

Cloud Container Engine

Getting Started

Issue 01
Date 2023-09-28



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2023. All rights reserved.

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

Trademarks and Permissions



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

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

Notice

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

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

Contents

1 Introduction.....	1
2 Preparations.....	3
3 Creating a Kubernetes Cluster.....	6
4 Creating a Deployment (Nginx).....	9
5 Deploying WordPress and MySQL That Depend on Each Other.....	15
5.1 Overview.....	15
5.2 Creating a MySQL Workload.....	16
5.3 Creating a WordPress Workload.....	20
6 Deploying WordPress Using Helm.....	29

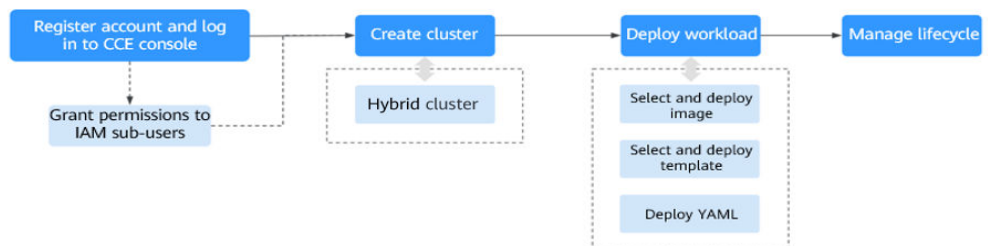
1 Introduction

This section describes how to use Cloud Container Engine (CCE) and provides frequently asked questions (FAQs) to help you quickly get started with CCE.

Procedure

Complete the following tasks to get started with CCE.

Figure 1-1 Procedure for getting started with CCE



Step 1 Register a Huawei Cloud account and grant permissions to IAM users.

Huawei Cloud accounts have the permissions to use CCE. However, IAM users created by a Huawei Cloud account do not have the permission. You need to manually grant the permission to IAM users. For details, see [Permissions Overview](#).

Step 2 Create a cluster.

For details on how to create a regular Kubernetes cluster, see [Creating a Kubernetes Cluster](#).

Step 3 Create a workload from an image or chart.

- [Creating a Deployment \(Nginx\)](#)
- [Deploying WordPress and MySQL That Depend on Each Other](#)

Step 4 View workload status and logs. Upgrade, scale, and monitor the workload.

For details, see [Managing Workloads and Jobs](#).

----End

FAQs

1. **Is CCE suitable for users who are not familiar with Kubernetes?**

Yes. The CCE console is easy-to-use, and the *Getting Started* guide helps you quickly understand and use CCE.

2. **Is CCE suitable for users who have little experience in building images?**

In addition to storing images created by yourself in **My Images**, CCE allows you to create containerized applications using open source images. For details, see [Creating a Deployment \(Nginx\)](#).

3. **How do I create a workload using CCE?**

Create a cluster and then create a workload in the cluster. For details, see [Creating a Deployment \(Nginx\)](#).

4. **How do I create a workload accessible to public networks?**

CCE provides different workload access types to address diverse scenarios. For details, see [Overview](#).

5. **How can I allow multiple workloads in the same cluster to access each other?**

Select the access type ClusterIP, which allows workloads in the same cluster to use their cluster-internal domain names to access each other.

Cluster-internal domain names are in the format of <self-defined service name>.<workload's namespace>.svc.cluster.local:<port number>. For example, nginx.default.svc.cluster.local:80.

Example:

Assume that workload A needs to access workload B in the same cluster. Then, you can create a [ClusterIP](#) Service for workload B. After the ClusterIP Service is created, workload B is reachable at <self-defined service name>.<workload B's namespace>.svc.cluster.local:<port number>.

2 Preparations

Before using CCE, make the following preparations:

- [Registering a HUAWEI ID](#)
- [Creating an IAM user](#)
- [Obtaining Resource Permissions](#)
- [\(Optional\) Creating a VPC](#)
- [\(Optional\) Creating a Key Pair](#)

Registering a HUAWEI ID

If you already have a HUAWEI ID, skip this step. If you do not have one, proceed as follows:

1. Log in to the [Huawei Cloud official website](#), and click **Register** in the upper right corner.
2. On the page displayed, register an account as prompted.

After the registration, the system automatically redirects you to your personal information page.

Creating an IAM user

If you want to allow multiple users to manage your resources without sharing your password or keys, you can create users using IAM and grant permissions to the users. These users can use specified links and their own accounts to access Huawei Cloud and help you manage resources efficiently. You can also configure account security policies to ensure the security of these accounts.

Your accounts have the permissions to use CCE. However, IAM users created by your accounts do not have the permissions. You need to manually assign the permissions to IAM users. For details, see [Permissions Overview](#).

Obtaining Resource Permissions

CCE works closely with multiple cloud services to support computing, storage, networking, and monitoring functions. When you log in to the CCE console for the first time, CCE automatically requests permissions to access those cloud services in the region where you run your applications. Specifically:

- **Compute services**
When you create a node in a cluster, a cloud server is created accordingly. The prerequisite is that CCE has obtained the permissions to access Elastic Cloud Service (ECS) and Bare Metal Server (BMS).
- **Storage services**
CCE allows you to mount storage to nodes and containers in a cluster. The prerequisite is that CCE has obtained the permissions to access services such as Elastic Volume Service (EVS), Scalable File Service (SFS), and Object Storage Service (OBS).
- **Networking services**
CCE allows containers in a cluster to be published as services that can be accessed by external systems. The prerequisite is that CCE has obtained the permissions to access services such as Virtual Private Cloud (VPC) and Elastic Load Balance (ELB).
- **Container and monitoring services**
CCE supports functions such as container image pull, monitoring, and logging. The prerequisite is that CCE has obtained the permissions to access services such as SoftWare Repository for Container (SWR) and Application Operations Management (AOM).

After you agree to delegate the permissions, an agency named **cce_admin_trust** will be created for CCE in Identity and Access Management (IAM). The system account **op_svc_cce** will be delegated the **Tenant Administrator** role to perform operations on other cloud service resources. Tenant Administrator has the permissions on all cloud services except IAM, which calls the cloud services on which CCE depends. The delegation takes effect only in the current region. For details, see [Account Delegation](#).

To use CCE in multiple regions, request for cloud resource permissions in each region. You can go to the IAM console, choose **Agencies**, and click **cce_admin_trust** to view the delegation records of each region.

 **NOTE**

CCE may fail to run as expected if the Tenant Administrator role is not assigned. Therefore, do not delete or modify the **cce_admin_trust** agency when using CCE.

(Optional) Creating a VPC

A VPC provides an isolated, configurable, and manageable virtual network for CCE clusters.

Before creating the first cluster, ensure that a VPC has been created. For details, see [Creating a VPC](#).

If you already have a VPC available, skip this step.

Step 1 Log in to the management console.

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

Step 3 Under **Networking**, click **Virtual Private Cloud**.

Step 4 Click **Create VPC**.

Step 5 On the **Create VPC** page, configure parameters as prompted.

A default subnet will be created together with a VPC. You can click **Add Subnet** to create more subnets for the VPC.

Step 6 Click **Create Now**.

----End

(Optional) Creating a Key Pair

The cloud platform uses public key cryptography to protect the login information of your CCE nodes. Passwords or key pairs are used for identity authentication during remote login to nodes.

- You need to specify the key pair name and provide the private key when logging to CCE nodes using SSH if you choose the key pair login mode.
- If you choose the password login mode, skip this task.


NOTE

If you want to create pods in multiple regions, you need to create a key pair in each region.

Creating a Key Pair on the Management Console

If you have no key pair, create one on the management console. The procedure is as follows:

Step 1 Log in to the management console.

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

Step 3 Under **Computing**, click **Elastic Cloud Server**.

Step 4 In the navigation pane on the left, choose **Key Pair**.

Step 5 On the right pane, click **Create Key Pair**.

Step 6 Enter the key name and click **OK**.

Step 7 A key pair name consists of two parts: **KeyPair** and four random digits. You can enter an easy-to-remember name, for example, **KeyPair-xxxx_ecs**.

Step 8 Manually or automatically download the private key file. The file name is a specified key pair name with a suffix of .pem. Securely store the private key file. In the dialog box displayed, click **OK**.

NOTE

The private key file can be downloaded only once. Keep it secure. When creating an ECS, provide the name of your desired key pair. Each time you log in to the ECS using SSH, provide the private key.

----End

3 Creating a Kubernetes Cluster

Context

This section describes how to quickly create a CCE cluster. For details, see [Buying a CCE Cluster](#). In this example, the default or simple configurations are in use.

Creating a Cluster

Step 1 Log in to the [CCE console](#).

- If you have not created a cluster, click **Create** under **CCE cluster** on the wizard page.
- If you have created a cluster, choose **Clusters** from the navigation pane and click **Buy** next to **CCE cluster**.

Step 2 On the **Configure Cluster** page, configure cluster parameters.

In this example, a majority of parameters retain default values. Only mandatory parameters are described. For details, see [Table 3-1](#).

Table 3-1 Parameters for creating a cluster

Parameter	Description
Basic Settings	
*Cluster Name	Name of the cluster to be created. A cluster name contains 4 to 128 characters starting with a lowercase letter and not ending with a hyphen (-). Only lowercase letters, digits, and hyphens (-) are allowed.
*Enterprise Project	This parameter is displayed only for enterprise users who have enabled Enterprise Project Management.
*Cluster Version	Choose the latest version.
*Cluster Scale	Maximum number of worker nodes that can be managed by the cluster. If you select 50 nodes , the cluster can manage a maximum of 50 worker nodes.

Parameter	Description
*High Availability	The default value is Yes .
Network Settings	
*Network Model	You can select VPC network or Tunnel network . Retain the default value.
*VPC	VPC where the cluster will be located. If no VPC is available, click Create VPC to create one. After the VPC is created, click refresh.
*Master Node Subnet	Subnet where master nodes of the cluster are located.
*Container CIDR Block	CIDR block used by containers. The value determines the maximum number of containers in the cluster. Retain the default value.
*IPv4 Service CIDR Block	CIDR block for Services used by containers in the same cluster to access each other. The value determines the maximum number of Services you can create. The value cannot be changed after creation. Retain the default value.

Step 3 Click **Next: Add-on Configuration**. Retain the default settings.

Step 4 Click **Next: Confirm**. The cluster resource list is displayed. Confirm the information and click **Submit**.

It takes about 6 to 10 minutes to create a cluster.

The created cluster will be displayed on the **Clusters** page, and the number of nodes in the cluster is 0.

----End

Creating a Node

After a cluster is created, you need to create nodes in the cluster to run workloads.

Step 1 Log in to the CCE console.

Step 2 Click the created cluster.

Step 3 In the navigation pane, choose **Nodes**. Click **Create Node** in the upper right corner and configure node parameters.

The following describes only important parameters. For other parameters, retain the defaults.

Compute Settings

- **AZ**: Retain the default value.

- **Node Type:** Select **Elastic Cloud Server (VM)**.
- **Specifications:** Select node specifications that fit your business needs.
- **Container Engine:** Select a container engine as required.
- **OS:** Select the operating system (OS) of the nodes to be created.
- **Node Name:** Enter a node name.
- **Login Mode:**

Storage Settings

- **System Disk:** Configure the disk type and capacity based on your requirements. The default disk capacity is 50 GiB.
- **Data Disk:** Configure the disk type and capacity based on your requirements. The default disk capacity is 100 GiB.

Network Settings

- **VPC:** Use the default value, that is, the subnet selected during cluster creation.
- **Node Subnet:** Select a subnet in which the node runs.
- **Node IP:** private IP address of the node. Select **Random**.
- **EIP:** enables public network access. After an EIP is bound, the node can access the Internet, for example, downloading images from an external repository.
The default value is **Do not use**. You can also select **Use existing** or **Auto create**.

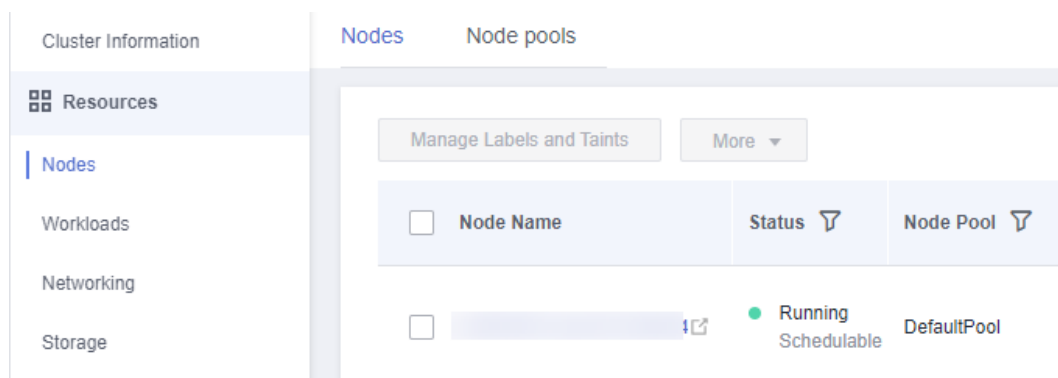
Step 4 At the bottom of the page, select the node quantity, and click **Next: Confirm**.

Step 5 Review the node specifications, read the instructions, select **I have read and understand the preceding instructions**, and click **Submit**.

It takes about 6 to 10 minutes to add a node.

The created node will be displayed on the **Nodes** page.

Figure 3-1 Node created successfully



----End

4 Creating a Deployment (Nginx)

You can use images to quickly create a single-pod workload that can be accessed from public networks. This section describes how to use CCE to quickly deploy an Nginx application and manage its lifecycle.

Prerequisites

You have created a CCE cluster that contains a node with 4 vCPUs and 8 GiB memory. The node is bound with an EIP.

A cluster is a logical group of cloud servers that run workloads. Each cloud server is a node in the cluster.

For details on how to create a cluster, see [Creating a Kubernetes Cluster](#).

Nginx Overview

Nginx is a lightweight web server. On CCE, you can quickly set up a Nginx web server.

This section uses the Nginx application as an example to describe how to create a workload. The creation takes about 5 minutes.

After Nginx is created successfully, you can access the Nginx web page.

Figure 4-1 Nginx web page

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Creating Nginx on the Console

The following is the procedure for creating a containerized workload from a container image.

- Step 1** Log in to the [CCE console](#).
- Step 2** Choose the target cluster.
- Step 3** In the navigation pane, choose **Workloads**. Then, click **Create Workload**.
- Step 4** Configure the following parameters and retain the default value for other parameters:

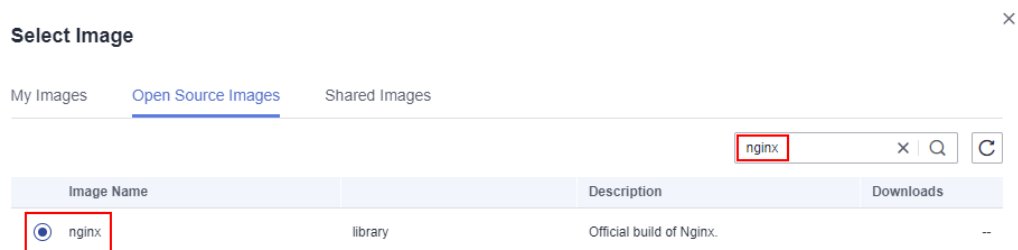
Basic Info

- **Workload Type:** Select **Deployment**.
- **Workload Name:** Set it to **nginx**.
- **Namespace:** Select **default**.
- **Pods:** Set the quantity of pods to **1**.

Container Settings

In the **Basic Info** area of **Container Information**, click **Select Image**. In the dialog box displayed, select **Open Source Images**, search for **nginx**, and select the **nginx** image.

Figure 4-2 Selecting the nginx image



Service Settings

Click the plus sign (+) to create a Service for accessing the workload from an external network. In this example, create a LoadBalancer Service. Configure the following parameters:

- **Service Name:** name of the Service exposed to external networks. In this example, the Service name is **nginx**.
- **Access Type:** Select **LoadBalancer**.
- **Service Affinity:** Retain the default value.
- **Load Balancer:** If a load balancer is available, select an existing load balancer. If not, select **Auto create** to create one.
- **Port:**
 - **Protocol:** Select **TCP**.
 - **Service Port:** Set this parameter to **8080**, which is mapped to the container port.

- **Container Port:** port on which the application listens. For containers created using the nginx image, set this parameter to **80**. For other applications, set this parameter to the port of the application.

Figure 4-3 Creating a Service

Create Service

Service Name:

Service Type: ClusterIP NodePort LoadBalancer DNAT

Service Affinity: Cluster-level Node-level

Load Balancer: [Create Load Balancer](#)

Port Configuration:

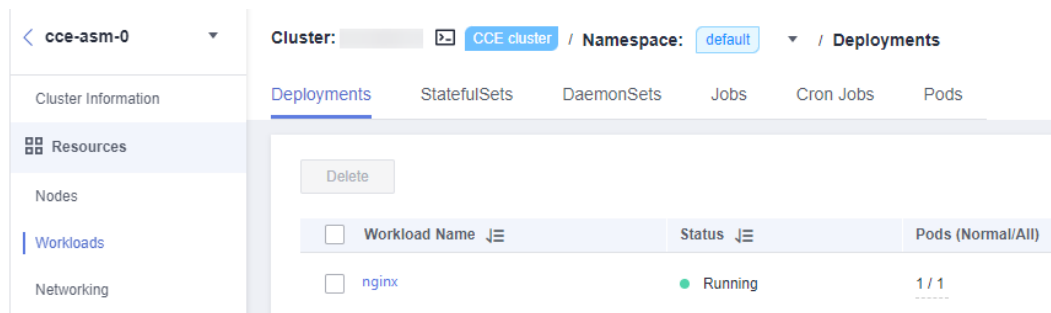
Protocol	Service Port	Container Port	Operation
TCP	8080	80	Delete

Step 5 Click **Create Workload**.

Wait until the workload is created.

The created Deployment will be displayed on the **Deployments** tab.

Figure 4-4 Workload created successfully



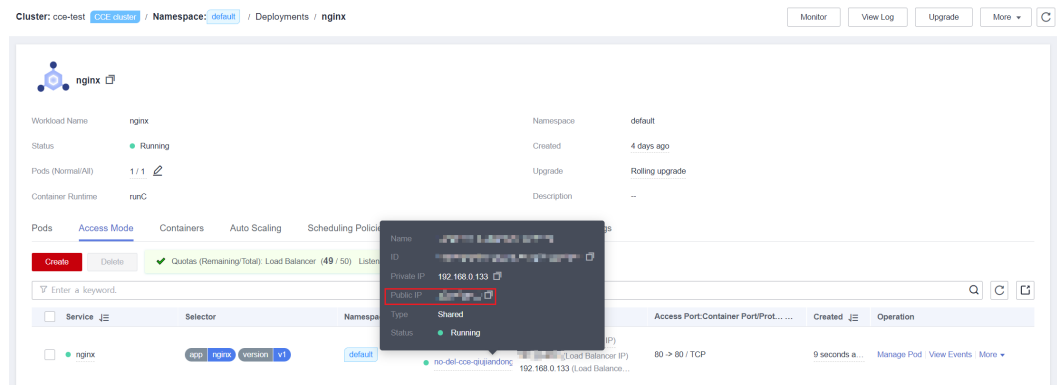
----End

Accessing Nginx

Step 1 Obtain the external access address of Nginx.

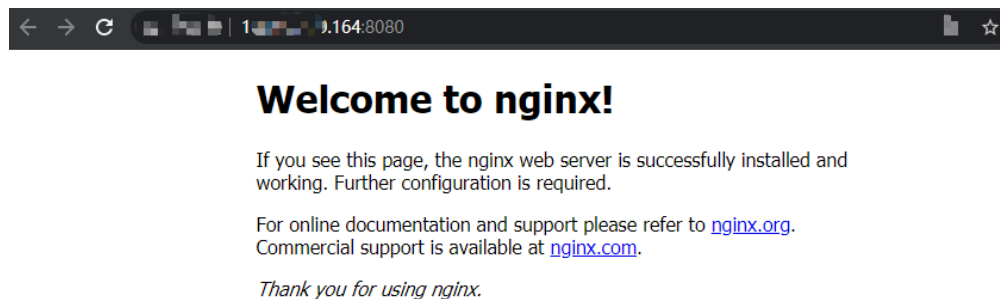
Click the Nginx workload to enter its details page. On the **Access Mode** tab, view the IP address of Nginx. The public IP address is the external access address.

Figure 4-5 Obtaining the external access address



Step 2 Enter the **external access address** in the address box of a browser. The following shows the welcome page if you successfully access the workload.

Figure 4-6 Accessing Nginx



----End

Creating Nginx Using kubectl

This section describes how to use kubectl to create a Deployment and expose the Deployment to the Internet through a LoadBalancer Service.

Step 1 Use kubectl to connect to the cluster. For details, see [Connecting to a Cluster Using kubectl](#).

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

vi nginx-deployment.yaml

```
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 3 Create a Deployment.

kubectl create -f nginx-deployment.yaml

If the following information is displayed, the Deployment is being created.

```
deployment "nginx" created
```

Check the Deployment.

kubectl get deployment

If the following information is displayed, the Deployment is running.

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

Parameter description

- **NAME:** specifies the name of a workload.
- **READY:** indicates the number of available pods/expected pods for the workload.
- **UP-TO-DATE:** indicates the number of replicas that have been updated.
- **AVAILABLE:** indicates the number of available pods.
- **AGE:** indicates the running period of the Deployment.

Step 4 Create a description file named **nginx-elb-svc.yaml**. 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.

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: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```


Step 5 Create a Service.

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

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

```
service/nginx created
```

```
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	10.78.42.242	80:31540/TCP	51s

Step 6 Enter the URL (for example, **10.78.42.242:80**) in the address box of a browser. **10.78.42.242** indicates the IP address of the load balancer, and **80** indicates the access port displayed on the CCE console.

Nginx is accessible.

Figure 4-7 Accessing Nginx through the LoadBalancer Service

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

----End

5 Deploying WordPress and MySQL That Depend on Each Other

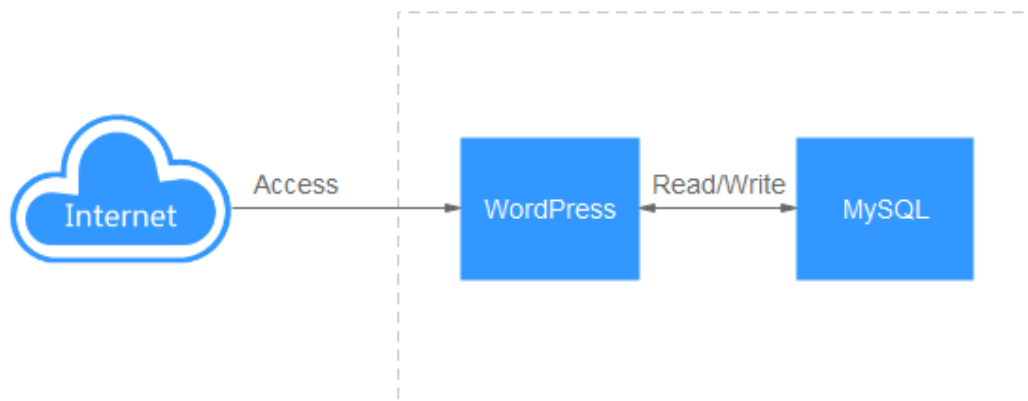
5.1 Overview

WordPress was originally a blog platform based on PHP and MySQL. It is gradually evolved into a content management system. You can set up your own blog website on any server that supports PHP and MySQL. Thousands of plug-ins and countless theme templates are available for WordPress and easy to install.

WordPress is a blog platform developed in hypertext preprocessor (PHP). You can set up your websites on the services that support PHP and MySQL databases, or use WordPress as a content management system. For more information about WordPress, visit <https://wordpress.org/>.

WordPress must be used together with MySQL. WordPress runs the content management program while MySQL serves as a database to store data. Generally, WordPress and MySQL run in different containers, as shown in the following figure.

Figure 5-1 WordPress



In this example, two container images are involved.

- **WordPress:** Select wordpress:php7.3 in this example.
- **MySQL:** Select mysql:5.7 in this example.

When WordPress accesses MySQL in a cluster, Kubernetes provides a resource object called Service for the workload access. In this example, a Service is created for MySQL and WordPress, respectively. For details about how to create and configure a Service, see the following sections.

5.2 Creating a MySQL Workload

WordPress must be used together with MySQL. WordPress runs the content management program while MySQL serves as a database to store data.

Prerequisites

You have created a CCE cluster that contains a node with 4 vCPUs and 8 GiB memory. For details on how to create a cluster, see [Creating a Kubernetes Cluster](#).

Operations on the Console

- Step 1** Log in to the [CCE console](#).
- Step 2** Choose the target cluster.
- Step 3** In the navigation pane, choose **Workloads**. Then, click **Create Workload**.
- Step 4** Set workload parameters.

Basic Info

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

Figure 5-2 Basic information about the MySQL workload

Basic Info

Workload Type: Deployment | StatefulSet | DaemonSet | Job | Cron Job

⚠ Switching the workload type will require you to configure workload parameters again.

Workload Name:

Namespace: [Create Namespace](#)

Pods:

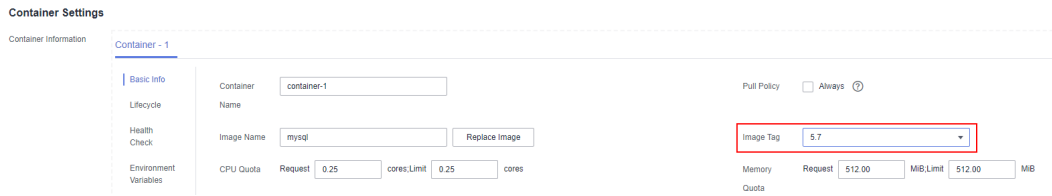
Time Zone:

Synchronization: Allows containers to use the same time zone as the node where they run. (This function is realized by the local disks mounted to the containers. Do not modify or delete the local disks.)

Container Settings

In the **Basic Info** area, click **Select Image**. In the dialog box displayed, select **Open Source Images**, search for **mysql**, select the **mysql** image, and set the image tag to **5.7**.

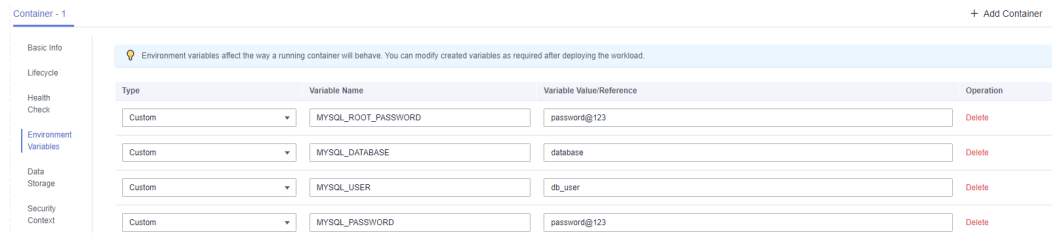
Figure 5-3 Selecting an image tag



Add the following four environment variables (details available in **MySQL**):

- **MYSQL_ROOT_PASSWORD**: password of the **root** user of MySQL.
- **MYSQL_DATABASE**: name of the database created during image startup.
- **MYSQL_USER**: name of the database user.
- **MYSQL_PASSWORD**: password of the database user.

Figure 5-4 Setting environment variables



Service Settings

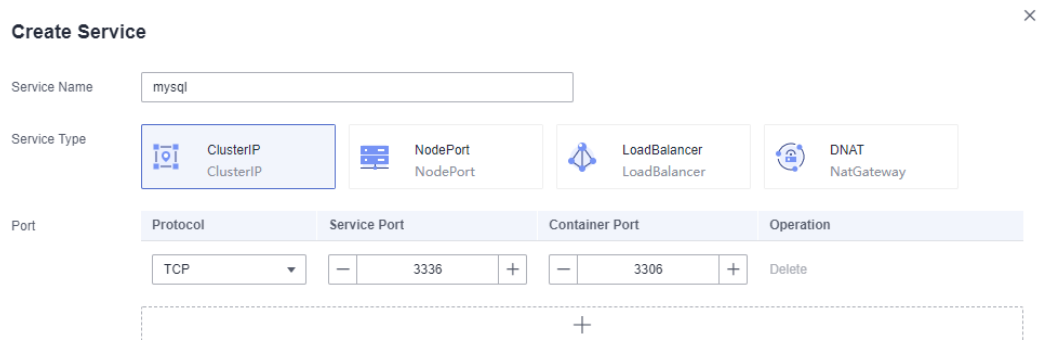
Click the plus sign (+) to create a Service for accessing MySQL from WordPress.

Select **ClusterIP** for **Access Type**, set **Service Name** to **mysql**, set both the **Container Port** and **Service Port** to **3306**, and click **OK**.

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

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

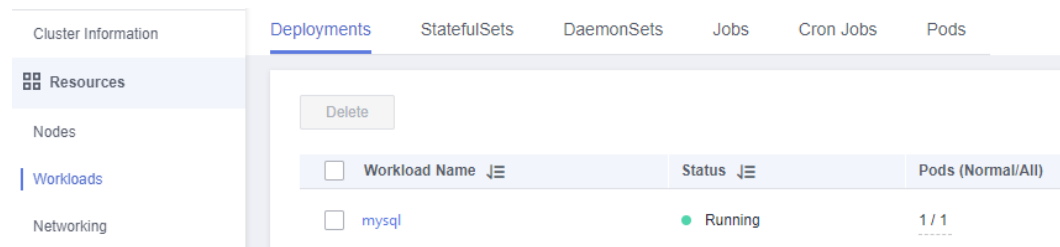
Figure 5-5 Creating a Service



Step 5 Click **Create Workload**.

Wait until the workload is created.

The created Deployment will be displayed on the **Deployments** tab.

Figure 5-6 Workload created successfully

----End

Operations Through kubectl

This section describes how to use kubectl to create a Deployment and expose the Deployment through a ClusterIP Service to allow access from within the cluster.

Step 1 Use kubectl to connect to the cluster. For details, see [Connecting to a Cluster Using kubectl](#).

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

vi mysql-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
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:5.7
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: password@123
            - name: MYSQL_DATABASE
              value: database
            - name: MYSQL_USER
              value: db_user
            - name: MYSQL_PASSWORD
              value: password@123
      resources:
        requests:
          cpu: 250m
```

```
memory: 512Mi
limits:
  cpu: 250m
  memory: 512Mi
imagePullSecrets:
  - name: default-secret
```

Step 3 Create a MySQL workload.

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

If the following information is displayed, the Deployment is being created.

```
deployment "mysql" created
```

Check the Deployment.

```
kubectl get deployment
```

If the following information is displayed, the Deployment is running.

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

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

```
vi mysql-service.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
  namespace: default
spec:
  selector:
    app: mysql
    version: v1
  ports:
    - name: cce-service-0
      targetPort: 3306
      nodePort: 0
      port: 3306
      protocol: TCP
  type: ClusterIP
```

Step 5 Create a Service.

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

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

```
service/mysql created
```

```
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
mysql     ClusterIP   10.247.202.20 <none>       3306/TCP   51s
```

----End

5.3 Creating a WordPress Workload

WordPress was originally a blog platform based on PHP and MySQL. It is gradually evolved into a content management system. You can set up your own blog website on any server that supports PHP and MySQL. Thousands of plug-ins and countless theme templates are available for WordPress and easy to install.

This section describes how to create a public WordPress website from images.

Prerequisites

- You have created a CCE cluster that contains a node with 4 vCPUs and 8 GiB memory. For details on how to create a cluster, see [Creating a Kubernetes Cluster](#).
- The MySQL database has been created by following the instructions in [Creating a MySQL Workload](#). In this example, WordPress data is stored in the MySQL database.

Operations on the Console

Step 1 Log in to the [CCE console](#).

Step 2 Choose the target cluster.

Step 3 In the navigation pane, choose **Workloads**. Then, click **Create Workload**.

Step 4 Set workload parameters.

Basic Info

- **Workload Type:** Select **Deployment**.
- **Workload Name:** Set it to **wordpress**.
- **Namespace:** Select **default**.
- **Pods:** Set this parameter to **2** in this example.

Figure 5-7 Setting the basic information about the workload

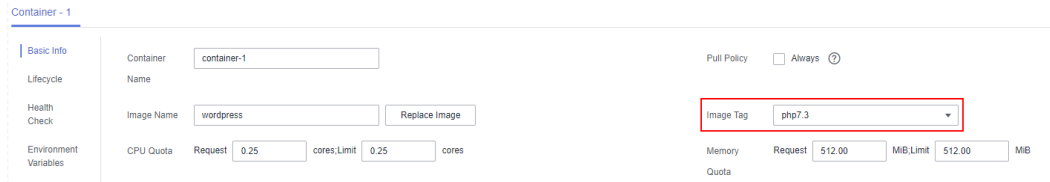
The screenshot shows the 'Basic Info' configuration page for a workload. It includes the following fields and options:

- Workload Type:** Three buttons are shown: 'Deployment' (selected), 'StatefulSet', and 'DaemonSet'. A warning message below reads: 'Switching the workload type will require you to configure workload parameters again.'
- Workload Name:** A text input field containing 'wordpress'.
- Namespace:** A dropdown menu showing 'default' and a 'Create Namespace' link.
- Pods:** A numeric input field with a minus sign, the number '2', and a plus sign.

Container Settings

In the **Basic Info** area, click **Select Image**. In the dialog box displayed, select **Open Source Images**, search for **wordpress**, select the **wordpress** image, and set the image tag to **php7.3**.

Figure 5-8 Selecting an image tag

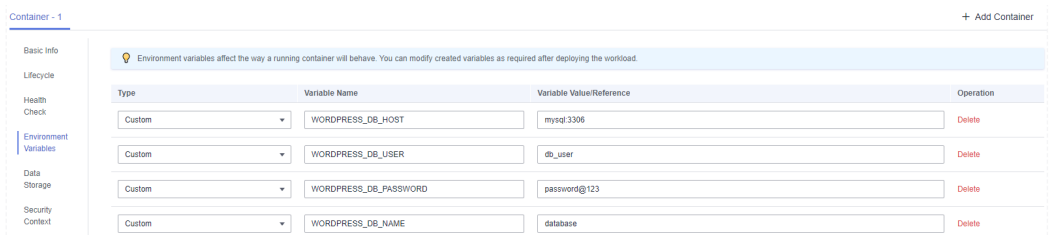


Add the following environment variables:

(These variables let WordPress know the information about the MySQL database.)

- **WORDPRESS_DB_HOST**: address for accessing the database, which can be found in the Service (on the **Services** tab page) of the MySQL workload. You can use the internal domain name **mysql.default.svc.cluster.local:3306** to access the database, or use only **mysql:3306** omitting **.default.svc.cluster.local**.
- **WORDPRESS_DB_USER**: username for accessing the database. The value must be the same as that of **MYSQL_USER** in [Creating a MySQL Workload](#), which is used to connect to MySQL.
- **WORDPRESS_DB_PASSWORD**: password for accessing the database. The value must be the same as that of **MYSQL_PASSWORD** in [Creating a MySQL Workload](#).
- **WORDPRESS_DB_NAME**: name of the database to be accessed. The value must be the same as that of **MYSQL_DATABASE** in [Creating a MySQL Workload](#).

Figure 5-9 Setting environment variables



Service Settings

Click the plus sign (+) to create a Service for accessing the workload from an external network. In this example, create a LoadBalancer Service. Configure the following parameters:

- **Service Name**: name of the Service exposed to external networks. In this example, the Service name is **wordpress**.
- **Access Type**: Select **LoadBalancer**.
- **Service Affinity**: Retain the default value.
- **Load Balancer**: If a load balancer is available, select an existing load balancer. If not, click **Create Load Balancer** to create one on the ELB console.

- **Port:**
 - **Protocol:** Select **TCP**.
 - **Service Port:** Set this parameter to **80**, which is mapped to the container port.
 - **Container Port:** port on which the application listens. For containers created using the wordpress image, set this parameter to **80**. For other applications, set this parameter to the port of the application.

Figure 5-10 Creating a Service

Create Service

Service Name:

Service Type:
 ClusterIP
 NodePort
 LoadBalancer
 DNAT

Service Affinity: Cluster-level Node-level ?

Load Balancer: Shared elb-373896-2 Create Load Balancer

Only shared load balancers in VPC vpc-asm where the cluster is deployed are supported.

Set ELB: Load balancing algorithm: Weighted round robin; Sticky session: Disable; Health check: Disable

I have read [Notes on Using Load Balancers](#).

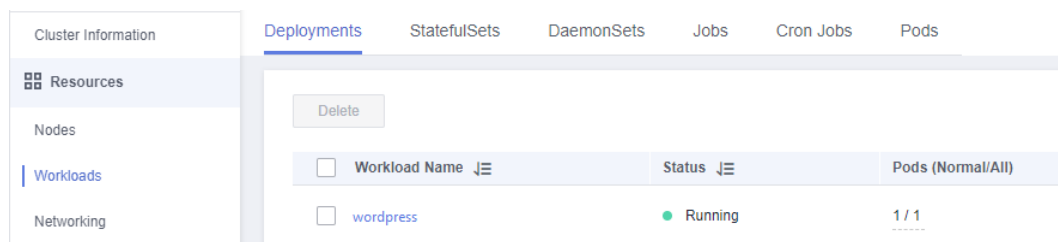
Port	Protocol	Service Port	Container Port	Operation
	TCP	80	80	Delete

Step 5 Click **Create Workload**.

Wait until the workload is created.

The created Deployment will be displayed on the **Deployments** tab.

Figure 5-11 Workload created successfully



----End

Operations Through kubectl

This section describes how to use kubectl to create a Deployment and expose the Deployment to the Internet through a LoadBalancer Service.

Step 1 Use kubectl to connect to the cluster. For details, see [Connecting to a Cluster Using kubectl](#).

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: wordpress
      version: v1
  template:
    metadata:
      labels:
        app: wordpress
        version: v1
    spec:
      containers:
        - name: container-1
          image: wordpress:php7.3
          env:
            - name: WORDPRESS_DB_HOST
              value: mysql:3306
            - name: WORDPRESS_DB_USER
              value: db_user
            - name: WORDPRESS_DB_PASSWORD
              value: password@123
            - name: WORDPRESS_DB_NAME
              value: database
          resources:
            requests:
              cpu: 250m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 128Mi
          imagePullSecrets:
            - name: default-secret
```

Step 3 Create a WordPress workload.

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

If the following information is displayed, the Deployment is being created.

```
deployment "wordpress" created
```

Check the Deployment.

```
kubectl get deployment
```

If the following information is displayed, the Deployment is running.

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

```
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:
    app: wordpress
  externalTrafficPolicy: Cluster
  ports:
  - name: cce-service-0
    targetPort: 80
    nodePort: 0
    port: 80
    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
```

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
mysql	ClusterIP	10.247.202.20	<none>	3306/TCP	8m
wordpress	LoadBalancer	10.247.130.196	**.**.**.**	80:31540/TCP	51s

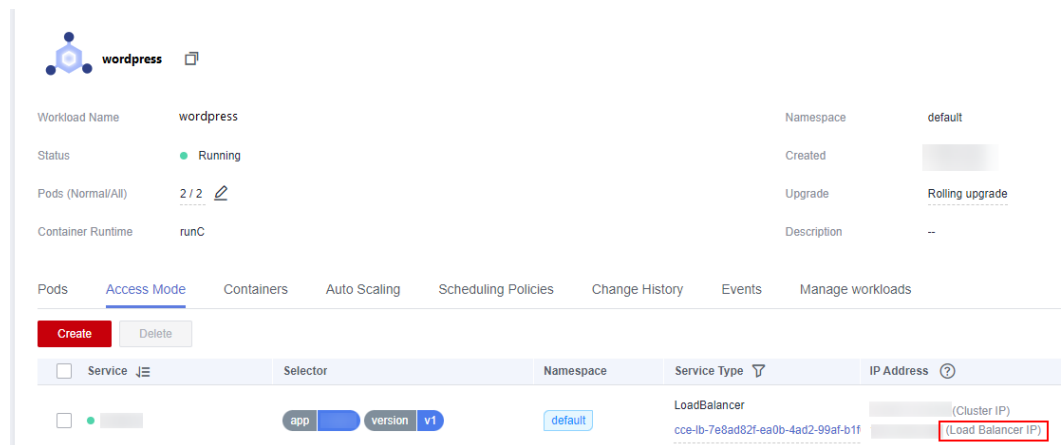
----End

Accessing WordPress

Step 1 Obtain the external access address of WordPress.

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

Figure 5-12 Accessing WordPress



Step 2 Enter the external access address in the address box of a browser to connect to the workload.

The following figure shows the accessed WordPress page.

Figure 5-13 WordPress workload

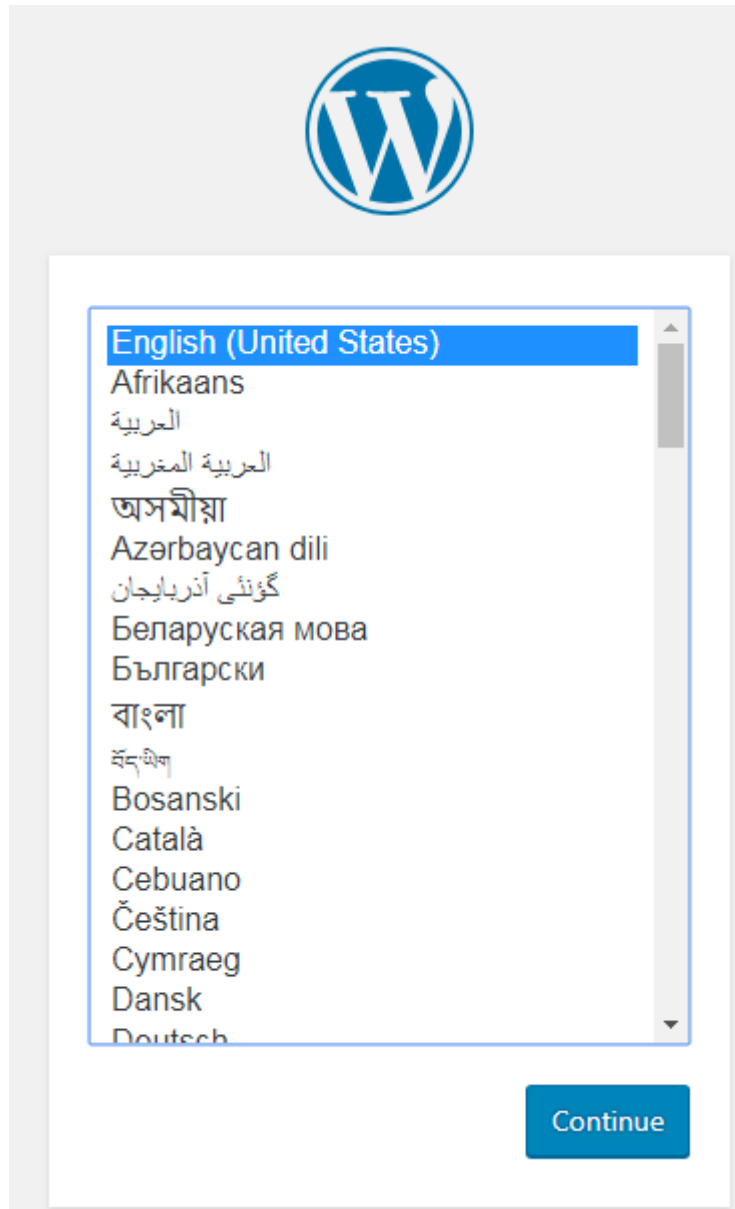
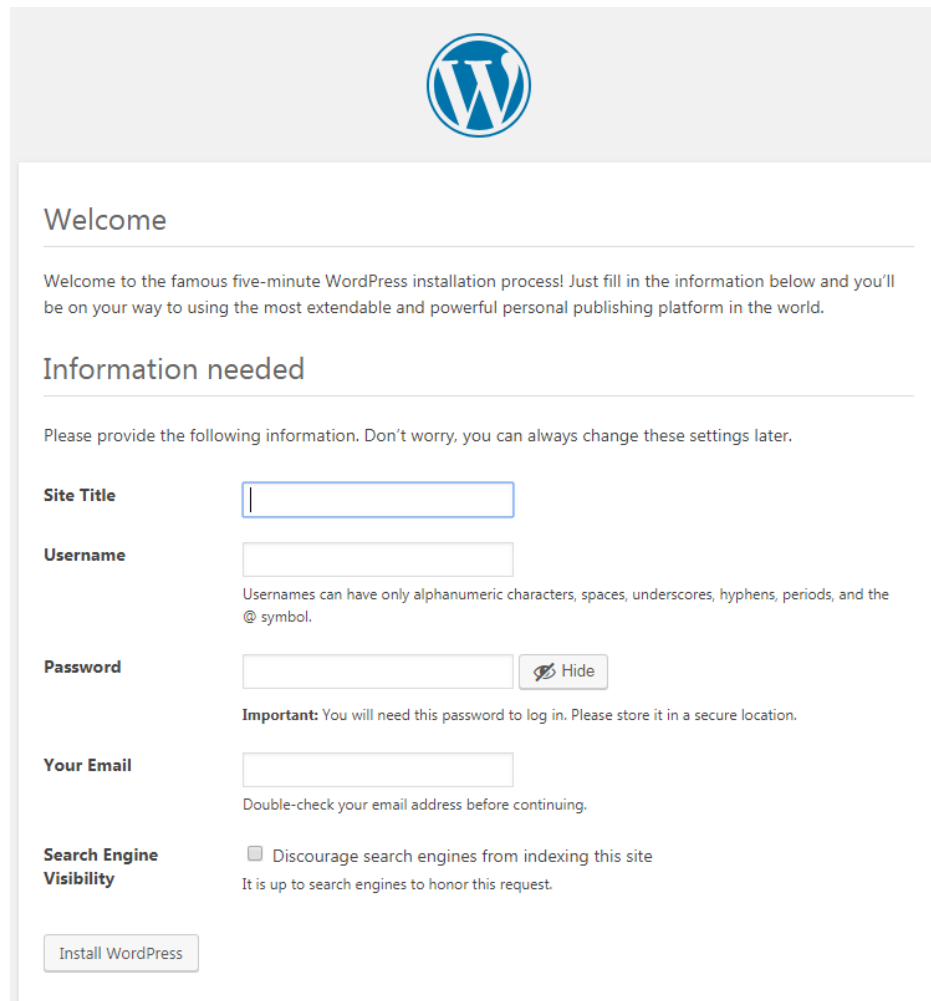


Figure 5-14 WordPress



----End

Deleting Resources

Until now, you have completed all the Getting Started walkthroughs and have understood how to use CCE. Fees are incurred while nodes are running. If the clusters used in the Getting Started walkthroughs are no longer in use, perform the following steps to delete them. If you will continue the CCE walkthroughs, retain the clusters.


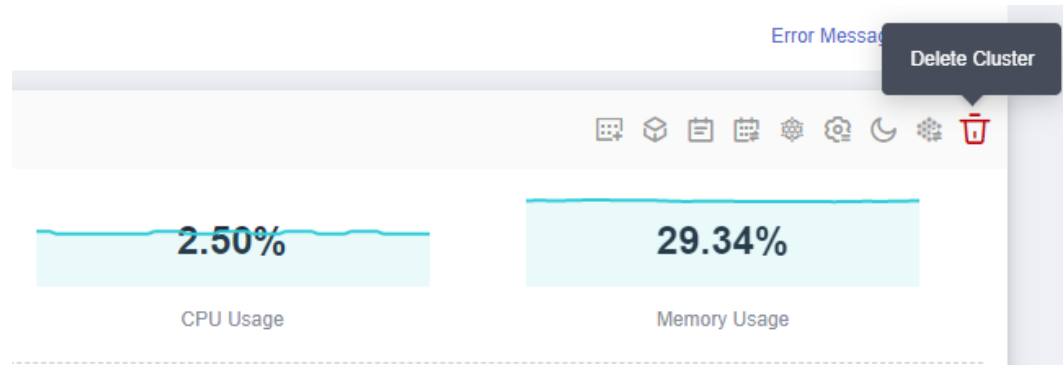
- Step 1** Log in to the CCE console.
- Step 2** In the navigation pane on the left, choose **Clusters**.
- Step 3** Click  next to a cluster and delete the cluster as prompted.

Figure 5-15 Deleting a cluster



----End

6 Deploying WordPress Using Helm

Helm is a package manager that simplifies the deployment, upgrade, and management of Kubernetes applications. Helm uses charts (a packaging format that defines Kubernetes resources) to encapsulate all elements deployed by Kubernetes, including application code, dependencies, configuration files, and deployment instructions. With Helm, you can easily deploy and manage complex Kubernetes applications, simplifying application development and deployment.

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

Prerequisites



You have created a CCE cluster that contains a node with 4 vCPUs and 8 GiB memory. The node has an EIP to pull the WordPress image from an external repository. For details on how to create a cluster, see [Creating a Kubernetes Cluster](#).

Preparations

1. Download and configure kubectl to connect to the cluster.
Log in to the CCE console and click the target cluster name. In the **Connection Information** area of the **Cluster Information** page, click **Configure** next to **kubectl** and configure kubectl as instructed.

Figure 6-1 kubectl

Connection Information

Private IP	https://192.168.0.198:5443 
EIP	-- Bind
Custom SAN	-- 
kubectl	Configure
Certificate Authentication	X.509 certificate Download

2. Install Helm 3.

Deploying WordPress

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
```

NOTE

- mariadb uses a persistent volume (PV) to store data. The PV is automatically created with an EVS disk of 10 GiB by configuring **StorageClassName**.
- Data persistence is not required for WordPress. Set **persistence.enabled** to **false** for the data volume.

Information similar to the following is displayed:

```
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
```

** Please 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. Login with the following credentials below to see your blog:

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

The command output shows how to obtain the WordPress URL and the username and password for logging in to the WordPress background.

----End

Accessing WordPress

Step 1 Modify the WordPress Service configuration.

Additional annotation configuration is required for using a LoadBalancer Service in CCE. However, **bitnami/wordpress** does not have this configuration. You need to manually modify the configuration.

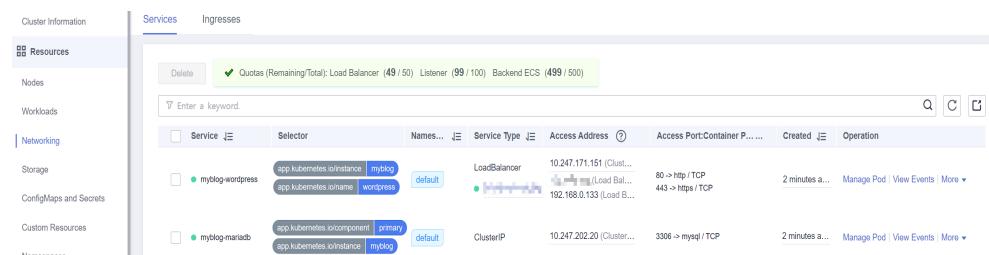
kubectl edit svc myblog-wordpress

Add **kubernetes.io/elb.autocreate** and **kubernetes.io/elb.class** to **metadata.annotations** and save the modification. The two annotations are used to create a shared load balancer so that the WordPress workload can be accessed through 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 Obtain the public network load balancer IP address of Service **myblog-wordpress** and access it.

On the **Services** tab, locate **myblog-wordpress** and find its public network load balancer IP address.



In the address box of a browser, enter **<Public IP address of the load balancer>:80** to access WordPress.

User's Blog!

Sample Page

Mindblown: a blog about philosophy.

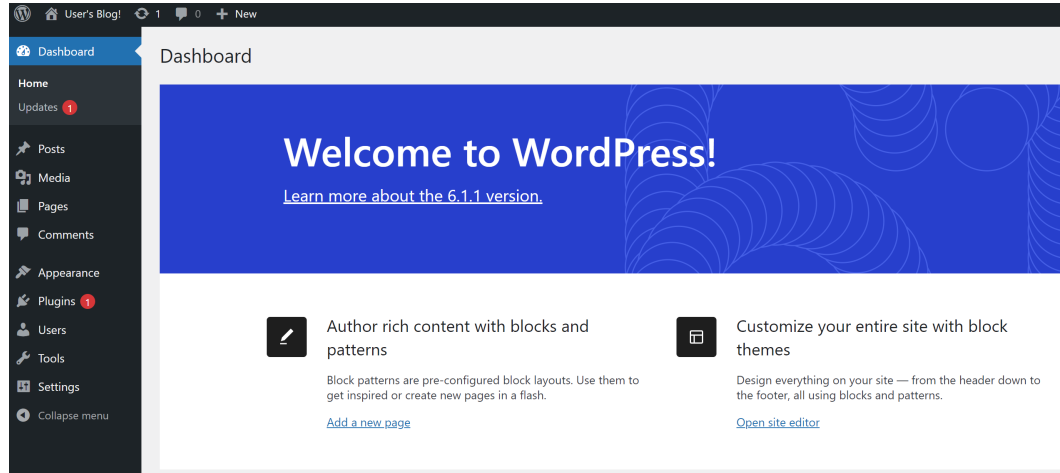
Hello world!

Welcome to WordPress. This is your first post.
Edit or delete it, then start writing!

March 27, 2023

In the address box of a browser, enter **<Public IP address of the load balancer>:80/login** to access the WordPress background as user **user**. Run the following command to obtain the password of **user**:

```
kubectl get secret --namespace default myblog-wordpress -o jsonpath="{.data.wordpress-password}" | base64 -d
```



----End

Deleting the WordPress Workload

Step 1 Run the following command to delete the WordPress workload:

```
helm delete myblog
```

Information similar to the following is displayed:
release "myblog" uninstalled

Step 2 Delete the PersistentVolumeClaim (PVC) used by the mariadb component in WordPress.

```
kubectl delete pvc data-myblog-mariadb-0
```

Information similar to the following is displayed:
persistentvolumeclaim "data-myblog-mariadb-0" deleted

----End