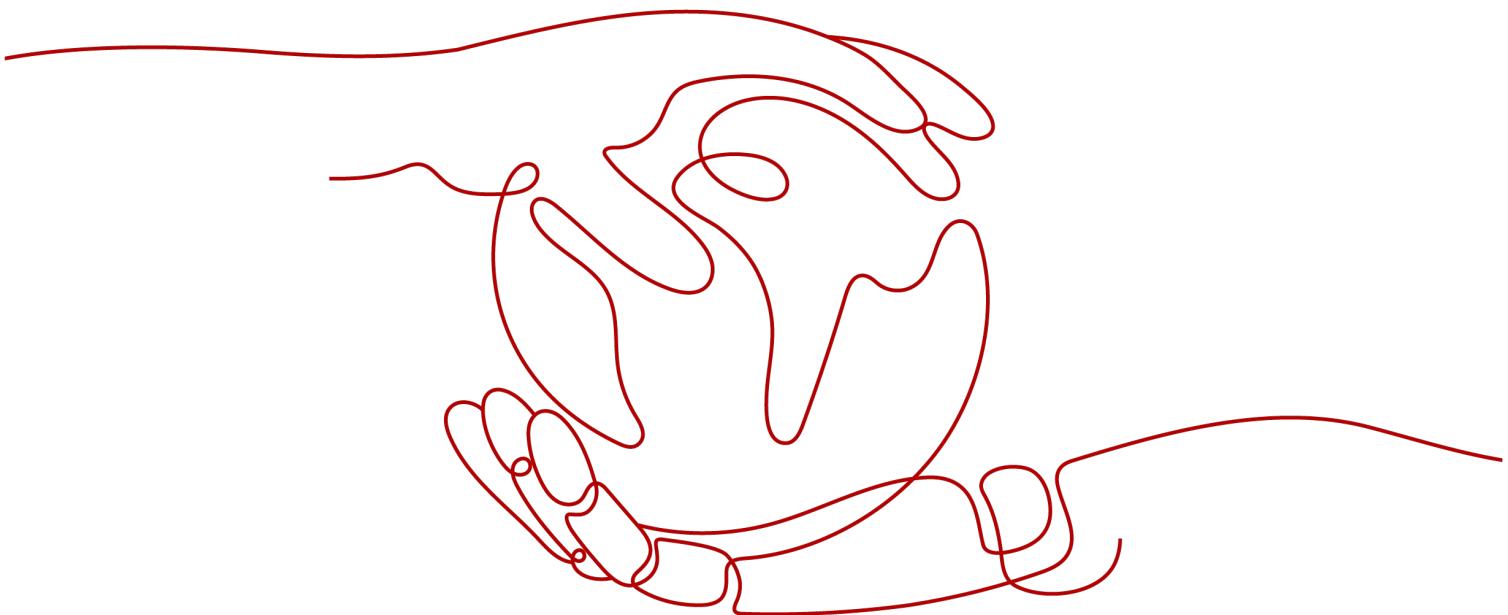


# Cloud Container Engine

## Best Practices

Issue 01

Date 2022-12-13



HUAWEI TECHNOLOGIES CO., LTD.



**Copyright © Huawei Technologies Co., Ltd. 2022. All rights reserved.**

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

## **Trademarks and Permissions**



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

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

## **Notice**

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

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

# Contents

---

<b>1 Checklist for Deploying Containerized Applications in the Cloud.....</b>	<b>1</b>
<b>2 Containerization.....</b>	<b>9</b>
2.1 Containerizing an Enterprise Application (ERP).....	9
2.1.1 Solution Overview.....	9
2.1.2 Resource and Cost Planning.....	13
2.1.3 Procedure.....	13
2.1.3.1 Containerizing an Entire Application.....	13
2.1.3.2 Containerization Process.....	15
2.1.3.3 Analyzing the Application.....	16
2.1.3.4 Preparing the Application Runtime.....	18
2.1.3.5 Compiling a Startup Script.....	20
2.1.3.6 Compiling the Dockerfile.....	21
2.1.3.7 Building and Uploading an Image.....	22
2.1.3.8 Creating a Container Workload.....	24
2.2 Containerizing a Game Application (WOW).....	27
2.2.1 Solution Overview.....	27
2.2.2 Resource and Cost Planning.....	31
2.2.3 Procedure.....	31
2.2.3.1 Deployment Process.....	31
2.2.3.2 Preparation: Reconstructing the Game Application Architecture.....	32
2.2.3.3 Analyzing the Game Application.....	34
2.2.3.4 Preparing the Environment.....	36
2.2.3.5 Deploying the Game Application.....	38
2.2.3.6 Scaling the Game Application.....	45
2.2.3.7 Upgrading the Game Application.....	46
2.2.3.8 Deleting Resources.....	47
2.2.3.9 FAQs.....	48
<b>3 Migration.....</b>	<b>49</b>
3.1 Migrating Container Images.....	49
3.1.1 Overview.....	49
3.1.2 Migrating Images to SWR Using Docker Commands.....	51
3.1.3 Migrating Images to SWR Using image-syncer.....	52

3.1.4 Synchronizing Images Across Clouds from Harbor to SWR.....	55
3.2 Migrating On-premises Kubernetes Clusters to CCE.....	62
3.2.1 Solution Overview.....	62
3.2.2 Planning Resources for the Target Cluster.....	67
3.2.3 Migrating Resources Outside a Cluster.....	70
3.2.4 Installing the Migration Tool.....	72
3.2.5 Migrating Resources in a Cluster (Velero).....	76
3.2.6 Migrating Resources in a Cluster (e-backup).....	79
3.2.7 Updating Resources Accordingly.....	84
3.2.8 Performing Additional Tasks.....	88
3.2.9 Troubleshooting.....	89
3.3 Migrating Clusters from Other Clouds to CCE.....	90
3.3.1 Solution Overview.....	90
3.3.2 Resource and Cost Planning.....	92
3.3.3 Procedure.....	93
3.3.3.1 Migrating Data.....	93
3.3.3.2 Installing the Migration Tool.....	93
3.3.3.3 Migrating Resources in a Cluster (Velero).....	98
3.3.3.4 Migrating Resources in a Cluster (e-backup).....	101
3.3.3.5 Preparing Object Storage and Velero.....	106
3.3.3.6 Backing Up Kubernetes Objects of the ACK Cluster.....	107
3.3.3.7 Restoring Kubernetes Objects in the Created CCE Cluster.....	108
3.3.3.8 Update and Adaptation.....	108
3.3.3.9 Debugging and Starting the Application.....	109
3.3.3.10 Others.....	109
3.3.4 Change History.....	110
<b>4 DevOps.....</b>	<b>111</b>
4.1 Installing, Deploying, and Interconnecting Jenkins with SWR and CCE Clusters.....	111
4.1.1 Solution Overview.....	111
4.1.2 Resource and Cost Planning.....	114
4.1.3 Procedure.....	114
4.1.3.1 Installing and Deploying Jenkins Master.....	115
4.1.3.2 Configuring Jenkins Agent.....	120
4.1.3.3 Using Jenkins to Build a Pipeline.....	131
4.1.3.4 Interconnecting Jenkins with RBAC of Kubernetes Clusters (Example).....	134
4.2 Interconnecting GitLab with SWR and CCE for CI/CD.....	140
<b>5 Disaster Recovery.....</b>	<b>146</b>
5.1 Implementing High Availability for Containers in CCE.....	146
<b>6 Security.....</b>	<b>149</b>
6.1 Suggestions on Selecting CCE Clusters.....	149
6.2 Cluster Security.....	150

6.3 Node Security.....	154
6.4 Container Security.....	156
6.5 Secret Security.....	158
6.6 Workload Identities.....	160
<b>7 Auto Scaling.....</b>	<b>166</b>
7.1 Using HPA and CA for Auto Scaling of Workloads and Nodes.....	166
7.2 Auto Scaling Based on ELB Monitoring Metrics.....	175
<b>8 Cluster.....</b>	<b>184</b>
8.1 Creating an IPv4/IPv6 Dual-Stack Cluster in CCE.....	184
8.2 Creating a Custom CCE Node Image.....	192
8.3 Using a Private Image to Build a Worker Node Image.....	197
8.4 Cleaning Up CCE Resources on a Deleted Node.....	202
8.5 Connecting to Multiple Clusters Using kubectl.....	204
8.6 Creating a Node Injection Script.....	209
8.7 Adding a Second Data Disk to a Node in a CCE Cluster.....	213
<b>9 Networking.....</b>	<b>216</b>
9.1 Planning CIDR Blocks for a Cluster.....	216
9.2 Selecting a Network Model.....	223
9.3 Allowing Containers and IDCs to Communicate with Each Other Through VPC and Direct Connect.....	228
9.4 Enabling a CCE Cluster to Resolve Domain Names on Both On-Premises IDCs and HUAWEI CLOUD.....	232
9.4.1 Solution Overview.....	232
9.4.2 Solution 1: Using a DNS Endpoint for Cascading Resolution.....	235
9.4.3 Solution 2: Changing the CoreDNS Configurations.....	239
9.5 Implementing Sticky Session Through Load Balancing.....	241
9.6 Obtaining the Client Source IP Address for a Container.....	244
9.7 Increasing the Listening Queue Length by Configuring Container Kernel Parameters.....	249
9.8 Using Multiple ENIs on a Node in a CCE Cluster.....	253
9.9 Enabling Passthrough Networking for LoadBalancer Services.....	254
9.10 CoreDNS Configuration Optimization.....	258
9.10.1 Overview.....	258
9.10.2 Client.....	259
9.10.2.1 Optimizing Domain Name Resolution Requests.....	259
9.10.2.2 Selecting a Proper Image.....	260
9.10.2.3 Avoiding Occasional DNS Resolution Timeout Caused by IPVS Defects.....	260
9.10.2.4 Using NodeLocal DNSCache.....	260
9.10.2.5 Upgrading the CoreDNS in the Cluster Timely.....	260
9.10.2.6 Adjusting the DNS Configuration of the VPC and VM.....	261
9.10.3 Server.....	261
9.10.3.1 Monitors the coredns Add-on.....	261
9.10.3.2 Adjusting the CoreDNS Deployment Status.....	262
9.10.3.3 Configuring CoreDNS.....	263

<b>10 Storage.....</b>	<b>269</b>
10.1 Expanding Node Disk Capacity.....	269
10.2 Mounting an Object Storage Bucket of a Third-Party Tenant.....	272
10.3 Dynamically Creating and Mounting Subdirectories of an SFS Turbo File System.....	277
10.4 How Do I Change the Storage Class Used by a Cluster of v1.15 from FlexVolume to CSI Everest?...281	
10.5 Using OBS Parallel File Systems.....	293
10.6 Custom Storage Classes.....	294
10.7 Realizing Automatic Topology for EVS Disks When Nodes Are Deployed Across AZs (csi-disk-topology).....	301
<b>11 Container.....</b>	<b>307</b>
11.1 Properly Allocating Container Computing Resources.....	307
11.2 Upgrading Pods Without Interrupting Services.....	309
11.3 Modifying Kernel Parameters Using a Privileged Container.....	310
11.4 Initializing a Container.....	311
11.5 Setting Time Zone Synchronization.....	313
11.6 Setting the Container Network Bandwidth Limit.....	316
11.7 Using hostAliases to Configure /etc/hosts in a Pod.....	318
11.8 Configuring Domain Name Resolution for CCE Containers.....	320
11.9 How Do I Select a Container Runtime?.....	325
11.10 Using Dual-Architecture Images (x86 and Arm) in CCE.....	328
11.11 Configuring Core Dumps.....	331
<b>12 Permission.....</b>	<b>333</b>
12.1 Configuring kubeconfig for Fine-Grained Management on Cluster Resources.....	333
12.2 Performing Cluster Namespace RBAC.....	337
<b>13 Release.....</b>	<b>342</b>
13.1 Overview.....	342
13.2 Using Services to Implement Simple Grayscale Release and Blue-Green Deployment.....	345
13.3 Using Nginx Ingress to Implement Grayscale Release and Blue-Green Deployment.....	351
<b>14 Batch Computing.....</b>	<b>361</b>
14.1 Using Kubeflow and Volcano to Train an AI Model.....	361
14.2 Running Kubeflow in CCE.....	367
14.2.1 Deploying Kubeflow.....	367
14.2.2 Training TensorFlow Models.....	369
14.3 Running Caffe in CCE.....	372
14.3.1 Prerequisites.....	373
14.3.2 Preparing Resources.....	375
14.3.3 Caffe Classification Example.....	376
14.4 Running TensorFlow in CCE.....	379
14.5 Running Flink in CCE.....	385
14.6 Deploying ClickHouse on CCE.....	390
14.6.1 Creating a CCE Cluster.....	390

14.6.2 Configuring kubectl.....	391
14.6.3 Deploying ClickHouse Operator.....	391
14.6.4 Examples.....	392
14.7 Running Spark on CCE.....	396
14.7.1 Installing Spark.....	396
14.7.2 Using Spark on CCE.....	399

# 1

# Checklist for Deploying Containerized Applications in the Cloud

---

## Overview

Security, efficiency, stability, and availability are common requirements on all cloud services. To meet these requirements, the system availability, data reliability, and O&M stability must be perfectly coordinated. This checklist describes the check items for deploying containerized applications on the cloud to help you efficiently migrate services to CCE, reducing potential cluster or application exceptions caused by improper use.

## Check Items

**Table 1-1** System availability

Category	Check Item	Type	Impact	FAQ & Example
Cluster	Before creating a cluster, properly plan the node network and container network based on service requirements to allow subsequent service expansion.	Network planning	If the subnet or container CIDR block where the cluster resides is small, the number of available nodes supported by the cluster may be less than required.	<ul style="list-style-type: none"><li>• <a href="#">Network Planning</a></li><li>• <a href="#">Planning CIDR Blocks for a Cluster</a></li><li>• <a href="#">How Do I Set the VPC CIDR Block and Subnet CIDR Block for a CCE Cluster?</a></li></ul>

Category	Check Item	Type	Impact	FAQ & Example
	Before creating a cluster, properly plan CIDR blocks for the related Direct Connect, peering connection, container network, service network, and subnet to avoid IP address conflicts.	Network planning	If CIDR blocks are not properly set and IP address conflicts occur, service access will be affected.	<ul style="list-style-type: none"><li>• <a href="#">Connectivity</a></li><li>• <a href="#">Planning CIDR Blocks for a Cluster</a></li></ul>
	When a cluster is created, the default security group is automatically created and bound to the cluster. You can set custom security group rules based on service requirements.	Deployment	Security groups are key to security isolation. Improper security policy configuration may cause security risks and service connectivity problems.	<ul style="list-style-type: none"><li>• <a href="#">Security Group Overview</a></li><li>• <a href="#">How Do I Harden the VPC Security Group Rules for CCE Cluster Nodes?</a></li></ul>
	Enable the multi-master node mode, and set the number of master nodes to 3 when creating a cluster.	Reliability	After the multi-master node mode is enabled, three master nodes will be created. If a master node is faulty, the cluster can still be available without affecting service functions. In commercial scenarios, it is advised to enable the multi-master node mode.	<a href="#">How Do I Check Whether a Cluster Is an HA Cluster?</a>  Once a cluster is created, the number of master nodes cannot be changed. Exercise caution when setting the number of master nodes.

Category	Check Item	Type	Impact	FAQ & Example
	When creating a cluster, select a proper network model, such as container tunnel network or VPC network.	Deployment	After a cluster is created, the network model cannot be changed. Exercise caution when selecting a network model.	<a href="#">Container Network Model Comparison</a>
Workload	When creating a workload, you need to set the CPU and memory limits to improve service robustness.	Deployment	When multiple applications are deployed on the same node, if the upper and lower resource limits are not set for an application, resource leakage occurs. As a result, resources cannot be allocated to other applications, and the application monitoring information will be inaccurate.	-
	When creating a workload, you can set probes for container health check, including <b>liveness probe</b> and <b>readiness probe</b> .	Reliability	If the health check function is not configured, a pod cannot detect service exceptions or automatically restart the service to restore it. This results in a situation where the pod status is normal but the service in the pod is abnormal.	<ul style="list-style-type: none"><li>• <a href="#">Setting Health Check for a Container</a></li><li>• <a href="#">Enabling ICMP Security Group Rules</a></li></ul>

Category	Check Item	Type	Impact	FAQ & Example
	When creating a workload, select a proper access mode (Service). Currently, the following types of Services are supported: ClusterIP, NodePort, DNAT, and LoadBalancer.	Deployment	Improper Service configuration may cause logic confusion for internal and external access and resource waste.	<ul style="list-style-type: none"><li>• <a href="#">Network Overview</a></li></ul>
	When creating a workload, do not set the number of replicas for a single pod. Set a proper node scheduling policy based on your service requirements.	Reliability	For example, if the number of replicas of a single pod is set, the service will be abnormal when the node or pod is abnormal. To ensure that your pods can be successfully scheduled, ensure that the node has idle resources for container scheduling after you set the scheduling rule.	-

Category	Check Item	Type	Impact	FAQ & Example
	Properly set affinity and anti-affinity.	Reliability	If affinity and anti-affinity are both configured for an application that provides Services externally, Services may fail to be accessed after the application is upgraded or restarted.	<p><b>Scheduling Policy Overview</b></p> <p><b>Negative example:</b> For application A, nodes 1 and 2 are set as affinity nodes, and nodes 3 and 4 are set as anti-affinity nodes. Application A exposes a Service through the ELB, and the ELB listens to node 1 and node 2. When application A is upgraded, it may be scheduled to a node other than nodes 1, 2, 3, and 4, and it cannot be accessed through the Service.</p> <p><b>Cause:</b> Scheduling of application A does not need to meet both affinity and anti-affinity policies. A node will be selected for application A according to either of the policies. In this example, the node selection is based on the anti-affinity scheduling policy.</p>

Category	Check Item	Type	Impact	FAQ & Example
	When creating a workload, set the pre-stop processing command ( <b>Lifecycle &gt; Pre-Stop</b> ) to ensure that the services running in the pods can be completed in advance in the case of application upgrade or pod deletion.	Reliability	If the pre-stop processing command is not configured, the pod will be directly killed and services will be interrupted during application upgrade.	<ul style="list-style-type: none"> <li><a href="#">Setting Container Lifecycle Parameters</a></li> <li><a href="#">When Is Pre-stop Processing Used?</a></li> </ul>

**Table 1-2** Data reliability

Category	Check Item	Type	Impact	FAQ & Example
Container data persistency	Select a proper data volume type based on service requirements.	Reliability	When a node is faulty and cannot be recovered, data in the local disk cannot be recovered. Therefore, you are advised to use cloud storage volumes to ensure data reliability.	<ul style="list-style-type: none"> <li><a href="#">Storage Management</a></li> </ul>
Backup	Back up application data.	Reliability	Data cannot be restored after being lost.	<a href="#">What Storage Classes Does CCE Support?</a> <a href="#">What Are the Differences Between These Storage Classes?</a>

**Table 1-3 O&M reliability**

Category	Check Item	Type	Impact	FAQ & Example
Project	The quotas of ECS, VPC, subnet, EIP, and EVS resources must meet customer requirements.	Deployment	If the quota is insufficient, resources will fail to be created. Specifically, users who have configured auto scaling must have sufficient resource quotas.	<ul style="list-style-type: none"><li>• <a href="#">Which Resource Quotas Should I Pay Attention To When Using CCE?</a></li><li>• <a href="#">Constraints</a></li></ul>
	You are not advised to modify kernel parameters, system configurations, cluster core component versions, security groups, and ELB-related parameters on cluster nodes, or install software that has not been verified.	Deployment	Exceptions may occur on CCE clusters or Kubernetes components on the node, making the node unavailable for application deployment.	<p>For details, see <a href="#">High-Risk Operations and Solutions</a>.</p> <p><b>Negative example:</b></p> <ol style="list-style-type: none"><li>1. The container network is interrupted after the node kernel is upgraded.</li><li>2. The container network is interrupted after an open-source Kubernetes network add-on is installed on a node.</li><li>3. The <code>/var/paas</code> or <code>/mnt/paas/kubernetes</code> directory is deleted from a node, which causes exceptions on the node.</li></ol>

Category	Check Item	Type	Impact	FAQ & Example
	Do not modify information about resources created by CCE, such as security groups and EVS disks. Resources created by CCE are labeled <b>cce</b> .	Deploy me nt	CCE cluster functions may be abnormal.	<b>Negative example:</b> 1. On the ELB console, a user changes the name of the listener created by CCE. 2. On the VPC console, a user modifies the security group created by CCE. 3. On the EVS console, a user deletes or uninstalls data disks mounted to CCE cluster nodes. 4. On the IAM console, a user deletes <b>cce_admin_trust</b> .  All the preceding actions will cause CCE cluster functions to be abnormal.
Proactive O&M	CCE provides multi-dimensional monitoring and alarm reporting functions, and supports basic resource monitoring based on fine-grained metrics by interconnecting with Application Operations Management (AOM). Alarms allow users to locate and rectify faults as soon as possible.	Mo nito ring	If the alarms are not configured, the standard of container cluster performance cannot be established. When an exception occurs, you cannot receive alarms and will need to manually locate the fault.	<a href="#">Monitoring Overview</a>

# 2 Containerization

## 2.1 Containerizing an Enterprise Application (ERP)

### 2.1.1 Solution Overview

This chapter provides CCE best practices to walk you through the application containerization.

#### What Is a Container?

A container is a lightweight high-performance resource isolation mechanism implemented based on the Linux kernel. It is a built-in capability of the operating system (OS) kernel.

CCE is an enterprise-class container service based on open-source Kubernetes. It is a high-performance and high-reliability service through which enterprises can manage containerized applications. CCE supports native Kubernetes applications and tools, allowing you to easily set up a container runtime in the cloud.

#### Why Is a Container Preferred?

- More efficient use of system resources

A container does not require extra costs such as fees for hardware virtualization and those for running a complete OS. Therefore, a container has higher resource usage. Compared with a VM with the same configurations, a container can run more applications.

- Faster startup

A container directly runs on the host kernel and does not need to start a complete OS. Therefore, a container can be started within seconds or even milliseconds, greatly saving the development, testing, and deployment time.

- Consistent runtime environment

A container image provides a complete runtime environment to ensure environment consistency. In this case, problems (for example, some code runs properly on machine A but fails to run on machine B) will not occur.

- Easier application migration, maintenance, and scaling  
A consistent runtime environment makes application migration easier. In addition, the in-use storage and image technologies facilitate the reuse of repeated applications and simplifies the expansion of images based on base images.

## Containerization Modes

The following modes are available for containerizing applications:

- Mode 1: Containerize a single application as a whole. Application code and architecture remain unchanged.
- Mode 2: Separate the components that are frequently upgraded or have high requirements on auto scaling from an application, and then containerize these components.
- Mode 3: Transform an application to microservices and then containerize the microservices one by one.

**Table 2-1** lists the advantages and disadvantages of the three modes.

**Table 2-1** Containerization modes

Containerization Mode	Advantage	Disadvantage
Method 1: Containerize a single application as a whole.	<ul style="list-style-type: none"><li>• Zero modification on services: The application architecture and code require no change.</li><li>• The deployment and upgrade efficiency is improved. Applications can be packed as container images to ensure application environment consistency and improve deployment efficiency.</li><li>• Reduce resource costs: Containers use system resources more efficiently. Compared with a VM with the same configurations, a container can run more applications.</li></ul>	<ul style="list-style-type: none"><li>• Difficult to expand the entire architecture of an application. As the code size increases, code update and maintenance would be complicated.</li><li>• Difficult to launch new functions, languages, frameworks, and technologies.</li></ul>

Containerization Mode	Advantage	Disadvantage
Method 2: Containerize first the application components that are frequently updated or have high requirements on auto scaling.	<ul style="list-style-type: none"><li>• Progressive transformation: Reconstructing the entire architecture involves a heavy workload. This mode containerizes only a part of components, which is easy to accept for customers.</li><li>• Flexible scaling: Application components that have high requirements on auto scaling are containerized. When the application needs to be scaled, you only need to scale the containers, which is flexible and reduces the required system resources.</li><li>• Faster rollout of new features: Application components that are frequently upgraded are containerized. In subsequent upgrades, only these containers need to be upgraded. This shortens the time to market (TTM) of new features.</li></ul>	Need to decouple some services.

Containerization Mode	Advantage	Disadvantage
Method 3: Transform an application to microservices and then containerize the microservices one by one.	<ul style="list-style-type: none"> <li>Independent scaling: After an application is split into microservices, you can independently increase or decrease the number of instances for each microservice.</li> <li>Increased development speed: Microservices are decoupled from one another. Code development of a microservice does not affect other microservices.</li> <li>Security assurance through isolation: For an overall application, if a security vulnerability exists, attackers can use this vulnerability to obtain the permission to all functions of the application. However, in a microservice architecture, if a service is attacked, attackers can only obtain the access permission to this service, but cannot intrude other services.</li> <li>Breakdown isolation: If one microservice breaks down, other microservices can still run properly.</li> </ul>	Need to transform the application to microservices, which involves a large number of changes.

**Mode 1** is used as an example in this tutorial to illustrate how to containerize an enterprise resource planning (ERP) system.

## 2.1.2 Resource and Cost Planning

### NOTICE

The fees listed here are estimates. The actual fees will be displayed on the Huawei Cloud console.

The required resources are as follows:

**Table 2-2** Resource and cost planning

Resource	Description	Quantity	Estimated Fee (USD)
Elastic Cloud Server (ECS)	<ul style="list-style-type: none"><li>Minimum ECS specifications: 4 vCPUs, 8 GB memory, Ubuntu 16.04</li><li>EIP specification: billed by bandwidth, 5 Mbit/s</li><li>Pay-per-use recommended</li></ul>	1	0.124/hour
Cloud Container Engine (CCE)	<ul style="list-style-type: none"><li>CCE cluster version: v1.21</li><li>Minimum node specifications: 4 vCPUs, 8 GB memory, EulerOS 2.9</li><li>Pay-per-use recommended</li></ul>	1	0.55/hour
Elastic Volume Service (EVS)	<ul style="list-style-type: none"><li>EVS disk specification: 100 GB</li><li>Pay-per-use recommended</li></ul>	1	0.01/hour

## 2.1.3 Procedure

### 2.1.3.1 Containerizing an Entire Application

This tutorial describes how to containerize an ERP system by migrating it from a VM to CCE.

No recoding or re-architecting is required. You only need to pack the entire application into a container image and deploy the container image on CCE.

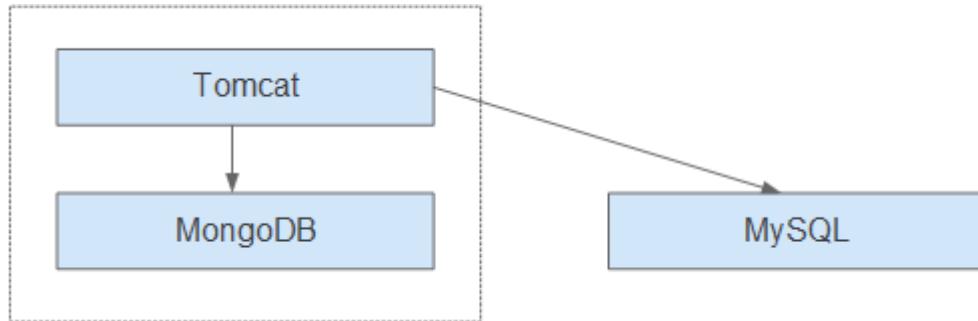
#### Introduction

In this example, the **enterprise management application** is developed by enterprise A. This application is provided for third-party enterprises for use, and enterprise A is responsible for application maintenance.

When a third-party enterprise needs to use this application, a suit of **Tomcat application** and **MongoDB database** must be deployed for the third-party

enterprise. The MySQL database, used to store data of third-party enterprises, is provided by enterprise A.

**Figure 2-1** Application architecture



As shown in **Figure 2-1**, the application is a standard Tomcat application, and its backend interconnects with MongoDB and MySQL databases. For this type of applications, there is no need to split its architecture. The entire application is packed as an image, and the mongoDB database is deployed in the same image as the Tomcat application. In this way, the application can be deployed or upgraded through the image.

- Interconnecting with the MongoDB database for storing user files.
- Interconnecting with the MySQL database for storing third-party enterprise data. The MySQL database is an external cloud database.

## Benefits

In this example, the application was deployed on a VM. During application deployment and upgrade, a series of problems is encountered, but application containerization has solved these problems.

By using containers, you can easily pack application code, configurations, and dependencies and convert them into easy-to-use building blocks. This achieves the environmental consistency and version management, as well as improves the development and operation efficiency. Containers ensure quick, reliable, and consistent deployment of applications and prevent applications from being affected by deployment environment.

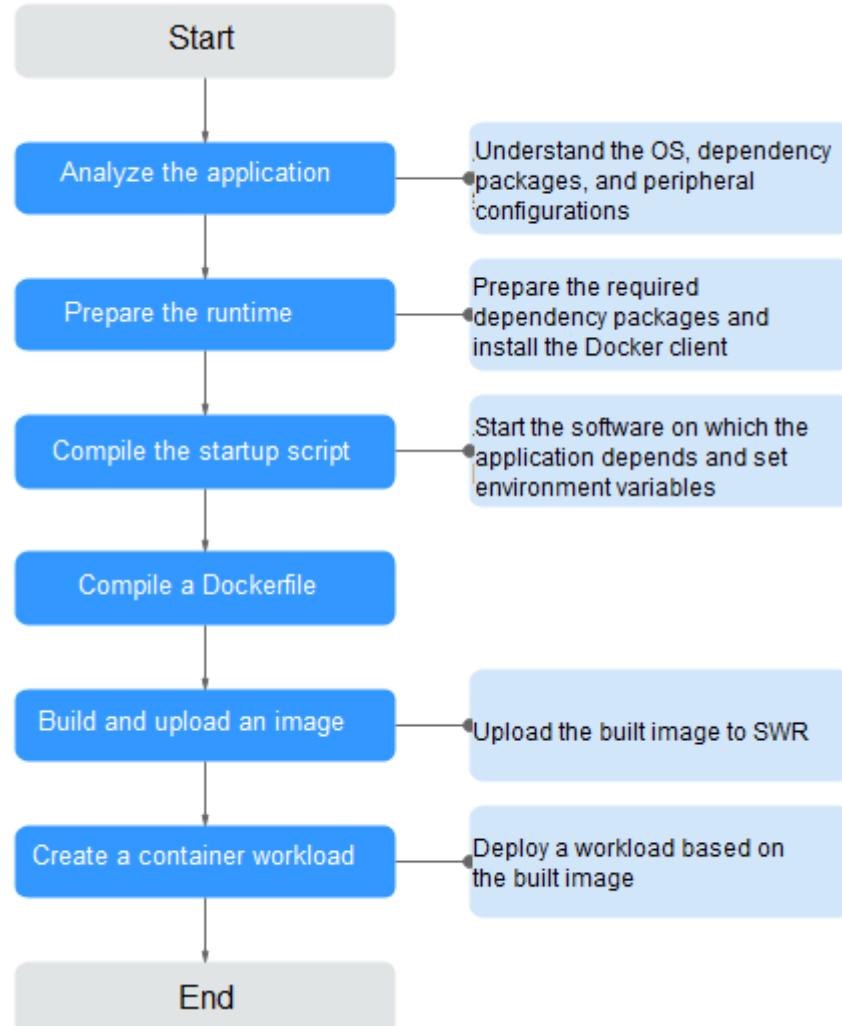
**Table 2-3** Comparison between the tow deployment modes

Item	Before: Application Deployment on VM	After: Application Deployment Using Containers
Deployment	High deployment cost. A VM is required for deploying a system for a customer.	More than 50% cost reduced. Container services achieve multi-tenant isolation, which allows you to deploy systems for different enterprises on the same VM.

Item	Before: Application Deployment on VM	After: Application Deployment Using Containers
Upgrade	Low upgrade efficiency. During version upgrades, you need to log in to VMs one by one and manually configure the upgrades, which is inefficient and error-prone.	Per-second level upgrade. Version upgrades can be completed within seconds by replacing the image tag. In addition, CCE provides rolling updates, ensuring zero service downtime during upgrades.
Operation and maintenance (O&M)	High O&M cost. As the number of applications deployed for customer grows, the number of VMs that need to be maintained increases accordingly, which requires a large sum of maintenance cost.	Automatic O&M Enterprises can focus on service development without paying attention to VM maintenance.

### 2.1.3.2 Containerization Process

The following figure illustrates the process of containerizing an application.

**Figure 2-2** Process of containerizing an application

### 2.1.3.3 Analyzing the Application

Before containerizing an application, you need to analyze the running environment and dependencies of the application, and get familiar with the application deployment mode. For details, see [Table 2-4](#).

**Table 2-4** Application environment

Item	Sub-category	Description
Runtime environment	OS	OS that the application runs on, such as CentOS or Ubuntu. In this example, the application runs on CentOS 7.1.

Item	Sub-category	Description
	Runtime environment	<p>The Java application requires Java Development Kit (JDK), the Go language requires GoLang, the web application requires Tomcat environment, and the corresponding version number needs to be confirmed.</p> <p>In this example, the web application of the Tomcat type is used. This application requires the runtime environment of Tomcat 7.0, and Tomcat requires JDK 1.8.</p>
	Dependency package	<p>Understand required dependency packages, such as OpenSSL and other system software, and their version numbers.</p> <p>In this example, no dependency package is required.</p>
Deployment mode	Peripheral configurations	<p>MongoDB database: In this example, the MongoDB database and Tomcat application are deployed on the same server. Therefore, their configurations can be fixed and there is no need to extract their configurations.</p>
		<p>External services with which the application needs to interconnect, such as databases and file systems.</p> <p>These configurations need to be manually configured each time you deploy an application on a VM. However, through containerized deployment, environment variables can be injected into a container, facilitating deployment.</p> <p>In this example, the application needs to interconnect with the MySQL database. You need to obtain the database configuration file. The server address, database name, database login username, and database login password are injected through environment variables.</p> <pre>url=jdbc:mysql://Server address/Database name      #Database connection URL username=****                                     #Username for logging in to the database password=****                                     #Password for logging in to the database</pre>
	Application configurations	<p>You need to sort out the configuration parameters, such as configurations that need to be modified frequently and those remain unchanged during the running of the application.</p> <p>In this example, no application configurations need to be extracted.</p> <p><b>NOTE</b></p> <p>To avoid frequent image replacement, you are advised to classify configurations of the application.</p> <ul style="list-style-type: none"> <li>For the configurations (such as peripheral interconnection information and log levels) that are frequently changed, you are advised to configure them as environment variables.</li> <li>For the configurations that remain unchanged, directly write them into images.</li> </ul>

### 2.1.3.4 Preparing the Application Runtime

After application analysis, you have gained the understanding of the OS and runtime required for running the application. You need to make the following preparations:

- **Installing Docker:** During application containerization, you need to build a container image. To do so, you have to prepare a PC and install Docker on it.
- **Obtaining the base image tag:** Determine the base image based on the OS on which the application runs. In this example, the application runs on CentOS 7.1 and the base image can be obtained from an open-source image repository.
- **Obtaining the runtime:** Obtain the runtime of the application and the MongoDB database with which the application interconnects.

#### Installing Docker

Docker is compatible with almost all operating systems. Select a Docker version that best suits your needs.

##### NOTE

SWR uses Docker 1.11.2 or later to upload images.

You are advised to install Docker and build images as user **root**. Obtain the password of user **root** of the host where Docker is to be installed in advance.

**Step 1** Log in as user **root** to the device on which Docker is about to be installed.

**Step 2** Run the following commands to quickly install Docker on the device running Linux:

```
curl -fsSL get.docker.com -o get-docker.sh
```

```
sh get-docker.sh
```

**Step 3** Run the following command to query the Docker version:

```
docker version
```

```
Client:  
Version: 17.12.0-ce  
API Version: 1.35  
...
```

**Version** indicates the version number.

----End

#### Obtaining the Base Image Tag

Determine the base image based on the OS on which the application runs. In this example, the application runs on CentOS 7.1 and the base image can be obtained from an open-source image repository.

##### NOTE

Search for the image tag based on the OS on which the application runs.

**Step 1** Visit the Docker website.

**Step 2** Search for CentOS. The image corresponding to CentOS 7.1 is **centos7.1.1503**. You need to use this image name when compiling the Dockerfile.

**Figure 2-3** Obtaining the CentOS version



----End

## Obtaining the Runtime

In this example, the web application of the Tomcat type is used. This application requires the runtime of Tomcat 7.0, and Tomcat requires JDK 1.8. In addition, the application must interconnect with the MongoDB database in advance.

### NOTE

Download the environment required by the application.

**Step 1** Download Tomcat, JDK, and MongoDB installation packages of the specific versions.

1. Download JDK 1.8.  
Download address: <https://www.oracle.com/java/technologies/jdk8-downloads.html>.
2. Download Tomcat 7.0 from <http://archive.apache.org/dist/tomcat/tomcat-7/v7.0.82/bin/apache-tomcat-7.0.82.tar.gz>.
3. Download MongoDB 3.2 from [https://fastdl.mongodb.org/linux/mongodb-linux-x86\\_64-rhel70-3.2.9.tgz](https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-3.2.9.tgz).

**Step 2** Log in as user **root** to the device running Docker.

**Step 3** Run the following commands to create the directory where the application is to be stored: For example, set the directory to **apptest**.

```
mkdir apptest
```

```
cd apptest
```

**Step 4** Use Xshell to save the downloaded dependency files to the **apptest** directory.

**Step 5** Run the following commands to decompress the dependency files:

```
tar -zxf apache-tomcat-7.0.82.tar.gz
```

```
tar -zxf jdk-8u151-linux-x64.tar.gz
```

```
tar -zxf mongodb-linux-x86_64-rhel70-3.2.9.tgz
```

**Step 6** Save the enterprise application (for example, **apptest.war**) in the **webapps/apptest** directory of the Tomcat runtime environment.



**apptest.war** is used as an example only. Use your own application for actual configuration.

```
mkdir -p apache-tomcat-7.0.82/webapps/apptest
```

```
cp apptest.war apache-tomcat-7.0.82/webapps/apptest
```

```
cd apache-tomcat-7.0.82/webapps/apptest
```

```
./../..../jdk1.8.0_151/bin/jar -xf apptest.war
```

```
rm -rf apptest.war
```

----End

### 2.1.3.5 Compiling a Startup Script

During application containerization, you need to prepare a startup script. The method of compiling this script is the same as that of compiling a shell script. The startup script is used to:

- Start up the software on which the application depends.
- Set the configurations that need to be changed as the environment variables.



Startup scripts vary according to applications. You need to compile the script based on your service requirements.

## Procedure

**Step 1** Log in as user **root** to the device running Docker.

**Step 2** Run the following commands to create the directory where the application is to be stored:

```
mkdir apptest
```

```
cd apptest
```

**Step 3** Compile a script file. The name and content of the script file vary according to applications. You need to compile the script file based on your application. The following example is only for your reference.

```
vi start_tomcat_and_mongo.sh
```

```
#!/bin/bash
source /etc/profile
./usr/local/mongodb/bin/mongod --dbpath=/usr/local/mongodb/data --logpath=/usr/local/mongodb/logs
--port=27017 -fork
sed -i "s|mysql://.*|awcp_crmtile|mysql://$MYSQL_URL/$MYSQL_DB|g" /root/apache-tomcat-7.0.82/
webapps/awcp/WEB-INF/classes/conf/jdbc.properties
sed -i "$username=.*$username=$MYSQL_USER|g" /root/apache-tomcat-7.0.82/webapps/awcp/WEB-INF/
classes/conf/jdbc.properties
sed -i "$password=.*$password=$MYSQL_PASSWORD|g" /root/apache-tomcat-7.0.82/webapps/awcp/WEB-
INF/classes/conf/jdbc.properties
bash /root/apache-tomcat-7.0.82/bin/catalina.sh run
```

Script description:

- source /etc/profile: Load system environment variables.
- ./usr/local/mongodb/bin/mongod --dbpath=/usr/local/mongodb/data --logpath=/usr/local/mongodb/logs --port=27017 -fork: Start the MongoDB database. In this example, the data storage directory is **/usr/local/mongodb/data**, and this directory must correspond to the storage mount path specified when creating a container.
- sed -i "s|...": These three script commands indicate that the contents related to the MySQL database in the environment variables are written into the configuration file when Docker is started.
- bash /root/apache-tomcat-7.0.82/bin/catalina.sh run: Start Tomcat at the end.

----End

### 2.1.3.6 Compiling the Dockerfile

An image is the basis of a container. A container runs based on the content defined in the image. An image has multiple layers. Each layer includes the modifications made based on the previous layer.

Generally, Dockerfiles are used to customize images. Dockerfile is a text file and contains various instructions. Each instruction is used to build an image layer. That is, each instruction describes how to build an image layer.

This section describes how to compile a Dockerfile file.

 NOTE

Dockerfiles vary according to applications. Dockerfiles need to be compiled based on actual service requirements.

For details on how to write a quality Dockerfile, see [Writing a Quality Dockerfile](#).

## Procedure

**Step 1** Log in as the **root** user to the device running Docker.

**Step 2** Write a Dockerfile.

### vi Dockerfile

The following provides an example Dockerfile:

```
# CentOS:7.1.1503 is used as the base image.  
FROM centos:7.1.1503  
# Create a folder to store data and dependency files. You are advised to write multiple commands into one  
line to reduce the image size.  
RUN mkdir -p /usr/local/mongodb/data \  
    && mkdir -p /usr/local/mongodb/bin \  
    && mkdir -p /root/apache-tomcat-7.0.82 \  
    && mkdir -p /root/jdk1.8.0_151  
  
# Copy the files in the apache-tomcat-7.0.82 directory to the container path.  
COPY ./apache-tomcat-7.0.82 /root/apache-tomcat-7.0.82  
# Copy the files in the jdk1.8.0_151 directory to the container path.  
COPY ./jdk1.8.0_151 /root/jdk1.8.0_151  
# Copy the files in the mongodb-linux-x86_64-rhel70-3.2.9 directory to the container path.  
COPY ./mongodb-linux-x86_64-rhel70-3.2.9/bin /usr/local/mongodb/bin  
# Copy start_tomcat_and_mongo.sh to the /root directory of the container.  
COPY ./start_tomcat_and_mongo.sh /root/  
  
# Enter Java environment variables.  
RUN chown root:root -R /root \  
    && echo "JAVA_HOME=/root/jdk1.8.0_151" >> /etc/profile \  
    && echo "PATH=$JAVA_HOME/bin:$PATH" >> /etc/profile \  
    && echo "CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar" >> /etc/profile \  
    && chmod +x /root \  
    && chmod +x /root/start_tomcat_and_mongo.sh  
  
# When the container is started, commands in start_tomcat_and_mongo.sh are automatically run. The file  
can be one or more commands, or a script.  
ENTRYPOINT ["/root/start_tomcat_and_mongo.sh"]
```

In the preceding information:

- **FROM** statement: indicates that **centos:7.1.1503** is used as the base image.
- **Run** statement: indicates that a shell command is executed in the container.
- **Copy** statement: indicates that files in the local computer are copied to the container.
- **ENTRYPOINT** statement: indicates the commands that are run after the container is started.

----End

### 2.1.3.7 Building and Uploading an Image

This section describes how to build an entire application into a Docker image. After building an image, you can use the image to deploy and upgrade the application. This reduces manual configuration and improves efficiency.

 NOTE

When building an image, ensure that files used to build the image are stored in the same directory.

## Required Cloud Services

SoftWare Repository for Container (SWR) provides easy, secure, and reliable management over container images throughout their lifecycle, facilitating the deployment of containerized services.

## Basic Concepts

- **Image:** A Docker image is a special file system that includes everything needed to run containers: programs, libraries, resources, settings, and so on. It also includes corresponding configuration parameters (such as anonymous volumes, environment variables, and users) required within a container runtime. An image does not contain any dynamic data, and its content remains unchanged after being built.
- **Container:** Images become containers at runtime, that is, containers are created from images. A container can be created, started, stopped, deleted, or suspended.

## Procedure

**Step 1** Log in as the **root** user to the device running Docker.

**Step 2** Enter the **apptest** directory.

```
cd apptest
```

```
ll
```

Ensure that files used to build the image are stored in the same directory.

```
root@ecs-aos:~/apptest# ll
total 264456
drwxr-xr-x 5 root root    4096 Jan  2 19:59 .
drwxr----- 6 root root    4096 Jan  2 19:59 ..
drwxr-xr-x 9 root root    4096 Jan  2 19:55 apache-tomcat-7.0.82/
-rw-r--r-- 1 root root 8997403 Jan  2 19:52 apache-tomcat-7.0.82.tar.gz
-rw-r--r-- 1 root root    599 Jan  2 19:59 Dockerfile
drwxr-xr-x 8 uucp   143  4096 Sep  6 10:32 jdk1.8.0_151/
-rw-r--r-- 1 root root 189736377 Jan  2 19:54 jdk-8u151-linux-x64.tar.gz
drwxr-xr-x 3 root root    4096 Jan  2 19:55 mongodb-linux-x86_64-rhel70-3.2.9/
-rw-r--r-- 1 root root 72035914 Jan  2 19:53 mongodb-linux-x86_64-rhel70-3.2.9.tgz
-rw-r--r-- 1 root root    597 Jan  2 19:58 [start tomcat and mongo.sh]
```

**Step 3** Build an image.

```
docker build -t apptest .
```

**Step 4** Upload the image to SWR. For details, see [Uploading an Image Through a Container Engine Client](#).

----End

### 2.1.3.8 Creating a Container Workload

This section describes how to deploy a workload on CCE. When using CCE for the first time, create an initial cluster and add a node into the cluster.

#### NOTE

Containerized workloads are deployed in a similar way. The difference lies in:

- Whether environment variables need to be set.
- Whether cloud storage is used.

### Required Cloud Services

- Cloud Container Engine (CCE): a highly reliable and high-performance service that allows enterprises to manage containerized applications. With support for Kubernetes-native applications and tools, CCE makes it simple to set up an environment for running containers in the cloud.
- Elastic Cloud Server (ECS): a scalable and on-demand cloud server. It helps you to efficiently set up reliable, secure, and flexible application environments, ensuring stable service running and improving O&M efficiency.
- Virtual Private Cloud (VPC): an isolated and private virtual network environment that users apply for in the cloud. You can configure the IP address ranges, subnets, and security groups, as well as assign elastic IP addresses and allocate bandwidth in a VPC.

### Basic Concepts

- A cluster is a collection of computing resources, including a group of node resources. A container runs on a node. Before creating a containerized application, you must have an available cluster.
- A node is a virtual or physical machine that provides computing resources. You must have sufficient node resources to ensure successful operations such as creating applications.
- A workload indicates a group of container pods running on CCE. CCE supports third-party application hosting and provides the full lifecycle (from deployment to O&M) management for applications. This section describes how to use a container image to create a workload.

### Procedure

**Step 1** Prepare the environment as described in [Table 2-5](#).

**Table 2-5** Preparing the environment

No.	Item	Procedure
1	Creating a VPC	<p>Create a VPC before you create a cluster. A VPC provides an isolated, configurable, and manageable virtual network environment for CCE clusters.</p> <p>If you have a VPC already, skip to the next task.</p> <ol style="list-style-type: none"><li>1. Log in to the management console.</li><li>2. In the service list, choose <b>Networking &gt; Virtual Private Cloud</b>.</li><li>3. On the <b>Dashboard</b> page, click <b>Create VPC</b>.</li><li>4. Follow the instructions to create a VPC. Retain default settings for parameters unless otherwise specified.</li></ol>
2	Creating a key pair	<p>Create a key pair before you create a containerized application. Key pairs are used for identity authentication during remote login to a node. If you have a key pair already, skip this task.</p> <ol style="list-style-type: none"><li>1. Log in to the management console.</li><li>2. In the service list, choose <b>Security and Compliance &gt; Data Encryption Workshop</b>.</li><li>3. In the navigation pane, choose <b>Key Pair Service</b>. On the <b>Private Key Pairs</b> tab page, click <b>Create Key Pair</b>.</li><li>4. Enter a key pair name, select <b>I agree to have the private key managed on the cloud</b> and <b>I have read and agree to the Key Pair Service Disclaimer</b>, and click <b>OK</b>.</li><li>5. In the dialog box displayed, click <b>OK</b>.</li></ol> <p>View and save the key pair. For security purposes, a key pair can be downloaded only once. Keep it secure to ensure successful login.</p>

**Step 2** Create a cluster and a node.

1. Log in to the CCE console, choose **Clusters**, and click **Buy** next to **CCE cluster**. Configure cluster parameters and select the VPC created in **Step 1**. For details, see [Buying a CCE Cluster](#).
2. Buy a node and select the key pair created in **Step 1** as the login mode. For details, see [Creating a Node](#).

**Step 3** Deploy a workload on CCE.

1. Log in to the CCE console, click the created cluster, choose **Workloads** in the navigation pane, and click **Create Workload** in the upper right corner.
2. Set the following parameters, and retain the default settings for other parameters:
  - **Workload Name:** Set it to **apptest**.
  - **Pods:** Set it to **1**.

3. In the **Container Settings** area, select the image uploaded in [Building and Uploading an Image](#).
4. In the **Container Settings** area, choose **Environment Variables** and add environment variables for interconnecting with the MySQL database. These environment variables are set in [Compiling a Startup Script](#).

 **NOTE**

In this example, interconnection with the MySQL database is implemented through configuring the environment variables. Determine whether to use environment variables based on your service requirements.

**Table 2-6** Configuring environment variables

Variable Name	Variable Value/Variable Reference
MYSQL_DB	Database name.
MYSQL_URL	IP address and port number of the database.
MYSQL_USER	Database username.
MYSQL_PASSWORD	Database user password.

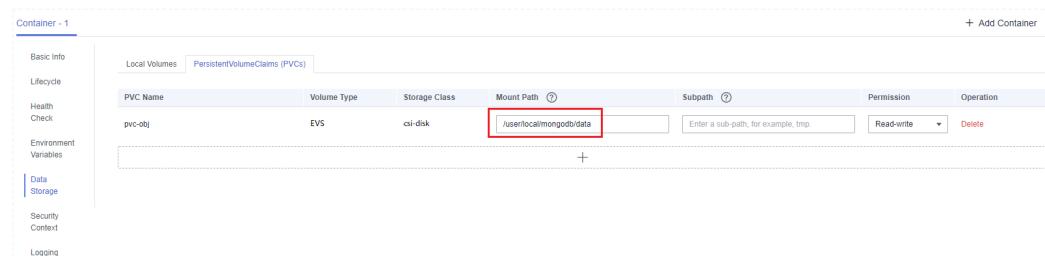
5. In the **Container Settings** area, choose **Data Storage** and configure cloud storage for persistent data storage.

 **NOTE**

In this example, the MongoDB database is used and persistent data storage is also needed, so you need to configure cloud storage. Determine whether to use cloud storage based on your service requirements.

In this example, set **Mount Path** as the path to the MongoDB storage directory that you configured in the Docker startup script. For details, see [Compiling a Startup Script](#).

**Figure 2-4** Configuring cloud storage



6. In the **Service Settings** area, click  to add a service, configure workload access parameters, and click **OK**.

 **NOTE**

In this example, the application will be accessible from public networks by using an elastic IP address.

- **Service Name:** name of the application that can be accessed externally. In this example, this parameter is set to **apptest**.

- **Service Type:** In this example, select **NodePort**.
- **Service Affinity**
  - **Cluster-level:** The IP addresses and access ports of all nodes in a cluster can be used to access the workload associated with the Service. Service access will cause performance loss due to route redirection, and the source IP address of the client cannot be obtained.
  - **Node-level:** Only the IP address and access port of the node where the workload is located can be used to access the workload associated with the Service. Service access will not cause performance loss due to route redirection, and the source IP address of the client can be obtained.
- **Port**
  - **Protocol:** Set it to **TCP**.
  - **Service Port:** port for accessing the Service.
  - **Container Port:** port that the application will listen on the container. In this example, this parameter is set to **8080**.
  - **Node Port:** Set it to **Auto**. The system automatically opens a real port on all nodes in the current cluster and then maps the port number to the container port.

7. Click **Create Workload**.

After the workload is created, you can view the running workload in the workload list.

----End

## Verifying a Workload

After a workload is created, you can access the workload to check whether the deployment is successful.

In the preceding configuration, the NodePort mode is selected to access the workload by using **IP address:Port number**. If the access is successful, the workload is successfully deployed.

You can obtain the access mode from the **Access Mode** tab page on the workload details page.

## 2.2 Containerizing a Game Application (WOW)

### 2.2.1 Solution Overview

This document provides best practices for CCE to guide you through containerizing game applications.

The features of the gaming industry, especially mobile games that are popular nowadays, are **short, flat, and fast**.

- **Short:** Games, especially mobile games, feature a short lifecycle, which generally lasts for only one year.
- **Flat:** Game development relies on a flat background architecture. For most games, one server is deployed for one region. One machine and one database can serve multiple game servers.
- **Fast:** Game players generally increase in an explosive way. However, a game server supports only a limited number of users. A new server must be deployed if the number of users reaches the upper limit. Due to a short lifecycle, it is necessary that a game can be developed and rolled out in a short time.

## What Is a Container?

A container is a lightweight OS-layer virtualization technology. It allows user space in an OS to be divided into several independent units running in the kernel, each of which is independent from each other. Such independent space is called a container.

With the development of virtualization technologies, one physical machine can be virtualized into multiple VMs, but virtualizing an independent OS incurs system loss, limiting the number of target VMs. In contrast, **one machine can run dozens or even hundreds of containers with basically no performance loss. In addition, starting a container is as simple as starting a process, which can be completed in seconds.** In gaming scenarios, the container technology is extremely advantageous.

## Why Is a Container Preferred?

- **A traditional game background architecture has the following disadvantages:** One machine runs a large number of game servers at the same time. When the machine breaks down, users are widely affected.  
**Container solution:** One of the most important advantages of the container technology lies in its lightweight, which supports virtualization of an independent system operating environment with a fine granularity. This enables a physical server, or a cloud server, to run hundreds of thousands of independent containers at the same time. **Each kind of service logic in a game, such as marching, fighting, and chatting control logic, can run in an independent container. This series of containers constitutes an independent game world.**

In addition, the resource usage of these containers can be properly planned based on service types, so that the resources are isolated for different containers.

Moreover, after containerization, a machine failure may affect only some service logic in some game servers. For example, when the machine running the container that controls marching tasks is faulty, frame freezing occurs instantaneously but the marching tasks recover immediately. Using monitoring methods, some important service logic can be run on multiple backup containers at the same time, and the service logic can be quickly switched to the backup containers in case of unavailability.

- **Games feature a short lifecycle and require quick development and rollout.**

**Container solution:** The core concepts of Docker containers are Build, Ship, and Run, covering the entire process from development to deployment. After the development is complete, containers are packaged into container images and stored in a repository before testing. After the testing is complete, the container images are stored in the repository again and finally deployed in the production environment. The three phases are smoothly connected, avoiding the workload of setting up a complex running environment. In this way, games can be quickly developed and rolled out.

- **The number of game players dramatically fluctuates, requiring auto scaling of game applications in a short time.**

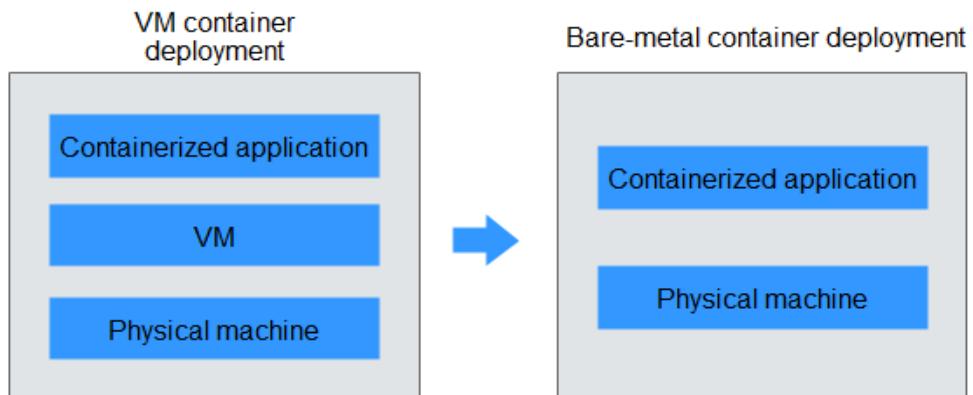
**Container solution:** A container can be started in seconds. When the number of game players dramatically increases, containers can be started in seconds to ensure service stability.

## Core Advantages of CCE

- **Advanced bare-metal container service, improving the gaming service performance by more than 200%**

The gaming industry places stringent requirements on network **performance** and computing capabilities of servers. CCE supports a bare-metal container service, as shown in **Figure 2-5**. Games can be deployed based on Bare Metal Server (BMS), and containers can directly run on physical machines. By using containers, excellent performance can be achieved without any performance loss caused by virtualization, thereby improving the gaming service performance by more than 200%.

**Figure 2-5** BMS container service



- **Auto scaling of containers in seconds, saving a lot of resource costs**  
Traffic unpredictability has become a "New Normal" of game applications. CCE supports auto scaling in seconds to ensure high service stability and improve user experience. In addition, reserved resources are reduced, saving investments by millions of dollars. Based on features of games, CCE provides flexible auto scaling policies, which can be selected and combined for use as required.

**Table 2-7** Flexible auto scaling policies

Recommended Auto Scaling Policy	Description
<ul style="list-style-type: none"><li>The number of game players dramatically fluctuates every day, so the <b>Periodic Policy</b> is recommended.</li></ul>	<p>For example, the number of players of a game reaches the peak in the afternoon and evening on a given day.</p> <p>You are advised to use the <b>Periodic Policy</b>. For example, 100 pods are added at 13:00 every day from January 1, 2018 to January 1, 2019.</p>
<ul style="list-style-type: none"><li>For new games, the <b>Metric-based Policy</b> is recommended.</li></ul>	<p>After a new game is released, it is uncertain whether how many players will play this game. It is difficult to reserve a proper number of machines based on existing experience.</p> <p>You are advised to use the <b>Metric-based Policy</b>. For example, if the CPU or memory usage exceeds 70%, one pod is added. If the CPU or memory usage is less than 40%, one pod is reduced.</p>
<ul style="list-style-type: none"><li>For various in-game events, the <b>Scheduled Policy</b> is recommended.</li></ul>	<p>In-game events are frequently held. You are advised to configure the <b>Scheduled Policy</b> before an event begins. For example, 100 pods are added at 12:00 on August 8.</p>

- Rolling upgrade policy, causing no service interruption and ensuring lag-free experience of game players**

Game requirements grow rapidly and versions change frequently. The upgrade efficiency and user experience during the upgrade are crucial for gaming services.

CCE provides a rolling upgrade policy to separately update pods one by one instead of updating all pods at the same time. This ensures that services are not interrupted during the upgrade.

- Support for stateful applications, solving the problem of containerizing game applications**

Game servers constitute an independent game world. In this world, data of players needs to be continuously updated and stored. The prerequisite for containerizing game applications is to ensure data storage. CCE supports stateful containerized applications (applications that store data or statuses during the running), and leverages high-availability volumes by means of storage capabilities such as Elastic Volume Service (EVS) and Scalable File Service (SFS) of Huawei Cloud. This solves the problem of containerizing game applications.

## 2.2.2 Resource and Cost Planning

### NOTICE

The fees listed here are estimates. The actual fees will be displayed on the Huawei Cloud console.

The required resources are as follows:

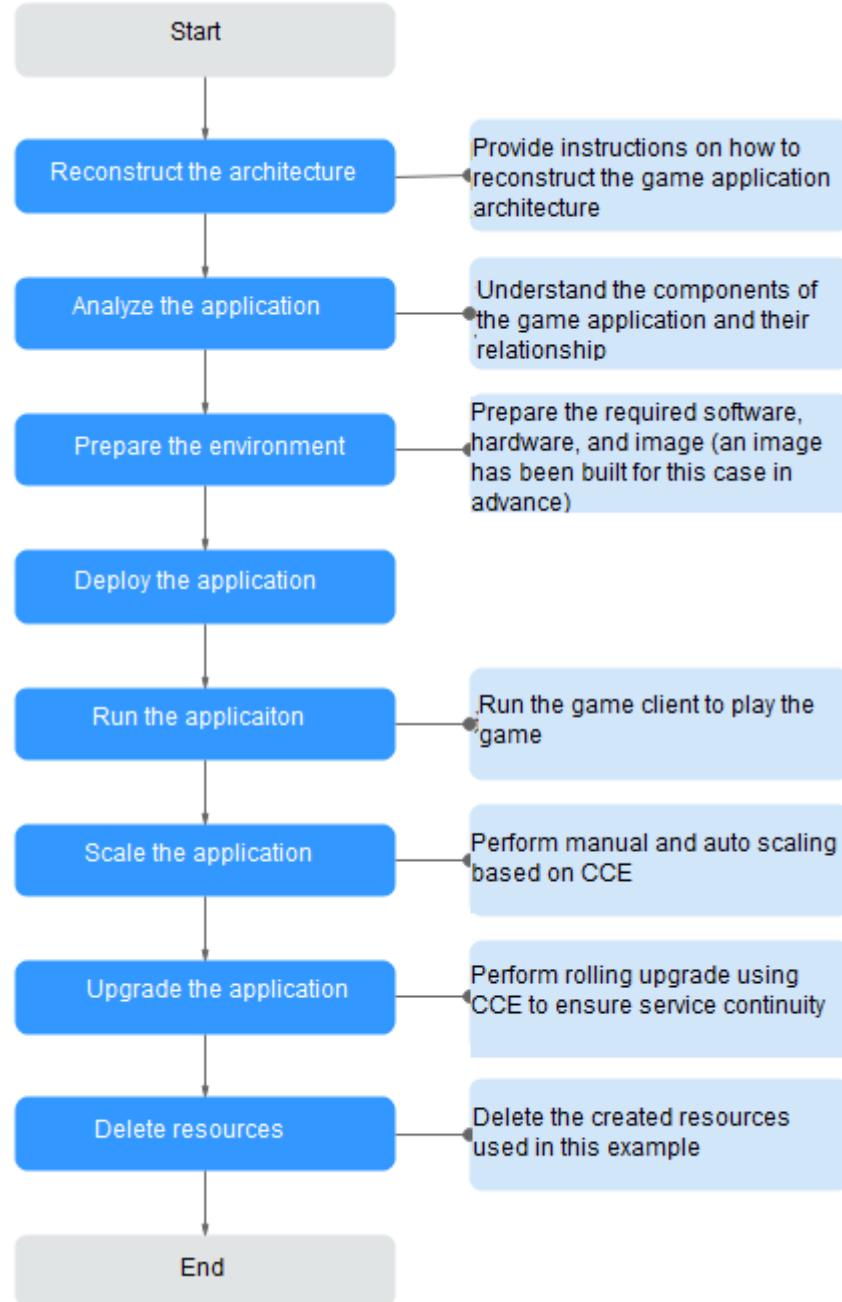
**Table 2-8** Resource and cost planning

Resource	Description	Quantity	Estimated Fee (USD)
Elastic Cloud Server (ECS)	<ul style="list-style-type: none"><li>Minimum ECS specifications: 4 vCPUs, 8 GB memory, Ubuntu 16.04</li><li>EIP specification: billed by bandwidth, 5 Mbit/s</li><li>Pay-per-use recommended</li></ul>	1	0.124/hour
Cloud Container Engine (CCE)	<ul style="list-style-type: none"><li>CCE cluster version: v1.21</li><li>Minimum node specifications: 4 vCPUs, 8 GB memory, EulerOS 2.9</li><li>Pay-per-use recommended</li></ul>	1	0.55/hour

## 2.2.3 Procedure

### 2.2.3.1 Deployment Process

This tutorial uses a game as an example to describe how to deploy a game application on CCE and demonstrate how to scale and upgrade this application.

**Figure 2-6** Deployment process

### 2.2.3.2 Preparation: Reconstructing the Game Application Architecture

This section describes reconstruction on the architecture of a game application into microservices before the game application is containerized.

Only overall reconstruction suggestions are provided in this section, and the reconstruction process is not described in detail.

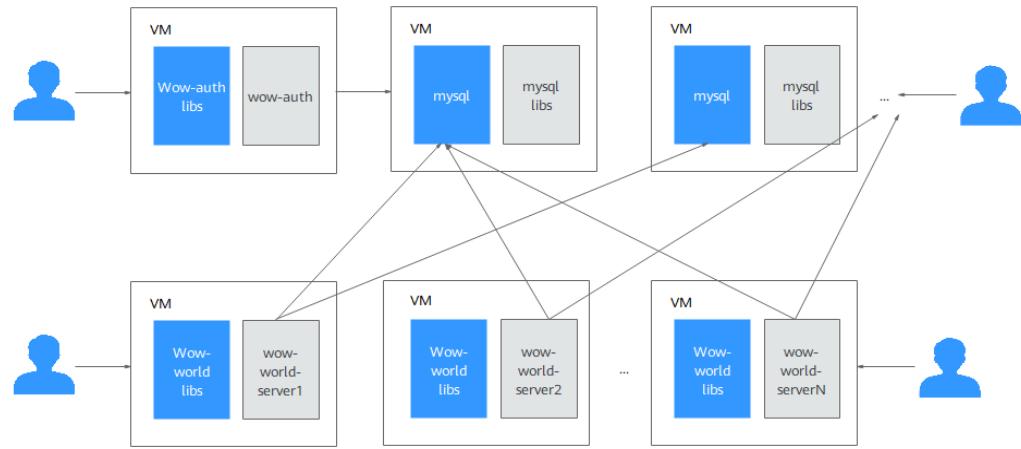
#### NOTE

This section provides suggestions on reconstruction of the game application architecture. You do not need to perform any operation. To perform operations, go to [Analyzing the Game Application](#).

## Suggestions on Containerization Reconstruction

The original game application architecture is as follows:

Figure 2-7 Original architecture

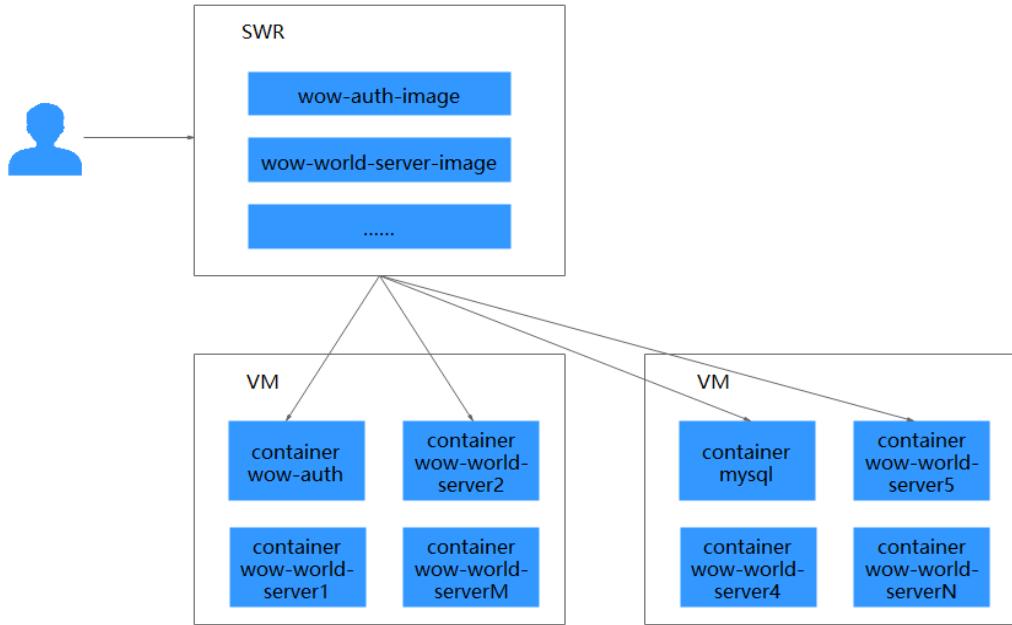


As shown in [Figure 2-7](#),

- The game application consists of three components: wow-auth login authentication system, wow-world game server, and MySQL database.
- The wow-auth login authentication system and its dependency are installed on one VM, the MySQL database and its dependency are installed on two or more VMs, and the wow-world game server and its dependency are installed on three or more VMs. In the current architecture, if there are multiple game servers, they must be installed on multiple VMs. In this case, **multiple VMs must be prepared, and independence packages required by different components must be installed on each VM, featuring heavy workload.**
- This architecture causes poor scalability and difficult scaling, and brings high costs in maintenance. A new VM must be installed before a game server is added. In addition, it is a difficult job to maintain multiple VMs.
- Difficult upgrade: Upgrading the VMs require you to upgrade the configurations of these VMs one by one, which is time-consuming and error-prone.

You are advised to reconstruct the original architecture as follows:

**Figure 2-8** New architecture



As shown in **Figure 2-8**, the three components (including the wow-auth login authentication system, wow-world game server, and MySQL database) of the game application are respectively containerized and deployed on VMs. The new feature has the following advantages:

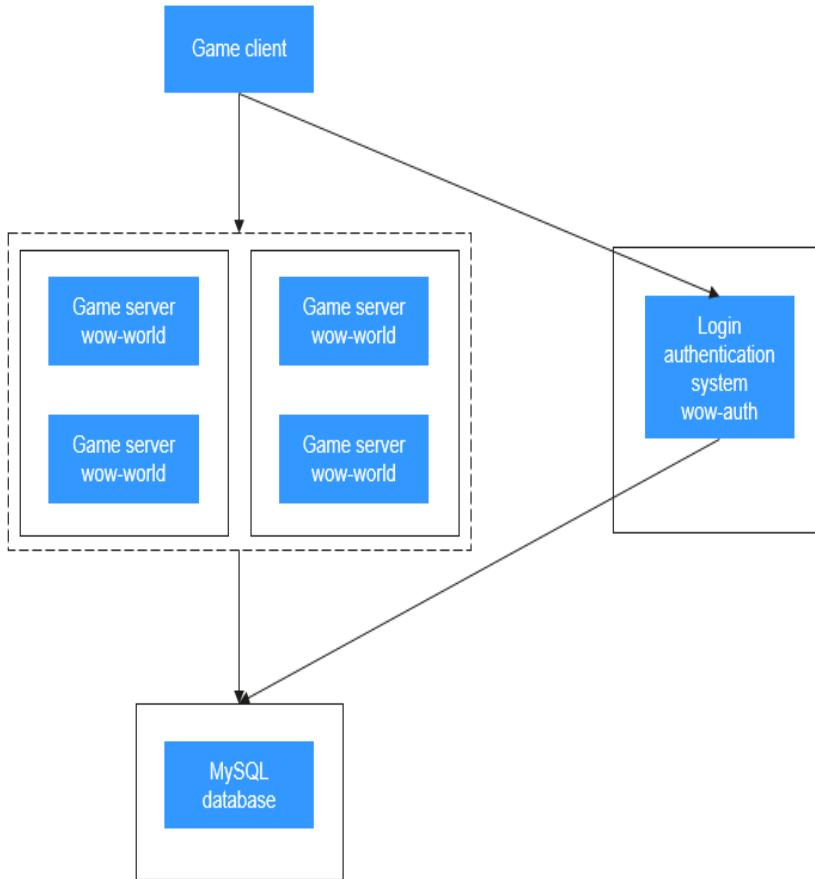
**NOTE**

In this tutorial, the MySQL database is used. During actual commercial use, select a proper database based on your service requirements.

- Easy deployment: The three components of the game application are made into container images, and the images are uploaded to a container image repository. You can directly deploy the containerized game application based on these images by using CCE.
- Good scalability and quick scaling: To add a game server, you only need to start a container. You can start a container in seconds.
- Easy upgrade: You can quickly upgrade the components by merely replacing the images. In addition, CCE provides rolling upgrade, causing no service interruption during the upgrade.

### 2.2.3.3 Analyzing the Game Application

Before deploying the game application, learn the application components to be deployed and the relationships between these components.

**Figure 2-9** Game application architecture

The entire game application consists of four components: game client on the foreground, wow-auth login authentication system, wow-world game server, and MySQL database on the background. The following table describes these components and their relationships.

**Table 2-9** Application environment

Item	Description
<b>Application Components</b>	
• Foreground: game client	The game client is prepared in advance. You can directly download the client to install the game.
• Background: database (MySQL)	Stores game data.
• Background: authentication system (wow-auth)	Authenticates login of game players.
• Background: game server (wow-world)	Provides the game.

Item	Description
<b>Relationships between the components</b>	
	<ul style="list-style-type: none"><li>Both wow-auth and wow-world need to connect to the MySQL database for data storage. In this example, they are connected by using environment variables.</li><li>wow-auth needs to connect to wow-world. In this example, they are connected through environment variables.</li></ul>

### 2.2.3.4 Preparing the Environment

Before deploying the game application, prepare the hardware and Huawei Cloud environment.

#### Hardware Environment

Prepare a Windows PC with a graphics card and at least 20 GB disk space for running the game client.

#### Game Application Images

As shown in [Figure 2-8](#), this game application consists of three components: wow-auth login authentication system, wow-world game server, and MySQL database.

CCE supports deployment of the MySQL database with just a few clicks, so you do not need to prepare the image of the MySQL database. Instead, prepare the images of the wow-auth login authentication system and wow-world game server.



In this tutorial, the images of the two components have been created, so you can directly download them.

##### Step 1 Buy a Huawei Cloud ECS for downloading and uploading images.

1. Log in to the management console, and set the region to **CN-Hong Kong** in the upper left corner.
2. In the service list, choose **Compute > Elastic Cloud Server**, and click **Buy ECS** in the upper right corner.
3. Set the parameters listed in [Table 2-10](#) on the displayed **Buy ECS** page. For the other parameters, retain their default values.

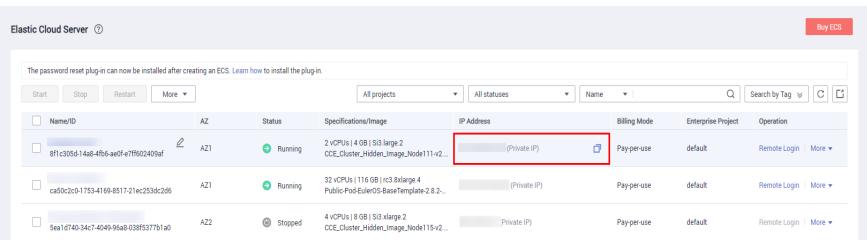
**Table 2-10** Buying ECS

Parameter	Description
Billing Mode	To save costs, you are advised to select <b>Pay-per-use</b> . You can clear the resources after deploying the game application.

Parameter	Description
Image	Select a Linux image from public images, for example, <b>Ubuntu 18.04 server 64bit</b> .
EIP	Create an EIP.
Login Mode	Set a password with high security level.
ECS Name	You can use the ECS name that is automatically generated. In this example, the ECS name is changed to <b>ecs-test</b> .

4. Click **Next: Confirm**. On the page displayed, confirm your order and click **Submit**.
5. After the ECS is created, you can view it in the ECS list and its status is **Running**.
6. Click  next to the EIP in the **IP Address** column to obtain the EIP.

**Figure 2-10** Obtaining the elastic IP address



Name/ID	AZ	Status	Specifications/Image	IP Address	Billing Mode	Enterprise Project	Operation
8f1c3056-14ab-4fb6-aedf-e1ff602409af	AZ1	Running	2 vCPUs   4 GB   S0 large-2 CCE_Cluster_Hidden_Image_Node11-v2	(Private IP) 	Pay-per-use	default	Remote Login More ▾
ca50c2c01753-4169-8517-21ec253dc2d6	AZ1	Running	32 vCPUs   116 GB   c3 Large 4 Public_Pod_EulerOS_BaseTemplate-2.8.2...	(Private IP)	Pay-per-use	default	Remote Login More ▾
fea1d740-34c7-4049-96a8-d38f5377b1a0	AZ2	Stopped	4 vCPUs   8 GB   S3 xlarge 2 CCE_Cluster_Hidden_Image_Node11-v2...	(Private IP)	Pay-per-use	default	Remote Login More ▾

**Step 2** Use a remote login tool, such as Xshell, to log in to the ECS.

**ssh root@elastic IP address bound to the ECS**

**Step 3** Run the following command to install Docker: If Docker cannot be automatically installed by running the following commands, manually install Docker based on the OS. For details, see [Docker Engine installation](#).

```
curl -fsSL get.docker.com -o get-docker.sh
```

```
sh get-docker.sh
```

It takes about five minutes to install Docker.

**Step 4** Log in to the SWR console, click **Generate Login Command** in the upper right corner of the **Dashboard** page, and copy and run the command on the node.

The command is successfully executed if the following information is displayed:  
Login Succeeded

**Step 5** Pull the prepared four images, including two images for the wow-auth authentication system (versions 5.0 and 5.1) and another two images for the wow-world game server (versions 5.0 and 5.1). Two versions of images are prepared to demonstrate subsequent upgrade operations. Run the following commands to pull images:

#### NOTE

It takes about 5 to 10 minutes to pull the images.

```
docker pull swr.cn-north-1.myhuaweicloud.com/wow/wow:wowauth-5.0
docker pull swr.cn-north-1.myhuaweicloud.com/wow/wow:wowworld-5.0-withmap
docker pull swr.cn-north-1.myhuaweicloud.com/wow/wow:wowauth-5.1
docker pull swr.cn-north-1.myhuaweicloud.com/wow/wow:wowworld-5.1-withmap
```

**Step 6** Run the following command to view the images:

**docker images**

**Step 7** Run the following command to upload the **wowauth-5.0** image to SWR:

```
docker tag [Image name:Tag] swr.ap-southeast-1.myhuaweicloud.com/[Organization name]/[Image name:Tag]
```

Example:

 **NOTE**

- In the example command, **gametest** indicates the organization name, which must be globally unique. If an organization has been created on SWR, you are advised to use the name of the existing organization.
- **ap-southeast-1** in **swr.ap-southeast-1.myhuaweicloud.com** indicates the region where the ECS that uploads the image is located.

```
docker tag swr.cn-north-1.myhuaweicloud.com/wow/wow:wowauth-5.0
swr.ap-southeast-1.myhuaweicloud.com/gametest/wow:wowauth-5.0
```

```
docker push swr.ap-southeast-1.myhuaweicloud.com/gametest/
wow:wowauth-5.0
```

Upload the **wowauth-5.1**, **wowworld-5.0-withmap**, and **wowworld-5.1-withmap** images in the same way.

**Step 8** On the SWR console, choose **My Images** in the navigation pane, and click **wow** in the image list. Then, you can view the four image tags on the page that is displayed.

----End

### 2.2.3.5 Deploying the Game Application

To deploy the game application on CCE, you need to perform the following operations:

1. **Create a cluster**: A container cluster is a logical group that runs applications and contains a group of cloud server resources. Each cluster node corresponds to a cloud server. When using CCE for the first time, create an initial cluster and add a node into the cluster.
2. **Deploy a MySQL database**: Deploy the database.
3. **Deploy the wow-auth authentication system**: Deploy the wow-auth authentication system on CCE.
4. **Deploy the wow-world game server**: Deploy the wow-world game server on CCE.

## Creating a Cluster

A container cluster is a logical group that runs applications and contains a group of cloud server resources. Each cluster node corresponds to a cloud server. When using CCE for the first time, create an initial cluster and add a node into the cluster.

- Step 1** Before creating a cluster, create a VPC and a key pair.



If a VPC and a key pair are available, skip to the next step.

**Table 2-11** Preparing the environment

No.	Item	Procedure
1	Creating a VPC	<p>Create a VPC before you create a cluster. A VPC provides an isolated, configurable, and manageable virtual network environment for CCE clusters.</p> <p>If you have a VPC already, skip to the next task.</p> <ol style="list-style-type: none"><li>1. Log in to the management console.</li><li>2. In the service list, choose <b>Networking &gt; Virtual Private Cloud</b>.</li><li>3. On the <b>Dashboard</b> page, click <b>Create VPC</b>.</li><li>4. Follow the instructions to create a VPC. Retain default settings for parameters unless otherwise specified.</li></ol>
2	Creating a key pair	<p>Create a key pair before you create a containerized application. Key pairs are used for identity authentication during remote login to a node. If you have a key pair already, skip this task.</p> <ol style="list-style-type: none"><li>1. Log in to the management console.</li><li>2. In the service list, choose <b>Security and Compliance &gt; Data Encryption Workshop</b>.</li><li>3. In the navigation pane, choose <b>Key Pair Service</b>. On the <b>Private Key Pairs</b> tab page, click <b>Create Key Pair</b>.</li><li>4. Enter a key pair name, select <b>I agree to have the private key managed on the cloud and I have read and agree to the Key Pair Service Disclaimer</b>, and click <b>OK</b>.</li><li>5. In the dialog box displayed, click <b>OK</b>. View and save the key pair. For security purposes, a key pair can be downloaded only once. Keep it secure to ensure successful login.</li></ol>

- Step 2** Choose **Clusters**. On the displayed page, click **Buy** next to **CCE cluster**.

Configure cluster parameters. Set **Cluster Name** to **cluster-wow** and select the VPC created in **Step 1**. For details, see [Buying a CCE Cluster](#).

**Step 3** Buy a node and select the key pair created in **Step 1** as the login mode. For details, see [Creating a Node](#).

Set the parameters for adding a node into the cluster. Set the node specifications, network, and login parameters as follows, and retain the default settings for the other parameters:

- **Specifications:** 4 vCPUs and 8 GB memory.

 NOTE

The 4-core CPU and 8 GB memory are the minimum specifications for the game application, supporting deployment of only one game server. To deploy more game servers, scale out the node or configure higher specifications.

- **Login Mode:** Select **Key pair**, and select the key pair you created in [Table 2-11](#) for logging in to the node.

**Step 4** After the node is created, log in to the VPC console, buy an EIP, and bind the EIP to the node.

----End

## Creating a MySQL Database

 NOTE

In this section, the database is deployed as a two-pod StatefulSet. In practice, complex clusters or active/standby deployments are often used, or database cloud services are directly used.

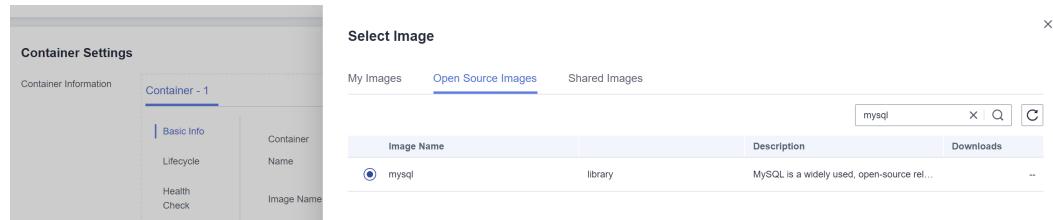
**Step 1** Log in to the CCE console, click the created cluster, choose **Workloads** in the navigation pane, and click **Create Workload** on the right.

**Step 2** Set the basic information about the workload.

- **Workload Type:** Select **StatefulSet**.
- **Workload Name:** Set it to **mysql**.
- **Namespace:** Set it to **default**.
- **Pods:** Change the value to **1** in this example.

**Step 3** In the **Container Settings** area, select the **mysql** image and click **OK**.

**Figure 2-11** Selecting the MySQL image



Select **5.7** for **Image Tag**.

Set environment variables. In this example, you need to set four environment variables. You can visit [MySQL](#) to view the environment variables that can be set for MySQL.

- **MYSQL\_ROOT\_PASSWORD**: password of the **root** user of MySQL.
- **MYSQL\_DATABASE**: name of the database created during image startup.
- **MYSQL\_USER**: name of the database user.
- **MYSQL\_PASSWORD**: password of the database user.

**Step 4** Set the **Startup Command** under the **Lifecycle** tap, for example, [Figure 2-12](#).

- **Command**:

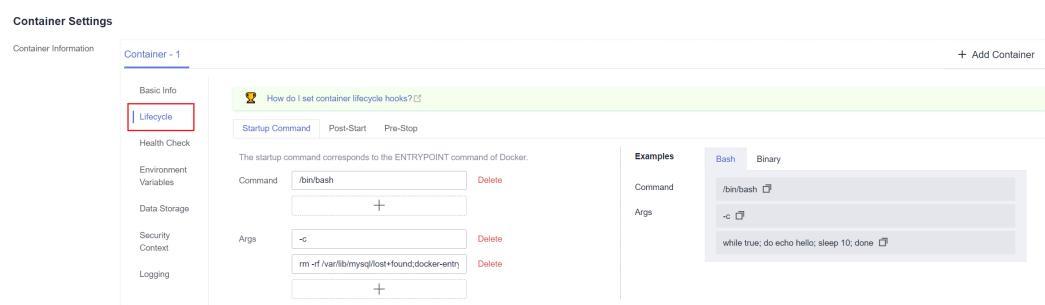
/bin/bash

- **Args**:

-c

rm -rf /var/lib/mysql/lost+found;docker-entrypoint.sh mysqld;

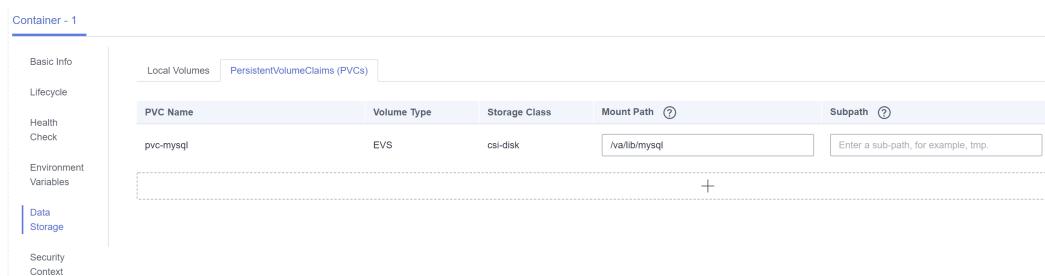
**Figure 2-12** Set startup commands



**Step 5** Choose **Data Storage > PersistentVolumeClaims (PVCs)** and add a cloud volume as the MySQL storage.

Mount the volume to the **/var/lib/mysql** path of the container, which is the default path used by MySQL.

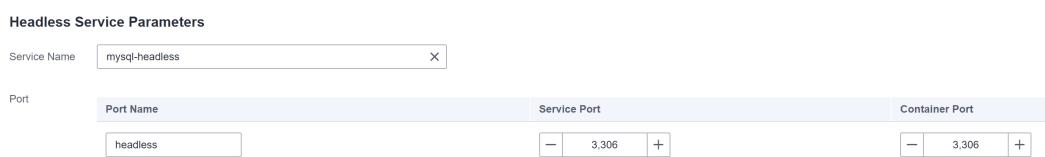
**Figure 2-13** Mounting storage for MySQL



**Step 6** In the **Headless Service Parameters** area, configure the headless Service parameters.

A headless Service needs to be configured for the StatefulSet to discover pods. In this example, this function is not used. You can set a random port number.

**Figure 2-14** Headless Service



**Step 7** In the **Service Settings** area, click to add a Service. Configure Service parameters and click **OK**.

Set **Service Name** to **mysql**, **Service Type** to **ClusterIP**, and **Container Port** and **Service Port** to **3306**, and click **OK**.

The default access port in the MySQL image is 3306. In this example, both the container port and service port are set to **3306** for convenience. The service port can be changed to another port.

In this way, the MySQL workload can be accessed through **Service name:Access port** (**mysql:3306** in this example) in the cluster.

**Figure 2-15** Adding a Service for MySQL

#### Create Service

Service Name	mysql		
Service Type	ClusterIP	NodePort	LoadBalancer
Port	Protocol	Service Port	Container Port
	TCP	- 3306 +	- 3306 +

**Step 8** Click **Create Workload**.

If the workload status is **Running**, the workload is successfully created.

**Step 9** Click the MySQL name to view its details. Click the **Access Mode** tab, and obtain and record the **Access Address**.

----End

## Deploying the wow-auth Authentication System

Deploy the wow-auth authentication system on CCE. During the deployment, set the wow-auth authentication system to connect to the MySQL database and wow-world game server through environment variables.

**Step 1** Log in to the CCE console, click the created cluster, choose **Workloads** in the navigation pane, and click **Create Workload** on the right.

**Step 2** Set the basic information about the workload.

- **Workload Type:** Select **Deployment**.
- **Workload Name:** Enter a workload name, for example, **wow-auth**.
- **Namespace:** Set it to **default**.
- **Pods:** Change the value to **1** in this example.

**Step 3** Select the **wowauth-5.0** image.

**Step 4** Retain the default values for other parameters and configure the container specifications.

Configure compute resources based on the application requirements. In this tutorial, the wow-auth component of the game application requires at least 0.5 CPU core and 0.5 GiB memory.

- Step 5** Configure the environment variables for connecting the wow-auth authentication system to the MySQL database and wow-world game server.

The environment variables listed in the following list are pre-set in the image. If you are unclear about the settings, see [How Can I Obtain the Values of Environment Variables When Deploying a Game Application?](#).

**Table 2-12** Setting environment variables

Variable Name	Description	Variable Value/Variable Reference
mysqlip	Set this variable to the database access address obtained in <a href="#">Step 9</a> .	10.247.59.224//10.247.130.188
mysqlroot passwd	Password of the database administrator, which must be the same as the administrator password set in <a href="#">Step 3</a> .	-
biboaddress	External access address of the game server. You can use the elastic IP address or elastic load balancer mode. Elastic IP address mode is used in this tutorial. Select the elastic IP address queried in <a href="#">Step 4</a> .	10.3.2.119
biboport	External access port of the game server. You must set a value ranging from 30000 to 32767 in advance.  <b>NOTE</b> This value must be globally unique in the current cluster. In this tutorial, for easy operations, a fixed value is specified for connection to the game server. During actual application deployment, you are advised to specify the dependencies when creating the images.	32500

- Step 6** In the **Service Settings** area, click to add a Service. Configure Service parameters and click **OK**.

- **Service Type:** In this example, select **NodePort**.
- **Service Name:** The service name can be the same as the application name, for example, **wow-auth**.
- **Service Affinity:** In this example, select **Cluster-level**.

- **Cluster-level:** The IP addresses and access ports of all nodes in a cluster can be used to access the workload associated with the Service. Service access will cause performance loss due to route redirection, and the source IP address of the client cannot be obtained.
- **Node-level:** Only the IP address and access port of the node where the workload is located can be used to access the workload associated with the Service. Service access will not cause performance loss due to route redirection, and the source IP address of the client can be obtained.
- **Port**
  - **Protocol:** In this example, TCP is selected.
  - **Service Port:** port for accessing the Service.
  - **Container Port:** listening port of the authentication system, which is port 3724 in this example. Do not change the port number because the port number 3724 has been set in the image.
  - **Node Port:** node port (with a private IP address) to which the container port will be mapped. Set it to **Auto**.

**Step 7** Click **Create Workload**.

----End

## Deploying the wow-world Game Server

Deploy the wow-world game server on CCE. During the deployment, set the wow-world game server to connect to the MySQL database through environment variables.

**Step 1** Log in to the CCE console, click the created cluster, choose **Workloads** in the navigation pane, and click **Create Workload** on the right.

**Step 2** Set the basic information about the workload.

- **Workload Type:** Select **Deployment**.
- **Workload Name:** Specify the workload name, for example, **wow-world**.
- **Namespace:** Retain the default value.
- **Instances:** You are advised to set the quantity to **1**. Otherwise, resources may be insufficient.
- **Description:** You can leave it blank.

**Step 3** Select the **wowworld-5.0-withmap** image.

**Step 4** Configure container specifications. Configure compute resources based on the application requirements. In this tutorial, the wow-world component of the game application requires at least 2 CPU cores and 2 GiB memory.

**Step 5** Set the environment variables used for interconnection with the MySQL database. **Table 2-13** describes the variables.

### NOTE

If you are unclear about the settings, see [How Can I Obtain the Values of Environment Variables When Deploying a Game Application?](#)

**Table 2-13** Setting environment variables

Variable Name	Description	Variable Value/Variable Reference
myslip	Set this variable to the database access address.	10.247.59.224
mysqlroot passwd	Password of the database administrator, which must be the same as the administrator password set in <a href="#">Step 3</a> .	-

- Step 6** In the **Service Settings** area, click to add a Service. Configure Service parameters and click **OK**.
- **Service Type:** In this example, select **NodePort**.
  - **Service Name:** The service name can be the same as the workload name, for example, **wow-world**.
  - **Service Affinity:** In this example, select **Cluster-level**.
    - **Cluster-level:** The IP addresses and access ports of all nodes in a cluster can be used to access the workload associated with the Service. Service access will cause performance loss due to route redirection, and the source IP address of the client cannot be obtained.
    - **Node-level:** Only the IP address and access port of the node where the workload is located can be used to access the workload associated with the Service. Service access will not cause performance loss due to route redirection, and the source IP address of the client can be obtained.
  - **Protocol:** In this example, TCP is selected.
  - **Service Port:** port for accessing the Service.
  - **Container Port:** Listening port of the game server, which is port 8085 in this tutorial. Do not change the port number because the port number has been set in the image.
  - **Node Port.** This port must be the same as the **biboport** in the environment variable settings added when the authentication system is deployed. If no such environment variable is found, see [How Can I Obtain the Access Port When Deploying a Game Application?](#)

**Step 7** Click **Create Workload**.

----End

### 2.2.3.6 Scaling the Game Application

Scale the **wow-auth** workload in manual or automatic mode by using CCE.

#### Manual Scaling

- Step 1** Log in to the CCE console, click the cluster, choose **Workloads** in the navigation pane, and click **wow-auth** to access its details page.

**Step 2** Click  next to **Pods**, change the value to **2**, and click **Save**.

**Step 3** On the **Pods** tab page of the details page, you can view that the new pod is being created. The creation will be completed within seconds. If the pod status is not

refreshed, click . The status of the instance is **Running**.

----End

## Auto Scaling

**Step 1** Log in to the CCE console, click the cluster, and choose **Workload Scaling** in the navigation pane.

**Step 2** Click **Create HPA Policy** in the upper right corner.

**Step 3** Configure HPA policy parameters.

Select the **CPU usage** metric and set **Desired Value** to **50%**. Set **30%** for **Scale-in** and **70%** for **Scale-out**.

**Figure 2-16** HPA policy



**Step 4** Click **Create**.

----End

### 2.2.3.7 Upgrading the Game Application

Game requirements grow rapidly and versions change frequently. The upgrade efficiency and user experience during the upgrade are crucial for gaming services.

CCE provides a rolling upgrade policy to separately update pods one by one instead of updating all pods at the same time. This ensures that services are not interrupted during the upgrade.

This section uses **wow-auth** as an example to demonstrate the rolling upgrade of applications.

## Prerequisites

Ensure that the workload to be upgraded has at least two pods. You are advised to manually scale the workload into two pods before performing the upgrade.

## Procedure

**Step 1** Log in to the CCE console, click the cluster, choose **Workloads** in the navigation pane, and click **wow-auth** to access its details page.

**Step 2** Click the **Containers** tab and click **Edit**.

**Step 3** After containerization, you can easily upgrade the workload by replacing its image. Click **Replace Image** and select **wowauth-5.1**.

**Step 4** Click **Upgrade Workload** in the lower right corner of the page.

On the **Pods** tab page, you can view that one pod is being created and then the other is being stopped. This ensures that there is always a pod running and the service is not interrupted during the upgrade.

**Step 5** Click  on the right. If both pods are in the running state, the upgrade is successful.

----End

### 2.2.3.8 Deleting Resources

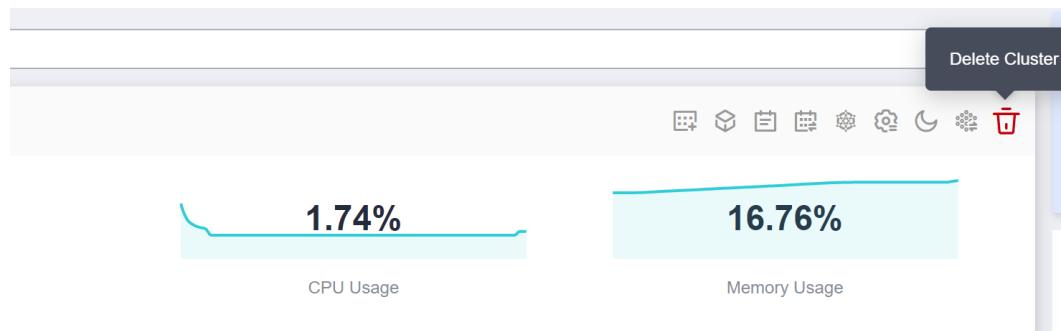
You have finished deploying the game application. Fees are generated during node and application running. When you finish deploying the game application, you are advised to delete the resources associated with it to avoid incurring charges for resources that you are not using.

#### Procedure

**Step 1** Delete cluster resources.

Log in to the CCE console and click  in the upper right corner of the cluster card.

**Figure 2-17** Deleting a cluster



**Step 2** Delete the ECS on which Docker is installed.

1. Log in to the ECS console.
2. Choose **More > Delete** in the **Operation** column of the newly created **ecs-cy** server. Then, on the page displayed, select **Release the EIPs bound to the following ECSs** and **Delete the data disks attached to the following ECSs**, and click **OK** to delete the ECS.

----End

### 2.2.3.9 FAQs

#### How Can I Obtain the Values of Environment Variables When Deploying a Game Application?

- Step 1** Log in to the CCE console, click the cluster, and choose **Workloads** in the navigation pane.
  - Step 2** Click **wow-mysql-mysqld** to access its details page.
  - Step 3** Click the **Access Mode** tab. The IP address is the value of **myslip** of wow-auth.
  - Step 4** Click the **Containers** tab. The value of the environment variable **MYSQL\_ROOT\_PASSWORD** is the value of **mysqlrootpasswd** in wow-auth.
  - Step 5** Choose **Nodes** in the navigation pane and view the elastic IP address of the node created in [Creating a Cluster](#). The elastic IP address is the value of **biboaddress**.
- End

#### How Can I Obtain the Access Port When Deploying a Game Application?

- Step 1** Log in to the CCE console, choose **Workloads** in the navigation pane, and click **wow-auth** to access its details page.
  - Step 2** Click the **Containers** tab, choose **Environment Variables** in the **Container Information** area, and view the value of **biboport**.
- End

# 3 Migration

## 3.1 Migrating Container Images

### 3.1.1 Overview

#### Challenges

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

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

This section describes three ways for migrating image repositories to SWR smoothly. You can select one as required.

## Migration Solutions

**Table 3-1** Comparison of migration solutions and application scenarios

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

Solution	Application Scenario	Precautions
Migrating images to SWR using image-syncer	A large number of images	<ul style="list-style-type: none"><li>• Many-to-many image repository synchronization is supported.</li><li>• Docker image repository services (such as Docker Hub, Quay, and Harbor) based on Docker Registry V2 are supported.</li><li>• Memory- and network-dependent synchronization is fast.</li><li>• Flushing the Blob information of synchronized images avoids repetition.</li><li>• Concurrent synchronization can be achieved by adjusting the number of concurrent tasks in the configuration files.</li><li>• Automatically retrying failed synchronization tasks can resolve most network jitter during image synchronization.</li><li>• Docker or other programs are not required.</li></ul>
Synchronizing images across clouds from Harbor to SWR	A customer deploys services in multiple clouds and uses Harbor as their image repository.	Only Harbor v1.10.5 and later versions are supported.

### 3.1.2 Migrating Images to SWR Using Docker Commands

#### Scenarios

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

#### Procedure

**Step 1** Pull images from the source repository.

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

Example: **docker pull nginx:latest**

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

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	22f2bf2e2b4f	5 hours ago	22.8MB

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

1. Log in to the VM where the target container is located and log in to SWR. For details, see [Uploading an Image Through a Container Engine Client](#).

2. Tag the images.

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

Example:

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

3. Run the following command to push the images to the target image repository.

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

Example:

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

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

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

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

----End

### 3.1.3 Migrating Images to SWR Using image-syncer

#### Scenarios

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

#### Procedure

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

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

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

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

image-syncer supports the Docker image repository based on Docker Registry V2. Enter the authentication information as required. In the following example, the image repository of AP-Singapore is migrated to CN-Hong Kong.

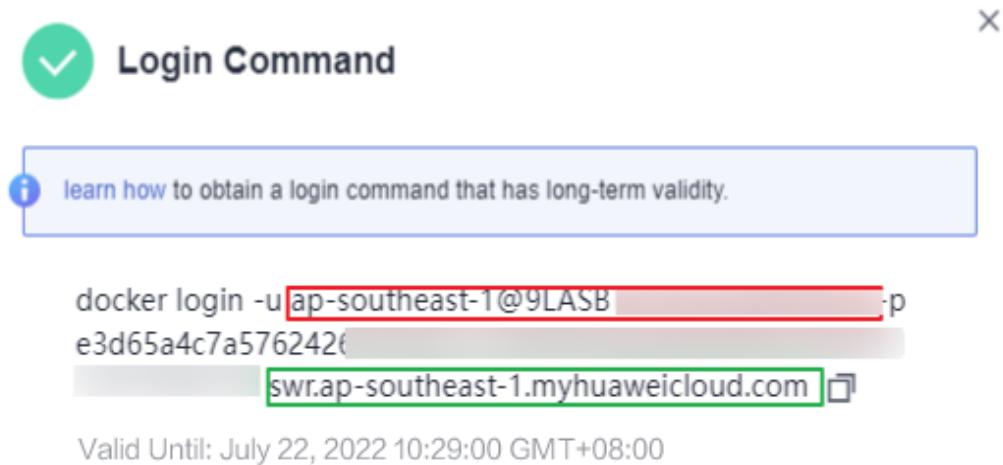
The following describes how to write the authentication information of the source and target repositories.

```
{  
    "swr.ap-southeast-3.myhuaweicloud.com": {  
        "username": "ap-southeast-3@F1I3Q.....",  
        "password": "2fd4c869ea0....."  
    },  
    "swr.ap-southeast-1.myhuaweicloud.com": {  
        "username": "ap-southeast-1@4N3FA.....",  
        "password": "f1c82b57855f9d35....."  
    }  
}
```

In the preceding commands, **swr.ap-southeast-1.myhuaweicloud.com** indicates the image repository address. You can obtain the username and password from the login command as follows:

Log in to the SWR console, and click **Generate Login Command** in the upper right corner to obtain the login command in the dialog box displayed, as shown in the following figure.

**Figure 3-1** Generating a login command



In **the above figure**, **ap-southeast-1@9LASB.....** is the username;  
**e3d65a4c7a57624264c.....** is the password;  
**swr.ap-southeast-1.myhuaweicloud.com** is the image repository address.

**⚠ CAUTION**

For security, the example username and password are not complete. You should use the actual username and password obtained from the console.

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

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

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

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

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

**Table 3-2** Command parameter description

Parameter	Description
--config	Sets the path of config file. This file needs to be created before starting synchronization. Default config file is at "current/working/directory/config.json". (This flag can be replaced with flag <b>--auth</b> and <b>--images</b> which for better organization.)
--images	Sets the path of image rules file. This file needs to be created before starting synchronization. Default config file is at "current/working/directory/images.json".
--auth	Sets the path of authentication file. This file needs to be created before starting synchronization. Default config file is at "current/working/directory/auth.json".
--log	Sets the path of log file. Logs will be printed to Stderr by default.
--namespace	Sets default-namespace. default-namespace can also be set by environment variable <b>DEFAULT_NAMESPACE</b> . If they are both set at the same time, <b>DEFAULT_NAMESPACE</b> will not work at this synchronization. default-namespace will work only if default-registry is not empty.
--proc	Number of goroutines. Default value is 5.
--retries	Number of retries. Default value is 2. The retries of failed sync tasks will start after all sync tasks are executed once. Retries of failed sync tasks will resolve most occasional network problems during synchronization.

Parameter	Description
--registry	Sets default-registry. Default-registry can also be set by environment variable <b>DEFAULT_REGISTRY</b> . If they are both set at the same time, <b>DEFAULT_REGISTRY</b> will not work at this synchronization. default-registry will work only if default-namespace is not empty.

After the migration command is executed, you can log in to the target image repository to view the migrated images.

----End

### 3.1.4 Synchronizing Images Across Clouds from Harbor to SWR

#### Scenarios

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

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

#### Background

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

#### Accessing SWR Through a Public Network

**Step 1** Configure a registry endpoint on Harbor.



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

1. Add an endpoint.

2. Configure the following parameters.

### New Registry Endpoint

Provider *	Huawei SWR
Name *	test
Description	
Endpoint URL *	https://swr...myhuaweicloud.com
Access ID	-east-3@CCRXJUTQ7QQFRCYBDMXC
Access Secret	.....
Verify Remote Cert ⓘ	<input type="checkbox"/>
<input type="button" value="TEST CONNECTION"/> <input type="button" value="CANCEL"/> <input type="button" value="OK"/>	

- **Provider:** Select **Huawei SWR**.
- **Name:** Enter a customized name.
- **Endpoint URL:** Enter the public network domain name of SWR in the format of **https://\{SWR image repository address\}**. To obtain the image repository address, log in to the SWR console, choose **My Images**, and

click **Upload Through Client**. You can view the image repository address of the current region on the page that is displayed.

- **Access ID:** Enter an access ID in the format of **Regional project name@[AK]**.
- Access Secret: Enter an AK/SK. To obtain an AK/SK, see [Obtaining a Long-Term Valid Login Command](#).
- **Verify Remote Cert:** Deselect the option.

## Step 2 Configure a replication rule.

1. Create a replication rule.

2. Configure the following parameters.

- **Name:** Enter a customized name.
- **Replication mode:** Select **Push-based**, indicating that images are pushed from the local Harbor to the remote repository.
- **Source resource filter:** Filters images on Harbor based on the configured rules.
- **Destination registry:** Select the endpoint created in [Step 1](#).
- **Destination Namespace:** Enter the organization name on SWR.  
**Flattening:** Select **Flatten All Levels**, indicating that the hierarchy of the registry is reduced when copying images. If the directory of Harbor registry is **library/nginx** and the directory of the endpoint namespace is **dev-container**, after you flatten all levels, the directory of the endpoint namespace is **library/nginx -> dev-container/nginx**.
- **Trigger Mode:** Select **Manual**.
- **Bandwidth:** Set the maximum network bandwidth when executing the replication rule. The value **-1** indicates no limitation.

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

The screenshot shows the Harbor interface with the sidebar navigation open. The 'Replications' section is selected. A table lists a single replication rule named 'test'. The 'REPLICATE' button next to the rule is highlighted with a red box.

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

----End

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

**Step 1** Configure a VPC endpoint.

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

The screenshot shows the VPC endpoint details page for a specific endpoint. The 'Private Domain Name' field is highlighted with a red box and contains the value 'vpcpep-b6cf5b9d-2b46-49a9-a0de-f877a505ec42.svr.cn-east-3.myhuaweicloud.com'.

**Step 3** Configure a registry endpoint on Harbor.

### NOTE

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

1. Add an endpoint.
2. Configure the following parameters.

## New Registry Endpoint

The screenshot shows a configuration dialog for a new registry endpoint. The provider is set to "Huawei SWR". The name is "test". The description field is empty. The endpoint URL is "https://vpcep-b6cf5b9d-2b46-49a9". The access ID is "cn-east-3@CCRXJUTQ7QQFRCYBDM". The access secret is redacted. The "Verify Remote Cert" checkbox is unchecked. At the bottom, there are "TEST CONNECTION", "CANCEL", and "OK" buttons.

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

### Step 4 Configure a replication rule.

1. Create a replication rule.

The screenshot shows the Harbor UI interface. On the left, there's a sidebar with various navigation options: Projects, Logs, Administration (with sub-options like Users, Robot Accounts, Registries), Replications (which is currently selected and highlighted in blue), Distributions, Labels, Project Quotas, Interrogation Services, Garbage Collection, and Configuration. At the top right, there's a search bar labeled 'Search Harbor...' and some global actions like 'REPLICATE' and 'ACTIONS'. The main content area is titled 'Replications' and contains a table with the following columns: Name, Status, Source registry, Replication Mode, Destination Registry:Namespace, and Destination Repository Flattening. A large red box highlights the 'New Replication Rule' button ('+ NEW REPLICATION RULE'). Below the table, a message says 'We couldn't find any replication rules!' with a small icon of a funnel.

2. Configure the following parameters.

## New Replication Rule

Name \* test

Description

Replication mode  Push-based  Pull-based

Source resource filter

Name:	library/*
Tag:	matching
Label:	matching
Resource:	image

Destination registry \* test-https://vpcep-b6cf5b9d-2b46-49a9-2b46

Destination

Namespace:	dev-container
------------	---------------

Flattening: Flatten All Levels

Trigger Mode \* Manual

Bandwidth \* -1 Kbps

Override

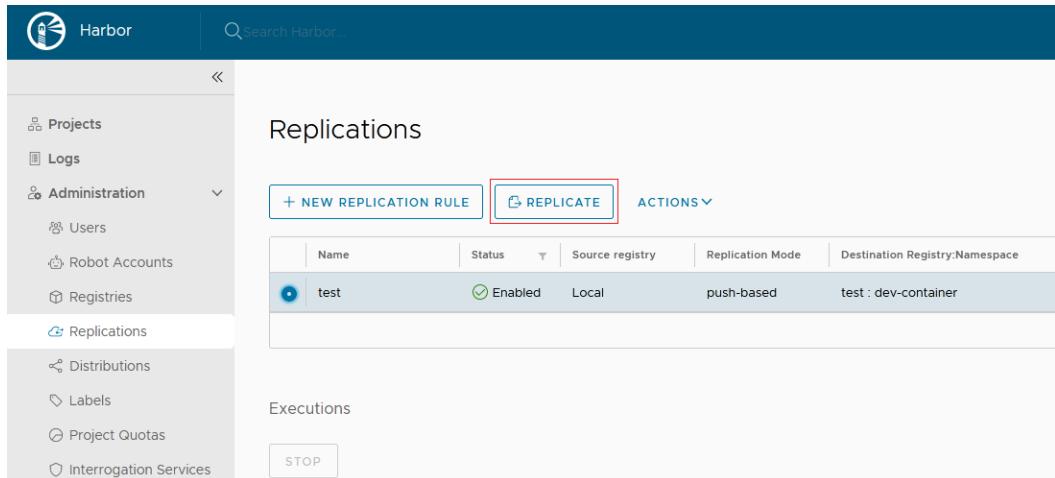
CANCEL SAVE

- **Name:** Enter a customized name.
- **Replication mode:** Select **Push-based**, indicating that images are pushed from the local Harbor to the remote repository.
- **Source resource filter:** Filters images on Harbor based on the configured rules.
- **Destination registry:** Select the endpoint created in [Step 3](#).
- **Destination Namespace:** Enter the organization name on SWR.
- Flattening:** Select **Flatten All Levels**, indicating that the hierarchy of the registry is reduced when copying images. If the directory of Harbor registry is **library/nginx** and the directory of the endpoint namespace is

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

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

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



The screenshot shows the Harbor interface with the 'Replications' tab selected. On the left sidebar, under 'Administration', the 'Replications' option is highlighted. In the main area, there's a table titled 'Replications' with one row. The row contains the following information:

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

Below the table, there's a 'STOP' button. At the top of the main area, there are three buttons: '+ NEW REPLICATION RULE', 'REPLICATE' (which is highlighted with a red box), and 'ACTIONS'.

----End

## 3.2 Migrating On-premises Kubernetes Clusters to CCE

### 3.2.1 Solution Overview

#### Scenario

Containers are growing in popularity and Kubernetes simplifies containerized deployment. Many companies choose to build their own Kubernetes clusters. However, the O&M workload of on-premises clusters is heavy, and O&M personnel need to configure the management systems and monitoring solutions by themselves. This increases the labor costs while decreasing the efficiency.

In terms of performance, an on-premises cluster has poor scalability due to its fixed specifications. Auto scaling cannot be implemented in case of traffic surges, which may easily result in the insufficient or waste of cluster resources. In addition, an on-premises cluster is usually deployed on a single node without considering disaster recovery risks. Once a fault occurs, the entire cluster cannot be used, which may cause serious production incident.

Now you can address the preceding challenges by using CCE, a service that allows easy cluster management and flexible scaling, integrated with application service mesh and Helm charts to simplify cluster O&M and reduce operations costs. CCE is easy to use and delivers high performance, security, reliability, openness, and compatibility. This section describes the solution and procedure for migrating on-premises clusters to CCE.

## Precautions

Compared with on-premises Kubernetes clusters, CCE clusters have multiple advantages. For details, see [Comparative Analysis of the Cloud Container Engine and On-Premises Kubernetes Clusters](#). There are some restrictions when using CCE clusters. For details, see [Notes and Constraints](#). Evaluate the restrictions before using CCE clusters.

## Migration Solution

This section describes a cluster migration solution, which applies to the following types of clusters:

- Kubernetes clusters built in local IDCs
- On-premises clusters built using multiple ECSS
- Cluster services provided by other cloud service providers

Before the migration, you need to analyze all resources in the source clusters and then determine the migration solution. Resources that can be migrated include resources inside and outside the clusters, as listed in the following table.

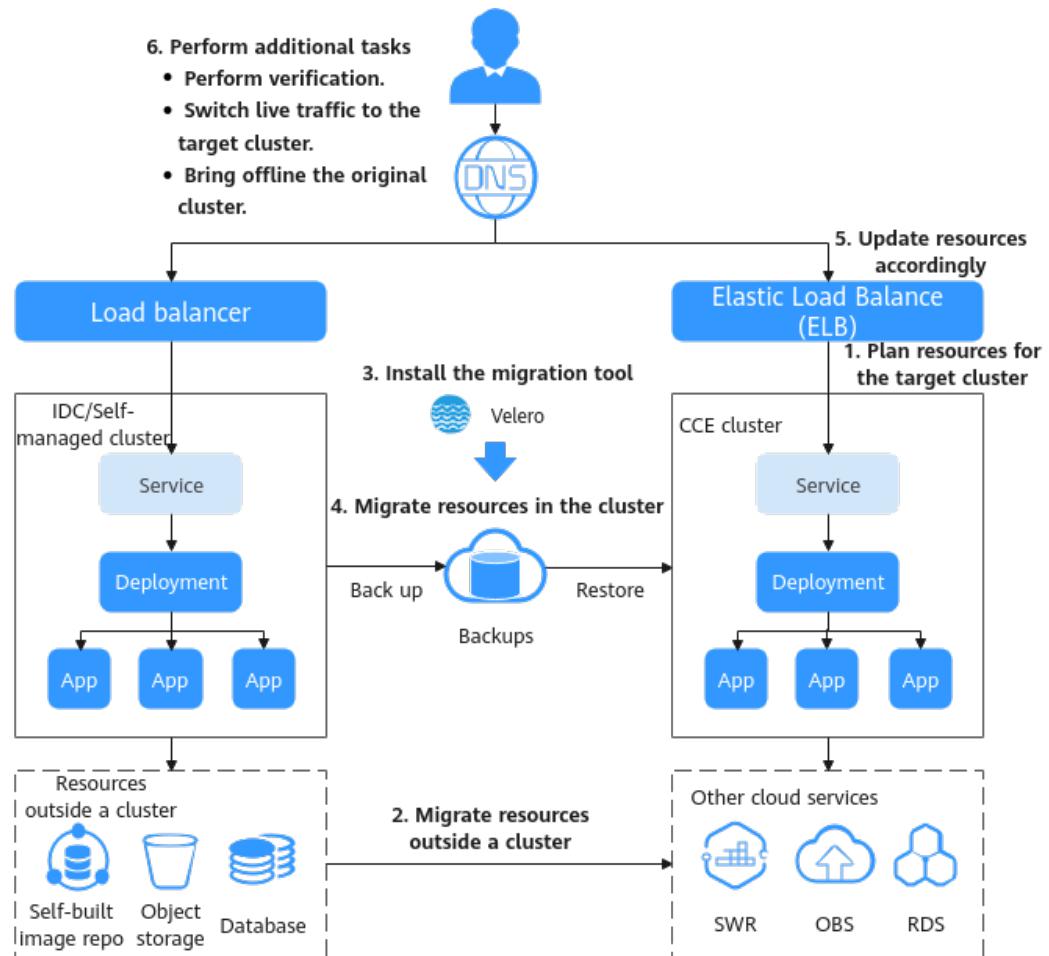
**Table 3-3** Resources that can be migrated

Category	Migration Object	Remarks
Resources inside a cluster	All objects in a cluster, including pods, jobs, Services, Deployments, and ConfigMaps.	<p>You are not advised to migrate the resources in the <b>velero</b> and <b>kube-system</b> namespaces.</p> <ul style="list-style-type: none"><li>• <b>velero</b>: Resources in this namespace are created by the migration tool and do not need to be migrated.</li><li>• <b>kube-system</b>: Resources in this namespace are system resources. If this namespace of the source cluster contains resources created by users, migrate the resources on demand.</li></ul> <p><b>CAUTION</b></p> <p>If you are migrating or backing up cluster resources in CCE, for example, from a namespace to another, do not back up Secret <b>paas.elb</b>. It is because secret <b>paas.elb</b> is periodically updated. After the backup is complete, the secret may become invalid when it is restored. As a result, network storage functions are affected.</p>

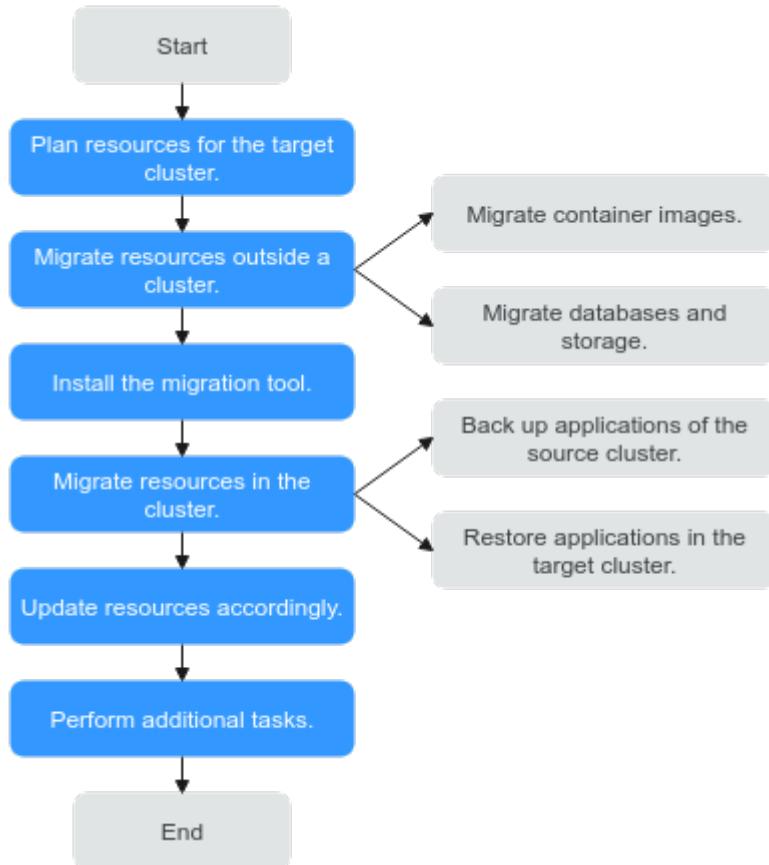
Category	Migration Object	Remarks
	PersistentVolumes (PVs) mounted to containers	Due to restrictions of the Restic tool, migration is not supported for the hostPath storage volume. For details about how to solve the problem, see <a href="#">Storage Volumes of the HostPath Type Cannot Be Backed Up</a> .
Resources outside a cluster	On-premises image repository	Resources can be migrated to SoftWare Repository for Container (SWR).
	Non-containerized database	Resources can be migrated to Relational Database Service (RDS).
	Non-local storage, such as object storage	Resources can be migrated to Object Storage Service (OBS).

[Figure 3-2](#) shows the migration process. You can migrate resources outside a cluster as required.

**Figure 3-2** Migration solution diagram



## Migration Process



The cluster migration process is as follows:

### Step 1 Plan resources for the target cluster.

For details about the differences between CCE clusters and on-premises clusters, see **Key Performance Parameter** in [Planning Resources for the Target Cluster](#). Plan resources as required and ensure that the performance configuration of the target cluster is the same as that of the source cluster.

### Step 2 Migrate resources outside a cluster.

If you need to migrate resources outside the cluster, see [Migrating Resources Outside a Cluster](#).

### Step 3 Install the migration tool.

After resources outside a cluster are migrated, you can use a migration tool to back up and restore application configurations in the source and target clusters. For details about how to install the tool, see [Installing the Migration Tool](#).

### Step 4 Migrate resources in the cluster.

Use Velero to back up resources in the source cluster to OBS and restore the resources in the target cluster. For details, see [Migrating Resources in a Cluster \(Velero\)](#).

- [Backing Up Applications in the Source Cluster](#)

To back up resources, use the Velero tool to create a backup object in the original cluster, query and back up cluster data and resources, package the data, and upload the package to the object storage that is compatible with the S3 protocol. Cluster resources are stored in the JSON format.

- **Restoring Applications in the Target Cluster**

During restoration in the target cluster, Velero specifies the temporary object bucket that stores the backup data, downloads the backup data to the new cluster, and redeploys resources based on the JSON file.

**Step 5 Update resources accordingly.**

After the migration, cluster resources may fail to be deployed. You need to update the faulty resources. The possible adaptation problems are as follows:

- **Updating Images**
- **Updating Services**
- **Updating the Storage Class**
- **Updating Databases**

**Step 6 Perform additional tasks.**

After cluster resources are properly deployed, verify application functions after the migration and switch service traffic to the target cluster. After confirming that all services are running properly, bring the source cluster offline.

----End

### 3.2.2 Planning Resources for the Target Cluster

CCE allows you to customize cluster resources to meet various service requirements. **Table 3-4** lists the key performance parameters of a cluster and provides the planned values. You can set the parameters based on your service requirements. It is recommended that the performance configuration be the same as that of the source cluster.

---

**NOTICE**

After a cluster is created, the resource parameters marked with asterisks (\*) in **Table 3-4** cannot be modified.

---

**Table 3-4** CCE cluster planning

Resource	Key Performance Parameter	Description	Example Value
Cluster	*Cluster Type	<ul style="list-style-type: none"><li><b>CCE cluster:</b> supports hybrid deployment of VMs and bare-metal servers (BMSs), and heterogeneous nodes such as GPU and NPU nodes. You can run your containers in a secure and stable container runtime environment based on a high-performance network model.</li><li><b>CCE Turbo cluster:</b> runs on a cloud native infrastructure that features software-hardware synergy to support passthrough networking, high security and reliability, and intelligent scheduling.</li></ul>	CCE cluster
	*Network Model	<ul style="list-style-type: none"><li><b>VPC network:</b> The container network uses VPC routing to integrate with the underlying network. This network model is applicable to performance-intensive scenarios. The maximum number of nodes allowed in a cluster depends on the route quota in a VPC network.</li><li><b>Tunnel network:</b> The container network is an overlay tunnel network on top of a VPC network and uses the VXLAN technology. This network model is applicable when there is no high requirements on performance.</li><li><b>Cloud Native Network 2.0:</b> The container network deeply integrates the elastic network interface (ENI) capability of VPC, uses the VPC CIDR block to allocate container addresses, and supports passthrough networking to containers through a load balancer.</li></ul>	VPC network
	*Number of master nodes	<ul style="list-style-type: none"><li><b>3:</b> Three master nodes will be created to deliver better DR performance. If one master node is faulty, the cluster can still be available without affecting service functions.</li><li><b>1:</b> A single master node will be created. This mode is not recommended in commercial scenarios.</li></ul>	3
Node	OS	<ul style="list-style-type: none"><li>EulerOS</li><li>CentOS</li><li>Ubuntu</li></ul>	Euler OS

Resource	Key Performance Parameter	Description	Example Value
	Node Specifications (vary depending on the actual region)	<ul style="list-style-type: none"><li><b>General-purpose:</b> provides a balance of computing, memory, and network resources. It is a good choice for many applications. General-purpose nodes can be used for web servers, workload development, workload testing, and small-scale databases.</li><li><b>Memory-optimized:</b> provides higher memory capacity than general-purpose nodes and is suitable for relational databases, NoSQL, and other workloads that are both memory-intensive and data-intensive.</li><li><b>General computing-basic:</b> provides a balance of computing, memory, and network resources and uses the vCPU credit mechanism to ensure baseline computing performance. Nodes of this type are suitable for applications requiring burstable high performance, such as light-load web servers, enterprise R&amp;D and testing environments, and low- and medium-performance databases.</li><li><b>GPU-accelerated:</b> provides powerful floating-point computing and is suitable for real-time, highly concurrent massive computing. Graphical processing units (GPUs) of P series are suitable for deep learning, scientific computing, and CAE. GPUs of G series are suitable for 3D animation rendering and CAD. GPU-accelerated nodes can be added only to clusters of v1.11 or later.</li><li><b>High-performance computing:</b> provides stable and ultra-high computing performance and is suitable for scientific computing and workloads that demand ultra-high computing power and throughput.</li><li><b>General computing-plus:</b> provides stable performance and exclusive resources to enterprise-class workloads with high and stable computing performance.</li><li><b>Disk-intensive:</b> supports local disk storage and provides high networking performance. It is designed for workloads requiring high throughput and data switching, such as big data workloads.</li><li><b>Ultra-high I/O:</b> delivers ultra-low SSD access latency and ultra-high IOPS performance. This type of specifications is ideal for high-performance relational databases, NoSQL databases (such as Cassandra and MongoDB), and Elasticsearch.</li><li><b>Ascend-accelerated:</b> Ascend-accelerated nodes powered by HiSilicon Ascend 310 AI processors are</li></ul>	General-purpose (node specifications: 4 vCPUs and 8 GiB memory)

Resource	Key Performance Parameter	Description	Example Value
		applicable to scenarios such as image recognition, video processing, inference computing, and machine learning.	
	System Disk	<ul style="list-style-type: none"><li><b>High I/O:</b> The backend storage media is SAS disks.</li><li><b>Ultra-high I/O:</b> The backend storage media is SSD disks.</li></ul>	High I/O
	Storage Type	<ul style="list-style-type: none"><li><b>EVS volumes:</b> Mount an EVS volume to a container path. When containers are migrated, the attached EVS volumes are migrated accordingly. This storage mode is suitable for data that needs to be permanently stored.</li><li><b>SFS volumes:</b> Create SFS volumes and mount them to a container path. The file system volumes created by the underlying SFS service can also be used. SFS volumes are applicable to persistent storage for frequent read/write in multiple workload scenarios, including media processing, content management, big data analysis, and workload analysis.</li><li><b>OBS volumes:</b> Create OBS volumes and mount them to a container path. OBS volumes are applicable to scenarios such as cloud workload, data analysis, content analysis, and hotspot objects.</li><li><b>SFS Turbo volumes:</b> Create SFS Turbo volumes and mount them to a container path. SFS Turbo volumes are fast, on-demand, and scalable, which makes them suitable for DevOps, containerized microservices, and enterprise office applications.</li></ul>	EVS volumes

### 3.2.3 Migrating Resources Outside a Cluster

If your migration does not involve resources outside a cluster listed in [Table 3-3](#) or you do not need to use other services to update resources after the migration, skip this section.

#### Migrating Container Images

To ensure that container images can be properly pulled after cluster migration and improve container deployment efficiency, you are advised to migrate private images to SoftWare Repository for Container (SWR). CCE works with SWR to provide a pipeline for automated container delivery. Images are pulled in parallel, which greatly improves container delivery efficiency.

You need to manually migrate container images.

- Step 1** Remotely log in to any node in the source cluster and run the **docker pull** command to pull all images to the local host.
- Step 2** Log in to the SWR console, click **Login Command** in the upper right corner of the page, and copy the command.
- Step 3** Run the copied login command on the node.

The message "Login Succeeded" will be displayed upon a successful login.

- Step 4** Add tags to all local images.

```
docker tag [Image name 1:tag 1] [Image repository address]/[Organization name]/[Image name 2:tag 2]
```

- *[Image name 1:tag 1]*: name and tag of the local image to be pulled.
- *[Image repository address]*: You can query the image repository address on the SWR console.
- *[Organization name]*: Enter the name of the organization you created on the SWR console.
- *[Image name 2:tag 2]*: image name and tag displayed on the SWR console.

#### Example

```
docker tag nginx:v1 swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/mynginx:v1
```

- Step 5** Run the **docker push** command to upload all local container image files to SWR.

```
docker push [Image repository address]/[Organization name]/[Image name 2:tag 2]
```

#### Example

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

----End

## Migrating Databases and Storage (On-Demand)

You can determine whether to use **Relational Database Service (RDS)** and **Object Storage Service (OBS)** based on your production requirements. After the migration is complete, you need to reconfigure the database and storage for applications in the target CCE cluster.

### Database migration

If your database is deployed without using containers and needs to be migrated to the cloud, **Data Replication Service (DRS)** can help you do this. DRS provides multiple capabilities, including online migration, backup migration, real-time disaster recovery, data synchronization, and data subscription. Contact O&M or development personnel to migrate the database. For details, see [Migrating Databases Across Cloud Platforms](#). After the migration is complete, interconnect with the database by following the procedure described in [Updating Databases](#).

### Storage migration

If your cluster has connected to an object storage service and needs to be migrated to the cloud, **Object Storage Migration Service (OMS)** can help you migrate data to OBS. Other storage types are not supported by official tools.

Contact O&M or development personnel to migrate object storage data. For details, see [Creating a Migration Task](#). After the migration is complete, attach

the object storage to the application by referring to [Connecting to an Existing OBS File System](#).

 NOTE

Currently, you can use OMS to migrate object storage data from AWS China, Alibaba Cloud, Microsoft Azure, Baidu Cloud, Kingsoft Cloud, Ucloud, QingCloud, Qiniu Cloud, and Tencent Cloud to Huawei Cloud [OBS](#).

### 3.2.4 Installing the Migration Tool

Velero is an open-source backup and migration tool for Kubernetes clusters. It integrates the persistent volume (PV) data backup capability of the Restic tool and can be used to back up Kubernetes resource objects (such as Deployments, jobs, Services, and ConfigMaps) in the source cluster. Data in the PV mounted to the pod is backed up and uploaded to the object storage. When a disaster occurs or migration is required, the target cluster can use Velero to obtain the corresponding backup data from OBS and restore cluster resources as required.

According to [Migration Solution](#), you need to prepare temporary object storage to store backup files before the migration. Velero supports OSB or [MinIO](#) as the object storage. OBS requires sufficient storage space for storing backup files. You can estimate the storage space based on your cluster scale and data volume. You are advised to use OBS for backup. For details about how to deploy Velero, see [Installing Velero](#).

CCE supports backup and restore using the e-backup add-on, which is compatible with Velero and uses OBS as the storage backend. You can use Velero in on-premises clusters and use e-backup in CCE.

- Without e-backup: Install Velero in the source and target clusters by following the instructions described in this topic, and migrate resources by referring to [Migrating Resources in a Cluster \(Velero\)](#).
- With e-backup: Install Velero in the source cluster and use OBS as the storage backend by following the instructions described in [Installing Velero](#), and install e-backup in the target CCE cluster and migrate resources by referring to [Migrating Resources in a Cluster \(e-backup\)](#).

## Prerequisites

- The Kubernetes version of the source on-premises cluster must be 1.10 or later, and the cluster can use DNS and Internet services properly.
- If you use OBS to store backup files, you need to obtain the AK/SK of a user who has the right to operate OBS. For details about how to obtain the AK/SK, see [Obtaining Access Keys \(AK/SK\)](#).
- If you use MinIO to store backup files, bind an EIP to the server where MinIO is installed and enable the API and console port of MinIO in the security group.
- The target CCE cluster has been created.
- The source cluster and target cluster must each have at least one idle node. It is recommended that the node specifications be 4 vCPUs and 8 GB memory or higher.

## Installing MinIO

MinIO is an open-source, high-performance object storage tool compatible with the S3 API protocol. If MinIO is used to store backup files for cluster migration, you need a temporary server to deploy MinIO and provide services for external systems. If you use OBS to store backup files, skip this section and go to [Installing Velero](#).

MinIO can be installed in any of the following locations:

- Temporary ECS outside the cluster

If the MinIO server is installed outside the cluster, backup files will not be affected when a catastrophic fault occurs in the cluster.

- Idle nodes in the cluster

You can remotely log in to a node to install the MinIO server or install MinIO in a container. For details, see the official Velero documentation at <https://velero.io/docs/v1.7/contributions/minio/#set-up-server>.

### NOTICE

For example, to install MinIO in a container, run the following command:

- The storage type in the YAML file provided by Velero is **emptyDir**. You are advised to change the storage type to **HostPath** or **Local**. Otherwise, backup files will be permanently lost after the container is restarted.
- Ensure that the MinIO service is accessible externally. Otherwise, backup files cannot be downloaded outside the cluster. You can change the Service type to NodePort or use other types of public network access Services.

Regardless of which deployment method is used, the server where MinIO is installed must have sufficient storage space, an EIP must be bound to the server, and the MinIO service port must be enabled in the security group. Otherwise, backup files cannot be uploaded or downloaded.

In this example, MinIO is installed on a temporary ECS outside the cluster.

**Step 1** Download MinIO.

```
mkdir /opt/minio
mkdir /opt/miniodata
cd /opt/minio
wget https://dl.minio.io/server/minio/release/linux-amd64/minio
chmod +x minio
```

**Step 2** Set the username and password of MinIO.

The username and password set using this method are temporary environment variables and must be reset after the service is restarted. Otherwise, the default root credential **minioadmin:minioadmin** will be used to create the service.

```
export MINIO_ROOT_USER=minio
export MINIO_ROOT_PASSWORD=minio123
```

**Step 3** Create a service. In the command, **/opt/miniodata/** indicates the local disk path for MinIO to store data.

The default API port of MinIO is 9000, and the console port is randomly generated. You can use the **--console-address** parameter to specify a console port.

```
./minio server /opt/miniodata/ --console-address ":30840" &
```

 NOTE

Enable the API and console ports in the firewall and security group on the server where MinIO is to be installed. Otherwise, access to the object bucket will fail.

- Step 4** Use a browser to access `http://[EIP of the node where MinIO resides]:30840`. The MinIO console page is displayed.

----End

## Installing Velero

Go to the OBS console or MinIO console and create a bucket named **velero** to store backup files. You can custom the bucket name, which must be used when installing Velero. Otherwise, the bucket cannot be accessed and the backup fails. For details, see [Step 4](#).

 NOTICE

- Velero instances need to be installed and deployed in both the **source and target clusters**. The installation procedures are the same, which are used for backup and restoration, respectively.
- The master node of a CCE cluster does not provide a port for remote login. You can install Velero using kubectl.
- If there are a large number of resources to back up, you are advised to adjust the CPU and memory resources of Velero and Restic to 1 vCPU and 1 GB memory or higher. For details, see [Backup Tool Resources Are Insufficient](#).
- The object storage bucket for storing backup files must be **empty**.

Download the latest, stable binary file from <https://github.com/vmware-tanzu/velero/releases>. This section uses Velero 1.7.0 as an example. The installation process in the source cluster is the same as that in the target cluster.

- Step 1** Download the binary file of Velero 1.7.0.

```
wget https://github.com/vmware-tanzu/velero/releases/download/v1.7.0/velero-v1.7.0-linux-amd64.tar.gz
```

- Step 2** Install the Velero client.

```
tar -xvf velero-v1.7.0-linux-amd64.tar.gz  
cp ./velero-v1.7.0-linux-amd64/velero /usr/local/bin
```

- Step 3** Create the access key file **credentials-velero** for the backup object storage.

```
vim credentials-velero
```

Replace the AK/SK in the file based on the site requirements. If OBS is used, obtain the AK/SK by referring to [Obtaining Access Keys \(AK/SK\)](#). If MinIO is used, the AK and SK are the username and password created in [Step 2](#).

```
[default]  
aws_access_key_id = {AK}  
aws_secret_access_key = {SK}
```

- Step 4** Deploy the Velero server. Change the value of **--bucket** to the name of the created object storage bucket. In this example, the bucket name is **velero**. For more information about custom installation parameters, see [Customize Velero Install](#).

```
velero install \
--provider aws \
--plugins velero/velero-plugin-for-aws:v1.2.1 \
--bucket velero \
--secret-file ./credentials-velero \
--use-restic \
--use-volume-snapshots=false \
--backup-location-config region=ap-southeast-1,s3ForcePathStyle="true",s3Url=http://obs.ap-southeast-1.myhuaweicloud.com
```

**Table 3-5** Installation parameters of Velero

Parameter	Description
--provider	Vendor who provides the plug-in.
--plugins	API component compatible with AWS S3. Both OBS and MinIO support the S3 protocol.
--bucket	Name of the object storage bucket for storing backup files. The bucket must be created in advance.
--secret-file	Secret file for accessing the object storage, that is, the <b>credentials-velero</b> file created in <a href="#">Step 3</a> .
--use-restic	Whether to use Restic to support PV data backup. You are advised to enable this function. Otherwise, storage volume resources cannot be backed up.
--use-volume-snapshots	Whether to create the VolumeSnapshotLocation object for PV snapshot, which requires support from the snapshot program. Set this parameter to <b>false</b> .
--backup-location-config	OBS bucket configurations, including region, s3ForcePathStyle, and s3Url.
region	Region to which object storage bucket belongs. <ul style="list-style-type: none"><li>If OBS is used, set this parameter according to your region, for example, <b>ap-southeast-1</b>.</li><li>If MinIO is used, set this parameter to <b>minio</b>.</li></ul>
s3ForcePathStyle	The value <b>true</b> indicates that the S3 file path format is used.

Parameter	Description
s3Url	<p>API access address of the object storage bucket.</p> <ul style="list-style-type: none"><li>If OBS is used, set this parameter to <b>http://obs</b>. <b>{region}.myhuaweicloud.com</b> (<i>region</i> indicates the region where the object storage bucket is located). For example, if the region is Hong Kong (ap-southeast-1), the value is <b>http://obs.ap-southeast-1.myhuaweicloud.com</b>.</li><li>If MinIO is used, set this parameter to <b>http://[EIP of the node where minio is located]:9000</b>. The value of this parameter is determined based on the IP address and port of the node where MinIO is installed.</li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>The access port in s3Url must be set to the API port of MinIO instead of the console port. The default API port of MinIO is 9000.</li><li>To access MinIO installed outside the cluster, you need to enter the public IP address of MinIO.</li></ul>

**Step 5** By default, a namespace named **velero** is created for the Velero instance. Run the following command to view the pod status:

```
$ kubectl get pod -n velero
NAME           READY   STATUS    RESTARTS   AGE
restic-rn29c   1/1     Running   0          16s
velero-c9ddd56-tkzpk 1/1     Running   0          16s
```

 **NOTE**

To prevent memory insufficiency during backup in the actual production environment, you are advised to change the CPU and memory allocated to Restic and Velero by referring to [Backup Tool Resources Are Insufficient](#).

**Step 6** Check the interconnection between Velero and the object storage and ensure that the status is **Available**.

```
$ velero backup-location get
NAME   PROVIDER   BUCKET/PREFIX   PHASE   LAST VALIDATED   ACCESS MODE   DEFAULT
default  aws        velero        Available  2021-10-22 15:21:12 +0800 CST  ReadWrite   true
```

----End

### 3.2.5 Migrating Resources in a Cluster (Velero)

#### Scenario

WordPress is used as an example to describe how to migrate an application from an on-premises Kubernetes cluster to a CCE cluster. The WordPress application consists of the WordPress and MySQL components, which are containerized. The two components are bound to two local storage volumes of the Local type respectively and provide external access through the NodePort Service.

Before the migration, use a browser to access the WordPress site, create a site named **Migrate to CCE**, and publish an article to verify the integrity of PV data after the migration. The article published in WordPress will be stored in the **wp\_posts** table of the MySQL database. If the migration is successful, all contents

in the database will be migrated to the new cluster. You can verify the PV data migration based on the migration result.

## Prerequisites

- Before the migration, clear the abnormal pod resources in the source cluster. If the pod is in the abnormal state and has a PVC mounted, the PVC is in the pending state after the cluster is migrated.
- Ensure that the cluster on the CCE side does not have the same resources as the cluster to be migrated because Velero does not restore the same resources by default.
- To ensure that container image images can be properly pulled after cluster migration, migrate the images to SWR.
- CCE does not support EVS disks of the **ReadWriteMany** type. If resources of this type exist in the source cluster, change the storage type to **ReadWriteOnce**.
- Velero integrates the Restic tool to back up and restore storage volumes. Currently, the storage volumes of the HostPath type are not supported. For details, see [Restic Restrictions](#). If you need to back up storage volumes of this type, replace the hostPath volumes with local volumes by referring to [Storage Volumes of the HostPath Type Cannot Be Backed Up](#). If a backup task involves storage of the HostPath type, the storage volumes of this type will be automatically skipped and a warning message will be generated. This will not cause a backup failure.

## Backing Up Applications in the Source Cluster

**Step 1** (Optional) If you need to back up the data of a specified storage volume in the pod, add an annotation to the pod. The annotation template is as follows:

```
kubectl -n <namespace> annotate <pod/pod_name> backup.velero.io/backup-volumes=<volume_name_1>,<volume_name_2>,...
```

- **<namespace>**: namespace where the pod is located.
- **<pod\_name>**: pod name.
- **<volume\_name>**: name of the persistent volume mounted to the pod. You can run the **describe** statement to query the pod information. The **Volume** field indicates the names of all persistent volumes attached to the pod.

Add annotations to the pods of WordPress and MySQL. The pod names are **wordpress-758fbf6fc7-s7fsr** and **mysql-5ffdcbc498-c45lh**. As the pods are in the default namespace **default**, the **-n <NAMESPACE>** parameter can be omitted.

```
kubectl annotate pod/wordpress-758fbf6fc7-s7fsr backup.velero.io/backup-volumes=wp-storage
kubectl annotate pod/mysql-5ffdcbc498-c45lh backup.velero.io/backup-volumes=mysql-storage
```

**Step 2** Back up the application. During the backup, you can specify resources based on parameters. If no parameter is added, the entire cluster resources are backed up by default. For details about the parameters, see [Resource filtering](#).

- **--default-volumes-to-restic**: indicates that the Restic tool is used to back up all storage volumes mounted to the pod. Storage volumes of the HostPath type are not supported. If this parameter is not specified, the storage volume specified by annotation in **Step 1** is backed up by default. This parameter is available only when **--use-restic** is specified during [Velero installation](#).  
`velero backup create <backup-name> --default-volumes-to-restic`

- **--include-namespaces:** backs up resources in a specified namespace.  
`velero backup create <backup-name> --include-namespaces <namespace>`
- **--include-resources:** backs up the specified resources.  
`velero backup create <backup-name> --include-resources deployments`
- **--selector:** backs up resources that match the selector.  
`velero backup create <backup-name> --selector <key>=<value>`

In this section, resources in the namespace **default** are backed up. **wordpress-backup** is the backup name. You need to specify the same backup name when restoring applications. Example:

```
velero backup create wordpress-backup --include-namespaces default --default-volumes-to-restic
```

If the following information is displayed, the backup task is successfully created:

```
Backup request "wordpress-backup" submitted successfully. Run `velero backup describe wordpress-backup` or `velero backup logs wordpress-backup` for more details.
```

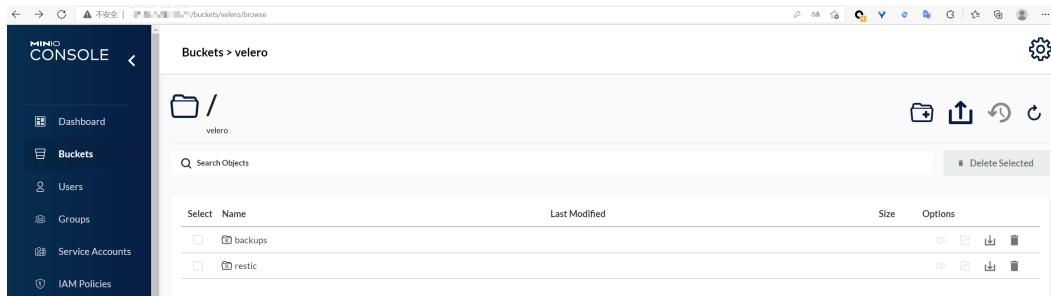
### Step 3 Check the backup status.

```
velero backup get
```

Information similar to the following is displayed:

NAME	STATUS	ERRORS	WARNINGS	CREATED	EXPIRES	STORAGE
LOCATION	SELECTOR					
wordpress-backup	Completed	0	0	2021-10-14 15:32:07 +0800 CST	29d	default
<none>						

In addition, you can go to the object bucket to view the backup files. The backups path is the application resource backup path, and the restic path is the PV data backup path.



----End

## Restoring Applications in the Target Cluster

The storage infrastructure of an on-premises cluster is different from that of a cloud cluster. After the cluster is migrated, PVs cannot be mounted to pods. Therefore, during the migration, you need to update the storage class of the target cluster to shield the differences of underlying storage interfaces between the two clusters when creating a workload and request storage resources of the corresponding type. For details, see [Updating the Storage Class](#).

### Step 1 Use kubectl to connect to the CCE cluster. Create a storage class with the same name as that of the source cluster.

In this example, the storage class name of the source cluster is **local** and the storage type is local disk. Local disks completely depend on the node availability. The data DR performance is poor. When the node is unavailable, the existing

storage data is affected. Therefore, EVS volumes are used as storage resources in CCE clusters, and SAS disks are used as backend storage media.

 NOTE

- When an application containing PV data is restored in a CCE cluster, the defined storage class dynamically creates and mounts storage resources (such as EVS volumes) based on the PVC.
- The storage resources of the cluster can be changed as required, not limited to EVS volumes. To mount other types of storage, such as file storage and object storage, see [Updating the Storage Class](#).

YAML file of the migrated cluster:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

The following is an example of the YAML file of the migration cluster:

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local
  selfLink: /apis/storage.k8s.io/v1/storageclasses/csi-disk
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi-everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS
  everest.io/passthrough: "true"
  provisioner: everest-csi-provisioner
  reclaimPolicy: Delete
  volumeBindingMode: Immediate
```

**Step 2** Use the Velero tool to create a restore and specify a backup named **wordpress-backup** to restore the WordPress application to the CCE cluster.

```
velero restore create --from-backup wordpress-backup
```

You can run the **velero restore get** statement to view the application restoration status.

**Step 3** After the restoration is complete, check whether the application is running properly. If other adaptation problems may occur, rectify the fault by following the procedure described in [Updating Resources Accordingly](#).

----End

### 3.2.6 Migrating Resources in a Cluster (e-backup)

#### Scenario

This section describes how to use Velero to back up resources in an on-premises cluster and use e-backup to restore resources in a CCE cluster.

WordPress is used as an example to describe how to migrate an application from an on-premises Kubernetes cluster to a CCE cluster. The WordPress application consists of the WordPress and MySQL components, which are containerized. The two components are bound to two local storage volumes of the Local type respectively and provide external access through the NodePort Service.

Before the migration, use a browser to access the WordPress site, create a site named **Migrate to CCE**, and publish an article to verify the integrity of PV data after the migration. The article published in WordPress will be stored in the **wp\_posts** table of the MySQL database. If the migration is successful, all contents in the database will be migrated to the new cluster. You can verify the PV data migration based on the migration result.

## Prerequisites

- Before the migration, clear the abnormal pod resources in the source cluster. If the pod is in the abnormal state and has a PVC mounted, the PVC is in the pending state after the cluster is migrated.
- Ensure that the cluster on the CCE side does not have the same resources as the cluster to be migrated because Velero does not restore the same resources by default.
- To ensure that container images can be properly pulled after cluster migration, migrate the images to SWR. For details, see [Uploading an Image Through a Container Engine Client](#).
- CCE does not support EVS disks of the **ReadWriteMany** type. If resources of this type exist in the source cluster, change the storage type to **ReadWriteOnce**.
- Velero integrates the Restic tool to back up and restore storage volumes. Currently, the storage volumes of the HostPath type are not supported. For details, see [Restic Restrictions](#). If you need to back up storage volumes of this type, replace the hostPath volumes with local volumes by referring to [Storage Volumes of the HostPath Type Cannot Be Backed Up](#). If a backup task involves storage of the HostPath type, the storage volumes of this type will be automatically skipped and a warning message will be generated. This will not cause a backup failure.

## Backing Up Applications in the Source Cluster

**Step 1** (Optional) If you need to back up the data of a specified storage volume in the pod, add an annotation to the pod. The annotation template is as follows:

```
kubectl -n <namespace> annotate <pod/pod_name> backup.velero.io/backup-volumes=<volume_name_1>,<volume_name_2>,...
```

- **<namespace>**: namespace where the pod is located.
- **<pod\_name>**: pod name.
- **<volume\_name>**: name of the persistent volume mounted to the pod. You can run the **describe** statement to query the pod information. The **Volume** field indicates the names of all persistent volumes attached to the pod.

Add annotations to the pods of WordPress and MySQL. The pod names are **wordpress-758fbf6fc7-s7fsr** and **mysql-5ffdfbc498-c45lh**. As the pods are in the default namespace **default**, the **-n <NAMESPACE>** parameter can be omitted.

```
kubectl annotate pod/wordpress-758fbf6fc7-s7fsr backup.velero.io/backup-volumes=wp-storage  
kubectl annotate pod/mysql-5ffdfbc498-c45lh backup.velero.io/backup-volumes=mysql-storage
```

**Step 2** Back up the application. During the backup, you can specify resources based on parameters. If no parameter is added, the entire cluster resources are backed up by default. For details about the parameters, see [Resource filtering](#).

- **--default-volumes-to-restic:** indicates that the Restic tool is used to back up all storage volumes mounted to the pod. Storage volumes of the HostPath type are not supported. If this parameter is not specified, the storage volume specified by annotation in **Step 1** is backed up by default. This parameter is available only when **--use-restic** is specified during **Velero installation**.  

```
velero backup create <backup-name> --default-volumes-to-restic
```
- **--include-namespaces:** backs up resources in a specified namespace.  

```
velero backup create <backup-name> --include-namespaces <namespace>
```
- **--include-resources:** backs up the specified resources.  

```
velero backup create <backup-name> --include-resources deployments
```
- **--selector:** backs up resources that match the selector.  

```
velero backup create <backup-name> --selector <key>=<value>
```

In this section, resources in the namespace **default** are backed up. **wordpress-backup** is the backup name. You need to specify the same backup name when restoring applications. Example:

```
velero backup create wordpress-backup --include-namespaces default --default-volumes-to-restic
```

If the following information is displayed, the backup task is successfully created:

Backup request "wordpress-backup" submitted successfully. Run `velero backup describe wordpress-backup` or `velero backup logs wordpress-backup` for more details.

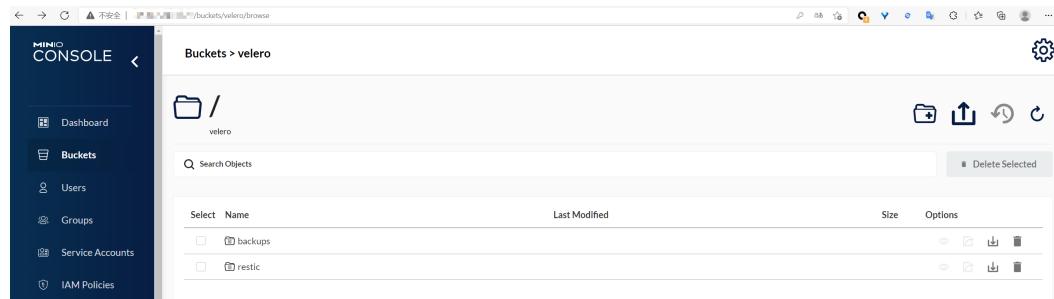
### Step 3 Check the backup status.

```
velero backup get
```

Information similar to the following is displayed:

NAME	LOCATION	STATUS	SELECTOR	ERRORS	WARNINGS	CREATED	EXPIRES	STORAGE
wordpress-backup	<none>	Completed		0	0	2021-10-14 15:32:07 +0800 CST	29d	default

In addition, you can go to the object bucket to view the backup files. The backups path is the application resource backup path, and the restic path is the PV data backup path.



----End

## Installing e-backup in the Target Cluster

CCE provides e-backup for cluster backup and restore. You can install this add-on and create a storage location to restore resources.

### Installing the e-backup add-on

#### Step 1 Log in to the CCE console. In the navigation pane, choose **Add-ons**. Locate the e-backup add-on and click **Install** under it.

**Step 2** In the **Install Add-on** drawer, select the target cluster, configure parameters, and click **Install**.

The following parameter can be configured:

**volumeWorkerNum**: number of concurrent volume backup jobs. The default value is **3**.

----End

### Creating a secret

**Step 1** Obtain an access key.

Log in to the CCE console, move the cursor to the username in the upper right corner, and choose **My Credentials**. In the navigation pane on the left, choose **Access Keys**. On the page displayed, click **Add Access Key**.

**Step 2** Create a key file and format it into a string using Base64.

```
# Create a key file.  
$ vi credential-for-huawei-obs  
HUAWEI_CLOUD_ACCESS_KEY_ID=your_access_key  
HUAWEI_CLOUD_SECRET_ACCESS_KEY=your_secret_key  
  
# Use Base64 to format the string.  
$ base64 -w 0 credential-for-huawei-obs  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXHWOBS
```

**Step 3** Create a secret.

Create a secret using the following YAML content:

```
apiVersion: v1  
kind: Secret  
metadata:  
  labels:  
    backup.everest.io/secret: 'true'  # Indicates that the secret is used by e-backup to access the backup storage location.  
  name: secret-secure-opaque  
  namespace: velero      # The value must be velero. The secret must be in the same namespace as e-backup.  
  type: cfe/secure-opaque  
data:  
  # String obtained after the credential file is Base64-encoded.  
  cloud: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXHWOBS
```

- The secret must be in the same namespace as e-backup, that is, **velero**.
- **secret.data** stores the secret for accessing OBS. The key must be **cloud**, and the value is the string Base64-encoded in **Step 2**. Generally, the displayed Base64-encoded string contains line breaks. You need to manually delete them when writing the string into **secret.data**.
- The secret must be labeled **secret.everest.io/backup: true**, indicating that the secret is used to manage the backup storage location.

----End

### Creating a storage location

Create a Kubernetes resource object used by e-backup as the backup storage location to obtain and detect information about the backend OBS.

```
apiVersion: velero.io/v1  
kind: BackupStorageLocation  
metadata:
```

```

name: backup-location-001
namespace: velero      # The object must be in the same namespace as e-backup.
spec:
  config:
    endpoint: obs.ap-southeast-1.myhuaweicloud.com # OBS endpoint
  credential:
    name: secret-secure-opaque # Name of the created secret
    key: cloud                 # Key in secret.data
  objectStorage:
    bucket: velero          # OBS bucket name
    provider: huawei           # Uses the OBS service.

```

- The **prefix** field is optional, and other fields are mandatory. The value of **provider** is fixed at **huawei**.
- You can obtain the endpoint from [Regions and Endpoints](#). Ensure that all nodes in the cluster can access the endpoint. If the endpoint does not carry a protocol header (http or https), **https** is used by default.
- Correctly set **name** and **key** in the credential. Otherwise, e-backup cannot access the backend storage location.

After the creation is complete, wait for 30 seconds for check and synchronization of the backup storage location. Then check whether **PHASE** is **Available**. The backup location is available only when the value is **Available**.

```
$ kubectl get backupstoragelocations.velero.io backup-location-001 -n velero
NAME      PHASE   LAST VALIDATED AGE  DEFAULT
backup-location-001  Available  23s      23m
```

If the value of **PHASE** does not change to **Available** for a long time, you can view e-backup logs to locate the fault. After e-backup is installed, a workload named **velero** is created in the **velero** namespace, recorded in the logs of **velero**.

The screenshot shows the Kubernetes UI with the path `/ Deployments / velero`. On the left, there are tabs for `Scheduling Policies` and `Change`. The main area is titled `View Log`. A message at the top says: "By default, 100 logs are displayed. You can go to the AOM console to view more logs or export logs to a local directory." Below this is a section titled "Log Policy" with details: "0.189 GB" for Log Usage, "Collection Policy" (not specified), and "Storage Duration" set to "day". The log viewer shows the following log entries:

```

time="2022-04-20T17:21:09+08:00" level=warning msg="There is no existing backup storage location set as default. Please see `velero backup-location -h` for options." controller=backup-storage-location logSource="pkg/controller/backup_storage_location_controller.go:173"
time="2022-04-20T17:21:09+08:00" level=warning msg="There is no existing backup storage location set as default. Please see `velero backup-location -h` for options." controller=backup-storage-location logSource="pkg/controller/backup_storage_location_controller.go:173"
time="2022-04-20T17:21:09+08:00" level=info msg="Backup storage location valid, marking as available" backup-storage-location=backup-location-001 controller=backup-storage-location logSource="pkg/controller/backup_storage_location_controller.go:121"

```

## Restoring Applications in the Target Cluster (e-backup)

Perform the following steps to restore a cluster in CCE using e-backup:

Use an immediate backup as the data source and restore data to another cluster. This mode applies to all scenarios.

You can use the Restore manifest below and run the **kubectl create** command to create a backup deletion request.

```

apiVersion: velero.io/v1
kind: Restore
metadata:

```

```
name: restore-01
namespace: velero
spec:
  backupName: wordpress-backup
  includedNamespaces:
    - default
  storageClassMapping:
    local: csi-disk
  imageRepositoryMapping:
    quay.io/coreos: swr.ap-southeast-1.myhuaweicloud.com/everest
```

- **backupName:** (mandatory) immediate backup that is used as the data source.
- **storageClassMapping:** changes the storageClassName used by backup resources such as PVs and PVCs. The storageClass types must be the same. In this example, **local** is changed to **csi-disk** supported by CCE.
- **imageRepositoryMapping:** changes the **images** field of the backup. It is used for repository mapping, excluding the change of the image name and tag (to prevent the migration and upgrade from being coupled). For example, after you migrate **quay.io/coreos/etcdb:2.5** to SWR, you can use **swr.ap-southeast-1.myhuaweicloud.com/everest/etcdb:2.5** in the local image repository. The configuration format is as follows: **quay.io/coreos: swr.ap-southeast-1.myhuaweicloud.com/everest**

If **storageClassMapping** and **imageRepositoryMapping** are configured, you can skip their configuration in [Updating Images](#) and [Updating the Storage Class](#).

For details about other parameters, see [e-backup](#).

After the restore is performed, run the following command to **check the task status**:

```
$ kubectl -n velero get restores restore-01 -o yaml | grep " phase"
phase: Completed
```

If the status is **Completed**, the restore is complete. You can view the application restore details on the CCE console.

### 3.2.7 Updating Resources Accordingly

#### Updating Images

The WordPress and MySQL images used in this example can be pulled from SWR. Therefore, the image pull failure (ErrImagePull) will not occur. If the application to be migrated is created from a private image, perform the following steps to update the image:

- Step 1** Migrate the image resources to SWR. For details, see [Uploading an Image Through a Container Engine Client](#).
- Step 2** Log in to the SWR console and obtain the image path used after the migration.

The image path is in the following format:  
`'swr.{Region}.myhuaweicloud.com/{Organization name}/{Image name}:{Tag name}'`

- Step 3** Run the following command to modify the workload and replace the **image** field in the YAML file with the image path:

```
kubectl edit deploy wordpress
```

**Step 4** Check the running status of the workload.

----End

## Updating Services

After the cluster is migrated, the Service of the source cluster may fail to take effect. You can perform the following steps to update the Service. If ingresses are configured in the source cluster, you need to connect the new cluster to ELB again after the migration. For details, see [Creating an Ingress - Interconnecting with an Existing Load Balancer](#).

**Step 1** Connect to the cluster using kubectl.

**Step 2** Edit the YAML file of the corresponding Service to change the Service type and port number.

```
kubectl edit svc wordpress
```

To update load balancer resources, you need to connect to ELB again. Add the annotations by following the procedure described in [Creating an Ingress - Interconnecting with an Existing Load Balancer](#).

annotations:

```
kubernetes.io/elb.class: union # Shared load balancer
kubernetes.io/elb.id: 9d06a39d-xxxx-xxxx-xxxx-c204397498a3 # Load balancer ID, which can be queried on the ELB console.
kubernetes.io/elb.subnet-id: f86ba71c-xxxx-xxxx-xxxx-39c8a7d4bb36 # ID of the cluster where the subnet resides
kubernetes.io/session-affinity-mode: SOURCE_IP # Enable the sticky session based on the source IP address.
```

**Step 3** Use a browser to check whether the Service is available.

----End

## Updating the Storage Class

As the storage infrastructures of clusters may be different, storage volumes cannot be mounted to the target cluster. You can use either of the following methods to update the volumes:

### NOTICE

Both update methods can be performed only before the application is restored in the target cluster. Otherwise, PV data resources may fail to be restored. In this case, use the Velero to restore applications after the storage class update is complete. For details, see [Restoring Applications in the Target Cluster](#).

### Method 1: Creating a ConfigMap mapping

**Step 1** Create a ConfigMap in the CCE cluster and map the storage class used by the source cluster to the default storage class of the CCE cluster.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: change-storageclass-plugin-config
  namespace: velero
  labels:
```

```

app.kubernetes.io/name: velero
velero.io/plugin-config: "true"
velero.io/change-storage-class: RestoreItemAction
data:
  {Storage class name01 in the source cluster}: {Storage class name01 in the target cluster}
  {Storage class name02 in the source cluster}: {Storage class name02 in the target cluster}

```

- Step 2** Run the following command to apply the ConfigMap configuration:

```
$ kubectl create -f change-storage-class.yaml
configmap/change-storageclass-plugin-config created
```

----End

### Method 2: Creating a storage class with the same name

- Step 1** Run the following command to query the default storage class supported by CCE:  
 kubectl get sc

Information similar to the following is displayed:

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	
ALLOWVOLUMEEXPANSION	AGE			
csi-disk	everest-csi-provisioner	Delete	Immediate	3d23h
csi-disk-topology	everest-csi-provisioner	Delete	WaitForFirstConsumer	3d23h
csi-nas	everest-csi-provisioner	Delete	Immediate	3d23h
csi-obs	everest-csi-provisioner	Delete	Immediate	3d23h
csi-sfsturbo	everest-csi-provisioner	Delete	false	3d23h
			true	3d23h

**Table 3-6** Storage classes

Storage Class	Storage Resource
csi-disk	EVS
csi-disk-topology	EVS with delayed binding
csi-nas	SFS
csi-obs	OBS
csi-sfsturbo	SFS Turbo

- Step 2** Run the following command to export the required storage class details in YAML format:

```
kubectl get sc <storageclass-name> -o=yaml
```

- Step 3** Copy the YAML file and create a new storage class.

Change the storage class name to the name used in the source cluster to call basic storage resources of the cloud.

The YAML file of csi-obs is used as an example. Delete the unnecessary information in italic under the **metadata** field and modify the information in bold. You are advised not to modify other parameters.

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  creationTimestamp: "2021-10-18T06:41:36Z"
  name: <your_storageclass_name> # Use the name of the storage class used in the source cluster.
  resourceVersion: "747"
  selfLink: /apis/storage.k8s.io/v1/storageclasses/csi-obs
  uid: 4dbbe557-ddd1-4ce8-bb7b-7fa15459aac7

```

```
parameters:  
  csi.storage.k8s.io/csi-driver-name: obs.csi-everest.io  
  csi.storage.k8s.io/fstype: obsfs  
  everest.io/obs-volume-type: STANDARD  
  provisioner: everest-csi-provisioner  
  reclaimPolicy: Delete  
  volumeBindingMode: Immediate
```

#### NOTE

- SFS Turbo file systems cannot be directly created using StorageClass. You need to go to the SFS Turbo console to create SFS Turbo file systems that belong to the same VPC subnet and have inbound ports (111, 445, 2049, 2051, 2052, and 20048) enabled in the security group.
- CCE does not support EVS disks of the `ReadWriteMany` type. If resources of this type exist in the source cluster, change the storage type to `ReadWriteOnce`.

**Step 4** Restore the cluster application by referring to [Restoring Applications in the Target Cluster](#) and check whether the PVC is successfully created.

```
kubectl get pvc
```

In the command output, the **VOLUME** column indicates the name of the PV automatically created using the storage class.

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
pvc	Bound	pvc-4c8e655a-1dbc-4897-ae6c-446b502f5e77	5Gi	RWX	local	13s

----End

## Updating Databases

In this example, the database is a local MySQL database and does not need to be reconfigured after the migration. If you use **DRS** to migrate a local database to RDS, you need to configure database access based on site requirements after the migration.

#### NOTE

- If the RDS instance is in the same VPC as the CCE cluster, it can be accessed using the private IP address. Otherwise, it can only be accessed only through public networks by binding an EIP. You are advised to use the private network access mode for high security and good RDS performance.
- Ensure that the inbound rule of the security group to which RDS belongs has been enabled for the cluster. Otherwise, the connection will fail.

**Step 1** Log in to the RDS console and obtain the private IP address and port number of the DB instance on the **Basic Information** page.

**Step 2** Run the following command to modify the WordPress workload:

```
kubectl edit deploy wordpress
```

Set the environment variables in the `env` field.

- **WORDPRESS\_DB\_HOST**: address and port number used for accessing the database, that is, the internal network address and port number obtained in the previous step.
- **WORDPRESS\_DB\_USER**: username for accessing the database.
- **WORDPRESS\_DB\_PASSWORD**: password for accessing the database.
- **WORDPRESS\_DB\_NAME**: name of the database to be connected.

**Step 3** Check whether the RDS database is properly connected.

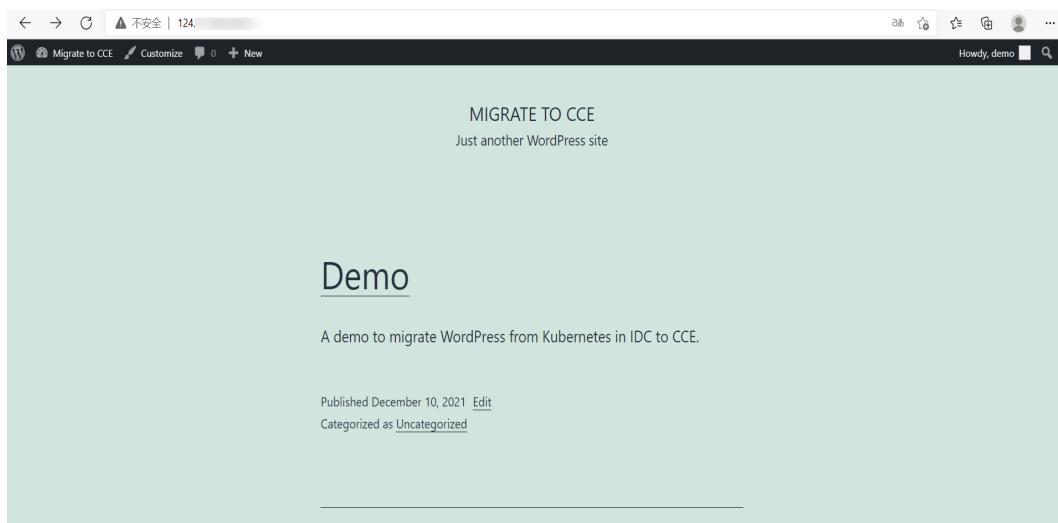
----End

### 3.2.8 Performing Additional Tasks

#### Verifying Application Functions

Cluster migration involves full migration of application data, which may cause intra-application adaptation problems. In this example, after the cluster is migrated, the redirection link of the article published in WordPress is still the original domain name. If you click the article title, you will be redirected to the application in the source cluster. Therefore, you need to search for the original domain name in WordPress and replace it with the new domain name, change the values of **site\_url** and primary URL in the database. For details, see [Changing The Site URL](#).

Access the new address of the WordPress application. If the article published before the migration is displayed, the data of the persistent volume is successfully restored.



#### Switching Live Traffic to the Target Cluster

O&M personnel switch DNS to direct live traffic to the target cluster.

- DNS traffic switching: Adjust the DNS configuration to switch traffic.
- Client traffic switching: Upgrade the client code or update the configuration to switch traffic.

#### Bringing the Source Cluster Offline

After confirming that the service on the target cluster is normal, bring the source cluster offline and delete the backup files.

- Verify that the service on the target cluster is running properly.
- Bring the source cluster offline.

- Delete backup files.

### 3.2.9 Troubleshooting

#### Storage Volumes of the HostPath Type Cannot Be Backed Up

Both HostPath and Local volumes are local storage volumes. However, the Restic tool integrated in Velero cannot back up the PVs of the HostPath type and supports only the Local type. Therefore, you need to replace the storage volumes of the HostPath type with the Local type in the source cluster.



It is recommended that Local volumes be used in Kubernetes v1.10 or later and can only be statically created. For details, see [local](#).

##### Step 1 Create a storage class for the Local volume.

Example YAML:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

##### Step 2 Change the **hostPath** field to the **local** field, specify the original local disk path of the host machine, and add the **nodeAffinity** field.

Example YAML:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
  labels:
    app: mysql
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 5Gi
  storageClassName: local    # Storage class created in the previous step
  persistentVolumeReclaimPolicy: Delete
  local:
    path: "/mnt/data"      # Path of the attached local disk
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: Exists
```

##### Step 3 Run the following commands to verify the creation result:

```
kubectl get pv
```

Information similar to the following is displayed:

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS
REASON	AGE					
mysql-pv	5Gi	RWO	Delete	Available	local	3s

----End

## Backup Tool Resources Are Insufficient

In the production environment, if there are many backup resources, for example, the default resource size of the backup tool is used, the resources may be insufficient. In this case, perform the following steps to adjust the CPU and memory size allocated to the Velero and Restic:

### Before installing Velero:

You can specify the size of resources used by Velero and Restic when [installing Velero](#).

The following is an example of installation parameters:

```
velero install \
  --velero-pod-cpu-request 500m \
  --velero-pod-mem-request 1Gi \
  --velero-pod-cpu-limit 1000m \
  --velero-pod-mem-limit 1Gi \
  --use-restic \
  --restic-pod-cpu-request 500m \
  --restic-pod-mem-request 1Gi \
  --restic-pod-cpu-limit 1000m \
  --restic-pod-mem-limit 1Gi
```

### After Velero is installed:

#### Step 1 Edit the YAML files of the Velero and Restic workloads in the **velero** namespace.

```
kubectl edit deploy velero -n velero
kubectl edit deploy restic -n velero
```

#### Step 2 Modify the resource size under the **resources** field. The modification is the same for the Velero and Restic workloads, as shown in the following:

```
resources:
  limits:
    cpu: "1"
    memory: 1Gi
  requests:
    cpu: 500m
    memory: 1Gi
```

----End

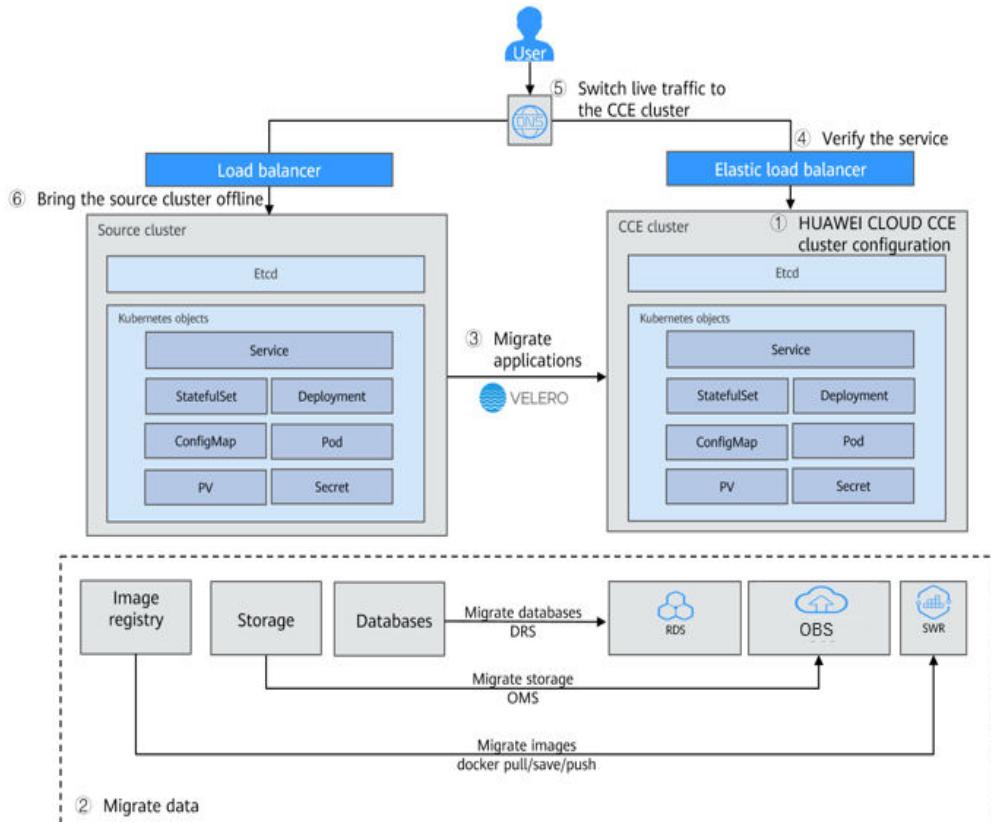
## 3.3 Migrating Clusters from Other Clouds to CCE

### 3.3.1 Solution Overview

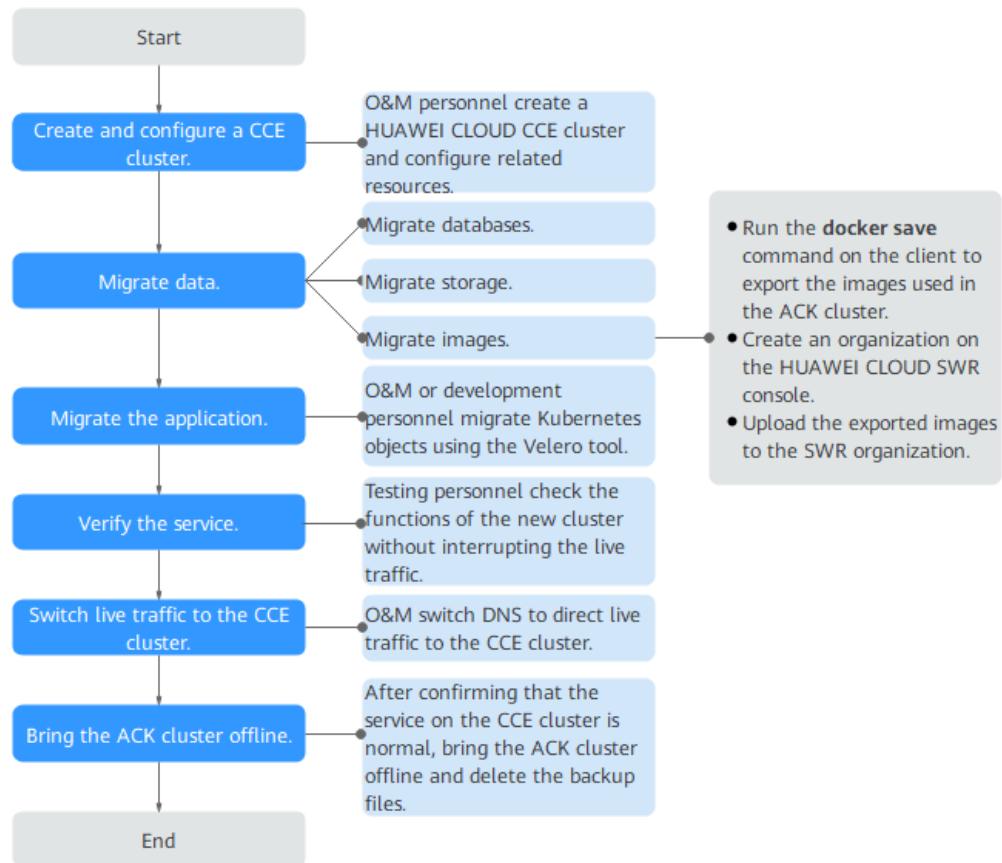
This section takes the WordPress application as an example to describe how to migrate an application from Alibaba Cloud ACK to Huawei Cloud CCE. Assume that you have deployed the WordPress application on Alibaba Cloud and created your own blog.

This document briefly describes how to smoothly migrate an application from an Alibaba Cloud ACK cluster to a Huawei Cloud CCE cluster in six steps without interrupting the service.

## Migration Scheme



## Procedure



### 3.3.2 Resource and Cost Planning

#### NOTICE

The fees listed here are estimates. The actual fees will be displayed on the Huawei Cloud console.

The required resources are as follows:

**Table 3-7** Resource and cost planning

Resource	Description	Quantity	Monthly Fee (CNY)
Cloud Container Engine (CCE)	<ul style="list-style-type: none"><li>CCE cluster version: v1.21</li><li>Minimum node specifications: 4 vCPUs, 8 GB memory, EulerOS 2.9</li><li>Pay-per-use recommended</li></ul>	1	0.55/hour

### 3.3.3 Procedure

#### 3.3.3.1 Migrating Data

##### Migrating Databases and Storage

###### Database migration

O&M personnel or development personnel migrate databases using Data Replication Service (DRS). For details, see [Migrating Databases Across Cloud Platforms](#).

###### Storage migration

O&M personnel or development personnel migrate data in object storage using Object Storage Migration Service (OMS). For details on OMS, see [Object Storage Migration Service](#).



Currently, you can use OMS to migrate object storage data from Amazon Web Services (AWS), Alibaba Cloud, Microsoft Azure, Baidu Cloud, Kingsoft Cloud, QingCloud, Qiniu Cloud, and Tencent Cloud to Huawei Cloud [OBS](#).

- Create a bucket on OBS. For details, see [Creating a Bucket](#).
- Create a migration task on OMS. For details, see [Creating an Object Storage Migration Task](#).

##### Migrating Container Images

**Step 1** Export the container images used in ACK clusters.

Pull the images to the client by referring to the operation guide of Alibaba Cloud Container Registry (ACR).

**Step 2** Upload the image files to Huawei Cloud SWR.

Run the `docker pull` command to push the image to Huawei Cloud. For details, see [Uploading an Image Through a Container Engine Client](#).

----End

#### 3.3.3.2 Installing the Migration Tool

Velero is an open-source backup and migration tool for Kubernetes clusters. It integrates the persistent volume (PV) data backup capability of the Restic tool and can be used to back up Kubernetes resource objects (such as Deployments, jobs, Services, and ConfigMaps) in the source cluster. Data in the PV mounted to the pod is backed up and uploaded to the object storage. When a disaster occurs or migration is required, the target cluster can use Velero to obtain the corresponding backup data from OBS and restore cluster resources as required.

According to [Migration Solution](#), you need to prepare temporary object storage to store backup files before the migration. Velero supports OSB or [MinIO](#) as the object storage. OBS requires sufficient storage space for storing backup files. You can estimate the storage space based on your cluster scale and data volume. You

are advised to use OBS for backup. For details about how to deploy Velero, see [Installing Velero](#).

CCE supports backup and restore using the e-backup add-on, which is compatible with Velero and uses OBS as the storage backend. You can use Velero in on-premises clusters and use e-backup in CCE.

- Without e-backup: Install Velero in the source and target clusters by following the instructions described in this topic, and migrate resources by referring to [Migrating Resources in a Cluster \(Velero\)](#).
- With e-backup: Install Velero in the source cluster and use OBS as the storage backend by following the instructions described in [Installing Velero](#), and install e-backup in the target CCE cluster and migrate resources by referring to [Migrating Resources in a Cluster \(e-backup\)](#).

## Prerequisites

- The Kubernetes version of the source on-premises cluster must be 1.10 or later, and the cluster can use DNS and Internet services properly.
- If you use OBS to store backup files, you need to obtain the AK/SK of a user who has the right to operate OBS. For details about how to obtain the AK/SK, see [Obtaining Access Keys \(AK/SK\)](#).
- If you use MinIO to store backup files, bind an EIP to the server where MinIO is installed and enable the API and console port of MinIO in the security group.
- The target CCE cluster has been created.
- The source cluster and target cluster must each have at least one idle node. It is recommended that the node specifications be 4 vCPUs and 8 GB memory or higher.

## Installing MinIO

MinIO is an open-source, high-performance object storage tool compatible with the S3 API protocol. If MinIO is used to store backup files for cluster migration, you need a temporary server to deploy MinIO and provide services for external systems. If you use OBS to store backup files, skip this section and go to [Installing Velero](#).

MinIO can be installed in any of the following locations:

- Temporary ECS outside the cluster  
If the MinIO server is installed outside the cluster, backup files will not be affected when a catastrophic fault occurs in the cluster.
- Idle nodes in the cluster  
You can remotely log in to a node to install the MinIO server or install MinIO in a container. For details, see the official Velero documentation at <https://velero.io/docs/v1.7/contributions/minio/#set-up-server>.

### NOTICE

For example, to install MinIO in a container, run the following command:

- The storage type in the YAML file provided by Velero is **emptyDir**. You are advised to change the storage type to **HostPath** or **Local**. Otherwise, backup files will be permanently lost after the container is restarted.
- Ensure that the MinIO service is accessible externally. Otherwise, backup files cannot be downloaded outside the cluster. You can change the Service type to NodePort or use other types of public network access Services.

Regardless of which deployment method is used, the server where MinIO is installed must have sufficient storage space, an EIP must be bound to the server, and the MinIO service port must be enabled in the security group. Otherwise, backup files cannot be uploaded or downloaded.

In this example, MinIO is installed on a temporary ECS outside the cluster.

**Step 1** Download MinIO.

```
mkdir /opt/minio
mkdir /opt/miniodata
cd /opt/minio
wget https://dl.minio.io/server/minio/release/linux-amd64/minio
chmod +x minio
```

**Step 2** Set the username and password of MinIO.

The username and password set using this method are temporary environment variables and must be reset after the service is restarted. Otherwise, the default root credential **minioadmin:minioadmin** will be used to create the service.

```
export MINIO_ROOT_USER=minio
export MINIO_ROOT_PASSWORD=minio123
```

**Step 3** Create a service. In the command, **/opt/miniodata/** indicates the local disk path for MinIO to store data.

The default API port of MinIO is 9000, and the console port is randomly generated. You can use the **--console-address** parameter to specify a console port.  
.minio server /opt/miniodata/ --console-address ":30840" &

### NOTE

Enable the API and console ports in the firewall and security group on the server where MinIO is to be installed. Otherwise, access to the object bucket will fail.

**Step 4** Use a browser to access `http://[EIP of the node where MinIO resides]:30840`. The MinIO console page is displayed.

----End

## Installing Velero

Go to the OBS console or MinIO console and create a bucket named **velero** to store backup files. You can custom the bucket name, which must be used when installing Velero. Otherwise, the bucket cannot be accessed and the backup fails. For details, see **Step 4**.

**NOTICE**

- Velero instances need to be installed and deployed in both the **source and target clusters**. The installation procedures are the same, which are used for backup and restoration, respectively.
- The master node of a CCE cluster does not provide a port for remote login. You can install Velero using kubectl.
- If there are a large number of resources to back up, you are advised to adjust the CPU and memory resources of Velero and Restic to 1 vCPU and 1 GB memory or higher. For details, see [Backup Tool Resources Are Insufficient](#).
- The object storage bucket for storing backup files must be **empty**.

Download the latest, stable binary file from <https://github.com/vmware-tanzu/velero/releases>. This section uses Velero 1.7.0 as an example. The installation process in the source cluster is the same as that in the target cluster.

**Step 1** Download the binary file of Velero 1.7.0.

```
wget https://github.com/vmware-tanzu/velero/releases/download/v1.7.0/velero-v1.7.0-linux-amd64.tar.gz
```

**Step 2** Install the Velero client.

```
tar -xvf velero-v1.7.0-linux-amd64.tar.gz  
cp ./velero-v1.7.0-linux-amd64/velero /usr/local/bin
```

**Step 3** Create the access key file **credentials-velero** for the backup object storage.

```
vim credentials-velero
```

Replace the AK/SK in the file based on the site requirements. If OBS is used, obtain the AK/SK by referring to [Obtaining Access Keys \(AK/SK\)](#). If MinIO is used, the AK and SK are the username and password created in [Step 2](#).

```
[default]  
aws_access_key_id = {AK}  
aws_secret_access_key = {SK}
```

**Step 4** Deploy the Velero server. Change the value of **--bucket** to the name of the created object storage bucket. In this example, the bucket name is **velero**. For more information about custom installation parameters, see [Customize Velero Install](#).

```
velero install \  
  --provider aws \  
  --plugins velero/velero-plugin-for-aws:v1.2.1 \  
  --bucket velero \  
  --secret-file ./credentials-velero \  
  --use-restic \  
  --use-volume-snapshots=false \  
  --backup-location-config region=ap-southeast-1,s3ForcePathStyle="true",s3Url=http://obs.ap-  
southeast-1.myhuaweicloud.com
```

**Table 3-8** Installation parameters of Velero

Parameter	Description
--provider	Vendor who provides the plug-in.
--plugins	API component compatible with AWS S3. Both OBS and MinIO support the S3 protocol.

Parameter	Description
--bucket	Name of the object storage bucket for storing backup files. The bucket must be created in advance.
--secret-file	Secret file for accessing the object storage, that is, the <b>credentials-velero</b> file created in <b>Step 3</b> .
--use-restic	Whether to use Restic to support PV data backup. You are advised to enable this function. Otherwise, storage volume resources cannot be backed up.
--use-volume-snapshots	Whether to create the VolumeSnapshotLocation object for PV snapshot, which requires support from the snapshot program. Set this parameter to <b>false</b> .
--backup-location-config	OBS bucket configurations, including region, s3ForcePathStyle, and s3Url.
region	Region to which object storage bucket belongs. <ul style="list-style-type: none"><li>If OBS is used, set this parameter according to your region, for example, <b>ap-southeast-1</b>.</li><li>If MinIO is used, set this parameter to <b>minio</b>.</li></ul>
s3ForcePathStyle	The value <b>true</b> indicates that the S3 file path format is used.
s3Url	API access address of the object storage bucket. <ul style="list-style-type: none"><li>If OBS is used, set this parameter to <b>http://obs.{region}.myhuaweicloud.com</b> (<i>region</i> indicates the region where the object storage bucket is located). For example, if the region is Hong Kong (ap-southeast-1), the value is <b>http://obs.ap-southeast-1.myhuaweicloud.com</b>.</li><li>If MinIO is used, set this parameter to <b>http://{{IP of the node where minio is located}}:9000</b>. The value of this parameter is determined based on the IP address and port of the node where MinIO is installed.</li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>The access port in s3Url must be set to the API port of MinIO instead of the console port. The default API port of MinIO is 9000.</li><li>To access MinIO installed outside the cluster, you need to enter the public IP address of MinIO.</li></ul>

**Step 5** By default, a namespace named **velero** is created for the Velero instance. Run the following command to view the pod status:

```
$ kubectl get pod -n velero
NAME          READY   STATUS    RESTARTS   AGE
restic-rn29c  1/1     Running   0          16s
velero-c9ddd56-tkzpk  1/1     Running   0          16s
```

 NOTE

To prevent memory insufficiency during backup in the actual production environment, you are advised to change the CPU and memory allocated to Restic and Velero by referring to [Backup Tool Resources Are Insufficient](#).

**Step 6** Check the interconnection between Velero and the object storage and ensure that the status is **Available**.

```
$ velero backup-location get
NAME      PROVIDER   BUCKET/PREFIX   PHASE      LAST VALIDATED          ACCESS MODE   DEFAULT
default    aws        velero        Available  2021-10-22 15:21:12 +0800 CST  ReadWrite     true
```

----End

### 3.3.3.3 Migrating Resources in a Cluster (Velero)

#### Scenario

WordPress is used as an example to describe how to migrate an application from an on-premises Kubernetes cluster to a CCE cluster. The WordPress application consists of the WordPress and MySQL components, which are containerized. The two components are bound to two local storage volumes of the Local type respectively and provide external access through the NodePort Service.

Before the migration, use a browser to access the WordPress site, create a site named **Migrate to CCE**, and publish an article to verify the integrity of PV data after the migration. The article published in WordPress will be stored in the **wp\_posts** table of the MySQL database. If the migration is successful, all contents in the database will be migrated to the new cluster. You can verify the PV data migration based on the migration result.

#### Prerequisites

- Before the migration, clear the abnormal pod resources in the source cluster. If the pod is in the abnormal state and has a PVC mounted, the PVC is in the pending state after the cluster is migrated.
- Ensure that the cluster on the CCE side does not have the same resources as the cluster to be migrated because Velero does not restore the same resources by default.
- To ensure that container image images can be properly pulled after cluster migration, migrate the images to SWR.
- CCE does not support EVS disks of the **ReadWriteMany** type. If resources of this type exist in the source cluster, change the storage type to **ReadWriteOnce**.
- Velero integrates the Restic tool to back up and restore storage volumes. Currently, the storage volumes of the HostPath type are not supported. For details, see [Restic Restrictions](#). If you need to back up storage volumes of this type, replace the hostPath volumes with local volumes by referring to [Storage Volumes of the HostPath Type Cannot Be Backed Up](#). If a backup task involves storage of the HostPath type, the storage volumes of this type will be automatically skipped and a warning message will be generated. This will not cause a backup failure.

## Backing Up Applications in the Source Cluster

**Step 1** (Optional) If you need to back up the data of a specified storage volume in the pod, add an annotation to the pod. The annotation template is as follows:

```
kubectl -n <namespace> annotate <pod/pod_name> backup.velero.io/backup-volumes=<volume_name_1>,<volume_name_2>,...
```

- <namespace>: namespace where the pod is located.
- <pod\_name>: pod name.
- <volume\_name>: name of the persistent volume mounted to the pod. You can run the **describe** statement to query the pod information. The **Volume** field indicates the names of all persistent volumes attached to the pod.

Add annotations to the pods of WordPress and MySQL. The pod names are **wordpress-758fbf6fc7-s7fsr** and **mysql-5ffdfbc498-c45lh**. As the pods are in the default namespace **default**, the **-n <NAMESPACE>** parameter can be omitted.

```
kubectl annotate pod/wordpress-758fbf6fc7-s7fsr backup.velero.io/backup-volumes=wp-storage  
kubectl annotate pod/mysql-5ffdfbc498-c45lh backup.velero.io/backup-volumes=mysql-storage
```

**Step 2** Back up the application. During the backup, you can specify resources based on parameters. If no parameter is added, the entire cluster resources are backed up by default. For details about the parameters, see [Resource filtering](#).

- **--default-volumes-to-restic**: indicates that the Restic tool is used to back up all storage volumes mounted to the pod. Storage volumes of the HostPath type are not supported. If this parameter is not specified, the storage volume specified by annotation in **Step 1** is backed up by default. This parameter is available only when **--use-restic** is specified during [Velero installation](#).  
`velero backup create <backup-name> --default-volumes-to-restic`
- **--include-namespaces**: backs up resources in a specified namespace.  
`velero backup create <backup-name> --include-namespaces <namespace>`
- **--include-resources**: backs up the specified resources.  
`velero backup create <backup-name> --include-resources deployments`
- **--selector**: backs up resources that match the selector.  
`velero backup create <backup-name> --selector <key>=<value>`

In this section, resources in the namespace **default** are backed up. **wordpress-backup** is the backup name. You need to specify the same backup name when restoring applications. Example:

```
velero backup create wordpress-backup --include-namespaces default --default-volumes-to-restic
```

If the following information is displayed, the backup task is successfully created:

```
Backup request "wordpress-backup" submitted successfully. Run `velero backup describe wordpress-backup` or `velero backup logs wordpress-backup` for more details.
```

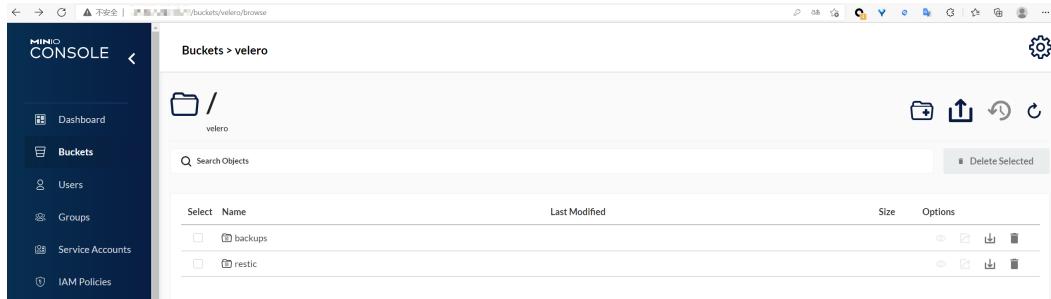
**Step 3** Check the backup status.

```
velero backup get
```

Information similar to the following is displayed:

NAME	STATUS	ERRORS	WARNINGS	CREATED	EXPIRES	STORAGE
LOCATION	SELECTOR					
wordpress-backup	Completed	0	0	2021-10-14 15:32:07 +0800 CST	29d	default
<none>						

In addition, you can go to the object bucket to view the backup files. The backups path is the application resource backup path, and the restic path is the PV data backup path.



----End

## Restoring Applications in the Target Cluster

The storage infrastructure of an on-premises cluster is different from that of a cloud cluster. After the cluster is migrated, PVs cannot be mounted to pods. Therefore, during the migration, you need to update the storage class of the target cluster to shield the differences of underlying storage interfaces between the two clusters when creating a workload and request storage resources of the corresponding type. For details, see [Updating the Storage Class](#).

- Step 1** Use kubectl to connect to the CCE cluster. Create a storage class with the same name as that of the source cluster.

In this example, the storage class name of the source cluster is **local** and the storage type is local disk. Local disks completely depend on the node availability. The data DR performance is poor. When the node is unavailable, the existing storage data is affected. Therefore, EVS volumes are used as storage resources in CCE clusters, and SAS disks are used as backend storage media.

### NOTE

- When an application containing PV data is restored in a CCE cluster, the defined storage class dynamically creates and mounts storage resources (such as EVS volumes) based on the PVC.
- The storage resources of the cluster can be changed as required, not limited to EVS volumes. To mount other types of storage, such as file storage and object storage, see [Updating the Storage Class](#).

YAML file of the migrated cluster:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

The following is an example of the YAML file of the migration cluster:

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local
  selfLink: /apis/storage.k8s.io/v1/storageclasses/csi-disk
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS
  everest.io/passthrough: "true"
```

```
provisioner: everest-csi-provisioner  
reclaimPolicy: Delete  
volumeBindingMode: Immediate
```

- Step 2** Use the Velero tool to create a restore and specify a backup named **wordpress-backup** to restore the WordPress application to the CCE cluster.

```
velero restore create --from-backup wordpress-backup
```

You can run the **velero restore get** statement to view the application restoration status.

- Step 3** After the restoration is complete, check whether the application is running properly. If other adaptation problems may occur, rectify the fault by following the procedure described in [Updating Resources Accordingly](#).

----End

### 3.3.3.4 Migrating Resources in a Cluster (e-backup)

#### Scenario

This section describes how to use Velero to back up resources in an on-premises cluster and use e-backup to restore resources in a CCE cluster.

WordPress is used as an example to describe how to migrate an application from an on-premises Kubernetes cluster to a CCE cluster. The WordPress application consists of the WordPress and MySQL components, which are containerized. The two components are bound to two local storage volumes of the Local type respectively and provide external access through the NodePort Service.

Before the migration, use a browser to access the WordPress site, create a site named **Migrate to CCE**, and publish an article to verify the integrity of PV data after the migration. The article published in WordPress will be stored in the **wp\_posts** table of the MySQL database. If the migration is successful, all contents in the database will be migrated to the new cluster. You can verify the PV data migration based on the migration result.

#### Prerequisites

- Before the migration, clear the abnormal pod resources in the source cluster. If the pod is in the abnormal state and has a PVC mounted, the PVC is in the pending state after the cluster is migrated.
- Ensure that the cluster on the CCE side does not have the same resources as the cluster to be migrated because Velero does not restore the same resources by default.
- To ensure that container images can be properly pulled after cluster migration, migrate the images to SWR. For details, see [Uploading an Image Through a Container Engine Client](#).
- CCE does not support EVS disks of the **ReadWriteMany** type. If resources of this type exist in the source cluster, change the storage type to **ReadWriteOnce**.
- Velero integrates the Restic tool to back up and restore storage volumes. Currently, the storage volumes of the HostPath type are not supported. For details, see [Restic Restrictions](#). If you need to back up storage volumes of this type, replace the hostPath volumes with local volumes by referring to

**Storage Volumes of the HostPath Type Cannot Be Backed Up.** If a backup task involves storage of the HostPath type, the storage volumes of this type will be automatically skipped and a warning message will be generated. This will not cause a backup failure.

## Backing Up Applications in the Source Cluster

**Step 1** (Optional) If you need to back up the data of a specified storage volume in the pod, add an annotation to the pod. The annotation template is as follows:

```
kubectl -n <namespace> annotate <pod/pod_name> backup.velero.io/backup-volumes=<volume_name_1>,<volume_name_2>,...
```

- **<namespace>**: namespace where the pod is located.
- **<pod\_name>**: pod name.
- **<volume\_name>**: name of the persistent volume mounted to the pod. You can run the **describe** statement to query the pod information. The **Volume** field indicates the names of all persistent volumes attached to the pod.

Add annotations to the pods of WordPress and MySQL. The pod names are **wordpress-758fbf6fc7-s7fsr** and **mysql-5ffdfbc498-c45lh**. As the pods are in the default namespace **default**, the **-n <NAMESPACE>** parameter can be omitted.

```
kubectl annotate pod/wordpress-758fbf6fc7-s7fsr backup.velero.io/backup-volumes=wp-storage
kubectl annotate pod/mysql-5ffdfbc498-c45lh backup.velero.io/backup-volumes=mysql-storage
```

**Step 2** Back up the application. During the backup, you can specify resources based on parameters. If no parameter is added, the entire cluster resources are backed up by default. For details about the parameters, see [Resource filtering](#).

- **--default-volumes-to-restic**: indicates that the Restic tool is used to back up all storage volumes mounted to the pod. Storage volumes of the HostPath type are not supported. If this parameter is not specified, the storage volume specified by annotation in **Step 1** is backed up by default. This parameter is available only when **--use-restic** is specified during [Velero installation](#).  
velero backup create <backup-name> --default-volumes-to-restic
- **--include-namespaces**: backs up resources in a specified namespace.  
velero backup create <backup-name> --include-namespaces <namespace>
- **--include-resources**: backs up the specified resources.  
velero backup create <backup-name> --include-resources deployments
- **--selector**: backs up resources that match the selector.  
velero backup create <backup-name> --selector <key>=<value>

In this section, resources in the namespace **default** are backed up. **wordpress-backup** is the backup name. You need to specify the same backup name when restoring applications. Example:

```
velero backup create wordpress-backup --include-namespaces default --default-volumes-to-restic
```

If the following information is displayed, the backup task is successfully created:

```
Backup request "wordpress-backup" submitted successfully. Run `velero backup describe wordpress-backup` or `velero backup logs wordpress-backup` for more details.
```

**Step 3** Check the backup status.

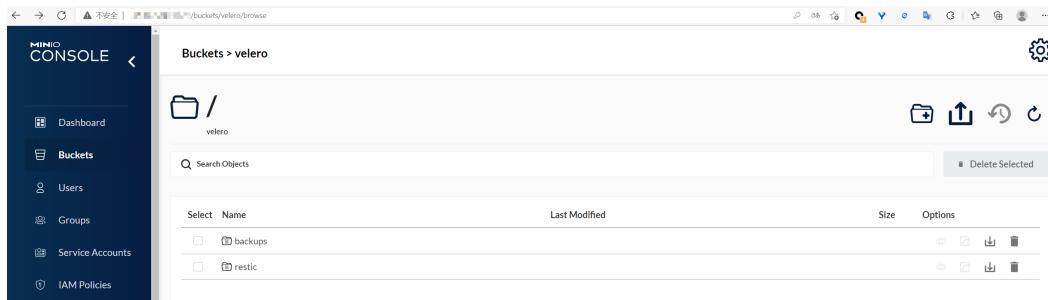
```
velero backup get
```

Information similar to the following is displayed:

NAME	STATUS	ERRORS	WARNINGS	CREATED	EXPIRES	STORAGE
LOCATION	SELECTOR					

```
wordpress-backup Completed 0 0 2021-10-14 15:32:07 +0800 CST 29d default  
<none>
```

In addition, you can go to the object bucket to view the backup files. The backups path is the application resource backup path, and the restic path is the PV data backup path.



----End

## Installing e-backup in the Target Cluster

CCE provides e-backup for cluster backup and restore. You can install this add-on and create a storage location to restore resources.

### Installing the e-backup add-on

- Step 1** Log in to the CCE console. In the navigation pane, choose **Add-ons**. Locate the e-backup add-on and click **Install** under it.
- Step 2** In the **Install Add-on** drawer, select the target cluster, configure parameters, and click **Install**.

The following parameter can be configured:

**volumeWorkerNum**: number of concurrent volume backup jobs. The default value is 3.

----End

### Creating a secret

- Step 1** Obtain an access key.

Log in to the CCE console, move the cursor to the username in the upper right corner, and choose **My Credentials**. In the navigation pane on the left, choose **Access Keys**. On the page displayed, click **Add Access Key**.

- Step 2** Create a key file and format it into a string using Base64.

```
# Create a key file.  
$ vi credential-for-huawei-obs  
HUAWEI_CLOUD_ACCESS_KEY_ID=your_access_key  
HUAWEI_CLOUD_SECRET_ACCESS_KEY=your_secret_key  
  
# Use Base64 to format the string.  
$ base64 -w 0 credential-for-huawei-obs  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXHWOBS
```

- Step 3** Create a secret.

Create a secret using the following YAML content:

```
apiVersion: v1
kind: Secret
metadata:
  labels:
    backup.everest.io/secret: 'true' # Indicates that the secret is used by e-backup to access the backup storage location.
  name: secret-secure-opaque
  namespace: velero # The value must be velero. The secret must be in the same namespace as e-backup.
  type: cfe/secure-opaque
data:
  # String obtained after the credential file is Base64-encoded.
  cloud: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXHWOBS
```

- The secret must be in the same namespace as e-backup, that is, **velero**.
- **secret.data** stores the secret for accessing OBS. The key must be **cloud**, and the value is the string Base64-encoded in [Step 2](#). Generally, the displayed Base64-encoded string contains line breaks. You need to manually delete them when writing the string into **secret.data**.
- The secret must be labeled **secret.everest.io/backup: true**, indicating that the secret is used to manage the backup storage location.

----End

### Creating a storage location

Create a Kubernetes resource object used by e-backup as the backup storage location to obtain and detect information about the backend OBS.

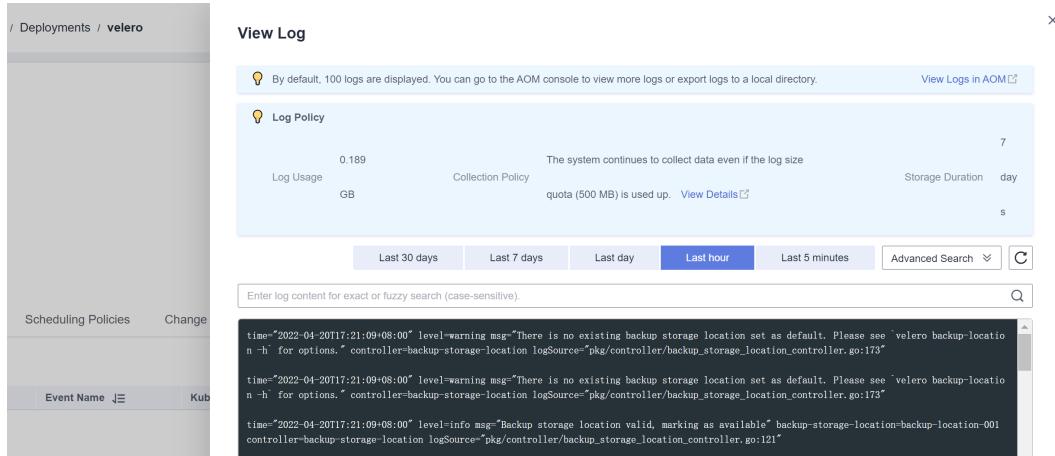
```
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  name: backup-location-001
  namespace: velero # The object must be in the same namespace as e-backup.
spec:
  config:
    endpoint: obs.ap-southeast-1.myhuaweicloud.com # OBS endpoint
  credential:
    name: secret-secure-opaque # Name of the created secret
    key: cloud # Key in secret.data
  objectStorage:
    bucket: velero # OBS bucket name
    provider: huawei # Uses the OBS service.
```

- The **prefix** field is optional, and other fields are mandatory. The value of **provider** is fixed at **huawei**.
- You can obtain the endpoint from [Regions and Endpoints](#). Ensure that all nodes in the cluster can access the endpoint. If the endpoint does not carry a protocol header (http or https), **https** is used by default.
- Correctly set **name** and **key** in the credential. Otherwise, e-backup cannot access the backend storage location.

After the creation is complete, wait for 30 seconds for check and synchronization of the backup storage location. Then check whether **PHASE** is **Available**. The backup location is available only when the value is **Available**.

```
$ kubectl get backupstoragelocations.velero.io backup-location-001 -n velero
NAME      PHASE      LAST VALIDATED AGE   DEFAULT
backup-location-001  Available  23s        23m
```

If the value of **PHASE** does not change to **Available** for a long time, you can view e-backup logs to locate the fault. After e-backup is installed, a workload named **velero** is created in the **velero** namespace, recorded in the logs of **velero**.



## Restoring Applications in the Target Cluster (e-backup)

Perform the following steps to restore a cluster in CCE using e-backup:

Use an immediate backup as the data source and restore data to another cluster. This mode applies to all scenarios.

You can use the Restore manifest below and run the **kubectl create** command to create a backup deletion request.

```

apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-01
  namespace: velero
spec:
  backupName: wordpress-backup
  includedNamespaces:
  - default
  storageClassMapping:
    local: csi-disk
  imageRepositoryMapping:
    quay.io/coreos: swr.ap-southeast-1.myhuaweicloud.com/everest

```

- **backupName:** (mandatory) immediate backup that is used as the data source.
- **storageClassMapping:** changes the storageClassName used by backup resources such as PVs and PVCs. The storageClass types must be the same. In this example, **local** is changed to **csi-disk** supported by CCE.
- **imageRepositoryMapping:** changes the **images** field of the backup. It is used for repository mapping, excluding the change of the image name and tag (to prevent the migration and upgrade from being coupled). For example, after you migrate **quay.io/coreos/etcdb2.5** to SWR, you can use **swr.ap-southeast-1.myhuaweicloud.com/everest/etcdb2.5** in the local image repository. The configuration format is as follows: **quay.io/coreos: swr.ap-southeast-1.myhuaweicloud.com/everest**

If **storageClassMapping** and **imageRepositoryMapping** are configured, you can skip their configuration in [Updating Images](#) and [Updating the Storage Class](#).

For details about other parameters, see [e-backup](#).

After the restore is performed, run the following command to **check the task status**:

```
$ kubectl -n velero get restores restore-01 -o yaml | grep " phase"  
phase: Completed
```

If the status is **Completed**, the restore is complete. You can view the application restore details on the CCE console.

### 3.3.3.5 Preparing Object Storage and Velero

O&M or development personnel migrate Kubernetes objects using the Velero tool.

#### Preparing Object Storage MinIO

MinIO official website: <https://docs.min.io/>

Prepare the object storage and save its AK/SK.

**Step 1** Install the MinIO.

MinIO is a high performance,distributed,Kubernetes Native Object Storage.

```
# Binary installation  
mkdir /opt/minio  
mkdir /opt/miniodata  
cd /opt/minio  
wget https://dl.minio.io/server/minio/release/linux-amd64/minio  
chmod +x minio  
export MINIO_ACCESS_KEY=minio  
export MINIO_SECRET_KEY=minio123  
.minio server /opt/miniodata/ &  
Enter http://{EIP of the node where MinIO is deployed}:9000 in the address box of a browser. Note that  
the corresponding ports on the firewall and security group must be enabled.  
  
# Installing kubectl in containers  
# To release the MinIO service as a service that can be accessed from outside the cluster, change the service  
type in 00-minio-deployment.yaml to NodePort or LoadBalancer.  
kubectl apply -f ./velero-v1.4.0-linux-amd64/examples/minio/00-minio-deployment.yaml
```

**Step 2** Create a bucket, which will be used in the migration.

Open the web page of the MinIO service.

Use **MINIO\_ACCESS\_KEY/MINIO\_SECRET\_KEY** to log in to the MinIO service. In this example, use **minio/minio123**.

Click **Create bucket** above +. In this example, create a bucket named **velero**.

----End

#### Preparing Velero

Velero official website: <https://velero.io/docs/v1.4/contributions/minio/>

Velero is an open source tool to safely back up, restore, perform disaster recovery, and migrate Kubernetes cluster resources and persistent volumes.

Perform the following operations on the ACK and CCE nodes that can run kubectl commands:

**Step 1** Download the migration tool Velero.

Download the latest stable version from <https://github.com/heptio/velero/releases>.  
This document uses **velero-v1.4.0-linux-amd64.tar.gz** as an example.

**Step 2** Install the Velero client.

```
mkdir /opt/ack2cce  
cd /opt/ack2cce
```

```
tar -xvf velero-v1.4.0-linux-amd64.tar.gz -C /opt/ack2cce  
cp /opt/ack2cce/velero-v1.4.0-linux-amd64/velero /usr/local/bin
```

**Step 3** Install the Velero server.

```
cd /opt/ack2cce  
# Prepare the MinIO authentication file. The AK/SK must be correct.  
vi credentials-velero  
  
[default]  
aws_access_key_id = minio  
aws_secret_access_key = minio123  
  
# Install the Velero server. Note that s3Url must be set to the correct MinIO address.  
velero install \  
--provider aws \  
--plugins velero/velero-plugin-for-aws:v1.0.0 \  
--bucket velero \  
--secret-file ./credentials-velero \  
--use-restic \  
--use-volume-snapshots=false \  
--backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://{{EIP of the node where minio runs}}:9000
```

----End

### 3.3.3.6 Backing Up Kubernetes Objects of the ACK Cluster

**Step 1** If you need to back up a WordPress application with PV data, add an annotation to the corresponding pod. If you do not need to back up the PV data, skip this step.

```
# kubectl -n YOUR_POD_NAMESPACE annotate pod/YOUR_POD_NAME backup.velero.io/backup-volumes=YOUR_VOLUME_NAME_1,YOUR_VOLUME_NAME_2,...  
  
[root@iZbp1cqobeh1iyf7qgvvZ ack2cce]# kubectl get pod -n wordpress  
NAME READY STATUSRESTARTS AGE  
wordpress-67796d86b5-f9bfm 1/1 Running 1 39m  
wordpress-mysql-645b796d8d-6k8wh 1/1 Running 0 38m  
  
[root@iZbp1cqobeh1iyf7qgvvZ ack2cce]# kubectl -n wordpress annotate pod/wordpress-67796d86b5-f9bfm backup.velero.io/backup-volumes=wordpress-pvc  
pod/wordpress-67796d86b5-f9bfm annotated  
[root@iZbp1cqobeh1iyf7qgvvZ ack2cce]# kubectl -n wordpress annotate pod/wordpress-mysql-645b796d8d-6k8wh backup.velero.io/backup-volumes=wordpress-mysql-pvc  
pod/wordpress-mysql-645b796d8d-6k8wh annotated
```

**Step 2** Execute the backup task.

```
[root@iZbp1cqobeh1iyf7qgvvZ ack2cce]# velero backup create wordpress-ack-backup --include-namespaces wordpress  
Backup request "wordpress-ack-backup" submitted successfully.  
Run `velero backup describe wordpress-ack-backup` or `velero backup logs wordpress-ack-backup` for more details.
```

**Step 3** Check whether the backup task is successful.

```
[root@iZbp1cqobeh1iyf7qgvvZ ack2cce]# velero backup get  
NAME STATUS CREATED EXPIRES STORAGE LOCATION SELECTOR  
wordpress-ack-backup InProgress 2020-07-07 20:31:19 +0800 CST 29d default<none>  
[root@iZbp1cqobeh1iyf7qgvvZ ack2cce]# velero backup get  
NAME STATUS CREATED EXPIRES STORAGE LOCATION SELECTOR  
wordpress-ack-backup Completed 2020-07-07 20:31:19 +0800 CST 29d default<none>
```

----End

### 3.3.3.7 Restoring Kubernetes Objects in the Created CCE Cluster

#### Creating a StorageClass

In this example, the WordPress application uses Alibaba Cloud SSD persistent data volumes, which need to be replaced with HUAWEI CLOUD SSDs.

The StorageClass used in this example is alicloud-disk-ssd. Create an SC with the same name and use HUAWEI CLOUD SSDs as backend storage media. Set this parameter based on the application to migrate.

```
[root@ccenode-roprr hujun]# cat cce-sc-csidisk-ack.yaml
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-disk-ssd
  selfLink: /apis/storage.k8s.io/v1/storageclasses/csi-disk
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SSD
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate

[root@ccenode-roprr hujun]# kubectl create -f cce-sc-csidisk-ack.yaml
```

#### Restoring the Application

```
[root@ccenode-roprr hujun]# velero restore create --from-backup wordpress-ack-backup
Restore request "wordpress-ack-backup-20200707212519" submitted successfully.
Run `velero restore describe wordpress-ack-backup-20200707212519` or `velero restore logs wordpress-ack-backup-20200707212519` for more details

[root@ccenode-roprr hujun]# velero restore get
NAME      BACKUP STATUS   WARNINGS   ERRORS   CREATED SELECTOR
wordpress-ack-backup-20200708112940   wordpress-ack-backup   Completed   0 02020-07-08 11:29:42
+0800 CST <none>
```

Check the running status of the WordPress application. Make adaptation if issues such as image pulling failures and service access failures occur.

### 3.3.3.8 Update and Adaptation

Application configuration items include image address, Service, and storage disk mounting. Update and make adaptation based on the live environment.

In this example, update the WordPress service.

**Step 1** On the ELB console, buy an elastic load balancer.

**Step 2** Record the ELB ID and the corresponding subnet ID.

**Step 3** Log in to the CCE console, choose **Resource Management > Network** and find the WordPress service. Add the following annotations to its YAML file:

```
annotations:
  kubernetes.io/elb.class: union
  kubernetes.io/elb.id: 9d06a39d-12e1-4ada-835d-c204397498a3
  kubernetes.io/elb.subnet-id: f86ba71c-beb2-46e1-8910-39c8a7d4bb36
  kubernetes.io/session-affinity-mode: SOURCE_IP
```

----End

If the image of your application fails to be pulled, edit the YAML file on the workload page of the CCE console. Set the image address to the repository address obtained from the SWR console.

### 3.3.3.9 Debugging and Starting the Application

Debug and access the application to check data.

**Step 1** Log in to the [CCE console](#). In the navigation pane, choose **Resource Management > Network**. Click the EIP next to the WordPress service.

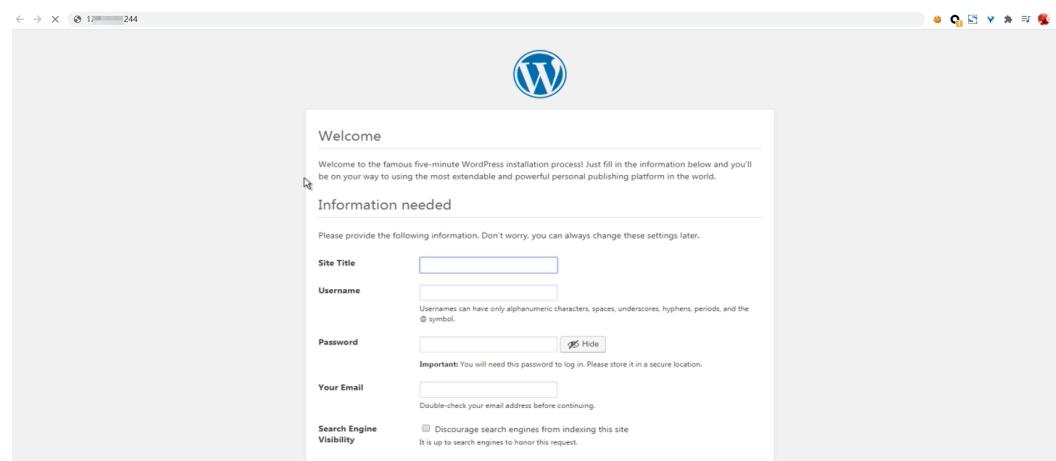
**Figure 3-3** Obtaining the access address

The screenshot shows the CCE Network Services interface. On the left, there's a sidebar with 'Clusters', 'Nodes', 'Node Pools', 'Network', 'Storage', and 'Namespaces'. The main area is titled 'Network' and has tabs for 'Services', 'Ingresses', 'Network Policies', and 'Network Attachment Definitions'. Under the 'Services' tab, there are two entries:

Service Name	Internal Domain Name	Workload	Access Address	Access Type	Access Port -> Container Port / Protocol	Operation
wordpress-mysql	wordpress-mysql.wordpress.svc.cluster.local	wordpress-mysql	None	ClusterIP	3306 -> 3306 / TCP	<a href="#">Update</a> <a href="#">Edit YAML</a> <a href="#">Delete</a>
wordpress	wordpress.wordpress.svc.cluster.local	wordpress	127.0.0.1:244 (IP) 192.168.0.2 (Private)	LoadBalancer (ELB)	80 -> 80 / TCP	<a href="#">Update</a> <a href="#">Edit YAML</a> <a href="#">Delete</a>

**Step 2** If the access is normal, and the migration is successful.

**Figure 3-4** WordPress welcome page



----End

### 3.3.3.10 Others

#### Service Verification

Testing personnel check the functions of the new cluster without interrupting the live traffic.

- Configure a test domain name.
- Test service functions.
- Check O&M functions, such as log monitoring and alarm reporting.

## Switching Live Traffic to the CCE Cluster

O&M switch DNS to direct live traffic to the CCE cluster.

- DNS traffic switching: Adjust the DNS configuration to switch traffic.
- Client traffic switching: Upgrade the client code or update the configuration to switch traffic.

## Bringing the ACK Cluster Offline

After confirming that the service on the CCE cluster is normal, bring the ACK cluster offline and delete the backup files.

- Verify that the service on the CCE cluster is running properly.
- Bring the ACK cluster offline.
- Delete backup files.

### 3.3.4 Change History

**Table 3-9** Change History

Date	Description
2022-05-30	This issue is the first official release.

# 4 DevOps

## 4.1 Installing, Deploying, and Interconnecting Jenkins with SWR and CCE Clusters

### 4.1.1 Solution Overview

#### What Is Jenkins?

Jenkins is an open source continuous integration (CI) tool that provides user-friendly GUIs. It originates from Hudson and is used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins is written in Java and can run in popular servlet containers such as Tomcat, or run independently. It is usually used together with the version control tools (or SCM tools) and build tools. Jenkins supports project building in various languages and is compatible with multiple third-party build tools, such as Maven, Ant, and Gradle. In addition, Jenkins seamlessly integrates with common version control tools, such as SVN and Git, and can directly connect to source code hosting services, such as GitHub.

#### Notes and Constraints

This solution can be deployed only in CCE clusters. It is not supported in DeC.

#### Solution Architecture

Jenkins can be deployed in either of the following modes:

- Use a single Master to install Jenkins. The Master handles tasks and builds and releases services. However, security risks may exist.
- Use the Master to install Jenkins with Agents. The Master schedules build jobs to Agents for execution, and monitors Agents' status. Agents execute build jobs dispatched by the Master and returns the task progress and result.

You can install the Master and Agents on VMs, containers, or combination of the two. For details, see [Table 4-1](#).

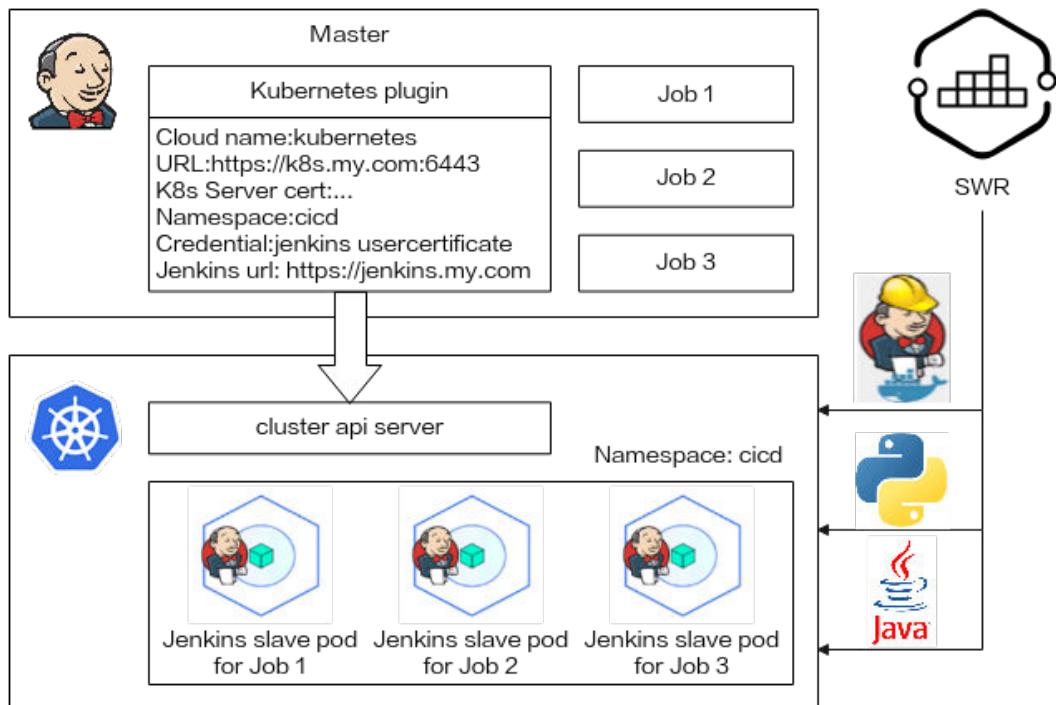
**Table 4-1** Jenkins deployment mode

Deployment Mode	Master	Agent	Advantages and Disadvantages
Single Master	VMs	-	<ul style="list-style-type: none"><li>• Advantage: Localized build is easy to operate.</li><li>• Disadvantage: Task management and execution are performed on the same VM, resulting in high security risks.</li></ul>
Single Master	Containers	-	<ul style="list-style-type: none"><li>• Advantage: The Kubernetes container scheduling supports self-healing.</li><li>• Disadvantage: Task management and execution are not isolated. Security risks exist.</li></ul>
Master +Agent	VMs	VMs	<ul style="list-style-type: none"><li>• Advantage: Task management and execution are isolated, reducing security risks.</li><li>• Disadvantages: Agents are fixed. Resources cannot be scheduled and the resource utilization is low and the maintenance cost is high.</li></ul>
		Containers (Kubernetes cluster)	<ul style="list-style-type: none"><li>• Advantage: Agents can be fixed or dynamic after containerization. Kubernetes schedules the dynamic Agents, improving resource utilization. In addition, tasks can be evenly allocated based on the scheduling policy, which is easy to maintain.</li><li>• Disadvantage: The Master may break down and the recovery cost is high.</li></ul>
Master +Agent	Containers (Kubernetes cluster)	Containers (Kubernetes cluster)	<ul style="list-style-type: none"><li>• Advantage: Agents can be fixed or dynamic after containerization. Kubernetes schedules the dynamic Agents, improving the resource utilization. In addition, The Master is self-healing and the maintenance cost is low. Agents can be deployed in the same cluster as the Master or in different clusters.</li><li>• Disadvantage: The system is complex and the environment is difficult to set up.</li></ul>

In this section, Jenkins is installed with the containerized Master and Agents. Kubernetes schedules the dynamic Agents. For details about the architecture, see [Figure 4-1](#).

- The Master handles jobs. To use resources on the Kubernetes platform, install Kubernetes add-ons on the Master.
- The Kubernetes platform generates pods for Agents to execute jobs. When a job is scheduled on the Master, the Master sends a request to the Kubernetes platform using the Kubernetes add-on. After receiving the request, Kubernetes builds a pod using the pod template to send requests to the Master. After the Master is successfully connected, you can execute the job on the pod.

**Figure 4-1** Installing the Jenkins on Kubernetes



## Operation Process

### Step 1 **Installing and Deploying Jenkins Master.**

Jenkins Master is deployed in the CCE cluster using container images.

### Step 2 **Configuring Jenkins Agent.**

Jenkins can fix Agents in the cluster or use the pipeline to interconnect with CCE to dynamically provide Agent pods. The dynamic Agents need to use Kubernetes add-ons to configure cluster authentication and user permissions.

### Step 3 **Using Jenkins to Build a Pipeline.**

The Jenkins pipeline interconnects with SWR and calls **docker build/login/push** commands in Agents to package and push images automatically.

You can also use pipelines to deploy and upgrade Kubernetes resources (such as Deployments, Services, ingresses, and jobs).

----End

## 4.1.2 Resource and Cost Planning

### NOTICE

The fees listed here are estimates. The actual fees will be displayed on the Huawei Cloud console.

The required resources are as follows:

**Table 4-2** Resource and cost planning

Resource	Description	Quantity	Estimated Fee (USD)
Cloud Container Engine (CCE)	Pay-per-use recommended <ul style="list-style-type: none"><li>• Cluster type: CCE cluster</li><li>• CCE cluster version: v1.21</li><li>• Cluster scale: ≤ 5,000 nodes</li><li>• <b>High Availability:</b> Yes</li></ul>	1	2.91/hour
VM	Pay-per-use recommended <ul style="list-style-type: none"><li>• VM type: General computing-plus</li><li>• Specifications: 4 vCPUs   8 GiB</li><li>• OS: EulerOS 2.9</li><li>• System disk: 50 GiB   General-purpose SSD</li><li>• Data disk: 100 GiB   General-purpose SSD</li></ul>	1	1.00/hour
Elastic Volume Service (EVS)	Pay-per-use recommended <ul style="list-style-type: none"><li>• EVS disk specification: 100 GB</li><li>• EVS disk type: general-purpose SSD</li></ul>	1	0.01/hour
Load Balancer (ELB)	Pay-per-use recommended <ul style="list-style-type: none"><li>• Type: Shared</li><li>• Billed By: Traffic</li><li>• Bandwidth: 5 Mbit/s</li></ul>	1	0.32/hour + 0.80/GB (The traffic fee is charged based on the actual outbound traffic.)

## 4.1.3 Procedure

### 4.1.3.1 Installing and Deploying Jenkins Master

#### NOTE

The UI strings on the Jenkins page may vary depending on the version. For example, the UI strings in Chinese and English are different. The screenshots in this section are for your reference only.

### Selecting an Image

Select a new stable image from Docker Hub. The image used by Jenkins for the test is **jenkinsci/blueocean:2.346.3**. This image is bound with all Blue Ocean add-ons and functions. For details, see [Installing Jenkins](#).

### Preparations

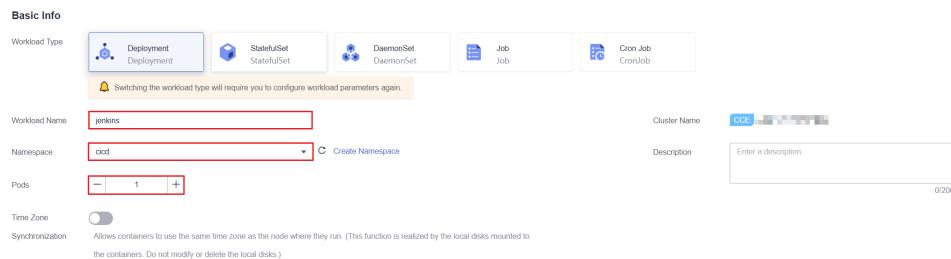
- Before creating a containerized workload, you need to buy a cluster (the cluster must contain at least one node with four vCPUs and 8 GB memory). For details, see [Buying a CCE Cluster](#).
- To enable access to a workload from a public network, ensure that an elastic IP address (EIP) has been bound to or a load balancer has been configured for at least one node in the cluster.

### Installing and Deploying Jenkins on CCE

**Step 1** Log in to the CCE console, click the target cluster. Choose **Workloads > Deployments** and click **Create Workload** on the right.

**Step 2** Set basic workload parameters.

- Workload Name:** Jenkins (user-defined)
- Namespace:** Select the namespace where Jenkins will be deployed. You can create a namespace.
- Pods:** Set it to 1.

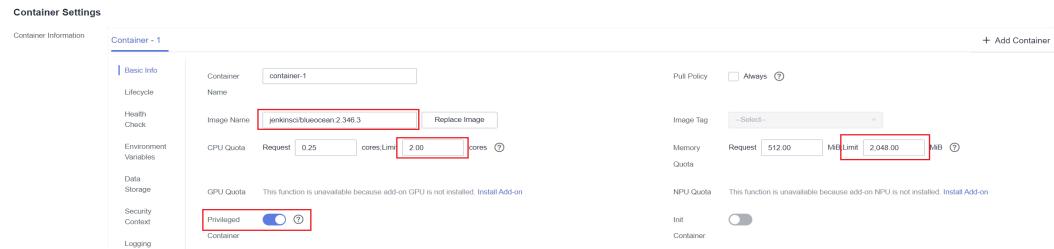


**Step 3** Complete the container settings.

- Image Name:** Enter **jenkinsci/blueocean**. Select an image version as required. In this example, if no version is set, the latest version is used by default.
- CPU Quota:** In this example, set **Limit** to 2 cores.
- Memory Quota:** set **Limit** to 2048 MiB.
- Privileged Container:** If Jenkins is deployed with a single Master, enable the **Privileged Container** so that the container can obtain permissions to operate the host. Otherwise, Docker commands cannot be executed in the Jenkins Master container.

Retain the default values for other parameters, as shown in the following figure.

**Figure 4-2** Basic container information parameters



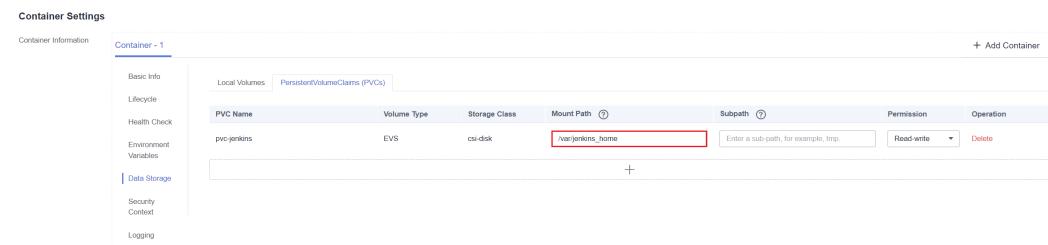
**Step 4** Choose **Data Storage > PersistentVolumeClaims (PVCs)** and add a persistent volume.

In the displayed dialog box, select a cloud volume and enter **/var/jenkins\_home** in the mount path to mount the cloud volume for Jenkins to retain data.

**NOTE**

The cloud storage type can be **EVS** or **SFS**. If no cloud storage is available, click **Create PVC**. If you select **EVS**, the AZ of the EVS disk must be the same as that of the node.

**Figure 4-3** Adding a cloud volume



**Step 5** Add permissions to the Jenkins container so that Docker commands can be executed in the Jenkins container.

1. Ensure that **Privileged Container** is enabled in 3.
2. Choose **Data Storage > Local Volumes**, add a local volume, and mount the host path to the corresponding container path.

**Table 4-3** Mounting path

Storage Type	Host Path	Mount Path
Host Path	/var/run/docker.sock	/var/run/docker.sock
Host Path	/usr/bin/docker	/usr/bin/docker
Host Path	/usr/lib64/libltdl.so.7	/usr/lib/x86_64-linux-gnu/libltdl.so.7
Host Path	/usr/bin/kubectl	/usr/local/bin/kubectl

After the mounting is complete, the page shown in **Figure 4-4** is displayed.

**Figure 4-4** Mounting the host paths to the corresponding container paths

Type	Disk Source	Mount Path	Subpath	Capacity (GiB)	Permission	Operation
HostPath	/var/run/docker.sock	/var/run/docker.sock	--	--	Read-write	Delete
HostPath	/usr/bin/docker	/usr/bin/docker	--	--	Read-write	Delete
HostPath	/usr/lib64/libstdc++.so.7	/usr/lib64/_linux-gnu/libstdc++.so.7	--	--	Read-write	Delete
HostPath	/usr/bin/kubectl	/usr/local/bin/kubectl	--	--	Read-write	Delete

3. In **Security Context**, set User ID to 0 (user root).

**Figure 4-5** Configuring the user

You can specify the user who runs the processes in a container.

User ID: 0

Containers run with the user permissions you specify. For example, enter user ID 0 to run containers as user root.

### Step 6 In Service Configuration, set the access mode.

The Jenkins container image has two ports: 8080 and 50000. You need to configure them separately. Port 8080 is used for web login, and port 50000 is used for the connection between the Master and Agents.

In this example, two Services are created:

- LoadBalancer: provides external web access using port 8080. You can also select **NodePort** to provide external access.  
The Service name is **jenkins** (user-defined), the container port is **8080**, the access port is **8080**, and retain the default values for other parameters.
- ClusterIP: used by the Agent to connect to the Master. Jenkins requires the IP address of Jenkins-web be the same as that of Jenkins-agent. Therefore, port 8080 for web access and port 50000 for agent access are included.  
The Service name is **agent** (user-defined), the container port 1 is **8080**, the access port 1 is **8080**, the container port 2 is **50000**, the access port 2 is **50000**. Retain the default values for other parameters.

#### NOTE

In this example, Agents created in the subsequent steps are in the same cluster as the Master. Therefore, the Agents use the ClusterIP Service to connect to the Master.

If Agents need to connect to the Master across clusters or through the public network, select a proper Service. Note that Jenkins requires the IP address of Jenkins-web to be the same as that of Jenkins-agent. Therefore, **ports 8080 and 50000 must be enabled for the IP address connected to Jenkins-agent**, for addresses used only for web access, only port 8080 can be enabled.

**Figure 4-6 Adding a Service**

Service Settings			
Service	Access Mode	Access Port/Container Port/Protocol	Operation
jenkins	LoadBalancer	8080 → 8080 / TCP	<a href="#">Delete</a>
agent	ClusterIP	8080 → 8080 / TCP 50000 → 50000 / TCP	<a href="#">Delete</a>
+			

**Step 7** Retain the default settings for **Advanced Settings** and click **Create Workload**.

**Step 8** Click **Back to Deployment List** to view the Deployment status. If the workload is in **Running** state, the Jenkins application can be accessed.

**Figure 4-7 Viewing the workload status**

Deployments							Quick Links	<a href="#">Create Workload</a>	<a href="#">Create from YAML</a>
Workload Name	Status	Pods (Normal/AI)	Namespace	Created	Image Name	Operation	Filter by label	Enter a name	Q C
jenkins	Running	1 / 1	cicd	24 minutes ago	blueocean:latest	<a href="#">Monitor</a>   <a href="#">View Log</a>   <a href="#">Upgrade</a>   More ▾			

----End

## Logging In and Initializing Jenkins

**Step 1** On the CCE console, click the target cluster. Choose **Networking** in the navigation pane. On the **Services** tab page, view the Jenkins access mode.

**Figure 4-8 Access mode corresponding to port 8080**

Services								Ingresses		
Service	Selector	Namespace	Service Type	IP Address	Port/Protocol	Created	Operation	Filter by selector	Enter a name	C
agent	app:jenkins version:v1	cicd	ClusterIP	10.247.173.132 (Cluster IP)	8080 / TCP 50000 / TCP	26 minutes ago	<a href="#">Manage Pod</a>   <a href="#">View Events</a>   More ▾			
jenkins	app:jenkins version:v1	cicd	LoadBalancer	10.247.57.210 (Cluster IP) 10.247.57.210 (Load Balancer IP)	8080 / TCP	26 minutes ago	<a href="#">Manage Pod</a>   <a href="#">View Events</a>   More ▾			

**Step 2** Enter **EIP:8080** of the load balancer in the address box of the browser to open the Jenkins configuration page.

When you visit the page for the first time, you are prompted to obtain the initial administrator password. You can obtain the password from the Jenkins pod. Before running the following commands, you need to connect to the cluster using `kubectl`. For details, see [Connecting to a Cluster Using kubectl](#).

```
# kubectl get pod -n cicd
NAME           READY   STATUS    RESTARTS   AGE
jenkins-7c69b6947c-5gvlm   1/1     Running   0          17m
# kubectl exec -it jenkins-7c69b6947c-5gvlm -n cicd -- /bin/sh
# cat /var/jenkins_home/secrets/initialAdminPassword
b10eabe29a9f427c9b54c01a9c3383ae
```

**Step 3** When you log in to the system for the first time, select the default recommended add-on and create an administrator as prompted. After the initial configuration is complete, the Jenkins page is displayed.

The screenshot shows the Jenkins dashboard. On the left sidebar, there are links for 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', 'Open Blue Ocean', and 'New View'. Below these are 'Build Queue' and 'Build Executor Status'. The 'Build Queue' section says 'No builds in the queue.' The 'Build Executor Status' section shows 1 idle and 2 idle executors. The main area has a 'Welcome to Jenkins!' message and a 'Start building your software project' section with links for 'Create a job', 'Set up a distributed build', 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'. At the top right, there are user icons for 'admin' and 'log out'.

----End

## Modifying the Number of Concurrent Build Jobs

**Step 1** On the Jenkins dashboard page, click **Manage Jenkins** on the left, choose **System Configuration > Manage nodes and clouds**, and select **Configure** from the drop-down list of the target node.

The screenshot shows the 'Manage nodes and clouds' page. On the left sidebar, there are links for 'Back to Dashboard', 'Manage Jenkins', 'New Node', 'Configure Clouds', and 'Node Monitoring'. Below these are 'Build Queue' and 'Build Executor Status'. The main area lists a single node named 'Built-In Node'. A context menu is open over this node, with the 'Configure' option highlighted with a red box. The table columns include Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The Built-In Node row shows Linux (amd64), In sync, 9.32 GB, 0 B, 80.07 GB, 0 ms, and 23 min.

### NOTE

- The number of concurrent build jobs can be modified on both the Master and Agent. The following uses Master as an example.
- If the **Master is used with Agents**, you are advised to set the number of concurrent build jobs of Master to **0**. That is, all build jobs are performed using Agent. If a **single Master** is used, you do not need to change the value to **0**.

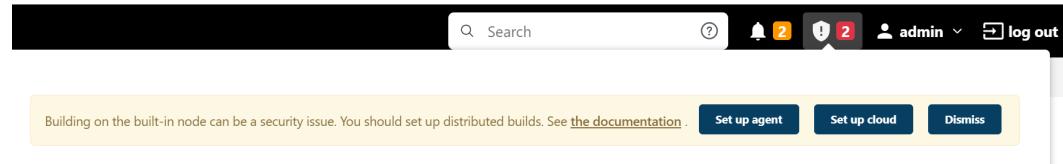
**Step 2** Modify the maximum number of concurrent build jobs. In the example, the value is changed to **2**. You can change the value as required.

The screenshot shows a configuration dialog for 'Number of executors'. A text input field contains the value '2'. Below the input field is a note: 'The maximum number of concurrent builds that Jenkins may perform on this node. A good value to start with would be the number of CPU cores on the machine. Setting a higher value would cause each build to take longer, but could increase the overall throughput. For example, one build might be CPU-bound, while a second build running at the same time might be I/O-bound — so the second build could take advantage of the spare I/O capacity at that moment.' At the bottom, there is a note: 'Agents (nodes that are not the built-in node) must have at least one executor. To temporarily prevent any builds from being executed on an agent, use the *Mark this node temporarily offline* button on the agent's page.' Another note at the bottom states: 'For the built-in node, set the number of executors to zero to prevent it from executing builds locally on the controller. Note: The built-in node will always be able to run flyweight tasks including Pipeline's top-level task.'

----End

### 4.1.3.2 Configuring Jenkins Agent

After Jenkins is installed, the following information may be displayed, indicating that Jenkins uses the Master for local build and Agents are not configured.



The screenshot shows the Jenkins dashboard with a yellow banner at the top stating: "Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#)." Below the banner are three buttons: "Set up agent", "Set up cloud", and "Dismiss".

If you install Jenkins using a Master, you can build a pipeline after performing operations in [Installing and Deploying Jenkins Master](#). For details, see [Using Jenkins to Build a Pipeline](#).

If you install Jenkins using Master with Agents, continue to configure Agents. You can select either of the following methods:

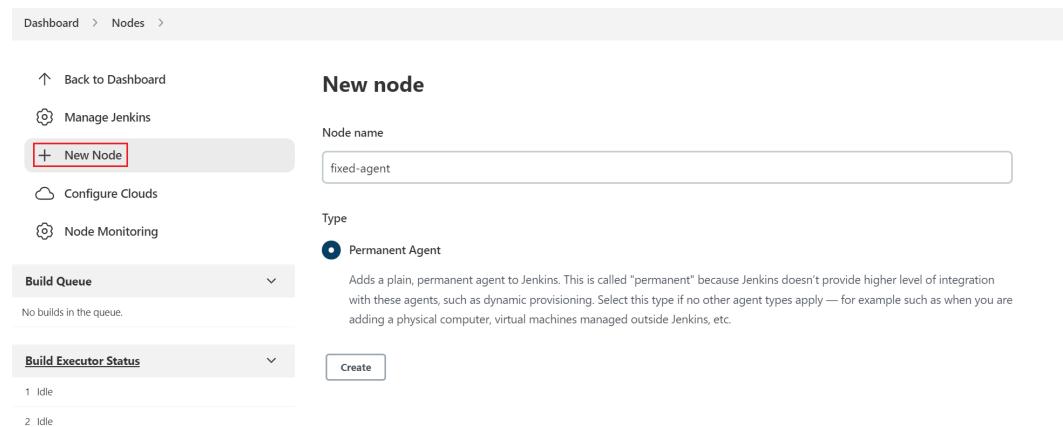
- **Fixed Agent:** The Agent container keeps running and is not killed after the job is built. After the job is built, cluster resources are occupied. The configuration process is simple.
- **Dynamic Agent:** the Agent container is created when the job is being built and destroyed after the task is built. In this way, resources can be dynamically allocated and the resource utilization is high. However, the configuration process is complex.

In this section, the Agent is containerized using the [jenkins/inbound-agent: 4.13.3-1](#) image.

### Adding a Fixed Agent to Jenkins

**Step 1** Log in to the Jenkins Dashboard, click **Manage Jenkins** on the left, and choose **System Configuration > Manage nodes and clouds**.

**Step 2** Click **New Node** on the left, enter the node name **fixed-agent** (which can be customized), and select **Permanent Agent** for **Type**.



The screenshot shows the "New node" configuration page. On the left, there's a sidebar with links: Back to Dashboard, Manage Jenkins, New Node (highlighted with a red border), Configure Clouds, and Node Monitoring. The main area has a title "New node". It includes a "Node name" input field containing "fixed-agent" and a "Type" section where "Permanent Agent" is selected. A note below explains: "Adds a plain, permanent agent to Jenkins. This is called 'permanent' because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc." At the bottom right is a "Create" button.

**Step 3** Configure the following node information:

- **Number of executors:** The default value is 1. Set this parameter as required.
- **Remote root directory:** enter **/home/jenkins/agent**.
- **Launch method:** Select **Launch agent by connecting it to the controller**.

Retain the values for other parameters and click **Save**.

Number of executors ?  
1

Remote root directory ?  
/home/jenkins/agent

Labels ?

Usage ?  
Use this node as much as possible

Launch method ?  
Launch agent by connecting it to the controller

**Step 4** In the **Nodes** page, click the new node. The Agent status is disconnected, and the method for connecting the node to Jenkins is provided. This command applies to VM installation. In this example, container-based installation is used. Therefore, you only need to copy the secret, as shown in the following figure.



**Step 5** Log in to the CCE console, click the target cluster. Choose **Workloads > Deployments** and click **Create Workload** on the right.

**Step 6** Set the basic information about the workload.

- Workload Name:** agent (user-defined)
- Namespace:** Select the namespace where Jenkins will be deployed. You can create a namespace.
- Pods:** Set it to 1.

Basic Info

Workload Type

Deployment StatefulSet DaemonSet Job CronJob

⚠️ Switching the workload type will require you to configure workload parameters again.

Workload Name: agent

Namespace: ociid

Cluster Name: cce-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Pods: 1

Description: Enter a description.

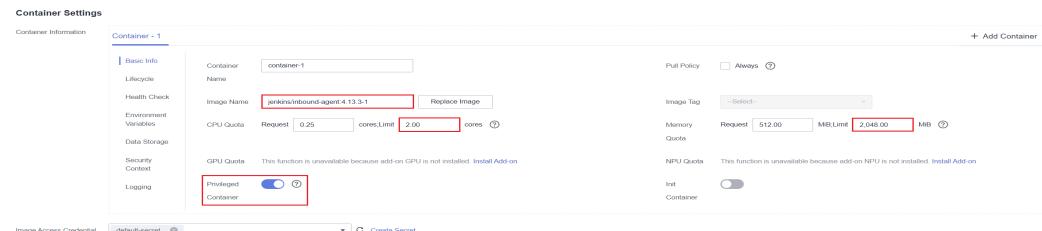
Time Zone: ⓘ Allows containers to use the same time zone as the node where they run. (This function is realized by the local disks mounted to the containers. Do not modify or delete the local disks.)

**Step 7** Set basic container parameters.

- Image Name:** Enter **jenkins/inbound-agent:4.13.3-1**. The image version may change with time. Select an image version as required or use the latest version.

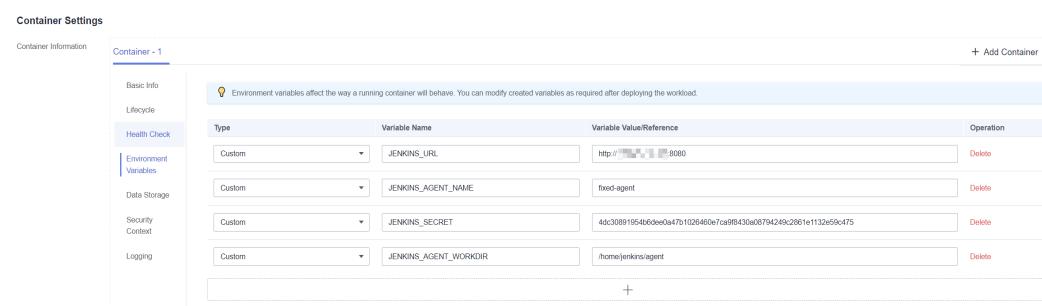
- **CPU Quota:** In this example, set **Limit** to **2** cores.
- **Memory Quota:** In this example, set **Limit** to **2048** MiB.
- **Privileged Container:** must be enabled so that the container can obtain permissions to operate the host. Otherwise, Docker commands cannot be executed in the container.

Retain the values for other parameters.



#### Step 8 Run the following commands to set the environment variables:

- **JENKINS\_URL:** indicates the access path of Jenkins. You need to enter the IP address of port 8080 set in **Step 6** (ports 8080 and 50000 must be enabled for the IP address), for example, **http://10.247.222.254:8080**.
- **JENKINS\_AGENT\_NAME:** name of the Agent set in **Step 2**. In this example, the value is **fixed-agent**.
- **JENKINS\_SECRET:** secret copied from **Step 4**.
- **JENKINS\_AGENT\_WORKDIR:** remote work directory configured in **Step 3**, that is, **/home/jenkins/agent**.



#### Step 9 Add permissions to the Jenkins container so that Docker commands can be executed in the Jenkins container.

1. Ensure that **Privileged Container** is enabled in **3**.
2. Choose **Data Storage > Local Volumes**, add a local volume, and mount the host path to the corresponding container path.

**Table 4-4 Mounting path**

Storage Type	Host Path	Mount Path
Host Path	/var/run/docker.sock	/var/run/docker.sock
Host Path	/usr/bin/docker	/usr/bin/docker
Host Path	/usr/lib64/libltdl.so.7	/usr/lib/x86_64-linux-gnu/libltdl.so.7

Storage Type	Host Path	Mount Path
Host Path	/usr/bin/kubectl	/usr/local/bin/kubectl

After the mounting is complete, the page shown in [Figure 4-9](#) is displayed.

**Figure 4-9** Mounting the host paths to the corresponding container paths

The screenshot shows the 'Container Settings' interface for 'Container - 1'. In the 'Data Storage' section, under 'Local Volumes', there are four entries, each with a red box highlighting the 'Mount Path' column:

Type	Disk Source	Mount Path	Subpath	Capacity (GB)	Permission	Operation
HostPath	/var/run/docker.sock	/var/run/docker.sock	—	—	Read+write	Delete
HostPath	/usr/bin/docker	/usr/bin/docker	—	—	Read+write	Delete
HostPath	/usr/lib64/libstdc++.so.7	/usr/lib64_84-linux-gnu/libstdc++.so.7	—	—	Read+write	Delete
HostPath	/usr/bin/kubectl	/usr/local/bin/kubectl	—	—	Read+write	Delete

3. In **Security Context**, set **User ID** to **0** (user root).

**Figure 4-10** Configuring the user

The screenshot shows the 'Container Settings' interface for 'Container - 1'. In the 'Security Context' section, the 'User ID' field is highlighted with a red box and contains the value '0'.

**Step 10** Retain the settings for **Advanced Settings** and click **Create Workload**.

**Step 11** Go to the Jenkins page and refresh the node status to **In sync**.

The screenshot shows the 'Manage nodes and clouds' page in Jenkins. It lists two nodes: 'Built-In Node' and 'fixed-agent'. Both nodes are marked as 'In sync' with a green status indicator. The 'fixed-agent' node has a red box around it.

Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
Built-In Node	Linux (amd64)	In sync	9.39 GB	0 B	80.84 GB	0ms
fixed-agent	Linux (amd64)	In sync	80.84 GB	0 B	80.84 GB	79ms

#### BOOK NOTE

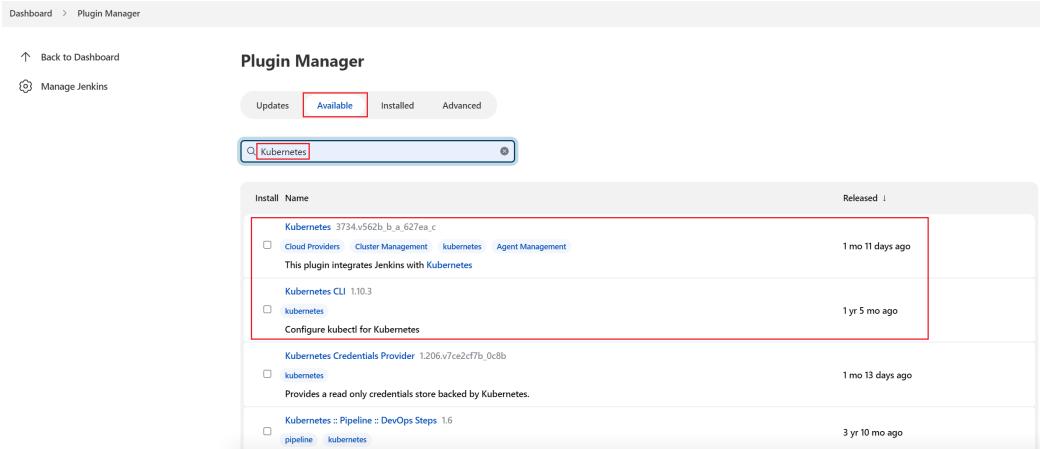
After the Agent is configured, you are advised to set the number of concurrent build jobs of the Master to **0**. That is, you use the Agent to perform build. For details, see [Modifying the Number of Concurrent Build Jobs](#).

----End

## Setting a Dynamic Agent for Jenkins

**Step 1** Install the plug-in.

On the Jenkins dashboard page, click **Manage Jenkins** on the left and choose **System Configuration > Manage Plugins**. On the **Available** tab page, filter and install the **Kubernetes CLI Plugin** and **Kubernetes plugin**.



The plug-in version installed in this document may change with time. Select a plug-in version as required.

- **Kubernetes Plugin:** 3734.v562b\_b\_a\_627ea\_c  
It is used to run dynamic Agents in the Kubernetes cluster, create a Kubernetes pod for each started Agent, and stop the pod after each build is complete.
- **Kubernetes CLI Plugin:** 1.10.3  
kubectl can be configured for jobs to interact with Kubernetes clusters.

#### NOTE

The Jenkins plug-ins are provided by the plug-in maintenance personnel and may be iterated due to security risks.

#### Step 2 Add cluster access credentials to Jenkins.

Add the cluster access credentials to Jenkins in advance. For details, see [Setting Cluster Access Credentials](#).

#### Step 3 Configure basic cluster information.

On the Jenkins dashboard page, click **Manage Jenkins** on the left and choose **System Configuration > Manage nodes and clouds**. Click **Configure Clouds** on the left to configure the cluster. Click **Add a new cloud** and select **Kubernetes**. The cluster name can be customized.

#### Step 4 Enter Kubernetes Cloud details.

Set the following cluster parameters and retain the values for other parameters, as shown in [Figure 4-11](#).

- **Kubernetes URL:** The cluster API server address. You can enter `https://kubernetes.default.svc.cluster.local:443`.
- **Credentials:** Select the cluster credential added in [Step 2](#). You can click **Test Connection** to check whether the cluster is properly connected.
- **Jenkins URL:** Enter the Jenkins access path. You need to enter the IP address of port 8080 set in [Step 6 \(ports 8080 and 50000 must be enabled for the](#)

IP address, that is, the intra-cluster access address), for example, <http://10.247.222.254:8080>.

**Figure 4-11 Example**

The screenshot shows a Jenkins configuration page for connecting to a Kubernetes cluster. The fields include:

- Kubernetes URL: https://kubernetes.default.svc.cluster.local:443
- Use Jenkins Proxy: Unchecked
- Kubernetes server certificate key: (empty text area)
- Disable https certificate check: Unchecked
- Kubernetes Namespace: (empty text area)
- JNLP Docker Registry: (empty text area)
- Credentials: k8s-token (selected from dropdown)
- + Add: (button)
- Test Connection: (button) - This button is highlighted with a red border.
- WebSocket: Unchecked
- Direct Connection: Unchecked
- Jenkins URL: http://10.247.222.254:8080

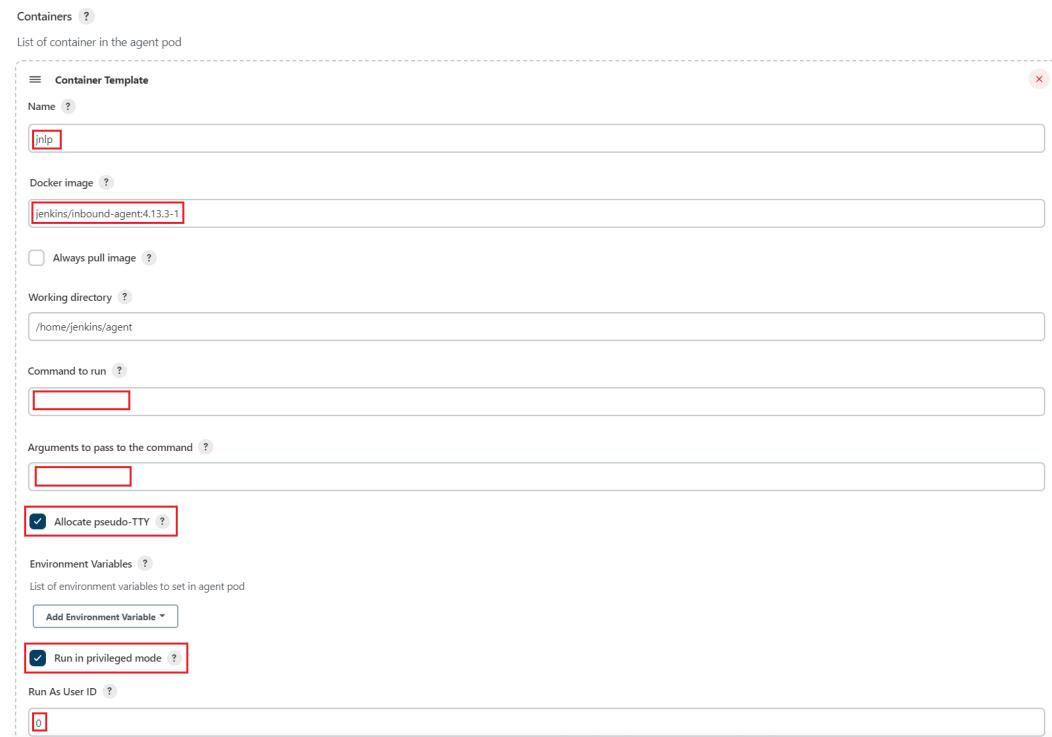
**Step 5 Pod Template:** Click **Add Pod Template > Pod Template details** and set pod template parameters.

- Set the basic parameters of the pod template, as shown in [Figure 4-12](#).
  - Name:** jenkins-agent
  - Namespace:** cicd
  - Labels:** jenkins-agent
  - Usage:** Select **Use this node as much as possible**.

Figure 4-12 Basic parameters of the pod template

The screenshot shows the 'Pod Template' configuration interface. It includes fields for Name (jenkins-agent), Namespace (cicd), Labels (jenkins-agent), Usage (Use this node as much as possible), Pod template to inherit from (empty), and Containers (empty). A 'Add Container' button is visible.

- Add a container. Click **Add Container > Container Template**. [Figure 4-13](#) shows the parameters.
  - **Name:** The value must be **jnlp**.
  - **Docker image:** **jenkins/inbound-agent:4.13.3-1**. The image version may change with time. Select an image version as required or use the latest version.
  - **Working directory:** **/home/jenkins/agent** is selected by default.
  - **Command to run/Arguments to pass to the command:** You need to delete the existing default value and leave these two parameters empty.
  - **Allocate pseudo-TTY:** Select this parameter.
  - Select **Run in privileged mode** and set **Run As User ID to 0 (root user)**.

**Figure 4-13** Container template parameters

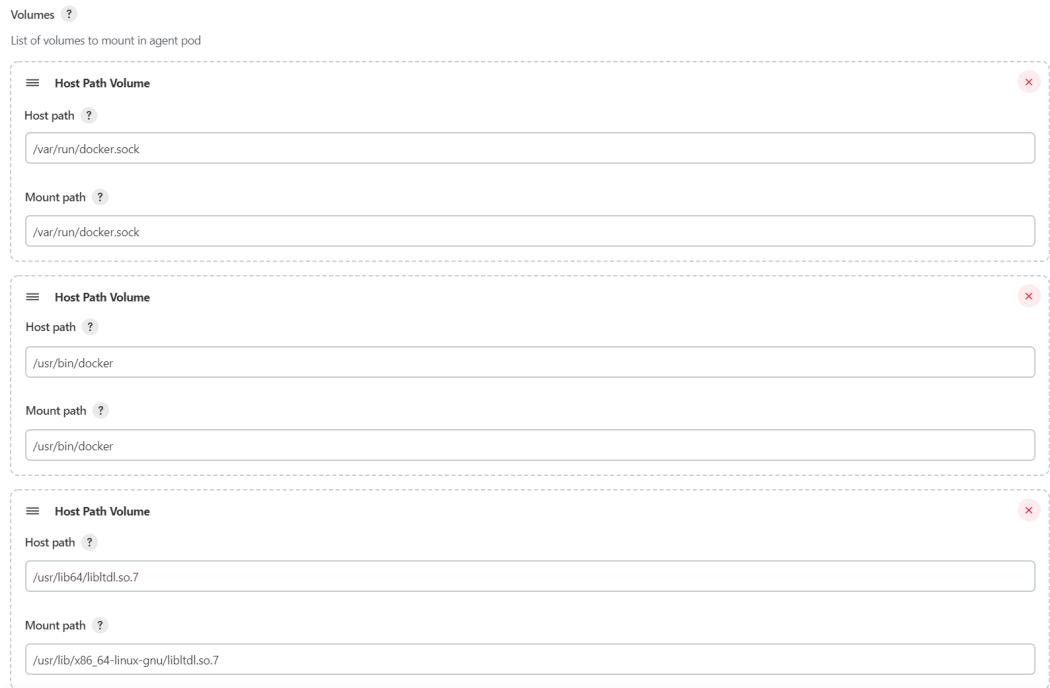
- Add a volume: Choose **Add Volume > Host Path Volume** to mount the host path in **Table 4-5** to the corresponding path of the container.

**Table 4-5** Mount path

Storage Type	Host Path	Mount Path
HostPath	/var/run/docker.sock	/var/run/docker.sock
HostPath	/usr/bin/docker	/usr/bin/docker
HostPath	/usr/lib64/libltdl.so.7	/usr/lib/x86_64-linux-gnu/libltdl.so.7
HostPath	/usr/bin/kubectl	/usr/local/bin/kubectl

After the mounting is complete, the page shown in **Figure 4-14** is displayed.

**Figure 4-14** Mounting the host paths to the corresponding container paths



- **Service Account:** Enter **jenkins-admin**, which is the name of the service account created in [Step 2](#).
- **Run As User ID:** **0 (root user)**.
- **Workspace Volume:** working directory of the agent. Persistence is recommended. Select **Host Path Workspace Volume** and set **Host path** to **/home/jenkins/agent**.



## Step 6 Click Save.

### NOTE

After the Agent is configured, you are advised to set the number of concurrent build jobs of the Master to **0**. That is, you do not use the Master to perform local build but use the Agent for build. For details, see [Modifying the Number of Concurrent Build Jobs](#).

----End

## Setting Cluster Access Credentials

Jenkins supports multiple types of cluster access credentials. You can obtain a token or use a certificate to access Jenkins. The following describes how to obtain a token or use a certificate to access Jenkins.

### Solution 1: Token Access

#### Step 1 Use kubectl to connect to the cluster.

**Step 2** Create a **sa.yaml** file.

```
vi sa.yaml
```

Press **i** to enter the editing mode. The file content is as follows:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins-admin      # ServiceAccount username.
  namespace: cicd    # Namespace where Jenkins is deployed.
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jenkins-admin      # ServiceAccount username.
subjects:
  - kind: ServiceAccount
    name: jenkins-admin      # ServiceAccount username.
    namespace: cicd    # Namespace where Jenkins is deployed.
roleRef:          # Indicates that the cluster-admin permission is granted in the cicd space. No
modification is required.
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

**NOTE**

- In clusters earlier than v1.21, a token is obtained by mounting the secret of the service account to a pod. Tokens obtained this way are permanent. This approach is no longer recommended starting from version 1.21. Service accounts will stop auto creating secrets in clusters from version 1.25.

In clusters of version 1.21 or later, you can use the [TokenRequest API](#) to [obtain the token](#) and use the projected volume to mount the token to the pod. Such tokens are valid for a fixed period. When the mounting pod is deleted, the token automatically becomes invalid. For details, see [Service Account Token Security Improvement](#).

- If you need a token that never expires, you can also [manually manage secrets for service accounts](#). Although a permanent service account token can be manually created, you are advised to use a short-lived token by calling the [TokenRequest API](#) for higher security.

**Step 3** Create ServiceAccount in the cluster. After the ServiceAccount is created, it cannot be deleted. Otherwise, the token becomes invalid.

```
kubectl apply -f sa.yaml
```

**Step 4** Obtain a token. If you have customized the namespace and ServiceAccount name, modify the parameters in the command.

```
kubectl get secret -n cicd $(kubectl get sa jenkins-admin -n cicd -o jsonpath={.secrets[0].name}) -o
jsonpath={.data.token} |base64 -d ;echo
```

**Step 5** On the Jenkins dashboard page, click **Manage Jenkins** on the left, choose **Security > Manage Credentials**, and click the default global credential storage domain of Jenkins. You can also create a domain.

T	P	Store	I	Domain	ID	Name

P	Store	I	Domains
	Jenkins		(global)

**Step 6** Click **Add Credentials** to create a credential.

- **Kind:** Select **Secret text**.
- **Scope:** Select **Global**.
- **Secret:** Enter the token value obtained in **Step 4**.
- **ID:** Set this parameter to **k8s-token** (which can be changed as required).

The screenshot shows the 'New credentials' page in the CCE console. The 'Kind' field is set to 'Secret text'. The 'Scope' field is set to 'Global'. The 'Secret' field contains a long, redacted token. The 'ID' field is set to 'k8s-token'. There is a 'Description' field which is empty. At the bottom is a 'Create' button.

----End

**Solution 2: Certificate Access**

The certificate file that can be identified in Jenkins is in the PKCS#12 certificate. Therefore, you need to convert the cluster certificate to a PFX certificate file in PKCS#12 format.

**Step 1** Download the cluster certificate from the CCE console. The certificate contains the **ca.crt**, **client.crt**, and **client.key** files.

The screenshot shows the 'Cluster Information' tab in the CCE console. On the left sidebar, under 'O&M', the 'Add-ons' item has a red notification badge with the number '1'. In the main content area, under 'Connection Information', there is a row for 'Certificate Authentication' with options for 'X.509 certificate' and a 'Download' button, which is highlighted with a red box.

**Step 2** Log in to a Linux host, place the three certificate files in the same directory, and use OpenSSL to convert the certificate format to generate a certificate in the **cert.pfx** format. When a certificate is generated, the system prompts you to enter the password. The password can be customized.

```
openssl pkcs12 -export -out cert.pfx -inkey client.key -in client.crt -certfile ca.crt
```

**Step 3** On the Jenkins console, choose **Manage Jenkins > Manage Credentials** and click **Global**. You can also create a domain.

The screenshot shows the Jenkins 'Credentials' management interface. In the top navigation bar, there is a link to 'Dashboard'. Below the navigation, there is a sidebar with links like '+ New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', 'Open Blue Ocean', and 'New View'. The main area is titled 'Credentials' and shows a table with columns 'Domain', 'ID', and 'Name'. A single row is visible, labeled 'Jenkins' under 'Domain' and '(global)' under 'Name'. Below the table, there is a 'Build Queue' section indicating 'No builds in the queue.' At the bottom, there are icons for 'S' (Stable), 'M' (Medium), and 'L' (Long). The entire interface has a light gray background with blue and black text.

**Step 4** Click **Add Credential**.

- **Kind:** Select **Certificate**.
- **Scope:** Select **Global**.
- **Certificate:** Select **Upload PKCS#12 certificate** and upload the **cert.pfx** file generated in **Step 2**.
- **Password:** The password customized during cert.pfx conversion.
- **ID:** Set this parameter to **k8s-test-cert** (which can be changed as required).

#### New credentials

The screenshot shows the 'New credentials' form for adding a 'Certificate' kind. The 'Kind' field is set to 'Certificate'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. Under the 'Certificate' section, the 'Upload PKCS#12 certificate' option is selected, and a file input field shows 'cert.pfx' being chosen. The 'Password' field contains '\*\*\*\*\*'. The 'ID' field is set to 'k8s-test-cert'. The entire form is contained within a light gray box with a white background and black text.

----End

### 4.1.3.3 Using Jenkins to Build a Pipeline

#### Obtaining a Long-Term Valid Login Command

During Jenkins installation and deployment, the Docker commands have been configured in the container (see 5). Therefore, no additional configuration is

required for interconnecting Jenkins with SWR. You can directly run the Docker command. You only need to obtain a long-term valid SWR login command. For details, see [Obtaining a Long-Term Valid Login Command](#).

For example, the command of this account is as follows:

```
docker login -u ap-southeast-1@xxxxx -p xxxx swr.ap-southeast-1.myhuaweicloud.com
```

## Creating a Pipeline to Build and Push Images

In this example, Jenkins is used to build a pipeline to pull code from the code repository, package the code into an image, and push the image to SWR.

The pipeline creation procedure is as follows:

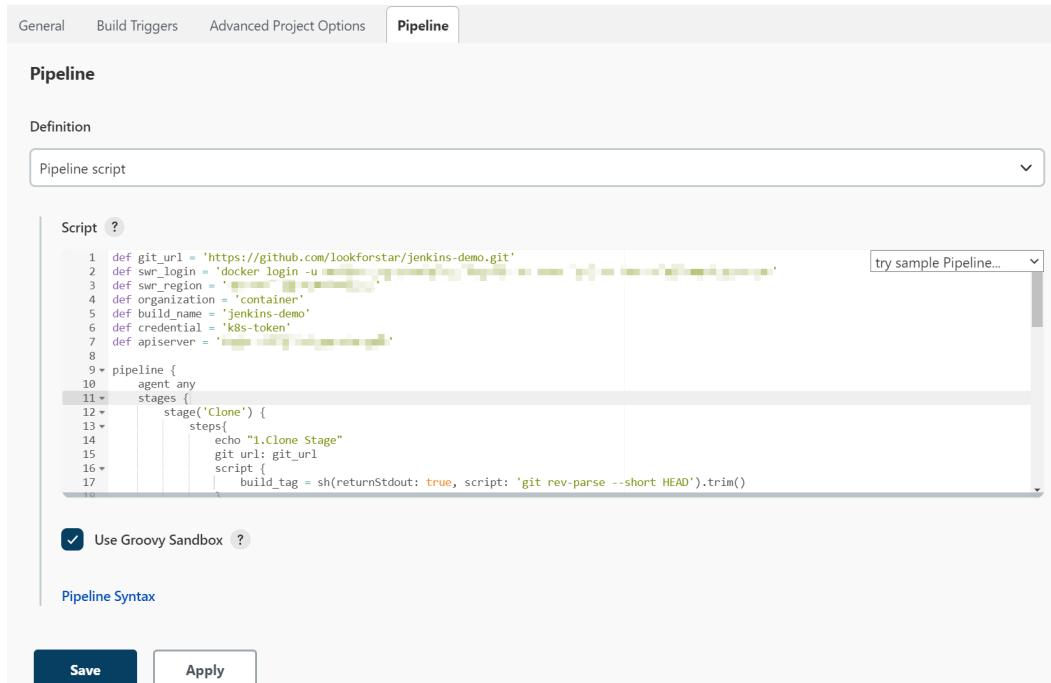
**Step 1** Click **New Item** on the Jenkins page.

**Step 2** Enter a task name and select **Create Pipeline**.

The screenshot shows the Jenkins 'Create New Item' dialog. In the 'Enter an item name' field, 'test-pipe' is entered. Below it, there are five project types listed: 'Freestyle project', 'Pipeline' (which is highlighted with a red border), 'Multi-configuration project', 'Folder', and 'Multibranch Pipeline'. Each item has a brief description and an icon.

Project Type	Description
Freestyle project	This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
Pipeline	Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
Multi-configuration project	Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
Folder	Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
Multibranch Pipeline	Creates a set of Pipeline projects according to detected branches in one SCM repository.

**Step 3** Configure only the pipeline script.



The following pipeline scripts are for reference only. You can customize the script content based on your service requirements. For details about the syntax, see [Pipeline](#).

Some parameters in the example need to be modified:

- **git\_url**: Address of your code repository. Replace it with the actual address.
- **swr\_login**: The login command obtained in [Obtaining a Long-Term Valid Login Command](#).
- **swr\_region**: SWR region.
- **organization**: The actual organization name in SWR.
- **build\_name**: indicates the name of the created image.
- **credential**: The cluster credential added to Jenkins. Enter the credential ID. If you want to deploy the service in another cluster, add the access credential of the cluster to Jenkins again. For details, see [Setting Cluster Access Credentials](#).
- **apiserver**: IP address of the API server where the application cluster is deployed. Ensure that the IP address can be accessed from the Jenkins cluster.

```

//Define the code repository address.
def git_url = 'https://github.com/lookforstar/jenkins-demo.git'
//Define the SWR login command.
def swr_login = 'docker login -u ap-southeast-1@xxxxx -p xxxx swr.cn-east-3.myhuaweicloud.com'
//Define the SWR zone.
def swr_region = 'ap-southeast-1'
//Define the name of the SWR organization to be uploaded.
def organization = 'container'
//Define the image name.
def build_name = 'jenkins-demo'
//Certificate ID of the cluster to be deployed
def credential = 'k8s-token'
//API server address of the cluster. Ensure that the address can be accessed from the Jenkins cluster.
def apiserver = 'https://192.168.0.100:6443'

pipeline {

```

```
agent any
stages {
    stage('Clone') {
        steps{
            echo "1.Clone Stage"
            git url: git_url
            script {
                build_tag = sh(returnStdout: true, script: 'git rev-parse --short HEAD').trim()
            }
        }
    }
    stage('Test') {
        steps{
            echo "2.Test Stage"
        }
    }
    stage('Build') {
        steps{
            echo "3.Build Docker Image Stage"
            sh "docker build -t swr.${swr_region}.myhuaweicloud.com/${organization}/${build_name}:$ {build_tag} ."
            // ${build_tag} indicates that the build_tag variable is obtained as the image tag. It is the return value of the git rev-parse --short HEAD command, that is, commit ID.
        }
    }
    stage('Push') {
        steps{
            echo "4.Push Docker Image Stage"
            sh swr_login
            sh "docker push swr.${swr_region}.myhuaweicloud.com/${organization}/${build_name}:$ {build_tag}"
        }
    }
    stage('Deploy') {
        steps{
            echo "5. Deploy Stage"
            echo "This is a deploy step to test"
            script {
                sh "cat k8s.yaml"
                echo "begin to config kubernetes"
                try {
                    withKubeConfig([credentialsId: credential, serverUrl: apiserver]) {
                        sh 'kubectl apply -f k8s.yaml'
                        //The YAML file is stored in the code repository. The following is only an example. Replace it as required.
                    }
                    println "hooray, success"
                } catch (e) {
                    println "oh no! Deployment failed! "
                    println e
                }
            }
        }
    }
}
```

**Step 4** Save the settings and execute the Jenkins job.

----End

#### 4.1.3.4 Interconnecting Jenkins with RBAC of Kubernetes Clusters (Example)

##### Prerequisites

RBAC must be enabled for the cluster.

## Scenario 1: Namespace-based Permissions Control

Create a service account and a role, and add a RoleBinding.

```
$ kubectl create ns dev
$ kubectl -n dev create sa dev

$ cat <<EOF > dev-user-role.yml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: dev
  name: dev-user-pod
rules:
- apiGroups: ["*"]
  resources: ["deployments", "pods", "pods/log"]
  verbs: ["get", "watch", "list", "update", "create", "delete"]
EOF
kubectl create -f dev-user-role.yml

$ kubectl create rolebinding dev-view-pod \
  --role=dev-user-pod \
  --serviceaccount=dev:dev \
  --namespace=dev
```

Generate the kubeconfig file of a specified service account.



- In clusters earlier than v1.21, a token is obtained by mounting the secret of the service account to a pod. Tokens obtained this way are permanent. This approach is no longer recommended starting from version 1.21. Service accounts will stop auto creating secrets in clusters from version 1.25.

In clusters of version 1.21 or later, you can use the [TokenRequest API](#) to [obtain the token](#) and use the projected volume to mount the token to the pod. Such tokens are valid for a fixed period. When the mounting pod is deleted, the token automatically becomes invalid. For details, see [Service Account Token Security Improvement](#).

- If you need a token that never expires, you can also [manually manage secrets for service accounts](#). Although a permanent service account token can be manually created, you are advised to use a short-lived token by calling the [TokenRequest API](#) for higher security.

```
$ SECRET=$(kubectl -n dev get sa dev -o go-template='{{range .secrets}}{{.name}}{{end}}')
$ API_SERVER="https://172.22.132.51:6443"
$ CA_CERT=$(kubectl -n dev get secret ${SECRET} -o yaml | awk '/ca.crt:/{print $2}')
$ cat <<EOF > dev.conf
apiVersion: v1
kind: Config
clusters:
- cluster:
  certificate-authority-data: $CA_CERT
  server: $API_SERVER
  name: cluster
EOF

$ TOKEN=$(kubectl -n dev get secret ${SECRET} -o go-template='{{.data.token}}')
$ kubectl config set-credentials dev-user \
  --token=`echo ${TOKEN} | base64 -d` \
  --kubeconfig=dev.conf

$ kubectl config set-context default \
  --cluster=cluster \
  --user=dev-user \
  --kubeconfig=dev.conf

$ kubectl config use-context default \
  --kubeconfig=dev.conf
```

## Verification in the CLI

```
$ kubectl --kubeconfig=dev.conf get po
Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:dev:dev" cannot list pods in the namespace "default"
```

```
$ kubectl -n dev --kubeconfig=dev.conf run nginx --image nginx --port 80 --restart=Never
$ kubectl -n dev --kubeconfig=dev.conf get po
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   0          39s
```

### Verify whether the permissions meet the expectation in Jenkins.

**Step 1** Add the kubeconfig file with permissions control settings to Jenkins.

**Step 2** Start the Jenkins job. In this example, Jenkins fails to be deployed in namespace **default** but is successfully deployed in namespace **dev**.

```
* cat k8s.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: jenkins-demo
  namespace: default
spec:
  template:
    metadata:
      labels:
        app: jenkins-demo
    spec:
      containers:
        - image: cnyh/jenkins-demo:716a813
          imagePullPolicy: IfNotPresent
          name: jenkins-demo
          env:
            - name: branch
              value: <BRANCH_NAME>
      [Pipeline] echo
      begin to config kubernetes
      [Pipeline] kubernetesDeploy
      Starting Kubernetes deployment
      Loading configuration: /var/jenkins_home/workspace/liuyi-rvr-cce-pipe01/k8s.yaml
      Created Deployment: Deployment@v1(annotations=null, clusterName=null, creationTimestamp=2019-02-18T08:05:47Z, deletionGracePeriodSeconds=null, deletionTimestamp=null, finalizers=[], generateName=null, generation=1, initializers=null, labels=[app:jenkins-demo, name:jenkins-demo, namespace:dev, ownerReferences=null], resourceVersion=862129, selfLink=/apis/extensions/v1beta1/namespaces/dev/deployments/jenkins-demo, uid:ffd4d740-x-3553-11e9-9411-fa153e99047, additionalProperties=null), spec=DeploymentSpec(minReadySeconds=null, pause=false, progressDeadlineSeconds=null, replicas=1, revisionHistoryList=null, rollbackTo=null, selector=LabelSelector(matchExpressions=[], matchLabels=[app:jenkins-demo], additionalProperties=null), strategy=DeploymentStrategy(reconcilingUpdate=RollingUpdateStrategy(type=RollingUpdateType(rollbackEnabled=true), maxSurge=InOrString(IntVal=1, Kind=Null, StrVal=null, additionalProperties=null)), type=RollingUpdate, additionalProperties=null)), template=PodTemplateSpec(metadata=ObjectMeta(name:jenkins-demo, namespace:dev, ownerReferences=null, resources=ResourceRequirements(limits=null, requests=null), termination=Termination(type=Delete, gracePeriodSeconds=30, terminationMessagePath=/dev/termination-log, terminationMessagePolicy=File, tty=null, volumeMonitor=null, additionalProperties=null)), spec=ContainerSpec(image=ImageSpec(name:jenkins-demo, port=32000, readinessProbe=ReadinessProbe(name:jenkins-demo, port=32000, resources=ResourceRequirements(limits=null, requests=null), termination=Termination(type=Delete, gracePeriodSeconds=30, terminationMessagePath=/dev/termination-log, terminationMessagePolicy=File, tty=null, volumeMonitor=null, additionalProperties=null)), terminationMessagePath=/dev/termination-log, terminationMessagePolicy=File, tty=null, volumeMonitor=null, additionalProperties=null), additionalProperties=null), securityContext=SecurityContext(hostNetwork=null, hostPID=null, hostPorts=null, imagePullSecrets=[], initContainers=[], nodeName=null, nodeSelector=null, restartPolicy=Always, schedulerName=null, hostAliases=null, hostIPC=null, terminationGracePeriodSeconds=30, tolerations=[], volumeName=null, supplementalGroups=[], additionalProperties=null), additionalProperties=null), status=DeploymentStatus(availableReplicas=null, collisionCount=null, conditions=[], observedGeneration=null, readyReplicas=null, replicas=null, unavailableReplicas=null, updatedReplicas=null, additionalProperties=null), additionalProperties=null)
      Finished Kubernetes deployment
      [Pipeline] echo
      hooray, success
```

----End

## Scenario 2: Resource-based Permissions Control

**Step 1** Generate the service account, role, and binding.

```
kubectl -n dev create sa sa-test0304
```

```
cat <<EOF > test0304-role.yaml
```

```

kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: dev
  name: role-test0304
rules:
- apiGroups: ["*"]
  resources: ["deployments"]
  resourceNames: ["tomcat03", "tomcat04"]
  verbs: ["get", "update", "patch"]
EOF
kubectl create -f test0304-role.yml

kubectl create rolebinding test0304-bind \
--role=role-test0304 \
--serviceaccount=dev:sa-test0304\
--namespace=dev

```

## Step 2 Generate the kubeconfig file.



- In clusters earlier than v1.21, a token is obtained by mounting the secret of the service account to a pod. Tokens obtained this way are permanent. This approach is no longer recommended starting from version 1.21. Service accounts will stop auto creating secrets in clusters from version 1.25.

In clusters of version 1.21 or later, you can use the [TokenRequest API](#) to [obtain the token](#) and use the projected volume to mount the token to the pod. Such tokens are valid for a fixed period. When the mounting pod is deleted, the token automatically becomes invalid. For details, see [Service Account Token Security Improvement](#).

- If you need a token that never expires, you can also [manually manage secrets for service accounts](#). Although a permanent service account token can be manually created, you are advised to use a short-lived token by calling the [TokenRequest API](#) for higher security.

```

SECRET=$(kubectl -n dev get sa sa-test0304 -o go-template='{{range .secrets}}{{.name}}{{end}}')
API_SERVER="https://192.168.0.153:5443"
CA_CERT=$(kubectl -n dev get secret ${SECRET} -o yaml | awk '/ca.crt:/{print $2}')
cat <<EOF > test0304.conf
apiVersion: v1
kind: Config
clusters:
- cluster:
  certificate-authority-data: $CA_CERT
  server: $API_SERVER
  name: cluster
EOF

TOKEN=$(kubectl -n dev get secret ${SECRET} -o go-template='{{.data.token}}')
kubectl config set-credentials test0304-user \
--token=`echo ${TOKEN} | base64 -d` \
--kubeconfig=test0304.conf

kubectl config set-context default \
--cluster=cluster \
--user=test0304-user \
--kubeconfig=test0304.conf

kubectl config use-context default \
--kubeconfig=test0304.conf

```

## Step 3 Verify that Jenkins is running as expected.

In the pipeline script, update the Deployments of tomcat03, tomcat04, and tomcat05 in sequence.

```

try {
  kubernetesDeploy(

```

```
kubeconfigId: "test0304",
configs: "test03.yaml")
println "hooray, success"
} catch (e) {
    println "oh no! Deployment failed! "
    println e
}
echo "test04"
try {
    kubernetesDeploy(
        kubeconfigId: "test0304",
        configs: "test04.yaml")
    println "hooray, success"
} catch (e) {
    println "oh no! Deployment failed! "
    println e
}
echo "test05"
try {
    kubernetesDeploy(
        kubeconfigId: "test0304",
        configs: "test05.yaml")
    println "hooray, success"
} catch (e) {
    println "oh no! Deployment failed! "
    println e
}
```

Viewing the running result:

Figure 4-15 test03

```
test03
[Pipeline] kubernetesDeploy
Starting Kubernetes deployment
Loading configuration: /var/jenkins_home/workspace/liuyi-sw...
Applied Deployment: Deployment(apiVersion=extensions/v1beta1,
deletionTimestamp=null, finalizers=[], generateName=null, ge...
selfLink=/apis/extensions/v1beta1/namespaces/dev/deployments...
progressDeadlineSeconds=null, replicas=1, revisionHistoryLi...
strategy=DeploymentStrategy(rollingUpdate=RollingUpdateDepl...
additionalProperties={}), additionalProperties={}), type=R...
deletionGracePeriodSeconds=null, deletionTimestamp=null, fir...
resourceVersion=null, selfLink=null, uid=null, additionalPro...
[EnvVar(name=branch, value=<BRANCH_NAME>, valueFrom=null, ac...
livenessProbe=null, name=tomcat03, ports=[], readinessProbe=...
terminationMessagePath=/dev/termination-log, terminationMes...
hostNetwork=null, hostPID=null, hostname=null, imagePullSecr...
securityContext=PodSecurityContext(fsGroup=null, runAsNonRoot...
terminationGracePeriodSeconds=30, tolerations=[], volumes=[...
conditions=[DeploymentCondition(lastTransitionTime=2019-02-...
type=Available, additionalProperties={}), DeploymentCondition...
reason>NewReplicaSetAvailable, status=True, type=Progressing...
additionalProperties={})
Finished Kubernetes deployment
[Pipeline] echo
hooray, success
```

Figure 4-16 test04

```
test04
[Pipeline] kubernetesDeploy
Starting Kubernetes deployment
Loading configuration: /var/jenkins_home/workspace/liuyi-swr-cce-pipe01/test04.yaml
Applied Deployment: Deployment(apiVersion=extensions/v1beta1, kind=Deployment, metadata=ObjectMeta(deletionTimestamp=null, finalizers=[], generateName=null, generation=3, initializers=null, labels={name=test04}, name=test04, namespace=default, resourceVersion="1234567890", selfLink="/apis/extensions/v1beta1/namespaces/default/deployments/tomcat04", uid="06af3b14-3356-11e8-84d3-00163a2f3b14"), spec=DeploymentSpec(replicas=1, selector=LabelSelector(selectorLabels={name=tomcat04}), strategy=DeploymentStrategy(rollingUpdate=RollingUpdateDeployment(maxSurge="IntOrString(1)", maxUnavailable="0", rollingUpdatePeriodSeconds=30), type=RollingUpdate), template=PodTemplateSpec(containers=[Container(name=tomcat04, image="tomcat:8.0", ports=[ContainerPort(containerPort=80)], resources=ResourceRequirements(limits={}, requests={})), envFrom=[EnvVarSource(secretKeyRef=SecretKeySelector(name="branch-secret", key="branch"))], terminationMessagePath="/dev/termination-log", terminationMessagePolicy=File), volumeMounts=[VolumeMount(name="branch-secret", mountPath="/etc/branch")]), status=DeploymentStatus(lastUpdated="2019-02-18T08:56:55Z", lastTransitionTime="2019-02-18T08:56:55Z", observedGeneration=3, replicas=1, readyReplicas=1, status="True", type="Available"), conditions=[DeploymentCondition(lastTransitionTime="2019-02-18T08:56:55Z", lastUpdateTime="2019-02-18T08:56:55Z", status="True", type="Available"), DeploymentCondition(lastTransitionTime="2019-02-18T08:56:55Z", lastUpdateTime="2019-02-18T08:56:55Z", reason="ReplicaSetUpdated", status="True", type="Progressing")])
Finished Kubernetes deployment
[Pipeline] echo
hooray, success
[Pipeline] echo
test05
[Pipeline] kubernetesDeploy
Starting Kubernetes deployment
Loading configuration: /var/jenkins_home/workspace/liuyi-swr-cce-pipe01/test05.yaml
ERROR: ERROR: io.fabric8.kubernetes.client.KubernetesClientException: Failure executing: GET /apis/extensions/deployments/test0304-user. User doesn't have permission.
ERROR: io.fabric8.kubernetes.client.KubernetesClientException: Forbidden! User test0304-user doesn't have permission.
----End
```

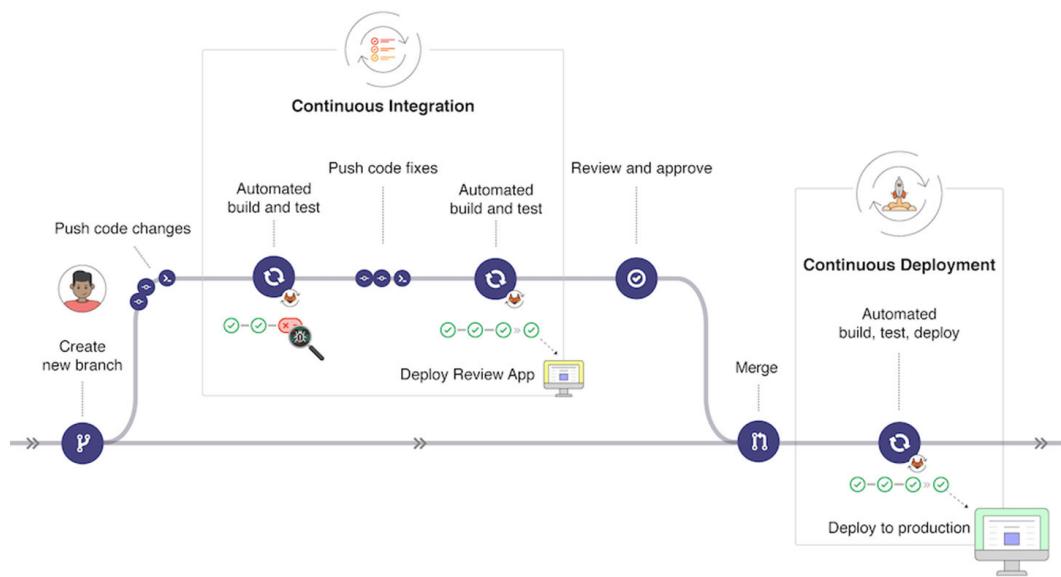
## 4.2 Interconnecting GitLab with SWR and CCE for CI/CD

### Challenges

GitLab is an open-source version management system developed with Ruby on Rails for Git project repository management. It supports web-based access to public and private projects. Similar to GitHub, GitLab allows you to browse source code, manage bugs and comments, and control team member access to repositories. You will find it very easy to view committed versions and file history database. Team members can communicate with each other using the built-in chat program (Wall).

GitLab provides powerful CI/CD functions and is widely used in software development.

**Figure 4-17 GitLab CI/CD process**

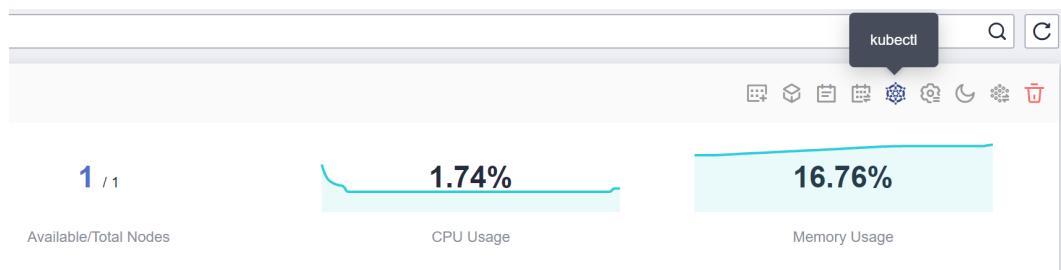


This section describes how to interconnect GitLab with SWR and CCE for CI/CD.

## Preparations

1. Create a CCE cluster and a node and bind an EIP to the node for downloading an image during GitLab Runner installation.
2. Download and configure kubectl to connect to the cluster.

Log in to the CCE console, click  on the right of the cluster, and configure kubectl as instructed.



3. **Install Helm 3.**

## Installing GitLab Runner

Log in to [GitLab](#), choose **Settings > CI/CD** in the project view, click **Expand** next to **Runners**, and search for the GitLab Runner registration URL and token.

The screenshot shows the GitLab interface with the sidebar collapsed. The main area is titled 'Runners' under the 'CI/CD' section. It includes sections for 'Specific runners' (runners specific to this project) and 'Shared runners' (runners shared across the instance). A 'Set up a specific runner for a project' section provides instructions: 1. Install GitLab Runner and ensure it's running, 2. Register the runner with this URL (<https://gitlab.com/>). Below this is a registration token input field with placeholder text 'And this registration token: [REDACTED]'. Buttons for 'Reset registration token' and 'Show runner installation instructions' are present. On the right, a section for 'Shared runners' notes that they run in autoscale mode and are powered by Google Cloud Platform. A toggle switch 'Enable shared runners for this project' is turned on, and the count 'Available shared runners: 42' is displayed.

Create the **values.yaml** file and fill in the following information:

```
# Registration URL
gitlabUrl: https://gitlab.com/
# Registration token
runnerRegistrationToken: "GR13489411dKVzmTyaywEDTF_1QXb"
rbac:
  create: true
runners:
  privileged: true
```

Create a GitLab namespace.

```
kubectl create namespace gitlab
```

Install GitLab Runner using Helm.

```
helm repo add gitlab https://charts.gitlab.io
helm install --namespace gitlab gitlab-runner -f values.yaml gitlab/gitlab-runner
```

After the installation, you can query the workload of gitlab-runner on the CCE console and view the connection information in GitLab later.

The screenshot shows the CCE console interface with the sidebar collapsed. The main area displays a list of 'Available specific runners'. One runner is highlighted with a red border: '#14806619 (Nob3Ymgb)' with status 'Up' and IP 'gitlab-runner-gitlab-runner-7d76d4fdbf-z5gfr'. Buttons for 'Remove runner' and 'Edit' are visible. Other runners listed include '#11574084 (EuhIQzPR)' and '#12270835 (zxwgk4AP)'. A section for 'Shared runners' is also visible, showing a list of runners associated with the namespace '3-green.shared-gitlab-org.runners-manager.gitlab.com.gitlab-org'.

## Creating an Application

Place the application to be created in the GitLab project repository. This section takes Nginx modification as an example. For details, visit <https://gitlab.com/c8147/cidemo/-/tree/main>.

The following files are included:

- **.gitlab-ci.yml**: Gitlab CI file, which will be described in detail in [Creating a Pipeline](#).
- **Dockerfile**: used to build Docker images.
- **index.html**: used to replace the index page of Nginx.
- **k8s.yaml**: used to deploy the Nginx app. A Deployment named **nginx-test** and a Service named **nginx-test** will be created.

The preceding files are only examples. You can replace or modify them accordingly.

## Configuring Global Variables

When using pipelines, you need to build an image, upload it to SWR, and run kubectl commands to deploy the image in the cluster. Before performing these operations, you must log in to SWR and obtain the credential for connecting to the cluster. You can define the information as variables in GitLab.

Log in to [GitLab](#), choose **Settings > CI/CD** in the project view, and click **Expand** next to **Variables** to add variables.

Type	Key	Value	Protected	Masked	Environments
Variable	kube_config	*****	✓	✗	All (default)
Variable	project	*****	✓	✗	All (default)
Variable	swr_ak	*****	✓	✗	All (default)
Variable	swr_sk	*****	✓	✗	All (default)

- **kube\_config**

**kubeconfig.json** file used for kubectl command authentication. Run the following command on the host where kubectl is configured to convert the file to the Base64 format:

```
echo $(cat ~/.kube/config | base64) | tr -d " "
```

The command output is the content of **kubeconfig.json**.

- **project**: project name.

Log in to the management console, click your username in the upper right corner, and click **My Credentials**. In the **Projects** area on the **API Credentials** page, check the name of the project in your current region.

- **swr\_ak**: access key.

Log in to the management console, click your username in the upper right corner, and click **My Credentials**. In the navigation pane on the left, choose **Access Keys**. Click **Create Access Key**, enter the description, and click **OK**. In the displayed **Information** dialog box, click **Download**. After the certificate is downloaded, obtain the AK and SK information from the **credentials** file.

- **swr\_sk**: secret key for logging in to SWR.

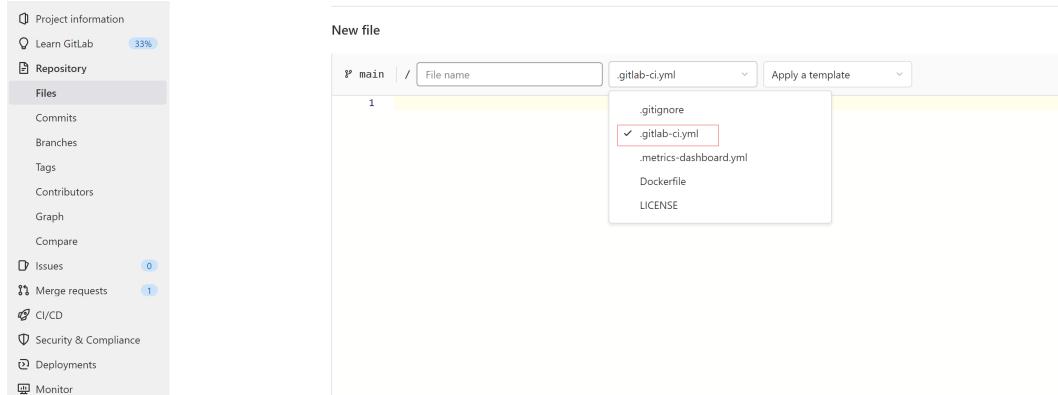
Run the following command to obtain the key pair. Replace **\$AK** and **\$SK** with the AK and SK obtained in the preceding steps.

```
printf "$AK" | openssl dgst -binary -sha256 -hmac "$SK" | od -An -vtx1 | sed 's/[ \n]//g' | sed 'N;s/\n//'
```

The command output displays the login key pair.

## Creating a Pipeline

Log in to **Gitlab** and add the **.gitlab-ci.yml** file to **Repository**.

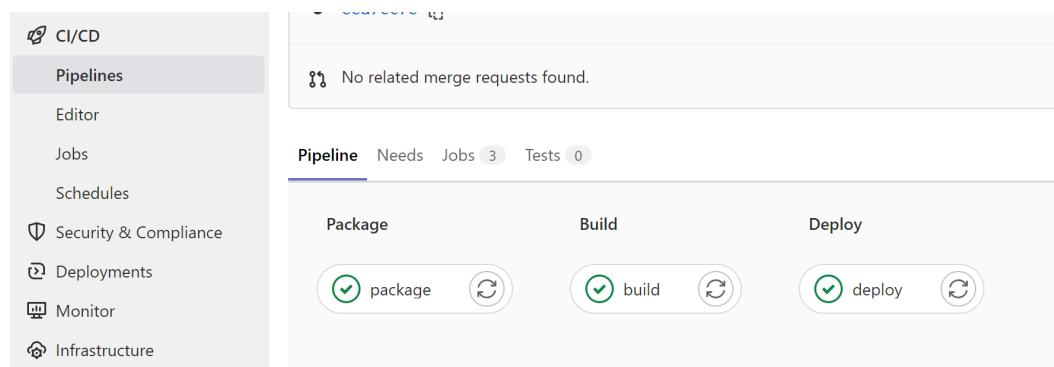


The content is as follows:

```
# Define pipeline stages, including package, build, and deploy.
stages:
  - package
  - build
  - deploy
# If no image is specified in each stage, the default image docker:latest is used.
image: docker:latest
# In the package stage, only printing is performed.
package:
  stage: package
  script:
    - echo "package"
# In the build stage, the Docker-in-Docker mode is used.
build:
  stage: build
  # Define environment variables for the build stage.
  variables:
    DOCKER_HOST: tcp://docker:2375
  # Define the image for running Docker-in-Docker.
  services:
    - docker:18.09-dind
  script:
    - echo "build"
    # Log in to SWR.
    - docker login -u $project@$swr_ak -p $swr_sk swr.ap-southeast-1.myhuaweicloud.com
    # Build an image. k8s-dev is the organization name in SWR. Replace it to the actual name.
    - docker build -t swr.ap-southeast-1.myhuaweicloud.com/k8s-dev/nginx:$CI_PIPELINE_ID .
    # Push the image to SWR.
    - docker push swr.ap-southeast-1.myhuaweicloud.com/k8s-dev/nginx:$CI_PIPELINE_ID
```

```
deploy:  
  # Use the kubectl image.  
  image:  
    name: bitnami/kubectl:latest  
    entrypoint: [""]  
  stage: deploy  
  script:  
    # Configure the kubeconfig file.  
    - echo $kube_config |base64 -d > $KUBECONFIG  
    # Replace the image in the k8s.yaml file.  
    - sed -i "s/<IMAGE_NAME>/swr.ap-southeast-1.myhuaweicloud.com/k8s-dev/nginx:$CI_PIPELINE_ID/g"  
k8s.yaml  
  - cat k8s.yaml  
  # Deploy an application.  
  - kubectl apply -f k8s.yaml
```

After the **.gitlab-ci.yml** file is saved, the pipeline is started immediately. You can view the pipeline execution status in GitLab.



## Verifying Deployment

After the pipeline is deployed, locate the **nginx-test** Service on the CCE console, query its access address, and run the **curl** command to access the Service.

```
# curl xxx.xxx.xxx.xxx:31111  
Hello Gitlab!
```

If the preceding information is displayed, the deployment is correct.

# 5 Disaster Recovery

## 5.1 Implementing High Availability for Containers in CCE

### Basic Principles

To achieve high availability for your CCE containers, you can do as follows:

1. Deploy three master nodes for the cluster.
2. When nodes are deployed across AZs, set custom scheduling policies based on site requirements to maximize resource utilization.
3. Create multiple node pools in different AZs and use them for node scaling.
4. Set the number of pods to be greater than 2 when creating a workload.
5. Set pod affinity rules to distribute pods to different AZs and nodes.

### Procedure

Assume that there are four nodes in a cluster distributed in the following AZs:

```
$ kubectl get node -L topology.kubernetes.io/zone,kubernetes.io/hostname
NAME      STATUS  ROLES   AGE VERSION      ZONE      HOSTNAME
192.168.5.112 Ready   <none>  42m v1.21.7-r0-CCE21.11.1.B007  ap-southeast-3a  192.168.5.112
192.168.5.179 Ready   <none>  42m v1.21.7-r0-CCE21.11.1.B007  ap-southeast-3a  192.168.5.179
192.168.5.252 Ready   <none>  37m v1.21.7-r0-CCE21.11.1.B007  ap-southeast-3b  192.168.5.252
192.168.5.8  Ready   <none>  33h v1.21.7-r0-CCE21.11.1.B007  ap-southeast-3c  192.168.5.8
```

Create workloads according to the following two podAntiAffinity rules:

- The first one is the pod anti-affinity in an AZ. Set the parameters as follows:
  - **weight**: A larger weight value indicates a higher priority. In this example, set it to **50**.
  - **topologyKey**: a default or custom key for the node label that the system uses to denote a topology domain. A topology key determines the scope where the pod should be scheduled to. In this example, set this parameter to **topology.kubernetes.io/zone**, which is the label for identifying the AZ where the node is located.

- **labelSelector:** Select the label of the workload to realize the anti-affinity between this container and the workload.
- The second one is the pod anti-affinity in the node host name. Set the parameters as follows:
  - **weight:** Set it to **50**.
  - **topologyKey:** Set it to **kubernetes.io/hostname**.
  - **labelSelector:** Select the label of the pod, which is anti-affinity with the pod.

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - weight: 50
              podAffinityTerm:
                labelSelector: # Select the label of the workload to realize the anti-affinity
between this container and the workload.
                matchExpressions:
                  - key: app
                    operator: In
                    values:
                      - nginx
                namespaces:
                  - default
                topologyKey: topology.kubernetes.io/zone # It takes effect in the same AZ.
            - weight: 50
              podAffinityTerm:
                labelSelector: # Select the label of the workload to realize the anti-affinity
between this container and the workload.
                matchExpressions:
                  - key: app
                    operator: In
                    values:
                      - nginx
                namespaces:
                  - default
                topologyKey: kubernetes.io/hostname # It takes effect on the node.
        imagePullSecrets:
          - name: default-secret
```

Create a workload and view the node where the pod is located.

```
$ kubectl get pod -owide
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE
nginx-6ffffd8d664-dpwbk  1/1   Running   0        17s   10.0.0.132  192.168.5.112
nginx-6ffffd8d664-qhclc  1/1   Running   0        17s   10.0.1.133  192.168.5.252
```

Increase the number of pods to 3. The pod is scheduled to another node, and the three nodes are in three different AZs.

```
$ kubectl scale --replicas=3 deploy/nginx
deployment.apps/nginx scaled
$ kubectl get pod -owide
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE
nginx-6ffffd8d664-8t7rv  1/1   Running   0        3s    10.0.0.9   192.168.5.8
nginx-6ffffd8d664-dpwbk  1/1   Running   0        2m45s  10.0.0.132  192.168.5.112
nginx-6ffffd8d664-qhclc  1/1   Running   0        2m45s  10.0.1.133  192.168.5.252
```

Increase the number of pods to 4. The pod is scheduled to the last node. With podAntiAffinity rules, pods can be evenly distributed to AZs and nodes.

```
$ kubectl scale --replicas=4 deploy/nginx
deployment.apps/nginx scaled
$ kubectl get pod -owide
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE
nginx-6ffffd8d664-8t7rv  1/1   Running   0        2m30s  10.0.0.9   192.168.5.8
nginx-6ffffd8d664-dpwbk  1/1   Running   0        5m12s  10.0.0.132  192.168.5.112
nginx-6ffffd8d664-h796b  1/1   Running   0        78s   10.0.1.5    192.168.5.179
nginx-6ffffd8d664-qhclc  1/1   Running   0        5m12s  10.0.1.133  192.168.5.252
```

# 6 Security

## 6.1 Suggestions on Selecting CCE Clusters

Based on the shared security responsibility model, CCE safeguards the master nodes in a cluster and CCE components, and provides a series of hierarchical security capabilities at the cluster and container levels. Users are responsible for the security of cluster nodes and comply with the security best practices provided by CCE to perform security configuration and O&M.

### CCE Application Scenarios

CCE is a container service built on popular Docker and Kubernetes technologies and offers a wealth of features best suited to enterprises' demand for running container clusters at scale. With unique advantages in system reliability, performance, and compatibility with open-source communities, CCE can suit the diverse needs of enterprises interested in building container clouds.

CCE provides a function list and typical application scenarios. For details about the function list, see [Function Overview](#). For details about the application scenarios, see [Application Scenarios](#).

### Exception Scenarios

You are not advised to use clusters in scenarios that require strong resource isolation. CCE provides tenants with a dedicated, exclusive cluster. Currently, resources such as nodes and networks are not strictly isolated. If no strict security protection measures are available, security risks exist when the cluster is used by multiple external uncontrollable users at the same time. For example, in a development pipeline scenario, when multiple users are allowed to use the pipeline, the service code logic of different users is uncontrollable, and the cluster and services in the cluster may be attacked.

### Enabling HSS

Host Security Service (HSS) provides host management, risk prevention, intrusion detection, advanced defense, security operations, and web page anti-tamper functions to comprehensively identify and manage information assets on hosts,

monitor risks on hosts in real time, and prevent unauthorized intrusions. You are advised to enable HSS to protect hosts in CCE clusters. For details about HSS and how to use it, see [HSS](#).

## Enabling CGS

CCE can be used together with Container Guard Service (CGS). CGS scans vulnerabilities and configurations in images, helping enterprises detect the container environment, which cannot be found by the traditional security software. CGS also delivers functions such as process whitelist configuration, read-only file protection, and container escape detection to minimize the security risks for a running container. For details about CGS and how to use it, see [CGS](#).

## 6.2 Cluster Security

For security purposes, you are advised to configure a cluster as follows.

### Using the CCE Cluster of the Latest Version

Kubernetes releases a major version in about four months. CCE follows the same frequency as Kubernetes to release major versions. To be specific, a new CCE version is released about three months after a new Kubernetes version is released in the community. For example, Kubernetes v1.19 was released in September 2020 and CCE v1.19 was released in March 2021.

The latest cluster version has known vulnerabilities fixed or provides a more comprehensive security protection mechanism. You are advised to select the latest cluster version when creating a cluster. Before a cluster version is deprecated and removed, upgrade your cluster to a supported version.

### Handling Vulnerabilities Released on the Official Website Promptly

CCE releases vulnerabilities irregularly. You need to handle the vulnerabilities in a timely manner. For details, see [Vulnerability Notice](#).

### Disabling the Automatic Token Mounting Function of the Default Service Account

By default, Kubernetes associates the default service account with every pod. That is, the token is mounted to a container. The container can use this token to pass the authentication by the kube-apiserver and kubelet components. In a cluster with RBAC disabled, the service account who owns the token has the control permissions for the entire cluster. In a cluster with RBAC enabled, the permissions of the service account who owns the token depends on the roles associated by the administrator. The service account's token is generally used by workloads that need to access kube-apiserver, such as coredns, autoscaler, and prometheus. For workloads that do not need to access kube-apiserver, you are advised to disable the automatic association between the service account and token.

Two methods are available:

- Method 1: Set the **automountServiceAccountToken** field of the service account to **false**. After the configuration is complete, newly created workloads

will not be associated with the default service account by default. Set this field for each namespace as required.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
automountServiceAccountToken: false
...
```

When a workload needs to be associated with a service account, explicitly set the **automountServiceAccountToken** field to **true** in the YAML file of the workload.

```
...
spec:
  template:
    spec:
      serviceAccountName: default
      automountServiceAccountToken: true
...
```

- Method 2: Explicitly disable the function of automatically associating with service accounts for workloads.

```
...
spec:
  template:
    spec:
      automountServiceAccountToken: false
...
```

## Configuring Proper Cluster Access Permissions for Users

CCE allows you to create multiple IAM users. Your account can create different user groups, assign different access permissions to different user groups, and add users to the user groups with corresponding permissions when creating IAM users. In this way, users can control permissions on different regions and assign read-only permissions. Your account can also assign namespace-level permissions for users or user groups. To ensure security, it is advised that minimum user access permissions are assigned.

If you need to create multiple IAM users, configure the permissions of the IAM users and namespaces properly.

- For details about how to configure cluster permissions, see [Assigning Cluster-level Permissions](#).
- For details about how to configure namespace permissions, see [Assigning Namespace-level Permissions](#).

## Configuring Resource Quotas for Cluster Namespaces

CCE provides resource quota management, which allows users to limit the total amount of resources that can be allocated to each namespace. These resources include CPU, memory, storage volumes, pods, Services, Deployments, and StatefulSets. Proper configuration can prevent excessive resources created in a namespace from affecting the stability of the entire cluster.

For details, see [Setting Resource Quotas and Constraints](#).

## Configuring LimitRange for Containers in a Namespace

With resource quotas, cluster administrators can restrict the use and creation of resources by namespace. In a namespace, a pod or container can use the

maximum CPU and memory resources defined by the resource quota of the namespace. In this case, a pod or container may monopolize all available resources in the namespace. You are advised to configure LimitRange to restrict resource allocation within the namespace. The LimitRange parameter has the following restrictions:

- Limits the minimum and maximum resource usage of each pod or container in a namespace.

For example, create the maximum and minimum CPU usage limits for a pod in a namespace as follows:

cpu-constraints.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-min-max-demo-lr
spec:
  limits:
    - max:
        cpu: "800m"
      min:
        cpu: "200m"
    type: Container
```

Then, run **kubectl -n <namespace> create -f cpu-constraints.yaml** to complete the creation. If the default CPU usage is not specified for the container, the platform automatically configures the default CPU usage. That is, the default configuration is automatically added after the container is created.

```
...
spec:
  limits:
    - default:
        cpu: 800m
      defaultRequest:
        cpu: 800m
    max:
      cpu: 800m
    min:
      cpu: 200m
    type: Container
```

- Limits the maximum and minimum storage space that each PersistentVolumeClaim can apply for in a namespace.

storagelimit.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: storagelimit
spec:
  limits:
    - type: PersistentVolumeClaim
      max:
        storage: 2Gi
      min:
        storage: 1Gi
```

Then, run **kubectl -n <namespace> create -f storagelimit.yaml** to complete the creation.

## Configuring Network Isolation in a Cluster

- Container tunnel network

If networks need to be isolated between namespaces in a cluster or between workloads in the same namespace, you can configure network policies to isolate the networks. For details, see [Network Policies](#).

- Cloud Native Network 2.0

In the Cloud Native Network 2.0 model, you can configure security groups to isolate networks between pods. For details, see [SecurityGroups](#).

- VPC network

Network isolation is not supported.

## Enabling the Webhook Authentication Mode with kubelet

### NOTICE

CCE clusters of v1.15.6-r1 or earlier are involved, whereas versions later than v1.15.6-r1 are not.

Upgrade the CCE cluster version to 1.13 or 1.15 and enable the RBAC capability for the cluster. If the version is 1.13 or later, no upgrade is required.

When creating a node, you can enable the kubelet authentication mode by injecting the **postinstall** file (by setting the kubelet startup parameter **--authorization-node=Webhook**).

**Step 1** Run the following command to create clusterrolebinding:

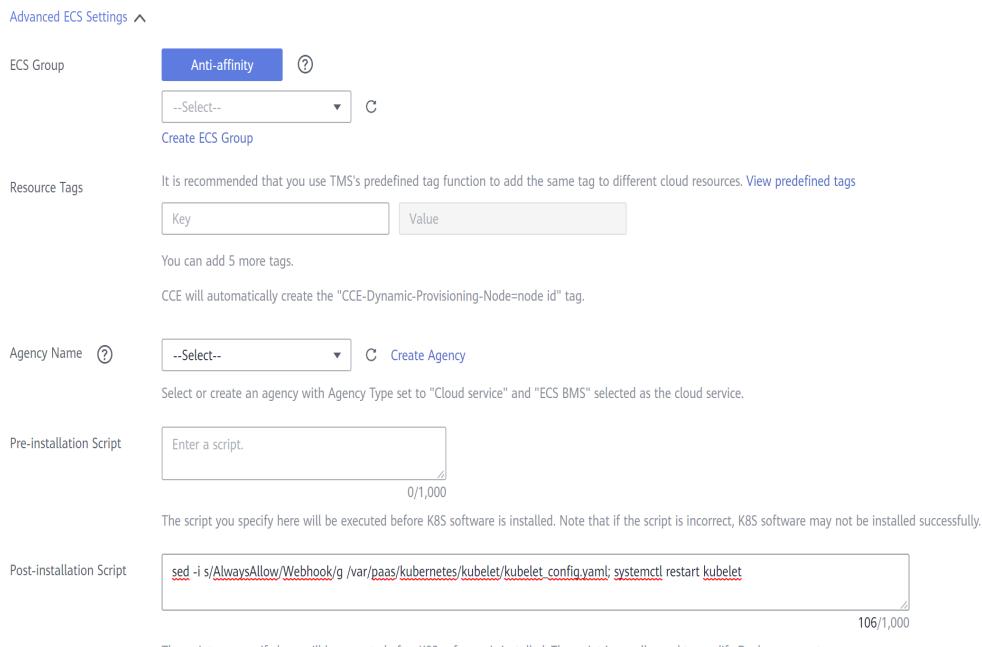
```
kubectl create clusterrolebinding kube-apiserver-kubelet-admin --clusterrole=system:kubelet-api-admin --user=system:kube-apiserver
```

**Step 2** For an existing node, log in to the node, change **authorization mode** in **/var/paas/kubernetes/kubelet/kubelet\_config.yaml** on the node to **Webhook**, and restart kubelet.

```
sed -i s/AlwaysAllow/Webhook/g /var/paas/kubernetes/kubelet/kubelet_config.yaml; systemctl restart kubelet
```

**Step 3** For a new node, add the following command to the post-installation script to change the kubelet permission mode:

```
sed -i s/AlwaysAllow/Webhook/g /var/paas/kubernetes/kubelet/kubelet_config.yaml; systemctl restart kubelet
```



----End

## Uninstalling web-terminal After Use

The web-terminal add-on can be used to manage CCE clusters. Keep the login password secure and uninstall the add-on when it is no longer needed.

## 6.3 Node Security

### Handling Vulnerabilities Released on the Official Website Promptly

Before releasing a new image, fix the node vulnerabilities by referring to [Vulnerability Notice](#).

### Preventing Nodes from Being Exposed to Public Networks

- Do not bind an EIP to a node unless necessary to reduce the attack surface.
- If an EIP must be used, properly configure the firewall or security group rules to restrict access of unnecessary ports and IP addresses.

You may have configured the **kubeconfig.json** file on a node in your cluster. kubectl can use the certificate and private key in this file to control the entire cluster. You are advised to delete unnecessary files from the **/root/.kube** directory on the node to prevent malicious use.

```
rm -rf /root/.kube
```

### Hardening VPC Security Group Rules

CCE is a universal container platform. Its default security group rules apply to common scenarios. Based on security requirements, you can harden the security

group rules set for CCE clusters on the **Security Groups** page of **Network Console**.

For details, see [How Do I Harden the VPC Security Group Rules for CCE Cluster Nodes](#).

## Hardening Nodes on Demand

CCE cluster nodes use the default settings of open source OSs. After a node is created, you need to perform security hardening according to your service requirements.

In CCE, you can perform hardening as follows:

- Use the post-installation script after the node is created. For details, see the description about **Post-installation Script** in **Advanced Settings** when creating a node. This script is user-defined.
- Build custom images in CCE to create worker nodes. For details about the creation process, see [Creating a Custom CCE Node Image](#).

## Forbidding Containers to Obtain Host Machine Metadata

If a single CCE cluster is shared by multiple users to deploy containers, containers cannot access the management address (169.254.169.254) of OpenStack, preventing containers from obtaining metadata of host machines.

For details about how to restore the metadata, see the "Notes" section in [Obtaining Metadata](#).

---

 **WARNING**

This solution may affect the password change on the ECS console. Therefore, you must verify the solution before rectifying the fault.

---

**Step 1** Obtain the network model and container CIDR of the cluster.

On the **Clusters** page of the CCE console, view the network model and container CIDR of the cluster.

Network	
Network Model	VPC network
VPC	vpc-cce
Subnet	
Service Forwarding Mode	iptables
Service Network Segment	10.247.0.0/16
Container Network Segment	10.0.0.0/16
Internal API Server Address	https://192.168.0.107:5443
Public API Server Address	<a href="#">Bind EIP</a>

**Step 2** Prevent the container from obtaining host metadata.

- VPC network
  - a. Log in to each node in the CCE cluster as user **root** and run the following command:

```
iptables -I OUTPUT -s {container_cidr} -d 169.254.169.254 -j REJECT
```

{*container\_cidr*} indicates the container CIDR of the cluster, for example, 10.0.0.0/16.  
To ensure configuration persistence, you are advised to write the command to the **/etc/rc.local** script.
  - b. Run the following commands in the container to access the **userdata** and **metadata** interfaces of OpenStack and check whether the request is intercepted:

```
curl 169.254.169.254/openstack/latest/meta_data.json
curl 169.254.169.254/openstack/latest/user_data
```
- Container tunnel network
  - a. Log in to each node in the CCE cluster as user **root** and run the following command:

```
iptables -I FORWARD -s {container_cidr} -d 169.254.169.254 -j REJECT
```

{*container\_cidr*} indicates the container CIDR of the cluster, for example, 10.0.0.0/16.  
To ensure configuration persistence, you are advised to write the command to the **/etc/rc.local** script.
  - b. Run the following commands in the container to access the **userdata** and **metadata** interfaces of OpenStack and check whether the request is intercepted:

```
curl 169.254.169.254/openstack/latest/meta_data.json
curl 169.254.169.254/openstack/latest/user_data
```

----End

## 6.4 Container Security

### Controlling the Pod Scheduling Scope

The nodeSelector or nodeAffinity is used to limit the range of nodes to which applications can be scheduled, preventing the entire cluster from being threatened due to the exceptions of a single application. For details, see [Node Affinity](#).

### Suggestions on Container Security Configuration

- Set the computing resource limits (**request** and **limit**) of a container. This prevents the container from occupying too many resources and affecting the stability of the host and other containers on the same node.
- Unless necessary, do not mount sensitive host directories to containers, such as **/**, **/boot**, **/dev**, **/etc**, **/lib**, **/proc**, **/sys**, and **/usr**.
- Do not run the sshd process in containers unless necessary.
- Unless necessary, it is not recommended that containers and hosts share the network namespace.
- Unless necessary, it is not recommended that containers and hosts share the process namespace.

- Unless necessary, it is not recommended that containers and hosts share the IPC namespace.
- Unless necessary, it is not recommended that containers and hosts share the UTS namespace.
- Unless necessary, do not mount the sock file of Docker to any container.

## Container Permission Access Control

When using a containerized application, comply with the minimum privilege principle and properly set securityContext of Deployments or StatefulSets.

- Configure runAsUser to specify a non-root user to run a container.
- Configure privileged to prevent containers being used in scenarios where privilege is not required.
- Configure capabilities to accurately control the privileged access permission of containers.
- Configure allowPrivilegeEscalation to disable privilege escape in scenarios where privilege escalation is not required for container processes.
- Configure seccomp to restrict the container syscalls. For details, see [Restrict a Container's Syscalls with seccomp](#) in the official Kubernetes documentation.
- Configure ReadOnlyRootFilesystem to protect the root file system of a container.

Example YAML for a Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: security-context-example
  namespace: security-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: security-context-example
      label: security-context-example
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
      type: RollingUpdate
  template:
    metadata:
      annotations:
        seccomp.security.alpha.kubernetes.io/pod: runtime/default
      labels:
        app: security-context-example
        label: security-context-example
    spec:
      containers:
        - image: ...
          imagePullPolicy: Always
          name: security-context-example
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            runAsUser: 1000
            capabilities:
              add:
                - NET_BIND_SERVICE
              drop:
```

```
- all
volumeMounts:
- mountPath: /etc/localtime
  name: localtime
  readOnly: true
- mountPath: /opt/write-file-dir
  name: tmpfs-example-001
securityContext:
seccompProfile:
  type: RuntimeDefault
volumes:
- hostPath:
  path: /etc/localtime
  type: ""
  name: localtime
- emptyDir: {}
  name: tmpfs-example-001
```

## Restricting the Access of Containers to the Management Plane

If application containers on a node do not need to access Kubernetes, you can perform the following operations to disable containers from accessing kube-apiserver:

**Step 1** Query the container CIDR block and private API server address.

On the **Clusters** page of the CCE console, click the name of the cluster to find the information on the details page.

**Step 2** Log in to each node in the CCE cluster as user **root** and run the following command:

- VPC network:  
iptables -I OUTPUT -s {container\_cidr} -d {Private API server IP} -j REJECT
- Container tunnel network:  
iptables -I FORWARD -s {container\_cidr} -d {Private API server IP} -j REJECT

{container\_cidr} indicates the container network of the cluster, for example, 10.0.0.0/16, and {master\_ip} indicates the IP address of the master node.

To ensure configuration persistence, you are advised to write the command to the **/etc/rc.local** script.

**Step 3** Run the following command in the container to access kube-apiserver and check whether the request is intercepted:

```
curl -k https://{Private API server IP}:5443
```

----End

## 6.5 Secret Security

Currently, CCE has configured static encryption for secret resources. The secrets created by users will be encrypted and stored in etcd of the CCE cluster. Secrets can be used in two modes: environment variable and file mounting. No matter which mode is used, CCE still transfers the configured data to users. Therefore, it is recommended that:

1. Do not record sensitive information in logs.

2. For the secret that uses the file mounting mode, the default file permission mapped in the container is 0644. Configure stricter permissions for the file.

For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: redis
      volumeMounts:
        - name: foo
          mountPath: "/etc/foo"
  volumes:
    - name: foo
      secret:
        secretName: mysecret
        defaultMode: 256
```

In **defaultMode: 256**, 256 is a decimal number, which corresponds to the octal number **0400**.

3. When the file mounting mode is used, configure the secret file name to hide the file in the container.

```
apiVersion: v1
kind: Secret
metadata:
  name: dotfile-secret
data:
  .secret-file: dmFsdWUtMg0KDQo=
---
apiVersion: v1
kind: Pod
metadata:
  name: secret-dotfiles-pod
spec:
  volumes:
    - name: secret-volume
      secret:
        secretName: dotfile-secret
  containers:
    - name: dotfile-test-container
      image: k8s.gcr.io/busybox
      command:
        - ls
        - "-1"
        - "/etc/secret-volume"
      volumeMounts:
        - name: secret-volume
          readOnly: true
          mountPath: "/etc/secret-volume"
```

In this way, **.secret-file** cannot be viewed by running the **ls -l** command in the **/etc/secret-volume/** directory, but can be viewed by running the **ls -al** command.

4. Encrypt sensitive information before creating a secret and decrypt the information when using it.

## Using a Bound ServiceAccount Token to Access a Cluster

The secret-based ServiceAccount token does not support expiration time or auto update. In addition, after the mounting pod is deleted, the token is still stored in the secret. Token leakage may incur security risks. A bound ServiceAccount token is recommended for CCE clusters of version 1.23 or later. In this mode, the

expiration time can be set and is the same as the pod lifecycle, reducing token leakage risks. Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: security-token-example
  namespace: security-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: security-token-example
      label: security-token-example
  template:
    metadata:
      annotations:
        seccomp.security.alpha.kubernetes.io/pod: runtime/default
      labels:
        app: security-token-example
        label: security-token-example
    spec:
      serviceAccountName: test-sa
      containers:
        - image: ...
          imagePullPolicy: Always
          name: security-token-example
      volumes:
        - name: test-projected
          projected:
            defaultMode: 420
          sources:
            - serviceAccountToken:
              expirationSeconds: 1800
              path: token
            - configMap:
              items:
                - key: ca.crt
                  path: ca.crt
                  name: kube-root-ca.crt
            - downwardAPI:
              items:
                - fieldRef:
                  apiVersion: v1
                  fieldPath: metadata.namespace
                  path: namespace
```

For details, visit <https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/>.

## 6.6 Workload Identities

With workload identities, your workloads in a cluster can access cloud services like IAM users without using the AK/SK, reducing security risks.

This section describes how to use workload identities in CCE.

### Notes and Constraints

Your clusters must be version 1.19.16 or later.

## Procedure

1. Obtain the JSON Web Key Set (JWKS) of the cluster (the signature public key of the service account token) from CCE.
2. Create an identity provider on the IAM console.
3. Deploy the application and bind it with the identity provider.
  - a. Use the OIDC token to access IAM and obtain the IAM token (implemented by you).
  - b. Use the IAM token to access cloud services (implemented by you).

## Obtaining JWKS of a CCE Cluster

**Step 1** Use kubectl to connect to the cluster.

**Step 2** Run the following command to obtain the public key:

**kubectl get --raw /openid/v1/jwks**

```
# kubectl get --raw /openid/v1/jwks
>{"keys": [
  {"use":"sig","kty":"RSA","kid":"XWHyMGivzlc3plctTTG7qupYrjZ6YI8rSudsBr0cRIM","alg":"RS256","n":"8uOBh
h4yjDWVnGFxPeuc3NzNIWUbH-
WhrzZlilyh88EKRzLbvEAfFRq5sXviNz1IUSAN5mFJZZwMD6pbho1beyGeYXG0Quq4ZYmwSeu7ATEpSuc2ksQm
Hq7xRzEewKetupA-2oBJaz4LShHpS6bHOnQL5m_OBzd8Eh7t7cEzPX-
ID_dd16qqoRO3iPEISlhq0rm2gSe6mirvibQ7NSUjmSJt4LxLVF-lqXgbcfH1JzCoV7Xi1PRc8viCGYPs05o_Bqqm2-
XXKqAtwmbMg_Z5NCESmKeJyuRqiFYrw2aCHpQaeVeUnOBabfA1d4crWVG0r_00Fat5yDnQmy5GFUGuSQ","e
":"AQAB"}]}
```

The returned field is the public key of the cluster.

----End

## Configuring an Identity Provider

**Step 1** Log in to the IAM console, create an identity provider, and select **OpenID Connect for Protocol**.

Identity Providers / Create Identity Provider

\* Name: workload\_identity

\* Protocol: OpenID Connect

\* SSO Type: Virtual user

\* Status: Enabled

Description: Enter a brief description.

0/255

OK Cancel

**Step 2** Click **OK**. After the creation, modify the identity provider information.

**Access Type:** Select **Programmatic access**.

#### Configuration Information

- Identity Provider URL:** Enter **https://kubernetes.default.svc.cluster.local**.
- Client ID:** Enter an ID, which will be used when you create a container.
- Signing Key:** Enter the JWKS of the CCE cluster obtained in [Obtaining JWKS of a CCE Cluster](#).

Configuration Information ②

Identity Provider URL: https://kubernetes.default.svc.cluster.local

Client ID: workload\_identity

Signing Key:

```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "XWHyMGivzlc3plctTTG7qupYrjZ6Yl8rSudsBr0cRIM",
      "alg": "RS256",
      "n": "BuOBhh4yjDWVnGFxPeuc3NzNIWUbh-WhrzZillyh8EKRzLbvEAfFRq5sXvlnz1IUSAN5mfJZ2wMD6pbho1beyGeYXG0Quq4ZYmwSeu7ATEpSuc2ksQmHq7xRzEewKetupA-2oBJaz4LShP66hOnQL5m_OBzd8Eh7t7cEzPX-ID_dD16qqaRO3IPEISlhq0rm2gSe6mirvibQ7NSUjmSJt4LxLVF-IqXpbcfH1jZco7X1iPrC8vICGYPs05o_Bqqm2-XXKqAtwmibMg_Z5NCESmKeJyuRqlFYrw2aChpQaeVeUnOBabfA1d4crWVG0r_00Fa5yDnQmy5GFUGuSQ"
    }
  ]
}
```

#### Identity Conversion Rules

An identity conversion rule maps the ServiceAccount of a workload to an IAM user group.

For example, create a ServiceAccount named **oidc-token** in namespace **default** of the cluster and map it to user group **demo**. If you use the identity provider ID to

access cloud services, you have the permissions of the **demo** user group. The attribute must be **sub**. The value format is **system:serviceaccount:Namespace:ServiceAccountName**.

The screenshot shows the 'Create Rule' dialog. Under 'Rule Conditions', there is one condition: 'sub' under 'any\_one\_of' with the value 'system:serviceaccount:default:oidc-token'. The 'Operation' column has a 'Delete' link.

Rules are in the JSON format as follows:

```
[  
  {  
    "local": [  
      {  
        "user": {  
          "name": "test"  
        }  
      },  
      {  
        "group": {  
          "name": "demo"  
        }  
      }  
    ],  
    "remote": [  
      {  
        "type": "sub",  
        "any_one_of": [  
          "system:serviceaccount:default:oidc-token"  
        ]  
      }  
    ]  
  }  
]
```

**Step 3** Click OK.

----End

## Using an Workload Identity

Create a ServiceAccount, whose name must be the value of **ServiceAccountName** set in [Configuring an Identity Provider](#).

```
apiVersion: v1  
kind: ServiceAccount  
metadata:  
  name: oidc-token
```

Example configuration for the workload:

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

```
name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
      version: v1
  template:
    metadata:
      labels:
        app: nginx
        version: v1
    spec:
      containers:
        - name: container-1
          image: nginx:latest
          volumeMounts:
            - mountPath: "/var/run/secrets/tokens"    # Mount the serviceAccountToken generated by Kubernetes
              to the /var/run/secrets/tokens/oidc-token file.
              name: oidc-token
          imagePullSecrets:
            - name: default-secret
          serviceAccountName: oidc-token    # Name of the created ServiceAccount
      volumes:
        - name: oidc-token
          projected:
            defaultMode: 420
            sources:
              - serviceAccountToken:
                  audience: workload_identity  # Must be the client ID of the identity provider.
                  expirationSeconds: 7200    # Expiry period
                  path: oidc-token         # Path name, which can be customized.
```

After the creation, log in to the container. The content of the **/var/run/secrets/tokens/oidc-token** file is the serviceAccountToken generated by Kubernetes. You can obtain the IAM token by calling the API for [obtaining a token with an OpenID Connect ID token](#), and then access cloud services.

#### NOTE

If the serviceAccountToken is used for over 24 hours or 80% of the expiry period, kubelet automatically rotates the serviceAccountToken.

Example:

```
curl -k https://{{iam endpoint}}/v3.0/OS-AUTH/id-token/tokens -d "${token_body}" -XPOST -H "X-Idp-Id:workload_identity" -H "Content-Type: application/json" -i -s
```

Where,

- **{{iam endpoint}}** indicates the endpoint of IAM. For details, see [Regions and Endpoints](#).
- **workload\_identity** is the identity provider name, which is the same as that configured in [Configuring an Identity Provider](#).
- **\${token\_body}** is defined as follows:

```
{
  "auth": {
    "id_token": {
      "id": "eyJhbGciOiJSU...” // The value is the content of the /var/run/secrets/tokens/oidc-token
file.
    },
    "scope": {
      "project": {
        "id": "46419baef4324...”, // Project ID
        "name": "cn-north-4" // Project name
      }
    }
}
```

```
    }  
}
```

The **X-Subject-Token** field in the response header is the IAM token.

# 7 Auto Scaling

## 7.1 Using HPA and CA for Auto Scaling of Workloads and Nodes

### Scenario

The best way to handle surging traffic is to automatically adjust the number of machines based on the traffic volume or resource usage, which is called scaling.

In CCE, the resources that can be used by containers are fixed during application deployment. Therefore, in auto scaling, pods are scaled first. The node resource usage increases only after the number of pods increases. Then, nodes can be scaled based on the node resource usage. How to configure auto scaling in CCE?

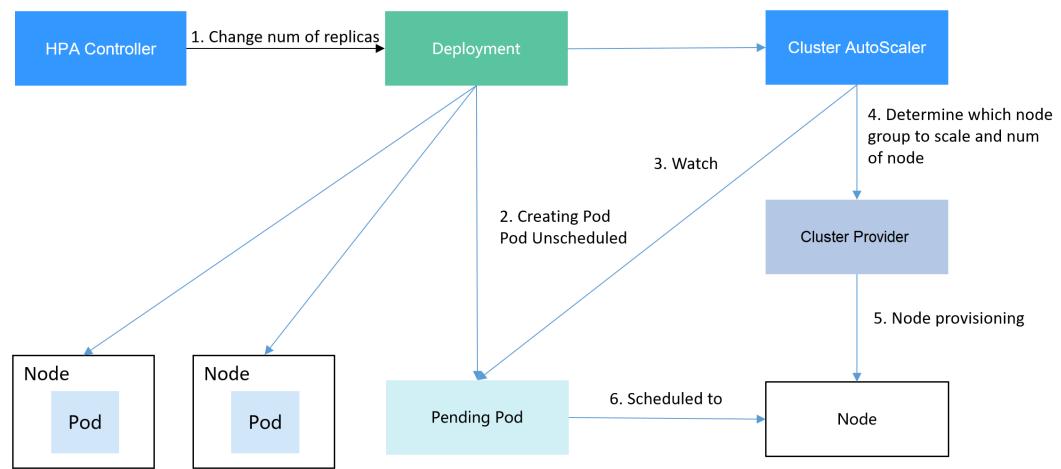
### Solution

Two major auto scaling policies are HPA (Horizontal Pod Autoscaling) and CA (Cluster AutoScaling). HPA is for workload auto scaling and CA is for node auto scaling.

HPA and CA work with each other. HPA requires sufficient cluster resources for successful scaling. When the cluster resources are insufficient, CA is needed to add nodes. If HPA reduces workloads, the cluster will have a large number of idle resources. In this case, CA needs to release nodes to avoid resource waste.

As shown in [Figure 7-1](#), HPA performs scale-out based on the monitoring metrics. When cluster resources are insufficient, newly created pods are in Pending state. CA then checks these pending pods and selects the most appropriate node pool based on the configured scaling policy to scale out the node pool. For details about how HPA and CA work, see [Workload Scaling Mechanisms](#) and [Node Scaling Mechanisms](#).

**Figure 7-1 HPA and CA working flows**

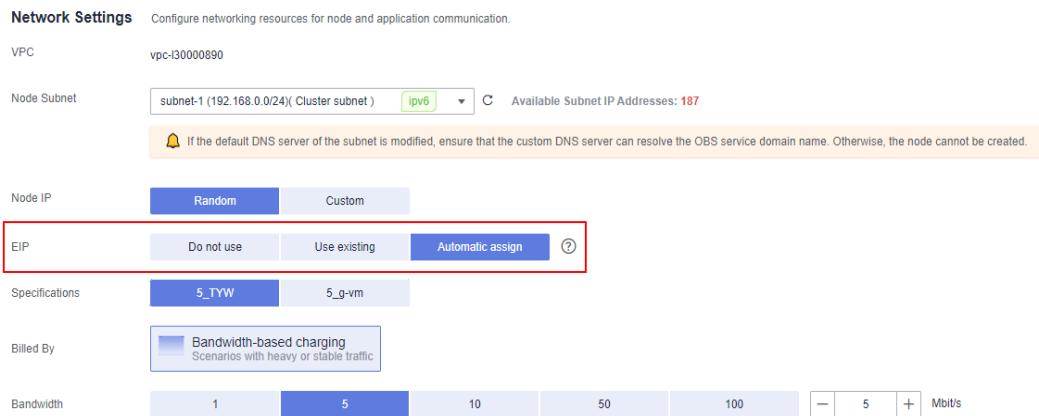


Using HPA and CA can easily implement auto scaling in most scenarios. In addition, the scaling process of nodes and pods can be easily observed.

This section uses an example to describe the auto scaling process using HPA and CA policies together.

## Preparations

- Step 1** Create a cluster with one node. The node should have 2 cores of CPU and 4 GB of memory, or a higher specification, as well as an EIP to allow external access. If no EIP is bound to the node during node creation, you can manually bind one on the ECS console after creating the node.



- Step 2** Install add-ons for the cluster.

- autoscaler: node scaling add-on
- metrics-server: an aggregator of resource usage data in a Kubernetes cluster. It can collect measurement data of major Kubernetes resources, such as pods, nodes, containers, and Services.

- Step 3** Log in to the cluster node and run a computing-intensive application. When a user sends a request, the result needs to be calculated before being returned to the user.

1. Create a PHP file named **index.php** to calculate the square root of the request for 1,000,000 times before returning **OK!**.  
`vi index.php`

Example file content:

```
<?php
$x = 0.0001;
for ($i = 0; $i <= 1000000; $i++) {
    $x += sqrt($x);
}
echo "OK!";
?>
```

2. Compile a Dockerfile to build an image.  
`vi Dockerfile`

Example Dockerfile:

```
FROM php:5-apache
COPY index.php /var/www/html/index.php
RUN chmod a+r index.php
```

3. Run the following command to build an image named **hpa-example** with the tag **latest**.  
`docker build -t hpa-example:latest .`
4. (Optional) Log in to the SWR console, choose **Organization Management** in the navigation pane, and click **Create Organization** in the upper right corner to create an organization.  
Skip this step if you already have an organization.
5. In the navigation pane, choose **My Images** and then click **Upload Through Client**. On the page displayed, click **Generate a temporary login command** and click  to copy the command.
6. Run the login command copied in the previous step on the cluster node. If the login is successful, the message "Login Succeeded" is displayed.
7. Tag the hpa-example image.

**docker tag [Image name 1:Tag 1] [Image repository address]/[Organization name]/[Image name 2:Tag 2]**

- **[Image name 1:Tag 1]**: name and tag of the local image to be uploaded.
- **[Image repository address]**: The domain name at the end of the login command in [5](#) is the image repository address, which can be obtained on the SWR console.
- **[Organization name]**: name of the organization created in [4](#).
- **[Image name 2:Tag 2]**: desired image name and tag to be displayed on the SWR console.

Example:

**docker tag hpa-example:latest swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/hpa-example:latest**

8. Push the image to the image repository.

**docker push [Image repository address]/[Organization name]/[Image name 2:Tag 2]**

Example:

**docker push swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/hpa-example:latest**

The following information will be returned upon a successful push:

```
6d6b9812c8ae: Pushed
...
fe4c16cbf7a4: Pushed
latest: digest: sha256:eb7e3bbd*** size: **
```

To view the pushed image, go to the SWR console and refresh the **My Images** page.

----End

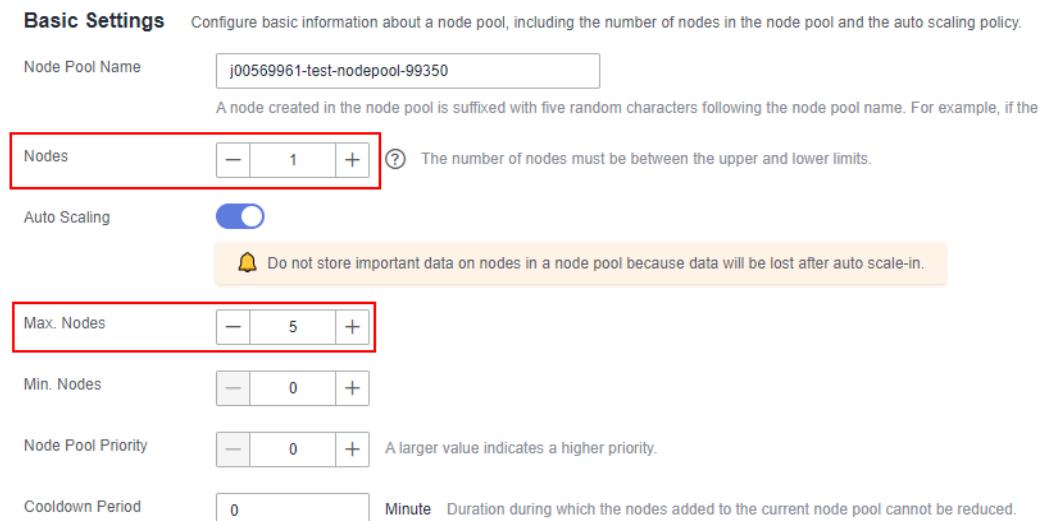
## Creating a Node Pool and a Node Scaling Policy

**Step 1** Log in to the CCE console, access the created cluster, click **Nodes** on the left, click the **Node Pools** tab, and click **Create Node Pool** in the upper right corner.

**Step 2** Set node pool parameters, add a node with 2 vCPUs and 4 GB memory, and enable auto scaling.

- **Nodes:** Set it to **1**, indicating that one node is created by default when a node pool is created.
- **Auto Scaling:** Enable the option, meaning that nodes will be automatically created or deleted in the node pool based on the cluster loads.
- **Max. Nodes:** Set it to **5**, indicating the maximum number of nodes in a node pool.
- **Specifications:** **2 vCPUs | 4 GiB**

Retain the defaults for other parameters. For details, see [Creating a Node Pool](#).



**Step 3** Click **Add-ons** on the left of the cluster console, click **Edit** under the autoscaler add-on, modify the add-on configuration, enable **Auto node scale-in**, and configure scale-in parameters. For example, trigger scale-in when the node resource utilization is less than 50%.

**Parameters**

Scaling

- Nodes are automatically added (from the node pool) when pods in the cluster cannot be scheduled.
- Auto node scale-in

Node Idle Time (min)	10	Minute
How long a node should be unneeded before it is eligible for scale down. The default value is 10 minutes.		
Scale-in Threshold	50	%
When the resource usage of a node is lower than a specified value (percentage), the node is considered idle (both CPUs and memory need to meet the requirements).		
Stabilization Window (s)	10	Minute
How long after a scale-out that a scale-in evaluation resumes.		
	10	Minute
How long after the node deletion that a scale-in evaluation resumes.		
	3	Minute
How long after a scale-in failure that a scale-in evaluation resumes.		
Max. Nodes for Batch Deletion	10	
Maximum number of empty nodes that can be deleted at the same time.		
Check Interval	5	Minute
Interval for checking again a node that could not be removed before.		

After the preceding configurations, scale-out is performed based on the pending status of the pod and scale-in is triggered when the node resource utilization decreases.

**Step 4** Click **Node Scaling** on the left of the cluster console and click **Create Node Scaling Policy** in the upper right corner. Node scaling policies added here trigger scale-out based on the CPU/memory allocation rate or periodically.

As shown in the following figure, when the cluster CPU allocation rate is greater than 70%, one node will be added. A node scaling policy needs to be associated with a node pool. Multiple node pools can be associated. When you need to scale nodes, node with proper specifications will be added or reduced from the node pool based on the minimum waste principle. For details, see [Creating a Node Scaling Policy](#).

**Edit Node Scaling Policy**

Policy Name	policy											
Associated Node Pool	<input style="width: 80%; height: 25px; border: 1px solid #ccc; margin-bottom: 5px;" type="text"/> <span style="font-size: small;">Enter a name. <input style="width: 20px; height: 20px; border: 1px solid #ccc; border-radius: 5px; vertical-align: middle;" type="button" value="Search"/></span> <span style="font-size: small; margin-left: 10px;"><input style="width: 20px; height: 20px; border: 1px solid #ccc; border-radius: 5px; vertical-align: middle;" type="button" value="C"/></span>											
	<input checked="" type="checkbox"/> Node Pool Name      Status      Actual/Desired N...      Specifications      AZ      Billing M...      Auto Scal...											
	<input checked="" type="checkbox"/> example-nodepool-25343 <span style="color: green;">Normal</span> 0/0      ECS   c7.large.2      Random <span style="background-color: orange; border: 1px solid black; padding: 2px;">Pay-per-...</span> <input type="checkbox"/> Enable											
Rules	<input style="width: 100px; height: 25px; border: 1px solid #ccc; border-radius: 5px; margin-bottom: 5px;" type="button" value="Add Rule"/> <p style="margin: 0;">You can add a maximum of 10 rules. Only one CPU-based rule and one memory-based rules can be added.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Rule Name</th> <th style="width: 15%;">Rule Type</th> <th style="width: 30%;">Trigger</th> <th style="width: 15%;">Action</th> <th style="width: 25%;">Operation</th> </tr> </thead> <tbody> <tr> <td>addnode</td> <td>Metric-based</td> <td>CPU allocation rate &gt; 70%</td> <td>Add1 nodes</td> <td><input style="width: 50px; height: 25px; border: 1px solid #ccc; border-radius: 5px;" type="button" value="Delete"/> <input style="width: 50px; height: 25px; border: 1px solid #ccc; border-radius: 5px;" type="button" value="Edit"/></td> </tr> </tbody> </table>		Rule Name	Rule Type	Trigger	Action	Operation	addnode	Metric-based	CPU allocation rate > 70%	Add1 nodes	<input style="width: 50px; height: 25px; border: 1px solid #ccc; border-radius: 5px;" type="button" value="Delete"/> <input style="width: 50px; height: 25px; border: 1px solid #ccc; border-radius: 5px;" type="button" value="Edit"/>
Rule Name	Rule Type	Trigger	Action	Operation								
addnode	Metric-based	CPU allocation rate > 70%	Add1 nodes	<input style="width: 50px; height: 25px; border: 1px solid #ccc; border-radius: 5px;" type="button" value="Delete"/> <input style="width: 50px; height: 25px; border: 1px solid #ccc; border-radius: 5px;" type="button" value="Edit"/>								

----End

## Creating a Workload

Use the hpa-example image to create a Deployment with one replica. The image path is related to the organization uploaded to the SWR repository and needs to be replaced with the actual value.

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: hpa-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hpa-example
  template:
    metadata:
      labels:
        app: hpa-example
    spec:
      containers:
        - name: container-1
          image: 'hpa-example:latest' # Replace it with the address of the image you uploaded to SWR.
          resources:
            limits:           # The value of limits must be the same as that of requests to prevent flapping
            during scaling.
              cpu: 500m
              memory: 200Mi
            requests:
              cpu: 500m
              memory: 200Mi
      imagePullSecrets:
        - name: default-secret
```

Then, create a NodePort Service for the workload so that the workload can be accessed from external networks.

### NOTE

To allow external access to NodePort Services, you need to create an EIP for the node in the cluster. After the creation, you need to synchronize node data. For details, see [Synchronizing Node Data](#). If the node already has an EIP, you do not need to create one.

Alternatively, you can create a Service with an ELB load balancer for external access. For details, see [Using kubectl to Create a Service \(Automatically Creating a Shared Load Balancer\)](#).

```
kind: Service
apiVersion: v1
metadata:
  name: hpa-example
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 31144
  selector:
    app: hpa-example
  type: NodePort
```

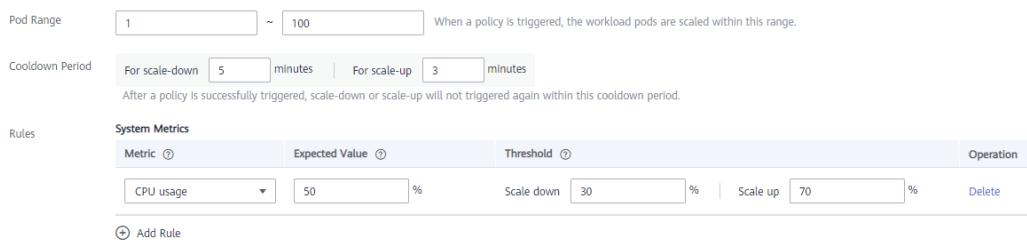
## Creating an HPA Policy

Create an HPA policy. As shown below, the policy is associated with the hpa-example workload, and the target CPU usage is 50%.

There are two other annotations. One annotation defines the CPU thresholds, indicating that scaling is not performed when the CPU usage is between 30% and 70% to prevent impact caused by slight fluctuation. The other is the scaling time window, indicating that after the policy is successfully executed, a scaling operation will not be triggered again in this cooling interval to prevent impact caused by short-term fluctuation.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-policy
  annotations:
    extendedhpa.metrics: '[{"type":"Resource","name":"cpu","targetType":"Utilization","targetRange":{"low":"30","high":"70"}}]'
    extendedhpa.option: '{"downscaleWindow":"5m","upscaleWindow":"3m"}'
spec:
  scaleTargetRef:
    kind: Deployment
    name: hpa-example
    apiVersion: apps/v1
  minReplicas: 1
  maxReplicas: 100
  metrics:
  - type: Resource
    resource:
      name: cpu
    targetAverageUtilization: 50
```

Set the parameters as follows if you are using the console.



## Observing the Auto Scaling Process

**Step 1** Check the cluster node status. In the following example, there are two nodes.

```
# kubectl get node
NAME     STATUS   ROLES   AGE     VERSION
192.168.0.183 Ready    <none>  2m20s  v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
192.168.0.26 Ready    <none>  55m    v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
```

Check the HPA policy. The CPU usage of the target workload is 0%.

```
# kubectl get hpa hpa-policy
NAME      REFERENCE          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-policy Deployment/hpa-example  0%/50%    1         100        1          4m
```

**Step 2** Run the following command to access the workload. In the following command, {ip:port} indicates the access address of the workload, which can be queried on the workload details page.

```
while true;do wget -q -O- http://{ip:port};done
```

 NOTE

If no EIP is displayed, the cluster node has not been assigned any EIP. You need to create one, bind it to the node, and synchronize node data. For details, see [Synchronizing Node Data](#).

Observe the scaling process of the workload.

```
# kubectl get hpa hpa-policy --watch
NAME      REFERENCE      TARGETS      MINPODS      MAXPODS      REPLICAS      AGE
hpa-policy Deployment/hpa-example 0%/50%      1           100          1           4m
hpa-policy Deployment/hpa-example 190%/50%     1           100          1           4m23s
hpa-policy Deployment/hpa-example 190%/50%     1           100          4           4m31s
hpa-policy Deployment/hpa-example 200%/50%     1           100          4           5m16s
hpa-policy Deployment/hpa-example 200%/50%     1           100          4           6m16s
hpa-policy Deployment/hpa-example 85%/50%      1           100          4           7m16s
hpa-policy Deployment/hpa-example 81%/50%      1           100          4           8m16s
hpa-policy Deployment/hpa-example 81%/50%      1           100          7           8m31s
hpa-policy Deployment/hpa-example 57%/50%      1           100          7           9m16s
hpa-policy Deployment/hpa-example 51%/50%      1           100          7           10m
hpa-policy Deployment/hpa-example 58%/50%      1           100          7           11m
```

You can see that the CPU usage of the workload is 190% at 4m23s, which exceeds the target value. In this case, scaling is triggered to expand the workload to four replicas/pods. In the subsequent several minutes, the CPU usage does not decrease until 7m16s. This is because the new pods may not be successfully created. The possible cause is that resources are insufficient and the pods are in Pending state. During this period, nodes are added.

At 7m16s, the CPU usage decreases, indicating that the pods are successfully created and start to bear traffic. The CPU usage decreases to 81% at 8m, still greater than the target value (50%) and the high threshold (70%). Therefore, 7 pods are added at 9m16s, and the CPU usage decreases to 51%, which is within the range of 30% to 70%. From then on, the number of pods remains 7.

In the following output, you can see the workload scaling process and the time when the HPA policy takes effect.

```
# kubectl describe deploy hpa-example
...
Events:
  Type  Reason        Age   From            Message
  ----  ----        --   --              --
  Normal  ScalingReplicaSet  25m  deployment-controller  Scaled up replica set hpa-example-79dd795485
  to 1
  Normal  ScalingReplicaSet  20m  deployment-controller  Scaled up replica set hpa-example-79dd795485
  to 4
  Normal  ScalingReplicaSet  16m  deployment-controller  Scaled up replica set hpa-example-79dd795485
  to 7
# kubectl describe hpa hpa-policy
...
Events:
  Type  Reason        Age   From            Message
  ----  ----        --   --              --
  Normal  SuccessfulRescale  20m  horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilization
  (percentage of request) above target
  Normal  SuccessfulRescale  16m  horizontal-pod-autoscaler  New size: 7; reason: cpu resource utilization
  (percentage of request) above target
```

Check the number of nodes. The following output shows that two nodes are added.

```
# kubectl get node
NAME      STATUS  ROLES   AGE    VERSION
192.168.0.120  Ready  <none>  3m5s  v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
```

```
192.168.0.136 Ready <none> 6m58s v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
192.168.0.183 Ready <none> 18m v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
192.168.0.26 Ready <none> 71m v1.17.9-r0-CCE21.1.1.3.B001-17.36.8
```

You can also view the scaling history on the console. For example, the CA policy is executed once when the CPU allocation rate in the cluster is greater than 70%, and the number of nodes in the node pool is increased from 2 to 3. The new node is automatically added by autoscaler based on the pending state of pods in the initial phase of HPA.

The node scaling process is as follows:

1. After the number of pods changes to 4, the pods are in Pending state due to insufficient resources. As a result, the default scale-out policy of the autoscaler add-on is triggered, and the number of nodes is increased by one.
2. The second node scale-out is triggered because the CPU allocation rate in the cluster is greater than 70%. As a result, the number of nodes is increased by one, which is recorded in the scaling history on the console. Scaling based on the allocation rate ensures that the cluster has sufficient resources.

### Step 3 Stop accessing the workload and check the number of pods.

```
# kubectl get hpa hpa-policy --watch
NAME      REFERENCE      TARGETS      MINPODS      MAXPODS      REPLICAS      AGE
hpa-policy Deployment/hpa-example 50%/50%    1           100          7           12m
hpa-policy Deployment/hpa-example 21%/50%    1           100          7           13m
hpa-policy Deployment/hpa-example 0%/50%     1           100          7           14m
hpa-policy Deployment/hpa-example 0%/50%     1           100          7           18m
hpa-policy Deployment/hpa-example 0%/50%     1           100          3           18m
hpa-policy Deployment/hpa-example 0%/50%     1           100          3           19m
hpa-policy Deployment/hpa-example 0%/50%     1           100          3           23m
hpa-policy Deployment/hpa-example 0%/50%     1           100          3           23m
hpa-policy Deployment/hpa-example 0%/50%     1           100          1           23m
```

You can see that the CPU usage is 21% at 13m. The number of pods is reduced to 3 at 18m, and then reduced to 1 at 23m.

In the following output, you can see the workload scaling process and the time when the HPA policy takes effect.

```
# kubectl describe deploy hpa-example
...
Events:
  Type  Reason      Age   From            Message
  ----  ----      --   --              --
  Normal  ScalingReplicaSet  25m  deployment-controller  Scaled up replica set hpa-example-79dd795485 to 1
  Normal  ScalingReplicaSet  20m  deployment-controller  Scaled up replica set hpa-example-79dd795485 to 4
  Normal  ScalingReplicaSet  16m  deployment-controller  Scaled up replica set hpa-example-79dd795485 to 7
  Normal  ScalingReplicaSet  6m28s deployment-controller  Scaled down replica set hpa-example-79dd795485 to 3
  Normal  ScalingReplicaSet  72s  deployment-controller  Scaled down replica set hpa-example-79dd795485 to 1
# kubectl describe hpa hpa-policy
...
Events:
  Type  Reason      Age   From            Message
  ----  ----      --   --              --
  Normal  SuccessfulRescale  20m  horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  16m  horizontal-pod-autoscaler  New size: 7; reason: cpu resource utilization
```

(percentage of request) above target

```
Normal SuccessfulRescale 6m45s horizontal-pod-autoscaler New size: 3; reason: All metrics below target
Normal SuccessfulRescale 90s horizontal-pod-autoscaler New size: 1; reason: All metrics below target
```

You can also view the HPA policy execution history on the console. Wait until the one node is reduced.

The reason why the other two nodes in the node pool are not reduced is that they both have pods in the kube-system namespace (and these pods are not created by DaemonSets). For details about node scale-in, see [How autoscaler Works](#).

----End

## Summary

Using HPA and CA can easily implement auto scaling in most scenarios. In addition, the scaling process of nodes and pods can be easily observed.

## 7.2 Auto Scaling Based on ELB Monitoring Metrics

### Issues

In [Using HPA and CA for Auto Scaling of Workloads and Nodes](#), auto scaling is performed based on the usage of resources such as CPU and memory.

However, resource usage usually lags. Such scaling cannot perfectly support services such as flash sales and social media that require quick and elastic scaling.

### Solution

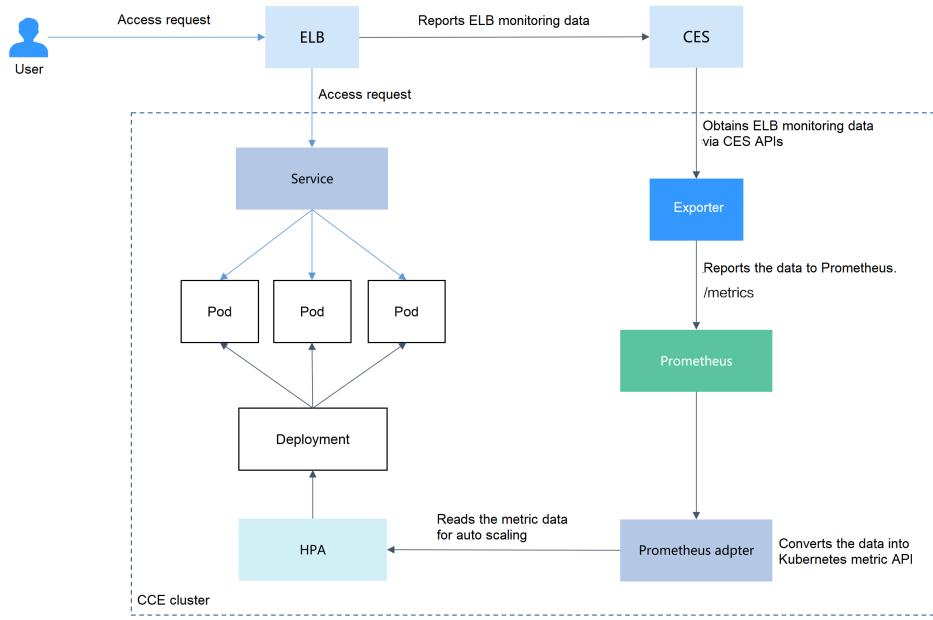
This section describes an auto scaling solution based on ELB monitoring metrics. Compared with CPU/memory usage-based auto scaling, auto scaling based on ELB QPS data is more targeted and timely.

The key of this solution is to obtain the ELB metric data and report the data to Prometheus, convert the data in Prometheus to the metric data that can be identified by HPA, and then perform auto scaling based on the converted data.

The implementation scheme is as follows:

1. Develop a Prometheus exporter to obtain ELB metric data, convert the data into the format required by Prometheus, and report it to Prometheus. This section uses [cloudeye-exporter](#) as an example.
2. Convert the Prometheus data into the Kubernetes metric API for the HPA controller to use.
3. Set an HPA rule to use ELB monitoring data as auto scaling metrics.

**Figure 7-2 ELB traffic flows and monitoring data**



#### NOTE

Other metrics can be collected in the similar way.

## Prerequisites

- You must be familiar with Prometheus and be able to write the Prometheus exporter.
- The prometheus add-on must be installed in the cluster.

## Building an Exporter Image

This section uses [cloudeye-exporter](#) to monitor load balancer metrics. If you need to develop an exporter, see [Appendix: Developing an Exporter](#).

- Step 1** Log in to a cluster node that can access the public network and compile a Dockerfile.

```
vi Dockerfile
```

Example Dockerfile:

```
FROM ubuntu:18.04
RUN apt-get update \
&& apt-get install -y git ca-certificates curl \
&& update-ca-certificates \
&& curl -O https://dl.google.com/go/go1.14.14.linux-amd64.tar.gz \
&& tar -zxf go1.14.14.linux-amd64.tar.gz -C /usr/local \
&& git clone https://github.com/huaweicloud/cloudeye-exporter \
&& export PATH=$PATH:/usr/local/go/bin \
&& export GO111MODULE=on \
&& export GOPROXY=https://goproxy.cn,direct \
&& export GONOSUMDB=* \
&& cd cloudeye-exporter \
&& go build
CMD ["/cloudeye-exporter/cloudeye-exporter -config=/tmp/clouds.yml"]
```

- Step 2** Build an image. The image name is **cloudeye-exporter** and the image version is 1.0.

```
docker build --network host . -t cloudeye-exporter:1.0
```

**Step 3** Push the image to the SWR.

1. (Optional) Log in to the SWR console, choose **Organization Management** in the navigation pane, and click **Create Organization** in the upper right corner to create an organization.  
Skip this step if you already have an organization.
2. In the navigation pane, choose **My Images** and then click **Upload Through Client**. On the page displayed, click **Generate a temporary login command** and click  to copy the command.
3. Run the login command copied in the previous step on the cluster node. If the login is successful, the message "Login Succeeded" is displayed.
4. Tag the **cloudeye-exporter** image.

**docker tag [Image name 1:Tag 1] [Image repository address]/[Organization name]/[Image name 2:Tag 2]**

- **[Image name 1:Tag 1]**: name and tag of the local image to be uploaded.
- **[Image repository address]**: The domain name at the end of the login command in 2 is the image repository address, which can be obtained on the SWR console.
- **[Organization name]**: name of the organization created in 1.
- **[Image name 2:Tag 2]**: desired image name and tag to be displayed on the SWR console.

Example:

**docker tag cloudeye-exporter:1.0 swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/cloudeye-exporter:1.0**

5. Push the image to the image repository.

**docker push [Image repository address]/[Organization name]/[Image name 2:Tag 2]**

Example:

**docker push swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/cloudeye-exporter:1.0**

The following information will be returned upon a successful push:

```
...  
030***: Pushed  
1.0: digest: sha256:eb7e3bbd*** size: **
```

To view the pushed image, go to the SWR console and refresh the **My Images** page.

----End

## Deploying the Exporter

Prometheus can dynamically monitor pods if you add Prometheus annotations to the pods (the default path is **/metrics**). This section uses **cloudeye-exporter** as an example.

Common annotations in Prometheus are as follows:

- **prometheus.io/scrape**: If the value is **true**, the pod will be monitored.
- **prometheus.io/path**: URL from which the data is collected. The default value is **/metrics**.
- **prometheus.io/port**: port number of the endpoint to collect data from.
- **prometheus.io/scheme**: Defaults to **http**. If HTTPS is configured for security purposes, change the value to **https**.

**Step 1** Use kubectl to connect to the cluster.

**Step 2** Create a secret, which will be used by **cloudeye-exporter** for authentication.

1. Create the **clouds.yml** file with the following content:

```
global:  
  prefix: "huaweicloud"  
  scrape_batch_size: 10  
  port: ":8087"  
  metric_path: "/metrics"  
auth:  
  auth_url: "https://iam.ap-southeast-1.myhuaweicloud.com/v3"  
  project_name: "ap-southeast-1"  
  access_key: "*****"  
  secret_key: "*****"  
  region: "ap-southeast-1"
```

The values of **access\_key** and **secret\_key** can be obtained from [Access Keys](#).

2. Obtain the Base64-encrypted string of the preceding file.

```
cat clouds.yml | base64 -w0 ;echo
```

3. Create the **clouds-secret.yaml** file with the following content:

```
apiVersion: v1  
kind: Secret  
data:  
  clouds.yml: ICAGa***** # Replace it with the Base64 encrypted string.  
metadata:  
  annotations:  
    description: "  
  name: 'clouds.yml'  
  namespace: default # Namespace where the key is located.  
  labels: {}  
type: Opaque
```

4. Create a secret.

```
kubectl apply -f clouds-secret.yaml
```

**Step 3** Create the **cloudeye-exporter-deployment.yaml** file with the following content:

```
kind: Deployment  
apiVersion: apps/v1  
metadata:  
  name: cloudeye-exporter  
  namespace: default  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: cloudeye-exporter  
      version: v1  
  template:  
    metadata:  
      labels:  
        app: cloudeye-exporter  
        version: v1  
    spec:  
      volumes:  
        - name: vol-166055064743016314  
      secret:  
        secretName: clouds.yml
```

```
defaultMode: 420
containers:
- name: container-1
  image: swr.ap-southeast-1.myhuaweicloud.com/cloud-develop/cloudeye-exporter:1.0
  command:
    - /cloudeye-exporter/cloudeye-exporter
    - '-config=/tmp/clouds.yml'
  resources: {}
  volumeMounts:
    - name: vol-166055064743016314
      readOnly: true
      mountPath: /tmp
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  imagePullPolicy: IfNotPresent
  restartPolicy: Always
  terminationGracePeriodSeconds: 30
  dnsPolicy: ClusterFirst
  securityContext: {}
  imagePullSecrets:
    - name: default-secret
  schedulerName: default-scheduler
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 25%
    maxSurge: 25%
  revisionHistoryLimit: 10
  progressDeadlineSeconds: 600
```

Create the preceding workload.

```
kubectl apply -f cloudeye-exporter-deployment.yaml
```

#### Step 4 Create the **cloudeye-exporter-service.yaml** file.

```
apiVersion: v1
kind: Service
metadata:
  name: cloudeye-exporter
  namespace: default
  labels:
    app: cloudeye-exporter
    version: v1
  annotations:
    prometheus.io/port: '8087'
    prometheus.io/scrape: 'true'
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 8087
      targetPort: 8087
  selector:
    app: cloudeye-exporter
    version: v1
  type: ClusterIP
```

Create the preceding Service.

```
kubectl apply -f cloudeye-exporter-service.yaml
```

----End

## Interconnecting with Prometheus

After collecting monitoring data, Prometheus needs to convert the data into the Kubernetes metric API for the HPA controller to perform auto scaling.

In this example, the ELB metrics associated with the workload need to be monitored. Therefore, the target workload must use the Service or ingress of the **LoadBalancer** type.

**Step 1** View the access mode of the workload to be monitored and obtain the ELB listener ID.

1. On the CCE cluster console, choose **Networking**. On the **Services** or **Ingresses** tab page, view the Service or ingress of the **LoadBalancer** type and click the load balancer to access the Load Balancer page.

Service	Selector	Namespace	Service Type
nginx-lb	app nginx version v1	default	LoadBalancer cce-lb-84855111-0680-4f23-9bf

2. On the **Listeners** tab, view the listener corresponding to the workload and copy the listener ID.

**Step 2** Use kubectl to connect to the cluster and add the configmap configuration of Prometheus. In this example, load balancer metrics are collected. For details about advanced usage, see [Configuration](#).

```
kubectl edit configmap prometheus -nmonitoring
```

Add the following content to the **scrape\_configs** field and save the file:

```
data:  
prometheus.yml: |-  
...  
scrape_configs:  
- job_name: 'huaweicloud'  
  params:  
    services: ['SYS.ELB']  
  kubernetes_sd_configs:  
  - role: endpoints  
  relabel_configs:  
  - action: keep  
    regex: '8087'  
    source_labels:  
    - __meta_kubernetes_service_annotation_prometheus_io_port  
  - action: replace  
    regex: ([^:]+)(?::(\d+))?(:(\d+))?  
    replacement: $1:$2  
    source_labels:  
    - __address__  
    - __meta_kubernetes_service_annotation_prometheus_io_port
```

```

target_label: __address__
- action: labelmap
  regex: __meta_kubernetes_service_label_(.+)
- action: replace
  source_labels:
    - __meta_kubernetes_namespace
  target_label: kubernetes_namespace
- action: replace
  source_labels:
    - __meta_kubernetes_service_name
  target_label: kubernetes_service
...

```

### Step 3 Redeploy the Prometheus StatefulSet in the **monitoring** namespace.

The screenshot shows the CCE UI with the navigation bar: Cluster, CCE, Namespace: monitoring, / StatefulSets. Below it is a table listing StatefulSets. One entry for 'prometheus' is shown with status 'Running'. On the right side of the table, there are several buttons: Monitor, View Log, Upgrade, More, Edit YAML, Redeploy (which is highlighted with a red box), Manage Label, and Delete.

### Step 4 Add the configmap configuration of **custom-metrics-apiserver** to **adapter-config**.

```
kubectl edit configmap adapter-config -n monitoring
```

Add the following content under the **rules** field and save the file. Replace the listener ID obtained in **Step 1** with the value of **seriesQuery**.

```

apiVersion: v1
data:
  config.yaml: |-  

    rules:  

    - metricsQuery: sum(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)  

      resources:  

        overrides:  

          kubernetes_namespace:  

            resource: namespace  

          kubernetes_service:  

            resource: service  

        name:  

          matches: huaweicloud_sys_elb_(*)  

          as: "elb01_${1}"  

        seriesQuery: '{lbaas_listener_id="94424*****"}' # ELB listener ID
...

```

### Step 5 Redeploy the **custom-metrics-apiserver** workload in the **monitoring** namespace.

Workload Name	Status	Pods (Normal/All)	Namespace	Created	Image Name	Operation
cluster-problem-detector	Running	1 / 1	monitoring	59 minutes ago	cluster-problem-detector...	Monitor   View Log   Upgrade   More ▾
custom-metrics-episerver	Running	1 / 1	monitoring	59 minutes ago	k8s-prometheus-adapter...	Monitor   View Log   Upgrade   More ▾
event-exporter	Running	1 / 1	monitoring	59 minutes ago	kube-event-exporter:3.6.1	Monitor   View   Edit YAML   Roll Back   Redeploy   Disable Upgrade   Manage Label   Delete
kube-state-metrics	Running	1 / 1	monitoring	59 minutes ago	kube-state-metrics:3.6.1	Monitor   View   Edit YAML   Roll Back   Redeploy   Disable Upgrade   Manage Label   Delete
prometheus-operator	Running	1 / 1	monitoring	59 minutes ago	prometheus-operator:3.6.1	Monitor   View   Edit YAML   Roll Back   Redeploy   Disable Upgrade   Manage Label   Delete

----End

## Creating an HPA Policy

After the data reported by the exporter to Prometheus is converted into the Kubernetes metric API by using the Prometheus adapter, you can create an HPA policy for auto scaling.

The following is an example HPA policy. The inbound traffic of the ELB load balancer is used to trigger scale-out. When the value of **m7\_in\_Bps** (inbound traffic rate) exceeds 1,000, the nginx Deployment will be scaled.

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Object
      object:
        target:
          kind: Service
          name: cloudeye-exporter
        metricName: elb01_listener_m7_in_Bps
        targetValue: 1k
```

After the workload is created, you can perform a pressure test on the workload (accessing the pods through ELB). Then, the HPA controller determines whether scaling is required based on the configured value.

Policy Name	Latest Status	Pod Range	Cooldown Period	Rule	Associated Worklo...	Namespace	Created	Operation
nginx	Lower limit reached	1 - 10	Scale-in: 0Minute   Scale-out: 0Mi...	1	nginx	default	Oct 21, 2022 15:58:37 GMT+08:00	View Events   Edit YAML   More ▾

Metric	Metric Source	Desired Value	Threshold
eb01_listener_m7_in_Bps	cloudeye-exporter (Service)	Value: 1k	Scale-in: --   Scale-out: --

## Appendix: Developing an Exporter

Prometheus periodically calls the [/metrics](#) API of the exporter to obtain metric data. Applications only need to report monitoring data through [/metrics](#). You can select a Prometheus client in a desired language and integrate it into applications to implement the [/metrics](#) API. For details about the client, see [Prometheus CLIENT LIBRARIES](#). For details about how to write the exporter, see [WRITING EXPORTERS](#).

The monitoring data must be in the format that Prometheus supports. Each data record provides the ELB ID, listener ID, namespace where the Service is located, Service name, and Service UID as labels, as shown in the following figure.

```
# HELP m1_cps Number of concurrent connections.  
# TYPE m1_cps gauge  
m1_cps{lb_instance_id="eab0f0fd-9997-468c-8ce2-bdaee416dc5",lb_listener_id="929747a9-ba55-472b-a1e4-8b8d6e076054",namespace="default",service_name="nginx",uid="3b74f807-addf-11e9-bccb-fa163ea2c926"} 0  
# HELP m5_in_pps the packets count that are currently flowing into.  
# TYPE m5_in_pps gauge  
m5_in_pps{lb_instance_id="eab0f0fd-9997-468c-8ce2-bdaee416dc5",lb_listener_id="929747a9-ba55-472b-a1e4-8b8d6e076054",namespace="default",service_name="nginx",uid="3b74f807-addf-11e9-bccb-fa163ea2c926"} 0  
# HELP m6_out_pps the packets count that are currently flowing out.  
# TYPE m6_out_pps gauge  
m6_out_pps{lb_instance_id="eab0f0fd-9997-468c-8ce2-bdaee416dc5",lb_listener_id="929747a9-ba55-472b-a1e4-8b8d6e076054",namespace="default",service_name="nginx",uid="3b74f807-addf-11e9-bccb-fa163ea2c926"} 0  
# HELP m7_in_Bps network traffic flowing into the measurement object per second.  
# TYPE m7_in_Bps gauge  
m7_in_Bps{lb_instance_id="eab0f0fd-9997-468c-8ce2-bdaee416dc5",lb_listener_id="929747a9-ba55-472b-a1e4-8b8d6e076054",namespace="default",service_name="nginx",uid="3b74f807-addf-11e9-bccb-fa163ea2c926"} 0  
# HELP m8_out_Bps network traffic flowing out of the measurement object per second.  
# TYPE m8_out_Bps gauge  
m8_out_Bps{lb_instance_id="eab0f0fd-9997-468c-8ce2-bdaee416dc5",lb_listener_id="929747a9-ba55-472b-a1e4-8b8d6e076054",namespace="default",service_name="nginx",uid="3b74f807-addf-11e9-bccb-fa163ea2c926"} 0
```

To obtain the preceding data, perform the following steps:

**Step 1** Query all Services.

The **annotations** field in the returned information contains the ELB associated with the Service.

- kubernetes.io/elb.id
- kubernetes.io/elb.class

**Step 2** Use the [listener query API](#) to query the listener ID based on the ELB instance ID obtained in the previous step.

**Step 3** Obtain the ELB monitoring data.

The ELB monitoring data is queried using the CES API [used to query monitoring data in batches](#). For details about ELB monitoring metrics, see [Monitoring Metrics](#). Example:

- **m1\_cps**: number of concurrent connections
- **m5\_in\_pps**: number of incoming data packets
- **m6\_out\_pps**: number of outgoing data packets
- **m7\_in\_Bps**: incoming rate
- **m8\_out\_Bps**: outgoing rate

**Step 4** Aggregate data in the format that Prometheus supports and expose the data through the [/metrics](#) API.

The Prometheus client can easily call the [/metrics](#) API. For details, see [CLIENT LIBRARIES](#). For details about how to develop an exporter, see [WRITING EXPORTERS](#).

----End

# 8 Cluster

## 8.1 Creating an IPv4/IPv6 Dual-Stack Cluster in CCE

This section describes how to set up a VPC with IPv6 CIDR block and create a cluster and nodes with an IPv6 address in the VPC, so that the nodes can access the Internet.

### Overview

IPv6 addresses are used to deal with the problem of IPv4 address exhaustion. If a worker node (such as an ECS) in the current cluster uses IPv4, the node can run in dual-stack mode after IPv6 is enabled. Specifically, the node has both IPv4 and IPv6 addresses, which can be used to access the intranet or public network.

### Application Scenarios

- If your application needs to provide Services for users who use IPv6 clients, you can use IPv6 EIPs or the IPv4 and IPv6 dual-stack function.
- If your application needs to both provide Services for users who use IPv6 clients and analyze the access request data, you can use only the IPv4 and IPv6 dual-stack function.
- If internal communication is required between your application systems or between your application system and another system (such as the database system), you can use only the IPv4 and IPv6 dual-stack function.

For details about the dual stack, see [IPv4 and IPv6 Dual-Stack Network](#).

### Notes and Constraints

- The IPv4/IPv6 dual stack is supported only for CCE clusters of v1.15 and later.
- Worker nodes and master nodes in Kubernetes clusters use IPv4 addresses to communicate with each other.
- If the Service type is set to **LoadBalancer (ELB)** or **LoadBalancer (DNAT)**, only IPv4 addresses are supported.
- Ingresses support only IPv4.

- Only one IPv6 address can be bound to each NIC.
- When IPv4/IPv6 dual stack is enabled for the cluster, DHCP unlimited lease cannot be enabled for the selected node subnet.

## Step 1: Create a VPC

Before creating your VPCs, determine how many VPCs, the number of subnets, and what IP address ranges you will need. For details, see [Network Planning](#).

### NOTE

- The basic operations for IPv4 and IPv6 dual-stack networks are the same as those for IPv4 networks. Only some parameters are different.
- For details about the IPv6 billing policy, supported ECS types, and supported regions, see [IPv4 and IPv6 Dual-Stack Network](#).

Perform the following operations to create a VPC named **vpc-ipv6** and its default subnet named **subnet-ipv6**.

1. Log in to the management console.
2. Click  in the upper left corner of the management console and select a region and a project.
3. Under **Networking**, click **Virtual Private Cloud**.
4. Click **Create VPC**.
5. Set the VPC and subnet parameters.

When configuring a subnet, select **Enable** for **IPv6 CIDR Block** to automatically allocate an IPv6 CIDR block to the subnet. IPv6 cannot be disabled after the subnet is created. Currently, you are not allowed to specify a custom IPv6 CIDR block.

**Table 8-1** VPC configuration parameters

Parameter	Description	Example Value
Region	Specifies the desired region. Regions are geographic areas that are physically isolated from each other. The networks inside different regions are not connected to each other, so resources cannot be shared across different regions. For lower network latency and faster access to your resources, select the region nearest you.	AP-Singapore
Name	VPC name.	vpc-ipv6

Parameter	Description	Example Value
IPv4 CIDR Block	<p>Specifies the Classless Inter-Domain Routing (CIDR) block of the VPC. The CIDR block of a subnet can be the same as the CIDR block for the VPC (for a single subnet in the VPC) or a subset (for multiple subnets in the VPC).</p> <p>The following CIDR blocks are supported:</p> <p>10.0.0.0/8–24 172.16.0.0/12–24 192.168.0.0/16–24</p>	192.168.0.0/16
Enterprise Project	<p>When creating a VPC, you can add the VPC to an enabled enterprise project.</p> <p>An enterprise project facilitates project-level management and grouping of cloud resources and users. The name of the default project is <b>default</b>.</p> <p>For details about creating and managing enterprise projects, see <a href="#">Enterprise Management User Guide</a>.</p>	default
Tag (Advanced Settings)	<p>Specifies the VPC tag, which consists of a key and value pair. You can add a maximum of ten tags for each VPC.</p> <p>The tag key and value must meet the requirements listed in <a href="#">Table 8-3</a>.</p>	<ul style="list-style-type: none"><li>• <b>Tag key:</b> vpc_key1</li><li>• <b>Key value:</b> vpc-01</li></ul>

**Table 8-2** Subnet parameter description

Parameter	Description	Example Value
AZ	An AZ is a geographic location with independent power supply and network facilities in a region. AZs are physically isolated, and AZs in the same VPC are interconnected through an internal network.	AZ2

Parameter	Description	Example Value
Name	Specifies the subnet name.	subnet-ipv6
IPv4 CIDR Block	Specifies the IPv4 CIDR block for the subnet. This value must be within the VPC CIDR range.	192.168.0.0/24
IPv6 CIDR Block	Select <b>Enable</b> for <b>IPv6 CIDR Block</b> . An IPv6 CIDR block will be automatically assigned to the subnet. IPv6 cannot be disabled after the subnet is created. Currently, you are not allowed to specify a custom IPv6 CIDR block.	N/A
Associated Route Table	Specifies the default route table to which the subnet will be associated. You can change the route table to a custom route table.	Default
Advanced Settings		
Gateway	Specifies the gateway address of the subnet.  This IP address is used to communicate with other subnets.	192.168.0.1
DNS Server Address	By default, two DNS server addresses are configured. You can change them if necessary. When multiple IP addresses are available, separate them with a comma (,).	100.125.x.x

Parameter	Description	Example Value
DHCP Lease Time	<p>Specifies the period during which a client can use an IP address automatically assigned by the DHCP server. After the lease time expires, a new IP address will be assigned to the client. If a DHCP lease time is changed, the new lease automatically takes effect when half of the current lease time has passed. To make the change take effect immediately, restart the ECS or log in to the ECS to cause the DHCP lease to automatically renew.</p> <p><b>CAUTION</b> When IPv4/IPv6 dual stack is enabled for the cluster, DHCP unlimited lease cannot be enabled for the selected node subnet.</p>	365 days or 300 hours
Tag	<p>Specifies the subnet tag, which consists of a key and value pair. You can add a maximum of ten tags to each subnet.</p> <p>The tag key and value must meet the requirements listed in <a href="#">Table 8-4</a>.</p>	<ul style="list-style-type: none"><li>• <b>Tag key:</b> subnet_key1</li><li>• <b>Key value:</b> subnet-01</li></ul>

**Table 8-3** VPC tag key and value requirements

Parameter	Requirement	Example Value
Tag key	<ul style="list-style-type: none"><li>• Cannot be left blank.</li><li>• Must be unique in a VPC.</li><li>• Can contain a maximum of 36 characters.</li><li>• Can contain letters, digits, underscores (_), and hyphens (-).</li></ul>	vpc_key1
Tag value	<ul style="list-style-type: none"><li>• Can contain a maximum of 43 characters.</li><li>• Can contain letters, digits, underscores (_), periods (.), and hyphens (-).</li></ul>	vpc-01

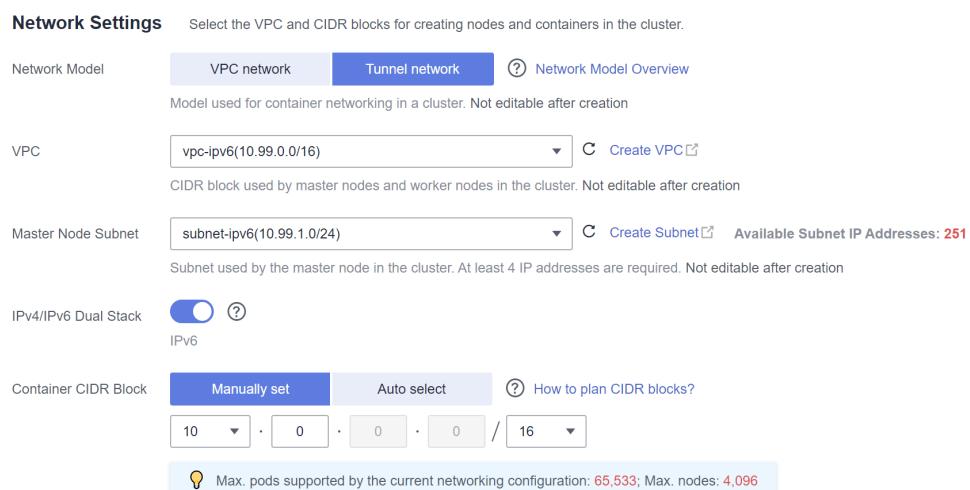
**Table 8-4** Subnet tag key and value requirements

Parameter	Requirement	Example Value
Tag key	<ul style="list-style-type: none"><li>Cannot be left blank.</li><li>Must be unique for each subnet.</li><li>Can contain a maximum of 36 characters.</li><li>Can contain letters, digits, underscores (_), and hyphens (-).</li></ul>	subnet_key1
Tag value	<ul style="list-style-type: none"><li>Can contain a maximum of 43 characters.</li><li>Can contain letters, digits, underscores (_), periods (.), and hyphens (-).</li></ul>	subnet-01

- Click **Create Now**.

## Step 2: Create a CCE Cluster

- Log in to the CCE console and create a cluster.  
Configure the network for the ECS as follows:
  - Network Model:** Select **Tunnel network**.
  - VPC:** Select the created VPC **vpc-ipv6**.
  - Master Node Subnet:** Select a subnet with IPv6 enabled. Otherwise, the **IPv6** option will not be displayed on the cluster creation page.
  - Container CIDR Block:** A proper mask must be set for the container CIDR block. The mask determines the number of available nodes in the cluster. If the mask of the container CIDR block in the cluster is set improperly, there will be only a small number of available nodes in the cluster.

**Figure 8-1** Configuring network settings

- Create a node.

The CCE console displays the nodes that support IPv6. You can directly select a node.

3. When creating a workload, if you select **LoadBalancer (ELB)** or **LoadBalancer (DNAT)** for the Service type, only IPv4 is supported.

After the creation is complete, access the cluster details page. Then, click the node name to go to the ECS details page and view the automatically allocated IPv6 address.

### Step 3: Buy a Shared Bandwidth and Adding an IPv6 Address to It

By default, the IPv6 address can only be used for private network communication. If you want to use this IPv6 address to access the Internet or be accessed by IPv6 clients on the Internet, you need to buy a shared bandwidth and add the IPv6 address to it.

If you already have a shared bandwidth, you can add the IPv6 address to the shared bandwidth without purchasing one.

#### Buying a Shared Bandwidth

1. Log in to the management console.
2. Click  in the upper left corner of the management console and select a region and a project.
3. Choose **Service List > Networking > Virtual Private Cloud**.
4. In the navigation pane, choose **Elastic IP and Bandwidth > Shared Bandwidths**.
5. In the upper right corner, click **Buy Shared Bandwidth**. On the displayed page, configure parameters as prompted.

**Table 8-5** Parameter description

Parameter	Description	Example Value
Billing Mode	Specifies the billing mode of a shared bandwidth. The billing mode can be: <ul style="list-style-type: none"><li>• <b>Yearly/Monthly:</b> You pay for the bandwidth by year or month before using it. No charges will be incurred for the bandwidth during its validity period.</li><li>• <b>Pay-per-use:</b> You pay for the bandwidth based on the amount of time you use the bandwidth.</li></ul>	Yearly/Monthly

Parameter	Description	Example Value
Region	Specifies the desired region. Regions are geographic areas that are physically isolated from each other. The networks inside different regions are not connected to each other, so resources cannot be shared across different regions. For lower network latency and faster access to your resources, select the region nearest you.	AP-Singapore
Billed By	Specifies the shared bandwidth billing factor.	Select <b>Bandwidth</b> .
Bandwidth	Specifies the shared bandwidth size in Mbit/s. The minimum bandwidth that can be purchased is 5 Mbit/s.	10
Bandwidth Name	Specifies the name of the shared bandwidth.	Bandwidth-001
Enterprise Project	When assigning the shared bandwidth, you can add the shared bandwidth to an enabled enterprise project.  An enterprise project facilitates project-level management and grouping of cloud resources and users. The name of the default project is <b>default</b> .  For details about creating and managing enterprise projects, see <a href="#">Enterprise Management User Guide</a> .	default
Required Duration	Specifies the required duration of the shared bandwidth to be purchased. You need to specify this parameter only in yearly/monthly billing mode.	2 months

6. Click **Next** to confirm the configurations and then click **Buy Now**.

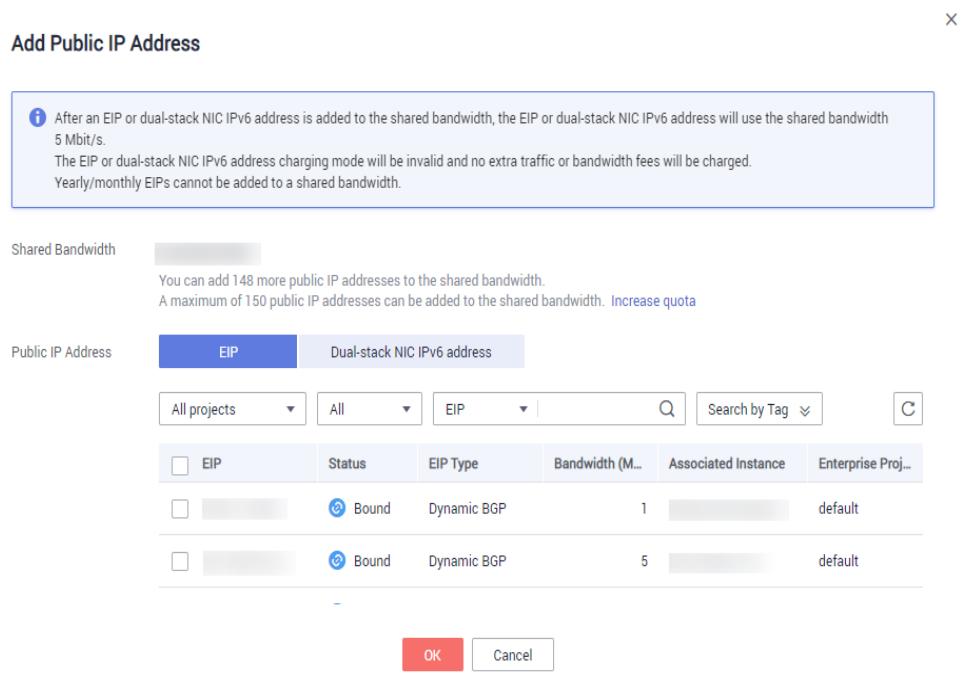
### Adding an IPv6 Address to a Shared Bandwidth

1. On the **Shared Bandwidths** page, choose **More > Add Public IP Address** in the **Operation** column.

**Figure 8-2** Adding an IPv6 address to a shared bandwidth



2. Add the IPv6 address to the shared bandwidth.

**Figure 8-3** Adding a dual-stack NIC IPv6 address

3. Click **OK**.

### Verifying the Result

Log in to an ECS and ping an IPv6 address on the Internet to verify the connectivity. **ping6 ipv6.baidu.com** is used as an example here. The execution result is displayed in **Figure 8-4**.

**Figure 8-4** Result verification

```
root@ecs-tang:~# ping6 ipv6.baidu.com
PING ipv6.baidu.com(2400:da00:2::29) 56 data bytes
64 bytes from 2400:da00:2::29: icmp_seq=1 ttl=42 time=45.6 ms
64 bytes from 2400:da00:2::29: icmp_seq=2 ttl=42 time=45.1 ms
64 bytes from 2400:da00:2::29: icmp_seq=3 ttl=42 time=44.8 ms
64 bytes from 2400:da00:2::29: icmp_seq=4 ttl=42 time=45.1 ms
```

## 8.2 Creating a Custom CCE Node Image

### Constraints

- Suggestions on using CCE node images:
  - You are advised to use the default node images maintained by CCE. These images have passed strict tests and updated in a timely manner, providing better compatibility, stability, and security.
  - Use the base images provided by CCE to create custom images.
  - Custom CCE node images are created using the open source tool **HashiCorp Packer** of v1.7.2 or later and the **open source plug-in**. The `cce-image-builder` template is provided to help you quickly build images.

### NOTE

Packer is an open-source tool used to create custom node images. Packer contains three components: builder, provisioner, and post-processor. It supports template files in JSON or HCL format. You can flexibly combine the three components to automatically create images in parallel.

Packer has the following advantages:

1. Automatic build process: You can use Packer configuration files to specify and automate the build process.
  2. High compatibility with cloud platforms: Packer can interconnect with most cloud platforms and various third-party plug-ins.
  3. Easy-to-use configuration files: Packer configuration files are simple and intuitive to write and read. Parameter definitions are easy to understand.
  4. Diverse image build functions: Common functional modules are supported. For example, the provisioner supports the shell module in remote script execution, the file module in remote file transfer, and the breakpoint module for process pauses.
- Before you create a custom node image, [submit a service ticket](#) to:
    - Obtain professional suggestions from CCE service experts on the customization.
    - **Obtain the latest node image ID of the CCE service for image customization.**
    - If later updates on the CCE node image are incompatible with the customized node image used in your production environment, you will be notified at least one month in advance and receive assistance in adaption to prevent possible failures in node and node pool scaling out.
  - When you create a custom node image, make sure:
    - You follow the instructions in this section to prevent unexpected problems.
    - You have the **sudo** or **root** permissions required to log in to VMs created from base images.
  - When the creation is complete:
    - The image creation process uses certain charging resources, including ECSs, EVS disks, EIPs, bandwidth, and IMS images. These resources are automatically released when the image is successfully created or fails to be created. Release the resources in time to ensure no charges are incurred unexpectedly.

## Precautions

- The component package on which user nodes depend is preset in the base image. The package version varies with the cluster version.
- Before you create an image, prepare:
  - An ECS executor: An ECS x86 server is used as the Linux executor. You are advised to select CentOS 7 and bind an EIP to it so that it can access the public network and install Packer.
  - Authentication credentials: Obtain the AK/SK of the tenant or user with required permissions. For details, see [How Do I Obtain an Access Key \(AK/SK\)](#).

- Security group: Packer creates a temporary ECS and uses a key pair to log in to the ECS using SSH. Ensure that **TCP:22** is enabled in the security group. For details, see [Security Group Configuration Examples](#).

## Creating a Node Image

### Step 1 Log in to the ECS executor, download and decompress cce-image-builder.

```
wget https://cce-north-4.obs.cn-north-4.myhuaweicloud.com/cce-image-builder/cce-image-builder.tgz
tar zvxf cce-image-builder.tgz
cd cce-image-builder/
```



The cce-image-builder contains:

- turbo-node.pkr.hcl # Packer configuration template used for creating the image
- scripts/\* # CCE image creation preset in the template. Do not modify it. Otherwise, the image might become unavailable.
- user-scripts/\* # **Custom package script directory preset in the template**. Take example.sh as an example. When you create a custom image, the image is automatically uploaded to the temporary server and executed.
- user-packages/\* # **Custom package directory preset in the template**. Take example.package as an example. When you create a custom image, the image is automatically uploaded to the temporary server /tmp/example.package.

### Step 2 Install [HashiCorp Packer](#). For details, see the [official documentation](#).



Install Packer of v1.7.2 or later.

Take the CentOS 7 executor as an example. Run the following command to automatically install Packer (**for reference only. For detailed operations, see the official guide**):

```
# Configure the yum repository and install Packer.
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo
sudo yum -y install packer

# Configure an alias to avoid duplicate Packer binary in the OS and check the Packer version.
rpm -q packer
alias packer=$(rpm -ql packer)
packer -v
```

### Step 3 Obtain the parameters required by **cce-image-builder/turbo-node.pkr.hcl** and configure the parameters using environment variables.

 NOTE

The **turbo-node.pkr.hcl** file is the configuration template of Packer. It defines the complete image build process. For details, see [Packer Documentation](#).

- **variables/variable**

Variable definition. **turbo-node.pkr.hcl** defines the parameters required in image build. You can set the parameters based on the live environment. For details about the parameters, see [Table 8-6](#).

- **packer**

Definition of the **packer** module. **required\_plugins** defines the add-on dependency of Packer, including the add-on source and version range. When you run **packer init**, the add-on is automatically downloaded and initialized. No manual intervention is required.

```
packer {  
    required_plugins {  
        huaweicloud = {  
            version = ">= 0.4.0"  
            source  = "github.com/huaweicloud/huaweicloud"  
        }  
    }  
}
```

- **source**

Definition of **source**. The preceding defined variables are referred to automatically set the parameters required for creating an ECS.

- **build**

Definition of **build**. The scripts are executed from top to bottom. Common modules such as the file upload module and script execution shell module are supported. For details, see the following example:

```
build {  
    sources = ["source.huaweicloud-ecs.builder"]
```

```
    provisioner "file" {  
        source      = "<source file path>"  
        destination = "<destination file path>"  
    }
```

```
    provisioner "shell" {  
        scripts = [  
            "<source script file: step1.sh>",  
            "<source script file: step2.sh>"  
        ]  
    }
```

```
    provisioner "shell" {  
        inline = ["echo foo"]  
    }
```

```
export REGION_NAME=xxx  
export IAM_ACCESS_KEY=xxx  
export IAM_SECRET_KEY=xxx  
export ECS_VPC_ID=xxx  
export ECS_NETWORK_ID=xxx  
export ECS_SECGRP_ID=xxx  
export CCE_SOURCE_IMAGE_ID=xxx
```

**Table 8-6** Variables configuration

Parameter	Description	Remarks
REGION_NAME	Region to which the project belongs	To obtain the region information, go to <a href="#">My Credentials</a> .

Parameter	Description	Remarks
IAM_ACCESS_KEY	Access key for user authentication	Apply for a temporary AK and delete it when the image is built successfully.
IAM_SECRET_KEY	Secret key for user authentication	Apply for a temporary SK and delete it when the image is built successfully.
ECS_VPC_ID	VPC ID	Used by the temporary ECS server, which must be the same as that of the executor
ECS_NETWORK_ID	Network ID of the subnet	Used by the temporary ECS server. It is recommended that the value be the same as that of the executor. It is not the subnet ID.
ECS_SECGRP_ID	Security group ID	Used by the temporary ECS server. It is recommended that the value be the same as that of the executor. Alternatively, allow the executor can be logged in to using SSH.
CCE_SOURCE_IMAGE_ID	Latest CCE node image ID	-

Note: Retain the default values of other parameters. To modify the value, refer to the description in the variable definition in **turbo-node.pkr.hcl** and configure the value using environment variables.

Use the ECS flavor variable **ecs\_az** as an example. If no AZ is specified, select a random AZ. If you want to specify an AZ, configure an environment variable as follows:

```
# export PKR_VAR_<variable name>=<variable value>
export PKR_VAR_ecs_az=xxx
```

- Step 4** Modify the **user-scripts** and **user-packages** directories in **cce-image-builder**. Modify or add the **file** and **shell** module configurations corresponding to **turbo-node.pkr.hcl** by referring to [Step 1](#).
- Step 5** Run the **make image** command and wait until the image creation is complete. The process takes about 3 to 5 minutes.

### NOTE

In the encapsulation script **packer.sh**:

- Automatic access of hashicorp.com by packer is disabled by default for privacy and security purposes.  
`export CHECKPOINT_DISABLE=false`
- The debugging detailed logs option is enabled by default for better visibility and traceability. The local packer build logs **packer\_{timestamp}.log** is specified so that the logs can be packed to the **/var/log/** directory during build. If sensitive information is involved, remove the related logic.  
`export PACKER_LOG=1`  
`export PACKER_BUILD_TIMESTAMP=$(date +%Y%m%d%H%M%S)`  
`export PACKER_LOG_PATH="packer_${PACKER_BUILD_TIMESTAMP}.log"`

For details about Packer configuration, see [Configuring Packer](#).

Log when the image build is completed:

```
--> huaweicloud-ecs.builder: Setting a 15m0s timeout for the next provisioner...
--> huaweicloud-ecs.builder: Provisioning with shell script: /tmp/packer-shell1759174699
--> huaweicloud-ecs.builder: Setting a 15m0s timeout for the next provisioner...
--> huaweicloud-ecs.builder: Uploading packer_20210530185050.log -> /var/log/packer_20210530185050.log
huaweicloud-ecs.builder: packer_20210530185050.log 43.63 KIB / 43.58 KIB [=====] 100.29% 0s
--> huaweicloud-ecs.builder: Stopping server: 9c981ac9-37b5-40af-934e-7190e6fa800e ...
huaweicloud-ecs.builder: Waiting for server to stop: 9c981ac9-37b5-40af-934e-7190e6fa800e ...
--> huaweicloud-ecs.builder: Creating the image: image-by-packer-20210530185050
--> huaweicloud-ecs.builder: Waiting for image image-by-packer-20210530185050 to become available ...
huaweicloud-ecs.builder: Image: 64e940f4-d674-4ae1-89cc-299501581c59
--> huaweicloud-ecs.builder: Deleted temporary floating IP '494617cc-a7c9-442a-b3e8-3b90c2c3f884' (94.74.101.22)
--> huaweicloud-ecs.builder: Terminating the source server: 9c981ac9-37b5-40af-934e-7190e6fa800e ...
--> huaweicloud-ecs.builder: Deleting volume: b769e29-e1fd-407b-hbec-79f353a3e671 ...
--> huaweicloud-ecs.builder: Deleting temporary keypair: packer_68b36e0b-1f16-acc5-df04-d045aba70056 ...
Build 'huaweicloud-ecs.builder' finished after 3 minutes 53 seconds.

--> Wait completed after 3 minutes 53 seconds

--> Builds finished. The artifacts of successful builds are:
--> huaweicloud-ecs.builder: An image was created: 64e940f4-d674-4ae1-89cc-299501581c59
[Sun May 30 18:54:45 CST 2021] packer.sh finish.
```

## Step 6 Clear the build files on the executor, mainly the authentication credentials in **turbo-node.pkr.hcl**.

- It is recommended that the executor be directly released. If the authentication credentials are temporary, delete them directly.
- If it is an automatic build, you are advised to add post-processor in the configuration file to execute related operations.

----End

## 8.3 Using a Private Image to Build a Worker Node Image

### Constraints

Only CCE clusters are supported. For details about how to create a custom node image for a CCE Turbo cluster, see [Creating a Custom CCE Node Image](#).

### Supported Image OSs and Kernel Versions

Currently, only clusters of version 1.15 and 1.17 are supported. **Table 8-7** lists the supported OSs and kernels.

The image OS version must be EulerOS 2.5 or CentOS 7.6.

**Table 8-7** Mappings between clusters, OSs, and kernels

OS	Cluster Version	Kernel
CentOS Linux release 7.6	v1.17.17	3.10.0-1160.15.1.el7.x86_64
	v1.17.9	3.10.0-1062.12.1.el7.x86_64
	v1.15.11	3.10.0-1062.12.1.el7.x86_64
	v1.15.6-r1	3.10.0-1062.1.1.el7.x86_64
EulerOS release 2.5	v1.17.17	3.10.0-862.14.1.5.h470.eulerosv2r7.x86_64
	v1.17.9	3.10.0-862.14.1.5.h428.eulerosv2r7.x86_64
	v1.15.11	3.10.0-862.14.1.5.h428.eulerosv2r7.x86_64
	v1.15.6-r1	3.10.0-862.14.1.5.h328.eulerosv2r7.x86_64

- When creating an image, follow the instructions in this section to prevent unexpected problems.
- To log in to VMs created from base images, users are required to have the **sudo** **root** or **root** permissions.

## Preparation

### Notes

- Check whether the dependencies required by the current OS version are installed on the ECS used to create an image.
- After the image is created, the ECSs will not be deleted. You need to delete them manually.
- The private image installation package contains the script and dependent components required for installing the node. The package version varies depending on the cluster version.
- Create a private image depends on the lvm2, conntrack, sudo, NetworkManager, haveged, ntp, numactl, ipset, and auditd components.

### Procedure

- Step 1** Before creating an image, you need to create two ECSs and bind EIPs to them. One ECS is used as the executor, and the other is used to create the image. The recommended ECS specifications are 4 vCPUs and memory of 8 GB.

 NOTE

- For details about how to use an image file to create a private image, see [Appendix](#).
- An EIP is bound to remotely transfer the installation package and send dependency installation commands.
- It takes about 10 minutes to create an image, which incurs network traffic and resource fees.
- Ensure that **TCP port 22** is enabled in the new inbound rule of the security group for both ECSs. For details about how to create a security group, see [Security Group Configuration Examples](#).

**Step 2** Creating a private image depends on the lvm2, conntrack, sudo, NetworkManager, haveged, ntp, numactl, ipset, and auditd components. Check whether the corresponding dependencies required by the current OS version have been installed on the ECS used to create the image.

eg: `yum install lvm2 conntrack sudo NetworkManager haveged ntp numactl ipset audit -y;`

If "no package" is displayed when you run the **yum install haveged** command, run the following commands:

```
yum install epel-release -y;  
yum install haveged -y;
```

Check and uninstall the unnecessary dependency: network-hotplug

```
eg: rpm -e network-hotplug
```

If the base image requires the auditd service to run properly, run the **systemctl status auditd** command to check the service status and disable selinux if the service runs improperly. In some cases, if selinux is enabled, the auditd service cannot run properly. (Run the **vim /etc/selinux/config** command and change the value of **SELINUX** to **disabled**. Check whether the **/var/log/audit** directory exists. If the directory does not exist, create it. Restart the node for the modification to take effect.) If the auditd service does not exist, install the auditd service.

**Step 3** Uploading the **init\_envs.conf** File

The **init\_envs.conf** file stores the configurations of the VM created from the base image. Apply for a server on the ECS console or use an existing server, log in the server, and upload the **init\_envs.conf** file to the **/root** directory on the server.

The following is an example of the **init\_envs.conf** file. Set the parameters based on the description in [Table 8-8](#).

You are advised to use a common tenant account to create an image. Otherwise, the API call may fail due to the security requirement of the **op\_svc\_xxx** account.

```
DOMAIN_NAME=""  
USER_NAME=""  
PROJECT_NAME=""  
PROJECT_ID=""  
IMS_ENDPOINT=""  
KEY_PAIR_NAME=""  
IMAGE_NAME=""
```

**Table 8-8** Description of the init\_envs.conf file

Parameter	Description
DOMAIN_NAME	Account that creates an image.
USER_NAME	User that creates an image.
PROJECT_NAME	Region to which the project belongs. View the region and project ID on the <a href="#">My Credentials</a> page.
PROJECT_ID	Project ID. View the region and project ID on the <a href="#">My Credentials</a> page.
IMS_ENDPOINT	ims. <i>region</i> .myhuaweicloud.com For details about regions, see <a href="#">Regions and Endpoints</a> .
KEY_PAIR_NAME	(Optional) Name of the key pair, which is the same as the name of the key pair file in the /root directory.
IMAGE_NAME	Optional. The default value is the <b>BASIC-NODE-IMG-timestamp</b> .

**Step 4** Obtain the key file. (Skip this step if you log in to the server using a password.)

A key file is the authentication file required for creating an ECS. You can use existing keys or create new keys. For example, log in to the server and upload the key file named **Keypair.pem** to the /root directory to create an ECS.

1. Log in to the management console.
2. Choose **Service List > Compute > Elastic Cloud Server**.
3. In the navigation pane, choose **Key Pair**. On the page displayed, click **Create Key Pair**.
4. Enter a key pair name and click **OK**.
5. In the dialog box displayed, click **OK**.

View and save the key pair. To ensure security, a key pair can be downloaded only once. Keep the key pair secure for login.

For details about how to create a key pair, see [Creating a Key Pair](#).

----End

## Creating a Node Image

**Step 1** Log in to the ECS that functions as the executor.

Check whether the **init\_envs.conf** and **Keypair.pem** files have been uploaded to the /root directory.

**Step 2** Run the image creation script.

**Table 8-9** Commands to be run

Site	Command
Huawei Cloud	<p>Click <a href="#">here</a> to obtain the installation package.</p> <p>Decompress the installation package. When executing <b>create.sh</b> in the <b>node-image/conf</b> directory, add the following five parameters.</p> <p>The following is an example:</p> <pre>bash create.sh \${NODE_EIP} \${PASSWORD} \${ECS_PASSWORD} \${ECS_INSTANCE_ID} \${LINUX_ROLE}</pre> <p>The parameters are described as follows:</p> <p><b>NODE_EIP</b>: EIP of the ECS used to create the image.</p> <p><b>PASSWORD</b>: password for logging in to Huawei Cloud. This password is used to obtain the token for creating an IMS image.</p> <p><b>ECS_PASSWORD</b>: password for logging in to the ECS used to create the image. If you use a key pair for login, do not set this parameter (a null value).</p> <p><b>ECS_INSTANCE_ID</b>: instance ID of the ECS used to create the image.</p> <p><b>LINUX_ROLE</b>: role of the user who creates the image. The default user is <b>root</b>. If the user is not <b>root</b>, set the permission as follows: <code>/etc/sudoers</code> <code>Username ALL=(ALL) NOPASSWD: ALL</code></p> <p><b>NOTE</b></p> <p>Example:</p> <p><b>Use a key pair</b>: bash create.sh 127.0.0.1 [Password of the user used to create the image] [Login password of the node corresponding to the machine used to create the image, which is empty when a key pair is used] d4d92ca7-256a-44ef-942d-ab326bed1d87 root</p> <p><b>Use a password</b>: bash create.sh 127.0.0.1 [Password of the user used to create the image] [Login password of the node corresponding to the machine used to create the image] d4d92ca7-256a-44ef-942d-ab326bed1d87 root</p>

**Step 3** After the image is created, use the image to create a worker node for verification.

----End

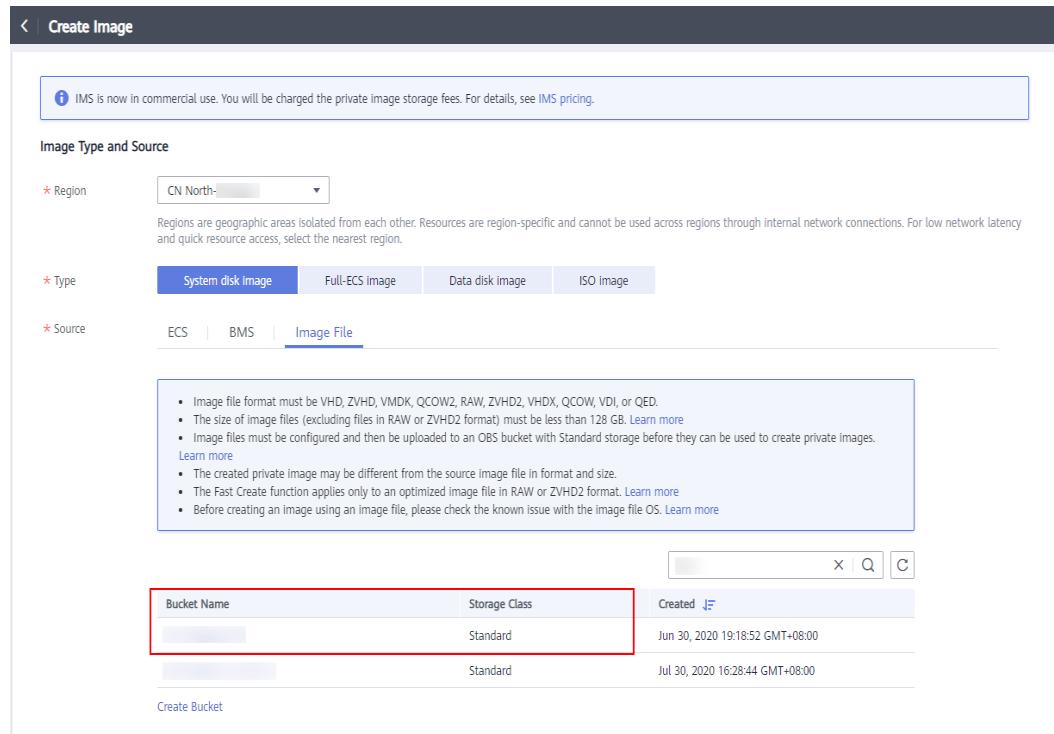
## Appendix

This operation is required only when you use image files to create a private image. Perform the following steps:

**Step 1** Obtain the base image file.

**Step 2** Import the obtained image file to an OBS bucket of your account.

**Step 3** On IMS, click **Create Image** on the **Private Images** tab page. Select **Image File** for **Source**, which is the image file in the OBS bucket. Set the system disk to 40 GB, configure other parameters as required, and click **Create Now**.

**Figure 8-5 Creating an image**

----End

## 8.4 Cleaning Up CCE Resources on a Deleted Node

### Application Scenario

If a cluster contains yearly/monthly billed nodes or nodes managed by the cluster, you will be prompted to clean up CCE resources on ECSs when deleting the cluster or these nodes. If you do not want to clean up CCE resources when deleting a cluster or nodes, follow the procedure described here when you want to do so.

#### NOTICE

Uninstalling an ECS will delete the CCE system user **paaS** and docker resources from the ECS. To preserve data before the cleanup, make a copy of the data or [submit a service ticket](#).

### Procedure

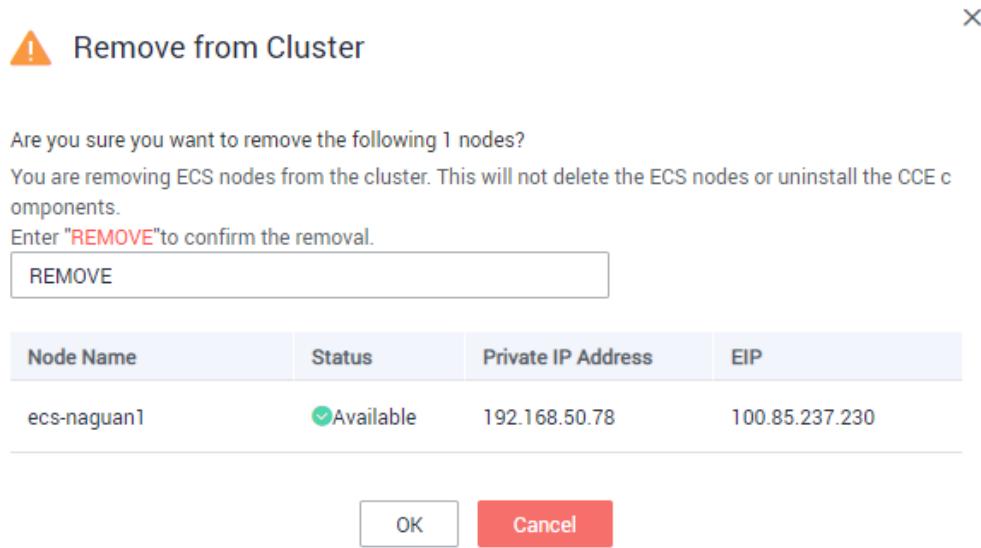
- Step 1** Log in to the CCE console. In the navigation pane, choose **Resource Management** > **Nodes**. In the same row as the node whose CCE resources will be cleaned up, choose **More** > **Remove**.

**Figure 8-6** Removing a managed node

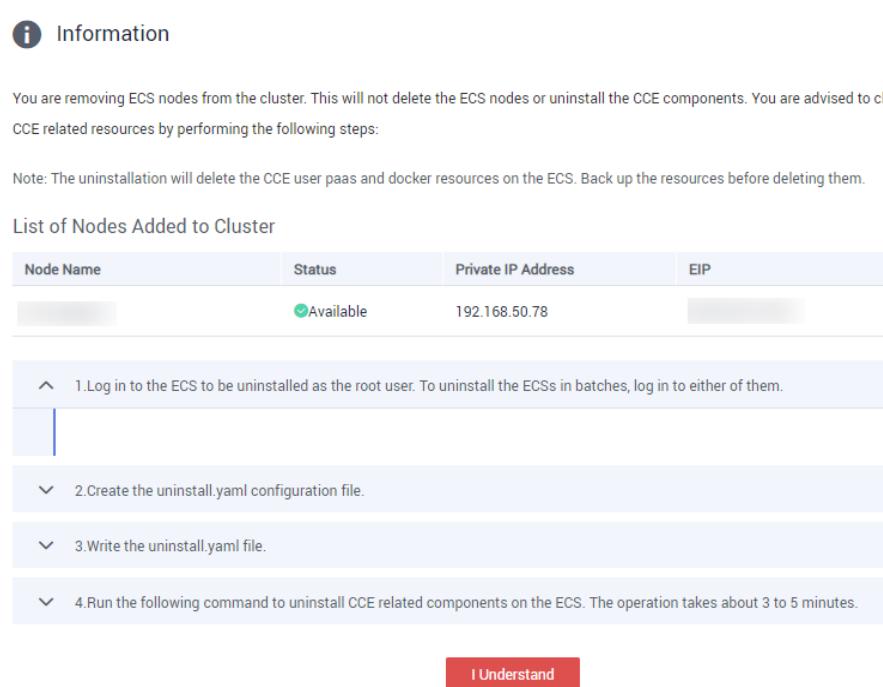
Node Name	Status	Node Pool	Specifications	Allocatable	IP	AZ	Node Type	Operation
[Redacted]	Available	DefaultPool	CPU: Intel Xeon Gold 5... Memory: 3840 DDR4 R...	CPU: 53.38 Core Memory: 360.10 GiB	192.168.50.34 (Private) 100.94.19.108 (EIP)	A21	Node created	Monitoring Manager Labels More ▾
[Redacted]	Available	DefaultPool	2 cores   4 GB	CPU: 1.63 Core Memory: 1.54 GiB	192.168.50.78 (Private) 100.85.237.230 (EIP)	A21	Add to Cluster	Monitoring Manager Labels More ▾
[Redacted]	Available	DefaultPool	8 cores   16 GB	CPU: 6.73 Core Memory: 11.77 GiB	192.168.50.222 (Private)	A22	Node created	Monitoring Manager Labels More ▾
[Redacted]	Available	DefaultPool	4 cores   8 GB	CPU: 3.00 Core Memory: 5.04 GiB	192.168.50.149 (Private)	A21	Node created	Monitoring Manager Labels More ▾

**Step 2** In the **Remove from Cluster** dialog box, enter **REMOVE** to confirm the removal of the node, and click **OK**.

**Figure 8-7** Confirming the removal of a managed node



**Step 3** Read the message displayed on the page and clean up CCE resources by following the on-screen instructions.

**Figure 8-8** On-screen instructions for cleaning up CCE resources on a node

----End

## 8.5 Connecting to Multiple Clusters Using kubectl

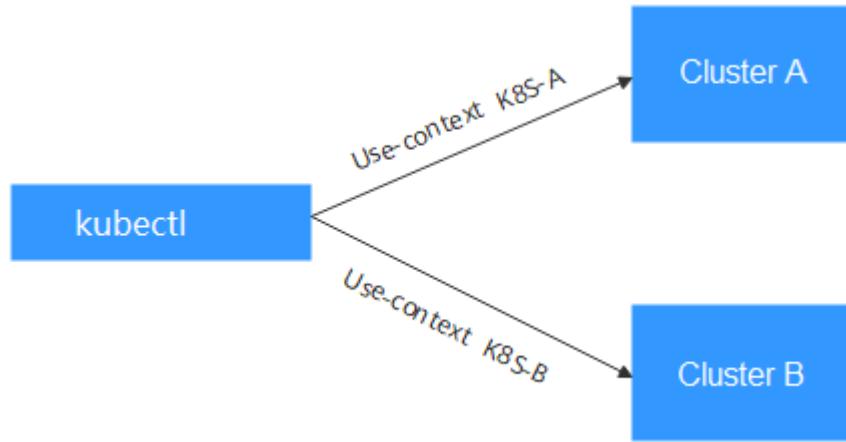
### Painpoint

When you have multiple CCE clusters, you may find it difficult to efficiently connect to all of them.

### Solution

This section describes how to configure access to multiple clusters by modifying **kubeconfig.json**. The file describes multiple clusters, users, and contexts. To access different clusters, run the **kubectl config use-context** command to switch between contexts.

**Figure 8-9** Using kubectl to connect to multiple clusters



## Prerequisites

kubectl can access multiple clusters.

## Introduction to kubeconfig.json

kubeconfig.json is the configuration file of kubectl. You can download it on the cluster details page.

The screenshot shows the 'Cluster Details' page for a cluster named 'vpc-demo'. It includes sections for Status, Networking Configuration, Connection Information, and Other. In the Connection Information section, the 'kubectl' button is highlighted with a red border. To the right, a modal window titled 'Access Cluster example Through kubectl' provides instructions for downloading and configuring kubectl to access the cluster. It includes steps for download, configuration, and setting access mode.

The content of kubeconfig.json is as follows:

```
{
  "kind": "Config",
  "apiVersion": "v1",
  "preferences": {},
  "clusters": [
    {
      "name": "internalCluster",
      "cluster": {
        "server": "https://192.168.0.85:5443",
        "certificate-authority-data": "LS0tLS1CRUUIE..."
      }
    },
    {
      "name": "externalCluster",
    }
  ]
}
```

```
"cluster": {
    "server": "https://xxx.xxx.xxx.xxx:5443",
    "insecure-skip-tls-verify": true
},
"users": [
    {
        "name": "user",
        "user": {
            "client-certificate-data": "LS0tLS1CRUdJTIBDRVJ...",
            "client-key-data": "LS0tLS1CRUdJTIBS..."
        }
    }
],
"contexts": [
    {
        "name": "internal",
        "context": {
            "cluster": "internalCluster",
            "user": "user"
        }
    },
    {
        "name": "external",
        "context": {
            "cluster": "externalCluster",
            "user": "user"
        }
    }
],
"current-context": "external"
}
```

It mainly consists of three sections.

- **clusters:** describes the cluster information, mainly the access address of the cluster.
- **users:** describes information about the users who access the cluster. It includes the **client-certificate-data** and **client-key-data** certificate files.
- **contexts:** describes the configuration contexts. You switch between contexts to access different clusters. A context is associated with **user** and **cluster**, that is, it defines which user accesses which cluster.

The preceding kubeconfig.json defines the private network address and public network address of the cluster as two clusters with two different contexts. You can switch the context to use different addresses to access the cluster.

## Configuring Access to Multiple Clusters

The following steps walk you through the procedure of configuring access to two clusters by modifying kubeconfig.json.

This example configures only the public network access to the clusters. If you want to access multiple clusters over private networks, retain the **clusters** field and ensure that the clusters can be accessed over private networks. Its configuration is similar to that described in this example.

- Step 1** Download kubeconfig.json of the two clusters and delete the lines related to private network access, as shown in the following figure.

- Cluster A:

```
{
    "kind": "Config",
    "apiVersion": "v1",
    "preferences": {},
    "clusters": [ {
        "name": "externalCluster",
        "cluster": {

```

```

        "server": "https://119.xxx.xxx:5443",
        "insecure-skip-tls-verify": true
    },
],
"users": [
    {
        "name": "user",
        "user": {
            "client-certificate-data": "LS0tLS1CRUdJTxM...",
            "client-key-data": "LS0tLS1CRUdJTiB..."
        }
    }
],
"contexts": [
    {
        "name": "external",
        "context": {
            "cluster": "externalCluster",
            "user": "user"
        }
    }
],
"current-context": "external"
}

```

- Cluster B:

```

{
    "kind": "Config",
    "apiVersion": "v1",
    "preferences": {},
    "clusters": [ {
        "name": "externalCluster",
        "cluster": {
            "server": "https://124.xxx.xxx:5443",
            "insecure-skip-tls-verify": true
        }
    }],
    "users": [
        {
            "name": "user",
            "user": {
                "client-certificate-data": "LS0tLS1CRUdJTxM...",
                "client-key-data": "LS0rTUideUdJTiB..."
            }
        }
    ],
    "contexts": [
        {
            "name": "external",
            "context": {
                "cluster": "externalCluster",
                "user": "user"
            }
        }
    ],
    "current-context": "external"
}

```

The preceding files have the same structure except that the **client-certificate-data** and **client-key-data** fields of **user** and the **clusters.cluster.server** field are different.

## Step 2 Modify the **name** field as follows:

- Cluster A:

```

{
    "kind": "Config",
    "apiVersion": "v1",
    "preferences": {},
    "clusters": [ {
        "name": "Cluster-A",
        "cluster": {
            "server": "https://119.xxx.xxx:5443",
            "insecure-skip-tls-verify": true
        }
    }],
    "users": [
        {
            "name": "Cluster-A-user",
        }
    ]
}

```

```

    "user": {
      "client-certificate-data": "LS0tLS1CRUdJTxM...",
      "client-key-data": "LS0tLS1CRUdJTiB...."
    },
  ],
  "contexts": [
    {
      "name": "Cluster-A-Context",
      "context": {
        "cluster": "Cluster-A",
        "user": "Cluster-A-user"
      }
    },
    "current-context": "Cluster-A-Context"
  ]
}

```

- Cluster B:

```

{
  "kind": "Config",
  "apiVersion": "v1",
  "preferences": {},
  "clusters": [ {
    "name": "Cluster-B",
    "cluster": {
      "server": "https://124.xxx.xxx.xxx:5443",
      "insecure-skip-tls-verify": true
    }
  }],
  "users": [ {
    "name": "Cluster-B-user",
    "user": {
      "client-certificate-data": "LS0tLS1CRUdJTxM...",
      "client-key-data": "LS0rTUideUdJTiB...."
    }
  }],
  "contexts": [
    {
      "name": "Cluster-B-Context",
      "context": {
        "cluster": "Cluster-B",
        "user": "Cluster-B-user"
      }
    },
    "current-context": "Cluster-B-Context"
  ]
}

```

**Step 3** Combine these two files.

The file structure remains unchanged. Combine the contents of **clusters**, **users**, and **contexts** as follows:

```
{
  "kind": "Config",
  "apiVersion": "v1",
  "preferences": {},
  "clusters": [ {
    "name": "Cluster-A",
    "cluster": {
      "server": "https://119.xxx.xxx.xxx:5443",
      "insecure-skip-tls-verify": true
    }
  },
  {
    "name": "Cluster-B",
    "cluster": {
      "server": "https://124.xxx.xxx.xxx:5443",
      "insecure-skip-tls-verify": true
    }
  }],
  "users": [ {
    "name": "Cluster-A-user",
    "user": {
      "client-certificate-data": "LS0tLS1CRUdJTxM...",
      "client-key-data": "LS0rTUideUdJTiB...."
    }
  }]
}
```

```
        "client-certificate-data": "LS0tLS1CRUdJTxM...",
        "client-key-data": "LS0tLS1CRUdJTiB...."
    }
}
{
    "name": "Cluster-B-user",
    "user": {
        "client-certificate-data": "LS0tLS1CRUdJTxM...",
        "client-key-data": "LS0tTUideUdJTiB...."
    }
},
"contexts": [
    {
        "name": "Cluster-A-Context",
        "context": {
            "cluster": "Cluster-A",
            "user": "Cluster-A-user"
        }
    },
    {
        "name": "Cluster-B-Context",
        "context": {
            "cluster": "Cluster-B",
            "user": "Cluster-B-user"
        }
    }
],
"current-context": "Cluster-A-Context"
}
```

----End

## Verification

Run the following commands to copy the file to the kubectl configuration path:

```
mkdir -p $HOME/.kube
mv -f kubeconfig.json $HOME/.kube/config
```

Run the kubectl commands to check whether the two clusters can be connected.

```
# kubectl config use-context Cluster-A-Context
Switched to context "Cluster-A-Context".
# kubectl cluster-info
Kubernetes control plane is running at https://119.xxx.xxx:5443
CoreDNS is running at https://119.xxx.xxx:5443/api/v1/namespaces/kube-system/services/coredns:dns/
proxy
```

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

```
# kubectl config use-context Cluster-B-Context
Switched to context "Cluster-B-Context".
# kubectl cluster-info
Kubernetes control plane is running at https://124.xxx.xxx:5443
CoreDNS is running at https://124.xxx.xxx:5443/api/v1/namespaces/kube-system/services/coredns:dns/
proxy
```

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

## 8.6 Creating a Node Injection Script

### Background

If you need to install some tools or perform custom security hardening on a node in advance, you need to inject some scripts when creating the node. CCE allows

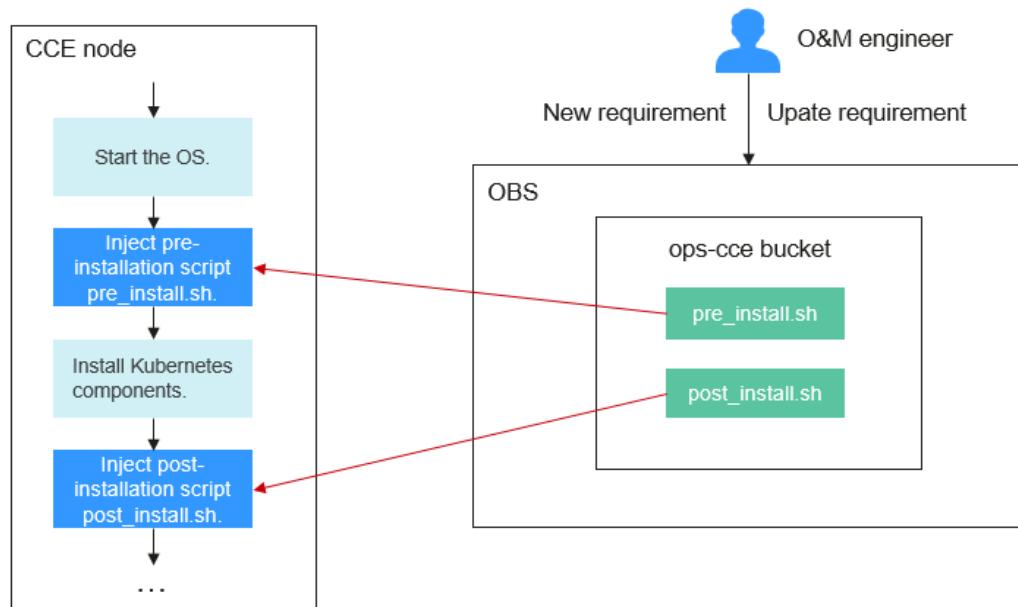
you to inject pre-installation and post-installation scripts when creating a node. However, this function has the following restrictions:

- The characters of the injection scripts are limited.
- The injected script content needs to be frequently modified to meet various requirements and scenarios. However, injection scripts of the CCE node pool are fixed and do not support frequent modifications.

## Solution

This section describes how to use CCE and OBS to provide a simplified, scalable, and easy-to-maintain solution, so that you can perform custom operations on CCE nodes.

Store the pre-installation and post-installation scripts in OBS. During node pool creation, CCE automatically pulls and executes the scripts stored in OBS based on the configured injection scripts. In this way, the configuration of the CCE node pool does not need to be changed. If there are new requirements, you only need to update the scripts in OBS.



## Suggestions on Maintaining OBS Buckets

- If there is no OBS bucket dedicated for O&M, create an OBS bucket dedicated for O&M.
- Create a multi-level directory **tools/cce** in the bucket to represent CCE-specific tools for easy maintenance. You can also store other tool scripts in this directory later.

## Precautions

- If the custom operation implemented by the script fails, the normal service running is affected. You are advised to add a check program at the end of the script. If the check fails, stop the kubelet process in the post-installation script to prevent services from being scheduled to the node.

- ```
systemctl stop kubelet-monitor
systemctl stop kubelet
```
- Do not include sensitive information in the scripts to prevent information leakage.

## Procedure

**Step 1** Create an OBS bucket.

**Step 2** Upload the pre-installation and post-installation scripts. The **pre\_install.sh** and **post\_install.sh** scripts are used as examples.

**Figure 8-10** Uploading scripts

| Name            | Storage Class | Size     | Encrypted |
|-----------------|---------------|----------|-----------|
| post_install.sh | Standard      | 11 bytes | No        |
| pre_install.sh  | Standard      | 11 bytes | No        |

**Step 3** Configure a read-only security policy for the scripts to ensure that the scripts can be downloaded without entering a password on the CCE nodes but cannot be downloaded from the Internet.

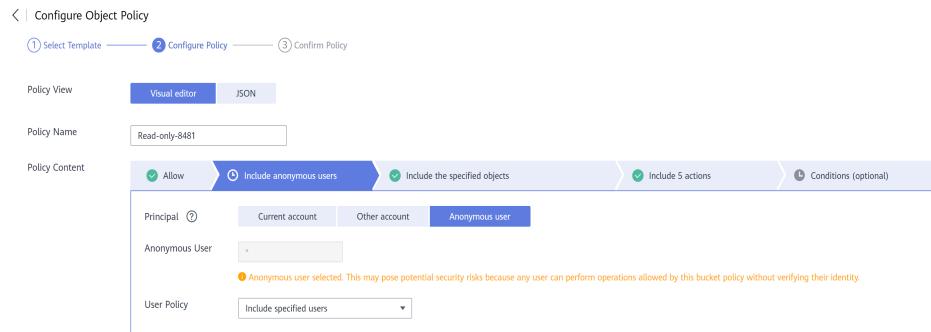
1. Configure a policy for the **tools/cce** directory and select a read-only policy template.

**Figure 8-11** Configuring an object policy

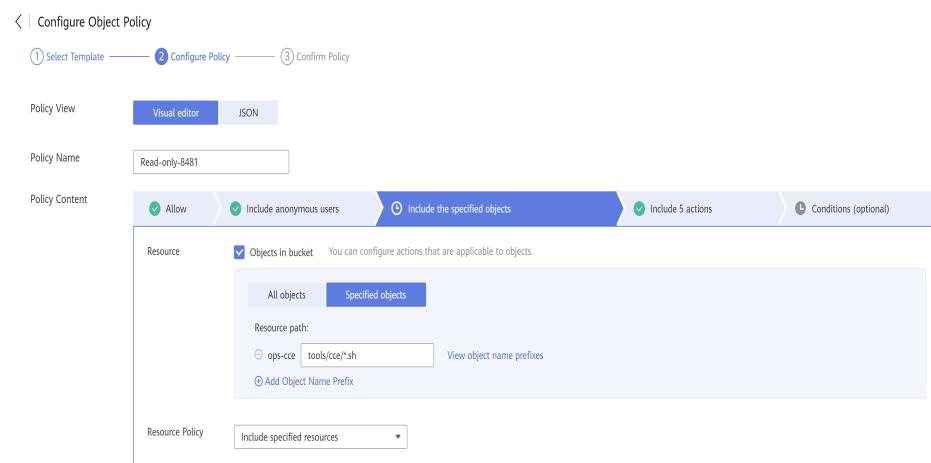
| Operation               |
|-------------------------|
| Delete                  |
| Configure Object Policy |

2. Modify the configuration information about the anonymous user, specified objects, and conditions.

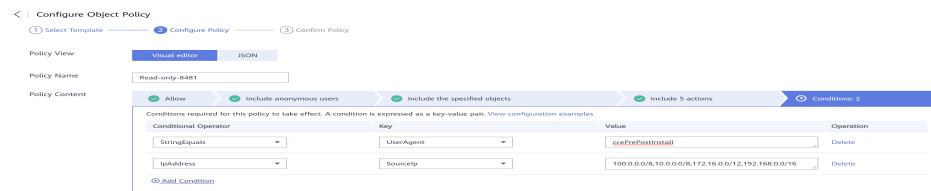
**Figure 8-12 Anonymous user**



**Figure 8-13 Specified objects**



**Figure 8-14 Conditions**



Two conditions are set: **UserAgent** and **Sourcelp**.

- **UserAgent** functions in the similar way as a key. The specified User-Agent request header and the corresponding key value must be carried during access.
- **Sourcelp** is used to prevent access from external networks. Set this parameter to **100.0.0.0/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16,214.0.0.0/8**, among which **100.0.0.0/8** and **214.0.0.0/8** are private IP ranges, among whom **10.0.0.0/8**, **172.16.0.0/12**, and **192.168.0.0/16** are commonly used.

For details about how to configure an OBS policy, see [Configuring an Object Policy](#) and [Bucket Policy Parameters](#).

**Step 4** Configure the pre-installation script and post-installation script when creating a node pool on CCE.

Enter the following scripts in the **Advanced ECS Settings** on the **Create Node Pool** page:

|                           |                                                                                                                                                                                                                                      |           |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| Pre-Installation Command  | <pre>curl -H "User-Agent: ccePrePostInstall" https://ops-cce.obs.cn-north-4.myhuaweicloud.com/tools/cce/pre_install.sh -o /tmp/pre_install.sh &amp;&amp; bash -x /tmp/pre_install.sh &gt; /tmp/pre_install.log 2&gt;&amp;1</pre>     | 196/1,000 |
| Post-Installation Command | <pre>curl -H "User-Agent: ccePrePostInstall" https://ops-cce.obs.cn-north-4.myhuaweicloud.com/tools/cce/post_install.sh -o /tmp/post_install.sh &amp;&amp; bash -x /tmp/post_install.sh &gt; /tmp/post_install.log 2&gt;&amp;1</pre> | 200/1,000 |

In the scripts below, the **curl** command is run to download **pre\_install.sh** and **post\_install.sh** from OBS to the **/tmp** directory, and then **pre\_install.sh** and **post\_install.sh** are executed.

Pre-installation script:

```
curl -H "User-Agent: ccePrePostInstall" https://ops-cce.obs.ap-southeast-1.myhuaweicloud.com/tools/cce/pre_install.sh -o /tmp/pre_install.sh && bash -x /tmp/pre_install.sh > /tmp/pre_install.log 2>&1
```

Post-installation script:

```
curl -H "User-Agent: ccePrePostInstall" https://ops-cce.obs.ap-southeast-1.myhuaweicloud.com/tools/cce/post_install.sh -o /tmp/post_install.sh && bash -x /tmp/post_install.sh > /tmp/post_install.log 2>&1
```

#### BOOK NOTE

- The actual value of **User-Agent** must be configured based on the OBS bucket policy.
- The bucket address in the link must be configured based on site requirements.

----End

## 8.7 Adding a Second Data Disk to a Node in a CCE Cluster

You can use the pre-installation script feature to configure CCE cluster nodes (ECSs). For details, see [Creating a Hybrid Cluster > Configure Advanced Settings](#).

#### BOOK NOTE

- When creating a node in a cluster of v1.13.10 or later, if a data disk is not managed by LVM, follow instructions in this section to format the data disk before adding the disk. Otherwise, the data disk will still be managed by LVM.
- When creating a node in a cluster earlier than v1.13.10, you must format the data disks that are not managed by LVM. Otherwise, either these data disks or the first data disk will be managed by LVM.

Before using this feature, write a script that can format data disks and save it to your OBS bucket. This script must be executed by user **root**.

### Input Parameters

1. Set the script name to **formatdisk.sh**, save the script to your OBS bucket, and obtain the address of the script in OBS. For details, see [Accessing an Object Using Its URL](#).
2. You need to specify the size of the Docker data disk (the data disk managed by LVM is called the Docker data disk). The size of the Docker disk must be

different from that of the second disk. For example, the Docker data disk is 100 GB and the new disk is 110 GB.

3. Set the mount path of the second data disk, for example, **/data/code**.

Run the following command in the pre-installation script to format the disk:

```
cd /tmp;curl -k -X GET OBS bucket address /formatdisk.sh -1 -O;fdisk -l;sleep 30;bash -x formatdisk.sh  
100 /data/code;fdisk -l
```

Example script (**formatdisk.sh**):

```
dockerdisksize=$1
mountdir=$2
systemdisksize=40
i=0
while [ 20 -gt $i ]; do
    echo $i;
    if [ $(lsblk -o KNAME,TYPE | grep disk | grep -v nvme | awk '{print $1}' | awk '{ print "/dev/"$1}' |wc -l) -ge 3 ]; then
        break
    else
        sleep 5
    fi;
    i=$((i+1))
done
all_devices=$(lsblk -o KNAME,TYPE | grep disk | grep -v nvme | awk '{print $1}' | awk '{ print "/dev/"$1}')
for device in ${all_devices[@]}; do
    isRawDisk=$(lsblk -n $device 2>/dev/null | grep disk | wc -l)
    if [[ ${isRawDisk} > 0 ]]; then
        # is it partitioned ?
        match=$(lsblk -n $device 2>/dev/null | grep -v disk | wc -l)
        if [[ ${match} > 0 ]]; then
            # already partited
            [[ -n "${DOCKER_BLOCK_DEVICES}" ]] && echo "Raw disk ${device} has been partition, will skip this device"
            continue
        fi
    else
        isPart=$(lsblk -n $device 2>/dev/null | grep part | wc -l)
        if [[ ${isPart} -ne 1 ]]; then
            # not parted
            [[ -n "${DOCKER_BLOCK_DEVICES}" ]] && echo "Disk ${device} has not been partition, will skip this device"
            continue
        fi
        # is used ?
        match=$(lsblk -n $device 2>/dev/null | grep -v part | wc -l)
        if [[ ${match} > 0 ]]; then
            # already used
            [[ -n "${DOCKER_BLOCK_DEVICES}" ]] && echo "Disk ${device} has been used, will skip this device"
            continue
        fi
        isMount=$(lsblk -n -o MOUNTPOINT $device 2>/dev/null)
        if [[ -n ${isMount} ]]; then
            # already used
            [[ -n "${DOCKER_BLOCK_DEVICES}" ]] && echo "Disk ${device} has been used, will skip this device"
            continue
        fi
        isLvm=$(sfdisk -lqL 2>>/dev/null | grep $device | grep "8e.*Linux LVM")
        if [[ ! -n ${isLvm} ]]; then
            # part system type is not Linux LVM
            [[ -n "${DOCKER_BLOCK_DEVICES}" ]] && echo "Disk ${device} system type is not Linux LVM, will skip this device"
            continue
        fi
        block_devices_size=$(lsblk -n -o SIZE $device 2>/dev/null | awk '{ print $1}')
        if [[ ${block_devices_size}"x" != "${dockerdisksize}Gx" ]] && [[ ${block_devices_size}"x" != "${systemdisksize}Gx" ]]; then
```

```
echo "n
p
1

w
" | fdisk $device
    mkfs -t ext4 ${device}1
    mkdir -p $mountdir
    uuid=$(blkid ${device}1 | awk '{print $2}')
    echo "${uuid} ${mountdir} ext4 noatime 0 0" | tee -a /etc/fstab >/dev/null
        mount $mountdir
    fi
done
```

 NOTE

If the preceding example cannot be executed, use the dos2unix tool to convert the format.

# 9 Networking

## 9.1 Planning CIDR Blocks for a Cluster

Before creating a cluster on CCE, determine the number of VPCs, number of subnets, container CIDR blocks, and Services for access based on service requirements.

This topic describes the addresses in a CCE cluster in a VPC and how to plan CIDR blocks.

### Notes and Constraints

To access a CCE cluster through a VPN, ensure that the VPN does not conflict with the VPC CIDR block where the cluster resides and the container CIDR block.

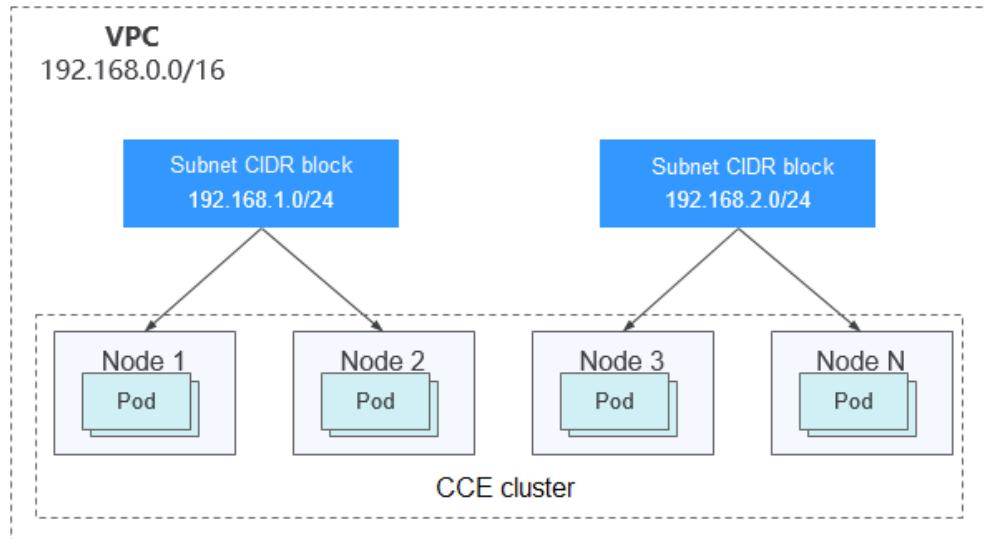
### Basic Concepts

- **VPC CIDR Block**

Virtual Private Cloud (VPC) enables you to provision logically isolated, configurable, and manageable virtual networks for cloud servers, cloud containers, and cloud databases. You have complete control over your virtual network, including selecting your own CIDR block, creating subnets, and configuring security groups. You can also assign EIPs and allocate bandwidth in your VPC for secure and easy access to your business system.

- **Subnet CIDR Block**

A subnet is a network that manages ECS network planes. It supports IP address management and DNS. The IP addresses of all ECSs in a subnet belong to the subnet.

**Figure 9-1** VPC CIDR block architecture

By default, ECSs in all subnets of the same VPC can communicate with one another, while ECSs in different VPCs cannot communicate with each other.

You can create a peering connection on VPC to enable ECSs in different VPCs to communicate with each other.

- **Container (Pod) CIDR Block**

Pod is a Kubernetes concept. Each pod has an IP address.

When creating a cluster on CCE, you can specify the pod (container) CIDR block, which cannot overlap with the subnet CIDR block. For example, if the subnet CIDR block is 192.168.0.0/16, the container CIDR block cannot be 192.168.0.0/18 or 192.168.1.0/18, because these addresses are included in 192.168.0.0/16.

- **Container Subnet (Only for CCE Turbo Clusters)**

In a CCE Turbo cluster, a container is assigned an IP address from the CIDR block of a VPC. The container subnet can overlap with the subnet CIDR block. Note that the subnet you select determines the maximum number of pods in the cluster. After a cluster is created, you can only add container subnets but cannot delete them.

- **Service CIDR Block**

Service is also a Kubernetes concept. Each Service has an address. When creating a cluster on CCE, you can specify the Service CIDR block. Similarly, the Service CIDR block cannot overlap with the subnet CIDR block or the container CIDR block. The Service CIDR block can be used only within a cluster.

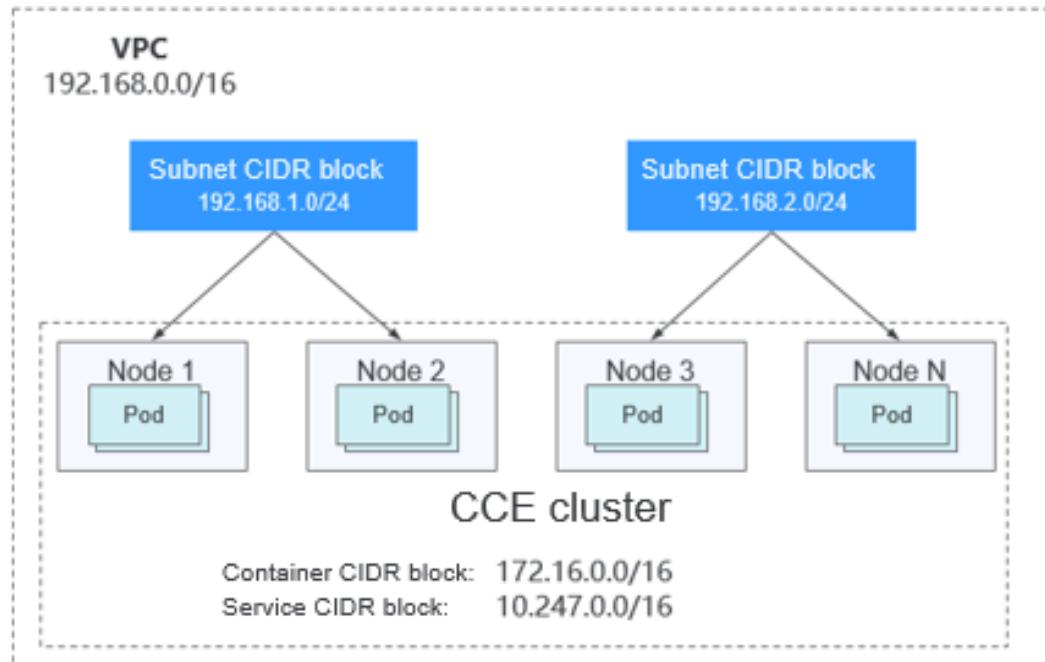
## Single-VPC Single-Cluster Scenarios

**CCE Clusters:** include clusters in VPC network model and container tunnel network model. [Figure 9-2](#) shows the CIDR block planning of a cluster.

- **VPC CIDR Block:** specifies the VPC CIDR block where the cluster resides. The size of this CIDR block affects the maximum number of nodes that can be created in the cluster.

- Subnet CIDR Block: specifies the subnet CIDR block where the node in the cluster resides. The subnet CIDR block is included in the VPC CIDR block. Different nodes in the same cluster can be allocated to different subnet CIDR blocks.
- Container CIDR Block: cannot overlap with the subnet CIDR block.
- Service CIDR Block: cannot overlap with the subnet CIDR block or the container CIDR block.

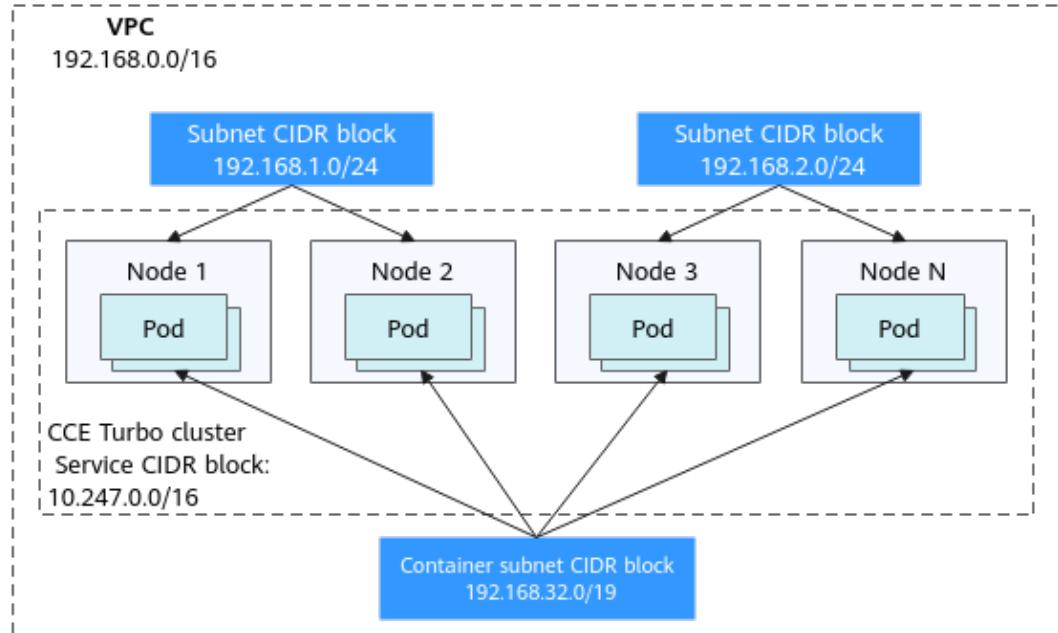
**Figure 9-2** Network CIDR block planning in the single-VPC single-cluster scenario (CCE cluster)



**Figure 9-3** shows the CIDR block planning for a **CCE Turbo cluster** (cloud native network 2.0).

- VPC CIDR Block: specifies the VPC CIDR block where the cluster resides. The size of this CIDR block affects the maximum number of nodes that can be created in the cluster.
- Subnet CIDR Block: specifies the subnet CIDR block where the node in the cluster resides. The subnet CIDR block is included in the VPC CIDR block. Different nodes in the same cluster can be allocated to different subnet CIDR blocks.
- Container Subnet CIDR Block: The container subnet is included in the VPC CIDR block and can overlap with the subnet CIDR block or even be the same as the subnet CIDR block. Note that the container subnet size determines the maximum number of containers in the cluster because IP addresses in the VPC are directly allocated to containers. After a cluster is created, you can only add container subnets but cannot delete them. You are advised to set a larger IP address segment for the container subnet to prevent insufficient container IP addresses.
- Service CIDR Block: cannot overlap with the subnet CIDR block or the container CIDR block.

**Figure 9-3** CIDR block planning in the single-VPC single-cluster scenario (CCE Turbo cluster)

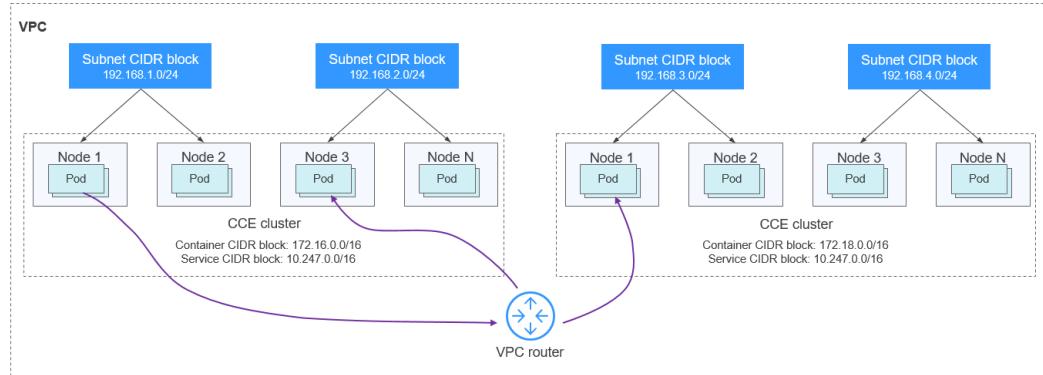


## Single-VPC Multi-Cluster Scenarios

### VPC network model

Pod packets are forwarded through VPC routes. CCE automatically configures a routing table on the VPC routes to each container CIDR block. The network scale is limited by the VPC route table. [Figure 9-4](#) shows the CIDR block planning of the cluster.

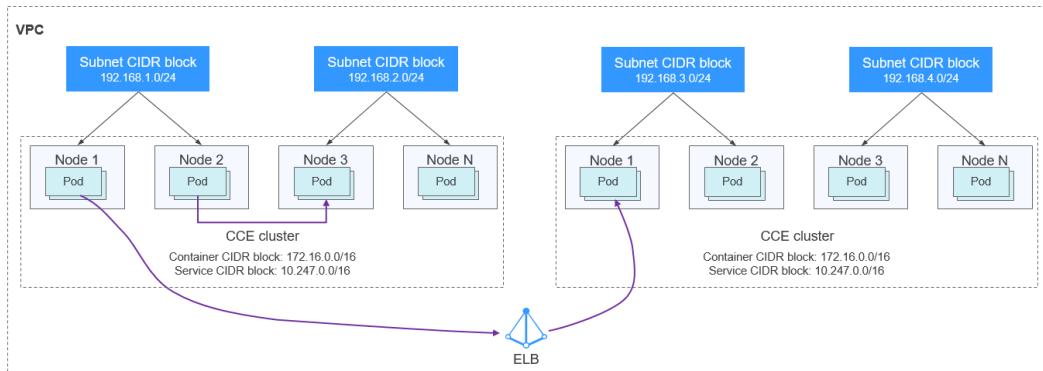
- **VPC CIDR Block:** specifies the VPC CIDR block where the cluster resides. The size of this CIDR block affects the maximum number of nodes that can be created in the cluster.
- **Subnet CIDR Block:** The subnet CIDR block in each cluster cannot overlap with the container CIDR block.
- **Container CIDR Block:** If multiple VPC network model clusters exist in a single VPC, the container CIDR blocks of all clusters cannot overlap because the clusters use the same routing table. In this case, CCE clusters are partially interconnected. A pod of a cluster can directly access the pods of another cluster, but cannot access the Services of the cluster.
- **Service CIDR Block:** can be used only in clusters. Therefore, the service CIDR blocks of different clusters can overlap, but cannot overlap with the subnet CIDR block and container CIDR block of the cluster to which the clusters belong.

**Figure 9-4** VPC network - multi-cluster scenario

## Tunnel Network

Though at some cost of performance, the tunnel encapsulation enables higher interoperability and compatibility with advanced features (such as network policy-based isolation), meeting the requirements of most applications. [Figure 9-5](#) shows the CIDR block planning of the cluster.

- VPC CIDR Block: specifies the VPC CIDR block where the cluster resides. The size of this CIDR block affects the maximum number of nodes that can be created in the cluster.
- Subnet CIDR Block: The subnet CIDR block in each cluster cannot overlap with the container CIDR block.
- Container CIDR Block: The container CIDR blocks of all clusters can overlap. In this case, pods in different clusters cannot be directly accessed using IP addresses. It is recommended that ELB be used for the cross-cluster access between containers.
- Service CIDR Block: can be used only in clusters. Therefore, the service CIDR blocks of different clusters can overlap, but cannot overlap with the subnet CIDR block and container CIDR block of the cluster to which the clusters belong.

**Figure 9-5** Tunnel network - multi-cluster scenario

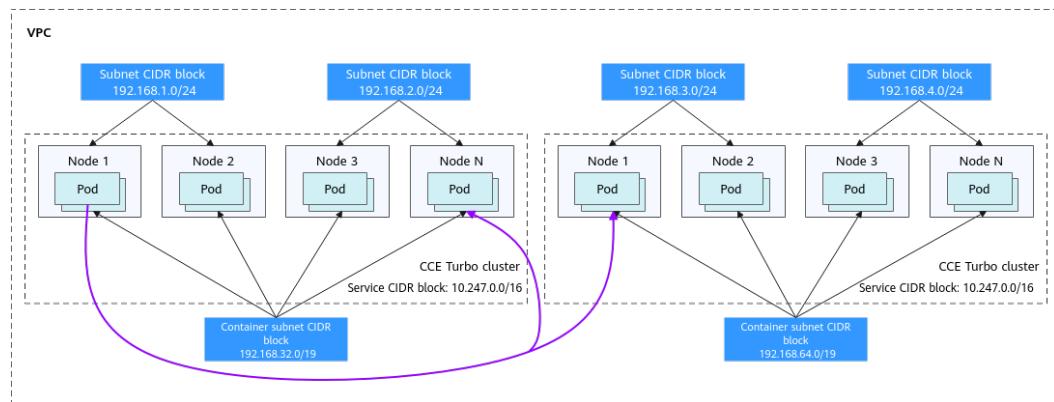
## Cloud native network 2.0 network model (CCE Turbo cluster)

In this mode, container IP addresses are allocated from the VPC CIDR block. ELB passthrough networking is supported to direct access requests to containers.

Security groups and multiple types of VPC networks can be bound to deliver high performance.

- VPC CIDR Block: specifies the VPC CIDR block where the cluster resides. In a CCE Turbo cluster, the CIDR block size affects the total number of nodes and containers that can be created in the cluster.
- Subnet CIDR Block: There is no special restriction on the subnet CIDR blocks in CCE Turbo clusters.
- Container Subnet: The CIDR block of the container subnet is included in the VPC CIDR block. Container subnets in different clusters can overlap with each other or overlap with the subnet CIDR block. However, you are advised to stagger the container CIDR blocks of different clusters and ensure that the container subnet CIDR blocks have sufficient IP addresses. In this case, pods in different clusters can directly access each other through IP addresses.
- Service CIDR Block: can be used only in clusters. Therefore, the service CIDR blocks of different clusters can overlap, but cannot overlap with the subnet CIDR block or container CIDR block.

**Figure 9-6** Cloud native network 2.0 network model - multi-cluster scenario



### Coexistence of Clusters in Multi-Network

When a VPC contains clusters created with different network models, comply with the following rules when creating a cluster:

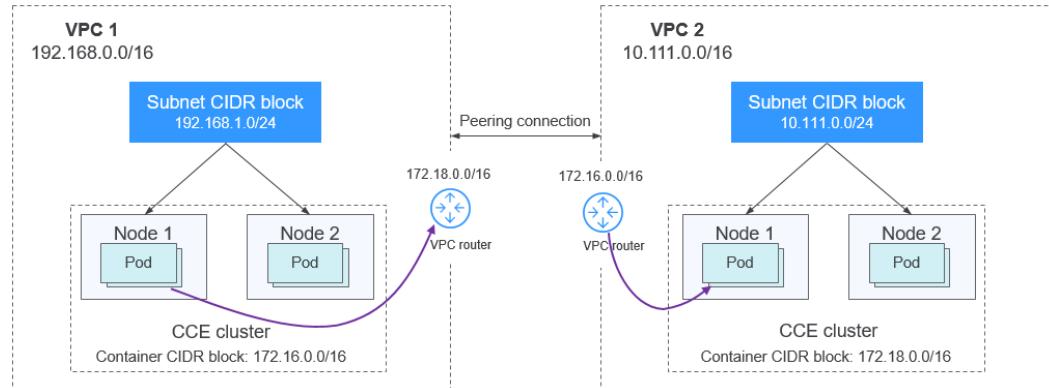
- VPC CIDR Block: In this scenario, all clusters are located in the same VPC CIDR block. Ensure that there are sufficient available IP addresses in the VPC.
- Subnet CIDR Block: Ensure that the subnet CIDR block does not overlap with the container CIDR block. Even in some scenarios (for example, coexistence with CCE Turbo clusters), the subnet CIDR block can overlap with the container (subnet) CIDR block. However, this is not recommended.
- Container CIDR Block: Ensure that the container CIDR blocks of clusters in **VPC network model** do not overlap.
- Service CIDR Block: The service CIDR blocks of all clusters can overlap, but cannot overlap with the subnet CIDR block and container CIDR block of the cluster.

## Cross-VPC Cluster Interconnection

When two VPC networks are interconnected, you can configure the packets to be sent to the peer VPC in the route table.

In the VPC network model, after creating a peering connection, you need to add routes for the peering connection to enable communication between the two VPCs.

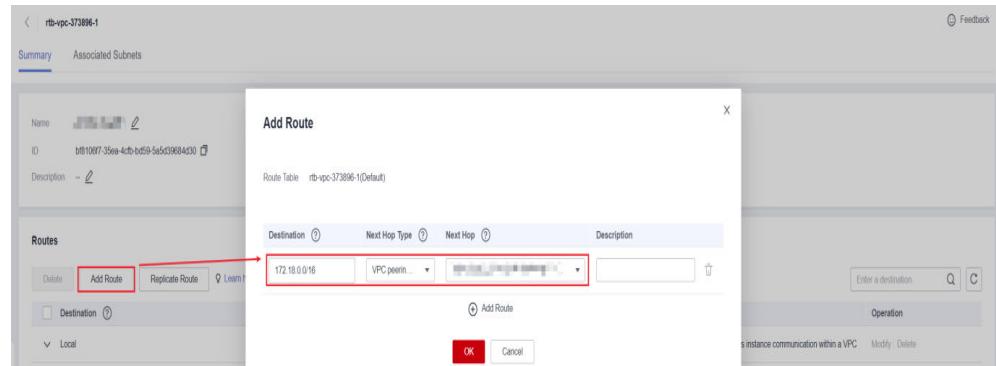
**Figure 9-7** VPC Network - VPC interconnection scenario



When creating a VPC peering connection between containers across VPCs, pay attention to the following points:

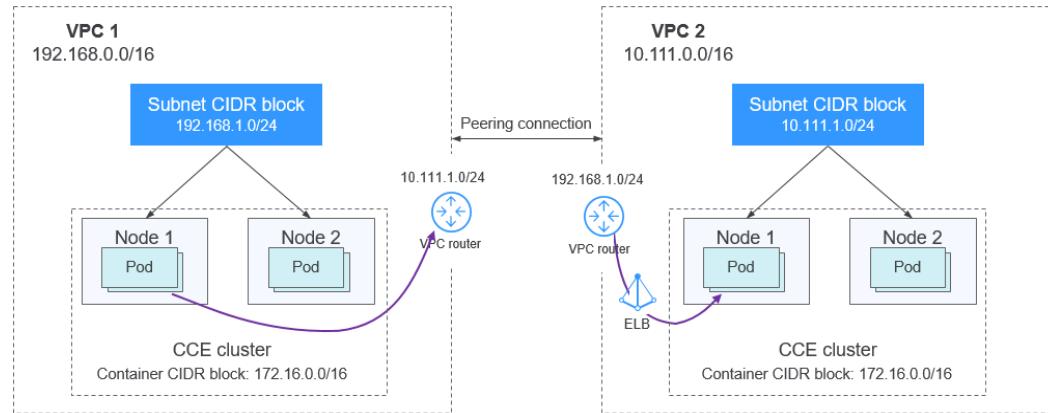
- The VPC to which the clusters belong must not overlap. In each cluster, the subnet CIDR block cannot overlap with the container CIDR block.
- The container CIDR blocks of clusters cannot overlap, but the Service CIDR blocks can.
- You need to add not only the peer VPC CIDR block but also the peer container CIDR block to the VPC routing tables at both ends. Note that this operation must be performed in the VPC route tables of the clusters.

**Figure 9-8** Adding the peer container CIDR block to the local route on the VPC console



In the tunnel network model, after creating a peering connection, you need to add routes for the peering connection to enable communication between the two VPCs.

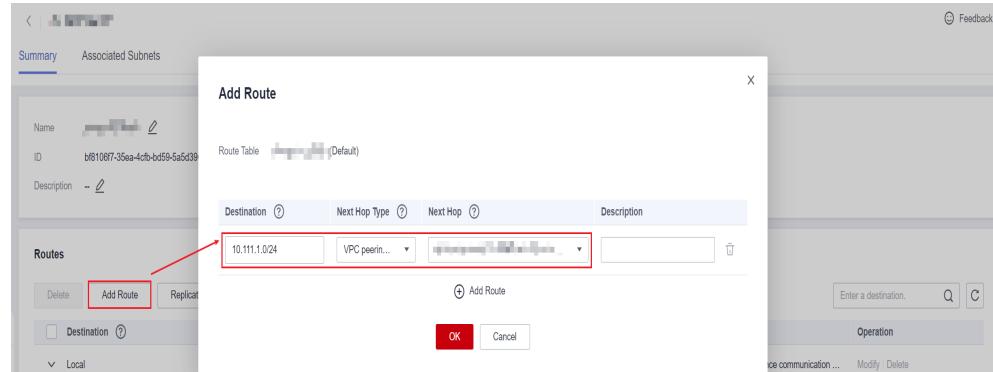
**Figure 9-9 Tunnel network - VPC interconnection scenario**



Pay attention to the following:

- The VPC of the clusters must not overlap.
- The container CIDR blocks of all clusters can overlap, so do the Service CIDR blocks.
- Add the peer subnet CIDR block to the route table of the VPC peering connection.

**Figure 9-10 Adding the subnet CIDR block of the peer cluster node to the local route on the VPC console**



In **Cloud Native Network 2.0** mode, after creating a VPC peering connection, you only need to add routes for the VPC peering connection to enable communication between the two VPCs. Ensure that the VPC of the clusters does not overlap.

## VPC-IDC Scenarios

Similar to the VPC interconnection scenario, some CIDR blocks in the VPC are routed to the IDC. The pod IP addresses of CCE clusters cannot overlap with the addresses within these CIDR blocks. To access the pod IP addresses in the cluster in the IDC, you need to configure the route table to the private line VBR on the IDC.

## 9.2 Selecting a Network Model

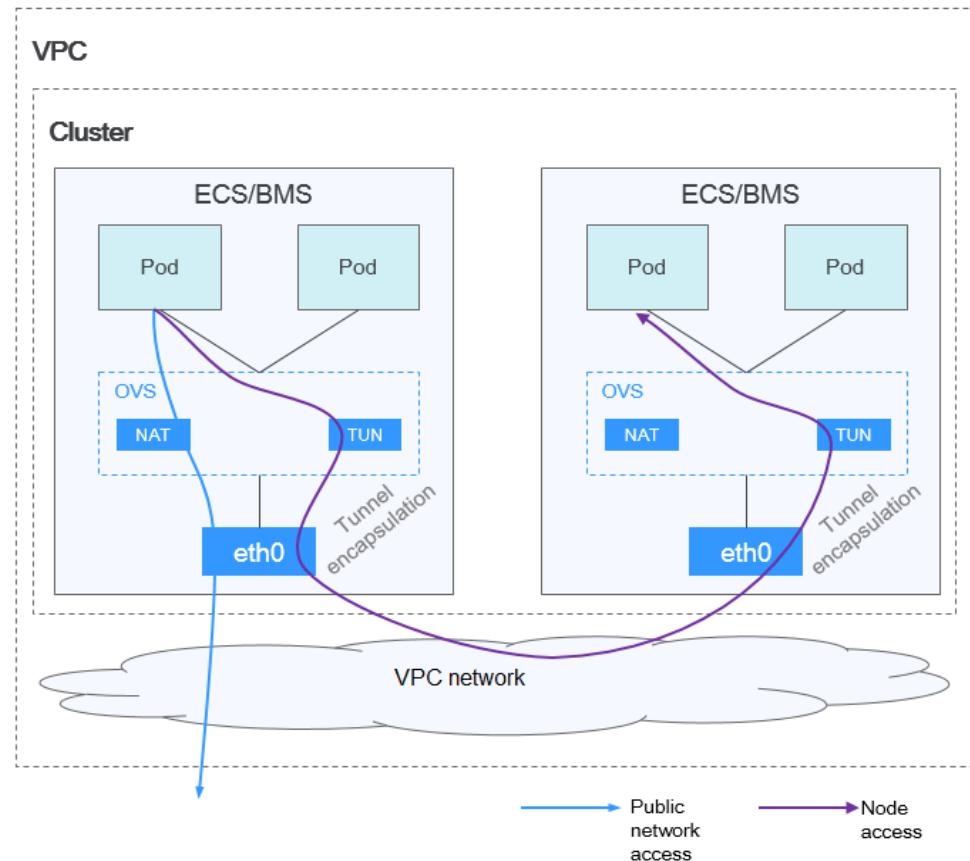
CCE uses self-proprietary, high-performance container networking add-ons to support the tunnel network, Cloud Native Network 2.0, and VPC network models.

**⚠ CAUTION**

After a cluster is created, the network model cannot be changed. Exercise caution when selecting a network model.

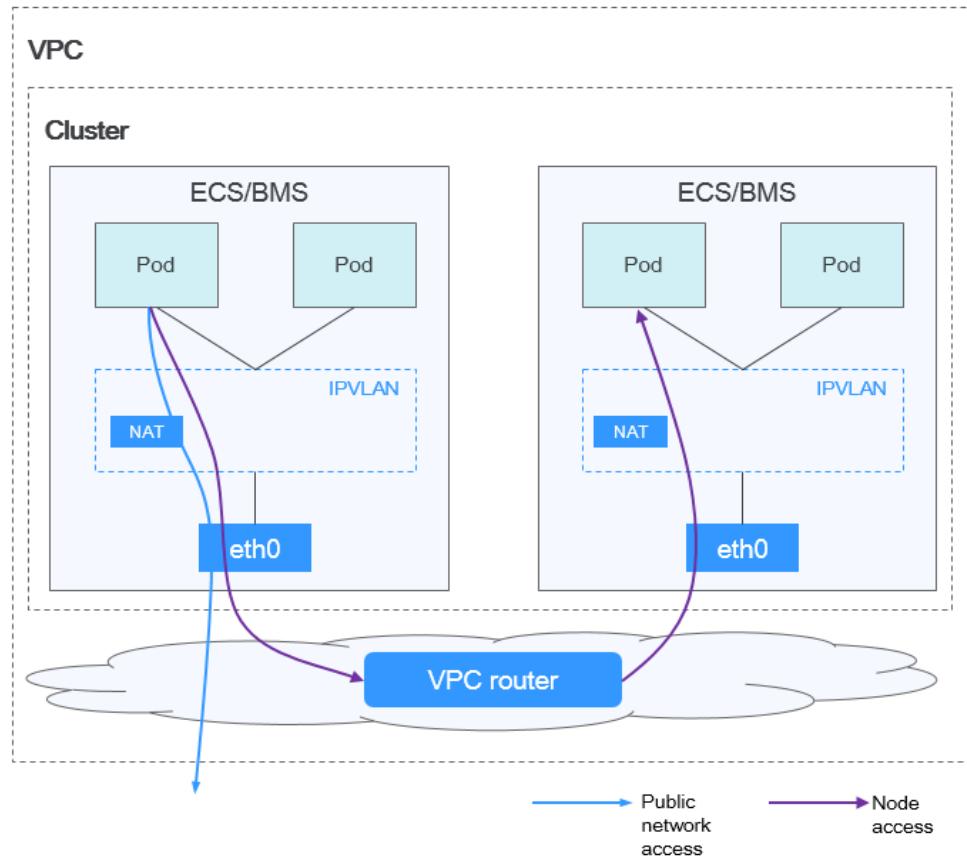
- **Tunnel network:** The container network is an overlay tunnel network on top of a VPC network and uses the VXLAN technology. This network model is applicable when there is no high requirements on performance. VXLAN encapsulates Ethernet packets as UDP packets for tunnel transmission. Though at some cost of performance, the tunnel encapsulation enables higher interoperability and compatibility with advanced features (such as network policy-based isolation), meeting the requirements of most applications.

**Figure 9-11** Container tunnel network



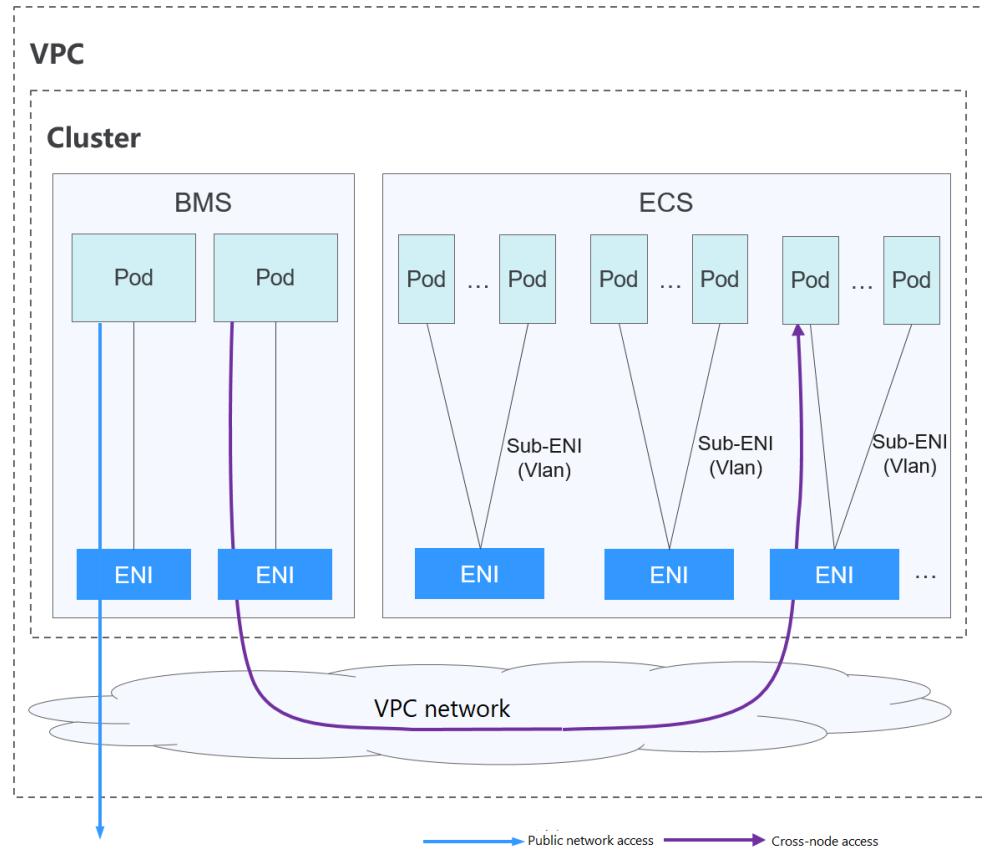
- **VPC network:** The container network uses VPC routing to integrate with the underlying network. This network model is applicable to performance-intensive scenarios. The maximum number of nodes allowed in a cluster depends on the route quota in a VPC network. Each node is assigned a CIDR block of a fixed size. VPC networks are free from tunnel encapsulation overhead and outperform container tunnel networks. In addition, as VPC routing includes routes to node IP addresses and container network segment, container pods in the cluster can be directly accessed from outside the cluster.

Figure 9-12 VPC network



- **Cloud Native Network 2.0:** The container network deeply integrates the elastic network interface (ENI) capability of VPC, uses the VPC CIDR block to allocate container addresses, and supports passthrough networking to containers through a load balancer.

**Figure 9-13 Cloud Native Network 2.0**



The following table lists the differences between the network models.

**Table 9-1** Networking model comparison

| Dimension               | Tunnel Network                           | VPC Network          | Cloud Native Network 2.0               |
|-------------------------|------------------------------------------|----------------------|----------------------------------------|
| Core technology         | OVS                                      | IPVlan and VPC route | VPC ENI/sub-ENI                        |
| Applicable Clusters     | CCE cluster                              | CCE cluster          | CCE Turbo cluster                      |
| Network isolation       | Kubernetes native NetworkPolicy for pods | No                   | Pods support security group isolation. |
| Passsthrough networking | No                                       | No                   | Yes                                    |

| Dimension             | Tunnel Network                                                                                                                                                                                                                              | VPC Network                                                                                                                                                                                                                       | Cloud Native Network 2.0                                                                              |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| IP address management | <ul style="list-style-type: none"> <li>The container CIDR block is allocated separately.</li> <li>CIDR blocks are divided by node and can be dynamically allocated (CIDR blocks can be dynamically added after being allocated.)</li> </ul> | <ul style="list-style-type: none"> <li>The container CIDR block is allocated separately.</li> <li>CIDR blocks are divided by node and statically allocated (the CIDR block cannot be changed after a node is created).</li> </ul> | The container CIDR block is divided from the VPC subnet and does not need to be allocated separately. |
| Performance           | Performance loss due to VXLAN encapsulation                                                                                                                                                                                                 | No tunnel encapsulation. Cross-node packets are forwarded through VPC routers, delivering performance equivalent to that of the host network.                                                                                     | The container network is integrated with the VPC network, eliminating performance loss.               |
| Networking scale      | A maximum of 2,000 nodes are supported.                                                                                                                                                                                                     | By default, 200 nodes are supported. Each time a node is added to the cluster, a route is added to the VPC routing table. Therefore, the cluster scale is limited by the VPC route table.                                         | A maximum of 2,000 nodes are supported.                                                               |

| Dimension | Tunnel Network                                                                                                                                                  | VPC Network                                                                                                                                                                                                                        | Cloud Native Network 2.0                                                                                                                                                                                                                         |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Scenario  | <ul style="list-style-type: none"><li>Common container services</li><li>Scenarios that do not have high requirements on network latency and bandwidth</li></ul> | <ul style="list-style-type: none"><li>Scenarios that have high requirements on network latency and bandwidth</li><li>Containers communicate with VMs using a microservice registration framework, such as Dubbo and CSE.</li></ul> | <ul style="list-style-type: none"><li>Scenarios that have high requirements on network latency, bandwidth, and performance</li><li>Containers communicate with VMs using a microservice registration framework, such as Dubbo and CSE.</li></ul> |

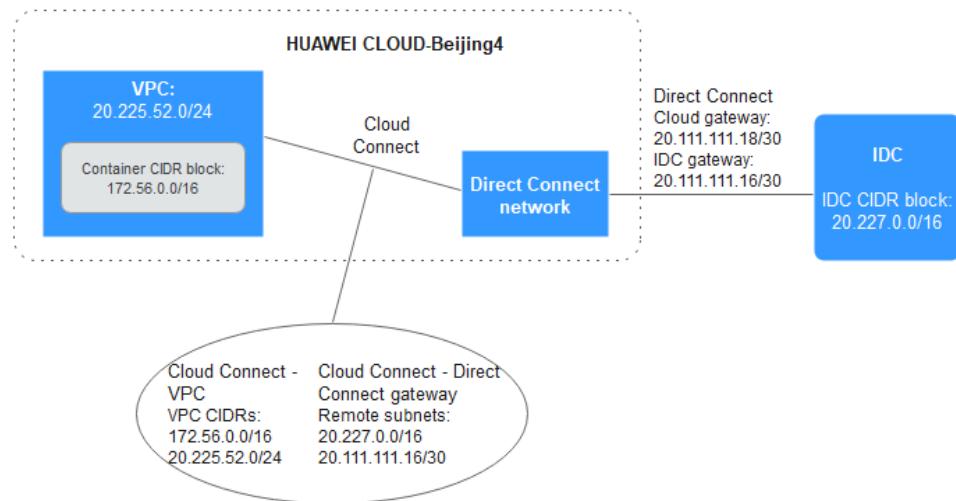
**NOTICE**

1. The scale of a cluster that uses the VPC network model is limited by the custom routes of the VPC. Therefore, you need to estimate the number of required nodes before creating a cluster.
2. By default, VPC routing network supports direct communication between containers and hosts in the same VPC. If a peering connection policy is configured between the VPC and another VPC, the containers can directly communicate with hosts on the peer VPC. In addition, in hybrid networking scenarios such as Direct Connect and VPN, communication between containers and hosts on the peer end can also be achieved with proper planning.

## 9.3 Allowing Containers and IDCs to Communicate with Each Other Through VPC and Direct Connect

### Scenario

By using VPC and Direct Connect, IP addresses in the container CIDR block (172.56.0.0/16) and IDC CIDR block (20.227.0.0/16) can communicate with each other in the VPC network model.

**Figure 9-14 Example network topology**

## Prerequisites

An IDC is available, and the Direct Connect service has been applied for.

## Procedure

### Step 1 Create a Direct Connect connection.

1. Log in to the management console, click in the upper left corner, and select the desired region and project. Click at the upper left corner and choose **Networking > Direct Connect** in the expanded list.
2. In the navigation pane on the left of the console, choose **Direct Connect > Connections**. On the displayed page, click **Create Connection**.
3. On the **Create Connection** page, click **Full Service Installation**.

The screenshot shows the "Create Connection" page with the "Full Service Installation" tab selected. The page includes the following sections:

- 1. Connection Requested**: Fields include "Equipment Room Address" (input field), "Available Carriers" (checkboxes for China Telecom, China Mobile, China Unicom, Any carrier, Other), "Port Type" (dropdown: GE single-mode optical port), and "Region" (dropdown: CN North-Beijing1).
- 2. Site Survey by HUAWEI CLOUD**: Fields include "Location" (dropdown: Beijing-Zhonglin, Langfang-Huawei Base) and "Port Type" (dropdown: TGE single-mode optical port).
- 3. Confirmation of Configuration and Payment Pending**: Fields include "Connection Name" (input: dc-4f54), "Leased Line Bandwidth (Mbps)" (dropdown: 1,000), "Billing Mode" (dropdown: Yearly/Monthly), "Required Duration" (dropdown: 1 year, 2 years, 3 years), and "Contact Person Name" and "Contact Number" (input fields).
- 4. Construction by HUAWEI CLOUD**: Fields include "Region" (dropdown: CN North-Beijing1), "Location" (dropdown: Beijing-Zhonglin, Langfang-Huawei Base), and "Port Type" (dropdown: TGE single-mode optical port).
- 5. Confirm Connection**: Fields include "Region" (dropdown: CN North-Beijing1), "Location" (dropdown: Beijing-Zhonglin, Langfang-Huawei Base), and "Port Type" (dropdown: TGE single-mode optical port).
- 6. Billing Confirmation**: Fields include "Region" (dropdown: CN North-Beijing1), "Location" (dropdown: Beijing-Zhonglin, Langfang-Huawei Base), and "Port Type" (dropdown: TGE single-mode optical port).

### Step 2 Create a virtual gateway.

Choose **Direct Connect > Virtual Gateways**, and click **Create Virtual Gateway** on the right. Add the VPC CIDR block and the container CIDR block in the VPC network model.

**Figure 9-15** Creating a virtual gateway

The screenshot shows the 'Create Virtual Gateway' dialog box. It includes fields for Name (vgw-183e), Enterprise Project (default), VPC (vpc-bcs-6i8y | 76ba953f-5543-490f-badf-aa6...), Local Subnet (a text area with placeholder text: 'Enter one or more subnets using CIDR notation. Separate each entry by a comma, for example, 192.168.52.0/24,192.168.54.0/24.'), BGP ASN (64512), and a Description text area (empty, 0/128 characters). At the bottom are OK and Cancel buttons.

Create Virtual Gateway

\* Name vgw-183e

\* Enterprise Project default C ⓘ Create Enterprise Project

\* VPC vpc-bcs-6i8y | 76ba953f-5543-490f-badf-aa6... C

\* Local Subnet ⓘ Enter one or more subnets using CIDR notation. Separate each entry by a comma, for example, 192.168.52.0/24,192.168.54.0/24.

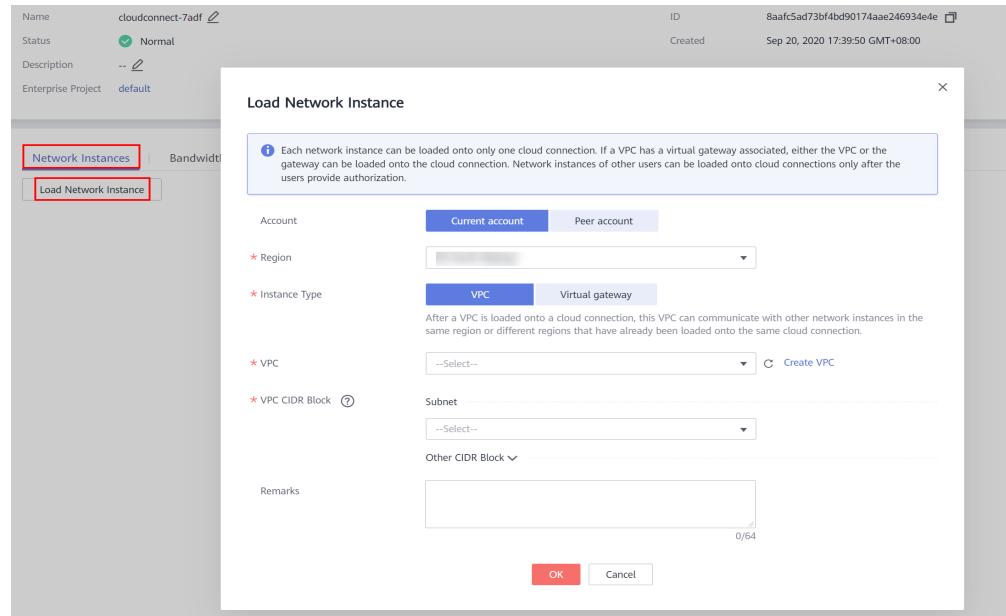
BGP ASN 64512

Description 0/128

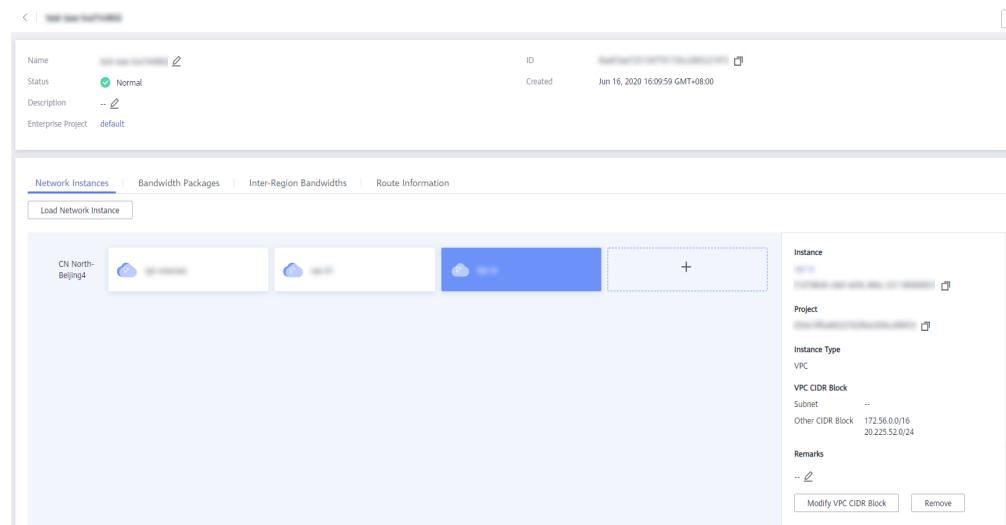
OK Cancel

### Step 3 Create a Cloud Connect connection.

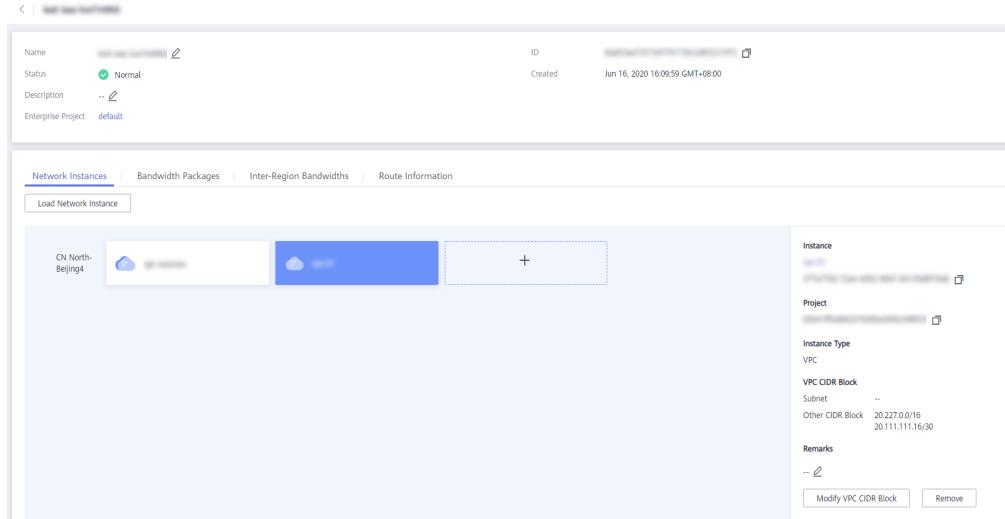
1. In the navigation pane on the left of the console, choose **Cloud Connect > Cloud Connections**. On the displayed page, click **Create Cloud Connection**.
2. After the connection is created, click the cloud connection name to go to its details page. On the **Network Instances** tab page, click **Load Network Instance** to add VPC information.



- Check the VPC CIDR blocks on the Cloud Connect VPC and ensure that the VPC and container CIDR blocks have been added.



- Add the VGW CIDR blocks on the Direct Connect gateway.
- Check the VGW CIDR blocks on the Direct Connect gateway and ensure that the remote subnets are correctly added.



**Step 4** Test the connectivity.

1. On an IDC host, traceroute the IP address of the container node or container on the cloud to check whether the route to the cloud gateway of Direct Connect is normal.
  - a. If the route is normal, Direct Connect has a return route.
  - b. If the route to the cloud gateway of Direct Connect is abnormal, check whether the route settings at both ends of Direct Connect are correct.
2. If the IP address cannot be tracerouted, try the ping or telnet operation. Before pinging the address, ensure that the ICMP policy has been enabled for the security group if the target is an ECS.

----End

## 9.4 Enabling a CCE Cluster to Resolve Domain Names on Both On-Premises IDCs and HUAWEI CLOUD

### 9.4.1 Solution Overview

#### Issues

Microservices are increasingly used to deploy applications. When microservices access each other, they need to resolve domain names.

When you have on-premises IDCs with internal domain names configured, and you have deployed containerized applications on both these IDCs and cloud, you need to enable the containers and nodes in CCE clusters to resolve domain names of both the IDC and cloud.

Suppose you have reconstructed one of your applications using microservices. You run the application management backend in a CCE cluster, deploy the content moderation service in the on-premises IDC, and use the image recognition service on Huawei Cloud. The VPC where CCE resides is connected to the IDC through a private line. **Figure 9-16** shows the deployment architecture.

When a user accesses this application, the following interaction is involved between different microservices:

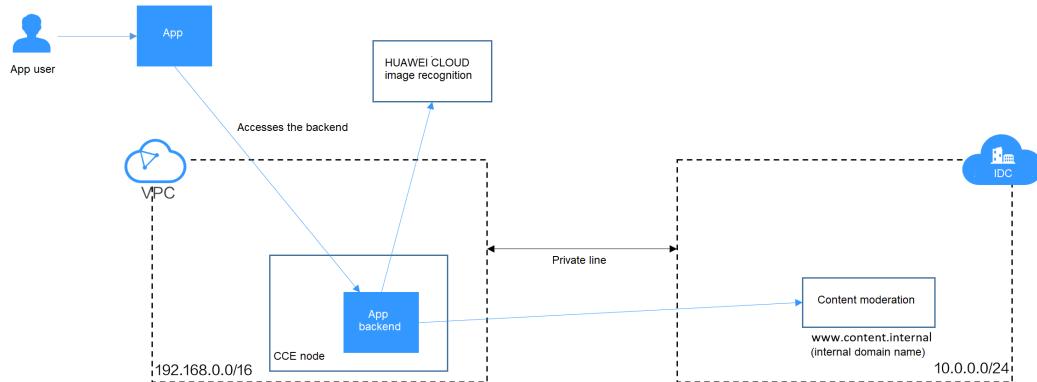
- The CCE cluster uses the Huawei Cloud DNS server, by default, to access the image recognition service.
- The CCE cluster uses the internal DNS server of the IDC to access the content moderation service deployed in the IDC.

In this case, the CCE cluster must be able to use both the Huawei Cloud DNS server and the internal DNS server of the IDC. If the DNS server on the CCE node points to the that of the IDC, the domain name of Huawei Cloud cannot be resolved. If the IP address of the IDC internal domain name is added to the **hosts** file, the configuration of the CCE node needs to be updated in real time when the IDC internal service IP changes. This is difficult to implement and may cause the CCE node to be unavailable.

#### NOTE

The content moderation and image recognition services are used only as examples.

**Figure 9-16 Application deployment architecture**



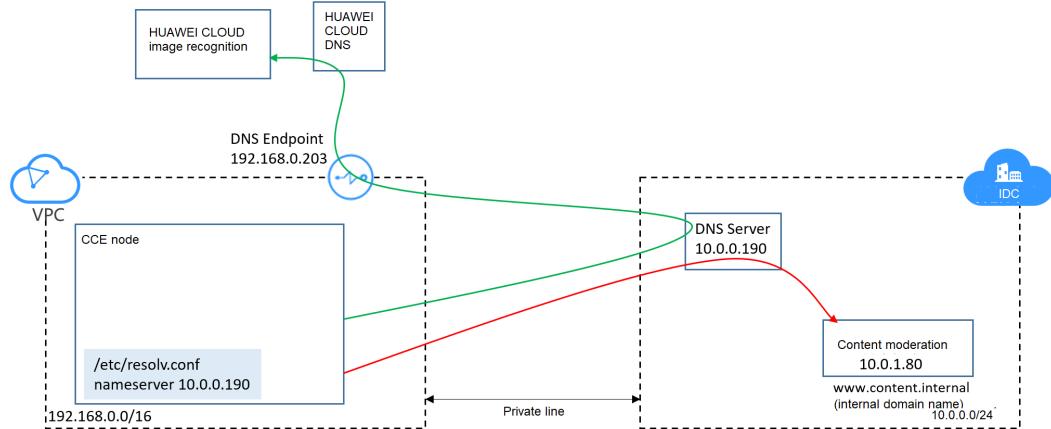
This section provides a solution for CCE clusters to resolve domain names of both on-premises IDCs and Huawei Cloud.

## Solution 1: Using the DNS Endpoint for Cascading Resolution

You can use the VPC endpoint service to create a DNS endpoint cascaded with the IDC DNS server, so that nodes and containers in the CCE cluster can use the IDC DNS server for domain name resolution.

- If the Huawei Cloud domain name needs to be resolved, the request is forwarded to the DNS endpoint, and the Huawei Cloud DNS server is used to resolve the address and return the result.
- If the IDC domain name needs to be resolved, the IDC DNS server directly resolves the address and returns the result.

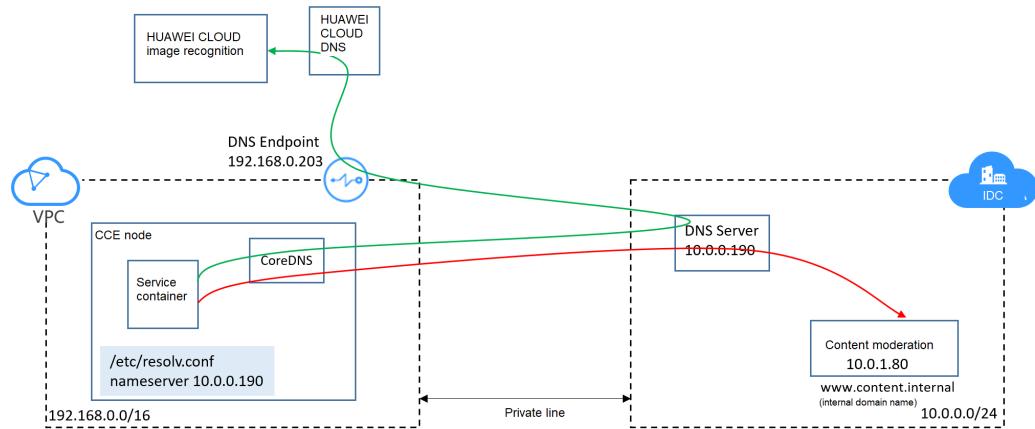
**Figure 9-17** Accessing both the Huawei Cloud domain name and external domain name (for nodes)



For domain name resolution in a container, you can set the DNS policy to **ClusterFirst** when creating a pod. In this way, the domain name resolution requests of the container are directly sent to CoreDNS.

- If a cluster-internal domain name needs to be resolved, CoreDNS directly returns the resolution result.
- If an external domain name needs to be resolved, CoreDNS forwards the request to the IDC DNS server for resolution.

**Figure 9-18** Accessing both the Huawei Cloud domain name and external domain name (for containers)



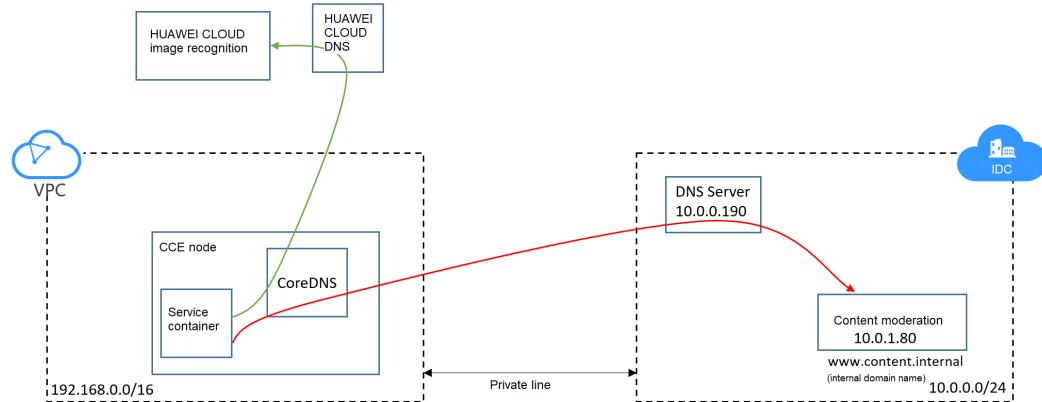
## Solution 2: Changing the CoreDNS Configurations

Set the DNS policy to **ClusterFirst** when creating a pod so that the domain name resolution requests of containers are directly sent to CoreDNS.

- If a cluster-internal domain name needs to be resolved, CoreDNS directly returns the resolution result.
- If an external domain name needs to be resolved, CoreDNS forwards the request to the IDC DNS server for resolution.

- If a container accesses a Huawei Cloud internal domain name, the domain name is resolved by the internal DNS server of Huawei Cloud.

**Figure 9-19** Domain name resolution in solution 2



## Solution Comparison

| Solution                                        | Advantage                                                                                                                                                | Disadvantage                                                                                                                                                                                                                         |
|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Using the DNS endpoint for cascading resolution | External domain names can be resolved for containers and nodes in a CCE cluster.                                                                         | An external DNS server is required to forward the requests for resolving internal domain names of Huawei Cloud, resulting in performance loss.                                                                                       |
| Changing the CoreDNS configuration              | No external DNS server is required to forward the requests for resolving internal domain names of Huawei Cloud. Therefore, there is no performance loss. | <ul style="list-style-type: none"> <li>External domain names cannot be resolved on CCE cluster nodes.</li> <li>The configuration will be lost if CoreDNS is upgraded or rolled back, and you need to reconfigure CoreDNS.</li> </ul> |

### 9.4.2 Solution 1: Using a DNS Endpoint for Cascading Resolution

#### Prerequisites

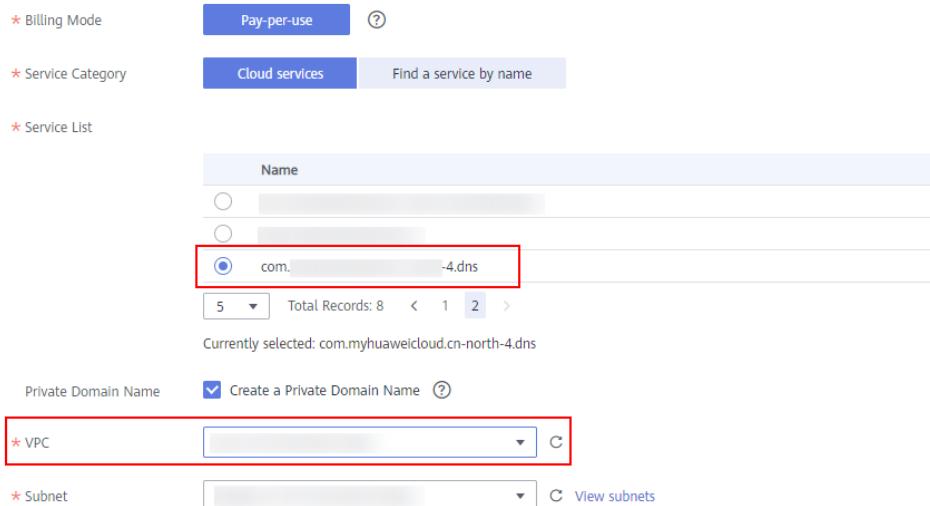
The VPC where the CCE cluster is deployed has been connected to the on-premises IDC through a private line (Direct Connect) or other channels. The IDC can access the IP addresses in the VPC CIDR block and CCE cluster container CIDR block. For details about how to create a Direct Connect connection, see [Getting Started with Direct Connect](#).

## Procedure

**Step 1** Create a DNS endpoint in the VPC where the CCE cluster is deployed.

1. Access the [VPC Endpoint](#) page on the network console.
2. Click **Buy VPC Endpoint** in the upper right corner.
3. Select the DNS service and VPC where the CCE cluster is deployed.

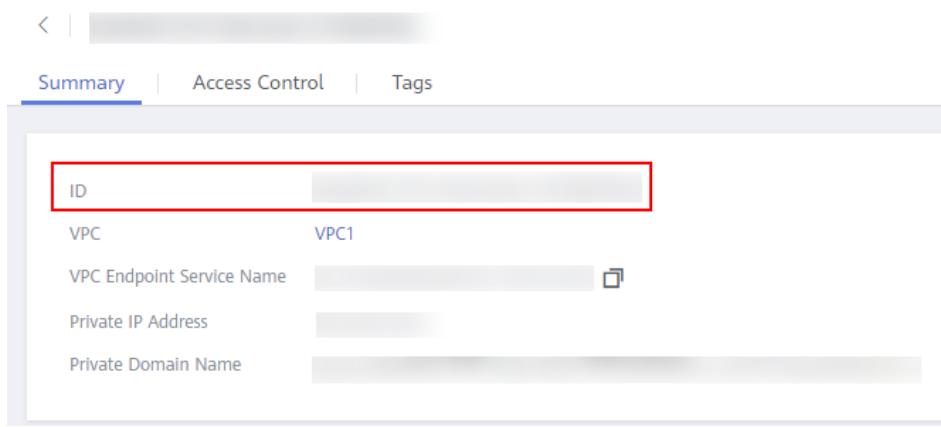
**Figure 9-20** Creating a DNS endpoint



4. Click **Next** and then **Submit**.

After the creation is complete, you can view the IP address of the DNS endpoint on its details page.

**Figure 9-21** IP address of the DNS endpoint



**Step 2** Configure cascading on the IDC DNS server.

### NOTE

The configuration varies depending on the DNS server. The following configurations are used only as example.

**BIND** (a commonly used DNS server software) is used for the following demonstration.

In this step, you configure the DNS server to forward the requests of resolving Huawei Cloud internal domain names to the DNS endpoint created in the previous step.

For example, in BIND, you can add the lines marked in red to the **/etc/named.conf** file. **192.168.0.203** is the IP address of the DNS endpoint created in

### Step 1.

```
options {  
    listen-on port 53 { any; };  
    listen-on-v6 port 53 { ::1; };  
    directory    "/var/named";  
    dump-file    "/var/named/data/cache_dump.db";  
    statistics-file "/var/named/data/named_stats.txt";  
    memstatistics-file "/var/named/data/named_mem_stats.txt";  
    recursing-file "/var/named/data/named.recurse";  
    secroots-file "/var/named/data/named.secroots";  
    allow-query   { any; };  
  
    forward first;  
    forwarders { 192.168.0.203; };  
  
};  
....
```

### Step 3 Change the DNS configuration of the node in the CCE cluster.

You can use either of the following methods to change the settings.

#### Method 1:

Change the settings after the node is created.

1. Log in to the worker node of the CCE cluster.
2. In the **/etc/resolv.conf** file, change the value of nameserver to the IP address of the IDC DNS server.  
`# vi /etc/resolv.conf  
nameserver 10.0.0.190`
3. Run the following command to lock the **resolv.conf** file to prevent it from being automatically updated by Huawei Cloud.  
`chattr +i /etc/resolv.conf`

For details about how to configure DNS, see [Configuring DNS](#).

#### Method 2:

Change the DNS settings of the VPC subnet where the CCE cluster resides. In this way, the IP address of the specified DNS server is used in the **/etc/resolv.conf** file for newly created worker nodes.

Before using this method, ensure that the node can use the IDC DNS server to resolve the Huawei Cloud internal domain name. Otherwise, the node cannot be created. You are advised to commission the DNS server before you change the DNS settings of the VPC subnet.

**Figure 9-22 Subnet DNS settings**



#### Step 4 Configure the workload DNS policy.

When creating a workload, you can set **dnsPolicy** to **ClusterFirst** in the YAML file for domain name resolution in containers. This is also the default configuration in Kubernetes. For details about how to configure DNS for workloads, see [DNS Configuration](#).

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
    - name: test
      image: nginx
  dnsPolicy: ClusterFirst
```

----End

## Verification

After the configuration is complete, run the **dig** command on the cluster node. The command output shows that the IP address of the server is **10.0.0.190**, which indicates that the domain name is resolved by the IDC DNS server.

```
# dig cce.ap-southeast-1.myhuaweicloud.com
; <>> DiG 9.9.4-61.1.h14.eulerosv2r7 <>> cce.ap-southeast-1.myhuaweicloud.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24272
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: udp: 4096
;; QUESTION SECTION:
;cce.ap-southeast-1.myhuaweicloud.com. IN A

;; ANSWER SECTION:
cce.ap-southeast-1.myhuaweicloud.com. 274 IN A 100.125.4.16

;; Query time: 4 msec
;; SERVER: 10.0.0.190#53(10.0.0.190)
;; WHEN: Tue Feb 23 19:16:08 CST 2021
;; MSG SIZE rcvd: 76
```

Access a domain name of Huawei Cloud from the cluster node. The following output shows that the domain name is resolved to the corresponding IP address.

```
# ping cce.ap-southeast-1.myhuaweicloud.com
PING cce.ap-southeast-1.myhuaweicloud.com (100.125.4.16) 56(84) bytes of data.
```

Create a pod to access the Huawei Cloud domain name. The following output shows that the domain name can be resolved.

```
# kubectl run -i --tty --image tutum/dnsutils dnsutils --restart=Never --rm /bin/sh
If you do not see a command prompt, try pressing Enter.
# ping cce.ap-southeast-1.myhuaweicloud.com
PING cce.ap-southeast-1.myhuaweicloud.com (100.125.4.16) 56(84) bytes of data.
```

## 9.4.3 Solution 2: Changing the CoreDNS Configurations

### Prerequisites

The VPC where the CCE cluster is deployed has been connected to the on-premises IDC through a private line (Direct Connect) or other channels. The IDC can access the IP addresses in the VPC CIDR block and CCE cluster container CIDR block. For details about how to create a Direct Connect connection, see [Getting Started with Direct Connect](#).

### Procedure

CoreDNS configurations are stored in the ConfigMap named **coredns**. You can find this ConfigMap in the kube-system namespace. Run the following command to view the default configurations.

```
kubectl get configmap coredns -n kube-system -oyaml
```

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: coredns
  namespace: kube-system
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: d54ed5df-f4a0-48ec-9bc0-3efc1ac76af0
  resourceVersion: '21789515'
  creationTimestamp: '2021-03-02T09:21:55Z'
  labels:
    app: coredns
    k8s-app: coredns
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: CoreDNS
    release: cceaddon-coredns
data:
  Corefile: |-
```

..5353 {  
 bind {\$POD\_IP}  
 cache 30  
 errors  
 health {\$POD\_IP}:8080  
 kubernetes cluster.local in-addr.arpa ip6.arpa {  
 pods insecure  
 upstream /etc/resolv.conf  
 fallthrough in-addr.arpa ip6.arpa  
 }  
 loadbalance round\_robin  
 prometheus {\$POD\_IP}:9153  
 forward . /etc/resolv.conf  
 reload  
}

The preceding example shows that all CoreDNS configurations are defined in **Corefile**. By default, resolution requests of any domain name that does not belong to the Kubernetes cluster are directed to the DNS server specified by **forward**. In **forward . /etc/resolv.conf**, the first period (.) indicates all domain names, and **/etc/resolv.conf** indicates the DNS server of the node.

If a specific external domain name needs to be resolved, you can add an extra configuration item. For example, if you want to forward the requests of resolving the domain name **content.internal** to the DNS server whose IP address is 10.0.0.190, run the following command to add configurations in **Corefile**.

### kubectl edit configmap coredns -n kube-system

```
apiVersion: v1
data:
  Corefile: |->
    :5353 {
      bind {$POD_IP}
      cache 30
      errors
      health {$POD_IP}:8080
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        upstream /etc/resolv.conf
        fallthrough in-addr.arpa ip6.arpa
      }
      loadbalance round_robin
      prometheus {$POD_IP}:9153
      forward . /etc/resolv.conf
      reload
    }
    content.internal:5353 {
      bind {$POD_IP}
      errors
      cache 30
      forward . 10.0.0.190
    }
}
```

For details about other CoreDNS configurations, see [Customizing DNS Service](#).

You can also add configurations on the CCE console, as shown in [Figure 9-23](#).

**Figure 9-23** Changing CoreDNS configurations on the console

The screenshot shows the CoreDNS configuration page in the CCE console. At the top, there's a summary section with details like Cluster Name (iaas-cce-w00401781), Version (1.17.4), Status (Available), Installed (Apr 09, 2021), and Latest Event (Deployed success). Below this, there are two tabs: 'Resources' and 'Parameters'. The 'Parameters' tab is active and contains an 'Edit' button, which is highlighted with a red box. Under the 'Parameters' tab, there's a 'Specifications' section showing Instances (2) and a table for Container (coredns) with CPU Quota (Apply 500m Limit 500m) and Memory Quota (Apply 512Mi Limit 512Mi).

## Verification

Create a pod to access the IDC domain name. The following output shows that the domain name can be resolved.

```
web-terminal-568c6566df-c9jhl:~# kubectl run -i --tty --image tutum/dnsutils dnsutils --restart=Never --rm /bin/sh
If you don't see a command prompt, try pressing enter.
# ping www.content.internal
PING www.content.internal (10.0.1.80) 56(84) bytes of data.
64 bytes from 10.0.1.80: icmp_seq=1 ttl=64 time=1.08 ms
64 bytes from 10.0.1.80: icmp_seq=2 ttl=64 time=0.337 ms
```

Access a Huawei Cloud domain name. The following output shows that the domain name can be resolved.

```
# ping cce.ap-southeast-1.myhuaweicloud.com
PING cce.ap-southeast-1.myhuaweicloud.com (100.125.4.16) 56(84) bytes of data.
```

Access the IDC domain name on the cluster node. The domain name cannot be pinged, indicating that the CoreDNS configurations do not affect the domain name resolution of the node.

## 9.5 Implementing Sticky Session Through Load Balancing

### Concepts

Session persistence is one of the most common while complex problems in load balancing.

Session persistence is also called sticky sessions. After the sticky session function is enabled, requests from the same client are distributed to the same backend ECS by the load balancer for better continuity.

In load balancing and sticky session, connection and session are two key concepts. When only load balancing is concerned, session and connection refer to the same thing.

Simply put, if a user needs to log in, it can be regarded as a session; otherwise, a connection.

The sticky session mechanism fundamentally conflicts with the basic functions of load balancing. A load balancer forwards requests from clients to multiple backend servers to avoid overload on a single server. However, sticky session requires that some requests be forwarded to the same server for processing. Therefore, you need to select a proper sticky session mechanism based on the application environment.

### Layer-4 Load Balancing (Service)

In layer-4 load balancing, source IP address-based sticky session (Hash routing based on the client IP address) can be enabled. To enable source IP address-based sticky session on Services, the following conditions must be met:

1. **Service Affinity** of the Service is set to **Node level** (that is, the value of the **externalTrafficPolicy** field of the Service is **Local**).
2. Enable the source IP address-based sticky session in the load balancing configuration of the Service.

```
apiVersion: v1
kind: Service
metadata:
  name: svc-example
  namespace: default
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.id: 56dcc1b4-8810-480c-940a-a44f7736f0dc
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN
    kubernetes.io/elb.session-affinity-mode: SOURCE_IP
spec:
  selector:
    app: nginx
  externalTrafficPolicy: Local
```

```
ports:  
  - name: cce-service-0  
    targetPort: 80  
    nodePort: 32633  
    port: 80  
    protocol: TCP  
    type: LoadBalancer
```

3. Anti-affinity is enabled for the backend application corresponding to the Service.

## Layer-7 Load Balancing (Ingress)

In layer-7 load balancing, sticky session based on HTTP cookies and app cookies can be enabled. To enable such sticky session, the following conditions must be met:

1. The application (workload) corresponding to the ingress is enabled with workload anti-affinity.
2. Node affinity is enabled for the Service corresponding to the ingress.

### Procedure

#### Step 1 Create a Nginx workload.

Set the number of pods to 3 and set the podAntiAffinity.

```
kind: Deployment  
apiVersion: apps/v1  
metadata:  
  name: nginx  
  namespace: default  
spec:  
  replicas: 3  
  selector:  
    matchLabels:  
      app: nginx  
  template:  
    metadata:  
      labels:  
        app: nginx  
    spec:  
      containers:  
        - name: container-0  
          image: 'nginx:perl'  
          resources:  
            limits:  
              cpu: 250m  
              memory: 512Mi  
            requests:  
              cpu: 250m  
              memory: 512Mi  
      imagePullSecrets:  
        - name: default-secret  
affinity:  
  podAntiAffinity: # Pod anti-affinity.  
    requiredDuringSchedulingIgnoredDuringExecution:  
      - labelSelector:  
          matchExpressions:  
            - key: app  
              operator: In  
            values:  
              - nginx  
    topologyKey: kubernetes.io/hostname
```

#### Step 2 Creating a NodePort Service

Configure the sticky session in a Service. An ingress can connect to multiple Services, and each Service can have different sticky sessions.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  annotations:
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN
    kubernetes.io/elb.session-affinity-mode: HTTP_COOKIE      # HTTP cookie type.
    kubernetes.io/elb.session-affinity-option: '{"persistence_timeout":"1440"}' # Session stickiness duration,
in minutes. The value ranges from 1 to 1440.
spec:
  selector:
    app: nginx
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 32633      # Node port number.
      type: NodePort
    externalTrafficPolicy: Local # Node-level forwarding.
```

You can also select **APP\_COOKIE**.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  annotations:
    kubernetes.io/elb.lb-algorithm: ROUND_ROBIN
    kubernetes.io/elb.session-affinity-mode: APP_COOKIE      # Select APP_COOKIE.
    kubernetes.io/elb.session-affinity-option: '{"app_cookie_name":"test"}' # Application cookie name.
...
...
```

**Step 3** Create an ingress and associate it with a Service. The following example describes how to automatically create a shared load balancer. For details about how to specify other types of load balancers, see [Using kubectl to Create an ELB Ingress](#).

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  namespace: default
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.port: '80'
    kubernetes.io/elb.autocreate:
      {
        "type": "public",
        "bandwidth_name": "cce-bandwidth-test",
        "bandwidth_chargemode": "traffic",
        "bandwidth_size": 1,
        "bandwidth_sharetype": "PER",
        "eip_type": "5_bgp"
      }
spec:
  rules:
    - host: 'www.example.com'
      http:
        paths:
          - path: '/'
            backend:
              service:
                name: nginx      #Service name
                port:
```

```
number: 80
property:
  ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
  pathType: ImplementationSpecific
  ingressClassName: cce
```

- Step 4** Log in to the ELB console, access the load balancer details page, and check whether the sticky session feature is enabled.

**Figure 9-24** Enabling the sticky session feature

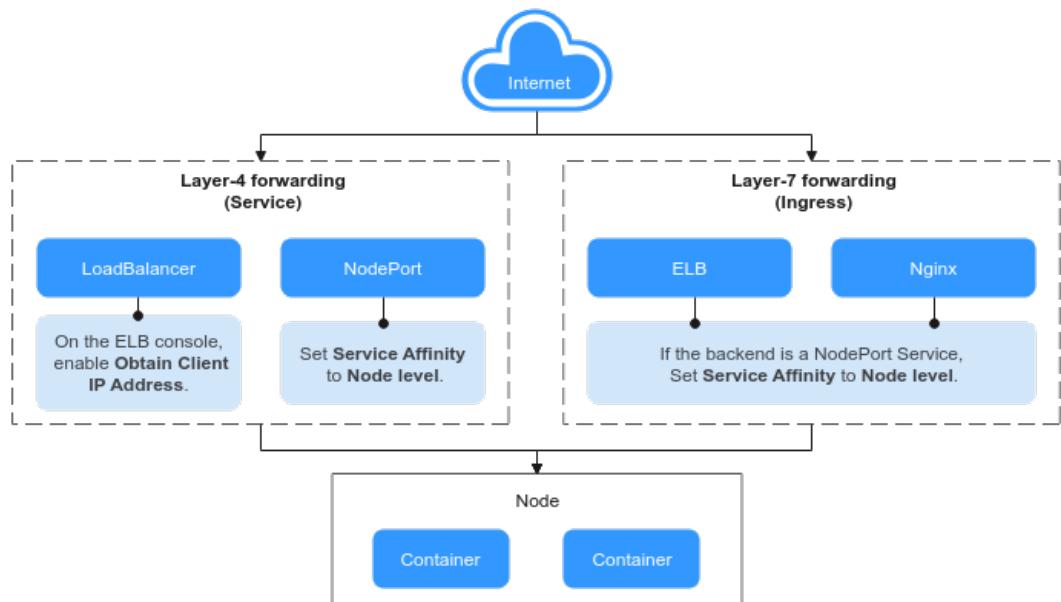
----End

## 9.6 Obtaining the Client Source IP Address for a Container

### Background

There may be different types of proxy servers between a client and a container server. How can a container obtain the real source IP address of the client? This section describes several scenarios you may encounter.

### Principles



### Layer-7 forwarding:

Ingress: If this access mode is used, the client source IP address is saved in the **X-Forwarded-For** HTTP header field by default. No other configuration is required.

- ELB ingress: A self-developed ingress to implement layer-7 network access between the internet and intranet (in the same VPC) based on ELB. If the backend Service type is **NodePort**, set **Service Affinity** to **Node** level.
- Nginx ingress: An ingress that implements layer-7 networking based on the nginx-ingress add-on. The backend Service type can be either **ClusterIP** or **NodePort**. If the backend Service type is **NodePort**, set **Service Affinity** to **Node** level.

### Layer-4 forwarding:

- LoadBalancer: Use ELB to achieve load balancing. You can manually enable the **Obtain Client IP Address** option for TCP and UDP listeners of shared load balancers. By default, the **Obtain Client IP Address** option is enabled for TCP and UDP listeners of dedicated load balancers. You do not need to manually enable it.
- NodePort: In this access mode, the container port is mapped to the node port. If cluster-level affinity is configured, access requests will be forwarded through the node and the client source IP address cannot be obtained. If node-level affinity is configured, access requests are not forwarded and the client source IP address can be obtained.

#### NOTE

If Istio is used, you can obtain the source IP address by referring to [How Do I Obtain the IP Address of a Client If I Use Istio for My Services](#).

## Scenarios in Which Source IP Address Can Be Obtained

Due to network model differences, CCE does not support obtaining source IP addresses in some scenarios, as listed in [Table 9-2](#). “-” in the table indicates that this scenario does not exist.

**Table 9-2** Scenarios in which source IP addresses can be obtained

| Level-1 Category             | Level-2 Category                                     | Load Balancer Type | VPC and Container Tunnel Network Models | Cloud Native Network 2.0 Model | Reference |
|------------------------------|------------------------------------------------------|--------------------|-----------------------------------------|--------------------------------|-----------|
| Layer-7 forwarding (ingress) | ELB                                                  | Shared             | Supported                               | Supported                      | Ingress   |
|                              |                                                      | Dedicated          | Supported                               | Supported                      |           |
|                              | Nginx (interconnected with the nginx-ingress add-on) | Shared             | Supported                               | Not supported                  |           |
|                              |                                                      | Dedicated          | Supported                               | Supported                      |           |

| Level-1 Category             | Level-2 Category | Load Balancer Type | VPC and Container Tunnel Network Models | Cloud Native Network 2.0 Model          | Reference                    |
|------------------------------|------------------|--------------------|-----------------------------------------|-----------------------------------------|------------------------------|
| Layer-4 forwarding (Service) | LoadBalancer     | Shared             | Supported                               | Not supported                           | <a href="#">LoadBalancer</a> |
|                              |                  | Dedicated          | Supported                               | Supported (only for ENI load balancing) |                              |
|                              | NodePort         | -                  | Supported                               | Not supported                           | <a href="#">NodePort</a>     |

## Ingress

Configure the application server and obtain the IP address of a client from the HTTP header.

The real IP address is placed in the **X-Forwarded-For** HTTP header field by the load balancer in the following format:

X-Forwarded-For: *IP address of the client,Proxy server 1-IP address,Proxy server 2-IP address,...*

If you use this method, the first IP address obtained is the IP address of the client.

For details, see [How Can I Obtain the IP Address of a Client?](#)

### NOTE

- In the Cloud Native Network 2.0 model, source IP addresses cannot be obtained if a shared load balancer is used when an ingress is interconnected with the nginx-ingress add-on. For details, see [Scenarios in Which Source IP Address Can Be Obtained](#). To obtain the source IP, uninstall the nginx-ingress add-on and use a dedicated load balancer during reinstallation.
- When adding an ingress, if the backend service is of the NodePort type, set **Service Affinity** to **Node level**, that is, set **spec.externalTrafficPolicy** to **Local**. For details, see [NodePort](#).

## LoadBalancer

For a LoadBalancer Service, different types of clusters obtain source IP addresses in different scenarios. In some scenarios, source IP addresses cannot be obtained currently. For details, see [Scenarios in Which Source IP Address Can Be Obtained](#).

- CCE clusters (using the VPC or container tunnel network model): Source IP addresses can be obtained when either a shared or dedicated load balancer is used.
- CCE Turbo clusters (using the Cloud Native Network 2.0 model): Source IP addresses can be obtained only when a dedicated load balancer (ENI LoadBalancer) is used.

## VPC and Container Tunnel Network Models

To obtain source IP addresses, perform the following steps:

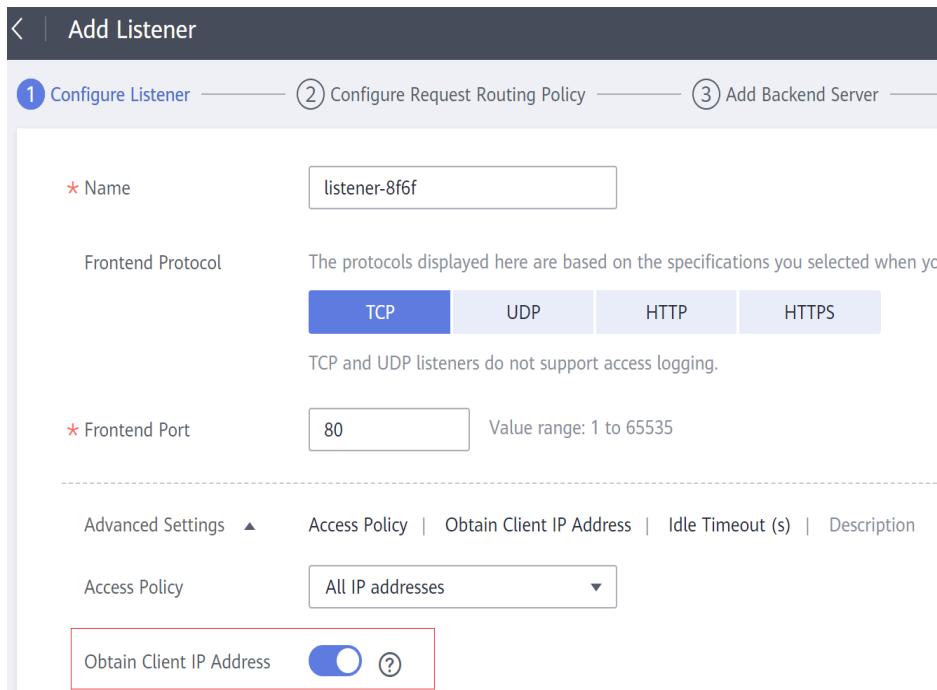
- Step 1** When creating a LoadBalancer Service on the CCE console, set **Service Affinity** to **Node level** instead of **Cluster level**.

### Create Service

The screenshot shows the 'Create Service' page. At the top, there's a 'Service Name' input field with placeholder text 'Enter a Service name.' Below it is an 'Access Type' section with three options: 'ClusterIP' (with a location pin icon), 'NodePort' (with a server icon), and 'LoadBalancer' (with a network icon). The 'LoadBalancer' option is highlighted with a blue border. Underneath, there's a 'Service Affinity' section with two buttons: 'Cluster-level' (grayed out) and 'Node-level' (highlighted in blue). A question mark icon is also present. At the bottom, there's a 'Namespace' field set to 'default'.

- Step 2** Go to the ELB console and enable the function of obtaining the client IP address of the listener corresponding to the load balancer. **Transparent transmission of source IP addresses is enabled for dedicated load balancers by default. You do not need to manually enable this function.**

1. Log in to the ELB console.
2. Click in the upper left corner to select the desired region and project.
3. Click **Service List**. Under **Networking**, click **Elastic Load Balance**.
4. On the **Load Balancers** page, click the name of the load balancer.
5. Click **Listeners**.
6. To add a listener, click **Add Listener**.
7. To modify a listener, locate the listener and click on the right of its name.
8. Enable **Obtain Client IP Address**.

**Figure 9-25** Enabling the function

----End

### Cloud Native Network 2.0 Model

In the Cloud Native Network 2.0 model, when a shared load balancer is used for load balancing, the service affinity cannot be set to **Node level**. As a result, source IP addresses cannot be obtained. To obtain a source IP address, you must use a **dedicated load balancer**. External access to the container does not need to pass through the forwarding plane.

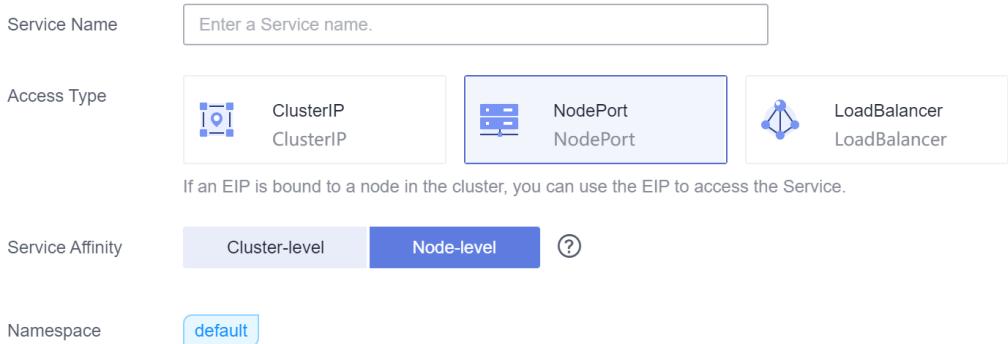
By default, transparent transmission of source IP addresses is enabled for dedicated load balancers. You do not need to manually enable **Obtain Client IP Address** on the ELB console. Instead, you only need to select a dedicated load balancer when creating an ENI LoadBalancer Service on the CCE console.

## NodePort

Set the service affinity of a NodePort Service to **Node level** instead of **Cluster level**. That is, set **spec.externalTrafficPolicy** of the Service to **Local**.

### NOTE

In the Cloud Native Network 2.0 model, the service affinity of the NodePort Service cannot be set to **Node level**. Therefore, source IP addresses cannot be obtained in this model.

**Figure 9-26** Selecting a node-level affinity**Create Service**

## 9.7 Increasing the Listening Queue Length by Configuring Container Kernel Parameters

### Scenario

**net.core.somaxconn** indicates the maximum number of half-open connections that can be backlogged in a listening queue. The default value is 128. If the queue is overloaded, you need to increase the listening queue length.

### Procedure

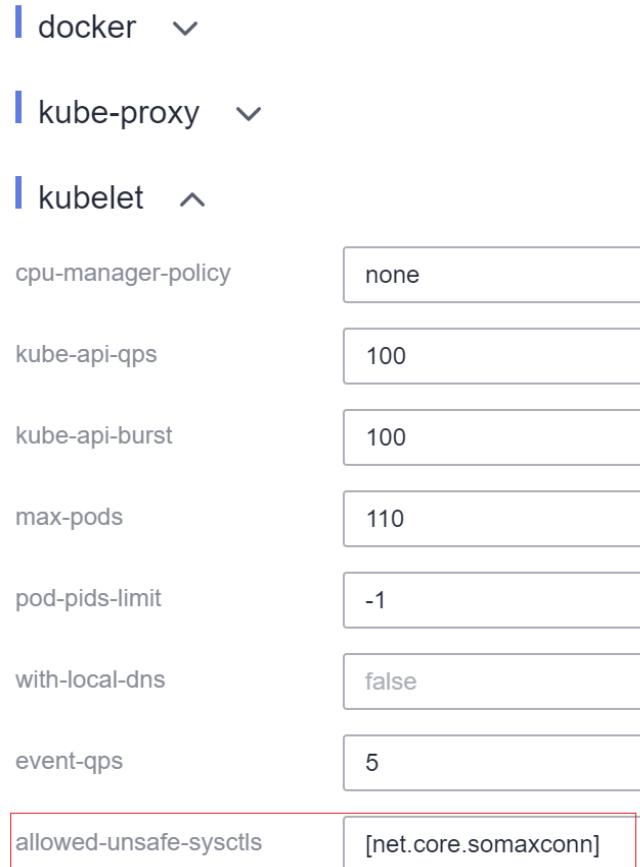
**Step 1** Modify kubelet configurations.

You can use either of the following methods to modify the kubelet parameters:

- **Modifying kubelet parameters in the node pool (only for clusters of v1.15 or later)**

Log in to the CCE console, go to the cluster details page, choose **More > Manage** in the node pool, and modify the kubelet parameters.

**Figure 9-27** Node pool configuration

**Figure 9-28** Modifying kubelet parameters

- **Modifying kubelet parameters of the node**

- a. Log in to the node.
- b. Edit the **/opt/cloud/cce/kubernetes/kubelet/kubelet** file. In versions earlier than 1.15, the file is **/var/paas/kubernetes/kubelet/kubelet**.  
Enable net.core.somaxconn.

```
--allowed-unsafe-sysctls=net.core.somaxconn
```

```
[root@test-565556-38106 ~]# cat /var/paas/kubernetes/kubelet/kubelet
DAEMON_ARGS="--bootstrap-kubeconfig=/var/paas/kubernetes/kubelet/boot.conf --cert-dir=/var/paas/kubernetes/kubelet/pki --rotate-certificates=true --network-plugin=cni --cni-conf-dir=/etc/cni/net.d/ --node-ip=192.168.116.149 --provider-id=acb56430-9a28-11e9-bf1e-0255ac101f9c --kubeconfig=/var/paas/kubernetes/kubelet/kubeconfig --config=/var/paas/kubernetes/kubelet/kubelet_config.yaml --authentication-token-webhook=true --root-dir=/mnt/paas/kubernetes/kubelet --hostname-override=192.168.116.149 --allow-privileged=True --v=2 --node-labels="os.name=EulerOS_2.0_SP5,os.version=3.10.0-862.14.0.1.h147.euleros2r7.x86_64,os.architecture=amd64,failure-domain.beta.kubernetes.io/zone=cn-north-1a,failure-domain.beta.kubernetes.io/region=cn-north-1,kubernetes.io/availableZone=cn-north-1a,failure-domain.beta.kubernetes.io/is-baremetal='false'" --machine-id-file=/var/paas/conf/server.conf --advisor-port=4194 --allowed-unsafe-sysctls=net.core.somaxconn"
[root@test-565556-38106 ~]#
```

- c. Restart kubelet.

```
systemctl restart kubelet
```

Check the kubelet status.

```
systemctl status kubelet
```

**NOTE**

After the kubelet configurations are changed for a cluster of v1.13 or earlier, the configurations will be restored if the cluster is upgraded to a later version.

**Step 2** Create a pod security policy.

Starting from **v1.17.17**, CCE enables pod security policies for kube-apiserver. You need to add **net.core.somaxconn** to **allowedUnsafeSysctls** of a pod security

policy to make the policy take effect. (This configuration is not required for clusters earlier than v1.17.17.)

- For details about CCE security policies, see [Pod Security Policies](#).
- For details about Kubernetes security policies, see [PodSecurityPolicy](#).
- For clusters with **net.core.somaxconn** enabled, add this configuration to **allowedUnsafeSysctls** of the corresponding pod security policy. For example, create a pod security policy as follows:

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
  name: sysctl-psp
spec:
  allowedUnsafeSysctls:
    - net.core.somaxconn
  allowPrivilegeEscalation: true
  allowedCapabilities:
    - '*'
  fsGroup:
    rule: RunAsAny
  hostIPC: true
  hostNetwork: true
  hostPID: true
  hostPorts:
    - max: 65535
      min: 0
    privileged: true
  runAsGroup:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  selinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
    - '*'
```

After creating the pod security policy **sysctl-psp**, you need to configure RBAC permission control for it.

Example:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sysctl-psp
rules:
  - apiGroups:
    - "*"
    resources:
    - podsecuritypolicies
  resourceNames:
  - sysctl-psp
  verbs:
  - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: sysctl-psp
roleRef:
  kind: ClusterRole
  name: sysctl-psp
  apiGroup: rbac.authorization.k8s.io
```

```
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

**Step 3** Create a workload, set kernel parameters, and configure the affinity with the node in [Step 1](#).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    description: ""
  labels:
    appgroup: ""
    name: test1
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test1
  template:
    metadata:
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api": "", "path": "", "port": "", "names": ""}]'
      labels:
        app: test1
    spec:
      containers:
        - image: 'nginx:1.14-alpine-perl'
          name: container-0
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
      imagePullSecrets:
        - name: default-secret
  securityContext:
    sysctls:
      - name: net.core.somaxconn
        value: '3000'
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                  - 192.168.x.x # Node name.
```

**Step 4** Log in to the node where the workload is deployed, access the container, and check whether the parameter configuration takes effect.

Run the following command in the container to check whether the configuration takes effect:

```
sysctl -a |grep somax
```

Figure 9-29 Viewing the parameter configuration

```
user@uw8i8he1ka75nyk-machine:~$ kubectl get pod
NAME           READY   STATUS    RESTARTS   AGE
test1-7794f95b55-fmsll  1/1     Running   0          17s
user@uw8i8he1ka75nyk-machine:~$ kubectl exec -it test1-7794f95b55-fmsll -- /bin/sh
/ # sysctl -a |grep somax
net.core.somaxconn = 3000
sysctl: error reading key 'net.ipv6.conf.all.stable_secret': I/O error
sysctl: error reading key 'net.ipv6.conf.default.stable_secret': I/O error
sysctl: error reading key 'net.ipv6.conf.eth0.stable_secret': I/O error
sysctl: error reading key 'net.ipv6.conf.lo.stable_secret': I/O error
```

----End

## 9.8 Using Multiple ENIs on a Node in a CCE Cluster

### Scenario

This section describes how to configure elastic network interfaces (ENIs) if multiple NICs are used on a node in a CCE cluster.

### Procedure

If the cluster version is **v1.15.11** or **v1.17.9**, **Network** is used as the CCE networking management component.

The following uses a CentOS 7 VM as an example to describe how to configure the VM networking.

**Step 1** Log in to the VM OS.

**Step 2** Run the **ifconfig** command to query the ENIs bound to the VM.

Assume that the name of the NIC is eth1. Run the following command:

```
vim /etc/sysconfig/network-scripts/ifcfg-eth1
```

Edit the file as follows:

```
DEVICE="eth1"
BOOTPROTO="dhcp"
ONBOOT="yes"
TYPE="Ethernet"
PERSISTENT_DHCLIENT="yes"
```

**Step 3** Run the following command to restart the VM network to make the networking configuration take effect:

```
systemctl restart network
```

----End

If the cluster version is **v1.17.11 or later**, **NetworkManager** is used as the CCE networking management component. When you configure ENIs on the CCE node, the configurations vary depending on the network model you use.

- **Tunnel network (overlay\_l2)**: ENIs can be automatically configured, for example, their IP addresses can be automatically obtained and renewed.
- **VPC network**: In addition to the primary NIC and the ENIs bound to containers, you need to configure extra ENIs.

- **Cloud Native Network 2.0:** In addition to the primary NIC and the ENIs bound to containers, you need to configure extra ENIs.

 NOTE

For a cluster that uses the Cloud Native Network 2.0 model (CCE Turbo cluster), the container networking is carried by ENIs and supplementary ENIs. If you want to use this network model, submit a service ticket to change the configuration of reserved ENIs in the cluster.

**Procedure**

The following uses a CentOS 7 VM as an example to describe how to configure the VM networking.

**Step 1** Log in to the VM OS.

**Step 2** Run the **ifconfig** command to query the ENIs bound to the VM.

Assume that the name of the NIC is eth1. Run the following command:

```
vim /etc/sysconfig/network-scripts/ifcfg-eth1
```

Edit the file as follows:

```
DEVICE="eth1"
BOOTPROTO="dhcp"
ONBOOT="yes"
TYPE="Ethernet"
PERSISTENT_DHCLIENT="yes"
```

**Step 3** Run the following command to restart the VM network to make the networking configuration take effect:

```
systemctl restart NetworkManager
```

**Step 4** Check whether the dhclient process of the eth1 NIC on the node is started. If yes, the configuration takes effect.

```
[root@jpp-jp-179-xingneng-56869 ~]# ps -ef | grep dhc
root    8783  8760  0 19:58 ?        00:00:00 /sbin/dhcclient -d -q -sf /usr/libexec/nm-dhcp-helper -pf /var/run/dhcclient-eth0.pid -lf /var/lib/NetworkManager/dhcClient-5fb06bd0-0bb0-7ffb-45f1-d6ed65f3e03-eth0.lease -cf /var/lib/NetworkManager/dhcClient-eth0.conf eth0
root    8785  8760  0 19:58 ?        00:00:00 /sbin/dhcclient -d -q -sf /usr/libexec/nm-dhcp-helper -pf /var/run/dhcclient-eth1.pid -lf /var/lib/NetworkManager/dhcClient-9c92fad9-6ecb-3e6c-eb4d-8a47c6f50c04-eth1.lease -cf /var/lib/NetworkManager/dhcClient-eth1.conf eth1
root     8809  2447  0 19:58 pts/1    00:00:00 grep --color=auto dhc
```

----End

## 9.9 Enabling Passthrough Networking for LoadBalancer Services

### Painpoint

A Kubernetes cluster can publish applications running on a group of pods as Services, which provide unified layer-4 access entries. For a Loadbalancer Service, kube-proxy configures the LoadbalanceIP in **status** of the Service to the local forwarding rule of the node by default. When a pod accesses the load balancer from within the cluster, the traffic is forwarded within the cluster instead of being forwarded by the load balancer.

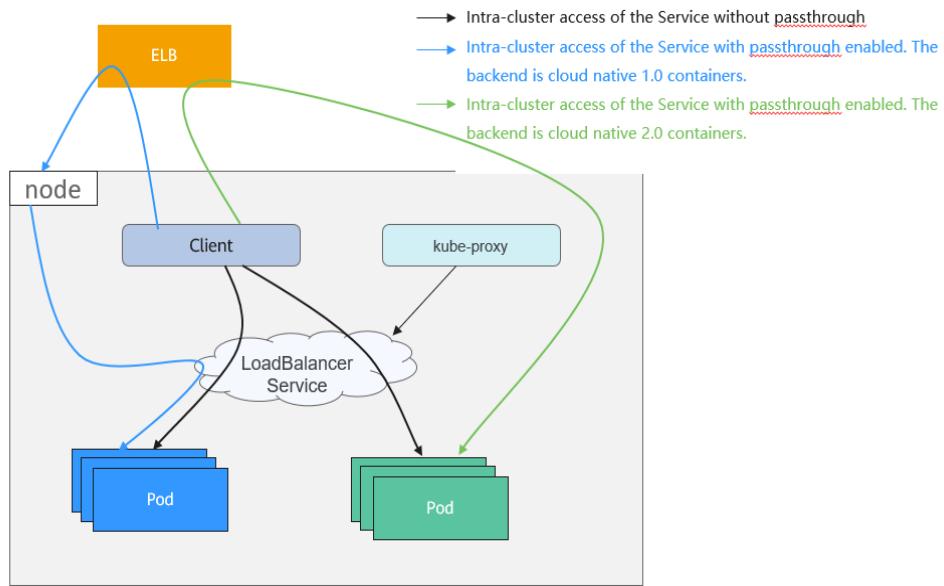
kube-proxy is responsible for intra-cluster forwarding. kube-proxy has two forwarding modes: iptables and IPVS. iptables is a simple polling forwarding mode. IPVS has multiple forwarding modes but it requires modifying the startup

parameters of kube-proxy. Compared with iptables and IPVS, load balancers provide more flexible forwarding policies as well as health check capabilities.

## Solution

CCE supports passthrough networking. You can configure the **annotation** of **kubernetes.io/elb.pass-through** for the Loadbalancer Service. Intra-cluster access to the Service load balancer address is then forwarded to backend pods by the load balancer.

**Figure 9-30** Passthrough networking illustration



- For CCE clusters:
  - When a LoadBalancer Service is accessed within the cluster, the access is forwarded to the backend pods using iptables/IPVS by default.
  - Configure **elb.pass-through** for LoadBalancer Services. Intra-cluster access to the Service is forwarded to the ELB load balancer first. The load balancer forwards the traffic back to the node.
- For CCE Turbo clusters:
  - When a LoadBalancer Service is accessed within the cluster, the access is forwarded to the backend pods using iptables/IPVS by default.
  - Configure **elb.pass-through** for LoadBalancer Services. Intra-cluster access to the Service is forwarded to the ELB first. The ELB load balancer forwards the traffic directly to the container.

## Notes and Constraints

- After passthrough networking is configured for a dedicated load balancer, containers on the node where the workload runs cannot be accessed through the Service.
- Passthrough networking is not supported for clusters of v1.15 or earlier.

- In IPVS network mode, the pass-through settings of Service connected to the same ELB must be the same.

## Procedure

This section describes how to create a Deployment using an Nginx image and create a Service with passthrough networking enabled.

**Step 1** Use the Nginx image to create a Deployment.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx:latest
          name: container-0
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
      imagePullSecrets:
        - name: default-secret
```

**Step 2** Create a LoadBalancer Service and configure **kubernetes.io/elb.pass-through** to **true**.

For details about how to create a LoadBalancer Service, see [Automatically Creating a LoadBalancer Service](#).

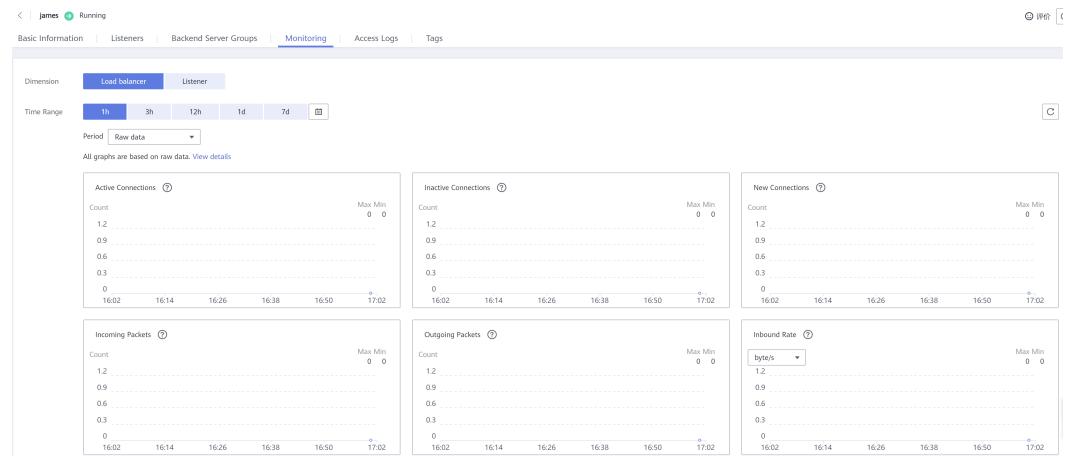
```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.pass-through: "true"
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"cce-bandwidth","bandwidth_chargeemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_bgp","name":"james"}'
  labels:
    app: nginx
    name: nginx
spec:
  externalTrafficPolicy: Local
  ports:
    - name: service0
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

A shared load balancer named **james** is automatically created. Use [kubernetes.io/elb.subnet-id](#) to specify the VPC subnet where the load balancer is located. The load balancer and the cluster must be in the same VPC.

----End

## Verification

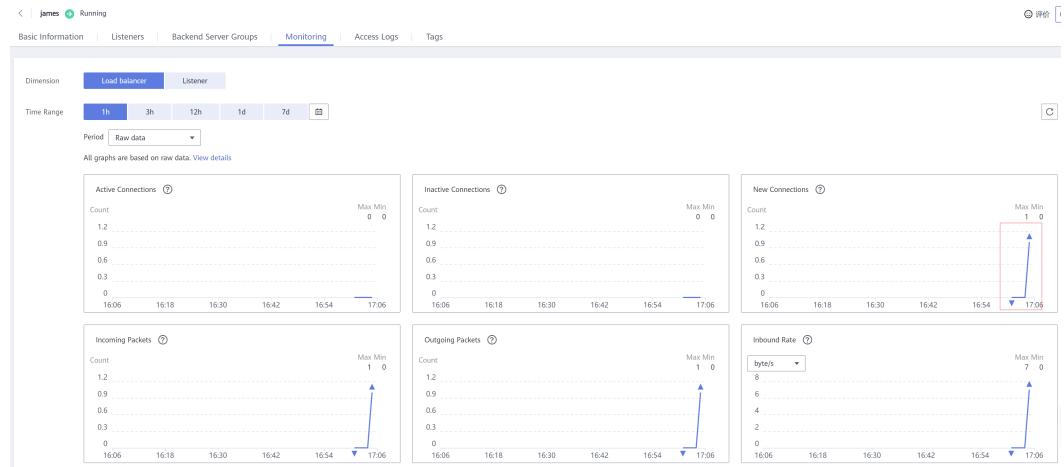
Check the ELB load balancer corresponding to the created Service. The load balancer name is **james**. The number of ELB connections is **0**, as shown in the following figure.



Use `kubectl` to connect to the cluster, go to an Nginx container, and access the ELB address. The access is successful.

```
# kubectl get pod
NAME           READY   STATUS    RESTARTS   AGE
nginx-7c4c5cc6b5-vpnvx  1/1     Running   0          9m47s
nginx-7c4c5cc6b5-xj5wl  1/1     Running   0          9m47s
# kubectl exec -it nginx-7c4c5cc6b5-vpnvx -- /bin/sh
# curl 120.46.141.192
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
body {
  width: 35em;
  margin: 0 auto;
  font-family: Tahoma, Verdana, Arial, sans-serif;
}
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Wait for a period of time and view the ELB monitoring data. A new access connection is created for the ELB, indicating that the access passes through the ELB load balancer as expected.



## 9.10 CoreDNS Configuration Optimization

### 9.10.1 Overview

#### Scenario

DNS is one of the important basic services in Kubernetes. When the container DNS policy is not properly configured and the cluster scale is large, DNS resolution may time out or fail. In extreme scenarios, a large number of services in the cluster may fail to be resolved. This section describes the best practices of CoreDNS configuration optimization in Kubernetes clusters to help you avoid such problems.

#### Solutions

CoreDNS configuration optimization includes clients and servers.

On the client, you can optimize domain name resolution requests to reduce resolution latency, and use proper container images and NodeLocal DNSCache to reduce resolution exceptions.

- [Optimizing Domain Name Resolution Requests](#)
- [Selecting a Proper Image](#)
- [Avoiding Occasional DNS Resolution Timeout Caused by IPVS Defects](#)
- [Using NodeLocal DNSCache](#)
- [Upgrading the CoreDNS in the Cluster Timely](#)
- [Adjusting the DNS Configuration of the VPC and VM](#)

On the server, you can adjust the CoreDNS deployment status or CoreDNS configuration to improve the availability and throughput of CoreDNS in the cluster.

- [Monitors the coredns Add-on](#)

- [Adjusting the CoreDNS Deployment Status](#)
- [Configuring CoreDNS](#)

For more information about CoreDNS configurations, see <https://coredns.io/>.

CoreDNS open source community: <https://github.com/coredns/coredns>

## Prerequisites

- A CCE cluster has been created. For details, see [Creating a Kubernetes Cluster](#).
- You have connected to the cluster using kubectl. For details, see [Connecting to a Cluster Using kubectl](#).
- The coredns add-on has been installed (the latest version is recommended). For details, see [CoreDNS](#).

## 9.10.2 Client

### 9.10.2.1 Optimizing Domain Name Resolution Requests

DNS resolution is frequently used in Kubernetes clusters. Based on the characteristics of DNS resolution in Kubernetes, you can optimize domain name resolution requests in the following ways:

#### Using Connection Pool

When a containerized application needs to frequently request another service, you are advised to use the connection pool. The connection pool can cache the link information of the upstream service to avoid the overhead of DNS resolution and TCP link reestablishment for each access.

#### Optimizing the resolve.conf File in the Container

The **ndots** and **search** parameters in the **resolve.conf** file determine the domain name resolution efficiency. For details about the two parameters, see [DNS Configuration](#).

#### Optimizing the Domain Name Configuration

When a container needs to access a domain name, configure the domain name based on the following rules to improve the domain name resolution efficiency.

1. When a pod accesses a Service in the same namespace, **<service-name>** is preferred. **service-name** indicates the service name.
2. When a pod accesses a service across namespaces, **<service-name>.<namespace-name>** is preferred. **namespace-name** indicates the namespace where the Service is located.
3. When a pod accesses an external domain name of a cluster, the FQDN domain name is preferentially used. This type of domain name is specified by adding a period (.) at the end of a common domain name to avoid multiple invalid searches caused by search domain combination. For example, you need to access **www.huawicoud.com**, the FQDN domain name **www.huawicoud.com.** is preferentially used.

## Using Local Cache

If the cluster specifications and the number of DNS resolution requests are large, you can cache the DNS resolution result on the node. You are advised to use NodeLocal DNSCache. For details, see [Using NodeLocal DNSCache to Improve DNS Performance](#).

### 9.10.2.2 Selecting a Proper Image

The musl libc library of the Alpine container image differs from the standard glibc library in the following aspects:

- Alpine 3.3 and earlier versions do not support parameter or domain search. As a result, service discovery cannot be completed.
- Multiple DNS servers configured in `/etc/resolve.conf` are concurrently requested. As a result, NodeLocal DNSCache cannot improve the DNS performance.
- When the same Socket is used to request A and AAAA records concurrently, the Conntrack source port conflict is triggered in the kernel of an earlier version. As a result, packet loss occurs.
- If the domain name cannot be resolved when Alpine is used as the base container image, update the base container image for testing.

For details about the function differences between glibc and glibc, see [Functional differences from glibc](#).

### 9.10.2.3 Avoiding Occasional DNS Resolution Timeout Caused by IPVS Defects

When the cluster uses IPVS as the kube-proxy load balancing, you may encounter DNS resolution timeout occasionally during CoreDNS scale-in or restart. This problem is caused by a Linux kernel defect. For details, see <https://github.com/torvalds/linux/commit/35dfb013149f74c2be1ff9c78f14e6a3cd1539d1?spm=a2c4g.11186623.0.0.5453eb84wzqSaC>.

You can use NodeLocal DNSCache to reduce the impact of IPVS defects. For details, see [Using NodeLocal DNSCache to Improve DNS Performance](#).

### 9.10.2.4 Using NodeLocal DNSCache

NodeLocal DNSCache can improve the performance of service discovery. For details about NodeLocal DNSCache and how to deploy it in a cluster, see [Using NodeLocal DNSCache to Improve DNS Performance](#).

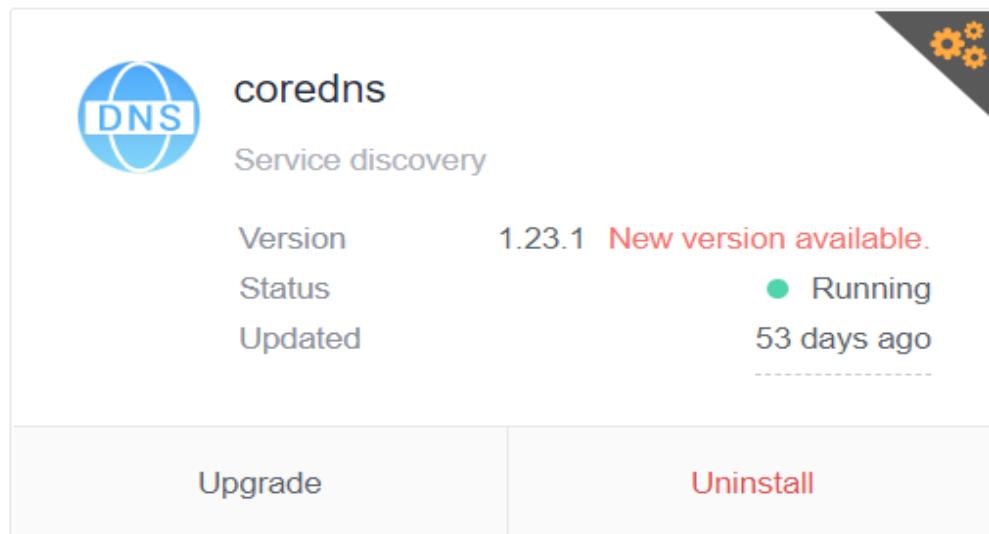
### 9.10.2.5 Upgrading the CoreDNS in the Cluster Timely

CoreDNS provides simple functions and is compatible with different Kubernetes versions. CCE periodically synchronizes bugs from the community and upgrades the coredns add-on version. You are advised to periodically upgrade the CoreDNS. The CCE add-on management center supports the CoreDNS installation and upgrade. You can define the CoreDNS version in the cluster. If the version can be upgraded, arrange services to seamlessly upgrade the CoreDNS component in the cluster as soon as possible.

You can upgrade CoreDNS in a cluster through the following process:

**Step 1** Log in to the CCE console, select a cluster, and click **Add-ons** in the navigation pane.

**Step 2** In the **Add-on Installed**, locate the coredns add-on and click **Upgrade**.



**Step 3** Set parameters as prompted. For details, see [coredns \(System Resource Add-on, Mandatory\)](#).

----End

### 9.10.2.6 Adjusting the DNS Configuration of the VPC and VM

When the coredns add-on is started, it obtains the DNS configuration in the **resolve.conf** file from the deployed instance by default and uses the configuration as the upstream resolution server address. Before the coredns add-on is restarted, the **resolve.conf** configuration on the node is not reloaded. Suggestion:

- Ensure that the **resolve.conf** configuration of each node in the cluster is the same. In this way, the coredns add-on can schedule requests to any node in the cluster.
- When modifying the **resolve.conf** file, if the node has a coredns add-on, restart the coredns add-on timely to ensure status consistency.

## 9.10.3 Server

### 9.10.3.1 Monitors the coredns Add-on

- CoreDNS exposes health metrics such as resolution results through the standard Prometheus API to detect exceptions on the CoreDNS server or even upstream DNS server.
- Port for obtaining coredns metrics. The default zone listening IP address is **{\$POD\_IP}:9153**. Retain the default value. Otherwise, CloudScope cannot collect coredns metrics.

- If you use self-built Prometheus to monitor Kubernetes clusters, you can observe related metrics on Prometheus and set alarms for the following key metrics. For details, see [enables Prometheus metrics](#).

### 9.10.3.2 Adjusting the CoreDNS Deployment Status

By default, the coredns add-on is installed in the CCE cluster and the CoreDNS application runs on the same cluster node as the service container. Pay attention to the following points when deploying the coredns add-on:

- [Adjusting the Number of CoreDNS Replicas Properly](#)
- [Allocating the Location of the CoreDNS Properly](#)
- [Manually Expanding the Number of Replicas](#)
- [Automatically Expanding the CoreDNS Capacity Based on the HPA](#)

#### Adjusting the Number of CoreDNS Replicas Properly

You are advised to set the number of CoreDNS replicas to at least 2 in any case and keep the number of replicas within a proper range to carry the resolution of the entire cluster. The default number of instances for installing the coredns add-on in a CCE cluster is 2.

- Modifying the number of CoreDNS replicas, CPUs, and memory size will change CoreDNS's parsing capability. Therefore, evaluate the impact before the operation.
- By default, podAntiAffinity (pod anti-affinity) is configured for the coredns add-on. If a node already has a CoreDNS pod, no new pod can be added. That is, only one CoreDNS pod can run on a node. If the number of configured CoreDNS replicas is greater than the number of cluster nodes, the excess pods cannot be scheduled. Therefore, keep the number of replicas less than or equal to the number of nodes.

#### Allocating the Location of the CoreDNS Properly

- You are advised to distribute CoreDNS replicas on nodes in different AZs and clusters to prevent single-node or single-AZ faults. By default, podAntiAffinity (pod anti-affinity) is configured for the coredns add-on. Some or all replicas may be deployed on the same node due to insufficient node resources. In this case, delete the pod and reschedule it again.
- The CPU and memory of the cluster node where the coredns add-on runs must not be used up. Otherwise, the QPS and response delay of domain name resolution will be affected.

#### Manually Expanding the Number of Replicas

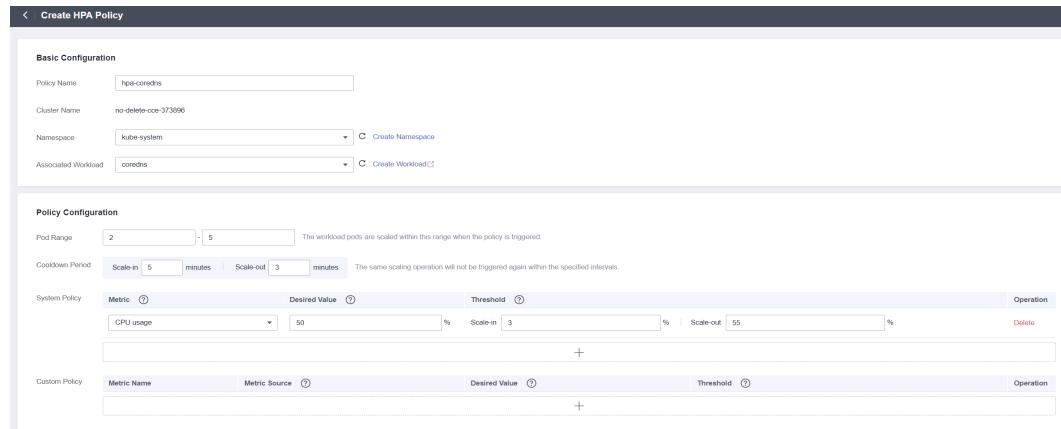
You can call the API to modify the coredns add-on specifications. Modify the number of replicas as required. For details, see [Customizing CoreDNS Specifications](#).

#### Automatically Expanding the CoreDNS Capacity Based on the HPA

HPA frequently scales down the number of the coredns add-on replicas. Therefore, you are advised not to use HPA. If HPA is required, use the CCE-developed add-on

cce-hpa-controller to configure an HPA automated scale-out policy. The process is as follows:

- Step 1** Log in to the CCE console and access the cluster details page. Choose **Add-ons** in the navigation pane, locate **cce-hpa-controller** on the right, and click **Install**.
- Step 2** You can select **Single** or **Custom** for **Add-on Specifications**, and click **Install**. For details, see [cce-hpa-controller](#).
- Step 3** On the CCE console, Choose **Workload > Scaling** in the navigation pane, select **kube-system** for the namespace, and click **Create HPA Policy**.



In the **Basic Settings** area, you can edit the policy name. Set **Namespace** to **kube-system** and associated workload to coredns.

On the **Policy Configuration** page, you can customize HPA policies based on metrics such as CPU usage and memory usage to automatically scale out the coredns add-on replicas.

- Step 4** Click **Create**. If the latest status is **Started**, the policy takes effect.

----End

### 9.10.3.3 Configuring CoreDNS

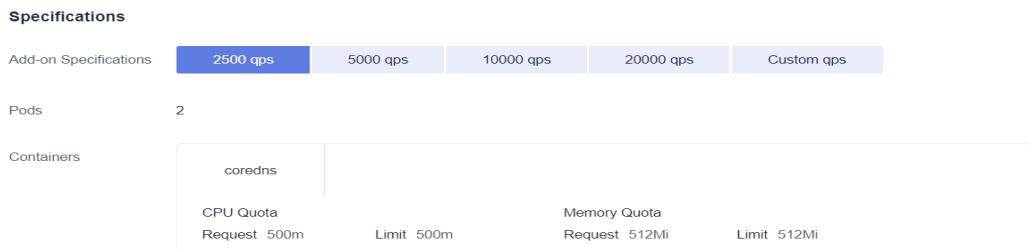
On the console, the coredns add-on can only be configured with the preset specifications, which can satisfy most of the services. In some scenarios where there are requirements on the CoreDNS resource usage, you may need to customize the add-on specifications.

For details about how to customize CoreDNS parameters, see [Customizing CoreDNS Specifications](#).

CoreDNS official document: <https://coredns.io/plugins/?spm=a2c4g.11186623.0.0.20c4eb84HHbdty>

## Configuring CoreDNS Specifications

- Step 1** Log in to the CCE console and click the cluster.
- Step 2** In the navigation pane on the left, choose **Add-ons**. Under **Add-ons Installed**, click **Edit** in the **Operation** column of the coredns add-on. The add-on details page is displayed.



**Step 3** In the **Specification Configuration** area, configure CoreDNS specifications.

**Step 4** You can select the domain name resolution QPS provided by CoreDNS based on service requirements.

**Step 5** You can also customize the CoreDNS parameter specifications of the cluster by selecting the number of instances, CPU quota, and memory quota.



- The request value must be less than or equal to the limit value, otherwise it cannot be created successfully.
- Please ensure that the node resources under the cluster are sufficient, otherwise it cannot be created successfully.

**Step 6** Click **Confirm**.

----End

## Properly Configuring the Stub Domain for DNS

**Step 1** Log in to the CCE console and access the cluster details page.

**Step 2** In the navigation pane, choose **Add-ons**. On the displayed page, click **Edit** under CoreDNS.

**Step 3** Add a stub domain in the **Parameters** area.

Modify the **stub\_domains** parameter in the format of a key-value pair. The key is a DNS suffix domain name, and the value is a DNS IP address or a group of DNS IP addresses.

```
{  
  "stub_domains": {  
    "consul.local": [  
      "10.150.0.1"  
    ]  
  },  
  "upstream_nameservers": []  
}
```

**Step 4** Click **OK**.

----End

You can also modify the ConfigMap as follows:

```
$ kubectl edit configmap coredns -n kube-system
apiVersion: v1
data:
  Corefile: |-  
    :5353 {  
      bind {$POD_IP}  
      cache 30  
      errors  
      health {$POD_IP}:8080  
      kubernetes cluster.local in-addr.arpa ip6.arpa {  
        pods insecure  
        fallthrough in-addr.arpa ip6.arpa  
      }  
      loadbalance round_robin  
      prometheus {$POD_IP}:9153  
      forward . /etc/resolv.conf {  
        policy random  
      }  
      reload  
    }  
  
    consul.local:5353 {  
      bind {$POD_IP}  
      errors  
      cache 30  
      forward . 10.150.0.1  
    }  
kind: ConfigMap
metadata:
  creationTimestamp: "2022-05-04T04:42:24Z"
  labels:  
    app: coredns  
    k8s-app: coredns  
    kubernetes.io/cluster-service: "true"  
    kubernetes.io/name: CoreDNS  
    release: cceaddon-coredns  
  name: coredns  
  namespace: kube-system  
  resourceVersion: "8663493"  
  uid: bba87142-9f8d-4056-b8a6-94c3887e9e1d
```

## Properly Configuring the Host

If you need to specify hosts for a specific domain name, you can use the Hosts plug-in. Example:

- Step 1** Use kubectl to connect to the cluster.
- Step 2** Modify the CoreDNS configuration file and add the custom domain name to the hosts file.

Point **www.example.com** to **192.168.1.1**. When CoreDNS resolves **www.example.com**, **192.168.1.1** is returned.

### NOTICE

The fallthrough field must be configured. **fallthrough** indicates that when the domain name to be resolved cannot be found in the hosts file, the resolution task is transferred to the next CoreDNS plug-in. If **fallthrough** is not specified, the task ends and the domain name resolution stops. As a result, the domain name resolution in the cluster fails.

For details about how to configure the hosts file, visit <https://coredns.io/plugins/hosts/>.

```
$ kubectl edit configmap coredns -n kube-system
apiVersion: v1
data:
  Corefile: |-
    :5353 {
      bind {$POD_IP}
      cache 30
      errors
      health {$POD_IP}:8080
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
      }
      hosts {
        192.168.1.1 www.example.com
        fallthrough
      }
      loadbalance round_robin
      prometheus {$POD_IP}:9153
      forward . /etc/resolv.conf
      reload
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2021-08-23T13:27:28Z"
  labels:
    app: coredns
    k8s-app: coredns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: CoreDNS
    release: cceaddon-coredns
  name: coredns
  namespace: kube-system
  resourceVersion: "460"
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: be64aaad-1629-441f-8a40-a3efc0db9fa9
```

After modifying the hosts file in CoreDNS, you do not need to configure the hosts file in each pod.

----End

## Configuring the Default Protocol Between the Forward Plug-in and the Upstream DNS Service

**Step 1** The NodeLocal DNSCache uses TCP to communicate with the CoreDNS. The CoreDNS communicates with the upstream DNS server based on the protocol used by the request source. By default, external domain name resolution requests from service containers pass through NodeLocal DNSCache and CoreDNS in sequence, and finally request the DNS server in the VPC using TCP.

**Step 2** However, the cloud server does not support TCP. To use NodeLocal DNSCache, you need to modify the CoreDNS configuration so that UDP is preferentially used to communicate with the upstream DNS server, preventing resolution exceptions. You are advised to use the following method to modify the CoreDNS configuration file:

The forward plug-in is used to set the upstream Nameservers DNS server. The following parameters are included:

**prefer\_udp:** Even if a request is received through TCP, UDP must be used first.

If you want CoreDNS to preferentially use UDP to communicate with upstream systems, set the protocol in the Forward plug-in to **prefer\_udp**. For details about the forward plug-in, see <https://coredns.io/plugins/forward/>.

Modify the ConfigMap of coredns.

```
kubectl edit configmap coredns -n kube-system
```

- Before:  
forward . /etc/resolv.conf
- After:  
forward . /etc/resolv.conf {  
 prefer\_udp  
}

----End

## Configuring IPv6 Resolution Properly

If the IPv6 kernel module is not disabled on the Kubernetes cluster host machine, the container initiates IPv4 and IPv6 resolution at the same time by default when requesting the coredns add-on. Generally, only IPv4 addresses are used. Therefore, if you only configure **DOMAIN in IPv4 address**, the coredns add-on forwards the request to the upstream DNS server for resolution because the local configuration cannot be found. As a result, the DNS resolution request of the container slows down.

CoreDNS provides the template plug-in. After being configured, CoreDNS can immediately return an empty response to all IPv6 requests to prevent the requests from being forwarded to the upstream DNS.

```
kubectl edit configmap coredns -n kube-system
```

Edit the YAML content as follows:

- AAAA indicates an IPv6 resolution request. If **NXDOMAIN** is returned in the **rcode** control response, no resolution result is returned.

For details about the template plug-in, visit <https://github.com/coredns/coredns/tree/master/plugin/template>.

```
apiVersion: v1
data:
  Corefile: |-  
    :5353 {  
      bind {$POD_IP}  
      cache 30  
      errors  
      health {$POD_IP}:8080  
      # Add the following template plug-in and retain other data.  
      template ANY AAAA {  
        rcode NXDOMAIN  
      }  
      kubernetes cluster.local in-addr.arpa ip6.arpa {  
        pods insecure  
        fallthrough in-addr.arpa ip6.arpa  
      }  
      hosts {  
        192.168.1.1 www.example.com  
        fallthrough  
      }  
      loadbalance round_robin  
      prometheus {$POD_IP}:9153  
      forward . /etc/resolv.conf {  
        prefer_udp  
      }  
      reload  
    }
```

```
kind: ConfigMap
metadata:
  annotations:
    meta.helm.sh/release-name: cceaddon-coredns
    meta.helm.sh/release-namespace: kube-system
  creationTimestamp: "2022-08-04T01:10:37Z"
  labels:
    app: coredns
    app.kubernetes.io/managed-by: Helm
    k8s-app: coredns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: CoreDNS
    release: cceaddon-coredns
  name: coredns
  namespace: kube-system
  resourceVersion: "2030659"
  uid: b2f43f8a-7920-41be-84bc-6b9f6d449e32
```

# 10 Storage

## 10.1 Expanding Node Disk Capacity

### System Disk

**Step 1** Expand the capacity of the system disk on the EVS console.

**Step 2** Restart the node on the ECS console.

**Step 3** Log in to the CCE console and click the cluster. In the navigation pane, choose **Nodes**. Click **More > Sync Server Data** at the row containing the target node.

----End

### Node Data Disk (Dedicated for Docker)

**Step 1** Expand the capacity of the data disk on the EVS console.

**Step 2** Log in to the CCE console and click the cluster. In the navigation pane, choose **Nodes**. Click **More > Sync Server Data** at the row containing the target node.

**Step 3** Log in to the target node.

**Step 4** Run the **lsblk** command to check the block device information of the node.

A data disk is divided depending on the container storage **Rootfs**:

- Overlayfs: No independent thin pool is allocated. Image data is stored in the **dockersys** disk.

```
# lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda        8:0    0   50G  0 disk
└─sda1     8:1    0   50G  0 part /
sdb        8:16   0 200G  0 disk
└─vgpaas-dockersys 253:0  0 90G  0 lvm /var/lib/docker      # Space used by Docker.
└─vgpaas-kubernetes 253:1  0 10G  0 lvm /mnt/paas/kubernetes/kubelet # Space used by Kubernetes.
```

Run the following commands on the node to add the new disk capacity to the **dockersys** disk:

```
pvresize /dev/sdb
lvextend -l+100%FREE -n vgpaas/dockersys
resize2fs /dev/vgpaas/dockersys
```

- **Devicemapper:** A thin pool is allocated to store image data.

```
# lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda        8:0    0   50G  0 disk
└─sda1     8:1    0   50G  0 part /
sdb        8:16   0  200G  0 disk
└─vgpaas-dockersys  253:0  0  18G  0 lvm  /var/lib/docker
└─vgpaas-thinpool_tmeta  253:1  0   3G  0 lvm
  └─vgpaas-thinpool  253:3  0  67G  0 lvm          # Thin pool space.
...
└─vgpaas-thinpool_tdata  253:2  0  67G  0 lvm
  └─vgpaas-thinpool  253:3  0  67G  0 lvm
...
└─vgpaas-kubernetes  253:4  0  10G  0 lvm  /mnt/paas/kubernetes/kubelet
```

Run the following commands on the node to add the new disk capacity to the **thinpool** disk:

```
pvresize /dev/sdb
lvextend -l+100%FREE -n vgpaas/thinpool
```

----End

## Node Data Disk (Kubernetes)

- Step 1 Expand the capacity of the data disk on the EVS console.
- Step 2 Log in to the CCE console and click the cluster. In the navigation pane, choose **Nodes**. Click **More > Sync Server Data** at the row containing the target node.
- Step 3 Log in to the target node.
- Step 4 Run the following commands on the node to add the new disk capacity to the Kubernetes disk:

```
pvresize /dev/sdb
lvextend -l+100%FREE -n vgpaas/kubernetes
resize2fs /dev/vgpaas/kubernetes
```

----End

## Container Disk Space (10 GB)

- Step 1 Log in to the CCE console and click the name of the target cluster in the cluster list.
- Step 2 Choose **Nodes** from the navigation pane.
- Step 3 Select the target node and choose **More > Reset Node** in the **Operation** column.

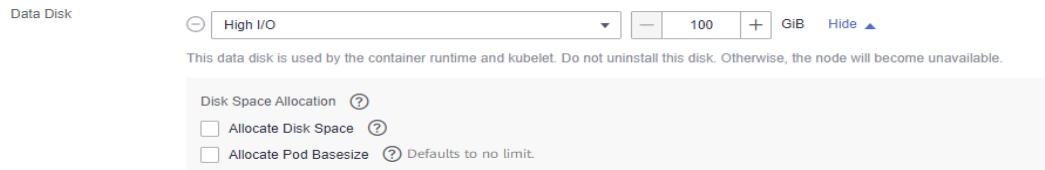
### NOTICE

Resetting a node may make unavailable the node-specific resources (such as local storage and workloads scheduled to this node). Exercise caution when performing this operation to avoid impact on running services.

- Step 4 Click **Yes**.

**Step 5** Reconfigure node parameters.

If you need to adjust the container storage space, pay attention to the following configurations:



**Storage Settings:** Click **Expand** next to the data disk to set the following parameters:

- **Allocate Disk Space:** storage space used by the container engine to store the Docker/containerd working directory, container image data, and image metadata. Defaults to 90% of the data disk.
- **Allocate Pod Basesize:** CCE allows you to set an upper limit for the disk space occupied by each workload pod (including the space occupied by container images). This setting prevents the pods from taking all the disk space available, which may cause service exceptions. It is recommended that the value be smaller than or equal to 80% of the container engine space.

 **NOTE**

- The capability of customizing pod basesize is related to the node OS and container storage rootfs.
  - When the rootfs uses Device Mapper, the node supports custom pod basesize. The default storage space of a single container is 10 GiB.
  - When the rootfs uses OverlayFS, most nodes do not support custom pod basesize. The storage space of a single container is not limited and defaults to the container engine space.  
Only EulerOS 2.9 nodes in clusters of 1.19.16, 1.21.3, 1.23.3, and later versions support custom pod basesize.  
For details about the relationship between node OSs and container storage rootfs, see [Mapping between Node OSs and Container Engines](#).
- In the case of using Docker on EulerOS 2.9 nodes, **basesize** will not take effect if **CAP\_SYS\_RESOURCE** or **privileged** is configured for a container.

For more information about container storage space allocation, see [Data Disk Space Allocation](#).

**Step 6** After the node is reset, log in to the node and run the following command to access the container and check whether the container storage capacity has been expanded:

```
docker exec -it container_id /bin/sh or kubectl exec -it container_id /bin/sh  
df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/mapper/docker-253:1-787293-631c1bde2cbe82e39f32253b216ba914cb183b168b54700b3e5b9a54ee40a0d1	15G	229M	15G	2%	/
tmpfs	32G	0	32G	0%	/dev
tmpfs	32G	0	32G	0%	/sys/firmware/group
/dev/mapper/vgpaas-kubernetes	9.0G	3.7M	9.0G	1%	/etc/hosts
tmpfs	49G	5.2G	39G	14%	/etc/hostname
shm	64M	0	64M	0%	/dev/shm
tmpfs	32G	16K	32G	1%	/run/secrets/kubernetes.io/serviceaccount
tmpfs	32G	0	32G	0%	/proc/acpi
tmpfs	32G	0	32G	0%	/sys/firmware
tmpfs	32G	0	32G	0%	/proc/scsi
tmpfs	32G	0	32G	0%	/proc/kbox
tmpfs	32G	0	32G	0%	/proc/oom_extend

----End

## PVC

Cloud storage:

- OBS buckets have no storage limits and do not need to be expanded.
- EVS and SFS
  - For pay-per-use EVS disks or SFS file systems, directly change the capacity requested in the PVC.  
# kubectl edit pvc xxxx  
...  
...
  - For yearly/monthly-billed EVS disks or SFS file systems, expand the capacity on the EVS or SFS console and then change the capacity in the PVC.
- For SFS Turbo, expand the capacity on the SFS console and then change the capacity in the PVC.

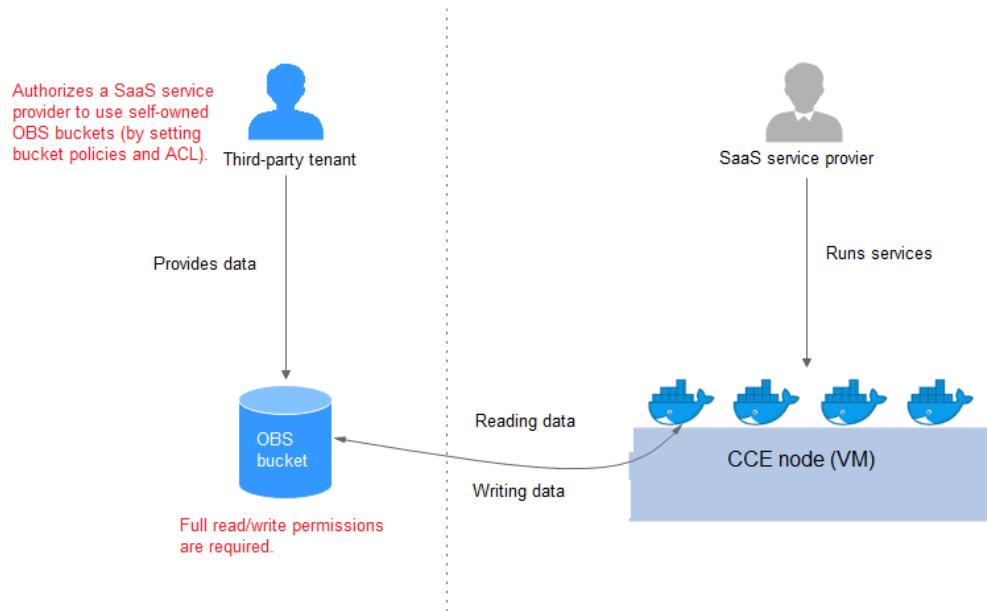
## 10.2 Mounting an Object Storage Bucket of a Third-Party Tenant

This section describes how to mount OBS buckets and OBS parallel file systems (preferred) of third-party tenants.

### Scenario

The CCE cluster of a SaaS service provider needs to be mounted with the OBS bucket of a third-party tenant, as shown in [Figure 10-1](#).

Figure 10-1 Mounting an OBS bucket of a third-party tenant



1. **The third-party tenant authorizes the SaaS service provider to access the OBS buckets or parallel file systems** by setting the bucket policy and bucket ACL.
2. **The SaaS service provider statically imports the OBS buckets and parallel file systems of the third-party tenant.**
3. The SaaS service provider processes the service and writes the processing result (result file or result data) back to the OBS bucket of the third-party tenant.

## Precautions

- Only parallel file systems and OBS buckets of third-party tenants in the same region can be mounted.
- Only clusters where the everest add-on of v1.1.11 or later has been installed (the cluster version must be v1.15 or later) can be mounted with OBS buckets of third-party tenants.
- The service platform of the SaaS service provider needs to manage the lifecycle of the third-party bucket PVs. When a PVC is deleted separately, the PV is not deleted. Instead, it will be retained. To do so, you need to call the native Kubernetes APIs to create and delete static PVs.

## Authorizing the SaaS Service Provider to Access the OBS Buckets

The following uses an OBS bucket as an example to describe how to set a bucket policy and bucket ACL to authorize the SaaS service provider. The configuration for an OBS parallel file system is the same.

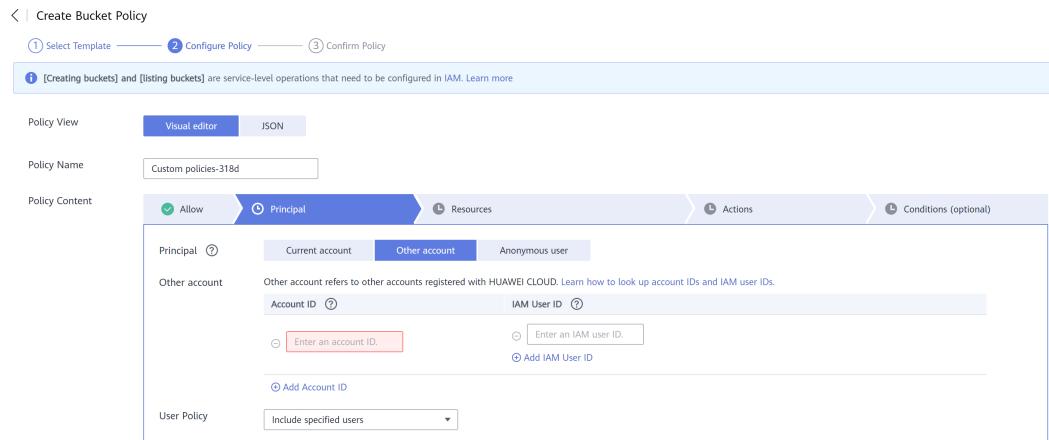
**Step 1** Log in to the OBS console. In the navigation pane, choose **Buckets**.

**Step 2** In the bucket list, click a bucket name to access the **Overview** page.

**Step 3** In the navigation pane, choose **Permissions > Bucket Policy**. In the right pane, click **Create**. In the Custom policy area, click **Create Custom Policy**.

Set the parameters as shown in the following figure.

**Figure 10-2** Creating a bucket policy



- **Allow:** Select **Allow**.
- **Principal:** Select **Other account**, and enter the account ID and user ID. The bucket policy takes effect for the specified users.
- **Resources:** Select the resources that can be operated.
- **Actions:** Select the actions that can be operated.

**Step 4** In the navigation pane, choose **Permissions > Bucket ACLs**. In the right pane, click **Add**. Enter the account ID of the authorized user, select **Read, Object read**, and **Write** for **Access to Bucket**, select **Read** and **Write** for **Access to ACL**, and click **OK**.

----End

## Statically Importing OBS Buckets and Parallel File Systems

- **Static PV of an OBS bucket:**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: objbucket    #Replace the name with the actual PV name of the bucket.
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
    - ReadWriteMany
  capacity:
    storage: 1Gi
  mountOptions:
    - default_acl=bucket-owner-full-control    #New OBS mounting parameters
  csi:
    driver: obs.csi.everest.io
    fsType: obsfs
    volumeAttributes:
      everest.io/obs-volume-type: STANDARD
      everest.io/region: ap-southeast-1    #Set it to the ID of the current region.
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
    volumeHandle: objbucket    #Replace the name with the actual bucket name of the third-
```

party tenant.

`persistentVolumeReclaimPolicy: Retain` #This parameter must be set to **Retain** to ensure that the bucket will not be deleted when a PV is deleted.

`storageClassName: csi-obs-mountoption` #You can associate a new custom OBS storage class or the built-in csi-obs of the cluster.

- **mountOptions:** This field contains the new OBS mounting parameters that allow the bucket owner to have full access to the data in the bucket. This field solves the problem that the bucket owner cannot read the data written into a mounted third-party bucket. If the object storage of a third-party tenant is mounted, **default\_acl** must be set to **bucket-owner-full-control**. For details about other values of **default\_acl**, see [ACLs](#).
- **persistentVolumeReclaimPolicy:** When the object storage of a third-party tenant is mounted, this field must be set to **Retain**. In this way, the OBS bucket will not be deleted when a PV is deleted. The service platform of the SaaS service provider needs to manage the lifecycle of the third-party bucket PVs. When a PVC is deleted separately, the PV is not deleted. Instead, it will be retained. To do so, you need to call the native Kubernetes APIs to create and delete static PVs.
- **storageClassName:** You can associate a new custom OBS storage class ([click here](#)) or the built-in csi-obs of the cluster.

#### PVC of a bound OBS bucket:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
annotations:
  csi.storage.k8s.io/fstype: obsfs
  everest.io/obs-volume-type: STANDARD
  volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
name: objbucketpvc  #Replace the name with the actual PVC name of the bucket.
namespace: default
spec:
accessModes:
- ReadWriteMany
resources:
requests:
  storage: 1Gi
storageClassName: csi-obs-mountoption  #The value must be the same as the storage class associated with the bound PV.
volumeName: objbucket  #Replace the name with the actual PV name of the bucket to be bound.
```

- **Static PV of an OBS parallel file system:**

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: obsfscheck  #Replace the name with the actual PV name of the parallel file system.
annotations:
  pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
accessModes:
- ReadWriteMany
capacity:
  storage: 1Gi
mountOptions:
- default_acl=bucket-owner-full-control  #New OBS mounting parameters
csi:
  driver: obs.csi.everest.io
  fsType: obsfs
  volumeAttributes:
    everest.io/obs-volume-type: STANDARD
    everest.io/region: ap-southeast-1
    storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
  volumeHandle: obsfscheck  #Replace the name with the actual name of the parallel file system of the third-party tenant.
  persistentVolumeReclaimPolicy: Retain  #This parameter must be set to Retain to ensure that
```

the bucket will not be deleted when a PV is deleted.  
storageClassName: **csi-obs-mountoption** #You can associate a new custom OBS storage class or the built-in csi-obs of the cluster.

- **mountOptions:** This field contains the new OBS mounting parameters that allow the bucket owner to have full access to the data in the bucket. This field solves the problem that the bucket owner cannot read the data written into a mounted third-party bucket. If the object storage of a third-party tenant is mounted, **default\_acl** must be set to **bucket-owner-full-control**. For details about other values of **default\_acl**, see [ACLs](#).
- **persistentVolumeReclaimPolicy:** When the object storage of a third-party tenant is mounted, this field must be set to **Retain**. In this way, the OBS bucket will not be deleted when a PV is deleted. The service platform of the SaaS service provider needs to manage the lifecycle of the third-party bucket PVs. When a PVC is deleted separately, the PV is not deleted. Instead, it will be retained. To do so, you need to call the native Kubernetes APIs to create and delete static PVs.
- **storageClassName:** You can associate a new custom OBS storage class ([click here](#)) or the built-in csi-obs of the cluster.

PVC of a bound OBS parallel file system:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    csi.storage.k8s.io/fstype: obsfs
    everest.io/obs-volume-type: STANDARD
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
  name: obsfscheckpvc #Replace the name with the actual PVC name of the parallel file system.
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-obs-mountoption #The value must be the same as the storage class associated with the bound PV.
  volumeName: obsfscheck #Replace the name with the actual PV name of the parallel file system.
```

- **(Optional) Creating a custom OBS storage class to associate with a static PV:**

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-obs-mountoption
mountOptions:
  - default_acl=bucket-owner-full-control
parameters:
  csi.storage.k8s.io/csi-driver-name: obs.csi.everest.io
  csi.storage.k8s.io/fstype: obsfs
  everest.io/obs-volume-type: STANDARD
  provisioner: everest-csi-provisioner
  reclaimPolicy: Retain
  volumeBindingMode: Immediate
```

- **csi.storage.k8s.io/fstype:** File type. The value can be **obsfs** or **s3fs**. If the value is **s3fs**, an OBS bucket is created and mounted using s3fs. If the value is **obsfs**, an OBS parallel file system is created and mounted using obsfs.
- **reclaimPolicy:** Reclaim policy of a PV. The value will be set in **PV.spec.persistentVolumeReclaimPolicy** dynamically created based on the new PVC associated with the storage class. If the value is **Delete**, the

external OBS bucket and the PV will be deleted when the PVC is deleted. If the value is **Retain**, the PV and external storage are retained when the PVC is deleted. In this case, you need to clear the PV separately. In the scenario where an imported third-party bucket is associated, the storage class is used only for associating static PVs (with this field set to **Retain**). Dynamic creation is not involved.

## 10.3 Dynamically Creating and Mounting Subdirectories of an SFS Turbo File System

### Background

The minimum capacity of an SFS Turbo file system is 500 GB, and the SFS Turbo file system cannot be billed by usage. By default, the root directory of an SFS Turbo file system is mounted to a container which, in most case, does not require such a large capacity.

The everest add-on allows you to dynamically create subdirectories in an SFS Turbo file system and mount these subdirectories to containers. In this way, an SFS Turbo file system can be shared by multiple containers to increase storage efficiency.

### Notes and Constraints

- Only clusters of v1.15 and later are supported.
- The cluster must use the everest add-on of version 1.1.13 or later.
- Kata containers are not supported.
- A maximum of 10 PVCs can be created concurrently at a time by using the subdirectory function.

### Creating an SFS Turbo Volume of the subpath Type



The CCE console has not yet supported the operations related to this feature, such as expanding, disassociating, and deleting subPath volumes.

---

**Step 1** Import an SFS Turbo file system that is located in the same VPC and subnet as the cluster.

**Step 2** Create a StorageClass YAML file, for example, **sfturbo-sc-test.yaml**.

Configuration example:

```
apiVersion: storage.k8s.io/v1
allowVolumeExpansion: true
kind: StorageClass
metadata:
  name: sfturbo-sc-test
mountOptions:
- lock
parameters:
```

```
csi.storage.k8s.io/csi-driver-name: sfsturbo.csi-everest.io
csi.storage.k8s.io/fstype: nfs
everest.io/archive-on-delete: "true"
everest.io/share-access-to: 7ca2dba2-1234-1234-1234-626371a8fb3a
everest.io/share-expand-type: bandwidth
everest.io/share-export-location: 192.168.1.1:/sfsturbo/
everest.io/share-source: sfs-turbo
everest.io/share-volume-type: STANDARD
everest.io/volume-as: subpath
everest.io/volume-id: 0d773f2e-1234-1234-1234-de6a35074696
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

In this example:

- **name:** name of the StorageClass.
- **mountOptions:** mount options. This field is optional.
  - In versions later than everest 1.1.13 and earlier than everest 1.2.8, only the **nolock** parameter can be configured. By default, the **nolock** parameter is used for the mount operation and does not need to be configured. If **nolock** is set to **false**, the **lock** field is used.
  - Starting from everest 1.2.8, more parameters are supported. The default parameter configurations are shown below. For details, see [Setting Mount Options](#). **Do not set nolock to true. Otherwise, the mount operation fails.**
- **everest.io/volume-as:** Set this parameter to **subpath**.
- **everest.io/share-access-to:** This parameter is optional. In subpath mode, set this parameter to the ID of the VPC where the SFS Turbo file system is located.
- **everest.io/share-expand-type:** This parameter is optional. If the type of the SFS Turbo file system is SFS Turbo Standard – Enhanced or SFS Turbo Performance – Enhanced, set this parameter to **bandwidth**.
- **everest.io/share-export-location:** root directory to be mounted. It consists of the SFS Turbo shared path and sub-directory. The shared path can be queried on the SFS Turbo console. The sub-directory is user-defined. The PVCs created by the StorageClass are located in the sub-directory.
- **everest.io/share-volume-type:** This parameter is optional. It specifies the SFS Turbo file system type. The value can be **STANDARD** or **PERFORMANCE**. For enhanced types, this parameter must be used together with **everest.io/share-expand-type** (whose value should be **bandwidth**).
- **everest.io/zone:** This parameter is optional. Set it to the AZ where the SFS Turbo file system is located.
- **everest.io/volume-id:** ID of the SFS Turbo volume. You can query the volume ID on the SFS Turbo page.
- **everest.io/archive-on-delete:** If this parameter is set to **true** and the recycling policy is set to **Delete**, the original PV file will be archived when the PVC is deleted. The archive directory is named in the format of *archived-\$PV name.timestamp*. If this parameter is set to **false**, the SFS Turbo sub-directory corresponding to the PV will be deleted. The default value is **true**.

```
mountOptions:
- vers=3
- timeo=600
- nolock
- hard
```

**Step 3** Run the **kubectl create -f sfsturbo-sc-test.yaml** command to create a StorageClass.

**Step 4** Create a PVC YAML file named **sfs-turbo-test.yaml**.

Configuration example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: sfs-turbo-test
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 50Gi
  storageClassName: sfsturbo-sc-test
  volumeMode: Filesystem
```

In this example:

- **name**: name of the PVC.
- **storageClassName**: name of the StorageClass created in the previous step.
- **storage**: In the subpath mode, this parameter is invalid. The storage capacity is limited by the total capacity of the SFS Turbo file system. If the total capacity of the SFS Turbo file system is insufficient, expand the capacity on the SFS Turbo page in a timely manner.

**Step 5** Run the **kubectl create -f sfs-turbo-test.yaml** command to create a PVC.

----End



It is meaningless to conduct capacity expansion on an SFS Turbo volume created in the subpath mode. This operation does not expand the capacity of the SFS Turbo file system. You need to ensure that the total capacity of the SFS Turbo file system is not used up.

## Creating a Deployment and Mounting an Existing Volume

**Step 1** Create a Deployment YAML file named **deployment-test.yaml**.

Configuration example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-turbo-subpath-example
  namespace: default
  generation: 1
  labels:
    appgroup: ""
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-turbo-subpath-example
  template:
    metadata:
      labels:
        app: test-turbo-subpath-example
    spec:
      containers:
```

```
- image: nginx:latest
  name: container-0
  volumeMounts:
    - mountPath: /tmp
      name: pvc-sfs-turbo-example
  restartPolicy: Always
  imagePullSecrets:
    - name: default-secret
  volumes:
    - name: pvc-sfs-turbo-example
      persistentVolumeClaim:
        claimName: sfs-turbo-test
```

In this example:

- **name:** name of the Deployment.
- **image:** image used by the Deployment.
- **mountPath:** mount path of the container. In this example, the volume is mounted to the **/tmp** directory.
- **claimName:** name of an existing PVC.

**Step 2** Run the **kubectl create -f deployment-test.yaml** command to create a Deployment.

----End

## Creating a StatefulSet That Uses a Volume Dynamically Created in subpath Mode

**Step 1** Create a StatefulSet YAML file named **statefulset-test.yaml**.

Configuration example:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: test-turbo-subpath
  namespace: default
  generation: 1
  labels:
    appgroup: ""
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test-turbo-subpath
  template:
    metadata:
      labels:
        app: test-turbo-subpath
      annotations:
        metrics.alpha.kubernetes.io/custom-endpoints: '[{"api": "", "path": "", "port": "", "names": ""}]'
        pod.alpha.kubernetes.io/initialized: 'true'
    spec:
      containers:
        - name: container-0
          image: 'nginx:latest'
          env:
            - name: PAAS_APP_NAME
              value: deploy-sfs-nfs-rw-in
            - name: PAAS_NAMESPACE
              value: default
            - name: PAAS_PROJECT_ID
              value: 8190a2a1692c46f284585c56fc0e2fb9
```

```
resources: {}
volumeMounts:
- name: sfs-turbo-160024548582479676
  mountPath: /tmp
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  imagePullPolicy: IfNotPresent
restartPolicy: Always
terminationGracePeriodSeconds: 30
dnsPolicy: ClusterFirst
securityContext: {}
imagePullSecrets:
- name: default-secret
affinity: {}
schedulerName: default-scheduler
volumeClaimTemplates:
- metadata:
  name: sfs-turbo-160024548582479676
  namespace: default
  annotations: {}
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: sfsturbo-sc-test
serviceName: www
podManagementPolicy: OrderedReady
updateStrategy:
  type: RollingUpdate
revisionHistoryLimit: 10
```

In this example:

- **name:** name of the StatefulSet.
- **image:** image used by the StatefulSet.
- **mountPath:** mount path of the container. In this example, the volume is mounted to the `/tmp` directory.
- **spec.template.spec.containers.volumeMounts.name** and **spec.volumeClaimTemplates.metadata.name** must be consistent because they have a mapping relationship.
- **storageClassName:** name of the created StorageClass.

**Step 2** Run the `kubectl create -f statefulset-test.yaml` command to create a StatefulSet.

----End

## 10.4 How Do I Change the Storage Class Used by a Cluster of v1.15 from FlexVolume to CSI Everest?

In clusters later than v1.15.11-r1, CSI (the everest add-on) has taken over all functions of fuxi FlexVolume (the storage-driver add-on) for managing container storage. You are advised to use CSI Everest.

To migrate your storage volumes, create a static PV to associate with the original underlying storage, and then create a PVC to associate with this static PV. When you upgrade your application, mount the new PVC to the original mounting path to migrate the storage volumes.



Services will be interrupted during the migration. Therefore, properly plan the migration and back up data.

## Procedure

**Step 1** (Optional) Back up data to prevent data loss in case of exceptions.

**Step 2** Configure a YAML file of the PV in the CSI format according to the PV in the FlexVolume format and associate the PV with the existing storage.

To be specific, run the following commands to configure the `pv-example.yaml` file, which is used to create a PV.

**touch pv-example.yaml**

**vi pv-example.yaml**

Configuration example of a PV for an EVS volume:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone: <zone name>
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
    name: pv-evs-example
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 10Gi
  csi:
    driver: disk.csi.everest.io
    fsType: ext4
    volumeAttributes:
      everest.io/disk-mode: SCSI
      everest.io/disk-volume-type: SAS
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      volumeHandle: 0992dbda-6340-470e-a74e-4f0db288ed82
    persistentVolumeReclaimPolicy: Delete
    storageClassName: csi-disk
```

Pay attention to the fields in bold and red. The parameters are described as follows:

**Table 10-1** EVS volume configuration parameters

Parameter	Description
failure-domain.beta.kubernetes.io/region	Region where the EVS disk is located. Use the same value as that of the FlexVolume PV.
failure-domain.beta.kubernetes.io/zone	AZ where the EVS disk is located. Use the same value as that of the FlexVolume PV.

Parameter	Description
name	Name of the PV, which must be unique in the cluster.
storage	EVS volume capacity in the unit of Gi. Use the value of <b>spec.capacity.storage</b> of the FlexVolume PV.
driver	Storage driver used to attach the volume. Set the driver to <b>disk.csi.everest.io</b> for the EVS volume.
volumeHandle	Volume ID of the EVS disk. Use the value of <b>spec.flexVolume.options.volumeID</b> of the FlexVolume PV.
everest.io/disk-mode	EVS disk mode. Use the value of <b>spec.flexVolume.options.disk-mode</b> of the FlexVolume PV.
everest.io/disk-volume-type	EVS disk type. Currently, high I/O (SAS) and ultra-high I/O (SSD) are supported. Use the value of <b>kubernetes.io/volumetype</b> in the storage class corresponding to <b>spec.storageClassName</b> of the FlexVolume PV.
storageClassName	Name of the Kubernetes storage class associated with the storage volume. Set this field to <b>csi-disk</b> for EVS disks.

#### Configuration example of a PV for an SFS volume:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-sfs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
    - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: nas.csi.everest.io
    fsType: nfs
    volumeAttributes:
      everest.io/share-export-location: sfs-nas01.ap-southeast-1.myhuaweicloud.com:/share-436304e8
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      volumeHandle: 682f00bb-ace0-41d8-9b3e-913c9aa6b695
    persistentVolumeReclaimPolicy: Delete
    storageClassName: csi-nas
```

Pay attention to the fields in bold and red. The parameters are described as follows:

**Table 10-2** SFS volume configuration parameters

Parameter	Description
name	Name of the PV, which must be unique in the cluster.

Parameter	Description
storage	File storage size in the unit of Gi. Use the value of <b>spec.capacity.storage</b> of the FlexVolume PV.
driver	Storage driver used to attach the volume. Set the driver to <b>nas.csi.everest.io</b> for the file system.
everest.io/share-export-location	Shared path of the file system. Use the value of <b>spec.flexVolume.options.deviceMountPath</b> of the FlexVolume PV.
volumeHandle	File system ID. Use the value of <b>spec.flexVolume.options.volumeID</b> of the FlexVolume PV.
storageClassName	Name of the Kubernetes storage class. Set this field to <b>csi-nas</b> .

Configuration example of a PV for an OBS volume:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-obs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
    - ReadWriteMany
  capacity:
    storage: 1Gi
  csi:
    driver: obs.csi.everest.io
    fsType: s3fs
    volumeAttributes:
      everest.io/obs-volume-type: STANDARD
      everest.io/region: ap-southeast-1
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      volumeHandle: obs-normal-static-pv
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-obs
```

Pay attention to the fields in bold and red. The parameters are described as follows:

**Table 10-3** OBS volume configuration parameters

Parameter	Description
name	Name of the PV, which must be unique in the cluster.
storage	Storage capacity, in the unit of Gi. Set this parameter to the fixed value <b>1Gi</b> .
driver	Storage driver used to attach the volume. Set the driver to <b>obs.csi.everest.io</b> for the OBS volume.

Parameter	Description
fsType	File type. Value options are <b>obsfs</b> or <b>s3fs</b> . If the value is <b>s3fs</b> , an OBS bucket is created and mounted using s3fs. If the value is <b>obsfs</b> , an OBS parallel file system is created and mounted using obsfs. Set this parameter according to the value of <b>spec.flexVolume.options.posix</b> of the FlexVolume PV. If the value of <b>spec.flexVolume.options.posix</b> is <b>true</b> , set this parameter to <b>obsfs</b> . If the value is <b>false</b> , set this parameter to <b>s3fs</b> .
everest.io/obs-volume-type	Storage class, including <b>STANDARD</b> (standard bucket) and <b>WARM</b> (infrequent access bucket). Set this parameter according to the value of <b>spec.flexVolume.options.storage_class</b> of the FlexVolume PV. If the value of <b>spec.flexVolume.options.storage_class</b> is <b>standard</b> , set this parameter to <b>STANDARD</b> . If the value is <b>standard_ia</b> , set this parameter to <b>WARM</b> .
everest.io/region	Region where the OBS bucket is located. Use the value of <b>spec.flexVolume.options.region</b> of the FlexVolume PV.
volumeHandle	OBS bucket name. Use the value of <b>spec.flexVolume.options.volumeID</b> of the FlexVolume PV.
storageClassName	Name of the Kubernetes storage class. Set this field to <b>csi-obs</b> .

#### Configuration example of a PV for an SFS Turbo volume:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-efs-example
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner
spec:
  accessModes:
    - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: sfsturbo.csi.everest.io
    fsType: nfs
    volumeAttributes:
      everest.io/share-export-location: 192.168.0.169:/
      storage.kubernetes.io/csiProvisionerIdentity: everest-csi-provisioner
      volumeHandle: 8962a2a2-a583-4b7f-bb74-fe76712d8414
  persistentVolumeReclaimPolicy: Delete
  storageClassName: csi-sfsturbo
```

Pay attention to the fields in bold and red. The parameters are described as follows:

**Table 10-4** SFS Turbo volume configuration parameters

Parameter	Description
name	Name of the PV, which must be unique in the cluster.
storage	File system size. Use the value of <b>spec.capacity.storage</b> of the FlexVolume PV.
driver	Storage driver used to attach the volume. Set it to <b>sfsturbo.csi.everest.io</b> .
everest.io/share-export-location	Shared path of the SFS Turbo volume. Use the value of <b>spec.flexVolume.options.deviceMountPath</b> of the FlexVolume PV.
volumeHandle	SFS Turbo volume ID. Use the value of <b>spec.flexVolume.options.volumeID</b> of the FlexVolume PV.
storageClassName	Name of the Kubernetes storage class. Set this field to <b>csi-sfsturbo</b> for SFS Turbo volumes.

**Step 3** Configure a YAML file of the PVC in the CSI format according to the PVC in the FlexVolume format and associate the PVC with the PV created in **Step 2**.

To be specific, run the following commands to configure the pvc-example.yaml file, which is used to create a PVC.

```
touch pvc-example.yaml
```

```
vi pvc-example.yaml
```

Configuration example of a PVC for an EVS volume:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    failure-domain.beta.kubernetes.io/region: ap-southeast-1
    failure-domain.beta.kubernetes.io/zone: <zone name>
  annotations:
    everest.io/disk-volume-type: SAS
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
  name: pvc-evs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  volumeName: pv-evs-example
  storageClassName: csi-disk
```

Pay attention to the fields in bold and red. The parameters are described as follows:

**Table 10-5** PVC configuration parameters for an EVS volume

Parameter	Description
failure-domain.beta.kubernetes.io/region	Region where the cluster is located. Use the same value as that of the FlexVolume PVC.
failure-domain.beta.kubernetes.io/zone	AZ where the EVS disk is deployed. Use the same value as that of the FlexVolume PVC.
everest.io/disk-volume-type	Storage class of the EVS disk. The value can be <b>SAS</b> or <b>SSD</b> . Set this parameter to the same value as that of the PV created in <a href="#">Step 2</a> .
name	PVC name, which must be unique in the namespace. The value must be unique in the namespace. (If the PVC is dynamically created by a stateful application, the value of this parameter must be the same as the name of the FlexVolume PVC.)
namespace	Namespace to which the PVC belongs. Use the same value as that of the FlexVolume PVC.
storage	Requested capacity of the PVC, which must be the same as the storage size of the existing PV.
volumeName	Name of the PV. Set this parameter to the name of the static PV in <a href="#">Step 2</a> .
storageClassName	Name of the Kubernetes storage class. Set this field to <b>csi-disk</b> for EVS disks.

**Configuration example of a PVC for an SFS volume:**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
  name: pvc-sfs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-nas
  volumeName: pv-sfs-example
```

Pay attention to the fields in bold and red. The parameters are described as follows:

**Table 10-6** PVC configuration parameters for an SFS volume

Parameter	Description
name	PVC name, which must be unique in the namespace. The value must be unique in the namespace. (If the PVC is dynamically created by a stateful application, the value of this parameter must be the same as the name of the FlexVolume PVC.)
namespace	Namespace to which the PVC belongs. Use the same value as that of the FlexVolume PVC.
storage	Storage capacity, in the unit of Gi. The value must be the same as the storage size of the existing PV.
storageClassName	Set this field to <b>csi-nas</b> .
volumeName	Name of the PV. Set this parameter to the name of the static PV in <a href="#">Step 2</a> .

Configuration example of a **PVC for an OBS volume**:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
    everest.io/obs-volume-type: STANDARD
    csi.storage.k8s.io/fstype: s3fs
  name: pvc-obs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-obs
  volumeName: pv-obs-example
```

Pay attention to the fields in bold and red. The parameters are described as follows:

**Table 10-7** PVC configuration parameters for an OBS volume

Parameter	Description
everest.io/obs-volume-type	OBS volume type, which can be <b>STANDARD</b> (standard bucket) and <b>WARM</b> (infrequent access bucket). Set this parameter to the same value as that of the PV created in <a href="#">Step 2</a> .
csi.storage.k8s.io/fstype	File type, which can be <b>obsfs</b> or <b>s3fs</b> . The value must be the same as that of <b>fsType</b> of the static OBS volume PV.

Parameter	Description
name	PVC name, which must be unique in the namespace. The value must be unique in the namespace. (If the PVC is dynamically created by a stateful application, the value of this parameter must be the same as the name of the FlexVolume PVC.)
namespace	Namespace to which the PVC belongs. Use the same value as that of the FlexVolume PVC.
storage	Storage capacity, in the unit of Gi. Set this parameter to the fixed value <b>1Gi</b> .
storageClassName	Name of the Kubernetes storage class. Set this field to <b>csi-obs</b> .
volumeName	Name of the PV. Set this parameter to the name of the static PV created in <a href="#">Step 2</a> .

Configuration example of a PVC for an SFS Turbo volume:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner
  name: pvc-efs-example
  namespace: default
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-sfsturbo
  volumeName: pv-efs-example
```

Pay attention to the fields in bold and red. The parameters are described as follows:

**Table 10-8** PVC configuration parameters for an SFS Turbo volume

Parameter	Description
name	PVC name, which must be unique in the namespace. The value must be unique in the namespace. (If the PVC is dynamically created by a stateful application, the value of this parameter must be the same as the name of the FlexVolume PVC.)
namespace	Namespace to which the PVC belongs. Use the same value as that of the FlexVolume PVC.
storageClassName	Name of the Kubernetes storage class. Set this field to <b>csi-sfsturbo</b> .

Parameter	Description
storage	Storage capacity, in the unit of Gi. The value must be the same as the storage size of the existing PV.
volumeName	Name of the PV. Set this parameter to the name of the static PV created in <a href="#">Step 2</a> .

#### Step 4 Upgrade the workload to use a new PVC.

##### For Deployments

- Run the **kubectl create -f** commands to create a PV and PVC.

**kubectl create -f pv-example.yaml**

**kubectl create -f pvc-example.yaml**



Replace the example file name **pvc-example.yaml** in the preceding commands with the names of the YAML files configured in [Step 2](#) and [Step 3](#).

- Go to the CCE console. On the workload upgrade page, click **Upgrade > Advanced Settings > Data Storage > Cloud Storage**.

- Uninstall the old storage and add the PVC in the CSI format. Retain the original mounting path in the container.
- Click **Submit**.
- Wait until the pods are running.

##### For StatefulSets that use existing storage

- Run the **kubectl create -f** commands to create a PV and PVC.

**kubectl create -f pv-example.yaml**

**kubectl create -f pvc-example.yaml**



Replace the example file name **pvc-example.yaml** in the preceding commands with the names of the YAML files configured in [Step 2](#) and [Step 3](#).

- Run the **kubectl edit** command to edit the StatefulSet and use the newly created PVC.

**kubectl edit sts sts-example -n xxx**

```

30      pod.alpha.Kubernetes.io/initialized: "true"
31  spec:
32    volumes:
33      - name: cce-efs-import-kjxmtzqn-z65j
34        persistentVolumeClaim:
35          claimName: pvc-csi-sfsturbo-f2ed93a7-468c-49c3-9a8b-9ded5c6e1533-1
36    containers:

```

 NOTE

Replace **sts-example** in the preceding command with the actual name of the StatefulSet to upgrade. **xxx** indicates the namespace to which the StatefulSet belongs.

3. Wait until the pods are running.

 NOTE

The current console does not support the operation of adding new cloud storage for StatefulSets. Use the kubectl commands to replace the storage with the newly created PVC.

#### For StatefulSets that use dynamically allocated storage

1. Back up the PV and PVC in the flexVolume format used by the StatefulSet.

**kubectl get pvc xxx -n {namespaces} -oyaml > pvc-backup.yaml**

**kubectl get pv xxx -n {namespaces} -oyaml > pv-backup.yaml**

2. Change the number of pods to **0**.

3. On the storage page, disassociate the flexVolume PVC used by the StatefulSet.

4. Run the **kubectl create -f** commands to create a PV and PVC.

**kubectl create -f pv-example.yaml**

**kubectl create -f pvc-example.yaml**

 NOTE

Replace the example file name **pvc-example.yaml** in the preceding commands with the names of the YAML files configured in [Step 2](#) and [Step 3](#).

5. Change the number of pods back to the original value and wait until the pods are running.

 NOTE

The dynamic allocation of storage for StatefulSets is achieved by using **volumeClaimTemplates**. This field cannot be modified by Kubernetes. Therefore, data cannot be migrated by using a new PVC.

The PVC naming rule of the **volumeClaimTemplates** is fixed. When a PVC that meets the naming rule exists, this PVC is used.

Therefore, disassociate the original PVC first, and then create a PVC with the same name in the CSI format.

6. (Optional) Recreate the stateful application to ensure that a CSI PVC is used when the application is scaled out. Otherwise, FlexVolume PVCs are used in scaling out.

- Run the following command to obtain the YAML file of the StatefulSet:

**kubectl get sts xxx -n {namespaces} -oyaml > sts.yaml**

- Run the following command to back up the YAML file of the StatefulSet:

**cp sts.yaml sts-backup.yaml**

- Modify the definition of **volumeClaimTemplates** in the YAML file of the StatefulSet.

**vi sts.yaml**

### Configuration example of **volumeClaimTemplates** for an EVS volume:

```
volumeClaimTemplates:  
  - metadata:  
      name: pvc-161070049798261342  
      namespace: default  
      creationTimestamp: null  
      annotations:  
        everest.io/disk-volume-type: SAS  
    spec:  
      accessModes:  
        - ReadWriteOnce  
      resources:  
        requests:  
          storage: 10Gi  
      storageClassName: csi-disk
```

The parameter value must be the same as the PVC of the EVS volume created in [Step 3](#).

### Configuration example of **volumeClaimTemplates** for an SFS volume:

```
volumeClaimTemplates:  
  - metadata:  
      name: pvc-161063441560279697  
      namespace: default  
      creationTimestamp: null  
    spec:  
      accessModes:  
        - ReadWriteMany  
      resources:  
        requests:  
          storage: 10Gi  
      storageClassName: csi-nas
```

The parameter value must be the same as the PVC of the SFS volume created in [Step 3](#).

### Configuration example of **volumeClaimTemplates** for an OBS volume:

```
volumeClaimTemplates:  
  - metadata:  
      name: pvc-161070100417416148  
      namespace: default  
      creationTimestamp: null  
      annotations:  
        csi.storage.k8s.io/fstype: s3fs  
        everest.io/obs-volume-type: STANDARD  
    spec:  
      accessModes:  
        - ReadWriteMany  
      resources:  
        requests:  
          storage: 1Gi  
      storageClassName: csi-obs
```

The parameter value must be the same as the PVC of the OBS volume created in [Step 3](#).

- Delete the StatefulSet.

**kubectl delete sts xxx -n {namespaces}**

- Create the StatefulSet.

**kubectl create -f sts.yaml**

**Step 5** Check service functions.

1. Check whether the application is running properly.
2. Checking whether the data storage is normal.



If a rollback is required, perform **Step 4**. Select the PVC in FlexVolume format and upgrade the application.

**Step 6** Uninstall the PVC in the FlexVolume format.

If the application functions normally, unbind the PVC in the FlexVolume format on the storage management page.

You can also run the kubectl command to delete the PVC and PV of the FlexVolume format.



Before deleting a PV, change the persistentVolumeReclaimPolicy of the PV to **Retain**. Otherwise, the underlying storage will be reclaimed after the PV is deleted.

If the cluster has been upgraded before the storage migration, PVs may fail to be deleted. You can remove the PV protection field **finalizers** to delete PVs.

```
kubectl patch pv {pv_name} -p '{"metadata":{"finalizers":null}}'
```

----End

## 10.5 Using OBS Parallel File Systems

Parallel File System (PFS), a sub-product of OBS, is a high-performance file system, with access latency in milliseconds. PFS can support a bandwidth performance up to the TB/s level and supports millions of IOPS, which makes it ideal for running high-performance computing (HPC) workloads. PFS is more stable than object storage. For details, see [About Parallel File System](#).

In commercial deployments, **you are advised to use parallel file systems instead of object storage**.

### Technical Description

CCE allows you to mount OBS parallel file systems by using **obsfs**. For details about obsfs, see [Introduction to obsfs](#).

An obsfs resident process is generated each time an object storage volume generated from the parallel file system is attached. Example:

```
[root@cluster-1198-prr-58064 ~]# ps -aux | grep obsfs
root      7533  0.1 53280 44048 ?        00:00:09 0:00 /usr/bin/obsfs pvc-e17fb8d-3367-4814-9e32-fba55d93cf1 /mnt/pas/kubernetes/kubelet/pods/49995932-36ac-41b8-1e9b-bb32e168bd01/volumes/kubernetes.io-csi/pvc-e17fb8d-3367-4814-9e32-fba55d93cf1 /mnt/pas/kubernetes/kubelet/pods/49995932-36ac-41b8-1e9b-bb32e168bd01/volumes/kubernetes.io-csi/pvc-e17fb8d-3367-4814-9e32-fba55d93cf1 -o 8
lunw other -o maxconcurrent=1000 -o max writes=1000000 -o max write=1000000 -o max background=100 -o use inc -o no check certificate -o umask=0
```

## Recommended Usage

You are advised to reserve 1 GB memory for each obsfs process. For example, for a node with 4 CPUs and 8 GB memory, the obsfs parallel file systems should be mounted to **no more than eight pods**.



obsfs resident processes run on the node. If the consumed memory exceeds the upper limit of the node, the node becomes abnormal.

On a node with 4 CPUs and 8 GB memory, if more than 100 pods are mounted with parallel file systems, the node will be unavailable.

You are advised to control the number of pods mounted with parallel file systems on a single node.

## Performance Metrics

**Table 10-9** lists the performance test result of obsfs resident processes (for reference only).

**Table 10-9** obsfs resource consumption

Test Item	Memory Usage
Long-term stable running	About 50 MB
Concurrently writing files of 10 MB (two processes)	About 110 MB
Concurrently writing files of 10 MB (four processes)	About 220 MB
Writing files of 100 GB (single process)	About 300 MB

## Verification Environment

Cluster version: 1.15.11

Add-on version: 1.2.0

obsfs version: 1.83 (commit:97e919f)

## 10.6 Custom Storage Classes

### Challenges

When using storage resources in CCE, the most common method is to specify **storageClassName** to define the type of storage resources to be created when creating a PVC. The following configuration shows how to use a PVC to apply for an SAS (high I/O) EVS disk (block storage).

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
name: pvc-evs-example
namespace: default
annotations:
  everest.io/disk-volume-type: SAS
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk
```

If you need to specify the EVS disk type, you can set the **everest.io/disk-volume-type** field. The value **SAS** is used as an example here, indicating the high I/O EVS disk type. Or you can choose **SSD** (ultra-high I/O).

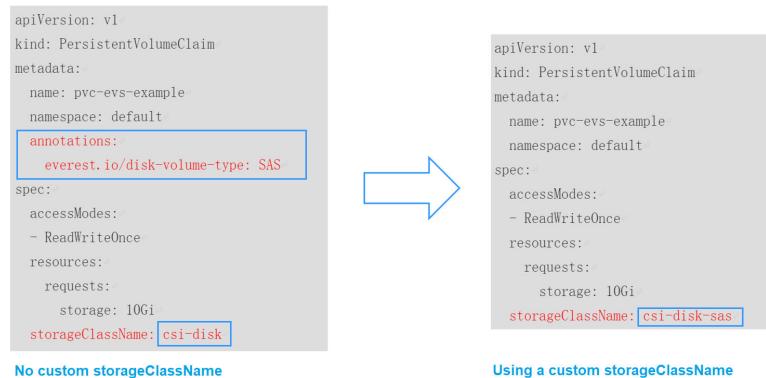
This configuration method may not work if you want to:

- Set **storageClassName** only, which is simpler than specifying the EVS disk type by using **everest.io/disk-volume-type**.
- Avoid modifying YAML files or Helm charts. Some users switch from self-built or other Kubernetes services to CCE and have written YAML files of many applications. In these YAML files, different types of storage resources are specified by different StorageClassNames. When using CCE, they need to modify a large number of YAML files or Helm charts to use storage resources, which is labor-consuming and error-prone.
- Set the default **storageClassName** for all applications to use the default storage class. In this way, you can create storage resources of the default type without needing to specify **storageClassName** in the YAML file.

## Solution

This section describes how to set a custom storage class in CCE and how to set the default storage class. You can specify different types of storage resources by setting **storageClassName**.

- For the first scenario, you can define custom storageClassNames for SAS and SSD EVS disks. For example, define a storage class named **csi-disk-sas** for creating SAS disks. The following figure shows the differences before and after you use a custom storage class.



- For the second scenario, you can define a storage class with the same name as that in the existing YAML file without needing to modify **storageClassName** in the YAML file.

- For the third scenario, you can set the default storage class as described below to create storage resources without specifying **storageClassName** in YAML files.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-evs-example
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

## Storage Classes in CCE

Run the following command to query the supported storage classes.

```
# kubectl get sc
NAME           PROVISIONER          AGE
csi-disk       everest-csi-provisioner   17d   # Storage class for EVS disks
csi-disk-topology everest-csi-provisioner   17d   # Storage class for EVS disks with delayed association
csi-nas        everest-csi-provisioner   17d   # Storage class for SFS file systems
csi-obs        everest-csi-provisioner   17d   # Storage Class for OBS buckets
csi-sfsturbo   everest-csi-provisioner   17d   # Storage class for SFS Turbo file systems
```

Check the details of **csi-disk**. You can see that the type of the disk created by **csi-disk** is SAS by default.

```
# kubectl get sc csi-disk -oyaml
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  creationTimestamp: "2021-03-17T02:10:32Z"
  name: csi-disk
  resourceVersion: "760"
  selfLink: /apis/storage.k8s.io/v1/storageclasses/csi-disk
  uid: 4db97b6c-853b-443d-b0dc-41cdcb8140f2
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

## Custom Storage Classes

You can customize a high I/O storage class in a YAML file. For example, the name **csi-disk-sas** indicates that the disk type is SAS (high I/O).

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-sas          # Name of the high I/O storage class, which can be customized.
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SAS      # High I/O EVS disk type, which cannot be customized.
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
```

```
volumeBindingMode: Immediate
allowVolumeExpansion: true # true indicates that capacity expansion is allowed.
```

For an ultra-high I/O storage class, you can set the class name to **csi-disk-ssd** to create SSD EVS disk (ultra-high I/O).

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-ssd # Name of the ultra-high I/O storage class, which can be customized.
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi-everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SSD # Ultra-high I/O EVS disk type, which cannot be customized.
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

**reclaimPolicy:** indicates the recycling policies of the underlying cloud storage. The value can be **Delete** or **Retain**.

- **Delete:** When a PVC is deleted, both the PV and the EVS disk are deleted.
- **Retain:** When a PVC is deleted, the PV and underlying storage resources are not deleted. Instead, you must manually delete these resources. After that, the PV resource is in the **Released** state and cannot be bound to the PVC again.

#### NOTE

The reclamation policy set here has no impact on the SFS Turbo storage. Therefore, the yearly/monthly SFS Turbo resources will not be reclaimed when the cluster or PVC is deleted.

If high data security is required, you are advised to select **Retain** to prevent data from being deleted by mistake.

After the definition is complete, run the **kubectl create** commands to create storage resources.

```
# kubectl create -f sas.yaml
storageclass.storage.k8s.io/csi-disk-sas created
# kubectl create -f ssd.yaml
storageclass.storage.k8s.io/csi-disk-ssd created
```

Query the storage class again. Two more types of storage classes are displayed in the command output, as shown below.

```
# kubectl get sc
NAME      PROVISIONER          AGE
csi-disk   everest-csi-provisioner  17d
csi-disk-sas everest-csi-provisioner 2m28s
csi-disk-ssd everest-csi-provisioner 16s
csi-disk-topology everest-csi-provisioner 17d
csi-nas    everest-csi-provisioner  17d
csi-obs    everest-csi-provisioner  17d
csi-sfsturbo everest-csi-provisioner 17d
```

Other types of storage resources can be defined in the similar way. You can use **kubectl** to obtain the YAML file and modify it as required.

- **File storage**

```
# kubectl get sc csi-nas -oyaml
kind: StorageClass
```

```
apiVersion: storage.k8s.io/v1
metadata:
  name: csi-nas
spec:
  provisioner: everest-csi-provisioner
  parameters:
    csi.storage.k8s.io/csi-driver-name: nas.csi-everest.io
    csi.storage.k8s.io/fstype: nfs
    everest.io/share-access-level: rw
    everest.io/share-access-to: 5e3864c6-e78d-4d00-b6fd-de09d432c632 # ID of the VPC to which the
    cluster belongs
    everest.io/share-is-public: 'false'
    everest.io/zone: xxxxx # AZ
  reclaimPolicy: Delete
  allowVolumeExpansion: true
  volumeBindingMode: Immediate
```

- Object storage

```
# kubectl get sc csi-obs -oyaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: csi-obs
spec:
  provisioner: everest-csi-provisioner
  parameters:
    csi.storage.k8s.io/csi-driver-name: obs.csi-everest.io
    csi.storage.k8s.io/fstype: s3fs # Object storage type. s3fs indicates an object bucket, and obfs
    indicates a parallel file system.
    everest.io/obs-volume-type: STANDARD # Storage class of the OBS bucket
  reclaimPolicy: Delete
  volumeBindingMode: Immediate
```

## Specifying an Enterprise Project for Storage Classes

CCE allows you to specify an enterprise project when creating EVS disks and OBS PVCs. The created storage resources (EVS disks and OBS) belong to the specified enterprise project. **The enterprise project can be the enterprise project to which the cluster belongs or the default enterprise project.**

If you do no specify any enterprise project, the enterprise project in StorageClass is used by default. The created storage resources by using the csi-disk and csi-obs storage classes of CCE belong to the default enterprise project.

If you want the storage resources created from the storage classes to be in the same enterprise project as the cluster, you can customize a storage class and specify the enterprise project ID, as shown below.

### NOTE

To use this function, the everest add-on must be upgraded to 1.2.33 or later.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: csi-disk-epid #Customize a storage class name.
spec:
  provisioner: everest-csi-provisioner
  parameters:
    csi.storage.k8s.io/csi-driver-name: disk.csi-everest.io
    csi.storage.k8s.io/fstype: ext4
    everest.io/disk-volume-type: SAS
    everest.io/enterprise-project-id: 86bfc701-9d9e-4871-a318-6385aa368183 #Specify the enterprise project
    ID.
    everest.io/passthrough: 'true'
  reclaimPolicy: Delete
  allowVolumeExpansion: true
  volumeBindingMode: Immediate
```

## Setting a Default Storage Class

You can specify a storage class as the default class. In this way, if you do not specify **storageClassName** when creating a PVC, the PVC is created using the default storage class.

For example, to specify **csi-disk-ssd** as the default storage class, edit your YAML file as follows:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-ssd
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" # Specifies the default storage class in a cluster. A cluster can have only one default storage class.
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi-everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SSD
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

Delete the created csi-disk-ssd disk, run the **kubectl create** command to create a csi-disk-ssd disk again, and then query the storage class. The following information is displayed.

```
# kubectl delete sc csi-disk-ssd
storageclass.storage.k8s.io "csi-disk-ssd" deleted
# kubectl create -f ssd.yaml
storageclass.storage.k8s.io/csi-disk-ssd created
# kubectl get sc
NAME      PROVISIONER      AGE
csi-disk   everest-csi-provisioner  17d
csi-disk-sas   everest-csi-provisioner  114m
csi-disk-ssd (default)   everest-csi-provisioner  9s
csi-disk-topology   everest-csi-provisioner  17d
csi-nas   everest-csi-provisioner  17d
csi-obs   everest-csi-provisioner  17d
csi-sfsturbo   everest-csi-provisioner  17d
```

## Verification

- Use **csi-disk-sas** to create a PVC.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: sas-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk-sas
```

Create a storage class and view its details. As shown below, the object can be created and the value of **STORAGECLASS** is **csi-disk-sas**.

```
# kubectl create -f sas-disk.yaml
persistentvolumeclaim/sas-disk created
# kubectl get pvc
NAME      STATUS  VOLUME          CAPACITY  ACCESS MODES
STORAGECLASS  AGE
```

```
sas-disk Bound pvc-6e2f37f9-7346-4419-82f7-b42e79f7964c 10Gi RWO csi-disk-sas
24s
# kubectl get pv
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS
CLAIM STORAGECLASS REASON AGE
pvc-6e2f37f9-7346-4419-82f7-b42e79f7964c 10Gi RWO Delete Bound default/
sas-disk csi-disk-sas 30s
```

View the PVC details on the CCE console. On the PV details page, you can see that the disk type is high I/O.

PV Name	3bd17	Creation Method	Unknown
PV Status	Bound	Storage Class	EVS disk
Access Mode	ReadWriteOnce	Sub-class	High I/O
PV Reclaim Policy	Delete	Capacity	10 GB
PV Created	Apr 26, 2021 16:26:57 GMT+08:00	Volume ID	jc956
PV ID	i8de		

- If **storageClassName** is not specified, the default configuration is used, as shown below.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ssd-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Create and view the storage resource. You can see that the storage class of PVC ssd-disk is csi-disk-ssd, indicating that csi-disk-ssd is used by default.

```
# kubectl create -f ssd-disk.yaml
persistentvolumeclaim/ssd-disk created
# kubectl get pvc
NAME STATUS VOLUME CAPACITY ACCESS MODES
STORAGECLASS AGE
sas-disk Bound pvc-6e2f37f9-7346-4419-82f7-b42e79f7964c 10Gi RWO csi-disk-sas
16m
ssd-disk Bound pvc-4d2b059c-0d6c-44af-9994-f74d01c78731 10Gi RWO csi-disk-ssd
10s
# kubectl get pv
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS
CLAIM STORAGECLASS REASON AGE
pvc-4d2b059c-0d6c-44af-9994-f74d01c78731 10Gi RWO Delete Bound
default/ssd-disk csi-disk-ssd 15s
pvc-6e2f37f9-7346-4419-82f7-b42e79f7964c 10Gi RWO Delete Bound default/
sas-disk csi-disk-sas 17m
```

View the PVC details on the CCE console. On the PV details page, you can see that the disk type is ultra-high I/O.

PV Name	pvc-39e488d1-07af-4590-b568-3bd9f9e3bd17	Creation Method	Unknown
PV Status	Bound	Storage Class	EVS disk
Access Mode	ReadWriteOnce	Sub-class	Ultra-high I/O
PV Reclaim Policy	Delete	Capacity	10 GB
PV Created	Apr 26, 2021 16:26:57 GMT+08:00	Volume ID	fae10df4-d9f4-476f-8f98-734dc0fc956
PV ID	55c2df67-5f1b-4011-8c06-d0eed34c98de		

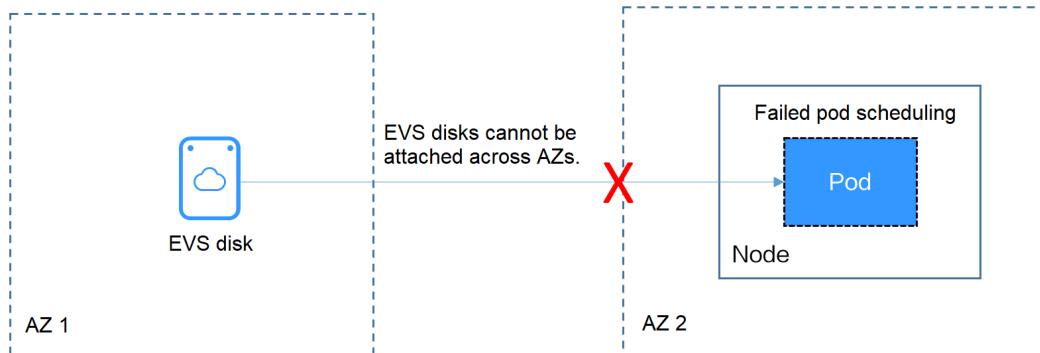
## 10.7 Realizing Automatic Topology for EVS Disks When Nodes Are Deployed Across AZs (csi-disk-topology)

### Challenges

EVS disks cannot be attached across AZs. For example, EVS disks in AZ 1 cannot be attached to nodes in AZ 2.

If the storage class csi-disk is used for StatefulSets, when a StatefulSet is scheduled, a PVC and a PV are created immediately (an EVS disk is created along with the PV), and then the PVC is bound to the PV.

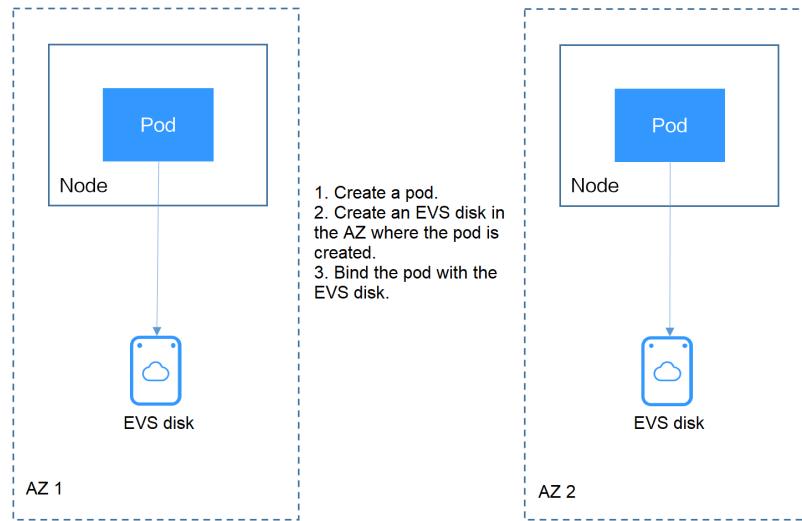
However, when the cluster nodes are located in multiple AZs, the EVS disk created by the PVC and the node to which the pods are scheduled may be in different AZs. As a result, the pods fail to be scheduled.



### Solution

CCE provides a storage class named **csi-disk-topology**. When you use this storage class to create a PVC, no PV will be created in pace with the PVC. Instead, the PV is created in the AZ of the node where the pod will be scheduled. An EVS disk is then created in the same AZ to ensure that the EVS disk can be attached and the pod can be successfully scheduled.

csi-disk-topology postpones the binding between a PVC and a PV for a while.



## Failed Pod Scheduling Due to csi-disk Used in Cross-AZ Node Deployment

Create a cluster with three nodes in different AZs.

Use the csi-disk storage class to create a StatefulSet and check whether the workload is successfully created.

```

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  serviceName: nginx          # Name of the headless Service
  replicas: 4
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: container-0
        image: nginx:alpine
        resources:
          limits:
            cpu: 600m
            memory: 200Mi
          requests:
            cpu: 600m
            memory: 200Mi
        volumeMounts:           # Storage mounted to the pod
          - name: data
            mountPath: /usr/share/nginx/html    # Mount the storage to /usr/share/nginx/html.
    imagePullSecrets:
      - name: default-secret
  volumeClaimTemplates:
    - metadata:
        name: data
        annotations:
          everest.io/disk-volume-type: SAS
    spec:
      accessModes:
        - ReadWriteOnce
      resources:

```

```
requests:
  storage: 1Gi
  storageClassName: csi-disk
```

The StatefulSet uses the following headless Service.

```
apiVersion: v1
kind: Service      # Object type (Service)
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
    - name: nginx  # Name of the port for communication between pods
      port: 80      # Port number for communication between pods
  selector:
    app: nginx    # Select the pod whose label is app:nginx.
  clusterIP: None  # Set this parameter to None, indicating the headless Service.
```

After the creation, check the PVC and pod status. In the following output, the PVC has been created and bound successfully, and a pod is in the Pending state.

```
# kubectl get pvc -owide
NAME      STATUS VOLUME                                     CAPACITY ACCESS MODES STORAGECLASS
AGE   VOLUMEMODE
data-nginx-0 Bound pvc-04e25985-fc93-4254-92a1-1085ce19d31e 1Gi      RWO        csi-disk
64s  Filesystem
data-nginx-1 Bound pvc-0ae6336b-a2ea-4ddc-8f63-cfc5f9efe189 1Gi      RWO        csi-disk
47s  Filesystem
data-nginx-2 Bound pvc-aa46f452-cc5b-4dbd-825a-da68c858720d 1Gi      RWO        csi-disk
30s  Filesystem
data-nginx-3 Bound pvc-3d60e532-ff31-42df-9e78-015cacb18a0b 1Gi      RWO        csi-disk
14s  Filesystem

# kubectl get pod -owide
NAME     READY STATUS RESTARTS AGE      IP          NODE      NOMINATED NODE READINESS GATES
nginx-0  1/1   Running 0       2m25s  172.16.0.12  192.168.0.121 <none>      <none>
nginx-1  1/1   Running 0       2m8s   172.16.0.136 192.168.0.211 <none>      <none>
nginx-2  1/1   Running 0       111s   172.16.1.7   192.168.0.240 <none>      <none>
nginx-3  0/1   Pending 0       95s    <none>      <none>      <none>      <none>
```

The event information of the pod shows that the scheduling fails due to no available node. Two nodes (in AZ 1 and AZ 2) do not have sufficient CPUs, and the created EVS disk is not in the AZ where the third node (in AZ 3) is located. As a result, the pod cannot use the EVS disk.

```
# kubectl describe pod nginx-3
Name:           nginx-3
...
Events:
  Type  Reason     Age From          Message
  ----  -----     --  --            --
  Warning FailedScheduling 111s default-scheduler 0/3 nodes are available: 3 pod has unbound immediate PersistentVolumeClaims.
  Warning FailedScheduling 111s default-scheduler 0/3 nodes are available: 3 pod has unbound immediate PersistentVolumeClaims.
  Warning FailedScheduling 28s  default-scheduler 0/3 nodes are available: 1 node(s) had volume node affinity conflict, 2 Insufficient cpu.
```

Check the AZ where the EVS disk created from the PVC is located. It is found that data-nginx-3 is in AZ 1. In this case, the node in AZ 1 has no resources, and only the node in AZ 3 has CPU resources. As a result, the scheduling fails. Therefore, there should be a delay between creating the PVC and binding the PV.

## Storage Class for Delayed Binding

If you check the cluster storage class, you can see that the binding mode of csi-disk-topology is **WaitForFirstConsumer**, indicating that a PV is created and bound when a pod uses the PVC. That is, the PV and the underlying storage resources are created based on the pod information.

# kubectl get storageclass		PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	
NAME	ALLOWVOLUMEEXPANSION	AGE	Delete	Immediate	true
csi-disk	everest-csi-provisioner	156m	Delete	<b>WaitForFirstConsumer</b>	true
csi-disk-topology	everest-csi-provisioner	156m	Delete	Immediate	true
csi-nas	everest-csi-provisioner	156m	Delete	Immediate	true
csi-obs	everest-csi-provisioner	156m	Delete	Immediate	false
csi-sfsturbo	everest-csi-provisioner	156m	Delete	Immediate	true

**VOLUMEBINDINGMODE** is displayed if your cluster is v1.19. It is not displayed in clusters of v1.17 or v1.15.

You can also view the binding mode in the csi-disk-topology details.

```
# kubectl describe sc csi-disk-topology
Name:           csi-disk-topology
IsDefaultClass: No
Annotations:    <none>
Provisioner:   everest-csi-provisioner
Parameters:    csi.storage.k8s.io/csi-driver-name=disk.csi.everest.io,csi.storage.k8s.io/fstype=ext4,everest.io/disk-volume-type=SAS,everest.io/passthrough=true
AllowVolumeExpansion: True
MountOptions:   <none>
ReclaimPolicy: Delete
VolumeBindingMode: WaitForFirstConsumer
Events:        <none>
```

Create PVCs of the csi-disk and csi-disk-topology classes. Observe the differences between these two types of PVCs.

- **csi-disk**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: disk
  annotations:
    everest.io/disk-volume-type: SAS
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk      # StorageClass
```

- **csi-disk-topology**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: topology
  annotations:
    everest.io/disk-volume-type: SAS
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk-topology      # StorageClass
```

View the PVC details. As shown below, the csi-disk PVC is in Bound state and the csi-disk-topology PVC is in Pending state.

```
# kubectl create -f pvc1.yaml
persistentvolumeclaim/disk created
# kubectl create -f pvc2.yaml
persistentvolumeclaim/topology created
# kubectl get pvc
NAME      STATUS    VOLUME          CAPACITY   ACCESS MODES
STORAGECLASS   AGE
disk       Bound    pvc-88d96508-d246-422e-91f0-8caf414001fc  10Gi     RWO        csi-disk
18s
topology    Pending                           csi-disk-topology  2s
```

View details about the csi-disk-topology PVC. You can see that "waiting for first consumer to be created before binding" is displayed in the event, indicating that the PVC is bound after the consumer (pod) is created.

```
# kubectl describe pvc topology
Name: topology
Namespace: default
StorageClass: csi-disk-topology
Status: Pending
Volume:
Labels: <none>
Annotations: everest.io/disk-volume-type: SAS
Finalizers: [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode: Filesystem
Used By: <none>
Events:
  Type  Reason          Age           From           Message
  ----  ----          --            --           --
  Normal  WaitForFirstConsumer  5s (x3 over 30s)  persistentvolume-controller  waiting for first
  consumer to be created before binding
```

Create a workload that uses the PVC. Set the PVC name to **topology**.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx:alpine
          name: container-0
          volumeMounts:
            - mountPath: /tmp
              name: topology-example
            # Mount path
          restartPolicy: Always
        volumes:
          - name: topology-example
            persistentVolumeClaim:
              claimName: topology
            # PVC name
```

After the PVC is created, check the PVC details. You can see that the PVC is bound successfully.

```
# kubectl describe pvc topology
Name: topology
```

```

Namespace: default
StorageClass: csi-disk-topology
Status: Bound
...
Used By: nginx-deployment-fcd9fd98b-x6tbs
Events:
  Type  Reason          Age
  From                           Message
  ----  -----          ---
  Normal  WaitForFirstConsumer  84s (x26 over 7m34s)  persistentvolume-
controller                                waiting for first consumer to be created before
binding
  Normal  Provisioning        54s      everest-csi-provisioner_everest-csi-
controller-7965dc48c4-5k799_2a6b513e-f01f-4e77-af21-6d7f8d4dbc98  External provisioner is provisioning
volume for claim "default/topology"
  Normal  ProvisioningSucceeded 52s      everest-csi-provisioner_everest-csi-
controller-7965dc48c4-5k799_2a6b513e-f01f-4e77-af21-6d7f8d4dbc98  Successfully provisioned volume
pvc-9a89ea12-4708-4c71-8ec5-97981da032c9

```

## Using csi-disk-topology in Cross-AZ Node Deployment

The following uses csi-disk-topology to create a StatefulSet with the same configurations used in the preceding example.

```

volumeClaimTemplates:
  - metadata:
      name: data
      annotations:
        everest.io/disk-volume-type: SAS
    spec:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 1Gi
      storageClassName: csi-disk-topology

```

After the creation, check the PVC and pod status. As shown in the following output, the PVC and pod can be created successfully. The nginx-3 pod is created on the node in AZ 3.

```

# kubectl get pvc -owide
NAME      STATUS VOLUME                                     CAPACITY ACCESS MODES
STORAGECLASS AGE VOLUMEMODE
data-nginx-0 Bound pvc-43802cec-cf78-4876-bcca-e041618f2470 1Gi      RWO      csi-disk-
topology 55s Filesystem
data-nginx-1 Bound pvc-fc942a73-45d3-476b-95d4-1eb94bf19f1f 1Gi      RWO      csi-disk-
topology 39s Filesystem
data-nginx-2 Bound pvc-d219f4b7-e7cb-4832-a3ae-01ad689e364e 1Gi      RWO      csi-disk-
topology 22s Filesystem
data-nginx-3 Bound pvc-b54a61e1-1c0f-42b1-9951-410ebd326a4d 1Gi      RWO      csi-disk-
topology 9s Filesystem

# kubectl get pod -owide
NAME      READY STATUS RESTARTS AGE IP           NODE      NOMINATED NODE READINESS
GATES
nginx-0   1/1  Running 0     65s  172.16.1.8  192.168.0.240 <none>    <none>
nginx-1   1/1  Running 0     49s  172.16.0.13  192.168.0.121 <none>    <none>
nginx-2   1/1  Running 0     32s  172.16.0.137 192.168.0.211 <none>    <none>
nginx-3   1/1  Running 0     19s  172.16.1.9   192.168.0.240 <none>    <none>

```

# 11 Container

## 11.1 Properly Allocating Container Computing Resources

If a node has sufficient memory resources, a container on this node can use more memory resources than requested, but no more than limited. If the memory allocated to a container exceeds the upper limit, the container is stopped first. If the container continuously uses memory resources more than limited, the container is terminated. If a stopped container is allowed to be restarted, kubelet will restart it, but other types of run errors will occur.

### Scenario 1

The node's memory has reached the memory limit reserved for the node. As a result, OOM killer is triggered.

#### Solution

You can either scale up the node or migrate the pods on the node to other nodes.

### Scenario 2

The upper limit of resources configured for the pod is too small. When the actual usage exceeds the limit, OOM killer is triggered.

#### Solution

Set a higher upper limit for the workload.

### Example

A pod will be created and allocated memory that exceeds the limit. As shown in the following configuration file of the pod, the pod requests 50 MB memory and the memory limit is set to 100 MB.

Example YAML file (memory-request-limit-2.yaml):

```
apiVersion: v1
kind: Pod
```

```
metadata:  
  name: memory-demo-2  
spec:  
  containers:  
    - name: memory-demo-2-ctr  
      image: vish/stress  
      resources:  
        requests:  
          memory: 50Mi  
        limits:  
          memory: "100Mi"  
      args:  
        - -mem-total  
        - 250Mi  
        - -mem-alloc-size  
        - 10Mi  
        - -mem-alloc-sleep  
        - 1s
```

The **args** parameters indicate that the container attempts to request 250 MB memory, which exceeds the pod's upper limit (100 MB).

Creating a pod:

```
kubectl create -f https://k8s.io/docs/tasks/configure-pod-container/memory-request-limit-2.yaml --namespace=mem-example
```

Viewing the details about the pod:

```
kubectl get pod memory-demo-2 --namespace=mem-example
```

In this stage, the container may be running or be killed. If the container is not killed, repeat the previous command until the container is killed.

NAME	READY	STATUS	RESTARTS	AGE
memory-demo-2	0/1	OOMKilled	1	24s

Viewing detailed information about the container:

```
kubectl get pod memory-demo-2 --output=yaml --namespace=mem-example
```

This output indicates that the container is killed because the memory limit is exceeded.

```
lastState:  
  terminated:  
    containerID: docker://7aae52677a4542917c23b10fb56fc2434c2e8427bc956065183c1879cc0dbd2  
    exitCode: 137  
    finishedAt: 2020-02-20T17:35:12Z  
    reason: OOMKilled  
    startedAt: null
```

In this example, the container can be automatically restarted. Therefore, kubelet will start it again. You can run the following command several times to see how the container is killed and started:

```
kubectl get pod memory-demo-2 --namespace=mem-example
```

The preceding command output indicates how the container is killed and started back and forth:

```
$ kubectl get pod memory-demo-2 --namespace=mem-example  
NAME      READY   STATUS    RESTARTS   AGE  
memory-demo-2  0/1    OOMKilled  1         37s  
$ kubectl get pod memory-demo-2 --namespace=mem-example  
NAME      READY   STATUS    RESTARTS   AGE  
memory-demo-2  1/1    Running   2         40s
```

Viewing the historical information of the pod:

```
kubectl describe pod memory-demo-2 --namespace=mem-example
```

The following command output indicates that the pod is repeatedly killed and started.

```
... Normal Created Created container with id  
66a3a20aa7980e61be4922780bf9d24d1a1d8b7395c09861225b0eba1b1f8511  
... Warning BackOff Back-off restarting failed container
```

## 11.2 Upgrading Pods Without Interrupting Services

### Scenario

You can use rolling upgrade to upgrade pods without interrupting services.

In this mode, pods are upgraded one by one, not all at once.

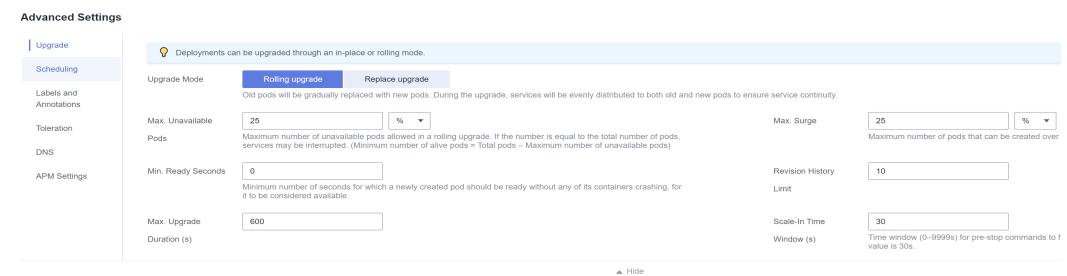
### Prerequisites

The workload to be upgraded has at least two pods. If there is only one pod, you are advised to perform the upgrade after manually scaling the workload into two pods.

### Procedure

- Step 1** Log in to the CCE console.
- Step 2** In the workload list, click the name of the workload to be upgraded. The workload details page is displayed.
- Step 3** In the upper right corner of the page, choose **More > Upgrade** to upgrade the workload in rolling upgrade mode.

**Figure 11-1** Rolling upgrade



After the configuration is complete, click **Upgrade Workload**.

On the **Pods** tab page, you can view that one pod is created and then the other is stopped. This ensures that there is always a pod running and the service is not interrupted during the upgrade.

----End

## 11.3 Modifying Kernel Parameters Using a Privileged Container

### Prerequisites

To access a Kubernetes cluster from a client, you can use the Kubernetes command line tool kubectl.

### Procedure

**Step 1** Create a DaemonSet in the background, select the Nginx image, enable the Privileged Container, configure the lifecycle, and add the **hostNetwork** field (value: **true**).

1. Create a DaemonSet file.

**vi daemonSet.yaml**

An example YAML file is provided as follows:

#### NOTICE

The **spec.spec.containers.lifecycle** field indicates the command that will be run after the container is started.

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: daemonset-test
  labels:
    name: daemonset-test
spec:
  selector:
    matchLabels:
      name: daemonset-test
  template:
    metadata:
      labels:
        name: daemonset-test
    spec:
      hostNetwork: true
      containers:
        - name: daemonset-test
          image: nginx:alpine-perl
          command:
            - "/bin/sh"
            args:
              - "-c"
              - "while ; do time=$(date);done"
          imagePullPolicy: IfNotPresent
          lifecycle:
            postStart:
              exec:
                command:
                  - sysctl
                  - "-w"
                  - net.ipv4.tcp_tw_reuse=1
          securityContext:
            privileged: true
```

```
imagePullSecrets:  
- name: default-secret
```

2. Create a DaemonSet.

```
kubectl create -f daemonSet.yaml
```

**Step 2** Check whether the DaemonSet is successfully created.

```
kubectl get daemonset DaemonSet name
```

In this example, run the following command:

```
kubectl get daemonset daemonset-test
```

Information similar to the following is displayed:

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
daemonset-test	2	2	2	2	2	<node>	2h

**Step 3** Query the container ID of DaemonSet on the node.

```
docker ps -a|grep DaemonSet name
```

In this example, run the following command:

```
docker ps -a|grep daemonset-test
```

Information similar to the following is displayed:

897b99faa9ce	3e094d5696c1	"/bin/sh -c while..."	31 minutes ago	Up 30 minutes	ault_fa7cc313-4ac1-11e9-a716-fa163e0aalba_0
--------------	--------------	-----------------------	----------------	---------------	---------------------------------------------

**Step 4** Access the container.

```
docker exec -it containerid /bin/sh
```

In this example, run the following command:

```
docker exec -it 897b99faa9ce /bin/sh
```

**Step 5** Check whether the configured command is executed after the container is started.

```
sysctl -a |grep net.ipv4.tcp_tw_reuse
```

If the following information is displayed, the system parameters are modified successfully:

```
net.ipv4.tcp_tw_reuse=1
```

----End

## 11.4 Initializing a Container

### Concepts

Before containers running 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.

## Scenario

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 exit, 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. Therefore, 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, 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:
    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
```

**Step 3** Query the created Docker container on the node where the workload is running.

```
docker ps -a|grep mysql
```

The init container will exit after it runs to completion. The query result **Exited (0)** shows the exit status of the init container.

```
9dc822969e3f      percona      "docker-entrypoint..."  34 seconds ago      Up 33 seconds
ql_mysql-76598b8c64-mmgw9  default_522566ea-bda5-11e9-a219-fa163e8b288b_0
a745881214e7      busybox      "sh -c 'sleep 20'"   About a minute ago   Exited (0) 50 seconds ago
resource_mysql-76598b8c64-mmgw9  default_522566ea-bda5-11e9-a219-fa163e8b288b_0
615db9e60a80      cfe-pause:11.23.1  "/pause"        About a minute ago   Up About a minute
mysql-76598b8c64-mmgw9  default_522566ea-bda5-11e9-a219-fa163e8b288b_0
```

----End

## 11.5 Setting Time Zone Synchronization

### Case Scenarios

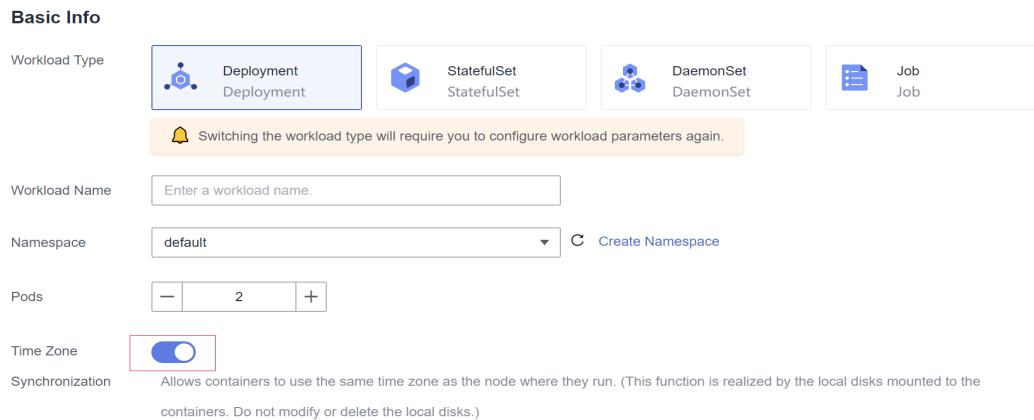
- **Scenario 1: Setting Time Zone Synchronization Between Containers and Nodes**
- **Scenario 2: Setting Time Zone Synchronization Among Containers, Container Logs, and Nodes**
- **Scenario 3: Setting Time Zone Synchronization Between Workloads and Nodes**

### Scenario 1: Setting Time Zone Synchronization Between Containers and Nodes

**Step 1** Log in to the CCE console.

**Step 2** In the **Basic Info** area of the **Create Workload** page, enable **Time Zone Synchronization** so that the same time zone will be used for both the container and the node.

Figure 11-2 Enabling the time zone synchronization



**Step 3** Log in to the node, go to the container, and check whether the time zone of the container is the same as that of the node.

**date -R**

Information similar to the following is displayed:

```
Tue, 04 Jun 2019 15:08:47 +0800
```

**docker ps -a|grep test**

Information similar to the following is displayed:

```
oedd74c66bdb      b2b9b536b744      "nginx -g 'daemon ...'"  6 hours ago      Up 6 hours
k8s_container-0_test-7d7d7f4965-xwqkx_default_abf6df2e-85f7-11e9-93df-fa163ee0f9
la_1
```

**docker exec -it oedd74c66bdb /bin/sh**

**date -R**

Information similar to the following is displayed:

```
Tue, 04 Jun 2019 15:09:20 +0800
```

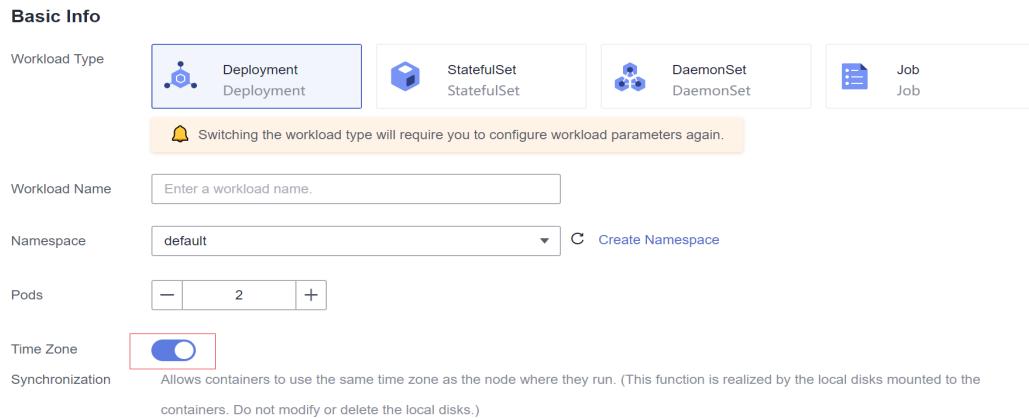
----End

## Scenario 2: Setting Time Zone Synchronization Among Containers, Container Logs, and Nodes

The difference between the time when the Java application prints logs and the container's standard time obtained in date -R mode is 8 hours.

**Step 1** Log in to the CCE console.

**Step 2** In the **Basic Info** area of the **Create Workload** page, enable **Time Zone Synchronization** so that the same time zone will be used for both the container and the node.

**Figure 11-3** Enabling the time zone synchronization

**Step 3** Log in to the node, go to the container, and modify the **catalina.sh** script.

```
cd /usr/local/tomcat/bin
```

```
vi catalina.sh
```

If you cannot run the **vi** command in the container, go to **Step 4** or run the **vi** command to add **-Duser.timezone=GMT+08** to the script, as shown in the following figure.

```
# Do this here so custom URL handles (specifically 'war:...') can be used in the security policy
JAVA_OPTS="$JAVA_OPTS -Djava.protocol.handler.pkgs=org.apache.catalina.webresources -Duser.timezone=GMT+08"
```

**Step 4** Copy the script from the container to the node, add **-Duser.timezone=GMT+08** to the script, and then copy the script from the node to the container.

Run the following commands to copy files in the container to the host machine:

```
docker cp mycontainer:/usr/local/tomcat/bin/catalina.sh /home/catalina.sh
```

Run the following commands to copy files from the host machine to the container:

```
docker cp /home/catalina.sh mycontainer:/usr/local/tomcat/bin/catalina.sh
```

**Step 5** Restart the container.

```
docker restart container_id
```

**Step 6** Check whether the time zone of the logs is the same as that of the node.

On the CCE console, click the workload name. On the workload details page displayed, click **Logs** in the upper right corner to view the log details. It takes about 5 minutes to load the logs.

----End

### Scenario 3: Setting Time Zone Synchronization Between Workloads and Nodes

- Method 1: Set the time zone to CST when creating a container image.
- Method 2: If you do not want to modify the container, when creating a workload on the CCE console, mount the **/etc/localtime** directory of the local host to the **/etc/localtime** directory of the container.

### Example:

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: test
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      volumes:
        - name: vol-162979628557461404
          hostPath:
            path: /etc/localtime
            type: ""
      containers:
        - name: container-0
          image: 'nginx:alpine'
          volumeMounts:
            - name: vol-162979628557461404
              readOnly: true
              mountPath: /etc/localtime
              imagePullPolicy: IfNotPresent
      imagePullSecrets:
        - name: default-secret
```

## 11.6 Setting the Container Network Bandwidth Limit

### Background

Containers on the same node share the host network bandwidth. Limiting the network bandwidth of containers can effectively prevent mutual interference between containers and improve container network stability.

### Scenario

This function is supported on all types of workloads.

### Constraints and Limitations

- This function is available only to clusters of v1.13.10 and later and whose network model is tunnel network.
- Clusters of v1.17 do not support this function.

### Procedure

**Step 1** Edit a YAML file for a workload.

**vi deployment.yaml**

Set the network bandwidth for the pod in `spec.template.metadata.annotations` to limit the network traffic of the container. For details about the network bandwidth limit fields, see [Table 11-1](#).

If the parameters are not specified, the network bandwidth is not limited by default.

The following is an example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
    annotations:
      # Ingress bandwidth
      kubernetes.io/ingress-bandwidth: 100M
      # Egress bandwidth
      kubernetes.io/egress-bandwidth: 1G
  spec:
    containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        imagePullSecrets:
          - name: default-secret
```

**Table 11-1** Fields for limiting the network bandwidth of pods

Field	Description	Mandatory
kubernetes.io/ingress-bandwidth	Ingress bandwidth for a pod. Value range: 1k-1P. If this field is set to a value greater than 32 Gbit/s, the actual ingress bandwidth that a pod can use is 32 Gbit/s.	No
kubernetes.io/egress-bandwidth	Egress bandwidth for a pod. Value range: 1k-1P. If this field is set to a value greater than 32 Gbit/s, the actual egress bandwidth that a pod can use is 32 Gbit/s.	No

**Step 2** Create a workload.

**kubectl create -f deployment.yaml**

Information similar to the following is displayed:

```
deployment.apps/nginx created
```

**----End**

## 11.7 Using hostAliases to Configure /etc/hosts in a Pod

### Scenario

If DNS or other related settings are inappropriate, you can use **hostAliases** to overwrite the resolution of the host name at the pod level when adding entries to the **/etc/hosts** file of the pod.

### Procedure

**Step 1** Use kubectl to connect to the cluster.

**Step 2** Create the **hostaliases-pod.yaml** file.

**vi hostaliases-pod.yaml**

The field in bold in the YAML file indicates the image name and tag. You can replace the example value as required.

```
apiVersion: v1
kind: Pod
metadata:
  name: hostaliases-pod
spec:
  hostAliases:
    - ip: 127.0.0.1
      hostnames:
        - foo.local
        - bar.local
    - ip: 10.1.2.3
      hostnames:
        - foo.remote
        - bar.remote
  containers:
    - name: cat-hosts
      image: tomcat:9-jre11-slim
      lifecycle:
        postStart:
          exec:
            command:
              - cat
              - /etc/hosts
  imagePullSecrets:
    - name: default-secret
```

**Table 11-2** pod field description

Parameter	Mandatory/ Optional	Description
apiVersion	Mandatory	API version number
kind	Mandatory	Type of the object to be created
metadata	Mandatory	Metadata definition of a resource object
name	Mandatory	Name of a pod

Parameter	Mandatory/ Optional	Description
spec	Mandatory	Detailed description of the pod. For details, see <a href="#">Table 11-3</a> .

**Table 11-3** spec field description

Parameter	Mandatory/ Optional	Description
hostAliases	Mandatory	Host alias
containers	Mandatory	For details, see <a href="#">Table 11-4</a> .

**Table 11-4** containers field description

Parameter	Mandatory/ Optional	Description
name	Mandatory	Container name
image	Mandatory	Container image name
lifecycle	Optional	Lifecycle

**Step 3** Create a pod.**kubectl create -f hostaliases-pod.yaml**

If information similar to the following is displayed, the pod is created.

pod/hostaliases-pod created

**Step 4** Query the pod status.**kubectl get pod hostaliases-pod**If the pod is in the **Running** state, the pod is successfully created.

NAME	READY	STATUS	RESTARTS	AGE
hostaliases-pod	1/1	Running	0	16m

**Step 5** Check whether the **hostAliases** functions properly.**docker ps |grep hostaliases-pod****docker exec -ti Container ID /bin/sh**

```
root@hostaliases-pod:/# cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1      localhost
::1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
10.0.0.25      hostaliases-pod

# Entries added by HostAliases.
127.0.0.1      foo.local      bar.local
10.1.2.3        foo.remote    bar.remote
```

----End

## 11.8 Configuring Domain Name Resolution for CCE Containers

This section describes how to configure domain name resolution for CCE containers.

### Service

- Create a Service before you create the appropriate back-end workload (Deployment or ReplicaSet) and any workloads that need to access it. When the Kubernetes starts a container, it provides environment variables that point to all the Services that are running when the container is started. For example, if a Service named foo exists, all containers will obtain the following variables when they are initialized.  
`FOO_SERVICE_HOST=<the host the Service is running on>`  
`FOO_SERVICE_PORT=<the port the Service is running on>`
- This requires that any Service that a pod wants to access must be created before the pod is created. Otherwise, environment variables will not take effect. This restriction does not apply to DNS.
- For a cluster, an optional add-on is a DNS server (mandatory for CCE clusters). The DNS server monitors the Kubernetes APIs for the new Services and creates a set of DNS records for each Service. If DNS is enabled throughout the cluster, all pods will be able to automatically resolve the names of Services.
- Do not specify a hostPort for a pod unless necessary. When a pod is bound to a hostPort, the number of locations to which the pod can be scheduled will be limited because each `<hostIP, hostPort, protocol>` must be unique. If you do not specify **hostIP** and **protocol**, Kubernetes uses 0.0.0.0 as the default host IP address and TCP as the default protocol.

If you only need to access the port for debugging, you can use apiserver proxies or kubectl port-forward.

If you want to open the pod port on the node, consider using the NodePort Service before using hostPort.

- Do not use hostNetwork. The reason is the same as that of using hostPort.
- When kube-proxy load balancing is not required, use headless Services (**ClusterIP** set to **None**) for service discovery.

## DNS

By default, CCE provides a DNS add-on Service named coredns to automatically assign DNS domain names for other Services. If it is running in the cluster, run the following command to check the status:

```
kubectl get services coredns --namespace=kube-system
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
kube-dns  ClusterIP  10.0.0.10  <none>        53/UDP,53/TCP  8m
```

If the pod is not running, you can run the **describe** command to check why the pod is not started. Assume that there is a Service that has a permanent IP address and a DNS server (coredns cluster add-on) that assigns domain name to the IP address. In this way, any pod in the cluster can communicate with the Service. You can run another application for testing. Enable a new pod, access the pod, and run the **curl** command to check whether the domain name of the Service can be correctly resolved. In some cases, the **curl** command cannot be executed due to the DNS search principles and configuration.

When a pod is created on the CCE console, not all dnsConfig configurations are opened and some default values of the pod domain name resolution parameters are used. You need to know well the default configurations. A typical case is **ndots**. If the number of dots is within the **ndots** threshold range, the domain name is considered as an internal domain name of the Kubernetes cluster and the **..svc.cluster.local** suffix is added to the domain name.

## DNS Search Principles and Rules

DNS configuration file: **/etc/resolv.conf**

```
nameserver 10.247.x.x
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:3
```

Parameter description:

- **nameserver**: domain name resolution server
- **search**: domain name suffix search rule. More search configurations indicate more matching times for domain name resolution. For example, if three suffixes are matched, at least six search operations are required because both IPv4 and IPv6 addresses need to be checked.
- **options**: domain name resolution option. Multiple KV values are available. A typical case is **ndots**. If the number of dots in the domain name to be accessed exceeds the value of **ndots**, the domain name is considered as a complete domain name and is directly parsed. If the number of dots is less than the value of **ndots**, the suffix **..svc.cluster.local** will be added.

## Parameters in Kubernetes dnsConfig

- **nameservers**: a list of IP addresses that will be used as DNS servers for the pod. A maximum of three IP addresses can be specified. If pod's **dnsPolicy** is set to **None**, the list must contain at least one IP address, otherwise this property is optional. The servers listed will be combined to the base nameservers generated from the specified DNS policy with duplicate addresses removed.
- **searches**: a list of DNS search domains for host name lookup in the pod. This property is optional. When specified, the provided list will be merged into the base search domain names generated from the chosen DNS policy. Duplicate domain names are removed. Kubernetes allows for at most 6 search domains.
- **options**: an optional list of objects where each object may have a **name** property (required) and a **value** property (optional). The contents in this property will be merged to the options generated from the specified DNS policy. Duplicate entries are removed.

For details, see [DNS for Services and Pods](#).

## Pod DNS Policies

DNS policies can be set on a per-pod basis. Currently, three types of DNS policies are supported: **Default**, **ClusterFirst**, and **None**.

- **Default**: The DNS configuration of the pod inherits from the host. That is, the DNS configuration of the pod is the same as that of the host and node.
- **ClusterFirst**: Unlike the **Default** policy, the **ClusterFirst** policy writes kube-dns (or CoreDNS) information to the DNS configuration of the pod in advance. **ClusterFirst** is the default pod policy. If **PodPolicy** is not specified for the pod, **dnsPolicy** is preset to **ClusterFirst**. However, **ClusterFirst** is mutually exclusive with **HostNetwork=true**. If **HostNetwork** is set to **true**, the **ClusterFirst** policy will be forcibly changed to the **Default** policy.
- **None**: This policy will clear the DNS configuration preset for the pod. If **dnsPolicy** is set to **None**, Kubernetes does not load any DNS configuration that is determined by its own logic in advance for the pod. Therefore, if you want to set **dnsPolicy** to **None**, you are advised to set **dnsConfig** to describe custom DNS parameters. This setting ensures that the pod has DNS configuration.

DNS configuration scenarios are provided as follows:

### Scenario 1: Using a custom DNS

The following example allows you to use a custom DNS to resolve the application domain name configuration in pods. After application migration, you do not need to modify the configuration.

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: dns-example
spec:
  containers:
    - name: test
      image: nginx
  dnsPolicy: "None"
```

```
dnsConfig:  
  nameservers:  
    - 1.2.3.4  
  searches:  
    - ns1.svc.cluster.local  
    - my.dns.search.suffix  
  options:  
    - name: ndots  
      value: "2"  
    - name: edns0
```

### Scenario 2: Using the Kubernetes DNS add-on CoreDNS

The DNS service of Kubernetes is preferentially used for domain name resolution. If the resolution fails, the DNS service of an external cascading system is used for domain name resolution.

```
apiVersion: v1  
kind: Pod  
metadata:  
  namespace: default  
  name: dns-example  
spec:  
  containers:  
    - name: test  
      image: nginx  
  dnsPolicy: ClusterFirst
```

### Scenario 3: Using the public network domain name resolution

This mode applies to the scenario where the domain names in pods are to be accessed from public networks. In this case, the applications in the pods resolve domain names from an external DNS.

```
apiVersion: v1  
kind: Pod  
metadata:  
  namespace: default  
  name: dns-example  
spec:  
  containers:  
    - name: test  
      image: nginx  
  dnsPolicy: Default
```

### Scenario 4: Using hostNetwork

If **hostNetwork: true** is used to configure the networking in the pod, the network ports of the host machine are exposed to the application running in the pod. All network ports on the LAN where the host machine is located can be used to access the application.

```
apiVersion: extensions/v1beta1  
kind: Deployment  
metadata:  
  name: nginx  
spec:  
  template:  
    metadata:  
      labels:  
        app: nginx  
  spec:  
    hostNetwork: true  
    dnsPolicy: ClusterFirstWithHostNet  
    containers:  
      - name: nginx  
        image: nginx:1.7.9
```

```
ports:  
- containerPort: 80
```

If **dnsPolicy: ClusterFirstWithHostNet** is not added, even if the pod uses the DNS of the host machine by default, other pods in the Kubernetes cluster cannot be accessed through the Service name in the container.

## CoreDNS Configuration

### 1. Configuring the CoreDNS ConfigMap

The default CoreDNS configuration file is as follows:

```
Corefile: |  
.53 {  
    errors  
    health  
    kubernetes cluster.local in-addr.arpa ip6.arpa {  
        pods insecure  
        upstream  
        fallthrough in-addr.arpa ip6.arpa  
    }  
    prometheus :9153  
    forward . /etc/resolv.conf  
    cache 30  
    loop  
    reload  
    loadbalance
```

Parameter description:

- **error**: Errors are recorded in stdout.
- **health**: The CoreDNS running status report can be obtained from <http://localhost:8080/health>.
- **kubernetes**: The CoreDNS returns a DNS query response based on the IP addresses of the Kubernetes Service and pod.
- **prometheus**: The measurement standard of CoreDNS can be found in the metrics in the format of <http://localhost:9153/Prometheus>. You can obtain monitoring data in Prometheus format from <http://localhost:9153/metrics>.
- **proxy and forward**: Any query that is not in the Kubernetes cluster domain is forwarded to the predefined resolver (/etc/resolv.conf). If the domain name cannot be resolved locally, query the upper-level address. By default, the **/etc/resolv.conf** configuration of the host machine is used.
- **cache**: The front-end cache is enabled.
- **loop**: Simple forwarding loops are detected. If a loop is detected, the CoreDNS process is stopped.
- **reload**: The changed Corefile can be automatically reloaded. After editing the ConfigMap, wait for two minutes for the modification to take effect.
- **loadbalance**: This is a round-robin DNS load balancer that randomizes the order of A, AAAA, and MX records in the answer.

### 2. Configuring an external DNS server

Some services are not in the Kubernetes environment and need to be accessed through the DNS. The suffix of the service name is **carey.com**.

```
carey.com:53 {  
    errors
```

```
    cache 30
    proxy . 10.150.0.1
}
```

Complete configuration file:

```
Corefile: |
.:53 {
  errors
  health
  kubernetes cluster.local in-addr.arpa ip6.arpa {
    pods insecure
    upstream
    fallthrough in-addr.arpa ip6.arpa
  }
  prometheus :9153
  forward . /etc/resolv.conf
  cache 30
  loop
  reload
  loadbalance
}
carey.com:53 {
  errors
  cache 30
  proxy . 10.150.0.1
}
```

Currently, CCE add-on management supports the configuration of stub-domains, which is more flexible and convenient than the direct editing of ConfigMaps. You do not need to configure the domain name resolution for pods.

## 11.9 How Do I Select a Container Runtime?

### containerd vs Docker

Container runtime, one of the most important components of Kubernetes, manages the lifecycle of images and containers. kubelet interacts with a container runtime through the Container Runtime Interface (CRI).

CCE supports containerd and Docker as your runtime. **containerd is recommended for its shorter traces, fewer components, and stability.**

Select Docker for the following scenarios:

- Docker-in-Docker needs to be used.
- Commands such as **docker build/push/save/load** need to be run on the CCE node.
- Docker APIs need to be called.
- Docker Compose or Docker Swarm needs to be used.

### Common Commands of containerd and Docker

containerd does not support Docker APIs and Docker CLI, but you can run crictl commands to implement similar functions.

**Table 11-5** Image-related commands

No.	Docker Command	containerd Command	Remarks
1	docker images [Option] [Image name[:Tag]]	crictl images [Option] [Image name[:Tag]]	List local images.
2	docker pull [Option] Image name[:Tag] @DIGEST]	crictl pull [Option] Image name[:Tag]@DIGEST]	Pull images.
3	docker push	None	Push an image.
4	docker rmi [Option] Image...	crictl rmi [Option] Image ID...	Delete a local image.
5	docker inspect Image ID	crictl inspect Image ID	Check a container.

**Table 11-6** Container-related commands

No.	Docker Command	containerd Command	Remarks
1	docker ps [Option]	crictl ps [Option]	List containers.
2	docker create [Option]	crictl create [Option]	Create a container.
3	docker start [Option] Container ID...	crictl start [Option] Container ID...	Start a container.
4	docker stop [Option] Container ID...	crictl stop [Option] Container ID...	Stop a container.
5	docker rm [Option] Container ID...	crictl rm [Option] Container ID...	Delete a container.
6	docker attach [Option] Container ID	crictl attach [Option] Container ID	Connect to a container.
7	docker exec [Option] Container ID Startup command [Parameter...]	crictl exec [Option] Container ID Startup command [Parameter...]	Access the container.
8	docker inspect [Option] Container name ID...	crictl inspect [Option] Container ID...	Query container details.
9	docker logs [Option] Container ID	crictl logs [Option] Container ID	View container logs.

No.	Docker Command	containerd Command	Remarks
10	docker stats [Option] <i>Container ID...</i>	crtctl stats [Option] <i>Container ID</i>	Check the resource usage of the container.
11	docker update [Option] <i>Container ID...</i>	crtctl update [Option] <i>Container ID...</i>	Update container resource limits.

**Table 11-7** Pod-related commands

No.	Docker Command	containerd Command	Remarks
1	None	crtctl pods [Option]	List pods.
2	None	crtctl inspectp [Option] <i>Pod ID...</i>	View pod details.
3	None	crtctl start [Option] <i>Pod ID...</i>	Start a pod.
4	None	crtctl runp [Option] <i>Pod ID...</i>	Run a pod.
5	None	crtctl stopp [Option] <i>Pod ID...</i>	Stop a pod.
6	None	crtctl rmp [Option] <i>Pod ID...</i>	Delete a pod.

 **NOTE**

Containers created and started by containerd are immediately deleted by kubelet. containerd does not support suspending, resuming, restarting, renaming, and waiting for containers, nor Docker image build, import, export, comparison, push, search, and labeling. containerd does not support file copy. You can log in to the image repository by modifying the configuration file of containerd.

## Differences in Tracing

- Docker:  
kubelet --> docker shim (in the kubelet process) --> dockerd --> containerd
- containerd:  
kubelet --> cri plugin (in the containerd process) --> containerd

Although Docker has added functions such as swarm cluster, docker build, and Docker APIs, it also introduces bugs. Compared with containerd, Docker has one more layer of calling.

## 11.10 Using Dual-Architecture Images (x86 and Arm) in CCE

### Challenges

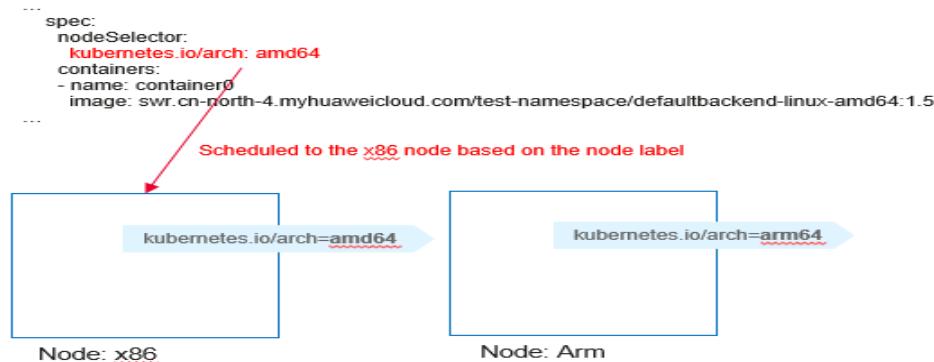
CCE provides CCE clusters and CCE Turbo clusters that support x86 nodes as well as Kunpeng clusters that support Arm nodes.

Due to different underlying architectures of Arm and x86, images (applications) based on the Arm architecture cannot run on x86 nodes, and vice versa. As a result, workloads may fail to be deployed in the clusters containing x86 and Arm nodes.

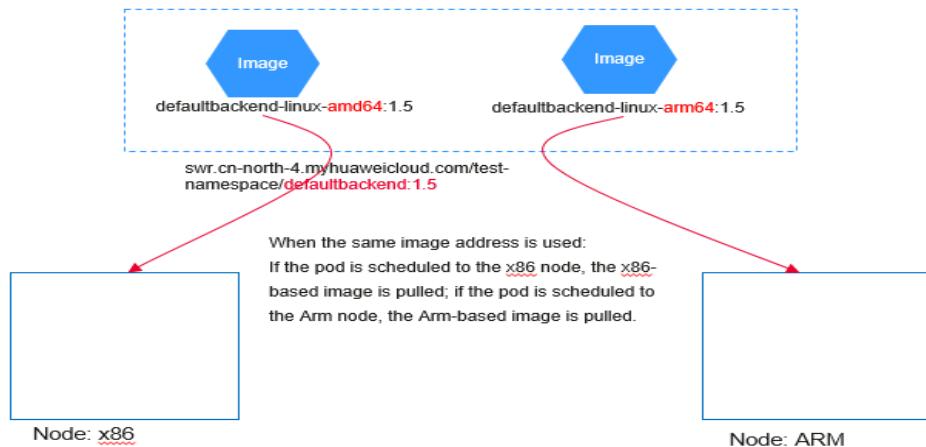
### Solution

To address this issue, use either of the following methods:

- Set the service affinity when you create a workload so that the pod can be scheduled to an Arm node when the Arm-based image is used or to an x86 node when the x86-based image is used.



- Build a dual-architecture image that supports both x86 and Arm architectures. When a pod is scheduled to an Arm node, the Arm variant in the image is pulled. When a pod is scheduled to an x86 node, the x86 variant in the image is pulled. A dual-architecture image has two variants but has one unified access path. When deploying a workload, you only need to specify one image path without configuring the service affinity. In this case, the workload description file is simpler and easier to maintain.



## Affinity Configuration Description

When creating a node, CCE automatically adds the **kubernetes.io/arch** label to the node to indicate the node architecture.

`kubernetes.io/arch=amd64`

The value **amd64** indicates the x86 architecture, and **arm64** indicates the Arm architecture.

When creating a workload, you can configure the node affinity to schedule pods to nodes using the corresponding architecture.

You can use **nodeSelector** in the YAML file to configure the architecture.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test
spec:
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      nodeSelector:
        kubernetes.io/arch: amd64
      containers:
        - name: container0
          image: swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-amd64:1.5
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
          imagePullSecrets:
            - name: default-secret
```

## Building a Dual-Architecture Image



To create a dual-architecture image, ensure that the Docker client version is later than 18.03.

The essence of building a dual-architecture image is to build images based on the x86 and Arm architectures separately and then build the dual-architecture image manifest.

For example, the **defaultbackend-linux-amd64:1.5** and **defaultbackend-linux-arm64:1.5** are images based on the x86 and Arm architectures, respectively.

Upload the two images to SWR. For details about how to upload an image, see [Uploading an Image Through a Container Engine Client](#).

```
# Add a tag to the original amd64 image defaultbackend-linux-amd64:1.5.  
docker tag defaultbackend-linux-amd64:1.5 swr.ap-southeast-1.myhuaweicloud.com/test-namespace/  
defaultbackend-linux-amd64:1.5  
# Add a tag to the original arm64 image defaultbackend-linux-arm64:1.5.  
docker tag defaultbackend-linux-arm64:1.5 swr.ap-southeast-1.myhuaweicloud.com/test-namespace/  
defaultbackend-linux-arm64:1.5  
# Push the amd64 image to the image repository.  
docker push swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-amd64:1.5  
# Push the arm64 image to the image repository.  
docker push swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-arm64:1.5
```

Create a dual-architecture **manifest** file and upload it.

```
# Enable DOCKER_CLI_EXPERIMENTAL.  
export DOCKER_CLI_EXPERIMENTAL=enabled  
# Create the manifest image file.  
docker manifest create --amend --insecure swr.ap-southeast-1.myhuaweicloud.com/test-namespace/  
defaultbackend:1.5 swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-  
arm64:1.5 swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-amd64:1.5  
# Add arch information to the manifest image file.  
docker manifest annotate swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend:1.5  
swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-amd64:1.5 --arch amd64  
docker manifest annotate swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend:1.5  
swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-arm64:1.5 --arch arm64  
# Push the manifest image file to the image repository.  
docker manifest push -p --insecure swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend:  
1.5
```

In this way, you only need to use the image path **swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend:1.5** when creating a workload.

- When a pod is scheduled to an x86 node, the **swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-amd64:1.5** image is pulled.
- When a pod is scheduled to an Arm node, the **swr.ap-southeast-1.myhuaweicloud.com/test-namespace/defaultbackend-linux-arm64:1.5** image is pulled.

## 11.11 Configuring Core Dumps

### Challenges

Linux allows you to create a core dump file if an application crashes, which contains the data the application had in memory at the time of the crash. You can analyze the file to locate the fault.

Generally, when a service application crashes, its container exits and is reclaimed and destroyed. Therefore, container core files need to be permanently stored on the host or cloud storage. This topic describes how to configure container core dumps.

### Enabling Core Dump on a Node

Log in to the node, run the following command to enable core dump, and set the path and format for storing core files:

```
echo "/tmp/cores/core.%h.%e.%p.%t" > /proc/sys/kernel/core_pattern
```

Parameters:

- **%h**: host name (or pod name). You are advised to configure this parameter.
- **%e**: program file name. You are advised to configure this parameter.
- **%p**: (optional) process ID.
- **%t**: (optional) time of the core dump.

You can also configure a pre-installation or post-installation script to automatically run this command when creating a node.

### Workaround for EulerOS 2.3

EulerOS2.3 Systemd has a **bug** that affects container core dump. To use core dump, perform the following operations:

**Step 1** In the `/usr/lib/systemd/system/docker.service` file on the node, change the value of `LimitCORE` to `infinity`.

**Step 2** Restart Docker.

**Step 3** Redeploy service containers.

----End

### Permanently Storing Core Dumps

A core file can be stored in your host (using a hostPath volume) or cloud storage (using a PVC). The following is an example YAML file for using a hostPath volume.

```
apiVersion: v1
kind: Pod
metadata:
  name: coredump
spec:
  volumes:
```

```
- name: coredump-path
  hostPath:
    path: /home/coredump
  containers:
    - name: ubuntu
      image: ubuntu:12.04
      command: ["/bin/sleep","3600"]
      volumeMounts:
        - mountPath: /tmp/cores
          name: coredump-path
```

Create a pod using kubectl.

```
kubectl create -f pod.yaml
```

## Verification

After the pod is created, access the container and trigger a segmentation fault of the current shell terminal.

```
$ kubectl get pod
NAME           READY   STATUS    RESTARTS   AGE
coredump       1/1     Running   0          56s
$ kubectl exec -it coredump -- /bin/bash
root@coredump:/# kill -s SIGSEGV $$
command terminated with exit code 139
```

Log in to the node and check whether a core file is generated in the **/home/coredump** directory. The following example indicates that a core file is generated.

```
# ls /home/coredump
core.coredump.bash.18.1650438992
```

# 12 Permission

## 12.1 Configuring kubeconfig for Fine-Grained Management on Cluster Resources

### Scenario

By default, the kubeconfig file provided by CCE for users has permissions bound to the **cluster-admin** role, which are equivalent to the permissions of user **root**. It is difficult to implement refined management on users with such permissions.

### Purpose

Cluster resources are managed in a refined manner so that specific users have only certain permissions (such as adding, querying, and modifying resources).

### Note

Ensure that kubectl is available on your host. If not, download it from [here](#) (corresponding to the cluster version or the latest version).

### Configuration Method



In the following example, only pods and Deployments in the **test** space can be viewed and added, and they cannot be deleted.

**Step 1** Set the service account name to **my-sa** and namespace to **test**.

```
kubectl create sa my-sa -n test
```

```
[root@test-arm-54016 ~]# kubectl create sa my-sa -n test
serviceaccount/my-sa created
[root@test-arm-54016 ~]#
```

**Step 2** Configure the role table and assign operation permissions to different resources.

```
vi role-test.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: myrole
  namespace: test
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - apps
  resources:
  - pods
  - deployments
  verbs:
  - get
  - list
  - watch
  - create
```

```
kubectl create -f role-test.yaml
```

```
[root@test-arm-54016 ~]# kubectl create -f role-test.yaml
role.rbac.authorization.k8s.io/myrole created
[root@test-arm-54016 ~]#
```

**Step 3** Create a RoleBinding and bind the service account to the role so that the user can obtain the corresponding permissions.

```
vi myrolebinding.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: myrolebinding
  namespace: test
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: myrole
subjects:
- kind: ServiceAccount
  name: my-sa
  namespace: test
```

```
kubectl create -f myrolebinding.yaml
```

```
[root@test-arm-54016 ~]# kubectl create -f myrolebinding.yaml
rolebinding.rbac.authorization.k8s.io/myrolebinding created
[root@test-arm-54016 ~]#
```

The user information is configured. Now perform **Step 4** to **Step 6** to write the user information to the configuration file.

**Step 4** Configure the cluster information.

1. Use the sa name **my-sa** to obtain the secret corresponding to the sa. In the following example, **my-sa-token-z4967** in the first column is the secret name.

```
kubectl get secret -n test |grep my-sa
```

```
[root@test-arm-54016 ~]# kubectl get secret -n test |grep my-sa
my-sa-token-5gp14      kubernetes.io/service-account-token  3        21m
[root@test-arm-54016 ~]#
```

2. Decrypt the **ca.crt** file in the secret and export it.

```
kubectl get secret my-sa-token-5gp14 -n test -oyaml |grep ca.crt:awk '{print $2}' | base64 -d > /home/ca.crt
```

3. Set the cluster access mode. **test-arm** indicates the cluster to be accessed, **10.0.1.100** indicates the IP address of the API server in the cluster (for details about how to obtain the IP address, see [Figure 12-1](#)), and **/home/test.config** indicates the path for storing the configuration file.

- If the internal API server address is used, run the following command:

```
kubectl config set-cluster test-arm --server=https://10.0.1.100:5443 --certificate-authority=/home/ca.crt --embed-certs=true --kubeconfig=/home/test.config
```

- If the public API server address is used, run the following command:

```
kubectl config set-cluster test-arm --server=https://10.0.1.100:5443 --kubeconfig=/home/test.config --insecure-skip-tls-verify=true
```

```
[root@test-arm-54016 home]# kubectl config set-cluster test-arm --server=https://10.0.1.100:5443 --certificate-authority=/home/ca.crt --embed-certs=true --kubeconfig=/home/test.config
Cluster "test-arm" set.
[root@test-arm-54016 home]# _
```

#### NOTE

If you **perform operations on a node in the cluster or the node that uses the configuration is a cluster node**, do not set the path of kubeconfig to **/root/.kube/config**.

The cluster API server address is an intranet API server address. After an EIP is bound to the cluster, the cluster API server address can also be a public API server address. You can obtain the API server address on the cluster details page on the CCE console.

**Figure 12-1** Obtaining the internal or public API server address

Basic Information	
Cluster Name	[REDACTED]
Cluster ID	41ec17f5-a65f-11ea-9e8b-0255ac101517
Cluster Type	Hybrid Cluster
Status	Available
Version	v1.13.10-r1
Cluster Spec	cce.s2.small   50 nodes
Docker Version	18.09.0.15
Enterprise project	cce-view
Created	Jun 04, 2020 20:31:04 GMT+08:00

Network	
Network Model	Tunnel network
VPC	vpc-b757
Subnet	wsdtest
Service Forwarding Mode	iptables
Service Network Segment	[REDACTED]
Container Network Segment	[REDACTED]
Internal API Server Address	5443
Public API Server Address	Bind EIP

### Step 5 Configure the cluster authentication information.

1. Obtain the cluster token. (If the token is obtained in GET mode, you need to run **base64 -d** to decode the token.)

```
token=$(kubectl describe secret my-sa-token-5gpl4 -n test | awk '/token:/ {print $2}')
```

2. Set the cluster user **ui-admin**.

```
kubectl config set-credentials ui-admin --token=$token --kubeconfig=/home/test.config
```

```
[root@test-arm-54016 home]# kubectl config set-credentials ui-admin --token=$token --kubeconfig=/home/test.config
User "ui-admin" set.
[root@test-arm-54016 home]#
```

### Step 6 Configure the context information for cluster authentication. **ui-admin@test** is the context name.

```
kubectl config set-context ui-admin@test --cluster=test-arm --user=ui-admin --kubeconfig=/home/test.config
```

```
[root@test-arm-54016 home]# kubectl config set-context ui-admin@test --cluster=test-arm --user=ui-admin --kubeconfig=/home/test.config
Context "ui-admin@test" created.
[root@test-arm-54016 home]#
```

### Step 7 Set the context. For details about how to use the context, see [Permissions Verification](#).

```
kubectl config use-context ui-admin@test --kubeconfig=/home/test.config
```

```
[paas@test-arm-54016 home]$ kubectl config use-context ui-admin@test --kubeconfig=/home/test.config
Switched to context "ui-admin@test".
```

#### NOTE

If you want to assign other users the above permissions to perform operations on the cluster, provide the generated configuration file **/home/test.config** to the user after performing step **Step 6**. The user must ensure that the host can access the API server address of the cluster. When performing step **Step 7** on the host and using kubectl, the user must set the kubeconfig parameter to the path of the configuration file.

----End

## Permissions Verification

1. Pods in the **test** namespace cannot access pods in other namespaces.

```
kubectl get pod -n test --kubeconfig=/home/test.config
```

```
[paas@test-arm-54016 home]$ kubectl get pod -n test --kubeconfig=/home/test.config
NAME          READY   STATUS    RESTARTS   AGE
test-pod-56cfcbf45b-12q92  0/1     CrashLoopBackOff  27          91m
[paas@test-arm-54016 home]$ kubectl get pod --kubeconfig=/home/test.config
Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:test:my-sa" cannot list resource "pods" in API group "" in the namespace "default"
[paas@test-arm-54016 home]$
```

2. Pods in the **test** namespace cannot be deleted.

```
[paas@test-arm-54016 home]$ kubectl delete pod -n test test-pod-56cfcbf45b-12q92 --kubeconfig=/home/test.config
Error from server (Forbidden): pods "test-pod-56cfcbf45b-12q92" is forbidden: User "system:serviceaccount:test:my-sa" cannot delete resource "pods" in API group "" in the namespace "test"
[paas@test-arm-54016 home]$
```

## Further Readings

For more information about users and identity authentication in Kubernetes, see [Authenticating](#).

## 12.2 Performing Cluster Namespace RBAC

### Painpoint

CCE permissions are classified into cluster permissions and namespace permissions. Namespace permissions are based on Kubernetes RBAC and can be used to grant permissions on resources in clusters and namespaces.

Currently, the CCE console provides four types of namespace-level ClusterRole permissions by default: cluster-admin, admin, edit, and view. However, these permissions apply to resources in the namespace regardless of resource types (pods, Deployments, and Services).

### Solution

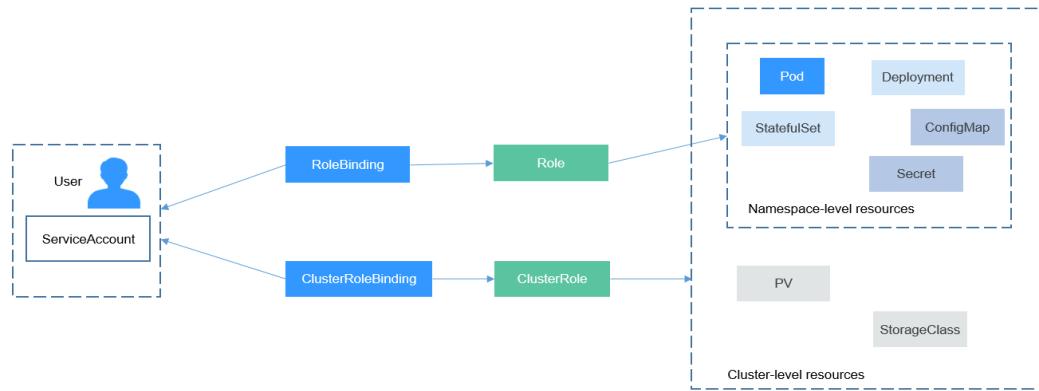
Kubernetes RBAC enables you to easily grant permissions on namespace resources.

- Role: defines a set of rules for accessing Kubernetes resources in a namespace.
- RoleBinding: defines the relationship between users and roles.
- ClusterRole: defines a set of rules for accessing Kubernetes resources in a cluster (including all namespaces).

- ClusterRoleBinding: defines the relationship between users and cluster roles.

Role and ClusterRole specify actions that can be performed on specific resources. RoleBinding and ClusterRoleBinding bind roles to specific users, user groups, or ServiceAccounts. See the following figure.

**Figure 12-2** Role binding



The user in the preceding figure can be an IAM user or user group in CCE. You can efficiently control permissions on namespace resources through RoleBindings.

The section describes how to use Kubernetes RBAC to grant user **user-example** with the permission for viewing pods. (This is the only permission the user has.)

## Prerequisites

RBAC is supported only on clusters of v1.11.7-r2 or later.

## Creating an IAM User and User Group

Log in to IAM and create an IAM user named **user-example** and a user group named **cce-role-group**. For details about how to create an IAM user and user group, see [Creating IAM Users](#) and [Creating User Groups](#).

Name	cce-role-group	Group ID	0c96fad22880f32a3f84c009862af6f7				
Description	--	Created	Jun 05, 2021 16:20:56 GMT+08:00				
<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <span>Permissions</span>                      Manage permissions for the user group.                 </div> <div style="flex: 1;"> <span>Users</span>                      Manage users in the user group.                 </div> </div>							
<div style="display: flex; justify-content: space-around;"> <span>Add</span> <span>Remove</span> </div>							
<table border="1"> <tr> <td><input type="checkbox"/> Username</td> <td>Description</td> </tr> <tr> <td><input type="checkbox"/> user-example</td> <td>--</td> </tr> </table>				<input type="checkbox"/> Username	Description	<input type="checkbox"/> user-example	--
<input type="checkbox"/> Username	Description						
<input type="checkbox"/> user-example	--						

Grant the **CCE FullAccess** permission to the **cce-role-group** user group. For details about how to grant permissions to a user group, see [Assigning Permissions to User Groups](#).

The screenshot shows the CCE console's User Groups interface. A user group named 'cce-role-group' is selected. The 'Permissions' tab is active, showing a table with columns for Policy/Role Name, Type, and Description. One row is highlighted, corresponding to the 'CCE FullAccess' policy.

Policy/Role Name	Type	Description
CCE FullAccess	System-defined policy	Common operation permissions on CCE cluster resources, excludin...

**CCE FullAccess** has the permissions for cluster operations (such as cluster creation), but does not have the permissions to operate Kubernetes resources (such as viewing pods).

## Creating a Cluster

Log in to CCE and create a cluster.

### NOTICE

Do not use IAM user **user-example** to create a cluster because CCE automatically assigns the cluster-admin permissions of all namespaces in the cluster to the user who creates the cluster. That is, the user can fully control the resources in the cluster and all its namespaces.

Log in to the CCE console as IAM user **user-example**, [download the kubectl configuration file in the cluster and connect to the cluster](#). Run the following command to obtain the pod information. The output shows that **user-example** does not have the permission to view the pods or other resources. This indicates that **user-example** does not have the permissions to operate Kubernetes resources.

```
# kubectl get pod
Error from server (Forbidden): pods is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list
resource "pods" in API group "" in the namespace "default"
# kubectl get deploy
Error from server (Forbidden): deployments.apps is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8"
cannot list resource "deployments" in API group "apps" in the namespace "default"
```

## Creating a Role and RoleBinding

Log in to the CCE console, [download the kubectl configuration file in the cluster and connect to the cluster](#). Create a Role and RoleBinding.

### NOTE

Log in as the account used to create the cluster because CCE automatically assigns the cluster-admin permissions to the account. That is, the account has the permissions to create Roles and RoleBindings. Alternatively, you can use IAM users who have the permissions to create Roles and RoleBindings.

The procedure for creating a Role is very simple. To be specific, specify a namespace and then define rules. The rules in the following example are to allow GET and LIST operations on pods in the default namespace.

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default          # Namespace
  name: role-example
rules:
- apiGroups: [""]
  resources: ["pods"]           # The pod can be accessed.
  verbs: ["get", "list"]         # The GET and LIST operations can be performed.
```

- **apiGroups** indicates the API group to which the resource belongs.
- **resources** indicates the resources that can be operated. Pods, Deployments, ConfigMaps, and other Kubernetes resources are supported.
- **verbs** indicates the operations that can be performed. **get** indicates querying a specific object, and **list** indicates listing all objects of a certain type. Other value options include **create**, **update**, and **delete**.

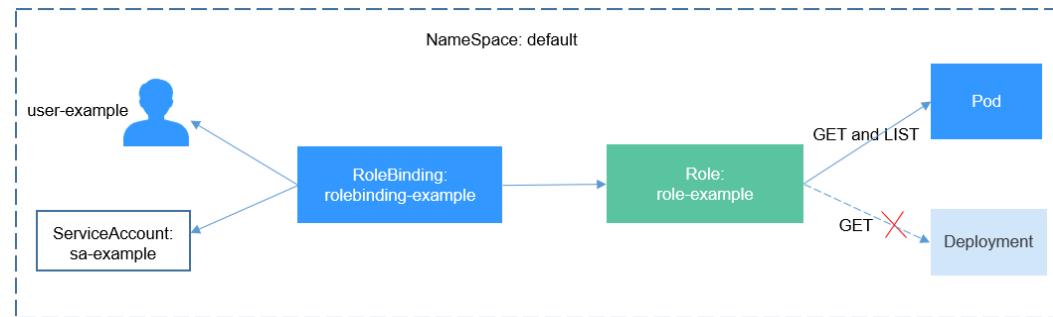
For details, see [Using RBAC Authorization](#).

After creating a Role, you can bind the Role to a specific user, which is called RoleBinding. The following is an example.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: RoleBinding-example
  namespace: default
  annotations:
    CCE.com/IAM: 'true'
roleRef:
  kind: Role
  name: role-example
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: User
  name: 0c97ac3cb280f4d91fa7c0096739e1f8  # IAM user ID
  apiGroup: rbac.authorization.k8s.io
```

The **subjects** section binds a Role with an IAM user so that the IAM user can obtain the permissions defined in the Role, as shown in the following figure.

**Figure 12-3** A RoleBinding binds the Role to the user.



You can also specify a user group in the **subjects** section. In this case, all users in the user group obtain the permissions defined in the Role.

```
subjects:
- kind: Group
```

```
name: 0c96fad22880f32a3f84c009862af6f7 # User group ID
apiGroup: rbac.authorization.k8s.io
```

## Verification

Use IAM user **user-example** to connect to the cluster and view the pods. The pods can be viewed.

```
# kubectl get pod
NAME           READY   STATUS    RESTARTS   AGE
nginx-658dff48ff-7rkph   1/1     Running   0          4d9h
nginx-658dff48ff-njdhj   1/1     Running   0          4d9h
# kubectl get pod nginx-658dff48ff-7rkph
NAME           READY   STATUS    RESTARTS   AGE
nginx-658dff48ff-7rkph 1/1     Running   0          4d9h
```

Try querying Deployments and Services in the namespace. The output shows **user-example** does not have the corresponding permissions. Try querying the pods in namespace kube-system. The output shows **user-example** does not have the corresponding permission, neither. This indicates that the IAM user **user-example** has only the GET and LIST Pod permissions in the default namespace, which is the same as expected.

```
# kubectl get deploy
Error from server (Forbidden): deployments.apps is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "deployments" in API group "apps" in the namespace "default"
# kubectl get svc
Error from server (Forbidden): services is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "services" in API group "" in the namespace "default"
# kubectl get pod --namespace=kube-system
Error from server (Forbidden): pods is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "pods" in API group "" in the namespace "kube-system"
```

# 13 Release

## 13.1 Overview

### Challenges

When switching between old and new services, you may be challenged in ensuring the system service continuity. If a new service version is directly released to all users at a time, it can be risky because once an online accident or bug occurs, the impact on users is great. It could take a long time to fix the issue. Sometimes, the version has to be rolled back, which severely affects user experience.

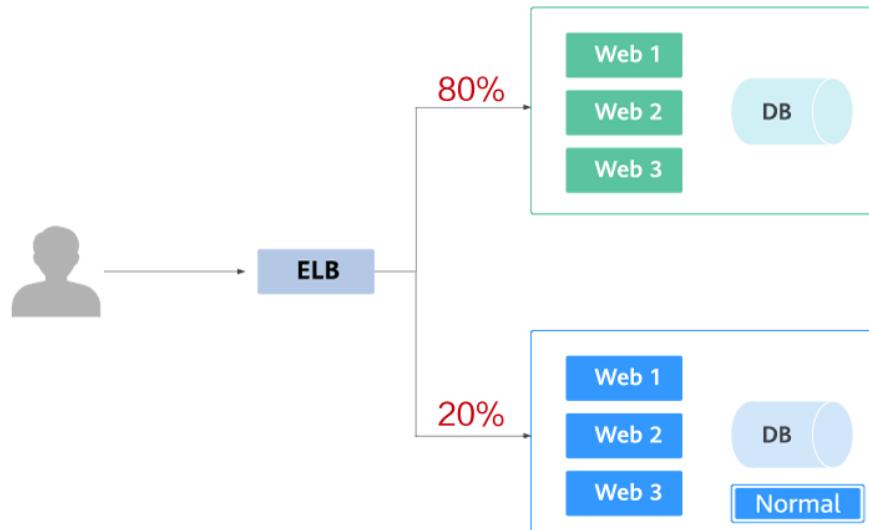
### Solutions

Several release policies are developed for service upgrade: grayscale release, blue-green deployment, A/B testing, rolling upgrade, and batch suspension of release. Traffic loss or service unavailability caused by releases can be avoided as much as possible.

This document describes the principles and practices of grayscale release and blue-green deployment.

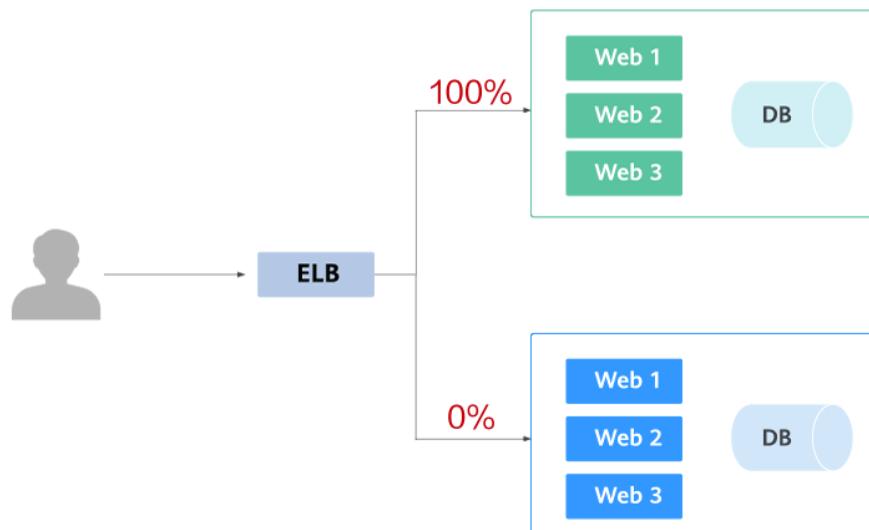
- Grayscale release, also called canary release, is a smooth iteration mode for version upgrade. During the upgrade, some users use the new version, while other users continue to use the old version. After the new version is stable and ready, it gradually takes over all the live traffic. In this way, service risks brought by the release of the new version can be minimized, the impact of faults can be reduced, and quick rollback is supported.

The following figure shows the general process of grayscale release. First, divide 20% of all service traffic to the new version. If the service version runs normally, gradually increase the traffic proportion and continue to test the performance of the new version. If the new version is stable, switch all traffic to it and bring the old version offline.



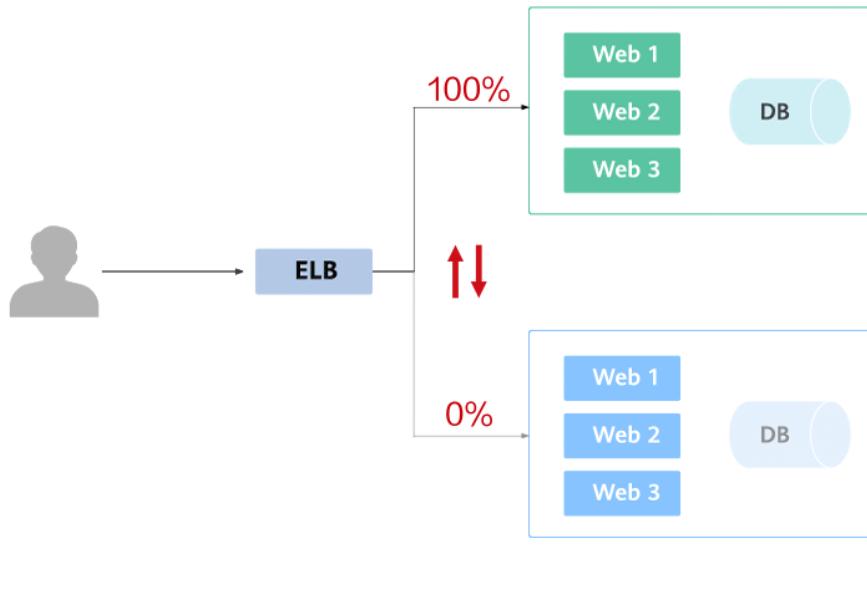
!

If an exception occurs in the new version when 20% of the traffic goes to the new version, you can quickly switch back to the old version.



!

- Blue-green deployment provides a zero-downtime, predictable manner for releasing applications to reduce service interruption during the release. A new version is deployed while the old version is retained. The two versions are online at the same time. The new and old versions work in hot backup mode. The route weight is switched (0 or 100) to enable different versions to go online or offline. If a problem occurs, the version can be quickly rolled back.



## Realizing Grayscale Release or Blue-Green Deployment

Kubernetes-native features can be used to implement simple grayscale release or blue-green deployment. For example, you can change the value of the label that determines the service version in the selector of a Service to change the pod backing the Service. In this way, the service can be directly switched from one version to another. If you have complex grayscale release or blue-green deployment requirements, you can deploy service meshes and open-source tools, such as Nginx Ingress and Traefik, to the cluster. You can obtain detailed instructions in the following sections:

- [Using Services to Implement Simple Grayscale Release and Blue-Green Deployment](#)
- [Using Nginx Ingress to Implement Grayscale Release and Blue-Green Deployment](#)

**Table 13-1** Comparison between the implementation modes

Implementation	Scenario	Feature	Disadvantage
Service	Simple release requirements and small-scale test scenarios	No need to introduce too many plug-ins or complex configurations	Manual operations and poor automation

Implementation	Scenario	Feature	Disadvantage
Nginx Ingress	No special requirements.	<ul style="list-style-type: none"><li>Only the annotation supported by Nginx Ingress needs to be configured.</li><li>Header-based, cookie-based, and service weight-based traffic division policies are supported.</li></ul>	The nginx-ingress add-on needs to be installed in the cluster, which consumes resources.

Both Services and Nginx Ingresses use open source Kubernetes capabilities to implement grayscale release and blue-green deployment. In this process, CCE allows you to easily perform the following operations:

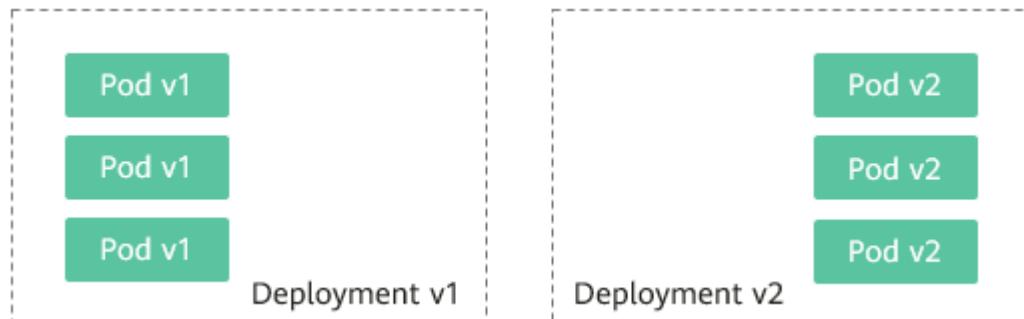
- All resources can be created, viewed, and modified on the console, which is more intuitive than the kubectl command line tool.
- LoadBalancer Services are supported by the ELB service. When creating a Service, you can use an existing ELB instance or create a new one.
- The nginx-ingress add-on can be installed in just a few clicks, and ELB load balancers can be created and interconnected during the installation.

## 13.2 Using Services to Implement Simple Grayscale Release and Blue-Green Deployment

To implement grayscale release for a CCE cluster, you need to deploy other open-source tools, such as Nginx Ingress, to the cluster or deploy services to a service mesh. These solutions are difficult to implement. If your grayscale release requirements are simple and you do not want to introduce too many plug-ins or complex configurations, you can refer to this section to implement simple grayscale release and blue-green deployment based on native Kubernetes features.

### Principles

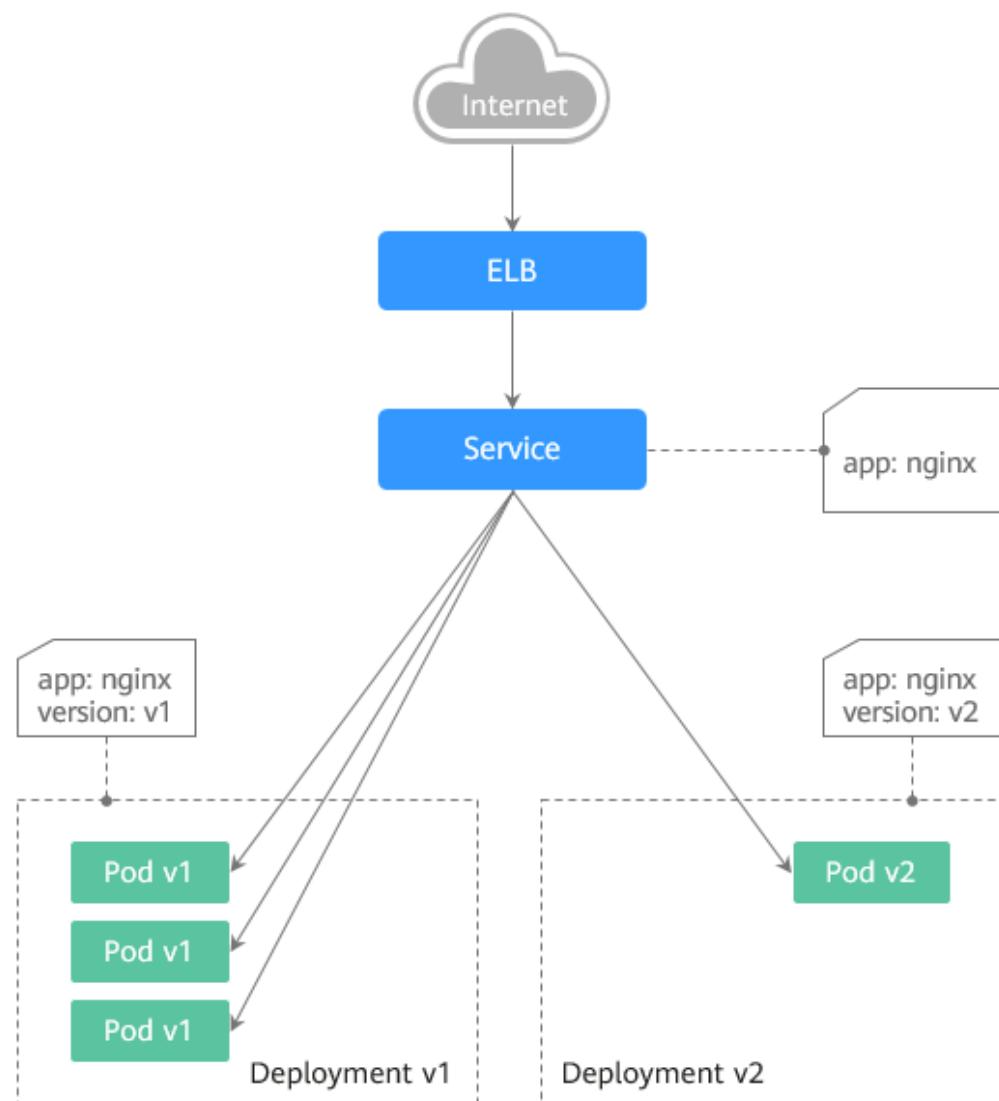
Users usually use Kubernetes objects such as Deployments and StatefulSets to deploy services. Each workload manages a group of pods. The following figure uses Deployment as an example.



Generally, a Service is created for each workload. The Service uses the selector to match the backend pod. Other Services or objects outside the cluster can access the pods backing the Service. If a pod needs to be exposed, set the Service type to LoadBalancer. The ELB load balancer functions as the traffic entrance.

- **Grayscale release principles**

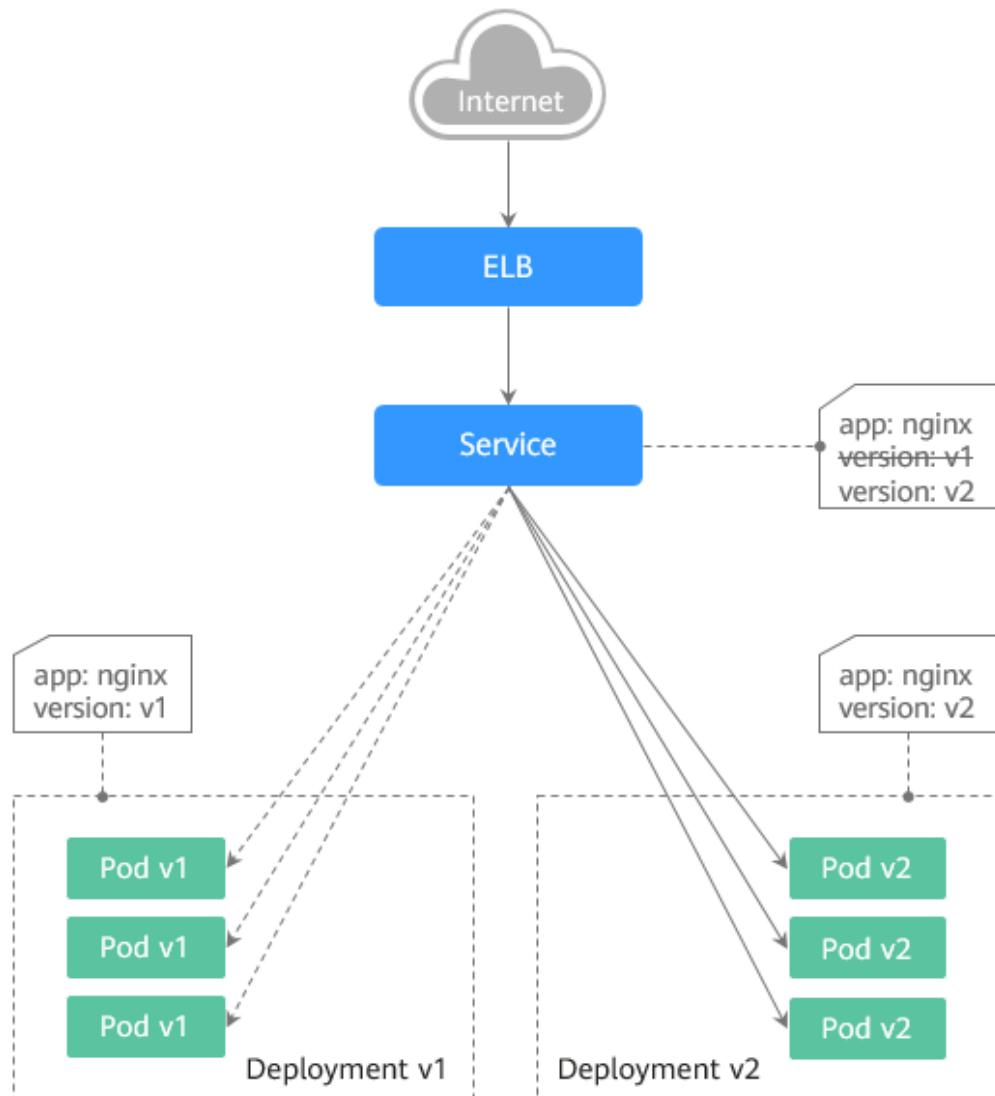
Take a Deployment as an example. A Service, in most cases, will be created for each Deployment. However, Kubernetes does not require that Services and Deployments correspond to each other. A Service uses a selector to match backend pods. If pods of different Deployments are selected by the same selector, a Service corresponds to multiple versions of Deployments. You can adjust the number of replicas of Deployments of different versions to adjust the weights of services of different versions to achieve grayscale release. The following figure shows the process:



- **Blue-green deployment principles**

Take a Deployment as an example. Two Deployments of different versions have been deployed in the cluster, and their pods are labeled with the same key but different values to distinguish versions. A Service uses the selector to select the pod of a Deployment of a version. In this case, you can change the

value of the label that determines the version in the Service selector to change the pod backing the Service. In this way, you can directly switch the service traffic from one version to another. The following figure shows the process:



## Prerequisites

The Nginx image has been uploaded to SWR. The Nginx images have two versions: v1 and v2. The welcome pages are **Nginx-v1** and **Nginx-v2**.

## Resource Creation

You can use YAML to deploy Deployments and Services in either of the following ways:

- On the **Create Deployment** page, click **Create YAML** on the right and edit the YAML file in the window.
- Save the sample YAML file in this section as a file and use `kubectl` to specify the YAML file. For example, run the `kubectl create -f xxx.yaml` command.

## Step 1: Deploy Services of Two Versions

Two versions of Nginx services are deployed in the cluster to provide external access through ELB.

- Step 1** Create a Deployment of the first version. The following uses nginx-v1 as an example. Example YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-v1
spec:
  replicas: 2          # Number of replicas of the Deployment, that is, the number of pods
  selector:             # Label selector
    matchLabels:
      app: nginx
      version: v1
  template:
    metadata:
      labels:           # Pod label
        app: nginx
        version: v1
    spec:
      containers:
        - image: swr.ap-southeast-1.myhuaweicloud.com/dev-container/nginx:v1 # The image used is nginx:v1.
          name: container-0
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
        imagePullSecrets:
          - name: default-secret
```

- Step 2** Create a Deployment of the second version. The following uses nginx-v2 as an example. Example YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-v2
spec:
  replicas: 2          # Number of replicas of the Deployment, that is, the number of pods
  selector:             # Label selector
    matchLabels:
      app: nginx
      version: v2
  template:
    metadata:
      labels:           # Pod label
        app: nginx
        version: v2
    spec:
      containers:
        - image: swr.ap-southeast-1.myhuaweicloud.com/dev-container/nginx:v2 # The image used is nginx:v2.
          name: container-0
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
        imagePullSecrets:
          - name: default-secret
```

You can log in to the CCE console to view the deployment status.

----End

## Step 2: Implement Grayscale Release

- Step 1** Create a LoadBalancer Service for the Deployment. Do not specify the version in the selector. Enable the Service to select the pods of the Deployments of two versions. Example YAML:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.id: 586c97da-a47c-467c-a615-bd25a20de39c  # ID of the ELB load balancer. Replace it with the actual value.
  name: nginx
spec:
  ports:
    - name: service0
      port: 80
      protocol: TCP
      targetPort: 80
  selector:          # The selector does not contain version information.
    app: nginx
  type: LoadBalancer # Service type (LoadBalancer)
```

- Step 2** Run the following command to test the access:

```
for i in {1..10}; do curl <EXTERNAL_IP>; done;
```

<EXTERNAL\_IP> indicates the IP address of the ELB load balancer.

The command output is as follows. Half of the responses are from the Deployment of version v1, and the other half are from version v2.

```
Nginx-v2
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v2
Nginx-v1
Nginx-v2
Nginx-v1
Nginx-v2
Nginx-v2
```

- Step 3** Use the console or kubectl to adjust the number of replicas of the Deployments. Change the number of replicas to 4 for v1 and 1 for v2.

```
kubectl scale deployment/nginx-v1 --replicas=4
```

```
kubectl scale deployment/nginx-v2 --replicas=1
```

- Step 4** Run the following command to test the access again:

```
for i in {1..10}; do curl <EXTERNAL_IP>; done;
```

<EXTERNAL\_IP> indicates the IP address of the ELB load balancer.

In the command output, among the 10 access requests, only two responses are from the v2 version. The response ratio of the v1 and v2 versions is the same as the ratio of the number of replicas of the v1 and v2 versions, that is, 4:1. Grayscale release is implemented by controlling the number of replicas of services of different versions.

```
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v2
Nginx-v1
Nginx-v2
Nginx-v1
Nginx-v1
Nginx-v1
```

 **NOTE**

If the ratio of v1 to v2 is not 4:1, you can set the number of access times to a larger value, for example, 20. Theoretically, the more the times, the closer the response ratio between v1 and v2 is to 4:1.

----End

### Step 3: Implement Blue-Green Deployment

**Step 1** Create a LoadBalancer Service for a deployed Deployment and specify that the v1 version is used. Example YAML:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.id: 586c97da-a47c-467c-a615-bd25a20de39c  # ID of the ELB load balancer. Replace it with the actual value.
  name: nginx
spec:
  ports:
    - name: service0
      port: 80
      protocol: TCP
      targetPort: 80
  selector:          # Set the version to v1 in the selector.
    app: nginx
    version: v1
  type: LoadBalancer # Service type (LoadBalancer)
```

**Step 2** Run the following command to test the access:

```
for i in {1..10}; do curl <EXTERNAL_IP>; done;
```

<EXTERNAL\_IP> indicates the IP address of the ELB load balancer.

The command output is as follows (all responses are from the v1 version):

```
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v1
Nginx-v1
```

**Step 3** Use the console or kubectl to modify the selector of the Service so that the v2 version is selected.

```
kubectl patch service nginx -p '{"spec":{"selector":{"version":"v2"}}}'
```

**Step 4** Run the following command to test the access again:

```
for i in {1..10}; do curl <EXTERNAL_IP>; done;
```

<EXTERNAL\_IP> indicates the IP address of the ELB load balancer.

The returned results show that all responses are from the v2 version. The blue-green deployment is successfully implemented.

```
Nginx-v2  
Nginx-v2  
Nginx-v2  
Nginx-v2  
Nginx-v2  
Nginx-v2  
Nginx-v2  
Nginx-v2  
Nginx-v2  
Nginx-v2
```

-----End

## 13.3 Using Nginx Ingress to Implement Grayscale Release and Blue-Green Deployment

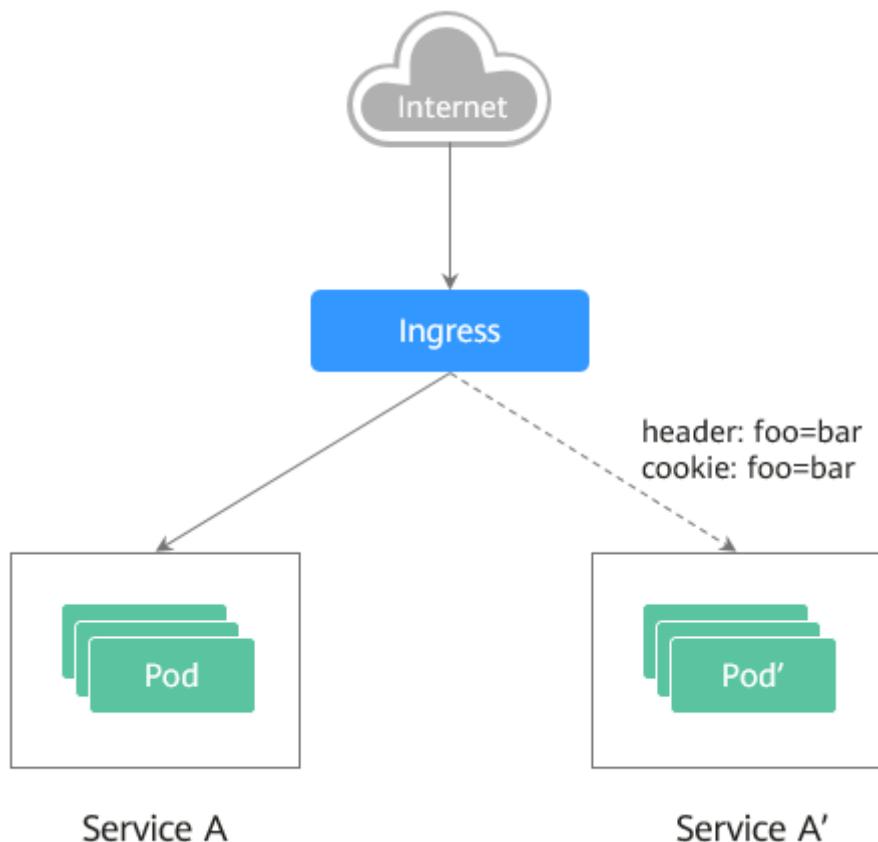
This section describes the scenarios and practices of using Nginx Ingress to implement grayscale release and blue-green deployment.

### Scenario

Nginx Ingress supports three traffic division policies based on the header, cookie, and service weight. Based on these policies, the following two release scenarios can be implemented:

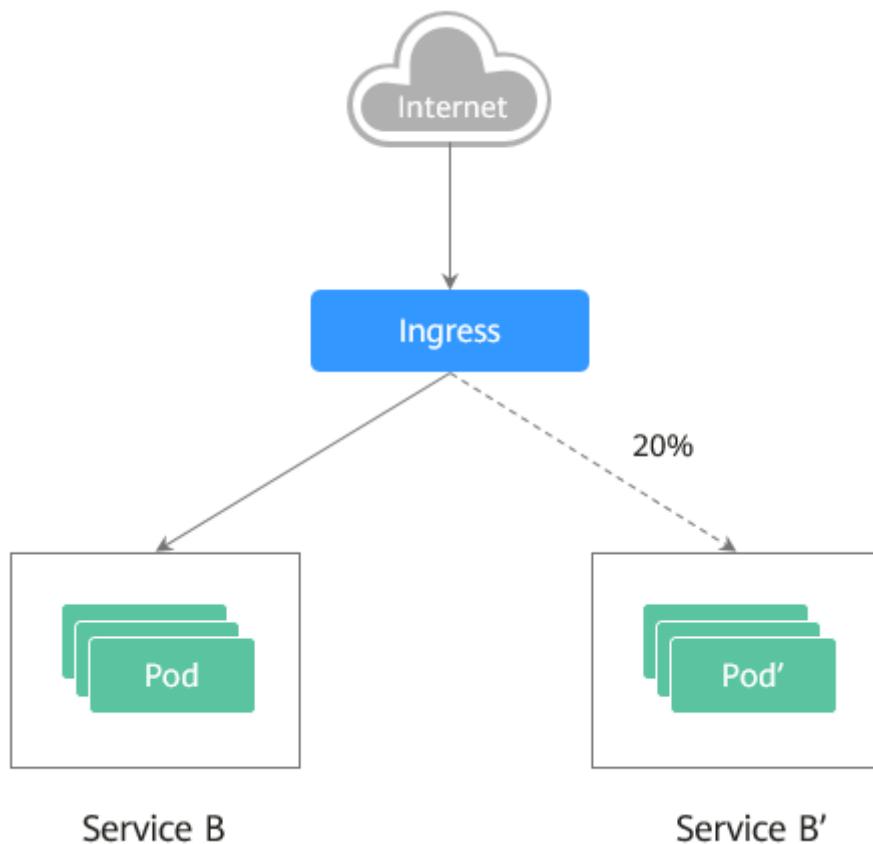
- **Scenario 1: Split some user traffic to the new version.**

Assume that Service A that provides layer-7 networking is running. A new version is ready to go online, but you do not want to replace the original Service A. You want to forward the user requests whose header or cookie contains **foo=bar** to the new version of Service A. After the new version runs stably for a period of time, you can gradually bring the new version online and smoothly bring the old version offline. The following figure shows the process:



- **Scenario 2: Split a certain proportion of traffic to the new version.**

Assume that Service B that provides layer-7 services is running. After some problems are resolved, a new version of Service B needs to be released. However, you do not want to replace the original Service B. Instead, you want to switch 20% traffic to the new version of Service B. After the new version runs stably for a period of time, you can switch all traffic from the old version to the new version and smoothly bring the old version offline.



## Annotations

Nginx Ingress supports release and testing in different scenarios by configuring annotations for grayscale release, blue-green deployment, and A/B testing. The implementation process is as follows: Create two ingresses for the service. One is a common ingress, and the other is an ingress with the annotation **nginx.ingress.kubernetes.io/canary: "true"**, which is called a canary ingress. Configure a traffic division policy for the canary ingress. The two ingresses cooperate with each other to implement release and testing in multiple scenarios. The annotation of Nginx Ingress supports the following rules:

- **nginx.ingress.kubernetes.io/canary-by-header**

Header-based traffic division, which is applicable to grayscale release. If the request header contains the specified header name and the value is **always**, the request is forwarded to the backend service defined by the canary ingress. If the value is **never**, the request is not forwarded and a rollback to the source version can be performed. If other values are used, the annotation is ignored and the request traffic is allocated according to other rules based on the priority.

- **nginx.ingress.kubernetes.io/canary-by-header-value**

This rule must be used together with canary-by-header. You can customize the value of the request header, including but not limited to **always** or **never**. If the value of the request header matches the specified custom value, the request is forwarded to the corresponding backend service defined by the canary ingress. If the values do not match, the annotation is ignored and the request traffic is allocated according to other rules based on the priority.

- **nginx.ingress.kubernetes.io/canary-by-header-pattern**

This rule is similar to canary-by-header-value. The only difference is that this annotation uses a regular expression, not a fixed value, to match the value of the request header. If this annotation and canary-by-header-value exist at the same time, this one will be ignored.

- **nginx.ingress.kubernetes.io/canary-by-cookie**

Cookie-based traffic division, which is applicable to grayscale release. Similar to canary-by-header, this annotation is used for cookies. Only **always** and **never** are supported, and the value cannot be customized.

- **nginx.ingress.kubernetes.io/canary-weight**

Traffic is divided based on service weights, which is applicable to blue-green deployment. This annotation indicates the percentage of traffic allocated by the canary ingress. The value ranges from 0 to 100. For example, if the value is set to **100**, all traffic is forwarded to the backend service backing the canary ingress.

 NOTE

- The preceding annotation rules are evaluated based on the priority. The priority is as follows: canary-by-header -> canary-by-cookie -> canary-weight.
- When an ingress is marked as a canary ingress, all non-canary annotations except **nginx.ingress.kubernetes.io/load-balance** and **nginx.ingress.kubernetes.io/upstream-hash-by** are ignored.
- For more information, see [Annotations](#).

## Prerequisites

- To use Nginx Ingress to implement grayscale release of a cluster, you need to install the nginx-ingress add-on as the Ingress Controller and expose a unified traffic entrance externally. For details, see [Installing the Add-on](#).
- The Nginx image has been uploaded to SWR. The Nginx images have two versions. The welcome pages are **Old Nginx** and **New Nginx**.

## Resource Creation

You can use YAML to deploy Deployments and Services in either of the following ways:

- On the **Create Deployment** page, click **Create YAML** on the right and edit the YAML file in the window.
- Save the sample YAML file in this section as a file and use kubectl to specify the YAML file. For example, run the **kubectl create -f xxx.yaml** command.

## Step 1: Deploy Services of Two Versions

Two versions of Nginx are deployed in the cluster, and Nginx Ingress is used to provide layer-7 domain name access for external systems.

**Step 1** Create a Deployment and Service for the first version. This section uses old-nginx as an example. Example YAML:

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

```
name: old-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: old-nginx
  template:
    metadata:
      labels:
        app: old-nginx
    spec:
      containers:
        - image: swr.ap-southeast-1.myhuaweicloud.com/dev-container/nginx:old  # The image used is
          nginx:old.
          name: container-0
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
          imagePullSecrets:
            - name: default-secret

---
apiVersion: v1
kind: Service
metadata:
  name: old-nginx
spec:
  selector:
    app: old-nginx
  ports:
    - name: service0
      targetPort: 80
      port: 8080
      protocol: TCP
      type: NodePort
```

**Step 2** Create a Deployment and Service for the second version. This section uses new-nginx as an example. Example YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: new-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: new-nginx
  template:
    metadata:
      labels:
        app: new-nginx
    spec:
      containers:
        - image: swr.ap-southeast-1.myhuaweicloud.com/dev-container/nginx:new  # The image used is
          nginx:new.
          name: container-0
          resources:
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
          imagePullSecrets:
```

```
- name: default-secret

---
apiVersion: v1
kind: Service
metadata:
  name: new-nginx
spec:
  selector:
    app: new-nginx
  ports:
    - name: service0
      targetPort: 80
      port: 8080
      protocol: TCP
      type: NodePort
```

You can log in to the CCE console to view the deployment status.

- Step 3** Create an ingress to expose the service and point to the service of the old version.  
Example YAML:

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: gray-release
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx # Use the Nginx ingress.
    kubernetes.io/elb.port: '80'
spec:
  rules:
    - host: www.example.com
      http:
        paths:
          - path: '/'
            backend:
              serviceName: old-nginx # Specify old-nginx as the backend service.
              servicePort: 80
```

- Step 4** Run the following command to verify the access:

```
curl -H "Host: www.example.com" http://<EXTERNAL_IP>
```

In the preceding command, <EXTERNAL\_IP> indicates the external IP address of the Nginx ingress.

Expected outputs:

```
Old Nginx
```

```
----End
```

## Step 2: Launch the New Version of the Service in Grayscale Release Mode

Set the traffic division policy for the service of the new version. CCE supports the following policies for grayscale release and blue-green deployment:

**Header-based, cookie-based, and weight-based** traffic division rules

Grayscale release can be implemented based on all these policies. Blue-green deployment can be implemented by adjusting the new service weight to 100%. For details, see the following examples.

 CAUTION

Pay attention to the following:

- Only one canary ingress can be defined for the same service so that the backend service supports a maximum of two versions.
- Even if the traffic is completely switched to the canary ingress, the old version service must still exist. Otherwise, an error is reported.

• **Header-based rules**

In the following example, only the requests whose header contains **Region** and the value is **bj** or **gz** can be forwarded to the service of the new version for users in Beijing and Guangzhou.

- a. Create a canary ingress, set the backend service to the one of the new versions, and add annotations.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: canary-ingress
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"          # Enable canary.
    nginx.ingress.kubernetes.io/canary-by-header: "Region"
    nginx.ingress.kubernetes.io/canary-by-header-pattern: "bj|gz"  # Requests whose header
contains Region with the value bj or gz are forwarded to the canary ingress.
    kubernetes.io/elb.port: '80'
spec:
  rules:
    - host: www.example.com
      http:
        paths:
          - path: '/'
            backend:
              serviceName: new-nginx   # Specify new-nginx as the backend service.
              servicePort: 80
```

- b. Run the following command to test the access:

```
$ curl -H "Host: www.example.com" -H "Region: bj" http://<EXTERNAL_IP>
New Nginx
$ curl -H "Host: www.example.com" -H "Region: sh" http://<EXTERNAL_IP>
Old Nginx
$ curl -H "Host: www.example.com" -H "Region: gz" http://<EXTERNAL_IP>
New Nginx
$ curl -H "Host: www.example.com" http://<EXTERNAL_IP>
Old Nginx
```

In the preceding command, <EXTERNAL\_IP> indicates the external IP address of the Nginx ingress.

Only requests whose header contains **Region** with the value **bj** or **gz** are responded by the service of the new version.

• **Cookie-based rules**

In the following example, only the requests whose cookie contains **user\_from\_bj** can be forwarded to the service of the new version for users in Beijing.

- a. Create a canary ingress, set the backend service to the one of the new versions, and add annotations.

 NOTE

If you have created a canary ingress in the preceding steps, delete it and then perform this step to create a canary ingress.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: canary-ingress
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"          # Enable canary.
    nginx.ingress.kubernetes.io/canary-by-cookie: "user_from_bj"  # Requests whose cookie
contains user_from_bj are forwarded to the canary ingress.
    kubernetes.io/elb.port: '80'
spec:
  rules:
    - host: www.example.com
      http:
        paths:
          - path: '/'
            backend:
              serviceName: new-nginx   # Specify new-nginx as the backend service.
              servicePort: 80
```

- b. Run the following command to test the access:

```
$ curl -s -H "Host: www.example.com" --cookie "user_from_bj=always" http://<EXTERNAL_IP>
New Nginx
$ curl -s -H "Host: www.example.com" --cookie "user_from_gz=always" http://<EXTERNAL_IP>
Old Nginx
$ curl -s -H "Host: www.example.com" http://<EXTERNAL_IP>
Old Nginx
```

In the preceding command, <EXTERNAL\_IP> indicates the external IP address of the Nginx ingress.

Only requests whose cookie contains **user\_from\_bj** with the value **always** are responded by the service of the new version.

- **Service weight-based rules**

Example 1: Only 20% of the traffic is allowed to be forwarded to the service of the new version to implement grayscale release.

- a. Create a canary ingress and add annotations to import 20% of the traffic to the backend service of the new version.

 NOTE

If you have created a canary ingress in the preceding steps, delete it and then perform this step to create a canary ingress.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: canary-ingress
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"          # Enable canary.
    nginx.ingress.kubernetes.io/canary-weight: "20"    # Forward 20% of the traffic to the canary
ingress.
    kubernetes.io/elb.port: '80'
spec:
  rules:
    - host: www.example.com
      http:
```

```
paths:
- path: '/'
  backend:
    serviceName: new-nginx # Specify new-nginx as the backend service.
    servicePort: 80
```

- b. Run the following command to test the access:

```
$ for i in {1..20}; do curl -H "Host: www.example.com" http://<EXTERNAL_IP>; done;
Old Nginx
Old Nginx
Old Nginx
New Nginx
Old Nginx
New Nginx
Old Nginx
New Nginx
Old Nginx
```

In the preceding command, <EXTERNAL\_IP> indicates the external IP address of the Nginx ingress.

It can be seen that there is a 4/20 probability that the service of the new version responds, which complies with the setting of the service weight of 20%.

#### NOTE

After traffic is divided based on the weight (20%), the probability of accessing the new version is close to 20%. The traffic ratio may fluctuate within a small range, which is normal.

Example 2: Allow all traffic to be forwarded to the service of the new version to implement blue-green deployment.

- a. Create a canary ingress and add annotations to import 100% of the traffic to the backend service of the new version.

#### NOTE

If you have created a canary ingress in the preceding steps, delete it and then perform this step to create a canary ingress.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: canary-ingress
  namespace: default
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"      # Enable canary.
    nginx.ingress.kubernetes.io/canary-weight: "100"  # All traffic is forwarded to the canary
spec:
  rules:
    - host: www.example.com
      http:
        paths:
```

```
- path: '/'
backend:
  serviceName: new-nginx  # Specify new-nginx as the backend service.
  servicePort: 80
```

- b. Run the following command to test the access:

```
$ for i in {1..10}; do curl -H "Host: www.example.com" http://<EXTERNAL_IP>; done;
New Nginx
```

In the preceding command, <EXTERNAL\_IP> indicates the external IP address of the Nginx ingress.

All access requests are responded by the service of the new version, and the blue-green deployment is successfully implemented.

# 14 Batch Computing

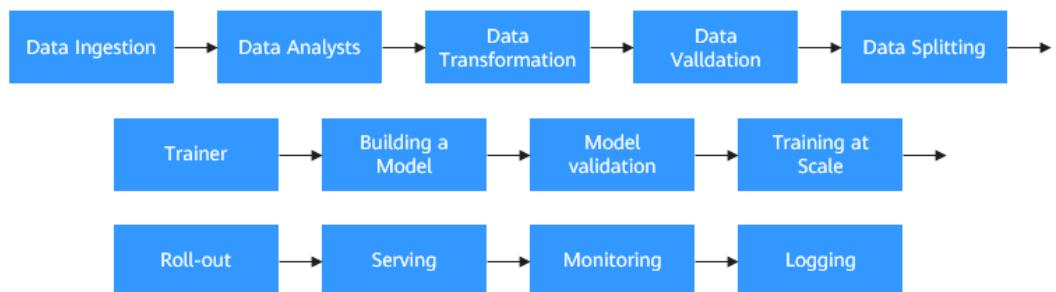
## 14.1 Using Kubeflow and Volcano to Train an AI Model

Kubernetes has become the de facto standard for cloud native application orchestration and management. An increasing number of applications are migrated to Kubernetes. AI and machine learning inherently involve a large number of computing-intensive tasks. Kubernetes is a preferential tool for developers building AI platforms because of its excellent capabilities in resource management, application orchestration, and O&M monitoring.

### Emergence and Constraints of Kubeflow

Building an end-to-end AI computing platform based on Kubernetes is complex and demanding. More than a dozen of phases is required, as shown in the following diagram. Apart from the familiar model training phase, the process also includes data collection, preprocessing, resource management, feature extraction, data verification, model management, model release, and monitoring, as shown in **Figure 14-1**. If AI algorithm engineers want to run a model training task, they have to build an entire AI computing platform first. Imagine how time- and labor-consuming that is and how much knowledge and experience it requires.

**Figure 14-1** Model training

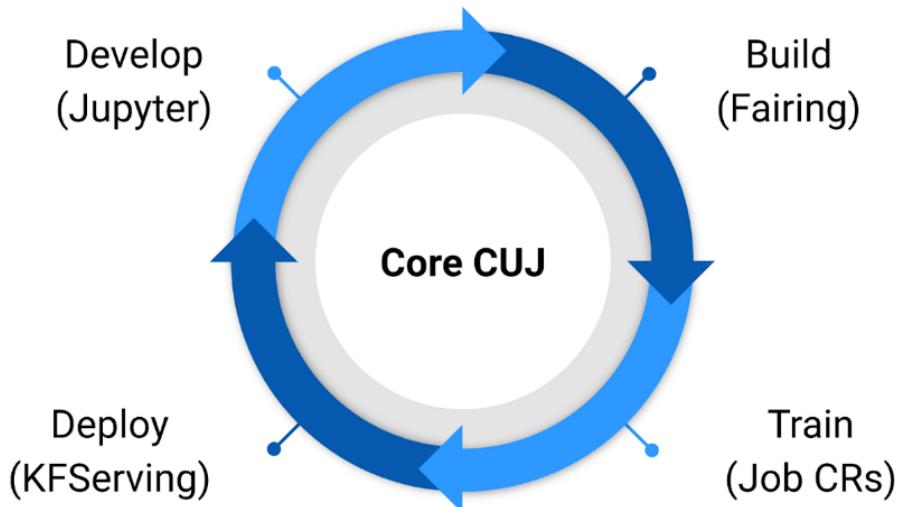


This is where Kubeflow comes in. Created in 2017, Kubeflow is a container and Kubernetes-based platform for agile deployment, development, training, release, and management in the machine learning field. It leverages cloud native technologies to make it faster and easier for data scientists, machine learning

engineers, and system O&M personnel to deploy, use, and manage popular machine learning software.

Kubeflow 1.0 is now available, providing capabilities in development, building, training, and deployment that cover the entire process of machine learning and deep learning for enterprise users.

Diagram:



With Kubeflow 1.0, you first develop a model using Jupyter, and then set up containers using tools such as Fairing (SDK). Next, you create Kubernetes resources to train the model. After the training is complete, you create and deploy servers for inference using KFServing. This is how you use Kubeflow to establish an end-to-end agile process of a machine learning task. This process can be fully automated using pipelines, which help achieve DevOps in the AI field.

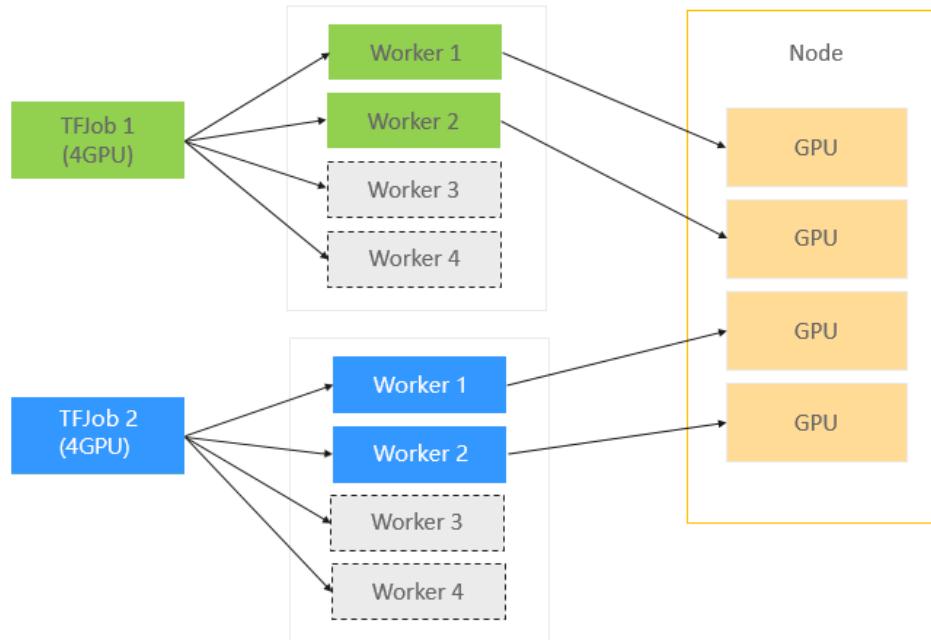
## Kubernetes Pain Points

Does that mean we can now sit back and relax? Not yet. Kubeflow uses the default scheduler of Kubernetes, which was initially designed for long services. Its scheduling capability is inadequate for tasks that involve batch computing and elastic scheduling in AI and big data scenarios. The main constraints are as follows:

### Resource preemption

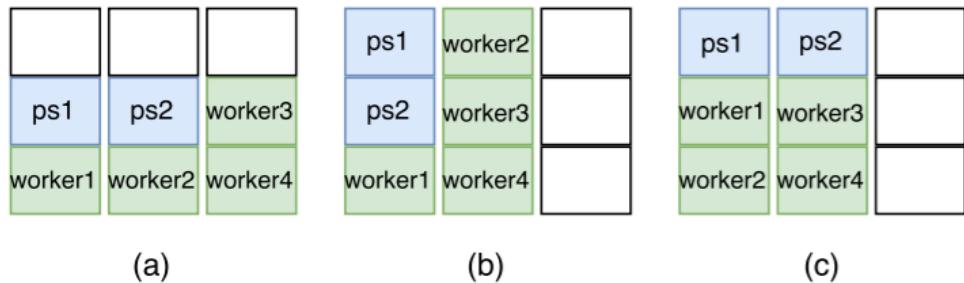
A TensorFlow job involves two roles: parameter server (ps) and worker. Only when pods of these two roles run properly at the same time can a TensorFlow job be executed normally. However, the default scheduler is insensitive to the roles of pods in a TensorFlow job. Pods are treated identically and scheduled one by one. This causes problems when there are multiple jobs to schedule and cluster resources are scarce. Each job could end up being allocated with only part of the resources it needs to finish the execution. That is, resources are used up while no job can be successfully executed. To better illustrate this dilemma, assume that you want to run two TensorFlow jobs, namely, TFJob1 and TFJob2. Each of these

jobs has four workers, which means each job requires four GPUs to run. However, your cluster only has four available GPUs in total. In this case, with the default scheduler, TFJob1 and TFJob2 could end up being allocated two GPUs each. They are waiting each other to finish and release the resources. However, this will not happen until you manually intervene. The deadlock created in this situation cause resource wastes and low efficiency in job execution.



### Lack of affinity-based scheduling

In distributed training, data is frequently exchanged between parameter servers and workers. To ensure higher efficiency, parameter servers and workers of the same job should be scheduled to the same node for faster transmission using local networks. However, the default scheduler is insensitive to the affinity between parameter servers and workers of the same job. Pods are randomly scheduled instead. As shown in the following figure, assume that you want to run two TensorFlow jobs with each having one ps and two workers. With the default scheduler, the scheduling results could be any of the following three situations. However, only result (c) can deliver the highest efficiency. In (c), the ps and the workers can use the local network to communicate more efficiently and shorten the training time.



## Volcano, a Perfect Batch Scheduling System for Accelerating AI Computing

Volcano is an enhanced batch scheduling system for high-performance computing workloads running on Kubernetes. It complements Kubernetes in machine learning, deep learning, HPC, and big data computing scenarios, providing capabilities such as gang scheduling, computing task queue management, task-topology, and GPU affinity scheduling. In addition, Volcano enhances batch task creation and lifecycle management, fair-share, binpack, and other Kubernetes-native capabilities. It fully addresses the constraints of Kubeflow in distributed training mentioned above.



For more information about Volcano, visit <https://github.com/volcano-sh/volcano>.

## Using Volcano in Huawei Cloud

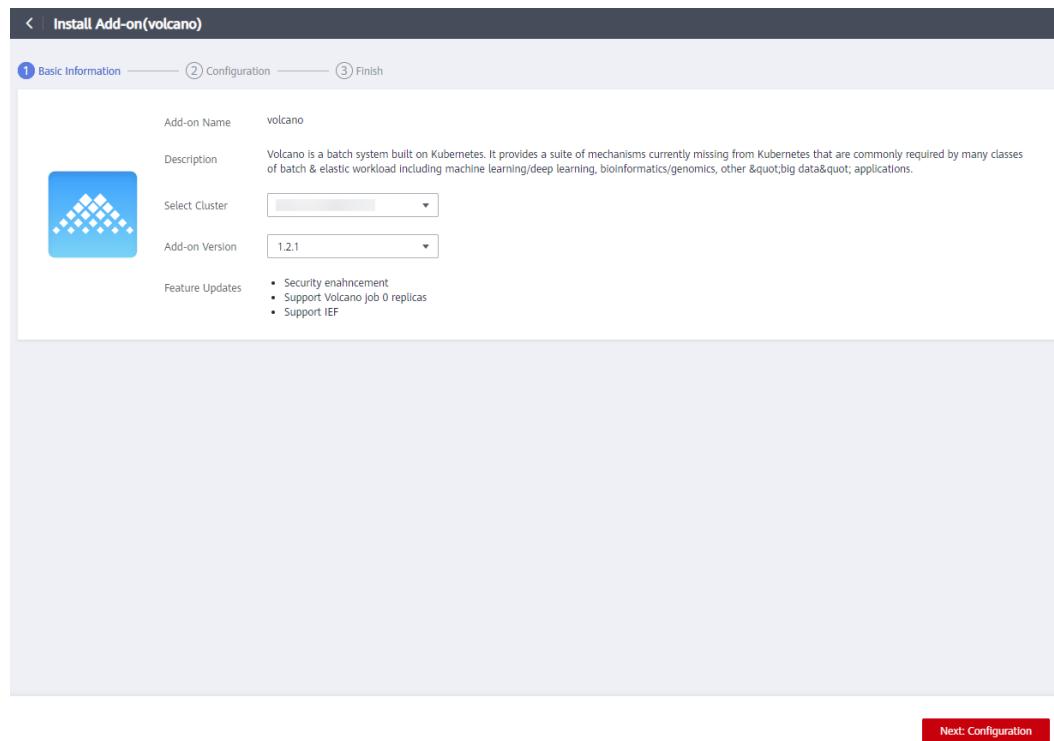
The convergence of Kubeflow and Volcano, two open-source projects, greatly simplifies and accelerates AI computing workloads running on Kubernetes. The two projects have been recognized by an increasing number of players in the field and applied in production environments. Volcano is used in Huawei Cloud CCE, Cloud Container Instance (CCI), and Kubernetes-Native Batch Computing Solution. Volcano will continue to iterate with optimized algorithms, enhanced capabilities such as intelligent scheduling, and new inference features such as GPU Share, to further improve the efficiency of Kubeflow batch training and inference.

## Implementing Typical Distributed AI Training Jobs

This section describes how to perform distributed training of a digital image classification model using the MNIST dataset based on Kubeflow and Volcano.

- Step 1** Log in to the CCE console and create a CCE cluster. For details, see [Buying a CCE Cluster](#).
- Step 2** Deploy Volcano on the created CCE cluster.

In the navigation pane on the left, choose **Add-ons**. On the **Add-on Marketplace** tab page, click **Install Add-on** under volcano. In the **Basic Information** area on the **Install Add-on** page, select the cluster and Volcano version, and click **Next: Configuration**.

**Figure 14-2** Installing the volcano add-on

The volcano add-on has no configuration parameters. Click **Install** and wait until the installation is complete.

### Step 3 Deploy the Kubeflow environment.

#### 1. Install kfctl and set environment variables.

##### a. Set environment variables as follows:

```
export KF_NAME=<your choice of name for the Kubeflow deployment>
export BASE_DIR=<path to a base directory>
export KF_DIR=${BASE_DIR}/${KF_NAME}
export CONFIG_URI="https://raw.githubusercontent.com/kubeflow/manifests/v1.0-branch/kfdef/
kfctl_k8s_istio.v1.0.2.yaml"
```

##### b. Install kfctl.

Download kfctl from <https://github.com/kubeflow/kfctl/releases/tag/v1.0.2>.

```
tar -xvf kfctl_v1.0.2_<platform>.tar.gz
chmod +x kfctl
mv kfctl /usr/local/bin/
```

#### 2. Deploy Kubeflow.

```
mkdir -p ${KF_DIR}
cd ${KF_DIR}
kfctl apply -V -f ${CONFIG_URI}
```

Delete the following PVCs and create four PVCs with the same name in CCE.

```
# kubectl get pvc -n kubeflow
NAME      STATUS   VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
katib-mysql  Pending
metadata-mysql  Pending
minio-pv-claim  Pending
mysql-pv-claim  Pending
3m56s
4m2s
3m55s
3m54s
```

### Step 4 Deploy the MNIST dataset.

1. Download **kubeflow/examples** to the local host and select an operation guide based on the environment.

```
yum install git
git clone https://github.com/kubeflow/examples.git
```

2. Install python3.

```
wget https://www.python.org/ftp/python/3.6.8/Python-3.6.8.tgz
tar -zvxf Python-3.6.8.tgz
cd Python-3.6.8 ./configure
make make install
```

After the installation, run the following commands to check whether the installation is successful:

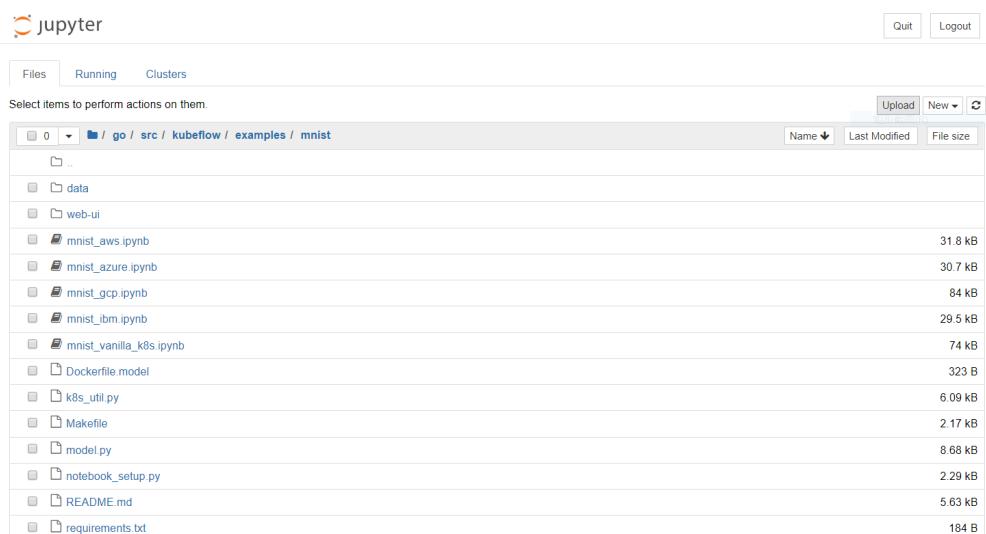
```
python3 -V
pip3 -V
```

3. Install and start Jupyter Notebook.

```
pip3 install jupyter notebook
jupyter notebook --allow-root
```

4. Configure an SSH tunnel on PuTTY and remotely connect to the notebook.

5. After the connection is successful, enter **localhost:8000** in the address box of a browser to log in to the notebook.



6. Create a distributed training job as prompted by Jupyter. Set the value of **schedulerName** to **volcano** to enable the Volcano scheduler.

```
kind: TFJob
metadata:
  name: {train_name}
spec:
  schedulerName: volcano
  tfReplicaSpecs:
    Ps:
      replicas: {num_ps}
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          serviceAccount: default-editor
          containers:
            - name: tensorflow
              command:
                ...
              env:
                ...
              image: {image}
```

```
        workingDir: /opt
        restartPolicy: OnFailure
Worker:
  replicas: 1
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
      serviceAccount: default-editor
      containers:
        - name: tensorflow
          command:
            ...
          env:
            ...
          image: {image}
          workingDir: /opt
          restartPolicy: OnFailure
```

**Step 5** Submit the job and start the training.

```
kubectl apply -f mnist.yaml
```

```
root@ecs-1953:~/tmp# kubectl get po
NAME           READY   STATUS    RESTARTS   AGE
tensorflow-mnist-ps-0   1/1     Running   0          5s
tensorflow-mnist-worker-0 1/1     Running   0          5s
```

After the training job is complete, you can query the training results on the Kubeflow UI. This is how you run a simple distributed training job using Kubeflow and Volcano. Kubeflow simplifies TensorFlow job configuration. Volcano, with simply one more line of configuration, saves you significant time and effort in large-scale distributed training by providing capabilities such as gang scheduling and task topology to eliminate deadlocks and achieve affinity scheduling.

----End

## 14.2 Running Kubeflow in CCE

### 14.2.1 Deploying Kubeflow

This section describes the process of deploying Kubeflow on Huawei Cloud CCE and using Kubeflow to build simple TensorFlow training jobs. It also compares the training performance in the single-GPU and multi-GPU scenarios.

For details about the deployment process, see the official document at <https://www.kubeflow.org/docs/started/getting-started/>.

#### Prerequisites

- A cluster named clusterA has been created on CCE. The cluster has an available GPU node that has two or more GPUs.
- An EIP has been bound to the node, and kubectl has been configured.

#### Installing ksonnet

You can click [here](#) to view the latest ksonnet version. The latest version is v0.13.1. The installation procedure is as follows:

```
export KS_VER=0.13.1
export KS_PKG=ks_${KS_VER}_linux_amd64
wget -O /tmp/${KS_PKG}.tar.gz
https://github.com/ksonnet/ksonnet/releases/download/v${KS_VER}/${KS_PKG}.tar.gz
mkdir -p ${HOME}/bin
tar -xvf /tmp/${KS_PKG}.tar.gz -C ${HOME}/bin
cp ${HOME}/bin/${KS_PKG}/ks /usr/local/bin
```

## Downloading kfctl.sh

Run the following commands:

```
export KUBEFLOW_SRC=/home/kubeflow_src
mkdir ${KUBEFLOW_SRC}
cd ${KUBEFLOW_SRC}
export KUBEFLOW_TAG=v0.4.1
curl
https://raw.githubusercontent.com/kubeflow/kubeflow/${KUBEFLOW_TAG}/scripts/download.sh | bash
```

- **KUBEFLOW\_SRC** is the download directory of Kubeflow.
- **KUBEFLOW\_TAG** indicates the kubeflow version, for example, v0.4.1.

## Deploying Kubeflow

Run the following commands:

```
 ${KUBEFLOW_SRC}/scripts/kfctl.sh init ${KFAPP} --platform none
cd ${KFAPP}
${KUBEFLOW_SRC}/scripts/kfctl.sh generate k8s
${KUBEFLOW_SRC}/scripts/kfctl.sh apply k8s
```

- **KFAPP** indicates the Kubeflow Deployment name.

After execution is complete, run the **kubectl get po -n kubeflow** command to check whether the related resources are started normally. Because the storage has not been configured, some pods are not running.

## Configuring the Storage Required by Kubeflow

Kubeflow v0.4.1 depends on the following storage volumes:

- katib-mysql
- mysql-pv-claim
- minio-pv-claim

Therefore, select cluster **clusterA** and namespace **kubeflow** on the **Resource Management > Storage** page of the CCE console to create three storage volumes.

After the volume creation is complete, modify the **volume-name** parameter of the following Deployments:

```
kubectl edit deploy minio -nkubeflow
:9s/minio-pv-claim/cce-sfs-kubeflow-minio/g
:wq!
kubectl edit deploy mysql -nkubeflow
:9s/mysql-pv-claim/cce-sfs-kubeflow-mysql/g
:wq!
kubectl edit deploy vizier-db -nkubeflow
:9s/katib-mysql/cce-sfs-kubeflow-katib/g
:wq!
```

After a period of time, all pods are in the running state.

## 14.2.2 Training TensorFlow Models

After Kubeflow is successfully deployed, it is easy to use the ps-worker mode to train TensorFlow models. This section provides a TensorFlow training example released at the official Kubeflow website. For more information, see <https://www.kubeflow.org/docs/guides/components/tftraining/>.

### Creating a TfCnn Training Job

Run the following commands to create a TfCnn training job:

```
CNN_JOB_NAME=mycnnjob
VERSION=v0.4.0

ks init ${CNN_JOB_NAME}
cd ${CNN_JOB_NAME}
ks registry add kubeflow-git github.com/kubeflow/kubeflow/tree/${VERSION}/kubeflow
ks pkg install kubeflow-git/examples

ks generate tf-job-simple-v1beta1 ${CNN_JOB_NAME} --name=${CNN_JOB_NAME}
ks apply ${KF_ENV} -c ${CNN_JOB_NAME}
```

You can run the **ks env list** command to obtain the value of **\${KF\_ENV}**. In this example, the value of **\${KF\_ENV}** is **default**. After the execution is complete, run the **kubectl get po** command to view the result.

### Using a Single GPU for Training

The preceding training job can be implemented by GPUs. Perform the following steps to modify the TFJob configuration file:

```
vi ${KS_APP}/components/${CNN_JOB_NAME}.jsonnet
```

Replace the file content with the following content in the **mycnnjob.jsonnet** file:

```
local env = std.extVar("__ksonnet/environments");
local params = std.extVar("__ksonnet/params").components.mycnnjob;

local k = import "k.libsonnet";

local name = params.name;
local namespace = env.namespace;
local image = "gcr.io/kubeflow/tf-benchmarks-cpu:v20171202-bdab599-dirty-284af3";

local tfjob = {
    apiVersion: "kubeflow.org/v1beta1",
    kind: "TFJob",
    metadata: {
        name: name,
        namespace: namespace,
    },
    spec: {
        tfReplicaSpecs: {
            Worker: {
                replicas: 1,
                template: {
                    metadata: {
                        annotations: {
                            sidecar.istio.io/inject: "false"
                        }
                    },
                    spec: {

```

```
containers: [
  {
    args: [
      "python",
      "tf_cnn_benchmarks.py",
      "--batch_size=64",
      "--num_batches=100",
      "--model=resnet50",
      "--variable_update=parameter_server",
      "--flush_stdout=true",
      "--num_gpus=1",
      "--local_parameter_device(cpu",
      "--device=gpu",
      "--data_format=NHWC",
    ],
    image: "swr.ap-southeast-1.myhuaweicloud.com/wubowen585/tf-benchmarks-gpu:v0",
    name: "tensorflow",
    ports: [
      {
        containerPort: 2222,
        name: "tfjob-port",
      },
    ],
    resources: {
      limits: {
        "nvidia.com/gpu": 1,
      },
      workingDir: "/opt/tf-benchmarks/scripts/tf_cnn_benchmarks",
    },
  ],
  restartPolicy: "OnFailure",
},
],
Ps: {
  replicas: 1,
  template: {
    spec: {
      containers: [
        {
          args: [
            "python",
            "tf_cnn_benchmarks.py",
            "--batch_size=64",
            "--num_batches=100",
            "--model=resnet50",
            "--variable_update=parameter_server",
            "--flush_stdout=true",
            "--num_gpus=1",
            "--local_parameter_device(cpu",
            "--device=cpu",
            "--data_format=NHWC",
          ],
          image: "swr.ap-southeast-1.myhuaweicloud.com/wubowen585/tf-benchmarks-cpu:v0",
          name: "tensorflow",
          ports: [
            {
              containerPort: 2222,
              name: "tfjob-port",
            },
          ],
          resources: {
            limits: {
              cpu: 4,
            },
            workingDir: "/opt/tf-benchmarks/scripts/tf_cnn_benchmarks",
          },
        ],
      ],
    }
  }
}
```

```
        ],
        restartPolicy: "OnFailure",
    },
},
tfReplicaType: "PS",
},
},
};

k.core.v1.list.new([
tfjob,
])
```

After the replacement is complete, restart the TFJob. After running the **ks delete** command, wait for about 30 seconds to confirm that the TFJob has been deleted.

```
ks delete ${KF_ENV} -c ${CNN_JOB_NAME}
ks apply ${KF_ENV} -c ${CNN_JOB_NAME}
```

After the worker runs the job (about 5 minutes if a GPU is used), run the following command to view the running result:

```
kubectl get po
kubectl logs ${CNN_JOB_NAME}-worker-0
```

In this example, the CNN ResNet50 model is used to train randomly generated images based on the TensorFlow distributed architecture. 64 images (specified by **batch\_size**) are trained each time, and a total of 100 training steps (specified by **step**) are performed. The CPU performance (image/sec) at each training step is recorded. The training result shows that the training performance of a single P100 GPU is 158.62 images/sec.

```
INFO|2019-02-27T09:18:14|/opt/launcher.py|27| Running warm up
INFO|2019-02-27T09:18:21|/opt/launcher.py|27| Done warm up
INFO|2019-02-27T09:18:21|/opt/launcher.py|27| Step      Img/sec loss
INFO|2019-02-27T09:18:22|/opt/launcher.py|27| 1 images/sec: 167.1 +/- 0.0 (jitter = 0.0)      9.578
INFO|2019-02-27T09:18:25|/opt/launcher.py|27| 10     images/sec: 161.6 +/- 1.7 (jitter = 6.1)      8.477
INFO|2019-02-27T09:18:29|/opt/launcher.py|27| 20     images/sec: 161.7 +/- 1.3 (jitter = 6.7)      8.277
INFO|2019-02-27T09:18:33|/opt/launcher.py|27| 30     images/sec: 160.3 +/- 1.1 (jitter = 8.0)      8.247
INFO|2019-02-27T09:18:37|/opt/launcher.py|27| 40     images/sec: 160.4 +/- 0.9 (jitter = 7.7)      7.968
INFO|2019-02-27T09:18:41|/opt/launcher.py|27| 50     images/sec: 159.3 +/- 0.9 (jitter = 9.0)      8.016
INFO|2019-02-27T09:18:45|/opt/launcher.py|27| 60     images/sec: 159.7 +/- 0.8 (jitter = 8.9)      7.888
INFO|2019-02-27T09:18:49|/opt/launcher.py|27| 70     images/sec: 159.5 +/- 0.7 (jitter = 8.3)      7.881
INFO|2019-02-27T09:18:53|/opt/launcher.py|27| 80     images/sec: 159.5 +/- 0.6 (jitter = 7.5)      7.909
INFO|2019-02-27T09:18:57|/opt/launcher.py|27| 90     images/sec: 159.2 +/- 0.6 (jitter = 7.5)      8.084
INFO|2019-02-27T09:19:02|/opt/launcher.py|27| 100    images/sec: 159.0 +/- 0.6 (jitter = 7.4)      7.882
INFO|2019-02-27T09:19:02|/opt/launcher.py|27| -----
INFO|2019-02-27T09:19:02|/opt/launcher.py|27| total images/sec: 158.62
INFO|2019-02-27T09:19:02|/opt/launcher.py|27| -----
INFO|2019-02-27T09:19:02|/opt/launcher.py|80| Finished: python tf_cnn_benchmarks.py --batch_size=64 --num_batches=100 --mode l=resnet50 --variable_update=parameter_server --flush_stdout=true --num_gpus=1 --local_parameter_device=cpu --device=gpu --data_format=NHWC --job_name=worker --ps_hosts=mvcnnjob-ps-0.default.svc:2222 --worker_hosts=mvcnnjob-worker-0.default.svc:2222 --task_index=0
INFO|2019-02-27T09:19:02|/opt/launcher.py|84| Command ran successfully sleep for ever.
```

## Using Multiple GPUs for Training

To demonstrate the advantages of distributed TensorFlow jobs, two GPUs are used to run the same training job. In this example, the number of workers is changed to 2. You can perform the following procedure to change the number of workers:

```
vi ${KS_APP}/components/${CNN_JOB_NAME}.jsonnet
```

Change the number of worker replicas to 2.

```
spec: {
  tfReplicaSpecs: {
    Worker: {
      replicas: 2,
      template: {
        spec: {
          containers: [
            {
              args: [
                "python",
                "tf_cnn_benchmarks.py",
                "--batch_size=64",

```

Save the modification and restart the TFJob.

```
ks delete ${KF_ENV} -c ${CNN_JOB_NAME}
ks apply ${KF_ENV} -c ${CNN_JOB_NAME}
```

Wait for about 5 minutes and query the training results of the two workers.

```
kubectl get po
kubectl logs ${CNN_JOB_NAME}-worker-0
```

The training results show that the training performance of two workers is almost twice that of a single worker.

```
INFO|2019-02-27T09:29:17|/opt/launcher.py|27| Running warm up
INFO|2019-02-27T09:29:27|/opt/launcher.py|27| Done warm up
INFO|2019-02-27T09:29:27|/opt/launcher.py|27| Step      Img/sec loss
INFO|2019-02-27T09:29:27|/opt/launcher.py|27| 1 images/sec: 150.0 +/- 0.0 (jitter = 0.0)      8.762
INFO|2019-02-27T09:29:31|/opt/launcher.py|27| 10     images/sec: 154.7 +/- 1.6 (jitter = 5.0)      8.208
INFO|2019-02-27T09:29:35|/opt/launcher.py|27| 20     images/sec: 154.1 +/- 1.1 (jitter = 5.9)      7.983
INFO|2019-02-27T09:29:39|/opt/launcher.py|27| 30     images/sec: 154.2 +/- 0.9 (jitter = 7.2)      8.051
INFO|2019-02-27T09:29:43|/opt/launcher.py|27| 40     images/sec: 153.0 +/- 0.8 (jitter = 5.0)      7.896
INFO|2019-02-27T09:29:48|/opt/launcher.py|27| 50     images/sec: 152.8 +/- 0.8 (jitter = 4.8)      7.904
INFO|2019-02-27T09:29:52|/opt/launcher.py|27| 60     images/sec: 152.1 +/- 0.7 (jitter = 4.9)      7.930
INFO|2019-02-27T09:29:56|/opt/launcher.py|27| 70     images/sec: 151.9 +/- 0.7 (jitter = 5.0)      7.870
INFO|2019-02-27T09:30:00|/opt/launcher.py|27| 80     images/sec: 152.0 +/- 0.6 (jitter = 5.3)      8.026
INFO|2019-02-27T09:30:05|/opt/launcher.py|27| 90     images/sec: 152.0 +/- 0.6 (jitter = 4.9)      8.000
INFO|2019-02-27T09:30:09|/opt/launcher.py|27| 100    images/sec: 151.6 +/- 0.6 (jitter = 5.3)      7.820
INFO|2019-02-27T09:30:09|/opt/launcher.py|27| -----
INFO|2019-02-27T09:30:09|/opt/launcher.py|27| total images/sec: 304.11
INFO|2019-02-27T09:30:09|/opt/launcher.py|27| -----
INFO|2019-02-27T09:30:10|/opt/launcher.py|80| Finished: python tf_cnn_benchmarks.py --batch_size=64 --num_batches=100 --model=resnet50 --variable_update=parameter_server --flush_stdout=true --num_gpus=1 --local_parameter_device=cpu --device=gpu --data_format=NHWC --job_name=worker --ps_hosts=mycnnjob-ps-0.default.svc:2222 --worker_hosts=mycnnjob-worker-0.default.svc:2222,mycnnjob-worker-1.default.svc:2222 --task_index=0
```

## 14.3 Running Caffe in CCE

### 14.3.1 Prerequisites

This section provides an example for running Caffe on CCE to classify an image. For more information, see <https://github.com/BVLC/caffe/blob/master/examples/00-classification.ipynb>.

#### Pre-configuring OBS Storage Data

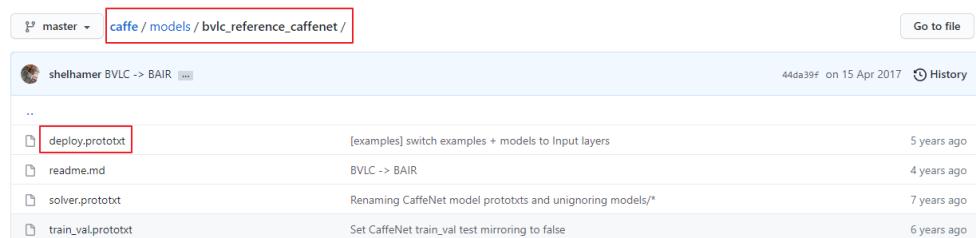
Create an OBS bucket and ensure that the following folders have been created and required files have been uploaded to the specified paths using the OBS Browser.

The folder name can be in the format of *File path in the bucket/File name*. You can search for the file download addresses in the specified paths of the specified project in GitHub, as shown in **1** and **2**.

1. models/bvlc\_reference\_caffenet/bvlc\_reference\_caffenet.caffemodel  
[https://github.com/BVLC/caffe/tree/master/models/bvlc\\_reference\\_caffenet](https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet)

name	caffemodel	caffemodel_url	license
BAIR/BVLC CaffeNet Model	bvlc_reference_caffenet.caffemodel	<a href="http://dl.caffe.berkeleyvision.org/bvlc_reference_caffenet.caffemodel">http://dl.caffe.berkeleyvision.org/bvlc_reference_caffenet.caffemodel</a>	unrestricted

2. models/bvlc\_reference\_caffenet/deploy.prototxt  
[https://github.com/BVLC/caffe/tree/master/models/bvlc\\_reference\\_caffenet](https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet)



3. python/caffe/imagenet/ilsvrc\_2012\_mean.npy  
<https://github.com/BVLC/caffe/tree/master/python/caffe/imagenet>



4. outputimg/
- An empty folder **outputimg** is created to store output files.
5. examples/images/cat.jpg  
<https://github.com/BVLC/caffe/blob/master/examples/00-classification.ipynb>
- Save the picture of the cat in the link.
6. data/ilsvrc12/\*

<https://github.com/BVLC/caffe/tree/master/data/ilsvrc12>

Obtain and execute the **get\_ilsvrc\_aux.sh** script. The script downloads a compressed package and decompresses it. After the script is executed, upload all decompressed files to the directory.

#### 7. caffeEx00.py

```
# set up Python environment: numpy for numerical routines, and matplotlib for plotting
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
# display plots in this notebook
#%matplotlib inline

# set display defaults
plt.rcParams['figure.figsize'] = (10, 10)      # large images
plt.rcParams['image.interpolation'] = 'nearest' # don't interpolate: show square pixels
plt.rcParams['image.cmap'] = 'gray' # use grayscale output rather than a (potentially misleading)
color heatmap

# The caffe module needs to be on the Python path;
# we'll add it here explicitly.
import sys
caffe_root = '/home/' # this file should be run from {caffe_root}/examples (otherwise change this
line)
sys.path.insert(0, caffe_root + 'python')

import caffe
# If you get "No module named _caffe", either you have not built pycaffe or you have the wrong path.

import os
#if os.path.isfile(caffe_root + 'models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel'):
#    print 'CaffeNet found.'
#else:
#    print 'Downloading pre-trained CaffeNet model...'
#    !./scripts/download_model_binary.py ..../models/bvlc_reference_caffenet

caffe.set_mode_cpu()

model_def = caffe_root + 'models/bvlc_reference_caffenet/deploy.prototxt'
model_weights = caffe_root + 'models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel'

net = caffe.Net(model_def,      # defines the structure of the model
                model_weights, # contains the trained weights
                caffe.TEST)   # use test mode (e.g., don't perform dropout)

# load the mean ImageNet image (as distributed with Caffe) for subtraction
mu = np.load(caffe_root + 'python/caffe/imagenet/ilsvrc_2012_mean.npy')
mu = mu.mean(1).mean(1) # average over pixels to obtain the mean (BGR) pixel values
print 'mean-subtracted values:', zip('BGR', mu)

# create transformer for the input called 'data'
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})

transformer.set_transpose('data', (2,0,1)) # move image channels to outermost dimension
transformer.set_mean('data', mu)           # subtract the dataset-mean value in each channel
transformer.set_raw_scale('data', 255)      # rescale from [0, 1] to [0, 255]
transformer.set_channel_swap('data', (2,1,0)) # swap channels from RGB to BGR

# set the size of the input (we can skip this if we're happy
# with the default; we can also change it later, e.g., for different batch sizes)
net.blobs['data'].reshape(50,          # batch size
                        3,            # 3-channel (BGR) images
                        227, 227)    # image size is 227x227

image = caffe.io.load_image(caffe_root + 'examples/images/cat.jpg')
transformed_image = transformer.preprocess('data', image)
plt.imshow(image)
```

```
plt.savefig(caffe_root + 'outputimg/img1.png')

# copy the image data into the memory allocated for the net
net.blobs['data'].data[...] = transformed_image

### perform classification
output = net.forward()

output_prob = output['prob'][0] # the output probability vector for the first image in the batch

print 'predicted class is:', output_prob.argmax()

# load ImageNet labels
labels_file = caffe_root + 'data/ilsvrc12/synset_words.txt'
#if not os.path.exists(labels_file):
#    !./data/ilsvrc12/get_ilsvrc_aux.sh

labels = np.loadtxt(labels_file, str, delimiter='\t')

print 'output label:', labels[output_prob.argmax()]

# sort top five predictions from softmax output
top_inds = output_prob.argsort()[:-1][-5:] # reverse sort and take five largest items

print 'probabilities and labels:'
zip(output_prob[top_inds], labels[top_inds])
```

## 14.3.2 Preparing Resources

### Creating a GPU Cluster and Adding a GPU Node

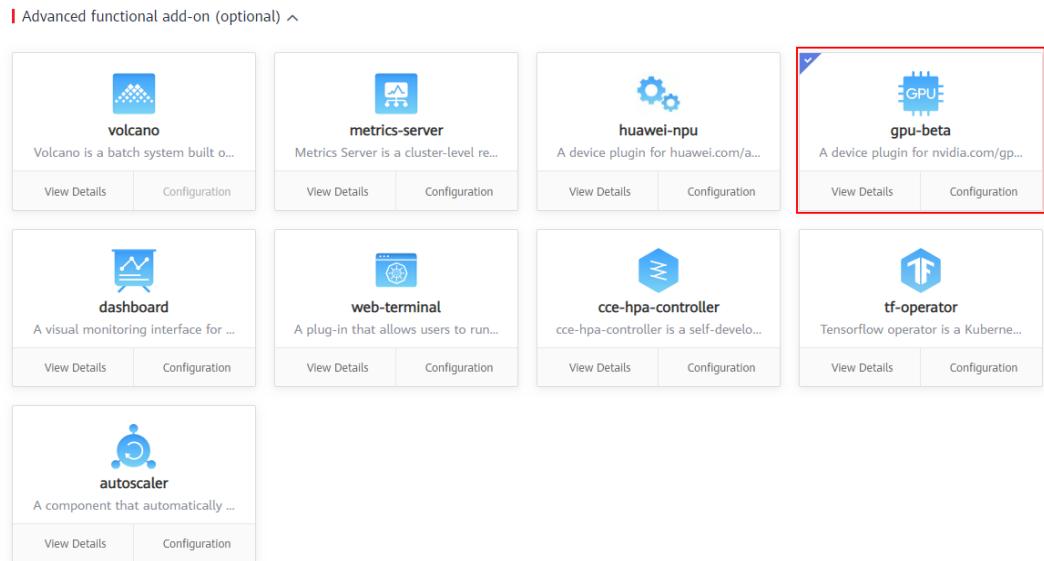
Log in to the CCE console, and click **Buy Cluster**. Then, enter the cluster name, select the cluster version, and click **Next: Create Node**.

Add a GPU node. Currently, the supported GPU node specifications are as follows:

Flavor name	CPU/Memory	Assured/Maximum Bandwidth	PPS
p12.2xlarge.4	8cores   32GB	4/10 Gbit/s	500,000 pps
p12.4xlarge.4	16cores   64GB	8/15 Gbit/s	1,000,000 pps
p12.8xlarge.4	32cores   128GB	15/25 Gbit/s	2,000,000 pps
p2v.2xlarge.8 (Sold Out)	8cores   64GB	4/10 Gbit/s	500,000 pps
p2v.4xlarge.8 (Sold Out)	16cores   128GB	8/15 Gbit/s	1,000,000 pps
p2v.8xlarge.8 (Sold Out)	32cores   256GB	15/25 Gbit/s	2,000,000 pps
p2v.16xlarge.8 (Sold Out)	64cores   512GB	30/30 Gbit/s	4,000,000 pps
g5.8xlarge.4 (Sold Out)	32cores   128GB	15/25 Gbit/s	2,000,000 pps

Current Specifications: GPU-accelerated | p12.2xlarge.4 | 8cores | 32GB | GPU Card: nvidia-t4x1 | AZ2  
To ensure node stability, CCE automatically reserves some resources to run necessary system components. [View Formula for Calculating Node Specifications](#)  
Quota Tip: Remaining CPU cores: 7,945; Remaining RAM (GB): 15,826 Increase quota

Click **Next: Install Add-on** and select the GPU add-on.



Click **Next: Confirm** and then **Pay Now**.

### Importing an OBS Volume

Go to the storage management page and import the OBS volume created in [Pre-configuring OBS Storage Data](#).

### 14.3.3 Caffe Classification Example

This section uses the official Caffe classification example at <https://github.com/BVLC/caffe/blob/master/examples/00-classification.ipynb> to illustrate how to run Caffe jobs on CCE.

### Using CPUs

Create a job using the third-party **image bvlc/caffe:cpu**. Set the container specifications.

The screenshot shows the 'Container Resources' section of the Cloud Container Engine interface. It includes fields for Request and Limit for CPU (2 cores), Memory (5 GiB), and GPU (100%). The GPU section also includes options for Any GPU type or nvidia-t4.

Add the startup command **python /home/caffeEx00.py**.

The screenshot shows the 'Lifecycle' tab in the Cloud Container Engine interface. It includes a 'Start Command' section where 'python /home/caffeEx00.py' is entered. There is also an 'Example' section showing binary and bash examples.

Mount the imported OBS volume.

The screenshot shows the 'Add Cloud Volume' dialog box. It includes fields for Type (OBS), Allocation Mode (Manual), and Sub-Type (Standard). A table lists a single container path entry with permission set to Read/Write.

Click **Create**. After the job execution is complete, go to the **outputimg** directory of the OBS volume to view the image used for inference.

Log in to the node added in [Creating a GPU Cluster and Adding a GPU Node](#) and run the `docker logs {Container ID}` command to view the classification result. The result is displayed as **tabby cat**.

```
I1119 09:42:08.196944    1 upgrade_proto.cpp:44] Attempting to upgrade input file specified using deprecated transformation parameters: /home/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
I1119 09:42:08.197065    1 upgrade_proto.cpp:47] Successfully upgraded file specified using deprecated data transformation parameters.
W1119 09:42:08.197030    1 upgrade_proto.cpp:49] Note that future Caffe releases will only support transform_param messages for transformation fields.
I1119 09:42:08.197032    1 upgrade_proto.cpp:53] Attempting to upgrade input file specified using deprecated V1LayerParameter: /home/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
I1119 09:42:08.432265    1 upgrade_proto.cpp:61] Successfully upgraded file specified using deprecated V1LayerParameter
I1119 09:42:08.494767    1 net.cpp:744] Ignoring source layer loss
/usr/local/lib/python2.7/dist-packages/skimage/transform/_warps.py:84: UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
mean-subtracted values: [('B', 104.0069879317889), ('G', 116.66876761896767), ('R', 122.6789143406786)]
predicted class is: 281
output label: n02123045 tabby, tabby cat
probabilities_and_labels:
```

## Using GPUs

Create a job using the third-party **bvlc/caffe:gpu**. Set the container specifications.

Image: bvlc/caffe | Change Image

\* Image Version: gpu

\* Container Name: container-0

**Container Resources**

- CPU**: Request 2 cores, Limit 2 cores
- Memory**: Request 5 GiB, Limit 5 GiB
- GPU**: Request 100% (Any GPU type selected)
- Ascend 310 Quota**: Use 1

Add the startup command `python /home/caffeEx00_GPU.py`.

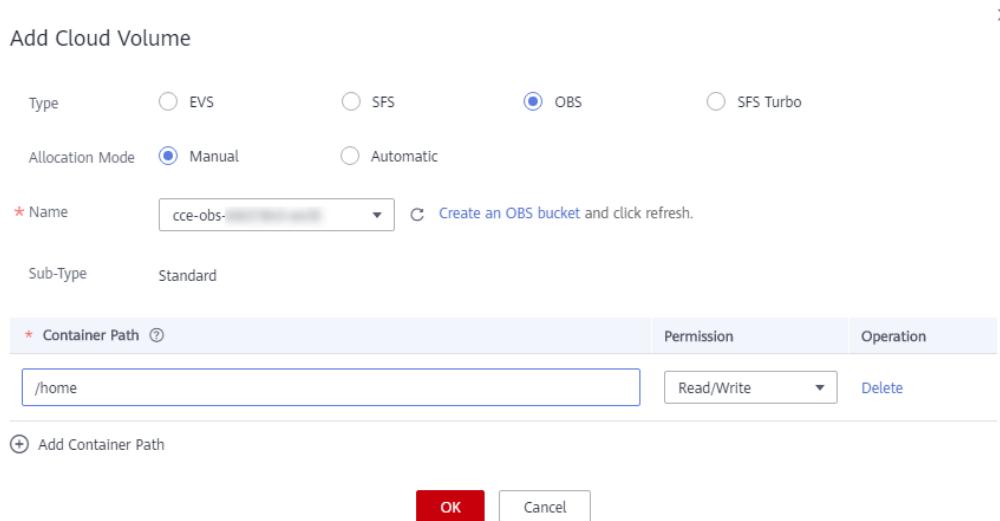
Lifecycle | Environment Variables | Data Storage | Log Policies

Set commands for starting and running containers. [Learn how to set container lifecycle parameters](#) [Learn how to set container start command](#)

Start Command

Command	python /home/caffeEx00_GPU.py	Example	Binary: python /var/tf_mnist/mnist_with_summaries.py
Args	Multi-Args per Input Box	Command	Bash: python --log_dir=/train --learning_rate=0.01 --batch_size=1
Enter the arguments for the command.		Args	50
<input type="button" value="Add"/>			

Mount the imported OBS volume.



Click **Create**. After the job execution is complete, go to the **outputimg** directory of the OBS volume to view the image used for inference.

Log in to the node added in [Creating a GPU Cluster and Adding a GPU Node](#) and run the **docker logs {Container ID}** command to view the classification result. The result is displayed as **tabby cat**.

```
I1119 09:42:08.196944    1 upgrade_proto.cpp:44] Attempting to upgrade input file specified using deprecated transformation parameters: /home/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
I1119 09:42:08.197065    1 upgrade_proto.cpp:47] Successfully upgraded file specified using deprecated data transformation parameters.
I1119 09:42:08.197010    1 upgrade_proto.cpp:49] Note that future Caffe releases will only support transform_param messages for transformation fields.
I1119 09:42:08.197012    1 upgrade_proto.cpp:53] Attempting to upgrade input file specified using deprecated V1LayerParameter: /home/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
I1119 09:42:08.432265    1 upgrade_proto.cpp:61] Successfully upgraded file specified using deprecated V1LayerParameter
I1119 09:42:08.494707    1 net.cpp:744] Ignoring source layer loss
/usr/local/lib/python2.7/dist-packages/skimage/transform/_warps.py:84: UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
mean-subtracted values: [('B', 104.0069879317889), ('G', 116.06876761696767), ('R', 122.6789143406786)]
predicted class is: 281
output label: n02123045 tabby, tabby cat
probabilities_and_labels:
```

## 14.4 Running TensorFlow in CCE

### Preparing Resources

- Create a CCE cluster and GPU nodes, and use the gpu-beta add-on to install the graphics card driver.
- Add an object storage volume to the cluster.

### Pre-configuring Data

Download data from <https://github.com/zalandoresearch/fashion-mnist>.

## Get the Data

Many ML libraries already include Fashion-MNIST data/API, give it a try!

You can use direct links to download the dataset. The data is stored in the same format as the original [MNIST data](#).

Name	Content	Examples	Size	Link	MD5 Checksum
<code>train-images-idx3-ubyte.gz</code>	training set images	60,000	26 MBytes	<a href="#">Download</a>	<code>8d4fb7e6c68d591d4c3dfe9ec88bf0d</code>
<code>train-labels-idx1-ubyte.gz</code>	training set labels	60,000	29 KBytes	<a href="#">Download</a>	<code>25c81989df183df01b3e8a0aad5dffbe</code>
<code>t10k-images-idx3-ubyte.gz</code>	test set images	10,000	4.3 MBytes	<a href="#">Download</a>	<code>bef4ecab320f06d8554ea6380940ec79</code>
<code>t10k-labels-idx1-ubyte.gz</code>	test set labels	10,000	5.1 KBytes	<a href="#">Download</a>	<code>bb300cfdad3c16e7a12a480ee83cd310</code>

Obtain the TensorFlow machine learning (ML) example and modify it based on your requirements.

### basicClass.py

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import gzip
from tensorflow.python.keras.utils import get_file
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt

print(tf.__version__)

#fashion_mnist = keras.datasets.fashion_mnist
#(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

def load_data():
    base = "file:///home/data/"
    files = [
        'train-labels-idx1-ubyte.gz', 'train-images-idx3-ubyte.gz',
        't10k-labels-idx1-ubyte.gz', 't10k-images-idx3-ubyte.gz'
    ]

    paths = []
    for fname in files:
        paths.append(get_file(fname, origin=base + fname))

    with gzip.open(paths[0], 'rb') as lbpath:
        y_train = np.frombuffer(lbpath.read(), np.uint8, offset=8)

    with gzip.open(paths[1], 'rb') as imgpath:
        x_train = np.frombuffer(
            imgpath.read(), np.uint8, offset=16).reshape(len(y_train), 28, 28)

    with gzip.open(paths[2], 'rb') as lbpath:
        y_test = np.frombuffer(lbpath.read(), np.uint8, offset=8)

    with gzip.open(paths[3], 'rb') as imgpath:
        x_test = np.frombuffer(
            imgpath.read(), np.uint8, offset=16).reshape(len(y_test), 28, 28)

    return (x_train, y_train), (x_test, y_test)

(train_images, train_labels), (test_images, test_labels) = load_data()
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.savefig('/home/img/basicimg1.png')

train_images = train_images / 255.0

test_images = test_images / 255.0

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.savefig('/home/img/basicimg2.png')

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5)

test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)

predictions = model.predict(test_images)

def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
   100*np.max(predictions_array),
   class_names[true_label]),
               color=color)

def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array[i], true_label[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)
```

```

thisplot[predicted_label].set_color('red')
thisplot[true_label].set_color('blue')

i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.savefig('/home/img/basicimg3.png')

i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.savefig('/home/img/basicimg4.png')

# Plot the first X test images, their predicted label, and the true label
# Color correct predictions in blue, incorrect predictions in red
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions, test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions, test_labels)
plt.savefig('/home/img/basicimg5.png')

```

Go to the OBS bucket page, create the **data** and **img** folders, and upload **basicClass.py**.

The screenshot shows the OBS Bucket Objects page. At the top, there are tabs for 'Objects' (which is selected), 'Deleted Objects', and 'Fragments'. Below the tabs are buttons for 'Upload Object', 'Create Folder', 'Restore', 'Delete', and 'Change Storage Class'. A table lists the uploaded objects:

Object Name	Storage Class	Size	Encrypted
img	--	--	--
data	--	--	--
basicClass.py	Standard	[redacted]	No

Go to the **data** folder and upload the four .gz files downloaded from GitHub.

## ML Example

In this section, the ML example from the TensorFlow official website is used. For details, see <https://www.tensorflow.org/tutorials/keras/classification?hl=en-us>.

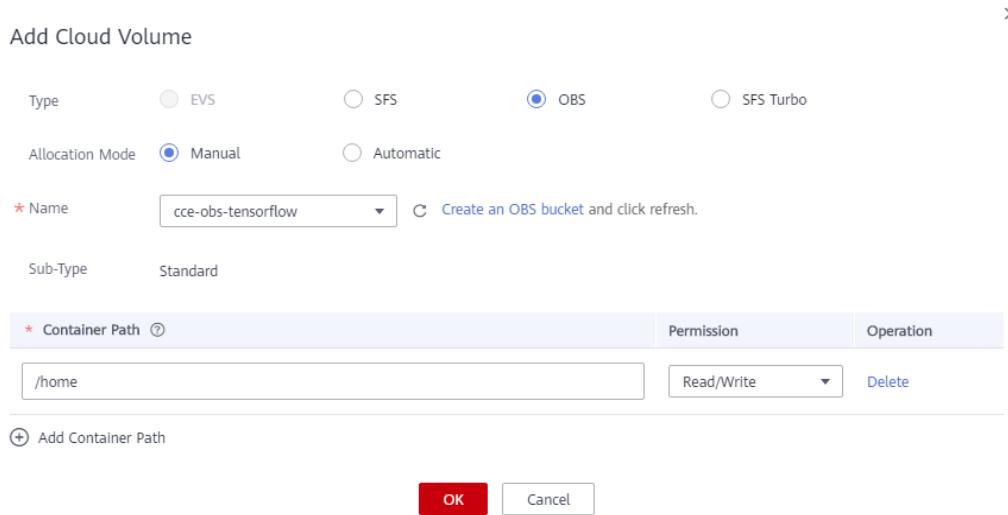
Create a job using the third-party **tensorflow/tensorflow:1.15.5-gpu**. Set the container specifications.

The screenshot shows the configuration page for a new container. At the top, the image name is set to "tensorflow" and the image version is "1.15.5-gpu". The container name is "container-0". A "Privileged Container" toggle switch is turned on, with a note explaining it gives access to all host devices. In the "Container Resources" section, under "CPU", the "Request" field is set to 2 cores, with a note that a container is guaranteed to have as much CPU/memory as it requests but cannot exceed its limit. Under "Memory", both "Request" and "Limit" fields are set to 4 MiB, with a note that exceeding the limit will terminate the container. Under "GPU", the "Use" field is set to 100%, with a note about reserving GPU resources. In the "GPU/Graphics Card" section, the "nvidia-p4" type is selected. The "Start Command" area is visible at the bottom.

Add `pip install matplotlib;python /home/basicClass.py` in the **Start Command** area.

The screenshot shows the "Lifecycle" tab with the "Start Command" section expanded. The "Command" field contains "/bin/bash". The "Args" field contains "pip install matplotlib;python /home/basicClass.py". To the right, there is an "Example" section with "Command" and "Args" fields, which are currently empty. A "Delete" button is also visible next to the Args input field.

Mount the created OBS volume.



Click **Create**. Wait until the job execution is complete. On the OBS page, you can view the execution results that are shown as images.

Objects Deleted Objects Fragments

Objects are basic units of data storage. In OBS, files and folders are treated as objects. Any file type can be uploaded and managed in a bucket. [Learn more](#)  
You can use [OBS Browser+](#) to move an object to any other folder in this bucket.

<input type="checkbox"/>	Name	Storage Class	Size	Encrypted
<input type="checkbox"/>	basicimg4.png	Standard	296.18 KB	No
<input type="checkbox"/>	basicimg2.png	Standard	275.08 KB	No
<input type="checkbox"/>	basicimg1.png	Standard	191.60 KB	No
<input type="checkbox"/>	basicimg3.png	Standard	113.71 KB	No
<input type="checkbox"/>	basicimg5.png	Standard	11.23 KB	No

[Back](#)

If you want to use kubectl, you can use the following example YAML:

```
kind: Job
apiVersion: batch/v1
metadata:
  name: testjob
  namespace: default
spec:
  parallelism: 1
  completions: 1
  backoffLimit: 6
  template:
    metadata:
      name: testjob
    spec:
      volumes:
        - name: cce-obs-tensorflow
      persistentVolumeClaim:
```

```
claimName: cce-obs-tensorflow
containers:
- name: container-0
  image: 'tensorflow/tensorflow:1.15.5-gpu'
  restartPolicy: OnFailure
  command:
    - /bin/bash
  args:
    - '-c'
    - pip install matplotlib;python /home/basicClass.py
resources:
limits:
  cpu: '2'
  memory: 4Gi
  nvidia.com/gpu: '1'
requests:
  cpu: '2'
  memory: 4Gi
  nvidia.com/gpu: '1'
volumeMounts:
- name: cce-obs-tensorflow
  mountPath: /home
imagePullPolicy: IfNotPresent
imagePullSecrets:
- name: default-secret
```

## 14.5 Running Flink in CCE

This section describes how to deploy a Flink cluster on CCE and run WordCount jobs.

### Prerequisites

A CCE cluster with available nodes has been created. An elastic IP address has been bound to the nodes, and kubectl has been configured.

### Deployment Process

For details about the process, see <https://ci.apache.org/projects/flink/flink-docs-stable/ops/deployment/kubernetes.html>.

### Creating a Flink Session Cluster

Create two Deployments, one Service, and one ConfigMap by following the instructions provided on the preceding web page.

flink-configuration-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: flink-config
  labels:
    app: flink
data:
  flink-conf.yaml: |+
    jobmanager.rpc.address: flink-jobmanager
    taskmanager.numberOfTaskSlots: 2
    blob.server.port: 6124
    jobmanager.rpc.port: 6123
    taskmanager.rpc.port: 6122
    queryable-state.proxy.ports: 6125
```

```

jobmanager.memory.process.size: 1600m
taskmanager.memory.process.size: 1728m
parallelism.default: 2
log4j-console.properties: |+
# This affects logging for both user code and Flink
rootLogger.level = INFO
rootLogger.appenderRef.console.ref = ConsoleAppender
rootLogger.appenderRef.rolling.ref = RollingFileAppender

# Uncomment this if you want to _only_ change Flink's logging
#logger.flink.name = org.apache.flink
#logger.flink.level = INFO

# The following lines keep the log level of common libraries/connectors on
# log level INFO. The root logger does not override this. You have to manually
# change the log levels here.
logger.akka.name = akka
logger.akka.level = INFO
logger.kafka.name= org.apache.kafka
logger.kafka.level = INFO
logger.hadoop.name = org.apache.hadoop
logger.hadoop.level = INFO
logger.zookeeper.name = org.apache.zookeeper
logger.zookeeper.level = INFO

# Log all infos to the console
appender.console.name = ConsoleAppender
appender.console.type = CONSOLE
appender.console.layout.type = PatternLayout
appender.console.layout.pattern = %d{yyyy-MM-dd HH:mm:ss,SSS} %-5p %-60c %x - %m%n

# Log all infos in the given rolling file
appender.rolling.name = RollingFileAppender
appender.rolling.type = RollingFile
appender.rolling.append = false
appender.rolling.fileName = ${sys:log.file}
appender.rolling.filePattern = ${sys:log.file}.%i
appender.rolling.layout.type = PatternLayout
appender.rolling.layout.pattern = %d{yyyy-MM-dd HH:mm:ss,SSS} %-5p %-60c %x - %m%n
appender.rolling.policies.type = Policies
appender.rolling.policies.size.type = SizeBasedTriggeringPolicy
appender.rolling.policies.size.size=100MB
appender.rolling.strategy.type = DefaultRolloverStrategy
appender.rolling.strategy.max = 10

# Suppress the irrelevant (wrong) warnings from the Netty channel handler
logger.netty.name = org.apache.flink.shaded.akka.org.jboss.netty.channel.DefaultChannelPipeline
logger.netty.level = OFF

```

### jobmanager-service.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: flink-jobmanager
spec:
  type: ClusterIP
  ports:
    - name: rpc
      port: 6123
    - name: blob-server
      port: 6124
    - name: webui
      port: 8081
  selector:
    app: flink
    component: jobmanager

```

### jobmanager-session-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flink-jobmanager
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flink
      component: jobmanager
  template:
    metadata:
      labels:
        app: flink
        component: jobmanager
    spec:
      containers:
        - name: jobmanager
          image: flink:1.11.0-scala_2.11
          args: ["jobmanager"]
          ports:
            - containerPort: 6123
              name: rpc
            - containerPort: 6124
              name: blob-server
            - containerPort: 8081
              name: webui
      livenessProbe:
        tcpSocket:
          port: 6123
      initialDelaySeconds: 30
      periodSeconds: 60
    volumeMounts:
      - name: flink-config-volume
        mountPath: /opt/flink/conf
    securityContext:
      runAsUser: 9999 # refers to user _flink_ from official flink image, change if necessary
  volumes:
    - name: flink-config-volume
      configMap:
        name: flink-config
        items:
          - key: flink-conf.yaml
            path: flink-conf.yaml
          - key: log4j-console.properties
            path: log4j-console.properties
```

#### taskmanager-session-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flink-taskmanager
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flink
      component: taskmanager
  template:
    metadata:
      labels:
        app: flink
        component: taskmanager
    spec:
      containers:
        - name: taskmanager
          image: flink:1.11.0-scala_2.11
          args: ["taskmanager"]
          ports:
```

```

- containerPort: 6122
  name: rpc
- containerPort: 6125
  name: query-state
livenessProbe:
  tcpSocket:
    port: 6122
initialDelaySeconds: 30
periodSeconds: 60
volumeMounts:
- name: flink-config-volume
  mountPath: /opt/flink/conf/
securityContext:
  runAsUser: 9999 # refers to user _flink_ from official flink image, change if necessary
volumes:
- name: flink-config-volume
configMap:
  name: flink-config
  items:
- key: flink-conf.yaml
  path: flink-conf.yaml
- key: log4j-console.properties
  path: log4j-console.properties

```

**kubectl create -f flink-configuration-configmap.yaml**

**kubectl create -f jobmanager-service.yaml**

**kubectl create -f jobmanager-session-deployment.yaml**

**kubectl create -f taskmanager-session-deployment.yaml**

```
[root@flink-config ~]# kubectl get cm |grep flink
2      17m
```

```
[root@flink-jobmanager ~]# kubectl get svc |grep flink
ClusterIP 10.247.4.151 <none> 6123/TCP,6124/TCP,8081/TCP 20m
```

```
[root@flink-jobmanager ~]# kubectl get pod -owide|grep flink
flink-jobmanager-b8545cf98-rnf5m 1/1 Running 0 15m 172.16.1.143 192.168.15.231 <none> <none>
flink-taskmanager-7674f748bd-h6z7w 1/1 Running 0 15m 172.16.1.144 192.168.15.231 <none> <none>
flink-taskmanager-7674f748bd-v6tsx 1/1 Running 0 15m 172.16.1.145 192.168.15.231 <none> <none>
```

## Exposing the Service

Log in to the CCE console. Choose **Workloads > Deployments**, click **flink-jobmanager**, and click the **Services** tab.

Workload Name	Type	Deployment	
[REDACTED]	Cluster	test	
Status			
Pods (Ready/All)	0/1	Namespace	default
Created	Dec 15, 2020 20:19:09 GMT+08:00	Access Address	<a href="#">View Access Mode</a>
Upgrade Mode	Rolling Upgrade	Labels	<a href="#">Manage Label</a>
Description	.. ↗		

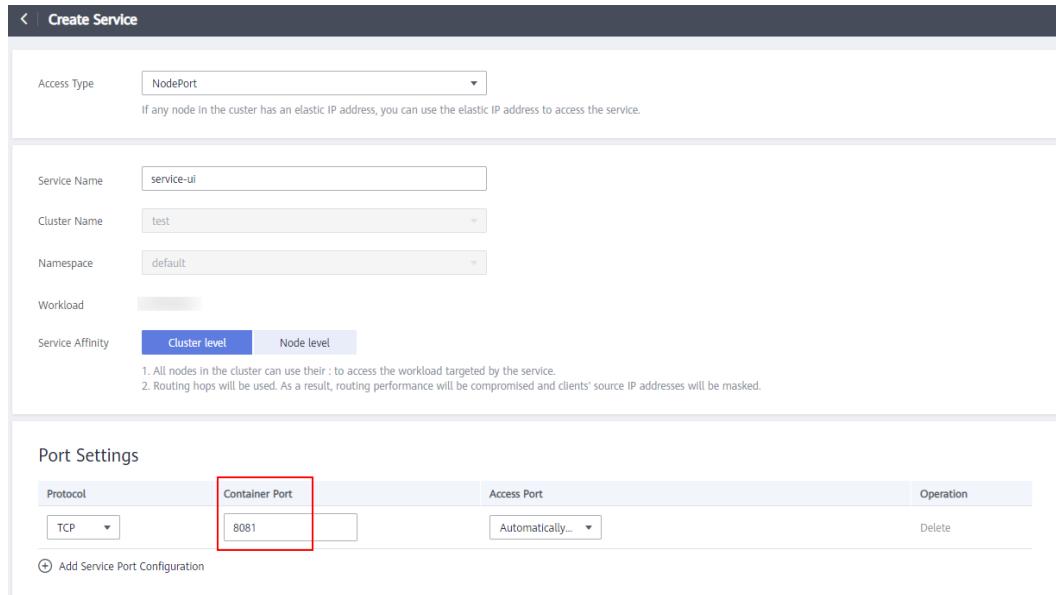
Pods | Monitoring | **Services** | Upgrade | Scaling | Scheduling Policies | Workload O&M | Events

A service defines a logical set of pods and a policy by which to access them. Go to Resource Management > Network > Services to view all services.

**Create Service**

Domain Name for Intra-Cluster Access	Access Address	Access Mode
--------------------------------------	----------------	-------------

Click **Create Service**, select **NodePort** for **Access Type**, and set **Container Port** to **8081**.



Check whether the Flink can be accessed by using the access address of the Service.

Service Name	Internal Domain Name	Workload	Access Address	Access Type	Access Port -> Container Port / Protocol
[redacted]	[redacted]	[redacted]	[redacted] (EIP) (Private)	LoadBalancer (ELB)	80 -> http / TCP [redacted]
[redacted]	[redacted]	[redacted]	121.36.30.220 (EIP) (Private)	NodePort	31687 -> 80 / TCP

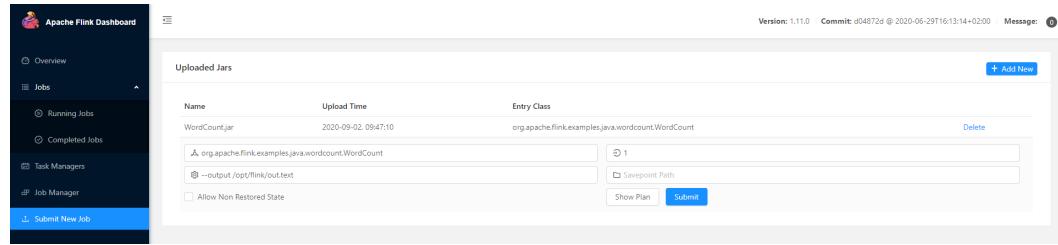
The Apache Flink Dashboard page is displayed.

## Running the Flink Job

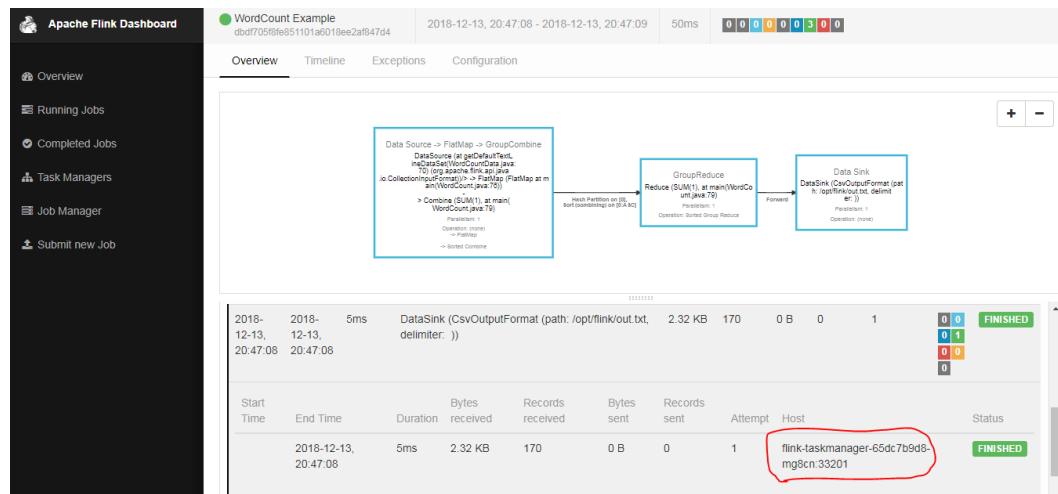
Use the official file **WordCount.jar** to run the Flink job.

Download [https://archive.apache.org/dist/flink/flink-1.11.0/flink-1.11.0-bin-scala\\_2.11.tgz](https://archive.apache.org/dist/flink/flink-1.11.0/flink-1.11.0-bin-scala_2.11.tgz) and decompress it. The **WordCount.jar** package is generated in **examples\streaming**.

Upload **WordCount.jar**, and set the parameters as follows:



Run the job.



After the job execution is complete, check the execution result. Access the specified task manager and check whether the count of each word is correctly recorded in the **/opt/flink/out** file.

```
[root@... ~]# kubectl exec -ti flink-taskmanager-65dc7b9d8-mg8cn bash
root@flink-taskmanager-65dc7b9d8-mg8cn:/opt/flink#
root@flink-taskmanager-65dc7b9d8-mg8cn:/opt/flink# ls
LICENSE NOTICE README.txt bin conf examples lib log opt out out.txt
root@flink-taskmanager-65dc7b9d8-mg8cn:/opt/flink#
root@flink-taskmanager-65dc7b9d8-mg8cn:/opt/flink#
root@flink-taskmanager-65dc7b9d8-mg8cn:/opt/flink# cat out.txt
a 5
action 1
after 1
against 1
all 2
and 12
arms 1
around 1
```

## 14.6 Deploying ClickHouse on CCE

### 14.6.1 Creating a CCE Cluster

ClickHouse Operator creates, configures, and manages ClickHouse clusters running on Kubernetes.

This section describes how to install and deploy ClickHouse Operator on CCE clusters and provides examples of creating ClickHouse cluster resources. For details, see <https://github.com/Altinity/clickhouse-operator>.

ClickHouse Operator can be installed in CCE clusters of v1.15.11 and later. In this example, ClickHouse Operator is installed in a cluster of v1.19.

<b>Cluster type</b>	CCE cluster
<b>Cluster version</b>	1.19
<b>Region</b>	CN East-Shanghai1
<b>Docker version</b>	18.09.0.91
<b>Network model</b>	VPC network
<b>Service forwarding mode</b>	iptables

## 14.6.2 Configuring kubectl

Log in to the CCE console, choose **Resource Management > Clusters**, and click the cluster name. The cluster details page is displayed. Click the **Kubectl** tab.

Configure the kubectl tool as prompted.

The screenshot shows the CCE Cluster Details page with the Kubectl tab selected. It includes the following content:

- Events** | **Auto Scaling** | **Kubectl** (selected) | **Istioctl**
- Use the kubectl tool to perform operations on the cluster.**
- Download the [kubectl](#) tool and configuration file and copy them to your client. They can be used after configuration. Use kubectl to access the cluster.
- Download the kubectl tool.**  
Go to [Kubernetes release page](#) to download kubectl corresponding to the cluster version or of a later version.
- Download the kubectl configuration file.**  
Click [here](#) to download the configuration file. (If the public API server address is changed, download the file again.)
- Install and set up kubectl.**  
The following procedure uses an Nginx application as an example. For more information, see [Install and Set Up kubectl](#).
- 1. Copy [kubectl](#) and its configuration file to the /home directory on your client.  
2. Log in to your client and configure [kubectl](#).  
1. cd /home  
2. chmod +x kubectl  
3. mv -f kubectl /usr/local/bin  
4. mkdir -p \$HOME/.kube  
5. mv -f kubeconfig.json \$HOME/.kube/config  
Copy Command

## 14.6.3 Deploying ClickHouse Operator

`kubectl apply -f https://github.com/Altinity/clickhouse-operator/blob/master/deploy/operator/clickhouse-operator-install-bundle.yaml`

```
[root@click-house-49066 ~]# kubectl apply -f https://raw.githubusercontent.com/Altinity/clickhouse-operator/master/deploy/operator/clickhouse-operator-install.yaml
customresourcedefinition.apiextensions.k8s.io/clickhouseinstallations.clickhouse.altinity.com created
customresourcedefinition.apiextensions.k8s.io/clickhouseinstallationstemplates.clickhouse.altinity.com created
customresourcedefinition.apiextensions.k8s.io/clickhouseoperatorconfigurations.clickhouse.altinity.com created
serviceaccount/clickhouse-operator created
clusterrolebinding/clickhouse-operator created
configmap/etc-clickhouse-operator-kube-system created
configmap/etc-clickhouse-operator-kube-system created
configmap/etc-clickhouse-operator-confd-files created
configmap/etc-clickhouse-operator-confd-files created
configmap/etc-clickhouse-operator-templated-files created
configmap/etc-clickhouse-operator-usersd-files created
deployment.apps/clickhouse-operator created
service/clickhouse-operator-metrics created
```

After a period of time, check the running status of ClickHouse Operator.

**kubectl get pod -n kube-system|grep clickhouse**

```
[root@click-house-49966 ~]# kubectl get pod -n kube-system|grep clickhouse
clickhouse-operator-5d56f7b4b7-zmp98      2/2     Running   0          16m
```

## 14.6.4 Examples

### Creating a Namespace

Create a namespace named **test-clickhouse-operator** to facilitate verification of basic functions.

**kubectl create namespace test-clickhouse-operator**

```
[root@click-house-49966 ~]# kubectl create namespace test-clickhouse-operator
namespace/test-clickhouse-operator created
```

### Simple Example

This example is available at <https://github.com/Altinity/clickhouse-operator/blob/master/docs/chi-examples/01-simple-layout-01-1shard-1repl.yaml>.

The YAML file is as follows:

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "simple-01"
```

Run the following command:

**kubectl apply -n test-clickhouse-operator -f https://raw.githubusercontent.com/Altinity/clickhouse-operator/master/docs/chi-examples/01-simple-layout-01-1shard-1repl.yaml**

```
[root@click-house-49966 ~]# kubectl apply -n test-clickhouse-operator -f https://raw.githubusercontent.com/Altinity/clickhouse-operator/master/docs/chi-examples/01-simple-layout-01-1shard-1repl.yaml
clickhouseinstallation.clickhouse.altinity.com/simple-01 created
```

After a period of time, check the resource running status.

**kubectl get pod -n test-clickhouse-operator**

**kubectl get service -n test-clickhouse-operator**

```
[root@click-house-49966 ~]# kubectl get pod -n test-clickhouse-operator
NAME           READY   STATUS    RESTARTS   AGE
chi-simple-01-cluster-0-0-0   1/1     Running   0          2m27s
[root@click-house-49966 ~]# kubectl get service -n test-clickhouse-operator
NAME         TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
chi-simple-01-cluster-0-0   ClusterIP  <none>       <none>        8123/TCP,9000/TCP,9009/TCP   2m3s
clickhouse-simple-01       LoadBalancer  10.247.1.133  <pending>     8123:30485/TCP,9000:30301/TCP   2m55s
```

Connect to the ClickHouse database.

**kubectl -n test-clickhouse-operator exec -ti chi-simple-01-cluster-0-0-0 -- clickhouse-client**

```
[root@click-house-49966 ~]# kubectl -n test-clickhouse-operator exec -ti chi-simple-01-cluster-0-0-0 -- clickhouse-client
ClickHouse client version 20.8.2.3 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 20.8.2 revision 54438.

chi-simple-01-cluster-0-0-0.chi-simple-01-cluster-0-0.test-clickhouse-operator.svc.cluster.local :)
```

## Simple Persistent Volume Example

This example is available at <https://github.com/Altinity/clickhouse-operator/blob/master/docs/chi-examples/03-persistent-volume-01-default-volume.yaml>.

Before using this YAML file to create a PVC on CCE, modify the file based on the storage volume you want to use.

- If an EVS disk is used as a storage volume, do as follows:

- Create a StorageClass.

By default, the CSI disk type supported by CCE is SAS. If you want to use ultra-high I/O EVS disks, you need to create the corresponding StorageClass.

**vim csi disk ssd.yaml**

Copy the following content:

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk-ssd
parameters:
  csi.storage.k8s.io/csi-driver-name: disk.csi.everest.io
  csi.storage.k8s.io/fstype: ext4
  everest.io/disk-volume-type: SSD
  everest.io/passthrough: "true"
provisioner: everest-csi-provisioner
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

Save the file and exit.

**kubectl create -f csi-disk-ssd.yaml**

- Set **accessModes** to **ReadWriteOnce**.
  - Add **storageClassName: csi-disk-ssd**.
- If an SFS file system is used as a storage volume, do as follows:
    - Set **accessModes** to **ReadWriteMany**.
    - Add **storageClassName: csi-nas**.

For example, if an SFS file system is used, the YAML file content is as follows:

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "pv-simple"
spec:
  defaults:
    templates:
      dataVolumeClaimTemplate: data-volume-template
      logVolumeClaimTemplate: log-volume-template
  configuration:
    clusters:
      - name: "simple"
        layout:
          shardsCount: 1
          replicasCount: 1
    templates:
      volumeClaimTemplates:
        - name: data-volume-template
          spec:
            accessModes:
```

```

- ReadWriteMany
resources:
  requests:
    storage: 10Gi
storageClassName: csi-nas
- name: log-volume-template
spec:
  accessModes:
    - ReadWriteMany
resources:
  requests:
    storage: 10Gi
storageClassName: csi-nas

```

Run the following command to create a PV:

**kubectl -n test-clickhouse-operator create -f 03-persistent-volume-01-default-volume.yaml**

```
[root@click-house-49966 ~]# kubectl -n test-clickhouse-operator create -f 03-persistent-volume-01-default-volume.yaml
clickhouseinstallation.clickhouse.altinity.com/pv-simple created
```

After a period of time, check the resource running status.

**kubectl get pvc -n test-clickhouse-operator**

```
[root@click-house-49966 ~]# kubectl get pvc -n test-clickhouse-operator
NAME                      STATUS   VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS  AGE
data-volume-template-chi-pv-simple-simple-0-0-0  Bound   pvc-b3ff9d71-e66a-4d19-96f4-cc92ed69a023  10Gi   RWX        csi-nas       28s
log-volume-template-chi-pv-simple-simple-0-0-0  Bound   pvc-a227b285-e955-414c-bb7f-89b7400bb0fa  10Gi   RWX        csi-nas       28s
```

**kubectl get pod -n test-clickhouse-operator**

```
[root@click-house-49966 ~]# kubectl get pod -n test-clickhouse-operator
NAME                  READY   STATUS    RESTARTS   AGE
chi-pv-simple-simple-0-0-0  2/2    Running   0          4m10s
chi-simple-01-cluster-0-0-0 1/1    Running   0          4h11m
```

Run the following command to check the mounting status of the storage volume:

**kubectl -n test-clickhouse-operator exec -ti chi-pv-simple-simple-0-0-0 -c clickhouse bash**

**df -h**

```
[root@click-house-49966 ~]# kubectl -n test-clickhouse-operator exec -ti chi-pv-simple-simple-0-0-0 -c clickhouse bash
root@chi-pv-simple-simple-0-0-0:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           64M   64M   0  100% /dev
tmpfs           7.8G  7.8G  0  100% /sys/fs/cgroup
tmpfs           9.3G  9.3G  0  100% /dev/mapper/vgpaas-kubernetc
tmpfs           18G  18G  0  100% /dev/mapper/vgpaas-dockersys
tmpfs           64M   64M   0  100% /dev/shm
/dev/nas01-cn-east-3a.myhuaweicloud.com/:share-89586dca  10G   0  10G  0% /var/lib/clickhouse
/dev/nas01-cn-east-3a.myhuaweicloud.com/:share-5bc94443  7.8G  7.8G  0  100% /var/lib/clickhouse-server
tmpfs           7.8G  7.8G  0  100% /proc/acpi
tmpfs           7.8G  7.8G  0  100% /proc/scsi
tmpfs           7.8G  7.8G  0  100% /sys/firmware
```

Connect to the ClickHouse database.

**kubectl -n test-clickhouse-operator exec -ti chi-pv-simple-simple-0-0-0 --clickhouse-client**

```
[root@click-house-49966 ~]# kubectl -n test-clickhouse-operator exec -ti chi-pv-simple-simple-0-0-0 --clickhouse-client
Defaulting container name to clickhouse.
Use 'kubectl describe pod/chi-pv-simple-simple-0-0-0 -n test-clickhouse-operator' to see all of the containers in this pod.
ClickHouse client version 20.8.2.3 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 20.8.2 revision 54438.
chi-pv-simple-simple-0-0-0.chi-pv-simple-simple-0-0-0.test-clickhouse-operator.svc.cluster.local :)
```

## Simple Load Balancer Example

```

apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "ck-elb"
spec:
  defaults:
    templates:
      dataVolumeClaimTemplate: data-volume-nas
      serviceTemplate: chi-service-elb

```

```

configuration:
clusters:
- name: "ck-elb"
  templates:
    podTemplate: pod-template-with-nas
  layout:
    shardsCount: 1
    replicasCount: 1
templates:
  podTemplates:
    - name: pod-template-with-nas
      spec:
        containers:
          - name: clickhouse
            image: yandex/clickhouse-server:21.6.3.14
            volumeMounts:
              - name: data-volume-nas
                mountPath: /var/lib/clickhouse
  volumeClaimTemplates:
    - name: data-volume-nas
      spec:
        accessModes:
          - ReadWriteMany
      resources:
        requests:
          storage: 20Gi
      storageClassName: csi-nas
  serviceTemplates:
    - name: chi-service-elb
      metadata:
        annotations:
          kubernetes.io/elb.class: union
          kubernetes.io/elb.autocreate: >-
            {"type":"public","bandwidth_name":"cce-bandwidth-
ck","bandwidth_chargemode":"bandwidth","bandwidth_size":5,"bandwidth_sharetype":"PER","eip_type":"5_bgp"}
      spec:
        ports:
          - name: http
            port: 8123
          - name: client
            port: 9000
      type: LoadBalancer

```

Add the information in bold to the YAML file. The following table describes the parameters supported by **annotations kubernetes.io/elb.autocreate**.

Parameter	Type	Description
name	String	Name of the automatically created load balancer.  Value range: a string of 1 to 64 characters, including lowercase letters, digits, and underscores (_). The value must start with a lowercase letter and end with a lowercase letter or digit.
type	String	Network type of the load balancer. <ul style="list-style-type: none"> <li>• <b>public</b>: public network load balancer</li> <li>• <b>inner</b>: private network load balancer</li> </ul>

Parameter	Type	Description
bandwidth_name	String	Bandwidth name. The default value is <b>cce-bandwidth-*****</b> . Value range: a string of 1 to 64 characters, including lowercase letters, digits, and underscores (_). The value must start with a lowercase letter and end with a lowercase letter or digit.
bandwidth_charge_mode	String	Bandwidth billing mode. <ul style="list-style-type: none"><li>• <b>bandwidth</b>: billed by bandwidth</li><li>• <b>traffic</b>: billed by traffic</li></ul>
bandwidth_size	Integer	Bandwidth.
bandwidth_share_type	String	Bandwidth sharing mode. <ul style="list-style-type: none"><li>• <b>PER</b>: dedicated bandwidth</li></ul>
eip_type	String	EIP type.

## 14.7 Running Spark on CCE

### 14.7.1 Installing Spark

#### Prerequisites

Prepare a Linux host that can access the public network and configure JDK 1.8 or later in the environment.

#### Obtaining the SDK Package

OBS matches Hadoop 2.8.3 and 3.1.1. Hadoop 3.1.1 is used in this example.

```
git clone -b v3.1.1 https://github.com/apache/spark.git
```

Modify the `/dev/make-distribution.sh` file. Search for the lines where **VERSION** is located (line 129 to line 147 in this case). Comment out all the lines from line 129 to the line containing **echo -n**) and add the following four lines so that the check can be skipped during compilation:

```
VERSION=3.1.3
SCALA_VERSION=2.12
SPARK_HADOOP_VERSION=3.1.1
SPARK_HIVE=1
```

Run the following command to compile the `make-distribution.sh` file:

```
./dev/make-distribution.sh --name hadoop3.1 --tgz -Pkubernetes -Pyarn -
Dhadoop.version=3.1.1
```

The `spark-3.1.3-bin-hadoop3.1.tgz` file is obtained.

## Obtaining the Huawei Cloud OBS JAR Package

The **hadoop-huaweicloud-3.1.1-hw-45.jar** package is used, which can be obtained from <https://github.com/huaweicloud/obsa-hdfs/tree/master/release>.

## Configuring Spark Running Environment

To simplify the operation, use the **root** user to place the compiled package **spark-3.1.3-bin-hadoop3.1.tgz** in the **/root** directory on the operation node.

Run the following command to install Spark:

```
tar -xvzf spark-3.1.3-bin-hadoop3.1.tgz
mv spark-3.1.3-bin-hadoop3.1 spark-obs
cat >> ~/bashrc <<EOF
PATH=/root/spark-obs/bin:$PATH
PATH=/root/spark-obs/sbin:$PATH
export SPARK_HOME=/root/spark-obs
EOF

source ~/bashrc
```

At this time, the **spark-submit** script is available. You can run the **spark-submit --version** command to check the Spark version.

## Interconnecting Spark with OBS

1. Copy the Huawei Cloud OBS JAR package to the corresponding directory.  
**cp hadoop-huaweicloud-3.1.1-hw-45.jar /root/spark-obs/jars/**
2. Modify Spark ConfigMaps.

To interconnect Spark with OBS, add ConfigMaps for Spark as follows:

Change the values of **AK\_OF\_YOUR\_ACCOUNT**, **SK\_OF\_YOUR\_ACCOUNT**, and **OBS\_ENDPOINT** to the actual values.

```
cp ~/spark-obs/conf/spark-defaults.conf.template ~/spark-obs/conf/spark-defaults.conf

cat >> ~/spark-obs/conf/spark-defaults.conf <<EOF
spark.hadoop.fs.obs.readahead.inputstream.enabled=true
spark.hadoop.fs.obs.buffer.max.range=6291456
spark.hadoop.fs.obs.buffer.part.size=2097152
spark.hadoop.fs.obs.threads.read.core=500
spark.hadoop.fs.obs.threads.read.max=1000
spark.hadoop.fs.obs.write.buffer.size=8192
spark.hadoop.fs.obs.read.buffer.size=8192
spark.hadoop.fs.obs.connection.maximum=1000
spark.hadoop.fs.obs.access.key=AK_OF_YOUR_ACCOUNT
spark.hadoop.fs.obs.secret.key=SK_OF_YOUR_ACCOUNT
spark.hadoop.fs.obs.endpoint=OBS_ENDPOINT
spark.hadoop.fs.obs.buffer.dir=/root/hadoop-obs/obs-cache
spark.hadoop.fs.obs.impl=org.apache.hadoop.fs.obs.OBSFileSystem
spark.hadoop.fs.obs.connection.ssl.enabled=false
spark.hadoop.fs.obs.fast.upload=true
spark.hadoop.fs.obs.socket.send.buffer=65536
spark.hadoop.fs.obs.socket.recv.buffer=65536
spark.hadoop.fs.obs.max.total.tasks=20
spark.hadoop.fs.obs.threads.max=20
spark.kubernetes.container.image.pullSecrets=default-secret
EOF
```

## Pushing an Image to SWR

Running Spark on Kubernetes requires the Spark image of the same version. A Dockerfile file has been generated during compilation. You can use this file to create an image and push it to SWR.

1. Create an image.

```
cd ~/spark-obs
docker build -t spark:3.1.3-obs --build-arg spark_uid=0 -f kubernetes/
dockerfiles/spark/Dockerfile .
```

2. Push the image.

Log in to the SWR console and obtain the login command.

Log in to the node where the image is created and run the login command.

```
docker tag [{Image name}:{Tag}] swr.ap-
southeast-1.myhuaweicloud.com/{Organization name}/{Image name}:
{Tag}
docker push swr.ap-southeast-1.myhuaweicloud.com/{Organization
name}/{Image name}:{Tag}
```

Record the image access address for later use.

For example, record the IP address as **swr.ap-**  
**southeast-1.myhuaweicloud.com/dev-container/spark:3.1.3-obs.**

## Configuring Spark History Server

```
cat >> ~/spark-obs/conf/spark-defaults.conf <<EOF
spark.eventLog.enabled=true
spark.eventLog.dir=obs://*****
EOF
```

Ensure that the bucket name and directory in the preceding command are valid.

For example, **obs://spark-sh1/history-obs/** is a valid OBS directory.

Modify the **~/spark-obs/conf/spark-env.sh** file. If the file does not exist, run the command to copy the template as a file:

```
cp ~/spark-obs/conf/spark-env.sh.template ~/spark-obs/conf/spark-env.sh

cat >> ~/spark-obs/conf/spark-env.sh <<EOF
SPARK_HISTORY_OPTS="-Dspark.history.fs.logDirectory=obs://*****"
EOF
```

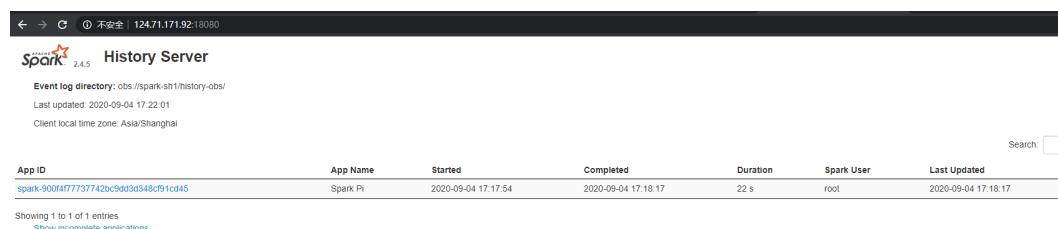
The OBS directory must be the same as that in **spark-default.conf**.

### start-history-server.sh

Start Spark History Server.

```
[root@... ~]# start-history-server.sh
starting org.apache.spark.deploy.history.HistoryServer, logging to /root/spark-obs/logs/spark-root-org.apache.spark.deploy.history.HistoryServer-1-mogujie-spark-zxx-03304.out
```

After the startup, you can access the server over port 18080.



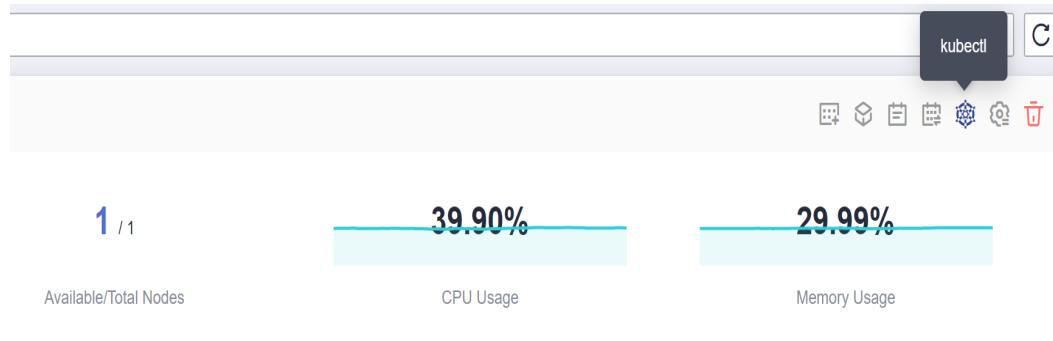
App ID	App Name	Started	Completed	Duration	Spark User	Last Updated
spark-9004f77737742bc9dd3d348cf91d45	Spark Pi	2020-09-04 17:17:54	2020-09-04 17:18:17	22 s	root	2020-09-04 17:18:17

## 14.7.2 Using Spark on CCE

### Running SparkPi on CCE

Install kubectl on the machine where Spark is executed. Log in to the CCE console.

Choose **Clusters** in the navigation pane and click  on the right.



Connect to the cluster as prompted.

After kubectl is installed, run the following commands to create Spark:

```
kubectl create serviceaccount spark
```

```
kubectl create clusterrolebinding spark-role --clusterrole=edit --serviceaccount=default:spark --namespace=default
```

The following describes how to submit a Spark-Pi job to CCE.

```
spark-submit \
--master k8s://https://aa.bb.cc.dd:5443 \
--deploy-mode cluster \
--name spark-pi \
--class org.apache.spark.examples.SparkPi \
--conf spark.executor.instances=2 \
--conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
--conf spark.kubernetes.container.image=swr.ap-southeast-1.myhuaweicloud.com/dev-container/spark:3.1.3-obs \
local:///opt/spark/examples/jars/spark-examples_2.12-3.1.1.jar
```

Configuration description:

1. **aa.bb.cc.dd** is the master address specified in `~/.kube/config`. You can run the **kubectl cluster-info** command to obtain the master address.
2. **spark.kubernetes.container.image** is the address of the pushed image.
3. All parameters that can be specified using `--conf` are read from the `~/spark-obs/conf/spark-defaults.conf` file by default. Therefore, the general configuration can be written to be the default settings, the same way as OBS access configuration.

### Accessing OBS

Use **spark-submit** to deliver an HDFS job. Change the value of **obs://bucket-name/filename** at the end of the script to the actual file name of the tenant.

```
spark-submit \
--master k8s://https://aa.bb.cc.dd:5443 \
--deploy-mode cluster \
```

```
--name spark-hdfs-test \
--class org.apache.spark.examples.HdfsTest \
--conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
--conf spark.kubernetes.container.image=swr.ap-southeast-1.myhuaweicloud.com/dev-container/spark:
3.1.3-obs \
local:///opt/spark/examples/jars/spark-examples_2.12-3.1.1.jar obs://bucket-name/filename
```

## Support for Spark Shell Commands to Interact with Spark-Scala

```
spark-shell \
--conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
--conf spark.kubernetes.container.image=swr.ap-southeast-1.myhuaweicloud.com/dev-container/spark:
3.1.3-obs \
--master k8s://https://aa.bb.cc.dd:5443
```

Run the following commands to define the algorithms of Spark computing jobs linecount and wordcount:

```
def linecount(input:org.apache.spark.sql.Dataset[String]):Long=input.filter(line => line.length()>0).count()
def wordcount(input:org.apache.spark.sql.Dataset[String]):Long=input.flatMap(value => value.split("\\\\s+")).groupByKey(value => value).count().count()
```

Run the following commands to define data sources:

```
var alluxio = spark.read.textFile("alluxio://alluxio-master:19998/sample-1g")
var obs = spark.read.textFile("obs://gene-container-gtest/sample-1g")
var hdfs = spark.read.textFile("hdfs://192.168.1.184:9000/user/hadoop/books/sample-1g")
```

Run the following command to start computing jobs:

```
spark.time(wordcount(obs))
spark.time(linecount(obs))
```