Cloud Container Instance

Developer Guide

Issue 01

Date 2024-11-04

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.

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

Trademarks and Permissions

HUAWEI 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.

Contents

1 Overview..1

2 Using Native kubectl (Recommended)..3

2.1 kubectl Configuration Guide.. 3

2.2 cci-iam-authenticator Usage Reference.. 7

3 Namespace and Network..12

4 Pod.. 19

4.1 Pod.. 19

4.2 Environment Variables.. 22

4.3 Startup Command.. 23

4.4 Initializing a Container..24

4.5 Calculating the Effective Resource Request or Limit of a Pod...25

4.6 Lifecycle Management.. 26

4.7 Liveness Probe..27

5 Label.. 31

6 Deployment.. 34

7 EIPPool.. 38

7.1 Overview... 38

7.2 Creating an EIPPool...38

7.2.1 Creating a Dynamic EIPPool... 39

7.2.2 Creating a Static EIPPool... 41

7.3 Using an EIPPool.. 41

7.4 Managing an EIPPool.. 42

8 EIP... 44

8.1 Overview... 44

8.2 Binding a New EIP to a Pod... 45

8.3 Binding an Existing EIP to a Pod... 47

9 Pod Resource Monitoring Metric...49

10 Collecting Pod Logs.. 56

11 Managing Network Access Through Service and Ingress.......................................61

11.1 Service.......... 61
11.2 Ingress.......... 66
11.3 Network Access Scenarios.......... 69
11.4 Readiness Probe.......... 70
12 Using PersistentVolumeClaim to Apply for Persistent Storage.......... 75
13 ConfigMap and Secret..........78
13.1 ConfigMap.......... 78
13.2 Secret.......... 79
14 Creating a Workload Using Job and Cron Job..........82
A YAML Syntax.......... 85

1 Overview

Cloud Container Instance (CCI) is a serverless container service that allows you to run containers without creating and managing server clusters. Under the serverless model, CCI allows you to directly create and use containerized workloads on the console or by using kubectl or Kubernetes APIs, and pay only for the resources consumed by these workloads.

This document describes how to use **kubectl** or call **CCI APIs** to implement functions.

Document Organization

This document includes:

1. **Using kubectl**

   This section describes how to configure kubectl on CCI. CCI allows you to use native or customized kubectl to create resources such as workloads. Native kubectl is recommended.

2. **Namespace and Network**

   This section describes the concepts of the namespace and network.

3. **Pod**

   This section describes the concept of the pod and how to use pods.

4. **Label**

   This section describes the functions of labels and how to use labels.

5. **Deployment**

   This section describes the application scenarios of Deployments, and how to deploy container images to CCI using a Deployment.

6. **Service and Ingress**

   This section describes how to use services and ingresses to manage workload access.

   – Service: an abstraction which defines a logical set of pods and a policy by which to access them.

   – Ingress: an API object that manages external access.

7. **Persistent Storage**

This section describes how to use storage in workloads. That is, how to use storage volumes in containers. Storage types that can be used include Elastic Volume Service (EVS), Scalable File Service (SFS), and Object Storage Service (OBS).

8. **ConfigMap and Secret**

   This section describes how to use ConfigMaps and secrets.

   ConfigMaps and secrets are used to store configuration and sensitive information to enable easier and flexible workload configuration.

9. **Job and Cron Job**

   This section describes how to use jobs. A job is a one-off task.

2 Using Native kubectl (Recommended)

2.1 kubectl Configuration Guide

CCI allows you to use native or customized kubectl to create resources such as workloads. Native kubectl is recommended.

Downloading kubectl

Download the kubectl of version 1.19 from the **Kubernetes version release page**.

NOTE

For a device that uses the Apple M1 chip, **download** the kubectl corresponding to the darwin-arm64 architecture.

Downloading cci-iam-authenticator

Download the cci-iam-authenticator binary from the CCI official website. **The latest version is v2.6.17.**

Table 2-1 lists the addresses for downloading cci-iam-authenticator.

Table 2-1 Download addresses

Operating System	Download Address	View Help
Linux AMD 64-bit	**cci-iam-authenticator_linux-amd64** **cci-iam-authenticator_linux-amd64_sha256**	**cci-iam-authenticator Usage Reference**
Darwin AMD 64-bit	**cci-iam-authenticator_darwin-amd64** **cci-iam-authenticator_darwin-amd64.sha256**	

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operating System	Download Address	View Help
Darwin Arm 64-bit	**cci-iam-authenticator_darwin-arm64** **cci-iam-authenticator_darwin-arm64.sha256**	

Installing and Configuring kubectl

Perform the following operations to install and configure kubectl on a Linux OS. For more details, see **Install Tools**.

Step 1 Grant the execute permission on kubectl downloaded in **Downloading kubectl** and save it to the PATH directory.

chmod +x ./kubectl

mv ./kubectl *$PATH*

In the preceding command, *$PATH* indicates the PATH directory (for example, **/usr/local/bin**). Replace it with the actual value.

You can run the following command to view the kubectl version:

kubectl version --client=true

```
Client Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.0",
GitCommit:"e19964183377d0ec2052d1f1fa930c4d7575bd50", GitTreeState:"clean",
BuildDate:"2020-08-26T14:30:33Z", GoVersion:"go1.15", Compiler:"gc", Platform:"linux/amd64"}
```

Step 2 Configure IAM authentication information and persistently store it to the local host.

1. Grant the execute permission on cci-iam-authenticator downloaded in **Downloading cci-iam-authenticator** and save it to the PATH directory.

   chmod +x ./cci-iam-authenticator

   mv ./cci-iam-authenticator *$PATH*

2. Initialize the cci-iam-authenticator configuration.

   You can initialize the cci-iam-authenticator configuration by using either of the following methods:

   – Using AK/SK

     cci-iam-authenticator generate-kubeconfig --cci-endpoint=*https://$endpoint* **--ak=***xxxxxxx* **--sk=***xxxxxx*

     endpoint is an endpoint of CCI. For details about CCI endpoints, see **Regions and Endpoints**. For details about how to obtain the AK and SK, see **Obtaining an AK/SK**. **ak** is the access key in the file, and **sk** is the secret key in the file.

     For example, if **endpoint** is **https://cci.cn-north-4.myhuaweicloud.com**, the value of **ak** is **my-ak**, and the value of **sk** is **ABCDEFAK..**, run the following command:

     cci-iam-authenticator generate-kubeconfig --cci-endpoint=*https://cci.cn-north-4.myhuaweicloud.com* **--ak=***my-ak* **--sk=***ABCDEFAK..*

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Information similar to the following is displayed:

```
Switched to context "cci-context-cn-north-4-my-ak"
```

In the preceding command, **cci-context-cn-north-4-my-ak** is the context name, which can be viewed by running **kubectl config get-contexts**.

– Using username and password

cci-iam-authenticator generate-kubeconfig --cci-endpoint=*https://$endpoint* **--domain-name=***xxxxxxx* **--user-name=***xxxxxx* **--password=***'xxxxxx'*

In the preceding command, **endpoint** is the CCI endpoint, **domain-name** is the tenant name, **user-name** is the IAM username, and **password** is the IAM user password. Replace them with the actual values.

NOTE

- If there is no IAM user, set **user-name** to the same value as **domain-name**. Alternatively, do not set **user-name**.
- For details about IAM endpoints, see **Regions and Endpoints**. Note that the endpoint must be in the same region as CCI.
- If kubectl is used in an insecure environment, you are advised to configure environment variables to reconfigure the authentication information after this step is complete. For details, see **Configuring kubectl in an Insecure Environment**.

Step 3 After the configuration is complete, you can run kubectl commands to perform operations on CCI resources.

For example, run the following command to view the namespace resources in CN North-Beijing4:

kubectl get ns

```
No resources found.
```

The command output shows that there is no namespace in CN North-Beijing4. Before creating resources in CCI, create a namespace by following the procedure described in **Namespace and Network**.

----End

WARNING

- When you access public cloud services through APIs, the username/password or AK/SK pair is required to encrypt the requests. This mechanism ensures the confidentiality and integrity of the requests as well as the correctness of the identities of both parties. Keep the **$HOME/.kube/config** configuration file secure to prevent unauthorized use of the AK/SK.
- If the function of caching tokens is enabled to improve access performance, the tokens are saved as files in the **$HOME/.cci/cache** directory. Delete the tokens in a timely manner when they are unnecessary.
- If your access key is used by an unauthorized person due to loss or leakage, you can delete the access key or notify the administrator of resetting it and then configure the access key again.
- Deleted access keys cannot be recovered.

Configuring kubectl in an Insecure Environment

Step 1 Follow the preceding steps to install and configure kubectl.

Step 2 Edit the kubeconfig file and delete sensitive information.

On a Linux OS, the kubeconfig file is stored in **$HOME/.kube/config** by default.

Table 2-2 Sensitive information to be deleted

Command Flag	Environment Value	Description
--domain-name	DOMAIN_NAME	Tenant name
--user-name	USER_NAME	Sub-user name
--password	PASSWORD	User password
--ak	ACCESS_KEY_ID	Access key ID
--sk	SECRET_ACCESS_KEY	Secret access key
--cache	CREDENTIAL_CACHE	Whether to cache tokens

For details about more parameters, see **cci-iam-authenticator Usage Reference**.

Step 3 Configure the environment variables corresponding to the deleted parameters. For example, configure AK, SK, and whether to cache tokens.

NOTE

Hardcoded or plaintext AK and SK are risky. For security, encrypt your AK and SK and store them in the configuration file or as environment variables.

export ACCESS_KEY_ID={Access Key} #Replace it with **HUAWEICLOUD_SDK_AK**.

export SECRET_ACCESS_KEY={Secret Key} #Replace it with **HUAWEICLOUD_SDK_SK**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

export CREDENTIAL_CACHE=false

kubectl get ns

Information similar to the following is displayed:

```
No resources found.
```

----End

Obtaining an AK/SK

AK: access key ID. It is a unique ID associated with an SK. AK is used together with SK to sign requests.

SK: secret access key. It is used together with an AK to sign requests. They can identify request senders and prevent requests from being modified.

Step 1 Log in to the management console.

Step 2 Hover over the username and select **My Credentials** from the drop-down list.

Step 3 Choose **Access Keys** from the navigation pane.

Step 4 Click **Create Access Key**, and enter the verification code.

Step 5 Click **OK** to generate an access key and download it.

NOTE

Keep the AK/SK file confidential to prevent information leakage.

----End

Obtaining CCI Endpoints

Obtain the endpoint from the **Regions and Endpoints** page, as shown in the following table.

Table 2-3

Region Name	Region	Endpoint
LA-Sao Paulo1	sa-brazil-1	cci.sa-brazil-1.myhuaweicloud.com

2.2 cci-iam-authenticator Usage Reference

cci-iam-authenticator is the authentication add-on of the Kubernetes client and provides two subcommands **generate-kubeconfig** and **token**.

```
A tool to authenticate to CCI using HuaweiCloud IAM credentials

Usage:
  cci-iam-authenticator [command]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
Available Commands:
  generate-kubeconfig Generate or modify kubeconfig files based on user configuration
  help                Help about any command
  token               Authenticate using HuaweiCloud IAM and get token for CCI

Flags:
      --alsologtostderr   log to standard error as well as files
  -h, --help              help for cci-iam-authenticator
      --log_dir string    If non-empty, write log files in this directory
      --log_file string   If non-empty, use this log file
      --logtostderr       log to standard error instead of files (default true)
  -v, --v Level           number for the log level verbosity
      --version           version for cci-iam-authenticator

Use "cci-iam-authenticator [command] --help" for more information about a command.
```

The **Flags** field lists log options.

token

The **token** command is used to obtain a user token. You can use either the username/password or AK/SK to obtain a token.

```
Authenticate using HuaweiCloud IAM and get token for CCI

Usage:
  cci-iam-authenticator token [flags]

Flags:
      --ak string                   IAM access key ID
      --cache                       Cache the token credential on disk until it expires (default true)
      --domain-name string          IAM domain name, typically your account name
  -h, --help                        help for token
      --iam-endpoint string         HuaweiCloud IAM endpoint, i.e. https://iam.cn-north-4.myhuaweicloud.com
(default "https://iam.myhuaweicloud.com")
      --insecure-skip-tls-verify    If true, the iam server's certificate will not be checked for validity. (default
true)
      --password string             IAM user password
      --project-id string           IAM project id, project id and project name should not be empty at same time
      --project-name string         IAM project name, project id and project name should not be empty at same
time
      --sk string                   IAM secret access key
      --token-only                  Return token only for other tool integration
      --user-name string            IAM user name. Same as domain-name when using main account, otherwise
use iam user name
```

The **Flags** field includes username, password, AK, SK, and common configurations.

Table 2-4 Username and password

Command Flag	Environment Value	Description
domain-name	DOMAIN_NAME	Tenant name, that is, the account name. For details, see **My Credentials**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Command Flag	Environment Value	Description
user-name	USER_NAME	Sub-user name, that is, the IAM username. If this parameter is not specified, the value of **domain-name** is used. For details, see **My Credentials**.
password	PASSWORD	Password of an account or IAM user.

Table 2-5 AK/SK

Command Flag	Environment Value	Description
ak	ACCESS_KEY_ID	For details about how to obtain the AK and SK, see **Obtaining an AK/SK**. AK is the access key in the file, and SK is the secret key in the file.
sk	SECRET_ACCESS_KEY	

Table 2-6 Common configurations

Command Flag	Environment Value	Description
iam-endpoint	IAM_ENDPOINT	IAM endpoint, which is mandatory. For details, see **Regions and Endpoints**.
project-name	PROJECT_NAME	Project name. For details, see **My Credentials**.
project-id	PROJECT_ID	Project ID. For details, see **My Credentials**.
insecure-skip-tls-verify	INSECURE_SKIP_TLS_VERIFY	Whether to skip the verification of the CCI or IAM server. The default value is **true**.

Command Flag	Environment Value	Description
cache	CREDENTIAL_CACHE	Whether to cache the IAM token to the local host to improve the access performance. The default value is **true**. **CAUTION** In an insecure environment, you are advised to disable this option.

generate-kubeconfig

This command is used to generate kubeconfig configurations. If the specified kubeconfig file already exists, new server, user, and context configurations will be injected and the newly injected context will be used for authentication. By default, the system verifies the user configuration and attempts to access IAM and CCI to ensure that the IAM authentication information is correct and the CCI address is available.

```
Generate or modify kubeconfig files based on user configuration.

Sets a cluster entry, a user entry and a context entry in kubeconfig and use this context as the current-
context.

  The loading order follows these rules:

   1.  If the --kubeconfig flag is set, then only that file is loaded. The flag may only be set once and no
merging takes
place.
   2.  If $KUBECONFIG environment variable is set, then it is used as a list of paths (normal path delimiting
rules for
your system). These paths are merged. When a value is modified, it is modified in the file that defines the
stanza. When
a value is created, it is created in the first file that exists. If no files in the chain exist, then it creates the
last file in the list.
   3.  Otherwise, ${HOME}/.kube/config is used and no merging takes place.

 Examples:
   # Generate kubeconfig to ${HOME}/.kube/config using aksk
   cci-iam-authenticator generate-kubeconfig --cci-endpoint=https://cci.cn-north-4.myhuaweicloud.com --
ak=*** --sk=***
   # Generate kubeconfig to ${HOME}/.kube/config using domain name and password
   cci-iam-authenticator generate-kubeconfig --cci-endpoint=https://cci.cn-north-4.myhuaweicloud.com --
domain-name=*** --password=***

Usage:
  cci-iam-authenticator generate-kubeconfig [flags]

Flags:
      --ak string                  IAM access key ID
      --cache                      Cache the token credential on disk until it expires (default true)
      --cci-endpoint string        CCI server endpoint, i.e. https://cci.cn-north-4.myhuaweicloud.com
      --domain-name string         IAM domain name, typically your account name
  -h, --help                       help for generate-kubeconfig
      --iam-endpoint string        HuaweiCloud IAM endpoint, i.e. https://iam.cn-north-4.myhuaweicloud.com
(default "https://iam.myhuaweicloud.com")
      --insecure-skip-tls-verify   If true, the iam server's certificate will not be checked for validity. (default
true)
      --kubeconfig string          use a particular kubeconfig file
```

```
      --password string            IAM user password
      --project-id string          IAM project id, project id and project name should not be empty at same time
      --project-name string        IAM project name, project id and project name should not be empty at same
time
      --sk string                  IAM secret access key
      --token-only                 Return token only for other tool integration
      --user-name string           IAM user name. Same as domain-name when using main account, otherwise
use iam user name
      --validation                 Validate kubeconfig by trying to access CCI with existing config (default true)
```

A kubeconfig file can contain multiple environment and authentication information records. You can use the same IAM authentication configuration and different **cci-endpoint** values to generate kubeconfig configurations for multiple regions. For example:

```
# Generate kubeconfig configuration for CN North-Beijing4 and switch to the corresponding context.
$ cci-iam-authenticator generate-kubeconfig --cci-endpoint=https://cci.cn-north-4.myhuaweicloud.com --
ak=my-ak --sk=xxxxxx
Switched to context "cci-context-cn-north-4-my-ak".
# Generate kubeconfig configuration for CN East-Shanghai1 and switch to the corresponding context.
$ cci-iam-authenticator generate-kubeconfig --cci-endpoint=https://cci.cn-east-3.myhuaweicloud.com --
ak=my-ak --sk=xxxxxx
Switched to context "cci-context-cn-east-3-my-ak".
# Switch to the context of CN North-Beijing4.
$ kubectl config use-context cci-context-cn-north-4-my-ak
```


Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3 Namespace and Network

A namespace provides a method of allocating resources among multiple users. It applies to scenarios where multiple teams or projects exist. Currently, CCI provides two types of resources: general-computing and GPU-accelerated resources. When creating a namespace, you need to select a resource type. Subsequently, new workloads will run on this type of cluster.

- **General-computing**: container instances and workloads with CPU resources. This namespace type is suitable for general computing scenarios.
- **GPU-accelerated**: container instances and workloads with GPU resources. This namespace type is suitable for scenarios such as deep learning, scientific computing, and video processing.

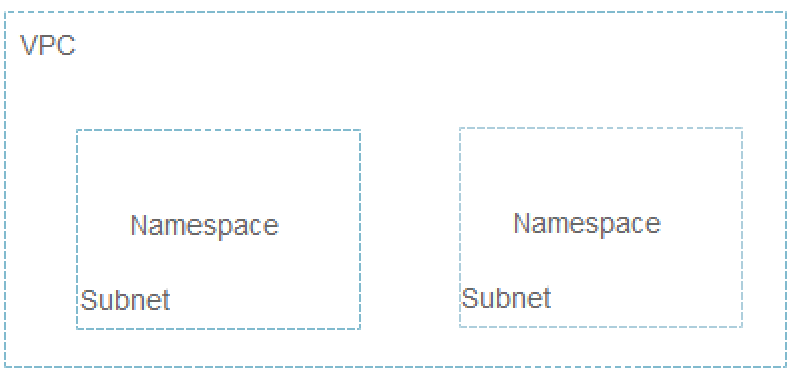
A network is a Kubernetes resource object extended for CCI. You can associate a network with a Virtual Private Cloud (VPC) and subnet so that CCI can use network resources of the public cloud.

Relationship Between a Namespace and Network

A namespace corresponds to a subnet in a VPC, as shown in **Figure 3-1**. When a namespace is created, it will be associated with an existing VPC or a newly created VPC, and a subnet will be created under the VPC. In this namespace, resources such as pods and Services are created in the corresponding VPC and subnet, and the IP addresses in the subnet are used.

If you want to run resources of multiple services in the same VPC, you need to plan subnet CIDR blocks and IP addresses.

Figure 3-1 Relationship between namespaces and VPC subnets



Scenarios Where Multiple Namespaces Are Used

Because namespaces enable partial environment isolation, you can create different namespaces, such as production, test, and development namespaces based on project attributes when there are a large number of projects and persons.

Creating a Namespace

Under a namespace, a network is required to associate with a VPC and subnet. After a namespace is created, a network needs to be created.

> NOTE
>
> In most cases, namespaces do not need to be frequently created. You are advised to log in to the CCI console to create a namespace. For details, see **Namespace**.

In the following example, create a namespace named **namespace-test**, and specify the CCI resource type to **general-computing**.

```
apiVersion: v1
kind: Namespace
metadata:
  name: namespace-test
  labels:
    sys_enterprise_project_id: "0"
  annotations:
    namespace.kubernetes.io/flavor: general-computing
spec:
  finalizers:
  - kubernetes
```

The definition file is in the YAML or JSON format. For more details about the YAML format, see **YAML Syntax**.

- **sys_enterprise_project_id**: enterprise project ID, which can be obtained from the project details page on the **Enterprise Project Management (EPS)** console. This field does not need to be set if you have not enabled EPS. If this parameter is not set, the default value **0** is used, indicating the **default** enterprise project.
- **namespace.kubernetes.io/flavor: general-computing**: namespace type.

  There are two namespace types:

- **General-computing**: container instances and workloads with CPU resources. This namespace type is suitable for general computing scenarios.
- **GPU-accelerated**: container instances and workloads with GPU resources. This namespace type is suitable for scenarios such as deep learning, scientific computing, and video processing.

If the file name of the namespace definition is **ns.yaml**, run **kubectl create -f ns.yaml** to create a namespace. **-f** indicates that the namespace is created from a file.

```
# kubectl create -f ns.yaml
namespace/namespace-test created
```

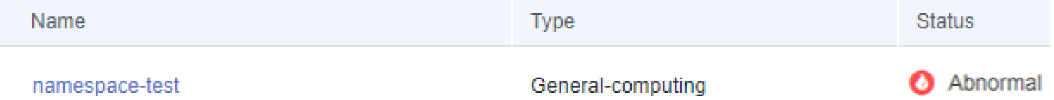
Run **kubectl get ns** to check whether the namespace is successfully created. In this command, **ns** indicates the namespace.

```
# kubectl get ns
NAME           STATUS    AGE
namespace-test   Active    23s
```

The preceding information indicates that the namespace **namespace-test** is created successfully and the duration is 23 seconds.

Log in to the CCI console. In the navigation pane, choose **Namespaces**. You can see that the namespace is created successfully but the status is **Abnormal**. This is because in CCI, you need to define a network policy for the namespace. For details, see **Creating a Network**.

Figure 3-2 Namespace - abnormal



Creating a Network

After creating a namespace, you need to create a network policy for the namespace and associate the namespace with the VPC and subnet.

The following example shows how to create a network named **test-network**.

```
apiVersion: networking.cci.io/v1beta1
kind: Network
metadata:
  annotations:
    network.alpha.kubernetes.io/default-security-group: security-group-id
    network.alpha.kubernetes.io/domain-id: domain-id
    network.alpha.kubernetes.io/project-id: project-id
  name: test-network
spec:
  cidr: 192.168.0.0/24
  attachedVPC: vpc-id
  networkID: network-id
  networkType: underlay_neutron
  subnetID: subnet-id
```


Copyright © Huawei Cloud Computing Technologies Co., Ltd.

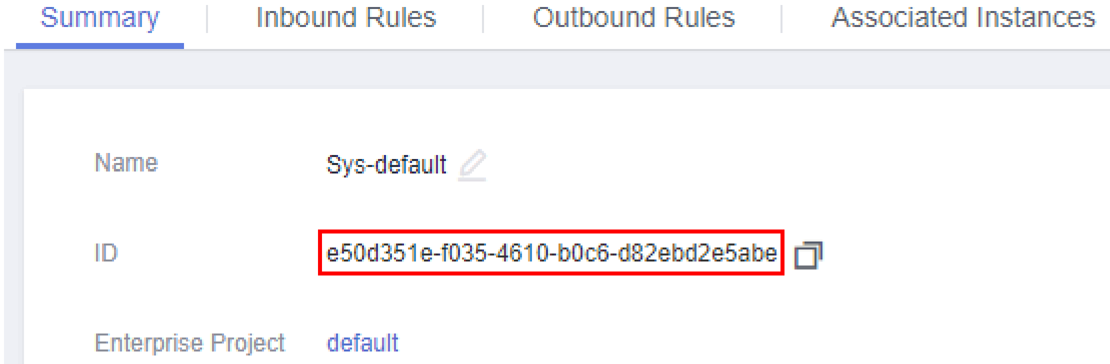
NOTE

The CIDR blocks of the VPC and subnet cannot be **10.247.0.0/16**, which is the CIDR block reserved by CCI for Services. If you use this CIDR block, IP address conflicts may occur, which may result in workload creation failures or service unavailability. If you do not need to access pods through Services, you can allocate this CIDR block to a VPC.

You can obtain the preceding parameters as follows:

- **network.alpha.kubernetes.io/domain-id**: account ID, which can be obtained from **My Credentials**.
- **network.alpha.kubernetes.io/project-id**: project ID, which can be obtained from **My Credentials**.
- **network.alpha.kubernetes.io/default-security-group**: security group ID, which can be obtained from the **Security Groups** page.

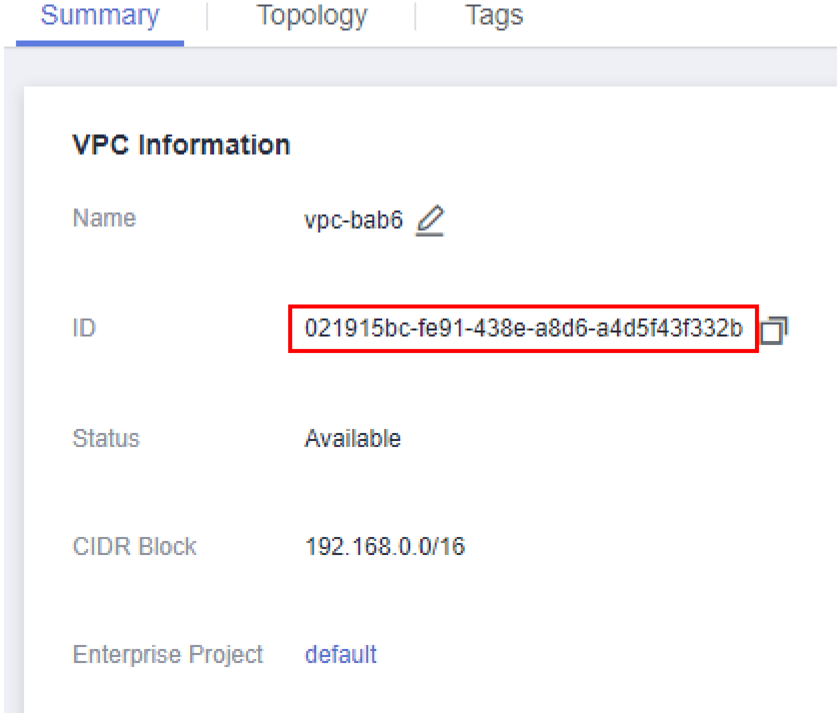
Figure 3-3 Obtaining the security group ID



- **cidr**: subnet CIDR block.
- **attachedVPC**: VPC ID, which can be obtained from the VPC console.

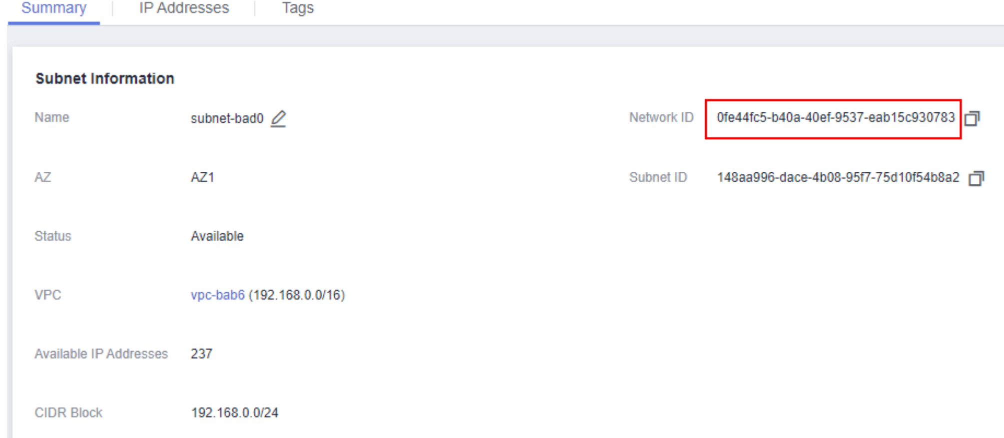
Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-4 Obtaining the VPC ID



- **networkID**: subnet network ID, which can be obtained by choosing **Virtual Private Cloud** under **Network** and then choosing **Subnets** in the navigation pane.

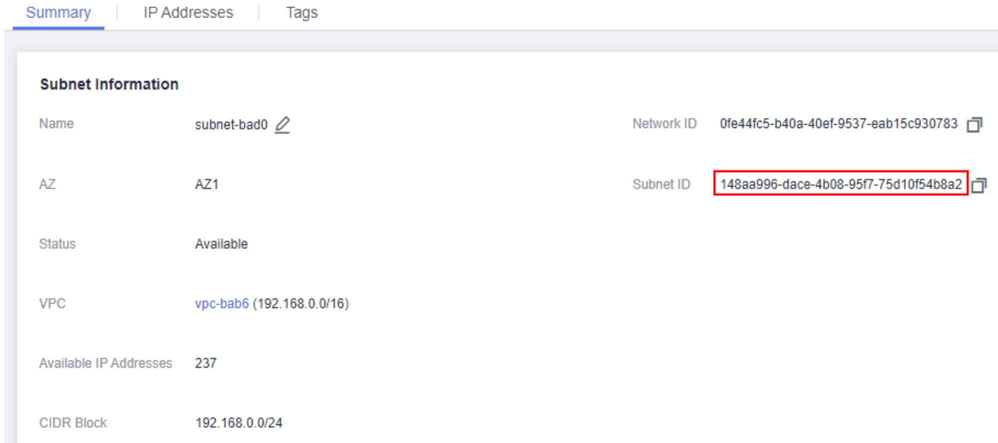
Figure 3-5 Obtaining the subnet network ID



- **networkType**: network type. Currently, only the **underlay_neutron** network type is supported.
- **subnetID**: subnet ID, which can be obtained by choosing **Virtual Private Cloud** under **Network** and then choosing **Subnets** in the navigation pane.

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-6 Obtain the subnet ID.

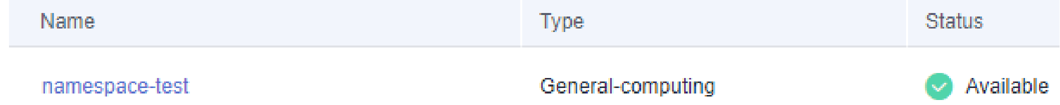


If the file name of the network definition is **network.yaml**, run **kubectl create -f network.yaml** to create a namespace. **-f** indicates that the namespace is created from a file. **namespace namespace-test** indicates that it is created in the namespace **namespace-test**.

```
# kubectl create -f network.yaml --namespace namespace-test
network.networking.cci.io/test-network created
```

Log in to the CCI console. In the navigation pane, choose **Namespaces**. You can see that the namespace is created successfully and the status is **Available**.

Figure 3-7 Namespace - available



Specifying a Namespace for the kubectl Context

The network above is created in a specified namespace. The subsequent resources are created in a namespace. It is time-consuming to specify the namespace each time. You can specify the namespace for a kubectl context. In this way, the resources created in the context are all under this namespace, which facilitates operations.

To specify the namespace, you only need to add the **--namespace** option to the context setting command, as shown in the following command:

kubectl config set-context *$context* **--namespace**=*$ns*

In the preceding command, *$ns* indicates the namespace name, and *$context* indicates the context name, which can be customized or obtained by running the following command:

```
# kubectl config get-contexts
CURRENT   NAME                                          CLUSTER                AUTHINFO
NAMESPACE
          cci-context-cn-east-3-1C8PNI0POPPCSFGXPM6S   cci-cluster-cn-east-3   cci-user-cn-
east-3-1C8PNI0POPPCSFGXPM6S
*         cci-context-cn-east-3-hwuser_xxx             cci-cluster-cn-east-3   cci-user-cn-east-3-hwuser_xxx
          kubernetes-admin@kubernetes                   kubernetes              kubernetes-admin
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For example, if the namespace created above is named **namespace-test**, the command is as follows:

```
# kubectl config set-context cci-context --namespace=namespace-test
```

After a namespace is specified, you can run kubectl commands to directly operate CCI resources. For example, run **kubectl get pod** to check pod resources. The result shows that all resources are normal.

```
# kubectl get pod
No resources found.
```


Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4 Pod

4.1 Pod

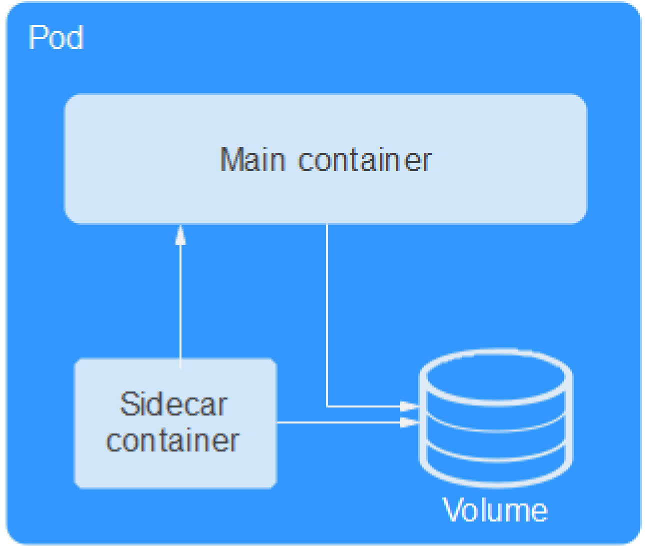
What Is a Pod?

A pod is the smallest and simplest unit in the Kubernetes object model that you create or deploy. A pod encapsulates one or more containers, storage resources, a unique network IP address, and options that govern how the container(s) should run.

Pods can be used in either of the following ways:

- One container runs in one pod. This is the most common usage of pods in Kubernetes. You can view the pod as a single encapsulated container, but Kubernetes directly manages pods instead of containers.
- Multiple containers that need to be coupled and share resources run in a pod. In this scenario, an application contains a main container and several sidecar containers, as shown in **Figure 4-1**. For example, the main container is a web server that provides file services from a fixed directory, and the sidecar container periodically downloads files to the directory.

Figure 4-1 Pod



In Kubernetes, pods are rarely created directly. Instead, controllers such as Deployments and jobs, are used to manage pods. Controllers can create and manage multiple pods, and provide replica management, rolling upgrade, and self-healing capabilities. A controller generally uses a pod template to create corresponding pods.

Container Specifications

You can use GPUs in CCI only if the **namespace** is of the GPU-accelerated type.

Currently, three types of pods are provided, including general-computing (used in general-computing namespaces), RDMA-accelerated, and GPU-accelerated (used in GPU-accelerated namespaces). For details about the specifications, see "Pod Specifications" in **Notes and Constraints**.

Creating a Pod

Kubernetes resources can be described using YAML or JSON files. For more details about the YAML format, see **YAML Syntax**. The following example describes a pod named **nginx**. This pod contains a container named **container-0** and uses the **nginx:alpine** image, 0.5 vCPUs, and 1024 MiB memory.

```
apiVersion: v1                      # Kubernetes API version
kind: Pod                          # Kubernetes resource type
metadata:
  name: nginx                       # Pod name
spec:                              # Pod specification
  containers:
  - image: nginx:alpine             # Used image is nginx:alpine
    name: container-0               # Container name
    resources:                     # Resources required for applying for a container. The values of limits and requests in CCI must be the same.
      limits:
        cpu: 500m                   # 0.5 vCPUs
        memory: 1024Mi
      requests:
        cpu: 500m                   # 0.5 vCPUs
        memory: 1024Mi
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
imagePullSecrets:              # Secret used to pull the image, which must be imagepull-secret.
- name: imagepull-secret
```

As shown in the annotation of YAML, the YAML description file includes:

- **metadata**: Information such as name, label, and namespace
- **spec**: Pod specification such as image and volume used

If you query a Kubernetes resource, you can see the **status** field. This field indicates the status of the Kubernetes resource, and does not need to set when the resource is created. This example is a minimum set, and other parameter definition will be described later.

For the parameter description of Kubernetes resources, see **API Reference**.

After the pod is defined, you can create it using kubectl. If the YAML file is named **nginx.yaml**, run the following command to create the file. **-f** indicates that it is created in the form of a file.

```
$ kubectl create -f nginx.yaml -n $namespace_name
pod/nginx created
```

NOTE

The kernel version of the OS for running on the containers has been upgraded from 4.18 to 5.10.

GPU-accelerated pods support the following GPU specifications:

- **nvidia.com/gpu-tesla-v100-16GB**: NVIDIA Tesla V100 16GB
- **nvidia.com/gpu-tesla-v100-32GB**: NVIDIA Tesla V100 32GB
- **nvidia.com/gpu-tesla-t4**: NVIDIA Tesla T4 GPU

Container Images

A container image is a special file system, which provides the programs, libraries, resources, and configuration files required for running containers. A container image also contains configuration parameters, for example, for anonymous volumes, environment variables, and users. An image does not contain any dynamic data. Its content remains unchanged after being built.

SoftWare Repository for Container (SWR) has synchronized some common images from the container registry so that you can use the images named in the format of **Image name:Tag** (for example, **nginx:alpine**) on the internal network. You can query the synchronized images on the SWR console.

Viewing Pod Information

After the pod is created, you can run the **kubectl get pods** command to query the pod information, as shown below.

```
$ kubectl get pods -n $namespace_name
NAME        READY   STATUS    RESTARTS   AGE
nginx       1/1     Running   0          40s
```

The preceding information indicates that the **nginx** pod is in the **Running** state, indicating that the pod is running. **READY** is **1/1**, indicating that there is one container in the pod, and the container is in the **Ready** state.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

You can run the **kubectl get** command to query the configuration information about a pod. In the following command, **-o yaml** indicates that the pod is returned in YAML format. **-o json** indicates that the pod is returned in JSON format.

```
$ kubectl get pod nginx -o yaml -n $namespace_name
```

You can also run the **kubectl describe** command to view the pod details.

```
$ kubectl describe pod nginx -n $namespace_name
```

Deleting a Pod

When a pod is deleted, Kubernetes stops all containers in the pod. Kubernetes sends the SIGTERM signal to the process and waits for a period (30 seconds by default) to stop the container. If it is not stopped within the period, Kubernetes sends a SIGKILL signal to kill the process.

You can stop and delete a pod in multiple methods. For example, you can delete a pod by name, as shown below.

```
$ kubectl delete po nginx -n $namespace_name
pod "nginx" deleted
```

Delete multiple pods at one time.

```
$ kubectl delete po pod1 pod2 -n $namespace_name
```

Delete all pods.

```
$ kubectl delete po --all -n $namespace_name
pod "nginx" deleted
```

Delete pods by labels. For details about **labels**, see the next section.

```
$ kubectl delete po -l app=nginx -n $namespace_name
pod "nginx" deleted
```

4.2 Environment Variables

Environment variables are set in the container running environment.

They provide great flexibility for applications. You can use environment variables in applications, assign values to environment variables when creating containers, and read the values of environment variables when containers are running, realizing flexible configuration. With environment variables, you do not need to rewrite application files to create images.

You can also use ConfigMap and secret as environment variables. For details, see **ConfigMap and Secret**.

The following shows how to use an environment variable. You only need to configure the **spec.containers.env** field.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
    containers:
    - image: nginx:1
      name: container-0
```

```
    resources:
      limits:
        cpu: 500m
        memory: 1024Mi
      requests:
        cpu: 500m
        memory: 1024Mi
      env:                          # Environment variable
    - name: env_key
      value: env_value
    - name: pod_name
       valueFrom:                   # Name of the referenced pod
        fieldRef:
          fieldPath: metadata.name
    - name: pod_ip
      valueFrom:                    # IP address of the referenced pod
        fieldRef:
          fieldPath: status.podIP
  imagePullSecrets:
  - name: imagepull-secret
```

4.3 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 may configure or initialize MySQL databases before running MySQL servers. You can set **ENTRYPOINT** or **CMD** in the Dockerfile when creating an image.

In the following Dockerfile, the **ENTRYPOINT ["top", "-b"]** command will be executed when the container is started.

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

When calling an API, you only need to configure the **containers.command** parameter of the pod. This parameter is of the list type. The first parameter is the execution command, while the subsequent parameters are the command parameters.

```
apiVersion: v1
kind: Pod
metadata:
  name: Ubuntu
spec:
  containers:
  - image: Ubuntu
    name: container-0
    resources:
      limits:
        cpu: 500m
        memory: 1024Mi
      requests:
        cpu: 500m
        memory: 1024Mi
    command:                    # Startup command
    - top
    - "-b"
  imagePullSecrets:
  - name: imagepull-secret
```

4.4 Initializing a Container

Concepts

Before containers that run applications are started, one or some init containers are started first. If there are multiple init containers, they will be started in the defined sequence. The application containers are started only after all init containers run to completion and exit. Storage volumes in a pod are shared. Therefore, the data generated in the init containers can be used by the application containers.

Init containers can be used in multiple Kubernetes resources, such as Deployments, DaemonSets, and jobs. They perform initialization before application containers are started.

Scenarios

Before deploying a service, you can use an init container to make preparations before the pod where the service is running is deployed. After the preparations are complete, the init container runs to completion and exits, and the container to be deployed will be started.

- **Scenario 1: Wait for other modules to be ready.** For example, an application contains two containerized services: web server and database. The web server service needs to access the database service. However, when the application is started, the database service may have not been started. The web server may fail to access database. To solve this problem, you can use an init container in the pod where web server is running to check whether database is ready. The init container runs to completion only when database is accessible. Then, the web server is started and initiates a formal access request to database.
- **Scenario 2: Initialize the configuration.** For example, the init container can check all existing member nodes in the cluster and prepare the cluster configuration information for the application container. After the application container is started, it can be added to the cluster using the configuration information.
- **Other scenarios**: For example, register a pod with a central database and download application dependencies.

For details, see **Init Containers**.

Procedure

Step 1 Edit the YAML file of the init container workload.

vi deployment.yaml

An example YAML file is provided as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  replicas: 1
  selector:
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    matchLabels:
      name: mysql
  template:
    metadata:
      labels:
        name: mysql
    spec:
      initContainers:
      - name: getresource
        image: busybox
        command: ['sleep','20']
      containers:
      - name: mysql
        image: percona:5.7.22
        imagePullPolicy: Always
        ports:
        - containerPort: 3306
        resources:
          limits:
            memory: "500Mi"
            cpu: "500m"
          requests:
            memory: "500Mi"
            cpu: "250m"
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "mysql"
```

Step 2 Create an init container workload.

kubectl create -f deployment.yaml

Information similar to the following is displayed:

```
deployment.apps/mysql created
```

----End

4.5 Calculating the Effective Resource Request or Limit of a Pod

To calculate the pod's effective resource request or limit, perform the following steps:

Step 1 Obtain the request or limit on each init container in the pod. The highest request or limit for a resource is the effective init request or limit for the resource.

Step 2 Compare the following two items. The pod's effective request or limit for a resource is the higher of the two items:

- The sum of all application container requests or limits for the resource
- The effective init request or limit for the resource

----End

The following example shows how to calculate pod specifications.

```
apiVersion: v1
kind: Pod
metadata:
  name: web-app
spec:
  initContainers:
  - name: config-generator
```

```
  image: busybox
  resources:
    requests:
      memory: "256Mi"
      cpu: "250m"
    limits:
      memory: "256Mi"
      cpu: "250m"
- name: mysql-checker
  image: centos
  resources:
    requests:
      memory: "1Gi"
      cpu: "500m"
    limits:
      memory: "1Gi"
      cpu: "500m"
containers:
- name: app
  image: images.my-company.example/app:v4
  env:
  - name: MYSQL_ROOT_PASSWORD
    value: "password"
  resources:
    requests:
      memory: "1Gi"
      cpu: "500m"
    limits:
      memory: "1Gi"
      cpu: "500m"
- name: log-aggregator
  image: images.my-company.example/log-aggregator:v6
  resources:
    requests:
      memory: "1Gi"
      cpu: "250m"
    limits:
      memory: "1Gi"
      cpu: "250m"
```

Step 1 Find the highest resource request or limit on init containers. In this example, 1 Gi for memory and 500m for CPU.

Step 2 Calculate the sum of all application container requests or limits. In this example, 2 Gi for memory and 750m for CPU.

Step 3 Compare the values and use the higher one as the effective request or limit: 2 Gi for memory and 750m for CPU.

----End

4.6 Lifecycle Management

Based on Kubernetes, CCI provides containers with **lifecycle hooks**. The hooks enable containers to run code triggered by events during their management 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:

- Post-Start Processing: triggered immediately after the workload is started
- Pre-Stop Processing: triggered immediately before the workload is stopped

When calling an API, you only need to set the **lifecycle.postStart** or **lifecycle.preStop** parameter of the pod, as shown in the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: container-0
    resources:
      limits:
        cpu: 500m
        memory: 1024Mi
      requests:
        cpu: 500m
        memory: 1024Mi
    lifecycle:
      postStart:                 # Post-start processing
        exec:
          command:
          - "/postStart.sh"
      preStop:                   # Pre-stop processing
        exec:
          command:
          - "/preStop.sh"
  imagePullSecrets:
  - name: imagepull-secret
```

4.7 Liveness Probe

Overview

Kubernetes provides the self-healing capability, that is, Kubernetes can detect the container crash and restart the container. However, sometimes memory leakage occurs in a Java program, and the program cannot work normally, while the JVM process is still running. For such issues, Kubernetes provides the liveness probe mechanism to determine whether to restart the container by checking whether the container responses normally. This is a good health check mechanism.

A liveness probe should be defined for each pod. Otherwise, Kubernetes cannot detect whether the pod is running properly.

CCI supports the following detection mechanisms:

- HTTP GET: An HTTP GET request is sent to the container. If the probe receives **2xx** or **3xx**, the container is healthy.

  NOTE

  You need to configure the following annotation for the pod to make **timeoutSeconds** take effect:

  cci.io/httpget-probe-timeout-enable:"true"

  For details, see the example in **Advanced Configuration of Liveness Probe**.

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

HTTP GET

HTTP GET is the most common detection method. The mechanism is to send an HTTP GET request to the container. If the probe receives **2xx** or **3xx**, the container is healthy. The method is defined as follows:

```
apiVersion: v1
kind: Pod
metadata:
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:          # liveness probe
      httpGet:              # HTTP GET definition
        path: /healthz
        port: 8080
```

Create a pod.

```
$ kubectl create -f liveness-http.yaml -n $namespace_name
pod/liveness-http created
```

As shown above, the probe sends an HTTP GET request to port 8080 of the container. The preceding program returns the status code **500** for the fifth request. Then Kubernetes restarts the container.

View pod details.

```
$ kubectl describe po liveness-http -n $namespace_name
Name:         liveness-http
......
Containers:
  container-0:
    ......
    State:          Running
      Started:      Mon, 12 Nov 2018 22:57:28 +0800
    Last State:     Terminated
      Reason:       Error
      Exit Code:    137
      Started:      Mon, 12 Nov 2018 22:55:40 +0800
      Finished:     Mon, 12 Nov 2018 22:57:27 +0800
    Ready:          True
    Restart Count:  1
    Liveness:       http-get http://:8080/ delay=0s timeout=1s period=10s #success=1 #failure=3
......
Events:
  Type    Reason     Age                From               Message
  ----    ------     ----               ----               -------
  Normal  Scheduled  3m5s               default-scheduler  Successfully assigned default/pod-liveness to 
node2
  Normal  Pulling    74s (x2 over 3m4s)  kubelet, node2     pulling image "pod-liveness"
  Normal  Killing    74s                kubelet, node2     Killing container with id docker://container-0:Container 
failed liveness probe.. Container will be killed and recreated.
```

As shown, the pod is in the **Running** state, the **Last State** is **Terminated**, and the **Restart Count** is **1**, indicating that the pod is restarted once. In addition, you can see the following information from the event "Killing container with id docker://container-0:Container failed liveness probe.." **Container will be killed and recreated.**

After the container is killed, a new container is created.

Exec

Exec is to execute a specific command. The mechanism is that the probe executes the command in the container and checks the exit status code of the command. If the status code is 0, the pod is healthy. The method is defined as follows:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:           # liveness probe
      exec:                  # Exec definition
        command:
        - cat
        - /tmp/healthy
```

Run the **cat /tmp/healthy** command in the container. If the command is executed successfully and **0** is returned, the container is healthy.

Advanced Configuration of Liveness Probe

In output of the **$ kubectl describe po liveness-http** command, the following information is displayed:

```
Liveness: http-get http://:8080/ delay=0s timeout=1s period=10s #success=1 #failure=3
```

This line indicates the parameter configuration of the liveness probe. The meanings of the parameters are as follows:

- **delay=0s** indicates that the probe starts immediately after the container is started.
- **timeout=1s** indicates that the container must respond to the probe within 1s. Otherwise, the detection fails.
- **period=10s** indicates that the detection is performed every 10s.
- **#success=1** indicates that the detection is successful after succeeding once.
- **#failure=3** indicates that the container will be restarted after three consecutive detection failures.

These are set by default when the probe is created. You can also manually configure the parameters as follows:

```
apiVersion: v1
kind: Pod
metadata:
  name: liveness-http
spec:
  template:
    metadata:
      annotations:
        cci.io/httpget-probe-timeout-enable:"true"
  containers:
  - image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        path: /
        port: 8080
      initialDelaySeconds: 10    # When does the container start detection after the container is started?
      timeoutSeconds: 2          # The container must respond to the probe within 2s, or the detection fails.
      periodSeconds: 30          # The probe is performed every 30s.
```

```
  successThreshold: 1        # The container is considered healthy as long as the probe succeeds once.
  failureThreshold: 3       # The container will be restarted after three consecutive detection failures.
```

Generally, the value of **initialDelaySeconds** must be greater than 0, because in most cases, although the container is started successfully, it takes a while for the application to be ready. After the application is ready, a success message is returned. Otherwise, the probe may fail frequently.

In addition, you can set **failureThreshold** to allow multiple times of loop detection, so that you do not have to repeatedly run the health check program.

Configuring an Effect Liveness Probe

- **What should a liveness probe detect?**

  A liveness probe should check whether all the key parts of an application are healthy and use a dedicated URL, such as */health*. This function is performed when */health* is accessed, and then the result is returned. Note that authentication cannot be performed. Otherwise, the probe will repeatedly fail and be restarted.

  In addition, the check can be performed only within the application, and cannot be performed outside the dependency. For example, if the frontend web server cannot connect to the database, the web server cannot be considered as unhealthy.

- **A liveness probe must be lightweight.**

  A liveness probe cannot occupy too many resources or too much time. Otherwise, the health check is wasting resources. For example, for Java applications, the HTTP GET method is recommended. If the Exec method is used, the JVM startup occupies too many resources.

5 Label

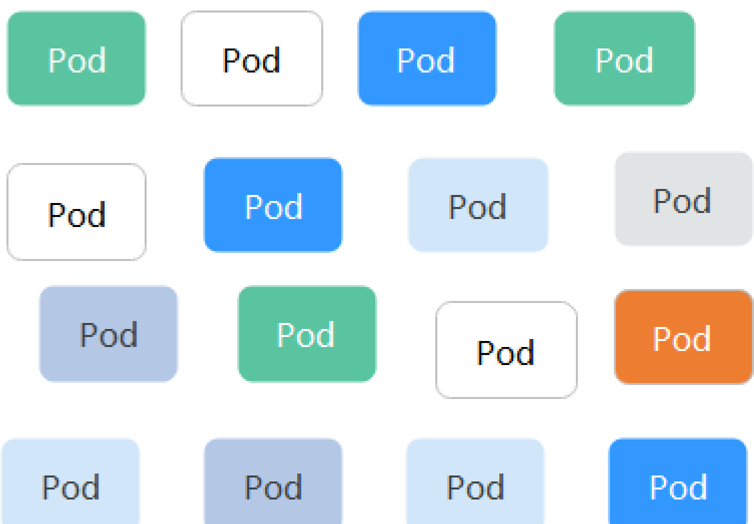
Why Are Labels Required?

As resources increase, how to classify and manage resources becomes important. Kubernetes provides a mechanism to classify resources, that is, using labels. Labels are simple but powerful. **Almost all resources** in the Kubernetes can be organized by labels.

A label is a key-value pair, which can be set when a resource is created, or can be added or modified later.

Taking pods as an example, as the number of pods increases, pods become cluttered and difficult to manage, as shown in the following figure.

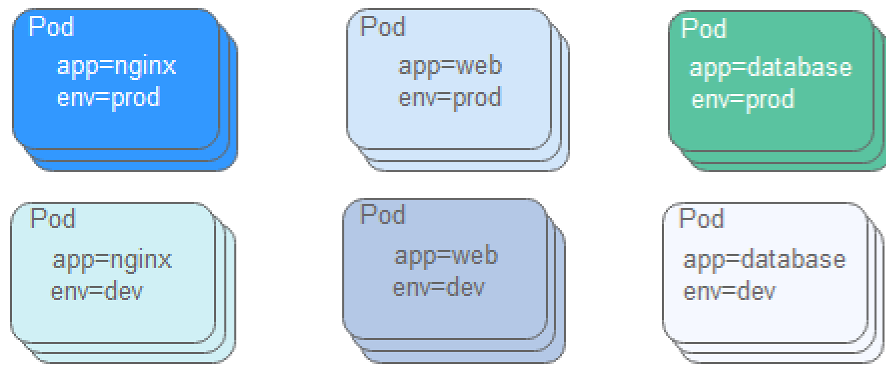
Figure 5-1 Pods without classification



If we attach different labels to the pods, the situation is totally different, as shown in the following figure.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 5-2 Pods organized with labels



Adding a Label

A label is a key-value pair. As shown below, two labels **app=nginx** and **env=prod** are set for a pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:                    # Add the following two labels to the pod.
    app: nginx
    env: prod
spec:
  containers:
  - image: nginx:latest
    name: container-0
    resources:
      limits:
        cpu: 500m
        memory: 1024Mi
      requests:
        cpu: 500m
        memory: 1024Mi
  imagePullSecrets:
  - name: imagepull-secret
```

When a pod has labels, you can view the label of the pod by using **--show-labels** when querying the pod.

```
$ kubectl get pod --show-labels -n $namespace_name
NAME          READY   STATUS    RESTARTS   AGE   LABELS
nginx         1/1     Running   0          50s   app=nginx,env=prod
```

You can also use **-L** to query only certain labels.

```
$ kubectl get pod -L app,env -n $namespace_name
NAME          READY   STATUS    RESTARTS   AGE   APP     ENV
nginx         1/1     Running   0          1m    nginx   prod
```

For an existing pod, you can directly run the **kubectl label** command to add a label.

```
$ kubectl label po nginx creation_method=manual -n $namespace_name
pod "nginx" labeled

$ kubectl get pod --show-labels -n $namespace_name
NAME          READY   STATUS    RESTARTS   AGE   LABELS
nginx         1/1     Running   0          50s   app=nginx,env=prod,creation_method=manual
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Modifying a Label

If you want to modify an existing label, you need to add **--overwrite** to the command, as shown below:

```
$ kubectl label po nginx env=debug --overwrite -n $namespace_name
pod "nginx" labeled

$ kubectl get pod --show-labels -n $namespace_name
NAME              READY   STATUS    RESTARTS   AGE   LABELS
nginx             1/1     Running   0          50s   app=nginx,env=debug,creation_method=manual
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6 Deployment

Pod describes pod, which is the smallest and simplest unit in the Kubernetes object model that you create or deploy. However, a pod is designed to be a one-off entity. A pod can be evicted (when node resources are insufficient) and disappears as the cluster node fails. Kubernetes provides controllers to manage pods. Controllers can create and manage multiple pods, and provide replica management, rolling upgrade, and self-healing capabilities. The most commonly used is Deployment.

A Deployment can contain one or more pod replicas. Each pod replica has the same role. Therefore, the system automatically distributes requests to multiple pod replicas of a Deployment.

A Deployment integrates a lot of functions, including online deployment, rolling upgrade, replica creation, and restoration of online jobs. To some extent, Deployments can be used to realize unattended rollout, which greatly reduces communication difficulties and operation risks in the rollout process.

Creating a Deployment

In the following example, a Deployment named **nginx** is created, and two pod replicas are created from the **nginx:latest** image. Each pod replica occupies 500m and 1024Mi memory.

```
apiVersion: apps/v1      # Note the difference from that of pods. It is apps/v1 instead of v1 for a Deployment.
kind: Deployment         # The resource type is Deployment.
metadata:
  name: nginx            # Name of the Deployment
spec:
  replicas: 2            # Number of pod replicas. The Deployment ensures that two pod replicas are running.
  selector:              # Label Selector
    matchLabels:
      app: nginx
  template:              # Definition of a pod, which is used to create pods. It is also known as pod template.
    metadata:
      labels:
        app: nginx
    spec:
      volumes:
        - name: cci-sfs-test                    # SFS volume name
          persistentVolumeClaim:
            claimName: cci-sfs-test
      containers:
      - image: nginx:latest
```

```
        name: container-0
        resources:
          limits:
            cpu: 500m
            memory: 1024Mi
          requests:
            cpu: 500m
            memory: 1024Mi
        volumeMounts:
          - name: cci-sfs-test
            mountPath: /tmp/sfs0/krlp2k8j       # SFS volume's mount path in a container
      imagePullSecrets:          # Secret used to pull the image, which must be imagepull-secret.
      - name: imagepull-secret
```

In this definition, the name of the Deployment is **nginx**, and **spec.replicas** defines the number of pods. That is, the Deployment controls two pods. **spec.selector** is a label selector, indicating that the Deployment selects the pod whose label is **app=nginx**. **spec.template** is the definition of the pod and is the same as that defined in **Pod**.

Save the definition of the Deployment to **deployment.yaml** and use kubectl to create the Deployment.

Run the **kubectl get** command to view the Deployment and the pods. The value of **DESIRED** is **2**, indicating that the Deployment desires two pods. The value of **CURRENT** is **2**, indicating that there are two pods. The value of **AVAILABLE** is **2**, indicating that two pods are available.

```
$ kubectl create -f deployment.yaml -n $namespace_name

$ kubectl get deployment -n $namespace_name
NAME     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx    2         2         2            2           8s
```

How Does the Deployment Control Pods?

Continue to query pods, as shown below.

```
$ kubectl get pods -n $namespace_name
NAME                     READY     STATUS    RESTARTS   AGE
nginx-7f98958cdf-tdmqk   1/1       Running   0          13s
nginx-7f98958cdf-txckx   1/1       Running   0          13s
```

If you delete a pod, a new pod is immediately created, as shown below. As mentioned above, the Deployment ensures that there are two pods running. If a pod is deleted, the Deployment creates another pod. If a pod crashes or is faulty, the Deployment automatically restarts the pod.

```
$ kubectl delete pod nginx-7f98958cdf-txckx -n $namespace_name

$ kubectl get pods -n $namespace_name
NAME                     READY     STATUS    RESTARTS   AGE
nginx-7f98958cdf-tdmqk   1/1       Running   0          21s
nginx-7f98958cdf-tesqr   1/1       Running   0          21s
```

There are two pods, **nginx-7f98958cdf-tdmqk** and **nginx-7f98958cdf-tesqr**, in which **nginx** is the name of the Deployment, and **-7f98958cdf-tdmqk** and **-7f98958cdf-tesqr** are the suffixes randomly generated by Kubernetes.

You may notice that the first part of the two suffixes is the same, that is, **7f98958cdf**. This is because the Deployment does not control the pod directly, but through a controller named ReplicaSet. You can run the following command to query the ReplicaSet, in which **rs** is the abbreviation of ReplicaSet.

```
$ kubectl get rs -n $namespace_name
NAME              DESIRED   CURRENT   READY     AGE
nginx-7f98958cdf   3         3         3         1m
```

The name of the ReplicaSet is **nginx-7f98958cdf**, and the suffix **-7f98958cdf** is generated randomly.

Figure 6-1 shows how the Deployment controls the pod via the ReplicaSet.

Figure 6-1 Control flow

If you run the **kubectl describe** command to view the details of the Deployment, you can see the ReplicaSet. As shown below, you can see a line **NewReplicaSet: nginx-7f98958cdf (2/2 replicas created)**. In events, the number of pods of the ReplicaSet is scaled out to 2. In practice, you may not operate ReplicaSet directly, but understanding that a Deployment controls a pod by controlling a ReplicaSet helps you locate problems.

```
$ kubectl describe deploy nginx -n $namespace_name
Name:                   nginx
Namespace:              default
CreationTimestamp:      Sun, 16 Dec 2018 19:21:58 +0800
Labels:                 app=nginx

...

NewReplicaSet:   nginx-7f98958cdf (2/2 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  5m    deployment-controller  Scaled up replica set nginx-7f98958cdf to 2
```

Upgrade

In actual applications, upgrade is a common operation. A Deployment can easily support application upgrade.

You can set different upgrade policies for a Deployment:

- RollingUpdate: Gradually create new pods and delete old pods). This is the default policy.
- Recreate: Delete the current pods and then create new pods.

The Deployment upgrade can be in declarative mode. That is, you only need to modify the YAML definition of the Deployment. For example, you can run the **kubectl edit** command to change the Deployment image to **nginx:alpine**. After the modification, query the ReplicaSet and pod, a new ReplicaSet is created, and the pod is recreated.

```
$ kubectl edit deploy nginx -n $namespace_name

$ kubectl get rs -n $namespace_name
NAME               DESIRED   CURRENT   READY     AGE
nginx-6f9f58dffd   2         2         2         1m
nginx-7f98958cdf   0         0         0         48m

$ kubectl get pods -n $namespace_name
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
NAME                    READY     STATUS    RESTARTS   AGE
nginx-6f9f58dffd-tdmqk   1/1       Running   0          21s
nginx-6f9f58dffd-tesqr   1/1       Running   0          21s
```

The Deployment can use the **maxSurge** and **maxUnavailable** parameters to control the proportion of pods to be recreated during the upgrade. This is useful in many scenarios. The configuration is as follows:

```
spec:
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
```

- **maxSurge**: specifies the percent of pods that can exist over **spec.replicas** in the Deployment. The default value is **25%**. For example, if **spec.replicas** is set to **4**, no more than five pods can exist during the upgrade, and the upgrade step is 1. The absolute number is calculated from the percentage by rounding up. The value can also be set to an absolute number.
- **maxUnavailable**: specifies the percent of pods that can be unavailable during the update. The default value is **25%**. For example, if **spec.replicas** is set to **4**, there must be at least three pods during the upgrade, so the deleting step is 1. The value can also be set to an absolute number.

In the preceding example, the value of **spec.replicas** is **2**. Suppose both **maxSurge** and **maxUnavailable** are the default value 25%, **maxSurge** allows a maximum of three pods to exist (2 * 1.25 = 2.5, rounded up to 3), and **maxUnavailable** does not allow unavailable pods (2 * 0.75 = 1.5, rounded up to 2). As a result, two pods are running during the upgrade. Each time a new pod is created, an old pod is deleted, until all pods are new.

Rollback

Rollback is to roll an application back to the earlier version when a fault occurs during the upgrade. A Deployment can be easily rolled back to the earlier version.

For example, if the upgraded image is faulty, you can run the **kubectl rollout undo** command to roll back.

```
$ kubectl rollout undo deployment nginx -n $namespace_name
deployment "nginx" rolled back
```

A Deployment can be easily rolled back because a Deployment uses a ReplicaSet to control a pod. After the upgrade, the ReplicaSet still exists. The Deployment is rolled back by recreating the pod using the ReplicaSet. The number of ReplicaSets stored in a Deployment can be restricted by the **revisionHistoryLimit** parameter. The default value is **10**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7 EIPPool

7.1 Overview

CCI introduces a user-defined resource object named EIPPool to automatically bind EIPs to pods. EIPPools support two EIP resource management modes: dynamic mode (EIP resources are automatically created by CCI) and static mode (EIP resources are created by users in advance).

Constraints

- Only one EIP can be bound to a pod.
- To enable a pod with an EIP bound to be accessed from the public network, you need to add security group rules to allow access from the public network.
- An EIPPool being used by a pod cannot be deleted. You need to delete the associated pod and then delete the EIPPool.
- EIPPools are namespace-level resources and cannot be used across namespaces.
- In the rolling upgrade of a workload, by default, new pods are created before old pods are deleted (for details, see **the upgrade policy**). As a result, the upgrade may fail due to insufficient EIPs in the EIPPool. To avoid this, you can set the number of EIPs in the EIPPool to be slightly greater than the total number of Deployment replicas that use the EIPPool, or set **maxSurge** to **0**.

Related Operations

You can perform the following operations on EIPPools:

Creating an EIPPool

Using an EIPPool

Managing an EIPPool

7.2 Creating an EIPPool

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7.2.1 Creating a Dynamic EIPPool

A dynamic EIPPool dynamically creates underlying EIP resources based on your configuration and creates EIP objects in the CCI namespace.

The following example shows how to create a dynamic EIPPool named **eippool-demo1**. For details about the fields, see **Table 7-1**.

- To dynamically create an EIPPool that uses a dedicated bandwidth, you do not need to specify the bandwidth ID. Example:

```
apiVersion: crd.yangtse.cni/v1
kind: EIPPool
metadata:
  name: eippool-demo1
  namespace: xxx                    # Namespace where the EIPPool is located, which must be the same as that 
of the pod.
spec:
  amount: 3                         # Number of EIPs in the EIPPool
  eipAttributes:
    networkType: 5_bgp
    ipVersion: 4
    bandwidth:
      name: cci-eippool-demo1
      chargeMode: bandwidth
      shareType: PER
      size: 5
```

- To dynamically create an EIPPool that uses a shared bandwidth, you must and only need to specify the bandwidth ID. Example:

```
apiVersion: crd.yangtse.cni/v1
kind: EIPPool
metadata:
  name: eippool-demo1
  namespace: xxx
spec:
  amount: 3
  eipAttributes:
    networkType: 5_bgp
    ipVersion: 4
    bandwidth:
      id: xxx
      shareType: WHOLE              # If the shareType is WHOLE, the bandwidth ID must be specified.
```

Table 7-1 Parameter description

Parameter	Description	Constraint
name	Name of the EIPPool	The name should be less than or equal to 29 bytes. If it exceeds 29 bytes, extra characters will be truncated, but the use of the EIPPool is not affected.
namespace	Namespace to which the EIPPool belongs	The value must be the same as the namespace of the pod.
amount	Number of EIPs in the EIPPool	The value ranges from 0 to 500.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Constraint
networkType	EIP type	The value can be **5_telcom** (China Telecom), **5_union** (China Unicom), **5_bgp** (dynamic BGP), **5_sbgp** (static BGP), or **5_ipv6**. The configured value must be supported by the system.
ipVersion	EIP version	The value can be **4** (IPv4 address) or **6** (IPv6 address). The configured value must be supported by the system. If this parameter is left blank or is an empty string, an IPv4 address is assigned by default.
chargeMode	Whether the billing is based on traffic or bandwidth	The value can be **bandwidth** or **traffic**. The value **bandwidth** indicates that the bandwidth is billed by fixed bandwidth. The value **traffic** indicates that the bandwidth is billed by traffic. If this parameter is left blank, the default value is **bandwidth**. For IPv6 addresses, the default value is **bandwidth** outside China and is **traffic** in China.
shareType	Bandwidth type	The value can be **PER** (dedicated bandwidth) or **WHOLE** (shared bandwidth). If this parameter is set to **WHOLE**, the bandwidth ID must be specified.
id	Bandwidth ID	The value can be the ID of the bandwidth whose type is set to **WHOLE**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Constraint
size	Bandwidth	The value ranges from 1 to 200. Unit: Mbit/s The specific range may vary with the configuration in each region. For the specific range, see the console.

For the function description, value range, and constraints of the above EIP-related fields in the YAML file, see the **EIP parameters** page.

Run the following command to view the EIPPool details. (**-n** indicates the namespace to which the EIPPool belongs.)

If the EIPPool name **eippool-demo1** is displayed in the command output, the dynamic EIPPool is created successfully.

```
# kubectl get eippool -n $namespace_name
NAME              EIPS         USAGE        AGE
eippool-demo1                   0/3          39m
```

7.2.2 Creating a Static EIPPool

A static EIPPool statically manages underlying EIP resources based on multiple unused EIPs that you specify and creates EIP objects in the CCI namespace. If EIPs in the EIPPool are used by NAT or ELB, the management fails.

The following example shows how to create a static EIPPool named **eippool-demo2** and manage public IP addresses 10.246.173.254 and 10.246.172.3 in it.

Example:

```
apiVersion: crd.yangtse.cni/v1
kind: EIPPool                   # Type of the created object
metadata:                       # Metadata definition of the resource object
  name:eippool-demo2
spec:                         # EIPPool configuration
  eips:                       # Public IP addresses to be managed
  - 10.246.173.254
  - 10.246.172.3
```

7.3 Using an EIPPool

After creating an EIPPool in the namespace, add the specified annotation **yangtse.io/eip-pool** to the pod template to use the EIPs of the EIPPool. When a pod is created, an available EIP is automatically obtained from the EIPPool and bound to the pod.

NOTE

EIPs that are being used by an EIPPool cannot be bound, unbound, or deleted on the VPC console. Therefore, you are not advised to perform operations on such EIPs on the VPC console.

The following uses **eippool-demo1** as an example.

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
   yangtse.io/eip-pool:eippool-demo1    # Use EIPs by specifying an EIPPool.
...
```

Run the following command to view the EIPPool details. (**-n** indicates the namespace to which the EIPPool belongs.)

In the command output, the usage of the **eippool-demo1** EIPPool increases by 1, indicating that an EIP is successfully bound to the pod.

```
# kubectl get eippool -n $namespace_name
NAME                 EIPS            USAGE   AGE
eippool-demo1                         1/3     64m
```

After the pod is started, you can add the annotation **yangtse.io/allocated-ipv4-eip** to query the EIP used by the pod.

```
apiVersion:v1
kind:Pod
metadata:
  annotations:
    yangtse.io/allocated-ipv4-eip: 116.205.XXX.XXX      # EIP allocated to the pod
```

NOTE

If a pod is rebuilt, it will re-obtain an available EIP from the EIPPool.

7.4 Managing an EIPPool

Updating an EIPPool

Currently, you can only adjust the number of EIPs in an EIPPool, that is, scale in or out an EIPPool. If you want to modify other EIP parameters, create an EIPPool to replace the existing one in the workload configuration.

During EIPPool scale-in, if resources of an EIP are occupied, the EIP will not be deleted until the resources are released.

In the following example of the dynamic EIPPool **eippool-demo1**, change **amount: 3** to **amount: 5**.

```
apiVersion: crd.yangtse.cni/v1
kind: EIPPool
metadata:
  name: eippool-demo1
  namespace: xxx
spec:
  amount: 5                       # Number of EIPs in the EIPPool
  eipAttributes:                  # EIP attributes
 ...
```

Run the following command to view the EIPPool details. (**-n** indicates the namespace to which the EIPPool belongs.)

In the command output, if the **USAGE** value of **eippool-demo1** changes from **0/3** to **0/5**, the EIPPool is successfully updated.

```
# kubectl get eippool -n $namespace_name
NAME              EIPS        USAGE       AGE
eippool-demo1                  0/5         39m
```

As shown in the following example of the static EIPPool **eippool-demo2**, updating an EIPPool is to add or delete public IP addresses managed by the EIPPool.

```
apiVersion: crd.yangtse.cni/v1
kind: EIPPool                 # Type of the created object
metadata:                     # Metadata definition of the resource object
  name:eippool-demo2
spec:                         # EIPPool configuration
  eips:                       # Public IP addresses to be managed
  - 10.246.173.254
  - 10.246.172.3
  - 10.246.172.59
```

Deleting an EIPPool

When an EIPPool is deleted, its EIPs are also deleted. If an EIP in the EIPPool is occupied by a pod or other resources, you cannot delete the EIPPool.

8 EIP

8.1 Overview

To make it easier for you to bind an EIP to a pod in CCI, you only need to configure the annotation when creating the pod. The EIP will be automatically bound to the pod.

Constraints

- A pod can only have one EIP bound, and an EIP can only be bound to one pod.
- You can set **annotations** when creating a pod. After the pod is created, the annotations related to the EIP cannot be updated.
- The EIP allocated during pod creation takes precedence over the EIP created by an EIPPool.
- When you create a pod, binding an existing EIP has priority over binding a new EIP.
- To enable a pod with an EIP bound to be accessed from the public network, you need to add security group rules to allow access from the public network.
- For an EIP that has been bound to a pod, do not perform operations such as changing the alias, deleting, unbinding, and binding the EIP, and changing the billing mode on the EIP console or through APIs. Otherwise, some resources may not be deleted.
- If an existing EIP is to be bound to the pod and the pod is deleted and then rebuilt, the time for the new pod to be ready is prolonged because the EIP needs to be unbound from the old pod.
- An existing EIP must be one that is manually allocated. An EIP created using an EIPPool cannot be used, or the EIP status will be abnormal.

Related Operations

You can perform the following operations on EIPs:

Binding a New EIP to a Pod

Binding an Existing EIP to a Pod

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

8.2 Binding a New EIP to a Pod

Allocating an EIP During Pod Creation

When creating a pod, configure **pod-with-eip** under **annotations**. An EIP is automatically assigned and bound to the pod.

The following uses a Deployment named **nginx** as an example. For details about the parameters, see **Table 8-1**.

- To create a Deployment that uses a dedicated bandwidth, you do not need to specify the bandwidth ID.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "14"
    description: ""
  name: nginx
  namespace: eip
spec:
  ...
  template:
    metadata:
      annotations:
        yangtse.io/pod-with-eip: "true"
        yangtse.io/eip-bandwidth-size: "5"
        yangtse.io/eip-network-type: 5_g-vm
        yangtse.io/eip-charge-mode: bandwidth
        yangtse.io/eip-bandwidth-name: "xxx"
```

- To create a Deployment that uses a shared bandwidth, you must and only need to specify the bandwidth ID.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "14"
    description: ""
  name: nginx
  namespace: eip
spec:
  ...
  template:
    metadata:
      annotations:
        yangtse.io/pod-with-eip: "true"
        yangtse.io/eip-bandwidth-id: "xxx"
```

Table 8-1 Parameter description

Parameter	Description	Mandatory	Constraint
yangtse.io/pod-with-eip	The EIP allocated during pod creation	Yes	This option is enabled only when the parameter value is set to **true**.

Parameter	Description	Mandatory	Constraint
yangtse.io/eip-bandwidth-size	The bandwidth that will be used by the EIP	No	The default value is **5**.
yangtse.io/eip-network-type	The bandwidth type	No	The default value is **5_bgp**. If your region does not support **5_bgp** and this parameter is left empty, an event is reported for the pod.
yangtse.io/eip-charge-mode	Bandwidth billing option	No	The value can be **bandwidth** (default) or **traffic**.
yangtse.io/eip-bandwidth-id	ID of the shared bandwidth	Yes (for shared bandwidths)	If you have set this parameter, you do not need to set other values.
yangtse.io/eip-bandwidth-name	Bandwidth name	No	The default value is the same as the EIP name.

Verifying the EIP Allocation

The startup time of the pod may be earlier than the time when the EIP allocation result is returned. During pod startup, the EIP may fail to be bound.

You can use an init container to check whether the EIP is allocated. After the pod IP address is allocated, the container network controller binds an EIP to the pod and returns the allocation result to the annotation (yangtse.io/allocated-ipv4-eip) of the pod. You can configure an init container in the pod and use the downward API to mount the annotation to the init container through a volume to check whether the EIP is allocated. You can configure the init container as follows:

NOTICE

CCI allows EIPs to be automatically bound to pods. EIPs are allocated after pod scheduling is complete. EIP annotations cannot be injected to pods through **ENV**.

```
apiVersion: v1
kind: Pod
metadata:
  name: example
  namespace: demo
  annotations:
    yangtse.io/pod-with-eip: "true"
    yangtse.io/eip-bandwidth-size: "5"
    yangtse.io/eip-network-type: 5_g-vm
    yangtse.io/eip-charge-mode: bandwidth
    yangtse.io/eip-bandwidth-name: "xxx"
```

```
spec:
  initContainers:
  - name: init
    image: busybox:latest
    command: ['timeout', '60', 'sh', '-c', "until grep -E '[0-9]+' /etc/eipinfo/allocated-ipv4-eip; do echo waiting 
for allocated-ipv4-eip; sleep 2; done"]
    volumeMounts:
        - name: eipinfo
          mountPath: /etc/eipinfo
  volumes:
    - name: eipinfo
      downwardAPI:
        items:
          - path: "allocated-ipv4-eip"
            fieldRef:
              fieldPath: metadata.annotations['yangtse.io/allocated-ipv4-eip']
```

Releasing an EIP During Pod Deletion

When you delete a pod, the EIP bound to it is also released. You can run the following command to delete a pod:

```
# kubectl delete pod nginx -n $namespace_name
```

8.3 Binding an Existing EIP to a Pod

Specifying the EIP ID for a Pod

When creating a pod, enter the **yangtse.io/eip-id** annotation. An EIP is automatically assigned and bound to the pod.

The following example creates a Deployment named **nginx**. This Deployment has one pod, and an EIP is automatically assigned and bound to the pod. **Table 8-2** describes the parameters.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 annotations:
   deployment.kubernetes.io/revision: "14"
   description: ""
 name: nginx
 namespace: eip
spec:
 ...
 replicas: 1
  template:
   metadata:
    annotations:
     yangtse.io/eip-id: 65eb3679-7a8d-4b24-b681-0b661axxxxcb
```

Table 8-2 Parameter description

Parameter	Description	Mandatory	Constraint
yangtse.io/eip-id	EIP ID	Yes	The ID that can be queried on the EIP page.

Verifying the EIP Allocation

The startup time of the pod may be earlier than the time when the EIP allocation result is returned. During pod startup, the EIP may fail to be bound.

You can use an init container to check whether the EIP is allocated. After the pod IP address is allocated, the container network controller binds an EIP to the pod and returns the allocation result to the annotation (yangtse.io/allocated-ipv4-eip) of the pod. You can configure an init container in the pod and use the downward API to mount the annotation to the init container through a volume to check whether the EIP is allocated. You can configure the init container as follows:

```
apiVersion: v1
kind: Pod
metadata:
  name: example
  namespace: demo
  annotations:
    yangtse.io/eip-id: 65eb3679-7a8d-4b24-b681-0b661axxxxcb
spec:
  initContainers:
  - name: init
    image: busybox:latest
    command: ['timeout', '60', 'sh', '-c', "until grep -E '[0-9]+' /etc/eipinfo/allocated-ipv4-eip; do echo waiting 
for allocated-ipv4-eip; sleep 2; done"]
    volumeMounts:
        - name: eipinfo
          mountPath: /etc/eipinfo
  volumes:
    - name: eipinfo
      downwardAPI:
        items:
          - path: "allocated-ipv4-eip"
            fieldRef:
              fieldPath: metadata.annotations['yangtse.io/allocated-ipv4-eip']
```

9 Pod Resource Monitoring Metric

CCI supports basic monitoring of pod resources with multiple metrics, such as metrics for CPU, memory, disk, and network.

Pods have built-in system agents, which provide pod and container monitoring metrics in HTTP services by default. Reserve 30 MB for the agent that integrated into a pod.

Resource Metrics

Basic monitoring metrics include CPU, memory, and disk usage. For details, see **Resource Metrics**.

Table 9-1 Resource metrics

Category	Metric	Description
CPU	container_cpu_system_seconds_total	Cumulative system CPU time consumed (unit: second)
	container_cpu_usage_seconds_total	Cumulative time that the container consumed on all CPU cores (unit: second)
	container_cpu_user_seconds_total	Cumulative user CPU time consumed (unit: second)
	container_cpu_cfs_periods_total	Number of elapsed enforcement period intervals
	container_cpu_cfs_throttled_periods_total	Number of throttled period intervals
	container_cpu_cfs_throttled_seconds_total	Total time duration the container has been throttled (unit: second)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Category	Metric	Description
File system and disk I/O	container_fs_inodes_free	Number of available inodes in the file system
	container_fs_usage_bytes	File system usage (unit: byte)
	container_fs_inodes_total	Total number of inodes in the file system
	container_fs_io_current	Number of I/Os currently in progress in the disk or file system
	container_fs_io_time_seconds_total	Cumulative seconds spent on doing I/Os by the disk or file system
	container_fs_io_time_weighted_seconds_total	Cumulative weighted I/O time of the disk or file system
	container_fs_limit_bytes	Total disk or file system capacity that can be consumed by the container (unit: byte)
	container_fs_reads_bytes_total	Cumulative amount of disk or file system data read by the container (unit: byte)
	container_fs_read_seconds_total	Cumulative count of seconds the container spent on reading disk or file system data
	container_fs_reads_merged_total	Cumulative count of merged disk or file system reads made by the container
	container_fs_reads_total	Cumulative count of disk or file system reads completed by the container
	container_fs_sector_reads_total	Cumulative count of sector reads completed by the container in the disk or file system
	container_fs_sector_writes_total	Cumulative count of sector writes completed by the container to the disk or file system
	container_fs_writes_bytes_total	Total amount of data written by the container to the disk or file system (unit: byte)
	container_fs_write_seconds_total	Cumulative count of seconds the container spent on writing data to the disk or file system

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Category	Metric	Description
	container_fs_writes_merged_total	Cumulative count of merged container writes to the disk or file system
	container_fs_writes_total	Cumulative count of completed container writes to the disk or file system
	container_blkio_device_usage_total	Blkio device usage (unit: byte)
Memory	container_memory_failures_total	Cumulative count of container memory allocation failures
	container_memory_failcnt	Number of memory usage hits limits
	container_memory_cache	Total page cache memory of the container (unit: byte)
	container_memory_mapped_file	Size of memory mapped files (unit: byte)
	container_memory_max_usage_bytes	Maximum memory usage recorded for the container (unit: byte)
	container_memory_rss	Size of the resident memory set for the container (unit: byte)
	container_memory_swap	Container swap usage (unit: byte)
	container_memory_usage_bytes	Current memory usage of the container (unit: byte)
	container_memory_working_set_bytes	Memory usage of the working set of the container (unit: byte)
Network	container_network_receive_bytes_total	Total volume of data received by the container network (unit: byte)
	container_network_receive_errors_total	Cumulative count of errors encountered during reception
	container_network_receive_packets_dropped_total	Cumulative count of packets dropped during reception
	container_network_receive_packets_total	Cumulative count of packets received
	container_network_transmit_bytes_total	Total volume of data transmitted on the container network (unit: byte)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Category	Metric	Description
	container_network_transmit_errors_total	Cumulative count of errors encountered during transmission
	container_network_transmit_packets_dropped_total	Cumulative count of packets dropped during transmission
	container_network_transmit_packets_total	Cumulative count of packets transmitted
Process	container_processes	Number of processes running inside the container
	container_sockets	Number of open sockets for the container
	container_file_descriptors	Number of open file descriptors for the container
	container_threads	Number of threads running inside the container
	container_threads_max	Maximum number of threads allowed inside the container
	container_ulimits_soft	Soft ulimit value of process 1 in the container. Unlimited if the value is -1, except priority and nice.
	container_spec_cpu_period	CPU period of the container
	container_spec_cpu_shares	CPU share of the container
	container_spec_memory_limit_bytes	Memory limit for the container
	container_spec_memory_reservation_limit_bytes	Memory reservation limit for the container
	container_spec_memory_swap_limit_bytes	Memory swap limit for the container
	container_start_time_seconds	Running time of the container (unit: second)
	container_last_seen	Last time a container was seen by the exporter
gpu	container_accelerator_memory_used_bytes	GPU accelerator memory that is being used by the container (unit: byte)
	container_accelerator_memory_total_bytes	Total available GPU accelerator memory (unit: byte)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Category	Metric	Description
	container_accelerator_duty_cycle	Percentage of time when the GPU accelerator is actually running

The total number of monitoring metrics is 59, which is the same as that provided by cAdvisor.

For details about the metrics, see the **cAdvisor document**.

Basic Configuration

The following example describes how to configure pod resource monitoring metrics, including enabling or disabling pod-level features and customizing ports.

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx-exporter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-exporter
  template:
    metadata:
      labels:
        app: nginx-exporter
      annotations:
        monitoring.cci.io/enable-pod-metrics: "true"
        monitoring.cci.io/metrics-port: "19100"
    spec:
      containers:
        - name: container-0
          image: 'nginx:alpine'
          resources:
            limits:
              cpu: 1000m
              memory: 2048Mi
            requests:
              cpu: 1000m
              memory: 2048Mi
      imagePullSecrets:
        - name: imagepull-secret
```

Table 9-2 Parameter description

Annotation	Function	Available Value	Default Value
monitoring.cci.io/enable-pod-metrics	Whether to enable the monitoring metrics	true, false (case insensitive)	true
monitoring.cci.io/metrics-port	Listening port of the pod exporter	Valid ports (1 to 65535)	19100

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Advanced Configuration

Creating a Secret

A secret is a resource object for encrypted storage. You can save the authentication information, certificates, and private keys in a secret for configuring sensitive data such as passwords, tokens, and keys.

The secret defined in the following example contains three key-value pairs.

```
apiVersion: v1
kind: Secret
metadata:
  name: cert
type: Opaque
data:
   ca.crt: ...
   server.crt: ...
   server.key: ...
```

Configuring a TLS Certificate

You can configure annotations to specify the TLS certificate suite of the exporter server for encrypted communication and use the file mounting mode to associate the certificate secret. Example:

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx-tls
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-tls
  template:
    metadata:
      labels:
        app: nginx-tls
      annotations:
        monitoring.cci.io/enable-pod-metrics: "true"
        monitoring.cci.io/metrics-port: "19100"
        monitoring.cci.io/metrics-tls-cert-reference: cert/server.crt
        monitoring.cci.io/metrics-tls-key-reference: cert/server.key
        monitoring.cci.io/metrics-tls-ca-reference: cert/ca.crt
        sandbox-volume.openvessel.io/volume-names: cert
    spec:
      volumes:
        - name: cert
          secret:
            secretName: cert
            defaultMode: 384
      containers:
        - name: container-0
          image: 'nginx:alpine'
          resources:
            limits:
              cpu: 1000m
              memory: 2048Mi
            requests:
              cpu: 1000m
              memory: 2048Mi
          volumeMounts:
            - name: cert
              mountPath: /tmp/secret0
      imagePullSecrets:
        - name: imagepull-secret
```

Table 9-3 TLS certificate parameters

Annotation	Function	Available Value	Default Value
monitoring.cci.io/metrics-tls-cert-reference	TLS certificate volume reference	${volume-name}/${volume-keyOrPath} (Volume/Path)	None (HTTP is used.)
monitoring.cci.io/metrics-tls-key-reference	TLS private key volume reference	${volume-name}/${volume-keyOrPath}	None (HTTP is used.)
monitoring.cci.io/metrics-tls-ca-reference	TLS CA volume reference	${volume-name}/${volume-keyOrPath}	None (HTTP is used.)

The values of the preceding parameters are the names and paths of the storage volume where the TLS certificate, private key, and CA file are located.

Obtaining Resource Monitoring Metrics

After configuring the preceding monitoring attributes, run the following command in a VPC that can access the pod to obtain the pod monitoring data:

```
curl $podIP:$port/metrics
```

<*podIP*> indicates the IP address of the pod, and <*port*> indicates the listening port, for example, **curl 192.168.XXX.XXX:19100/metrics**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

10 Collecting Pod Logs

This section describes how to collect logs in a pod. You can configure the log files in a user-defined path in a container to collect logs, process the logs based on user-defined policies, and report the logs to the Kafka log center.

Resource Restriction

You are advised to reserve 50 MiB memory for Fluent Bit.

Constraints

- Logs of soft link paths in containers cannot be collected.
- Container stdout logs cannot be collected and reported to Kafka.
- Log rotation is not supported. You need to control the log file size by yourself.
- A log larger than 250 KB cannot be collected.
- Logs cannot be collected from the directory that a specified system, device, cgroup, tmpfs, or localdir is mounted to.
- In a container, if a log name exceeds 190 characters, the log will not be collected. If there are logs whose name contains 180 to 190 characters, only the first log can be collected.
- When a container is stopped, if log collection is delayed due to network latency or high resource usage, some logs generated before the container is stopped may be lost.

Basic Configuration

Fluent Bit is an open source multi-platform log processor. It consists of modules SERVICE, INPUT, FILTER, PARSER, and OUTPUT. Currently, you can only define the destination of the log contents in the OUTPUT module.

You can use the following ConfigMap to send Fluent Bit process logs to Kafka.

Constraints

- The size of **output.conf** must be less than 1 MB.
- **[OUTPUT]** is the outermost parameter and must not be indented. Configuration items below it are indented by four spaces.

Basic Configuration

Configure the following parameters in your main configuration file:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: cci-logging-conf
  labels:
    logconf.k8s.io/discovery: "true"
data:
    output.conf: |
        [OUTPUT]
            Name       kafka
            Match      *
            Brokers    192.168.1.3:9092
            Topics     test
```

Table 10-1 Parameter description

Parameter	Description	Mandatory	Constraint
logconf.k8s.io/discovery	Labels the ConfigMap as a Fluent Bit log configuration file.	Yes	Mandatory value: **true**
Name	Add-on name.	Yes	Mandatory value: **kafka** Currently, only the Kafka add-on is supported.
Match	Matches the label of the transferred record. The asterisk (*) is used as a wildcard.	No	If this parameter is set, the value must be *.
Brokers	Broker (Kafka) address. You can configure multiple broker addresses at the same time.	Yes	Example: **192.168.1.3:9092, 192.168.1.4:9092, 192.168.1.5:9092**
Topics	Log topic.	No	The default value is **fluent-bit**. The transferred topic must exist.

You can configure the volume on the pod and configure annotations to specify the sandbox volume and the corresponding log output configuration file.

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: kafka-dey
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kafka
  template:
    metadata:
      labels:
        app: kafka
      annotations:
        logpath.k8s.io/container-0: /var/log/*.log;/var/paas/sys/log/virtual-kubelet.log
        logconf.k8s.io/fluent-bit-configmap-reference: cci-logging-conf
    spec:
      containers:
        - name: container-0
          image: 'nginx:alpine'
          resources:
            limits:
              cpu: 1000m
              memory: 2048Mi
            requests:
              cpu: 1000m
              memory: 2048Mi
      imagePullSecrets:
        - name: default-secret
```

Table 10-2 Parameter description

Annotation	Function	Constraint
logpath.k8s.io/ $containerName	Configures the collection file using the environment variables of the pod container. *$containerName* is a container name variable.	Multiple paths can be configured. Each path must be an absolute path starting with a slash (/) and paths are separated by semicolons (;). Only complete log file paths or file names with the wildcard (*) are supported. If a file name contains the wildcard (*), the directory where the file is located must exist when the container is started. The maximum length of a file name is 190 characters.
logconf.k8s.io/fluent-bit-configmap-reference	Specifies the ConfigMap configured for the Fluent Bit log collection.	The ConfigMap must exist and meet the requirements of configuring Fluent Bit.

Advanced Configuration

A secret is a resource object for encrypted storage. You can save the authentication information, certificates, and private keys in a secret for configuring sensitive data such as passwords, tokens, and keys.

```
apiVersion: v1
kind: Secret
metadata:
  name: cci-sfs-kafka-tls
type: Opaque
data:
   ca.crt: ...
   server.crt: ...
   server.key: ...
```

You can configure SSL parameters to implement encrypted secure connections. Files such as certificates can be referenced by the sandbox volume feature.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: cci-logging-conf-tls
  labels:
     logconf.k8s.io/discovery: true
data:
    output.conf: |
        [OUTPUT]
            Name         kafka
            Match        *
            Brokers      192.168.1.3:9092
            Topics       test
            rdkafka.security.protocol ssl
            rdkafka.ssl.certificate.location ${sandbox_volume_kafkatls}/client.crt
            rdkafka.ssl.key.location ${sandbox_volume_kafkatls}/client.key
            rdkafka.ssl.ca.location ${sandbox_volume_kafkatls}/ca.crt
            rdkafka.enable.ssl.certificate.verification true
            rdkafka.request.required.acks 1
```

Table 10-3 Parameter description

Parameter	Description	Mandatory	Available Value
rdkafka.security.protocol	Protocol used to communicate with the agent	Mandatory if SSL authentication is enabled	ssl
rdkafka.ssl.certificate.location	Path for storing SSL public keys	Mandatory if SSL authentication is enabled	${sandbox_volume_**${VOLUME_NAME}**}/**some.cert**
rdkafka.ssl.key.location	Path for storing SSL private keys	Mandatory if SSL authentication is enabled	${sandbox_volume_**${VOLUME_NAME}**}/**some.key**

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Description	Mandatory	Available Value
rdkafka.ssl.ca.location	File path or directory of the CA certificate	Mandatory if server certificate authentication is enabled	${sandbox_volume_**${VOLUME_NAME}**}/**some-bundle.crt**
rdkafka.enable.ssl.certificate.verification	Whether to start server certificate authentication	No	The value can be **true** (default) or **false**.

You can configure the volume on the pod and configure annotations to specify the sandbox volume and the corresponding log output configuration file.

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: kafka-tls
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kafka
  template:
    metadata:
      labels:
        app: kafka
      annotations:
        logpath.k8s.io/container-0: /var/log/*.log;/var/paas/sys/log/virtual-kubelet.log
        logconf.k8s.io/fluent-bit-configmap-reference: cci-logging-conf
        sandbox-volume.openvessel.io/volume-names: kafkatls
    spec:
      volumes:
        - name: kafkatls
          secret:
            secretName: cci-sfs-kafka-tls
      containers:
        - name: container-0
          image: 'nginx:alpine'
          resources:
            limits:
              cpu: 1000m
              memory: 2048Mi
            requests:
              cpu: 1000m
              memory: 2048Mi
          volumeMounts:
            - name: kafkatls
              mountPath: /tmp/sfs
      imagePullSecrets:
        - name: default-secret
```

For details about Kafka configuration items, see **https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

11 Managing Network Access Through Service and Ingress

11.1 Service

Direct Access to a Pod

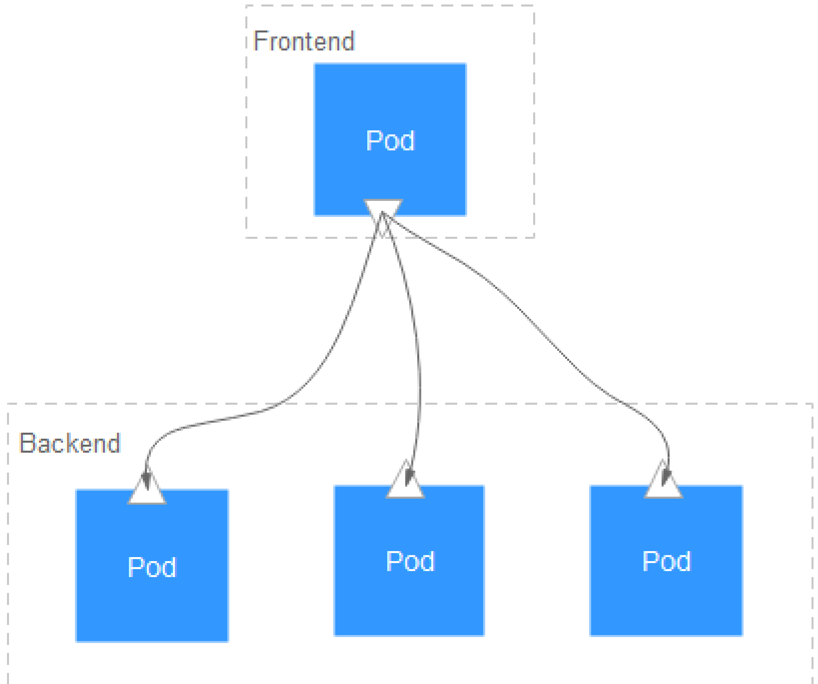
How can I access a workload after it is created? Accessing a workload is to access a pod. However, the following problems may occur when you access a pod directly:

- The pod can be deleted and recreated at any time by a controller such as a Deployment, and the result of accessing the pod becomes unpredictable.
- The IP address of the pod is allocated only after the pod is started. Before the pod is started, the IP address of the pod is unknown.
- An application is usually composed of multiple pods that run the same image. Accessing pods one by one is not efficient.

For example, an application uses Deployments to create the frontend and backend. The frontend calls the backend for computing, as shown in **Figure 11-1**. Three pods are running in the backend, which are independent and replaceable. When a backend pod is recreated, the new pod is assigned with a new IP address and the frontend pod is unaware.

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 11-1 Inter-workload access



How Services Work

Kubernetes Services are used to solve the preceding pod access problems. A Service has a fixed IP address and forwards the traffic to the pods based on labels. In addition, the Service can perform load balancing for these pods.

In the preceding example, two Services are added for accessing the frontend and backend pods. In this way, the frontend pod does not need to sense changes on backend pods, as shown in **Figure 11-2**.

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 11-2 Accessing pods through a Service

Creating a Service

In the following example, create a Service named **nginx**, and use a selector to select the pod with the label of **app:nginx**. The port of the target pod is port 80 while the exposed port of the Service is port 8080.

The Service can be accessed using **Service name:Exposed port**. In the example, **nginx:8080** is used. In this case, other workloads can access the pod associated with **nginx** using **nginx:8080**.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx        #Service name
spec:
  selector:          #Label selector, which selects pods with the label of app=nginx
    app: nginx
  ports:
```


Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
- name: service0
  targetPort: 80   #Pod port
  port: 8080       #Service external port
  protocol: TCP    #Forwarding protocol type. The value can be TCP or UDP.
type: ClusterIP    #Service type
```

NOTE

NodePort Services are not supported in CCI.

Save the Service definition to **nginx-svc.yaml** and use kubectl to create the Service.

```
# kubectl create -f nginx-svc.yaml -n $namespace_name
service/nginx created

# kubectl get svc -n $namespace_name
NAME       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)         AGE
kube-dns   ClusterIP   10.247.9.190     <none>        53/UDP,53/TCP   7m
nginx      ClusterIP   10.247.148.137   <none>        8080/TCP        1h
```

You can see that the Service has a ClusterIP, which is fixed unless the Service is deleted. You can use this ClusterIP to access the Service internally.

NOTE

kube-dns is a Service reserved for domain name resolution. It is automatically created in CCI. For details about domain name resolution, see **Using ServiceName to Access a Service**.

Using ServiceName to Access a Service

In CCI, you can use the **coredns** add-on to resolve the domain name for a Service, and use **ServiceName:Port** to access to the Service. This is the most common mode in Kubernetes. For details about how to install coredns, see **Add-on Management**.

After coredns is installed, it becomes a DNS. After the Service is created, coredns records the Service name and IP address. In this way, the pod can obtain the Service IP address by querying the Service name from coredns.

nginx.<namespace>.svc.cluster.local is used to access the Service. **nginx** is the Service name, **<namespace>** is the namespace, and **svc.cluster.local** is the domain name suffix. In actual use, you can omit **<namespace>.svc.cluster.local** and use the Service name.

For example, if the Service named **nginx** is created, you can access the Service through **nginx:8080** and then access backend pods.

An advantage of using ServiceName is that you can write ServiceName into the program when developing the application. In this way, you do not need to know the IP address of a specific Service.

NOTICE

The coredns add-on occupies compute resources. It runs two pods, with each pod occupies 0.5 vCPUs and 1 GiB of memory. You need to pay for the resources.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

LoadBalancer Services

You have known that you can create ClusterIP Services. You can access backend pods of the Service through the IP address.

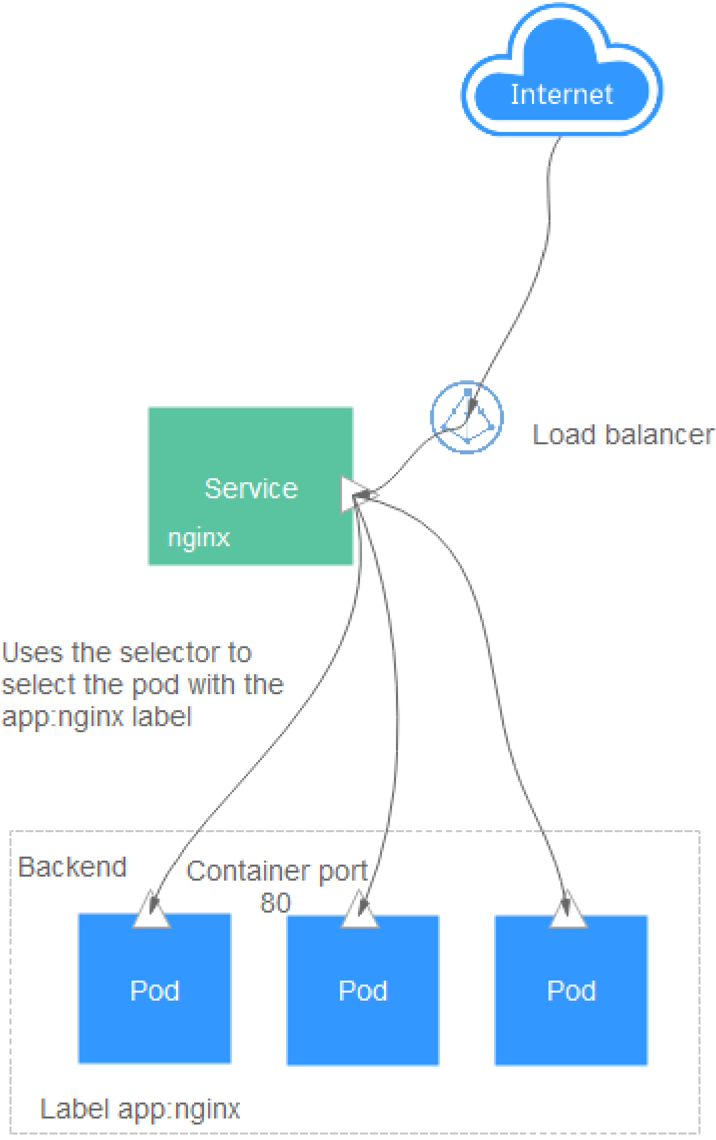
CCI also supports LoadBalancer Services. You can bind a load balancer to a Service. In this way, the traffic for accessing the load balancer is forwarded to the Service.

A load balancer can work on a private network or public network. If the load balancer has a public IP address, it can route requests over the public network. You can create a load balancer by using the **API** or the **ELB console**.

NOTE

- The load balancer must be in the same VPC as the Service.
- Cross-namespace access cannot be achieved using a Service or ELB domain name. It can be implemented only through *Private IP address of load balancer:Port*.

Figure 11-3 LoadBalancer Service



The following is an example of creating a LoadBalancer Service. After a load balancer is created, you can access pods using the IP address and port of the load balancer in the format of *IP address*:*Port*.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    kubernetes.io/elb.id: 77e6246c-a091-xxxx-xxxx-789baa571280  #ID of the load balancer
spec:
  selector:
    app: nginx
  ports:
  - name: service0
    targetPort: 80
    port: 8080         #Port configured for the load balancer
    protocol: TCP
  type: LoadBalancer   #Service type
```

11.2 Ingress

The previous section describes how to create a LoadBalancer Service that uses a load balancer to access pods.

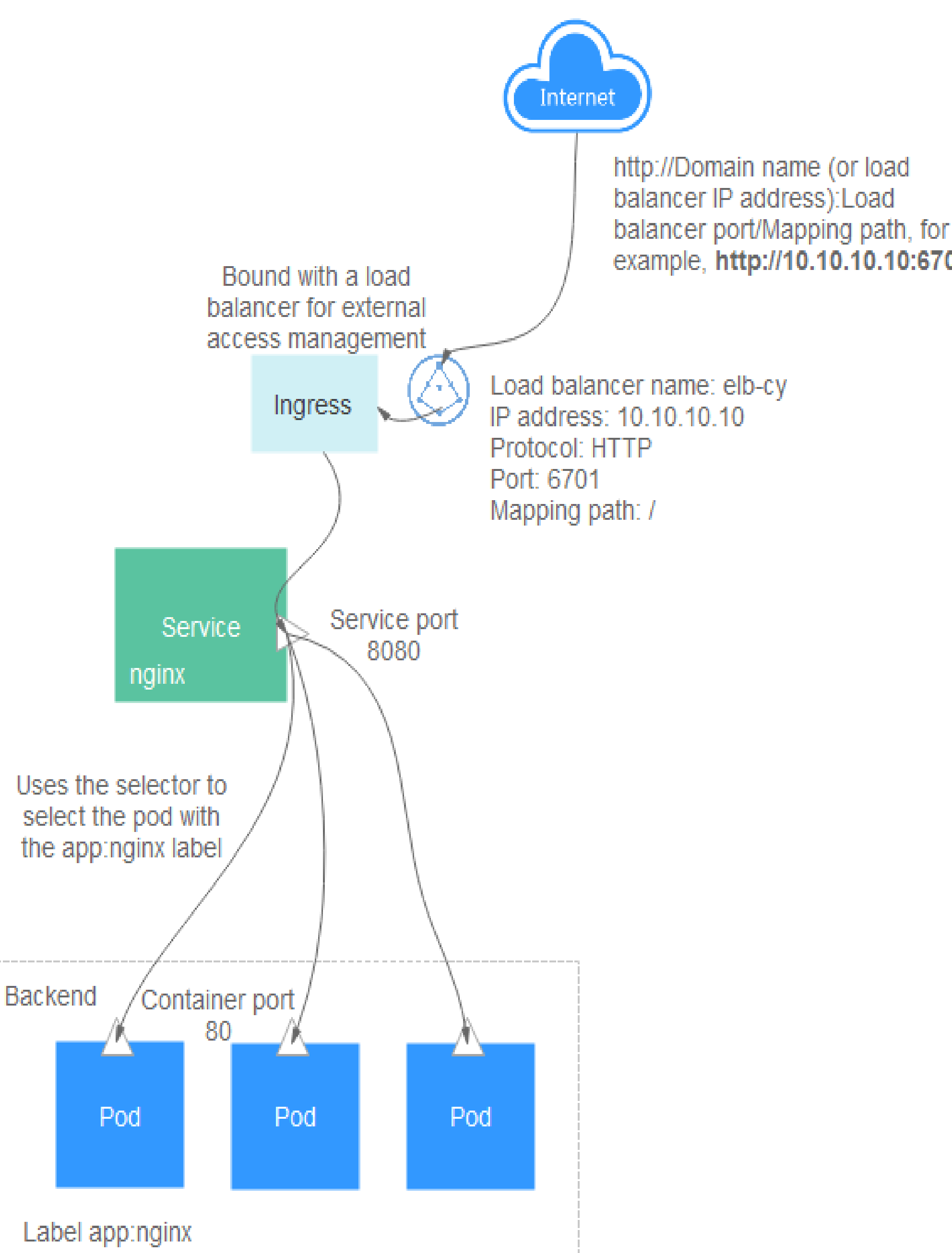
Services forward requests using TCP and UDP at Layer 4. Ingresses can forward requests using HTTP and HTTPS at Layer 7. Domain names and paths can be used for finer granularities.

Figure 11-4 Ingress-Service

In CCI, external access is implemented by binding the load balancer's IP address and port number to an ingress, as shown in **Figure 11-5**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 11-5 Ingress



Load Balancers

Ingresses can be bound to load balancers. You can create a load balancer by using the **API** or the **ELB console**.

A load balancer can work on a private network or public network. If the load balancer has a public IP address, it can route requests over the public network.

Creating an Ingress

- Creating an HTTP ingress

In the following example, the associated backend is **nginx:8080**. When **http://10.10.10.10:6071/** is accessed, the traffic is forwarded to the Service corresponding to **nginx:8080**, and then to the corresponding pod.

```
apiVersion: extensions/v1beta1                          # Ingress version
kind: Ingress
metadata:
  name: nginx
  labels:
    app: nginx
    isExternal: "true"   # This parameter is mandatory and is reserved. The value must be true.
    zone: data        # Data plane mode. This parameter is reserved. The value must be data.
  annotations:
    kubernetes.io/elb.id: 2d48d034-6046-48db-8bb2-53c67e8148b5   # ID of the load balancer. This
parameter is mandatory.
    kubernetes.io/elb.ip: 192.168.137.182                    # IP address of the load balancer. This
parameter is optional.
    kubernetes.io/elb.port: '6071'                       # Port configured for the load balancer. This
parameter is mandatory.
spec:
  rules:                                  # Routing rules
  - http:                                 # Using HTTP protocol
      paths:
      - path: /                           # Route
        backend:
          serviceName: nginx                           # Name of the Service to which requests are
forwarded
          servicePort: 8080                          # Port of the Service to which requests are
forwarded
```

You can also set the external domain name in an ingress so that you can access the load balancer through the domain name and then access backend Services.

NOTE

Domain name-based access depends on domain name resolution. You need to point the domain name to the IP address of the load balancer. For example, you can use **Domain Name Service (DNS)** to resolve domain names.

```
spec:
  rules:
  - host: www.example.com       # Domain name
    http:
      paths:
      - path: /
        backend:
          serviceName: nginx
          servicePort: 80
```

- Creating an HTTPS ingress

  In the following example, the associated backend is **nginx:8080**. When **https://10.10.10.10:6071/** is accessed, the traffic is forwarded to the Service corresponding to **nginx:8080**, and then to the corresponding pod.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/elb.id: 2d48d034-6046-48db-8bb2-53c67e8148b5
    kubernetes.io/elb.ip: 192.168.137.182
    kubernetes.io/elb.port: '6071'
  labels:
    app: nginx
    isExternal: 'true'
    zone: data
  name: nginx
spec:
  rules:
  - http:
```


Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    paths:
    - backend:
        serviceName: nginx
        servicePort: 8080
      path: /
  tls:
  - secretName: cci-sslcertificate-20214221                  # Name of the uploaded SSL 
certificate
```

Accessing Multiple Services

An ingress can access multiple Services at the same time. The configuration is as follows:

- When accessing **http://foo.bar.com/foo**, you access the backend **s1:80**.
- When accessing **http://foo.bar.com/bar**, you access the backend **s2:80**.

```
spec:
  rules:
  - host: foo.bar.com          # Host address
    http:
      paths:
      - path: "/foo"
        backend:
          serviceName: s1
          servicePort: 80
      - path: "/bar"
        backend:
          serviceName: s2
          servicePort: 80
```

Configuring the Routing Service for URL Redirection

In the following example template, an ingress is connected to a backend service named **service-test**, and access requests to the **/service-test** path of the ingress will be redirected to the **/** path of **service-test**.

```
cat <<-EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-redirect-test
  namespace: default
 spec:
  rules:
  - host: ingress-test.com
    http:
      paths:
      - path: /
        backend:
          serviceName: service-test
          servicePort: 80
EOF
```

11.3 Network Access Scenarios

The previous two sections describe how to access pods through Services and ingresses. This section describes scenarios of accessing pods in CCI, as shown in **Figure 11-6**. In each scenario, Services and ingresses can be used to solve access problems.

- Intra-namespace access: You only need to create a Service. A workload can be accessed from other workloads in the same namespace by using **Service name:Service port**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Intra-VPC access: A workload can be accessed from other workloads in the same VPC by using the IP address of a Service or the IP address of the private network load balancer bound to an ingress.
- Inter-VPC access: You can create a VPC **peering connection** to connect two VPCs. A workload is accessible to workloads in the other VPC by using the IP address of a Service or the IP address of the private network load balancer.
- Access to a workload from the public network: A workload can be accessed from the outside of Huawei Cloud by using the IP address of the public network load balancer bound to an ingress.
- Access to the public network from a workload: You can configure source network address translation (SNAT) rules in **NAT Gateway**, so that containers can access the public network. For details, see **Accessing Public Networks from a Container**.

Figure 11-6 Network access diagram

11.4 Readiness Probe

After a pod is created, the Service can immediately select it and forward requests to it. However, it takes time to start a pod. If the pod is not ready (it takes time to load the configuration or data, or a preheating program may need to be executed), the pod cannot process requests, and the requests will fail.

Kubernetes solves this problem by adding a readiness probe to pods. The Service can forward requests to a pod only after the probe detects that the pod is ready.

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The readiness probe periodically detects a pod and determines whether the pod is ready based on the response. Similar to **Liveness Probe**, CCI also supports two types of readiness probes.

- HTTP GET: The probe sends an HTTP GET request to the container by using **IP:port**. If the probe receives a 2xx or 3xx status code, the container is ready.

  NOTE

  You need to configure the following annotation for the pod to make **timeoutSeconds** take effect:

  cci.io/httpget-probe-timeout-enable:"true"

  For details, see the example in **Advanced Configuration of Liveness Probe**.

- Exec: The probe runs a command in the container and checks the exit status code. If the exit status code is 0, the container is ready.

Working Principles of the Readiness Probe

If you run the **kubectl describe** command to query the Service, information similar to the following is displayed:

```
$ kubectl describe svc nginx -n $namespace_name
Name:             nginx
......
Endpoints:        192.168.113.81:80,192.168.165.64:80,192.168.198.10:80
......
```

Endpoints is displayed, which is also a resource object in Kubernetes.

```
$ kubectl get endpoints -n $namespace_name
NAME       ENDPOINTS                                                  AGE
nginx      192.168.113.81:80,192.168.165.64:80,192.168.198.10:80   14m
```

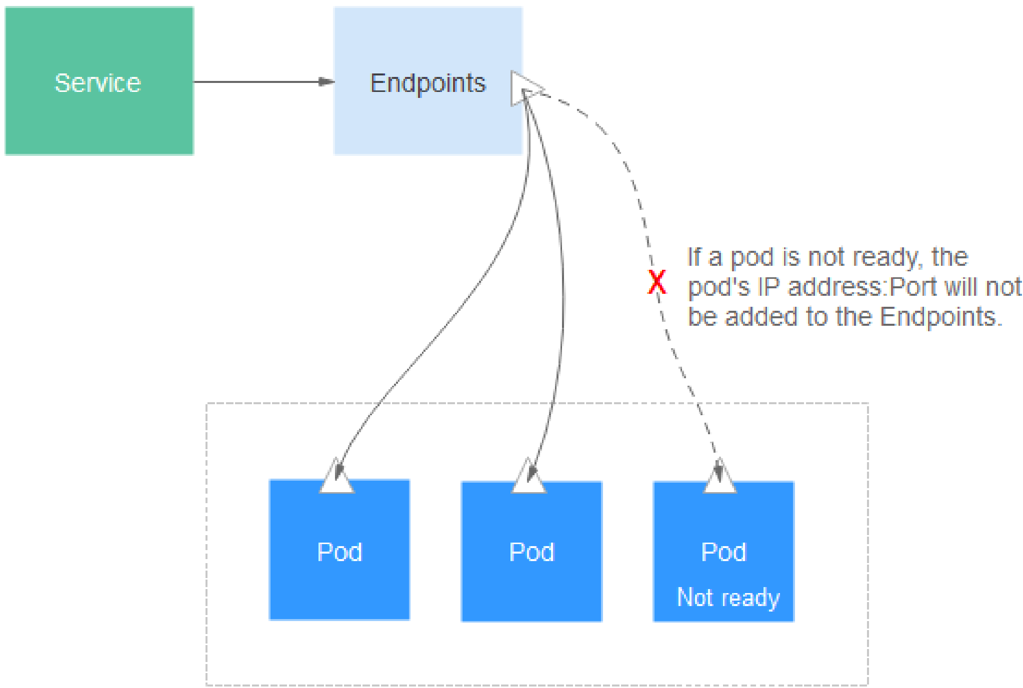
192.168.113.81:80 is the **IP:port** of the pod. You can run the following command to view the IP address of the pod, which is the same as the preceding IP address.

```
# kubectl get pods -o wide -n $namespace_name
NAME                    READY     STATUS    RESTARTS   AGE
IP
nginx-55c54cc5c7-49chn   1/1       Running   0          1m        192.168.198.10
nginx-55c54cc5c7-x87lb   1/1       Running   0          1m        192.168.165.64
nginx-55c54cc5c7-xp4c5   1/1       Running   0          1m        192.168.113.81
```

Endpoints can be used as a readiness probe. When the pod is not ready, **IP:port** is deleted from the Endpoints and is added to the Endpoints after the pod is ready, as shown in the following figure.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 11-7 Working principles of the readiness probe



Exec

The Exec mode is the same as the HTTP GET mode. As shown below, the probe runs the **ls /ready** command. If the file exists, **0** is returned, indicating that the pod is ready. Otherwise, another status code is returned.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        name: container-0
        resources:
          limits:
            cpu: 500m
            memory: 1024Mi
          requests:
            cpu: 500m
            memory: 1024Mi
        readinessProbe:      # Readiness Probe
          exec:              # Define the ls /ready command.
            command:
            - ls
            - /ready
      imagePullSecrets:
      - name: imagepull-secret
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Save the definition of the Deployment to **deploy-read.yaml**, delete the previously created Deployment, and use **deploy-read.yaml** to recreate the Deployment.

```
# kubectl delete deploy nginx -n $namespace_name
deployment.apps "nginx" deleted

# kubectl create -f deploy-read.yaml -n $namespace_name
deployment.apps/nginx created
```

The **nginx** image does not contain the **/ready** file. Therefore, the container is not in **Ready** state after the creation, as shown below. Note that the value in the **READY** column is **0/1**, indicating that the container is not ready.

```
# kubectl get po -n $namespace_name
NAME                    READY     STATUS    RESTARTS   AGE
nginx-7955fd7786-686hp   0/1       Running   0          7s
nginx-7955fd7786-9tgwq   0/1       Running   0          7s
nginx-7955fd7786-bqsbj   0/1       Running   0          7s
```

Check the Service again. If there are no values in the **Endpoints** line, no Endpoints are found.

```
$ kubectl describe svc nginx -n $namespace_name
Name:              nginx
......
Endpoints:
......
```

If a **/ready** file is created in the container to make the readiness probe succeed, the container is in the **Ready** state. Check the pod and **Endpoints**. It is found that the container for which the **/ready** file is created is ready and an **Endpoints** record is added.

```
# kubectl exec -n $namespace_name nginx-7955fd7786-686hp -- touch /ready

# kubectl get po -o wide -n $namespace_name
NAME                    READY     STATUS    RESTARTS   AGE       IP
nginx-7955fd7786-686hp   1/1       Running   0          10m       192.168.93.169
nginx-7955fd7786-9tgwq   0/1       Running   0          10m       192.168.166.130
nginx-7955fd7786-bqsbj   0/1       Running   0          10m       192.168.252.160

# kubectl get endpoints -n $namespace_name
NAME       ENDPOINTS           AGE
nginx      192.168.93.169:80   14d
```

HTTP GET

The configuration of a readiness probe is the same as that of a **liveness probe**, which is also in the **containers** field of the pod description template. As shown below, the readiness probe sends an HTTP request to the pod. If the probe receives **2xx** or **3xx**, the pod is ready.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    containers:
    - image: nginx:latest
      name: container-0
      resources:
        limits:
          cpu: 500m
          memory: 1024Mi
        requests:
          cpu: 500m
          memory: 1024Mi
      readinessProbe:          # readinessProbe
        httpGet:               # HTTP GET definition
          path: /read
          port: 80
    imagePullSecrets:
    - name: imagepull-secret
```

Advanced Configuration of Readiness Probe

Similar to the liveness probe, the readiness probe also has the same advanced configuration items. The output of the **describe** command of the **nginx** pod is as follows:

```
Readiness: exec [ls /var/ready] delay=0s timeout=1s period=10s #success=1 #failure=3
```

This line indicates the parameter configuration of the readiness probe. The meanings of the parameters are as follows:

- **delay=0s** indicates that the probe starts immediately after the container is started.
- **timeout=1s** indicates that the container must respond to the probe within 1s. Otherwise, the detection fails.
- **period=10s** indicates that the detection is performed every 10s.
- **#success=1** indicates that the detection is successful after succeeding once.
- **#failure=3** indicates that the container will be restarted after three consecutive detection failures.

These are set by default when the probe is created. You can also manually configure the parameters as follows:

```
      readinessProbe:      # Readiness Probe
        exec:              # Define the ls /readiness/ready command.
          command:
          - ls
          - /readiness/ready
        initialDelaySeconds: 10    # Readiness probes are initiated after the container has started for 10s.
        timeoutSeconds: 2          # The container must respond to the probe within 2s, or the detection fails.
        periodSeconds: 30          # The probe is performed every 30s.
        successThreshold: 1        # The container is considered ready as long as the probe succeeds once.
        failureThreshold: 3        # The probe is considered to be failed after three consecutive failures.
```

12 Using PersistentVolumeClaim to Apply for Persistent Storage

CCI supports the following persistent storage services in containers:

- **Elastic Volume Service (EVS)** is a block storage service that provides three specifications: common I/O (previous-generation), high I/O (SAS), and ultra-high I/O (SSD).
- SFS Turbo is expandable to 320 TB, and provides fully hosted shared file storage. It features high availability and durability, and supports massive quantities of small files and applications requiring low latency and high IOPS. You can use SFS Turbo in high-traffic websites, log storage, compression/decompression, DevOps, enterprise OA, and containerized applications.
- **Object Storage Service (OBS)** is an object-based storage service, and provides massive, secure, highly reliable, and low-cost data storage.

EVS need to be mounted before being used. The following describes how to use EVS.

PersistentVolumeClaim (PVC)

Kubernetes provides PVC to apply for persistent storage. The PVC allows you to specify the type and capacity of storage without concerning about how to create and release underlying storage resources.

In practice, you can associate a PVC with the volume in the pod and use the persistent storage through the PVC, as shown in **Figure 12-1**.

Figure 12-1 Using persistent storage

Creating a PVC

- Creating a PVC to apply for a 100-GB SAS EVS disk

  To create an encrypted EVS volume, add the **paas.storage.io/cryptKeyId** field in **metadata.annotations**.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-evs
  namespace: namespaces-test
  annotations: {
  paas.storage.io/cryptKeyId: ee9b610c-e356-11e9-aadc-d0efc1b3bb6b
  }
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
  storageClassName: sas
```

  accessModes indicates the volume access modes. The following three modes are supported:

  – **ReadWriteOnce**: A volume can be mounted to a single node for reading and writing.

  – **ReadOnlyMany**: A volume can be mounted to multiple nodes for reading.

  – **ReadWriteMany**: A volume can be mounted to multiple nodes for reading and writing.

  storageClassName indicates the applied storage class. Currently, the following 3 storage classes are supported:

  – **sas**: SAS (high I/O) EVS disk

  – **ssd**: SSD (ultra-high I/O) EVS disk

  – **nfs-rw**: SFS file storage of the standard file protocol

Using a PVC

After applying for storage resources using a PVC, you can use a volume in the pod to associate the PVC and mount the volume to containers.

The following example shows how to use a PVC in a pod. A volume named **pvc-test-example** is defined and mounted to the **/tmp/volume0** directory of the container. In this way, the data written to **/tmp** is written to the PVC named **pvc-test**.

- Writing data to the applied SAS EVS disk

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - image: nginx:latest
    name: container-0
    resources:
      limits:
        cpu: 500m
```

```
      memory: 1024Mi
    requests:
      cpu: 500m
      memory: 1024Mi
  volumeMounts:
  - mountPath: "/tmp/volume0"   # Mount the PVC to the /tmp/volume0 directory of the container.
    name: pvc-test-example      # Volume name.
volumes:                        # Define a volume, and associate it with the PVC.
- name: pvc-test-example
  persistentVolumeClaim:
    claimName: pvc-test         # PVC name.
imagePullSecrets:
- name: imagepull-secret
```

13 ConfigMap and Secret

13.1 ConfigMap

A ConfigMap is a resource object for storing configuration information required by applications. It uses the key-value pair to save configuration data. It can be used to save a single attribute or configuration file.

A ConfigMap can be used to decouple configuration and make different configurations in different environments. Compared with the environment variables, the ConfigMap referenced in the pod can be updated in real time. After the ConfigMap data is updated, the ConfigMap referenced in the pod is updated synchronously.

Creating a ConfigMap

In the following example, a ConfigMap named **configmap-test** is created. The ConfigMap configuration data is defined in the **data** field.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-test
data:                     # Configuration data
  property_1: Hello
  property_2: World
```

Referencing a ConfigMap in Environment Variables

A ConfigMap is usually referenced in environment variables and volumes.

In the following example, the **property_1** of **configmap-test** is used as the value of the environment variable **EXAMPLE_PROPERTY_1**. In this case, the value of **EXAMPLE_PROPERTY_1** is the value of **property_1** after the container is started, that is, **Hello**.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
```

```
- image: nginx:latest
  name: container-0
  resources:
    limits:
      cpu: 500m
      memory: 1024Mi
    requests:
      cpu: 500m
      memory: 1024Mi
  env:
  - name: EXAMPLE_PROPERTY_1
    valueFrom:
       configMapKeyRef:         # Reference the ConfigMap.
        name: configmap-test
        key: property_1
imagePullSecrets:
- name: imagepull-secret
```

Referencing a ConfigMap in a Volume

Referencing a ConfigMap in a volume is to fill its data in configuration files in the volume. Each piece of data is saved in a file. The key is the file name, and the key value is the file content.

In the following example, create a volume named **vol-configmap**, reference the ConfigMap named **configmap-test** in the volume, and mount the volume to the **/tmp** directory of the container. After the pod is created, there are two files **property_1** and **property_2** in the **/tmp** directory of the container, and the values are **Hello** and **World**.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: container-0
    resources:
      limits:
        cpu: 500m
        memory: 1024Mi
      requests:
        cpu: 500m
        memory: 1024Mi
    volumeMounts:
    - name: vol-configmap           # Mount the volume named vol-configmap.
      mountPath: "/tmp1"
  imagePullSecrets:
  - name: imagepull-secret
  volumes:
  - name: vol-configmap
    configMap:                   # Reference the ConfigMap.
      name: configmap-test
```

13.2 Secret

A secret is a resource object for encrypted storage. You can save the authentication information, certificates, and private keys in a secret, solving the configuration problems of sensitive data such as passwords, tokens, and keys. In this case, sensitive data will not be exposed to images or pod specification files. You only need to load such data as environment variables to containers during container startup.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Similar to a ConfigMap, a secret saves data using key-value pairs. The difference is that a secret is encrypted and suitable for storing sensitive information.

Base64 Encoding

Similar to a ConfigMap, a secret saves data using key-value pairs. The difference is that secret values must be encoded using the Base64 method.

To encrypt a character string using Base64, run the **echo -n *to-be-encoded content* | base64** command. The following is an example:

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

Creating a Secret

The secret defined in the following example contains two key-value pairs.

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  key1: 
VkZNME0wVlpVbEpQVHpGTFdrSkRWVWhCV2s5T1ZrNUxUVlZNUjBzMFRWcElVMFpVUkVWV1N3PT0=   # 
Base64 encoded value
  key2: T0VkR1RGRlZVRlpVU2xCWFdUZFBVRUZCUmtzPQ==                                 # Base64 encoded 
value
```

Referencing a Secret in Environment Variables

In most cases, a secret is injected into a container as an environment variable, as shown in the following example.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: container-0
    resources:
      limits:
        cpu: 500m
        memory: 1024Mi
      requests:
        cpu: 500m
        memory: 1024Mi
    env:
    - name: key
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: key1
  imagePullSecrets:
  - name: imagepull-secret
```

Referencing a Secret in a Volume

Referencing a secret in a volume is to fill its data in configuration files in the volume. Each piece of data is saved in a file. The key is the file name, and the key value is the file content.

In the following example, create a volume named **vol-secret**, reference the secret named **mysecret** in the volume, and mount the volume to the **/tmp** directory of the container. After the pod is created, there are two files **key1** and **key2** in the **/tmp** directory of the container, and the values are **VkZNME0wVlpVbEpQVHpGTFdrSkRWVWhCV2s5T1ZrNUxUVlZNUjBzMFRWcElVMFpVUkVWV1N3PT0=** and **T0VkR1RGRlZVRlpVU2xCWFdUZFBVRUZCUmtzPQ==**.

NOTE

The values of **key1** and **key2** are the values encoded using Base64.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: container-0
    resources:
      limits:
        cpu: 500m
        memory: 1024Mi
      requests:
        cpu: 500m
        memory: 1024Mi
    volumeMounts:
    - name: vol-secret              # Mount the volume named vol-secret
      mountPath: "/tmp"             # Mount path. The value contains a maximum of 256 characters.
  imagePullSecrets:
  - name: imagepull-secret
  volumes:
  - name: vol-secret
    secret:                         # Reference a secret
      secretName: mysecret
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

14 Creating a Workload Using Job and Cron Job

A job workload is responsible for batch processing of short lived one-off tasks, that is, tasks that are executed only once. It ensures that one or more pods are successfully completed.

- A job is a resource object that Kubernetes uses to control batch tasks. A job is different from a long-term servo workload (such as Deployment and StatefulSet). The former is completed when a specified number of successful completions is reached, while the latter runs unceasingly if not terminated. The pods managed by the job will be automatically removed after successfully completing the job based on user configurations.
- A cron job runs a job periodically on a specified schedule. A cron job object is similar to a line of a crontab file in Linux.

This run-and-stop feature of the task workload is especially suitable for one-off tasks, such as CI. It works with the per-second billing of the CCI to implement pay-per-use in real sense.

Constraints

EVS volumes created using **flexVolume** can only be deleted when the pods are in the **Terminated** state. When the pod status is **Completed**, the EVS volumes created using this field will not be deleted.

Creating a Job

The following is an example job, which calculates π till the 2000th digit and prints the output. 50 pods need to be run before the job is ended. In this example, print π calculation results for 50 times, and run five pods concurrently. If a pod fails to be run, a maximum of five retries are supported.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-with-timeout
  namespace: cci-namespace-test1
spec:
  completions: 50            # Number of pods that need to run successfully to end the job
  parallelism: 5            # Number of pods that run concurrently. The default value is 1.
  backoffLimit: 5            # Maximum number of retries performed if a pod fails. When the limit is reached,
```

```
it will not try again.
  activeDeadlineSeconds: 10  # Timeout duration of pods. Once the time is reached, all pods of the job are
terminated.
  template:                  # Pod definition
    spec:
      containers:
      - name: pi
        image: perl
        command:
        - perl
        - "-Mbignum=bpi"
        - "-wle"
        - print bpi(2000)
      restartPolicy: Never
```

Based on the **completions** and **Parallelism** settings, jobs can be classified as follows:

Table 14-1 Job types

Job Type	Description	Example
One-off job	One pod runs until it is successfully ends.	Database migration
Jobs with a fixed completion count	One pod runs until the specified completion count is reached.	Pod for processing work queues
Parallel jobs with a fixed completion count	Multiple pods run until the specified completion count is reached.	Multiple pods for processing work queues concurrently
Parallel jobs	One or more pods run until one pod is successfully ended.	Multiple pods for processing work queues concurrently

Creating a Cron Job

Compared with a job, a cron job is a scheduled job. A cron job runs a job periodically on a specified schedule, and the job creates a pod.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-example
  namespace: cci-namespace-test1
spec:
  schedule: "0,15,30,45 * * * *"           # Scheduling configuration
  jobTemplate:                             # Job definition
    spec:
      template:
        spec:
          restartPolicy: OnFailure
          containers:
          - name: main
            image: pi
```

The format of the cron is as follows:

- Minute
- Hour
- Day of month
- Month
- Day of week

For example, in **0,15,30,45 * * * ***, commas separate minutes, the first asterisk (*) indicates the hour, the second asterisk indicates the day of the month, the third asterisk indicates the month, and the fourth asterisk indicates the day of the week.

If you want to run the job every half an hour on the first day of each month, set this parameter to **0,30 * 1 * ***. If you want to run the job at 3:00 a.m. every Sunday, set this parameter to **0 3 * * 0**.

For details about the cron format, see **https://en.wikipedia.org/wiki/Cron**.

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

A YAML Syntax

YAML is a simple and powerful language. It is designed to make the language easy to read.

Basic Syntax Rules

- Characters are case-sensitive.
- Indentation is used for denoting structure.
- Only spaces can be used for indentation, but tab characters are not allowed.
- The specific number of spaces in the indentation is unimportant as long as parallel elements have the same left justification.
- Comments begin with the number sign (#).

Data Types Supported by YAML

- Object: A set of key-value pairs, which is also known as maps, hashes, or dictionaries.
- Array: A group of values arranged in sequence, which is also known as sequence or list.
- Scalar: A single and irreducible value, which is the minimum data unit.

Object

An object is a group of key-value pairs. For key: value, the colon (:) must be followed by a space or newline character. The valid expression is as follows:

```
animal: pets
plant:
  tree
```

You can also write multiple key-value pairs into an inline object.

```
hash: {name: Steve, foo: bar}
```

However, an error occurs in the following scenario:

```
foo: somebody said I should put a colon here: so I did
windows_drive: c:
```

To resolve the issue, you can enclose values in single quotation marks (' ') as follows:

```
foo: 'somebody said I should put a colon here: so I did'
windows_drive: 'c:'
```

Array

An array is represented by a hyphen (-) and space. The valid expression is as follows:

```
animal:
- Cat
- Dog
- Goldfish
```

You can also use the inline representation.

```
animal: [Cat, Dog, Goldfish]
```

Objects and arrays can be used in combination to form a composite structure.

```
languages:
 - Ruby
 - Perl
 - Python
websites:
 YAML: yaml.org
 Ruby: ruby-lang.org
 Python: python.org
 Perl: use.perl.org
```

Scalar

Scalars include strings, Boolean values, integers, floats, null, time, and dates.

- String:

  By default, a string is not enclosed in quotation marks.

  ```
  str:This_is_a_line
  ```

  If a string contains spaces or special characters, the string needs to be enclosed in quotation marks.

  ```
  str: 'content: a string'
  ```

  Both single and double quotation marks can be used. The difference between them is that the former can identify escape characters while the latter cannot convert special characters.

  ```
  s1: 'content:\n a string'
  s2: "content:\n a string"
  ```

  If there is a single quotation mark between two single quotation marks, ensure that two consecutive single quotation marks are used to achieve conversion.

  ```
  str: 'labor''s day'
  ```

  Strings can be written into multiple lines. The lines except the first line must be indented with one space. The newline character will be converted to a space.

  ```
  str: This_is
   a_multi_line
  ```

- Integer:

  ```
  int_value: 314
  ```

- Float:

  ```
  float_value: 3.14
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Null:

```
parent: ~
```

- Time

  The time is in the ISO8601 format.

```
iso8601: 2018-12-14t21:59:43.10-05:00
```

- Date:

  The date is in the compound ISO8601 format: year-month-day.

```
date: 1976-07-31
```

Special Symbols

- Three hyphens (---) indicate the start of a YAML file. Three periods (...) indicate the end of a YAML file.

```
---
# A list of delicious fruits
- Apple
- Orange
- Strawberry
- Mango
...
```

- You can use two exclamation marks (!!) to forcibly convert an integer, a float, or a Boolean value.

```
strbool: !!str true
strint: !!str 10
```

- For a string occupying multiple lines, you can use a literal block scalar (|) to preserve newlines or folded block scalar (>) to fold newlines. The two symbols are often used in the character strings in YAML files.

```
this: |
 Foo
 Bar
that: >
 Foo
 Bar
```

The corresponding objects are as follows:

```
{ this: 'Foo\nBar\n', that: 'Foo Bar\n' }
```

It is recommended that you use "|" to meet the requirements of most scenarios.

Comment

YAML supports comments. This is an advantage of YAML compared with JSON.

Comments in YAML files begin with the number sign (#), as shown in the following:

```
languages:
 - Ruby             # Ruby programming language
 - Go               # Go programming language
 - Python           # Python programming language
```

Reference Documents

- **YAML 1.2 Specification**
- **Ansible YAML Syntax**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.