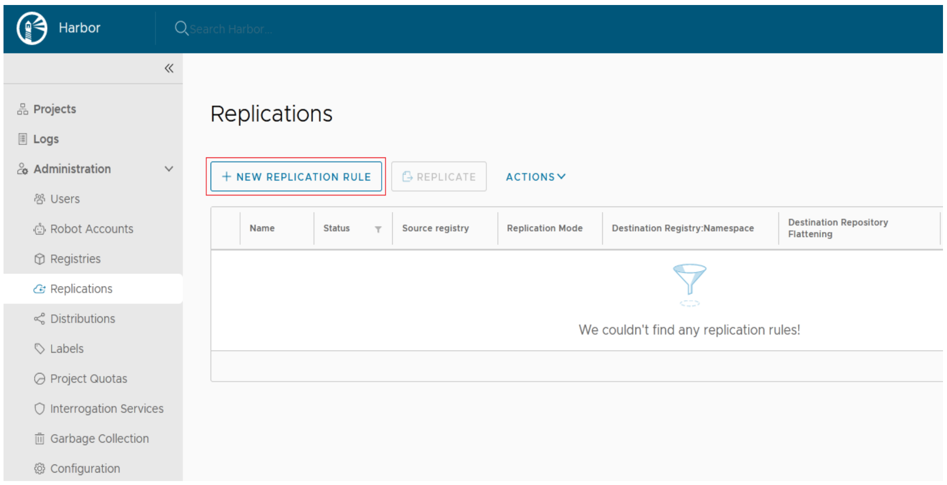

New Registry Endpoint

Provider *	Huawei SWR
Name *	test
Description	
Endpoint URL *	https://swr[redacted].myhuaweiclou
Access ID	[redacted]@CCRXJUTQ7QG[redacted]
Access Secret	•••••••••••••••••••••••••••••••••
Verify Remote Cert	☐

- **Provider**: Select **Huawei SWR**.
- **Name**: Enter a customized name.
- **Endpoint URL**: Enter the public network domain name of SWR in the format of **https://{*SWR image repository address*}**. To obtain the image repository address, log in to the SWR console, choose **My Images**, and click **Upload Through Client**. You can view the image repository address of the current region on the page that is displayed.
- **Access ID**: Enter an access ID in the format of **Regional project name@[AK]**.
- **Access Secret**: Enter an AK/SK. To obtain an AK/SK, see **Obtaining a Long-Term Valid Login Command**.
- **Verify Remote Cert**: **Deselect** the option (recommended).

Step 2 Configure a replication rule.

1. Create a replication rule.



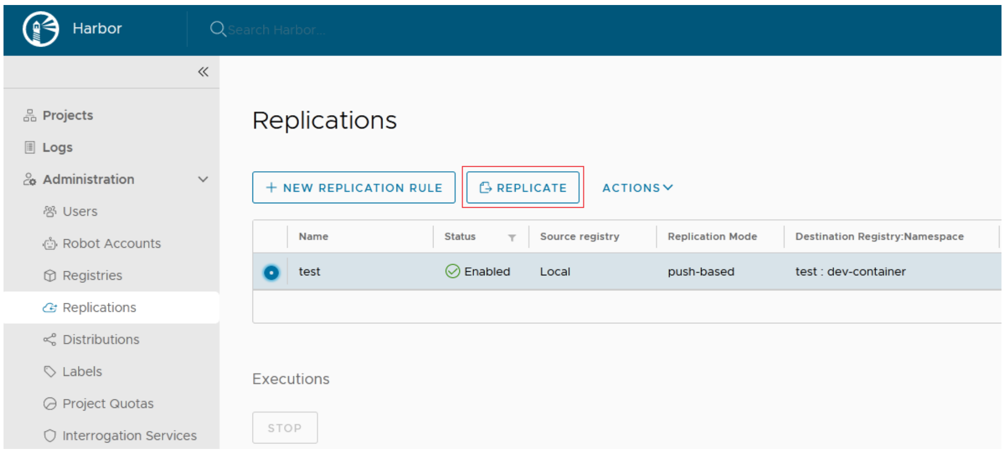
2. Configure the following parameters.

   – **Name**: Enter a customized name.
   – **Replication mode**: Select **Push-based**, indicating that images are pushed from the local Harbor to the remote repository.
   – **Source resource filter**: Filters images on Harbor based on the configured rules.
   – **Destination registry**: Select the endpoint created in **Step 1**.
   – **Destination**

     Namespace: Enter the organization name on SWR.

     Flattening: Select **Flatten All Levels**, indicating that the hierarchy of the registry is reduced when replicating images. If the directory of Harbor registry is **library/nginx** and the directory of the endpoint namespace is **dev-container**, after you flatten all levels, the directory of the endpoint namespace is **library/nginx -> dev-container/nginx**.
   – **Trigger Mode**: Select **Manual**.
   – **Bandwidth**: Set the maximum network bandwidth when executing the replication rule. The value **–1** indicates no limitation.

Step 3 After creating the replication rule, select it and click **REPLICATE** to complete the replication.



	Name	Status	Source registry	Replication Mode	Destination Registry:Namespace
●	test	Enabled	Local	push-based	test : dev-container

----End

Accessing SWR Through a VPC Endpoint by Using a Private Line

Step 1 Configure a VPC endpoint.

Step 2 Obtain the private network IP address and domain name of the VPC. (By default, the domain name resolution rule is automatically added to Huawei Cloud VPCs, so you only need to configure hosts for non-Huawei Cloud endpoints.) You can query the IP address and domain name in **Private Domain Name** on the VPC endpoint details page.

Step 3 Configure a registry endpoint on Harbor.

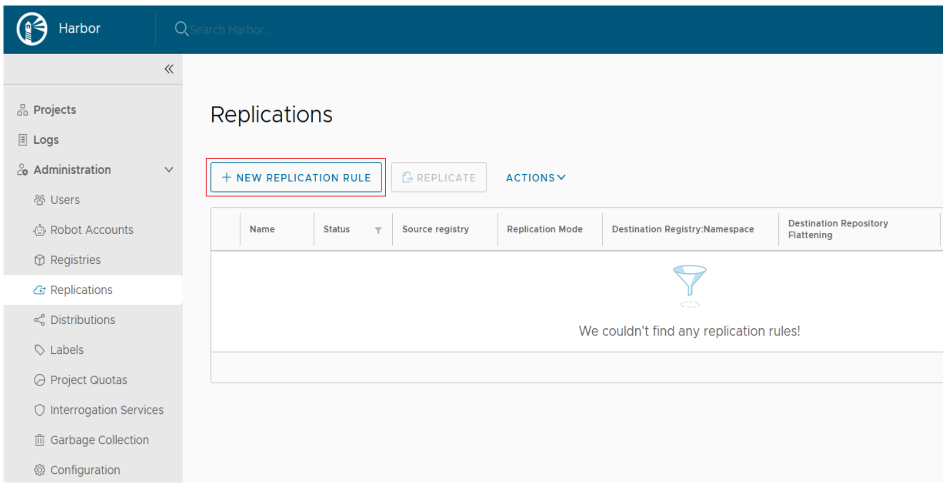
NOTE

Huawei Cloud SWR has integrated with Harbor 1.10.5 and later versions. You only need to set **Provider** to **Huawei SWR** when configuring your endpoint. This document uses Harbor 2.4.1 as an example.

1. Add an endpoint.
2. Configure the following parameters.
   - **Provider**: Select **Huawei SWR**.
   - **Name**: Enter a customized name.
   - **Endpoint URL**: Enter **the private network domain name of the VPC endpoint**, which must start with **https**. In addition, the domain name mapping must be configured in the container where Harbor is located.
   - **Access ID**: Enter an access ID in the format of **Regional project name@[AK]**.
   - **Access Secret**: Enter an AK/SK. To obtain an AK/SK, see **Obtaining a Long-Term Valid Login Command**.
   - **Verify Remote Cert**: **Deselect** the option.

Step 4 Configure a replication rule.

1. Create a replication rule.

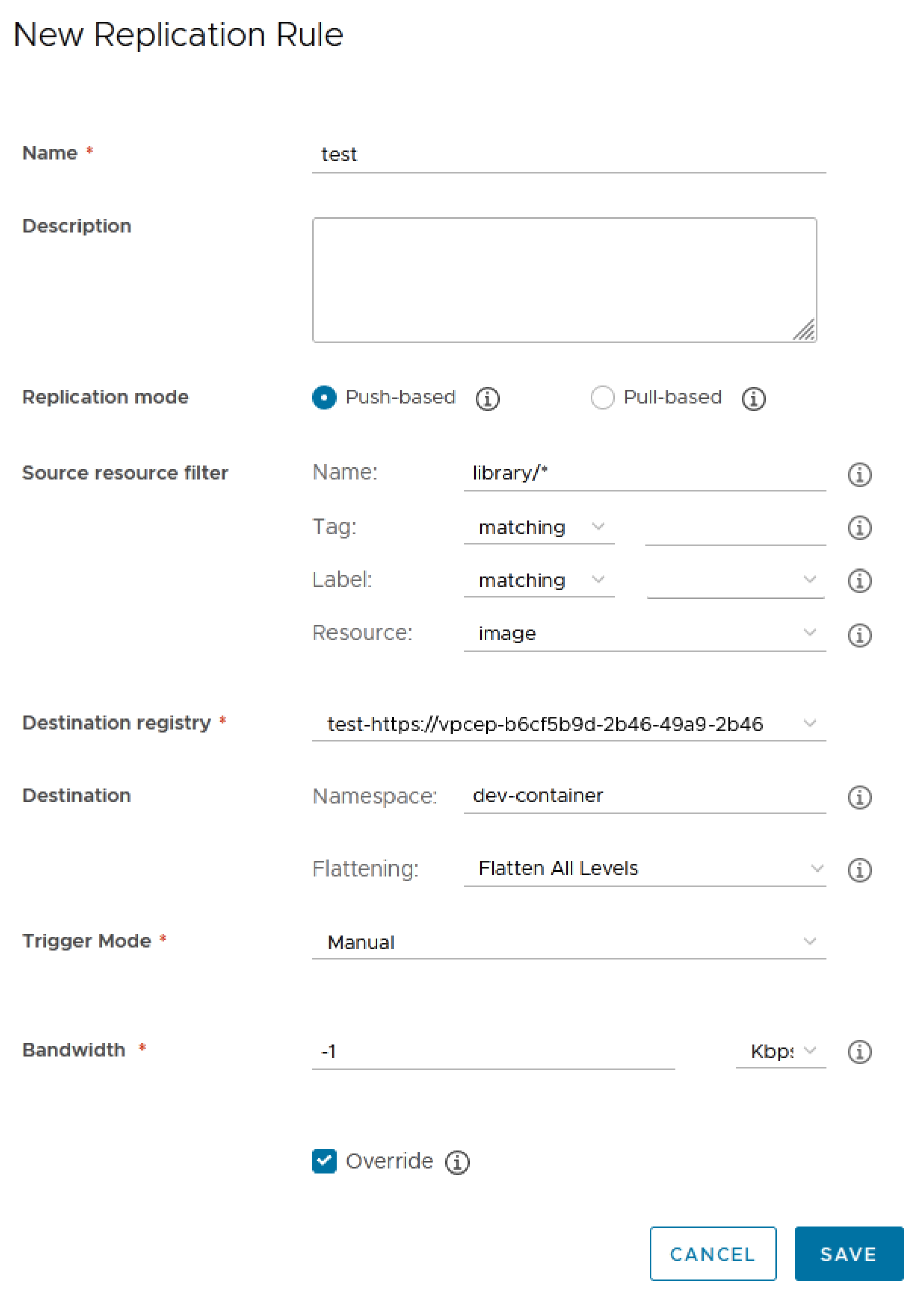


Replications

Name	Status	Source registry	Replication Mode	Destination Registry:Namespace	Destination Repository Flattening

We couldn't find any replication rules!

2. Configure the following parameters.


New Replication Rule

Name *	test
Description	
Replication mode	[✓ Push-based] Pull-based
Source resource filter	Name: library/*
	Tag: matching
	Label: matching
	Resource: image
Destination registry *	test-https://vpcep-b6cf5b9d-2b46-49a9-2b46
Destination	Namespace: dev-container
	Flattening: Flatten All Levels
Trigger Mode *	Manual
Bandwidth *	-1 Kbps
	[✓] Override

- **Name**: Enter a customized name.
- **Replication mode**: Select **Push-based**, indicating that images are pushed from the local Harbor to the remote repository.
- **Source resource filter**: Filters images on Harbor based on the configured rules.
- **Destination registry**: Select the endpoint created in **Step 3**.
- **Destination**

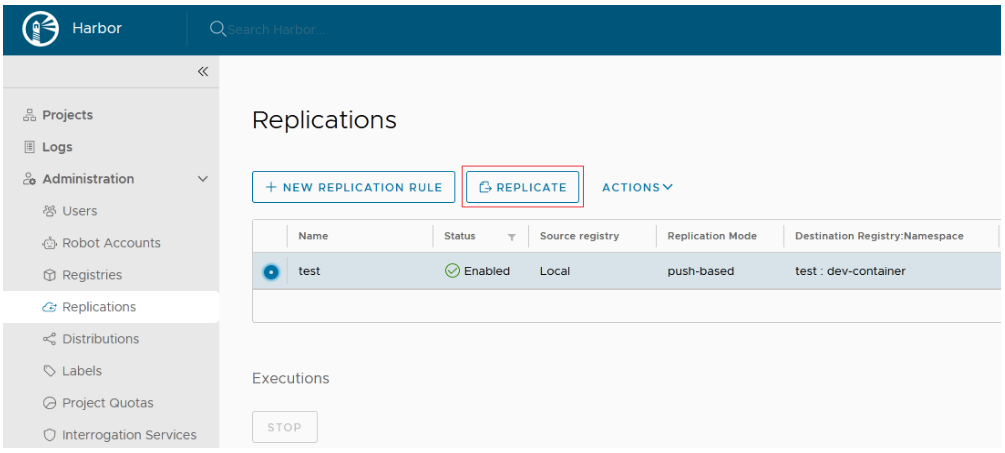
  Namespace: Enter the organization name on SWR.

  Flattening: Select **Flatten All Levels**, indicating that the hierarchy of the registry is reduced when replicating images. If the directory of Harbor registry is **library/nginx** and the directory of the endpoint namespace is

dev-container, after you flatten all levels, the directory of the endpoint namespace is **library/nginx -> dev-container/nginx**.

- **Trigger Mode**: Select **Manual**.
- **Bandwidth**: Set the maximum network bandwidth when executing the replication rule. The value **–1** indicates no limitation.

Step 5 After creating the replication rule, select it and click **REPLICATE** to complete the replication.



	Name	Status	Source registry	Replication Mode	Destination Registry:Namespace
●	test	Enabled	Local	push-based	test : dev-container

----End