## **Cloud Container Instance**

## **Best Practice**

 Issue
 01

 Date
 2024-05-24





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

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

## **Trademarks and Permissions**

NUAWEI 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 Auto Scaling	1
1.1 Elastic Scaling of CCE Pods to CCI	1
2 Workload Creation	4
2.1 Overview	4
2.2 Using docker run Commands to Run Containers	4
2.3 Using the CCI Console to Create Workloads	6
2.4 Calling APIs to Create Workloads	
2.5 Configuring Dockerfile Parameters for CCI	17
3 Workload Management	20
3.1 Performing Graceful Rolling Upgrade for CCI Applications	20
3.2 Exposing Basic Pod Information to Containers Through Environment Variables	24
3.3 Configuring Kernel Parameters	26
3.4 Resizing /dev/shm	27

# Auto Scaling

## 1.1 Elastic Scaling of CCE Pods to CCI

CCE Cloud Bursting Engine for CCI functions as a virtual kubelet to connect Kubernetes clusters to APIs of other platforms. This add-on is mainly used to extend Kubernetes APIs to serverless container services such as Huawei Cloud CCI.

With this add-on, you can scale Deployments, StatefulSets, Jobs, and CronJobs running in CCE clusters to **Cloud Container Instance (CCI)** during peak hours. In this way, you can reduce consumption caused by cluster scaling.

## Installing the Add-on

1. Log in to the CCE console.

Install Add-on

- 2. Click the name of the target CCE cluster to go to the cluster console.
- 3. In the navigation pane on the left, choose **Add-ons**.
- 4. Select the CCE Cloud Bursting Engine for CCI add-on and click Install.
- 5. Configure the add-on parameters.

CCE CI	oud Bursting Engine for CCI Scheduling and Elasticity	Quick
An add-or	n that schedules CCE pods onto CCI clusters	
Version	1.5.0 v	
Specifications	5	
Add-on Specifica	tions Single HA Custom Resources	
Pods	1	
Parameters		
After the the CCI c	plug-in is installed, if the workload instance (Pod) is scheduled to the CCI service, it will be billed according to harging standard.	
Networking	Pods in the CCE cluster can communicate with pods in the CCI cluster through Kubernetes services.	
Subnet	subnet-3833 (192.168.240.0/20) V Q Available Subnet IP Addresses: 4,084	
	Pods scheduled to CCI occupy the IP addresses in the selected subnet. Plan the CIDR block properly to ensure IP provisioning.	

Table	1-1	Add-on	parameters
-------	-----	--------	------------

Paramete r	Description
Version	Add-on version. There is a mapping between add-on versions and CCE cluster versions. For more details, see "Change History" in <b>CCE Cloud Bursting Engine for CCI</b> .
Specificati ons	Number of pods required for a workload.
Networki ng	If this option is enabled, pods in a CCE cluster can communicate with the pods in CCI. For details, see <b>Networking</b> .

## Creating a Workload

- 1. Log in to the CCE console.
- 2. Click the name of the target CCE cluster to go to the cluster console.
- 3. In the navigation pane on the left, choose **Workloads**.
- 4. Click Create Workload. For details, see Creating a Workload.
- Specify basic information. Set Burst to CCI to Force scheduling. For more information about scheduling policies, see Scaling Pods to CCI.

Basic Info				
Workload Type	, Deployment 📦 StadefulSet 🔹 DaemonSet 🗎 Job	Cron Job		
	Switching workload type requires reconfiguring workload parameters.			
Workload Name	Einter a workload name.		Cluster Name wotest CCE Standard	
Namespace	default v Q Create Namespace		Description A maximum of 200 characters are supported.	
Pods	- <b>2</b> +			0200
Low-priority			Time Zone	
services	In the offline hybrid deployment scenario, high-priority online services and low-priority offline services are deployed on the same node to		Synchronization Allows containers to use the same time zone as the node	where they run. (This function is realized by the
	improve deployment density. When online services require more resources, the system automatically suppresses low-priority offine		local disks mounted to the containers. Do not modify or de	iete the local disks.)
	services to guarantee high-priority online services and improve resource utilization.			
	Low-priority services cannot be enabled because the Volcano plug-in is not installed in the cluster/install now Q			
Burst to CCI	Disable scheduling Automatic scheduling Local priority scheduling Force scheduling			
	Bearer Node bursting-node virtual-kubelet			
	Supports the rapid elastic creation of Pods to the cloud container instance CCI service in short-term high load scenarios to reduce			

- 6. Configure the container parameters.
- 7. Click Create Workload.
- 8. On the **Workloads** page, click the name of the created workload to go to the workload details page.
- 9. View the node where the workload is running. If the workload is running on a CCI node, it has been scheduled to CCI.

## Uninstalling the Add-on

- 1. Log in to the CCE console.
- 2. Click the name of the target CCE cluster to go to the cluster console.
- 3. In the navigation pane on the left, choose **Add-ons**.
- 4. Select the CCE Cloud Bursting Engine for CCI add-on and click Uninstall.

Ð	CCE Cloud Bursti     Running	ng Engine for C	CI
Version 1.5.	0 Updated 2 min	utes ago	
ର୍ View Det	ails	Edit	Jninstall

 Table 1-2 Special scenarios for uninstalling the add-on

Scenario	Symptom	Description
There are no nodes in the CCE cluster that the bursting add-on needs to be uninstalled from.	Failed to uninstall the bursting add-on.	If the bursting add-on is uninstalled from the cluster, a job for clearing resources will be started in the cluster. To ensure that the job can be started, there is at least one node in the cluster that can be scheduled.
The CCE cluster is deleted, but the bursting add-on is not uninstalled.	There are residual resources in the namespace on CCI. If the resources are not free, additional expenditures will be generated.	The cluster is deleted, but the resource clearing job is not executed. You can manually clear the namespace and residual resources.

For more information about the bursting add-on, see **CCE Cloud Bursting Engine for CCI**.

# **2** Workload Creation

## 2.1 Overview

You can create workloads on the CCI console or by calling APIs. What are the differences between the two methods and running the **docker run** commands?

This chapter uses the WordPress and MySQL as examples to compare the three methods.

The 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 the WordPress as a content management system. For more information about the WordPress, visit https://wordpress.org/.

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



## 2.2 Using docker run Commands to Run Containers

Docker is an open source engine that manages images and containers. A Docker image includes all dependencies required for running an application. The processes contained in the image are isolated from each other.

Docker containers are built on Docker images.

## **Preparing Images**

WordPress and MySQL images are general-purpose images and can be obtained from the container registry.

You can run the **docker pull** command on the device where the container engine is installed to download images.

docker pull mysql:5.7 docker pull wordpress

Run the **docker images** command to view the images. As shown in the following figure, two images exist on the local host.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
wordpress	latest	6a837ea4bd22	6 days ago	408MB
mysql		0d16d0a97dd1	5 weeks ago	372MB

## **Running Containers**

You can use the container engine to run the WordPress and MySQL containers, and use the **--link** parameter to connect the two containers. In this way, the WordPress container can access the MySQL container without code changes.

Run the following command to run the MySQL container:

docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=\*\*\*\*\*\*\*\* -e MYSQL\_DATABASE=wordpress -d mysql:5.7

The parameters are described as follows:

- --name: specifies the container name as some-mysql.
- -e: specifies the environment variable of the container. In this example, the value of MYSQL\_ROOT\_PASSWORD is \*\*\*\*\*\*\*\* (replace \*\*\*\*\*\*\* with your password). The environment variable MYSQL\_DATABASE indicates the name of the database to be created when the image is started. Its value is wordpress in this example.
- -d: indicates that the container runs in the backend.

Run the following command to run the WordPress container:

docker run --name some-wordpress --link some-mysql:mysql -p 8080:80 -e WORDPRESS\_DB\_PASSWORD=\*\*\*\*\*\*\*\* -e WORDPRESS\_DB\_USER=root -d wordpress

The parameters are described as follows:

- --name: specifies the container name as some-wordpress.
- --link: connects the some-wordpress container to the some-mysql container and changes the name of the some-mysql container to mysql. --link provides an easy way to connect two containers. Alternatively, you can configure the environment variable WORDPRESS\_DB\_HOST of the some-wordpress container to access the IP address and port of the mysql container.
- **-p**: specifies ports for mapping. In this example, port 80 of the container is mapped to port 8080 of the host.
- -e: specifies the environment variable of the container. In this example, the value of WORDPRESS\_DB\_PASSWORD is \*\*\*\*\*\*\*\* (replace \*\*\*\*\*\*\*\* with your

password). The value of **WORDPRESS\_DB\_PASSWORD** must be the same as that of **MYSQL\_ROOT\_PASSWORD** because the WordPress requires a password to access the MySQL database. **WORDPRESS\_DB\_USER** indicates the username for accessing the MySQL database. Set it to **root**.

• -d: indicates that the container runs in the backend.

After the WordPress runs, you can access WordPress blogs through http:// 127.0.0.1:8080.



## 2.3 Using the CCI Console to Create Workloads

**Using docker run Commands to Run Containers** describes how you can run the WordPress workload by running the **docker run** commands. However, it is not convenient to use a container engine in many scenarios, such as auto scaling and rolling upgrade.

CCI provides a serverless container engine that frees you from managing clusters or servers. CCI delivers agility and high performance with only three steps. It enables you to create stateless workloads (Deployments) and stateful workloads (StatefulSets). It also enhances container security isolation and supports fast workload deployment, elastic load balancing, auto scaling, and blue-green deployment based on the Kubernetes workload model.

## Creating a Namespace

- **Step 1** Log in to the CCI console. In the navigation pane, choose **Namespaces**.
- **Step 2** Click **Create** for the target namespace type.
- **Step 3** Enter a namespace name.
- Step 4 Configure a VPC.

Select an existing VPC or create one. Recommended CIDR blocks for the new VPC are 10.0.0/8-24, 172.16.0.0/12-24, and 192.168.0.0/16-24.

**Step 5** Configure a subnet.

Ensure that there are sufficient available IP addresses in the subnet. If IP addresses are insufficient, workload creation will fail.

Step 6 Click Create.

----End

## Creating a MySQL Workload

- Step 1 Log in to the CCI console. In the navigation pane, choose Workloads > Deployments. On the page displayed, click Create from Image.
- **Step 2** Specify basic information.
  - Workload Name: mysql
  - Namespace: Select the namespace created in Creating a Namespace.
  - **Pods**: Change the value to **1** in this example.
  - **Pod Specifications**: Select the general-computing pod with 0.5-core CPU and 1 GiB of memory.

* Workload Name	mysql	×			
	Enter 1 to 63 characters starting and en hyphen.	ding with a letter or digit. Only lowercase lett	ters, digits, hyphens (-), and periods (.)	are allowed. Do not enter two co	onsecutive periods or a period adjacent to a
* Namespace	😚 🛞 gene-test1	- C C	reate Namespace		
	Available General-computing   Computing   Computing	CI-VPC-1928286404 192.168.0.0/16			
Description	Enter a description.				
				0/250	
* Pods	- 1 +				
* Pod Specifications	General-computing				
	1X	2X	4X	X8	
	CPU 0.5 cores	CPU 1 core	CPU 2 cores	CPU 4 cores	🖉 Custom
	Memory 1 GB	Memory 2 GB	Memory 4 GB	Memory 8 GB	

- Container Settings
  - a. On the **Open Source Images** tab, search for the mysql image and click **Use Image**.

* Configure Container	Select Image + Add Container	
	Note: Image data is sourced from Software Repository for Container (SWR). My Images Open Source Images Shared Images	
	mysql X Q C	
	My SQL	
	Download: 999231875	

b. Set image parameters. Specifically, set the image version to **5.7**, CPU to **0.50**, and memory to **1.000**.

* Configure Container	mysql       Speck     0.5 corres   1 GB
	Image Name mySQI Change Image
	Image Version 5.7 V
	Container Name container-0
	CPU (cores) - 0.50 +
	Memory (GB) - 1.000 +
	Standard Output: ⑦ 🥢 Application Operations Management (AOM) will bill you according to the actual usage. File Collection
	✓ Advanced Settings

c. In the advanced settings, enter the environment variable MYSQL\_ROOT\_PASSWORD and its value. The value is the password of the MySQL database. You need to set the password by yourself.

Environment Variables: Environment variables are set in	the container runtime environment, and can be modified after a	application deployment.	^
Add Manually Variable Reference			
Variable Name	Variable Value	Operation	
MYSQL_ROOT_PASSWORD		Delete	
Add Environment Variable			

Step 3 Click Next: Configure Access Settings. Set Access Type to Intranet access. In this case, the workload can be accessed by other workloads in CCI by using Service name:Port. In addition, set Service Name to mysql, and map workload access port 3306 to container port 3306 (default access port of the MySQL image).

In this way, other workloads in CCI can access the MySQL workload by using **mysql:3306**.

Access Mode					
Access Type	Intranet access	Internet access	Do not use		
	For intranet access, you can configure the intranet. There are 2 access modes	a workload domain name or internal don s: service and ELB. Learn how to configu	nain name (or ELB VIP) for the current w re intranet access for a workload.	vorkload so that the workload can be ac	cessed by other workloads in
* Access Mode	Service	ELB			
	In the service access mode, the curren supported.	t workload can be accessed by other wo	rkloads in the intranet based on the wor	kload domain name and workload port.	The TCP/UDP protocols are
* Service Name	mysql				
<ul> <li>Workload Port Settings</li> </ul>	(Sets the mapping between the workloai instance:container port.)	d access port and container port. Access	s requests are forwarded from the workld	ad domain name:workload access port	to the container
Protocol	Worklo	ad Access Port	Container Port		Operation
TCP	▼ 3306	i	3306		Delete

**Step 4** Click **Next**. On the page that is displayed, check the configurations and then click **Submit**.

In the workload list, if the workload is in the **Running** state, the workload is successfully created.

----End

## Creating a WordPress Workload

- Step 1 Log in to the CCI console. In the navigation pane, choose Workloads > Deployments. On the page displayed, click Create from Image.
- **Step 2** Specify basic information.
  - Workload Name: wordpress
  - Namespace: Select the namespace created in Creating a Namespace.
  - **Pods**: Change the value to **2** in this example.
  - **Pod Specifications**: Select the general-computing pod with 0.5-core CPU and 1 GiB of memory.

* Workload Name	wordproce	~			
	Enter 1 to 63 characters starting and ending with hyphen.	h a letter or digit. Only lowercase lette	ers, digits, hyphens (-), and periods (.)	are allowed. Do not enter two conse	ecutive periods or a period adjacent to a
* Namespace	🛇 😻 gene-test1	• C 0	eate Namespace		
	Available General-computing   CCI-VPC	- <b>1928286404</b> 192.168.0.0/16			
Description	Enter a description.				
* Pods	- 2 +			0/250	
* Pod Specifications	General-computing				
	1X	2X	4X	8X	
	CPU 0.5 cores	CPU 1 core	CPU 2 cores	CPU 4 cores	🖉 Custom
	Memory 1 GB	Memory 2 GB	Memory 4 GB	Memory 8 GB	

- Configure Container
  - a. On the **Open Source Images** tab page, search for the wordpress image and click **Use Image**.

* Configure Container	Select Image + Add Container	
	Note: Image data is sourced from Software Repository for Container (SWR). My Images Open Source Images Shared Images	
	wordpress X Q	C
	wordpress	
	Downloads: 37318115 Use Image	

b. Set image parameters. Specifically, set the image version to **php7.1**, CPU to **0.50**, and memory to **1.000**.

Configure Container	Spect. 0.5 cores   1 GB
	Image Name WOrdpress Change Image Image Version php7.1
	Container Name container-0
	CPU (cores) - 0.50 +
	Memory (GB) - 1.000 +
	Standard Output 💿 💽 🔺 Application Operations Management (AOM) will bill you according to the actual usage. File Collection
	Advanced Settings

c. In the **Advanced Settings** area, expand **Environment Variables** and add environment variables to enable the wordpress application to access the MySQL database.

Add Manually	Variable Reference		
ariable Name		Variable Value	Operation
WORDPRESS	_DB_HOST	:3306	Delete
WORDPRESS	_DB_PASSWORD		Delete

Table 2-1 Description of environment variables

Variable Name	Variable Value/Variable Reference
WORDPRESS_DB_HOS T	Address for accessing the MySQL database. Example: 10.***.***:3306
WORDPRESS_DB_PAS SWORD	Password for accessing the MySQL database. The password must be the same as the MySQL password set in <b>Creating a MySQL</b> Workload.

#### Step 3 Click Next: Configure Access Settings.

Set Access Type to Internet access and Service Name to wordpress, and select a load balancer. If no load balancers are available, click Create Share Load Balancer. Set ELB Protocol to HTTP and ELB Port to 9012. In the Workload Port Settings area, set a mapping between workload access port 8080 and container port 80 (default access port in the WordPress image). In the HTTP Route Settings area, set Mapping Path to / (so http://Load balancer IP address.Port can be used to access the WordPress) and Workload Access Port to 8080.

Access Mode					
Access Type	Intranet access	Internet access	Do not use		
	An Internet access portal is provided for the WordPress). Learn how to configure Internet.	ne workload. Access requests are forw net access for a workload.	arded through the HTTP protocol and l	JRL. This access mode is su	itable for frontend services (such as
* Service Name	wordpress				
* Load Balancer 🕐	elb-test 🔹 C	Create an enhanced load balancer	and click refresh to make it available for	selection.	
ELB Protocol	HTTP/HTTPS	TCP/UDP			
★ Ingress Name	wordpress				
Public Domain	Enter a maximum of 34 characters for e	ach level of the domain name. /			
Name	Access the workload through the public de this parameter is left unspecified, the workload the second secon	omain name. You need to purchase th doad is accessed through the ELB Elf	e public domain name and point the res <sup>9</sup> address.	olved domain name to the E	IP address of the selected load balancer. If
* ELB Port	HTTP • 9	012			
	To provide HTTPS-based Internet access	select HTTPS. This port is used to a	ccess the workload.		
* Workload Port Protocol	TCP				
<ul> <li>Workload Port Settings</li> </ul>	(Sets the mapping between the workload a instance:container port.)	access port and container port. Access	requests are forwarded from the workl	oad domain name:workload	access port to the container
Workload Access P	Port	Container Port			Operation
8080		80			Delete
Add Port					
₭ HTTP Route ( Settings a	Set the route relationship from the mapping address):External port/mapping path to the v	path to the backend workload access vorkload domain name:workload acce	port. The Internet access requests are ss port.	forwarded from the http://pu	blic domain name (or ELB EIP
Mapping Path		Workload Access	Port (TCP Protocol)		Operation
1		8080		•	Delete

## **Step 4** Click **Next**. On the page that is displayed, check the configurations and then click **Submit**.

In the workload list, if the workload is in the **Running** state, the workload is successfully created. In this case, you can click the workload to go to its details page.

In the **Access Settings** area, click **Internet Access** and view the access address (*Load balancer IP address.Port*).

Access Settings				
Internet access Intranet access	Events			
Public Network Access Address	EIP	Internal Access Address	Internal Workload Domain Name Address	Protocol
http:/ :9012/	0.000	http://192.168.24.162:9012/	wordpress:8080	HTTP

----End

## 2.4 Calling APIs to Create Workloads

CCI supports Kubernetes APIs. Compared with using the console to create workloads, calling APIs is much easier.

In Kubernetes, a pod is the minimum unit for container running and can encapsulate one or more containers, storage resources, and an independent network IP address. In practice, pods are rarely created directly. Kubernetes uses controllers such as Deployment and StatefulSet to manage pods. In addition, Kubernetes uses Services to define pods and their access policies, and uses ingresses to manage external access. For more information about Kubernetes resources, see *Cloud Container Instance Developer Guide*.

For the WordPress application, you can call APIs to create a series of resources, as shown in the following figure.

- MySQL: Create a Deployment to deploy the MySQL, and create a Service to define the access policy of the MySQL.
- WordPress: Create a Deployment to deploy the WordPress, and create a Service and an ingress to define the access policy of the WordPress.



{

}

## Namespace

**Step 1** Call the API described in **Creating a Namespace** to create a namespace and specify a namespace type.

```
"apiVersion": "v1",
"kind": "Namespace",
"metadata": {
    "name": "namespace-test",
    "annotations": {
        "namespace.kubernetes.io/flavor": "gpu-accelerated"
    }
},
"spec": {
    "finalizers": [
        "kubernetes"
    ]
}
```

**Step 2** Call the API described in **Creating a Network** to create a network, and associate the network with a Virtual Private Cloud (VPC) and subnet.



----End

## **MySQL**

Step 1 Call the API described in Creating a Deployment to deploy the MySQL.

- Set the Deployment name to mysql.
- Set the pod label to **app:mysql**.
- Use the mysql:5.7 image.
- Set the value of the environment variable **MYSQL\_ROOT\_PASSWORD** to \*\*\*\*\*\*\*\* (replace \*\*\*\*\*\*\* with your password).

```
{
    "apiVersion": "apps/v1",
    "kind": "Deployment",
    "metadata": {
        "name": "mysql"
    },
    "spec": {
        "replicas": 1,
        "selector": {
    }
}
```

```
"matchLabels": {
           "app": "mysql"
        }
     },
"template": {
        "metadata": {
           "labels": {
              "app": "mysql"
           }
        },
         "spec": {
           "containers": [
              {
                 "image": "mysql:5.7",
                 "name": "container-0",
                 "resources": {
                    "limits": {
"cpu": "500m",
                       "memory": "1024Mi"
                   },
"requests": {
....": "50
                       "cpu": "500m",
                       "memory": "1024Mi"
                   }
                },
"env": [
                   {
                      "name": "MYSQL_ROOT_PASSWORD",
                       "value": "*******"
                   }
                ]
              }
           ],
           "imagePullSecrets": [
              {
                 "name": "imagepull-secret"
              }
          ]
       }
     }
  }
}
```

```
Step 2 Call the API described in Creating a Service to create a Service, and define the access policy for the pod created in Step 1.
```

- Set the Service name to **mysql**.
- Select the pod whose label is app:mysql to associate the pod created in Step
   1.
- Map workload access port 3306 to container port 3306.
- The access type of the Service is **ClusterIP**, that is, ClusterIP is used to access the Service inside the cluster.

```
"name": "service0",
"targetPort": 3306,
"port": 3306,
"protocol": "TCP"
}
],
"type": "ClusterIP"
}
}
```

----End

## WordPress

Step 1 Call the API described in Creating a Deployment to deploy the WordPress.

- Set the Deployment name to wordpress.
- Set the value of replicas to **2**, indicating that two pods are created.
- Set the pod label to app:wordpress.
- Use the wordpress:latest image.
- Set the value of the environment variable **WORDPRESS\_DB\_PASSWORD** to \*\*\*\*\*\*\*\* (replace \*\*\*\*\*\*\* with your password). This password must be the same as **MYSQL\_ROOT\_PASSWORD** set for the MySQL.

```
{
  "apiVersion": "apps/v1",
  "kind": "Deployment",
  "metadata": {
"name": "wordpress"
  },
   "spec": {
      "replicas": 2,
     "selector": {
         "matchLabels": {
           "app": "wordpress"
        }
     },
"template": {
        "metadata": {
           "labels": {
"app": "wordpress"
           }
        },
         "spec": {
            "containers": [
              {
                 "image": "wordpress:latest",
                 "name": "container-0",
                 "resources": {
                    "limits": {
"cpu": "500m",
                       "memory": "1024Mi"
                    },
                    "requests": {
                       "cpu": "500m",
                       "memory": "1024Mi"
                   }
                },
"env": [
                    {
                       "name": "WORDPRESS_DB_PASSWORD",
                       "value": "*******
                    }
                 ]
          }
],
```

}

- **Step 2** Call the API described in **Creating a Service** to create a Service, and define the access policy for the pod created in **Step 1**.
  - Set the Service name to wordpress.
  - Select the pod whose label is app:wordpress to associate the pod created in Step 1.
  - Map workload access port 8080 to container port 80. For the WordPress image, port 80 is the default externally exposed port.
  - The access type of the Service is **ClusterIP**, that is, ClusterIP is used to access the Service inside the cluster.

```
"apiVersion": "v1",
"kind": "Service",
"metadata": {
"name": "wordpress",
   "labels": {
      "app": "wordpress"
   }
},
 "spec": {
   "selector": {
      "app": "wordpress"
   },
   "ports": [
      {
         "name": "service0",
         "targetPort": 80,
         "port": 8080.
          "protocol": "TCP"
      }
    'type": "ClusterIP"
}
```

- **Step 3** Call the API described in **Creating an Ingress** to create an ingress to define the external access policy of the WordPress. In this step, you need to configure a load balancer that is in the same VPC as the WordPress.
  - metadata.annotations.kubernetes.io/elb.id: ID of the load balancer
  - metadata.annotations.kubernetes.io/elb.ip: IP address of the load balancer
  - metadata.annotations.kubernetes.io/elb.port: Port configured for the load balancer
  - spec.rules: a set of rules for accessing the Service. path lists the paths for accessing the Service, for example, "/". Each path is associated with a backend (for example, wordpress:8080). A backend represents a combination of service:port. Ingress traffic will be forwarded to the corresponding backend.

After the configuration is complete, the traffic destined for the load balancer (*Load balancer IP address:Port*) is transmitted to the wordpress:8080 Service. Because the Service is associated with the WordPress pod, the traffic finally accesses the WordPress container deployed in **Step 1**.

```
"apiVersion": "extensions/v1beta1",
"kind": "Ingress",
"metadata": {
"name": "wordpress",
   "labels": {
"app": "wordpress",
      "isExternal": "true",
      "zone": "data"
  },
"annotations": {
      "kubernetes.io/elb.id": "2d48d034-6046-48db-8bb2-53c67e8148b5",
      "kubernetes.io/elb.ip": "10.10.10.10",
      "kubernetes.io/elb.port": "9012"
   }
},
"spec": {
   "rules": [
     {
         "http": {
            "paths": [
              {
                  "path": "/",
                  "backend": {
                     "serviceName": "wordpress",
                     "servicePort": 8080
                 }
              }
       }
     }
  ]
}
```

----End

## 2.5 Configuring Dockerfile Parameters for CCI

## Scenario

Dockerfiles are generally used to customize images. A Dockerfile is a text file that contains instructions, each of which builds an image layer.

This topic describes the CCI settings corresponding to the Dockerfile configurations.

## **Using Dockerfile Parameters in CCI**

The following uses an example to describe the relationship between them.

FROM ubuntu:16.04 ENV VERSION 1.0 VOLUME /var/lib/app EXPOSE 80 ENTRYPOINT ["./entrypoint.sh"] CMD ["start"] In the preceding example, the Dockerfile contains common parameters, including **ENV**, **VOLUME**, **EXPOSE**, **ENTRYPOINT**, and **CMD**. These parameters can be configured for CCI as follows:

• **ENV** indicates an environment variable. Corresponding to **ENV VERSION 1.0** in the Dockerfile, set **Environment Variables** in the advanced settings as follows when creating a workload on the CCI console.

nvironment Variable	s: Environment variables are set in the	container runtime environment, and can be modified after ap	oplication deployment.
Add Manually	Variable Reference		
Variable Name		Variable Value	Operation
VERSION		1.0	Delete

• **VOLUME** indicates a container volume. Generally, this parameter is used together with **docker run -v host path**:container volume path.

For CCI, Elastic Volume Service (EVS) disks can be mounted to containers. You only need to add EVS volumes, and configure their sizes and mount paths (that is, container volume paths) when creating workloads.

EVS Volumes	SFS Volumes	SFS Turbo Volumes			
ect existing EVS volu	umes, create a new EV	'S volume, or import EVS o	disks to generate EVS volume	s. C	
Name		Capacity (GB)	Container Path	Туре	Operation
Aut 🔻 cci	-evs-kdikl7mv-9zx6	- 10 +	/var/lib/app	High I/O	Delete

• **ENTRYPOINT** and **CMD** correspond to the startup command of the advanced settings in CCI. For details, see **Setting Container Startup Commands**.

		or commands. Learn	now to conligure starte	ip commands.	1
Executable Command	./entrypoint.sh	Example	Binary Mode	Bash Mode	
Parameters	Switch to single line input mode stsrt Add	Executable Command	python //	var/tf_mnist/mnist_with_s	
	0	Parameters	01log_dir	=/trainlearning_rate=0.	

 EXPOSE indicates an exposed port. Generally, this parameter is used together with docker run -p <host port>:<container port> when a container is started. To set an exposed port for a CCI container, you only need to configure the Workload access port:Container port when creating a workload. In this way, you can access the container through the Workload domain name:Workload access port.

*	Service Name	арр				
*	Workload Port Settings	(Sets the mapping between the instance:container port.)	workload access port and container port. Acc	ess requests are fo	warded from the workload domain name:workload access port	to the container
	Protocol		Workload Access Port		Container Port	Operation
	TCP	•	8080		80	Delete



## 3.1 Performing Graceful Rolling Upgrade for CCI Applications

## Scenario

When you deploy a workload in CCI to run an application, the application is exposed as a LoadBalancer Service or ingress, and connected to a dedicated ELB load balancer to allow access traffic to reach the containers directly. When rolling upgrade or auto scaling is performed on the application, your pods may fail to work with ELB and 5xx errors may occur. This section guides you to configure container probes and readiness time to achieve graceful upgrade and auto scaling.

## Procedure

The following uses an Nginx Deployment as an example.

Step 1 On the CCI console, choose Workloads > Deployments in the navigation pane, and click Create from Image in the upper right corner.

#### Figure 3-1 Creating a Deployment

Deployments ⑦ 🕞 Namespace:	Available General-computing   vpc- 192.168.0.0/16		Create from YAML Create from Image
U Delete		All statuses •	Enter a workload name. Q BB III C
	HUAWEI CLOUD	HLT gateway Public network ELB + Physics network ELB (child wateway VC) + Bost - DECS	

- **Step 2** In the **Container Settings** area, click **Use Image** to select an image.
- Step 3 Click Advanced Settings of the image, click Health Check > Application Readiness Probe, and configure the probe.

#### Figure 3-2 Configuring the application readiness probe

Health Check: Health check regularly ch	ecks the health status of containers or applica	tions. Learn how to configure a health check.	
✓ Application Liveness Probe: Use	d to check whether the application is started.		
<ul> <li>Application Readiness Probe: U:</li> </ul>	ed to check whether the application is ready	for use.	
Time Window (s) 3 Timeout Period (s) 5	Provides a time windo	w for the probe to start after the workload is started. d. Default value: 1s.	
Command Line Script	HTTP Request Mode		
Path	Port	Protocol	
1	80	НТТР	•

#### **NOTE**

The probe checks whether your container is ready. If the container is not ready, requests will not be forwarded to the container.

**Step 4** Expand **Lifecycle** and configure the parameters of **Pre-Stop Processing** for the container.

#### Figure 3-3 Configuring lifecycle parameters

Lifecycle: Lifecycle scripts specify actions that applications take when a lifecycle event occurs. Learn how to configure a lifecycle script.								
Post-Start Processing	Pre-Stop Processing							
Triggered before the applicat Command Line Script	tion is stopped.		Example	Binary Mode	Bash Mode			
/bin/bash		Command	🗇 /run/star	t				
-c sleep 30	Delete	Parameter	🗇port=8					
⊕ Add		_						

#### **NOTE**

This configuration ensures that the container can provide services for external systems during its exit.

**Step 5** Click **Next: Configure Access Settings** and configure settings as shown in **Figure 3-4**.

Access Mode					
Access Type	Intranet access	Internet access	Do not use		
	An Internet access portal is provided fo WordPress). Learn how to configure Int	r the workload. Access requests are for ternet access for a workload.	warded through the HTTP protocol and	d URL. This access mode is su	uitable for frontend services (such as
* Service Name					
* Load Balancer	elb-b544 👻 (	Create Shared Load Balancer			
ELB Protocol	HTTP/HTTPS	TCP/UDP			
* Workload Port Protocol	ТСР	UDP			
* Workload Port ( Settings	(Set the mapping between the ELB port	and container port. Access requests ar	e forwarded from the workload domain	n name:ELB port to the conta	ainer instance:container port.)
ELB Port (not in use	2)	Container Port			Operation
6044		80			Delete
⊕ Add Port					

Figure 3-4 Configuring the access type and port

- Step 6 Click Next and complete the Deployment creation.
- **Step 7** Configure the minimum readiness time.

A pod is considered available only when the minimum readiness time is exceeded without any of its containers crashing.

In the upper right corner of the **Deployments** page, click **Create YAML** to configure the minimum readiness time as below.

- 1	04	V DI IV DASIL
		- ' -c'
		- sleep 30
		terminationMessagePath: /dev/termination-log
		terminationMessagePolicy: File
		imagePullPolicy: IfNotPresent
		restartPolicy: Always
		terminationGracePeriodSeconds: 30
		dnsPolicy: ClusterFirst
		securityContext: {}
		imagePullSecrets:
		- name: imagepull-secret
		schedulerName: default-scheduler
		dnsConfig: {}
		strategy:
		type: RollingUpdate
		rollingUpdate:
		maxUnavailable: 1
		maxSurge: O
		minReadySeconds: 10
		revisionHistoryLimit: 10
		progressDeadlineSeconds: 600
		status:
		observedGeneration: 2
		replicas: 2
		updatedReplicas: 2
		readyReplicas: 2
		availableKeplicas: 2
		conditions:
		- type: Available
		status: lrue l svil, min s (coost co composto traff)
		lastUpdatelime: 2021-08-24109:16:172
		lastiransitionlime: 2021-08-24109:16:172
		reason: MinimumnepiicasAvailable
		message. Deproyment has minimum availability.
		- type. Trogressing
		Status, irue
		Tastopdatellme. 2021-00-24109.16.252

Figure 3-5 Configuring the minimum readiness time

## D NOTE

- The recommended value of **minReadySeconds** is the expected time for starting the service container plus the duration from the time when the ELB service delivers the member to the time when the member takes effect.
- The value of **minReadySeconds** must be smaller than that of **sleep** to ensure that the new container is ready before the old container stops and exits.

Step 8 Test the application upgrade and auto scaling.

```
Prepare a client outside the cluster, and configure the detection script

detection_script.sh with the following content (100.85.125.90:7552 indicates the

public network address for accessing the Service):

#! /bin/bash

for ((;;))

do

    curl -1 100.85.125.90:7552 | grep "200 OK"

    if [ $? -ne 0 ]; then

        echo "response error!"

        exit 1
```

fi done

**Step 9** Run the detection script (**bash detection\_script.sh**) and trigger the rolling upgrade of the application on the CCI console. You can change the specifications of the container to trigger the rolling upgrade of the application.

#### Figure 3-6 Modifying container specifications

Workload Name											
* Pod Specifications	General-c	omputing									
		1X		2X			4)		8X		
	CPU	0.5 cores	CPU	1 core	1	CPU	2 cores	CPU	4 cores	D	Curtore
	Memory	1 GB	Memory	2 GB		Memory	4 GB	Memory	8 GB	<u>v</u>	Custom

If the access to the application is not interrupted, and the returned responses are all **200OK**, the graceful upgrade is successfully triggered.

----End

## **3.2 Exposing Basic Pod Information to Containers** Through Environment Variables

If you want a pod to expose its basic information to containers running in the pod, you can use the Kubernetes *Downward API* to inject environment variables. This section describes how to add environment variables to the definition of a Deployment or a pod to obtain the namespace, name, UID, IP address, region, and AZ of the pod.

When CCI creates a pod and allocates it to a node, the region and AZ information of the node is added to the pod's annotations.

In this case, the format of the pod's annotations is as follows:

apiVersion: v1 kind: Pod metadata: annotations: topology.kubernetes.io/region: "{{**region**}}" topology.kubernetes.io/zone: "{{**available-zone**}}"

topology.kubernetes.io/region indicates the region of the node.

topology.kubernetes.io/zone indicates the AZ of the node.

## **Deployment Configuration Example**

The following example shows how to use environment variables to obtain basic pod information.

kind: Deployment apiVersion: apps/v1 metadata: name: cci-downwardapi-test namespace: cci-test # Enter a specific namespace. spec: replicas: 2 selector: matchLabels: app: cci-downwardapi-test template: metadata: labels: app: cci-downwardapi-test spec: containers: - name: container-0 image: 'library/euleros:latest' command: - /bin/bash - '-c' - while true; do echo hello; sleep 10; done env: - name: MY\_POD\_UID valueFrom: fieldRef: fieldPath: metadata.uid - name: MY\_POD\_NAME valueFrom: fieldRef: fieldPath: metadata.name - name: MY\_POD\_NAMESPACE valueFrom: fieldRef: fieldPath: metadata.namespace - name: MY\_POD\_IP valueFrom: fieldRef: fieldPath: status.podIP - name: REGION valueFrom: fieldRef: fieldPath: metadata.annotations['topology.kubernetes.io/region'] - name: ZONE valueFrom: fieldRef: fieldPath: metadata.annotations['topology.kubernetes.io/zone'] resources: limits: cpu: 500m memory: 1Gi requests: cpu: 500m memory: 1Gi

When the Deployment starts, you can view the pod information exposed to the container through environment variables.

#### Figure 3-7 Basic pod Information



## **3.3 Configuring Kernel Parameters**

CCI uses Kata containers to build an industry-leading serverless container platform. Kata containers are isolated from the physical machine system kernel. They do not affect each other. kernel parameter optimization is a common practice in advanced service deployment scenarios. In a safe situation, CCI allows you to configure kernel parameters through a security context of a pod based on the solution recommended by the Kubernetes community, greatly improving the flexibility of service deployment. For details of security contexts, see *Configure a Security Context for a Pod or Container*.

In Linux, kernel parameters are usually configured through the sysctl interface. In Kubernetes, kernel parameters are configured through the sysctl security context of the pod. For details of sysctl, see *Using sysctls in a Kubernetes Cluster*. The security context is applied to all containers in the pod.

CCI allows you to modify the following kernel parameters:

kernel.shm\*, kernel.msg\*, kernel.sem, fs.mqueue.\*, net.\* (excluding net.netfilter.\* and net.ipv4.vs.\*)

In the following example, the pod's **securityContext** is used to set the sysctl parameters **net.core.somaxconn** and **net.ipv4.tcp\_tw\_reuse**.

apiVersion:v1 kind:Pod metadata: name: xxxxx namespace: auto-test-namespace spec: securityContext: sysctls: - name: net.core.somaxconn



Go to the container to check whether the configuration takes effect.

<pre>[root@master-2 ~]# kubectl get pod -n au</pre>	to-test-na	amespace							
NAME	READY	STATUS	RESTARTS	AGE					
cci-deployment-20225241-76dff9f854-6fwlm	1/1	Running	Θ	15m					
cci-deployment-20225241-76dff9f854-nwst7	1/1	Running	Θ	29m					
[root@master-2 ~]# kubectl exec -it cci-	deployment	-20225241-	76dff9f854-	nwst7	/bin/bash -n auto-test-namespace				
root@cci-deployment-20225241-76dff9f854-n	wst7:/#								
root@cci-deployment-20225241-76dff9f854-n	root@cci-deployment-20225241-76dff9f854-nwst7:/# cat /proc/sys/net/core/somaxconn								
65536									
root@cci-deployment-20225241-76dff9f854-n	wst7:/# ca	at /proc/sys	s/net/ipv4/	tcp tw	/ reuse				
1				_					
root@cci-deployment-20225241-76dff9f854-n	wst7:/#								

## 3.4 Resizing /dev/shm

## Scenario

**/dev/shm** is a temporary file system (tmpfs), which is a memory-based file system implemented in Linux or Unix and has high read/write efficiency.

If you use **/dev/shm** for data interaction between processes or for temporary data storage, the default size of **/dev/shm** (64 MB) in CCI cannot meet your requirements. CCI allows you to modify the size.

This practice shows how to resize **/dev/shm** by setting memory-backed emptyDir or running **securityContext** and **mount** commands.

## Constraints

- /dev/shm uses a memory-based tmpfs to temporarily store data. Data is not retained after the container is restarted.
- You can use either of the following methods to modify the size of **/dev/shm**. However, do not use both methods in one pod.
- The emptyDir uses the memory requested by the pod and does not occupy extra resources.
- Writing data to **/dev/shm** is to request memory. In this scenario, you need to evaluate the memory usage of processes. When the sum of the memory requested by processes in the container plus the data volume in the emptyDir exceeds the memory limit of the container, memory overflow occurs.
- When resizing /dev/shm, set the size to 50% of the pod's memory request.

## Resizing /dev/shm Using Memory-backed emptyDir

emptyDir is applicable to temporary data storage, disaster recovery, and runtime data sharing. It will be deleted upon deletion or transfer of workload pods.

CCI supports the mounting of memory-backed emptyDir. You can specify the memory size allocated to the emptyDir and mount it to the **/dev/shm** directory in the container to resize **/dev/shm**.

apiVersion: v1 kind: Pod metadata:

name: pod-emptydir-name spec: containers: - image: 'library/ubuntu:latest' volumeMounts: - name: volume-emptydir1 mountPath: /dev/shm name: container-0 resources: limits: cpu: '4' memory: 8Gi requests: cpu: '4' memory: 8Gi volumes: - emptyDir: medium: Memory sizeLimit: 4Gi name: volume-emptydir1

After the pod is started, run the **df** -**h** command to go to the **/dev/shm** directory. If the following information is displayed, the size is successfully modified.

Figure 3-8 /dev/shm directory details

root@pod-emptydir-name:/# df -h										
Filesystem	Size	Used	Avail	Use%	Mounted on					
/dev/vdc	20G	182M	19G	1%						
tmpfs	64M	0	64M	0%	/dev					
tmpfs	4.0G	0	4.0G	0%	/sys/fs/cgroup					
tmpfs	4.0G	52K	4.0G	1%	/etc/hosts					
kataShared	20G	45M	19G	1%	/dev/termination-log					
shm	4.0G	0	4.0G	0%	/dev/shm					
tmpfs	4.0G	0	4.0G	0%	/proc/acpi					
tmpfs	4.0G	0	4.0G	0%	/proc/scsi					
tmpfs	4.0G	0	4.0G	0%	/sys/firmware					

## Resizing /dev/shm by Running securityContext and mount Commands

## • Grant the SYS\_ADMIN permission to the container.

Linux provides the SYS\_ADMIN permission. To apply this permission to the container, Kubernetes needs to add this information to pods by adding the description of the **securityContext** field to the pod's description file. For example:

Another description field **CapAdd** also needs to be added to the container description.

```
"CapAdd": [
"SYS_ADMIN"
].
```

In this case, a parameter is added when the container is automatically started by kubelet.

docker run --cap-add=SYS\_ADMIN

• Insert the mount command in the startup command to resize /dev/shm.

```
apiVersion: v1
kind: Pod
metadata:
name: pod-emptydir-name
spec:
 containers:
  - command:
     - /bin/sh
     - '-C'
    - mount -o size=4096M -o remount /dev/shm;bash
    securityContext:
     capabilities:
      add: ["SYS_ADMIN"]
    image: 'library/ubuntu:latest'
    name: container-0
    resources:
     limits:
      cpu: '4'
      memory: 8Gi
     requests:
      cpu: '4'
      memory: 8Gi
```

After the pod is started, run the **df** -**h** command to go to the **/dev/shm** directory. If the following information is displayed, the size is successfully modified.

root@pod-emptydir-name:/# df -h										
Filesystem	Size	Used	Avail	Use%	Mounted on					
/dev/vdc	20G	182M	19G	1%						
tmpfs	64M	0	64M	0%	/dev					
tmpfs	4.0G	0	4.0G	0%	/sys/fs/cgroup					
tmpfs	4.0G	52K	4.0G	1%	/etc/hosts					
kataShared	20G	45M	19G	1%	/dev/termination-log					
shm	4.0G	0	4.0G	0%	/dev/shm					
tmpfs	4.0G	0	4.0G	0%	/proc/acpi					
tmpfs	4.0G	0	4.0G	0%	/proc/scsi					
tmpfs	4.0G	0	4.0G	0%	/sys/firmware					

Figure 3-9 /dev/shm directory details