

# SoftWare Repository for Container

## Best Practices

**Issue**            01  
**Date**             2026-03-26



**Copyright © Huawei Technologies Co., Ltd. 2026. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

# 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

---

<b>1 Fast Image Loading.....</b>	<b>1</b>
<b>2 Pulling an Image Through Dragonfly.....</b>	<b>3</b>
<b>3 Writing a Quality Dockerfile.....</b>	<b>11</b>
<b>4 Creating a JDK8 Base Image and Pushing It to SWR.....</b>	<b>20</b>
<b>5 Creating Multi-Architecture Images.....</b>	<b>23</b>
<b>6 Configuring an Access Network.....</b>	<b>25</b>
6.1 Overview.....	25
6.2 Private Network Access.....	25
6.3 Public Network Access.....	28
6.4 VPN/Direct Connect Access.....	31
<b>7 Migrating Container Images.....</b>	<b>35</b>
7.1 Overview.....	35
7.2 Migrating Images to SWR Using Docker Commands.....	38
7.3 Migrating Images to SWR Using image-syncer.....	39
7.4 Migrating Images to SWR Using image-migrator.....	41
7.5 Migrating Images Across Clouds from Harbor to SWR.....	47
<b>8 SWR Security Best Practices.....</b>	<b>55</b>
<b>9 CCE Clusters Pull Images from SWR Basic Edition Without Passwords.....</b>	<b>57</b>



**Step 4** Push the source image to the container image server. If the image has been pushed, skip this step. For details about how to push images, see [Pushing an Image to a Container Image Server](#).

**Step 5** Generate an index image. The index image is automatically pushed to the repository on the container image server. The index image is the key to speeding up the initial image pull.

```
apull-image-build convert --oci-ref --source <source-image> --target <index-image>
```

Example:

```
apull-image-build convert --oci-ref --source rnd-dockerhub.huawei.com:88/apull/centos-oci:latest --target rnf-dockerhub.huawei.com:88/apull/centos-oci:apull
```

---

 **CAUTION**

- The names of the source image and index image must be the same. If the names are different, the index image will fail to be generated.
- The image tags must be different. If the tags are the same, the source image will be overwritten.

---

----End

### Configuring the Index Image on CCE

**Step 1** Log in to the [CCE console](#) and purchase a cluster. For details, see [Buying a Cluster](#).

**Step 2** Click the name of the cluster to go to its console. In the navigation pane, choose **Nodes > Node Pools**. Then click **Create Node Pool**. In the **Advanced Settings** area, copy and paste the following commands for **Post-installation Command**. Configure other parameters based on service requirements. For details, see [Creating a Node Pool](#).

```
cd /tmp
wget --no-check-certificate https://cce-statics.ap-southeast-3.obs.ap-southeast-3.myhuaweicloud.com/job-platform-package/cce-addons/addons-apull-20250415193050793.zip
unzip addons-apull-20250415193050793.zip
cd addons
bash install.sh
```

**Step 3** Create a workload in the cluster. When you create the workload, enter the image name or select the image for **Image Name (Container Settings > Basic Info)**, and select the index image tag (**apull**) set in [Step 5](#) for **Image Tag**. Configure other parameters based on service requirements. For details, see [Creating a Deployment](#).

Now the image can be quickly loaded when the workload runs for the first time.

----End

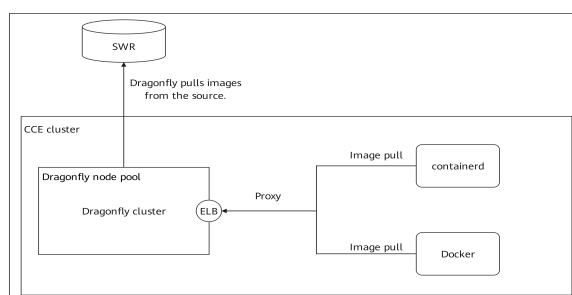
# 2 Pulling an Image Through Dragonfly

## Scenario

As business grows, the scale of container clusters for running enterprise applications also increases. When a cluster reaches a certain scale, the distribution of container images in the cluster faces some challenges. The first challenge is the bandwidth of the SWR service or of the backend storage. In some large clusters, hundreds or even thousands of nodes may pull an image at the same time, and high bandwidth is required.

There are many methods to solve this problem, such as dividing clusters, increasing caches, and load balancing. However, a better solution is P2P container image distribution that stores image data in a P2P network cluster and pulls the image from multiple P2P nodes. There are many P2P container image distribution technologies such as Kraken, BitTorrent, and Dragonfly. This section uses Dragonfly as an example to describe how to deploy Dragonfly in a CCE cluster and pull images through Dragonfly and a proxy. For details about other functions or configurations of Dragonfly, see the documents provided at the end of this section.

The following figure shows the deployment structure of Dragonfly.





- If the task is downloaded for the first time, the Seed Peer is triggered to download the image from the source and divide the task into pieces. After the registration is successful, the Scheduler schedules the Seed Peer to the Peer for streaming based on the piece level. When a piece is successfully downloaded, the metadata of the piece is reported to the Scheduler for the next scheduling. The piece is distributed to multiple Peers.
- If the piece is not downloaded for the first time, the Scheduler schedules another group of optimal Peers to download the piece. The HTTP proxy Peer downloads the piece from different Peers, combines the pieces, and returns the entire file. The P2P download is complete.

## Prerequisites

- A CCE cluster is available and a node pool has been created in the cluster. Dragonfly will be installed on the nodes in the node pool. The following uses the **test-dragonfly-nodepool** node pool as an example.
- Helm has been installed on a node in the CCE cluster.

## Deploying Dragonfly Components

**Step 1** Download and decompress the [Helm chart package](#) of Dragonfly to the node where Helm has been installed in the CCE cluster.

**Step 2** Change the value of **storageClass** in **dragonfly/values.yaml** to **csi-disk**.

Change the value of **nodeSelector** to the label (**cce.cloud.com/cce-nodepool: "test-dragonfly-nodepool"**) of the Dragonfly node pool. The label is automatically generated after the node pool is created. Change **test-dragonfly-nodepool** to the name of the node pool. This configuration is used to deploy the Dragonfly components in the Dragonfly node pool that is isolated from the service node pool.

```
global:
  # -- Global Docker image registry.
  imageRegistry: ""
  # -- Global Docker registry secret names as an array.
  imagePullSecrets: []
  # -- Global node labels for pod assignment.
  nodeSelector:
    cce.cloud.com/cce-nodepool: "test-dragonfly-nodepool"
  # -- Global storageClass for Persistent Volume(s).
  storageClass: "csi-disk"
```

**Step 3** Start to install Dragonfly.

1. Go to the Dragonfly directory generated after the Helm chart package is decompressed and run the following command:

```
helm install --wait --create-namespace --namespace dragonfly-system dragonfly .
```

 **CAUTION**

If the installation fails, uninstall the Dragonfly and install it again. After the uninstallation, you need to clear the Dragonfly-related PVCs for your cluster. If the PVCs are not deleted, the next Dragonfly reinstallation will fail. The uninstallation command is as follows:

```
helm uninstall --namespace dragonfly-system dragonfly
```

2. Wait for several minutes until information similar to the following is displayed (indicating all Dragonfly components are running normally):

```
[root@test-dragonfly-16758-e7uvz dragonfly]# helm install --wait --create-namespace --namespace dragonfly-system dragonfly
NAME: dragonfly
LAST DEPLOYED: Sat Apr 19 11:18:40 2025
NAMESPACE: dragonfly-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the manager address by running these commands:
  export MANAGER_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=manager" -o jsonpath={.items[0].metadata.name})
  export MANAGER_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $MANAGER_POD_NAME -o jsonpath={.spec.containers[0].ports[0].containerPort})
  kubectl --namespace dragonfly-system port-forward $MANAGER_POD_NAME 8080:$MANAGER_CONTAINER_PORT
  echo "Visit http://127.0.0.1:8080 to use your manager"
2. Get the scheduler address by running these commands:
  export SCHEDULER_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=scheduler" -o jsonpath={.items[0].metadata.name})
  export SCHEDULER_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $SCHEDULER_POD_NAME -o jsonpath={.spec.containers[0].ports[0].containerPort})
  kubectl --namespace dragonfly-system port-forward $SCHEDULER_POD_NAME 8082:$SCHEDULER_CONTAINER_PORT
  echo "Visit http://127.0.0.1:8082 to use your scheduler"
3. Configure runtime to use dragonfly:
  https://dfty.io/docs/getting-started/quick-start/kubernetes/
```

```
[root@test-dragonfly-16758-e7uvz ~]# kubectl get all -n dragonfly-system
NAME                                READY   STATUS    RESTARTS   AGE
pod/dragonfly-client-9xc8d          1/1    Running   0           22h
pod/dragonfly-client-jgzfp          1/1    Running   1 (22h ago) 22h
pod/dragonfly-client-mc67z          1/1    Running   2 (22h ago) 22h
pod/dragonfly-manager-6d6447d5b9-9s442 1/1    Running   1 (22h ago) 22h
pod/dragonfly-manager-6d6447d5b9-cfbmx 1/1    Running   0           22h
pod/dragonfly-manager-6d6447d5b9-fczqv 1/1    Running   1 (22h ago) 22h
pod/dragonfly-mysql-0                1/1    Running   0           22h
pod/dragonfly-redis-master-0         1/1    Running   0           22h
pod/dragonfly-redis-replicas-0       1/1    Running   0           22h
pod/dragonfly-redis-replicas-1       1/1    Running   0           22h
pod/dragonfly-scheduler-0            1/1    Running   0           22h
pod/dragonfly-scheduler-1            1/1    Running   0           22h
pod/dragonfly-scheduler-2            1/1    Running   0           22h
pod/dragonfly-seed-client-0          1/1    Running   3 (22h ago) 22h
pod/dragonfly-seed-client-1          1/1    Running   0           22h
pod/dragonfly-seed-client-2          1/1    Running   0           22h
```

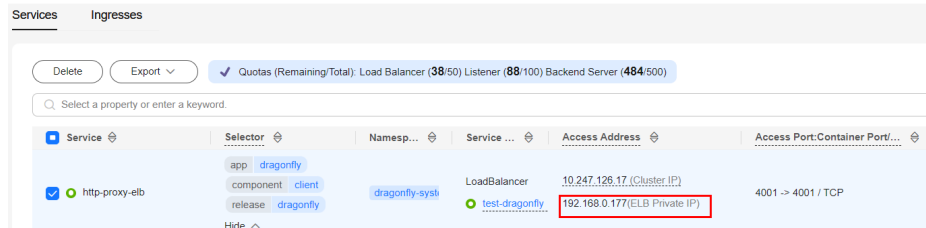
```
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/dragonfly-manager            NodePort       10.247.99.122   <none>           8080:31783/TCP,65003:32035/TCP 22h
service/dragonfly-manager-metrics    ClusterIP      10.247.230.242 <none>           8000/TCP          22h
service/dragonfly-mysql              ClusterIP      10.247.214.197 <none>           3306/TCP          22h
service/dragonfly-mysql-headless     ClusterIP      None            <none>           3306/TCP          22h
service/dragonfly-redis-headless     ClusterIP      None            <none>           6379/TCP          22h
service/dragonfly-redis-master       ClusterIP      10.247.237.184 <none>           6379/TCP          22h
service/dragonfly-redis-replicas     ClusterIP      10.247.115.34  <none>           6379/TCP          22h
service/dragonfly-scheduler          ClusterIP      10.247.127.130 <none>           8002/TCP          22h
service/dragonfly-scheduler-metrics  ClusterIP      10.247.226.143 <none>           8000/TCP          22h
service/dragonfly-seed-client        ClusterIP      10.247.41.129  <none>           4001/TCP,4003/TCP,4004/TCP      22h
```

 **NOTE**

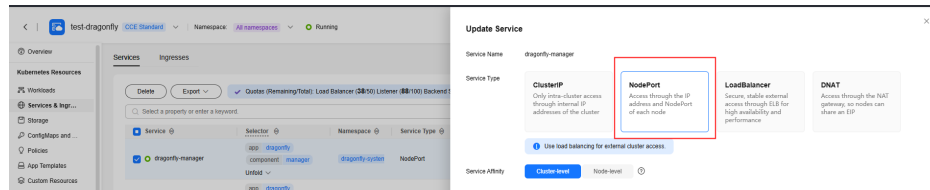
**dragonfly-client** indicates each Peer in the architecture diagram, and **dragonfly-seed-client** indicates each Seed Peer.

After the deployment is complete, some Services, such as **dragonfly-manager** and **dragon-mysql**, are created. The **dragonfly-manager** Service needs to be exposed to users on the console.

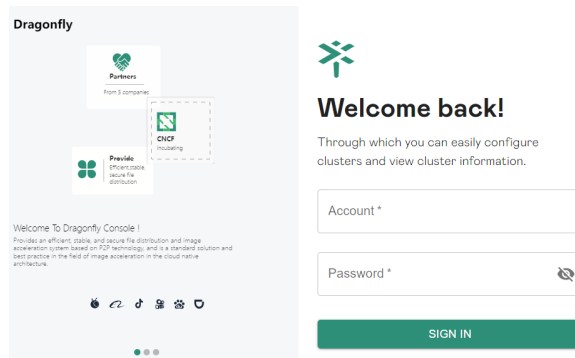
3. Wait until all pods are running and then create a LoadBalancer Service for the DaemonSet named **dragonfly-client** in the cluster. Set both **Container Port** and **Service Port** to **4001** so that traffic is forwarded to port 4001 of each dragonfly-client pod. Record the load balancer private IP address in the **Access Address** column (192.168.0.177:4001 in this example) of the LoadBalancer Service. This address will be used as the proxy address of the Docker or containerd client.



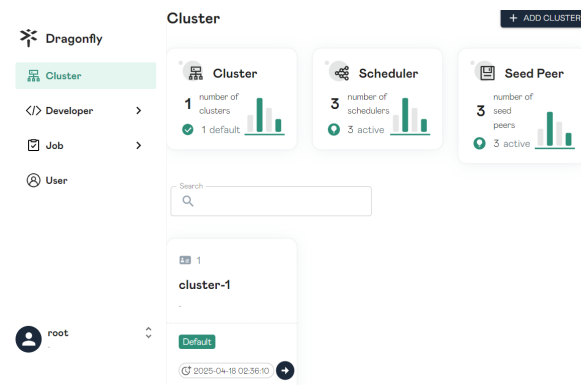
**Step 4** Log in to the **CCE console**, access your cluster, and choose **Services** in the navigation pane. Update the access type of the **dragonfly-manager** Service to NodePort so that the Service can be accessed externally.



Access `<node-IP-address>:<node-port>`. The node port in this example is 8080.



The preset username is **root** and the preset password is **dragonfly**. You can view the information about each component and job on the console.



Now, Dragonfly has been deployed and port 4001 of **dragonfly-client** has been exposed as the HTTP proxy port through the LoadBalancer Service.

----End

## Verifying that Images Can Be Pulled Using Dragonfly

You have set up a Dragonfly cluster and created a LoadBalancer Service (whose access address is 192.168.0.177:4001 in this example) to expose **dragonfly-client**.

Now, you can use this Service to receive the traffic from the containerd client and Docker client to Dragonfly.

## containerd Client

**Step 1** Configure the containerd client to connect to Dragonfly through an HTTP proxy.

### CAUTION

When configuring the proxy, pay special attention **NO\_PROXY**. You are advised to configure all requests in **NO\_PROXY** except image pull requests. This prevents network access exceptions or even service exceptions.

1. Edit the **http-proxy.conf** file in the `/etc/systemd/system/containerd.service.d/` directory. If this directory does not exist, manually create it.

```
vim /etc/systemd/system/containerd.service.d/http-proxy.conf
```

2. Copy the following content to the **http-proxy.conf** file and change **192.168.0.177:4001** to the address recorded in [Step 3.3](#) (the HTTP proxy forwards the traffic for pulling images to Dragonfly):

```
[Service]
Environment="HTTP_PROXY=http://192.168.0.177:4001"
Environment="HTTPS_PROXY=http://192.168.0.177:4001"
Environment="NO_PROXY=localhost,127.0.0.1,127.0.0.0/8,.cluster.local,svc"
```

**Step 2** Ignore the certificate verification.

- If no HTTPS certificate is configured for Dragonfly, edit the containerd configuration file **config.toml** to ignore the certificate verification.

```
vim /etc/containerd/config.toml
```

Copy the following content to the **config.toml** file (replace **test-uvpv1j.swr-pro.myhuaweicloud.com/library/busybox:latest** with the address of the target image):

```
[plugins."io.containerd.grpc.v1.cri".registry.configs]
[plugins."io.containerd.grpc.v1.cri".registry.configs."test-uvpv1j.swr-pro.myhuaweicloud.com".tls]
insecure_skip_verify = true
```

- If a valid HTTPS certificate has been configured for Dragonfly, skip this step.

**Step 3** Restart containerd and pull the image.

```
systemctl daemon-reload && systemctl restart containerd.service
crictl -debug pull test-uvpv1j.swr-pro.myhuaweicloud.com/library/mysql:8.0.36-debian-12-r10
```

Replace **test-uvpv1j.swr-pro.myhuaweicloud.com/library/busybox:latest** in the **crictl -debug pull** command with the address of the target image.

### NOTE

If you verify the image repeatedly, you need to delete the image that has been pulled locally. If the image to be pulled already exists locally, the image will not be pulled again.

```
crictl rmi test-uvpv1j.swr-pro.myhuaweicloud.com/library/mysql:8.0.36-debian-12-r10
```

**Step 4** Log in to the [CCE console](#), access your cluster, and choose **Workloads** in the navigation pane. On the **DaemonSets** tab, locate the **dragonfly-client** DaemonSet and click **View Log** in the **Operation** column. If the following information is displayed, the image is successfully pulled using Dragonfly.

```
0025-04-23T01:40:28.121152923480-00 INFO download_task:download/download_partial_from_local: dragonfly-client/src/resource/task.rs:1573: finished piece 43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4-16
1 From local host_id=192.168.0.44-test-dragonfly-ndepool-vg8a task_id=43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4 peer_id=192.168.0.44-test-dragonfly-ndepool-vg8a-d616b07-7a78-45bd-a105-110
ccf7836c0
0025-04-23T01:40:28.121152923480-00 INFO download_task:download/download_partial_from_local: dragonfly-client/src/resource/task.rs:1573: finished piece 43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4-16
1 From local host_id=192.168.0.44-test-dragonfly-ndepool-vg8a task_id=43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4 peer_id=192.168.0.44-test-dragonfly-ndepool-vg8a-d616b07-7a78-45bd-a105-110
ccf7836c0
0025-04-23T01:40:28.121152923480-00 INFO download_task:download/download_partial_from_local: dragonfly-client/src/resource/task.rs:1573: finished piece 43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4-17
1 From local host_id=192.168.0.44-test-dragonfly-ndepool-vg8a task_id=43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4 peer_id=192.168.0.44-test-dragonfly-ndepool-vg8a-d616b07-7a78-45bd-a105-110
ccf7836c0
0025-04-23T01:40:28.121152923480-00 INFO download_task:download/download_partial_from_local: dragonfly-client/src/resource/task.rs:1573: finished piece 43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4-17
1 From local host_id=192.168.0.44-test-dragonfly-ndepool-vg8a task_id=43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4 peer_id=192.168.0.44-test-dragonfly-ndepool-vg8a-d616b07-7a78-45bd-a105-110
ccf7836c0
0025-04-23T01:40:28.121152923480-00 INFO download_task:download: dragonfly-client/src/resource/task.rs:1510: interested pieces after removing the finished pieces: [] host_id=192.168.0.44-test-dragonfly-ndepool-vg8a ta
sk_id=43409207d63409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4 peer_id=192.168.0.44-test-dragonfly-ndepool-vg8a-d616b07-7a78-45bd-a105-110ccf7836c0
0025-04-23T01:40:28.121152923480-00 INFO download_task:download: dragonfly-client/src/resource/task.rs:1510: all pieces are downloaded from local host_id=192.168.0.44-test-dragonfly-ndepool-vg8a task_id=43409207d63
409e6b7b4ac93c7734c3290e9946ac2c962770f1915f46bd4 peer_id=192.168.0.44-test-dragonfly-ndepool-vg8a-d616b07-7a78-45bd-a105-110ccf7836c0
0025-04-23T01:40:28.121152923480-00 INFO download_task:dragonfly-client/src/runner/diagnostic/download.rs:107: download task succeeded host_id=192.168.0.44-test-dragonfly-ndepool-vg8a task_id=43409207d63409e6b7b4ac
93c7734c3290e9946ac2c962770f1915f46bd4 peer_id=192.168.0.44-test-dragonfly-ndepool-vg8a-d616b07-7a78-45bd-a105-110ccf7836c0
0025-04-23T01:40:33.330277252480-00 INFO upgraded_tunnel:upgraded_tunnel/proxy_via_docker: dragonfly-client/src/proxy/mod.rs:184: message is none url=/v2/library/busybox/manifests/latest?nsd=002770f1915f46bd4 peer_id=192.168.0.44-test-dragonfly-ndepool-
vg8a-d616b07-7a78-45bd-a105-110ccf7836c0
```

----End

## Docker Client

**Step 1** Configure the Docker client to connect to Dragonfly through an HTTP proxy.



When configuring the proxy, pay special attention **NO\_PROXY**. You are advised to configure all requests in **NO\_PROXY** except image pull requests. This prevents network access exceptions or even service exceptions.

1. Edit the **http-proxy.conf** file in the **/etc/systemd/system/docker.service.d/** directory. If this directory does not exist, manually create it.  
`vim /etc/systemd/system/docker.service.d/http-proxy.conf`

2. Copy the following content to the **http-proxy.conf** file and change **192.168.0.177:4001** to the address recorded in **Step 3.3** (the HTTP proxy forwards the traffic for pulling images to Dragonfly):

```
[Service]
Environment="HTTP_PROXY=http://192.168.0.177:4001"
Environment="HTTPS_PROXY=http://192.168.0.177:4001"
Environment="NO_PROXY=localhost,127.0.0.1,127.0.0.0/8,cluster.local,svc"
```

**Step 2** Ignore the certificate verification.

- If no HTTPS certificate is configured for Dragonfly, edit **/etc/docker/daemon.json** and set **insecure-registries** to the domain name of the target image repository.

```
[root@test-dragonfly-33468 ~]# cat /etc/docker/daemon.json
{
  "storage-driver": "overlay2",
  "max-concurrent-downloads": 3,
  "log-opts": {
    "max-size": "50m",
    "max-file": "20"
  },
  "insecure-registries": ["test-uvpv1j.swr-pro.myhuaweicloud.com"],
  "runtimes": {
    "nvidia": {
      "path": "/usr/bin/nvidia-container-runtime",
      "runtimeArgs": []
    }
  }
}
```

- If a valid HTTPS certificate has been configured for Dragonfly, skip this step.

**Step 3** Restart Docker and pull the image.

```
systemctl daemon-reload && systemctl restart docker.service
docker pull test-uvpv1j.swr-pro.myhuaweicloud.com/library/busybox:latest
```

Replace **test-uvpv1j.swr-pro.myhuaweicloud.com/library/busybox:latest** in the **docker pull** command with the address of the target image.

 NOTE

If you verify the image repeatedly, you need to delete the image that has been pulled locally. If the image to be pulled already exists locally, the image will not be pulled again.

```
docker rmi test-uvpv1j.swr-pro.myhuaweicloud.com/library/busybox:latest
```

**Step 4** Log in to the **CCE console**, access your cluster, and choose **Workloads** in the navigation pane. On the **DaemonSets** tab, locate the **dragonfly-client** DaemonSet and click **View Log** in the **Operation** column. If the following information is displayed, the image is successfully pulled using Dragonfly.

```
2025-04-23T01:49:36.1103112840000 [INFO] download_task:download:download_partial_with_scheduler:announce_peer:client: dragonfly-client/src/grpc/scheduler.rs:470: picked Wode { addr: 10.0.0.240:8082 } host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77 task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.277041467490000 [INFO] download_task:download:download_partial_with_scheduler: dragonfly-client/src/resource/task.rs:572: announce_peer has been completed host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.277041467490000 [INFO] download_task:download:download_partial_with_scheduler: dragonfly-client/src/resource/task.rs:812: need back to source response: NeedBackToSourceResponse { description: Some("such request exceeded MaxRequestSizeLimit") } host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.277041467490000 [INFO] download_task:download:download_partial_with_scheduler: dragonfly-client/src/resource/task.rs:820: sent DownloadPeerBackToSourceRequest host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.277041467490000 [INFO] download_task:download:download_partial_with_scheduler: dragonfly-client/src/resource/task.rs:830: start to download piece c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225-0 from source host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.4107144948000 [INFO] download_task:download:download_partial_with_scheduler: dragonfly-client/src/resource/task.rs:1407: finished piece c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225-0 from source host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.4122233694000 [INFO] download_task:download:download_partial_with_scheduler: dragonfly-client/src/resource/task.rs:890: sent DownloadPeerBackToSourceInShotRequest host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.4122233694000 [INFO] download_task:download: dragonfly-client/src/resource/task.rs:476: interested pieces after removing the finished pieces: [] host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.412700791490000 [INFO] download_task:download: dragonfly-client/src/resource/task.rs:486: all pieces are downloaded with scheduler host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.412714555490000 [INFO] download_task: dragonfly-client/src/grpc/dragonfly_download.rs:437: Download task successful host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
2025-04-23T01:49:36.4125486949000 [INFO] upgraded_tunnel:upgraded_handler:proxy_via_dfdemon: dragonfly-client/src/proxy/mod.rs:849: message is none url=/v2/Library/busybox/objs/sha256:1f7d793e936c4b70c1522922d45c830e215608b100720111b0c?method=GET host_id=192.168.0.44-test-dragonfly-nodpool-ugtkba task_id=c0599a020545f04fd75a0b7147c173184636a8ba8b7632f026075d4225 peer_id=192.168.0.44-test-dragonfly-nodpool-ugtkba-b17905ee-464f-4262-a4c2-836f839c3b77
```

----End

Reference

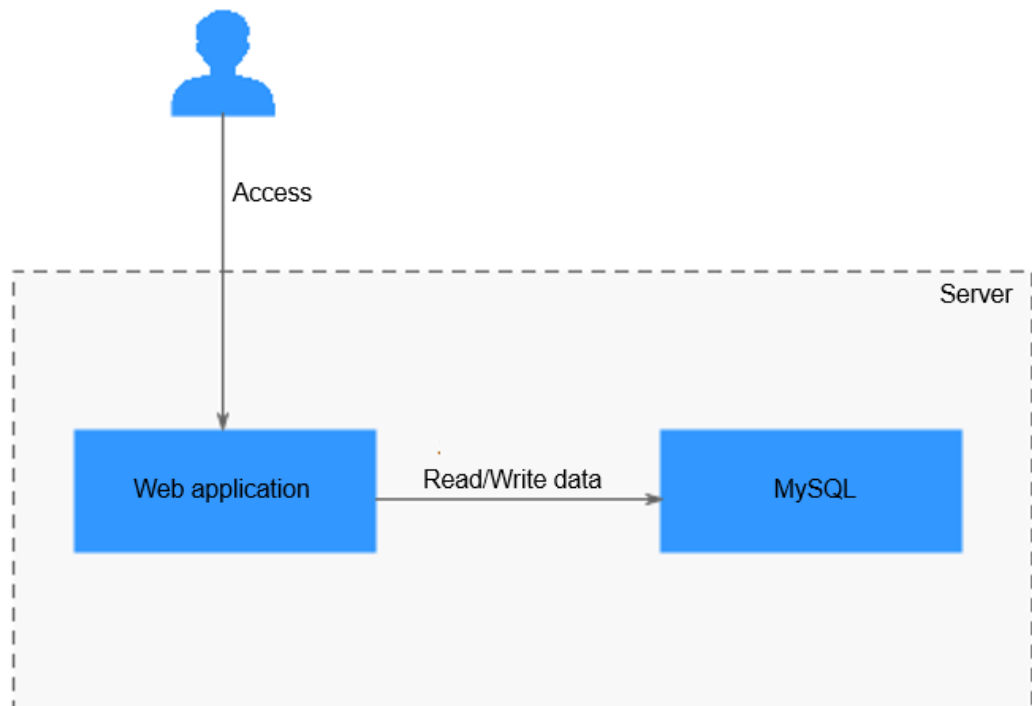
<https://d7y.io/docs/next/>

<https://github.com/dragonflyoss/dragonfly>

# 3 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, you can write a Dockerfile 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 of Similar Update Frequency**
- **Specify an Image Tag**
- **Delete Unnecessary Files**
- **Select a Suitable Base Image**
- **Set WORKDIR and CMD**
- **(Optional) Use ENTRYPOINT**
- **Use exec in ENTRYPOINT**
- **Use COPY Preferentially**
- **Change the Order of COPY and RUN**
- **Set Default Environment Variables, Mapping Ports, and Data Volumes**
- **Use EXPOSE to Set Listening Ports**
- **Use VOLUME to Manage Data Volumes**
- **Use Labels to Configure Image Metadata**
- **Use HEALTHCHECK**
- **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.

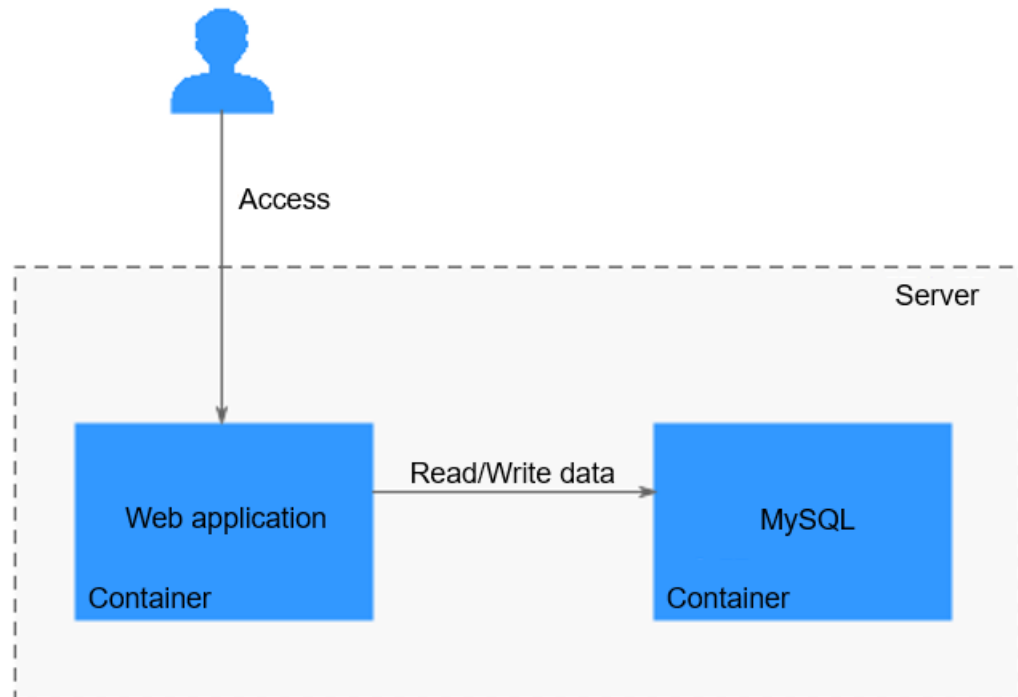
Therefore, usually, an application is split into microservices before containerization. You will benefit a lot from the microservice architecture:

- **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 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 Tag

If no tag is specified for an image, the image will be tagged with **latest** by default. For example, the **FROM ubuntu** command is equivalent to **FROM ubuntu:latest**. During an image update, **latest** will point to a new tag. The image build may fail.

In the sample Dockerfile, tag the **ubuntu** 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 a node program, there is no need to use a general base image. A **node** image would be a better choice.

A **node** image tagged with **alpine** is recommended. Alpine is a lightweight Linux distribution 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

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

**CMD** provides default commands to be executed when running a container from an image. Write the commands in an array.

```
FROM node:7-alpine
WORKDIR /app
ADD . /app
RUN npm install

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

## (Optional) Use ENTRYPOINT

**ENTRYPOINT** 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"]
```

## Use exec in ENTRYPOINT

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

## Use COPY Preferentially

**COPY** is simply used to copy files to images. **ADD** is more complex and can be used to download remote files and decompress packages.

```
FROM node:7-alpine
WORKDIR /app
COPY . /app
RUN npm install

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

## Change the Order of COPY and RUN

Place the parts that are infrequently changed in the front of your Dockerfile to make the most out of the image cache.

In the sample Dockerfile, the source code changes frequently. Every time the image is built, npm needs to be reinstalled. To avoid this issue, copy **package.json** first, then install npm, and 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"]
```

Environment variables specified by **ENV** can be used in containers. If you only need to specify variables for image build, you can use **ARG** instead.

## Use EXPOSE to Set Listening Ports

**EXPOSE** is used to describe which ports your containers will listen on. For example, set **EXPOSE 80** for an Apache image and **EXPOSE 27017** for a MongoDB image.

For external access, use a flag to map ports when executing **docker run**.

```
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 VOLUME to Manage Data Volumes

**VOLUME** is used to access database storage files, configuration files, or files and directories of created containers. You are advised to use **VOLUME** to manage the image modules that can change or the modules modifiable for users.

In the sample Dockerfile, a media directory is added.

```
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 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 the spaces into escape characters. If the string itself contains quotation marks, convert the quotation marks.

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

## Use HEALTHCHECK

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? **HEALTHCHECK** 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/
```

# 4 Creating a JDK8 Base Image and Pushing It to SWR

---

## Scenarios

A base image can be used as a basis to create other images. This section describes how to create a JDK8 base image on a CCE node and push it to SWR.

## Procedure

**Step 1** Buy a CCE cluster.

1. Log in to the [CCE console](#).
2. On the CCE cluster purchase page, configure parameters. For details, see [Buying a CCE Standard/Turbo Cluster](#).
3. Wait until the cluster is created. The new cluster will be displayed in the cluster list. The cluster status is running and the number of nodes is 0.

**Step 2** Create a CCE node.

After a cluster is created, you need to create nodes for running workloads in the cluster. Assume that the node you are creating has Linux and Docker installed. You can use it to create a base image.

In the following steps, CentOS 7.6 is used as an example to describe how to create a JDK8 base image and push it to SWR.

1. Log in to the [CCE console](#).
2. Click the cluster created in [Step 1](#).
3. In the navigation pane, choose **Nodes**. On the **Nodes** tab, click **Create Node** in the upper right corner. In the displayed dialog box, [configure node parameters](#).
4. For network settings, select **Auto create** for **EIP**. Set the bandwidth to **5 Mbit/s**.
5. Click **Next: Confirm**.
6. Check the node specifications, read the instructions, select **I have read and agree to CCE Disclaimer**, and click **Submit**.

Wait until the node is created. The new node will be displayed in the node list. The node status is **Running**.

### Step 3 Download a JDK package.

1. After a node is created, click the node name to go to its details page.
2. In the upper right corner, click **Remote Login**.
3. Select a login mode and click **Log In**.
4. Log in to the node as **root**.
5. Create a directory **image**.

```
mkdir image
```

6. Go to the **image** directory.

```
cd image
```

7. Download a JDK package.

```
wget https://builds.openlogic.com/downloadJDK/openlogic-openjdk/8u352-b08/openlogic-openjdk-8u352-b08-linux-x64.tar.gz
```

### Step 4 Build an image.

1. Run **vi dockerfile** to write a Dockerfile as follows:

```
FROM centos # Use CentOS as the base image.
RUN useradd -d /home/springboot -m springboot # Create a user in
the working directory.
ADD ./openlogic-openjdk-8u352-b08-linux-x64.tar.gz /home/springboot # Add the
JDK software package to the image and decompress it automatically.
RUN chown springboot:springboot /home/springboot/openlogic-openjdk-8u352-b08-linux-x64 -R
USER springboot # Set the user to springboot.
ENV JAVA_HOME=/home/springboot/openlogic-openjdk-8u352-b08-linux-x64 # Set
environment variables.
ENV PATH=$JAVA_HOME/bin:$PATH \
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
WORKDIR /home/springboot/ # Set the working
directory of the image.
```

2. Press **Esc** and enter **:wq** to save the Dockerfile and exit.

3. Build an image.

```
docker build -t openjdk:8 .
```

4. Run **docker images** to check whether the image is built successfully.

Figure 4-1 Checking the built image

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
openjdk	8	40a78b1544f1	28 hours ago	621MB
swr.cn				
icloud.com/devcloud-t/openjdk	8	40a78b1544f1	28 hours ago	621MB
centos	7	eeb6ee3f44bd	15 months ago	204MB
swr.cn				
icloud.com/root/swr-demo-2048	latest	647c57f8354f	4 years ago	109MB
swr.cn				
icloud.com/testawa0306/swr-demo-2048	latest	647c57f8354f	4 years ago	109MB

### Step 5 Log in to the **SWR console** and create an organization.

Example: Create an organization **testawa0306**.

### Step 6 Push the image to the organization created in **Step 5**.

1. Log in to **SWR console** as user **root**.
2. Tag the image.
3. Push the image to the organization created in **Step 5**.

```
docker push swr.ap-southeast-3.myhuaweicloud.com/testawa0306/
openjdk:v8.8
```

After an image is pushed, you can find it on the **My Images** page of the SWR console.

**Step 7** (Optional) Use the pushed image to [deploy a workload in CCE](#).

----**End**

# 5 Creating Multi-Architecture Images

## Scenario

When images are required in the x86 and Arm architectures, and different image tags need to be set, you can create multi-architecture images and manage these images of the same tag using the **docker manifest** command.

## Preparations

1. Check the Docker version.

Ensure that you use Docker 19.03 or later that supports the manifest extension. Run the following command to view the version:

```
docker --version
```

2. If your Docker version is earlier than 19.03, set the **experimental** environment variable.

Open the Docker daemon configuration file **/etc/docker/daemon.json**.

```
vim /etc/docker/daemon.json
```

Change the value of **experimental** to **true**. If this field does not exist, add it.

```
"experimental": true
```

Save the file and restart Docker to apply the modification.

```
sudo systemctl restart docker
```

3. Check whether experimental Docker CLI features are enabled.

```
echo $DOCKER_CLI_EXPERIMENTAL
```

```
[root@test-auto-45915 ~]# echo $DOCKER_CLI_EXPERIMENTAL  
enabled
```

- If **enabled** is returned, experimental Docker CLI features are enabled.
- If **disabled** or no information is returned, experimental Docker CLI features are not enabled.

## Procedure

- Step 1** Build and push two images to the image repository by following the instructions in [Creating a JDK8 Base Image and Pushing It to SWR](#).

```
docker push ${repository_url}/${organization}/${image_name}:${image_tag}-amd64  
docker push ${repository_url}/${organization}/${image_name}:${image_tag}-arm64
```

**Step 2** Build a manifest for the multi-architecture images.

## 1. Create a manifest.

```
docker manifest create --amend --insecure ${repository_url}/${organization}/${image_name}:${image_tag} $
${repository_url}/${organization}/${image_name}:${image_tag}-amd64 $
${repository_url}/${organization}/${image_name}:${image_tag}-arm64
```

2. Modify the manifest to add **arch** information.

```
docker manifest annotate ${repository_url}/${organization}/${image_name}:${image_tag} $
${repository_url}/${organization}/${image_name}:${image_tag}-amd64 --arch amd64
docker manifest annotate ${repository_url}/${organization}/${image_name}:${image_tag} $
${repository_url}/${organization}/${image_name}:${image_tag}-arm64 --arch arm64
```

**Step 3** Push the manifest.


```
docker manifest push ${repository_url}/${organization}/${image_name}:${image_tag}
```

----End

## Downloading an Image Across CPU Architectures

Use the **--platform** parameter to specify the architecture of the image to be pulled.

```
docker pull ${repository_url}/${organization}/${image_name}:${image_tag} --platform=linux/amd64
```



```
[root@ecs-f6fc ~]# docker pull swr.          icloud.com/swr-ee/duojiagou:duojiagouv1 --platform=linux/arm64
duojiagouv1: Pulling from swr-ee/duojiagou
70928ed4d12f: Pull complete
1f601ffade39: Pull complete
e3c93cc0c1c6: Pull complete
Digest: sha256:b682c82ccd78fe82ca74b6276955137170ee022e5e490573ad0d86ec6e6cc9db
Status: Downloaded newer image for swr.          icloud.com/swr-ee/duojiagou:duojiagouv1
swr          icloud.com/swr-ee/duojiagou:duojiagouv1
[root@ecs-f6fc ~]#
```

**--platform** parameter description:

- **--platform=linux/amd64**: Linux system of the x86\_64 architecture
- **--platform=linux/arm64**: Linux system of the ARM64 architecture

# 6 Configuring an Access Network

---

## 6.1 Overview

For SWR Basic Edition, if your container engine client is installed on a CCE node or an ECS, you have three choices for the network used to pull and push images.

- [Private Network Access](#)
- [Public Network Access](#)
- [VPN/Direct Connect Access](#)

## 6.2 Private Network Access

SWR supports two ways of private network access.

- [Default way](#)
- [Access through VPC Endpoint](#)

### Access Through the Default Way

If your container engine client is installed on a CCE node or an ECS that is in the same region as the image repository, you can push or pull images over 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 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.

---

## Access Through VPC Endpoint

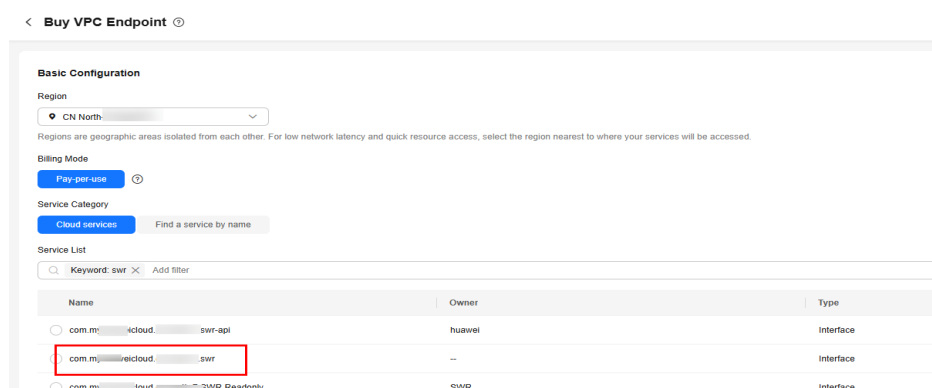
### Scenarios

VPC Endpoint is used to establish a convenient, secure, and private connection channel between the VPC and SWR. If you have higher security requirements, you can purchase a VPC endpoint service to access SWR.

### Procedure

- Step 1** Log in to the [VPC Endpoint console](#).
- Step 2** On the **VPC Endpoints** page, click **Buy VPC Endpoint**. In the **Select List**, select the VPC endpoint service below and then configure other parameters as needed:

com.myhuaweicloud.{region\_id}.swr



### NOTE

In some regions, the VPC endpoint service for SWR is not displayed in the **Service List**. You can search for it by name.

**Table 6-1** VPC endpoint services for SWR in each region

Region Name	VPC Endpoint Service for SWR	OBS Bucket for SWR
CN Southwest-Guiyang1	com.myhuaweicloud.cn-southwest-2.swr	Primary bucket: op-svc-swr-b051-10-205-14-19-3az Standby bucket: op-svc-swr-cn-southwest-2-backup
CN East2	com.myhuaweicloud.cn-east-4.swr	op-svc-swr-cn-east-4-multiaz
CN South-Guangzhou	com.myhuaweicloud.cn-south-1.swr	Primary bucket: op-svc-swr-b051-10-230-33-197-3az Standby bucket: op-svc-swr-cn-south-1-backup
ME-Riyadh	com.myhuaweicloud.me-east-1.swr	op-svc-swr-me-east-1-multiaz

Region Name	VPC Endpoint Service for SWR	OBS Bucket for SWR
AF-Johannesburg	com.myhuaweicloud.af-south-1.swr	Primary bucket: op-svc-swr-b051-10.21.20.226 Standby bucket: op-svc-swr-b051-10-21-20-226-3az
LA-Mexico City2	com.myhuaweicloud.la-north-2.swr	Primary bucket: op-svc-swr-b051-10.57.26.156 Standby bucket: op-svc-swr-la-north-2-multiaz
AP-Singapore	com.myhuaweicloud.ap-southeast-3.swr	Primary bucket: op-svc-swr-b051-10-38-34-172-3az Standby bucket: op-svc-swr-ap-southeast-3-backup
CN East-Shanghai1	com.myhuaweicloud.cn-east-3.swr	Primary bucket: op-svc-swr-b051-10-147-7-14-3az Standby bucket: op-svc-swr-cn-east-3-backup
CN North-Beijing4	com.myhuaweicloud.cn-north-4.swr	Primary bucket: op-svc-swr-b051-10-38-19-62-3az Standby bucket: op-svc-swr-cn-north-4-backup
CN North-Ulanqab1	com.myhuaweicloud.cn-north-9.swr	Primary bucket: op-svc-swr-b051-10.67.13.15 Standby bucket: op-svc-swr-cn-north-9-backup
CN North-Guangzhou-InvitationOnly	com.myhuaweicloud.cn-south-4.swr	Primary bucket: op-svc-swr-b051-26.0.9.255 Standby bucket: op-svc-swr-cn-south-4-backup
CN North3	com.myhuaweicloud.cn-north-12.swr	op-svc-swr-cn-north-12-multiaz
LA-Sao Paulo1	com.myhuaweicloud.sa-brazil-1.swr	op-svc-swr-b051-10-22-233-67
Automotive II	com.myhuaweicloud.cn-southwest-3.swr	op-svc-swr-cn-southwest-3-multiaz
Qingdao	com.myhuaweicloud.cn-east-5.swr	op-svc-swr-cn-east-5-multiaz

Region Name	VPC Endpoint Service for SWR	OBS Bucket for SWR
CN-Hong Kong	com.myhuaweicloud.ap-southeast-1.swr	Primary bucket: op-svc-swr-b051-10.229.19.28 Standby bucket: op-svc-swr-ap-southeast-1-backup
AP-Bangkok	com.myhuaweicloud.ap-southeast-2.swr	Primary bucket: op-svc-swr-b051-10.17.19.92 Standby bucket: op-svc-swr-ap-southeast-2-backup
AP-Manila	com.myhuaweicloud.ap-southeast-5.swr	op-svc-swr-ap-southeast-5-multiaz
AF-Cairo	com.myhuaweicloud.af-north-1.swr	op-svc-swr-af-north-1-multiaz
TR-Istanbul	com.myhuaweicloud.tr-west-1.swr	op-svc-swr-b051-26.174.223.88
AP-Jakarta	com.myhuaweicloud.ap-southeast-4.swr	op-svc-swr-b051-26.130.169.73
LA-Santiago	com.myhuaweicloud.la-south-2.swr	op-svc-swr-b051-10.53.5.19

**Step 3** Click **Next**.

**Step 4** Confirm the order details and click **Submit**.

----End

## 6.3 Public Network Access

### Scenarios

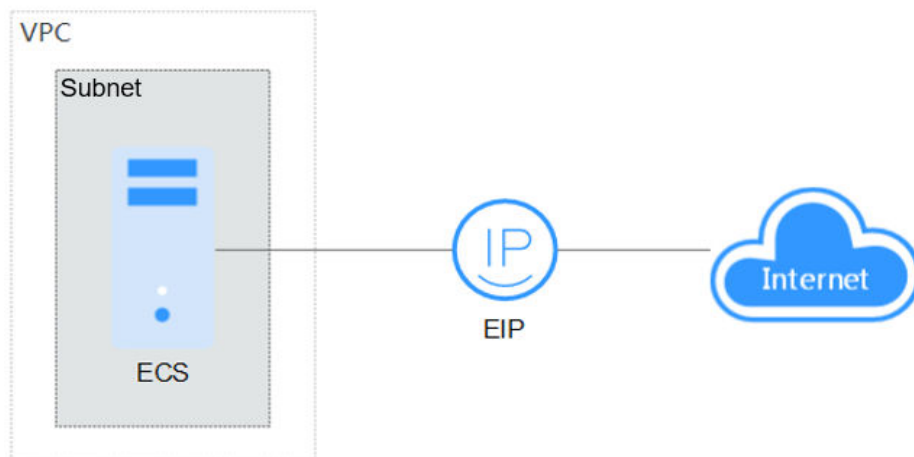
If your container engine client is installed on a CCE node or an ECS that is in a different region from the image repository, you can push or pull images over a public network. An EIP needs to be bound to the CCE node or ECS. For public network access, there are two scenarios.

- [Public Network Access for Single ECS](#)
- [Public Network Access for Multiple ECSs](#)

### Public Network Access for Single ECS

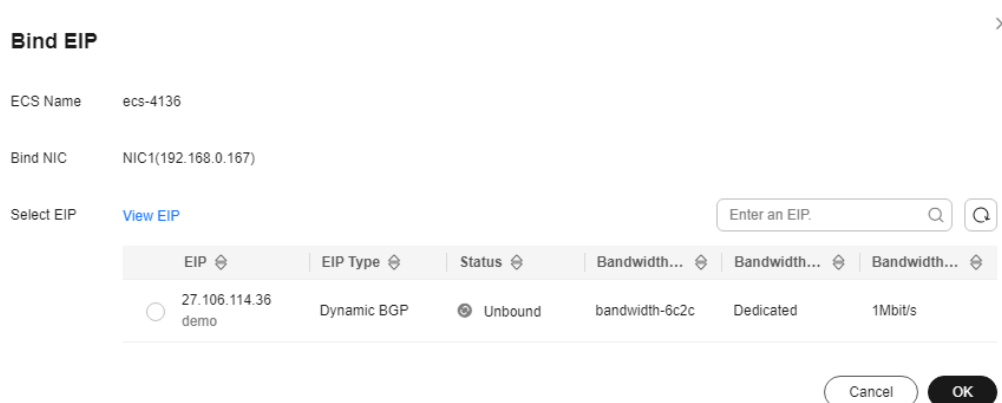
If an ECS needs to access a public network, you can bind it to an EIP. Huawei Cloud provides multiple billing modes (such as pay-per-use and pay-per-traffic). You can select one as required and flexibly unbind an unnecessary EIP.

**Figure 6-1** Network topology



- Step 1** Log in to the [ECS console](#).
- Step 2** 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 3** Select an EIP and click **OK**.

**Figure 6-2** Binding an EIP



- Step 4** After the ECS is bound to the EIP, you can view the bound EIP in the ECS list.

**NOTE**

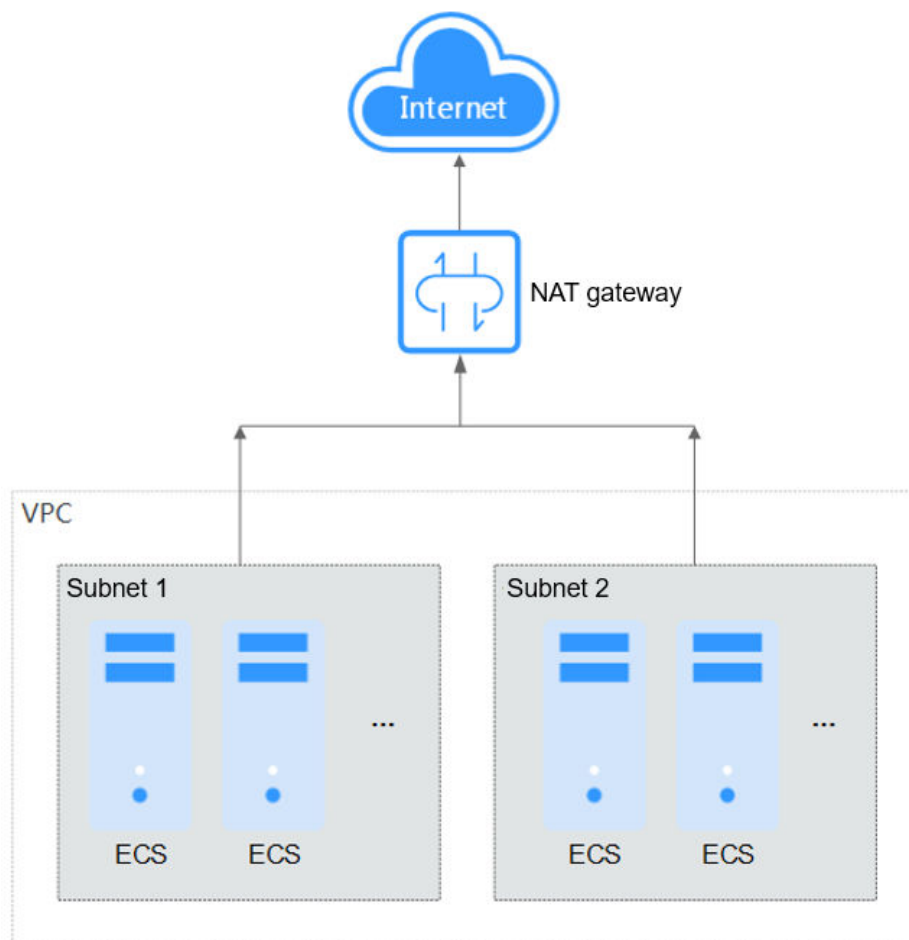
If no EIP is available in the current region, the EIP list is empty. In this case, purchase an EIP and bind it again.

----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 6-3** Network topology



**Step 1** Bind the ECS to an EIP. For details, see [Public Network Access for Single ECS](#).

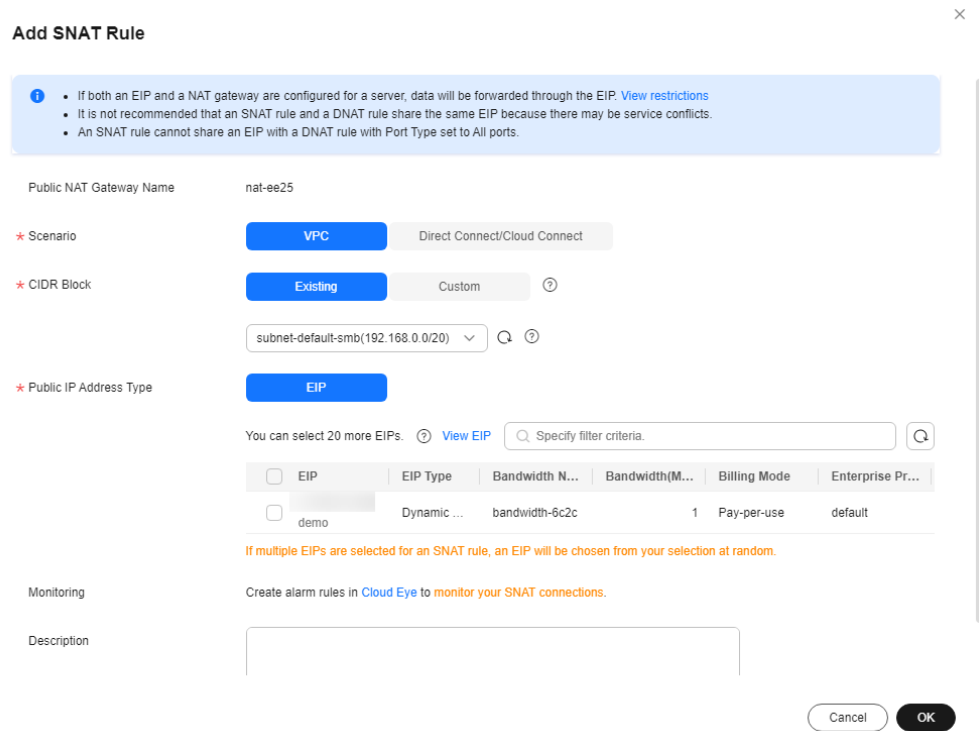
**Step 2** Create a NAT gateway. For details, see [Buying a Public NAT Gateway](#).

1. Log in to the [NAT console](#).
2. On the displayed page, click **Buy Public NAT Gateway**.
3. Configure the parameters as prompted.

**Step 3** Configure SNAT rules and bind EIPs to subnets. For details, see [Adding an SNAT Rule](#).

1. Log in to the [NAT console](#).
2. On the displayed page, click the name of the NAT gateway for which you want to add the SNAT rule.
3. On the **SNAT Rules** tab, click **Add SNAT Rule**.
4. Configure the parameters as prompted.

**Figure 6-4** Adding an SNAT rule



----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.

## 6.4 VPN/Direct Connect Access

### Scenarios

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 applies only to pushing images. To pull images, you also need to [configure a VPC endpoint for accessing the private IP address of OBS](#).

### 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.

1. Log in to the [VPC Endpoint console](#).
2. On the displayed page, click **Buy VPC Endpoint**.
3. Configure the parameters as prompted. In **Service List**, select **com.myhuaweicloud.\*\*\*\*\*.SWR** or **com.myhuaweicloud.\*\*\*\*\*.swr.\*\*\*\*\***. \*\*\*\*\* indicates the region ID.

 **NOTE**

In some regions, the VPC endpoint service for SWR is not displayed in the **Service List**. You can search for it by name.

4. Click **Next**.
5. Confirm the order details and click **Submit**.

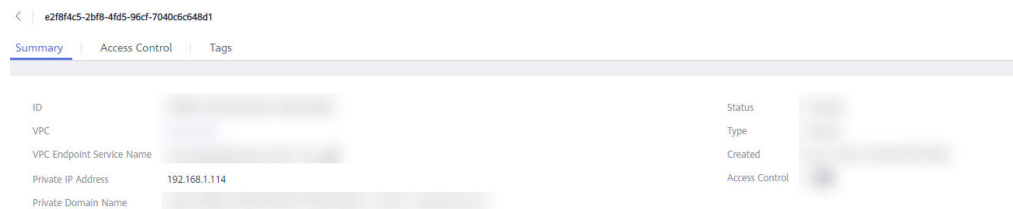
**Step 4** Obtain the private IP address and domain name for accessing the VPC.

 **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.

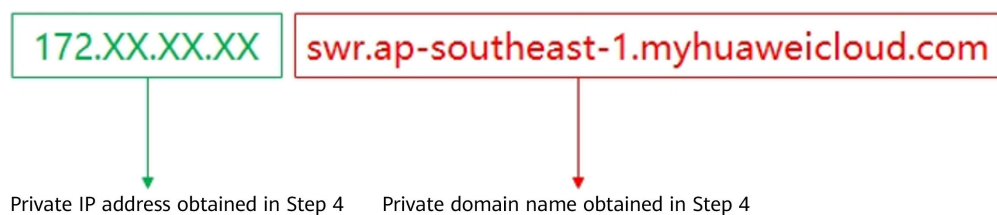
1. Go to the VPC endpoint list.
2. Locate the purchased VPC endpoint and click the ID to go to the details page.
3. On the page displayed, you can view the IP address and private domain name of the VPC endpoint.

**Figure 6-5** Endpoint details page



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

**Figure 6-6** Example hosts



 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 domain name.

There are two configuration methods:

**Configuring the hosts file in the Linux OS**

**Configuring the on-premises DNS server**

- **Configuring the hosts file in the Linux OS**

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`

- **Configuring the on-premises DNS server**

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

The methods of configuring DNS forwarding rules vary depending on OSs. 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 domain name obtained in **Step 4**.

- b. Configure forward DNS resolution. Create a file **/var/named/swr.xx-xx.myhuaweicloud.com.zone** mentioned in **Step 5.2.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:

```
ping swr.xx-xx.myhuaweicloud.com
```

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

----End

# 7 Migrating Container Images

---

## 7.1 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 7-1** Comparison of migration solutions and application scenarios

Solution	Application Scenario	Precautions
Migrating images to SWR using Docker commands	A few images	<ul style="list-style-type: none"> <li>● Disk storage leads to the timely deletion of local images and time-consuming flushing.</li> <li>● Docker daemon strictly restricts the number of concurrent pull/push operations, so high-concurrency synchronization cannot be performed.</li> <li>● Scripts are complex because HTTP APIs are needed to perform the operations that cannot be implemented through Docker CLI.</li> </ul>

Solution	Application Scenario	Precautions
Migrating images to SWR using image-syncer	A large number of images	<ul style="list-style-type: none"> <li>● Images can be synchronized from multiple source repositories to multiple destination repositories.</li> <li>● Docker Registry V2-based image repositories (such as Docker Hub, Quay, and Harbor) can be migrated to SWR.</li> <li>● Memory- and network-dependent synchronization is fast.</li> <li>● A file records the blob information about synchronized images to avoid repeated synchronization.</li> <li>● You can modify the number of concurrent synchronization tasks in a configuration file.</li> <li>● Automatic retries of failed synchronization tasks can resolve most image synchronization issues caused by network jitter.</li> <li>● Docker or other programs are not required.</li> </ul>
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.

## 7.2 Migrating Images to SWR Using Docker Commands

### Scenarios

SWR provides easy-to-use image hosting and efficient distribution services. If a 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 destination 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 destination 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

## 7.3 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](#).

### Prerequisites

Public network access is supported. You can bind an EIP for public network access.

### Procedure

**Step 1** [Download image-syncer](#). Then, decompress and run it.

The following uses image-syncer v1.5.0 as an example.

```
wget https://github.com/AliyunContainerService/image-syncer/releases/download/v1.5.0/image-syncer-v1.5.0-linux-amd64.tar.gz
```

```
tar -zxvf image-syncer-v1.5.0-linux-amd64.tar.gz
```

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

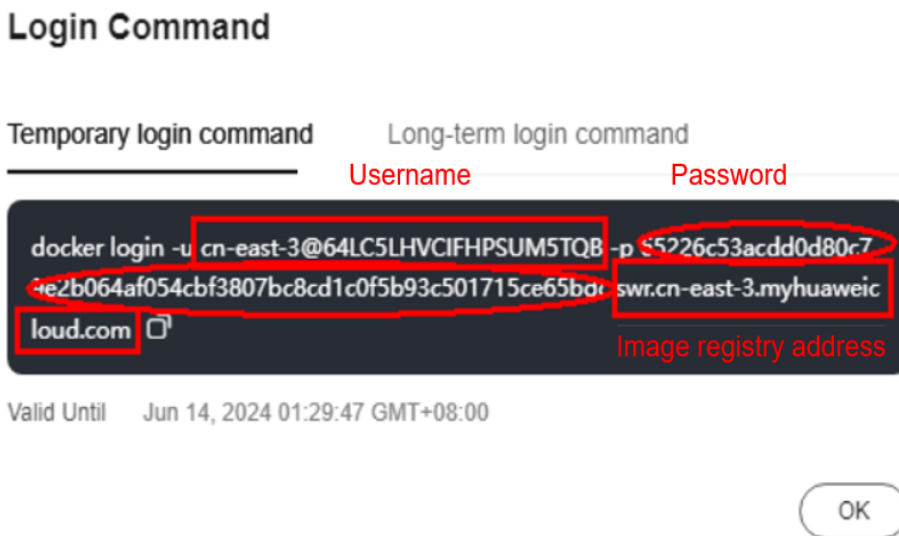
image-syncer can migrate Docker Registry V2-based repositories. Write the authentication information of the source and destination registries. The following is an example:

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

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

Log in to the [SWR console](#). In the upper right corner of the **Dashboard** page, click **Login Command** and obtain the login command.

**Figure 7-1** Generating a login command



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

In the following example, the source registry address is on the left, and the destination registry 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 7-2** 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 <b>config.json</b> file in the current directory. (This parameter can be replaced with parameters <b>--auth</b> and <b>--images</b> 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 <b>images.json</b> 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 <b>auth.json</b> 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 <b>cat</b> to check logs.
--namespace	default-namespace. default-namespace can also be set by environment variable <b>DEFAULT_NAMESPACE</b> . If both the parameter and environment variable are set, the parameter will take precedence. default-namespace will work only if the destination registry is empty but the default-registry is not empty.
--proc	Number of goroutines. The default value is <b>5</b> . You are advised to use this value and do not change it.
--retries	Number of retries. The default value is <b>2</b> . 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 <b>DEFAULT_REGISTRY</b> . If both the parameter and environment variable are set, the parameter will take precedence. default-registry will work only if the destination registry is empty but the default-namespace is not empty.

**Step 5** Log in to the destination image registry to check the migrated images.

----End

## 7.4 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 registries 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.

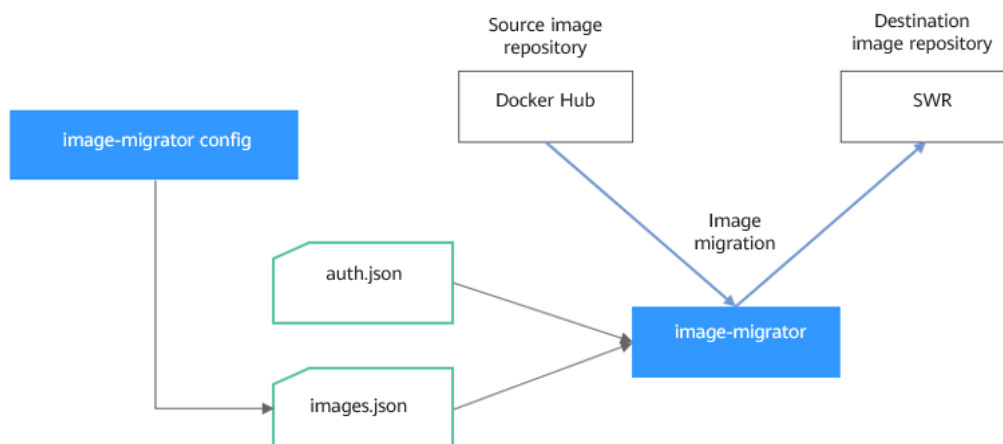
Before using image-migrator in Linux, run the **chmod u+x *Tool name*** command (for example, **chmod u+x image-migrator-linux-amd64**) to grant the execute permission.

**Table 7-3** 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: <a href="https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-linux-amd64">https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-linux-amd64</a> Linux Arm: <a href="https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-linux-arm64">https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-linux-arm64</a> Windows: <a href="https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-windows-amd64.exe">https://ucs-migration.obs.cn-north-4.myhuaweicloud.com/toolkits/image-migrator-windows-amd64.exe</a>
----------------	--	--

## How image-migrator Works

**Figure 7-2** 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.json** 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 based on the AK and SK. For details, see [Obtaining a Long-Term 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:

- The largest unit that can be synchronized using one rule is a repository. An entire namespace or registry cannot be synchronized using one rule.
- The formats of the source and destination repositories are similar to those of the image URL used by the **docker pull** or **docker push** command (*registry/namespace/repository:tag*).
- Both the source and destination repositories (if the destination repository is not an empty string) contain at least *registry/namespace/repository*.

- d. The source repository cannot be empty. To synchronize data from one source repository to multiple destination repositories, 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 any tag, all tags of the repository will be synchronized to the destination repository. In this case, the destination repository field cannot contain a tag.
- g. If the source repository field contains a tag, only that tag in the source repository will be synchronized to the destination repository. If the destination repository does not contain a tag, the source tag will be used at the destination 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 **config**, a subcommand of `image-migrator`, to automatically obtain the images that are being used by workloads in a 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 retries allowed 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 **image-migrator** 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 kubeconfig files of kubectl. The default value is **\$HOME/.kube/config**. A kubeconfig file is used to configure access to a Kubernetes cluster. The file contains the authentication credential and endpoint (address) required for accessing a 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 registry 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 one 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 a registry access permission file **auth.json**.

Create an **auth.json** file and modify it based on the required format. If a registry 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](#).

### Step 2 Prepare an 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 of image-migrator to generate an **images.json** file.

You can refer to the methods and examples in [Usage of image-migrator config](#) to obtain the images used in the source cluster applications without specifying a namespace, or by specifying one or multiple namespaces.

3. Modify the `images.json` file to make it meet the requirements in [images.json](#).

### 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 results.

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

## 7.5 Migrating Images Across Clouds from Harbor to SWR

### Scenarios

If you are using a self-built Harbor registry on other cloud, you can replicate images from Harbor to SWR.

1. If Harbor can access SWR over the Internet, perform the replication as instructed in [Accessing SWR over the Internet](#).
2. If Harbor accesses SWR through a VPC endpoint by using a private line, perform the replication as instructed in [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.

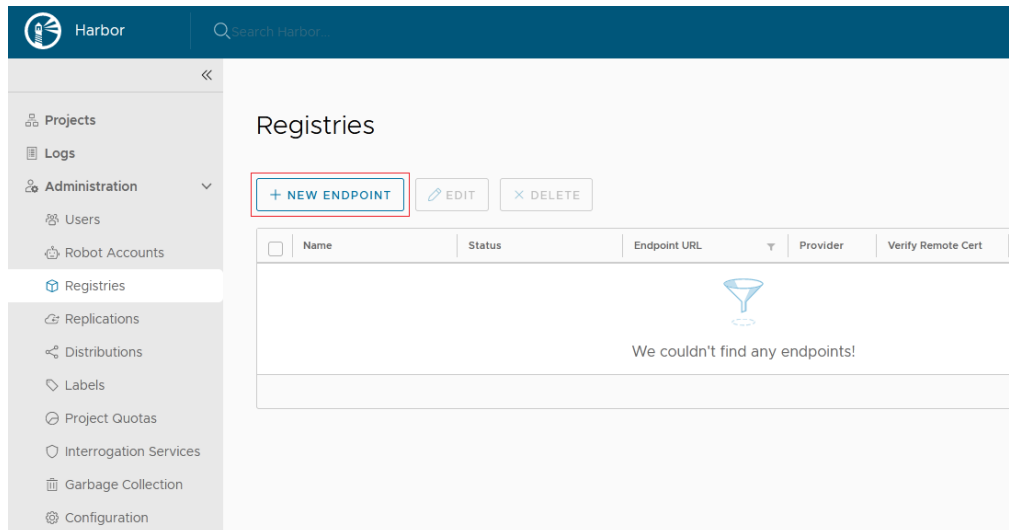
### Accessing SWR over the Internet

- Step 1 Configure a registry endpoint on Harbor.

 **NOTE**

Harbor 1.10.5 and later can interconnect with Huawei Cloud SWR. You only need to set **Provider** to **Huawei SWR** when configuring an endpoint on Harbor. Harbor 2.4.1 is used as an example.

1. Add an endpoint.



2. Configure the following parameters.

## New Registry Endpoint

The screenshot shows a configuration form for a new registry endpoint. The fields are as follows:

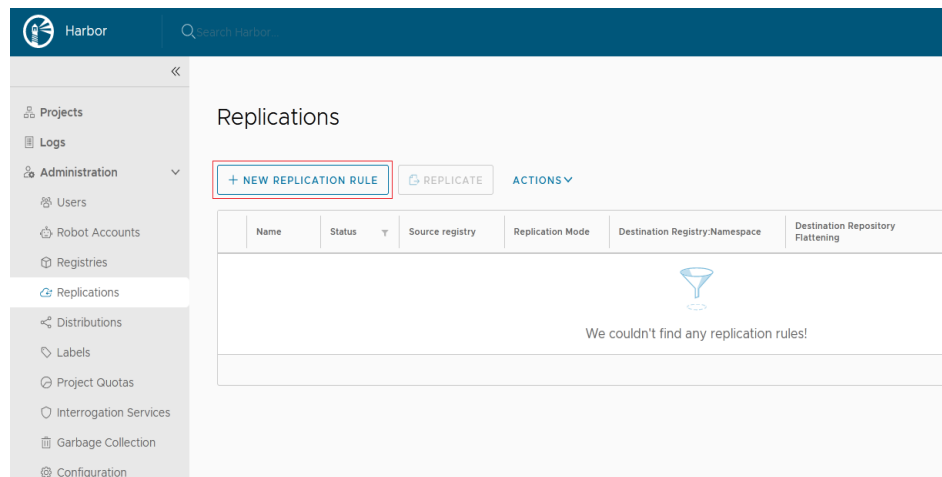
- Provider \***: A dropdown menu with "Huawei SWR" selected.
- Name \***: A text input field containing "test".
- Description**: An empty text area.
- Endpoint URL \***: A text input field containing "https://swr...myhuaweicloud".
- Access ID**: A text input field containing "@CCRJUTQ7QG".
- Access Secret**: A text input field with the content masked by dots.
- Verify Remote Cert (i)**: A checkbox that is currently unchecked.

At the bottom of the form, there are three buttons: "TEST CONNECTION", "CANCEL", and "OK".

- **Provider**: Select **Huawei SWR**.
- **Name**: Enter a name for the endpoint.
- **Endpoint URL**: Enter the public network domain name of SWR in the format of **https://{SWR image registry address}**. To obtain the image registry address, log in to the **SWR console**, choose **My Images** in the navigation pane, and then click **Upload Through Client**. On the page that is displayed, you can view the image registry address in the current region.
- **Access ID**: Enter an access ID in the format of *Regional project name@[AK]*.
- **Access Secret**: Enter an access secret generated from a pair of AK and SK. **Access Secret** will be used together with **Access ID** to generate a long-term login command. For details, see **Obtaining a Long-Term 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 name for the rule.
- **Replication mode:** Select **Push-based**, indicating that images are pushed from the local Harbor to a remote registry.
- **Source resource filter:** Images in Harbor will be filtered to replicate based on the configured rule.
- **Destination registry:** Select the endpoint created in [Step 1](#).
- **Destination Namespace:** Enter an organization of SWR.

---

**CAUTION**

If the **Namespace** value contains a slash (/), for example, **ns1/test**, Huawei SWR considers **ns1** as the organization name and **test** as the path to the image. So **ns1/test** means the image name under the **ns1** organization is **test/\***.

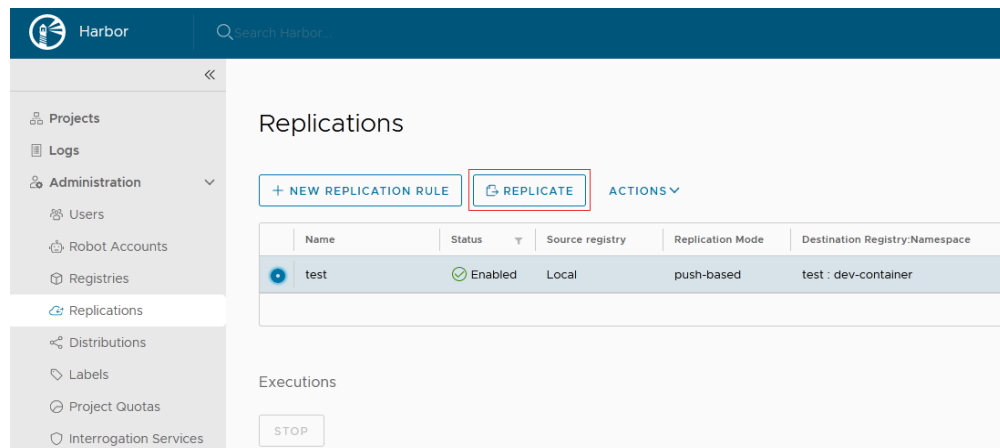
---

**Flattening:** Select **Flatten All Levels** to remove all hierarchy from the replicated images. For example, if the hierarchy of Harbor is **library/nginx** and the destination namespace is **dev-container**, **library/nginx** replicates to **dev-container/nginx**.

- **Trigger Mode:** Select **Manual**.
- **Bandwidth:** Set the maximum network bandwidth for each replication task. **-1** indicates unlimited bandwidth.

For more information, see the [Harbor official website](#).

**Step 3** Select the replication rule and click **REPLICATE** to replicate images.



----End

## Accessing SWR Through a VPC Endpoint by Using a Private Line

**Step 1** Configure a VPC endpoint.

**Step 2** Obtain the private 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.

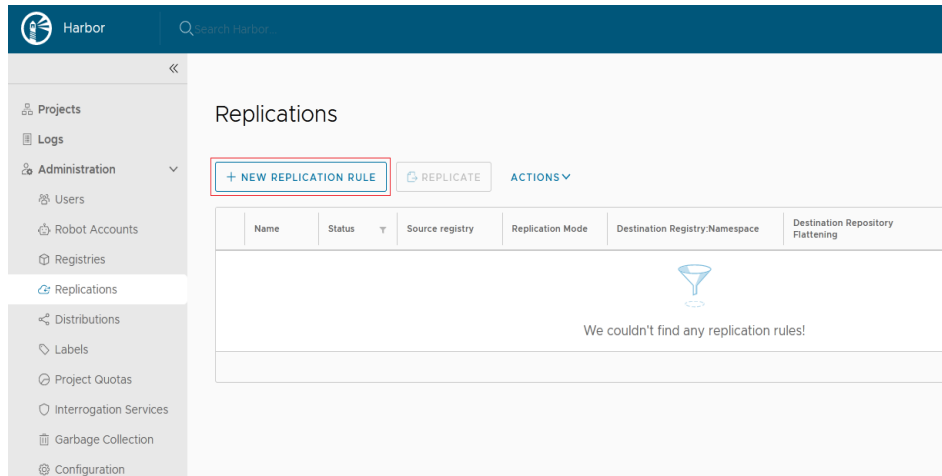
### NOTE

Harbor 1.10.5 and later can interconnect with Huawei Cloud SWR. You only need to set **Provider** to **Huawei SWR** when configuring an endpoint on Harbor. Harbor 2.4.1 is used as an example.

1. Add an endpoint.
2. Configure the following parameters.
  - **Provider:** Select **Huawei SWR**.
  - **Name:** Enter a name for the endpoint.
  - **Endpoint URL:** Enter the private domain name of a 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 access secret generated from a pair of AK and SK. **Access Secret** will be used together with **Access ID** to generate a long-term login command. For details, see [Obtaining a Long-Term Login Command](#).
  - **Verify Remote Cert:** Deselect the option.

**Step 4** Configure a replication rule.

1. Create a replication rule.



2. Configure the following parameters.

## New Replication Rule

**Name \***

**Description**

**Replication mode**  Push-based ⓘ  Pull-based ⓘ

**Source resource filter**

Name:  ⓘ

Tag:  ⓘ

Label:  ⓘ

Resource:  ⓘ

**Destination registry \***  ⓘ

**Destination**

Namespace:  ⓘ

Flattening:  ⓘ

**Trigger Mode \***  ⓘ

**Bandwidth \***  Kbp:  ⓘ

Override ⓘ

- **Name:** Enter a name for the rule.
- **Replication mode:** Select **Push-based**, indicating that images are pushed from the local Harbor to a remote registry.
- **Source resource filter:** Images in Harbor will be filtered to replicate based on the configured rule.
- **Destination registry:** Select the endpoint created in [Step 3](#).
- **Destination Namespace:** Enter an organization of SWR.

**CAUTION**

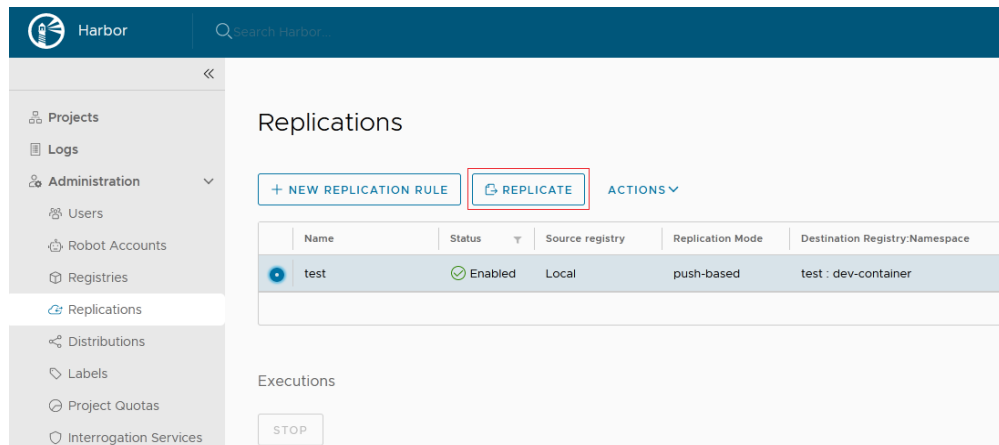
If the **Namespace** value contains a slash (/), for example, **ns1/test**, Huawei SWR considers **ns1** as the organization name and **test** as the path to the image. So **ns1/test** means the image name under the **ns1** organization is **test/\***.

**Flattening:** Select **Flatten All Levels** to remove all hierarchy from the replicated images. For example, if the hierarchy of Harbor is **library/nginx** and the destination namespace is **dev-container**, **library/nginx** replicates to **dev-container/nginx**.

- **Trigger Mode:** Select **Manual**.
- **Bandwidth:** Set the maximum network bandwidth for each replication task. **-1** indicates unlimited bandwidth.

For more information, see the [Harbor official website](#).

**Step 5** Select the replication rule and click **REPLICATE** to replicate images.



----End

# 8 SWR Security Best Practices

---

Security is a shared responsibility between Huawei Cloud and you. Huawei Cloud ensures the security of cloud services. As a tenant, you should take advantage of the security capabilities provided by Huawei Cloud to protect your data and use the cloud securely. For details, see [Shared Responsibilities](#).

This document provides guidance for enhancing the overall security of SoftWare Repository for Container (SWR). You can continuously evaluate the security of your SWR resources and enhance their overall defensive capabilities by combining different security capabilities provided by SWR. By doing this, images stored in SWR can be protected from leakage and tampering both at rest and in transit.

Consider the following aspects for your security configurations:

- [Scan images to detect container image vulnerabilities and security risks in advance, reducing the risk of attacks.](#)
- [Strengthen permissions management to reduce related risks.](#)
- [Enable image access audit for post-event backtracking.](#)

## Scanning Images to Detect Container Image Vulnerabilities and Security Risks in Advance

You are advised to [scan images](#) stored in SWR periodically. You can scan system vulnerabilities, application vulnerabilities, malicious files, software information, file information, baseline configurations, weak passwords, sensitive information, software compliance, and basic image information to identify and fix potential risks. This ensures that all images deployed in the production environment have passed strict security checks, ensuring that the system and applications can run securely and stably.

## Strengthening Permissions Management to Reduce Related Risks

1. Do not allow IAM users to access SWR using administrator permissions. Create Huawei Cloud IAM users and grant them [access permissions](#) on different container images to isolate permissions between employees.
2. Access SWR through a VPC endpoint for refined control over different image organizations.

Accessing SWR through a VPC endpoint enables refined data boundary control over SWR. You can configure the VPC endpoint and IAM policies to

control the push and pull permissions in a specified VPC and allow only images in a fixed organization to be pulled in that VPC.

## **Enabling Image Access Audit for Post-event Backtracking**

The audit function records all user operations on SWR in real time. By recording, analyzing, and reporting user access to SWR, the audit function helps you generate compliance reports and trace the root cause of an accident, improving data asset security. For details, see [Viewing Logs in CTS](#).

# 9 CCE Clusters Pull Images from SWR Basic Edition Without Passwords

Each time a CCE cluster pulls an image from SWR Basic Edition, the username and password must be provided for authentication. CCE clusters store these credentials in secrets. However, when you deploy a workload, you still need to manually configure **imagePullSecrets** for each workload to pull images.

To simplify the deployment process and reduce the complexity of manual operations, CCE provides a solution that does not require **imagePullSecrets**. This solution makes image pull automated and more convenient.

## Procedure

**Step 1** Use **kubectl to connect to the cluster**. Run the following command to create the service account **swr-service-account** in namespace **test-namespace**:

```
kubectl create serviceaccount swr-service-account -n test-namespace
```

### NOTE

- A service account is also called a ServiceAccount.
- If you already have a service account or you want to use the default service account, skip this step.

**Step 2** Run the following command to associate **default-secret** of CCE with the service account created in **Step 1**.

```
kubectl patch serviceaccount swr-service-account -p '{"imagePullSecrets": [{"name": "default-secret"}]}' -n test-namespace
```

In this command,

- **test-namespace** is the namespace of CCE. You need to specify the namespace images can be pulled for application deployment without a password.
- **swr-service-account** is the name of the service account created in **Step 1**. If you use the default service account, replace **swr-service-account** in the command with **default**.

**Step 3** Use the YAML file below to create a workload and set **ServiceAccountName** to the service account created in **Step 1**. If you use the default service account, you do not need to set **ServiceAccountName**.

```
apiVersion: apps/v1  
kind: Deployment
```

```
metadata:
  name: nginx
  namespace: test-namespace # Specify the namespace where CCE can pull images without a password.
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      serviceAccountName: swr-service-account #Specify the service account associated with default-secret
of CCE.
  containers:
    - name: nginx
      image: swr.cn-north-4.myhuaweicloud.com/nginx/nginx:latest # Specify the SWR image path.
      ports:
        - containerPort: 80
```

**Step 4** Verify that the configuration is valid.

Verify that an image can be pulled when you deploy a workload by following the instructions in [Step 3](#). After a period of time, the workload status changes to **Running**.

----End