Cloud Container Engine

Getting Started

Issue 01

Date 2025-07-29

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.

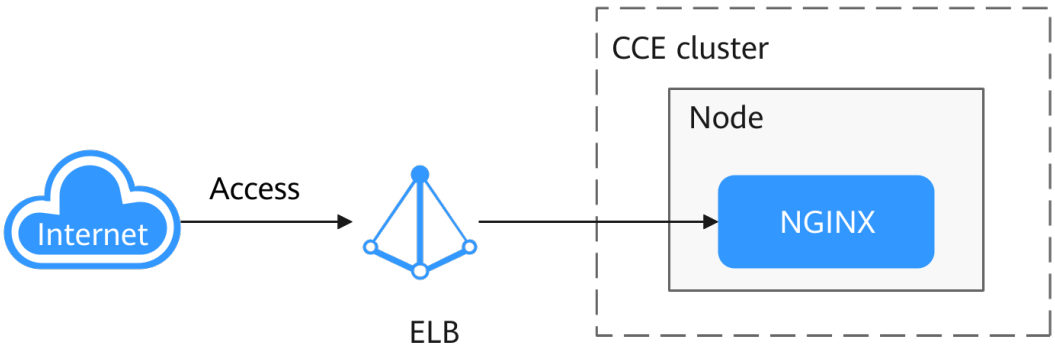
The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Deploying an Nginx Deployment in a CCE Cluster

Deployments are a type of workload in Kubernetes. They are ideal for stateless applications, such as web and application servers, which do not require data consistency or durability. The pods in a Deployment are independent, identical, and replaceable. If a pod fails, traffic is automatically redirected to healthy pods, ensuring service continuity. You can also adjust the number of pods to dynamically adapt to service demands in real time. For instance, you may need to add pods during peak hours to handle traffic surges.

This section uses Nginx, a lightweight web server, as an example to describe how to deploy a Deployment in a CCE cluster.



Procedure

Step	Description
Preparations	Sign up for a HUAWEI ID.
Step 1: Enable CCE and Perform Authorization	Obtain the required permissions for your account when you use CCE in the deployment region for the first time.
Step 2: Create a Cluster	Create a CCE cluster to provide Kubernetes services.

Step	Description
Step 3: Create a Node Pool and Nodes	Create nodes in the cluster to run containerized applications.
Step 4: Create a Workload and Access It	Create a workload in the cluster to run your containers and create a Service to enable public access.
Follow-up Operations: Release Resources	Release resources promptly if the cluster is no longer needed to avoid extra charges.

Preparations

- Before you start, sign up for a HUAWEI ID. For details, see **Signing up for a HUAWEI ID and Enabling Huawei Cloud Services**.

Step 1: Enable CCE and Perform Authorization

When you first log in to the CCE console, CCE automatically requests permissions to access related cloud services (compute, storage, networking, and monitoring) in the region where the cluster is deployed. If you have authorized CCE in the deployment region, skip this step.

Step 1 Log in to the **CCE console** using your HUAWEI ID.

Step 2 Click  in the upper left corner and select a region.

Step 3 If this is your first login to the CCE console in the selected region, the **Authorization Statement** dialog box will appear. Read it carefully and click **OK**.

After you agree to delegate permissions, CCE uses IAM to create an agency named **cce_admin_trust**. This agency is granted **Tenant Administrator** permissions for the resources of other cloud services (excluding IAM). These permissions are required for CCE to access dependent cloud services and are only valid for the current region. You can view the authorization records in each region by navigating to the IAM console, choosing **Agencies** in the navigation pane, and clicking **cce_admin_trust**. For more details, see **Cloud Service Agency**.

To ensure CCE can run normally, do not delete or modify the **cce_admin_trust** agency, as CCE requires **Tenant Administrator** permissions.

NOTE

CCE has updated the **cce_admin_trust** agency permissions to enhance security while accommodating dependencies on other cloud services. The new permissions no longer include **Tenant Administrator** permissions. This update is only available in certain regions. If your clusters are of v1.21 or later, a message will appear on the console asking you to re-grant permissions. After re-granting, the **cce_admin_trust** agency will be updated to include only the necessary cloud service permissions, with the **Tenant Administrator** permissions removed.

When creating the **cce_admin_trust** agency, CCE creates a custom policy named **CCE admin policies**. Do not delete this policy.

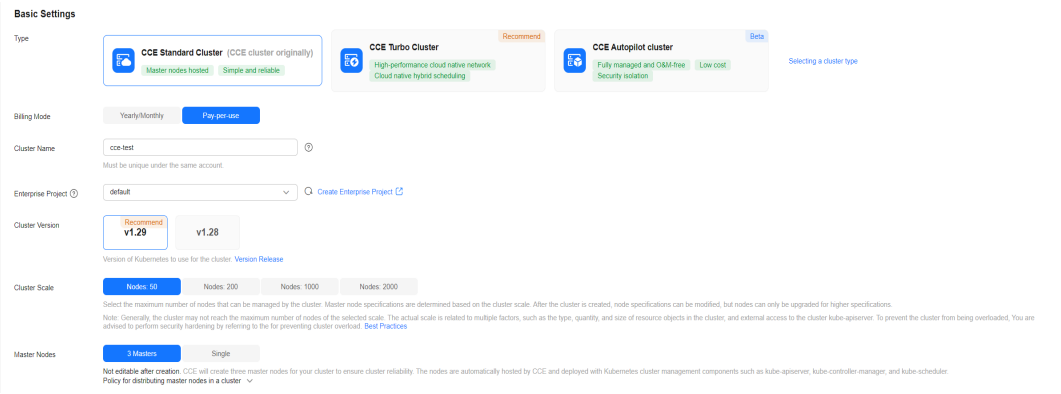
----End

Step 2: Create a Cluster

Step 1 Log in to the **CCE console** and click **Buy Cluster**.

Step 2 Configure basic cluster parameters.

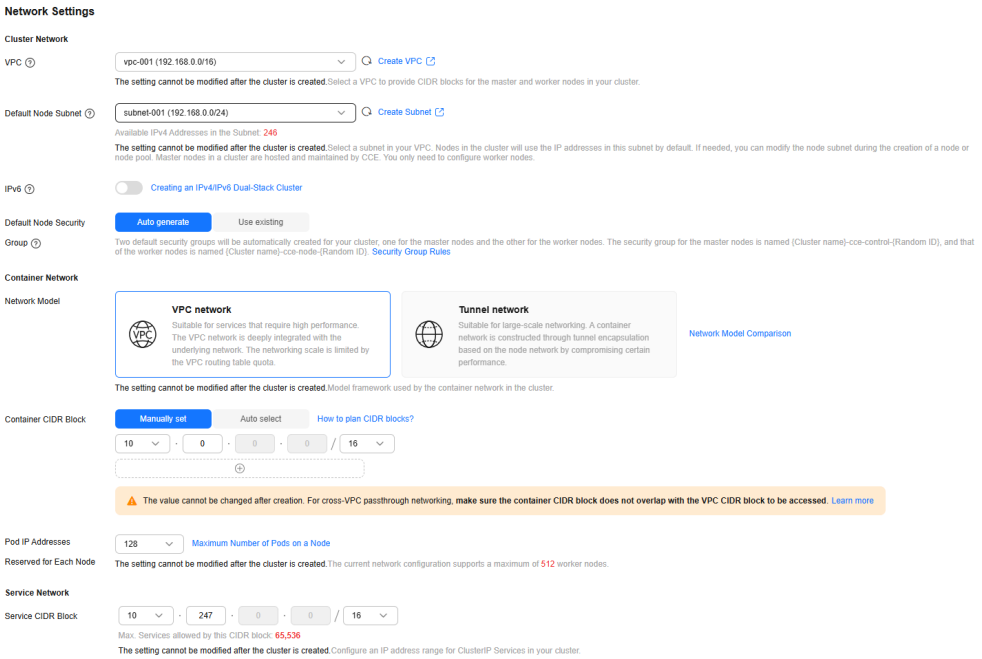
This example describes only mandatory parameters. You can keep default settings for most other parameters. For details, see **Buying a CCE Standard/Turbo Cluster**.



Parameter	Example	Description
Type	CCE Standard Cluster	CCE supports multiple cluster types to meet diverse needs, including highly reliable, highly secure, commercial containers. You can select **CCE Standard Cluster** or **CCE Turbo Cluster** as required. • CCE standard clusters provide highly reliable, highly secure, commercial containers. • CCE Turbo clusters use high-performance cloud native networks and support cloud native hybrid scheduling. These clusters offer improved resource utilization and are suitable for a wider range of scenarios. For more details, see **Comparison Between Cluster Types**.

Parameter	Example	Description
Billing Mode	Pay-per-use	Select a billing mode for the cluster. • **Yearly/Monthly**: a prepaid billing mode. Resources are billed based on how long you will need the resources for. This mode is more cost-effective when resource usage periods are predictable. If you choose this option, select the **desired duration** and decide whether to enable automatic renewal (monthly or yearly). Monthly subscriptions renew automatically every month, and yearly subscriptions renew automatically every year. • **Pay-per-use**: a postpaid billing mode. Resources are billed based on actual usage duration. This mode is suitable for flexible scenarios where you may need to provision or delete resources at any time. For more details, see **Billing Modes**.
Cluster Name	cce-test	Enter a name for the cluster.
Enterprise Project	default	Enterprise projects facilitate project-level management and grouping of cloud resources and users. This parameter is only displayed for enterprise users who have enabled enterprise projects.
Cluster Version	v1.29 (recommended)	Select the latest Kubernetes commercial release to benefit from new, reliable, and production-ready features. CCE offers multiple Kubernetes versions.
Cluster Scale	Nodes: 50	This parameter controls the maximum number of worker nodes the cluster can manage. Configure it as needed. After the cluster is created, it can be scaled out, but it cannot be scaled in.
Master Nodes	3 Masters	Select the number of master nodes, also known as control plane nodes. These nodes are hosted on CCE and run Kubernetes components like kube-apiserver, kube-controller-manager, and kube-scheduler. • **Multiple**: Three master nodes will be created to ensure high availability. • **Single**: Only one master node will be created in your cluster. This setting cannot be changed after the cluster is created.

Step 3 Configure network parameters.

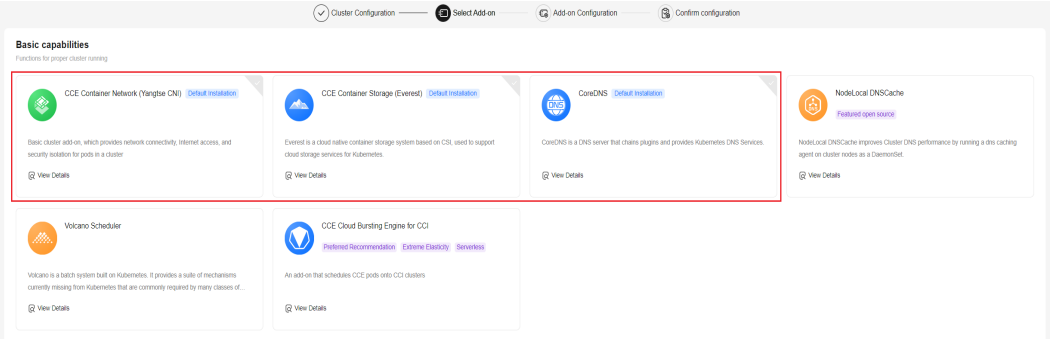


Parameter	Example	Description
VPC	vpc-001	Select a VPC for the cluster. If no VPC is available, click **Create VPC** to create one. After creating the VPC, click the refresh icon. For details, see **Creating a VPC and Subnet**.
Default Node Subnet	subnet-001	Select a subnet. Nodes in the cluster will be assigned IP addresses from this subnet.
Network Model	VPC network	Select **VPC network** or **Tunnel network**. The default value is **VPC network**. For details about the differences between container network models, see **Container Networks**.
Container CIDR Block	Manually set (10.0.0.0/16)	Specify the container CIDR block. The CIDR block determines how many containers you can deploy in the cluster. You can select **Manually set** or **Auto select**.
Pod IP Addresses Reserved for Each Node	128	Specify the number of allocatable container IP addresses (the **alpha.cce/fixPoolMask** parameter) on each node. This determines the maximum number of pods that can be created on each node.

Parameter	Example	Description
Service CIDR Block	10.247.0.0/16	Configure the cluster-wide IP address range. This controls how many Services can be created. This setting cannot be changed later.

Step 4 Click **Next: Select Add-on**. On the page displayed, select the add-ons to be installed during cluster creation.

This example only includes mandatory add-ons, which are installed automatically.



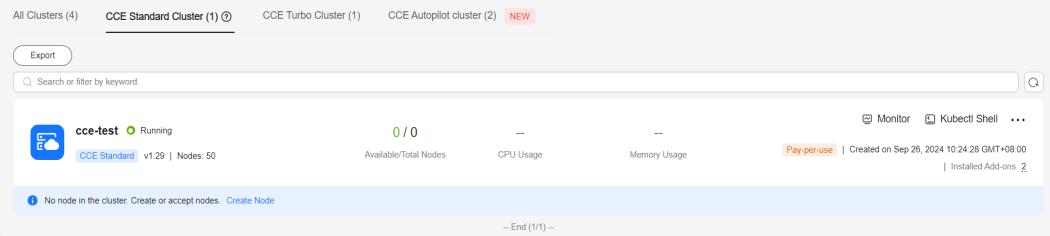
Step 5 Click **Next: Configure Add-on**. No setup is needed for the default add-ons.

Step 6 Click **Next: Confirm Settings**, check the displayed cluster resource list, and click **Submit**.

Wait for the cluster to be created. It takes approximately 5 to 10 minutes.

The newly created cluster in the **Running** state with zero nodes will be displayed on the **Clusters** page.

Figure 1-1 Cluster created



----End

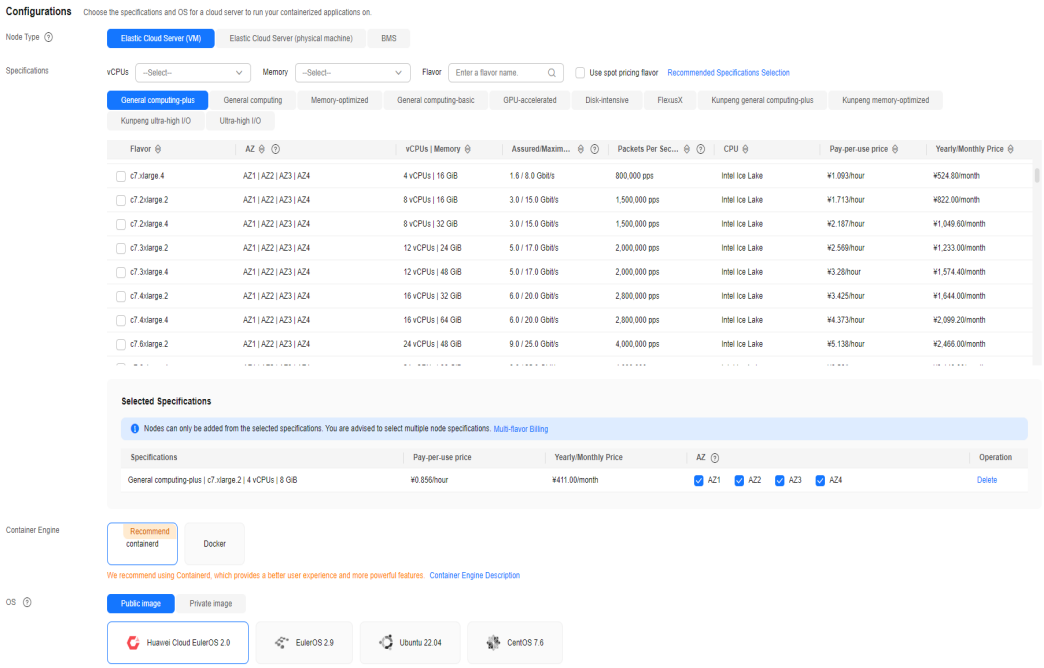
Step 3: Create a Node Pool and Nodes

Step 1 Log in to the CCE console and click the cluster name to access the cluster console.

Step 2 In the navigation pane, choose **Nodes**. On the **Node Pools** tab, click **Create Node Pool** in the upper right corner.

Step 3 Configure the node pool parameters.

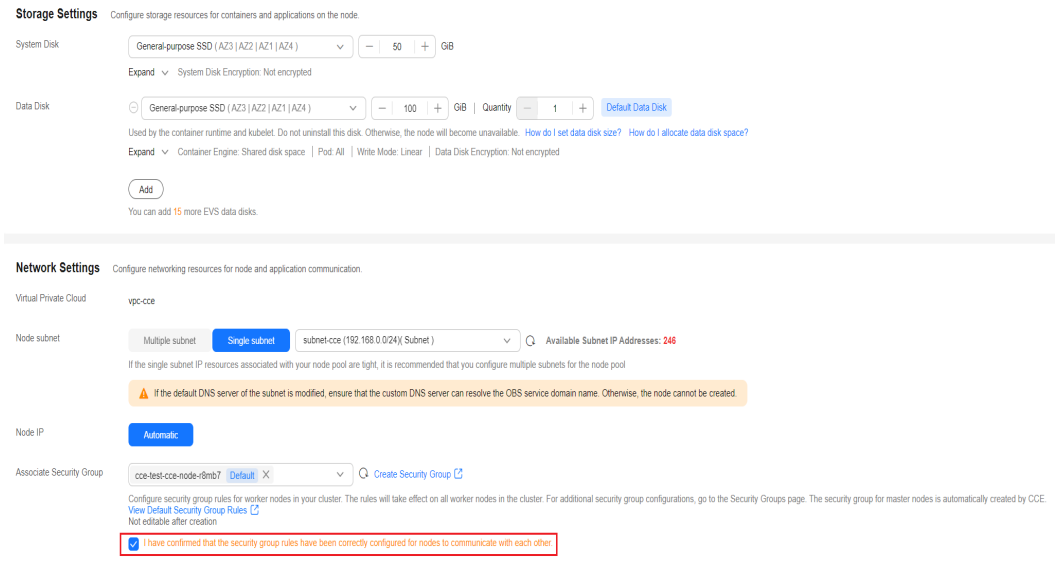
This example describes only mandatory parameters. You can keep default settings for most other parameters. For details, see **Creating a Node Pool**.



Parameter	Example	Description
Node Type	Elastic Cloud Server (VM)	Select a node type based on service requirements. The available node flavors will then be displayed in the **Specifications** area for you to choose from.
Specifications	General computing-plus 4 vCPUs | 8 GiB	Select a node flavor that best fits your service needs. • In this example, there are no specific requirements for memory or GPU resources. General computing-plus or general computing nodes are recommended. – General computing-plus nodes use dedicated vCPUs and next-generation network acceleration engines to provide strong compute and network performance. – General computing nodes provide a balance of compute, memory, and network resources and a baseline level of vCPU performance with the ability to burst above the baseline. • For optimal cluster component performance, choose a node with at least 4 vCPUs and 8 GiB of memory.

Parameter	Example	Description
Container Engine	containerd	Select a container engine based on service requirements. For details about the differences between container engines, see **Container Engines**.
OS	Huawei Cloud EulerOS 2.0	Select an OS for the nodes.
Login Mode	A custom password	• **Password**: Set and confirm a password for node login. The default username is **root**. Keep the password secure. It cannot be retrieved if forgotten. • **Key Pair**: Select a key pair for node login and confirm that you have the key file. Without this file, you will not be able to log in. Key pairs are used for identity authentication when you remotely access nodes. If you do not have a key pair, click **Create Key Pair**. For details, see **Creating a Key Pair on the Management Console**.

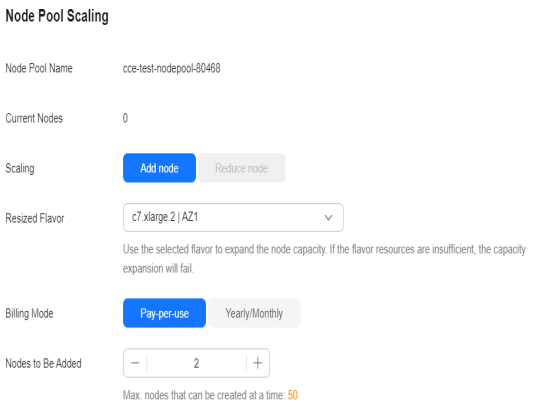
Step 4 Configure parameters in **Storage Settings** and **Network Settings**. In this example, keep the default settings. Select **I have confirmed that the security group rules have been correctly configured for nodes to communicate with each other.** and click **Next: Confirm**.



Step 5 Review the node specifications, read and confirm the instructions on the page, and click **Submit**.

Step 6 Locate the newly created node pool and click **Scaling**. The node pool initially contains zero nodes.

Step 7 Set the number of nodes to add to **2**. This will create two more nodes in the node pool.



Step 8 Wait for the nodes to be created. It takes approximately 5 to 10 minutes.

----End

Step 4: Create a Workload and Access It

You can deploy a workload using the console or **kubectl**. In this example, an Nginx image is used to create a workload.

Using the CCE Console

Step 1 In the navigation pane, choose **Workloads**. In the right pane, click **Create Workload** in the upper right corner.

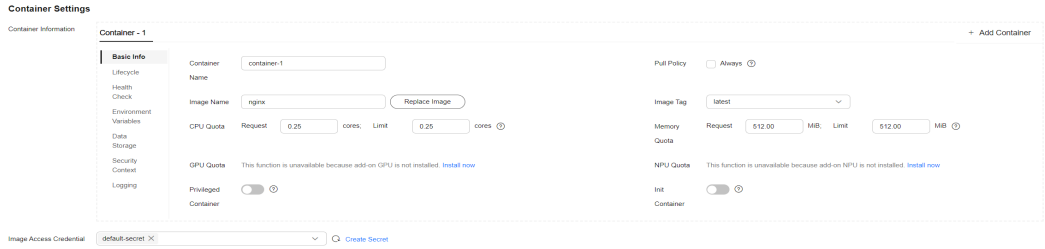
Step 2 Configure the basic information for the workload.

In this example, configure the parameters shown in the table and keep the default settings for other parameters. For more details, see **Creating a Deployment**.

Parameter	Example	Description
Workload Type	Deployment	A workload is an application running on Kubernetes. Various built-in workload types are available, each designed for different functions and scenarios. For more details, see **Workload Overview**.
Workload Name	nginx	Enter a workload name.
Namespace	default	In Kubernetes, a namespace is a conceptual grouping of resources or objects, providing isolation from other namespaces. After a cluster is created, a namespace named **default** is generated by default. You can use this namespace directly.
Pods	1	Specify the number of pods for the workload.

Step 3 Configure container parameters.

Configure the parameters shown in the table and keep the default settings for other parameters.

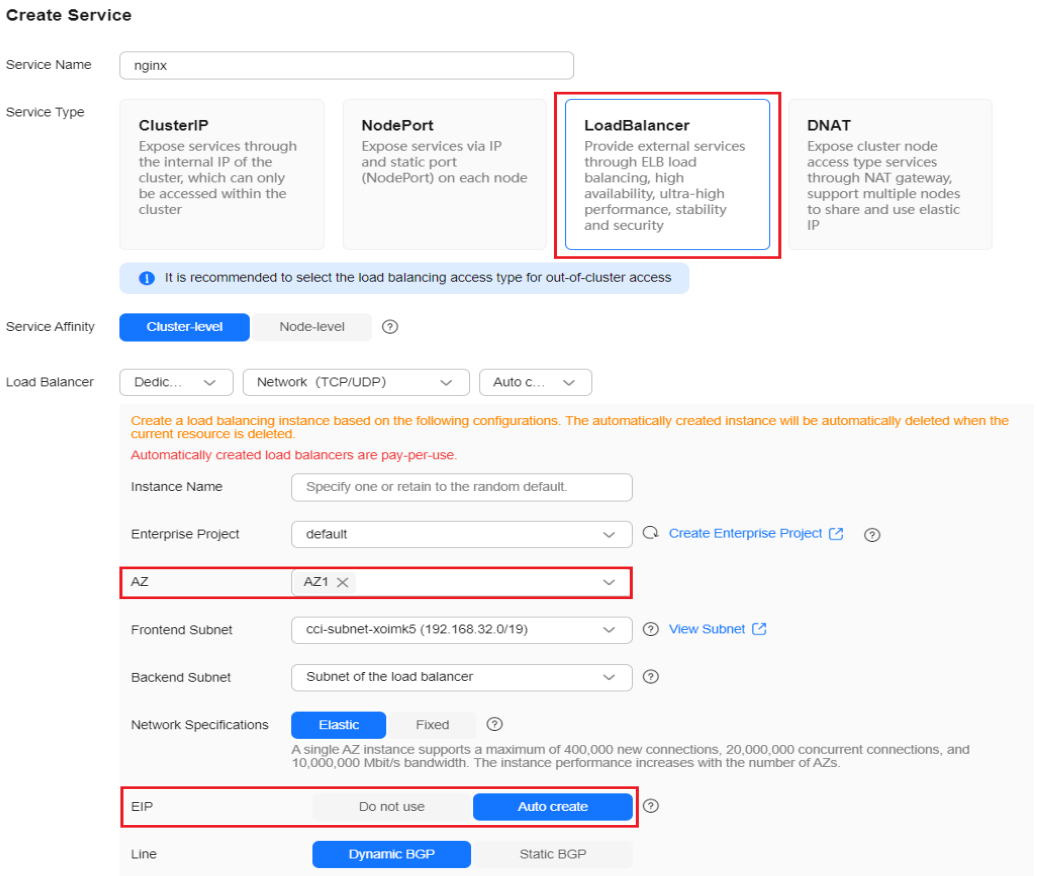


Parameter	Example	Description
Image Name	nginx (latest version)	Click **Select Image**. In the displayed dialog box, click the **Open Source Images** tab and select a public image.
CPU Quota	Request: 0.25 cores Limit: 0.25 cores	• **Request**: the number of CPU cores pre-allocated to containers. The default value is **0.25** cores. • **Limit**: the maximum number of CPU cores that containers can use. The default value is the same as the **Request**. If **Limit** exceeds **Request**, containers can temporarily use more resources in burst scenarios. For details, see **Configuring Container Specifications**.

Parameter	Example	Description
Memory Quota	Request: 512 MiB Limit: 512 MiB	• **Request**: the memory pre-allocated to containers. The default value is **512** MiB. • **Limit**: the maximum amount of memory that containers can use. The default value is the same as the **Request**. If **Limit** exceeds **Request**, containers can temporarily use more resources in burst scenarios. For details, see **Configuring Container Specifications**.

Step 4 Configure access settings.

In the **Service Settings** area, click the plus sign (+) and create a Service for external network access to the workload. This example shows how to create a LoadBalancer Service. Configure the below parameters in the sliding window on the right.



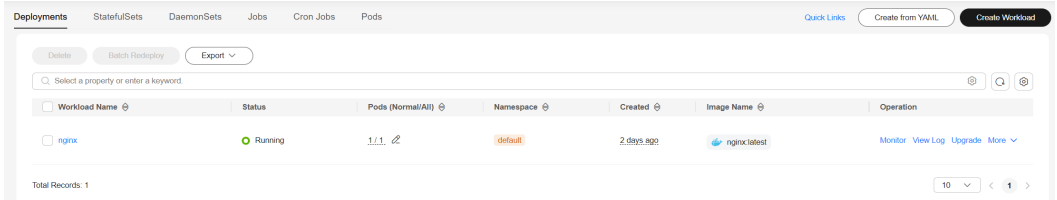
Parameter	Example	Description
Service Name	nginx	Specify the Service name.

Parameter	Example	Description
Service Type	LoadBalancer	Select a Service type. The Service type refers to the Service access mode. For details about the differences between Service types, see **Service Overview**.
Load Balancer	• Type: Dedicated • AZ: at least one AZ, for example, AZ1 • EIP: Auto create Keep the default settings for other parameters.	Select **Use existing** if a load balancer is available. If there is no load balancer available, select **Auto create** to create one and bind an EIP to it. For details, see **Creating a LoadBalancer Service**.
Ports	• Protocol: TCP • Container Port: 80 • Service Port: 8080	• **Protocol**: applies to the load balancer listener. • **Container Port**: The port that the containerized application listens on. This value must be the same as the listening port provided by the application for external access. If the **nginx** image is used, set this parameter to **80**. • **Service Port**: The port used by the load balancer to create a listener and provide an entry for external access. You can change this port if necessary.

Step 5 Click **Create Workload**.

Wait until the workload is created. The created workload in the **Running** state will be displayed on the **Deployments** tab.

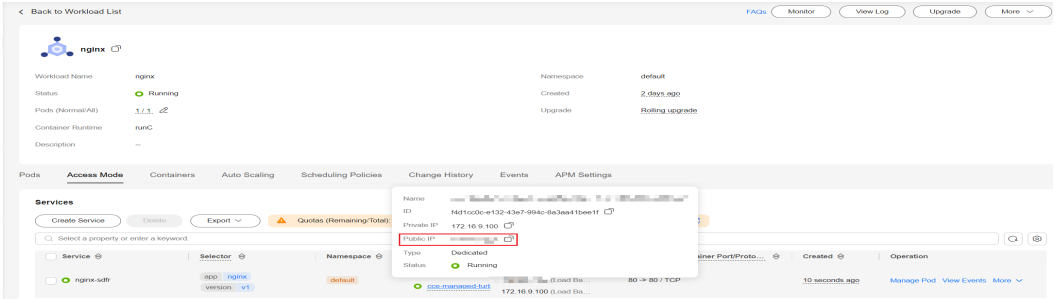
Figure 1-2 Workload created



Step 6 Obtain the external access address of Nginx.

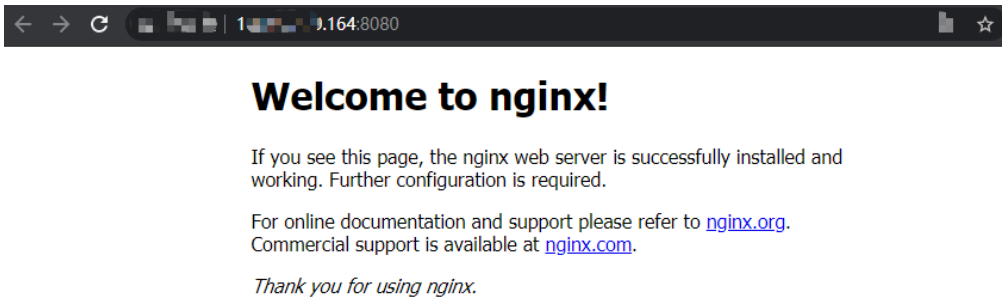
Click the name of the **nginx** workload to go to its details page. On the page displayed, click the **Access Mode** tab, view the Nginx IP address. The public IP address is the external access address.

Figure 1-3 Obtaining the external access address



Step 7 In the address bar of a browser, enter *<external-access-address>:service-port* to access the application. The value of *service-port* is the same as the service port specified in **Step 4**. In this example, it is **8080**.

Figure 1-4 Accessing Nginx



----End

Using kubectl

NOTICE

If you use kubectl to access the cluster, prepare an ECS **that has been bound with an EIP in the same VPC as the cluster**.

Step 1 Log in to the target ECS. For details, see **Login Overview (Linux)**.

Step 2 Install kubectl on the ECS.

You can check whether kubectl has been installed by running **kubectl version**. If kubectl has been installed, you can skip this step.

The Linux environment is used as an example to describe how to install and configure kubectl. For more installation methods, see **kubectl**.

1. Download kubectl.
   ```
   cd /home
   curl -LO https://dl.k8s.io/release/{v1.29.0}/bin/linux/amd64/kubectl
   ```
   {v1.29.0} specifies the version. You can replace it as required.
2. Install kubectl.
   ```
   chmod +x kubectl
   mv -f kubectl /usr/local/bin
   ```

Step 3 Configure a credential for kubectl to access the Kubernetes cluster.

1. Log in to the **CCE console** and click the cluster name to access the cluster console. Choose **Overview** in the navigation pane.
2. On the page displayed, locate the **Connection Information** area, click **Configure** next to **kubectl**, and view the kubectl connection information.
3. In the window that slides out from the right, locate the **Download the kubeconfig file** area, select **Intranet access** for **Current data**, and download the corresponding configuration file.
4. Log in to the VM where the kubectl client has been installed and copy and paste the configuration file (for example, **kubeconfig.yaml**) downloaded in the previous step to the **/home** directory.
5. Save the kubectl authentication file to the configuration file in the **$HOME/.kube** directory.
   ```
   cd /home
   mkdir -p $HOME/.kube
   mv -f kubeconfig.yaml $HOME/.kube/config
   ```
6. Run the kubectl command to see whether the cluster can be accessed.

   For example, to view the cluster information, run the following command:

   ```
   kubectl cluster-info
   ```

   Information similar to the following is displayed:

   ```
   Kubernetes master is running at https://*.*.*.*:5443
   CoreDNS is running at https://*.*.*.*:5443/api/v1/namespaces/kube-system/services/coredns:dns/proxy
   To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
   ```

Step 4 Create a YAML file named **nginx-deployment.yaml**. **nginx-deployment.yaml** is an example file name. You can rename it as required.

```
vi nginx-deployment.yaml
```

The file content is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:alpine
        name: nginx
      imagePullSecrets:
      - name: default-secret
```


Step 5 Run the following command to deploy the workload:

```
kubectl create -f nginx-deployment.yaml
```

If information similar to the following is displayed, the workload is being created:

```
deployment "nginx" created
```

Step 6 Run the following command to check the workload status:

```
kubectl get deployment
```

If information similar to the following is displayed, the workload has been created:

```
NAME         READY    UP-TO-DATE   AVAILABLE   AGE
nginx        1/1      1            1           4m5s
```

The parameters in the command output are described as follows:

- **NAME**: specifies the name of a workload.
- **READY**: indicates the number of available pods/expected pods for the workload.
- **UP-TO-DATE**: specifies the number of pods that have been updated for the workload.
- **AVAILABLE**: specifies the number of pods available for the workload.
- **AGE**: specifies how long the workload has run.

Step 7 Create a YAML file named **nginx-elb-svc.yaml** and change the value of **selector** to that of **matchLabels** (**app: nginx** in this example) in the **nginx-deployment.yaml** file to associate the Service with the backend application.

```
vi nginx-elb-svc.yaml
```

An example is shown below. For details about the parameters in the following example, see **Using kubectl to Create a Service (Automatically Creating a Load Balancer)**.

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate:
        '{
            "type": "public",
            "bandwidth_name": "cce-bandwidth",
            "bandwidth_chargemode": "bandwidth",
            "bandwidth_size": 5,
            "bandwidth_sharetype": "PER",
            "eip_type": "5_bgp"
        }'
  labels:
    app: nginx
  name: nginx
spec:
  ports:
  - name: service0
    port: 8080
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

Step 8 Run the following command to create the Service:

```
kubectl create -f nginx-elb-svc.yaml
```


If information similar to the following is displayed, the Service has been created:

```
service/nginx created
```

Step 9 Run the following command to check the Service:

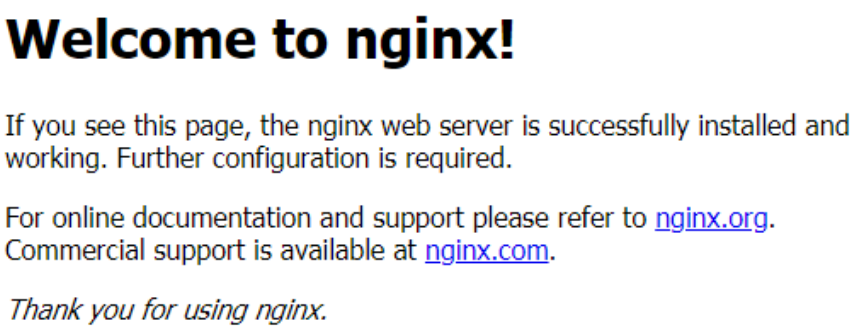
```
kubectl get svc
```

If information similar to the following is displayed, the access type has been configured, and the workload is accessible:

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
kubernetes   ClusterIP      10.247.0.1      <none>          443/TCP          3d
nginx        LoadBalancer   10.247.130.196  **.**.**.**     8080:31540/TCP   51s
```

Step 10 Enter the URL (for example, ****.**.**.**:8080**) in the address bar of a browser. ****.**.**.**** specifies the EIP of the load balancer, and **8080** indicates the access port.

Figure 1-5 Accessing Nginx using the LoadBalancer Service



----End

Follow-up Operations: Release Resources

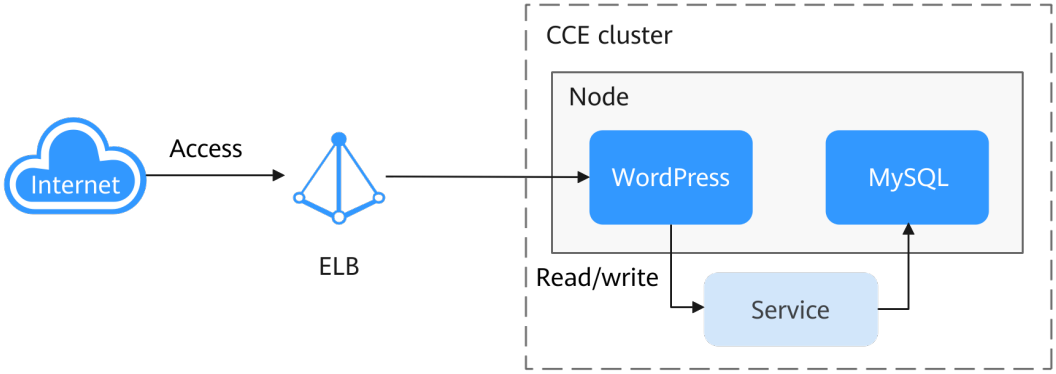
Release resources promptly if the cluster is no longer needed to avoid extra charges. For details, see **Deleting a Cluster**.

2 Deploying a WordPress StatefulSet in a CCE Cluster

StatefulSets are a type of workload in Kubernetes. They are designed to manage stateful applications. Unlike Deployments, StatefulSets are ideal for applications that require data consistency and durability. Each StatefulSet pod has its own unique identifier and must be deployed and scaled in a specific sequence. Examples of stateful applications include databases (like MySQL) and message queues (such as Kafka). This section uses WordPress, a popular content management system, and a MySQL database, as an example to describe how to deploy a StatefulSet in a CCE cluster.

WordPress started as a blogging platform using PHP and MySQL, but it has evolved into a complete content management system. You can use a CCE cluster to quickly set up your own blog. For more information about WordPress, see the **WordPress official website**.

WordPress and a database (a MySQL database in this example) are often used together, with WordPress managing content and the database storing website data. In a containerized deployment, WordPress and MySQL typically run in separate containers. WordPress accesses MySQL through a Service.



Video Tutorial

Procedure

Step	Description
Preparations	Sign up for a HUAWEI ID.
Step 1: Enable CCE and Perform Authorization	Obtain the required permissions for your account when you use CCE in the deployment region for the first time.
Step 2: Create a Cluster	Create a CCE cluster to provide Kubernetes services.
Step 3: Create a Node Pool and Nodes	Create nodes in the cluster to run containerized applications.
Step 4: Deploy MySQL	Create a MySQL workload in the cluster and create a ClusterIP Service for WordPress access.
Step 5: Deploy WordPress	Create a WordPress workload in the cluster and create a LoadBalancer Service for the workload for Internet access.
Step 6: Access WordPress	Access the WordPress website from the Internet to start your blog.
Follow-up Operations: Release Resources	Release resources promptly if the cluster is no longer needed to avoid extra charges.

Preparations

- Before you start, sign up for a HUAWEI ID. For details, see **Signing up for a HUAWEI ID and Enabling Huawei Cloud Services**.

Step 1: Enable CCE and Perform Authorization

When you first log in to the CCE console, CCE automatically requests permissions to access related cloud services (compute, storage, networking, and monitoring) in the region where the cluster is deployed. If you have authorized CCE in the deployment region, skip this step.

Step 1 Log in to the **CCE console** using your HUAWEI ID.

Step 2 Click  in the upper left corner and select a region.

Step 3 If this is your first login to the CCE console in the selected region, the **Authorization Statement** dialog box will appear. Read it carefully and click **OK**.

After you agree to delegate permissions, CCE uses IAM to create an agency named **cce_admin_trust**. This agency is granted **Tenant Administrator** permissions for the resources of other cloud services (excluding IAM). These permissions are required for CCE to access dependent cloud services and are only valid for the current region. You can view the authorization records in each region by navigating to the IAM console, choosing **Agencies** in the navigation pane, and clicking **cce_admin_trust**. For more details, see **Cloud Service Agency**.

To ensure CCE can run normally, do not delete or modify the **cce_admin_trust** agency, as CCE requires **Tenant Administrator** permissions.

NOTE

CCE has updated the **cce_admin_trust** agency permissions to enhance security while accommodating dependencies on other cloud services. The new permissions no longer include **Tenant Administrator** permissions. This update is only available in certain regions. If your clusters are of v1.21 or later, a message will appear on the console asking you to re-grant permissions. After re-granting, the **cce_admin_trust** agency will be updated to include only the necessary cloud service permissions, with the **Tenant Administrator** permissions removed.

When creating the **cce_admin_trust** agency, CCE creates a custom policy named **CCE admin policies**. Do not delete this policy.

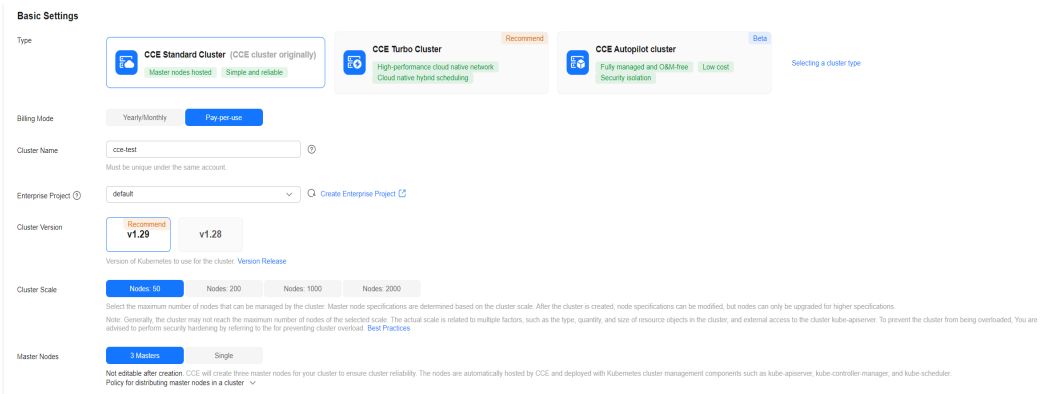
----End

Step 2: Create a Cluster

Step 1 Log in to the **CCE console** and click **Buy Cluster**.

Step 2 Configure basic cluster parameters.

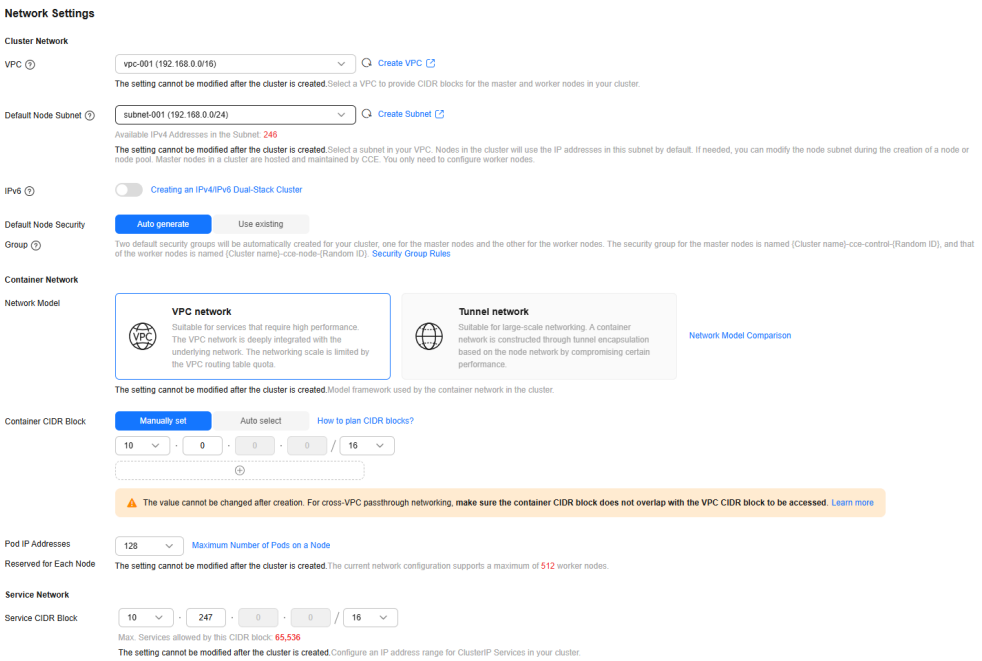
This example describes only mandatory parameters. You can keep default settings for most other parameters. For details, see **Buying a CCE Standard/Turbo Cluster**.



Parameter	Example	Description
Type	CCE Standard Cluster	CCE supports multiple cluster types to meet diverse needs, including highly reliable, highly secure, commercial containers. You can select **CCE Standard Cluster** or **CCE Turbo Cluster** as required. • CCE standard clusters provide highly reliable, highly secure, commercial containers. • CCE Turbo clusters use high-performance cloud native networks and support cloud native hybrid scheduling. These clusters offer improved resource utilization and are suitable for a wider range of scenarios. For more details, see **Comparison Between Cluster Types**.
Billing Mode	Pay-per-use	Select a billing mode for the cluster. • **Yearly/Monthly**: a prepaid billing mode. Resources are billed based on how long you will need the resources for. This mode is more cost-effective when resource usage periods are predictable. If you choose this option, select the **desired duration** and decide whether to enable automatic renewal (monthly or yearly). Monthly subscriptions renew automatically every month, and yearly subscriptions renew automatically every year. • **Pay-per-use**: a postpaid billing mode. Resources are billed based on actual usage duration. This mode is suitable for flexible scenarios where you may need to provision or delete resources at any time. For more details, see **Billing Modes**.
Cluster Name	cce-test	Enter a name for the cluster.
Enterprise Project	default	Enterprise projects facilitate project-level management and grouping of cloud resources and users. This parameter is only displayed for enterprise users who have enabled enterprise projects.
Cluster Version	v1.29 (recommended)	Select the latest Kubernetes commercial release to benefit from new, reliable, and production-ready features. CCE offers multiple Kubernetes versions.

Parameter	Example	Description
Cluster Scale	Nodes: 50	This parameter controls the maximum number of worker nodes the cluster can manage. Configure it as needed. After the cluster is created, it can be scaled out, but it cannot be scaled in.
Master Nodes	3 Masters	Select the number of master nodes, also known as control plane nodes. These nodes are hosted on CCE and run Kubernetes components like kube-apiserver, kube-controller-manager, and kube-scheduler. • **Multiple**: Three master nodes will be created to ensure high availability. • **Single**: Only one master node will be created in your cluster. This setting cannot be changed after the cluster is created.

Step 3 Configure network parameters.

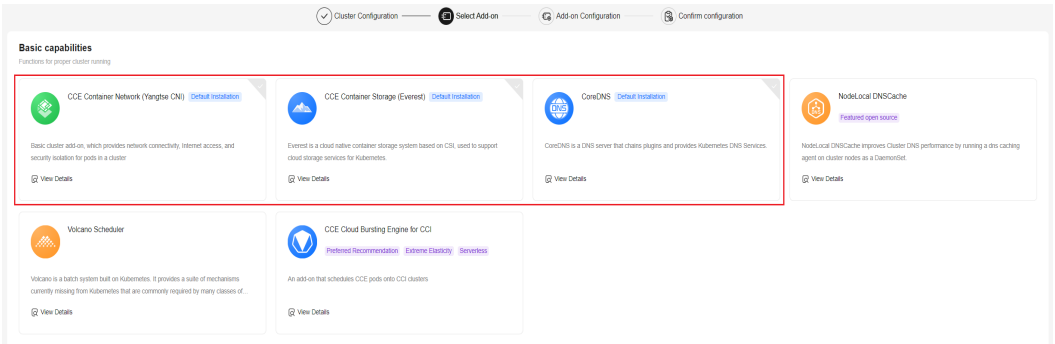


Parameter	Example	Description
VPC	vpc-001	Select a VPC for the cluster. If no VPC is available, click **Create VPC** to create one. After creating the VPC, click the refresh icon. For details, see **Creating a VPC and Subnet**.

Parameter	Example	Description
Default Node Subnet	subnet-001	Select a subnet. Nodes in the cluster will be assigned IP addresses from this subnet.
Network Model	VPC network	Select **VPC network** or **Tunnel network**. The default value is **VPC network**. For details about the differences between container network models, see **Container Networks**.
Container CIDR Block	Manually set (10.0.0.0/16)	Specify the container CIDR block. The CIDR block determines how many containers you can deploy in the cluster. You can select **Manually set** or **Auto select**.
Pod IP Addresses Reserved for Each Node	128	Specify the number of allocatable container IP addresses (the **alpha.cce/fixPoolMask** parameter) on each node. This determines the maximum number of pods that can be created on each node.
Service CIDR Block	10.247.0.0/16	Configure the cluster-wide IP address range. This controls how many Services can be created. This setting cannot be changed later.

Step 4 Click **Next: Select Add-on**. On the page displayed, select the add-ons to be installed during cluster creation.

This example only includes mandatory add-ons, which are installed automatically.



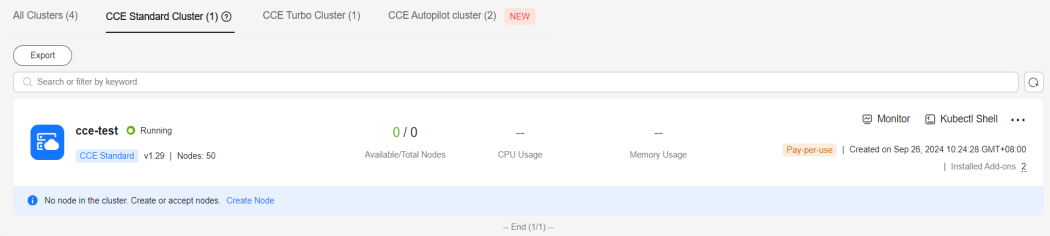
Step 5 Click **Next: Configure Add-on**. No setup is needed for the default add-ons.

Step 6 Click **Next: Confirm Settings**, check the displayed cluster resource list, and click **Submit**.

Wait for the cluster to be created. It takes approximately 5 to 10 minutes.

The newly created cluster in the **Running** state with zero nodes will be displayed on the **Clusters** page.

Figure 2-1 Cluster created



----End

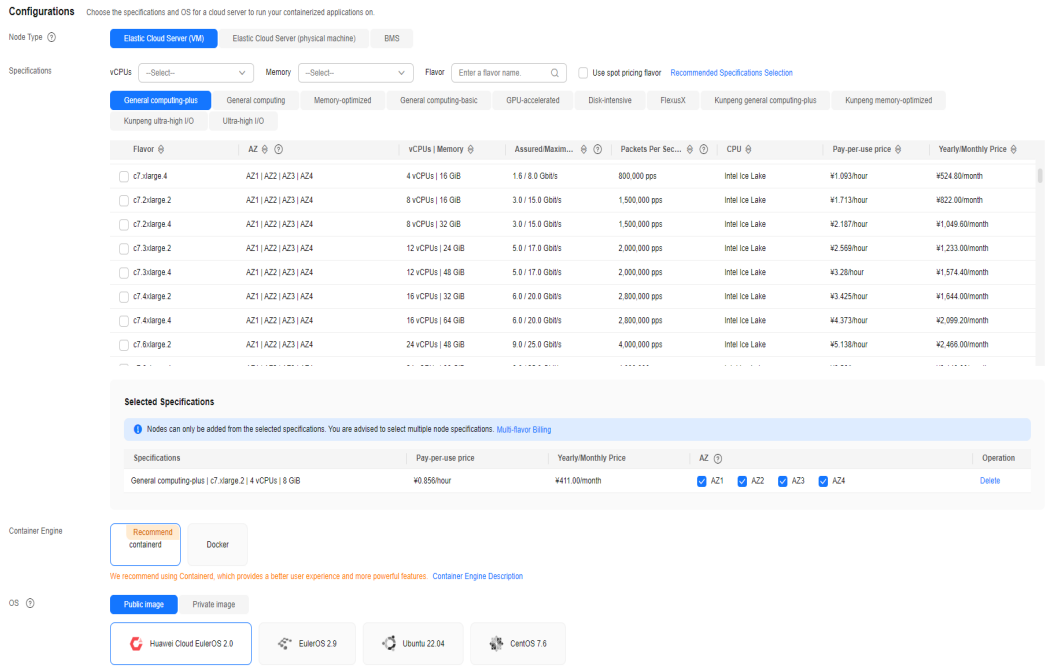
Step 3: Create a Node Pool and Nodes

Step 1 Log in to the CCE console and click the cluster name to access the cluster console.

Step 2 In the navigation pane, choose **Nodes**. On the **Node Pools** tab, click **Create Node Pool** in the upper right corner.

Step 3 Configure the node pool parameters.

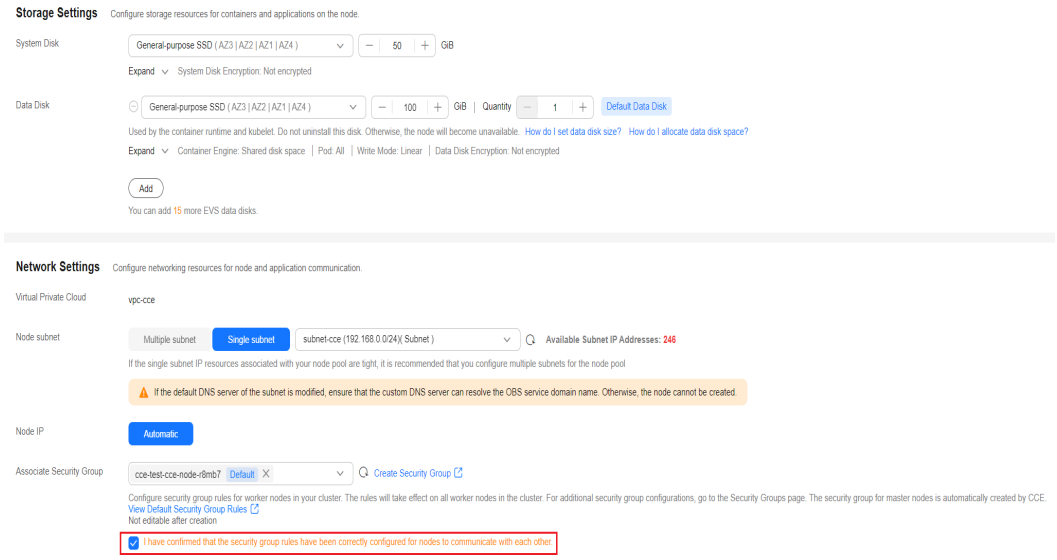
This example describes only mandatory parameters. You can keep default settings for most other parameters. For details, see **Creating a Node Pool**.



Parameter	Example	Description
Node Type	Elastic Cloud Server (VM)	Select a node type based on service requirements. The available node flavors will then be displayed in the **Specifications** area for you to choose from.

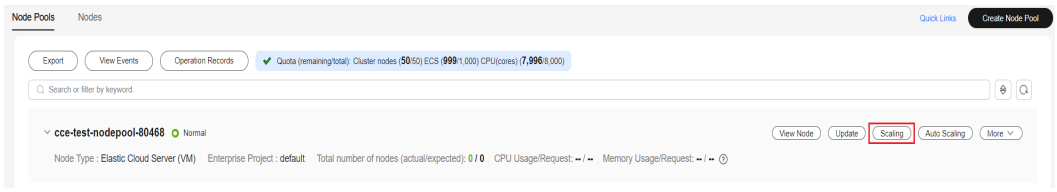
Parameter	Example	Description
Specifications	General computing-plus 4 vCPUs | 8 GiB	Select a node flavor that best fits your service needs. • In this example, there are no specific requirements for memory or GPU resources. General computing-plus or general computing nodes are recommended. – General computing-plus nodes use dedicated vCPUs and next-generation network acceleration engines to provide strong compute and network performance. – General computing nodes provide a balance of compute, memory, and network resources and a baseline level of vCPU performance with the ability to burst above the baseline. • For optimal cluster component performance, choose a node with at least 4 vCPUs and 8 GiB of memory.
Container Engine	containerd	Select a container engine based on service requirements. For details about the differences between container engines, see **Container Engines**.
OS	Huawei Cloud EulerOS 2.0	Select an OS for the nodes.
Login Mode	A custom password	• **Password**: Set and confirm a password for node login. The default username is **root**. Keep the password secure. It cannot be retrieved if forgotten. • **Key Pair**: Select a key pair for node login and confirm that you have the key file. Without this file, you will not be able to log in. Key pairs are used for identity authentication when you remotely access nodes. If you do not have a key pair, click **Create Key Pair**. For details, see **Creating a Key Pair on the Management Console**.

Step 4 Configure parameters in **Storage Settings** and **Network Settings**. In this example, keep the default settings. Select **I have confirmed that the security group rules have been correctly configured for nodes to communicate with each other.** and click **Next: Confirm**.

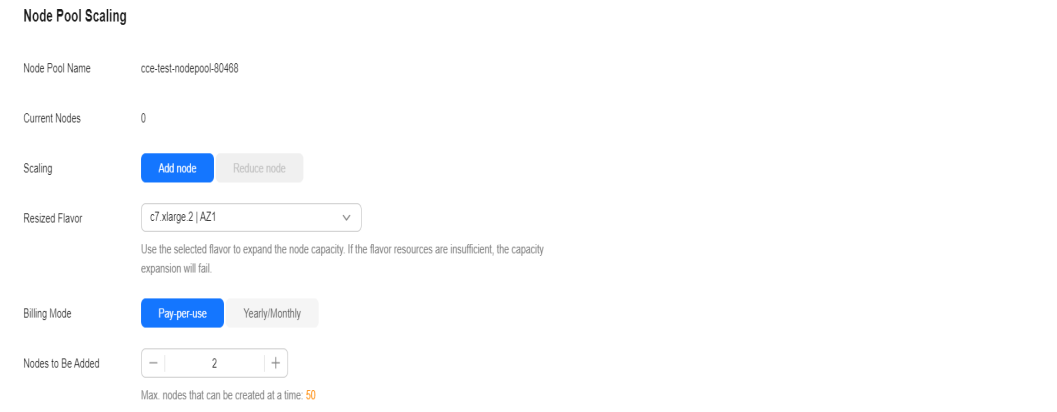


Step 5 Review the node specifications, read and confirm the instructions on the page, and click **Submit**.

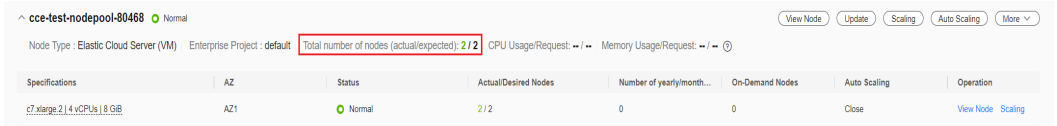
Step 6 Locate the newly created node pool and click **Scaling**. The node pool initially contains zero nodes.



Step 7 Set the number of nodes to add to **2**. This will create two more nodes in the node pool.



Step 8 Wait for the nodes to be created. It takes approximately 5 to 10 minutes.



----End

Step 4: Deploy MySQL

You can deploy a MySQL workload in different ways.

Using the CCE Console

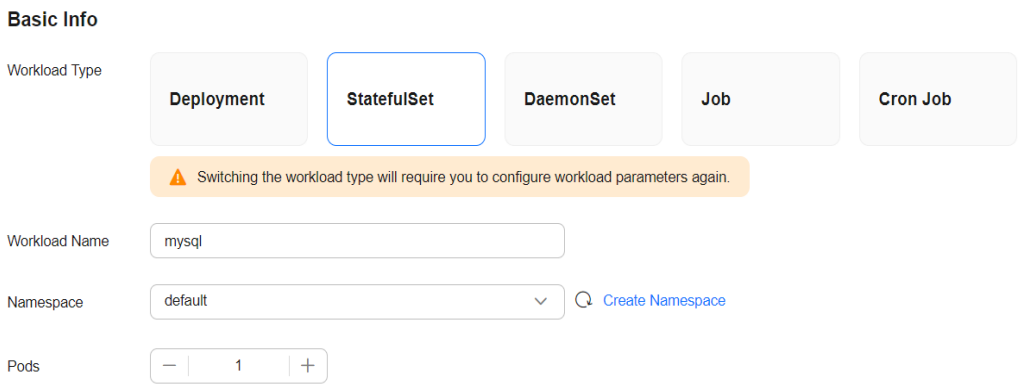
Step 1 Log in to the **CCE console**.

Step 2 Click the name of the cluster to access the cluster console.

Step 3 In the navigation pane, choose **Workloads**. In the right pane, click **Create Workload** in the upper right corner.

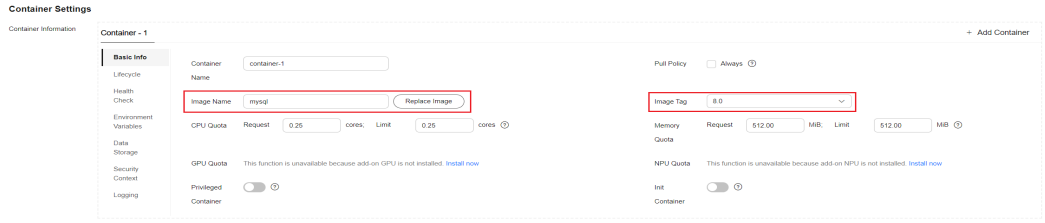
Step 4 Configure the basic information for the workload.

In this example, configure the parameters shown in the table and keep the default settings for other parameters. For more details, see **Creating a StatefulSet**.



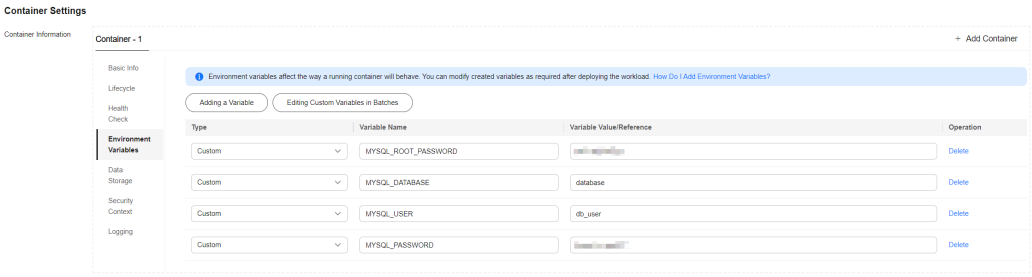
Parameter	Example	Description
Workload Type	StatefulSet	A workload is an application running on Kubernetes. Various built-in workload types are available, each designed for different functions and scenarios. For more details, see **Workload Overview**.
Workload Name	mysql	Enter a workload name.
Namespace	default	In Kubernetes, a namespace is a conceptual grouping of resources or objects, providing isolation from other namespaces. After a cluster is created, a namespace named **default** is generated by default. You can use this namespace directly.
Pods	1	Specify the number of pods for the workload.

Step 5 Configure the basic information about the container.



Parameter	Example	Description
Image Name	A MySQL image of tag 8.0	In the **Container Settings** area, click **Basic Info** and click **Select Image**. In the dialog box displayed, select **Open Source Images**, search for **mysql**, select the **mysql** image, and configure the image tag. **To ensure compatibility, set the image tag to 8.0.** The compatibility between other tags and WordPress has not been verified. Any other tags may create compatibility issues with WordPress. For details, see **FAQs**.
CPU Quota	Request: 0.25 cores Limit: 0.25 cores	• **Request**: the number of CPU cores pre-allocated to containers. The default value is **0.25** cores. • **Limit**: the maximum number of CPU cores that containers can use. The default value is the same as the **Request**. If **Limit** exceeds **Request**, containers can temporarily use more resources in burst scenarios. For details, see **Configuring Container Specifications**.
Memory Quota	Request: 512 MiB Limit: 512 MiB	• **Request**: the memory pre-allocated to containers. The default value is **512** MiB. • **Limit**: the maximum amount of memory that containers can use. The default value is the same as the **Request**. If **Limit** exceeds **Request**, containers can temporarily use more resources in burst scenarios. For details, see **Configuring Container Specifications**.

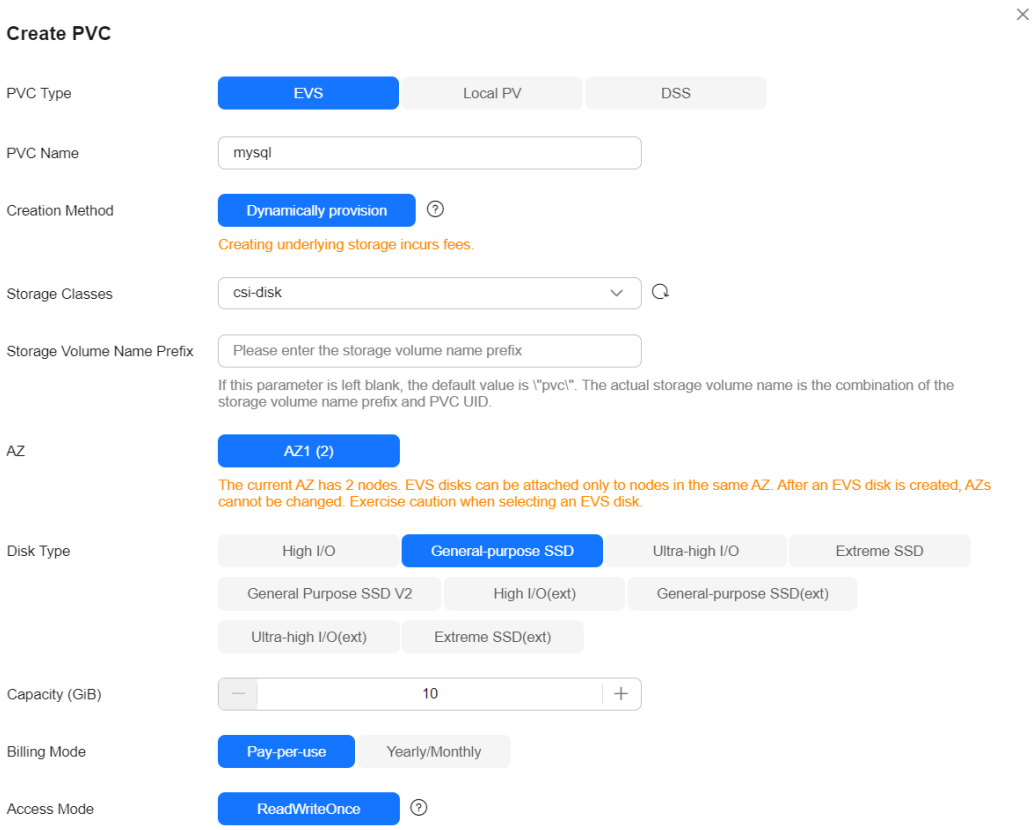
Step 6 Click **Environment Variables** and add the four required environment variables. For details about the environment variables supported by MySQL, see **MySQL**.



Environment Variable	Example	Description
MYSQL_ROOT_PASSWORD	**A custom password**	The password of the **root** user of the MySQL database. You configure your own password.
MYSQL_DATABASE	database	The name of the database to be created when the image is started. You configure your own database name.
MYSQL_USER	db_user	The database username. You configure your own username.
MYSQL_PASSWORD	A custom password	The database user password. You configure your own password.

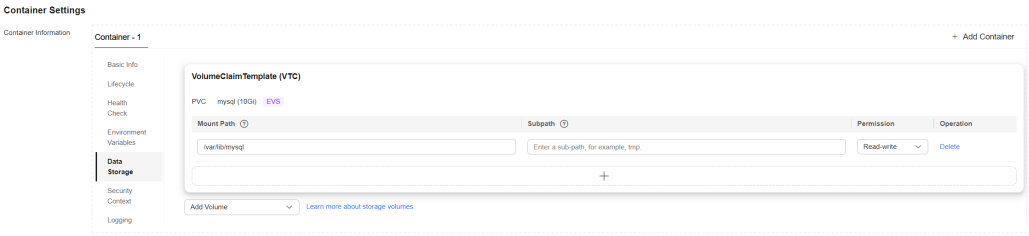
Step 7 Click **Data Storage**, click **Add Volume**, select **VolumeClaimTemplate (VTC)** from the drop-down list, and add an EVS disk for MySQL.

Click **Create PVC**, configure the parameters shown here, and keep the default values for other parameters.



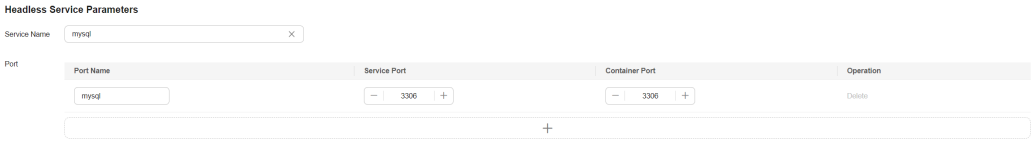
Parameter	Example	Description
PVC Type	EVS	Select a type for the underlying storage volume used by the PersistentVolumeClaim (PVC).
PVC Name	mysql	Enter a custom PVC name, for example, **mysql**.
Storage Classes	csi-disk	The default value is **csi-disk**.
AZ	AZ1	Select an AZ. The EVS disk can only be attached to nodes in the same AZ. After an EVS disk is created, the AZ where the disk locates cannot be changed.
Disk Type	General-purpose SSD	Select a proper type as required.
Capacity (GiB)	10	Enter the capacity required. The default capacity is 10 GiB.

Click **Create** and enter the path for mounting the storage volume to the container. The default path used by MySQL is **/var/lib/mysql**.



Step 8 In the **Headless Service Parameters** area, configure a headless Service.

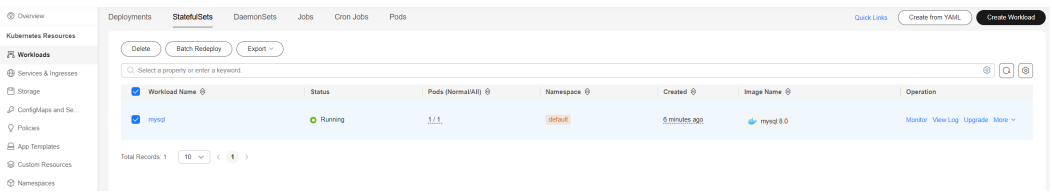
The headless Service is used for networking between StatefulSet pods. It generates a domain name for each pod for accessing a specific StatefulSet pod. For a MySQL database that has primary/secondary relationship and multiple replicas, a headless Service is needed to read and write data from and into the MySQL database server (known as a source) and copy the data to other replicas. In this example, MySQL is deployed in a single pod. For details about how to deploy MySQL in multiple pods, see **Run a Replicated Stateful Application**.



Parameter	Example	Description
Service Name	mysql	Enter a custom headless Service name.
Port Name	mysql	Enter a custom port name. The port name is used to distinguish different ports in a given Service. In this example, only one port is used.
Service Port	3306	Enter a custom port number. This port is used by the Service for external access. In this example, the port is the same as the container port.
Container Port	3306	The actual listening port of the application in the container. It is determined by the port opened by the application image. For example, the MySQL database open port is 3306.

Step 9 Click **Create Workload**.

Wait until the workload is created. After it is created, it will be displayed on the **StatefulSets** tab.



----End

Using kubectl

NOTICE

You need to create an ECS **bound with an EIP in the same VPC as the cluster** first.

Step 1 Install kubectl on the ECS.

You can check whether kubectl has been installed by running **kubectl version**. If kubectl has been installed, you can skip this step.

The Linux environment is used as an example to describe how to install and configure kubectl. For more installation methods, see **kubectl**.

1. Download kubectl.
   ```
   cd /home
   curl -LO https://dl.k8s.io/release/{v1.29.0}/bin/linux/amd64/kubectl
   ```
   {v1.29.0} specifies the version. You can replace it as required.
2. Install kubectl.
   ```
   chmod +x kubectl
   mv -f kubectl /usr/local/bin
   ```

Step 2 Configure a credential for kubectl to access the Kubernetes cluster.

1. Log in to the **CCE console** and click the cluster name to access the cluster console. Choose **Overview** in the navigation pane.
2. On the page displayed, locate the **Connection Information** area, click **Configure** next to **kubectl**, and view the kubectl connection information.
3. In the window that slides out from the right, locate the **Download the kubeconfig file** area, select **Intranet access** for **Current data**, and download the corresponding configuration file.
4. Log in to the VM where the kubectl client has been installed and copy and paste the configuration file (for example, **kubeconfig.yaml**) downloaded in the previous step to the **/home** directory.
5. Save the kubectl authentication file to the configuration file in the **$HOME/.kube** directory.
   ```
   cd /home
   mkdir -p $HOME/.kube
   mv -f kubeconfig.yaml $HOME/.kube/config
   ```
6. Run the kubectl command to see whether the cluster can be accessed.

   For example, to view the cluster information, run the following command:
   ```
   kubectl cluster-info
   ```
   Information similar to the following is displayed:
   ```
   Kubernetes master is running at https://*.*.*.*:5443
   CoreDNS is running at https://*.*.*.*:5443/api/v1/namespaces/kube-system/services/coredns:dns/proxy
   To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
   ```

Step 3 Create a description file named **mysql.yaml**. **mysql.yaml** is an example file name. You can rename it as required.


```
vi mysql.yaml
```

The file content is as follows:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
      version: v1
  template:
    metadata:
      labels:
        app: mysql
        version: v1
    spec:
      containers:
        - name: container-1
          image: mysql:8.0
          env:
            - name: MYSQL_ROOT_PASSWORD  # The password of the root user of the MySQL database. You configure your own password.
              value: *****
            - name: MYSQL_DATABASE       # The name of the database to be created when the image is started. You configure your own database name.
              value: database
            - name: MYSQL_USER           # The database username. You configure your own username.
              value: db_user
            - name: MYSQL_PASSWORD       # The database user password. You configure your own password.
              value: *****
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
          volumeMounts:
            - name: mysql
              mountPath: /var/lib/mysql
      imagePullSecrets:
        - name: default-secret
  serviceName: mysql     # The name of the headless Service, which must be the same as that defined in the headless Service
  volumeClaimTemplates:  # Dynamically attach an EVS disk to the workload.
    - apiVersion: v1
      kind: PersistentVolumeClaim
      metadata:
        name: mysql
        namespace: default
        annotations:
          everest.io/disk-volume-type: SSD  # EVS disk type
        labels:
          failure-domain.beta.kubernetes.io/region: eu-west-101  # Region where the EVS disk is in
          failure-domain.beta.kubernetes.io/zone:    # AZ where the EVS disk is in. It must be the same as the AZ of the node that runs the workload.
      spec:
        accessModes:
          - ReadWriteOnce  # ReadWriteOnce for an EVS disk
        resources:
          requests:
            storage: 10Gi
        storageClassName: csi-disk  # Storage class name. The value is csi-disk for an EVS disk.

---
```



```
# Define the headless Service.
apiVersion: v1
kind: Service
metadata:
  name: mysql         #  Name of the headless Service, which must be the same as the name of the Service mounted to the workload
  namespace: default
  labels:
    app: mysql
    version: v1
spec:
  selector:           # Label selector, which must be the same as the labels of the workload. It is used to match the desired workload.
    app: mysql
    version: v1
  clusterIP: None
  ports:
    - name: mysql      # The port name. You configure your own port name.
      protocol: TCP
      port: 3306       # The Service port. You configure your own Service port.
      targetPort: 3306 # The container port. The value is fixed to 3306 for MySQL.
  type: ClusterIP
```

Step 4 Create a MySQL workload.

```
kubectl apply -f mysql.yaml
```

If information similar to the following is displayed, the workload is being created:

```
statefulset.apps/mysql created
service/mysql created
```

Step 5 Check the workload status.

```
kubectl get statefulset
```

If information similar to the following is displayed, the workload has been created:

```
NAME          READY      AGE
mysql         1/1        4m5s
```

Step 6 Check the Service.

```
kubectl get svc
```

If information similar to what is shown here is displayed, the workload's access mode has been configured.

```
NAME         TYPE          CLUSTER-IP       EXTERNAL-IP    PORT(S)          AGE
kubernetes   ClusterIP     10.247.0.1       <none>         443/TCP          3d
mysql        ClusterIP     None             <none>         3306/TCP         51s
```

----End

Step 5: Deploy WordPress

You can deploy a WordPress workload using either of the following ways.

Using the CCE Console

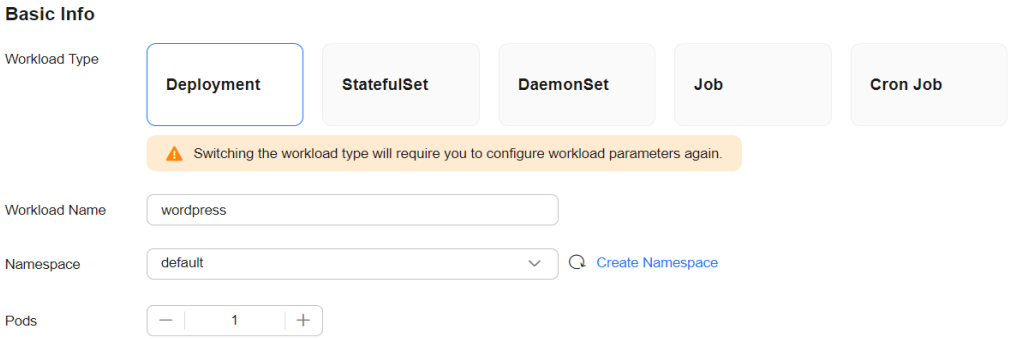
Step 1 Log in to the **CCE console**.

Step 2 Click the name of the cluster to access the cluster console.

Step 3 In the navigation pane, choose **Workloads**. In the right pane, click **Create Workload** in the upper right corner.

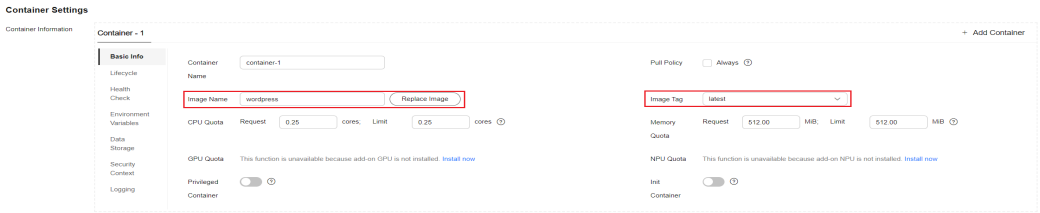
Step 4 Configure the basic information for the workload.

In this example, configure the parameters shown in the table and keep the default settings for other parameters. For more details, see **Creating a StatefulSet**.



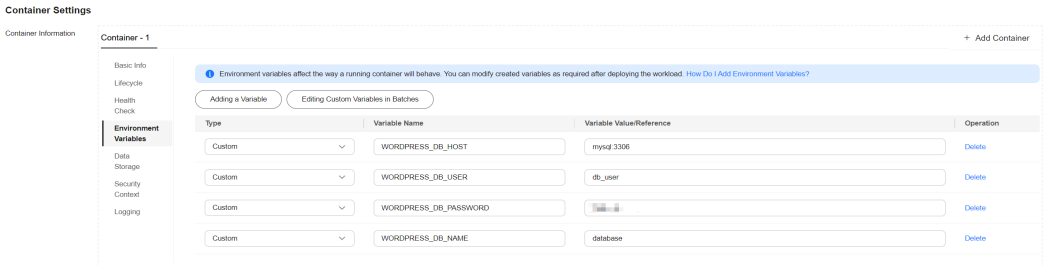
Parameter	Example	Description
Workload Type	Deployment	A workload is an application running on Kubernetes. Various built-in workload types are available, each designed for different functions and scenarios. For more details, see **Workload Overview**.
Workload Name	wordpress	Enter a workload name.
Namespace	default	In Kubernetes, a namespace is a conceptual grouping of resources or objects, providing isolation from other namespaces. After a cluster is created, a namespace named **default** is generated by default. You can use this namespace directly.
Pods	1	Specify the number of pods for the workload.

Step 5 Configure the basic information about the container.



Parameter	Example	Description
Image Name	The wordpress image of the latest version	In the **Container Information** area, click **Basic Info** and click **Select Image**. In the dialog box displayed, select **Open Source Images**, search for **wordpress**, select the **wordpress** image, and select **latest** from the drop-down list for **Image Tag**.
CPU Quota	Request: 0.25 cores Limit: 0.25 cores	• **Request**: the number of CPU cores pre-allocated to containers. The default value is **0.25** cores. • **Limit**: the maximum number of CPU cores that containers can use. The default value is the same as the **Request**. If **Limit** exceeds **Request**, containers can temporarily use more resources in burst scenarios. For details, see **Configuring Container Specifications**.
Memory Quota	Request: 512 MiB Limit: 512 MiB	• **Request**: the memory pre-allocated to containers. The default value is **512** MiB. • **Limit**: the maximum amount of memory that containers can use. The default value is the same as the **Request**. If **Limit** exceeds **Request**, containers can temporarily use more resources in burst scenarios. For details, see **Configuring Container Specifications**.

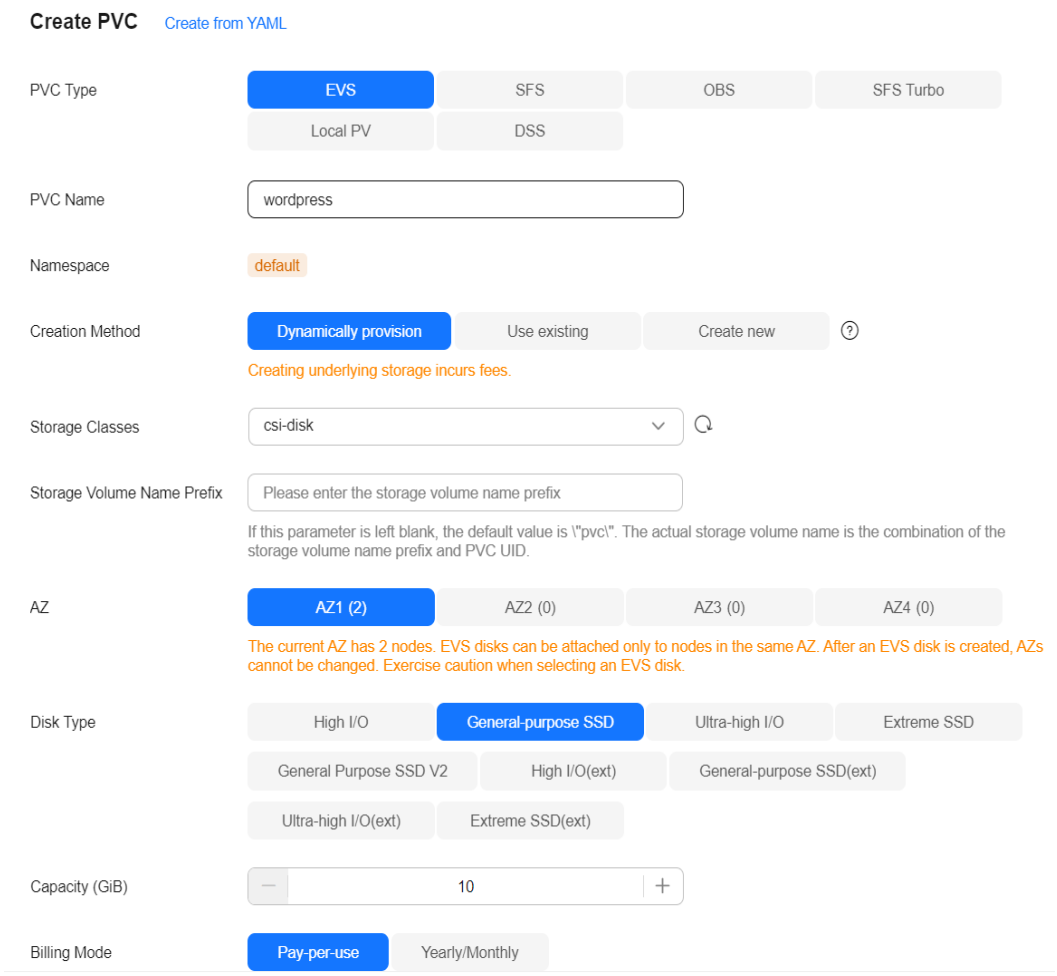
Step 6 Click **Environment Variables** and add environment variables listed in the table to add the MySQL database information to WordPress.



Environment Variable	Example	Description
WORDPRESS_DB_HOST	mysql:3306	The IP address for accessing the database. In this example, you need to enter the access mode of the MySQL workload, that is, the headless Service in **Step 4: Deploy MySQL**. You can use the internal domain name **mysql.default.svc.cluster.local:3306** of the cluster to access the workload. You can omit **.default.svc.cluster.local** and simply use **mysql:3306**.
WORDPRESS_DB_USER	db_user	The username for accessing data. The value must be the same as that of **MYSQL_USER** in **Step 4: Deploy MySQL**. This username is used to establish a connection with the MySQL database.
WORDPRESS_DB_PASSWORD	A custom database password	The password for accessing the database. The value must be the same as that of **MYSQL_PASSWORD** in **Step 4: Deploy MySQL**.
WORDPRESS_DB_NAME	database	The name of the database to be accessed. The value must be the same as that of **MYSQL_DATABASE** in **Step 4: Deploy MySQL**.

Step 7 Click **Data Storage**, click **Add Volume**, select **PVC**, and add an EVS disk as the MySQL storage.

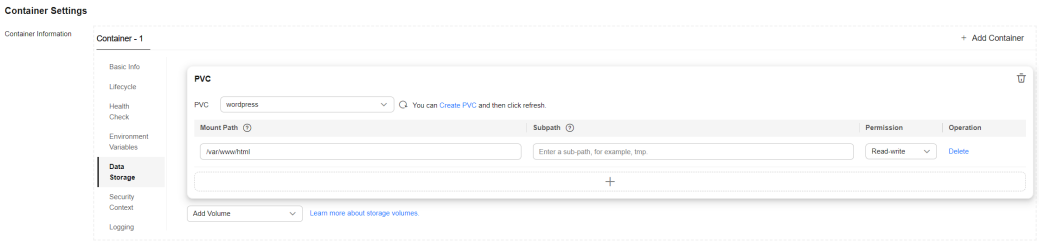
Click **Create PVC**, configure the parameters shown here, and keep the default values for other parameters.



Parameter	Example	Description
PVC Type	EVS	Select a type for the underlying storage volume used by the PVC.
PVC Name	wordpress	Enter a custom PVC name.
Creation Method	Dynamically provision	In this example, select **Dynamically provision**. The PVC, PV, and underlying storage volume will be automatically created. This method is ideal when no underlying storage volume is available.
Storage Classes	csi-disk	The default value is **csi-disk**.
AZ	AZ1	Select an AZ. The EVS disk can only be attached to nodes in the same AZ. After an EVS disk is created, the AZ where the disk locates cannot be changed.
Disk Type	General-purpose SSD	Select a proper type as required.

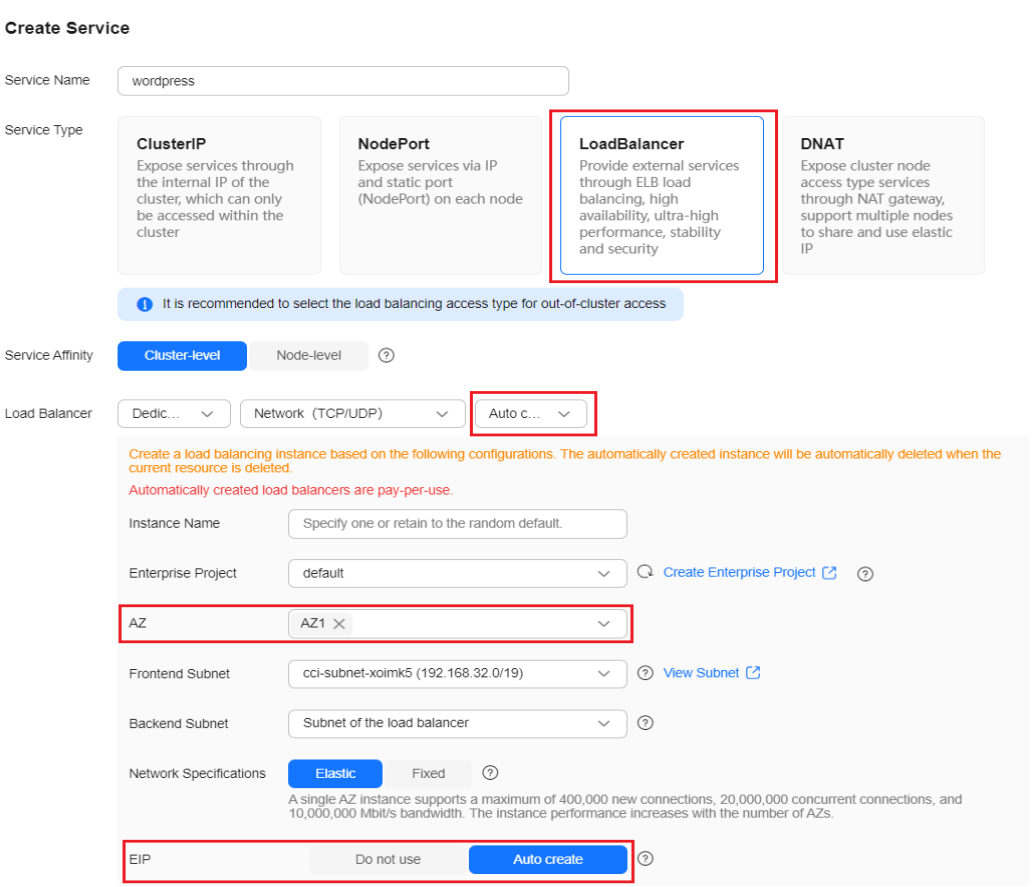
Parameter	Example	Description
Capacity (GiB)	10	Enter the capacity required. The default capacity is 10 GiB.

Click **Create** and enter the path for mounting the storage volume to the container. The default path used by WordPress is **/var/www/html**.



Step 8 Configure access settings.

In the **Service Settings** area, click the plus sign (+) and create a Service for external network access to the workload. This example shows how to create a LoadBalancer Service. Configure the below parameters in the sliding window on the right.



Parameter	Example	Description
Service Name	wordpress	Specify the Service name.
Service Type	LoadBalancer	Select a Service type. The Service type refers to the Service access mode. For details about the differences between Service types, see **Service Overview**.
Load Balancer	• Type: Dedicated • AZ: at least one AZ, for example, **AZ1** • EIP: Auto create Keep the default settings for other parameters.	Select **Use existing** if a load balancer is available. If there is no load balancer available, select **Auto create** to create one and bind an EIP to it. For details, see **Creating a LoadBalancer Service**.
Ports	• Protocol: TCP • Container Port: 80 • Service Port: 8080	• **Protocol**: applies to the load balancer listener. • **Container Port**: The port that the containerized application listens on. This value must be the same as the listening port provided by the application for external access. If the **wordpress** image is used, set this parameter to **80**. • **Service Port**: The port used by the load balancer to create a listener and provide an entry for external access. You can change this port if necessary.

Step 9 Click **Create Workload**.

Wait until the workload is created. After it is created, it will be displayed on the **Deployments** tab.

----End

Using kubectl

Step 1 Log in to the ECS where kubectl has been installed.

Step 2 Create a description file named **wordpress-deployment.yaml**. **wordpress-deployment.yaml** is an example file name. You can rename it as required.

```
vi wordpress-deployment.yaml
```

The file content is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
      version: v1
  template:
    metadata:
      labels:
        app: wordpress
        version: v1
    spec:
      containers:
        - name: container-1
          image: wordpress:latest
          env:
            - name: WORDPRESS_DB_HOST  # Database access address, that is, <headless-service-name>:service-port in MySQL. In this example, the value is mysql:3306.
              value: mysql:3306
            - name: WORDPRESS_DB_USER  # Username for accessing the database, which must be the same as the value of MYSQL_USER in MySQL. This username is used to establish a connection with MySQL.
              value: db_user
            - name: WORDPRESS_DB_PASSWORD # Password for accessing the database, which must be the same as the value of MYSQL_PASSWORD in MySQL
              value: *****
            - name: WORDPRESS_DB_NAME  # Name of the database to be accessed, which must be the same as the value of MYSQL_DATABASE in MySQL
              value: database
          resources:
            requests:
              cpu: 250m
              memory: 512Mi
            limits:
              cpu: 250m
              memory: 512Mi
          volumeMounts:    # Mount storage.
            - name: wordpress
              readOnly: false
              mountPath: /var/www/html # The default path used by WordPress is /var/www/html.
      imagePullSecrets:
        - name: default-secret
      volumes:
        - name: wordpress
          persistentVolumeClaim:
            claimName: wordpress
---
# Dynamically create a storage volume.
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wordpress
  namespace: default
  annotations:
```



```
    everest.io/disk-volume-type: SSD
    everest.io/enterprise-project-id: '0'
  labels:
    failure-domain.beta.kubernetes.io/region: eu-west-101  # Region where the EVS disk is in
    failure-domain.beta.kubernetes.io/zone:    # AZ where the EVS disk is in. It must be the same as the AZ of the node that runs the workload.
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: csi-disk
```

Step 3 Create the WordPress workload.

```
kubectl apply -f wordpress-deployment.yaml
```

Check the workload status.

```
kubectl get deployment
```

If information similar to the following is displayed, the workload has been created:

```
NAME              READY    UP-TO-DATE   AVAILABLE   AGE
wordpress         1/1      1            1           4m5s
```

Step 4 Create a description file named **wordpress-service.yaml**. **wordpress-service.yaml** is an example file name. You can rename it as required.

```
vi wordpress-service.yaml
```

The file content is as follows:

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  namespace: default
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate:
        '{
            "type": "public",
            "bandwidth_name": "cce-wordpress",
            "bandwidth_chargemode": "bandwidth",
            "bandwidth_size": 5,
            "bandwidth_sharetype": "PER",
            "eip_type": "5_bgp"
        }'
spec:
  selector:      # Label selector, which must be the same as the labels of the WordPress workload. It is used to match the desired workload.
    app: wordpress
  externalTrafficPolicy: Cluster
  ports:
    - name: cce-service-0
      targetPort: 80  # Container port. If the wordpress image is used, set this parameter to 80.
      nodePort: 0
      port: 8080      # Service port. You configure your own Service port.
      protocol: TCP
  type: LoadBalancer
```

Step 5 Create a Service.

```
kubectl create -f wordpress-service.yaml
```

If information similar to the following is displayed, the Service has been created:

```
service/wordpress created
```


Step 6 Check the Service.

```
kubectl get svc
```

If information similar to what is shown here is displayed, the workload's access mode has been configured. You can use the LoadBalancer Service to access the WordPress workload from the Internet. ****.**.**.**** specifies the EIP of the load balancer, and **8080** indicates the access port.

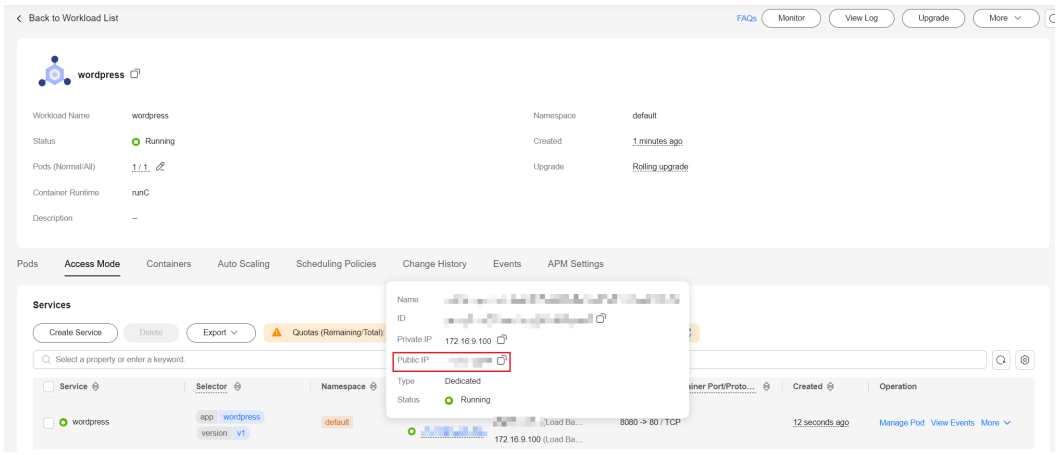
```
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)          AGE
kubernetes   ClusterIP      10.247.0.1       <none>         443/TCP          3d
mysql        ClusterIP      10.247.202.20    <none>         3306/TCP         8m
wordpress    LoadBalancer   10.247.130.196   **.**.**.**    8080:31540/TCP   51s
```

----End

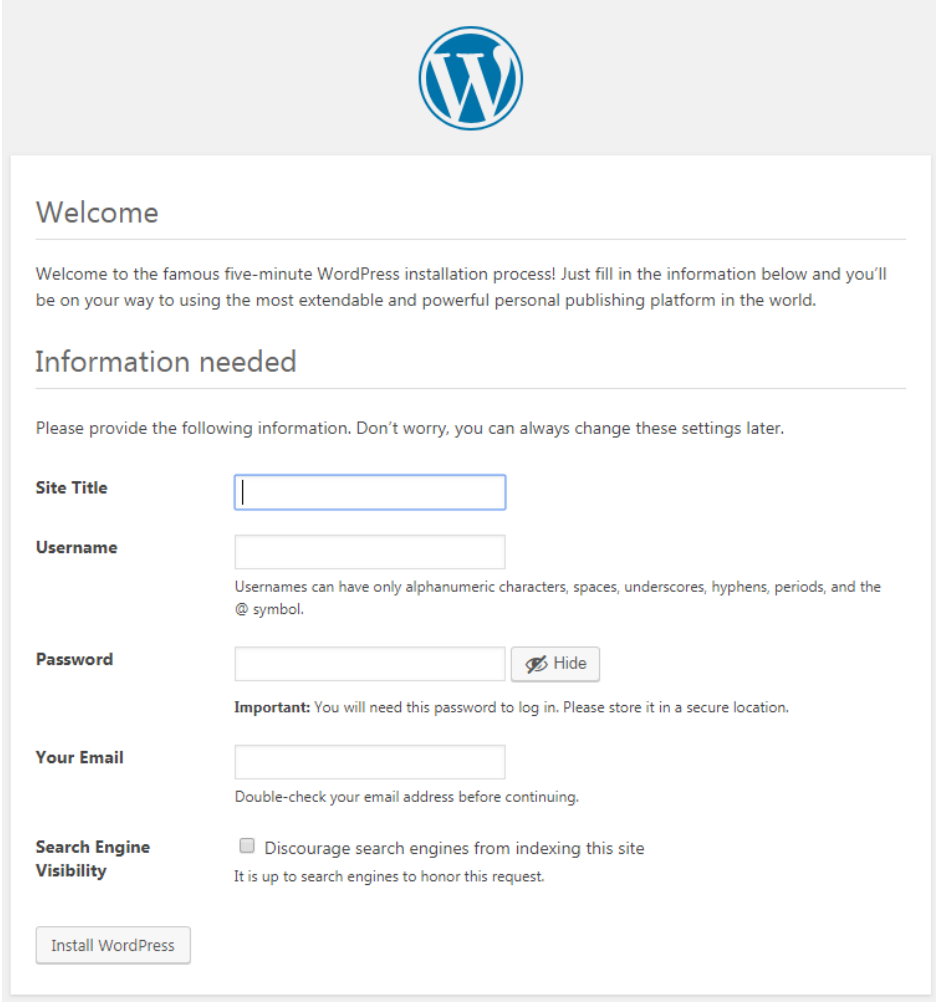
Step 6: Access WordPress

Step 1 Obtain the external access address of WordPress.

Click the name of the **wordpress** workload to enter its details page. On the page displayed, click the **Access Mode** tab, view the IP address of WordPress. The public IP address is the external access address.



Step 2 Enter *<external-access-address>:port* in the address box of a browser to access the application. The port is the same as the Service port configured in **Step 8**. It should be **8080**.



----End

Follow-up Operations: Release Resources

Release resources promptly if the cluster is no longer needed to avoid extra charges. For details, see **Deleting a Cluster**.

FAQs

Problem: WordPress cannot be accessed. The error message **Error establishing a database connection** is displayed.

Cause: This problem is usually related to the MySQL databases. The typical causes include:

- In **Step 3**, the MySQL image tag was incompatible with WordPress, so MySQL could not access WordPress.
- In **Step 2**, the database access address in WordPress was inconsistent with the headless Service access address in MySQL.

Solution

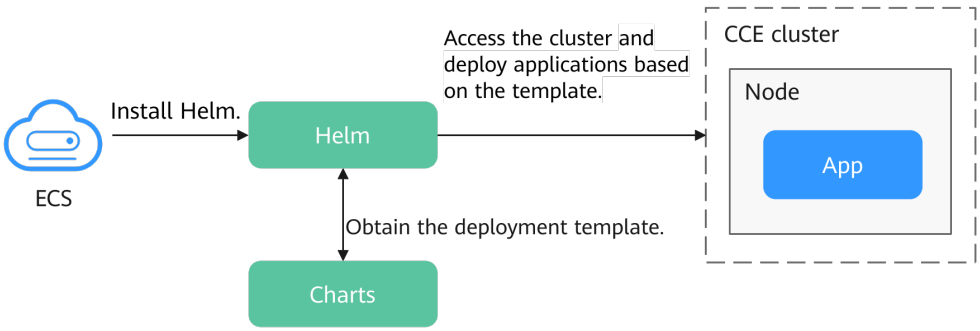
- Version incompatibility: Change the MySQL image tag to the recommended MySQL 8.0 tag.

- Inconsistent access address: Change the value of the **WORDPRESS_DB_HOST** environment variable in WordPress to the access address of the headless Service in MySQL.

3 Deploying an Application in a CCE Cluster Using a Helm Chart

Helm is a package manager that streamlines the deployment, upgrade, and management of Kubernetes applications. Helm uses charts, which are a packaging format that defines Kubernetes resources, to package all components deployed by Kubernetes. This includes application code, dependencies, configuration files, and deployment instructions. By doing so, Helm enables the distribution and deployment of complex Kubernetes applications in a more efficient, consistent manner. Moreover, Helm facilitates application upgrade and rollback, simplifying application lifecycle management.

This section describes how to deploy a WordPress workload using Helm.



Procedure

Step	Description
Preparations	Sign up for a HUAWEI ID.
Step 1: Enable CCE and Perform Authorization	Obtain the required permissions for your account when you use the CCE service in the current region for the first time.
Step 2: Create a Cluster	Create a CCE cluster to provide Kubernetes services.
Step 3: Create a Node Pool and Nodes	Create a node in the cluster to run your containerized applications.

Step	Description
Step 4: Access the Cluster Using kubectl	Before using Helm charts, access the cluster on a VM using kubectl.
Step 5: Install Helm	Install Helm on the VM with kubectl installed.
Step 6: Deploy the Template	Create a WordPress workload in the cluster using the Helm installation command and create a Service for the workload for Internet access.
Step 7: Access WordPress	Access the WordPress website from the Internet to start your blog.
Follow-up Operations: Release Resources	To avoid additional charges, delete the cluster resources promptly if you no longer require them after practice.

Preparations

- Before you start, sign up for a HUAWEI ID. For details, see **Signing up for a HUAWEI ID and Enabling Huawei Cloud Services**.

Step 1: Enable CCE and Perform Authorization

When you first log in to the CCE console, CCE automatically requests permissions to access related cloud services (compute, storage, networking, and monitoring) in the region where the cluster is deployed. If you have authorized CCE in the deployment region, skip this step.

Step 1 Log in to the **CCE console** using your HUAWEI ID.

Step 2 Click  in the upper left corner and select a region.

Step 3 If this is your first login to the CCE console in the selected region, the **Authorization Statement** dialog box will appear. Read it carefully and click **OK**.

After you agree to delegate permissions, CCE uses IAM to create an agency named **cce_admin_trust**. This agency is granted **Tenant Administrator** permissions for the resources of other cloud services (excluding IAM). These permissions are required for CCE to access dependent cloud services and are only valid for the current region. You can view the authorization records in each region by navigating to the IAM console, choosing **Agencies** in the navigation pane, and clicking **cce_admin_trust**. For more details, see **Cloud Service Agency**.

To ensure CCE can run normally, do not delete or modify the **cce_admin_trust** agency, as CCE requires **Tenant Administrator** permissions.

> **NOTE**
>
> CCE has updated the **cce_admin_trust** agency permissions to enhance security while accommodating dependencies on other cloud services. The new permissions no longer include **Tenant Administrator** permissions. This update is only available in certain regions. If your clusters are of v1.21 or later, a message will appear on the console asking you to re-grant permissions. After re-granting, the **cce_admin_trust** agency will be updated to include only the necessary cloud service permissions, with the **Tenant Administrator** permissions removed.
>
> When creating the **cce_admin_trust** agency, CCE creates a custom policy named **CCE admin policies**. Do not delete this policy.

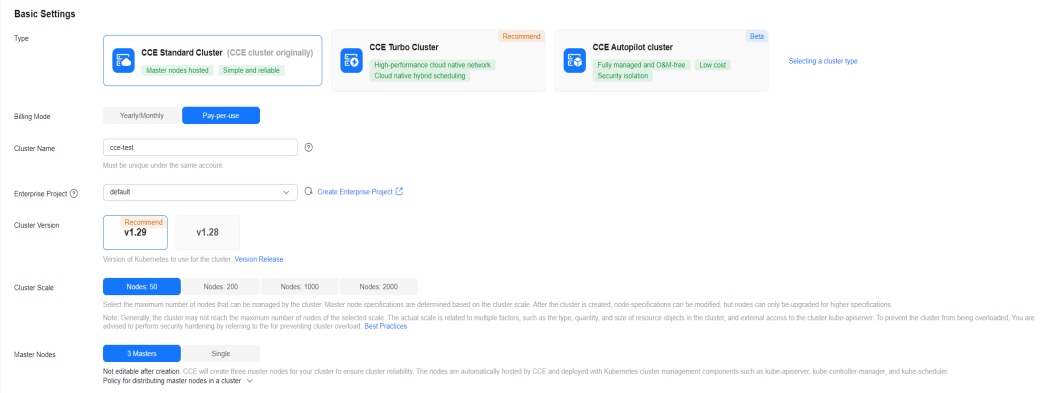
----End

Step 2: Create a Cluster

Step 1 Log in to the **CCE console** and click **Buy Cluster**.

Step 2 Configure basic cluster parameters.

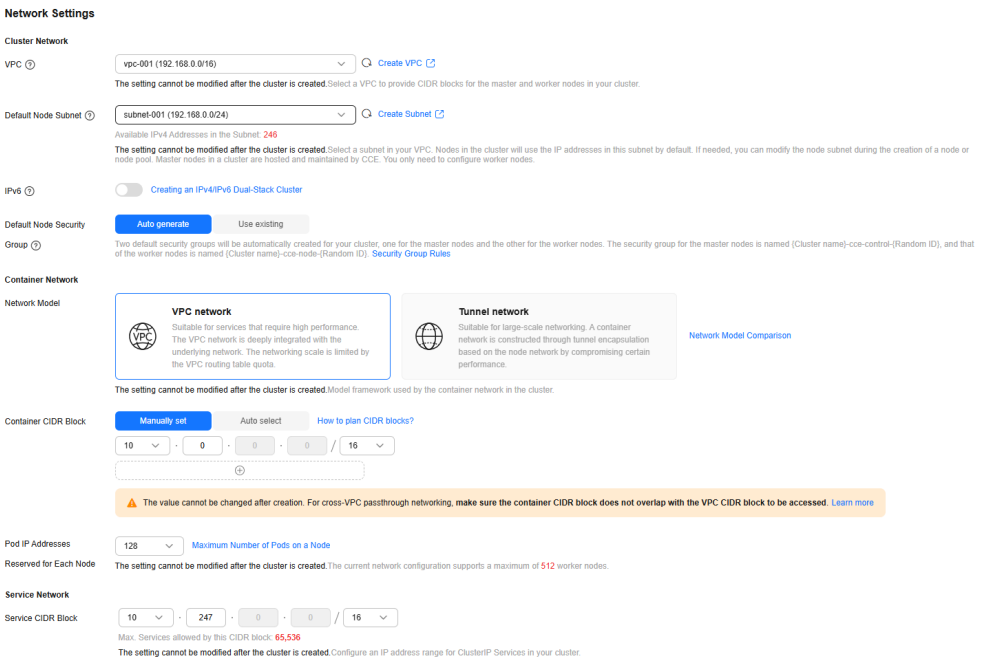
This example describes only mandatory parameters. You can keep default settings for most other parameters. For details, see **Buying a CCE Standard/Turbo Cluster**.



Parameter	Example	Description
Type	CCE Standard Cluster	CCE supports multiple cluster types to meet diverse needs, including highly reliable, highly secure, commercial containers. You can select **CCE Standard Cluster** or **CCE Turbo Cluster** as required. • CCE standard clusters provide highly reliable, highly secure, commercial containers. • CCE Turbo clusters use high-performance cloud native networks and support cloud native hybrid scheduling. These clusters offer improved resource utilization and are suitable for a wider range of scenarios. For more details, see **Comparison Between Cluster Types**.
Billing Mode	Pay-per-use	Select a billing mode for the cluster. • **Yearly/Monthly**: a prepaid billing mode. Resources are billed based on how long you will need the resources for. This mode is more cost-effective when resource usage periods are predictable. If you choose this option, select the **desired duration** and decide whether to enable automatic renewal (monthly or yearly). Monthly subscriptions renew automatically every month, and yearly subscriptions renew automatically every year. • **Pay-per-use**: a postpaid billing mode. Resources are billed based on actual usage duration. This mode is suitable for flexible scenarios where you may need to provision or delete resources at any time. For more details, see **Billing Modes**.
Cluster Name	cce-test	Enter a name for the cluster.
Enterprise Project	default	Enterprise projects facilitate project-level management and grouping of cloud resources and users. This parameter is only displayed for enterprise users who have enabled enterprise projects.
Cluster Version	v1.29 (recommended)	Select the latest Kubernetes commercial release to benefit from new, reliable, and production-ready features. CCE offers multiple Kubernetes versions.

Parameter	Example	Description
Cluster Scale	Nodes: 50	This parameter controls the maximum number of worker nodes the cluster can manage. Configure it as needed. After the cluster is created, it can be scaled out, but it cannot be scaled in.
Master Nodes	3 Masters	Select the number of master nodes, also known as control plane nodes. These nodes are hosted on CCE and run Kubernetes components like kube-apiserver, kube-controller-manager, and kube-scheduler. • **Multiple**: Three master nodes will be created to ensure high availability. • **Single**: Only one master node will be created in your cluster. This setting cannot be changed after the cluster is created.

Step 3 Configure network parameters.

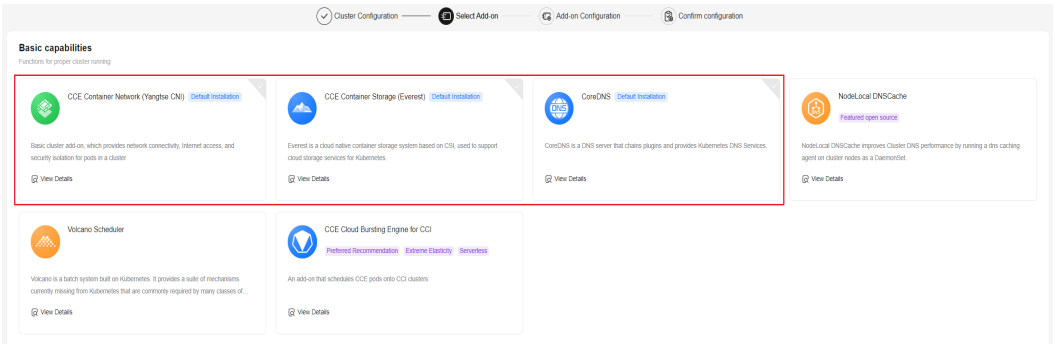


Parameter	Example	Description
VPC	vpc-001	Select a VPC for the cluster. If no VPC is available, click **Create VPC** to create one. After creating the VPC, click the refresh icon. For details, see **Creating a VPC and Subnet**.

Parameter	Example	Description
Default Node Subnet	subnet-001	Select a subnet. Nodes in the cluster will be assigned IP addresses from this subnet.
Network Model	VPC network	Select **VPC network** or **Tunnel network**. The default value is **VPC network**. For details about the differences between container network models, see **Container Networks**.
Container CIDR Block	Manually set (10.0.0.0/16)	Specify the container CIDR block. The CIDR block determines how many containers you can deploy in the cluster. You can select **Manually set** or **Auto select**.
Pod IP Addresses Reserved for Each Node	128	Specify the number of allocatable container IP addresses (the **alpha.cce/fixPoolMask** parameter) on each node. This determines the maximum number of pods that can be created on each node.
Service CIDR Block	10.247.0.0/16	Configure the cluster-wide IP address range. This controls how many Services can be created. This setting cannot be changed later.

Step 4 Click **Next: Select Add-on**. On the page displayed, select the add-ons to be installed during cluster creation.

This example only includes mandatory add-ons, which are installed automatically.



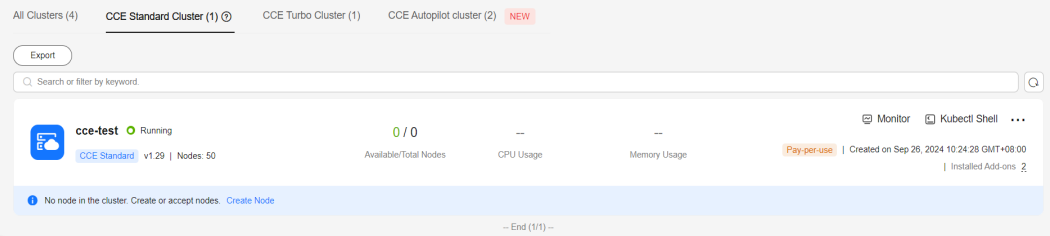
Step 5 Click **Next: Configure Add-on**. No setup is needed for the default add-ons.

Step 6 Click **Next: Confirm Settings**, check the displayed cluster resource list, and click **Submit**.

Wait for the cluster to be created. It takes approximately 5 to 10 minutes.

The newly created cluster in the **Running** state with zero nodes will be displayed on the **Clusters** page.

Figure 3-1 Cluster created



----End

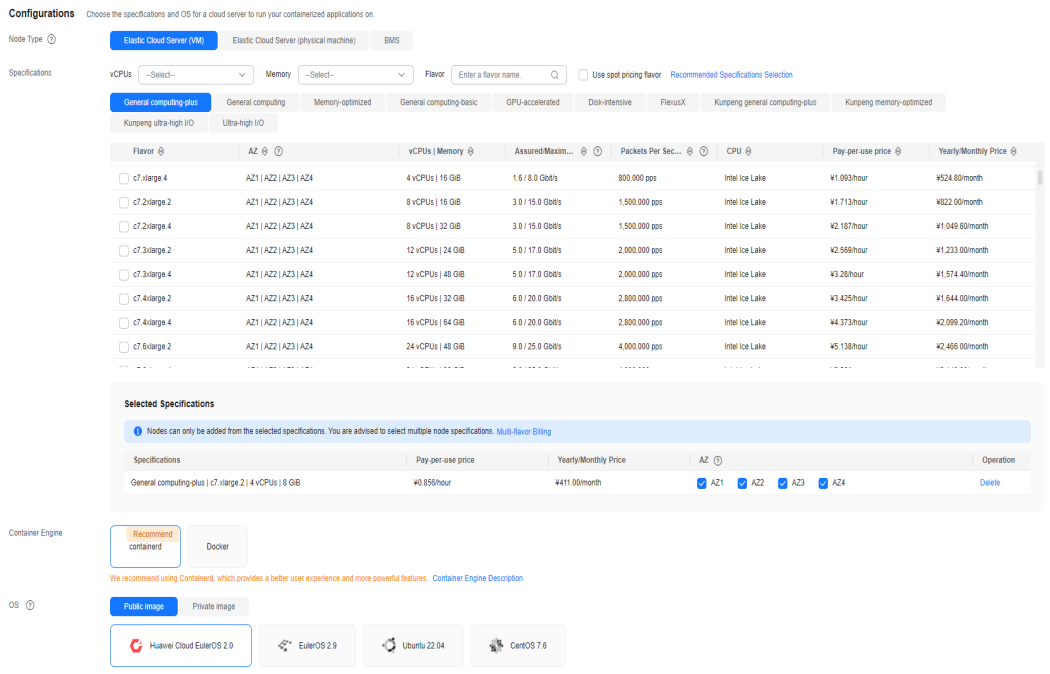
Step 3: Create a Node Pool and Nodes

Step 1 Log in to the CCE console and click the cluster name to access the cluster console.

Step 2 In the navigation pane, choose **Nodes**. On the **Node Pools** tab, click **Create Node Pool** in the upper right corner.

Step 3 Configure the node pool parameters.

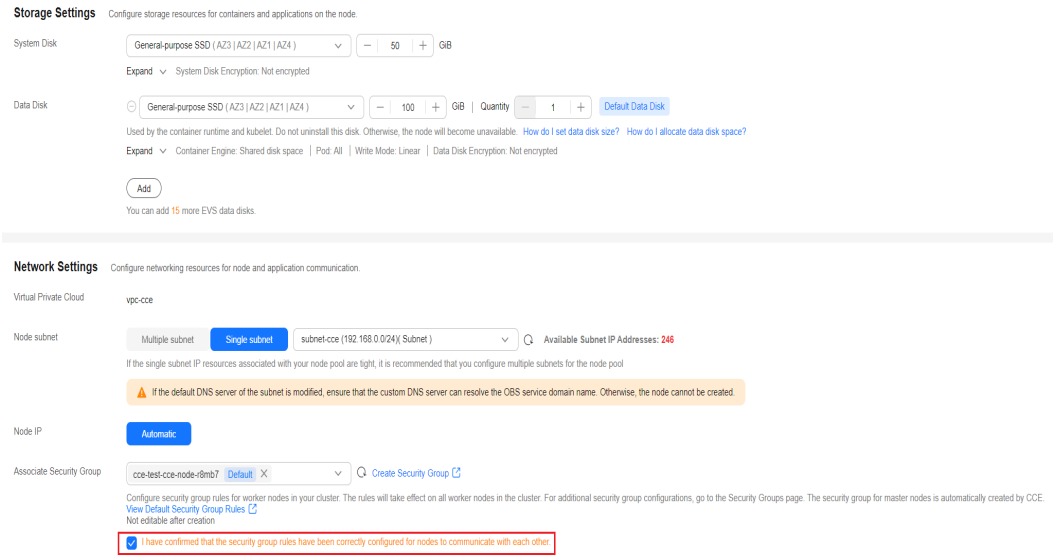
This example describes only mandatory parameters. You can keep default settings for most other parameters. For details, see **Creating a Node Pool**.



Parameter	Example	Description
Node Type	Elastic Cloud Server (VM)	Select a node type based on service requirements. The available node flavors will then be displayed in the **Specifications** area for you to choose from.

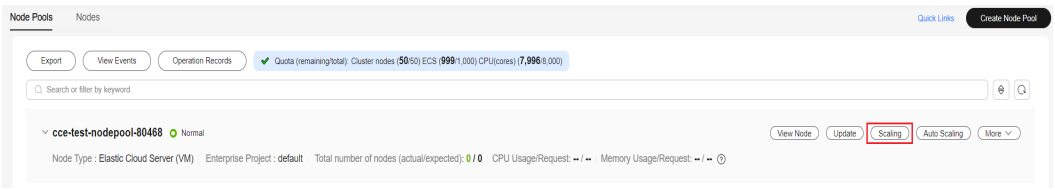
Parameter	Example	Description
Specifications	General computing-plus 4 vCPUs | 8 GiB	Select a node flavor that best fits your service needs. • In this example, there are no specific requirements for memory or GPU resources. General computing-plus or general computing nodes are recommended. – General computing-plus nodes use dedicated vCPUs and next-generation network acceleration engines to provide strong compute and network performance. – General computing nodes provide a balance of compute, memory, and network resources and a baseline level of vCPU performance with the ability to burst above the baseline. • For optimal cluster component performance, choose a node with at least 4 vCPUs and 8 GiB of memory.
Container Engine	containerd	Select a container engine based on service requirements. For details about the differences between container engines, see **Container Engines**.
OS	Huawei Cloud EulerOS 2.0	Select an OS for the nodes.
Login Mode	A custom password	• **Password**: Set and confirm a password for node login. The default username is **root**. Keep the password secure. It cannot be retrieved if forgotten. • **Key Pair**: Select a key pair for node login and confirm that you have the key file. Without this file, you will not be able to log in. Key pairs are used for identity authentication when you remotely access nodes. If you do not have a key pair, click **Create Key Pair**. For details, see **Creating a Key Pair on the Management Console**.

Step 4 Configure parameters in **Storage Settings** and **Network Settings**. In this example, keep the default settings. Select **I have confirmed that the security group rules have been correctly configured for nodes to communicate with each other.** and click **Next: Confirm**.

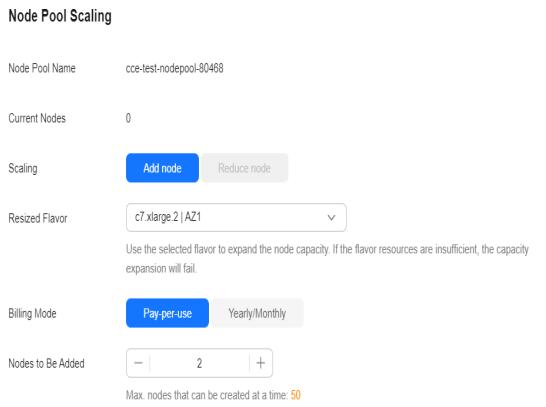


Step 5 Review the node specifications, read and confirm the instructions on the page, and click **Submit**.

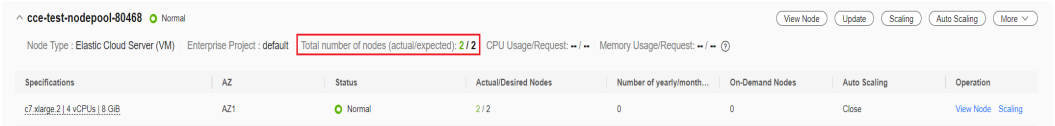
Step 6 Locate the newly created node pool and click **Scaling**. The node pool initially contains zero nodes.



Step 7 Set the number of nodes to add to **2**. This will create two more nodes in the node pool.



Step 8 Wait for the nodes to be created. It takes approximately 5 to 10 minutes.



----End

Step 4: Access the Cluster Using kubectl

NOTICE

You need to create an ECS **bound with an EIP in the same VPC as the cluster** first.

Step 1 Install kubectl on the ECS.

You can check whether kubectl has been installed by running **kubectl version**. If kubectl has been installed, you can skip this step.

The Linux environment is used as an example to describe how to install and configure kubectl. For more installation methods, see **kubectl**.

1. Download kubectl.
   ```
   cd /home
   curl -LO https://dl.k8s.io/release/{v1.29.0}/bin/linux/amd64/kubectl
   ```
   {v1.29.0} specifies the version. You can replace it as required.
2. Install kubectl.
   ```
   chmod +x kubectl
   mv -f kubectl /usr/local/bin
   ```

Step 2 Configure a credential for kubectl to access the Kubernetes cluster.

1. Log in to the **CCE console** and click the cluster name to access the cluster console. Choose **Overview** in the navigation pane.
2. On the page displayed, locate the **Connection Information** area, click **Configure** next to **kubectl**, and view the kubectl connection information.
3. In the window that slides out from the right, locate the **Download the kubeconfig file** area, select **Intranet access** for **Current data**, and download the corresponding configuration file.
4. Log in to the VM where the kubectl client has been installed and copy and paste the configuration file (for example, **kubeconfig.yaml**) downloaded in the previous step to the **/home** directory.
5. Save the kubectl authentication file to the configuration file in the **$HOME/.kube** directory.
   ```
   cd /home
   mkdir -p $HOME/.kube
   mv -f kubeconfig.yaml $HOME/.kube/config
   ```
6. Run the kubectl command to see whether the cluster can be accessed.

   For example, to view the cluster information, run the following command:

   ```
   kubectl cluster-info
   ```

   Information similar to the following is displayed:

   ```
   Kubernetes master is running at https://*.*.*.*:5443
   CoreDNS is running at https://*.*.*.*:5443/api/v1/namespaces/kube-system/services/coredns:dns/proxy
   To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
   ```

----End

Step 5: Install Helm

This section uses Helm v3.7.0 as an example. If other versions are needed, see **Helm**.

Step 1 Download the Helm client to a VM in a cluster.

```
wget https://get.helm.sh/helm-v3.7.0-linux-amd64.tar.gz
```

Step 2 Decompress the Helm package.

```
tar -xzvf helm-v3.7.0-linux-amd64.tar.gz
```

Step 3 Copy and paste Helm to the system path, for example, **/usr/local/bin/helm**.

```
mv linux-amd64/helm /usr/local/bin/helm
```

Step 4 Check the Helm version.

```
helm version
version.BuildInfo{Version:"v3.7.0",GitCommit:"eeac83883cb4014fe60267ec6373570374ce770b",GitTreeState:"clean",GoVersion:"g01.16.8"}
```

----End

Step 6: Deploy the Template

This section uses the WordPress template as an example.

Step 1 Add the official WordPress repository.

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

Step 2 Run the following commands to create a WordPress workload:

```
helm install myblog bitnami/wordpress \
  --set mariadb.primary.persistence.enabled=true \
  --set mariadb.primary.persistence.storageClass=csi-disk \
  --set mariadb.primary.persistence.size=10Gi \
  --set persistence.enabled=false
```

The custom instance name is specified by *myblog*. The remaining parameters serve the following functions:

- Persistent storage volumes are used by the MariaDB database that is connected to WordPress to store data. StorageClass is used to automatically create persistent storage. The EVS disk type (csi-disk) is used, with a size of 10 GiB.
- WordPress requires no data persistence, so you can set **persistence.enabled** to **false** for the PV.

The command output is as follows:

```
coalesce.go:223: warning: destination for mariadb.networkPolicy.egressRules.customRules is a table. Ignoring non-table value ([])
NAME: myblog
LAST DEPLOYED: Mon Mar 27 11:47:58 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: wordpress
CHART VERSION: 15.2.57
APP VERSION: 6.1.1

** Be patient while the chart is being deployed.**
```



```
Your WordPress site can be accessed through the following DNS name from within your cluster:

    myblog-wordpress.default.svc.cluster.local (port 80)

To access your WordPress site from outside the cluster, follow the steps below:

1. Get the WordPress URL by running these commands:

  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        Watch the status with: 'kubectl get svc --namespace default -w myblog-wordpress'

   export SERVICE_IP=$(kubectl get svc --namespace default myblog-wordpress --template "{{ range
(index .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}")
   echo "WordPress URL: http://$SERVICE_IP/"
   echo "WordPress Admin URL: http://$SERVICE_IP/admin"

2. Open a browser and access WordPress using the obtained URL.

3. Log in with the following credentials to see your blog:

  echo Username: user
  echo Password: $(kubectl get secret --namespace default myblog-wordpress -o
jsonpath="{.data.wordpress-password}" | base64 -d)
```

----End

Step 7: Access WordPress

Step 1 Modify the WordPress Service configuration.

To use a LoadBalancer Service in CCE, you need to configure it with additional annotations. Unfortunately, **bitnami/wordpress** does not come with this configuration, so you will have to modify it manually.

```
kubectl edit svc myblog-wordpress
```

Add **kubernetes.io/elb.autocreate** and **kubernetes.io/elb.class** to **metadata.annotations** and save the changes. These two annotations are used to create a shared load balancer, which allows access to WordPress via the EIP of the load balancer.

```
apiVersion: v1
kind: Service
metadata:
  name: myblog-wordpress
  namespace: default
  annotations:
    kubernetes.io/elb.autocreate: '{ "type": "public", "bandwidth_name": "myblog-wordpress",
"bandwidth_chargemode": "bandwidth", "bandwidth_size": 5, "bandwidth_sharetype": "PER", "eip_type":
"5_bgp" }'
    kubernetes.io/elb.class: union
spec:
  ports:
    - name: http
...
```

Step 2 Check the Service.

```
kubectl get svc
```

If information similar to the following is displayed, the workload's access mode has been configured. You can use the LoadBalancer Service to access the WordPress workload from the Internet. ****.**.**.**** specifies the EIP of the load balancer, and **80** indicates the access port.

```
NAME              TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)         AGE
kubernetes        ClusterIP     10.247.0.1      <none>        443/TCP         3d
```

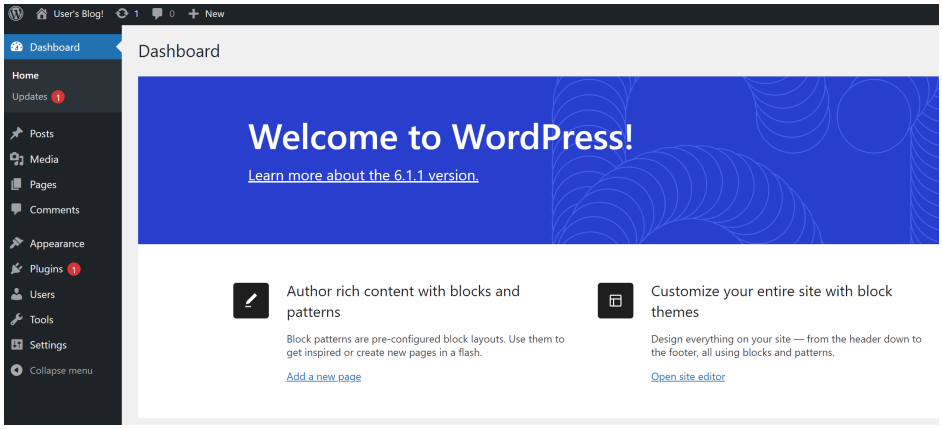


```
myblog-mariadb     ClusterIP      10.247.202.20   <none>        3306/TCP        8m
myblog-wordpress   LoadBalancer   10.247.130.196  **.**.**.**   80:31540/TCP    8m
```

Step 3 Access WordPress.

- To access the WordPress web page, enter *<EIP of the load balancer>***:80** in the address box of a browser.

- To access the WordPress management console:
  a. Run the following command to obtain the password of **user**:
  ```
  kubectl get secret --namespace default myblog-wordpress -o jsonpath="{.data.wordpress-password}" | base64 -d
  ```
  b. In the address box of a browser, enter *<EIP of the load balancer>***:80/login** to access the WordPress backend. The user name is **user**, and the password is the character string obtained in the previous step.



----End

Follow-up Operations: Release Resources

Release resources promptly if the cluster is no longer needed to avoid extra charges. For details, see **Deleting a Cluster**.

