Cloud Container Instance

FAQs

Issue	01
Date	2020-10-26

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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.

Contents

1 Basic Concept FAQs 1
1.1 What Is CCI? 1
1.2 What Are the Differences Between a Cloud Container Instance and a Cloud Container Engine? 2
1.3 What Is an Environment Variable? 2
1.4 What Is a Service? 2
1.5 What Is Mcore? 2
1.6 What Are the Relationships Between Images, Containers, and Workloads? 3
1.7 What Is a Secure Container? 5
1.8 Can kubectl Be Used to Manage Container Instances? 5
2 Workload Abnormalities 6
2.1 Locating Method 6
2.2 What Do I Do If an Event Indicating That the Image Failed to Be Pulled Occurs? 6
2.3 What Do I Do If an Event Indicating That the Container Failed to Be Restarted Occurs? 8
3 Container Workload FAQs 11
3.1 How Do I Set the Quantity of Instances (Pods)? 11
3.2 How Do I Set Probes for a Workload? 12
3.3 How Do I Configure an Auto Scaling Policy? 12
3.4 What Do I Do If the Workload Created from the sample Image Fails to Run? 12
3.5 How Do I View Pods After I Call the API to Delete a Deployment? 12
3.6 Why an Error Is Reported When a GPU-Related Operation Is Performed on the Container Entered by Using exec? 13
4 Image Repository FAQs 14
4.1 Can I Export Public Images? 14
4.2 How Do I Create a Container Image? 14
4.3 How Do I Upload Images? 15
4.4 Does CCI Provide Base Container Images for Download? 15
4.5 Does CCI Administrator Have the Permission to Upload Image Packages? 15
4.6 What Do I Do If Authentication Is Required During Image Push? 15
5 Network Management FAQs 17
5.1 How Do I View the VPC CIDR Block? 17
5.2 Does CCI Support Load Balancing? 17
5.3 How Do I Configure the DNS Service on CCI? 18

5.4 Does CCI Support InfiniBand (IB) Networks? 18
5.5 How Do I Access a Container from a Public Network? 19
5.6 How Do I Access a Public Network from a Container? 19
5.7 What Do I Do If Access to a Workload from a Public Network Fails? 19
5.8 What Do I Do If Error 504 Is Reported When I Access a Workload? 20
5.9 What Do I Do If the Connection Timed Out? 21

6 Storage Management FAQs 22

6.1 Which Cloud Storage Types Are Supported and Which Types Require Backup? 22
6.2 How Do I Use Cloud Storage? 22
6.3 Where Is the Data Generated by a Container Stored If No Cloud Storage Is Mounted? 22
6.4 Why Volume Mounting Failure Is Reported When a Job Pod Completes Its Task? 23

1 Basic Concept FAQs

1.1 What Is CCI?

Cloud Container Instance (CCI) is a serverless container engine that allows you to run containers without creating or managing server clusters.

With the serverless architecture, you can focus on building and operating applications without having to create or manage servers, or worrying about server health. All you have to do is to specify resource requirements (such as the required CPU cores and memory space). This gives you a more focused approach to business needs and helps you reduce management and maintenance costs. Traditionally, to run containerized workloads using Kubernetes, you need to create a Kubernetes server cluster first. That is not the case with CCI. Under the serverless architecture, CCI allows you to directly create and use containerized workloads by using the console, kubectl, or Kubernetes APIs, and pay only for the resources consumed by these workloads.

CCI provides the following functions:

- Automated continuous delivery (CD)

  Verifies every container image change by running the new container images in just a few clicks.
- Full hosting of workloads during runtime

  Provides hosting for Deployments to ensure stable running.
- Fast auto scaling

  Allows you to customize auto scaling policies and scales resources automatically within 1s.
- High Availability (HA) assurance for applications

  Allows concurrent running of multiple instances and provides global load balancing.
- Monitoring of container status

  Provides health checks for containers and monitors container metrics in real time.
- Persistent data storage

Supports mounting of network storage volumes to persist service data.

1.2 What Are the Differences Between a Cloud Container Instance and a Cloud Container Engine?

HUAWEI CLOUD provides enterprises with two CNCF-certified Kubernetes services that feature high-performance, high-availability, and high-security: Cloud Container Engine (CCE) and Cloud Container Instance (CCI).

CCE is a high-performance, high-reliability service through which enterprises can deploy, manage, and scale containerized applications on HUAWEI CLOUD. It supports container runtime environment. CCE is a one-stop container platform that provides full-stack container services from Kubernetes cluster management, lifecycle management of containerized applications, Istio application mesh, and Helm charts to add-on management, application scheduling, and monitoring and O&M. For more details, see **CCE**.

CCI is a serverless container engine that allows you to run containers without creating and managing servers or clusters. With CCI, enterprises only need to manage containerized services running on Kubernetes. Under the serverless architecture, containerized applications are free of O&M and enterprises can focus on building and operating applications. For more details, see **CCI**.

1.3 What Is an Environment Variable?

An environment variable is a variable whose value can affect the way a running container will behave. You can modify environment variables even after workloads are deployed, increasing flexibility in workload configuration.

The effect of setting environment variables on CCI is the same as that of specifying **ENV** in a Dockerfile.

1.4 What Is a Service?

A service defines a set of instances and a means for accessing them, such as a single stable IP address and corresponding DNS name.

CCI uses service names instead of IP addresses to address communication between components. A service name is specified during workload creation.

1.5 What Is Mcore?

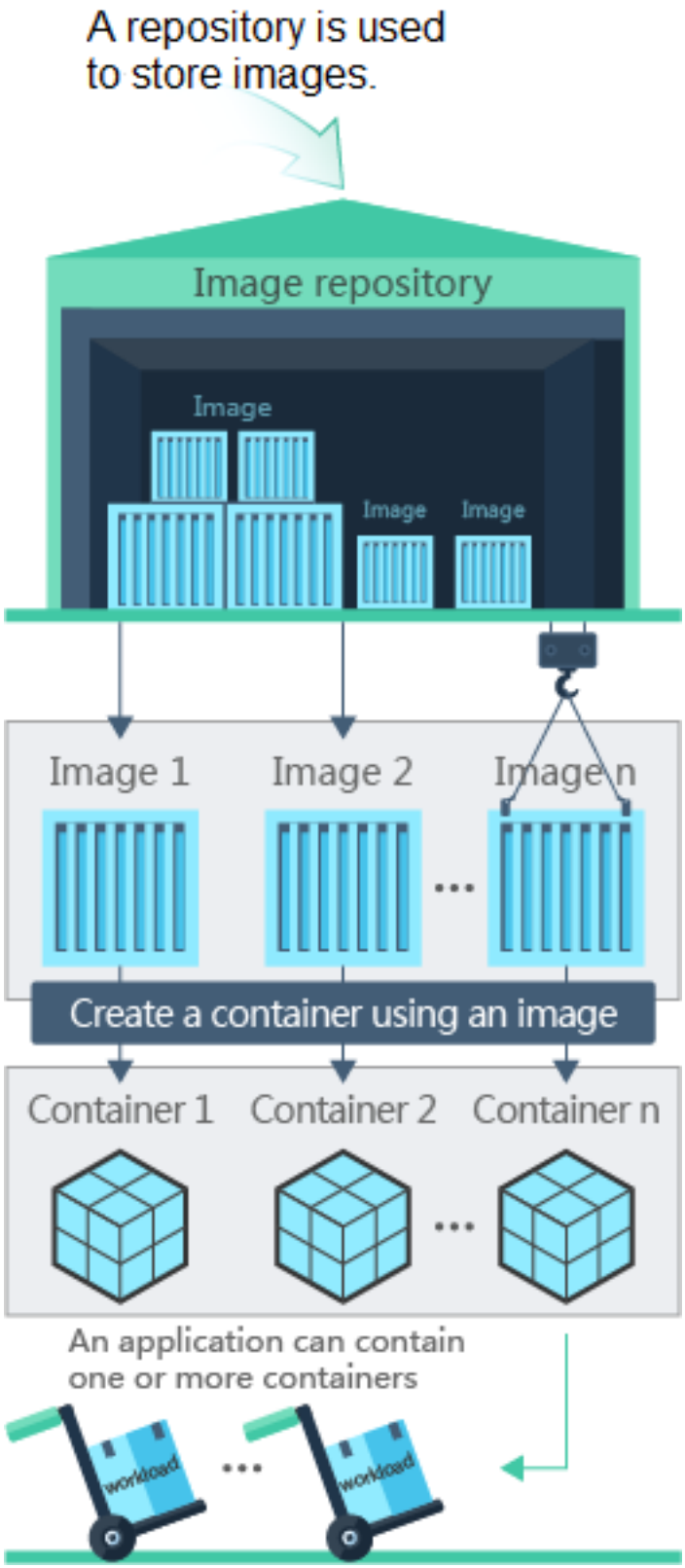
One CPU core is equal to 1000 mcores. Generally, the CPU usage of a containerized workload is measured in mcores.

1.6 What Are the Relationships Between Images, Containers, and Workloads?

- Image: A container image is a special file system that includes all the programs, libraries, resources, and configuration files for running containers. It also includes some required configuration parameters, such as anonymous volumes, environment variables, and users. An image does not contain any dynamic data, and its content remains unchanged after being built.
- Container: A container is a runtime instance of an image. The relationship between an image and a container is similar to that between a class and an instance in the object-oriented program design. A container can be created, started, stopped, deleted, or suspended.

The following figure shows the relationships between images, containers, and workloads.

Figure 1-1 Relationships between images, containers, and workloads



1.7 What Is a Secure Container?

Secure containers are distinguished from common containers in a few aspects.

The most important difference is that each secure container (pod) runs on an independent micro-VM, has an independent OS kernel, and is securely isolated at the virtualization layer. As CCI uses shared multi-tenant clusters, security isolation of containers is more stringent than that in the scenarios where users have independent, private Kubernetes clusters. Secure containers, kernels, computing resources, storage resources, and networks can be isolated between different tenants, protecting users' resources and data from being preempted or stolen by other users.

1.8 Can kubectl Be Used to Manage Container Instances?

CCI allows you to use kubectl to manage container instances. For details, see **https://support.huaweicloud.com/en-us/devg-cci/cci_kubectl_01.html**.

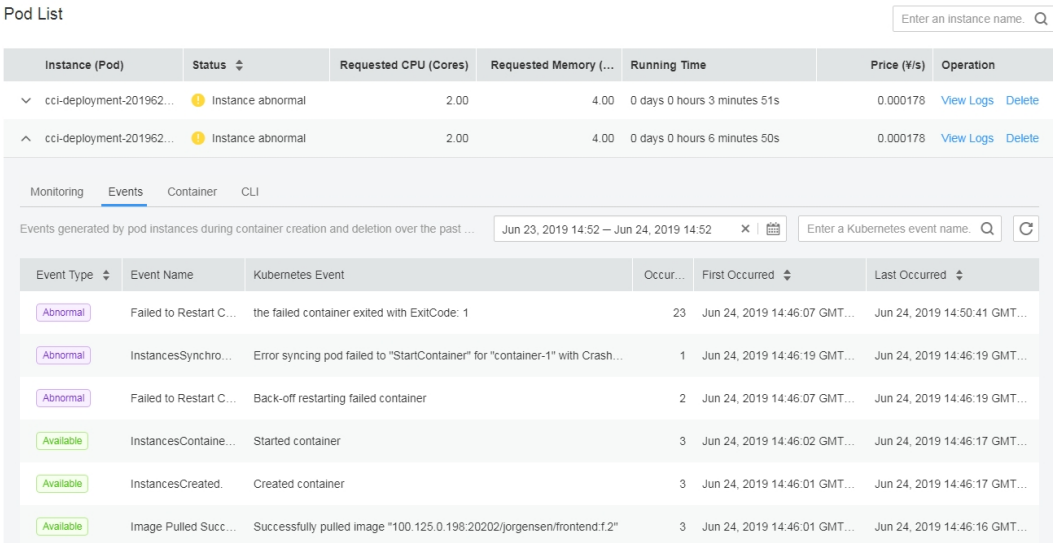
2 Workload Abnormalities

2.1 Locating Method

If a workload is faulty, check events first.

On the CCI console, click **Workloads** in the navigation pane and click the name of the faulty workload. In the **Pod List** area of the workload details page, click ⌄ at the left of the faulty pod (workload instance) to show more details about the pod. The **Events** tab page will display pod events.

Figure 2-1 Viewing events



2.2 What Do I Do If an Event Indicating That the Image Failed to Be Pulled Occurs?

If the details page of a workload shows an event indicating that the image fails to be pulled, perform the following operations to locate the fault:

Check Item 1: imagePullSecret (This Is Required When You Use kubectl to Create a Workload)

In the following example, a Deployment named **nginx** is created. Check whether the .yaml file contains the **imagePullSecrets** field (bold field in the following information), which indicates the name of the secret used for pulling the image.

To pull an image from the **SoftWare Repository for Container (SWR)**, set this field to **imagepull-secret**.

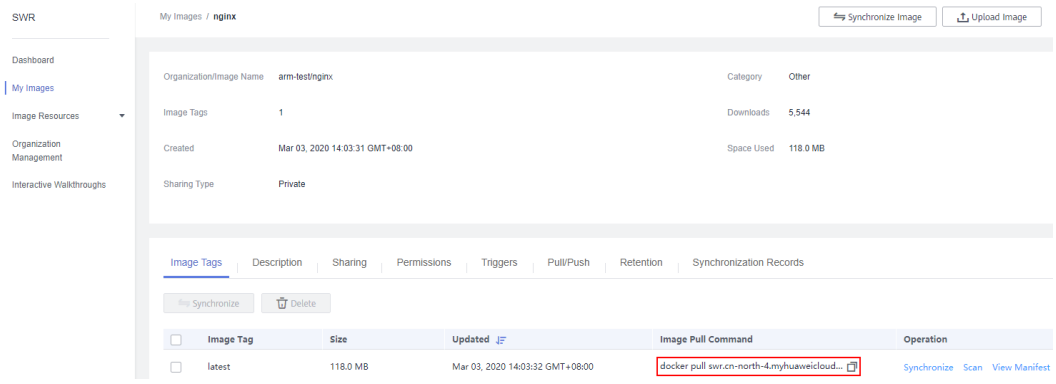
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:alpine
        imagePullPolicy: Always
        name: nginx
      imagePullSecrets:
      - name: imagepull-secret
```

Check Item 2: Image Address

CCI allows you to create workloads using images pulled from **SWR**.

SWR images can be obtained by using **Template Address**. After an image is pushed, you can obtain its address.

Figure 2-2 Template address



Check Item 2: IAM Users' Permissions to Download Images

If you have enabled the Enterprise Management service, you need to use your account to grant IAM users with permissions to access SWR so that the IAM users can download private images of the account.

You can grant permissions to IAM users in either of the following ways:

- Grant permissions on the image details page. After permissions granted, IAM users can read, edit, and manage the image. For more details, see **Granting Permissions to Users on the Image Details Page**.
- Grant permissions on the organization details page. After permissions granted, IAM users can read, edit, and manage all images in the organization. For more details, see **Granting Permissions to Users on the Organization Details Page**.

2.3 What Do I Do If an Event Indicating That the Container Failed to Be Restarted Occurs?

If the details page of a workload shows an event indicating that the container fails to be restarted, perform the following operations to locate the fault:

Check Item 1: Port Conflict

Step 1 Configure the kubectl. For details, see **Using Native kubectl (Recommended)**.

Step 2 On the CCI console, click the name of the workload whose container failed to restart. In the **Pod List** area of the workload details page, obtain the pod name.

Step 3 View the name of the failed container.

kubectl describe pod $name -n $namespace | grep "Error syncing pod failed to"

Figure 2-3 Viewing the name of the failed container

```
[root@master-1 ~]# kubectl describe pod -n cce-burst-2425753f-0b12-11e9-a771-0255ac1057f2 | grep "Error syncing pod failed to"
  Warning  FailedSync           7m               kubelet, 1ce9a08e-dd39-e911-a1fb-9835ed8fc6a9  Error syncing pod failed to "Sta
rtContainer" for "container-1" with CrashLoopBackOff: "Back-off 10s restarting failed container=container-1 pod=port-conflict-7d4f
c654bb-tc9gw_cce-burst-2425753f-0b12-11e9-a771-0255ac1057f2(c719d941-55e4-11e9-b64c-e84dd0c9d022)"
  Warning  FailedSync           6m               kubelet, 1ce9a08e-dd39-e911-a1fb-9835ed8fc6a9  Error syncing pod failed to "Sta
rtContainer" for "container-1" with CrashLoopBackOff: "Back-off 20s restarting failed container=container-1 pod=port-conflict-7d4f
c654bb-tc9gw_cce-burst-2425753f-0b12-11e9-a771-0255ac1057f2(c719d941-55e4-11e9-b64c-e84dd0c9d022)"
```

Step 4 View the logs of the container.

kubectl logs $podName -n $namespace -c $containerName

```
[root@master-1 ~]# kubectl logs port-conflict-7d4fc654bb-tc9gw -n cce-burst-2425753f
2019/04/03 07:52:42 [emerg] 24#24: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2019/04/03 07:52:42 [emerg] 24#24: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2019/04/03 07:52:42 [emerg] 24#24: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2019/04/03 07:52:42 [emerg] 24#24: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2019/04/03 07:52:42 [emerg] 24#24: bind() to 0.0.0.0:80 failed (98: Address in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)
2019/04/03 07:52:42 [emerg] 24#24: still could not bind()
```

Solution: Re-create a workload and configure correct ports to avoid port conflicts.

----End

Check Item 2: Workload Bugs

Check whether the workload startup command is correctly executed or whether the workload has a bug.

Step 1 Configure the kubectl. For details, see **Using Native kubectl (Recommended)**.

Step 2 On the CCI console, click the name of the workload whose container failed to restart. In the **Pod List** area of the workload details page, obtain the pod name.

Step 3 View the name of the failed container.

kubectl describe pod $name -n $namespace | grep "Error syncing pod failed to"

Figure 2-4 Viewing the name of the failed container

```
[root@master-1 ~]# kubectl describe pod -n cce-burst-2425753f-0b12-11e9-a771-0255ac1057f2 | grep "Error syncing pod failed to"
  Warning  FailedSync            7m                 kubelet, 1ce9a08e-dd39-e911-a1fb-9835ed8fc6a9  Error syncing pod failed to "Sta
rtContainer" for "container-1" with CrashLoopBackOff: "Back-off 10s restarting failed container=container-1 pod=port-conflict-7d4f
c654bb-tc9gw_cce-burst-2425753f-0b12-11e9-a771-0255ac1057f2(c719d941-55e4-11e9-b64c-e84dd0c9d022)"
  Warning  FailedSync            6m                 kubelet, 1ce9a08e-dd39-e911-a1fb-9835ed8fc6a9  Error syncing pod failed to "Sta
rtContainer" for "container-1" with CrashLoopBackOff: "Back-off 20s restarting failed container=container-1 pod=port-conflict-7d4f
c654bb-tc9gw_cce-burst-2425753f-0b12-11e9-a771-0255ac1057f2(c719d941-55e4-11e9-b64c-e84dd0c9d022)"
```

Step 4 View the logs of the container.

kubectl logs $podName -n $namespace -c $containerName

Fix the workload bugs indicated in logs.

Figure 2-5 Incorrect startup command configuration of the container

```
[root@master-1 ~]# kubectl logs no-running-proces
ERROR: exec failed: No such file or directory
```

Solution: Re-create a workload and configure a correct startup command.

----End

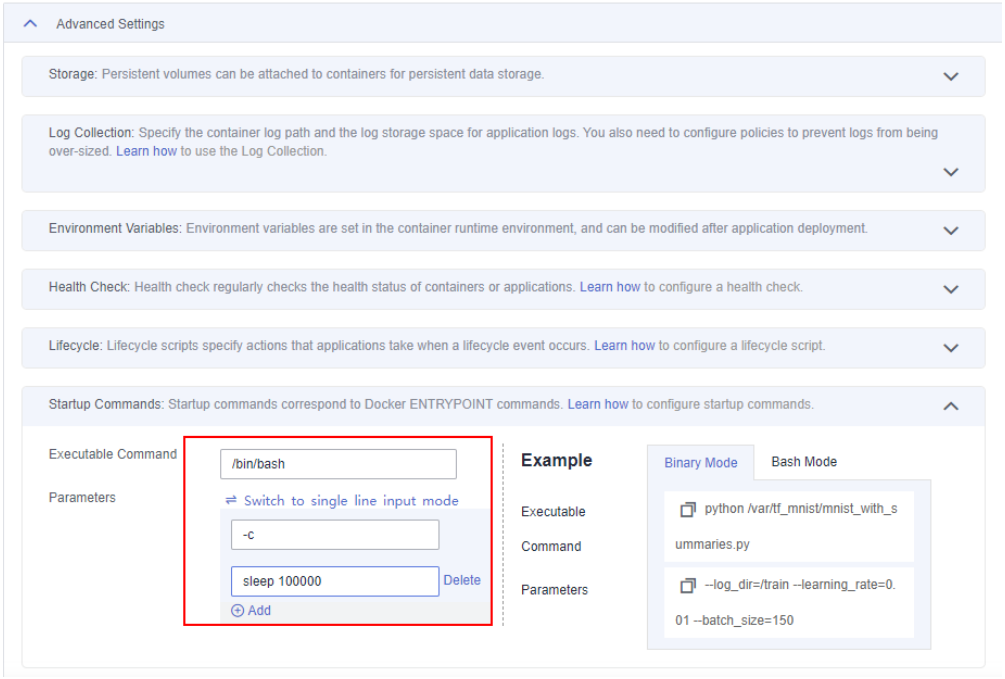
Check Item 3: Workload Health Check

If the liveness probing (a type of health check) is enabled for the workload and the number of health check failures exceeds the threshold, the workload details page shows the Kubernetes event **Liveness probe failed: .....** and containers in the workload instance will be restarted. In this case, reconfigure the health check policies.

Other Check Items

A workload fault may be caused by the failure to start the workload inside the container. In this case, you can manually run the startup command inside the pod to which the container belongs and rectify the fault based on the error message. The specific procedure is as follows:

1. Configure the following startup command for the workload. In this way, the application will not be started after the pod is started and no operation will be performed.



NOTE

Before running the startup command, ensure that the **/bin/bash** command in the image is available.

2. After the pod is started, run the **kubectl exec** command to enter the inside of the pod. Then, manually run the startup command inside the pod, and rectify the fault based on the error message.

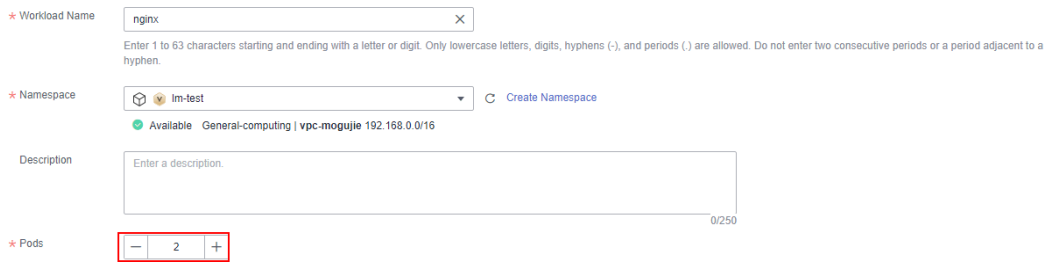
3 Container Workload FAQs

3.1 How Do I Set the Quantity of Instances (Pods)?

When Creating a Workload

Click the plus and minus signs or enter a value to set the quantity of instances (pods).

Figure 3-1 Setting the quantity of instances (pods)



After a Workload Is Created

Click the workload whose instance quantity needs to be adjusted. In the **Scaling** area on the workload details page, click **Manual Scaling** and then , and set the instance quantity.

Figure 3-2 Manual scaling

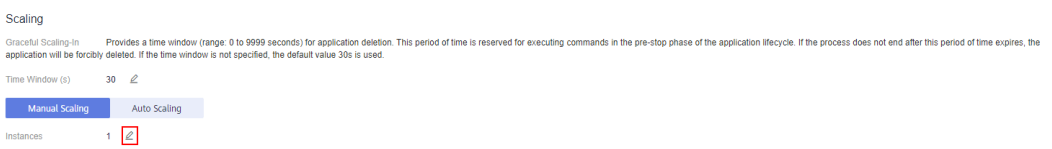
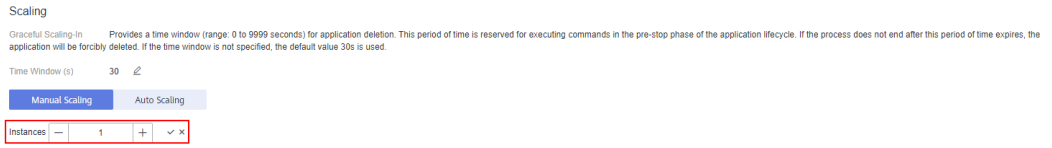


Figure 3-3 Adjusting the instance quantity



3.2 How Do I Set Probes for a Workload?

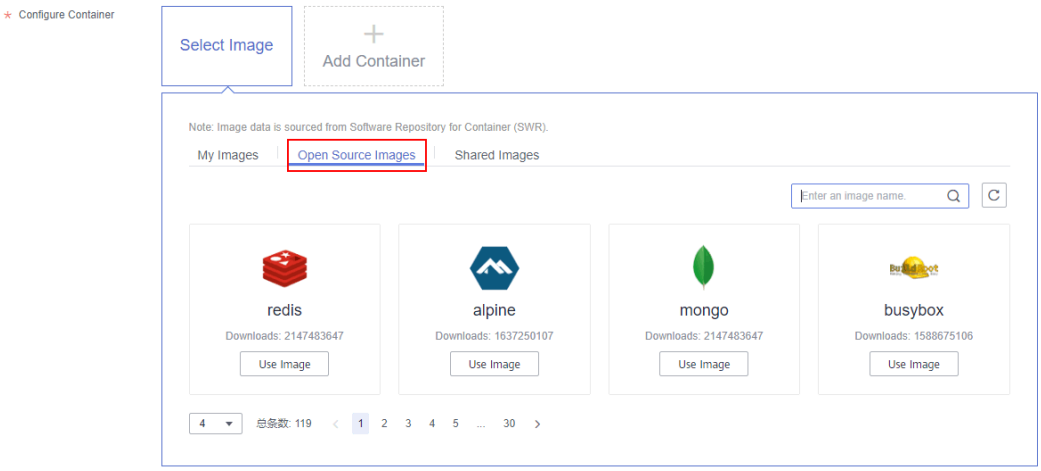
CCI supports Kubernetes-based liveness probing and readiness probing. You can set them when creating a workload. For details, see **Health Check**.

3.3 How Do I Configure an Auto Scaling Policy?

CCI supports three types of auto scaling policies: metric-based, scheduled, and periodic. For details, see **Workload Scaling**.

3.4 What Do I Do If the Workload Created from the sample Image Fails to Run?

If you have used SWR but have not pushed any image to SWR, SWR will build an image named **sample** for you. However, this image cannot run. Therefore, you are advised to use an image on the **Open Source Images** tab page to create a workload.



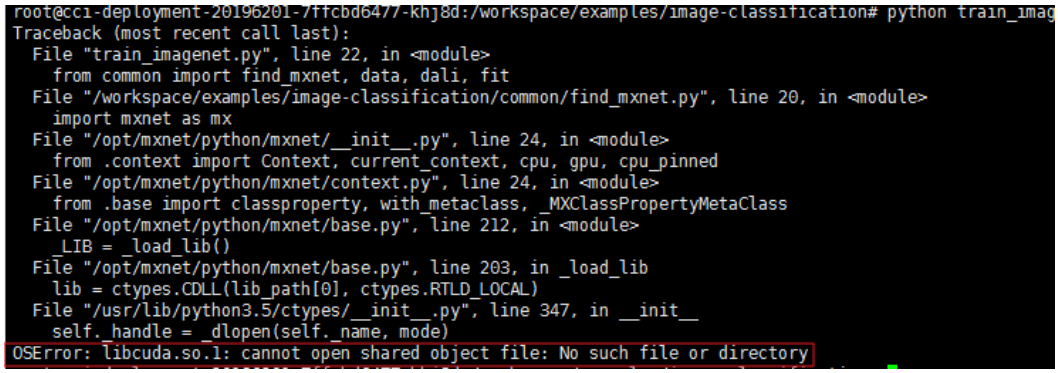
3.5 How Do I View Pods After I Call the API to Delete a Deployment?

The **propagationPolicy** field in a Deployment deletion API request indicates whether to delete pods along with the Deployment. This field can be set to **Orphan**, **Foreground**, or **Background**. For details about this field, see **Deleting a Deployment**.

3.6 Why an Error Is Reported When a GPU-Related Operation Is Performed on the Container Entered by Using exec?

Symptom

After I enter a container using exec and perform a GPU-related operation (such as using nvidia-smi or running a GPU training task using TensorFlow), the error message "cannot open shared object file: No such file or directory" is displayed.



Possible Cause

The CUDA library in a container is located in **/usr/local/nvidia/lib64**. This directory must be added to **LD_LIBRARY_PATH** to ensure that the CUDA library can be found.

Solution

Log in to the GPU-accelerated container by using kubectl exec or console, run the **export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64** command, and then perform other GPU-related operations.

4 Image Repository FAQs

4.1 Can I Export Public Images?

You cannot export or download images uploaded by other users.

4.2 How Do I Create a Container Image?

This section describes how to use Dockerfile to create a container image for a simple web workload.

Context

After you create a containerized workload using an official Nginx image, the default Nginx welcome page is displayed. The following describes how to customize an image to change the default welcome message to **Hello, CCI!**.

Procedure

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

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

mkdir mynginx

cd mynginx

touch Dockerfile

Step 3 Run the following command to edit the **Dockerfile** file:

vi Dockerfile

File content:

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

Where:

- FROM statement: specifies that an Nginx image is used as a base image.
- RUN statement: indicates that the **echo** command is executed to display **Hello, CCI!**.

Step 4 Create a container image.

docker build -t nginx:v3 .

Step 5 Run the following command to check the created image. The command output shows that the nginx image has been created with a tag of v3.

docker images

----End

4.3 How Do I Upload Images?

Software Repository for Container (SWR) manages images. It provides two methods to upload images:

- **Uploading an image through a Docker client**
- **Uploading an image through the SWR console**

4.4 Does CCI Provide Base Container Images for Download?

The image repository of CCI is provided by SWR, which provides base container images for download.

4.5 Does CCI Administrator Have the Permission to Upload Image Packages?

To upload images for CCI, you need to use SWR.

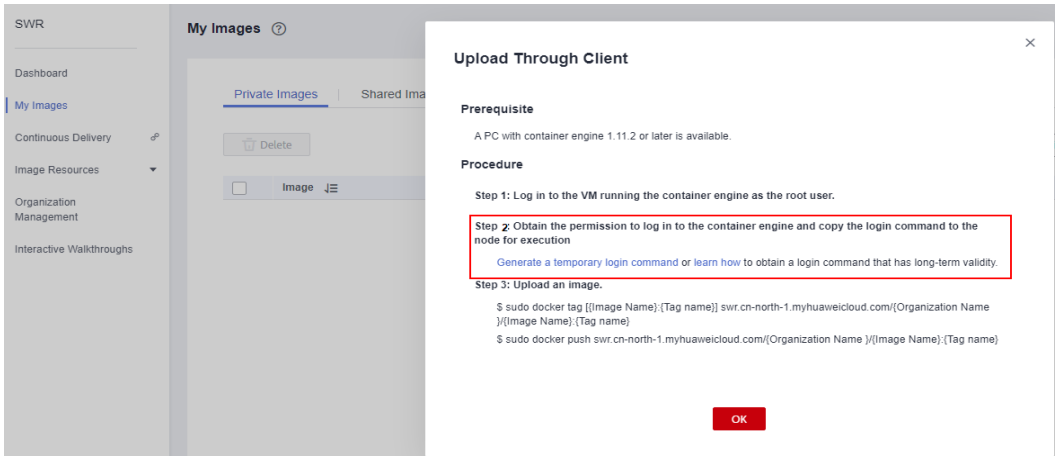
You also need to add **SWR Admin** permissions for your account.

4.6 What Do I Do If Authentication Is Required During Image Push?

SWR is used when you upload images on the CCI console.

To upload images using SWR, first obtain the permission to access SWR. For details about how to upload images, see **Uploading an Image Through a Docker Client**.

Figure 4-1 Uploading an image

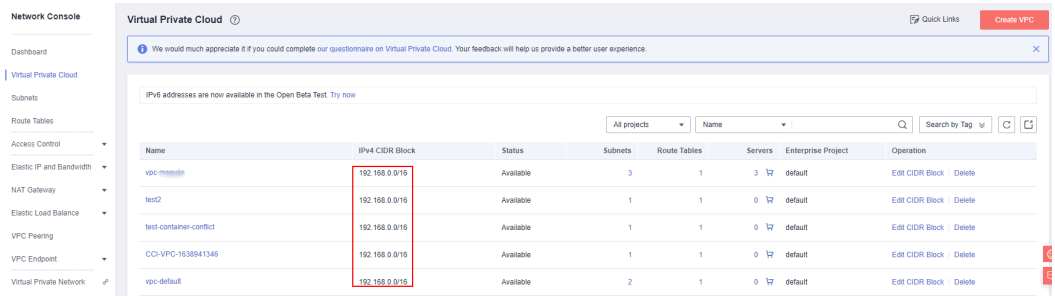


5 Network Management FAQs

5.1 How Do I View the VPC CIDR Block?

On the home page of the VPC console, view **Name** and **IPv4 CIDR Block** of VPCs. You can modify the CIDR block of a VPC or re-create a VPC.

Figure 5-1 Viewing the VPC CIDR blocks



5.2 Does CCI Support Load Balancing?

CCI supports load balancing. On the **Configure Access Settings** page for creating a workload on the CCI console, the access mode can be set to **ELB** for both intranet access and Internet access. For details about ELB-based intranet access, see **Workload Access Through a Private Network Load Balancer**.

Generally, load balancing refers to public network load balancing. CCI connects to Elastic Load Balance (ELB) of HUAWEI CLOUD to implement load balancing.

When creating a workload on the CCI console, you can choose **Intranet access** or **Internet access** and configure a load balancer on the **Configure Access Settings** page.

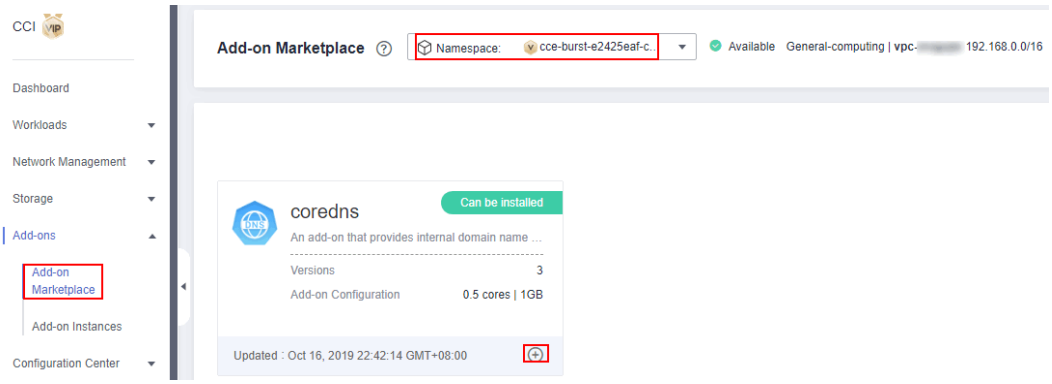
- For details about public network load balancing, see **Public Network Access**.
- For details about private network load balancing, see **Private Network Access**.

5.3 How Do I Configure the DNS Service on CCI?

If workloads require the internal domain name resolution provided by Kubernetes, the coredns add-on must be installed. In this case, **dnsPolicy** of pods must be set to **ClusterFirst**.

On the **Add-on Marketplace** page, click on the card of the coredns add-on to install it under the specified namespace.

Figure 5-2 Installing an add-on



If workloads do not require the internal domain name resolution provided by Kubernetes but require HUAWEI CLOUD DNS, **dnsPolicy** of pods must be set to **Default**.

In addition, you can set **dnsPolicy** of pods to **None** to use the custom DNS service. The following is a YAML example:

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
    - name: test
      image: nginx
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 1.2.3.4
    searches:
      - ns1.svc.cluster-domain.example
      - my.dns.search.suffix
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

5.4 Does CCI Support InfiniBand (IB) Networks?

No.

5.5 How Do I Access a Container from a Public Network?

ELB can be bound to workloads. When creating a workload, you can configure the public network access mode and bind a load balancer. In this way, you can access the workload by using the IP address of the load balancer. For details, see **Public Network Access**.

If you use the kubectl, you can create the service and ingress objects and bind load balancers to the two objects, enabling container access from the public network. For details, see **Service** and **Ingress**.

5.6 How Do I Access a Public Network from a Container?

You can use the NAT Gateway service available on the public cloud platform. This service offers NAT for containers in a VPC, allowing these containers to access the Internet using an EIP. The NAT Gateway service supports a large number of concurrent connections, which makes it suitable for applications involving a large number of requests and connections. For details, see **Accessing Public Networks from a Container**.

5.7 What Do I Do If Access to a Workload from a Public Network Fails?

1. A workload must be in running state before it can be accessed from a public network. If your workload is abnormal or not ready, it cannot be accessed properly from a public network.
2. The workload cannot be accessed properly from the public network until the network route is configured, which takes 1 to 3 minutes.
3. If the workload cannot be accessed 3 minutes after being created, click the workload on the CCI console. On the details page that is displayed, choose **Access Settings** to check whether any alarm events are reported. The container cannot be accessed from the public network if either of the following events is displayed:
   - Listener port is repeated: This event occurs when you delete a workload for which a load balancer port is configured, and immediately after that, create a workload using the same load balancer port. It takes some time for a load balancer port to be deleted. Wait for 5 to 10 minutes, and then the workload can be accessed from the public network.
   - Create listener failed: This event occurs usually because the listener quota is exceeded. Select another load balancer with a sufficient quota.
4. The workload is still inaccessible after it has been created for 3 minutes, and there is no alarm event. The possible reason is that no corresponding process is actually listening to the user-configured container port. Currently, CCI cannot detect this type of exception. You need to check whether the image is

listening to this container port. If the container port is properly listened to, the access failure may be caused by the load balancer. In this case, check the status of the load balancer.

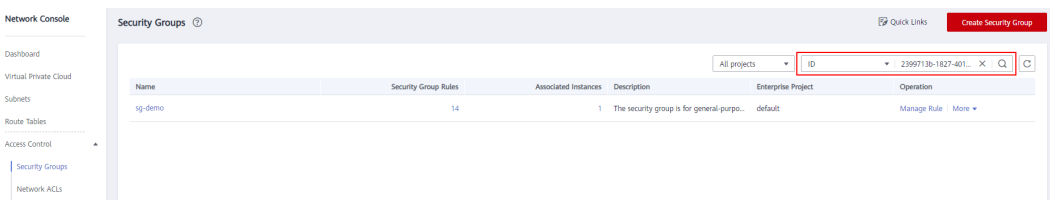
5.8 What Do I Do If Error 504 Is Reported When I Access a Workload?

Error 504 is reported generally when the security group between the port bound to the load balancer and the CCI workload pod is not bypassed. Check the security group used by the CCI workload pod and ensure that the security group rule permits the port bound to the load balancer.

The security group bound to the pod can be obtained by reading the network corresponding to the workload. You can call the API to read the network. For details, see **Help Center > Cloud Container Instance > API Reference > Proprietary APIs > Network > Reading a Network**. In a response, **network.alpha.kubernetes.io/default-security-group** in **metadata.annotations** is the security group ID.


```
{
   "kind": "Network",
   "apiVersion": "networking.cci.io/v1beta1",
   "metadata": {
      "name": "namespace-test-dc1-default-network",
      "namespace": "namespace-test",
      "selfLink": "/apis/networking.cci.io/v1beta1/namespaces/namespace-test/networks/namespace-test-dc1-default-network",
      "uid": "6fb85414-af6b-11e8-b6ef-f898ef6c78b4",
      "resourceVersion": "5016899",
      "creationTimestamp": "2018-09-03T11:21:00Z",
      "annotations": {
         "network.alpha.kubernetes.io/project-id": "51bf52609f2a49c68bfda3398817b376",
         "network.alpha.kubernetes.io/default-security-group": "19c5d024-aed5-4856-b958-c0f65ce70855",
         "network.alpha.kubernetes.io/domain-id": "aadb43c0b14c4cafbccfff483d075987"
      },
      "enable": true
   },
   "spec": {
      "cidr": "192.168.244.0/23",
      "attachedVPC": "0d4080e5-546a-46c4-86fe-f3e26d685177",
      "networkType": "underlay_neutron",
      "physicalNetwork": "phy_net0",
      "networkID": "0022e356-f730-4226-802e-9cdaa6e7da17",
      "subnetID": "1ffd839d-e534-4fa8-a59d-42356335bf74",
      "availableZone": "cnnorth1a"
   },
   "status": {
      "state": "Active"
   }
}
```


Log in to the **Network Console**, and search for the security group based on the obtained security group ID.



Click the security group name, and add the rules shown in the following figure on the **Inbound Rules** tab page.

NOTICE

If you access the workload from the public network through UDP, an ICMP rule must be added, which will be used during health check.

Protocol & Port	Type	Source
All	IPv4	sg-test
ICMP : Echo	IPv4	0.0.0.0/0
TCP : 22	IPv4	0.0.0.0/0
TCP : 3389	IPv4	0.0.0.0/0
TCP : 1-65535	IPv4	0.0.0.0/0
UDP : 1-65535	IPv4	0.0.0.0/0

5.9 What Do I Do If the Connection Timed Out?

Symptom

Pods can be created, but the message "[Errno 110] Connection timed out" is displayed when the Python Django SMTP service is used to send emails.

Possible Cause

- You have purchased only ELB but not NAT Gateway. Therefore, containers can be accessed only from the external network. Containers can access the external network after you purchase a NAT Gateway.
- Port 25 is prohibited from sending messages to secure the network environment.

Solution

- Solution 1: Use the **NAT Gateway** service to enable container instances (pods) in a VPC to access public networks. The NAT Gateway service provides source network address translation (SNAT), which translates private IP addresses to a public IP address by binding an elastic IP address (EIP) to the gateway, providing secure and efficient access to the Internet. For details, see **Accessing Public Networks from a Container**.
- Solution 2: Contact technical support to allow port 25 of your new EIP.

6 Storage Management FAQs

6.1 Which Cloud Storage Types Are Supported and Which Types Require Backup?

Currently, CCI supports the following cloud storage types:

- EVS disks
- SFS file systems
- SFS Turbo file systems
- OBS buckets

If an EVS disk is used, you need to manually configure a backup policy for it. For details, see **VBS Backup Management**.

6.2 How Do I Use Cloud Storage?

You can create cloud storage on the CCI console. For details, see **Overview**.

You can also use the kubectl to create cloud storage. For details, see **Using Persistent Storage**.

6.3 Where Is the Data Generated by a Container Stored If No Cloud Storage Is Mounted?

If no EVS disk or other cloud storage is mounted, the application data is stored in the physical machine disks of the container. Each pod can be allocated with a maximum of 10 GB disk of the CPU host or 30 GB disk of the GPU host for data storage. If the node is a dedicated node, the storage space can be adjusted based on customer requirements.

NOTE

To ensure data security, the container engine obtains a virtual disk from devicemapper when creating a container. Other container engines cannot access the virtual disk of this container.

6.4 Why Volume Mounting Failure Is Reported When a Job Pod Completes Its Task?

Symptom

When a job pod completes its task, there are still events showing that a volume failed to be mounted into the pod.

Possible Cause

When a pod of a Deployment, StatefulSet, job, or Cron job is started on a node:

- The kubelet creates a podWorker (independent goroutine) to check volume mounting into the pod at a regular interval of 0.3 seconds. If the volume mounting is not complete within 0.3 seconds or the timeout duration (123 seconds) expires, the event "Unable to mount volumes for pod ..." is reported.
- The VolumeManager (independent goroutine) in the kubelet is responsible for mounting volumes into the pod.

For a long running pod of a Deployment or StatefulSet: If the pod is started without experiencing failures to pull images, mount volumes, assign container CIDR block, or provide node CPU or memory resources that match the pod's needs, the volume mounting will eventually be successful after a few check cycles.

For a short-lived pod of a job or Cron job: The pod is very likely to exit gracefully soon after it is started, for example, the GCS Demo job in question runs only a few echo and ls commands and exits in less than 1 second. If the pod exits gracefully during the interval between consecutive volume mounting checks, a false event notification will be generated but services are not affected. In practice, pod jobs always run longer than 1 second.

The Kubelet's capability to check volume mounting is already considered in the volume mounting framework provided by the community.

Solution

A pod of a job or Cron job usually runs for a short period of time. If the pod completes its task much earlier than the timeout duration of volume mounting, the volume mounting timeout event is a false event notification and can be ignored. The event has no impact on the service run by the pod.

