Cloud Container Instance (CCI)

User Guide

Issue	01
Date	2025-08-12

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property 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 Cloud 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.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Permissions Management

1.1 Permissions Management for CCI

CCI permissions management allows you to grant permissions to your IAM users and user groups. It works with Identity and Access Management (IAM) to provide a variety of authorization methods, including IAM fine-grained authorization, IAM token authorization, namespace authorization, and resource authorization in namespaces.

- **CCI permissions:** permissions granted based on IAM fine-grained authorization. You can authorize users to perform operations on namespaces, such as creating and deleting namespaces.

1.2 Creating a User and Granting Permissions

This section describes how to use **IAM** to implement fine-grained permissions control for your CCI resources. With IAM, you can:

- Create IAM users for employees based on your enterprise's organizational structure. Each IAM user will have their own security credentials for accessing CCI resources.
- Grant users only the permissions required to perform a given task based on their job responsibilities.
- Entrust an account or cloud service to perform efficient O&M on your CCI resources.

If your account does not require individual IAM users, skip this section.

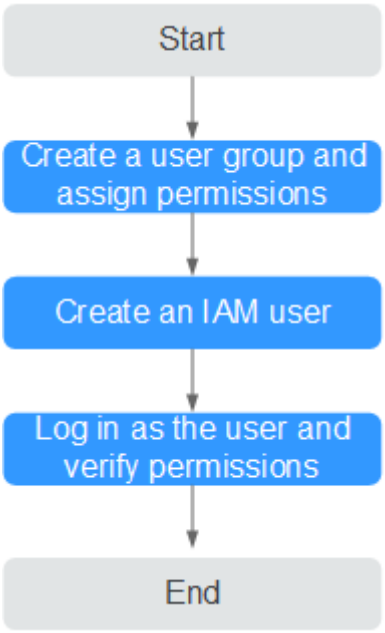
The following is the procedure for granting permissions (see **Figure 1-1**).

Prerequisites

You have learned about the permissions supported by CCI.

Process Flow

Figure 1-1 Process of granting CCI permissions



1. **Create a user group and assign permission**.

   Create a user group (for example, **Developers**) on the IAM console and assign the **CCI CommonOperations** policy to the group. CCI is a project-level service. When assigning system-defined policies to users, you also need to assign the **IAM ReadOnlyAccess** policy to the users.

2. **Create a user and add it to a user group**.

   Create a user (for example, **James**) on the IAM console and add the user to the group created in **1**.

3. **Log in as the user you created** and verify permissions.

   Log in to the management console as the user you created and verify that the user has the assigned permissions.

   - Choose **Service List** > **Cloud Container Instance**. In the navigation pane, choose **Workloads**. On the **Deployments** tab, click **Create Deployment**. If the Deployment is created successfully, the **CCI CommonOperations** policy has taken effect.
   - Choose **Service List** > **Cloud Container Instance**. In the navigation pane, choose **Namespaces**. On the page displayed, click **Create Namespace**. If the namespace cannot be created, the **CCI CommonOperations** policy has taken effect.

1.3 CCI Custom Policies

You can create custom policies to supplement the system-defined policies of CCI.

To create a custom policy, choose either visual editor or JSON.

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.
- JSON: Create a policy in the JSON format from scratch or based on an existing policy.

For details, see **Creating a Custom Policy**. The following section contains examples of common CCI custom policies.

Example Custom Policies

- Example 1: Updating a namespace

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "cci:namespace:update"
         ]
      }
   ]
}
```

- Example 2: Denying namespace deletion

  A policy with only "Deny" permissions must be used in conjunction with other policies for it to take effect. If you assign both "Allow" and "Deny" to a user, the "Deny" permissions take precedence over the "Allow" permissions.

  The following method can be used if you need to assign permissions of the **CCIFullAccess** policy to a user but you want to prevent the user from deleting namespaces (**cci:namespace:delete**). Create a custom policy for denying namespace deletion, and attach both policies to the group that the user belongs to. Then, the user can perform all operations on CCI except deleting namespaces. The following is an example of a deny policy:

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Action": [
            "cci:namespace:delete"
         ],
         "Effect": "Deny"
      }
   ]
}
```

- Example 3: Defining permissions for multiple services in a policy

  A custom policy can contain the actions of multiple services that are of the global or project-level type. The following is an example policy containing actions of multiple services:

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Action": [
            "ecs:cloudServers:resize",
            "ecs:cloudServers:delete",
            "ims:images:list",
            "ims:serverImages:create"
         ],
         "Effect": "Allow"
      }
```



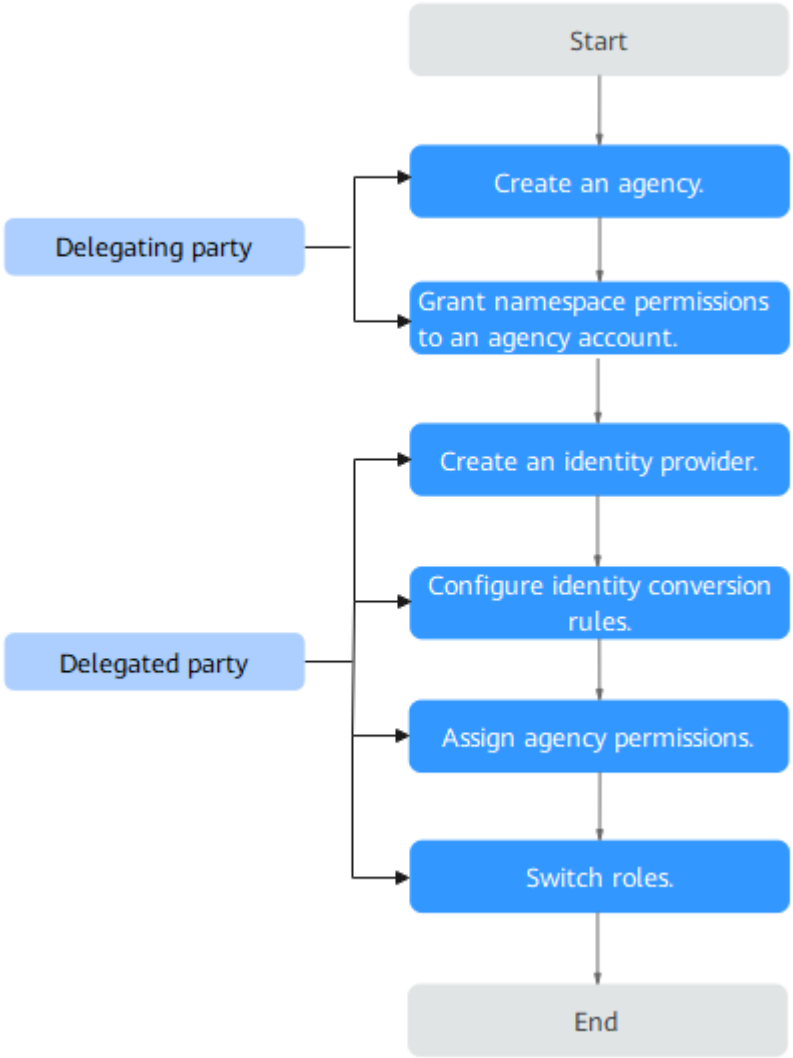
```
  ]
}
```

1.4 Delegating a Federated User to Manage Resources

If you want to delegate a federated user of another account (account B) to manage resources in your account (account A), log in using account A, create an agency for account B, and grant namespace permissions to account B. Then, log in using account B and perform federated identity authentication for it. After the authentication is complete, account B assigns the agency permissions to the federated user so that the federated user can switch to the agency of account A. Log in to Huawei Cloud as the federated user and switch the role to manage resources in account A.

This section describes how to delegate federated users to manage resources. **Figure 1-2** shows the operation process.

Figure 1-2 Process of delegating a federated user to manage resources



Procedure

To delegate account B to manage resources in account A as a federated user, take the following steps:

Step 1 **Create an agency (by the delegating party)**.

Log in to the IAM console as the delegating party (account A). Create an agency, enter the account name of the delegated party (account B), and grant permissions of the **CCIFullAccess** policy to the delegated party. Users granted these permissions can create, delete, query, and update all CCI resources.

Step 2 **Perform the federated identity authentication as the delegated party**.

Log in to as the delegated party (account B) and perform federated identity authentication.

Before delegating a federated user to manage resources, you need to perform federated identity authentication on the delegated party. The authentication process consists of two steps: establish a trust relationship and create an identity provider, and then configure identity conversion rules.

NOTE

After an identity provider is created, a default identity conversion rule is also created. You need to click **Edit Rule** to update or delete the default rule and create one. If you add a new rule without deleting the default rule, the default rule may be matched, and the new rule does not take effect.

Step 3 **Assign permissions to a user (by the delegated party)**.

If a user under the delegated party (account B) wants to manage account A's resources, the delegated party (account B) must assign agency permissions to the user. To enable a federated user to manage resources of the delegating party (account A), the delegated party (account B) needs to assign the permissions of the custom policy **federation_agency** to the user group (**federation_group**) that the federated user belongs to. **federation_group** is the user group that the federated user belongs to. It is user-defined and written into the identity conversion rules.

Step 4 **Switch roles (by the delegated party)**.

Account B and the federated user with agency permissions can switch their roles to the delegating party (account A) to manage its resources.

----End

2 Environment Configuration

Purchasing VPC Endpoints

VPC endpoints are required for accessing cloud services that use the network segment starting with 100.

- To pull images from a repository of SWR Enterprise Edition, you need to purchase a VPC endpoint for accessing OBS.
- To pull images from an SWR public image repository, you need a VPC endpoint for accessing SWR and a VPC endpoint for accessing OBS in the VPC where the workload is deployed.

Step 1 Go to the **VPC endpoint list** page.

Step 2 On the **VPC Endpoints** page, click **Buy VPC Endpoint**.

The **Buy VPC Endpoint** page is displayed.

Step 3 Configure the parameters. Both the VPC endpoint for accessing SWR and the VPC endpoint for accessing OBS work only in the VPC where they are created.

Figure 2-1 Buying a VPC endpoint for accessing SWR

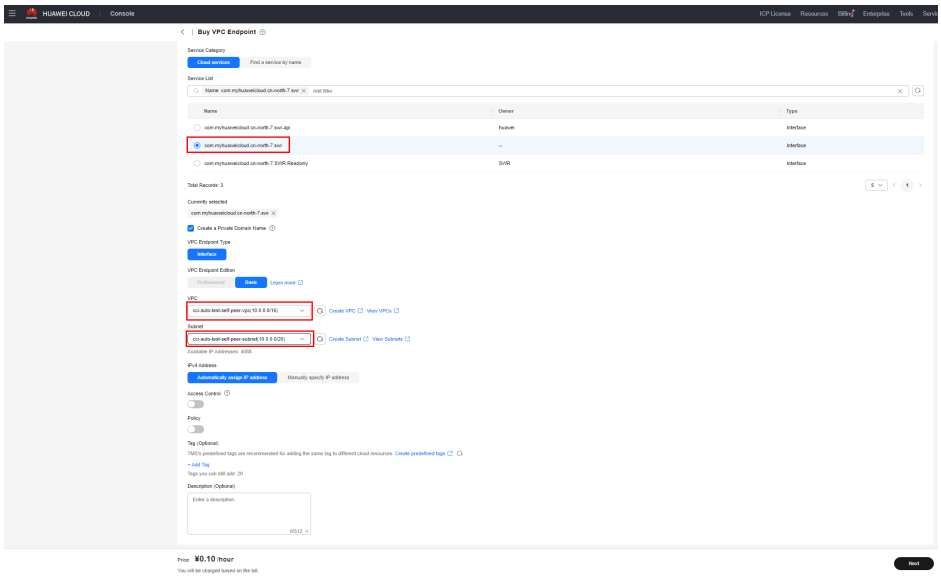


Figure 2-2 Buying a VPC endpoint for accessing OBS

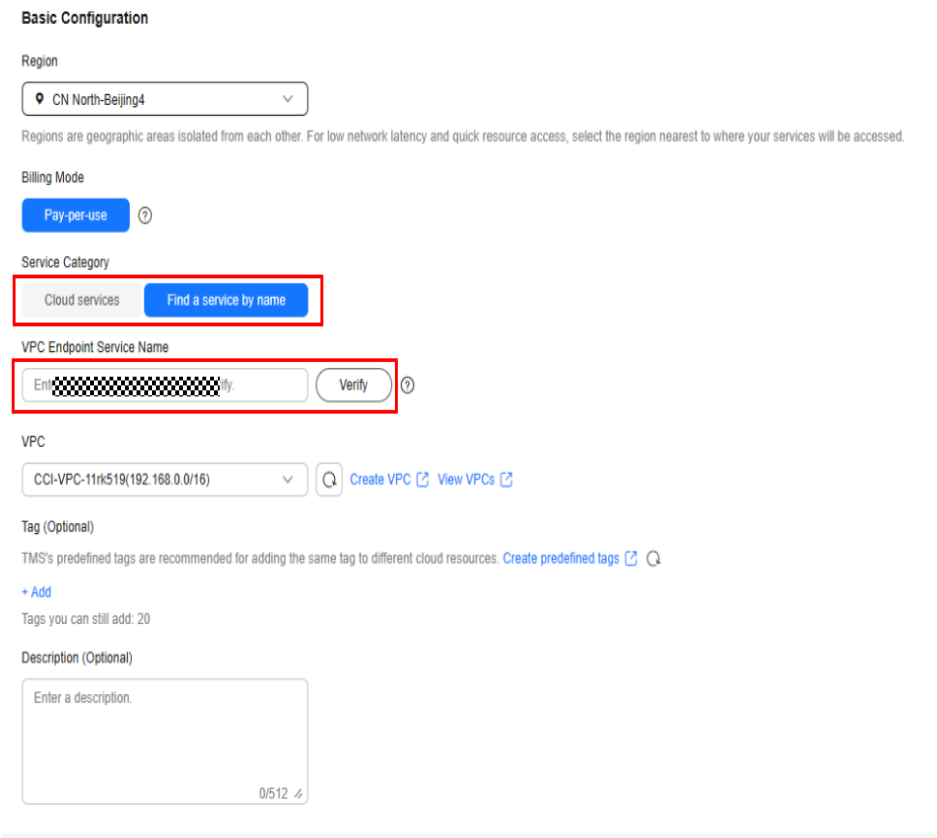


Table 2-1 VPC endpoint parameters

Parameter	Example	Description
Region	EU-Dublin	Specifies the region where the VPC endpoint will be used to connect a VPC endpoint service. Resources in different regions cannot communicate with each other over an intranet. For lower latency and quicker access, select the region nearest to your on-premises data center.
Billing Mode	Pay-per-use	Specifies the billing mode of the VPC endpoint. VPC endpoints can be used or deleted at any time. VPC endpoints support only pay-per-use billing based on the usage duration.

Parameter	Example	Description
Service Category	Cloud services	There are two options: • **Cloud services**: Select this option if the VPC endpoint service to be accessed is a cloud service. • **Find a service by name**: Select this option if the VPC endpoint service to be accessed is a private service of your own. **CAUTION** • Select **Cloud services** when you buy a VPC endpoint for accessing SWR. • Select **Find a service by name** when you buy a VPC endpoint for accessing OBS.
Service List	-	This parameter is available only when you select **Cloud services** for **Service Category**. VPC endpoint services have been created. You can select one of them. **NOTE** If you select **Find a service by name** for **Service Category** when you buy a VPC endpoint for accessing OBS, submit a service ticket to get the service name.
VPC	-	Specifies the VPC where the VPC endpoint is to be deployed.
Subnet	-	Specifies the subnet where the VPC endpoint is to be deployed.
Route Table	-	This parameter is available only when you create a VPC endpoint for connecting to a gateway VPC endpoint service. **NOTE** This parameter is available only in the regions where the route table function is enabled. You are advised to select all route tables. Otherwise, the access to the gateway VPC endpoint service may fail. Select a route table required for the VPC where the VPC endpoint is to be deployed. For details about how to add a route, see **Adding Routes to a Route Table** in the *Virtual Private Cloud User Guide*.

Parameter	Example	Description
Tag	example_key1 example_value1	Specifies the tag that is used to classify and identify the VPC endpoint. The tag settings can be modified after the VPC endpoint is purchased
Description	-	Provides supplementary information about the VPC endpoint.

Table 2-2 Tag requirements for VPC endpoints

Parameter	Requirement
Tag key	• Cannot be left blank. • Must be unique for each resource. • Can contain a maximum of 36 characters. • Cannot start or end with a space or contain special characters =*<>\,|/ • Can contain only letters, digits, hyphens (-), and underscores (_).
Tag value	• Cannot be left blank. • Can contain a maximum of 43 characters. • Cannot start or end with a space or contain special characters =*<>\,|/ • Can contain only letters, digits, hyphens (-), and underscores (_).

Step 4 Confirm the settings and click **Next**.

- If the configuration is correct, click **Submit**.
- If any parameter is incorrect, click **Previous** to modify it as needed and then click **Submit**.

Step 5 Click **Back to VPC Endpoint List** after the task is submitted.

Step 6 View the endpoint details by clicking each endpoint ID.

----End

Logging In to the CCI Console

Log in to the CCI console and grant CCI the permissions to access other cloud services.

Step 1 Log in to the management console.

Step 2 Click  in the upper left corner to select the desired region.

> **NOTE**
>
> CCI does not allow you to create resources in sub-projects.

Step 3 Choose **Service List** > **Containers** > **Cloud Container Instance**.

Switch to the CCI console.

Step 4 If this is the first time you are logging in to the CCI console, click **Agree** to grant CCI the permissions to access other cloud services.

After the permissions are granted, an agency named **cci_admin_trust** is created. You can view the agency on the IAM console.

----End

(Optional) Uploading Images

The cloud platform provides the SoftWare Repository for Container (SWR) service for you to upload container images to the image repository. You can easily pull these images when creating workloads on CCI. For details about how to upload images, see **Pushing an Image Through a Container Engine Client**.

NOTICE

3 Namespaces

3.1 Creating a Namespace

Namespaces are used to logically divide your resources into different groups, especially in scenarios where a large number of users from multiple teams work on different projects.

CCI provides general computing resources and allows you to create pods with vCPUs for general computing.

NOTE

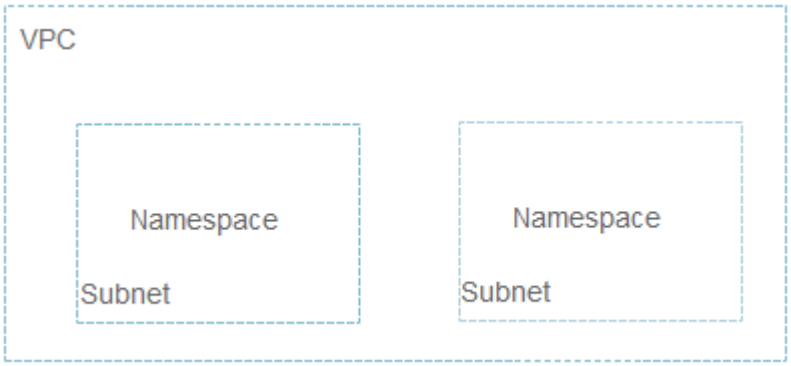
- One account can create a maximum of five namespaces in a region.
- x86 images are supported.

Relationship Between Namespaces and Networks

Each namespace requires a separate subnet, as shown in **Figure 3-1**. When you create a namespace, you need to associate it with a VPC. A subnet will be created for the namespace in the VPC. Containers and other resources created in this namespace will run in the VPC and subnet you select.

If you want to run resources of multiple services in the same VPC, you need to consider network planning, including subnet CIDR block division and IP address planning.

Figure 3-1 Relationship between namespaces and VPC subnets



Application Scenarios

Namespaces can implement partial environment isolation. If you have a large number of projects and personnel, you can create different namespaces based on project attributes, such as production, test, and development.

Creating a Namespace

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Namespaces**.

Step 3 On the **Namespaces** page, click **Create Namespace** in the upper right corner.

Step 4 Enter a name for the namespace.

NOTE

- The name of each namespace must be unique.
- Enter 1 to 63 characters starting and ending with a lowercase letter or digit. Only lowercase letters, digits, and hyphens (-) are allowed.

Step 5 (Optional) Specify monitoring settings.

Parameter	Description
AOM (Optional)	If this option is enabled, you need to select an AOM instance.

Step 6 Configure the network plane.

Table 3-1 Network plane settings

Parameter	Description
IPv6	If this option is enabled, IPv4/IPv6 dual stack is supported.

Parameter	Description
VPC	Select the VPC where the workloads are running. If no VPC is available, create one first. The VPC cannot be changed once selected. Recommended CIDR blocks: 10.0.0.0/8-22, 172.16.0.0/12-22, and 192.168.0.0/16-22 **NOTICE** • You cannot set the VPC CIDR block and subnet CIDR block to 10.247.0.0/16, because this CIDR block is reserved for workloads. If you select this CIDR block, there may be IP address conflicts, which may result in workload creation failure or service unavailability. If you do not need to access pods through workloads, you can select this CIDR block. • After the namespace is created, you can choose **Namespaces** in the navigation pane and view the VPC and subnet in the **Subnet** column.
Subnet	Select the subnet where the workloads are running. If no subnet is available, create one first. The subnet cannot be changed once selected. • A certain number of IP addresses (**10** by default) in the subnet will be warmed up for the namespace. • You can set the number of IP addresses to be warmed up in **Advanced Settings**. • If warming up IP addresses for the namespace is enabled, the VPC and subnet can only be deleted after the namespace is deleted. **NOTE** Ensure that there are sufficient available IP addresses in the subnet. If IP addresses are insufficient, workload creation will fail.
Security Group	Select a security group. If no security group is available, create one first. The security group cannot be changed once selected.

Step 7 (Optional) Specify advanced settings.

Each namespace provides an IP pool. You can specify the pool size to reduce the duration for assigning IP addresses and speed up the workload creation.

For example, 200 pods are running routinely, and 200 IP addresses are required in the IP pool. During peak hours, the IP pool instantly scales out to provide 65,535 IP addresses. After a specified interval (for example, 23 hours), the IP addresses that exceed the pool size (65535 – 200 = 65335) will be recycled.

Table 3-2 (Optional) Advanced namespace settings

Parameter	Description
IP Pool Warm-up for Namespace	• An IP pool is provided for each namespace, with the number of IP addresses you specify here. IP addresses will be assigned in advance to accelerate workload creation. • An IP pool can contain a maximum of 65,535 IP addresses. • When using general-computing pods, you are advised to configure an appropriate size for the IP pool based on service requirements to accelerate workload startup. • Configure the number of IP addresses to be assigned properly. If the number of IP addresses exceeds the number of available IP addresses in the subnet, other services will be affected.
IP Address Recycling Interval (h)	Pre-assigned IP addresses that become idle can be recycled within the duration you specify here. **NOTE** Recycling mechanism: • Recycling time: The **yangtse.io/warm-pool-recycle-interval** field configured on the network determines when the IP addresses can be recycled. If **yangtse.io/warm-pool-recycle-interval** is set to **24**, the IP addresses can only be recycled 24 hours later. • Recycling rate: A maximum of 50 IP addresses can be recycled at a time. This prevents IP addresses from being repeatedly assigned or released due to fast or frequent recycling.

Step 8 Click **OK**.

You can view the VPC and subnet on the namespace details page.

----End

Deleting a Namespace

NOTICE

Deleting a namespace will delete all resources (such as workloads, Services, pods, ConfigMaps, and secrets) related to the namespace.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Namespaces**.

Step 3 On the **Namespaces** page, locate the namespace you want to delete and click **Delete** in the **Operation** column. In the **Delete Namespace** dialog box, enter **DELETE** and click **OK**.

NOTE

To delete a VPC or subnet, go to the **VPC console**.

----End

4 Using CCI Through the Console

4.1 Workload Management

4.1.1 Deployments

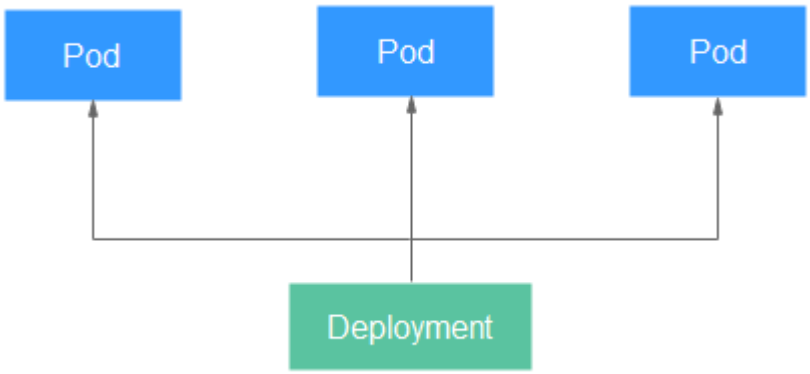
A Deployment is a service-oriented encapsulation of pods. A Deployment may manage one or more pods. These pods have the same role, and requests are distributed across the pods. All pods in a Deployment share the same volume.

Pods are the smallest deployable units. CCI provides controllers to manage pods. Controllers can create and manage pods, and provide replica management, rolling upgrade, and self-healing capabilities. The most commonly used controller is Deployment.

A Deployment can contain one or more pod replicas. These pod replicas have the same role, and requests are routed across the pod replicas.

A Deployment integrates a lot of functions, including rollout deployment, rolling upgrade, replica creation, and restoration of online jobs. To some extent, you can use Deployments to realize unattended rollout, which greatly reduces operation risks and improves rollout efficiency.

Figure 4-1 Deployment



Creating a Deployment on the Console

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Workloads**. On the **Deployments** tab, click **Create Deployment**.

Step 3 Add basic configuration.

Parameter	Description
Workload Name	Enter 1 to 63 characters starting and ending with a lowercase letter or digit. Only lowercase letters, digits, hyphens (-), and periods (.) are allowed. Do not enter consecutive periods or place a hyphen before or after a period. The workload name cannot be changed after the workload is created. If you need to change the name, create another workload.
Namespace	Select a namespace. If no namespaces are available, create one by following the steps provided in **Creating a Namespace**.
CPU Architecture	x86 or Kunpeng
Pod Type	**General** or **General-computing-lite** **NOTE** Only x86-based general-computing-lite pods are supported.
vCPUs	Select a value from **0.25** to **64**.
Memory	Select the memory based on the selected vCPUs.
Data Storage (Optional)	Only emptyDir volumes, ConfigMaps, and secrets are supported. Add a volume to the pod and then mount the volume to the specified container. Click **Add Data Store**, select a volume type, and enter a volume name. • **emptyDir volume**: By default, CCI provides 30 GiB of free storage space, which is shared by emptyDir volumes and the system disk. • **ConfigMap**: Select a ConfigMap. If no ConfigMaps are available, create one first. For details, see **Creating a ConfigMap**. • **Secret**: Select a secret. If no secrets are available, create one first. For details, see **Creating a Secret**.
Pods	Specify the number of pods. A workload can have one or more pods. A pod can have one or more containers with the same flavor. Configure multiple pods for a workload if you want higher reliability. If one pod becomes faulty, the workload can still run normally.

Step 4 Add containers. A pod generally contains one or more containers created from different images. If your application needs to run on multiple containers in a pod,

click **Add Container** and then select an image each time when you need another container.

1. Click **Add Container**.
2. Specify basic information.

Table 4-1 Basic parameters

Parameter	Description
Container Name	Enter 1 to 63 characters starting and ending with a lowercase letter or digit. Only lowercase letters, digits, and hyphens (-) are allowed.
Image	Select an image. **NOTICE** If different containers in a pod listen to the same port, there will be port conflicts, and the pod may fail to start. For example, if a container created from an Nginx image (which listens to port 80) has been added to a pod, there will be a port conflict when another container created from an HTTP image in the pod tries to listen to port 80. – **My Images**: images you have pushed to SWR. **NOTE** ▪ If you are an IAM user, you must obtain permissions before you can use the private images in the account. For details on how to obtain permissions, see **Uploading Images**. ▪ Currently, CCI does not support third-party image repositories. ▪ A single layer of a decompressed image cannot exceed 20 GiB. – **Shared Images**: images shared by others through SWR. – **Open Source Images**: public images in the image center.
Image Tag	Select a tag.
vCPUs	The value cannot be less than 0 or greater than the remaining quota.
Memory	The value cannot be less than 0 or greater than the remaining quota.

3. (Optional) Specify advanced settings.

- **Lifecycle**: Lifecycle scripts specify actions that applications take when a lifecycle event occurs. For details, see **Lifecycle**.
- **Health Check**: Container health can be checked regularly when the container is running. For details about how to configure health checks, see **Health Check**.
- **Environment Variables**: You can manually set environment variables or add variable references. Environment variables provide flexibility in configuring workloads. The environment variables for which you have assigned values

during container creation will take effect upon container startup. This saves you the trouble of rebuilding the container image.

To manually set variables, enter the variable name and value.

To reference variables, set the variable name, reference type, and referenced value for each variable. The following variables can be referenced: **PodIP** (pod IP address), **PodName** (pod name), and **Secret**. For details about how to reference a secret, see **Secrets**.

- **Data Storage**: Volumes can be mounted to containers for data access in the containers.
- **Security Settings**: Users who run containers can be configured.

Step 5 Select an upgrade policy.

- **Upgrade Policy**: **Rolling upgrade** and **Replace** are available.
  - **Rolling upgrade**: New pods gradually replace old ones, and service traffic is evenly distributed to both old and new pods to maintain service continuity.

    Max Unavailable Pods: Maximum number of unavailable pods allowed in a rolling upgrade. If the number is equal to the total number of pods, services may be interrupted. (Minimum number of alive pods = The total number of pods – Maximum number of unavailable pods)
  - **Replace**: Old pods are deleted, and then new pods are created. Services are interrupted during the upgrade.

Step 6 Click **Create Deployment**.

In the workload list, if the workload status changes to **Running**, the workload is created. You can click the workload name to view the details and refresh the page to view the real-time workload status.

----End

Creating a Deployment Using YAML

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Workloads**. On the **Deployments** tab, click **Create from YAML**.

Step 3 Import or add a YAML file and click **OK** to create a Deployment.

The following is an example file:

- Resource description in the **deployment.yaml** file

```
apiVersion: cci/v2
kind: Deployment
metadata:
  annotations:
    description: ''
  labels: {}
  name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
```



```
    annotations:
      vm.cci.io/pod-size-specs: 2.00_4.0
      resource.cci.io/pod-size-specs: 2.00_4.0
      metrics.alpha.kubernetes.io/custom-endpoints: '[{api:"",path:"",port:"",names:""}]'
      log.stdoutcollection.kubernetes.io: '{"collectionContainers": ["container-0"]}'
    labels:
      app: nginx
  spec:
    containers:
      - image: library/nginx:stable-alpine-perl
        name: container-0
        resources:
          limits:
            cpu: 2000m
            memory: 4096Mi
          requests:
            cpu: 2000m
            memory: 4096Mi
        command: []
        lifecycle: {}
    dnsPolicy: ''
    imagePullSecrets:
      - name: imagepull-secret
    dnsConfig: {}
minReadySeconds: 0
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 0
    maxUnavailable: 1
```

- Resource description in the **deployment.json** file

```
{
    "apiVersion": "cci/v2",
    "kind": "Deployment",
    "metadata": {
        "annotations": {
            "description": ""
        },
        "labels": {},
        "name": "nginx"
    },
    "spec": {
        "replicas": 2,
        "selector": {
            "matchLabels": {
                "app": "nginx"
            }
        },
        "template": {
            "metadata": {
                "annotations": {
                    "vm.cci.io/pod-size-specs": "2.00_4.0",
                    "resource.cci.io/pod-size-specs": "2.00_4.0",
                    "metrics.alpha.kubernetes.io/custom-endpoints": "[{api:'',path:'',port:'',names:''}]",
                    "log.stdoutcollection.kubernetes.io": "{\"collectionContainers\": [\"container-0\"]}"
                },
                "labels": {
                    "app": "nginx"
                }
            },
            "spec": {
                "containers": [
                    {
                        "image": "library/nginx:stable-alpine-perl",
                        "name": "container-0",
                        "resources": {
                            "limits": {
                                "cpu": "2000m",
                                "memory": "4096Mi"
```



```
                    },
                    "requests": {
                        "cpu": "2000m",
                        "memory": "4096Mi"
                    }
                },
                "command": [],
                "lifecycle": {}
            }
        ],
        "dnsPolicy": "",
        "imagePullSecrets": [
            {
                "name": "imagepull-secret"
            }
        ],
        "dnsConfig": {}
    }
  },
  "minReadySeconds": 0,
  "strategy": {
      "type": "RollingUpdate",
      "rollingUpdate": {
          "maxSurge": 0,
          "maxUnavailable": 1
      }
  }
 }
}
```


----End

Updating a Deployment

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Workloads**. On the **Deployments** tab, locate the deployment you want to update and click **Edit YAML** in the **Operation** column.

Step 3 Edit the YAML file and click **OK** to update the Deployment.

----End

Deleting a Pod

You can manually delete pods. Because pods are controlled by a controller, a new pod will be created immediately after you delete a pod. Manual pod deletion is useful when an upgrade fails halfway or when service processes need to be restarted.

Deleting a Deployment

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Workloads**. On the **Deployments** tab, locate the target Deployment and click **Delete** in the **Operation** column.

----End

4.1.2 Pods

4.1.2.1 Overview

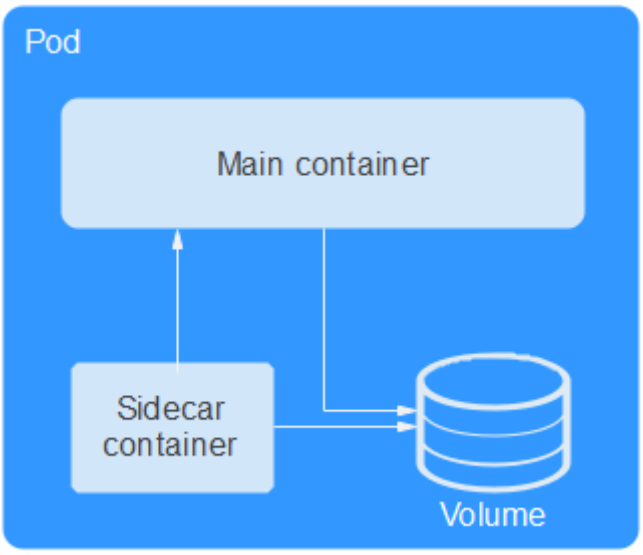
What Is a Pod?

Pods are the smallest deployable units of computing that you can create and manage. A pod is a group of one or more containers, with shared storage, a unique IP address, and a specification for how to run the containers.

Pods can be used in either of the following ways:

- A pod runs a single container. This is the most common use case. You can consider a pod as a wrapper around a single container. Pods can be managed directly rather than containers.
- A pod runs multiple containers that need to work together. In this scenario, a pod can encapsulate an application that is running in a main container and several sidecar containers. As shown in **Figure 4-2**, the main container serves as a web server that provides file services from a fixed directory, and the sidecar container periodically downloads files to the directory.

Figure 4-2 Pod



4.1.2.2 Creating a Pod

Scenario

You can use CCI to quickly create pods for running workloads. On the CCI console, you can view details about all pods, such as basic information, container list, storage volumes, and events. In addition, you can use remote terminals to access pods. You can also delete pods if you no longer need them.

Prerequisites

- A namespace has been created. If there are no namespaces, create one by referring to **Creating a Namespace**.
- The image has been uploaded. For details, see **Pushing an Image Through a Container Engine Client**.

Constraints

There are system components that help run the pods. These system components occupy some underlying resources, such as vCPU and memory. As a result, the resource usages for the pods may not reach the expected limits. To avoid this, refer to Reserved System Overhead.

Creating a Pod

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Workloads**. On the **Pods** tab, click **Create Pod**.

Step 3 On the **Create Pod** page, enter the basic information.

Parameter	Description
Pod Name	• Enter a name for the pod. The name must be unique in the same namespace. • Enter 1 to 204 characters. Start and end with a lowercase letter or digit. Only lowercase letters, digits, hyphens (-), and periods (.) are allowed. The total length of the pod name and namespace name cannot exceed 217 characters.
Namespace	Namespace that the pod belongs to.
Description (Optional)	Enter a description, which cannot exceed 250 characters.
CPU Architecture	x86 or Kunpeng
Pod Type	**General** or **General-computing-lite** **NOTE** Only x86-based general-computing-lite pods are supported.
vCPUs	Select a value from 0.25 to 64.
Memory	Select the memory based on the selected vCPUs.
Data Storage (Optional)	Only emptyDir volumes, ConfigMaps, and secrets are supported. Add a volume to the pod and then mount the volume to the specified container. Click **Add Data Store**, select a volume type, and enter a volume name. • **emptyDir volume**: By default, CCI provides 30 GiB of free storage space, which is shared by emptyDir volumes and the system disk. • **ConfigMap**: Select a ConfigMap. If no ConfigMaps are available, create one first. For details, see **Creating a ConfigMap**. • **Secret**: Select a secret. If no secrets are available, create one first. For details, see **Creating a Secret**.

Step 4 Specify container settings.

1. Add basic container information. The total resources of a container cannot exceed the pod flavor.

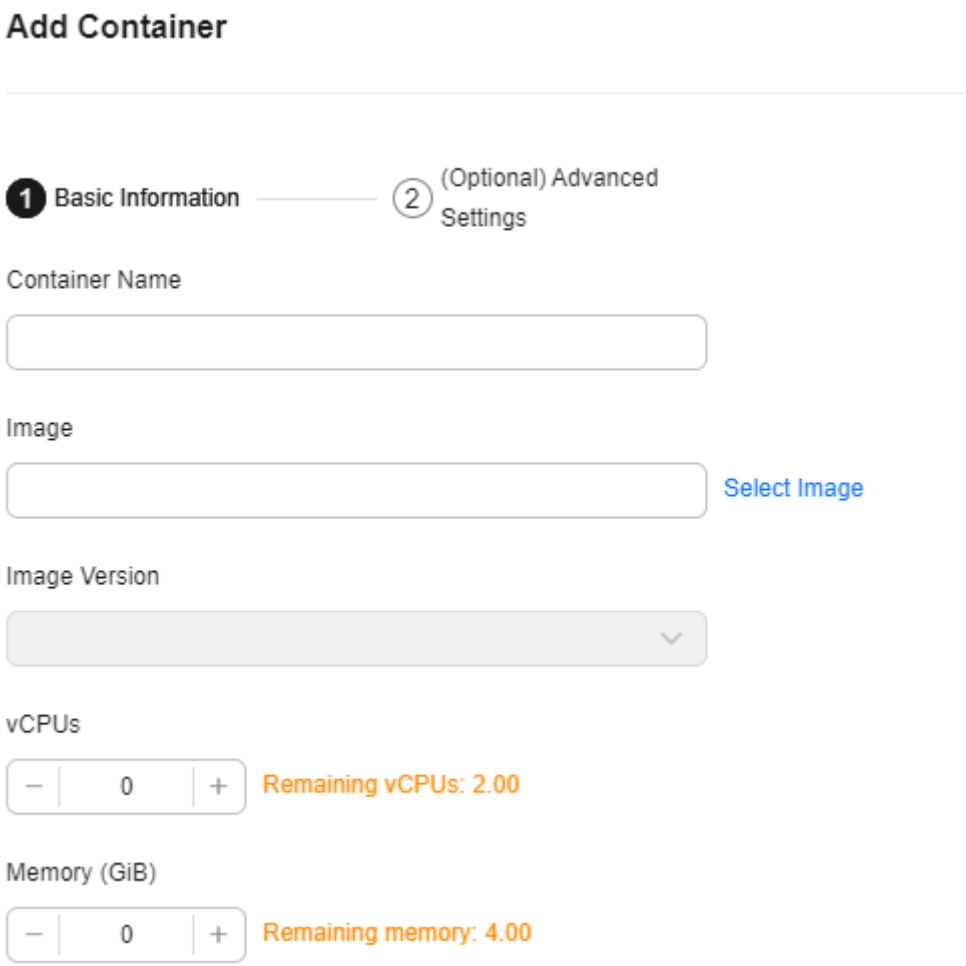


Table 4-2 Basic container information

Parameter	Description
Container Name	– The container name must be unique. – Enter 1 to 63 characters starting and ending with a lowercase letter or digit. Only lowercase letters, digits, and hyphens (-) are allowed.
Image	Select a container image. **CAUTION** Custom domain name images of the SWR Enterprise Edition cannot be used to create workloads.
Image Version	Select a container image tag.
vCPUs	Specify the vCPUs. The value cannot exceed that in the pod flavor.

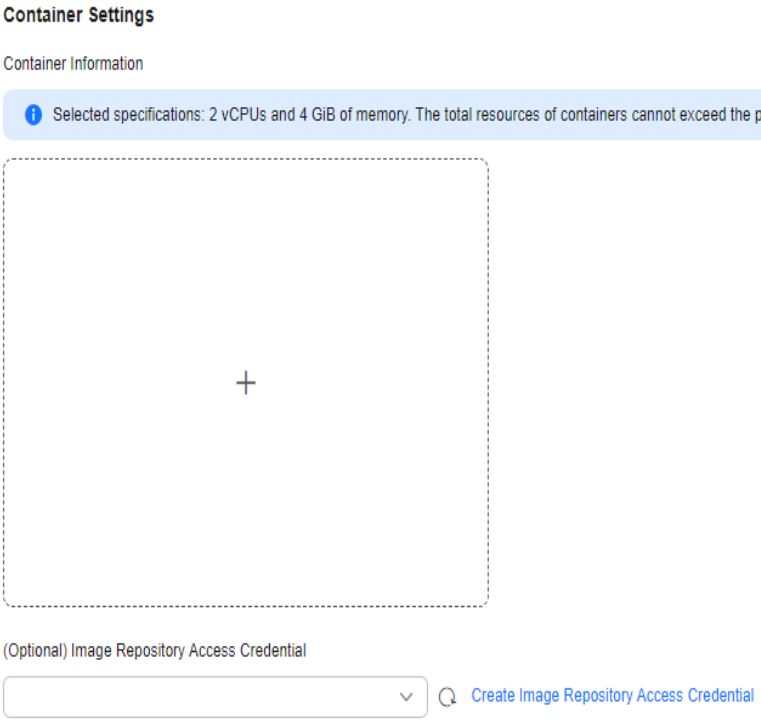
Parameter	Description
Memory	Specify the memory. The value cannot exceed that in the pod flavor.

2. (Optional) Specify advanced container settings.

Table 4-3 Advanced container settings

Parameter	Description
Lifecycle	CCI provides containers with **lifecycle hooks**, which enable containers to run code triggered by events during their lifecycle. For example, if you want a container to perform a certain operation before it is stopped, you can register a hook. For details, see **Lifecycle**. CCI provides the following lifecycle hooks: – Startup command: Docker ENTRYPOINT commands are used. – PostStart: This hook is triggered after an application is started. – PreStop: This hook is triggered before an application is stopped.
Health Check	Container health can be checked regularly when the container is running. For details, see **Health Check**. CCI supports the following types of probes: – Liveness probe: checks whether a container is normal and a restart is required. – Readiness probe: checks whether a container is ready to respond to requests. – Startup probe: checks whether an application has already started.
Environment Variables	Environment variables affect the way a running container will behave. You can update them after deploying the workload.
Data Storage	Volumes can be mounted to containers to read data from files or store data files persistently. To mount a volume to a container, add the volume to the pod first.
Security Settings	Specify a user ID for all the containers to run with. For example, enter **0** to run as **root**.

Step 5 (Optional) Select an image repository access credential.



Step 6 Click **Create Now**.

----End

4.1.2.3 Pod Flavor

4.1.2.3.1 Upgrading Pod Flavors

For pods created on the CCI console or using APIs, their flavors will be automatically upgraded based on the vCPUs and memory of containers to make full use of resources. This section describes how pod flavors are upgraded.

Table 4-4 Supported pod flavors

Container vCPUs	Container Memory (GiB)
0.25	0.5, 1, and 2
0.5	0.5, 1, 2, 3, and 4
1	1 to 8 (increment: 1 GiB)
2	2 to 16 (increment: 1 GiB)
4	4 to 32 (increment: 1 GiB)
8	8 to 64 (increment: 4 GiB)
16	16 to 128 (increment: 8 GiB)
32	32, 64, 128, and 256

Container vCPUs	Container Memory (GiB)
48	96, 192, and 384
64	128, 256, and 512

There are two ways to specify pods flavors.

Method 1: Specifying the Memory and vCPUs for the Pod

- Use the pod annotation **resource.cci.io/pod-size-specs** to specify the pod flavor. The specified pod flavor must be the same as one listed in **Table 4-4**, or the API for creating a pod returns an error message and refuses to create the pod. The value of **resource.cci.io/pod-size-specs** can be in the format of *${CPU}_${MEM}*, for example, **2.00_4.0**.
- The total vCPUs and memory of containers in a pod cannot exceed the pod flavor specified by **resource.cci.io/pod-size-specs**, or the API for creating a pod returns an error message and refuses to create the pod.

Method 2: Specifying vCPUs and Memory of Containers

- If the pod flavor is not specified, the memory and vCPUs required by the pod are calculated based on the total vCPUs and memory of containers in the pod. If the calculated result matches a flavor listed in **Table 4-4**, this flavor is used for the pod. If the calculated result does not match any listed flavor, it will be automatically upgraded to the next available flavor. Assume that the calculated result is 6 vCPUs and 15-GiB memory. The pod flavor will be automatically upgraded to 8 vCPUs and 16-GiB memory.
- The pod flavor is calculated as follows:

  Pod flavor = max(max(sum(spec.Containers.Request), container.Limit), max(spec.initContainer))

  sum(spec.Containers.Request): total vCPU requests and memory requests of all containers

  max(sum(spec.Containers.Request), container.Limit): the larger values between the total requests and the limits of each container

  max(spec.initContainer): the larger values between vCPU requests and limits and the larger values between memory requests and limits

  max(max(sum(spec.Containers.Request), container.Limit), max(spec.initContainer resource value)): the larger values between container resource specifications and init container resource specifications
- The rules for upgrading pod flavors as follows:

  The pod flavor is upgraded to the next available flavor, and the vCPUs and memory must not exceed those allowed in the pod flavor. If the pod flavor cannot be upgraded, the pod fails to be created.

  The upgraded pod flavor is included in the pod annotation: **resource.cci.io/size=***${cpuCeil}_${memoryCeil}*.

  If the number of vCPUs is greater than 32, the pod flavor will not be upgraded. If the vCPUs of containers in the pod do not match any pod flavor in **Table 4-4**, the pod fails to be created. To upgrade the pod flavor to one

with more than 32 vCPUs, set **resource.cci.io/resizing-large-size-instance-greater-than-32-cores** to **true** in the pod annotation.

CAUTION

- If **resource.cci.io/pod-size-specs** is not set for the pod and requests and limits are not specified for all containers in the pod, an error message is returned, and the pod fails to be created.
- The vCPUs and memory of each container must meet the Kubernetes validation requirements. If both requests and limits are specified, both requests and limits must be greater than or equal to 0, and the requests must be greater than or equal to the limits.
- A maximum of 10 containers can be created for each pod. If you have special requirements, you can apply for removing this restriction.

Examples of Automatic Pod Flavor Upgrades

Table 4-5 Automatic upgrades

Scenario	Container Settings	Resource Requests or Limits	Upgraded Pod Flavor	Description
Single container, with only requests specified	containers: - resources: requests: cpu: '1.5' memory: 2Gi	1.5 vCPUs and 2 GiB of memory	2 vCPUs and 4 GiB of memory	The pod flavors are upgraded based on the requests.
Single container, with both requests and limits specified	containers: - resources: limits: cpu: '3.5' requests: cpu: '1.5' memory: 4.5Gi	3.5 vCPUs and 4.5 GiB of memory	4 vCPUs and 5 GiB of memory	The requests and limits are compared, and the larger values are used.
Multiple containers, with only requests specified	containers: - resources: requests: cpu: '1.5' memory: 2Gi - resources: requests: cpu: '1.5' memory: 2Gi	3 vCPUs and 4 GiB of memory	4 vCPUs and 4 GiB of memory	The sum of requests is used.

Scenario	Container Settings	Resource Requests or Limits	Upgraded Pod Flavor	Description
Multiple containers, with only limits specified	containers: - resources: limits: cpu: '1.5' memory: 2Gi - resources: limits: cpu: '1.5' memory: 2Gi	1.5 vCPUs and 2 GiB of memory	2 vCPUs and 4 GiB of memory	The maximum limit is used.
Multiple containers, with both requests and limits specified	containers: - resources: limits: cpu: '5' memory: 8Gi requests: cpu: '1.5' memory: 2Gi - resources: requests: cpu: '1.5' memory: 2Gi	5 vCPUs and 8 GiB of memory	8 vCPUs and 8 GiB of memory	The requests and limits are compared, and the larger values are used.
Containers + init containers, with a large number of container resources	initContainers: - resources: requests: cpu: '1.5' memory: 2Gi containers: - resources: requests: cpu: '3' memory: 4Gi	3 vCPUs and 4 GiB of memory	4 vCPUs and 4 GiB of memory	The requests and limits of containers are compared with those of init containers, and the larger values are used.

Scenario	Container Settings	Resource Requests or Limits	Upgraded Pod Flavor	Description
Containers + init containers, with a large number of init container resources	initContainers: - resources: requests: cpu: '5' memory: 8Gi - resources: requests: cpu: '6' memory: 9Gi containers: - resources: requests: cpu: '3' memory: 4Gi	6 vCPUs and 9 GiB of memory	8 vCPUs and 12 GiB of memory	The requests and limits of containers are compared with those of init containers, and the larger values are used.
Resource requests, same as those in the flavor	containers: - resources: requests: cpu: '48' memory: 96Gi	48 vCPUs and 96 GiB of memory	48 vCPUs and 96 GiB of memory	There is no need to upgrade to the flavor.
Upgrading to 32 vCPUs	containers: - resources: requests: cpu: '31' memory: 32Gi	31 vCPUs and 32 GiB of memory	32 vCPUs and 32 GiB of memory	Fewer than 32 vCPUs after the upgrade
More than 32 vCPUs after the upgrade	containers: - resources: requests: cpu: '33' memory: 96Gi	33 vCPUs and 96 GiB of memory	No matched flavor	If the vCPU request is more than 32 vCPUs, no upgrade is performed.
More than 32 vCPUs, and upgrading to a large flavor allowed	annotations: "resource.cci.io/resizing-large-size-instance-greater-than-32-cores": "true" containers: - resources: requests: cpu: '33' memory: 96Gi	33 vCPUs and 96 GiB of memory	48 vCPUs and 96 GiB of memory	Upgrading to a large flavor is allowed.

> **CAUTION**
>
> When a workload pod that has multiple containers is scheduled from CCE to CCI and only limits are set for the containers, the pod creation complies with the default Kubernetes resource configuration rule as follows:
>
> If only limits are specified, requests that have the same values as limits will be automatically specified.
>
> Take a Deployment created on the CCE console as an example. The **spec.template** file of the Deployment contains two containers, and only limits are set for each container. After a pod is created, the requests of each container are specified and equal to the limits.
>
> The pod flavor may be greater than expected. You can manually specify the requests for the containers to avoid this issue.

4.1.2.3.2 Reserved System Overhead

Some necessary system components required by the pods occupy system resources. There is a difference between the memory in the pod flavor and the allocatable pod memory. CCI calculates the pod memory that can be allocated as follows:

- Memory in the pod flavor ≤ 2 GiB

  Allocatable pod memory = **Memory in the pod flavor**
- Memory in the pod flavor > 2 GiB

  Allocatable pod memory = **Memory in the pod flavor** – **Memory reserved for CCI** – **Memory reserved for the OS**

> **CAUTION**
>
> **Memory in the pod flavor** is the value of **${memoryCeil}** displayed in the pod annotation in the format of *resource.cci.io/size=${cpuCeil}_${memoryCeil}*.

Rules for Reserving Pod Memory on CCI

The total reserved pod memory is equal to the sum of that reserved for the OS and that reserved for CCI to manage the pod.

The memory reserved for the OS includes basic memory and floating memory that varies with the memory in the pod flavor. The memory reserved for CCI to manage the pod is 250 MiB.

Table 4-6 Rules for reserving pod memory

Scenario	Reserved for	Basic/ Floating	Reservation	Used by
Memory in the pod flavor ≤ 2 GiB	None	\	\	\
Memory in the pod flavor > 2 GiB	OS	Basic	Fixed at 150 MiB	OS services
		Floating reservation (varying with the memory in the pod flavor)	20 MiB per GiB	OS kernel
	CCI	Basic	Fixed at 250 MiB	CCI components

Example

Table 4-7 Reservation example

Memory in the Pod Flavor	Available Pod Memory	Calculation	Remarks
0.5 GiB	0.5 GiB	/	Memory in the pod flavor ≤ 2 GiB
1 GiB	1 GiB	/	
2 GiB	2 GiB	/	
3 GiB	2.55 GiB	3 GiB – 250 MiB – (150 MiB + 3 × 20 MiB) = 2.55 GiB	Memory in the pod flavor > 2 GiB
4 GiB	3.53 GiB	4 GiB – 250 MiB – (150 MiB + 4 × 20 MiB) = 3.53 GiB	
8 GiB	7.45 GiB	8 GiB – 250 MiB – (150 MiB + 8 × 20 MiB) = 7.45 GiB	
16 GiB	15.29 GiB	16 GiB – 250 MiB – (150 MiB + 16 × 20 MiB) = 15.29 GiB	

Memory in the Pod Flavor	Available Pod Memory	Calculation	Remarks
32 GiB	30.98 GiB	32 GiB – 250 MiB – (150 MiB + 32 × 20 MiB) = 30.98 GiB	
64 GiB	62.35 GiB	64 GiB – 250 MiB – (150 MiB + 64 × 20 MiB) = 62.35 GiB	
128 GiB	125.1 GiB	128 GiB – 250 MiB – (150 MiB + 128 × 20 MiB) = 125.1 GiB	
256 GiB	250.6 GiB	256 GiB – 250 MiB – (150 MiB + 256 × 20 MiB) = 250.6 GiB	

4.1.2.3.3 Increasing Reserved System Overhead

For pods running on CCI, the OS and CCI occupy some underlying resources. In some extreme scenarios, the actual memory usage of a pod may not reach the memory in the pod flavor. To solve this issue, two annotations are provided to reserve the overhead for system components.

The **resource.cci.io/memory-reservation** annotation specifies whether to enable system overhead reservation. The default value is **false**, and the value **true** indicates that system overhead reservation is enabled. After this option is enabled, CCI reserves 1 GiB of memory for the system components. If the memory exceeds that in the current pod flavor, the pod flavor is automatically rounded up to a flavor that is lightly larger than the current flavor.

The **resource.cci.io/memory-burst-size** annotation specifies whether to increase the container memory limits to the memory in the pod flavor. The default value is **false**, and the value **true** indicates that this option is enabled. After this option is enabled, the memory limits of all containers in the pod are adjusted to the memory in the pod flavor to increase the upper limit of the cgroup memory in the host environment. After system overhead reservation is enabled, the memory limited by cgroup can be released for application containers after the pod flavor is rounded up.

Table 4-8 Example

Scenario	Pod Configuration	Resource Requests	Rounded-up Pod Flavor	Final Pod Flavor on CCI	Description
The pod has only one container, and only resource requests are configured. System overhead reservation and increasing memory limits to the memory in the pod flavor are enabled.	annotations: "resource.cci.io/memory-reservation": "true" "resource.cci.io/memory-burst-size": "true" ... containers: - resources: requests: cpu: '1.5' memory: 2Gi	1.5 vCPUs and 3 GiB of memory	2 vCPUs and 4 GiB of memory	containers: - resources: limits: memory: 4Gi requests: cpu: '1.5' memory: 2Gi	How to calculate the required resources: 1. The pod flavor is rounded up based on the resource requests (1.5c2Gi). 2. System overhead reservation is enabled, and 1 GiB of memory (1.5c3Gi) is added. 3. The pod flavor is rounded up (2c4Gi) based on the rules described in **Upgrading Pod Flavors**. 4. The container memory limits are increased to the memory (4 GiB) in the pod flavor.

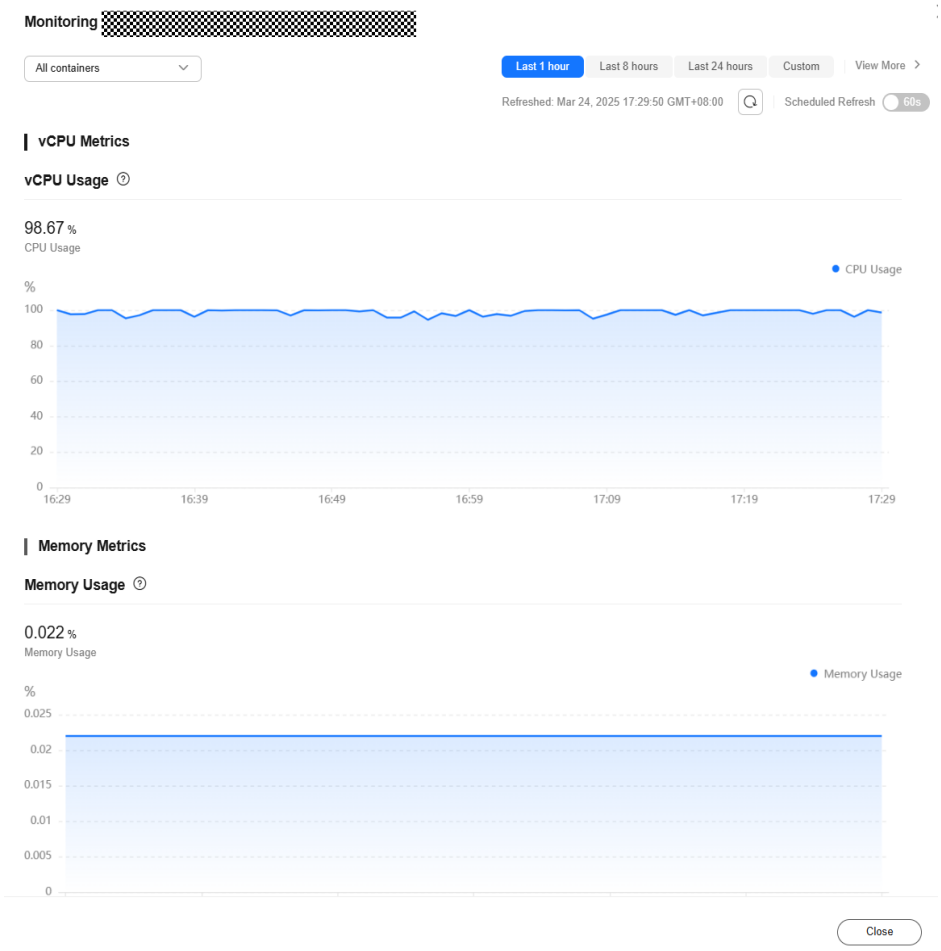
4.1.3 Viewing Resource Usages

If you enable AOM when creating a namespace, you can log in to the CCI console to view the resource usages of each pod after a workload is created.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 Locate the Deployment or pod and click **Monitor** in the **Operation** column to view the resource usages.

Figure 4-3 Viewing monitoring data



----End

4.1.4 Lifecycle

4.1.4.1 Startup Command

Starting the container is to start the main process. However, some preparations must be made before the main process is started. For example, you configure or initialize MySQL databases before running MySQL servers. You can set **ENTRYPOINT** or **CMD** in the Dockerfile when you create an image. As shown in

the following, the **ENTRYPOINT ["top", "-b"]** command is set in the Dockerfile. This command will be executed during container startup.

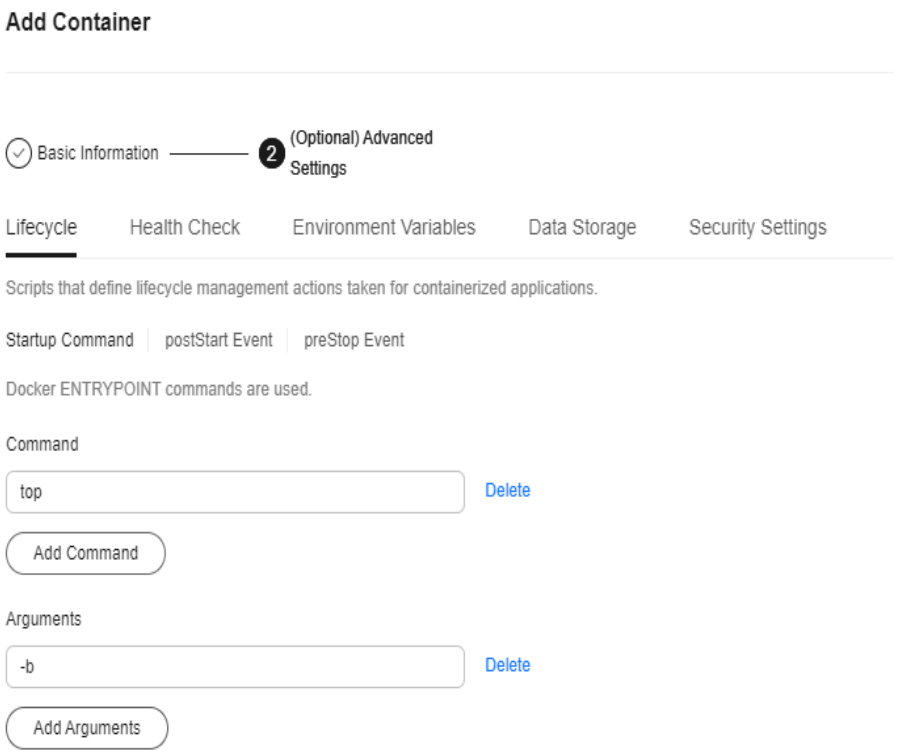
```
FROM ubuntu
ENTRYPOINT ["top", "-b"]
```

NOTICE

The startup command must be supported by the container image. Otherwise, the container startup will fail.

You can also set the container startup command. For example, to add the preceding command in the Dockerfile, you can click **Add** and enter the **top** command, and then click **Add** again and enter **-b** in the **Advanced Settings** area when you create a workload.

Figure 4-4 Startup command



When the container engine runs, only one **ENTRYPOINT** command is supported. The startup command that you set in CCI will overwrite the **ENTRYPOINT** and **CMD** commands that you set in the Dockerfile when you created the image. The following table lists the rules.

Image Entrypoint	Image CMD	Command for Running a Container	Parameter for Running a Container	Command Executed
[touch]	[/root/test]	Not set	Not set	[touch /root/test]
[touch]	[/root/test]	[mkdir]	Not set	[mkdir]
[touch]	[/root/test]	Not set	[/opt/test]	[touch /opt/test]
[touch]	[/root/test]	[mkdir]	[/opt/test]	[mkdir /opt/test]

4.1.4.2 PostStart/PreStop Processing

Setting Container Lifecycle Parameters

CCI provides containers with **lifecycle hooks**, which enable containers to run code triggered by events during their lifecycle. For example, if you want a container to perform a certain operation before it is stopped, you can register a hook. The following lifecycle hooks are provided:

- PostStart: triggered immediately after the container is started
- PreStop: triggered immediately before the container is stopped

NOTE

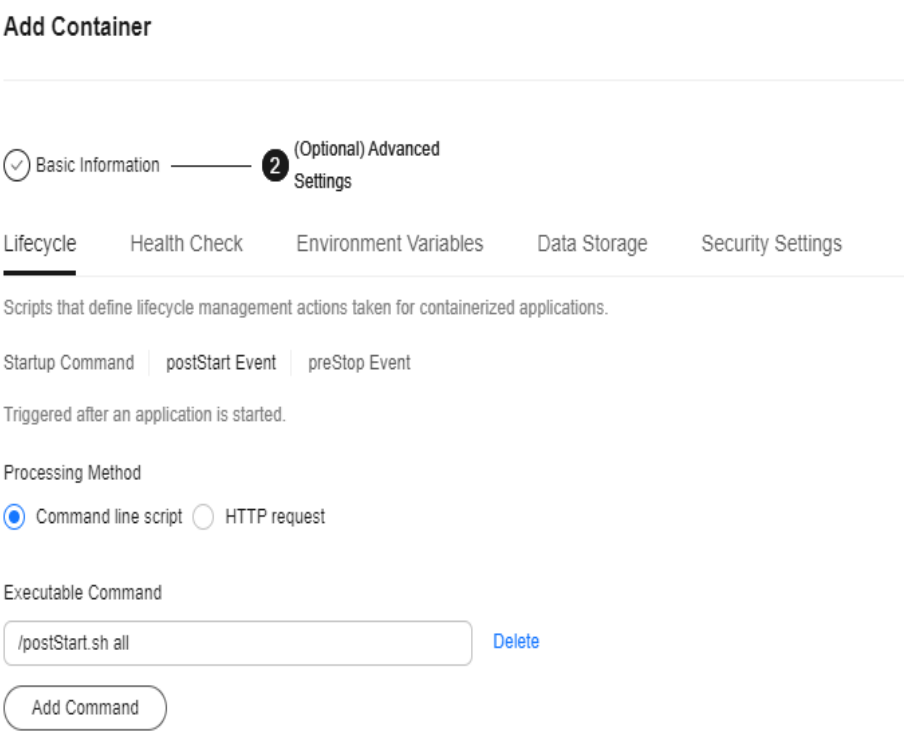
CCI supports only hook handlers of the Exec type, which execute a specific command.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 On the **Lifecycle** tab, configure PostStart or PreStop.

For example, if you want to run the **/PostStart.sh all** command in the container, configure data on the page shown in the following figure. The first row indicates the script name and the second row indicates a parameter setting.

Figure 4-5 Command settings



----End

4.1.5 Health Check

The health of a running container can be checked regularly.

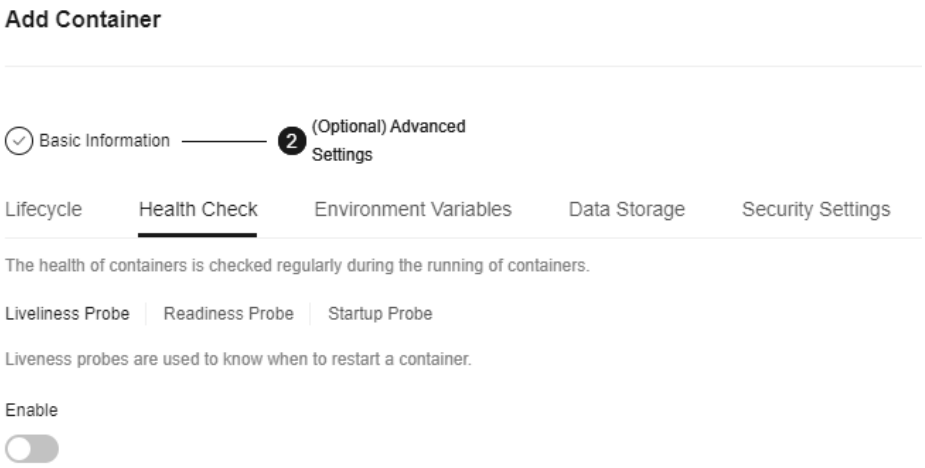
CCI supports the following types of probes:

- Liveness probe: checks whether a containerized application is alive. The liveness probe is similar to the **ps** command for checking whether a process is running. If the application fails the check, the container will be restarted. If the application passes the check, no operation will be performed.
- Readiness probe: checks whether a containerized application is ready to handle requests. An application may take a long time to start up and provide services due to some reasons, for example, it needs to load disk data or wait for the startup of an external module. In this case, application processes are running, but the application is not ready to provide services. This is where the readiness probe is useful.
- Startup probe: checks whether a container is started successfully. It checks when the container is started to ensure that the application can be started and run normally.

Health Check Methods

Step 1 Log in to the **CCI 2.0 console**.

Step 2 On the **Health Check** tab, configure the liveness probe, readiness probe, or startup probe.



- **HTTP request**

  The probe sends an HTTP GET request to the container. If the probe receives a 2xx or 3xx status code, the container is healthy.

- **TCP connection**

  The probe sends a TCP request to the container. If the probe receives a 2xx or 3xx status code, the container is healthy.

- **Command**

  The probe runs a command in the container and checks the exit status code. If the exit status code is 0, the container is healthy.

  For example, if you want to run the **cat /tmp/healthy** command to check whether the **/tmp/healthy** directory exists, configure data as shown in the following figure.

----End

Common Parameters

Table 4-9 Health check parameters

Parameter	Description
Interval (s)	Specifies the interval for performing a health check, in seconds. For example, if this parameter is set to **10**, the health check is performed every 10 seconds.
Delay (s)	Specifies the time required for the probe to be initiated after the container has started, in seconds. For example, if you set this parameter to **10**, the probe is initiated within 10 seconds after the container is started.

Parameter	Description
Timeout (s)	Specifies the amount of time after which the probe times out, in seconds. For example, if you set this parameter to **10**, the container must return a response within 10 seconds. Otherwise, the probe is counted as failed. If you set this parameter to **0** or do not specify any value, the default value (1 second) is used. **NOTE** If the executed command generates a subprocess in the image, **timeoutSeconds** may not take effect.
Success Threshold	Specifies the number of consecutive successful health checks for a container to be considered healthy. For example, if this parameter is set to **1**, the health check is successful if the probe succeeds for one time after a probe failure.
Failure Threshold	Specifies the number of consecutive failed health checks for a container to be considered unhealthy. For example, if this parameter is set to **3**, the health check fails after the probe fails for three consecutive times. In this case, the container is considered unhealthy and will be restarted.

4.1.6 Web Terminal

A web terminal allows you to connect to the containers for debugging.

Constraints

- By default, sh is used for login so containers must support sh.
- You can only log in to running containers using the terminal.
- To exit the terminal, you need to enter **exit**, or the sh process will not be terminated.

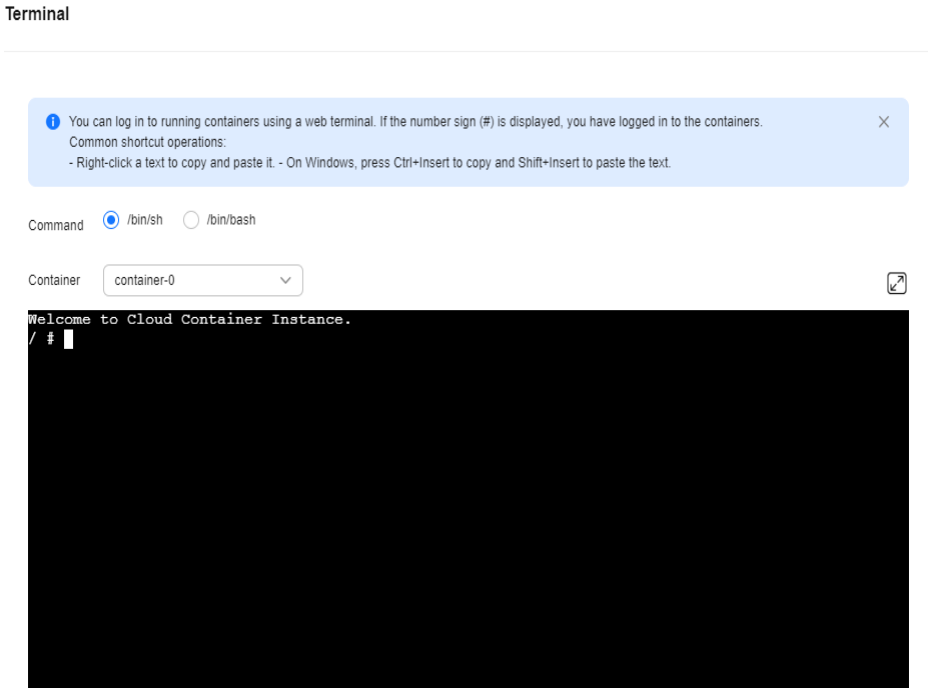
Connecting to the Container Using the Terminal

Step 1 In the navigation pane, choose **Workloads**. Then click the **Pods** tab.

Step 2 Locate the target pod and click **View Terminal** in the **Operation** column.

If **#** is displayed, the login is successful.

Figure 4-6 Terminal



----End

4.1.7 Workload Upgrade

You can upgrade a workload after you create it. There are two ways to upgrade a workload.

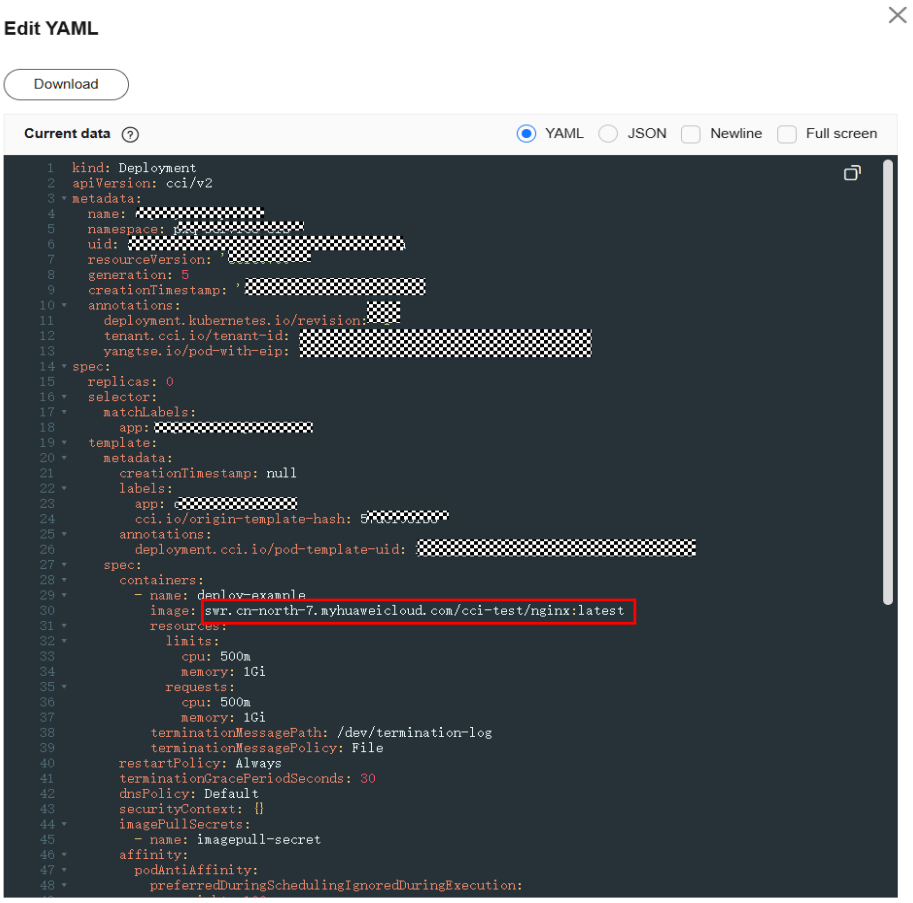
- **Rolling upgrade**: gradually replaces old pods with new pods. During the upgrade, service traffic is evenly distributed to the old and new pods to ensure service continuity.
- **In-place upgrade**: deletes an old pod and then creates a new one. Services will be interrupted during the upgrade.

Upgrading a Workload

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Workloads**. On the **Deployments** tab, locate the target workload and click **Edit YAML** in the **Operation** column.

Step 3 Modify the fields in the YAML file to upgrade the workload. For example, to update an image, modify the fields highlighted in the following figure:



Step 4 Click **OK**.

----End

4.2 Network Management

4.2.1 Services

4.2.1.1 Service Overview

A Service is a resource used to define a pod network access interface. It allows a group of pods to be accessed in a stable manner so you do not need to consider the location and quantity of the pods. CCI supports Services of the LoadBalancer type.

Workloads can be accessed over a private or public network, and they can also access the public network.

- **Private network access**: Create a Service of the LoadBalancer type, configure a private network load balancer, and use the private IP address of the private network load balancer to access the workload. You can also configure a

trustlist and blocklist for access control. This method can be used in the following scenarios: mutual access between workloads in the same namespace, mutual access between other cloud resources (such as ECSs) and CCI workloads in the same VPC, and mutual access between workloads in different namespaces of the same VPC. Services are provided over the private network through the private IP address and port of the load balancer in the format of *<private-IP-address>:<port>*.

- **Public network access**: Create a Service of the LoadBalancer type and configure a public network load balancer. You can use the public IP address and port of the load balancer to access the workload from the public network. You can also configure a trustlist or blocklist for access control.

Constraints on Services

- For Services of the LoadBalancer type, only dedicated load balancers are available. If a Service of the LoadBalancer type is used, the pods can have IPv4 IP addresses.
- If the CCE Cloud Bursting Engine for CCI add-on is used to schedule the workloads to CCI, dedicated load balancers can be configured for ingresses and Services of the LoadBalancer type. The CCE Cloud Bursting Engine for CCI add-on does not support Services of the LoadBalancer type if its version is earlier than 1.5.5.

4.2.1.2 Private Network Access

Overview

If you create a Service of the LoadBalancer type and configure a private network load balancer for the Service, you can use the private IP address and port of the load balancer to access the workload. This method can be used in the following scenarios: mutual access between workloads in the same namespace, mutual access between other cloud resources (such as ECSs) and CCI workloads in the same VPC, and mutual access between workloads in different namespaces of the same VPC. Services are provided over the private network through the private IP address and port of the load balancer in the format of *<private-IP-address>:<port>*.

Workloads run in pods. Accessing a workload is to access the pods for that workload.

Constraints

- The load balancer must be in the same VPC as the workload.
- Only dedicated load balancers are supported.

Creating a Service for an Existing Workload

You can create a Service for a workload after it is created. Creating a Service has no impact on the workload. Once created, the Service can be used by the workload for network access immediately.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Services**. On the right of the page, click **Create from YAML**.

Step 3 Import or add a YAML file.

The following is an example YAML file:

- Resource description in the **service.yaml** file

```
apiVersion: cci/v2
kind: Service
metadata:
  name: kubectl-test
  namespace: kubectl
  annotations:
    kubernetes.io/elb.class: elb
    kubernetes.io/elb.id: 1234567890 # Load balancer ID. Only dedicated load balancers are supported.
spec:
  selector:
    app: kubectl-test # Label of the associated workload
  ports:
    - name: service-0
      targetPort: 80   # Container port
      port: 12222      # Access port (load balancer's port for accessing the workload)
      protocol: TCP    # Protocol used to access the workload
  type: LoadBalancer
```

- Resource description in the **service.json** file

```
{
    "apiVersion": "cci/v2",
    "kind": "Service",
    "metadata": {
        "name": "kubectl-test",
        "namespace": "kubectl",
        "annotations": {
                    "kubernetes.io/elb.class": "elb",
            "kubernetes.io/elb.id": "1234567890"  # Load balancer ID. Only dedicated load balancers are supported.
        }
    },
    "spec": {
        "selector": {
            "app": "kubectl-test" # Label of the associated workload
        },
        "ports": [
            {
                "name": "service-0",
                "targetPort": 80,     # Container port
                "port": 12222,        # Access port (load balancer's port for accessing the workload)
                "protocol": "TCP",    # Protocol used to access the workload
                "type": "LoadBalancer"
            }
        ]
    }
}
```

Step 4 Click **OK**. Access the workload through the load balancer's private IP address and port in the format of *<private-IP-address>:<port>*.

----End

Updating a Service

After you add a Service, you can update the access port of the Service.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Services**. On the **Services** page, select the target namespace, locate the Service and click **Edit YAML** in the **Operation** column.

Step 3 Only the access port can be modified.

spec.ports[i].port indicates the access port. The port number ranges from **1** to **65535**.

Step 4 Click **OK**. The Service will be updated for the workload.

----End

4.2.1.3 Public Network Access

Overview

Workloads can be accessed from the public network. For this to work, you need to create a Service of the LoadBalancer type and create a public network load balancer in the same VPC as the workload.

Constraints

- The load balancer must be in the same VPC as the workload.
- You must familiarize yourself with the constraints on EIPs. For details, see **EIP Notes and Constraints**.
- Only dedicated load balancers are supported, and each load balancer must have an EIP bound.

Creating a Service for an Existing Workload

You can create a Service for a workload after it is created. Creating a Service has no impact on the workload. Once created, the Service can be used by the workload for network access immediately.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Services**. On the right of the page, click **Create from YAML**.

Step 3 Import or add a YAML file.

The following is an example YAML file.

- Resource description in the **service.yaml** file

```
apiVersion: cci/v2
kind: Service
metadata:
  name: kubectl-test
  namespace: kubectl
  annotations:
    kubernetes.io/elb.class: elb
    kubernetes.io/elb.id: 1234567890 # Load balancer ID. Only dedicated load balancers are supported.
spec:
  selector:
    app: kubectl-test # Label of the associated workload
  ports:
    - name: service-0
      targetPort: 80   # Container port
```



```
    port: 12222      # Access port (load balancer's port for accessing the workload)
    protocol: TCP     # Protocol used to access the workload
  type: LoadBalancer
```

- Resource description in the **service.json** file

```
{
   "apiVersion": "cci/v2",
   "kind": "Service",
   "metadata": {
      "name": "kubectl-test",
      "namespace": "kubectl",
      "annotations": {
                  "kubernetes.io/elb.class": "elb",
         "kubernetes.io/elb.id": "1234567890"  # Load balancer ID. Only dedicated load balancers are
supported.
      }
   },
   "spec": {
      "selector": {
         "app": "kubectl-test" # Label of the associated workload
      },
      "ports": [
         {
            "name": "service-0",
            "targetPort": 80,     # Container port
            "port": 12222,        # Access port (load balancer's port for accessing the workload)
            "protocol": "TCP",    # Protocol used to access the workload
            "type": "LoadBalancer"
         }
      ]
   }
}
```

Step 4 Click **OK**. Access the workload through the load balancer's EIP and port in the format of *<EIP-of-the-load-balancer>:<port>*.

----End

What If a Workload Cannot Be Accessed from the Public Network?

- A workload can only be accessed from the public network when it is in the running state. If your workload is abnormal or not ready, it cannot be accessed from the public network.
- It may take one to three minutes from the time when the workload was created to the time for it to be ready for access from the public network. During this time period, the network routes have not yet been configured. As a result, the workload cannot be accessed from the public network.
- If a workload is inaccessible 3 minutes after it is created, and there is no alarm event, a possible cause is that the container port is not being listened to. You need to use the image to check whether the container port is being listened to. If the container port is being listened to, the access failure may be caused by the load balancer. In this case, you need to check the load balancer.

Updating a Service

After you add a Service, you can update the access port of the Service.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Services**. On the **Services** page, select the target namespace, locate the Service and click **Edit YAML** in the **Operation** column.

Step 3 Only the access port can be modified.

spec.ports[i].port: indicates the access port. The port number ranges from **1** to **65535**.

Step 4 Click **OK**. The Service will be updated for the workload.

----End

4.2.1.4 Configuring HTTP/HTTPS for a LoadBalancer Service

4.2.1.4.1 Configuring HTTP for a LoadBalancer Service

Configuring an HTTP Service

You can create a Service for a workload after it is created. Creating a Service has no impact on the workload. Once created, the Service can be used by the workload for network access immediately.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Services**. On the right of the page, click **Create from YAML**.

Step 3 Import or add a YAML file.

The following is an example YAML file.

- Resource description in the **service.yaml** file:

```
apiVersion: cci/v2
kind: Service
metadata:
  name: kubectl-test
  namespace: kubectl
  annotations:
    kubernetes.io/elb.class: elb
    kubernetes.io/elb.id: 1234567890 # Load balancer ID. Only dedicated load balancers are supported.
    kubernetes.io/elb.protocol-port: "http:80"  # HTTP and port number, which must be the same as the port number in spec.ports
spec:
  selector:
    app: kubectl-test # Label of the associated workload
  ports:
    - name: service-0
      targetPort: 80   # Container port
      port: 80        # Access port (load balancer's port for accessing the workload)
      protocol: TCP    # Set the protocol to TCP.
  type: LoadBalancer
```

- Resource description in the **service.json** file:

```
{
    "apiVersion": "cci/v2",
    "kind": "Service",
    "metadata": {
        "name": "kubectl-test",
        "namespace": "kubectl",
        "annotations": {
                    "kubernetes.io/elb.class": "elb",
            "kubernetes.io/elb.id": "1234567890"  # Load balancer ID. Only dedicated load balancers are supported.
                    "kubernetes.io/elb.protocol-port": "http:80"  # HTTP and port number, which must be the same as the port number in spec.ports.
        }
    },
```



```
  "spec": {
    "selector": {
      "app": "kubectl-test" # Label of the associated workload
    },
    "ports": [
      {
        "name": "service-0",
        "targetPort": 80,     # Container port
        "port": 80,           # Access port of the Service
        "protocol": "TCP",    # Set the protocol to TCP.
        "type": "LoadBalancer"
      }
    ]
  }
}
```

Step 4 Click **OK**. Access the workload through the load balancer's IP address and port in the format of *<IP-address>:<port>*.

----End

Updating a Service

After you add a Service, you can update the access port of the Service.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Services**. On the **Services** page, select the target namespace, locate the Service, and click **Edit YAML** in the **Operation** column.

Step 3 Only the access port can be modified.

spec.ports[i].port: indicates the access port. The port number ranges from **1** to **65535**. You need to change the value of **kubernetes.io/elb.protocol-port** accordingly.

Step 4 Click **OK**. The Service will be updated for the workload.

----End

Parameters for an HTTP Service

You can refer to the following table to add the annotations to configure a listener for an HTTP Service.

Parameter	Description
kubernetes.io/elb.http2-enable	Whether HTTP/2 is enabled. Request forwarding using HTTP/2 improves the access performance between your application and the load balancer. However, the load balancer still uses HTTP/1.x to forward requests to the backend server. Value options: • **true**: enabled • **false**: disabled (default value) **CAUTION** HTTP/2 can be enabled or disabled only for HTTPS listeners. This option is invalid when the listeners use HTTP. The default value is **false**.
kubernetes.io/elb.tls-certificate-ids	IDs of SNI certificates used in ELB, separated by commas (,). Each SNI certificate must contain domain names. To obtain the value, log in to the ELB console and choose **Elastic Load Balance** > **Certificates**.
kubernetes.io/elb.security-pool-protocol	If the listener uses HTTPS, you can set the backend protocol to HTTPS. **The backend protocol of an existing listener cannot be changed. If you want to change the backend protocol, you need to add a new listener to the load balancer.** • **true**: enabled • **false**: disabled
kubernetes.io/elb.tls-ciphers-policy	Security policy used by a listener. Value options include **tls-1-0-inherit**, **tls-1-0**, **tls-1-1**, **tls-1-2**, **tls-1-2-strict**, **tls-1-2-fs**, **tls-1-0-with-1-3**, **tls-1-2-fs-with-1-3**, and **hybrid-policy-1-0**. The default value is **tls-1-0**. This option is available for HTTPS listeners of dedicated load balancers.
kubernetes.io/elb.x-forwarded-port	If this option is enabled, the listening port of the load balancer can be transferred to backend servers through the HTTP header of the packet. • **true**: enabled • **false**: disabled This option is available for HTTP/HTTPS listeners of dedicated load balancers.

Parameter	Description
kubernetes.io/elb.x-forwarded-for-port	If this option is enabled, the source port of the client can be transferred to backend servers through the HTTP header of the packet. • **true**: enabled • **false**: disabled This option is available for HTTP/HTTPS listeners of dedicated load balancers.
kubernetes.io/elb.x-forwarded-host	If this option is enabled, **X-Forwarded-Host** will be rewritten using the **Host** field in the request and transferred to backend servers. • **true**: enabled • **false**: disabled This option is available for HTTP/HTTPS listeners of dedicated load balancers.
kubernetes.io/elb.gzip-enabled	Data compression. • **true**: Data compression is enabled, and specific file types will be compressed. • **false**: Data compression is disabled, and no files will be compressed. By default, data compression is disabled. The files in the following format can be compressed: • Brotli can compress all file formats. • GZIP can compress the files of the following types: text/xml, text/plain, text/css, application/javascript, application/x-javascript, application/rss+xml, application/atom+xml, application/xml, and application/json. This option is available for HTTP/HTTPS listeners of dedicated load balancers.

4.2.1.4.2 Configuring HTTPS for a LoadBalancer Service

Configuring an HTTPS Service

You can create a Service for a workload after it is created. Creating a Service has no impact on the workload. Once created, the Service can be used by the workload for network access immediately.

Step 1 Log in to the ELB console and click **Create Certificate**.

Step 2 Log in to the **CCI 2.0 console**.

Step 3 In the navigation pane, choose **Services**. On the right of the page, click **Create from YAML**.

Step 4 Import or add a YAML file.

The following is an example YAML file.

- Resource description in the **service.yaml** file:

```
apiVersion: cci/v2
kind: Service
metadata:
  name: kubectl-test
  namespace: kubectl
  annotations:
    kubernetes.io/elb.class: elb
    kubernetes.io/elb.id: 1234567890 # Load balancer ID. Only dedicated load balancers are supported.
    kubernetes.io/elb.protocol-port: "http:443"  # HTTP and port number, which must be the same as that in spec.ports.
    kubernetes.io/elb.cert-id: "17e3b4f4bc40471c86741dc3aa211379"  # Certificate ID of the LoadBalancer Service
spec:
  selector:
    app: kubectl-test # Label of the associated workload
  ports:
    - name: service-0
      targetPort: 80   # Container port
      port: 443      # Access port (load balancer's port for accessing the workload)
      protocol: TCP    # Set the protocol to TCP.
  type: LoadBalancer
```

- Resource description in the **service.json** file:

```
{
    "apiVersion": "cci/v2",
    "kind": "Service",
    "metadata": {
        "name": "kubectl-test",
        "namespace": "kubectl",
        "annotations": {
                        "kubernetes.io/elb.class": "elb",
            "kubernetes.io/elb.id": "1234567890"  # Load balancer ID. Only dedicated load balancers are supported.
                        "kubernetes.io/elb.protocol-port": "https:443"  # HTTPS and port number, which must be the same as that in spec.ports.
                        "kubernetes.io/elb.cert-id": "17e3b4f4bc40471c86741dc3aa211379" # Certificate ID of the LoadBalancer Service
        }
    },
    "spec": {
        "selector": {
            "app": "kubectl-test" # Label of the associated workload
        },
        "ports": [
            {
                "name": "service-0",
                "targetPort": 80,     # Container port
                "port": 443,          # Access port of the Service
                "protocol": "TCP",    # Set the protocol to TCP.
                "type": "LoadBalancer"
            }
        ]
    }
}
```

Step 5 Click **OK**. Access the workload through the load balancer's IP address and port in the format of *<IP-address>:<port>*.

----End

Updating a Service

After you add a Service, you can update the access port of the Service.

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Services**. On the **Services** page, select the target namespace, locate the Service, and click **Edit YAML** in the **Operation** column.

Step 3 Only the access port can be modified.

spec.ports[i].port: indicates the access port. The port number ranges from **1** to **65535**. You need to change the value of **kubernetes.io/elb.protocol-port** accordingly.

Step 4 Click **OK**. The Service will be updated for the workload.

----End

Parameters for an HTTPS Service

You can refer to the following table to add the annotations to configure a listener for an HTTPS Service.

Parameter	Description
kubernetes.io/elb.http2-enable	Whether HTTP/2 is enabled. Request forwarding using HTTP/2 improves the access performance between your application and the load balancer. However, the load balancer still uses HTTP/1.x to forward requests to the backend server. Value options: • **true**: enabled • **false**: disabled (default value) **CAUTION** HTTP/2 can be enabled or disabled only for HTTPS listeners. This option is invalid when the listeners use HTTP. The default value is **false**.
kubernetes.io/elb.tls-certificate-ids	IDs of SNI certificates used in ELB, separated by commas (,). Each SNI certificate must contain domain names. To obtain the value, log in to the ELB console and choose **Elastic Load Balance** > **Certificates**.
kubernetes.io/elb.security-pool-protocol	If the listener uses HTTPS, you can set the backend protocol to HTTPS. **The backend protocol of an existing listener cannot be changed. If you want to change the backend protocol, you need to add a new listener to the load balancer.** • **true**: enabled • **false**: disabled

Parameter	Description
kubernetes.io/elb.tls-ciphers-policy	Security policy used by a listener. Value options include **tls-1-0-inherit**, **tls-1-0**, **tls-1-1**, **tls-1-2**, **tls-1-2-strict**, **tls-1-2-fs**, **tls-1-0-with-1-3**, **tls-1-2-fs-with-1-3**, and **hybrid-policy-1-0**. The default value is **tls-1-0**. This option is available for HTTPS listeners of dedicated load balancers.
kubernetes.io/elb.x-forwarded-port	If this option is enabled, the listening port of the load balancer can be transferred to backend servers through the HTTP header of the packet. • **true**: enabled • **false**: disabled This option is available for HTTP/HTTPS listeners of dedicated load balancers.
kubernetes.io/elb.x-forwarded-for-port	If this option is enabled, the source port of the client can be transferred to backend servers through the HTTP header of the packet. • **true**: enabled • **false**: disabled This option is available for HTTP/HTTPS listeners of dedicated load balancers.
kubernetes.io/elb.x-forwarded-host	If this option is enabled, **X-Forwarded-Host** will be rewritten using the **Host** field in the request and transferred to backend servers. • **true**: enabled • **false**: disabled This option is available for HTTP/HTTPS listeners of dedicated load balancers.

Parameter	Description
kubernetes.io/elb.gzip-enabled	Data compression. • **true**: Data compression is enabled, and specific file types will be compressed. • **false**: Data compression is disabled, and no files will be compressed. By default, data compression is disabled. The files in the following format can be compressed: • Brotli can compress all file formats. • GZIP can compress the files of the following types: text/xml, text/plain, text/css, application/javascript, application/x-javascript, application/rss+xml, application/atom+xml, application/xml, and application/json. This option is available for HTTP/HTTPS listeners of dedicated load balancers.

4.2.1.5 Configuring an Access Policy for a Service

Overview

You can add IP addresses to a trustlist or blocklist to control access to a load balancer associated with the Service.

- Trustlist: Only the IP addresses in the list can access the load balancer.
- Blocklist: IP addresses in the list are not allowed to access the load balancer.

Prerequisites

IP address groups have been created on the ELB console. For details, see **Creating an IP Address Group**.

Configuring an Access policy

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Services**. On the right of the page, click **Create from YAML**.

Step 3 Import or add the YAML file of the Service. For details about the parameters, see **Table 4-10**.

The following is an example YAML file:

- Resource description in the **service.yaml** file

```
apiVersion: cci/v2
kind: Service
metadata:
  name: kubectl-test
```



```
  namespace: kubectl
  annotations:
    kubernetes.io/elb.class: elb
    kubernetes.io/elb.id: 1234567890 # Load balancer ID. Only dedicated load balancers are supported.
    kubernetes.io/elb.acl-id: <your_acl_id>              # ID of an IP address group for accessing the load
balancer
    kubernetes.io/elb.acl-type: 'white'                  # Trustlist for access control
spec:
  selector:
    app: kubectl-test
  ports:
    - name: service-0
      targetPort: 80   # Container port
      port: 12222      # Access port (load balancer's port for accessing the workload)
      protocol: TCP    # Protocol used to access the workload
  type: LoadBalancer
```

- Resource description in the **service.json** file

```
{
    "apiVersion": "cci/v2",
    "kind": "Service",
    "metadata": {
        "name": "kubectl-test",
        "namespace": "kubectl",
        "annotations": {
                        "kubernetes.io/elb.class": "elb"
            "kubernetes.io/elb.id": "1234567890"  # Load balancer ID. Only dedicated load balancers are
supported.
                        kubernetes.io/elb.acl-id: <your_acl_id>              # ID of an IP address group for
accessing the load balancer
                        "kubernetes.io/elb.acl-type": "white"                # Trustlist for access control
        }
    },
    "spec": {
        "selector": {
            "app": "kubectl-test"
        },
        "ports": [
            {
                "name": "service-0",
                "targetPort": 80,     # Container port
                "port": 12222,        # Access port (load balancer's port for accessing the workload)
                "protocol": "TCP",    #Protocol used to access the workload
                "type": "LoadBalancer"
            }
        ]
    }
}
```

Step 4 Click **OK**. Access the workload through the load balancer's IP address and port in the format of *<IP-address>:<port>*.

- If a trustlist is used for access control, only the IP addresses in the trustlist can access the load balancer.
- If a blocklist is used for access control, the IP addresses in the blocklist cannot access the load balancer.

----End

Table 4-10 Annotations for ELB access control

Parameter	Type	Description
kubernetes.io/elb.acl-id	String	• If this parameter is not specified, CCI does not modify access control on ELB. • If this parameter is set to the IP address group ID of the load balancer, access control is enabled, and you need to configure an IP address blocklist or trustlist for the load balancer. • You can enter a maximum of five IP address group IDs separated by commas (,). • To obtain an IP address group ID, take the following steps: Log in to the console. In the **Service List**, choose **Networking** > **Elastic Load Balance**. On the Network Console, choose **Elastic Load Balance** > **IP Address Groups** and copy the **ID** of the target IP address group. For details, see **IP Address Group**.
kubernetes.io/elb.acl-type	String	This parameter is mandatory when you configure an IP address blocklist or trustlist for a load balancer. • **black**: The selected IP address group cannot access the load balancer. • **white**: Only the selected IP address group can access the load balancer. If **kubernetes.io/elb.acl-id** is specified but **kubernetes.io/elb.acl-type** is not, the trustlist is used by default.

4.2.2 Specifying a Subnet for a Pod

Scenario

If the pod network contains multiple subnets, you can use the **yangtse.io/subnets** annotation to specify the subnets after a pod is created.

Constraints

- A maximum of 20 subnets can be specified for a pod.
- The subnets specified for a pod must be included in the pod network configuration.
- Multiple subnets are separated using commas (,).
- There must be idle IP addresses in each subnet, or the pod fail to be created due to insufficient IP addresses.

Using kubectl

You can add annotations to a workload to specify the subnets for each pod.


```
apiVersion: cci/v2
kind: Deployment
metadata:
  annotations:
    description: ''
  labels: {}
  name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        vm.cci.io/pod-size-specs: 2.00_4.0
        resource.cci.io/pod-size-specs: 2.00_4.0
        metrics.alpha.kubernetes.io/custom-endpoints: '[{api:'''',path:'''',port:'''',names:''''}]'
        yangtse.io/subnets: ${subnetID1},${subnetID2}    # Subnets specified for the pod
        log.stdoutcollection.kubernetes.io: '{"collectionContainers": ["container-0"]}'
      labels:
        app: nginx
    spec:
      containers:
        - image: library/nginx:stable-alpine-perl
          name: container-0
          resources:
            limits:
              cpu: 2000m
              memory: 4096Mi
            requests:
              cpu: 2000m
              memory: 4096Mi
          command: []
          lifecycle: {}
      dnsPolicy: ''
      imagePullSecrets:
        - name: imagepull-secret
      dnsConfig: {}
  minReadySeconds: 0
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
```

NOTE

yangtse.io/subnets: IDs of the subnets to be specified for a pod

4.3 Storage Management

4.3.1 Overview

CCI supports both persistent storage and ephemeral storage to meet your requirements. You can use the following types of storage volumes when creating a workload.

Ephemeral Storage

By default, CCI allocates 30 GiB of free ephemeral storage space to a pod. The storage space is unavailable when the pod is not in use. If you need to use more

ephemeral storage space, you can expand the capacity. For details, see **Ephemeral Storage**.

SFS Turbo Volumes

CCI allows you to create SFS Turbo volumes and mount them to specific container paths. SFS Turbo volumes are fast, on-demand, and scalable. They are suitable for DevOps, containerized microservices, and enterprise office applications. For details, see **SFS Turbo Volumes**.

4.3.2 Ephemeral Storage

Scenarios

By default, CCI allocates 30 GiB of free ephemeral storage space to a pod. The storage space is unavailable when the pod is not in use. If the pod needs to write a large amount of data to rootfs or emptyDir or if the image size is greater than 30 GiB, you need to expand the ephemeral storage capacity.

Constraints

- The maximum ephemeral storage capacity is 994 GiB.
- When the ephemeral storage space is expanded, workload creation is slower than before because the capacity expansion takes time.

Using YAML

- When creating a pod, you need to add 10 GiB of ephemeral storage space. The YAML file is defined as follows:

```
apiVersion: cci/v2
kind: Pod
metadata:
  name: nginx
  namespace: ns
spec:
  extraEphemeralStorage:
    sizeInGiB: 10
  containers:
  - name: container-1
    image: nginx:latest
```

- When creating a Deployment, you need to add 10 GiB of ephemeral storage space. The YAML file is defined as follows:

```
apiVersion: cci/v2
kind: Deployment
metadata:
  name: nginx
  namespace: ns
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      extraEphemeralStorage:
        sizeInGiB: 10
      containers:
```

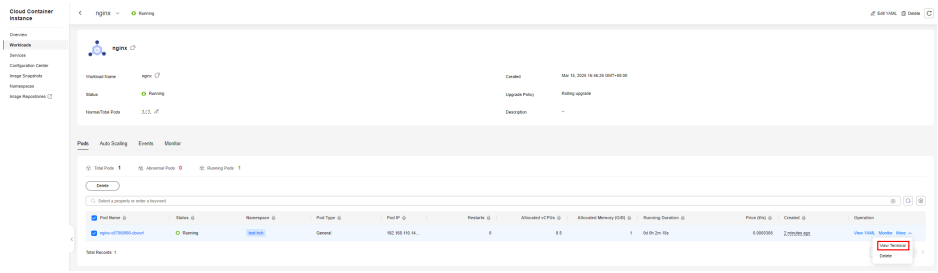


```
  - image: nginx:latest
    name: container-0
```

Verifying the Capacity Expansion

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Workloads**. Locate the target workload and click its name to go to the details page. On the **Pods** tab, locate the target pod and click **View Terminal**.



Step 3 Enter **lsblk** and press **Enter** to check the system disk size after the expansion. (The ephemeral storage capacity is 30 GiB by default.)

----End

4.3.3 SFS Turbo Volumes

CCI allows you to mount **SFS** Turbo volumes to container paths. SFS Turbo volumes are fast, on-demand, and scalable. They are suitable for DevOps, containerized microservices, and enterprise office applications.

Constraints

- SFS Turbo file systems are billed on a pay-per-use basis. For more information, see **SFS Turbo Pricing**.
- The SFS Turbo file system must be in the same VPC as the workload. If they are in different VPCs, the workload cannot use the SFS Turbo file system for persistent storage.

- If an SFS Turbo file system is in use, the VPC where the file system is deployed cannot be changed. Once the VPC is changed, the containers in CCI will not be able to access the file system.
- If an SFS Turbo file system is deleted, containers in CCI will become unavailable.
- SFS Turbo file systems do not involve AZs, so PVs of the SFS Turbo type do not support AZ affinity.

Importing SFS Turbo File Systems

Currently, SFS Turbo file systems can only be used by CCI containers through static PVC binding.

Step 1 Create an SFS Turbo file system. For details, see **Creating an SFS Turbo File System**.

Step 2 Create a PV.

1. Obtain the IAM token.

```
curl -i -k -H 'Accept:application/json' -H 'Content-Type:application/json;charset=utf8' -X POST -d '{"auth": {"identity":{"methods": ["password"],"password":{"user": {"name": "$username","password": "$password","domain": {"name": "$domain"}}}},"scope": {"project":{"name": "$project"}}}}' https://{iam-endpoint}/v3/auth/tokens | grep "X-Subject-Token"|awk -F ": " '{print $2}'|sed "s/\r//"
```

2. Call the API for creating a PV.

```
curl -H "X-Auth-Token:$token" -H 'Content-Type: application/json' -X POST -d @pv.json  -k -v https://{cci-endpoint}/apis/cci/v2/persistentvolumes
```

In this command, **$token** is the IAM token obtained in **Step 2.1**, and **pv.json** is the information about the PV to be created. The following is an example:

```
{
    "apiVersion": "cci/v2",
    "kind": "PersistentVolume",
    "metadata": {
        "annotations": {
        },
        "name": "pv-sfs-test"
    },
    "spec": {
        "accessModes": [
            "ReadWriteMany"
        ],
        "capacity": {
            "storage": "500Gi"
        },
        "csi": {
            "driver": "sfsturbo.csi.everest.io",
            "fsType": "nfs",
            "volumeHandle": "**************",
            "volumeAttributes": {
                "everest.io/share-export-location": "*******"
            }
        },
        "persistentVolumeReclaimPolicy": "Retain",
        "storageClassName": "csi-sfsturbo",
```

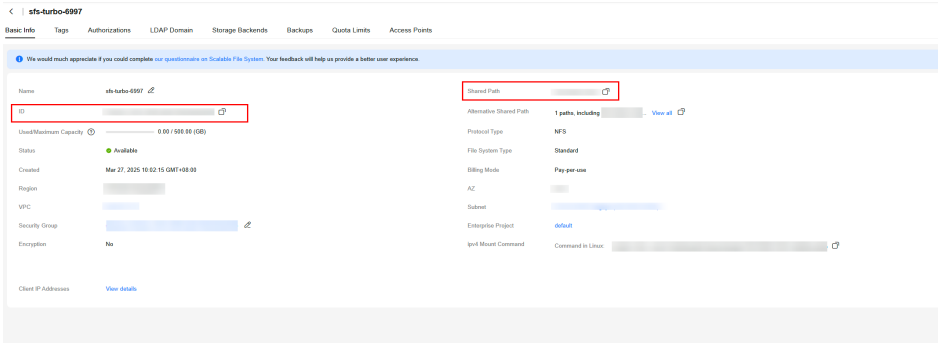


```
        "mountOptions": [
        ]
    }
}
```

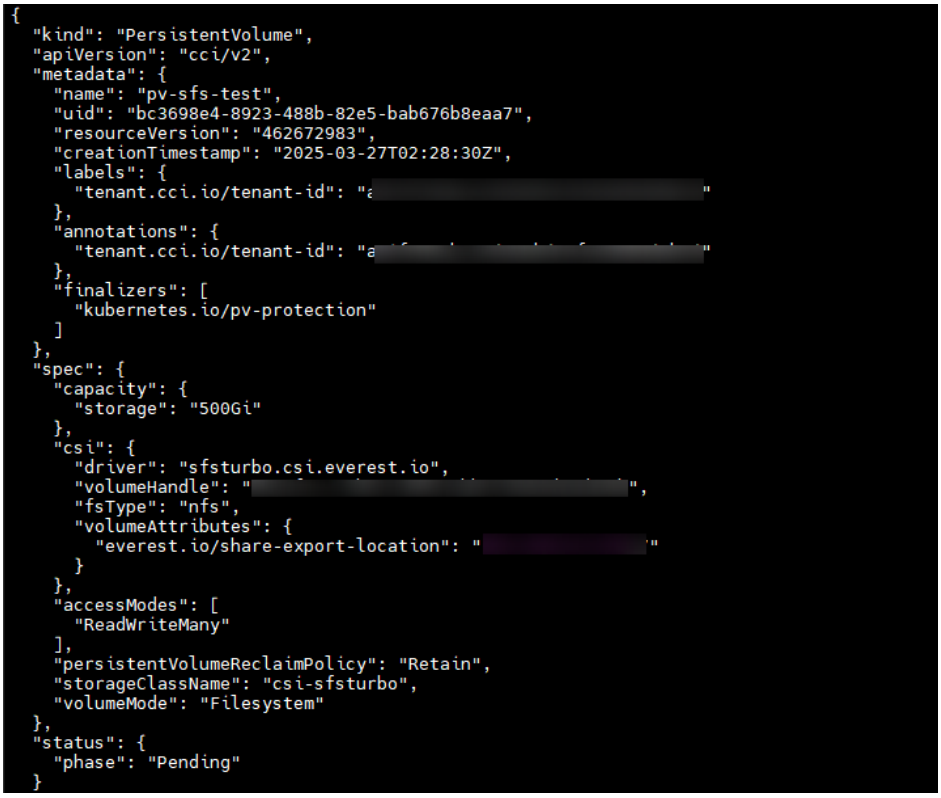
Table 4-11 Key parameters

Parameter	Mandatory	Type	Description
accessModes	Yes	List	Description: Storage access mode. Constraint: The value must be **ReadWriteMany** for SFS Turbo volumes.
driver	Yes	String	Description: Storage driver that the volume depends on. Constraint: The value must be **sfsturbo.csi.everest.io**.
fsType	Yes	String	Description: Storage instance type. Constraint: The value must be **nfs**, which indicates file system volumes.
volumeHandle	Yes	String	Description: ID of the SFS Turbo file system. Constraint: The ID must be that of an existing SFS Turbo file system.
persistentVolumeReclaimPolicy	Yes	String	Description: PV reclaim policy. Constraint: Only the **Retain** policy is supported. **Retain**: When a PVC is deleted, both the PV and underlying storage are retained. You need to manually delete these resources. After the PVC is deleted, the PV is in the **Released** state and cannot be bound to a PVC again.
storage	Yes	String	Description: Storage capacity, in Gi. Constraint: Set it to the size of the SFS Turbo file system.
storageClassName	Yes	String	Description: Storage class name of the SFS Turbo volume. Constraint: The storage class name of SFS Turbo volumes is **csi-sfsturbo**.

volumeHandle is the ID of the SFS Turbo file system created in **Step 1**, and **everest.io/share-export-location** is the shared path of the SFS Turbo file system.



The following figure shows the response to the request for creating a PV.



```
{
  "kind": "PersistentVolume",
  "apiVersion": "cci/v2",
  "metadata": {
    "name": "pv-sfs-test",
    "uid": "bc3698e4-8923-488b-82e5-bab676b8eaa7",
    "resourceVersion": "462672983",
    "creationTimestamp": "2025-03-27T02:28:30Z",
    "labels": {
      "tenant.cci.io/tenant-id": "a                                    "
    },
    "annotations": {
      "tenant.cci.io/tenant-id": "a                                    "
    },
    "finalizers": [
      "kubernetes.io/pv-protection"
    ]
  },
  "spec": {
    "capacity": {
      "storage": "500Gi"
    },
    "csi": {
      "driver": "sfsturbo.csi.everest.io",
      "volumeHandle": "                                                ",
      "fsType": "nfs",
      "volumeAttributes": {
        "everest.io/share-export-location": "                    "
      }
    },
    "accessModes": [
      "ReadWriteMany"
    ],
    "persistentVolumeReclaimPolicy": "Retain",
    "storageClassName": "csi-sfsturbo",
    "volumeMode": "Filesystem"
  },
  "status": {
    "phase": "Pending"
  }
```

Step 3 Create a PVC.

1. Obtain the IAM token. For details, see **Step 2.1**.
2. Call the API for creating a PVC and bind the created PV.

   ```
   curl -H "X-Auth-Token:$token" -H 'Content-Type: application/json' X POST -d @pvc.json  -k -v https://{cci-endpoint}/apis/cci/v2/namespaces/${namespace}/persistentvolumeclaims
   ```

   pvc.json indicates the information about the PVC to be created. The following is an example:

   ```
   {
       "apiVersion": "cci/v2",
       "kind": "PersistentVolumeClaim",
       "metadata": {
           "name": "pvc-sfs",
           "annotations": {
           }
   ```



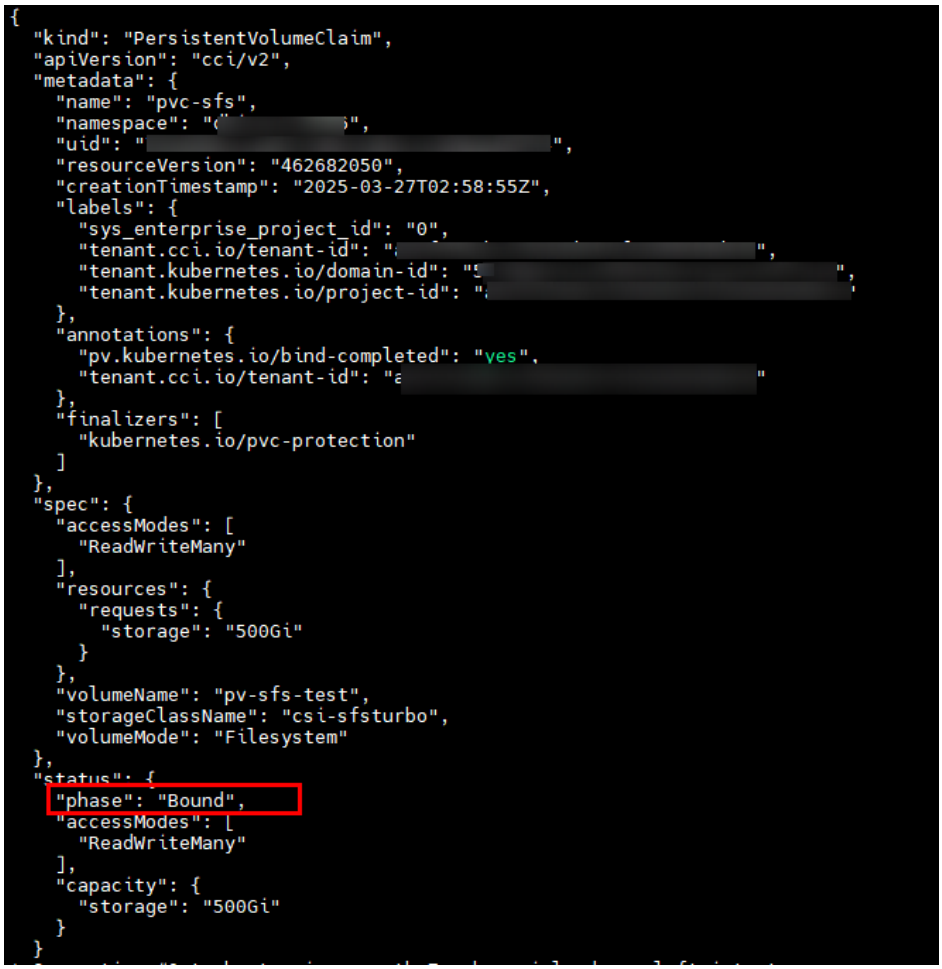
```
    },
    "spec": {
        "accessModes": [
            "ReadWriteMany"
        ],
        "resources": {
            "requests": {
                "storage": "500Gi"
            }
        },
        "storageClassName": "csi-sfsturbo",
        "volumeName": "pv-sfs-test"
    }
}
```

Table 4-12 Key parameters

Parameter	Mandatory	Type	Description
storage	Yes	String	Description: PVC capacity, in Gi. Constraints – Set it to the size of the SFS Turbo file system. – The value is the same as the capacity set for the PV in **Table 4-11**.
storageClassName	Yes	String	Description: Storage class name. Constraints: The value must be the same as the storage class of the PV in **Table 4-11**. The storage class name of SFS Turbo volumes is **csi-sfsturbo**.
volumeName	Yes	String	Description: PV name. Constraint: The value must be the same as the PV name in **Table 4-11**.

volumeName indicates the name of the PV created in **Step 3**.

The following figure shows the response to the request for creating a PVC, and the PVC has been bound to the PV.



----End

Using SFS Turbo Volumes

For details, see **Deployments**. Add the volume configuration to the workload YAML file.

```
kind: Deployment
apiVersion: cci/v2
metadata:
  name: nginx
  namespace: test-ns
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      volumes:
        - name: my-storage
          persistentVolumeClaim:
            claimName: PVC name
      containers:
        - name: nginx
          image: nginx:latest
```



```
        resources:
          limits:
            cpu: 500m
            memory: 1Gi
          requests:
            cpu: 500m
            memory: 1Gi
        volumeMounts:
          - name: my-storage
            mountPath: mount path of a container
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
    restartPolicy: Always
    terminationGracePeriodSeconds: 30
    dnsPolicy: Default
    securityContext: {}
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 25%
    maxSurge: 25%
```

NOTE

- When an SFS Turbo file system is being created, an independent VM will also be created, and this will take a long time. Therefore, you are advised to select existing SFS Turbo volumes.
- **subPath** is a sub-directory in the root path of the SFS Turbo file system. If there is no sub-directory, a sub-directory is automatically created in the SFS Turbo file system. Note that **subPath** must be a relative path.
- If SFS Turbo volumes are used, workloads can be created only using YAML.

4.4 Configuration Center

4.4.1 ConfigMaps

ConfigMaps are objects that you can use to store the configurations required by applications. After you create a ConfigMap, you can use it as a file in a containerized application.

Creating a ConfigMap

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Configuration Center**.

Step 3 Select a namespace and click the **ConfigMaps** tab.

Step 4 Click **Create from YAML** in the upper left corner and edit the YAML file. For details about the YAML file, see **YAML format**.

NOTE

CCI supports both JSON and YAML, and the file size cannot exceed 1 MiB.

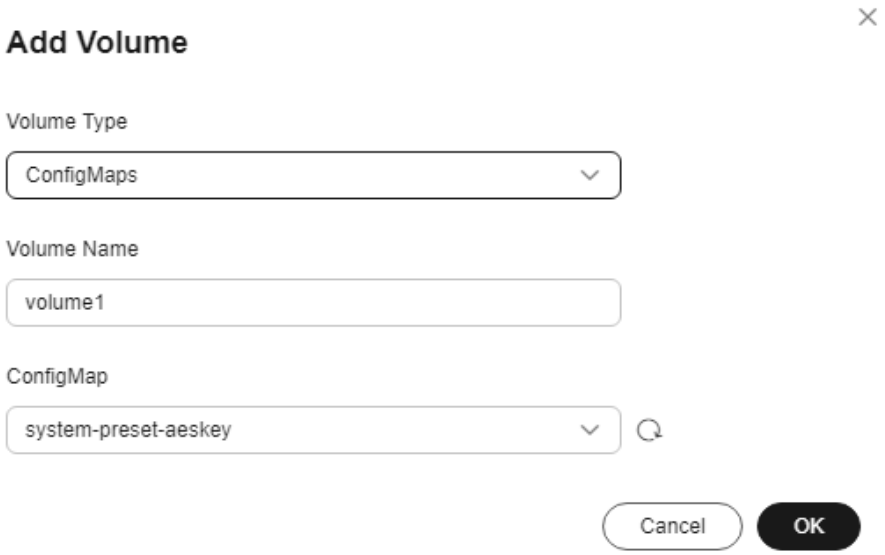
Step 5 Click **OK**.

----End

Using a ConfigMap

After a ConfigMap is created, you can mount it to a container as a storage volume during pod creation. For example, mount a ConfigMap named **system-preset-aeskey** to a container and set the storage volume name to **volume1**.

Figure 4-7 ConfigMaps



ConfigMap File Format

A ConfigMap resource file must be in either JSON or YAML format, and the file size cannot exceed 1 MiB.

- YAML format

  Example file: **configmap.yaml**

```
apiVersion: cci/v2
kind: ConfigMap
metadata:
  name: configmap-example
data:
  key1: value1
  key2: value2
```

- JSON format

  Example file: **configmap.json**

```
{
    "apiVersion": "cci/v2",
    "kind": "ConfigMap",
    "metadata": {
        "name": "configmap-example"
    },
    "data": {
        "key1": "value1",
        "key2": "value2"
    }
}
```


4.4.2 Secrets

Secrets are objects that you can use to store sensitive data such as authentication information, certificates, and private keys. You can load a secret to a container as an environment variable when the container is started or mount a secret to a container as a file.

NOTE

It is recommended that you encrypt the uploaded secrets.

Creating a Secret

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Configuration Center**.

Step 3 Select a namespace and click the **Secrets** tab.

Step 4 Click **Create from YAML** in the upper left corner and edit the YAML file. For details about the YAML file, see **YAML format**.

NOTE

CCI supports both JSON and YAML, and the file size cannot exceed 1 MiB.

Step 5 Click **OK**.

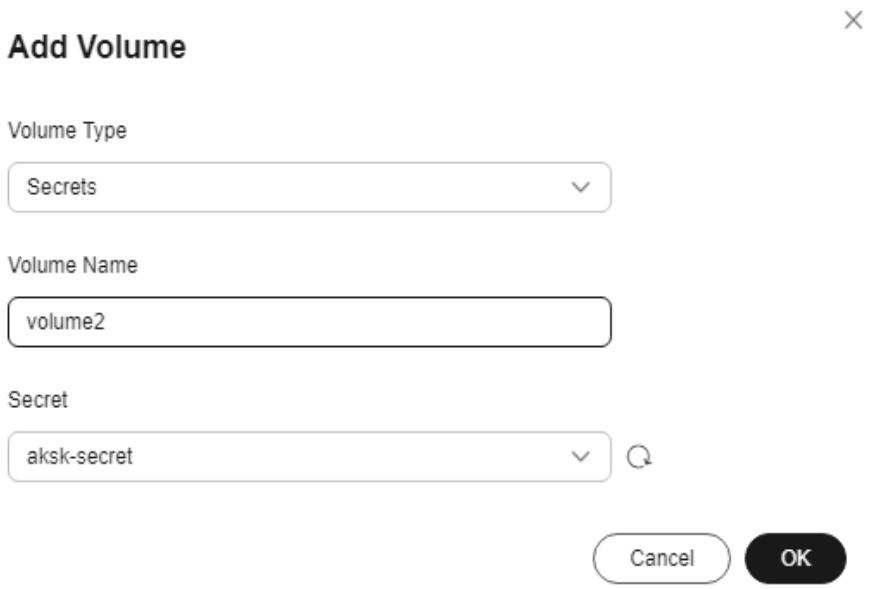
You can view the newly created secret in the secret list.

----End

Using a Secret

After a secret is created, you can mount it to a container as a storage volume during pod creation. For example, mount a secret named **aksk-secret** to a container and set the storage volume name to **volume2**.

Figure 4-8 Mounting a secret



Secret File Format

- **secret.yaml** resource description file

  For example, you can use a secret to obtain the following key-value pairs and encrypt them for an application:

  key1: value1

  key2: value2

  The **secret.yaml** file is defined as below. (Base64 encoding is required for the value of each key. For details about the Base64 encoding method, see **Base64 Encoding**.)

```
apiVersion: cci/v2
kind: Secret
metadata:
  name: mysecret          #Secret name
data:
  key1: dmFsdWUx    #Base64 encoding required
  key2: dmFsdWUy  #Base64 encoding required
type: Opaque         #The type must be Opaque.
```

- **secret.json** resource description file

  The content is as follows:

```
{
    "apiVersion": "cci/v2",
    "kind": "Secret",
    "metadata": {
        "name": "mysecret"
    },
    "data": {
        "key1": "dmFsdWUx",
        "key2": "dmFsdWUy"
    },
    "type": "Opaque"
}
```

Base64 Encoding

To perform Base64 encoding on a character string, run the **echo -n** *{Content to be encoded}* **| base64** command.

```
root@ubuntu:~# echo -n "3306" | base64
MzMwNg==
```

4.5 Images

4.5.1 Image Snapshots

4.5.1.1 Overview

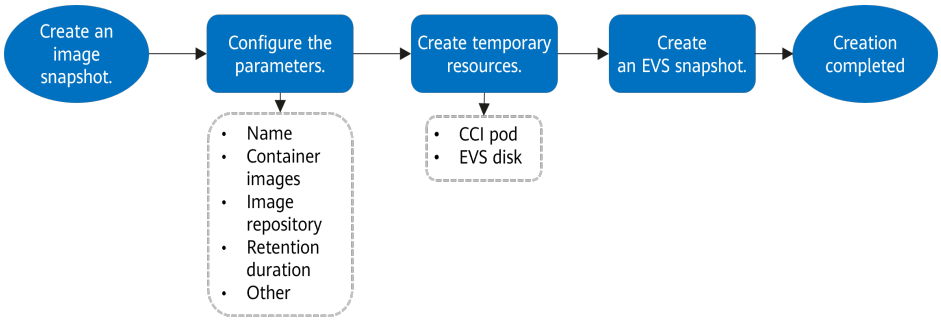
You can pull images from SWR, open-source image repositories, and self-managed image repositories to create image snapshots and then use them to create workloads without pulling the images, speeding up workload startup.

You can use an image snapshot to create a pod. There is no need to pull the image, and therefore pod startup time is reduced. This section describes the constraints on image snapshots and how you can use them.

Constraints

- An image snapshot can contain a maximum of 10 images.
- Private image repositories are supported. However, the access credentials of private image repositories must be provided, including the address and authentication information, such as the auth information in **~/.docker/config.json**.
- If the image needs to be pulled over the public network, you need to specify the public network access configuration in advance.
- If only some images in an image snapshot are consistent with those for the pod, inconsistent images still need to be pulled.
- If the image of a running pod changes and the new image is inconsistent with any image in the image snapshot, the new image needs to be pulled.
- By default, a CCI pod with 0.5 vCPUs and 1 GiB of memory is used to create an image snapshot. There are expenditures during the creation.

The following figure illustrates how to create an image snapshot.



How to Use

You can use an image snapshot to create a pod in either of the following ways:

- Automatic matching

The optimal image is selected from all available image snapshots you have created. The matching is performed in the following sequence:

a. Image matching degree: The images in the image snapshots are matched with those for the pod, and the mostly matched image snapshot is preferred.

b. Time when image snapshots were created: Image snapshots that were created later are preferred.

- Specific image snapshot

Specify the image snapshot to be used.

4.5.1.2 Creating an Image Snapshot

Scenario

This section describes how you can create an image snapshot. For details about how image snapshots work, see **Overview**.

Procedure

To run a container, you need to pull the specified container image first. However, due to factors such as the network and container image size, the pod startup speed slows down by image pull. You can create a snapshot using the image to be used. Then, you can use the snapshot to create pods without pulling the images, speeding up pod startup.

The following example creates an image snapshot named **my-imagesnapshot**.

```
apiVersion: cci/v2
kind: ImageSnapshot
metadata:
  name: 'my-imagesnapshot'
spec:
  buildingConfig:
    namespace: test-namespace
    eipID: xxxxxxxx
  imageSnapshotSize: 30
  ttlDaysAfterCreated: 7
  images:
    - image: 'nginx:stable-alpine-perl'
  registries:
    - imagePullSecret: imagepull-secret
      server: xxxxx.myhuaweicloud.com
      plainHTTP: true
```

Table 4-13 Parameter description

Field	Type	Mandatory	Example Value	Description
.metadata.name	String	Yes	my-imagesnapshot	Image snapshot name.
.spec.images.image	String	Yes	nginx:latest	Image used to create an image snapshot.
.spec.registries.server	String	Yes	serverA.com	Image repository address without the http:// or https:// prefix.
.spec.registries.imagePullSecret	String	No	imagepull-secret	Secret for accessing the image repository.
.spec.registries.plainHTTP	boolean	No	true	If the self-managed image repository address uses the HTTP protocol, set the value to **true**, or the image fails to be pulled due to different protocols. The default value is **false**.

Field	Type	Mandatory	Example Value	Description
.spec.registries.insecureSkipVerify	boolean	No	true	If the self-managed image repository address uses a self-issued certificate, set the value to **true** to skip certificate authentication, or the image fails to be pulled due to certificate authentication failure. The default value is **false**.
.spec.buildingConfig.namespace	String	Yes	my-namespace-a	User namespace. During image snapshot creation, you need to create a pod in the user namespace.
.spec.buildingConfig.eipID	String	No	3cxxxxe0-xxxx-xxxx-xxxx-8xxxxf3xxxx4	EIP used for pulling images from the public network.
.spec.buildingConfig.autoCreateEIP	boolean	No	true	Whether to automatically assign an EIP for the pod that will create the image snapshot. If **eipId** is specified, this parameter is ignored. If this parameter is set to **true**, you need to specify the EIP configuration using **autoCreateEIPAttribute**.
.spec.buildingConfig.autoCreateEIPAttribute.bandwidthChargeMode	String	No	bandwidth	Whether the billing is based on traffic or bandwidth. The value can be **traffic** or **bandwidth**. If this parameter is left blank or is an empty string, default value **bandwidth** is used. For IPv6 addresses, the default parameter value is **bandwidth** outside China and is **traffic** in China.
.spec.buildingConfig.autoCreateEIPAttribute.bandwidthSize	int	No	1000	Bandwidth size. The value ranges from 1 Mbit/s to 2,000 Mbit/s by default.

Field	Type	Mandatory	Example Value	Description
.spec.buildingConfig.autoCreateEIPAttribute.type	String	No	5_bgp	EIP type. The value can be **5_bgp** (dynamic BGP), **5_sbgp** (static BGP), or **5_youxuanbgp** (premium BGP).
.spec.buildingConfig.autoCreateEIPAttribute.ipVersion	int	No	4	EIP version. The value can be **4** or **6**. • **4** indicates IPv4. If this parameter is left empty or is an empty string, an IPv4 address is assigned by default. • **6** indicates IPv6. If the parameter is set to **6**, NAT64 is enabled.
.spec.buildingConfig.TimeoutMinutes	int	No	1440	Timeout interval for creating a snapshot, in minutes. The value is an integer from 30 to 10,080 (which is one week). The default value is **1440**, which is one day.
.spec.ttlDaysAfterCreated	integer	No	10	Retention period of the image snapshot, in days. Expired image snapshots will be deleted. The default value is **0**, indicating that the image snapshot never expires. When an image snapshot is used by a workload or pod, its expiration time is reset to the time when the image snapshot is used plus the image snapshot retention period. **NOTE** After an image snapshot expires, it still occupies the quota. You need to periodically review and delete expired image snapshots.

Field	Type	Mandatory	Example Value	Description
.spec.imageSnapshotSize	integer	No	20	Image snapshot size, in GiB. The default value is **20**.

4.5.1.3 Using an Image Snapshot

You can use an image snapshot to create a pod in either of the following ways:

- Automatic matching

  The optimal image is selected from all available image snapshots based on the following:

  a. Image matching degree: The images in the image snapshots are matched with those for the pod, and the mostly matched image snapshot is preferred.

  b. Time when image snapshots were created: Image snapshots that were created later are preferred.

- Specific image snapshot

  Specify the image snapshot to be used.

NOTE

If you create a pod using both the specified and automatically matched image snapshots and the two methods conflict, 400 will be returned.

Automatic Matching

When creating a pod, you can add the following annotation to enable automatic image snapshot matching.

Key	Example Value	Description
cci.io/image-snapshot-auto-match	"true"	Whether to enable automatic image snapshot matching.

Key	Example Value	Description
cci.io/image-snapshot-usage-strategy	"size"	Automatic matching policy for image snapshots. The default value is **size**. • **size**: selects the image snapshot with the largest sum of image sizes. • **quantity**: selects the image snapshot that matches the largest number of images.

In the following example, a Deployment will be created.

```
apiVersion: cci/v2
kind: Deployment
metadata:
  name: deployment-test
  namespace: ns-test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
      annotations:
        cci.io/image-snapshot-auto-match: "true"
    spec:
      containers:
        - image: redis
          name: container-0
          resources:
            limits:
              cpu: 500m
              memory: 1024Mi
            requests:
              cpu: 500m
              memory: 1024Mi
      imagePullSecrets:
        - name: imagepull-secret
```

Specifying an Image Snapshot

When creating a pod, you can add the following annotations to specify the image snapshot and interception policy.

Key	Example Value	Description
cci.io/image-snapshot-specified-name	"my-imagesnapshot"	Name of the specified image snapshot.

Key	Example Value	Description
cci.io/image-snapshot-reject-if-not-available	"true"	If the specified image snapshot does not exist or is unavailable, the image is pulled from the image repository by default. To intercept pod creation, set the value to **"true"**.

In the following example, a Deployment will be created.

```
apiVersion: cci/v2
kind: Deployment
metadata:
  name: deployment-test
  namespace: ns-test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
      annotations:
        cci.io/image-snapshot-specified-name: "my-imagesnapshot"
        cci.io/image-snapshot-reject-if-not-available: "true"
    spec:
      containers:
        - image: redis
          name: container-0
          resources:
            limits:
              cpu: 500m
              memory: 1024Mi
            requests:
              cpu: 500m
              memory: 1024Mi
      imagePullSecrets:
        - name: imagepull-secret
```

4.5.1.4 Managing Image Snapshots

Scenario

This section describes how to view and delete an image snapshot on the console. For details about how image snapshots work, see **Overview**.

Viewing an Image Snapshot

After an image snapshot is created, take the following steps to view the details:

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Image Snapshots**.

Step 3 View the name, status, remaining validity period, and size of the image snapshot. You can also view event details.

When the image snapshot status is **Available**, you can use the image snapshot.

If the image snapshot status is **Abnormal**, you can click **Events** to view the event details.

----End

Deleting Image Snapshots

If image snapshots are no longer used, take the following steps to delete them:

Step 1 Log in to the **CCI 2.0 console**.

Step 2 In the navigation pane, choose **Image Snapshots**.

Step 3 Select the image snapshots to be deleted and click **Delete** above the image snapshot list.

To delete an image snapshot, click **Delete** in the **Operation** column.

----End

4.5.2 Pulling an Image from a Self-Managed Image Repository

When an image is pulled from a self-managed image repository, the image may fail to be pulled due to different protocols or certificate authentication failures. In this section, HTTP and a self-issued certificate are used as examples to describe how to create a Deployment or pod by pulling an image from a self-managed image repository.

Configuration Description

Table 4-14 Configuration description

Annotation	Example Value	Configuration Description
cci.io/http-registries	"harbor.***.com,192.168.XX.XX:5000,100.95.XX.XX,http://harbor.***.com"	If you want to pull an image from a self-managed image repository using HTTP, you need to configure this annotation. The value can contain the https://prefix, port number, and relative path. Use commas (,) to separate multiple addresses, which can be private IP addresses, domain names, or public IP addresses. A maximum of 10 addresses are allowed.
cci.io/insecure-registries	"harbor.***.com,192.168.XX.XX:5000,100.95.XX.XX,https://harbor.***.com"	If you want to pull an image from a self-managed image repository using a self-issued certificate, you need to add this annotation to skip certificate authentication. The value can contain the https:// prefix, port number, and relative path. Use commas (,) to separate multiple addresses, which can be private IP addresses, domain names, or public IP addresses. A maximum of 10 addresses are allowed.

NOTE

- If the image repository address has a port number, the port number must be included. For example, if the image path is *192.168.XX.XX:5000/nginx:latest*, **cci.io/http-registries** can be set to *192.168.XX.XX:5000*.
- If HTTP is used, data transmission is not encrypted, and data is vulnerable to man-in-the-middle attacks and lacks identity authentication, which may cause data leak and service loss. HTTPS is recommended.

Example 1: Using HTTP for a Self-Managed Image Repository

- Creating a Deployment with 2 vCPUs and 4-GiB memory

```
apiVersion: cci/v2
kind: Deployment
metadata:
  labels:
    app: http
  name: http
spec:
  replicas: 1
  selector:
    matchLabels:
      app: http
  template:
    metadata:
      labels:
        app: http
      annotations:
        resource.cci.io/pod-size-specs: 2.00_4.0
        cci.io/http-registries: 192.168.XX.XX
    spec:
      containers:
        - image: 192.168.XX.XX/harbor/nginx:latest
          name: container-0
      imagePullSecrets:
        - name: harbor-secret-new
```

- Creating a pod with 2 vCPUs and 4-GiB memory

```
apiVersion: cci/v2
kind: Pod
metadata:
  annotations:
    resource.cci.io/pod-size-specs: 2.00_4.0
    cci.io/http-registries: 192.168.XX.XX
  name: http
spec:
  containers:
    - image: '192.168.XX.XX/harbor/nginx:latest'
      imagePullPolicy: IfNotPresent
      name: container-1
  imagePullSecrets:
    - name: harbor-secret
```

Example 2: Using a Self-Issued Certificate for a Self-Managed Image Repository

- Creating a Deployment with 2 vCPUs and 4-GiB memory

```
apiVersion: cci/v2
kind: Deployment
metadata:
  labels:
    app: insecure
  name: insecure
spec:
  replicas: 1
  selector:
```



```
  matchLabels:
    app: insecure
template:
  metadata:
    labels:
      app: insecure
    annotations:
      resource.cci.io/pod-size-specs: 2.00_4.0
      cci.io/insecure-registries: 192.168.XX.XX
  spec:
    containers:
      - image: 192.168.XX.XX/harbor/nginx:latest
        name: container-0
    imagePullSecrets:
      - name: harbor-secret-new
```

- Creating a pod with 2 vCPUs and 4 GiB-memory

```
apiVersion: cci/v2
kind: Pod
metadata:
  annotations:
    resource.cci.io/pod-size-specs: 2.00_4.0
    cci.io/insecure-registries: 192.168.XX.XX
  name: insecure
spec:
  containers:
    - image: '192.168.XX.XX/harbor/nginx:latest'
      imagePullPolicy: IfNotPresent
      name: container-1
  imagePullSecrets:
    - name: harbor-secret
```

4.6 Monitoring

A Prometheus instance for cloud services can be used to monitor multiple metrics of CCI. For details, see **Prometheus Monitoring Overview**.

Constraints

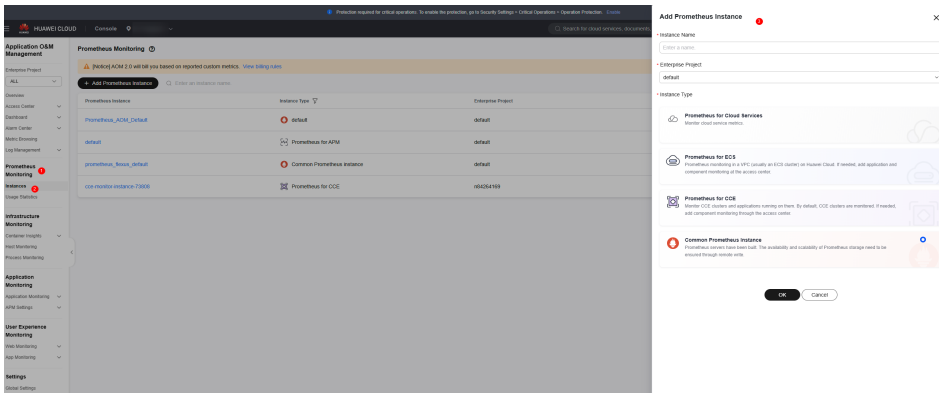
Only one Prometheus instance for cloud services can be created in an enterprise project.

Step 1: Create a Prometheus Instance

1. Log in to the AOM 2.0 console.
2. In the navigation pane on the left, choose **Metric Browsing** > **Prometheus Monitoring**. On the displayed page, click **Add Prometheus Instance**.
3. Set the instance name, enterprise project, and instance type.

   NOTE

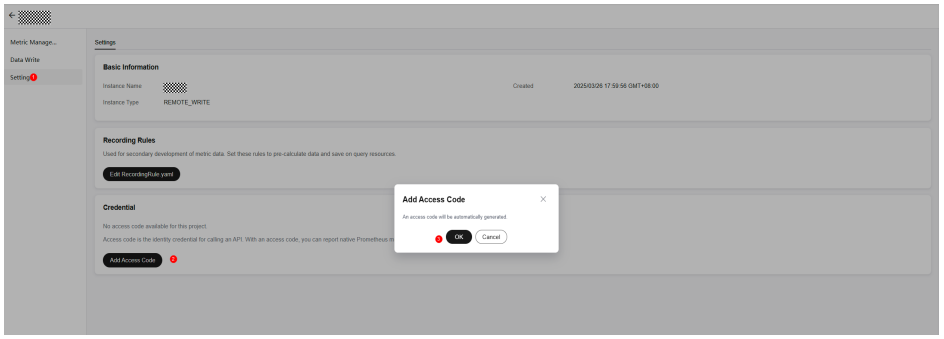
   If the CCE Cloud Bursting Engine for CCI add-on is being used or required, the CCE cluster also needs this Prometheus instance, with the instance type set to **Prometheus for CCE**.



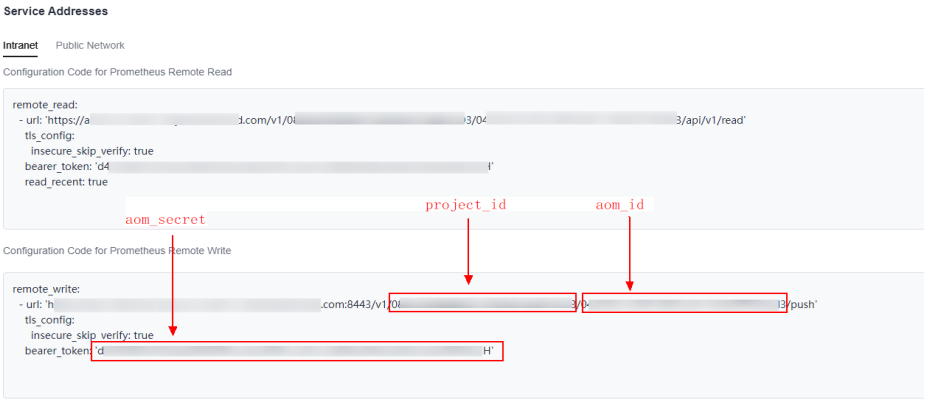
4. Click **OK**.

Step 2: Obtain the Access Code

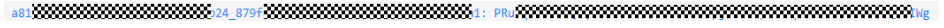
1. Click the name of the created Prometheus instance.
2. In the navigation pane, choose **Settings**. In the **Credential** area, click **Add Access Code**.
3. Click **OK**.



4. Obtain the values of *<project_id>*, *<aom_id>*, and *<aom_secret>*.



5. Combine the obtained values into a character string in the format of *<project_id>_<aom_id>:<aom_secret>*.



CAUTION

There must be a space after the colon (:).

6. Encode the character string using Base64.

   YTgxZ[redacted]KU3SVdn

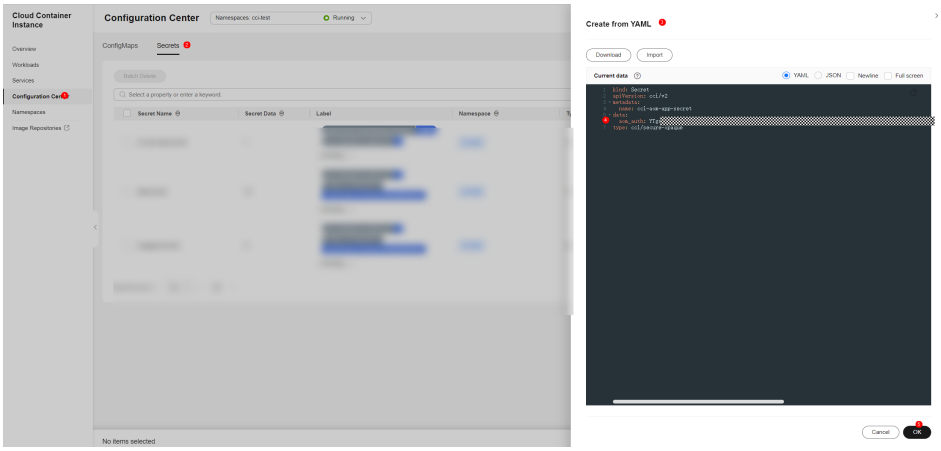
   You can run the following **shell** commands for encoding:

   ```
   project_id=<project_id>
   aom_id=<aom_id>
   aom_secret=<aom_secret>
   echo -n "${project_id}_${aom_id}: ${aom_secret}" |base64 -w 0
   ```

7. Fill the Base64-encoded character string in the following template to replace **{AOMAuthBase64}**, and then copy the code.

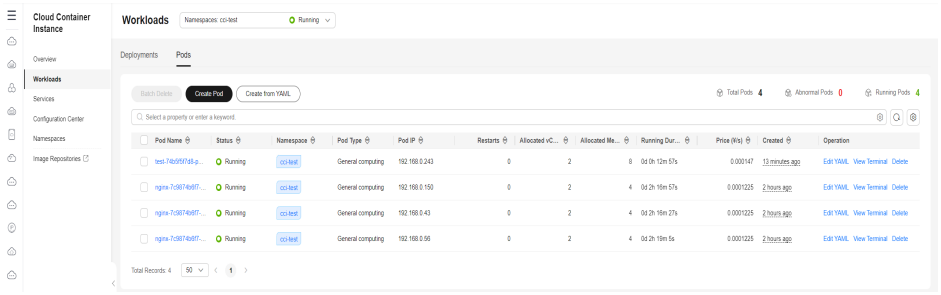
   ```
   kind: Secret
   apiVersion: cci/v2
   metadata:
     name: cci-aom-app-secret
   data:
     aom_auth: {AOMAuthBase64}
   type: cci/secure-opaque
   ```

8. Log in to the **CCI 2.0 console**.
9. In the navigation pane, choose **Configuration Center**.
10. Click the **Secrets** tab.
11. Click **Create from YAML** and use the copied code to replace the code on the CCI console to create an AOM app secret.



Step 3 Create a Pod

1. Log in to the **CCI 2.0 console**.
2. In the navigation pane, choose **Workloads**. On the displayed page, click the **Pods** tab.
3. Click **Create Pod** and configure the parameters. For details about the parameters, see **Creating a Pod**.



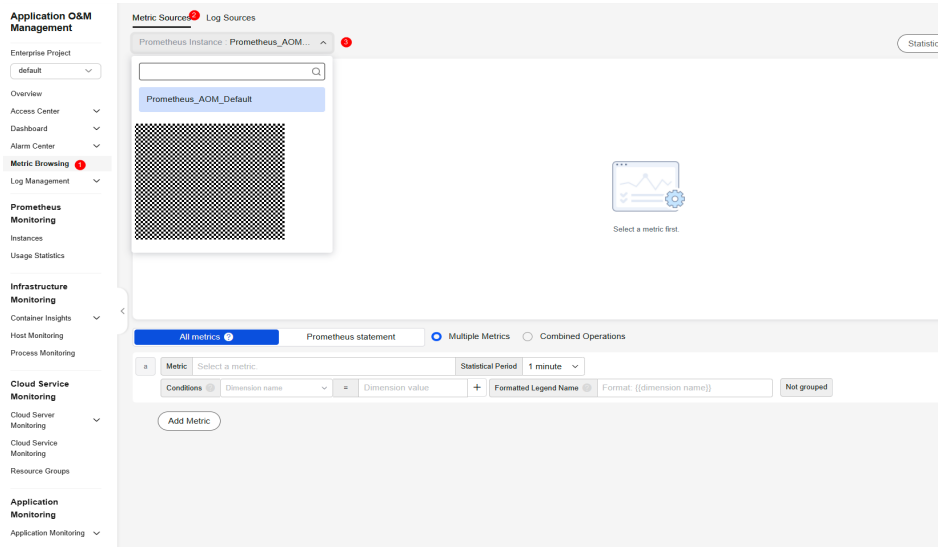
4. Click **Create Now**.

Step 4 View Monitoring Data

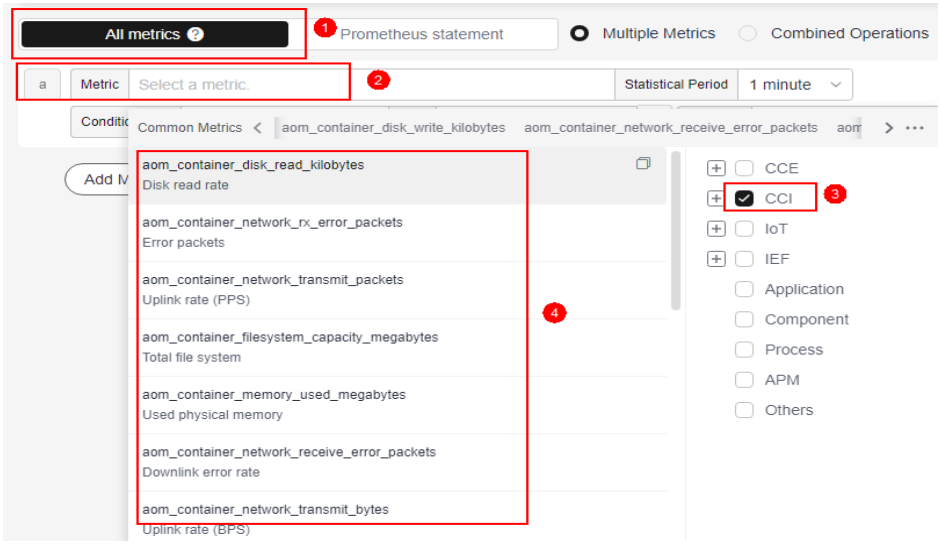
There are two ways to view monitoring data of a pod: All metrics and Prometheus statement.

Method 1: All Metrics

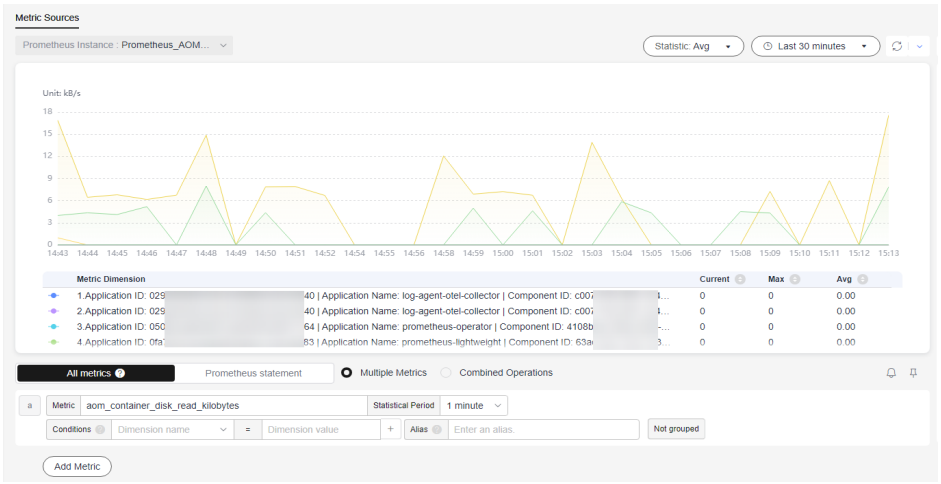
1. Log in to the AOM 2.0 console.
2. In the navigation pane, choose **Metric Analysis** > **Metric Browsing**.
3. On the **Metric Sources** tab, select the Prometheus instance created in **Step 1: Create a Prometheus Instance**.



4. On the **All metrics** tab, click the text box of **Metric**.
5. Select **CCI** on the right and select related common metrics.

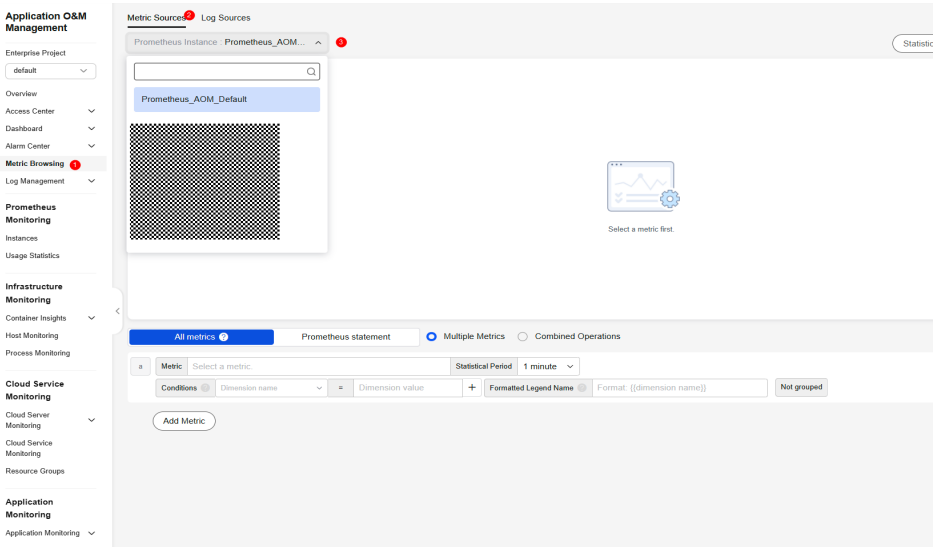


6. View metrics.



Method 2: Prometheus Statement

1. Log in to the AOM 2.0 console.
2. In the navigation pane, choose **Metric Analysis** > **Metric Browsing**.
3. On the **Metric Sources** tab, select the Prometheus instance created in **Step 1: Create a Prometheus Instance**.

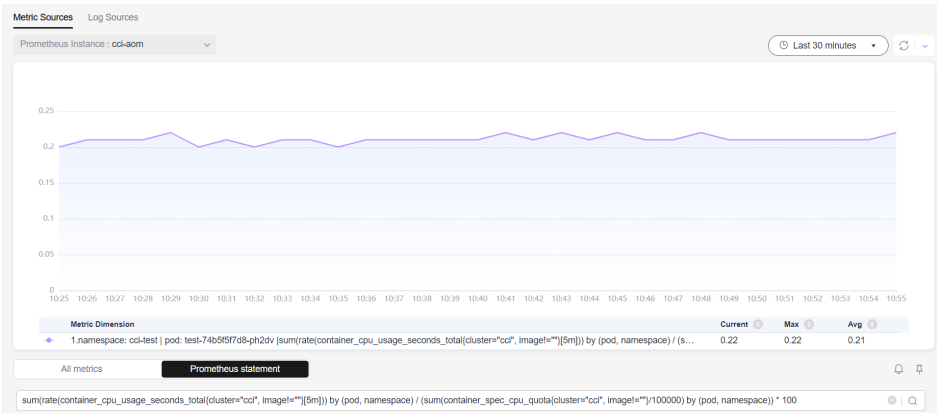


4. On the **Prometheus statement** tab, enter a Prometheus statement in the search box and click .

The following are examples of common Prometheus statements.

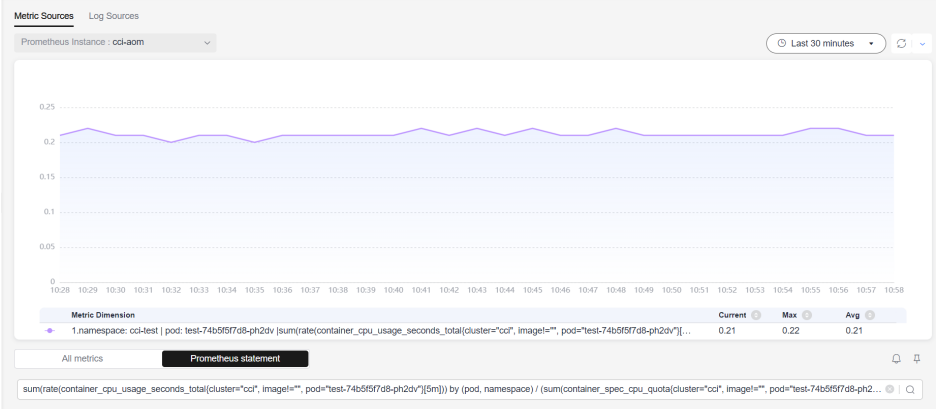
a. Prometheus statement for querying the vCPU usages of all pods:

```
sum(rate(container_cpu_usage_seconds_total{cluster="cci", image!=""}[5m])) by (pod, namespace) / (sum(container_spec_cpu_quota{cluster="cci", image!=""}/100000) by (pod, namespace)) * 100
```



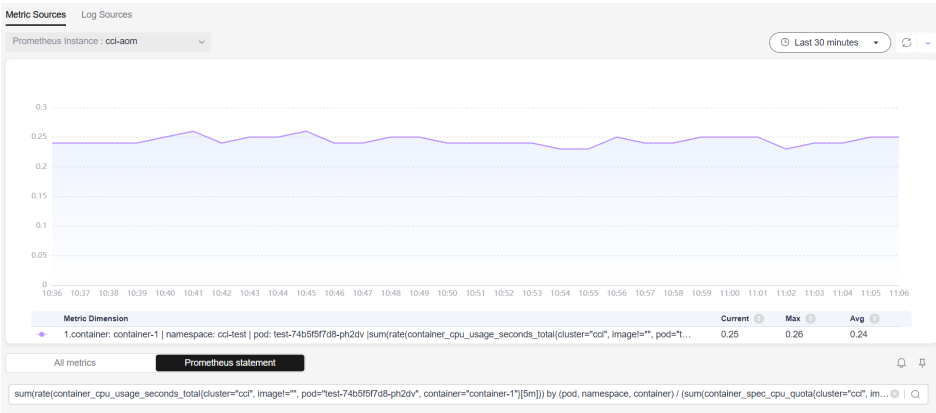
b. Prometheus statement for querying the vCPU usage of a pod (*<pod_name>* indicates the pod name):

```
sum(rate(container_cpu_usage_seconds_total{cluster="cci", image!="", pod="<pod_name>"}[5m])) by (pod, namespace) / (sum(container_spec_cpu_quota{cluster="cci", image!="", pod="<pod_name>"}/100000) by (pod, namespace)) * 100
```

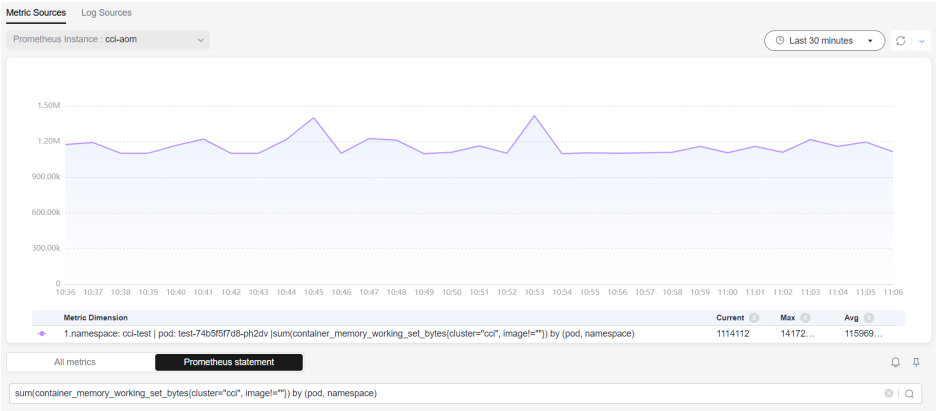
c. Prometheus statement for querying the vCPU usage of a container (*<pod_name>* indicates the pod name, and <container_name> indicates the container name):

```
sum(rate(container_cpu_usage_seconds_total{cluster="cci", image!="", pod="<pod_name>", container="<container_name>"}[5m])) by (pod, namespace, container) / (sum(container_spec_cpu_quota{cluster="cci", image!="", pod="<pod_name>", container="<container_name>"}/100000) by (pod, namespace, container)) * 100
```



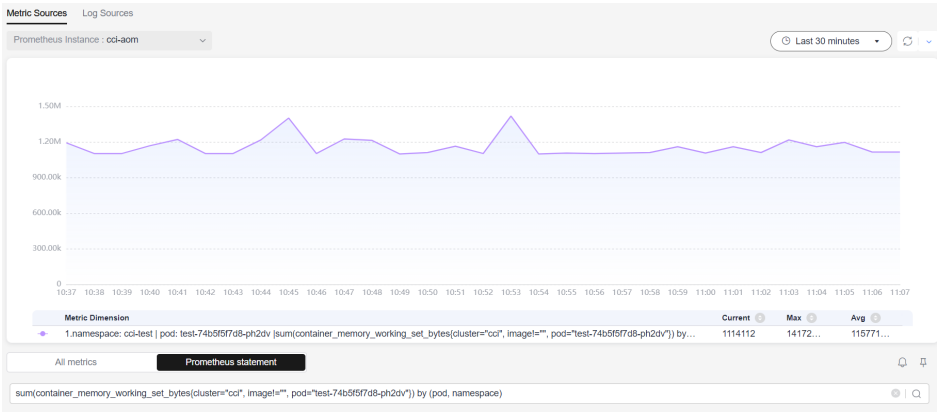
d. Prometheus statement for querying the used memory of all pods:

```
sum(container_memory_working_set_bytes{cluster="cci", image!=""}) by (pod, namespace)
```



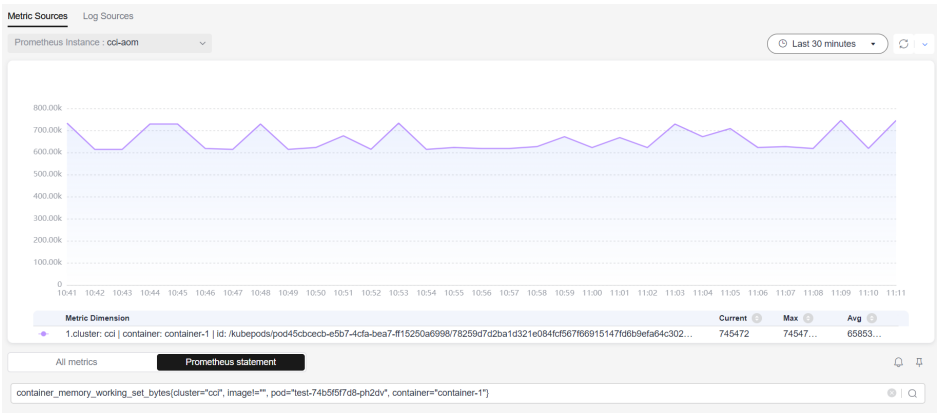
e. Prometheus statement for querying the used memory of a pod (*<pod_name>* indicates the pod name):

```
sum(container_memory_working_set_bytes{cluster="cci", image!="", pod="<pod_name>"}) by (pod, namespace)
```

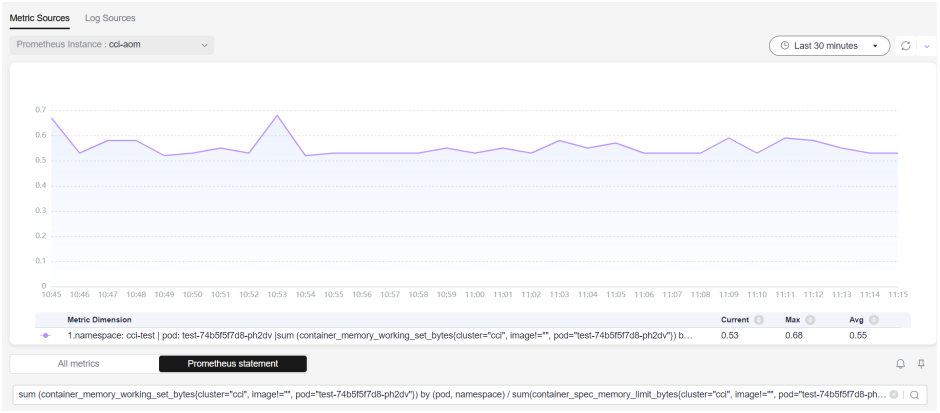
f. Prometheus statement for querying the used memory of a container (*<pod_name>* indicates the pod name, and *<container_name>* indicates the container name):

```
container_memory_working_set_bytes{cluster="cci", image!="", pod="<pod_name>", container="<container_name>"}
```



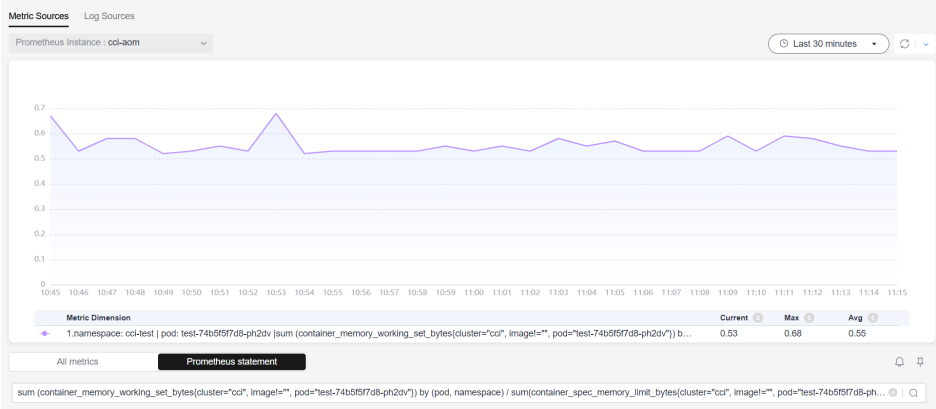
g. Prometheus statement for querying the memory usage of all pods:

```
sum (container_memory_working_set_bytes{cluster="cci", image!=""}) by (pod, namespace) / sum(container_spec_memory_limit_bytes{cluster="cci", image!=""}) by (pod, namespace) * 100 != +Inf
```



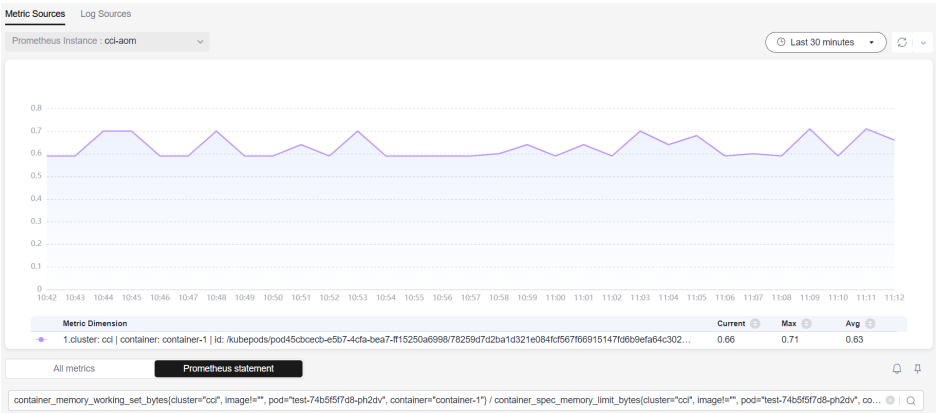
h. Prometheus statement for querying the memory usage of a pod (*<pod_name>* indicates the pod name):

```
sum (container_memory_working_set_bytes{cluster="cci", image!="", pod="<pod_name>"}) by (pod, namespace) / sum(container_spec_memory_limit_bytes{cluster="cci", image!="", pod="<pod_name>"}) by (pod, namespace) * 100 != +Inf
```

i. Prometheus statement for querying the memory usage of a container (*<pod_name>* indicates the pod name, and *<container_name>* indicates the container name):

```
container_memory_working_set_bytes{cluster="cci", image!="", pod="<pod_name>", container="<container_name>"} / container_spec_memory_limit_bytes{cluster="cci", image!="", pod="<pod_name>", container="<container_name>"} * 100 !=+Inf
```



For more monitoring and O&M methods, see **Creating a Dashboard**.

4.7 Log Management

4.7.1 Log Collection

CCI works with LTS to collect application logs and report these logs to LTS so that you can use them for troubleshooting.

Log Collection Reliability

The log system's main purpose is to record all stages of data for service components, including startup, initialization, exit, runtime details, and exceptions. It is primarily employed in O&M scenarios for tasks like checking component status and analyzing fault causes.

Standard streams (stdout and stderr) and local log files use non-persistent storage. However, data integrity may be compromised due to the following risks:

- Log rotation and compression potentially deleting old files

- Temporary storage volumes being cleared when Kubernetes pods end
- Automatic OS cleanup triggered by limited node storage space

While the Cloud Native Log Collection add-on employs techniques like multi-level buffering, priority queues, and resumable uploads to enhance log collection reliability, logs could still be lost in the following situations:

- The service log throughput surpasses the collector's processing capacity.
- The service pod is abruptly terminated and reclaimed by CCE.
- The log collector pod experiences exceptions.

The following lists some recommended best practices for cloud native log management. You can review and implement them thoughtfully.

- Use dedicated, high-reliable streams to record critical service data (for example, financial transactions) and store the data in persistent storage.
- Avoid storing sensitive information like customer details, payment credentials, and session tokens in logs.

Constraints

- Logs cannot be collected from the directory that a specified system, device, cgroup, or tmpfs is mounted to.
- A single-line log that exceeds 250 KB will not be collected.
- Regular expression match is only supported when a full path with a complete file name is specified.
- The name of each file to be logged must be unique in a container. If there are files with duplicate names, only the logs of one file are collected.
- After a pod is started, the log collection configuration cannot be updated. If the configuration is updated, the pod must be restarted for the configuration to take effect.
- A directory to be logged must exist before the container is started. If a directory is created after the container is started, the logs of the directory and of the files in that directory cannot be collected.
- If the name of a file exceeds 190 characters, its logs cannot be collected.
- Logs of init containers cannot be collected.
- Logs of Kunpeng pods cannot be collected.

Step 1 Create a Log Group

Step 1 Log in to the management console and choose **Management & Deployment** > **Log Tank Service**.

Step 2 On the **Log Management** page, click **Create Log Group**.

Step 3 On the displayed page, set log group parameters by referring to **Table 4-15**.

Table 4-15 Log group parameters

Parameter	Description
Log Group Name	A log group is the basic unit for LTS to manage logs. It is used to classify log streams. If there are too many logs to collect, separate logs into different log groups based on log types, and name log groups in an easily identifiable way. LTS automatically generates a default log group name. You are advised to customize one based on your service. You can also change it after the log group is created. The naming rules are as follows: • Enter 1 to 64 characters, including only letters, digits, hyphens (-), underscores (_), and periods (.). Do not start with a period or underscore or end with a period. • Each log group name must be unique.
Enterprise Project Name	Enterprise projects allow you to manage cloud resources and users by project. By default, the **default** enterprise project is used. You are advised to select an enterprise project that fits your service needs. To see all available options, click **View Enterprise Projects**.
Log Retention (Days)	Specify the log retention period for the log group, that is, how many days the logs will be stored in LTS after being reported to LTS. By default, logs are retained for 30 days. You can set the retention period to one to 365 days.

Parameter	Description
Tag	Tag the log group as required. Click **Add Tags** and enter a tag key and value. If you enable **Apply to Log Stream**, the tag will be applied to all log streams in the log group. To add more tags, repeat this step. A maximum of 20 tags can be added. **Tag key restrictions:** ● A tag key can contain letters, digits, spaces, and special characters (_.:=+-@), but cannot start or end with a space or start with _**sys**_. ● A tag key can contain up to 128 characters. ● Each tag key must be unique. **Tag value restrictions:** ● A tag value can contain letters, digits, spaces, and the following special characters: _.:=+-@ ● A tag value can contain up to 255 characters. **Deleting a tag:** Click **Delete** in the **Operation** column of the tag. **WARNING** Deleted tags cannot be recovered. If a tag is used by a transfer task, you need to modify the task configuration after deleting the tag.
Remark	Enter remarks. The value contains up to 1,024 characters.

Step 4 Click **OK**. The created log group will be displayed in the log group list.

- In the log group list, view information such as the log group name, tags, and log streams.
- Click the log group name to access the log details page.

----End

Step 2 Create a Log Stream

1. On the LTS console, click ⌄ on the left of a log group name.
2. In the upper left corner of the displayed page, click **Create Log Stream** and then enter a log stream name. The log stream name:
   - Can contain only letters, numbers, underscores (_), hyphens (-), and periods (.). The name cannot start with a period or underscore, or end with a period.
   - Can contain 1 to 64 characters.

   NOTE

   Collected logs are sent to the created log stream. If there are a large number of logs, you can create multiple log streams and name them for quick log search.
3. Select an enterprise project. You can click **View Enterprise Projects** to view all enterprise projects.

4. If you enable **Log Retention Duration** on this page, you can set the log retention duration specifically for the log stream. If you disable it, the log stream will inherit the log retention setting of the log group.
5. Anonymous write is disabled by default and is suitable for log reporting on Android, iOS, applets, and browsers. If anonymous write is enabled, the anonymous write permission is granted to the log stream, and no valid authentication is performed, which may generate dirty data.
6. Set the tag in the *Key=Value* format, for example, a=b.
7. Enter remarks. A maximum of 1,024 characters are allowed.
8. Click **OK**. In the log stream list, you can view information such as the log stream name and operations.

   NOTE

   - You can view the log stream billing information. For details, see **Price Calculator**.

Step 3 Obtain the Log Group ID and Log Stream ID

1. In the navigation pane, choose **Log Management**.
2. Select the log group created in **Step 1 Create a Log Group** and click **More** > **Details** in the **Operation** column.
3. Copy the log group ID.
4. Click on the left of the log group name.
5. Select the log stream created in **Step 2 Create a Log Stream** and click **Details** in the **Operation** column.
6. Copy the log stream ID.

Step 4 Create a Deployment on the CCI Console

1. Log in to the **CCI 2.0 console**.
1. In the navigation pane, choose **Workloads**. On the displayed page, click the **Deployments** tab.
2. Click **Create from YAML** to create a Deployment using **YAML** or **JSON**.
   - Method 1: Use YAML to create a Deployment.

```
apiVersion: cci/v2
kind: Deployment
metadata:
  annotations:
    description: ''
  labels: {}
  name: test
  namespace: default  # Namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      annotations:
        logconf.k8s.io/fluent-bit-log-type: lts      # (Mandatory) LTS is used to collect logs.
        logconfigs.logging.openvessel.io: |
          {
            "default-config": {   # You can set the log collection paths of multiple containers. stdout.log
```

indicates standard output. ***/root/out.log*** *contains the text logs in rootfs (volumes included).* ***/data/emptydir-xxx/*.log*** *indicates the directories in rootfs (volumes included).*

```
            "container_files": {
              "container-0": "stdout.log;/root/out.log;/data/emptydir-volume/*.log",
              "container-1": "/root/standard.log"
            },
            "regulation": "", #Regular expression for matching multi-line logs. For details, see Regular Expression Matching Rules.
            "lts-log-info": {    # Configure a log group and a log stream.
              "<log-group-ID>": "<log-stream-ID>"  #Replace <log-group-ID> and <log-stream-ID> with those obtained in Step 3.
            }
          },
          "multi-config": {
            "container_files": {
              "container-0": "/root/multi.log",
              "container-1": "stdout.log;/root/out.log;/data/emptydir-memory-volume/*.log"
            },
            "regulation": "/(?<log>\\d+-\\d+-\\d+ \\d+:\\d+:\\d+.*)/",
            "lts-log-info": {    # Configure the same log group and log stream.
              "<log-group-ID>": "<log-stream-ID>"  #Replace <log-group-ID> and <log-stream-ID> with those obtained in Step 3.
            }
          }
        }
      resource.cci.io/pod-size-specs: 2.00_8.0
      resource.cci.io/size: 2.00_8.0
      vm.cci.io/pod-size-specs: 2.00_8.0
    labels:
      app: test
      sys_enterprise_project_id: "0"
  spec:
    containers:
    - image: swr.***.com/paas_cci/vk-webhook:8.16.0
      imagePullPolicy: IfNotPresent
      command: ['sh', '-c', "while true; do echo hello; touch /root/out.log; echo hello >> /root/out.log; touch /data/emptydir-volume/emptydir.log; echo hello >> /data/emptydir-volume/emptydir.log; sleep 10; done"]
      lifecycle: {}
      volumeMounts:
      - name: emptydir-volume
        mountPath: /data/emptydir-volume
      - name: emptydir-memory-volume
        mountPath: /data/emptydir-memory-volume
      name: container-0
      resources:
        limits:
          cpu: 100m
          memory: 100Mi
        requests:
          cpu: 100m
          memory: 100Mi
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    - image: swr.***.com/paas_cci/vk-webhook:8.16.0
      imagePullPolicy: IfNotPresent
      command: ['sh', '-c', "while true; do echo hello; touch /root/out.log; echo hello >> /root/out.log; touch /data/emptydir-memory-volume/emptydir-memory.log; echo $(date +'%Y-%m-%d %H:%M:%S.%3N') hello >> /data/emptydir-memory-volume/emptydir-memory.log; echo hello >> /data/emptydir-memory-volume/emptydir-memory.log; sleep 10; done"]
      lifecycle: {}
      volumeMounts:
      - name: emptydir-volume
        mountPath: /data/emptydir-volume
      - name: emptydir-memory-volume
        mountPath: /data/emptydir-memory-volume
      name: container-1
      resources:
        limits:
```



```
          cpu: 100m
          memory: 100Mi
        requests:
          cpu: 100m
          memory: 100Mi
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: Default
    volumes:
    - name: emptydir-volume
      emptyDir: {}
    - name: emptydir-memory-volume
      emptyDir:
        sizeLimit: 1Gi
        medium: Memory
    enableServiceLinks: false
    restartPolicy: Always
    schedulerName: volcano
    securityContext: {}
    terminationGracePeriodSeconds: 30
  minReadySeconds: 0
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
```

Table 4-16 Key parameters

Parameter	Mandatory	Type	Description
logconf.k8s.io/fluent-bit-log-type	Yes	String	• Description: Log collection mode • Constraints: This parameter is mandatory. • Value: **lts**
logconfigs.logging.openvessel.io	Yes	String	• Description: Log collection configuration

– Method 2: Use JSON to create a Deployment.

```
{
   "default-config": {
      "container_files": { // You can set the log collection paths of multiple containers. stdout.log indicates standard output. /root/out.log indicates the text logs in rootfs (volumes included). /data/emptydir-xxx/*.log indicates the directories in rootfs (volumes included).
         "container-0": "stdout.log;/root/out.log;/data/emptydir-volume/*.log",
         "container-1": "stdout.log"
      },
      "regulation": "", // Regular expression matching rule for collecting multi-line logs. For details about regular expression matching rules, see https://docs.fluentbit.io/manual/pipeline/parsers/configuring-parser.
      "lts-log-info": { // Only one log group and one log stream are allowed.
         "log-group-ID": "log-stream-ID"  // Replace log-group-ID and log-stream-ID with those obtained in Step 3.
      }
   },
   "multi-config": {
      "container_files": { // You can set the log collection paths of multiple containers. stdout.log indicates standard output. /root/out.log indicates the text logs in rootfs (volumes included). /data/
```



```
***emptydir-xxx/*.log*** *indicates the directories in rootfs (volumes included).*
            "container-0": "stdout.log;/root/out.log;/data/emptydir-memory-volume/*.log"
        },
        "regulation": "/(?<log>\\d+-\\d+-\\d+ \\d+:\\d+:\\d+.*)/",  // Regular expression matching rule for collecting multi-line logs
        "lts-log-info": { // Only one log group and one log stream are allowed.
            "<log-group-ID>": "<log-stream-ID>"  // Replace <log-group-ID> and <log-stream-ID> with those obtained in Step 3.
        }
    }
}
```

CAUTION

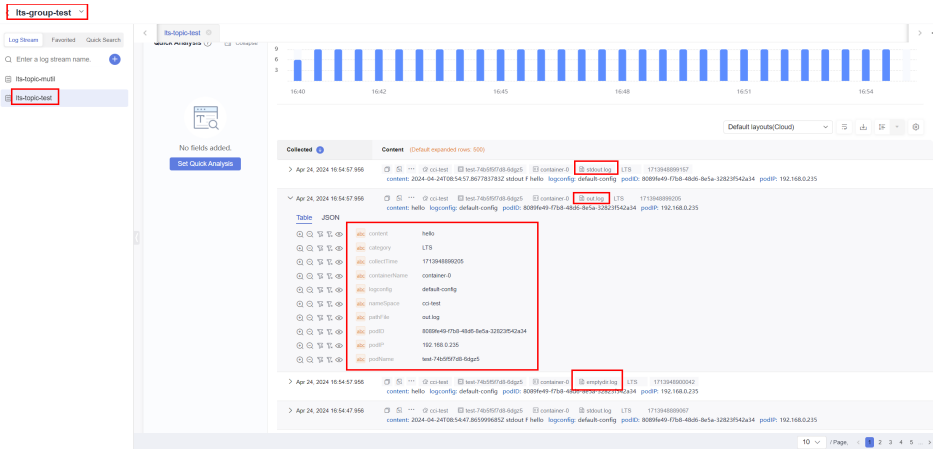
When you are creating a workload using YAML or JSON, you need to add the log group ID and log stream ID obtained in **Step 3 Obtain the Log Group ID and Log Stream ID** to the **lts-log-info** parameter, for example, **\"lts-log-info\":{\"***log-group-ID***\":\"***log-stream-ID***\"}**

- Replace *log-group-ID* with the log group ID obtained in **Step 3 Obtain the Log Group ID and Log Stream ID**.
- Replace *log-stream-ID* with the log stream ID obtained in **Step 3 Obtain the Log Group ID and Log Stream ID**.

3. Click **OK**.
4. Click the created Deployment to view its status.

Step 5 Check Log Reporting

1. Log in to the LTS console. In the navigation pane, choose **Log Management** and click the name of the log group.
2. Go to the log details page to view logs.



4.7.2 Mounting Standard Output Logs

This section describes how to mount container standard output logs in a pod to a specified container for easy log retrieval.

Constraints

volume.cci.io/mount-stdlog-containers and **volume.cci.io/mount-stdlog-containers-path** cannot be configured at the same time.

Using a YAML File

This example shows how to configure this function during workload creation, with the key configuration marked in red:

```
kind: Deployment
apiVersion: cci/v2
metadata:
  name: deploy-example
  namespace: namespace-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: deploy-example
  template:
    metadata:
      labels:
        app: deploy-example
      annotations:
        volume.cci.io/mount-stdlog-containers: sidecar  #Name of the container where standard output logs are mounted.
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          resources:
            limits:
              cpu: '1'
              memory: 2Gi
            requests:
              cpu: '1'
              memory: 2Gi
        - name: sidecar
          image: sidecar:latest
          resources:
            limits:
              cpu: '0'
              memory: '0'
            requests:
              cpu: '0'
              memory: '0'
      dnsPolicy: Default
      imagePullSecrets:
        - name: imagepull-secret
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 100%
```


Table 4-17 Pod annotations

Annotation	Type	Description	Example Value

volume.cci.io/mount-stdlog-containers	String	1. Name of the container where standard output logs are mounted. To mount the standard output logs of multiple containers, separate their names with commas (,). You can also set the value to an asterisk (*) to mount the standard output logs of all containers. If the value is set to *****, it cannot be set to any other container name.
		2. After a pod that matches the annotation starts, the directory that contains the standard output logs of all containers in the pod is mounted to the **/var/log/pods** directory of the container.
			Example 1:
			`"container-0,container-1"`
			Example 2:
			`"*"`

volume.cci.io/mount-stdlog-containers-path	String	1. Container name and mount path that the standard output logs are mounted. The value is in JSON format. If the container name is set to *****, the logs of all containers are mounted to the specified container. If the value is set to *****, it cannot be set to any other container name.
		2. After any container that matches the annotation starts, the directory that contains the standard output logs of all containers in the pod is mounted to the specified path of the container.
			Example 1:
			`"{\"container-0\":\"/var/log/pods\",\"container-1\":\"/tmp/log/pods\"}"`
			Example 2:
			`"{\"*\":\"/tmp/log/pods\"}"`

5 Cost Tag Management

Scenario

You can configure cost tags to classify and track pod costs.

Tag Naming Rules

- Each tag consists of a key-value pair. Example:
  pod-tag.cci.io/<tag-key>:tag-value
- Each pod can have a maximum of 20 tags.
- A tag key must be unique for each pod, and each tag key can have only one tag value.
- A tag consists of a tag key and a tag value. **Table 5-1** lists the tag key and value requirements.

Table 5-1 Tag naming rules

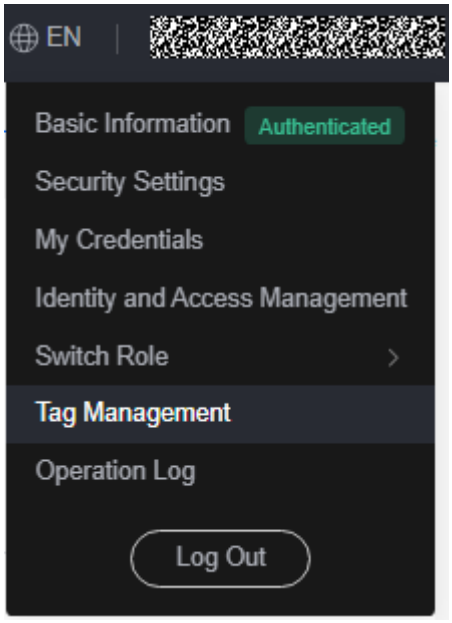
Item	Rules	Example
Tag key	• Cannot be empty. • Must be unique for each pod. • Can contain a maximum of 128 characters. • Can contain letters, digits, spaces, and special characters _.:=+-@. • Cannot start or end with a space.	Organization
Tag value	• Can contain a maximum of 255 characters. • Can contain letters, digits, spaces, and special characters _.:=+-@/.	Apache

Procedure

Step 1 Log in to the management console.

Step 2 In the upper right corner of the page, click the username and select **Tag Management** from the drop-down list.

Figure 5-1 Tag Management



Step 3 Create a predefined tag. In the navigation pane, choose **Predefined Tags**. Click **Create Tag**, enter the tag key and value, and click **OK**.

Step 4 In the top navigation bar, choose **Billing** > **Cost Center**.

Step 5 In the navigation pane, choose **Cost Organization** > **Cost Tags**. On the **Cost Tags** page, select the tag created in **Step 3**, click **Activate**, and then click **OK**.

Step 6 Wait until the tag status changes to **Activated**.

Step 7 Add the following fields to the **Podtemplate.metadata.labels** file when creating a pod in CCI. The following is an example YAML file:

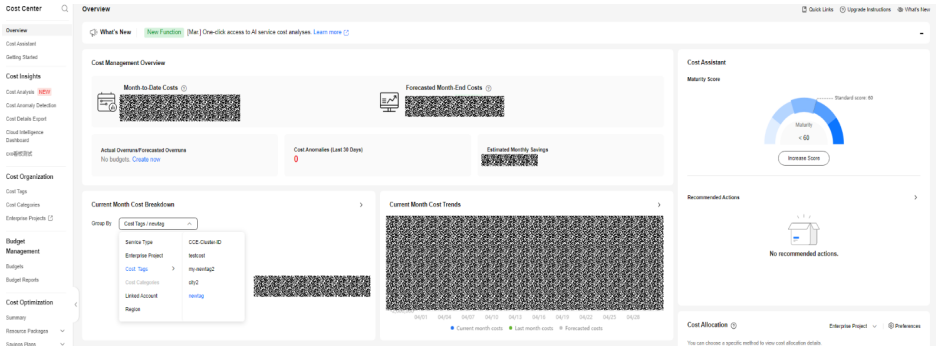
```
kind: Deployment
apiVersion: cci/v2
metadata:
  name: tag-example
  namespace: bursting-d0089910820250427-2031
spec:
  replicas: 3
  selector:
    matchLabels:
      app: tag-example
  template:
    metadata:
      labels:
        app: tag-example
        pod-tag.cci.io/newtag: tag-example-1  # pod-tag.cci.io/<custom-tag-key>: custom-tag-value
    spec:
      containers:
        - name: deploy-example
          image: swr.region-id.domain.com/org-name/image-name:tag
          env:
            - name: ENV1
              value: 'false'
            - name: ENV2
```



```
            value: xxx
        resources:
          limits:
            cpu: 500m
            memory: 1Gi
          requests:
            cpu: 500m
            memory: 1Gi
      dnsPolicy: Default
      imagePullSecrets:
        - name: imagepull-secret
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 0
    maxSurge: 100%
```

Step 8 In the top navigation bar, choose **Billing** > **Cost Center**. In the navigation pane, choose **Overview**. In the **Current Month Cost Breakdown** area, select **Cost Tags** and then the created tag to view the cost details.

Figure 5-2 Current Month Cost Breakdown



----End

6 Auditing

6.1 CCI Operations Supported by CTS

Cloud Trace Service (CTS) records operations on cloud service resources, allowing you to query, audit, and backtrack the resource operation requests initiated from the CCI console or open APIs as well as responses to the requests.

Table 6-1 CCI operations that can be recorded by CTS

Operation	Trace Name
Creating a Service	createService
Deleting a Service	deleteService
Replacing a Service	replaceService
Updating a Service	updateService
Creating a Deployment	createDeployment
Deleting a Deployment	deleteDeployment
Replacing a Deployment in a specified namespace	replaceDeployment
Updating a Deployment in a specified namespace	updateDeployment
Creating a namespace	createNamespace
Deleting a namespace	deleteNamespace
Creating a pod	createPod
Updating a pod	updatePod
Replacing a pod	replacePod
Deleting a pod	deletePod

Operation	Trace Name
Replacing the ObservabilityConfiguration	replaceObservabilityconfiguration
Creating a ConfigMap	createConfigmap
Updating a ConfigMap	updateConfigmap
Replacing a ConfigMap	replaceConfigmap
Deleting a ConfigMap	deleteConfigmap
Creating a secret	createSecret
Updating a secret	updateSecret
Replacing a secret	replaceSecret
Deleting a secret	deleteSecret
Deleting a network	deleteNetwork
Creating a network	createNetwork
Updating a network	updateNetwork
Replacing a network	replaceNetwork
Creating a PVC	createPersistentvolumeclaim
Replacing a PVC	replacePersistentvolumeclaim
Updating a PVC	updatePersistentvolumeclaim
Deleting a PVC	deletePersistentvolumeclaim
Creating an image snapshot	createImagesnapshot
Deleting an image snapshot	deleteImagesnapshot
Querying an image snapshot	getImagesnapshot
Querying all image snapshots in a specified namespace	listImagesnapshots
Creating a PV	createPV
Updating a PV	updatePV
Replacing a PV	replacePV
Deleting a PV	deletePV
Querying a PV	getPV
Listing all PVs in a specified namespace	listPVs
Querying a ReplicaSet	getReplicaset

Operation	Trace Name
Listing all ReplicaSets in a specified namespace	listReplicasets
Creating an HPA policy	createHPA
Deleting an HPA policy	deleteHPA
Updating an HPA policy	updateHPA
Replacing an HPA policy	replaceHPA
Querying an HPA policy	getHPA
Listing all HPA policies in a specified namespace	listHPAs
Creating a PoolBinding	createPoolbinding
Deleting a PoolBinding	deletePoolbinding
Querying a PoolBinding	getPoolbinding
Listing all PoolBindings in a specified namespace	listPoolbindings

6.2 Viewing a Trace

Scenarios

Once enabled, CTS starts recording operations on CCI resources. Operation records of the last seven days can be viewed on the CTS console.

Procedure

Step 1 Log in to the management console.

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

Step 3 Specify the filters used for querying traces. You can query traces using a combination of the following filters:

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

  Select the desired filters from the drop-down lists. Select **CCI** from the **Trace Source** drop-down list.

  If you select **Trace name** for **Search By**, you need to select a trace name.

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

  If you select **Resource name**, you need to select a resource name.
- **Operator**: Select an operator (a user not a tenant).
- **Trace Status**: Available options include **All trace statuses**, **Normal**, **Warning**, and **Incident**. You can select only one of them.

- Start time and end time: You can specify the time period in which to query traces.

Step 4 Click on the left of a trace to expand its details.

Step 5 Click **View Trace** in the **Operation** column. In the displayed dialog box, the trace structure details are displayed.

----End