Cloud Container Engine Autopilot

Best Practices

Issue	01
Date	2025-01-03

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

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

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

Contents

1 Deploying Jenkins in a CCE Autopilot Cluster

1.1 Overview

Jenkins is an open-source automation server widely used for continuous integration (CI) and continuous delivery (CD). When your code library changes, Jenkins helps you automatically build, test, and deploy applications, improving development efficiency and product quality. Jenkins can be deployed in different environments. Each environment has their advantages. For details, see **Table 1-1**. In addition, Jenkins can be deployed on a single node or in a distributed mode.

- **Single-node deployment**: Jenkins runs as an independent instance. All builds and operations are performed on the Jenkins master, which is responsible for job scheduling, system management, and execution of specific build jobs. All jobs are running on the same node, which may cause excessive consumption of system resources. In addition, as the project scale and the number of build jobs increase, single-node deployment may become a performance bottleneck. This deployment mode is suitable for small teams or individuals.
- **Distributed deployment**: The Jenkins master is responsible for job scheduling and system management, and the Jenkins agents for executing specific build jobs. The Jenkins master receives build requests from users and distributes jobs to available Jenkins agents. Each Jenkins agent can be independently configured to support different OSs and build tools, providing flexible build environments and scalability. In addition, the separation of management and execution can effectively improve system performance and response speed. This mode is suitable for large-scale production environments, especially when there are a large number of build jobs or there are high requirements for concurrent builds.

This section uses distributed deployment as an example to describe how to deploy and use Jenkins in a CCE Autopilot cluster.

Table 1-1 Comparisons of environments where Jenkins will be deployed

Item	CCE Autopilot	CCE Standard/CCE Turbo	VMs	Physical Machines
Scenario	CI/CD and scenarios that have high requirements for automation management.	Large-scale distributed environment and CI/CD.	Small- and medium-sized projects, or scenarios where multiple teams or projects share one physical machine.	Scenarios that have high requirements for performance and hardware, require stable resources, and do not require frequent expansion.
Performance	High	High	Relatively low	High
Resource utilization	High	High	Relatively low	Low
O&M	Simple	Simple	Less complex	Complex
Scalability	Auto scaling in seconds	Auto scaling in minutes	Relatively poor	Poor
Availability	High	High	High	Relatively low
Isolation level	High	Relatively low	High	High

Precautions

CCE does not provide maintenance and support for Jenkins. The maintenance is provided by the developers.

Basic Concepts of Jenkins

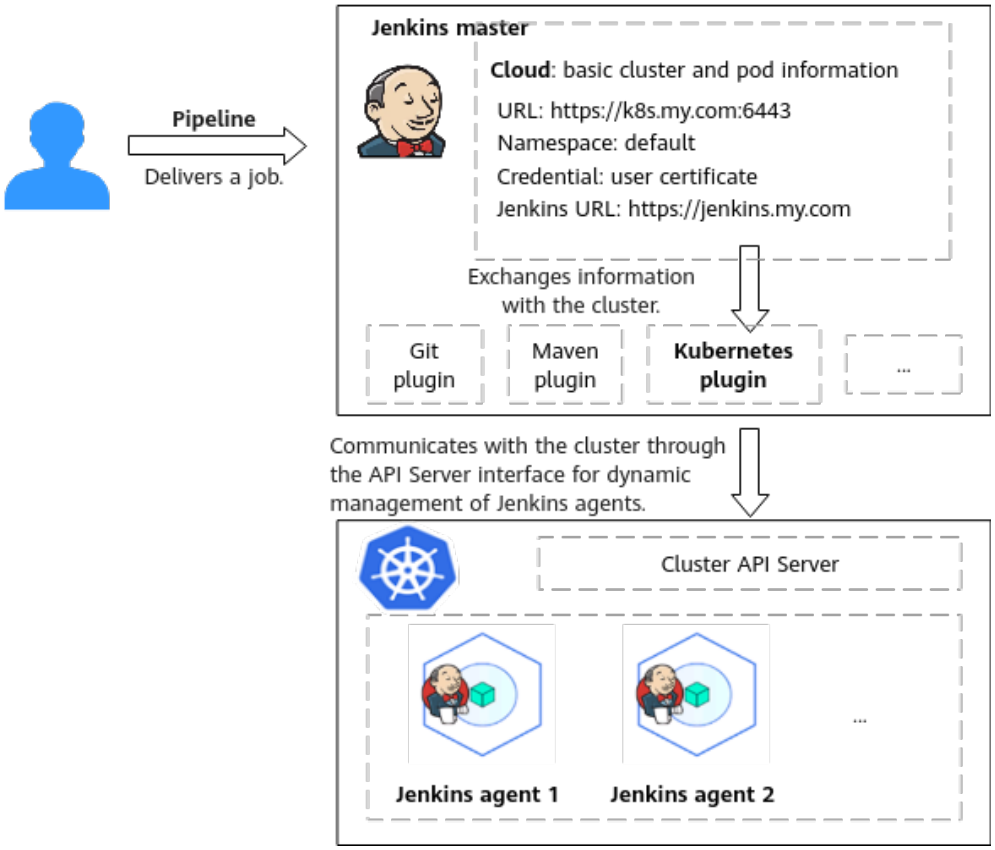
- Jenkins master: the core of the Jenkins system. It manages and coordinates all jobs. It can be regarded as a manager that does not directly execute jobs. Instead, it allocates jobs to other workers (Jenkins agents).

  NOTE

  The Jenkins master provides a web page for users to perform operations and view the task progress.

- Jenkins agent: a pod or machine that Jenkins uses to execute jobs. Multiple Jenkins agents can be configured at the same time to share the load and improve job concurrency and efficiency.
- Plugin: a component that extends the functionality of Jenkins. Jenkins allows users to install different plugins as needed to implement functions such as versioning, build tools, and deployment. In addition, plugins can integrate different tools and technologies, such as Kubernetes, Git, and Maven. The Kubernetes plugin is the key to information exchange between Jenkins and the cluster.
- Pipeline: an automated workflow that connects multiple phases (such as build, test, and deployment) in the software development process to ensure that each step can be automatically executed in a certain sequence and based on certain rules. With the pipeline, you can deliver jobs to the Jenkins master and use the pipeline script to define the entire automation process. The Jenkins master executes the jobs based on the script.
- Cloud: Various cloud environments, such as clusters, containers, and VMs, can be configured to flexibly use compute resources of external cloud platforms and implement dynamic management of Jenkins agents.

Figure 1-1 Logical relationships between basic concepts



Solution Architecture

Figure 1-2 shows the steps for deploying Jenkins and Table 1-2 provides more details.

Figure 1-2 Flowchart

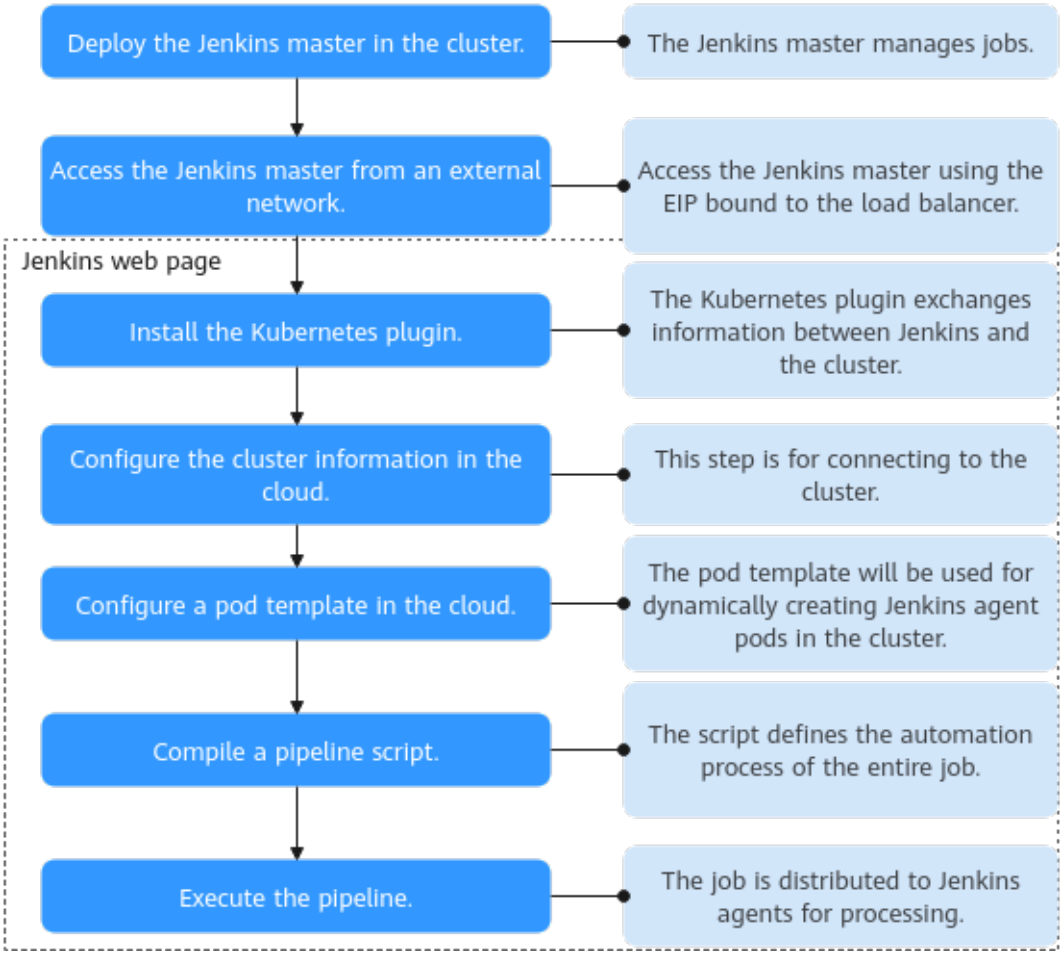


Table 1-2 Procedure

Step	Description	Image
Deploying the Jenkins Master in the Cluster	• Install and deploy the Jenkins master in the CCE Autopilot cluster for managing jobs. • Use a browser to access the Jenkins master through the public IP address of the load balancer.	jenkins/jenkins:lts **NOTE** **jenkins/jenkins:lts** indicates a Docker LTS image. The LTS version is a long-term release provided by Jenkins. It is relatively stable and will receive security updates and bug fixes for a longer time. It is suitable for production systems that require a stable environment. For more information, see **LTS Release Line**.

Step	Description	Image
Configuring the Jenkins Agent	• Install the Kubernetes plugin on the Jenkins web page. • Configure cluster information in the cloud to connect to the cluster. • Configure a pod template for dynamically creating Jenkins agent pods in the cloud.	The Jenkins agent requires three images: • **jenkins/inbound-agent:latest**: used to connect the Jenkins agent and Jenkins master to ensure continuous job execution. • **maven:3.8.1-jdk-8**: used to execute packing jobs in the pipeline. • **gcr.io/kaniko-project/executor:v1.23.2-debug**: used to build and push Docker images in the container.
Building and Executing a Pipeline on Jenkins	• Compile a pipeline script on the Jenkins web page, define the automation process of the entire job, and compile the job into a language that can be understood by the Jenkins master. • The Jenkins master coordinates the execution process of the pipeline, dynamically creates Jenkins agents (in the form of pods) in the cluster through the Kubernetes plugin, and distributes jobs to Jenkins agents for processing. After the jobs are complete, Jenkins agents are automatically deleted. In this example, the pipeline pulls code from the code repository, packs the code into an image, and pushes the image to the SWR image repository.	tomcat (This image needs to be pulled to SWR.)

1.2 Resource and Cost Planning

Figure 1-3 and Table 1-3 describe the resources required in this example and how they are related to each other.

Figure 1-3 Solution architecture

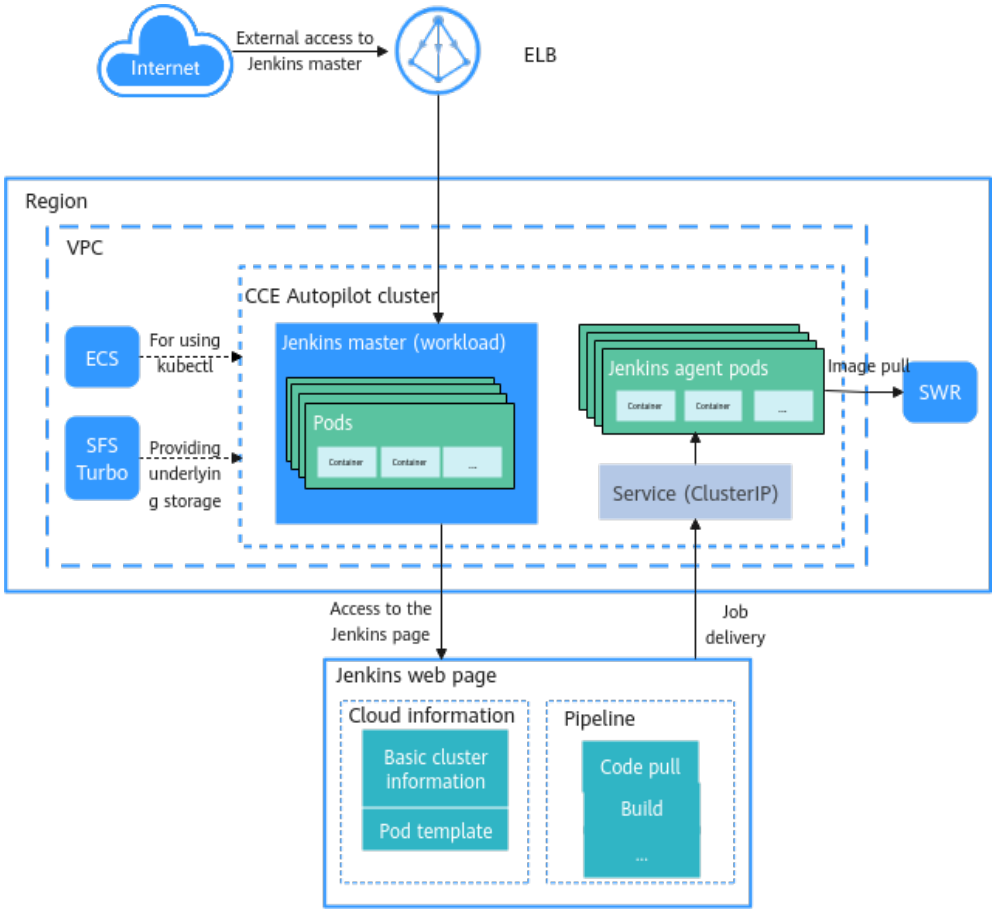


Table 1-3 Required resources and their prices

Resource	Specifications	Description
CCE Autopilot cluster	• Cluster type: CCE Autopilot • Billing mode: pay-per-use • Cluster version: v1.28 • Add-ons: CoreDNS and Kubernetes Metrics Server	One cluster needs to be created. Cluster management and VPC endpoints are billed. For details, see **Billing**.

Resource	Specifications	Description
Pod	Jenkins master: • vCPUs: 4 • Memory: 4 GiB • Storage: 30 GiB Jenkins agent: • vCPUs: 0.5 • Memory: 1 GiB • Storage: 30 GiB	Two pods are required, one for the Jenkins master and the other for the Jenkins agent. The functions of the two pods are as follows: • The Jenkins master is responsible for job scheduling and system management. • The Jenkins agent executes specific build jobs. Pods are billed. For details, see **Billing**.
ECS	• Billing mode: pay-per-use • VM type: General computing-plus • Specifications: 2 vCPUs and 4 GiB of memory • OS: CentOS 7.6 • System disk: 40 GiB | General purpose SSD • EIP – Type: exclusive EIP – Bandwidth billing option: traffic – Bandwidth: 5 Mbit/s	This ECS must be in the same VPC as the cluster. kubectl is installed on this ECS to deliver commands for creating workloads, PVs, PVCs, and secrets. After resources are created, you can delete the ECS in a timely manner to avoid extra expenditures. Deleting ECS does not affect the use of Jenkins. The ECS and EIP traffic are billed. For details, see **ECS Billing**.
SFS Turbo	• Billing mode: pay-per-use • Type: 40 MB/s/TiB • Capacity: 1.2 TB	One SFS Turbo file system is required. SFS Turbo provides underlying storage resources for clusters so that you can use PVs and PVCs to provide persistent storage for workloads. SFS Turbo is billed. For details, see **SFS Turbo Billing**.
Load balancer provided by ELB	• Billing mode: pay-per-use • Type: dedicated load balancer • Bandwidth billing option: traffic • Bandwidth: 5 Mbit/s	One load balancer is required. The load balancer is used by the LoadBalancer Service to allow access to the workloads. ELB is billed. For details, see **ELB Pricing Details**.

Resource	Specifications	Description
SWR shared edition	-	One organization is required. SWR is used to store the images created in **Building and Executing a Pipeline on Jenkins**. Billing is not involved.

1.3 Procedure

1.3.1 Deploying the Jenkins Master in the Cluster

Deploy the Jenkins master as a Deployment in the CCE Autopilot cluster to manage jobs.

NOTE

The Jenkins version used in this example is 2.440.2. The strings on the Jenkins page may vary depending on the version. The screenshots are for reference only.

Preparations

- Purchase a CCE Autopilot cluster. For details, see **Buying a CCE Autopilot Cluster**.
- Prepare a Linux ECS that is in the same VPC as the cluster and has an EIP bound. For details, see **Purchasing and Using a Linux ECS**. Check kubectl on the ECS and **connect to the cluster through kubectl**.
- Create an SFS Turbo file system is in the same VPC as the cluster. For details, see **Creating a File System**.
- Create an organization in SWR. This organization is in the same region as the cluster. For details, see **Creating an Organization**.

Procedure

Step 1 Log in to the ECS. For details, see **Logging In to a Linux ECS Using CloudShell**.

Step 2 Create a PV and PVC of the SFS Turbo type for the Jenkins master to store persistent data.

1. Create a YAML file named **pv-jenkins-master.yaml** for creating a PV. You can change the file name as needed.

   NOTE

   A Linux file name is case sensitive and can contain letters, digits, underscores (_), and hyphens (-), but cannot contain slashes (/) or null characters (\0). To improve compatibility, do not use special characters, such as spaces, question marks (?), and asterisks (*).

   ```
   vim pv-jenkins-master.yaml
   ```


The file content is as follows. In this example, only mandatory parameters are involved. For more parameters, see **Using an Existing SFS Turbo File System Through a Static PV.**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner  # Storage driver. The value is fixed to everest-csi-provisioner.
  name: pv-jenkins-master    # PV name. You can change the name.
spec:
  accessModes:
  - ReadWriteMany     # Access mode. The value must be ReadWriteMany for SFS Turbo.
  capacity:
    storage: 500Gi     # Requested PV capacity.
  csi:
    driver: sfsturbo.csi.everest.io    # Storage driver that the mounting depends on. The value is fixed to sfsturbo.csi.everest.io.
    fsType: nfs        # Storage type. The value is fixed to nfs.
    volumeHandle: ea8a59b6-485c-xxx    # SFS Turbo volume ID
    volumeAttributes:
     everest.io/share-export-location: ea8a59b6-485c-xxx.sfsturbo.internal:/     # Shared path of the SFS Turbo volume
  persistentVolumeReclaimPolicy: Retain    # Reclaim policy.
  storageClassName: csi-sfsturbo           # StorageClass name of the SFS Turbo volume.
```

Press **Esc** to exit editing mode and enter **:wq** to save the file.

Table 1-4 Descriptions of key parameters

Parameter	Example Value	Description
name	pv-jenkins-master	Indicates the PV name. You can use any name. The name can contain 1 to 64 characters and cannot start or end with a hyphen (-). Only lowercase letters, digits, and hyphens (-) are allowed.
accessModes	ReadWriteMany	Indicates the access mode. For SFS Turbo, the value is fixed to **ReadWriteMany**.
storage	**500Gi**	Indicates the requested PV capacity, in Gi.
volumeHandle	**ea8a59b6-485c-xxx**	Specifies the ID of an SFS Turbo volume. **How to obtain**: On the **CCE console**, click  in the upper left corner and choose **Storage** > **Scalable File Service**. In the navigation pane, choose **SFS Turbo** > **File Systems**. In the list, click the name of the target SFS Turbo file system. On the details page, copy the content following **ID**.

Parameter	Example Value	Description
everest.io/share-export-location	**ea8a59b6-485c-xxx.sfsturbo.internal:/**	Specifies the shared path of the SFS Turbo volume. Multiple pods can access the path through the network to share the same storage resource. **How to obtain**: On the **CCE console**, click  in the upper left corner and choose **Storage** > **Scalable File Service**. In the navigation pane, choose **SFS Turbo** > **File Systems**. In the list, click the name of the target SFS Turbo file system. On the details page, copy the content following **Shared Path**.
persistentVolumeReclaimPolicy	**Retain**	Indicates the PV reclamation policy. Only the **Retain** policy is supported. **Retain**: When a PVC is deleted, the PV and underlying storage resources are not deleted. Instead, you must manually delete these resources. After a PVC is deleted, the PV resource is in the **Released** state and cannot be bound to the PVC again.
storageClassName	**csi-sfsturbo**	Specifies the StorageClass name of an SFS Turbo volume. In this example, the built-in StorageClass is used and its name is fixed to **csi-sfsturbo**.

2. Run the following command to create a PV:
   ```
   kubectl create -f pv-jenkins-master.yaml
   ```
   If the following information is displayed, the PV named **pv-jenkins-master** has been created:
   ```
   persistentvolume/pv-jenkins-master created
   ```
3. Create a YAML file named **pvc-jenkins-master.yaml** for creating a PVC. You can change the file name as needed.
   ```
   vim pvc-jenkins-master.yaml
   ```
   The file content is as follows. In this example, only mandatory parameters are involved. For more parameters, see **Using an Existing SFS Turbo File System Through a Static PV**.
   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: pvc-jenkins-master    # PVC name. You can change the name.
     namespace: default    # Namespace. This is also the namespace of the workload.
     annotations:
       volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner  # Storage driver. The value is fixed to everest-csi-provisioner.
   spec:
     accessModes:
     - ReadWriteMany               # Access mode. The value must be ReadWriteMany for SFS Turbo.
     resources:
       requests:
         storage: 500Gi            # Requested capacity of the PVC, which must be the same as the PV
   ```



```
capacity.
  storageClassName: csi-sfsturbo      # StorageClass name of the SFS Turbo file system, which must be the same as that of the PV.
  volumeName: pv-jenkins-master       # Name of the associated PV.
```

Press **Esc** to exit editing mode and enter **:wq** to save the file.

Table 1-5 Descriptions of key parameters

Parameter	Example Value	Description
name	pvc-jenkins-master	Indicates the PVC name. You can use any name. The name can contain 1 to 64 characters and cannot start or end with a hyphen (-). Only lowercase letters, digits, and hyphens (-) are allowed.
namespace	default	Indicates the namespace, which must be the same as the namespace of the workload.
accessModes	ReadWriteMany	Indicates the access mode. For SFS Turbo, the value is fixed to **ReadWriteMany**.
storage	**500Gi**	Indicates the requested PVC capacity, in Gi. The value must be the same as the PV capacity requested in **Step 2.1**.
storageClassName	**csi-sfsturbo**	Indicates the StorageClass name. The value must be the same as the StorageClass of the PV in **Step 2.1**.
volumeName	**pv-jenkins-master**	Specifies the name of the associated PV. The value must be the same as the PV name in **Step 2.1**.

4. Run the following command to create a PVC:

   ```
   kubectl create -f pvc-jenkins-master.yaml
   ```

   If the following information is displayed, the PVC named **pvc-jenkins-master** has been created:

   ```
   persistentvolumeclaim/pvc-jenkins-master created
   ```

5. Verify that the PV has been bound to the PVC. After the PV and PVC are created, they are automatically bound. The PVC can be mounted to the pod only after the binding is successful. When both the PV and PVC are in the **Bound** state, the PV has been bound to the PVC.

   Run the following command to check the PV status:

   ```
   kubectl get pv
   ```


If the value of **STATUS** is **Bound**, the PV is bound.

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
pv-jenkins-master   500Gi      RWX            Retain           Bound    default/pvc-jenkins-master   csi-
sfsturbo               88s
```

Run the following command to check the PVC status:

```
kubectl get pvc
```

If the value of **STATUS** is **Bound**, the PVC is bound.

```
NAME                STATUS   VOLUME              CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc-jenkins-master  Bound    pv-jenkins-master   500Gi      RWX            csi-sfsturbo   61s
```

When both the PV and PVC are in the **Bound** state, the PV has been bound to the PVC.

Step 3 Use the **jenkins/jenkins:lts** image to create a Deployment named **jenkins-master** and mount the PVC created in **Step 2.4**.

NOTE

In this example, the **jenkins/jenkins:lts** image (Docker image of the Jenkins LTS version) is used. The LTS version is a long-term release provided by Jenkins. It is relatively stable and will receive security updates and bug fixes for a longer time. It is suitable for production systems that require a stable environment. For more information, see **LTS Release Line**.

In this example, the Jenkins master is deployed as a Deployment. The Jenkins master is mainly used to manage and schedule jobs and does not depend on persistent data. Deploying the Jenkins master as a Deployment can improve system flexibility and scalability.

You can select different images and workload types as required.

1. Create a YAML file named **jenkins-master** for creating the **jenkins-master** workload. You can change the file name as needed.

```
vim jenkins-master.yaml
```

The file content is as follows. In this example, only mandatory parameters are involved. For details about more parameters, see **Creating a Deployment**.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins-master    # Name of the Deployment.
  namespace: default     # Namespace, which must be the same as the name of the PVC.
spec:
  replicas: 1       # Number of pods running the Deployment.
  selector:
    matchLabels:    # Workload label selector, which is used to match the selected pod to ensure that
the required pod can be selected for the Deployment.
      app: jenkins-master
  template:
    metadata:
      labels:       # Pod label, which must be the same as the value of matchLabels of the workload to
ensure that the pod running the Deployment can be managed in a unified manner.
        app: jenkins-master
    spec:
      containers:
        - name: container-1
          image: jenkins/jenkins:lts   # The jenkins/jenkins:lts image is used.
          resources:      # Used to configure the resource limit and request of the container
            limits:       # Maximum number of resources that can be used by the container
              cpu: '4'
              memory: 4Gi
            requests:      # Resources required for starting the container
              cpu: '4'
              memory: 4Gi
          volumeMounts:    # Volume mounted to the container
            - name: pvc-jenkins-master
```



```
          mountPath: /var/jenkins_home   # Mount path. Generally, the value is /var/jenkins_home.
      volumes:    # Storage volume used by the pod, which corresponds to the created PVC.
        - name: pvc-jenkins-master    # Volume name. You can change the name.
          persistentVolumeClaim:
            claimName: pvc-jenkins-master    # The PVC to be used
      imagePullSecrets:
        - name: default-secret
```

Press **Esc** to exit editing mode and enter **:wq** to save the file.

2. Run the following command to create a Deployment named **jenkins-master**:

```
kubectl create -f jenkins-master.yaml
```

Information similar to the following will be displayed:

```
deployment/jenkins-master created
```

3. To ensure that the Deployment is created, check whether the pod created for the workload is in the **Running** state.

```
kubectl get pod
```

If **STATUS** of the pod whose name is **jenkins-master-xxx** is **Running**, the Deployment has been created.

```
NAME                              READY   STATUS    RESTARTS   AGE
jenkins-master-6f65c7b8f7-255gn   1/1     Running   0          72s
```

Step 4 Create Services for accessing the Jenkins master.

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 Jenkins master and Jenkins agent. In this example, two Services need to be created. For details, see **Table 1-6**.

NOTE

In this example, the Jenkins agent created in the subsequent steps is in the same cluster as the Jenkins master. Therefore, the Jenkins agent uses the ClusterIP Service to connect to the Jenkins master.

When the Jenkins web page needs to communicate with the Jenkins agent, port 8080 must be opened for the Jenkins agent. In this example, both ports 8080 and 50000 are opened for the ClusterIP Service.

If the Jenkins agent needs to connect to the Jenkins master across clusters or over the public network, select an appropriate Service type.

Table 1-6 Service

Service Type	Function	Basic Parameters
LoadBalancer	Allows access to the web from the public network.	• Service name: **jenkins-web** (You can change the name if needed.) • Container port: 8080 • Access port: 8080
ClusterIP	Used by the Jenkins agent to connect to the Jenkins master	• Service name: **jenkins-agent** (You can change the name if needed.) • Container port 1: 8080 • Access port 1: 8080 • Container port 2: 50000 • Access port 2: 50000

1. Create a YAML file named **jenkins-web** to create a LoadBalancer Service. You can change the file name as needed.

   This example describes how to create a Service using an automatically created load balancer. If you want to use an existing load balancer, see **Using kubectl to Create a Service (Using an Existing Load Balancer)**.

   ```
   vim jenkins-web.yaml
   ```

   The file content is as follows. In this example, only mandatory parameters are involved. For more parameters, see **Using kubectl to Automatically Create a Load Balancer**.

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: jenkins-web    # Service name. You change the name as needed.
     namespace: default   # Namespace of the Service.
     labels:
       app: jenkins-web   # Label of the Service.
     annotations:     #Automatic creation of a load balancer
       kubernetes.io/elb.class: performance    # Load balancer type. Only dedicated load balancers are supported.
       kubernetes.io/elb.autocreate: '{
         "type": "public",
         "bandwidth_name": "cce-bandwidth-xxx",
         "bandwidth_chargemode": "traffic",
         "bandwidth_size": 5,
         "bandwidth_sharetype": "PER",
         "eip_type": "5_bgp",
         "available_zone": ["cn-east-3a"
         ],
         "l4_flavor_name": "L4_flavor.elb.s1.small"
       }'
   spec:
     selector:    # Used to select the matched pod.
       app: jenkins-master
     ports:       # Service port information.
     - name: cce-service-0
       targetPort: 8080   # Port used by the Service to access the target pod. This port is closely related to the application running in the pod.
       port: 8080 # Port for accessing the Service. It is also the listening port of the load balancer.
       protocol: TCP
     type: LoadBalancer   # Service type. In this example, this is a LoadBalancer Service.
   ```

   Press **Esc** to exit editing mode and enter **:wq** to save the file.

Table 1-7 Key parameters in the **kubernetes.io/elb.autocreate** field

Parameter	Example Value	Description
type	**public**	Indicates the network type of the load balancer. – **public**: indicates a public network load balancer with an EIP bound to allow access from both public and private networks. – **inner**: indicates a private network load balancer, which does not need an EIP and can be accessed only over a private network. The Service is used to provide external web access, so set this parameter to **public**.

Parameter	Example Value	Description
bandwidth_name	**cce-bandwidth-xxx**	Specifies the bandwidth name. The default value is **cce-bandwidth-xxx**, where *xxx* can be changed as needed. The value can contain 1 to 64 characters. Only letters, digits, underscores (_), hyphens (-), and periods (.) are allowed.
bandwidth_chargemode	**traffic**	Indicates the bandwidth billing option. – **bandwidth**: You are billed by a fixed bandwidth. – **traffic**: You are billed based on the traffic you actually use.
bandwidth_size	**5**	Indicates the bandwidth. The default value is 1 Mbit/s to 2,000 Mbit/s. Configure this parameter based on the bandwidth allowed in your region. The minimum increment for modifying the bandwidth varies depending on the allowed bandwidth. You can only select an integer multiple of the minimum increment. – The minimum increment is 1 Mbit/s if the allowed bandwidth does not exceed 300 Mbit/s. – The minimum increment is 50 Mbit/s if the allowed bandwidth ranges from 300 Mbit/s to 1,000 Mbit/s. – The minimum increment is 500 Mbit/s if the allowed bandwidth exceeds 1,000 Mbit/s.
bandwidth_sharetype	**PER**	Specifies the bandwidth type. The only value **PER** indicates a dedicated bandwidth.
eip_type	**5_bgp**	Specifies the EIP type. – **5_bgp**: Dynamic BGP – **5_sbgp**: Static BGP
available_zone	***cn-east-3a***	Specifies the AZs where the load balancer is located. This parameter is only available for dedicated load balancers. You can obtain all supported AZs by **getting the AZ list**.
l4_flavor_name	**L4_flavor.elb.s1.small**	Specifies the flavor name of the Layer 4 load balancer. This parameter is only available for dedicated load balancers. You can obtain all supported types by **getting the flavor list**.

2. Run the following command to create a LoadBalancer Service to provide external web access:
```
kubectl create -f jenkins-web.yaml
```
Information similar to the following will be displayed:
```
service/jenkins-web created
```
3. Create a YAML file named **jenkins-agent** to create a ClusterIP Service. You can change the file name as needed.
```
vim jenkins-agent.yaml
```
The file content is as follows. In this example, only mandatory parameters are involved. For details about more parameters, see **ClusterIP**.
```
apiVersion: v1
kind: Service
metadata:
  name: jenkins-agent      # Service name. You change the name as needed.
  namespace: default      # Namespace of the Service.
  labels:
    app: jenkins-agent
spec:
  ports:    # Service port information.
  - name: service0         # Port 1: used to ensure that the external access address of the web is the same as the Jenkins agent access address.
    port: 8080             # Port for accessing a Service.
    protocol: TCP          # Protocol used for accessing a Service. The value can be TCP or UDP.
    targetPort: 8080       # Port used by the Service to access the target container. This port is closely related to the application running in a container.
  - name: service1         #Port 2: used for the connectivity between the Jenkins master and Jenkins agent.
    port: 50000
    protocol: TCP
    targetPort: 50000
  selector:                # Label selector. A Service selects a pod based on the label and forwards the requests for accessing the Service to the pod.
    app: jenkins-master
  type: ClusterIP          # Type of a Service. ClusterIP indicates that a Service is only reachable from within the cluster.
```
Press **Esc** to exit editing mode and enter **:wq** to save the file.
4. Run the following command to create a ClusterIP Service for the Jenkins agent to connect to the Jenkins master:
```
kubectl create -f jenkins-agent.yaml
```
Information similar to the following will be displayed:
```
service/jenkins-agent created
```
5. Check whether the Services are successfully created.
```
kubectl get svc
```
The following information is displayed. You can log in to Jenkins using *{EIP of the public network load balancer}:{8080}*.
```
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP                 PORT(S)             AGE
jenkins-agent   ClusterIP      10.247.22.139   <none>                      8080/TCP,50000/TCP  34s
jenkins-web     LoadBalancer   10.247.76.78    xx.xx.xx.xx,192.168.0.239   8080:31694/TCP      15m
kubernetes      ClusterIP      10.247.0.1      <none>                      443/TCP             3h3m
```

Step 5 Log in to and initialize Jenkins.

1. In the address box of the browser, enter *{EIP of the public network load balancer}:{8080}* to open the Jenkins configuration page.
2. Obtain the initial administrator password from the Jenkins pod upon the first login.

a. Return to the ECS and run the following command to query the pod name:

```
kubectl get pod|grep jenkins-master
```

The following information is displayed. **jenkins-master-6f65c7b8f7-255gn** indicates the pod name.

```
jenkins-master-6f65c7b8f7-255gn   1/1     Running   0          144m
```

b. Run the following command to enter the pod (**jenkins-master-6f65c7b8f7-255gn**):

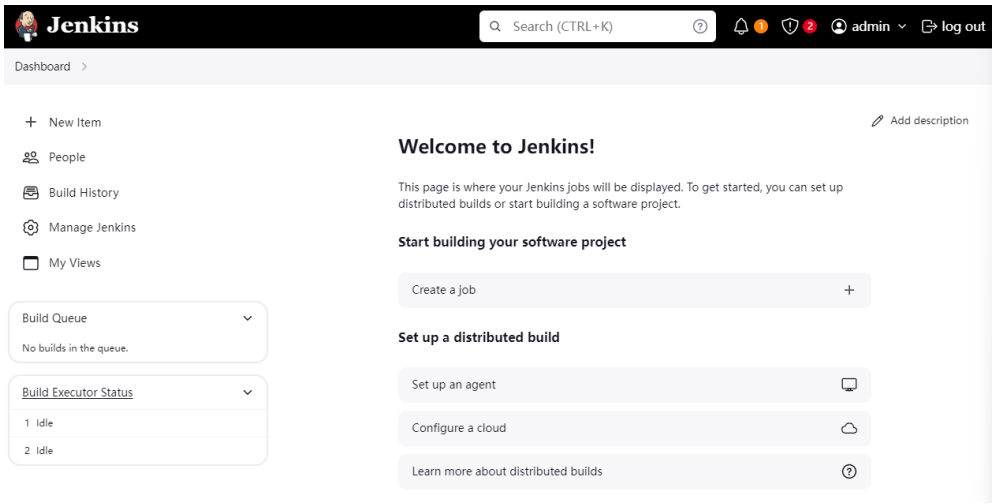
```
kubectl exec -it jenkins-master-6f65c7b8f7-255gn -- /bin/sh
```

c. Run the following command to obtain the initial administrator password:

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

3. Install the recommended add-ons and create an administrator as prompted upon the first login. After the initial configuration is complete, the Jenkins web page is displayed.

Figure 1-4 Jenkins web page



----End

1.3.2 Configuring the Jenkins Agent

In this section, you need to complete the following tasks:

- Install the Kubernetes plugin on the Jenkins web page and configure cluster information in the cloud for connecting to the cluster.
- Configure a pod template for dynamically creating Jenkins agent pods in the cloud.

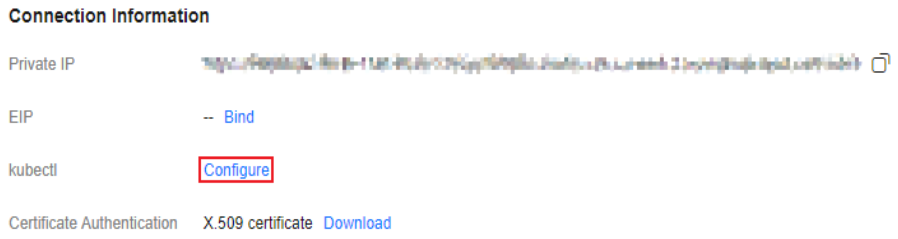
Before the installation and configuration, complete **Preparations for the Cluster**.

Preparations for the Cluster

Before configuring the Jenkins agent, you need to perform some operations on the cluster to support subsequent configuration of the Jenkins agent.

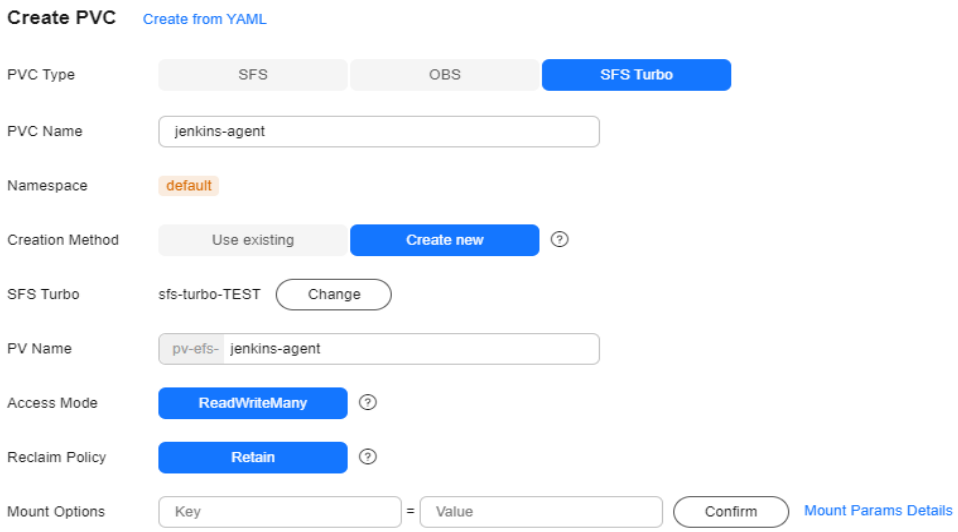
Step 1 Return to the **CCE console** and click the cluster name. In the **Connection Information** area on the right, click **Configure** to download the kubectl configuration file, which will be used as the credential for Jenkins to connect to the cluster.

Figure 1-5 Connection Information



Step 2 In the navigation pane of the cluster console, choose **Storage**. In the upper right corner, click **Create PVC**. In the **Create PVC** dialog, configure the following parameters and click **Create**. The created PVC persistently stores the data generated when the Jenkins agent completes jobs.

Figure 1-6 Creating a PVC



- **PVC Type**: **SFS Turbo**
- **PVC Name**: **jenkins-agent**
- **Creation Method**: **Create new**
- **SFS Turbo**: Select the SFS Turbo volume used in **Step 2**.
- **PV Name**: **pv-efs-jenkins-agent**

Step 3 Return to the ECS and create a secret with SWR authentication information as the credential for pushing images to SWR.

1. Download jq to process and operate JSON data. You can query, filter, modify, and format JSON data. The following uses an ECS running CentOS 7.6 as an example.
   ```
   yum install jq
   ```
2. Create the Docker registry secret to store SWR authentication information. Extract and decode the SWR authentication information and save it to the **/tmp/config.json** file.

- **docker-server**: Enter the SWR image repository address in the format of *swr.[Region].myhuaweicloud.com*.

  Obtain the region from **Regions and Endpoints**. Replace *[Region]* with the actual region name, for example, **swr.cn-east-3.myhuaweicloud.com** for CN East-Shanghai1.

- **docker-username**: Enter the username in the SWR login command.

  To obtain the username, log in to the **SWR console**, click **Login Command** in the upper right corner of the **Dashboard** page, and view the command on the **Temporary login command** tab. The content following **-u** in the command is the username.

- **docker-password**: Enter the password in the SWR login command.

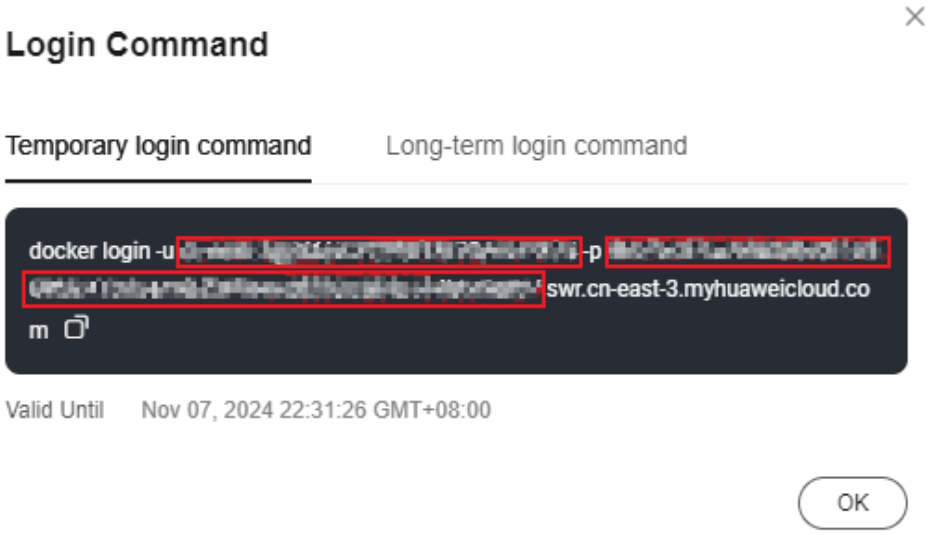
  The content following **-p** in the command on the **Temporary login command** tab is the password.

  NOTE

  The validity period of the temporary login command is 6 hours. After the temporary login command expires, you need to reconfigure the validity period.

  You can select **Long-term login command** on the **Login Command** page and configure related information as prompted to obtain the long-term login command and then the username and password.

Figure 1-7 Obtaining docker-username and docker-password




```
kubectl create secret docker-registry swr-secret \
  --docker-server=https://swr.xxx.myhuaweicloud.com \
  --docker-username=xxx \
  --docker-password=xxx \
  --dry-run=client -o json | jq -r '.data.".dockerconfigjson"' | base64 -d > /tmp/config.json
```

3. Use the **/tmp/config.json** file to create a generic secret. The secret can be directly mounted to the pod of the Jenkins agent created later.

```
kubectl create secret generic swr-secret --from-file=/tmp/config.json -n default
```

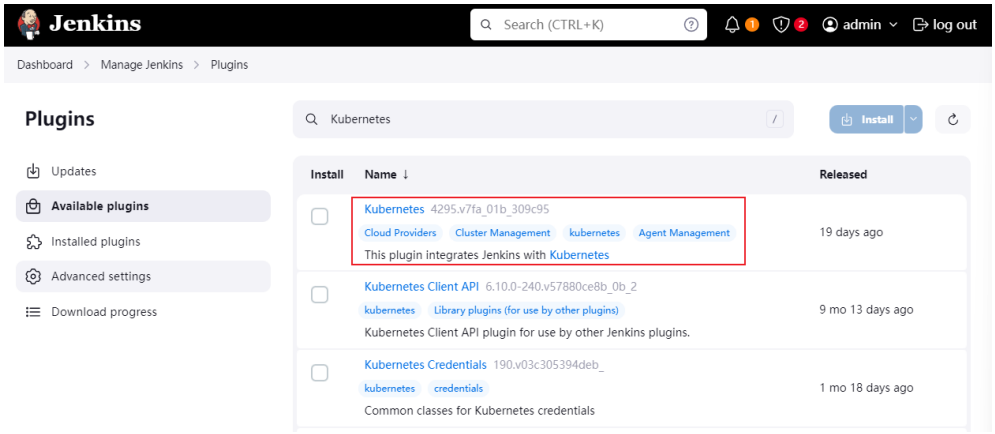
----End

Configuring Cloud Information on the Jenkins Web Page

Step 1 Return to the Jenkins web page. In the navigation pane, choose **Manage Jenkins** > **System Configuration** > **Plugins** > **Available plugins**. On the **Available plugins** tab, search for and install the Kubernetes plugin. The Kubernetes plugin dynamically creates a pod for the Jenkins agent in the cluster and deletes the pod after it is used.

The plugin version may change over time. Select a plugin version as required. In this example, the plugin version is **4295.v7fa_01b_309c95**. You can install other plugins as required, such as **Kubernetes CLI Plugin** (which allows kubectl to be configured for a job to interact with Kubernetes clusters).

Figure 1-8 Searching for the **Kubernetes** plugin



Step 2 In the upper left corner of the current page, click **Manage Jenkins** and then choose **Security** > **Security**. In the **CSRF Protection** area, select **Enable proxy compatibility** and click **Apply**.

NOTE

Selecting **Enable proxy compatibility** is to avoid "Error 403 No valid crumb was included in the request".

Jenkins uses CSRF protection to prevent cross-site request forgery attacks. When a user performs sensitive operations (such as building a project), Jenkins requires a valid "crumb". When a reverse proxy (such as Nginx or Apache) or load balancer is used, requests are forwarded from the client to the Jenkins server. The proxy or load balancer may modify the request header, and the CSRF token (crumb) will be lost or will not be passed correctly, resulting in the "Error 403 No valid crumb was included in the request" error.

After **Enable proxy compatibility** is selected, Jenkins uses a fault tolerance mechanism to ensure that it can properly process transferred requests in the proxy environment, so that CSRF tokens (crumbs) can be correctly transferred and verified through the proxy.

Figure 1-9 Selecting **Enable proxy compatibility**

Step 3 In the upper left corner of the current page, click **Manage Jenkins**, choose **Security** > **Credentials**, choose **Stores scoped to Jenkins** > **System** > **Global credentials (unrestricted)**, and click **Add Credentials** on the right to add a cluster credential.

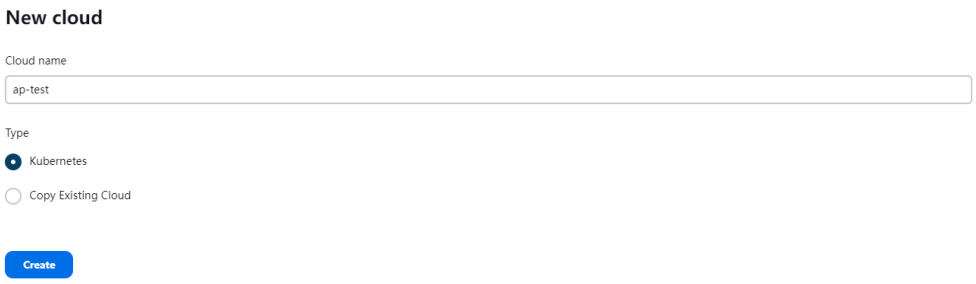
On the **New credentials** page, set **Kind** to **Secret file**, **Scope** to **Global (Jenkins, nodes, items, all child items, etc)**, and **File** to the downloaded kubectl configuration file. Retain the default values for other parameters and click **Create**.

Step 4 Create a cloud, which will be used to configure cluster information so that Jenkins can match the correct cluster.

1. In the upper left corner of the current page, click **Manage Jenkins**, then choose **System Configuration** > **Clouds**, click **New Cloud** to create a cloud, and enter the basic information about the cloud.

   Enter a cloud name, select **Kubernetes** for **Type**, and click **Create**.

   Figure 1-10 Basic cloud information

2. Specify cluster information.

   Figure 1-11 Cluster details

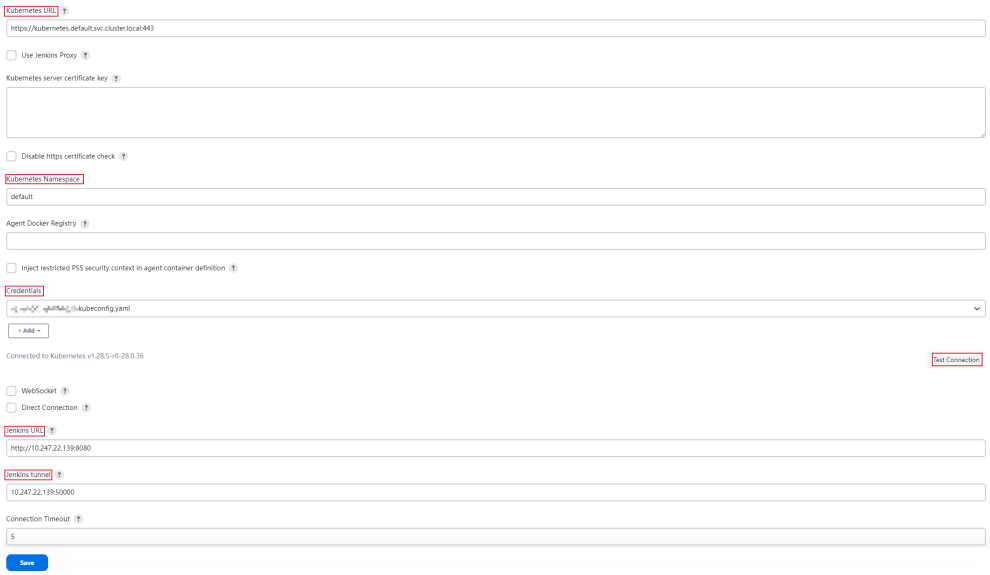
   

Table 1-8 Cluster parameters

Parameter	Example Value	Description
Kubernetes URL	**https://kubernetes.default.svc.cluster.local:443**	Indicates the address of the cluster APl Server. You can directly enter **https://kubernetes.default.svc.cluster.local:443**, which is the standard DNS address for accessing the Kubernetes API server in the cluster.
Kubernetes Namespace	**default**	Specifies the namespace where the dynamically created Jenkins agent is located. **NOTE** The namespace must be the same as that of the **jenkins-master** workload created in **Step 3**.
Credentials	**xxx-kubeconfig.yaml**	Specifies the cluster connection credential. Select the credential uploaded in **Step 3**. **NOTE** After selecting a credential, click **Test Connection** on the right to check whether the cluster can be connected. If **Connected to Kubernetes xxx** is displayed in the command output on the left, the cluster can be connected.
Jenkins URL	**http://10.247.22.139:8080**	Indicates the Jenkins access path. Enter the IP address for intra-cluster access in **Step 4**. The port number is **8080**.
Jenkins tunnel	**10.247.22.139:50000**	Indicates the tunnel that is used to establish connectivity between the Jenkins master and Jenkins agent. Enter the IP address for intra-cluster access in **Step 4**. The port number is **50000**.

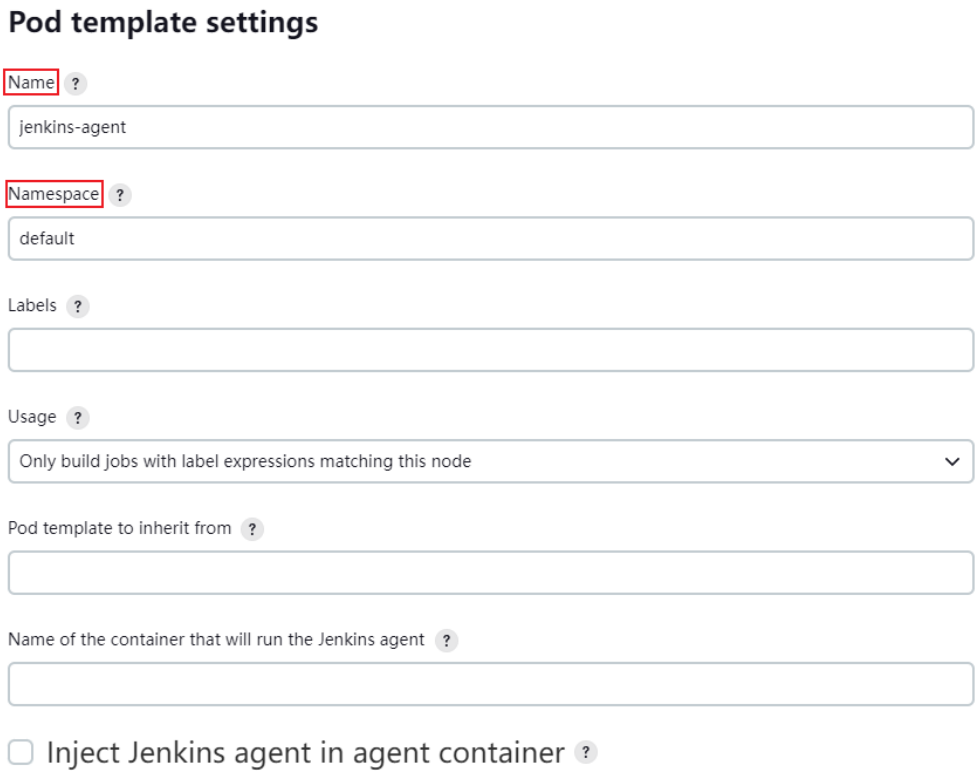
3. Confirm the preceding information and click **Save**.

Step 5 Configure a pod template. With this template, Jenkins can create pods for the Jenkins agent in the cluster as required and use the created pods to run Jenkin jobs. The pods are created on demand and are automatically deleted after the jobs are complete.

1. Click the cloud name and choose **Pod Templates** > **Add a pod template**.
2. Configure basic parameters for the pod template.
   - **Name**: name of the pod template. You can name the pod template as needed, for example, **jenkins-agent**.

- **Namespace**: namespace of the pod to be created. The namespace must be the same as that in the cloud, for example, **default**.
- Other parameters: You can configure them as required. In this example, retain the default values.

Figure 1-12 Configuring basic parameters for the pod template



3. Add a container template. In this example, three container templates need to be added. The parameters are described in **Table 1-9** in the form of container 1, container 2, and container 3. You can add three container templates based on the table.
   - Container 1: The **jenkins/inbound-agent:latest** image is used to connect the Jenkins agent to the Jenkins master to ensure continuous job execution.
   - Container 2: The **maven:3.8.1-jdk-8** image is used to execute packing jobs in the pipeline.
   - Container 3: The **gcr.io/kaniko-project/executor:v1.23.2-debug** image is used to build Docker images in the container.

     NOTE

     You should push the three images to the SWR image repository in advance to improve the container creation speed and reliability. For details, see **Uploading an Image Through a Client**.

     With images stored in the SWR image repository, Jenkins does not need to pull images from external sources, accelerating container creation and reducing network latency. This also reduces the risk of container creation failures caused by network fluctuation or image pull failures, ensuring a more stable, efficient build process.

Figure 1-13 Container template parameters

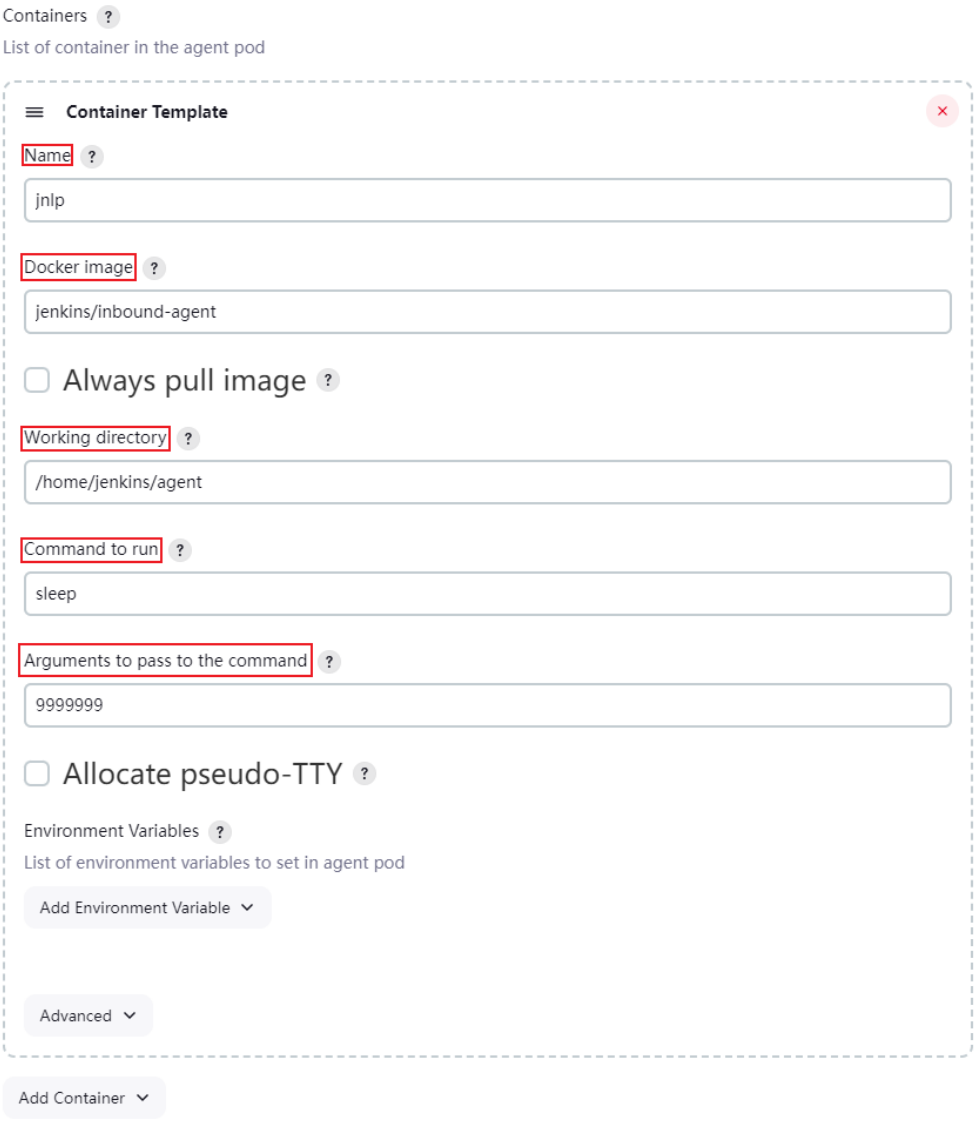


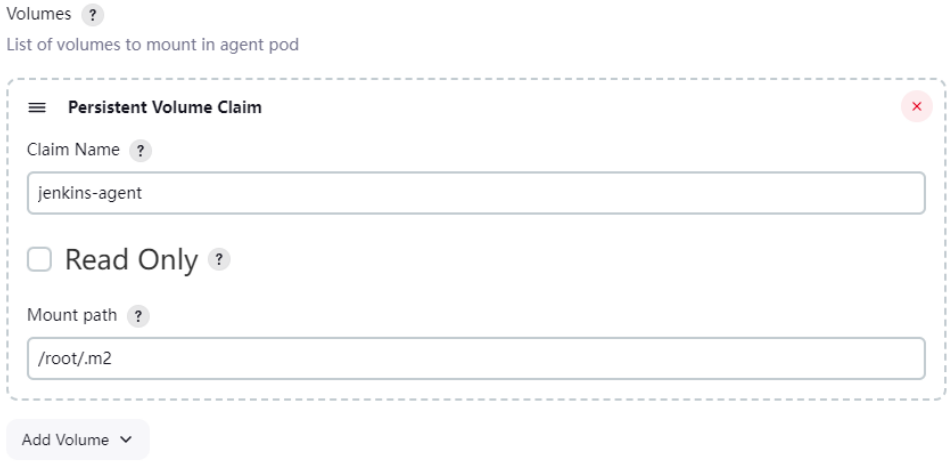
Table 1-9 Container template parameters

Parameter	Example Value	Description
Name	Container 1: **jnlp** Container 2: **maven** Container 3: **kaniko**	Indicates the name of each container created in the cluster. The name of container 1 is fixed to **jnlp**. You can name other containers as needed.

Parameter	Example Value	Description
Docker image	Container 1: **jenkins/inbound-agent:latest** Container 2: **maven:3.8.1-jdk-8** Container 3: **gcr.io/kaniko-project/executor:v1.23.2-debug**	Indicates the image required for creating a container. If you have pushed the images to SWR, change the value to the image path in SWR.
Working directory	Containers 1 to 3: **/home/jenkins/agent**	Indicates the default file storage location of the containers during the execution of build jobs. You can change the directory as needed.
Command to run	Containers 1 to 3: **sleep**	Indicates the command that is executed when the container is started.
Arguments to pass to the command	Containers 1 to 3: **9999999**	Specifies the parameters to be transferred to **Command to run**. The **sleep 9999999** command indicates that the container keeps running until it already runs for 9,999,999 seconds or is manually stopped. This configuration is used to keep the container active and prevent the container from automatically exiting when there is no job.

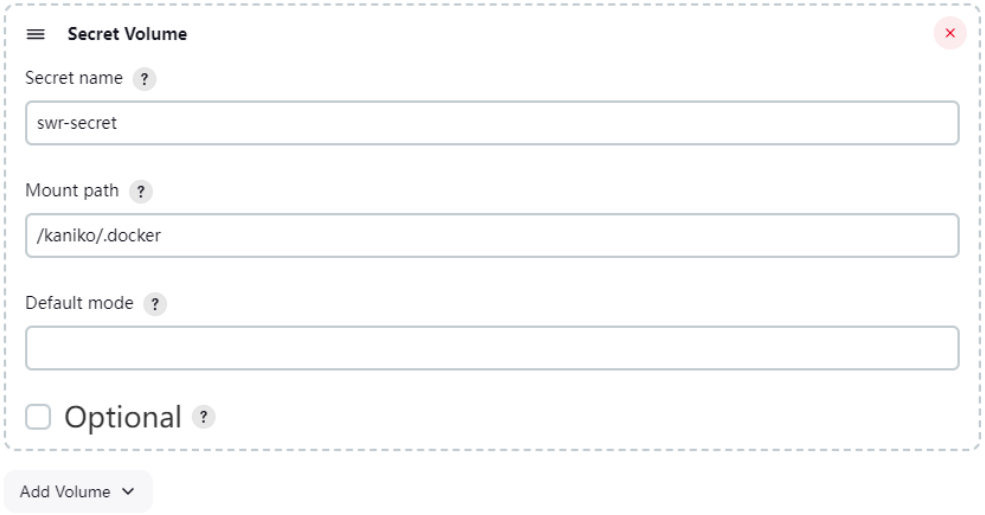
4. Click **Add Volume**, select **Persistent Volume Claim**, and configure the parameters. The PVC is mounted to all containers to provide persistent storage for each container.
   - **Claim Name**: Enter the name of the PVC created in **Step 2**.
   - **Mount path**: Enter the mount path. The value is fixed to **/root/.m2**.

Figure 1-14 Configuring a PVC



5. Click **Add Volume** again, select **Secret Volume**, and configure the parameters. When a pipeline job is being executed, the secret is used as a credential for the kaniko container to push images to SWR.
   - **Secret Name**: Enter the name of the secret created in **Step 3**.
   - **Mount path**: Enter the mount path. The value is fixed to **/kaniko/.docker**.

Figure 1-15 Configuring a secret



6. Configure the secret for pulling the image. In this example, **default-secret** is used.

   NOTE

   When pulling images in your account from SWR, you can use this secret. To use images in other accounts, you need to create a secret for a third-party image repository. For details, see **Creating a Secret for a Third-Party Image Repository**.

Figure 1-16 Configuring the image pull secret

7. Confirm the preceding information and click **Save**.

----End

1.3.3 Building and Executing a Pipeline on Jenkins

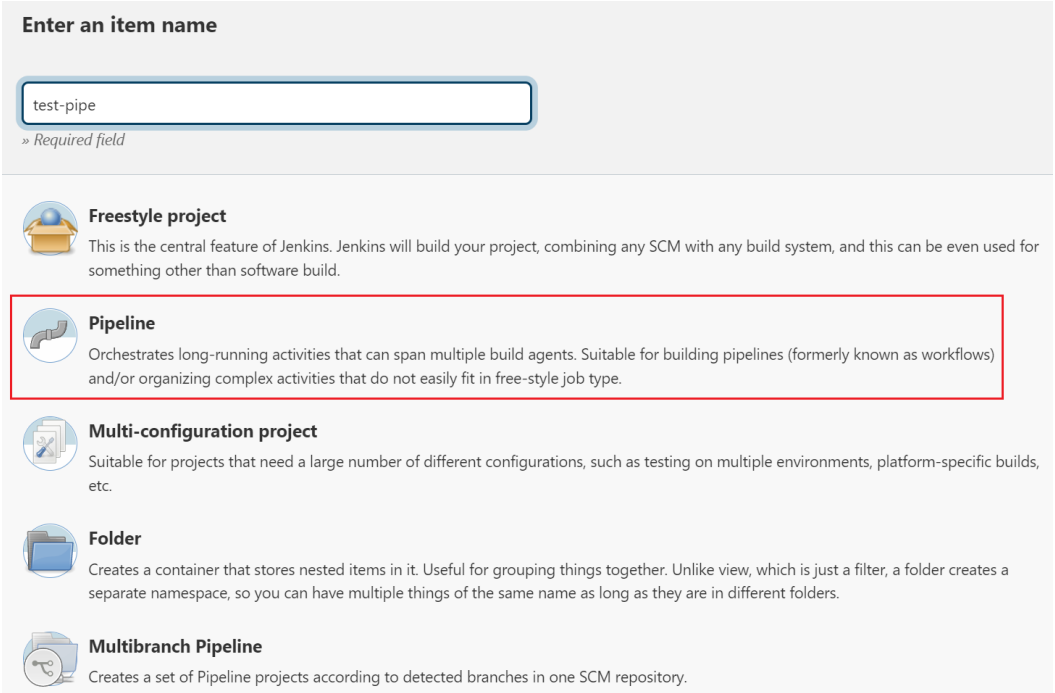
Building a Pipeline

Build a pipeline in Jenkins to pull code from the code repository, pack the code into an image, and push the image to the SWR image repository.

Step 1 Click **Dashboard** in the upper left corner of the page to switch to the **Jenkins Dashboard** page. In the navigation pane on the right, click **New Item**.

Step 2 Enter an item name (for example, **test-pipe**) and select **Pipeline**.

Figure 1-17 Pipeline



Step 3 Configure only the pipeline script and retain the default values for other parameters.

The following pipeline script is for reference only. You can modify the script content based on your service requirements. For details about the syntax, see **Pipeline**.

```
def swr_region = "cn-east-3"
def organization = "container"
```



```
def git_repo = "http://github.com/xxx.git"
def app_git_branch = "master"


podTemplate(
inheritFrom: 'jenkins-agent',    // Replace the value with the name of the pod template created in Step 5.
cloud: 'ap-test'    // Replace the value with the name of the cloud created in Step 4.
) {
  // Pull the code from the code repository.
  node(POD_LABEL) {
    stage('Pull the code.') {
         echo "pull clone"
         git branch: "${app_git_branch}",  url: "${git_repo}"
     }

  // Use the maven container to pack the code pulled from the code repository. (This packing method applies only to Java. Use another packing method for other languages.)
  container('maven'){
    stage ('Pack the code.') {
         echo "build package"
         sh "mvn clean package  -DskipTests"
     }
   }

  // Use the kaniko container to push the packed code to SWR and name the image tomcat.
  container('kaniko'){
    stage('Push the image.') {
         echo "build images and push  images"
         sh "/kaniko/executor -f Dockerfile -c . -d swr.${swr_region}.myhuaweicloud.com/${organization}/tomcat:${BUILD_ID} --force"
      }
   }
}
}
```

Table 1-10 Pipeline script parameters

Parameter	Example Value	Description
swr_region	**cn-east-3**	Region where the SWR image repository is located. For details, see **Regions and Endpoints**. **NOTE** The SWR image repository is used to store images packaged by code. The region must be the same as that in **Step 3**.
organization	**container**	Organization name in SWR. You can enter any organization name as needed.
git_repo	**https://github.com/xxx.git**	Specific address where the code is stored, which is the address of the code library.
app-git-branch	**master**	Branch of the code library.

Step 4 Click **Save**.

----End

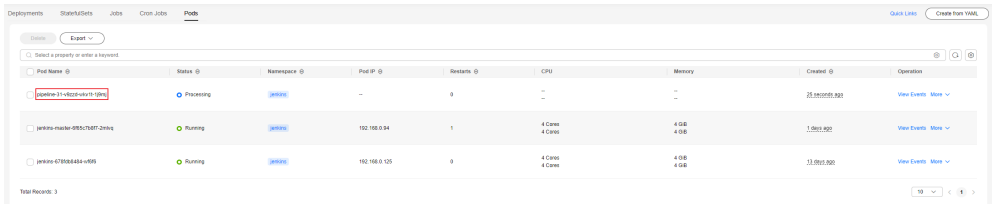
Executing the Pipeline and Viewing the Execution Result

After the pipeline is executed, a pod named **pipe-xxx** will be automatically created in the cluster, and three containers (named **jnlp**, **kaniko**, and **maven**) will be created in the pod based on the information in the pod template. The pod pulls code from the code repository, packs the code into an image, and pushes the image to the SWR image repository. After the operations are complete, the pod is automatically deleted.

Step 1 In the navigation pane, choose **Build Now** to execute the pipeline job.

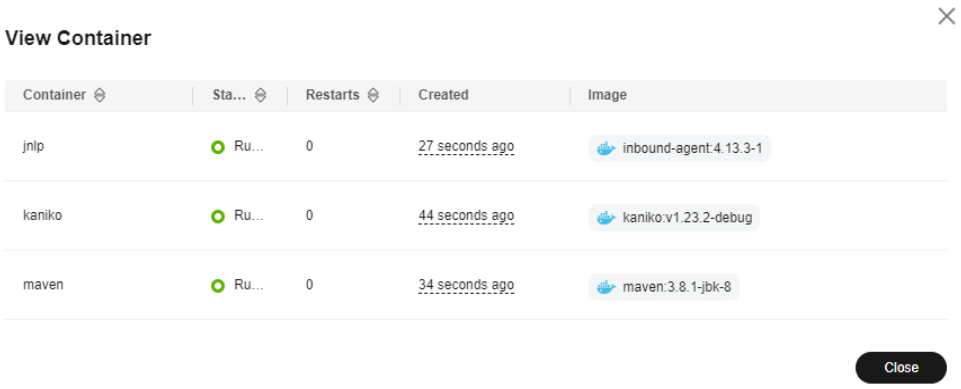
Step 2 Return to the **CCE console** and click the cluster name. In the navigation pane, choose **Workloads**. On the **Pods** tab, view the pod created by the pipeline.

Figure 1-18 Pod created by the pipeline



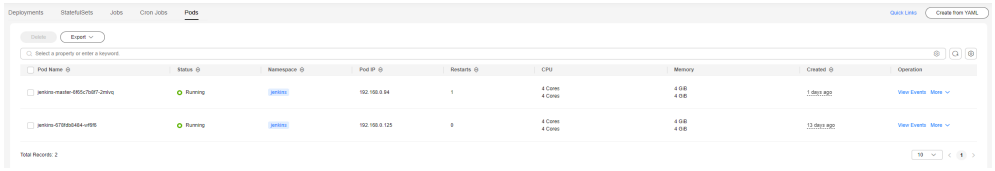
Step 3 Click **More** > **View Container** in the **Operation** column. You can see that three containers are created based on the pod template.

Figure 1-19 Creating a container



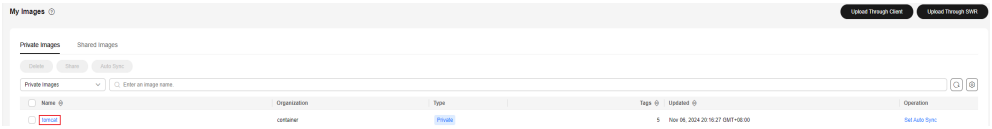
Step 4 Verify that the pod has been deleted automatically. After the code is pulled from the code repository and packed into an image, and the image is pushed to the SWR image repository, the pod will be automatically deleted.

Figure 1-20 Automatic pod deletion



Step 5 Log in to the **SWR console** and verify that the Tomcat image is available in the SWR image repository.

Figure 1-21 Pushed image



----End

Follow-up Operations: Releasing Resources

To avoid additional expenditures, release resources promptly if you no longer need them.

Step 1 Log in to the **CCE console**. In the navigation pane, choose **Clusters**.

Step 2 Locate the cluster, click ··· in the upper right corner of the cluster card, and click **Delete Cluster**.

Step 3 In the displayed **Delete Cluster** dialog box, delete related resources as prompted.

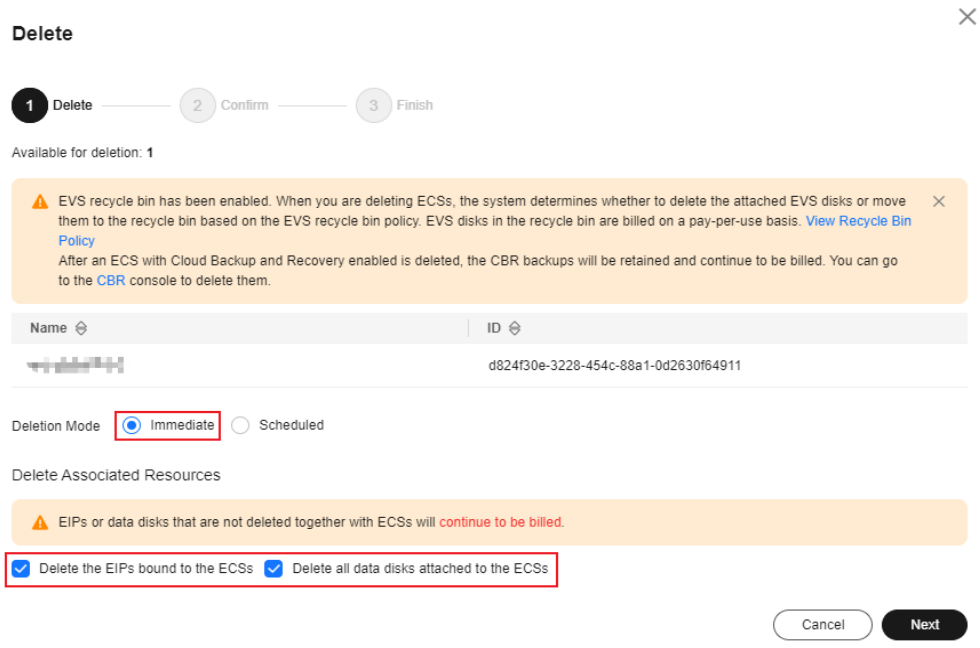
Enter **DELETE** and click **Yes** to start deleting the cluster.

It takes 1 to 3 minutes to delete a cluster. If the cluster name disappears from the cluster list, the cluster has been deleted.

Step 4 Log in to the **ECS console**. In the navigation pane, choose **Elastic Cloud Server**. Locate the target ECS and click **More** > **Delete**.

In the displayed dialog, select **Delete the EIPs bound to the ECSs** and **Delete all data disks attached to the ECSs**, and click **Next**.

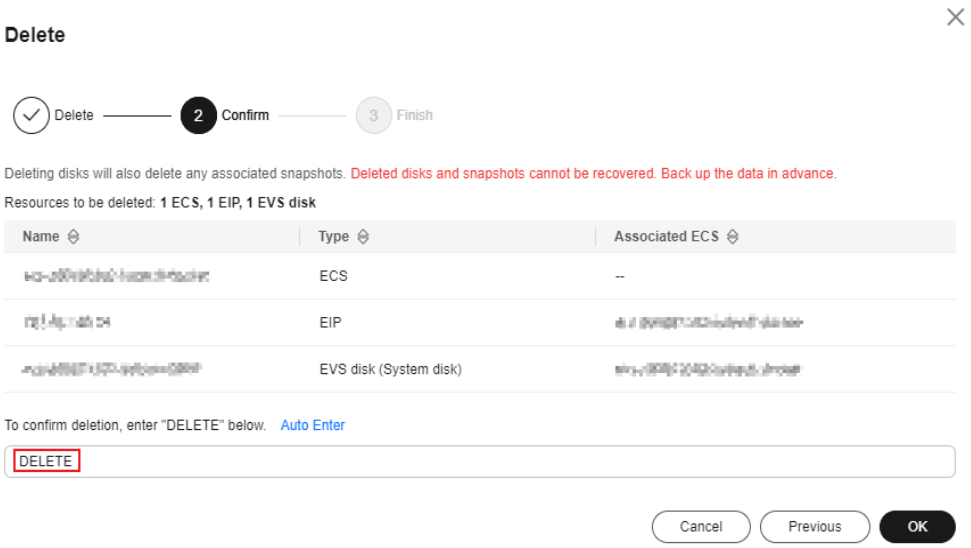
Figure 1-22 Deleting ECSs



Enter **DELETE** and click **OK** to start deleting the ECS.

It takes 0.5 minutes to 1 minute to delete an ECS. If the ECS name disappears from the ECS list, the ECS has been deleted.

Figure 1-23 Confirming the deletion



----End

