ServiceStage

Best Practices

Issue	01
Date	2025-07-10

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Huawei Cloud Computing Technologies Co., Ltd.

Address:	Huawei Cloud Data Center Jiaoxinggong Road Qianzhong Avenue Gui'an New District Gui Zhou 550029 People's Republic of China
Website:	https://www.huaweicloud.com/intl/en-us/

Contents

1 ServiceStage Best Practices

This document summarizes common ServiceStage operation practices and provides solutions and operation guide to help you easily use ServiceStage.

Table 1-1 ServiceStage best practices

Best Practice	Description
Hosting and Managing a Weather Forecast Microservice Application on ServiceStage	This section uses a weather forecast application to demonstrate the application scenarios of the microservice architecture and best practices of managing the runtime environment, building applications, and governing microservices on ServiceStage.
Enabling Security Authentication for an Exclusive ServiceComb Engine	The exclusive ServiceComb engine supports security authentication based on the Role-Based Access Control (RBAC) policy and allows you to enable or disable security authentication. After security authentication is enabled for an engine, the security authentication account and password must be configured for all microservices connected to the engine. Otherwise, the microservice fails to be registered, causing service loss. This section describes how to enable security authentication for an exclusive ServiceComb engine and ensure that services of microservice components connected to the engine are not affected.

Best Practice	Description
Connecting ServiceComb Engine Dashboard Data to AOM through ServiceStage	For Java chassis applications connected to the ServiceComb engine, the real-time monitoring data on the ServiceComb engine dashboard is retained for 5 minutes by default. To permanently store historical monitoring data for subsequent query and analysis, use the custom metric monitoring function of ServiceStage to connect the microservice data displayed on the ServiceComb engine dashboard to AOM. This section uses the application deployed using a software package as an example to describe how to complete the connection.
Migrating the Registered Microservice Engine Using ServiceStage Without Code Modification	This section describes how to migrate the microservice application components that are developed using the Java chassis microservice framework and registered with the professional ServiceComb engine to the exclusive ServiceComb engine without any code modification.
Hosting a Spring Boot Application on ServiceStage	**Spring Boot** is an open-source application development framework based on the Spring framework. It helps you quickly build production-level applications that can run independently. This best practice uses the sample code provided by Spring to help you quickly deploy, access, and upgrade Spring applications on ServiceStage.

Best Practice	Description
Using GitLab to Interconnect with Jenkins to Automatically Build and Perform Rolling Upgrade on Components Deployed on ServiceStage	After the code is developed, you need to pack the code into an image package or JAR package on Jenkins before each rollout, upload the image package to SWR or the JAR package to OBS, and then use ServiceStage to upgrade the component version. This process is complex. Frequent version tests cause low development and O&M efficiency and poor user experience. If you manage your code on GitLab and use ServiceStage with components deployed to host applications, you can use GitLab to interconnect with Jenkins for automatic build and packaging to upgrade the components deployed on ServiceStage. This practice uses the shell script output after Jenkins build and packaging to automatically build and package code after integration and upgrade components deployed on ServiceStage.

Best Practice	Description
Using ServiceStage to Migrate Components Across AZs and Upgrade Components in Sequence Based on Release Management	In actual services, services need to be deployed in different AZs to improve availability due to equipment room faults. However, when components are deployed in different AZs, each component must be configured as required. This is complex and error-prone. In addition, the components need to be deployed and run immediately after being created, and do not support on-demand deployment. If the component configurations are incorrect, the deployment fails. In this case, you need to delete the components, create them again, and then deploy them. ServiceStage release management can be used to migrate and upgrade components across AZs. ● Batch clone release tasks can be used to migrate components across AZs. ● Batch upgrade release tasks can be used to upgrade components across AZs and specify the upgrade sequence of components in different AZs.
Using ServiceStage Component Templates to Automatically Create and Upgrade Components	A component template is a specification file that describes ServiceStage components and defines resources (such as ConfigMaps, Secrets, and Services) and configurations required for component running. You can use a component template to provision components, resources required for component running, and configuration files to quickly create and upgrade components.

Best Practice	Description
Deploying Compressed Package Components Based on a VM	ServiceStage allows you to compress a Java or Tomcat application into a .zip or .tar.gz package for deploying a component on a VM. You can customize script installation, startup, and stop, configuration files, and health check for full lifecycle management from deployment to O&M.

2 Hosting and Managing a Weather Forecast Microservice Application on ServiceStage

2.1 Overview

A weather forecast microservice application provides weather forecasts as well as displays ultraviolet (UV) and humidity indexes. This section uses a weather forecast application to demonstrate the application scenarios of the microservice architecture and best practices of managing the runtime environment, building applications, and governing microservices on ServiceStage.

A weather forecast microservice application consists of a frontend component and backend components. The frontend component **weathermapweb** is developed using Node.js to discover backend components. The backend components are developed using the Java chassis and Spring Cloud microservice development frameworks and include microservices weather, forecast, fusionweather, weather-beta, and edge-service.

Where,

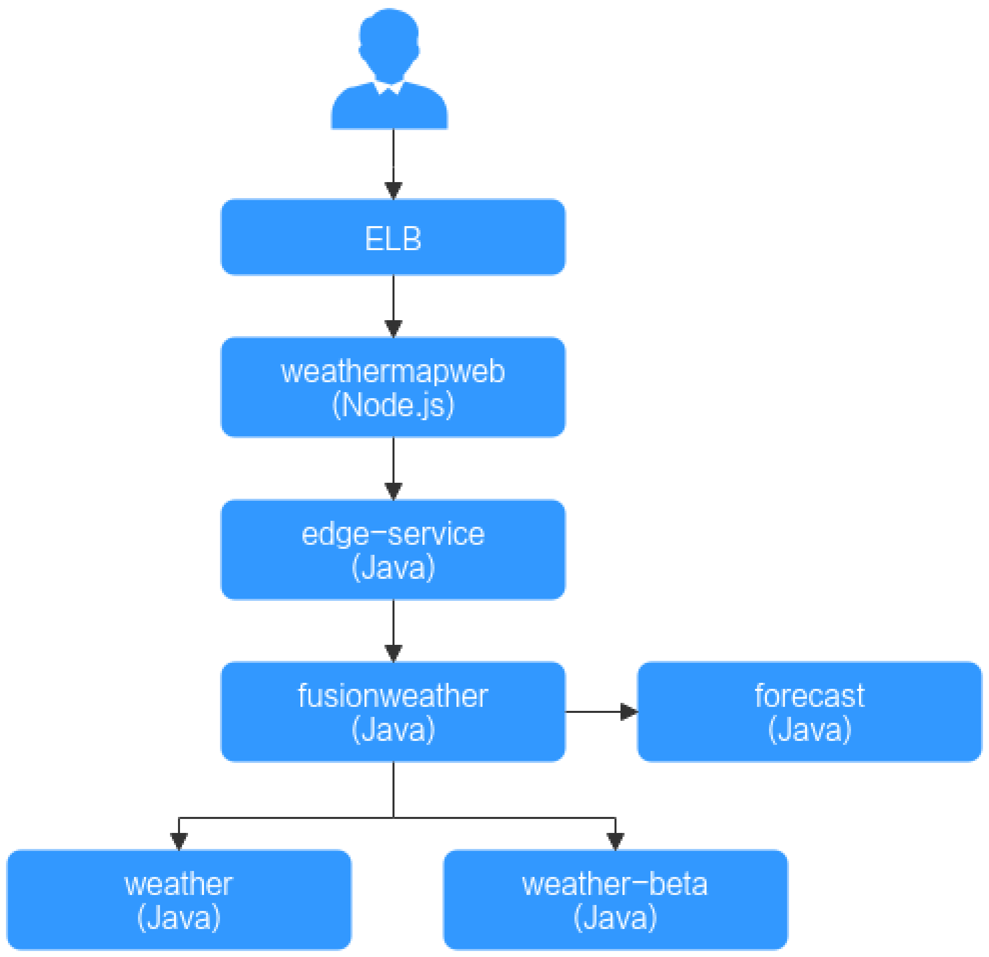
- weathermapweb is an interface microservice developed by Node.js.
- weather is a microservice that provides the current weather of a specified city.
- forecast is a microservice that provides weather forecast for a specified city in the next few days.
- fusionweather is an aggregation microservice that provides comprehensive weather forecast functions by accessing the weather and forecast microservices.
- weather-beta is a new version of the weather microservice. It allows you to query the UV index of a specified city.
- edge-service is the unified portal for all other microservices.

Table 2-1 lists the backend components.

Table 2-1 Components of the weather forecast microservice application

Microservice development framework	**Component Name**
Java chassis	weather
	forecast
	fusionweather
	weather-beta
	edge-service
	weathermapweb
Spring Cloud	weather
	forecast
	fusionweather
	weather-beta
	edge-service
	weathermapweb

The following figure shows the logical networking and calling relationship of the weather forecast application:



ServiceStage supports deployment and access of microservice applications developed based on Java chassis and Spring Cloud using source code and software packages.

This document describes how to host and manage a microservice application on ServiceStage by using the weather forecast microservice application developed based on Java chassis and two microservice application deployment methods (**Deploying a Weather Forecast Microservice Using Source Code** and **Deploying a Weather Forecast Microservice Using a Software Package**).

2.2 Deploying a Weather Forecast Microservice Using Source Code

2.2.1 Preparations

Preparing Resources

To facilitate subsequent operations, ensure that:

1. Create a VPC. For details, see **Creating a VPC**.
2. Create an exclusive ServiceComb engine with security authentication disabled. For details, see **Creating a Microservice Engine**.

The VPC to which the ServiceComb engine belongs is the one created in **1**. If the VPCs are inconsistent, correctly configure the VPC connectivity.

3. Create a CCE standard cluster. Set **Cluster Scale** to **50 nodes** and **Master Nodes** to **Single**. For details, see **Buying a CCE Standard/Turbo Cluster**.
   - The VPC to which the CCE cluster belongs is the one created in **1**.
   - The cluster must contain at least one ECS node with 8 vCPUs and 16 GB memory or two ECS nodes with 4 vCPUs and 8 GB memory and be bound to an EIP. For details, see **Creating a Node**.

Registering a GitHub Account and Forking the Weather Forecast Source Code

Step 1 **Register a GitHub account**.

Step 2 **Log in to GitHub**.

Step 3 Go to the **weather forecast source code repository**.

Step 4 Fork the weather forecast source code repository to your account. For details, see **Forking a repository**.

----End

Setting GitHub Repository Authorization

You can set GitHub repository authorization so that build projects and application components can use the authorization information to access the GitHub source code repository.

Step 1 Log in to ServiceStage.

Step 2 Choose **Continuous Delivery** > **Repository Authorization** > **Create Authorization** and configure authorization information by referring to the following table.

Parameter	Description
*Name	Use the default authorization name. The name cannot be changed after the authorization is created.
*Repository Authorization	1. Select **GitHub**. 2. Select **OAuth** for **Method**. 3. Click **Use OAuth Authorization** and complete the authorization for accessing the GitHub source code repository as prompted.

----End

Creating an Organization

Step 1 Choose **Deployment Source Management** > **Organization Management**.

Step 2 Click **Create Organization**. On the displayed page, specify **Organization Name**. For example, **org-test**.

Step 3 Click **OK**.

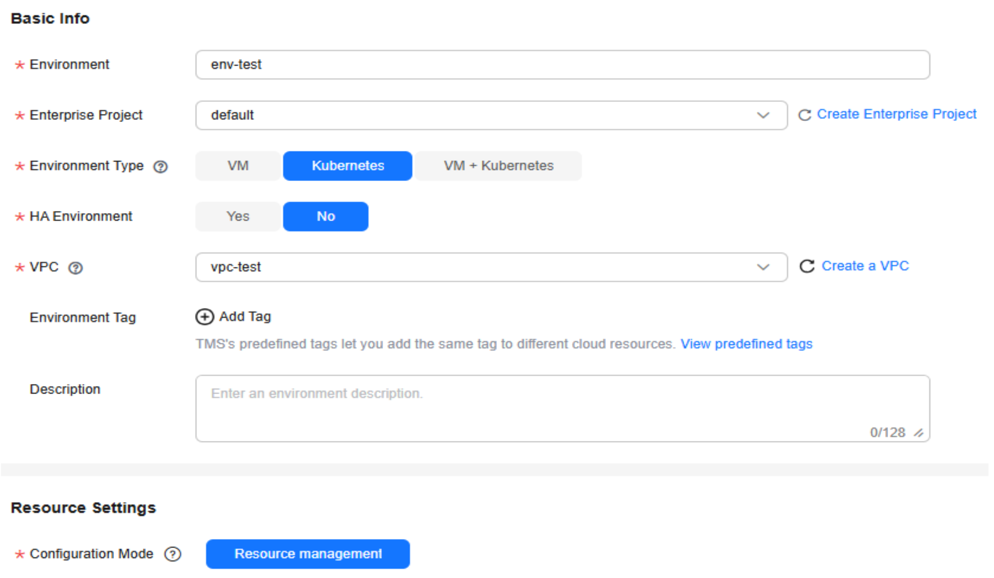
----End

Creating an Environment

Step 1 Choose **Environment Management** > **Create Environment**. Then set required environment parameters by referring to the following table, and retain the default values for other parameters.

Parameter	Description
Environment	Enter an environment name, for example, **env-test**.
Enterprise Project	**default** is selected by default. Enterprise projects let you manage cloud resources and users by project. It is available after you **enable the enterprise project function**.
Environment Type	Select **Kubernetes**.
HA Environment	Select **No**.
VPC	Select the VPC prepared in **Preparing Resources**. The VPC cannot be modified after the environment is created.
Configuration Mode	Select **Resource management**.

Figure 2-1 Configuring an environment



Basic Info

* Environment	env-test
* Enterprise Project	default
* Environment Type	VM / [✓ Kubernetes] / VM + Kubernetes
* HA Environment	Yes [✓ No]
* VPC	vpc-test
Environment Tag	Add Tag TMS's predefined tags let you add the same tag to different cloud resources. View predefined tags
Description	Enter an environment description. 0/128

Resource Settings

* Configuration Mode	[✓ Resource management]

Step 2 Click **Create Now**.

Step 3 Choose **Clusters** under **Compute** and click **Bind now**.

Step 4 In the dialog box that is displayed, select the CCE cluster created in **Preparing Resources** and click **OK**.

Step 5 Choose **ServiceComb Engines** under **Middleware** and click **Manage Resource**.

Step 6 In the dialog box that is displayed, select the ServiceComb engine created in **Preparing Resources** and click **OK**.

----End

Creating an Application

Step 1 Click < in the upper left corner to return to the **Environment Management** page.

Step 2 Choose **Application Management** > **Create Application** and set basic application information.

1. **Name**: Enter **weathermap**.

   NOTE

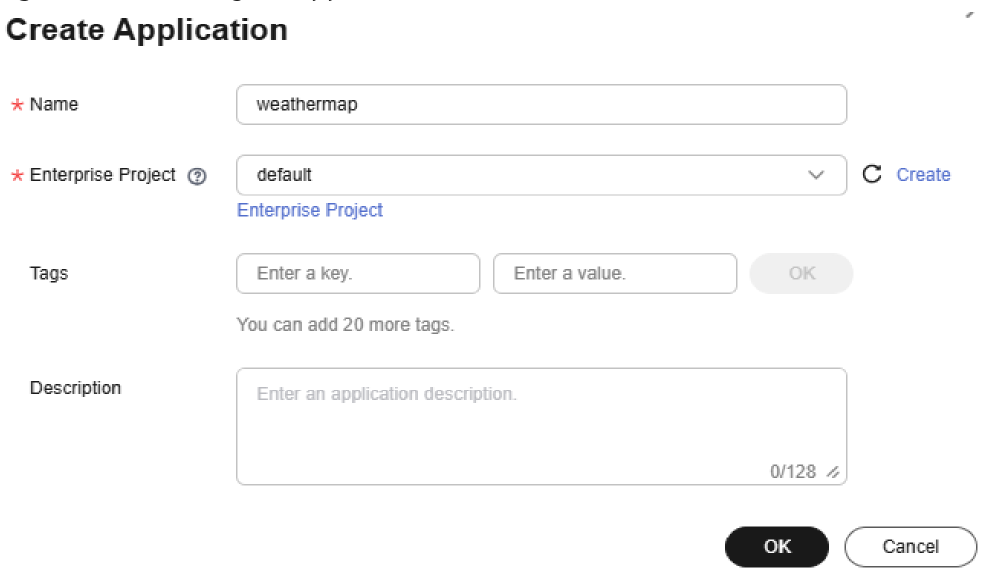
   If an application with the same name already exists in the application list, rectify the fault by referring to **What Should I Do If a Weather Forecast Application with the Same Name Exists in the Current Environment?**

2. **Enterprise Project**: **default** is selected by default. Enterprise projects let you manage cloud resources and users by project.

   It is available after you **enable the enterprise project function**.

Step 3 Click **OK**.

Figure 2-2 Creating an application



Create Application

* Name	weathermap
* Enterprise Project	default
Tags	Enter a key. / Enter a value.
Description	Enter an application description. 0/128

----End

2.2.2 Deploying a Microservice Using Source Code

Scenarios

ServiceStage allows you to quickly deploy microservices in containers (such as CCE) or VMs (such as ECS), and supports source code deployment, JAR/WAR package deployment, and Docker image package deployment. In addition, ServiceStage allows you to deploy, upgrade, roll back, start, stop, and delete applications developed in different programming languages, such as Java, PHP, Node.js, and Python.

In this practice, backend components developed in Java and frontend components developed in Node.js are used.

User Story

In this practice, you can deploy an application in containers and register microservice instances with the ServiceComb engine. The following components need to be created for the weathermap application:

1. Frontend component: weathermapweb, which is developed in Node.js.
2. Backend components: weather, fusionweather, forecast, and edge-service, which are developed based on Java.

The procedures for deploying a microservice are as follows:

1. **Creating and Deploying a Backend Application Component**
2. **Setting the Access Mode of the edge-service Component**
3. **Creating and Deploying a Frontend Component**
4. **Confirming the Deployment Result**
5. **Adding a Frontend Component Access Mode**
6. **Accessing an Application**

Creating and Deploying a Backend Application Component

You need to create and deploy four application components (weather, forecast, fusionweather, and edge-service), which correspond to the four software packages generated by the backend build jobs.

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**.

Step 3 Choose **More** > **Create Component** in the **Operation** column of the application created in **Creating an Application**, for example, **weathermap**.

Step 4 In the **Basic Information** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Component Name	Enter the name of the corresponding backend component (for example, **weather**).
Component Version	Click **Generate**. By default, the version number is generated based on the time when you click **Generate**. The format is yyyy.mmdd.hhmms, where **s** is the ones place of the second in the timestamp. For example, if the timestamp is 2022.0803.104321, the version number is 2022.0803.10431.

Parameter	Description
Application	Select the application created in **Creating an Application**, for example, **weathermap**.
Environmen t	Select the environment created in **Creating an Environment**, for example, **env-test**.
Namespace	Select **default** to isolate component instances.

Step 5 In the **Component Package** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Stack	Select **Java**.
Source Code/ Software Package	1. Select **Source code repository**. 2. Select **GitHub**. 3. **Authorization**: Select the authorization information created when **Setting GitHub Repository Authorization**. 4. **Username/Organization**: Select the GitHub account created when **Registering a GitHub Account and Forking the Weather Forecast Source Code**. 5. **Repository**: Select the weather forecast source code repository that has been forked to your GitHub account when **Registering a GitHub Account and Forking the Weather Forecast Source Code**. For example, **weathermap**. 6. **Branch**: Select **master**.

Step 6 In the **Build Job** area, set mandatory build parameters.

1. **Dockerfile Address**: Set this parameter by referring to the following table.

Component Name	Dockerfile Address
weather	./weather/
forecast	./forecast/
fusionweather	./fusionweather/
edge-service	./edge-service/

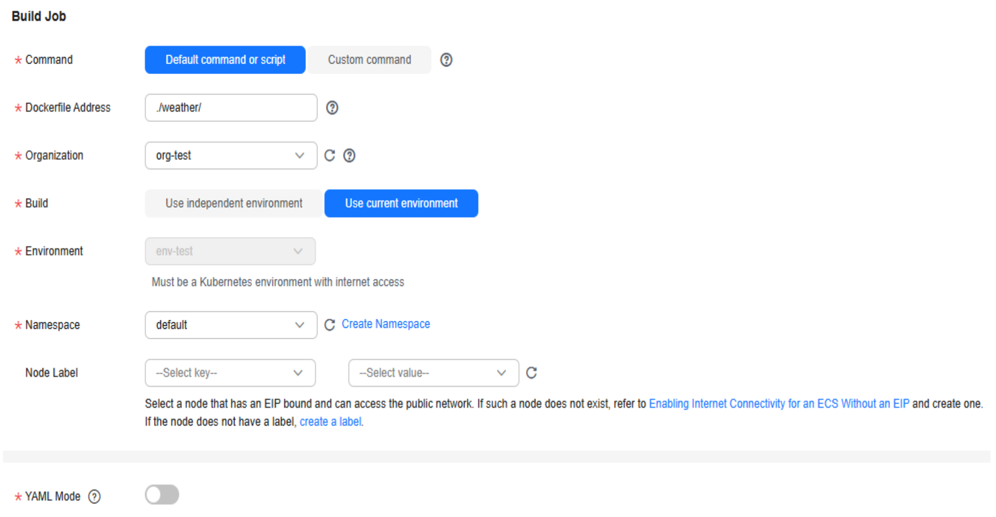
2. **Organization**: Select the organization created in **Creating an Organization**.
3. **Build**: Select **Use current environment**.

   Use the CCE cluster in the deployment environment to which the component belongs to build an image. In the current environment, masters and nodes in the CCE cluster must have the same CPU architecture. Otherwise, the component build fails.

4. **Namespace**: Select **default**.

   Used to isolate build data.

5. Retain the default values for other parameters.



Build Job

* Command	Default command or script (selected) / Custom command
* Dockerfile Address	./weather/
* Organization	org-test
* Build	Use independent environment / Use current environment (selected)
* Environment	env-test
	Must be a Kubernetes environment with internet access
* Namespace	default — Create Namespace
Node Label	--Select key-- / --Select value--
	Select a node that has an EIP bound and can access the public network. If such a node does not exist, refer to Enabling Internet Connectivity for an ECS Without an EIP and create one. If the node does not have a label, create a label.
* YAML Mode	

Figure 2-3 Configuring build parameters

Step 7 Click **Next**.

Step 8 In the **Resources** area, set **Instances** for each component and retain the default values for other parameters.

Component Name	Instances
weather	2
forecast	1
fusionweather	1
edge-service	1

Step 9 Bind the ServiceComb engine.

After a component is deployed, the microservice will be registered with the bound ServiceComb engine. All components must be registered with the same ServiceComb engine.

1. Choose **Cloud Service Settings** > **Microservice Engine**.
2. Click **Bind Microservice Engine**.
3. Select the managed exclusive ServiceComb engine in the current environment.
4. Click **OK**.

Step 10 Click **Create and Deploy**.

----End

Setting the Access Mode of the edge-service Component

Step 1 Click < in the upper left corner to return to the **Application Management** page.

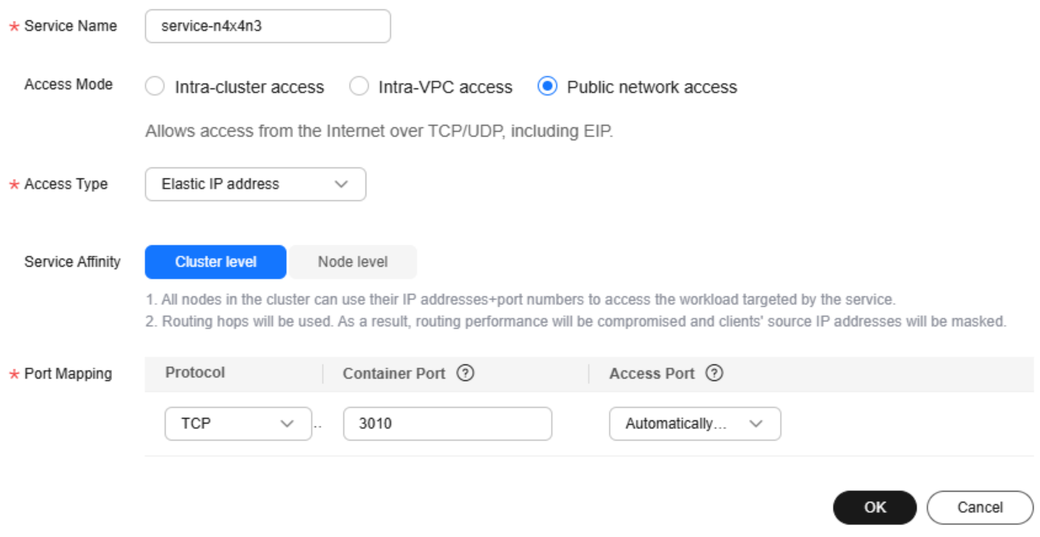
Step 2 Click the application created in **Creating an Application** (for example, **weathermap**). The **Overview** page is displayed.

Step 3 In the **Component List** area, locate the row that contains **edge-service** and click **View Access Mode** in the **External Access Address** column.

Step 4 Click **Add Service** in the **TCP/UDP Route Configuration** area and set parameters by referring to the following table.

Parameter	Description
Service Name	Retain the default value.
Access Mode	Select **Public network access**.
Access Type	Select **Elastic IP address**.
Service Affinity	Retain the default value.
Protocol	Select **TCP**.
Container Port	Enter **3010**.
Access Port	Select **Automatically generated**.

Figure 2-4 Setting the access mode of the edge-service component



* Service Name: service-n4x4n3

Access Mode: Intra-cluster access / Intra-VPC access / [✓ Public network access]

Allows access from the Internet over TCP/UDP, including EIP.

* Access Type: Elastic IP address

Service Affinity: [✓ Cluster level] / Node level

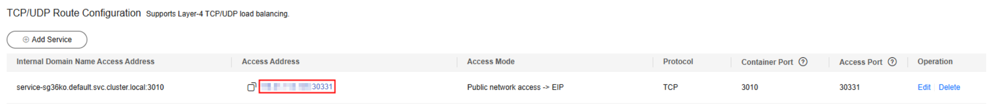
1. All nodes in the cluster can use their IP addresses+port numbers to access the workload targeted by the service.
2. Routing hops will be used. As a result, routing performance will be compromised and clients' source IP addresses will be masked.

* Port Mapping

Protocol	Container Port	Access Port
TCP	3010	Automatically...

Step 5 Click **OK** to generate an access address and record it.

Figure 2-5 Generating and recording the edge-service access address



TCP/UDP Route Configuration Supports Layer-4 TCP/UDP load balancing.

Internal Domain Name Access Address	Access Address	Access Mode	Protocol	Container Port	Access Port	Operation
service-sg36ko.default.svc.cluster.local:3010	[redacted]30331	Public network access -> EIP	TCP	3010	30331	Edit Delete

----End

Creating and Deploying a Frontend Component

Step 1 Click < in the upper left corner to return to the **Application Management** page.

Step 2 Choose **More** > **Create Component** in the **Operation** column of the application created in **Creating an Application**, for example, **weathermap**.

Step 3 In the **Basic Information** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Component Name	Enter the frontend component name **weathermapweb**.
Component Version	Click **Generate**. By default, the version number is generated based on the time when you click **Generate**. The format is yyyy.mmdd.hhmms, where **s** is the ones place of the second in the timestamp. For example, if the timestamp is 2022.0803.104321, the version number is 2022.0803.10431.
Application	Select the application created in **Creating an Application**, for example, **weathermap**.
Environment	Select the environment created in **Creating an Environment**, for example, **env-test**.
Namespace	Select **default** to isolate component instances.

Step 4 In the **Component Package** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Stack	Select **Node.js**.

Parameter	Description
Source Code/ Software Package	1. Select **Source code repository**. 2. Select **GitHub**. 3. **Authorization**: Select the authorization information created when **Setting GitHub Repository Authorization**. 4. **Username/Organization**: Select the username used to log in to GitHub in **Registering a GitHub Account and Forking the Weather Forecast Source Code**. 5. **Repository**: Select the weather forecast source code repository that has been forked to your GitHub account. For example, **weathermap**. 6. **Branch**: Select **master**.

Step 5 In the **Build Job** area, set mandatory build parameters.

1. **Dockerfile Address**: Set this parameter by referring to the following table.

Component Name	Dockerfile Address
weathermapweb	./weathermapweb/

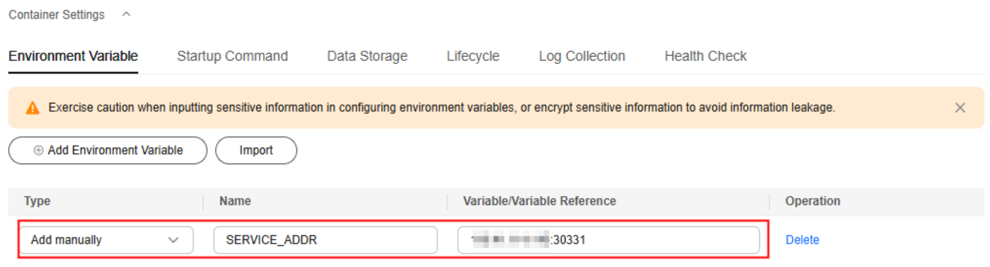
2. **Organization**: Select the organization created in **Creating an Organization**.
3. **Build**: Select **Use current environment**. Use the CCE cluster in the deployment environment to which the component belongs to build an image. In the current environment, masters and nodes in the CCE cluster must have the same CPU architecture. Otherwise, the component build fails.
4. **Namespace**: Select **default** to isolate build data.
5. Retain the default values for other parameters.

Step 6 Click **Next** to add an environment variable.

1. Choose **Container Settings** > **Environment Variable**.
2. Click **Add Environment Variable** to configure environment variables.

Type	Name	Variable/Variable Reference
Add manually	SERVICE_ADDR	Access address generated in **Setting the Access Mode of the edge-service Component**.

Figure 2-6 Adding a frontend component environment variable



Exercise caution when inputting sensitive information in configuring environment variables, or encrypt sensitive information to avoid information leakage.

Type	Name	Variable/Variable Reference	Operation
Add manually	SERVICE_ADDR	[redacted]:30331	Delete

Step 7 Click **Create and Deploy**.

----End

Confirming the Deployment Result

Step 1 Click < in the upper left corner to return to the **Application Management** page.

Step 2 Choose **Cloud Service Engine** > **Microservice Catalog**.

Step 3 Select the ServiceComb engine where the microservice application is deployed from the **Microservice Engine** drop-down list.

Step 4 Select the application (for example, weathermap) created in **Creating an Application** from **Microservice List**.

If the number of instances of each microservice is the same as listed in the following table, the deployment is successful.

Component Name	Instances
weather	2
forecast	1
fusionweather	1
edge-service	1

----End

Adding a Frontend Component Access Mode

Step 1 Choose **Application Management**.

Step 2 Click the application created in **Creating an Application** (for example, **weathermap**). The **Overview** page is displayed.

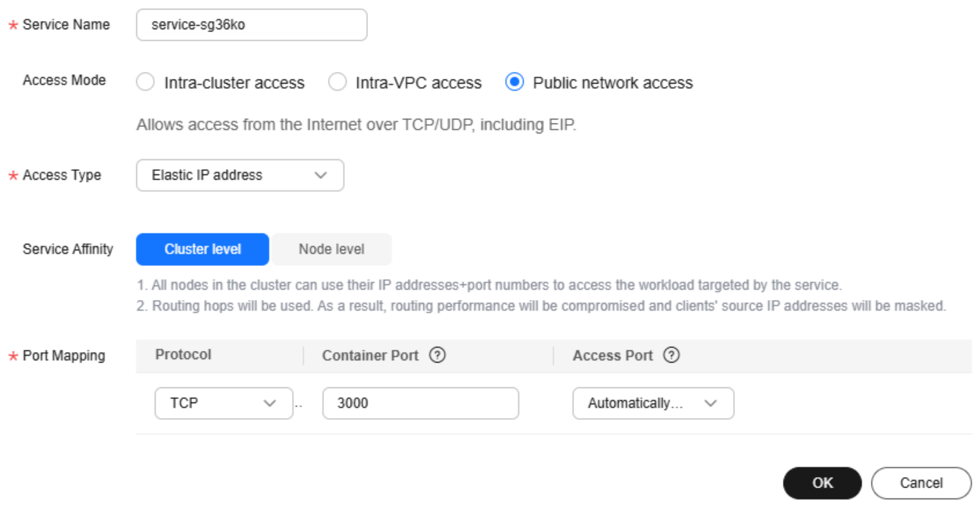
Step 3 In the **Component List** area, locate the row that contains **weathermapweb** and click **View Access Mode** in the **External Access Address** column.

Step 4 Click **Add Service** in the **TCP/UDP Route Configuration** area and set parameters by referring to the following table.

Parameter	Description
Service Name	Retain the default value.
Access Mode	Select **Public network access**.
Access Type	Select **Elastic IP address**.
Service Affinity	Retain the default value.
Protocol	Select **TCP**.
Container Port	Enter **3000**.

Parameter	Description
Access Port	Select **Automatically generated**.

Figure 2-7 Adding a frontend component access mode



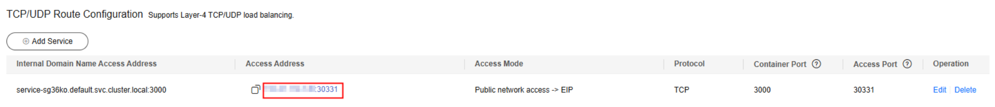
* Service Name	service-sg36ko
Access Mode	Intra-cluster access [✓ Public network access] Intra-VPC access
	Allows access from the Internet over TCP/UDP, including EIP.
* Access Type	Elastic IP address
Service Affinity	[✓ Cluster level] Node level
	1. All nodes in the cluster can use their IP addresses+port numbers to access the workload targeted by the service. 2. Routing hops will be used. As a result, routing performance will be compromised and clients' source IP addresses will be masked.

* Port Mapping

Protocol	Container Port	Access Port
TCP	3000	Automatically...

Step 5 Click **OK**.

Figure 2-8 Access address



TCP/UDP Route Configuration Supports Layer-4 TCP/UDP load balancing.

Internal Domain Name Access Address	Access Address	Access Mode	Protocol	Container Port	Access Port	Operation
service-sg36ko.default.svc.cluster.local:3000	[redacted]:30331	Public network access -> EIP	TCP	3000	30331	Edit Delete

----End

Accessing an Application

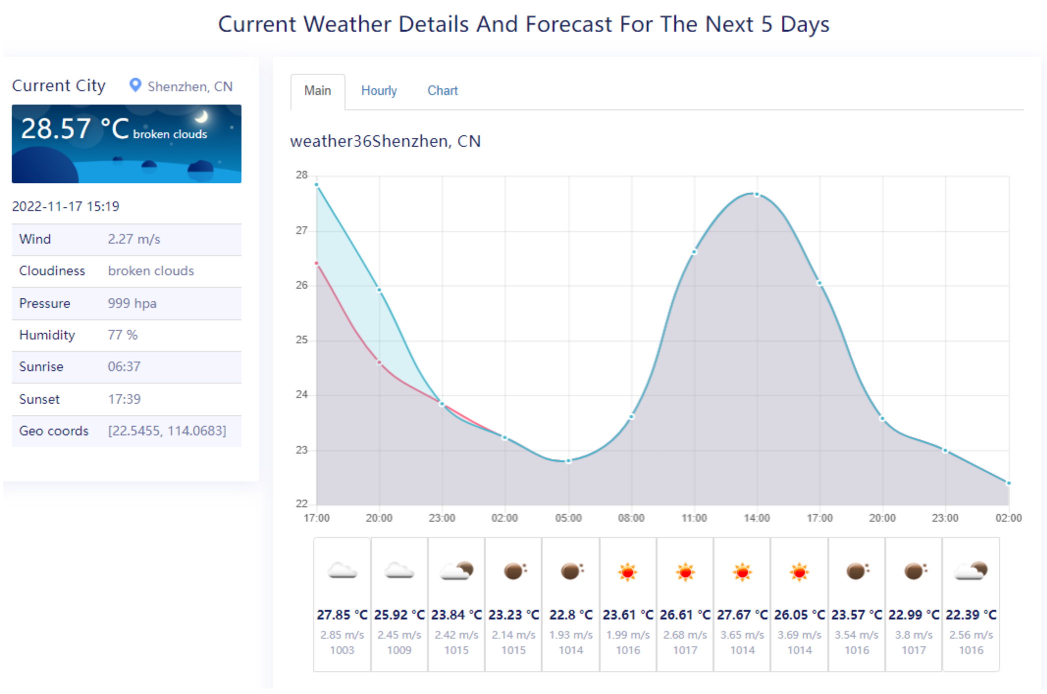
Step 1 Click < in the upper left corner to return to the **Application Management** page.

Step 2 Click the application created in **Creating an Application** (for example, **weathermap**). The **Overview** page is displayed.

Step 3 In the **Component List** area, locate the row that contains **weathermapweb** and click **View Access Mode** in the **External Access Address** column.

If the following page is displayed, the weather forecast microservice application is successfully deployed.

Figure 2-9 Application deployed successfully


Current Weather Details And Forecast For The Next 5 Days

Current City Shenzhen, CN

28.57 °C broken clouds

2022-11-17 15:19

Wind	2.27 m/s
Cloudiness	broken clouds
Pressure	999 hpa
Humidity	77 %
Sunrise	06:37
Sunset	17:39
Geo coords	[22.5455, 114.0683]

weather36Shenzhen, CN

27.85 °C	25.92 °C	23.84 °C	23.23 °C	22.8 °C	23.61 °C	26.61 °C	27.67 °C	26.05 °C	23.57 °C	22.99 °C	22.39 °C
2.85 m/s	2.45 m/s	2.42 m/s	2.14 m/s	1.93 m/s	1.99 m/s	2.68 m/s	3.65 m/s	3.69 m/s	3.54 m/s	3.8 m/s	2.56 m/s
1003	1009	1015	1015	1014	1016	1017	1014	1014	1016	1017	1016

NOTE

- The data is real-time data.
- When you access the application for the first time, it takes some time for the weather system to be ready. If the preceding page is not displayed, refresh the page.

----End

2.3 Deploying a Weather Forecast Microservice Using a Software Package

2.3.1 Preparations

Preparing Resources

To facilitate subsequent operations, ensure that:

1. Create a VPC. For details, see **Creating a VPC**.
2. Create an exclusive ServiceComb engine 2.4.0 or later with security authentication disabled. For details, see **Creating a Microservice Engine**.

   The VPC to which the ServiceComb engine belongs is the one created in **1**. If the VPCs are inconsistent, correctly configure the VPC connectivity.
3. Create a CCE standard cluster. Set **Cluster Scale** to **50 nodes** and **Master Nodes** to **Single**. For details, see **Buying a CCE Standard/Turbo Cluster**.
   – The VPC to which the CCE cluster belongs is the one created in **1**.

   – The cluster must contain at least one ECS node with 8 vCPUs and 16 GB memory or two ECS nodes with 4 vCPUs and 8 GB memory and be bound to an EIP. For details, see **Creating a Node**.

4. Create a bucket for storing software packages. For details, see **Creating a Bucket**.

Downloading and Uploading Component Software Packages

Step 1 Download the weather forecast component software package to the local PC by referring to **Table 2-2**. (This practice uses the component developed based on Java chassis.)

Table 2-2 Software packages of the weather forecast components

Microservice Development Framework	Component Name	Component Software Package Name	Description of Downloading a Component Software Package
Java chassis	weather	weather-1.0.0.jar	1. Access **software package repository of weather forecast components**. 2. Click **ServiceComb** to access the software package repository of weather forecast components developed using the Java chassis microservice development framework.
	weather-beta	weather-beta-2.0.0.jar	
	forecast	forecast-1.0.0.jar	
	fusionweather	fusionweather-1.0.0.jar	
	edge-service	edge-service-1.0.0.jar	
	weathermapweb	weathermapweb.zip	
Spring Cloud	weather	weather-1.0.0.jar	1. Access **software package repository of weather forecast components**. 2. Click **Spring Cloud** to access the software package repository of weather forecast components developed using the Spring Cloud microservice development framework.
	weather-beta	weather-beta-2.0.0.jar	
	forecast	forecast-1.0.0.jar	
	fusionweather	fusionweather-1.0.0.jar	
	edge-service	edge-service-1.0.0.jar	
	weathermapweb	weathermapweb.zip	

Step 2 Upload the preceding software packages to the bucket prepared in **Preparing Resources**.

For details how to upload a software package, see **Streaming Upload (PUT)**.

----End

Creating an Organization

Step 1 Choose **Deployment Source Management** > **Organization Management**.

Step 2 Click **Create Organization**. On the displayed page, specify **Organization Name**. For example, **org-test**.

Step 3 Click **OK**.

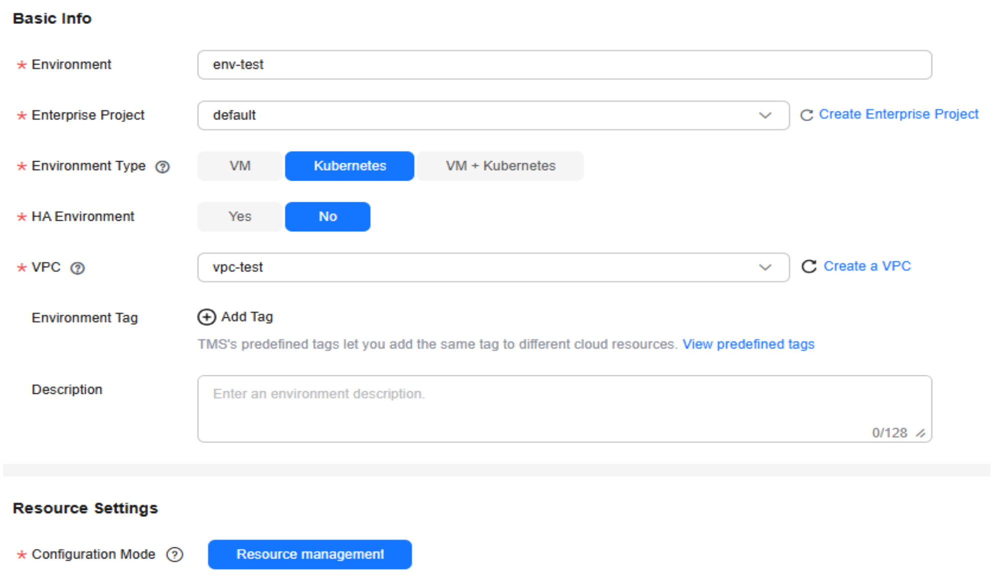
----End

Creating an Environment

Step 1 Choose **Environment Management** > **Create Environment**. Then set required environment parameters by referring to the following table, and retain the default values for other parameters.

Parameter	Description
Environment	Enter an environment name, for example, **env-test**.
Enterprise Project	**default** is selected by default. Enterprise projects let you manage cloud resources and users by project. It is available after you **enable the enterprise project function**.
Environment Type	Select **Kubernetes**.
HA Environment	Select **No**.
VPC	Select the VPC prepared in **Preparing Resources**. The VPC cannot be modified after the environment is created.
Configuration Mode	Select **Resource management**.

Figure 2-10 Configuring an environment



Basic Info

* Environment	env-test
* Enterprise Project	default
* Environment Type	Kubernetes
* HA Environment	No
* VPC	vpc-test
Environment Tag	Add Tag TMS's predefined tags let you add the same tag to different cloud resources. View predefined tags
Description	Enter an environment description. 0/128

Resource Settings

* Configuration Mode	Resource management

Step 2 Click **Create Now**.

Step 3 Choose **Clusters** under **Compute** and click **Bind now**.

Step 4 In the dialog box that is displayed, select the CCE cluster created in **Preparing Resources** and click **OK**.

Step 5 Choose **ServiceComb Engines** under **Middleware** and click **Manage Resource**.

Step 6 In the dialog box that is displayed, select the ServiceComb engine created in **Preparing Resources** and click **OK**.

----End

Creating an Application

Step 1 Click < in the upper left corner to return to the **Environment Management** page.

Step 2 Choose **Application Management** > **Create Application** and set basic application information.

1. **Name**: Enter **weathermap**.

   NOTE

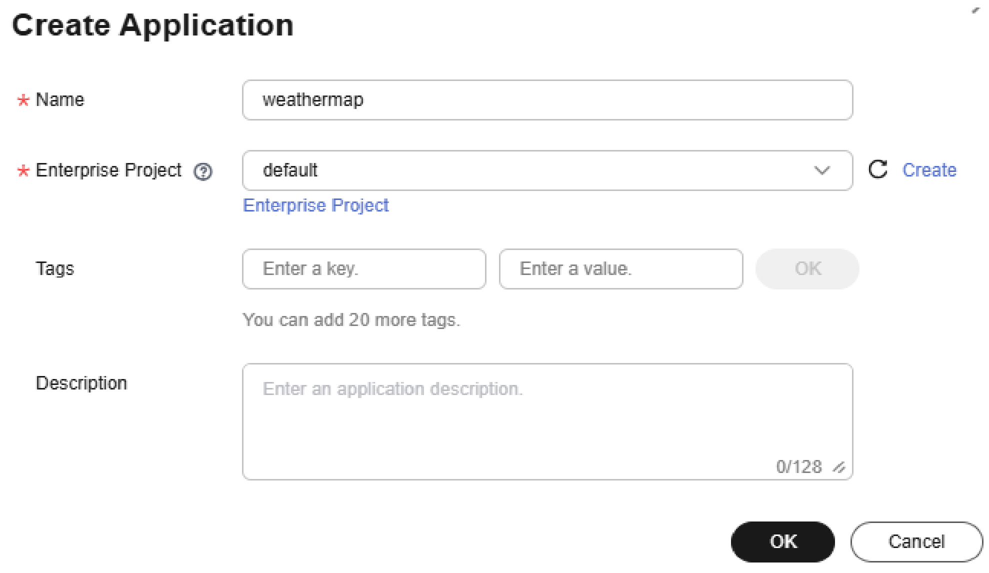
   If an application with the same name already exists in the application list, rectify the fault by referring to **What Should I Do If a Weather Forecast Application with the Same Name Exists in the Current Environment?**

2. **Enterprise Project**: **default** is selected by default. Enterprise projects let you manage cloud resources and users by project.

   It is available after you **enable the enterprise project function**.

Step 3 Click **OK**.

Figure 2-11 Creating an application



Create Application

* Name	weathermap
* Enterprise Project	default
Tags	Enter a key. / Enter a value.
Description	Enter an application description. 0/128

----End

2.3.2 Deploying a Microservice Using a Software Package

Scenarios

ServiceStage allows you to quickly deploy microservices in containers (such as CCE) or VMs (such as ECS), and supports source code deployment, JAR/WAR package deployment, and Docker image package deployment. In addition, ServiceStage allows you to deploy, upgrade, roll back, start, stop, and delete applications developed in different programming languages, such as Java, PHP, Node.js, and Python.

In this practice, backend components developed in Java and frontend components developed in Node.js are used.

User Story

In this practice, you can deploy an application in containers and register microservice instances with the ServiceComb engine. The following components need to be created and deployed for the weathermap application:

1. Frontend component: weathermapweb, which is developed in Node.js.
2. Backend components: weather, fusionweather, forecast, and edge-service, which are developed based on Java.

The procedures for deploying a microservice are as follows:

1. **Creating and Deploying a Backend Application Component**
2. **Setting the Access Mode of the edge-service Component**
3. **Creating and Deploying a Frontend Component**
4. **Confirming the Deployment Result**
5. **Adding a Frontend Component Access Mode**

6. Accessing an Application

Creating and Deploying a Backend Application Component

You need to create and deploy four application components (weather, forecast, fusionweather, and edge-service), which correspond to the four software packages generated by the backend build jobs.

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**. The application list is displayed.

Step 3 Choose **More** > **Create Component** in the **Operation** column of the application created in **Creating an Application**, for example, **weathermap**.

Step 4 In the **Basic Information** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Component Name	Enter the name of the corresponding backend component (for example, **weather**).
Component Version	Click **Generate**. By default, the version number is generated based on the time when you click **Generate**. The format is yyyy.mmdd.hhmms, where **s** is the ones place of the second in the timestamp. For example, if the timestamp is 2022.0803.104321, the version number is 2022.0803.10431.
Application	Select the application created in **Creating an Application**, for example, **weathermap**.
Environment	Select the environment created in **Creating an Environment**, for example, **env-test**.
Namespace	Select **default** to isolate component instances.

Step 5 In the **Component Package** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Stack	Select **Java**.
Source Code/ Software Package	Select **JAR package**.
Upload Method	1. Select **OBS**. 2. Click **Select Software Package** and select the uploaded software package of the corresponding component by referring to **Table 2-2**. 3. Click **OK**.

Step 6 In the **Build Job** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Organization	Select the organization created in **Creating an Organization**. An organization is used to manage images generated during component build.
Build	Select **Use current environment** to use the CCE cluster in the deployment environment to which the component belongs to build an image. In the current environment, masters and nodes in the CCE cluster must have the same CPU architecture. Otherwise, the component build fails.
Namespace	Select **default** to isolate build data.

Step 7 Click **Next**.

Step 8 In the **Resources** area, set **Instances** for each component and retain the default values for other parameters.

Component Name	Instances
weather	2
forecast	1
fusionweather	1
edge-service	1

Step 9 Bind the ServiceComb engine.

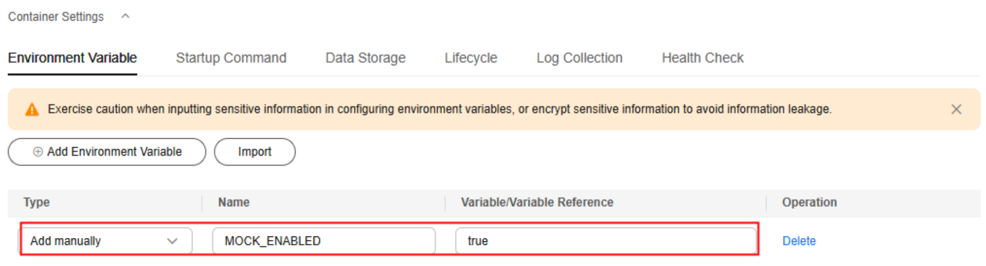
After a component is deployed, the microservice will be registered with the ServiceComb engine. All components must be registered with the same ServiceComb engine.

1. Choose **Cloud Service Settings** > **Microservice Engine**.
2. Click **Bind Microservice Engine**.
3. Select the managed ServiceComb engine in the current environment.
4. Click **OK**.

Step 10 (Optional) Choose **Container Settings** > **Environment Variable** > **Add Environment Variable**. Then add environment variables for the weather, forecast, and fusionweather components by referring to the following table.

Type	Name	Variable/Variable Reference
Add manually	MOCK_ENABLED	● **true**: If no EIP is bound to the ECS node in the CCE cluster created in **Preparing Resources** or the node cannot access the public network, set this parameter to **true**. The weather data used by the application is simulated data. ● **false**: If an EIP has been bound to the ECS node in the CCE cluster created in **Preparing Resources** and the node can access the public network, set this parameter to **false** or do not set this parameter. The weather data used by the application is real-time data.

Figure 2-12 Adding a backend component environment variable



Exercise caution when inputting sensitive information in configuring environment variables, or encrypt sensitive information to avoid information leakage.

Type	Name	Variable/Variable Reference	Operation
Add manually	MOCK_ENABLED	true	Delete

Step 11 Click **Create and Deploy**.

----End

Setting the Access Mode of the edge-service Component

Step 1 Click < in the upper left corner to return to the **Application Management** page.

Step 2 Click the application created in **Creating an Application** (for example, **weathermap**). The **Overview** page is displayed.

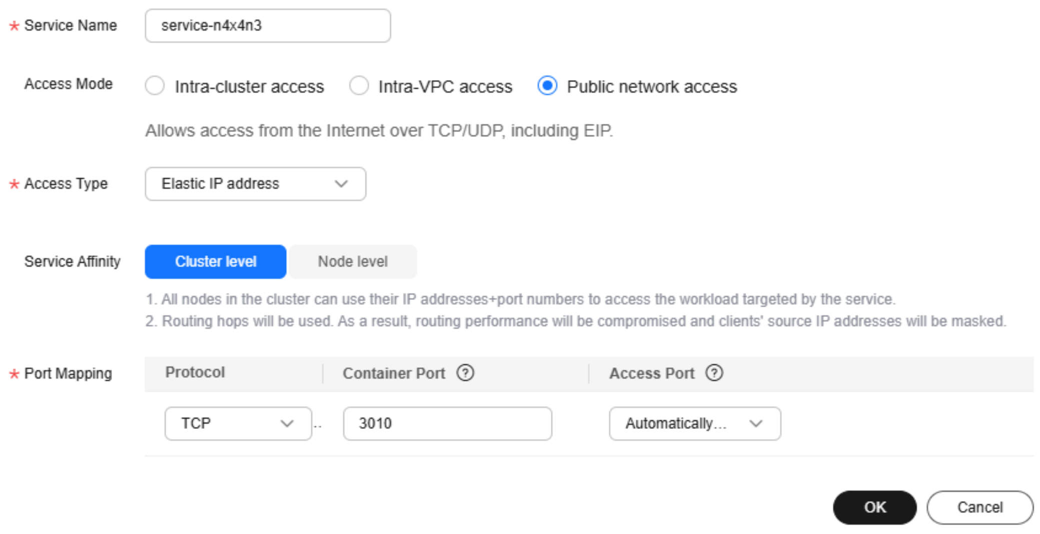
Step 3 In the **Component List** area, locate the row that contains **edge-service** and click **View Access Mode** in the **External Access Address** column.

Step 4 Click **Add Service** in the **TCP/UDP Route Configuration** area and set parameters by referring to the following table.

Parameter	Description
Service Name	Retain the default value.

Parameter	Description
Access Mode	Select **Public network access**.
Access Type	Select **Elastic IP address**.
Service Affinity	Retain the default value.
Protocol	Select **TCP**.
Container Port	Enter **3010**.
Access Port	Select **Automatically generated**.

Figure 2-13 Setting the access mode of the edge-service component



* Service Name: service-n4x4n3

Access Mode: Intra-cluster access / Intra-VPC access / [✓ Public network access]

Allows access from the Internet over TCP/UDP, including EIP.

* Access Type: Elastic IP address

Service Affinity: [✓ Cluster level] / Node level

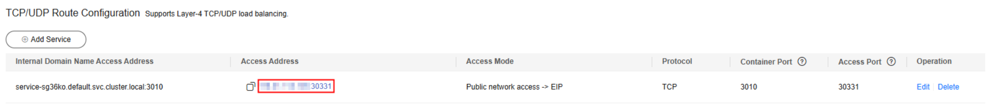
1. All nodes in the cluster can use their IP addresses+port numbers to access the workload targeted by the service.
2. Routing hops will be used. As a result, routing performance will be compromised and clients' source IP addresses will be masked.

* Port Mapping

Protocol	Container Port	Access Port
TCP	3010	Automatically...

Step 5 Click **OK** to generate an access address and record it.

Figure 2-14 Generating and recording the edge-service access address



TCP/UDP Route Configuration Supports Layer-4 TCP/UDP load balancing.

Internal Domain Name Access Address	Access Address	Access Mode	Protocol	Container Port	Access Port	Operation
service-sg36ko.default.svc.cluster.local:3010	[redacted]30331	Public network access -> EIP	TCP	3010	30331	Edit Delete

----End

Creating and Deploying a Frontend Component

Step 1 Click < in the upper left corner to return to the **Application Management** page.

Step 2 Choose **More** > **Create Component** in the **Operation** column of the application created in **Creating an Application**, for example, **weathermap**.

Step 3 In the **Basic Information** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Component Name	Enter the frontend component name **weathermapweb**.
Component Version	Click **Generate**. By default, the version number is generated based on the time when you click **Generate**. The format is yyyy.mmdd.hhmms, where **s** is the ones place of the second in the timestamp. For example, if the timestamp is 2022.0803.104321, the version number is 2022.0803.10431.
Application	Select the application created in **Creating an Application**, for example, **weathermap**.
Environment	Select the environment created in **Creating an Environment**, for example, **env-test**.
Namespace	Select **default** to isolate component instances.

Step 4 In the **Component Package** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Stack	Select **Node.js**.
Source Code/ Software Package	Select **ZIP package**.
Upload Method	1. Select **OBS**. 2. Click **Select Software Package** and select the uploaded software package of component **weathermapweb** by referring to **Table 2-2**. 3. Click **OK**.

Step 5 In the **Build Job** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Organization	An organization is used to manage images generated during component build. Select the organization created in **Creating an Organization**.
Build	Select **Use current environment** to use the CCE cluster in the deployment environment to which the component belongs to build an image. In the current environment, masters and nodes in the CCE cluster must have the same CPU architecture. Otherwise, the component build fails.

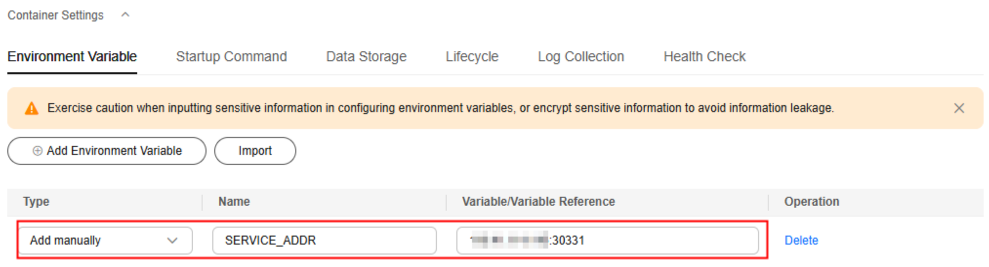
Parameter	Description
Namespace	Select **default** to isolate build data.

Step 6 Click **Next** to add an environment variable.

1. Choose **Container Settings** > **Environment Variable**.
2. Click **Add Environment Variable** to configure environment variables.

Type	Name	Variable/Variable Reference
Add manually	SERVICE_ADDR	Access address generated in **Setting the Access Mode of the edge-service Component**.

Figure 2-15 Adding an environment variable of a frontend component



Exercise caution when inputting sensitive information in configuring environment variables, or encrypt sensitive information to avoid information leakage.

Type	Name	Variable/Variable Reference	Operation
Add manually	SERVICE_ADDR	[redacted]:30331	Delete

Step 7 Click **Create and Deploy**.

----End

Confirming the Deployment Result

Step 1 Click < in the upper left corner to return to the **Application Management** page.

Step 2 Choose **Cloud Service Engine** > **Microservice Catalog**.

Step 3 Select the ServiceComb engine where the microservice application is deployed from the **Microservice Engine** drop-down list.

Step 4 Select the application (for example, weathermap) created in **Creating an Application** from **Microservice List**.

If the number of instances of each microservice is the same as listed in the following table, the deployment is successful.

Component Name	Instances
weather	2
forecast	1
fusionweather	1

Component Name	Instances
edge-service	1

----End

Adding a Frontend Component Access Mode

Step 1 Choose **Application Management**.

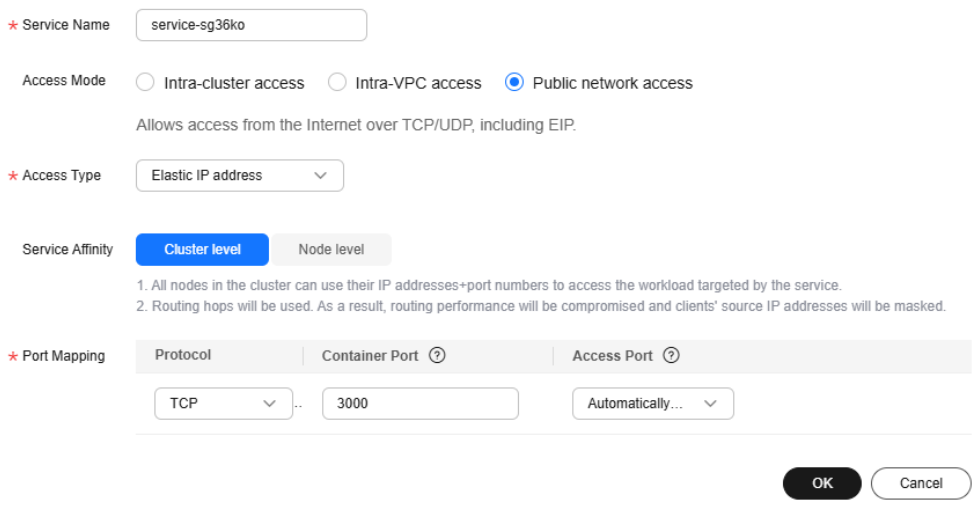
Step 2 Click the application created in **Creating an Application** (for example, **weathermap**). The **Overview** page is displayed.

Step 3 In the **Component List** area, locate the row that contains **weathermapweb** and click **View Access Mode** in the **External Access Address** column.

Step 4 Click **Add Service** in the **TCP/UDP Route Configuration** area and set parameters by referring to the following table.

Parameter	Description
Service Name	Retain the default value.
Access Mode	Select **Public network access**.
Access Type	Select **Elastic IP address**.
Service Affinity	Retain the default value.
Protocol	Select **TCP**.
Container Port	Enter **3000**.
Access Port	Select **Automatically generated**.

Figure 2-16 Adding a frontend component access mode



* Service Name: service-sg36ko

Access Mode: ( ) Intra-cluster access ( ) Intra-VPC access [✓ Public network access]

Allows access from the Internet over TCP/UDP, including EIP.

* Access Type: Elastic IP address

Service Affinity: [✓ Cluster level] Node level

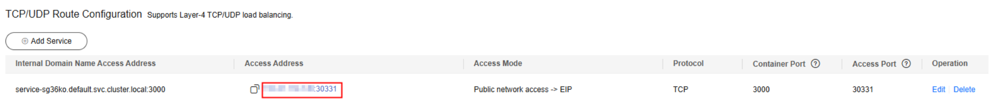
1. All nodes in the cluster can use their IP addresses+port numbers to access the workload targeted by the service.
2. Routing hops will be used. As a result, routing performance will be compromised and clients' source IP addresses will be masked.

* Port Mapping

Protocol	Container Port	Access Port
TCP	3000	Automatically...

Step 5 Click **OK**.

Figure 2-17 Access address


TCP/UDP Route Configuration Supports Layer-4 TCP/UDP load balancing.

Internal Domain Name Access Address	Access Address	Access Mode	Protocol	Container Port ⓘ	Access Port ⓘ	Operation
service-sg36ko.default.svc.cluster.local:3000	[redacted]:30331	Public network access -> EIP	TCP	3000	30331	Edit Delete

----End

Accessing an Application

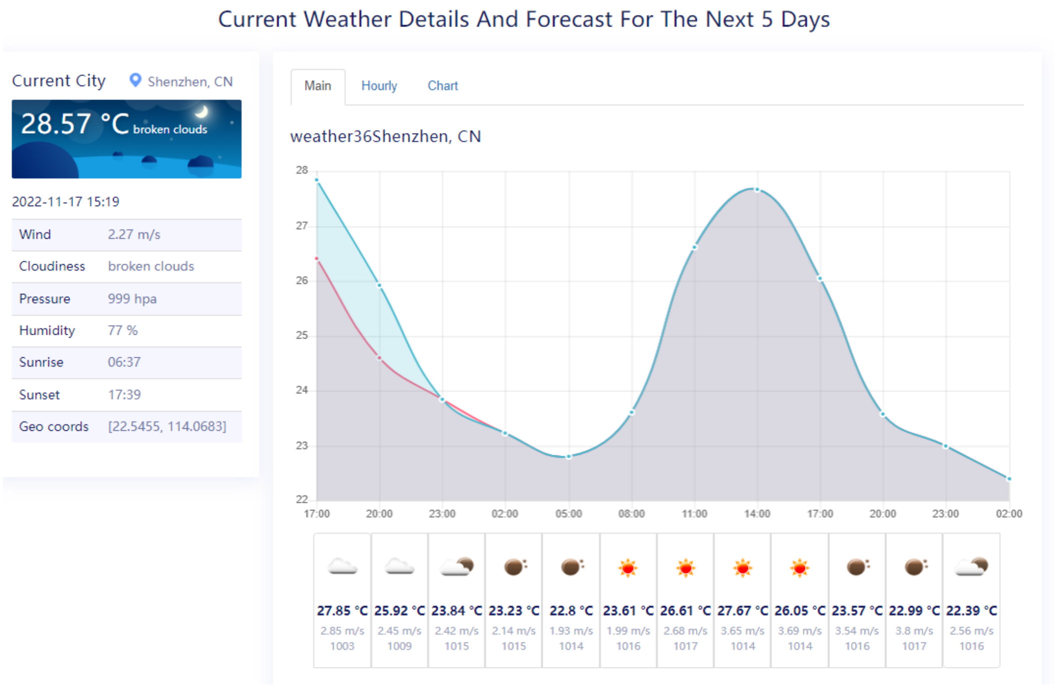
Step 1 Click < in the upper left corner to return to the **Application Management** page.

Step 2 Click the application created in **Creating an Application** (for example, **weathermap**). The **Overview** page is displayed.

Step 3 In the **Component List** area, locate the row that contains **weathermapweb** and click **View Access Mode** in the **External Access Address** column.

If the following page is displayed, the weather forecast microservice application is successfully deployed.

Figure 2-18 Application deployed successfully


Current Weather Details And Forecast For The Next 5 Days

Current City   Shenzhen, CN

28.57 °C broken clouds

2022-11-17 15:19

Wind	2.27 m/s
Cloudiness	broken clouds
Pressure	999 hpa
Humidity	77 %
Sunrise	06:37
Sunset	17:39
Geo coords	[22.5455, 114.0683]

27.85 °C	25.92 °C	23.84 °C	23.23 °C	22.8 °C	23.61 °C	26.61 °C	27.67 °C	26.05 °C	23.57 °C	22.99 °C	22.39 °C
2.85 m/s	2.45 m/s	2.42 m/s	2.14 m/s	1.93 m/s	1.99 m/s	2.68 m/s	3.65 m/s	3.69 m/s	3.54 m/s	3.8 m/s	2.56 m/s
1003	1009	1015	1015	1014	1016	1017	1014	1014	1016	1017	1016

NOTE

- The data is real-time data.
- When you access the application for the first time, it takes some time for the weather system to be ready. If the preceding page is not displayed, refresh the page.

----End

2.4 Microservice Routine O&M

Scenarios

ServiceStage supports application monitoring, events, alarms, logs, tracing diagnosis, and built-in AI capabilities, implementing easy O&M.

User Story

In actual application scenarios, you can monitor application running status in real time based on graphic metrics and threshold-crossing alarms. In addition, you can quickly locate application running problems and analyze performance bottlenecks based on performance management and log policies.

Procedure

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**.

Step 3 Click an application (for example, **weathermap**). The **Overview** page is displayed.

Step 4 In the **Component List** area, click the target component. The component overview page is displayed.

Perform routine O&M by referring to **Component O&M**.

----End

2.5 Dark Launch

The weather-beta microservice is a new version of the weather microservice and allows you to query the UV index. Before upgrading to weather-beta, a small number of requests are diverted to the later version for function verification. If the functions are normal, the earlier version will be brought offline. During the upgrade, customer requests should not be interrupted. During the deployment of the later version, traffic is not diverted to the later version. Before the earlier version is brought offline, traffic is migrated from the earlier version to the later version.

ServiceStage provides dark launch to achieve the preceding objectives.

This section describes how to deploy weather-beta using dark launch of ServiceStage.

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**.

Step 3 Click an application (for example, **weathermap**). The **Overview** page is displayed.

Step 4 In the **Component List** area, click the target weather component. The component overview page is displayed.

Step 5 In the upper right corner of the page, click **Upgrade**.

Step 6 Select **Dark Launch** and click **Next**.

Step 7 Configure the dark launch version based on how you deploy the weather forecast microservice.

- If **source code** is used for deployment, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Command	Select **Default command or script**.
Dockerfile Address	Enter `./weather-beta/`
Component Version	Click **Generate**. By default, the version number is generated based on the time when you click **Generate**. The format is yyyy.mmdd.hhmms, where **s** is the ones place of the second in the timestamp. For example, if the timestamp is 2022.0803.104321, the version number is 2022.0803.10431.

- If a **software package** is used for deployment, set the following mandatory parameters. Retain the default values for other parameters.

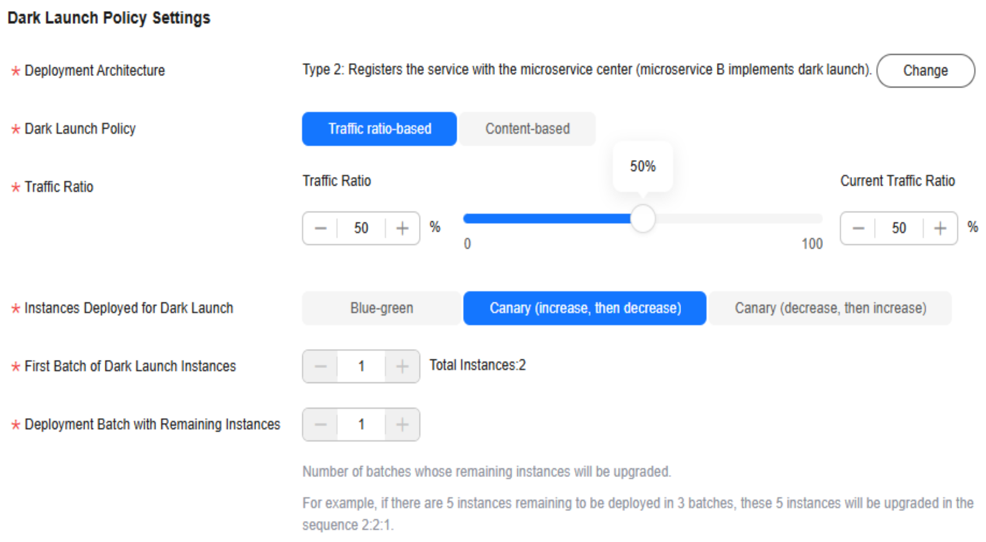
Parameter	Description
Upload Method	1. Move the cursor to the **weather-1.0.0.jar** software package. 2. Click ✎. 3. Select the **weather-beta-2.0.0.jar** software package that has been uploaded when **Downloading and Uploading Component Software Packages**.
Component Version	Click **Generate**. By default, the version number is generated based on the time when you click **Generate**. The format is yyyy.mmdd.hhmms, where **s** is the ones place of the second in the timestamp. For example, if the timestamp is 2022.0803.104321, the version number is 2022.0803.10431.

Step 8 Set mandatory parameters by referring to the following table. Retain the default values for other parameters.

Parameter	Description
Deployment Architecture	1. Click **Select**. 2. Select **Type 2: Registers the service with the microservice center (microservice B implements dark launch)**. 3. Click **OK**.

Parameter	Description
Dark Launch Policy	Select **Traffic ratio-based**.
Traffic Ratio	● **Traffic Ratio**: percentage of traffic directed to the new version. Set it to **50**. ● **Current Traffic Ratio**: percentage of traffic directed to the current version. It is automatically set to **50**.
Instances Deployed for Dark Launch	Select **Canary (increase, then decrease instances)**.
First Batch of Dark Launch Instances	Set this parameter to **1**.
Deployment Batch with Remaining Instances	Set this parameter to **1**.

Figure 2-19 Configuring the dark launch policy



Dark Launch Policy Settings

* Deployment Architecture: Type 2: Registers the service with the microservice center (microservice B implements dark launch). Change
* Dark Launch Policy: Traffic ratio-based (selected) / Content-based
* Traffic Ratio: Traffic Ratio 50 % (50%, 0–100); Current Traffic Ratio 50 %
* Instances Deployed for Dark Launch: Blue-green / Canary (increase, then decrease) (selected) / Canary (decrease, then increase)
* First Batch of Dark Launch Instances: 1 Total Instances:2
* Deployment Batch with Remaining Instances: 1

Number of batches whose remaining instances will be upgraded.

For example, if there are 5 instances remaining to be deployed in 3 batches, these 5 instances will be upgraded in the sequence 2:2:1.

Step 9 Click **Upgrade**.

Wait until the component status changes from **Upgrading/Rolling back the component** to **Releasing**, indicating that the component is released in dark launch.

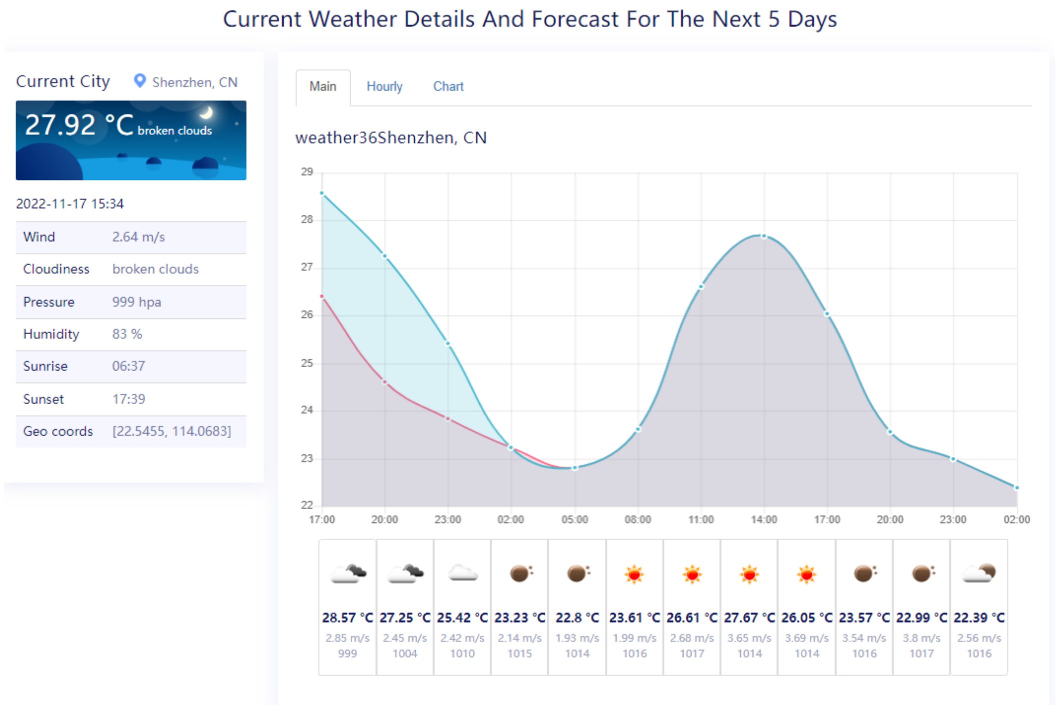
After the dark launch is successful, the **servicecomb.routeRule.weather** configuration item is delivered to the ServiceComb engine connected to the weather microservice.

You can view the configuration item from **Cloud Service Engine** > **Configuration Management**.

Step 10 Ensure that the dark launch version is working properly.

Access the application by referring to **Accessing an Application** and refresh the weather forecast page multiple times. The pages of the dark launch version and of the current version are periodically displayed based on the dark launch policy.

Figure 2-20 Current version (without UV data)



Current Weather Details And Forecast For The Next 5 Days

Current City Shenzhen, CN

27.92 °C broken clouds

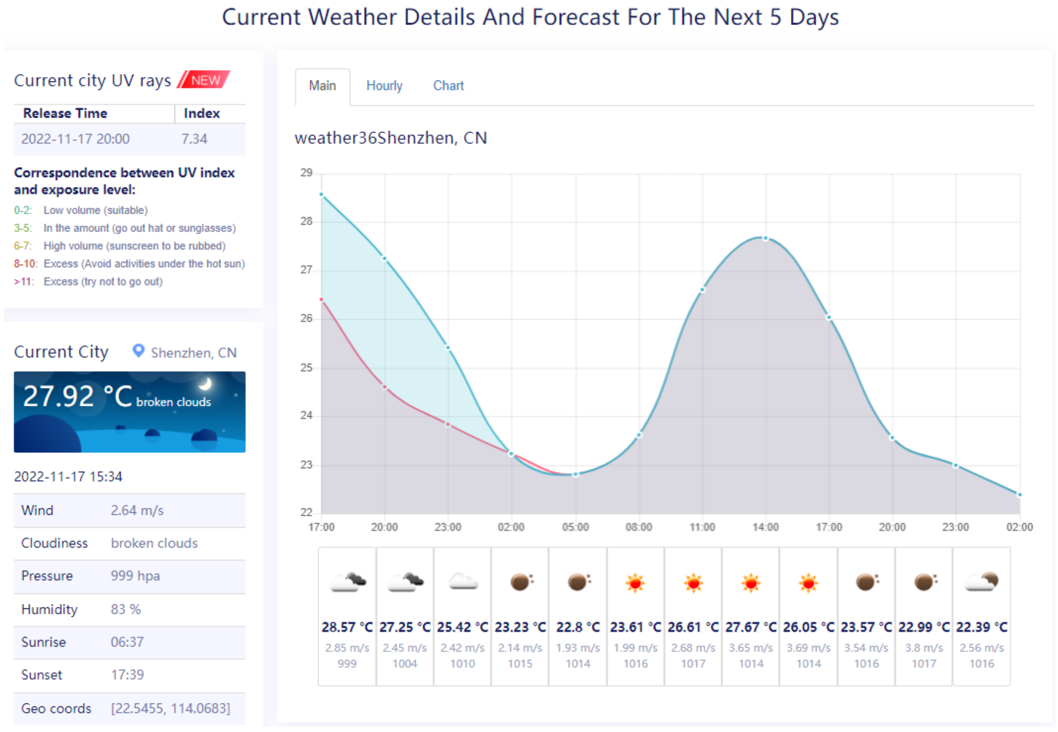
2022-11-17 15:34

Wind	2.64 m/s
Cloudiness	broken clouds
Pressure	999 hpa
Humidity	83 %
Sunrise	06:37
Sunset	17:39
Geo coords	[22.5455, 114.0683]

weather36Shenzhen, CN

28.57 °C	27.25 °C	25.42 °C	23.23 °C	22.8 °C	23.61 °C	26.61 °C	27.67 °C	26.05 °C	23.57 °C	22.99 °C	22.39 °C
2.85 m/s	2.45 m/s	2.42 m/s	2.14 m/s	1.93 m/s	1.99 m/s	2.68 m/s	3.65 m/s	3.69 m/s	3.54 m/s	3.8 m/s	2.56 m/s
999	1004	1010	1015	1014	1016	1017	1014	1014	1016	1017	1016

Figure 2-21 Dark launch version (with UV data)



Current Weather Details And Forecast For The Next 5 Days

Current city UV rays NEW

Release Time	Index
2022-11-17 20:00	7.34

Correspondence between UV index and exposure level:

0-2:	Low volume (suitable)
3-5:	In the amount (go out hat or sunglasses)
6-7:	High volume (sunscreen to be rubbed)
8-10:	Excess (Avoid activities under the hot sun)
>11:	Excess (try not to go out)

Current City Shenzhen, CN

27.92 °C broken clouds

2022-11-17 15:34

Wind	2.64 m/s
Cloudiness	broken clouds
Pressure	999 hpa
Humidity	83 %
Sunrise	06:37
Sunset	17:39
Geo coords	[22.5455, 114.0683]

weather36Shenzhen, CN

17:00	20:00	23:00	02:00	05:00	08:00	11:00	14:00	17:00	20:00	23:00	02:00
28.57 °C	27.25 °C	25.42 °C	23.23 °C	22.8 °C	23.61 °C	26.61 °C	27.67 °C	26.05 °C	23.57 °C	22.99 °C	22.39 °C
2.85 m/s	2.45 m/s	2.42 m/s	2.14 m/s	1.93 m/s	1.99 m/s	2.68 m/s	3.65 m/s	3.69 m/s	3.54 m/s	3.8 m/s	2.56 m/s
999	1004	1010	1015	1014	1016	1017	1014	1014	1016	1017	1016

----End

2.6 Microservice Governance

Scenarios

ServiceComb engines provide governance policies such as load balancing, service degradation, rate limiting, fault tolerance, circuit breaker, fault injection, blacklist, and whitelist.

User Story

You can configure governance policies in advance based on actual service scenarios to flexibly respond to service requirement changes and ensure stable running of applications.

Service degradation: In this practice, if the number of frontend requests increases sharply, the system responds slowly or may even break down. In this case, you can degrade the forecast microservice from fusionweather and request only important real-time weather data to ensure the proper running of important service functions and restore the service when traffic peaks are over.

Service Degradation

ServiceStage supports service degradation by microservice or API. The following uses the forecast microservice as an example.

Step 1 Log in to ServiceStage.

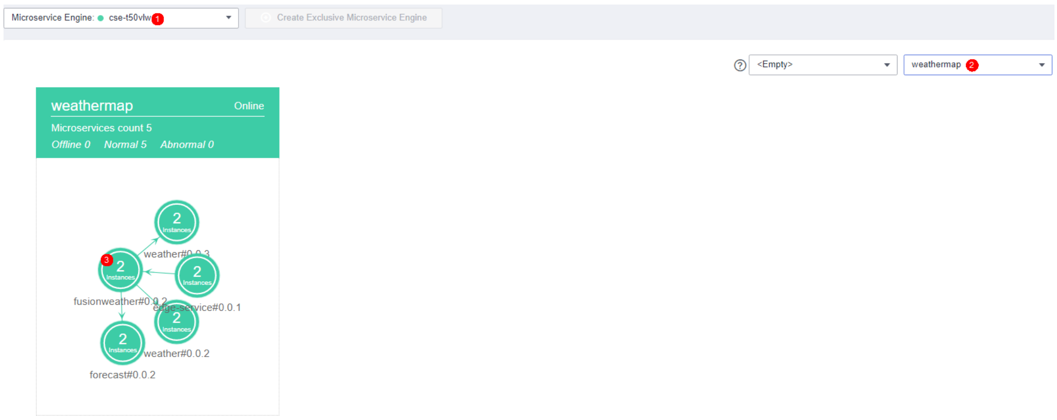
Step 2 Choose **Cloud Service Engine** > **Microservice Governance**.

Step 3 Select the ServiceComb engine where the weather forecast component is deployed from the **Microservice Engine** drop-down list.

Step 4 Select **weathermap** from the **All applications** drop-down list.

Step 5 Click the **fusionweather** microservice. The **Microservice Governance** page is displayed.

Figure 2-22 Accessing the Microservice Governance page

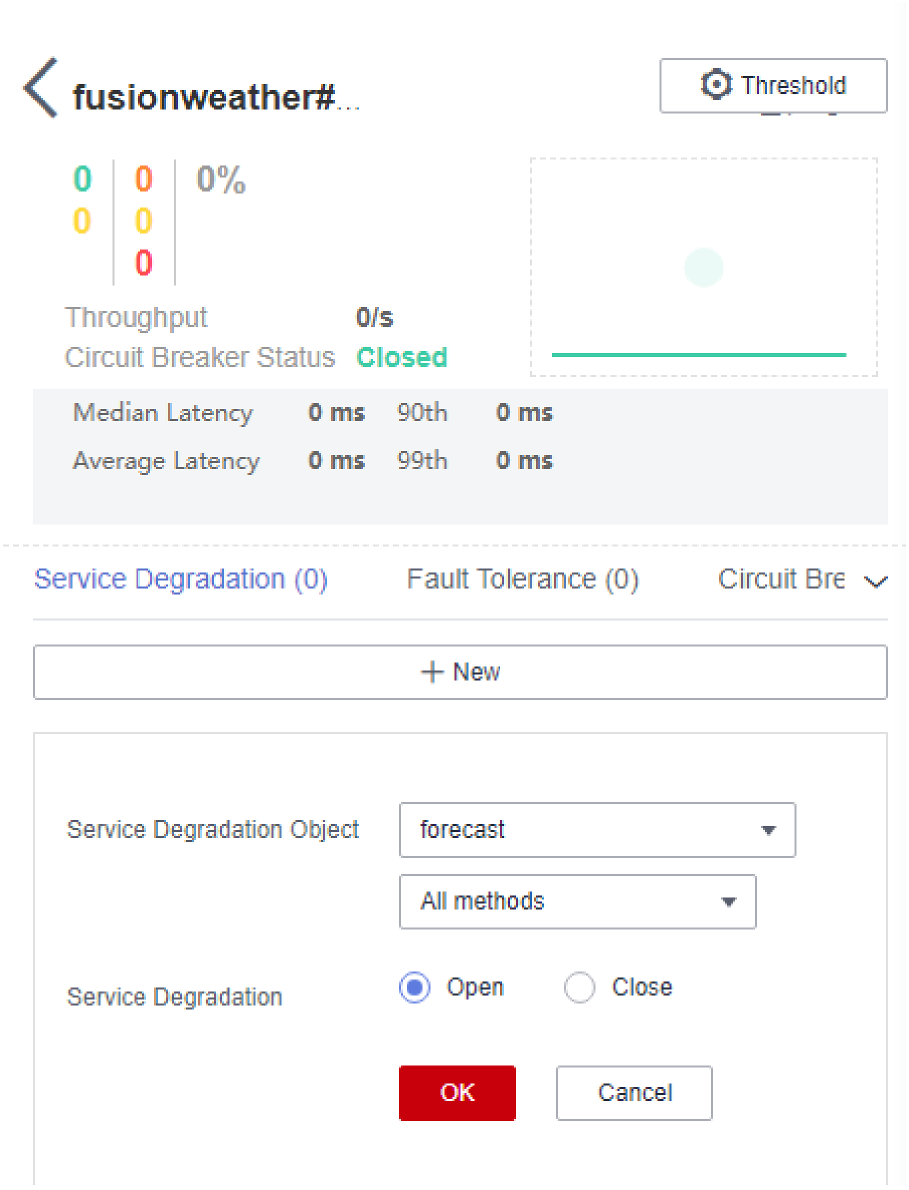


weathermap Online
Microservices count 5
Offline 0 Normal 5 Abnormal 0

Step 6 Set a service degradation policy.

1. Select **Service Degradation**.
2. Click **New**.
3. Set **Service Degradation Object** to **forecast**.
4. Set **Service Degradation** to **Open**.
5. Click **OK**.

Figure 2-23 Setting a service degradation policy



Throughput 0/s
Circuit Breaker Status Closed

Median Latency	0 ms	90th	0 ms
Average Latency	0 ms	99th	0 ms

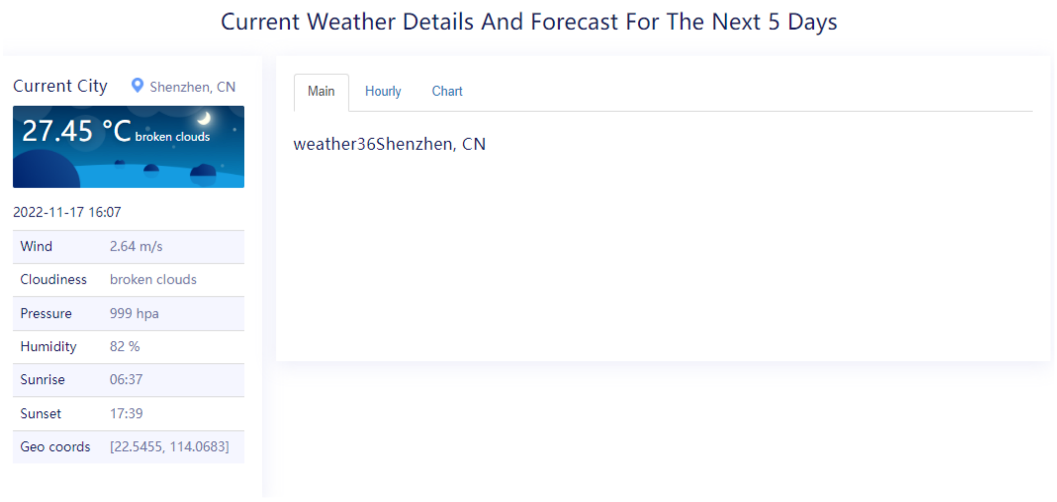
Service Degradation Object: forecast
All methods

Service Degradation: (● Open) ○ Close

Step 7 Check the configurations.

Access the application. The weather forecast on the right is blank.

Figure 2-24 Microservice degraded


Current Weather Details And Forecast For The Next 5 Days

Current City Shenzhen, CN

27.45 °C broken clouds

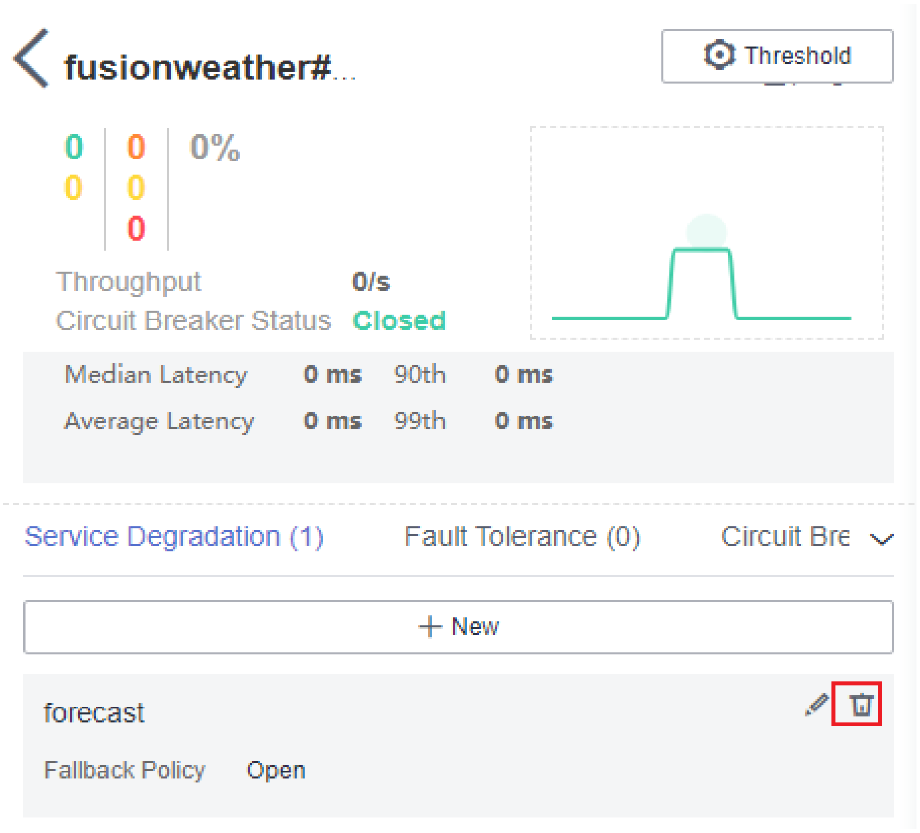
2022-11-17 16:07

Wind	2.64 m/s
Cloudiness	broken clouds
Pressure	999 hpa
Humidity	82 %
Sunrise	06:37
Sunset	17:39
Geo coords	[22.5455, 114.0683]

weather36Shenzhen, CN

Step 8 Click 🗑 to delete the service degradation policy to prevent it from affecting user experience.

Figure 2-25 Deleting a policy



Throughput 0/s
Circuit Breaker Status Closed

Median Latency 0 ms 90th 0 ms
Average Latency 0 ms 99th 0 ms

forecast
Fallback Policy Open

----End

2.7 FAQs

2.7.1 What Should I Do If a Weather Forecast Application with the Same Name Exists in the Current Environment?

Symptom

When you create a weather forecast application with a specified name (for example, **weathermap**) on ServiceStage, the system displays the error "SVCSTG.00100458: The application name already exists" indicating that an application with the same name exists in the application list.

Solution

Step 1 When creating an application, set **Name** to a unique application name, for example, **weathermap_test**.

Step 2 Click the created weather forecast application, for example, **weathermap_test**. The application **Overview** page is displayed.

Step 3 Click **Environment Variables** and select an environment (for example, **env-test**) from the drop-down list.

Step 4 Click **Add Environment Variable** to configure environment variables.

1. Set **Name** based on the selected source code repository branch by referring to the following table.

Technology Used by Microservice Components	Name
Java Chassis	servicecomb_service_application
Spring Cloud	spring_cloud_servicecomb_discovery_appName

2. Set **Variable/Variable Reference** to the name of the created application, for example, **weathermap_test**.

Step 5 Click **Submit**.

----End

3 Enabling Security Authentication for an Exclusive ServiceComb Engine

Overview

A microservice engine may be used by multiple users. Different users must have different microservice engine access and operation permissions based on their responsibilities and permissions. If **Security Authentication** is enabled for an exclusive microservice engine, grant different access and operation permissions to users based on the roles associated with the accounts used by the users to access the microservice engine.

The exclusive ServiceComb engine supports security authentication based on the Role-Based Access Control (RBAC) policy and allows you to enable or disable security authentication.

After security authentication is enabled for an engine, the security authentication account and password must be configured for all microservices connected to the engine. Otherwise, the microservice fails to be registered, causing service loss.

If security authentication is not enabled for the ServiceComb engine and security authentication parameters are configured for the microservice components connected to the ServiceComb engine, the normal service functions of the microservice components are not affected.

Application Scenarios

This section describes how to enable security authentication for an exclusive ServiceComb engine and ensure that services of microservice components connected to the engine are not affected.

Procedure

Step 1 Upgrade SDK used by microservice components.

To enable the security authentication function, SDK must support the security authentication function. If the SDK version used by the current microservice components is earlier than the required version (Spring Cloud Huawei requires 1.6.1 or later, and Java chassis requires 2.3.5 or later), you need to upgrade SDK.

Step 2 Configure security authentication parameters for microservice components.

Before enabling security authentication for a ServiceComb engine, configure security authentication parameters for the microservice components that have been connected to the engine. To configure security authentication parameters, you need to configure the security authentication account and password:

- Configuring the security authentication account and password for a Spring Cloud microservice component

  For details about how to obtain the security authentication account and password, see **Accounts**.

Table 3-1 Configuring the security authentication account and password for a Spring Cloud microservice component

Configuration File Configuration	Environmental Variables Injection

Add the following configurations to the **bootstrap.yml** file of the microservice. If they are configured, skip this step.
`spring:`
`  cloud:`
`    servicecomb:`
`      credentials:`
`        account:`
`          name: test   #Security authentication account. Set this parameter based on the site requirements.`
`          password: mima  #Password of the security authentication account. Set this parameter based on the site requirements.`
`          cipher: default`
By default, the user password is stored in plaintext, which cannot ensure security. You are advised to encrypt the password for storage. For details, see **Custom Encryption Algorithms for Storage**.
	Add the following environment variables. For details, see **Manually Adding an Environment Variable**.
	– spring_cloud_servicecomb_credentials_account_name: security authentication account. Set this parameter based on the site requirements.
	– spring_cloud_servicecomb_credentials_account_password: password of the security authentication account. Set this parameter based on the site requirements.

- Configuring the security authentication account and password for a Java chassis microservice component

  For details about how to obtain the security authentication account and password, see **Accounts**.

Table 3-2 Configuring the security authentication account and password for a Java chassis microservice component

Configuration File Configuration	Environmental Variables Injection

Add the following configurations to the **microservice.yml** file of the microservice. If they are configured, skip this step.
`servicecomb:`
`  credentials:`
`    rbac.enabled: true  #Whether to enable security authentication. Set this parameter based on the site requirements.`
`    cipher: default`
`    account:`
`      name: test #Security authentication account. Set this parameter based on the site requirements.`
`      password: mima #Password of the security authentication account. Set this parameter based on the site requirements.`
`      cipher: default`
	Add the following environment variables. For details, see **Manually Adding an Environment Variable**.
	– servicecomb_credentials_rbac_enabled: whether to enable security authentication. Set this parameter based on the site requirements. true: security authentication is enabled; false: security authentication is disabled.
	– servicecomb_credentials_account_name: security authentication account. Set this parameter based on the site requirements.
	– servicecomb_credentials_account_password: password of the security authentication account. Set this parameter based on the site requirements.

Step 3 Enable security authentication for an exclusive ServiceComb engine. For details, see **Enabling Security Authentication**.

NOTE

After security authentication is enabled, if security authentication parameters are not configured for the microservice components connected to the engine, or the security authentication account and password configured for the microservice components are incorrect, the heartbeat of the microservice components fails and the service is forced to go offline.

----End

4 Connecting ServiceComb Engine Dashboard Data to AOM through ServiceStage

Background

For Java chassis applications connected to the ServiceComb engine, the real-time monitoring data on the ServiceComb engine dashboard is retained for 5 minutes by default. To permanently store historical monitoring data for subsequent query and analysis, use the custom metric monitoring function of ServiceStage to connect the microservice data displayed on the ServiceComb engine dashboard to AOM.

This section uses the application deployed using a software package as an example to describe how to complete the connection.

Procedure

Step 1 Add dependency.

In the development environment, open the application project that requires persistent storage of historical monitoring data and add the following dependency to the **pom** file of the microservice:

```
<dependency>
    <groupId>org.apache.servicecomb</groupId>
    <artifactId>metrics-core</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.servicecomb</groupId>
    <artifactId>metrics-prometheus</artifactId>
</dependency>
```

Step 2 Recompile and package the application project to which the dependency has been added, and upload the package.

- Upload the software package to the SWR software repository. For details, see **Uploading the Software Package** .
- Upload the software package to the OBS bucket. For details, see **Streaming Upload (PUT)**.

- If you need to use JFrog (example) as the software package repository, you can download the package from the HTTP/HTTPS custom file address. Upload the software package to the corresponding custom file address in advance.

Step 3 Deploy the application component.

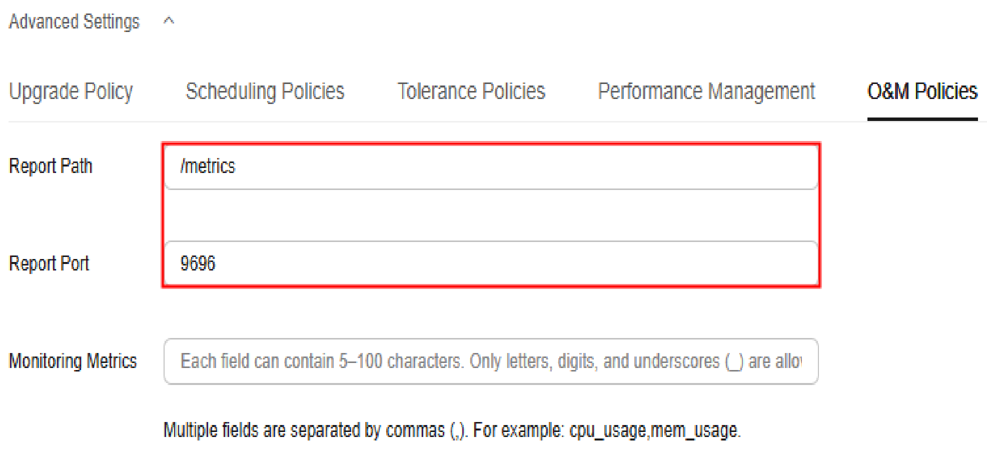
- To deploy a new component, go to **Step 4**.
- If the component has been deployed, go to **Step 5**.

Step 4 Deploy the component that is packaged and uploaded in **Step 2**. For details, see **Creating and Deploying a Component Based on a Container Using UI Configurations**.

1. During component deployment, choose **Advanced Settings** > **O&M Policy** and configure the following parameters:

Parameter	Value
Report Path	/metrics
Report Port	9696

Figure 4-1 Setting custom monitoring



Report Path: /metrics

Report Port: 9696

Monitoring Metrics: Each field can contain 5–100 characters. Only letters, digits, and underscores (_) are allo

Multiple fields are separated by commas (,). For example: cpu_usage,mem_usage.

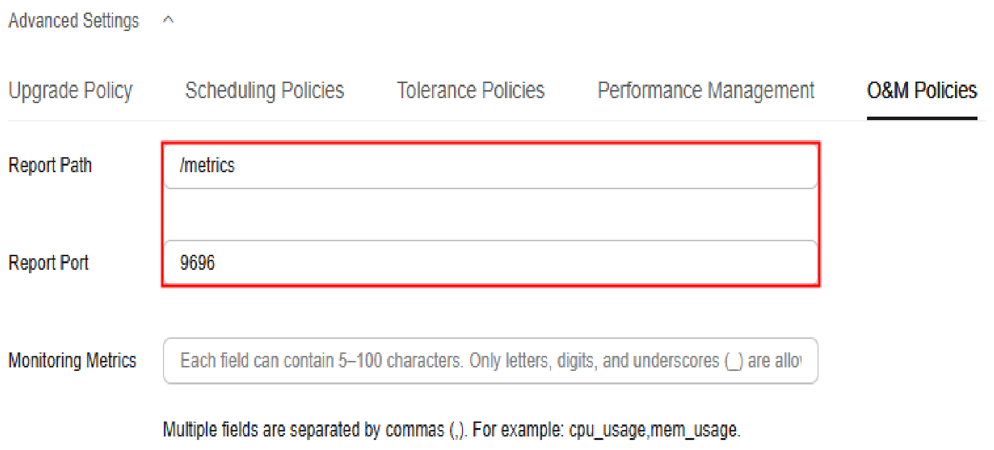
2. After the component is successfully deployed, go to **Step 6**.

Step 5 Connect monitoring metrics to AOM.

1. Log in to ServiceStage.
2. Choose **Application Management**.
3. Click the application where the component is located. The **Overview** page of the application is displayed.
4. In the **Component List** area, click the target component. The component overview page is displayed.
5. Click **Deploy**.
6. Select **Single-batch Release** and click **Next**.
7. Choose **Advanced Settings** > **O&M Policy** and configure the following parameters:

Parameter	Value
Report Path	/metrics
Report Port	9696

Figure 4-2 Monitoring metrics



Report Path: /metrics

Report Port: 9696

Monitoring Metrics: Each field can contain 5–100 characters. Only letters, digits, and underscores (_) are allo

Multiple fields are separated by commas (,). For example: cpu_usage,mem_usage.

8. Click **Deploy** and wait until the component is redeployed successfully.

Step 6 On the AOM console, view monitoring metrics and export monitoring data. For details, see **Metric Monitoring**.

----End

5 Migrating the Registered Microservice Engine Using ServiceStage Without Code Modification

Context

This section describes how to migrate the microservice application components that are developed using the Java chassis microservice framework and registered with the professional ServiceComb engine to the exclusive ServiceComb engine without any code modification.

Migrating the registered microservice engine will interrupt services. Evaluate the migration and select a proper time window before the migration.

Prerequisites

You have created an exclusive ServiceComb engine with security authentication disabled. For details, see **Creating a Microservice Engine**. The VPC and subnet where the engine is located are the same as those in the environment selected during microservice application component deployment.

Procedure

Step 1 Log in to ServiceStage.

Step 2 Delete the deployed microservice application component instances.

1. Choose **Application Management**.
2. Click the application where the microservice application is located. The **Overview** page is displayed.
3. In the **Component List** area, select the components to be deleted and click **Bulk Delete**.
4. In the displayed dialog box, click **OK**.

Step 3 Modify the environment for deploying microservice application components.

1. Click < in the upper left corner to return to the **Application Management** page.

2. Choose **Environment Management**.
3. Click the environment where the microservice application is deployed to go to the **Overview** page.
4. Choose **ServiceComb Engines** under **Middleware**.
5. Select **Cloud Service Engine** and click **Remove**.
6. Click **Manage Resource**.
7. Select the created exclusive ServiceComb engine and click **OK**.

Step 4 Redeploy the microservice application component. For details, see **Creating a Component Based on a Container Using UI Configurations**.

----End

6 Hosting a Spring Boot Application on ServiceStage

6.1 Preparations

Spring Boot is an open-source application development framework based on the Spring framework. It helps you quickly build production-level applications that can run independently.

This best practice uses the sample code provided by Spring to help you quickly deploy, access, and upgrade Spring applications on ServiceStage.

Preparing Resources

To facilitate subsequent operations, ensure that:

1. Create a VPC. For details, see **Creating a VPC**.
2. Create a CCE standard cluster. Set **Cluster Scale** to **50 nodes** and **Master Nodes** to **Single**. For details, see **Buying a CCE Standard/Turbo Cluster**.
   - The VPC to which the CCE cluster belongs is the one created in **1**.
   - The cluster must contain at least one ECS node with 8 vCPUs and 16 GB memory or two ECS nodes with 4 vCPUs and 8 GB memory and be bound to an EIP. For details, see **Creating a Node**.
3. Create an application load balancer and bind an EIP to it. For details, see **Creating a Dedicated Load Balancer**.
4. You have registered and obtained a public domain name from the domain name provider. For details, see **Creating a Public Zone**.

Registering a GitHub Account and Forking the Source Code

Step 1 **Register a GitHub account**.

Step 2 **Log in to GitHub**.

Step 3 Go to the source code repository.

- Source code repository address of the baseline version: **https://github.com/spring-guides/gs-spring-boot/tree/boot-2.7**
- Source code repository address of the dark launch version: **https://github.com/herocc19/gs-spring-boot-kubernetes**

Step 4 Fork the source code repository to your account. For details, see **Forking a repository**.

----End

Setting GitHub Repository Authorization

You can set GitHub repository authorization so that build projects and application components can use the authorization information to access the GitHub source code repository.

Step 1 Log in to ServiceStage.

Step 2 Choose **Continuous Delivery** > **Repository Authorization** > **Create Authorization**.

Step 3 Retain the default authorization name.

Step 4 Set **Repository Authorization**.

1. Select **GitHub**.
2. Select **OAuth** for **Method**.
3. Click **Use OAuth Authorization**.
4. After reading the service statement, select **I understand that the source code building function of the ServiceStage service collects the information above and agree to authorize the collection and use of the information**.
5. Click **OK**.
6. Enter your GitHub account and password to log in to GitHub for identity authentication. Wait until the authorization is complete.

Step 5 Click **OK**. You can view the created authorization in the repository authorization list.

----End

Creating an Organization

Step 1 Choose **Deployment Source Management** > **Organization Management**.

Step 2 Click **Create Organization**. On the displayed page, specify **Organization Name**. For example, **org-test**.

Step 3 Click **OK**.

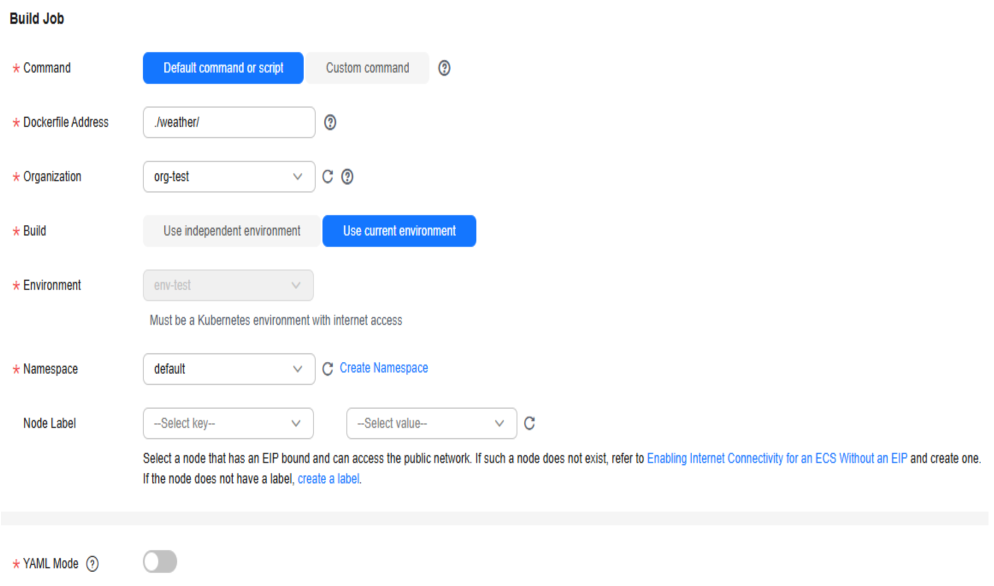
----End

Creating an Environment

Step 1 Choose **Environment Management** > **Create Environment**. Then set required environment parameters by referring to the following table, and retain the default values for other parameters.

Parameter	Description
Environment	Enter an environment name, for example, **env-test**.
Enterprise Project	**default** is selected by default. Enterprise projects let you manage cloud resources and users by project. It is available after you **enable the enterprise project function**.
Environment Type	Select **Kubernetes**.
HA Environment	Select **No**.
VPC	Select the VPC prepared in **Preparing Resources**. The VPC cannot be modified after the environment is created.
Configuration Mode	Select **Resource management**.

Figure 6-1 Configuring an environment



Build Job

* Command	Default command or script / Custom command
* Dockerfile Address	./weather/
* Organization	org-test
* Build	Use independent environment / Use current environment
* Environment	env-test Must be a Kubernetes environment with internet access
* Namespace	default (Create Namespace)
Node Label	--Select key-- --Select value-- Select a node that has an EIP bound and can access the public network. If such a node does not exist, refer to Enabling Internet Connectivity for an ECS Without an EIP and create one. If the node does not have a label, create a label.
* YAML Mode	

Step 2 Click **Create Now**.

Step 3 Choose **Clusters** under **Compute** and click **Bind now**.

Step 4 In the dialog box that is displayed, select the CCE cluster created in **Preparing Resources** and click **OK**.

Step 5 Choose **Load Balancers** under **Networking**. Then click **Manage Resource**.

Step 6 In the dialog box that is displayed, select the ELB resource created in **Preparing Resources** and click **OK**.

----End

Creating an Application

Step 1 Click < in the upper left corner to return to the **Environment Management** page.

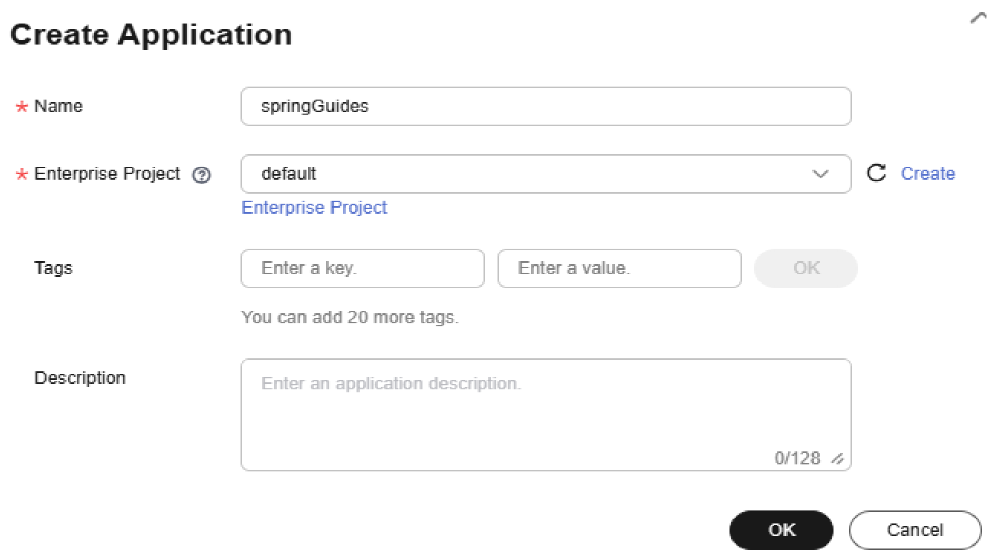
Step 2 Choose **Application Management** > **Create Application** and set basic application information.

1. **Name**: Enter an application name, for example, **springGuides**.
2. **Enterprise Project**: **default** is selected by default. Enterprise projects let you manage cloud resources and users by project.

   It is available after you **enable the enterprise project function**.

Step 3 Click **OK**.

Figure 6-2 Creating an application



Create Application

* Name	springGuides
* Enterprise Project	default
Tags	Enter a key. / Enter a value.
Description	Enter an application description. 0/128

----End

6.2 Deploying and Accessing a Spring Boot Application

To deploy and access a Spring Boot application, perform the following steps:

1. **Creating and Deploying a Spring Boot Application Component**
2. **Accessing a Spring Boot Application**

Creating and Deploying a Spring Boot Application Component

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**. The application list is displayed.

Step 3 Choose **More** > **Create Component** in the **Operation** column of the application created in **Creating an Application**, for example, **springGuides**.

Step 4 In the **Basic Information** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Component Name	Enter a component name, for example, **spring-boot**.
Component Version	Enter **1.0.0**.
Application	Select the application created in **Creating an Application**, for example, **springGuides**.
Environment	Select the environment created in **Creating an Environment**, for example, **env-test**.
Namespace	Select **default** to isolate component instances.

Step 5 In the **Component Package** area, set the following mandatory parameters. Retain the default values for other parameters.

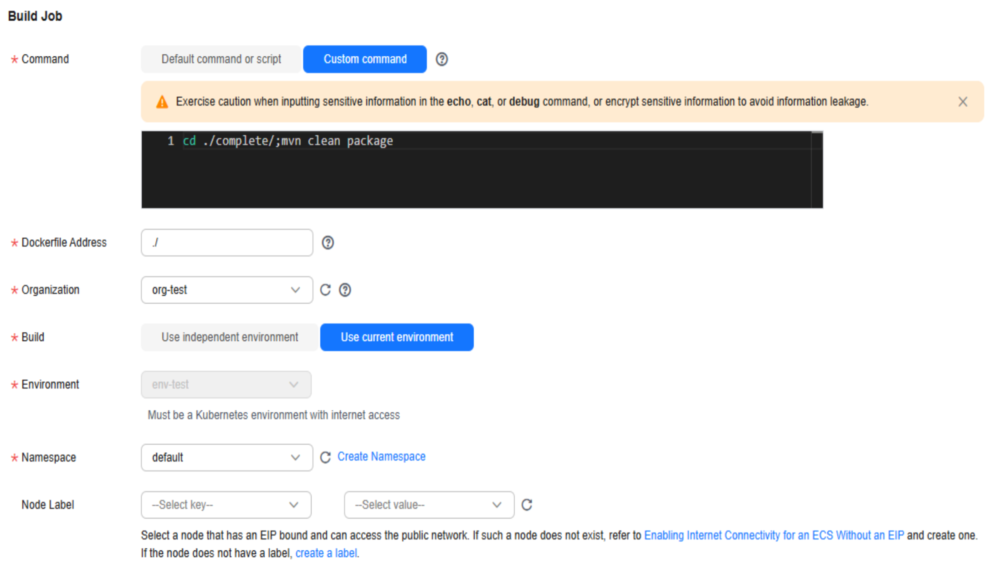
Parameter	Description
Stack	Select **Java**.
Source Code/ Software Package	1. Select **Source code repository**. 2. Select **GitHub**. 3. **Authorization**: Select the authorization information created when **Setting GitHub Repository Authorization**. 4. **Username/Organization**: Select the GitHub account created when **Registering a GitHub Account and Forking the Source Code**. 5. **Repository**: Select the Spring Boot source code repository that has been forked to your GitHub account when **Registering a GitHub Account and Forking the Source Code**. For example, **gs-spring-boot**. 6. **Branch**: Select **boot-2.7**.

Step 6 In the **Build Job** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Command	1. Select **Custom command**. 2. Enter the following command in the command text box: `cd ./complete/;mvn clean package`

Parameter	Description
Organization	Select the organization created in **Creating an Organization**. An organization is used to manage images generated during component build.
Build	Select **Use current environment** to use the CCE cluster in the deployment environment to which the component belongs to build an image. In the current environment, masters and nodes in the CCE cluster must have the same CPU architecture. Otherwise, the component build fails.
Namespace	Select **default** to isolate build data.

Figure 6-3 Configuring build parameters



Build Job

* Command: Default command or script | Custom command

Exercise caution when inputting sensitive information in the **echo**, **cat**, or **debug** command, or encrypt sensitive information to avoid information leakage.

```
cd ./complete/;mvn clean package
```

* Dockerfile Address: ./

* Organization: org-test

* Build: Use independent environment | Use current environment

* Environment: env-test

Must be a Kubernetes environment with internet access

* Namespace: default — Create Namespace

Node Label: --Select key-- --Select value--

Select a node that has an EIP bound and can access the public network. If such a node does not exist, refer to Enabling Internet Connectivity for an ECS Without an EIP and create one.
If the node does not have a label, create a label.

Step 7 Click **Next**.

Step 8 In the **Access Mode** area, click ⚪ to enable **Public Network Access** and set public network access parameters for the component by referring to the following table.

Parameter	Description
Public Network Access	Enable this option.
Public Network Load Balancer	By default, managed ELBs in the deployment environment to which the component belongs are selected.

Parameter	Description
Client Protocol	Retain the default value.
Domain Name	Select **Bind Domain Name** and enter the public domain name obtained in **Preparing Resources**.
Listening Port	Enter **8080**.

Step 9 Click **Create and Deploy**.

----End

Accessing a Spring Boot Application

Step 1 Click < in the upper left corner to return to the **Application Management** page.

Step 2 Click the application created in **Creating an Application** (for example, **springGuides**). The **Overview** page is displayed.

Step 3 In the **Component List** area, locate the row that contains the component name (for example, **spring-boot**) configured in **Creating and Deploying a Spring Boot Application Component** and click the access address in the **External Access Address** column to access the application.

If information similar to the following is displayed, the application is successfully deployed:

```
Greetings from Spring Boot!
```

----End

6.3 Upgrading a Component Version Using ELB Dark Launch

Step 1 Go to the ServiceStage console.

Step 2 Choose **Application Management**. The application list is displayed.

Step 3 Click the application created in **Creating an Application** (for example, **springGuides**). The **Overview** page is displayed.

Step 4 On the **Component List** tab, click the component created in **Deploying and Accessing a Spring Boot Application** (for example, **spring-boot**). The **Overview** page is displayed.

Step 5 In the upper right corner of the page, click **Upgrade**.

Step 6 Set **Upgrade Type** to **Dark Launch** and click **Next**.

Step 7 Set mandatory parameters for dark launch by referring to the following table. Retain the default values for other parameters.

Parameter	Description
Source Code/Image	The value is fixed to the GitHub source code repository selected during component creation and deployment. 1. Click **Modify**. 2. **Authorization**: Select the authorization information created when **Setting GitHub Repository Authorization**. 3. **Username/Organization**: Select the GitHub account created when **Registering a GitHub Account and Forking the Source Code**. 4. **Repository**: Select the Spring Boot source code repository that has been forked to your GitHub account when **Registering a GitHub Account and Forking the Source Code**. For example, **gs-spring-boot-kubernetes**. 5. **Branch**: Select **main**.
Command	1. Select **Custom command**. 2. Enter the following command in the command text box: `cd ./complete/;mvn clean package`
Component Version	Enter **1.0.1**.
Deployment Architecture	1. Click **Select**. 2. Select **Type 3: Connects the service to load balancer (microservice A implements dark launch)**. 3. Click **OK**.
Dark Launch Policy	Select **Traffic ratio-based**.
Traffic Ratio	● **Traffic Ratio**: percentage of traffic directed to the new version. Set it to **50**. ● **Current Traffic Ratio**: percentage of traffic directed to the current version. It is automatically set to **50**.
Instances Deployed for Dark Launch	Select **Canary (increase, then decrease instances)**.
First Batch of Dark Launch Instances	Set this parameter to **1**.
Deployment Batch with Remaining Instances	Set this parameter to **1**.

Step 8 Click **Upgrade**.

Wait until the component status changes from **Upgrading/Rolling back the component** to **Releasing**, indicating that the component is released in dark launch.

Step 9 Perform **Accessing a Spring Boot Application** multiple times. If "Greetings from Spring Boot!" and "Hello" are displayed alternately on the page, the dark launch version of ELB is released.

----End

7 Using GitLab to Interconnect with Jenkins to Automatically Build and Perform Rolling Upgrade on Components Deployed on ServiceStage

7.1 Overview

After the code is developed, you need to pack the code into an image package or JAR package on Jenkins before each rollout, upload the image package to SWR or the JAR package to OBS, and then use ServiceStage to upgrade the component version. This process is complex. Frequent version tests cause low development and O&M efficiency and poor user experience.

If you manage your code on GitLab and use ServiceStage with components deployed to host applications, you can use GitLab to interconnect with Jenkins for automatic build and packaging to upgrade the components deployed on ServiceStage.

This practice uses the shell script output after Jenkins build and packaging to automatically build and package code after integration and perform rolling upgrade on components deployed on ServiceStage.

7.2 Preparations

7.2.1 Preparing the Jenkins Environment

Environment Description

Install Jenkins on a Linux VM. The following lists the environment information used in this practice. If you use an image package for deployment, install Docker on the VM.

- VM: CentOS 7.9

- Jenkins: 2.319.3
- Git: installed using yum
- JDK: 11.0.8
- Apache Maven: 3.8.6

NOTE

The following parameter needs to be added to start Jenkins:

```
-Dhudson.security.csrf.GlobalCrumbIssuerConfiguration.DISABLE_CSRF_PROTECTION=true
```

Otherwise, GitLab fails to interconnect with Jenkins. The error is as follows:

```
HTTP Status 403 - No valid crumb was included in the request
```

Downloading and Installing Related Software

- Download and install Jenkins.

  Download: **https://mirrors.jenkins.io/war-stable/**. Install: **https://www.jenkins.io/doc/book/installing/**.
- Install Git to pull code for building commands.
  ```
  yum install git –y
  ```
- Download JDK.

  https://www.oracle.com/java/technologies/downloads/#java11
- Download Maven.

  https://maven.apache.org/download.cgi
- Install Docker to pack the image package and upload it to the image repository.
  ```
  yum install docker
  ```

Verifying the Installation

- Git
  ```
  [root@ecs-jenkins ~]# git version
  git version 1.8.3.1
  ```
- JDK
  ```
  [root@ecs-jenkins jar]# java -version
  openjdk version "1.8.0_345"
  OpenJDK Runtime Environment (build 1.8.0_345-b01)
  OpenJDK 64-Bit Server VM (build 25.345-b01, mixed mode)
  ```
- Maven
  ```
  [root@ecs-jenkins jar]# mvn -v
  Apache Maven 3.8.6 (84538c9988a25aec085021c365c560670ad80f63)
  Maven home: /root/app/maven/apache-maven-3.8.6
  Java version: 11.0.8, vendor: Huawei Technologies Co., LTD, runtime: /root/app/jdk11/jdk-11.0.8
  Default locale: en_US, platform encoding: UTF-8
  OS name: "linux", version: "3.10.0-1160.76.1.el7.x86_64", arch: "amd64", family: "unix"
  ```
- Docker
  ```
  [root@ecs-jenkins jar]# docker version
  Client:
   Version:         1.13.1
   API version:     1.26
   Package version: docker-1.13.1-209.git7d71120.el7.centos.x86_64
   Go version:      go1.10.3
   Git commit:      7d71120/1.13.1
   Built:           Wed Mar  2 15:25:43 2022
   OS/Arch:         linux/amd64
  Server:
   Version:         1.13.1
  ```

```
API version:     1.26 (minimum version 1.12)
Package version: docker-1.13.1-209.git7d71120.el7.centos.x86_64
Go version:      go1.10.3
Git commit:      7d71120/1.13.1
Built:           Wed Mar  2 15:25:43 2022
OS/Arch:         linux/amd64
Experimental:    false
```

7.2.2 Uploading Code to GitLab

This practice uses Java project code and uses Maven to build JAR packages.

Prerequisites

1. The Linux VM where Jenkins is located can access the GitLab code repository.
2. An account and a repository have been created on GitLab.

Procedure

Step 1 Log in to GitLab.

Step 2 Upload code to the code repository.

----End

7.2.3 Installing and Initializing obsutil

obsutil is used to upload software packages to OBS.

Prerequisites

1. You have obtained AK/SK. For details, see **Access Keys**.
2. You have obtained the endpoint of the region where ServiceStage is deployed. For details, see **Regions and Endpoints**.
3. You have created a bucket in OBS in the same region as ServiceStage where the component is deployed to store software packages. For details, see **Creating a Bucket**.

Procedure

Step 1 Log in to the Linux VM where Jenkins is installed and install obsutil. For details, see **Download and Installation**.

NOTE

Before installing obsutil, run the following command on the Linux VM where Jenkins is located to check the VM OS type:

```
echo $HOSTTYPE
```

- If the command output is **x86_64**, download the obsutil software package for the AMD 64-bit OS.
- If the command output is **aarch64**, download the obsutil software package for the Arm 64-bit OS.

Step 2 Initialize obsutil.

```
{path}/obsutil config -i=ak -k=sk -e={endpoint}
```

Where,

- *{path}* is the obsutil installation path, for example, **/root/tools/obsutil/obsutil_linux_amd64_5.4.6**.
- *{endpoint}* is the obtained endpoint of the region where ServiceStage is deployed.

Step 3 Check whether obsutil can be used to upload files to OBS.

1. Create a test file.
```
touch test.txt
```
2. Use obsutil to upload the file.
```
/root/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil cp test.txt obs://{OBS bucket name}
```
Replace *{OBS bucket name}* with the name of the OBS bucket to be used. In this example, the bucket name is **obs-mzc**. Upload the **test.txt** file created in the current directory to the obs-mzc bucket. If the "Upload successfully" is displayed, the upload is successful.
```
[root@ecs-jenkins jar]# /root/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil cp test1.txt obs://obs-mzc
Start at 2023-07-24 06:09:53.49127587 +0000 UTC
Parallel:      5                   Jobs:          5
Threshold:     50.00MB             PartSize:      auto
VerifyLength:  false               VerifyMd5:     false
CheckpointDir: /root/.obsutil_checkpoint
[-------------------------------------------------------------------------------------] 100.00% 138B/s 58B/58B 622ms
Upload successfully, 58B, n/a, /root/jar/test1.txt --> obs://obs-mzc/test1.txt, cost [621], status [200], request id [000001898684BD614014A659111ABF74]
```

----End

7.2.4 Installing and Initializing KooCLI

KooCLI is used to call ServiceStage APIs to upgrade components.

Install and initialize KooCLI to use it.

- Install KooCLI by **Method 1: Online Installation** or **Method 2: Using Software Package**
- **Initializing KooCLI**

Method 1: Online Installation

Step 1 Log in to the Linux VM where Jenkins is located.

Step 2 Run the following command:
```
curl -sSL https://hwcloudcli.obs.cn-north-1.myhuaweicloud.com/cli/latest/hcloud_install.sh -o ./hcloud_install.sh && bash ./hcloud_install.sh -y
```

----End

Method 2: Using Software Package

Step 1 Log in to the Linux VM where Jenkins is located and run the following command to check the VM OS type:
```
echo $HOSTTYPE
```
- If the command output is **x86_64**, the AMD 64-bit OS is used.
- If the command output is **aarch64**, the ARM 64-bit OS is used.

Step 2 Run the following command to download the software package:

- AMD
```
wget "https://hwcloudcli.obs.cn-north-1.myhuaweicloud.com/cli/latest/huaweicloud-cli-linux-amd64.tar.gz" -O huaweicloud-cli-linux-amd64.tar.gz
```
- ARM
```
wget "https://hwcloudcli.obs.cn-north-1.myhuaweicloud.com/cli/latest/huaweicloud-cli-linux-arm64.tar.gz" -O huaweicloud-cli-linux-arm64.tar.gz
```

Step 3 Run the following command to decompress the software package:

- AMD
```
tar -zxvf huaweicloud-cli-linux-amd64.tar.gz
```
- ARM
```
tar -zxvf huaweicloud-cli-linux-arm64.tar.gz
```

Step 4 Run the following command in the decompressed directory to create a soft link to the **/usr/local/bin** directory:

```
ln -s $(pwd)/hcloud /usr/local/bin/
```

Step 5 Run the following command to check whether the installation is successful:

```
hcloud version
```

If information similar to "KooCLI version: 3.4.1.1" is displayed, the installation is successful.

----End

Initializing KooCLI

Step 1 Log in to the Linux VM where Jenkins is located.

Step 2 Enter the command and press **Enter** to enter the interactive mode, and set the parameters as prompted. For details, see **Table 7-1**.

```
hcloud configure init
```

Table 7-1 Initial configurations

Parameter	Description
Access Key ID	Mandatory. For details, see **Access Keys**.
Secret Access Key	Mandatory. For details, see **Access Keys**.
Region	Optional. Region where ServiceStage is deployed. For details, see **Regions and Endpoints**.

Step 3 Add configuration parameters.

The corresponding CLI upgrade command may not be found. In this case, you need to add additional configuration.

```
hcloud configure set --cli-lang=cn
```

----End

7.2.5 Installing the Jenkins Plug-in and Configuring Jenkins

Before using GitLab to interconnect with Jenkins to automatically build and deploy components on ServiceStage, install the Jenkins plug-in and configure Jenkins global parameters.

- Install the Jenkins plug-in to interconnect with Git and use scripts during build.
- Configure Jenkins global parameters for the Jenkins pipeline packaging script to interconnect with Git to pull and package code.

Procedure

Step 1 Enter **http://***{IP address of the Linux VM where Jenkins is installed}***:8080** in the address box of the browser to log in to Jenkins.

Step 2 Choose **Manage Jenkins** > **Manage Plugins**.

Step 3 Click **Available**, search for plug-ins in **Table 7-2**, and install them.

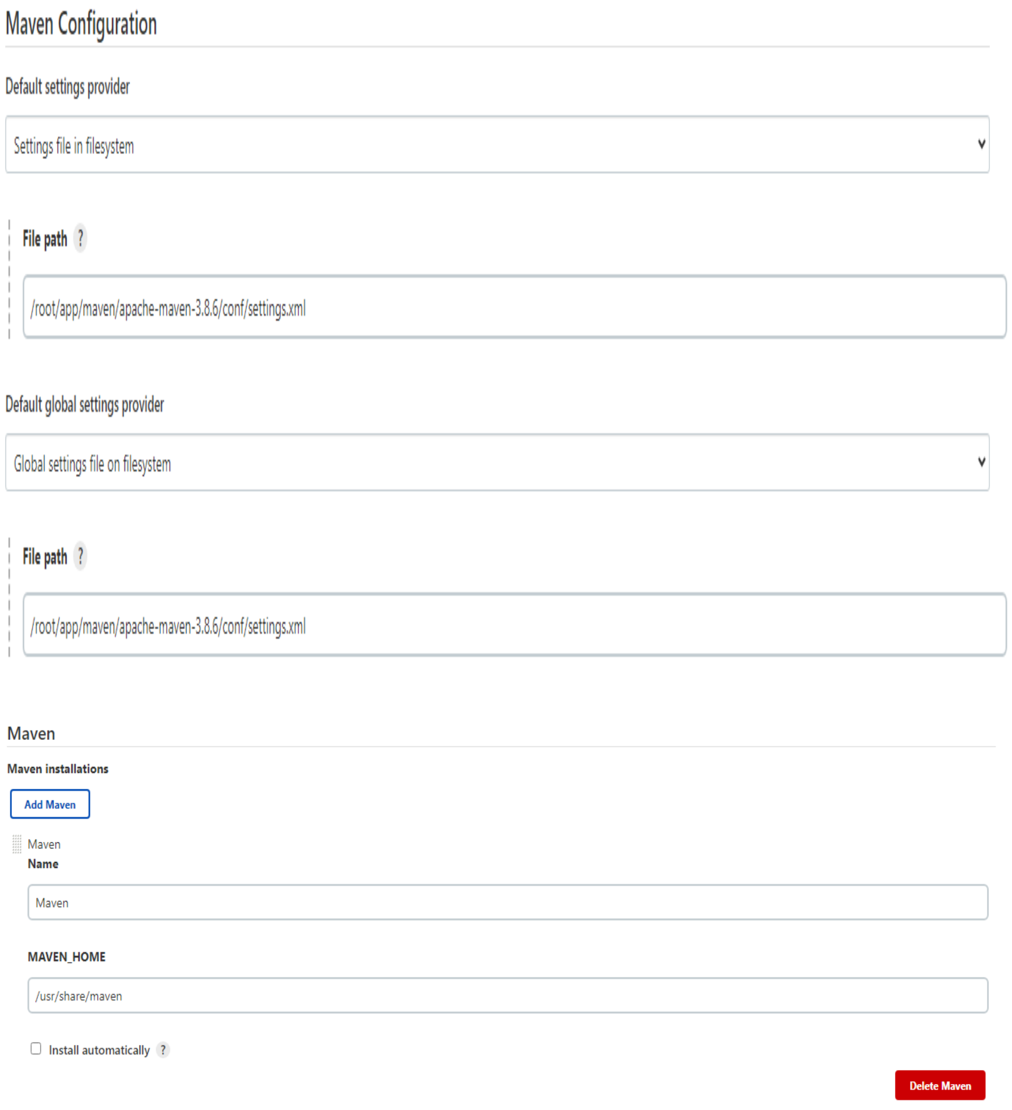
Table 7-2 Plug-in installation description

Plug-in	Mandatory	Description
Generic Webhook Trigger Plugin	Yes	Used to interconnect to the webhook of GitLab.
GitLab Plugin	Yes	Allows GitLab to trigger Jenkins build.
Pipeline: Basic Steps	Yes	Supports pipeline script syntax.
Pipeline: Build Step	Yes	Supports pipeline script syntax.
Pipeline: Stage Step	Yes	Supports pipeline script syntax.

Step 4 Choose **Manage Jenkins** > **Global Tool Configuration**.

Step 5 Configure Maven.

Replace **/root/app/maven/apache-maven-3.8.6** in the example with the actual Maven installation directory.



Maven Configuration

Default settings provider: Settings file in filesystem

File path: /root/app/maven/apache-maven-3.8.6/conf/settings.xml

Default global settings provider: Global settings file on filesystem

File path: /root/app/maven/apache-maven-3.8.6/conf/settings.xml

Maven

Maven installations

Maven

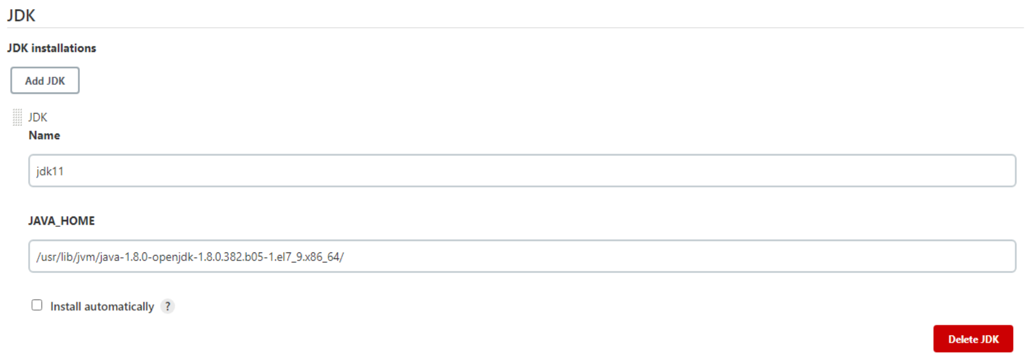
Name: Maven

MAVEN_HOME: /usr/share/maven

☐ Install automatically

Step 6 Configure JDK.

Replace **/root/app/jdk11/jdk-11.0.8** in the example with the actual JDK installation directory.



JDK

JDK installations

JDK

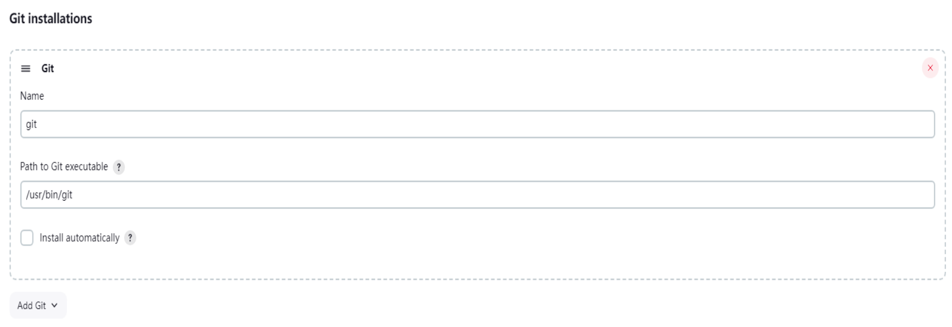
Name: jdk11

JAVA_HOME: /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.382.b05-1.el7_9.x86_64/

☐ Install automatically

Step 7 Configure Git.

Replace **/usr/bin/git** in the example with the actual Git installation directory.


Git installations

Git

Name: git

Path to Git executable: /usr/bin/git

☐ Install automatically

----End

7.2.6 Interconnection Tests

Before the operation, test the interconnection between Jenkins and GitLab to ensure that Jenkins can access GitLab through APIs.

Generating a GitLab Access Token

Step 1 Log in to GitLab.

Step 2 Move the cursor to the account name in the upper right corner and click **Edit profile**.

Step 3 Click **Access Tokens**, enter **Token name**, select **api**, and click **Create personal access token**.

The token will be displayed on the right of **Personal Access Tokens**.

NOTE

The token is displayed only when it is generated for the first time. Otherwise, you need to create it again next time. This token is used only for GitLab interconnection tests.

----End

Testing the Interconnection Between Jenkins and GitLab

Step 1 Enter **http://***{IP address of the Linux VM where Jenkins is installed}***:8080** in the address box of the browser to log in to Jenkins.

Step 2 Choose **Manage Jenkins** > **Jenkins Configuration**. In **Configuration**, select **Gitlab**.

Step 3 Configure the GitLab URL, click **Add** under **Credentials**, and select **Jenkins**.

Step 4 Select **Username with password** from the drop-down list, select **Gitlab API token**, and configure the GitLab access token in **Generating a GitLab Access Token** to the API token.

Step 5 Select **Gitlab API token** for **Credentials** and click **Test Connection**. If **Success** is displayed, the interconnection is successful.

----End

7.3 Using Jenkins to Automatically Build and Perform Rolling Upgrade on Components Deployed on ServiceStage

- Scenario 1: If a software package is generated using Jenkins, for example, a JAR package, use the software package deployment scenario in the script. During deployment, the built software package is uploaded to the OBS bucket and the ServiceStage component is upgraded.
- Scenario 2: If an image package is generated using Jenkins, use the image deployment scenario in the script. During deployment, the built image package is uploaded to the SWR image repository and the ServiceStage component is upgraded.

This section uses the scenario where the instance in **Configuring a Pipeline Script** is a JAR package as an example.

You need to create and deploy a component on ServiceStage in advance. For details, see **Creating and Deploying a Component Based on a Container Using UI Configurations**.

Creating a GitLab Credential

Use the account and password with the GitLab code repository permission to create a credential in Jenkins for pulling GitLab code.

Step 1 Enter **http://***{IP address of the Linux VM where Jenkins is installed}***:8080** in the address box of the browser to log in to Jenkins.

Step 2 Choose **Manage Jenkins** > **Jenkins Configuration**. In **Configuration**, select **Gitlab**.

Step 3 Click **Add** under **Credentials** and select **Jenkins**.

Step 4 Configure the GitLab account password and click **Add** to save the configuration.

Step 5 Choose **Manage Jenkins** > **Manage Credentials** to view the configured credentials.

The unique ID is used in **Configuring a Pipeline Script**.

----End

Creating a Pipeline Task

Step 1 Enter **http://***{IP address of the Linux VM where Jenkins is installed}***:8080** in the address box of the browser to log in to Jenkins.

Step 2 Click **New Item**.

Step 3 Enter the task name, for example, **test-upgrade**, select **Pipeline**, and click **OK**.

----End

Configuring a Build Trigger

Step 1 Configure the Jenkins build trigger.

1. Select **Build when a change is pushed to GitLab**, save the GitLab webhook URL (required when configuring GitLab webhook), and click **Advanced** in the lower right corner.
2. Select **Filter branches by regex** and configure the build task to be triggered after the specified branch is changed. In the example, the branch name is **main**. Click **Generate** to generate a secret token and save it. The token will be used for configuring GitLab webhook.

Step 2 Configure GitLab webhook.

1. Log in to GitLab and go to the code repository. In the example, the repository name is **test**. Select **Webhooks** in **settings**, and set **URL** and **Secret token** to the GitLab webhook URL and secret token obtained in **Step 1**.
2. Deselect **Enable SSL verification** for **SSL verification** and click **Add webhook**.

----End

Configuring a Pipeline Script

A pipeline script is a build command run during build.

Step 1 Click the **Pipeline** tab and select **Pipeline script** from the drop-down list.

Step 2 Configure the following pipeline script. In the example, the JAR package is used.

- Replace the parameters in the script with the actual parameters in your environment. For details about script parameters, see **Table 7-3**.
- The **upgrade.sh** script is invoked when the pipeline script is running. For details about the script, see **upgrade.sh**.
- Set **upgrade.sh** as an executable file.

```
node {
    //Code repository address, for example, http://10.95.156.58:8090/zmg/test.git.
    def git_url = '{Code repository address}'
    //GitLab credential ID.
    def credentials_id = '{GitLab credential ID}'
    //Name of the Git code repository branch, for example, main.
    def branch_name = '{Git code repository branch name}'
    //Path of the executable file for Maven installation, for example, /root/app/maven/apache-
maven-3.8.6/bin/mvn.
    def maven = '{Path of the executable file for Maven installation}'
    //Path for storing the upgrade.sh script, for example, /root/jar/upgrade.sh.
    def upgrade_shell = '{Path for storing the upgrade.sh script}'

    stage('Clone sources') {
        git branch: branch_name, credentialsId: credentials_id, url: git_url
    }
    stage('Build') {
        //Build a JAR package.
        sh "'$maven' clean package -Dmaven.test.failure.ignore=true -Dmaven.wagon.http.ssl.insecure=true -
Dmaven.wagon.http.ssl.allowall=true"
    }
    stage('upgrade') {
        //Execute the script and use the JAR package uploaded to OBS to upgrade the ServiceStage
component. The timeout period is 5 minutes.
        sh "timeout 300s '$upgrade_shell'"
```

```
  }
}
```

Table 7-3 Pipeline script parameters

Parameter	Mandatory	Type	Description
git_url	Yes	String	Address of the GitLab code repository.
credentials_id	Yes	String	GitLab credential ID configured using the account password. For details, see **Creating a GitLab Credential**.
branch_name	Yes	String	Name of the GitLab code repository branch.
maven	Yes	String	Path of the executable file for Maven installation, for example, **/root/app/maven/apache-maven-3.8.6/bin/mvn**.
upgrade_shell	Yes	String	Path for storing the **upgrade.sh** script on the VM where Jenkins is deployed, for example, **/root/jar/upgrade.sh**. For details about the script content, see **upgrade.sh**.

Step 3 Execute the build and verify the build result. For details, see **Build Verification**.

----End

upgrade.sh

Replace the parameters in the script with the actual parameters in your environment.

```
#!/bin/bash
#Project ID
project_id='{Project ID}'
#Application ID
application_id='{Application ID}'
#Component ID
component_id='{Component ID}'
#Batch information
rolling_release_batches=1
#Deployment type
deploy_type="package"


### Description:
### 1. Search for a string, as shown in key in the following code. If no string is found, defaultValue is returned.
### 2. Search for the nearest colon (:). The content following the colon is the value.
### 3. If there are multiple keys with the same name, only the first value is printed.
###
### 4. params: json, key, defaultValue
function getJsonValuesByAwk() {
    awk -v json="$1" -v key="$2" -v defaultValue="$3" 'BEGIN{
        foundKeyCount = 0
        pos = match(json, "\""key"\"[ \\t]*?:[ \\t]*");
        if (pos == 0) {if (foundKeyCount == 0) {print defaultValue;} exit 0;}
        ++foundKeyCount;
        start = 0; stop = 0; layer = 0;
        for (i = pos + length(key) + 1; i <= length(json); ++i) {
            lastChar = substr(json, i - 1, 1)
            currChar = substr(json, i, 1)

            if (start <= 0) {
                if (lastChar == ":") {
                    start = currChar == " " ? i + 1: i;
                    if (currChar == "{" || currChar == "[") {
                        layer = 1;
                    }
                }
            } else {
                if (currChar == "{" || currChar == "[") {
                    ++layer;
                }
                if (currChar == "}" || currChar == "]") {
                    --layer;
                }
                if ((currChar == "," || currChar == "}" || currChar == "]") && layer <= 0) {
                    stop = currChar == "," ? i : i + 1 + layer;
                    break;
                }
            }
        }

        if (start <= 0 || stop <= 0 || start > length(json) || stop > length(json) || start >= stop) {
            if (foundKeyCount == 0) {print defaultValue;} exit 0;
        } else {
            print substr(json, start, stop-start);
        }
    }'
}

#Query component information.
function getComponentInfo() {
    #Query component information.
    component_details=`hcloud ServiceStage ShowComponentInfo/v3 --project_id="$project_id" --
application_id="$application_id" --component_id="$component_id"`

    #Print component information.
    echo "$component_details"

    #Obtain the component name.
    test_name=`getJsonValuesByAwk  "$component_details" "name" "defaultValue"`
    lenj=${#test_name}
    component_name=${test_name:1:lenj-2}
    echo "name : $component_name"


    data_time=$(date +%Y.%m%d.%H%M)
    seconds=$(date +%S)
    component_version="${data_time}${seconds:1:1}"
    echo "version: $component_version"
}

#Image deployment scenario
function swr_image_upgrade() {

     #Image generated after project packaging: Image name:Version name
     machine_image_name='java-test:v1'
     #Path of the SWR image repository to which the image is uploaded
     swr_image_url='{Image repository address}/{Organization name}/{Image name}:{Version}'
     #AK, which is used to log in to the SWR image repository.
     AK='BMCKUPO9HZMI6BRDJGBD'
     #SWR login key, which is used to log in to the SWR image repository
     SK='{SWR login key}'
     #SWR image repository address
```

```
      swr_url='{SWR image repository address}'
      #Region
      region="{Region}"

      echo "upload image to swr"
      docker tag "$machine_image_name" "$swr_image_url"

      login_secret=`printf "$AK" | openssl dgst -binary -sha256 -hmac "$SK" | od -An -vtx1 | sed 's/[ \n]//g' |
sed 'N;s/\n//'`

      login_result=`docker login -u "$region"@"$AK" -p "$login_secret" "$swr_url"`
      #Print the result of logging in to the SWR image repository.
      echo "$login_result"
      push_result=`docker push "$swr_image_url"`
      #Print the image push result.
      #echo "$push_result"
      logout_result=`docker logout "$swr_url"`
      #Print the result of logging out of the SWR image repository.
      echo "$logout_result"
      #Clear all historical records. They may contain SWR login key information.
      #history -c

    echo "upgrade component"

    action_result=`hcloud ServiceStage ModifyComponent/v3 --project_id="$project_id" --
application_id="$application_id" --component_id="$component_id" --version="$component_version" --
runtime_stack.name="Docker" --runtime_stack.type="Docker" --source.kind="image" --
source.storage="swr" --source.url="$swr_image_url" --name="$component_name" --
deploy_strategy.rolling_release.batches=$rolling_release_batches --deploy_strategy.type="RollingRelease" `

}

#JAR package deployment scenario
function obs_jar_upgrade() {

    #Absolute path of the executable file for installing obsutil
    obsutil='/root/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil'
    #OBS bucket name
    bucket='obs://{OBS bucket name}'
    echo "upload jar to obs"
    #Upload the JAR package generated in the project to OBS.
    obs_result=`"$obsutil" cp ./target/*.jar "$bucket"`
    #Print the upload result.
    echo "$obs_result"
    #Link of the JAR package uploaded to OBS
    obs_jar_url='obs://{OBS bucket name}/{Jar package name}'

    echo "upgrade component"

    action_result=`hcloud ServiceStage ModifyComponent/v3 --project_id="$project_id" --
application_id="$application_id" --component_id="$component_id" --version="$component_version" --
runtime_stack.name="OpenJDK8" --runtime_stack.type="Java" --source.kind="package" --
source.storage="obs" --source.url="$obs_jar_url" --name="$component_name" --
deploy_strategy.rolling_release.batches=$rolling_release_batches --deploy_strategy.type="RollingRelease" `

}

#Query the job status every 15 seconds until the job is complete.
function waitDeployFinish() {
    sleep 10s
    id="$1"
    leni=${#id}
    id=${id:1:leni-2}
    echo "job_id= $id"
    job_status=""
    while [[ "$job_status" != "SUCCEEDED" ]]; do
        job_status_result=`hcloud ServiceStage ShowJobDetail/v2 --project_id="$project_id" --job_id="$id"`
        job_status=`getJsonValuesByAwk  "$job_status_result" "EXECUTION_STATUS" "defaultValue"`
        lenj=${#job_status}
        job_status=${job_status:1:lenj-2}
        echo "$job_status"
        if [[ "$job_status" != "RUNNING" && "$job_status" != "SUCCEEDED" ]]; then
            echo'Deployment failed'
            echo "$job_status_result"
            return
        fi
        sleep 15s
    done
    echo'Deployment succeeded'
}

function upgradeTask() {

    if [[ "$deploy_type" == "package" ]]; then
        obs_jar_upgrade
    elif [[ "$deploy_type" == "image" ]]; then
        swr_image_upgrade
    else
        return
    fi
    #Print the component upgrade result.
    echo "$action_result"
    #Obtain the job_id in the result.
    job_id=`getJsonValuesByAwk  "$action_result" "job_id" "defaultValue"`
    echo "$job_id"
    #Wait until the upgrade is complete.
    waitDeployFinish "$job_id"
}
function main() {
    getComponentInfo
    upgradeTask
}
main
```

Script Parameters

Parameter	Mandatory	Type	Description
region	Yes	String	Region name. For details, see **Obtaining Values**.
project_id	Yes	String	Project ID. For details, see **Obtaining Values**.
application_id	Yes	String	Application ID. For details, see **Obtaining Values**.
component_id	Yes	String	Component ID. For details, see **Obtaining Values**.
rolling_release_batches	Yes	int	Deployment batches.
deploy_type	Yes	String	Deployment type. ● package ● image

Parameter	Mandatory	Type	Description
obsutil	No	String	Absolute path for uploading JAR packages to OBS. This parameter is mandatory when software packages, such as JAR packages, are used for deployment. For example, **/root/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil**.
bucket	No	String	Path of the OBS bucket to which the package is uploaded. This parameter is mandatory when software packages are used for deployment. The format is **obs://***{Bucket name}*. For example, **obs://obs-mzc**.
obs_jar_url	No	String	This parameter is mandatory when software packages are used for deployment. Link of the software package uploaded to OBS. The format is **obs://***{Bucket name}***/***{Software package name}*. For example: **obs://obs-mzc/spring-demo-0.0.1-SNAPSHOT.jar**
machine_image_name	No	String	Image generated after Jenkins packaging and build. This parameter is mandatory when images are used for deployment. The format is *{Image name}***:***{Version}*. For example, **java-test:v1**.
swr_image_url	No	String	Path of the image package uploaded to the SWR image repository. This parameter is mandatory when images are used for deployment. The format is *{Image repository address}***/***{Organization name}***/***{Image package name}***:***{Version}*. The format of SWR image repository address is **swr.***{Project name of the region}***.myhuaweicloud.com**.
AK	No	String	This parameter is mandatory when images are used for deployment. AK is used to log in to the SWR image repository. For details, see **Access Keys**.
SK	No	String	This parameter is mandatory when images are used for deployment. SK is used to log in to the SWR image repository. For details, see **Access Keys**.
login_secret	No	String	This parameter is mandatory when images are used for deployment. Login secret is used to log in to the SWR image repository. Run the following command. The returned result is the login secret. printf "{AK}" | openssl dgst -binary -sha256 -hmac "{SK}" | od -An -vtx1 | sed 's/[\n]//g' | sed 'N;s/\n//' Replace {AK} and {SK} with the obtained AK and SK.

Parameter	Mandatory	Type	Description
swr_url	No	String	This parameter is mandatory when images are used for deployment. SWR image repository address. The format is **swr.***{Project name of the region}***.myhuaweicloud.com**.

Obtaining Values

- Obtain **region** and **project_id**.
  a. Log in to ServiceStage.
  b. Move the cursor to the username in the upper right corner and select **My Credentials** from the drop-down list.
  c. View the project and project ID of the region, that is, the values of **region** and **project_id**.
- Obtain **application_id** and **component_id**.
  a. Log in to ServiceStage.
  b. Choose **Component Management**.
  c. Click the corresponding component.
  d. In the **Configurations** area on the **Overview** page, choose **Container Settings** > **Environment Variable**.

     View **CAS_APPLICATION_ID** and **CAS_COMPONENT_ID**, that is, the values of **application_id** and **component_id**.

7.4 Build Verification

7.4.1 Manual Build

Step 1 Enter **http://***{IP address of the Linux VM where Jenkins is installed}***:8080** in the address box of the browser to log in to Jenkins.

Step 2 Click **My View**.

Step 3 Click the corresponding build task to go to the details page.

Step 4 Click **Build Now** to generate the build task.

The corresponding build task information is displayed in the **Build History** and **Stage View** areas. Move the cursor to a step to display the task status and log button. Click **log** to view logs.

Step 5 Log in to ServiceStage.

Step 6 Choose **Component Management**.

Step 7 In the **Component List** area, click the target component. The component overview page is displayed.

On the **Overview** page, check whether the component version and component package code source have been updated.

Step 8 Click **Deployment Record** to view the corresponding deployment record.

----End

7.4.2 Jenkins Build Triggered by GitLab

GitLab triggers Jenkins build in either of the following methods:

- Method 1: Use the configured webhook to push events and trigger Jenkins build task.
- Method 2: Modify the file of the specified branch in the build configuration to push events and trigger Jenkins build task.

This section uses method 1 as an example.

Procedure

Step 1 Log in to GitLab and go to the code repository.

Step 2 Click **Settings**, select **Webhooks**, and select **Push events** from the **Test** drop-down list.

Step 3 Enter **http://***{IP address of the Linux VM where Jenkins is installed}***:8080** in the address box of the browser to log in to Jenkins.

In the build execution status on the left, you can view the build tasks that have been triggered.

Step 4 Click the build task ID and choose **Console Output** to view the build output logs.

Step 5 Log in to ServiceStage.

Step 6 Choose **Component Management**.

Step 7 In the **Component List** area, click the target component. The component overview page is displayed.

On the **Overview** page, check whether the component version and component package code source have been updated.

Step 8 Click **Deployment Record** to view the corresponding deployment record.

----End

8 Using ServiceStage to Migrate Components Across AZs and Upgrade Components in Sequence Based on Release Management

8.1 Overview

In actual services, services need to be deployed in different AZs to improve availability due to equipment room faults.

However, when components are deployed in different AZs, each component must be configured as required. This is complex and error-prone. In addition, the components need to be deployed and run immediately after being created, and do not support on-demand deployment. If the component configurations are incorrect, the deployment fails. In this case, you need to delete the components, create them again, and then deploy them.

ServiceStage release management can be used to migrate and upgrade components across AZs.

- Batch clone release tasks can be used to migrate components across AZs.
- Batch upgrade release tasks can be used to upgrade components across AZs and specify the upgrade sequence of components in different AZs.

8.2 Preparations

Preparing Resources

1. Create a VPC. For details, see **Creating a VPC**.
2. Create two CCE clusters (for example, cce-az1 and cce-az2) in different AZs (for example, az1 and az2). Set **Cluster Scale** to **50 nodes** and **Master Nodes** to **Single**. When creating a cluster, select **Policy for distributing master nodes in a cluster** under **Master Nodes** and select **Custom** to allocate the master nodes of the two clusters to different AZs.

For details, see **Buying a CCE Standard/Turbo Cluster**.

- Each cluster must contain at least one ECS node with 8 vCPUs and 16 GB memory or two ECS nodes with 4 vCPUs and 8 GB memory. The AZ of the ECS nodes in the cluster must be the same as that of the master nodes in the CCE cluster.

  For details about how to add a node to the CCE cluster, see **Creating a Node**.
- The VPC to which the cluster belongs is the one created in **1**.

3. Create a bucket for storing software packages. For details, see **Creating a Bucket**.

Uploading Software Packages

Step 1 Download the following software packages:

- **weather-1.0.0.jar**
- **weather-beta-2.0.0.jar**

Step 2 Upload the preceding software packages to the bucket prepared in **Preparing Resources**.

For details about how to upload a software package, see **Streaming Upload (PUT)**.

----End

Creating Environments

Step 1 Log in to ServiceStage.

Step 2 Choose **Environment Management** > **Create Environment**. Then set required environment information by referring to the following table, and retain the default values for other parameters.

Parameter	Description
Environment	Enter an environment name. You can name the two environments (for example, env-cce-az1 and env-cce-az2) based on the AZs where the managed CCE clusters are located.
Enterprise Project	**default** is selected by default. Enterprise projects let you manage cloud resources and users by project. For details, see **Enabling the Enterprise Project Function**.
Environment Type	Select **Kubernetes**.
HA Environment	Select **Yes**.

Parameter	Description
VPC	Select the VPC prepared in **Preparing Resources**. The VPC cannot be modified after the environment is created.

Step 3 Click **Create Now**.

Step 4 Choose **Clusters** under **Compute** and click **Bind now**.

1. Select an AZ (for example, **az1**) from the **AZ** drop-down list.
2. Select a CCE cluster (for example, **cce-az1**) that can be bound in the AZ from the **Cluster** drop-down list.
3. Click **OK**.
4. Click **Add Cluster**.
5. Select the other AZ (for example, **az2**) from the **AZ** drop-down list.
6. Select a CCE cluster (for example, **cce-az2**) that can be bound in the AZ from the **Cluster** drop-down list.
7. Click **OK**.

----End

Creating an Application

Step 1 Click < in the upper left corner to return to the **Environment Management** page.

Step 2 Choose **Application Management** > **Create Application** and configure the application by referring to the following table.

Parameter	Description
Name	Enter an application name, for example, **test-app**.
Enterprise Project	**default** is selected by default. Enterprise projects let you manage cloud resources and users by project. For details, see **Enabling the Enterprise Project Function**.

Step 3 Click **OK**.

----End

Creating an Organization

Step 1 Choose **Deployment Source Management** > **Organization Management**.

Step 2 Click **Create Organization**. On the displayed page, specify **Organization Name**. For example, **org-test**.

Step 3 Click **OK**.

----End

8.3 Deploying a Component to a CCE Cluster

This section describes how to deploy a component to a CCE cluster in the environment (for example, **env-cce-az1**) created in **Preparations**.

Procedure

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**. The application list is displayed.

Step 3 Choose **More** > **Create Component** in the **Operation** column of the application created in **Creating an Application**, for example, **test-app**.

Step 4 In the **Basic Information** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Component Name	Enter a component name. It is recommended that the component name should contain the AZ information of the CCE cluster bound to the component environment, for example, **test-comp-az1**.
Component Version	Click **Generate**.
Application	Select the application created in **Creating an Application**, for example, **test-app**.
Environment	Select the environment created in **Creating Environments**, for example, **env-cce-az1**.
Cluster	Select the CCE cluster (for example, **cce-az1**) that is bound to the specified AZ in the environment.

Step 5 In the **Component Package** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Stack	Type of the technology stack used by the component. Select **Java**.
Source Code/ Software Package	Component package type. Select **JAR package**.

Parameter	Description
Upload Method	1. Select **OBS**. 2. Click **Software Package** and select **weather-1.0.0.jar** uploaded in **Uploading Software Packages**. 3. Click **OK**.

Step 6 In the **Build Job** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Organization	Select the organization created in **Creating an Organization**. An organization is used to manage images generated during component build.
Build	Select **Use current environment** to use the CCE cluster in the deployment environment to which the component belongs to build an image.
Select Cluster	Select the cluster (for example, **cce-az1**) selected in **Step 4** to build the component image.

Step 7 Click **Next**.

Step 8 Click **Create and Deploy** and wait until the component is created.

----End

8.4 Using a Release Task to Migrate Components Across AZs

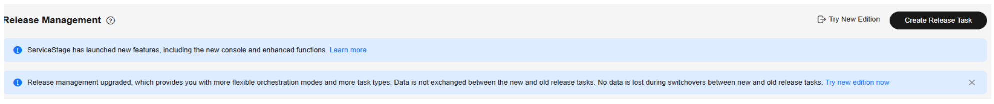
This section describes how to use batch clone of ServiceStage release management to migrate the components that have been deployed in cce-az1 in az1 when **Deploying a Component to a CCE Cluster** to cce-az2 in az2.

Procedure

Step 1 Log in to ServiceStage.

Step 2 Choose **Release Management**.

- If the **Release Management** page of the old version is displayed by default, go to **Step 4**.

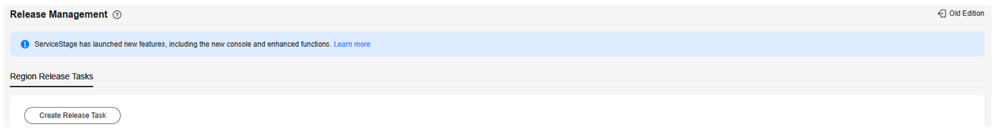


Release Management

ServiceStage has launched new features, including the new console and enhanced functions. Learn more

Release management upgraded, which provides you with more flexible orchestration modes and more task types. Data is not exchanged between the new and old release tasks. No data is lost during switchovers between new and old release tasks. Try new edition now

- If the **Release Management** page of the new version is displayed by default, go to **Step 3**.


Release Management
Old Edition
ServiceStage has launched new features, including the new console and enhanced functions. Learn more
Region Release Tasks
Create Release Task

Step 3 Click **Old Edition** to switch to the **Release Management** page of the old version.

Step 4 Click **Create Release Task**.

Step 5 Enter a release task name, for example, **release-clone**.

Step 6 Select **Bulk Clone** for **Operation Type**.

Step 7 Click **Add Component**.

Step 8 Select the component that has been successfully deployed in **Deploying a Component to a CCE Cluster** and click **OK**.

Step 9 Modify the component configuration information by referring to the following table.

Parameter	Description
Component Name	Enter a component name. It is recommended that the component name should contain the AZ information of the CCE cluster where the component is located, for example, **test-comp-az2**.
Cluster	Select a CCE cluster (for example, **cce-az2**) that is in a different AZ from the cluster in **Deploying a Component to a CCE Cluster**.

Step 10 Click **Complete and Execute** and wait until the release task is executed.

After the release task is executed, the component is cloned and deployed in another cluster.

----End

8.5 Using a Release Task to Upgrade Components in Batches Across AZs

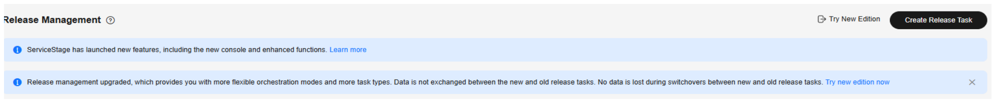
This section describes how to upgrade components across AZs based on batch upgrade of ServiceStage release management and specify the upgrade sequence of components in different AZs.

Procedure

Step 1 Log in to ServiceStage.

Step 2 Choose **Release Management**.

- If the **Release Management** page of the old version is displayed by default, go to **Step 4**.



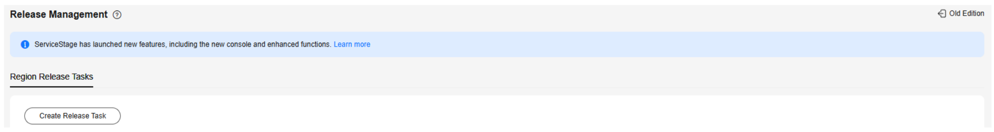
Release Management

Try New Edition | Create Release Task

ServiceStage has launched new features, including the new console and enhanced functions. Learn more

Release management upgraded, which provides you with more flexible orchestration modes and more task types. Data is not exchanged between the new and old release tasks. No data is lost during switchovers between new and old release tasks. Try new edition now

- If the **Release Management** page of the new version is displayed by default, go to **Step 3**.



Release Management

Old Edition

ServiceStage has launched new features, including the new console and enhanced functions. Learn more

Region Release Tasks

Create Release Task

Step 3 Click **Old Edition** to switch to the **Release Management** page of the old version.

Step 4 Click **Create Release Task**.

Step 5 Enter a release task name, for example, **release-upgrade**.

Step 6 Select **Bulk Upgrade** for **Operation Type**.

Step 7 Click **Add Component**.

Step 8 Select the components (for example, **test-comp-az1** and **test-comp-az2**) deployed in **Deploying a Component to a CCE Cluster** and **Using a Release Task to Upgrade Components in Batches Across AZs**, and click **OK**.

Step 9 Change the software package sources of the two components selected in **Step 8**.

1. Click ✎ in the **Image Package** column.
2. Move the cursor to the component software package card and click ✎.
3. Select the **weather-beta-2.0.0.jar** software package that has been uploaded when **Uploading Software Packages**.
4. Click **OK**.

Step 10 Change the **Deployment Sequence** of the two components to **1** and **2**.

Step 11 Click **Complete and Execute**.

View the component upgrade information. The release task upgrades two components in sequence as configured in **Step 10**.

----End

9 Using ServiceStage Component Templates to Automatically Create and Upgrade Components

9.1 Overview

A component template is a specification file that describes ServiceStage components and defines resources (such as ConfigMaps, Secrets, and Services) and configurations required for component running.

You can use a component template to provision components, resources required for component running, and configuration files to quickly create and upgrade components.

This practice provides the component template package **template-package-demo.zip** and image package **demo-app.tar**. **Table 9-1** describes the component template files contained in the component template package. The image package is the source of the component package for creating and deploying components and is defined by the image field in the **deployment.yaml** file in **Table 9-1**.

For details, see **Component Template Description**.

Table 9-1 Component template file description

Template File	Description
spec.yaml	Definition file, which describes the component file location and Kubernetes creation sequence. The file name cannot be changed.
variables.yaml	Variable file, which declares the variable information contained in the template package. The file name cannot be changed.
values.yaml	Value file, which contains the default values of variables. The file name cannot be changed.

Template File	Description
comp_demo	Directory where the component file is located. The directory name is specified by **spec.yaml**.
comp_demo/ss-config.yaml	Configuration file definition. The file name can be customized.
comp_demo/ss-component.yaml	Component definition. The file name can be customized.
comp_demo/deployment.yaml	Kubernetes resource definition. The file name can be customized.
comp_demo/configmap.yaml	
comp_demo/hpa.yaml	
comp_demo/role.yaml	
comp_demo/rolebinding.yaml	
comp_demo/secret.yaml	
comp_demo/service.yaml	
comp_demo/service-account.yaml	

This practice describes how to use a component template to quickly create and upgrade components.

9.2 Preparations

Preparing Resources

- Create a VPC. For details, see **Creating a VPC**.
- Create an elastic load balancer and bind it to an EIP. For details, see **Creating a Shared Load Balancer**.

  The VPC to which the load balancer belongs has been created.
- Create a CCE standard cluster. Set **Cluster Scale** to **50 nodes** and **Master Nodes** to **Single**.

  For details, see **Buying a CCE Standard/Turbo Cluster**.
  - The cluster must contain at least one ECS node with 8 vCPUs and 16 GB memory or two ECS nodes with 4 vCPUs and 8 GB memory.

    For details about how to add nodes to a CCE cluster, see **Creating a Node** .
  - The VPC to which the cluster belongs has been created.
- Create a DCS instance. For details, see **Buying a DCS Redis Instance**.
- Create a bucket for storing software packages. For details, see **Creating a Bucket**.

Obtaining and Uploading the Component Template Package

Step 1 Download the component template package **template-package-demo.zip**.

Step 2 Upload the package to the OBS bucket prepared in Preparing Resources.

For details, see Streaming Upload (PUT).

----End

Obtaining the Download Address of the Uploaded Image Package

Step 1 Download the image package **demo-app.tar**.

Step 2 Upload the package to the SWR image repository.

For details, see Uploading an Image.

Step 3 Obtain the image package download address. For details, see Obtaining an Image Pull Address.

----End

Creating an Environment

Step 1 Log in to ServiceStage.

Step 2 Choose **Environment Management** > **Create Environment**. Then set required environment information by referring to the following table, and retain the default values for other parameters.

Parameter	Description
Environment	Enter an environment name, for example, **template-demo-env**.
Enterprise Project	Enterprise projects let you manage cloud resources and users by project. For details, see Enabling the Enterprise Project Function.
Environment Type	Select **Kubernetes**.
HA Environment	Select **No**.
VPC	Select the VPC prepared in Preparing Resources. The VPC cannot be modified after the environment is created.
Configuration Mode	Select **Resource Management**.

Step 3 Click **Create Now**.

Step 4 Choose **Clusters** under **Compute** and click **Bind now**.

1. For **Cluster**, select the CCE cluster created in Preparing Resources.

2. Set **Namespace** to **default**.
3. Click **OK**.

Step 5 Choose **Load Balancers** under **Networking**. Then click **Manage Resource**.

Step 6 In the dialog box that is displayed, select the ELB resource created in **Preparing Resources** and click **OK**.

Step 7 Choose **Distributed Cache Service** under **Middleware** and click **Manage Resource**.

Step 8 In the dialog box that is displayed, select the DCS resource created in **Preparing Resources** and click **OK**.

----End

Creating an Application

Step 1 Click < in the upper left corner to return to the **Environment Management** page.

Step 2 Choose **Application Management** > **Create Application** and configure the application by referring to the following table.

Parameter	Description
Name	Enter an application name, for example, **template-demo-app**.
Enterprise Project	**default** is selected by default. Enterprise projects let you manage cloud resources and users by project. For details, see **Enabling the Enterprise Project Function**.

Step 3 Click **OK**.

----End

Creating an Organization

Step 1 Choose **Deployment Source Management** > **Organization Management**.

Step 2 Click **Create Organization**.

Step 3 Enter an organization name, for example, **org-test**.

Step 4 Click **OK**.

----End

9.3 Automatically Creating a Component Using a Component Template

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**. The application list is displayed.

Step 3 Click the application created in **Creating an Application** (for example, **template-demo-app**). The **Overview** page is displayed.

Step 4 Click **Create Component** > **Use Template** and set component parameters by referring to the following table.

Parameter	Description
Application	By default, the application selected in **Step 3** is selected, for example, **template-demo-app**.
Environment	Select the Kubernetes environment created when **Creating an Environment**, for example, **template-demo-env**.
Set Template Package	1. Select **OBS**. 2. Click **Select Software Package**. 3. Select **template-package-demo.zip** that has been uploaded to the OBS bucket when **Obtaining and Uploading the Component Template Package**. 4. Click **OK**.

Step 5 Click **Next**.

Step 6 In the **Set Parameters** area, set **Name** to **image** and change its **Value** to the image package address obtained in **Obtaining the Download Address of the Uploaded Image Package**.

Step 7 Click **Create and Deploy** and wait until the component is deployed.

Step 8 Click **Overview** and confirm that **Status** is **Running**.

Step 9 Add the application domain name **www.demo.com** and the elastic IP address bound during ELB creation in **Preparing Resources** to the local **hosts** file.

Step 10 Open the CLI and run the following command to access the service provided by the component:

```
curl http://www.demo.com/hello
```

If the following information is displayed, the component has been automatically created using the component template:

```
Hello World
```

----End

9.4 Automatically Upgrading a Component Using a Component Template

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**. The application list is displayed.

Step 3 Click the application created in **Creating an Application** (for example, **template-demo-app**). The **Overview** page is displayed.

Step 4 Click the component created in **Automatically Creating a Component Using a Component Template**. The **Overview** page is displayed.

Step 5 Click **Upgrade**.

Step 6 Click **Next**.

Step 7 In the **Set Parameters** area, set **Name** to **value** and change its **Value** to **New World**.

Step 8 Click **Upgrade** and wait until the deployment is complete.

Step 9 Click **Overview** and confirm that **Status** is **Running**.

Step 10 Add the application domain name **www.demo.com** and the elastic IP address bound during ELB creation in **Preparing Resources** to the local **hosts** file.

Step 11 Open the CLI and run the following command to access the service provided by the component:

```
curl http://www.demo.com/hello
```

If the following information is displayed, the component has been automatically upgraded using the component template:

```
Hello New World
```

----End

9.5 Deleting a Component

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**. The application list is displayed.

Step 3 Click the application created in **Creating an Application** (for example, **template-demo-app**). The **Overview** page is displayed.

Step 4 In the **Component List** area, select the component created in **Automatically Creating a Component Using a Component Template**.

Step 5 In the **Operation** column, choose **More** > **Delete**.

Step 6 Select **All** to delete the component configuration file and resources used for component running when deleting the component.

Step 7 Click **OK**.

Wait until the component is deleted.

----End

10 Deploying Compressed Package Components Based on a VM

10.1 Compressed Package Overview

ServiceStage allows you to compress a Java or Tomcat application into a .zip or .tar.gz package for deploying a component on a VM. You can customize script installation, startup, and stop, configuration files, and health check for full lifecycle management from deployment to O&M.

This practice provides a ZIP component package based on the Java technology stack. The package directory structure is as follows:

```
demoJavaExample/
│
├── scripts/
│   │       ├── pre-install.sh
│   │       ├── pre-stop.sh
│   │       ├── stop.sh
│   │       ├── uninstall.sh
│   │       ├── install.sh
│   │       ├── start.sh
│   │       ├── check.sh
│   │       ├── post-start.sh
├── packages/
│   │       ├── weather-1.0.0.jar
├── config/
│   │       ├── system.cfg
├── appspec.yml
```

The prefix of the package name must be the same as the directory name of the decompressed file. For example, a package named **demoJavaExample.zip** will have a directory after decompression named **demoJavaExample**. For details about the directories and files in the package, see **Table 10-1**.

Table 10-1 Compressed package description

Directory or File	Description
scripts	(Mandatory) Stores script files executed in each application lifecycle.

Directory or File	Description
packages	(Mandatory) Stores application JAR or WAR packages.
config	(Mandatory) Stores application configuration information. ● Stores the **system.cfg** file for Java applications. ● Stores the **system.cfg**, **logging.properties**, and **server.xml** files for Tomcat applications.
appspec.yaml	(Mandatory) Records the lifecycle definition and also specifies information such as health check.

For details, see **How Do I Package a Java or Tomcat Application?**

This practice describes how to deploy components of the ZIP package type based on the Java technology stack on VMs, helping you quickly understand how to deploy components of the ZIP package type on VMs using custom scripts and configuration files.

10.2 Preparations

Preparing Resources

1. Create a VPC. For details, see **Creating a VPC**.
2. Create an ECS. For details, see **Purchasing an ECS in Custom Config Mode**.

   The VPC to which the ECS belongs is the one created in **1**.
3. Create a bucket for storing software packages. For details, see **Creating a Bucket**.

Obtaining and Uploading the Software Package

Step 1 Download the software package **demoJavaExample.zip**.

Step 2 Upload the software package obtained in **Step 1** to the OBS bucket prepared in **Preparing Resources**.

For details, see **Streaming Upload (PUT)**.

----End

Creating an Environment

Step 1 Log in to ServiceStage.

Step 2 Choose **Environment Management** > **Create Environment**. Then set required environment information by referring to the following table, and retain the default values for other parameters.

Parameter	Description
Environment	Enter an environment name, for example, **test-env**.
Enterprise Project	Specify **Enterprise Project**. Enterprise projects let you manage cloud resources and users by project. For details, see **Enabling the Enterprise Project Function**.
Environment Type	Select **VM**.
VPC	Select the VPC prepared in **Preparing Resources**. The VPC cannot be modified after the environment is created.
Configuration Mode	Select **Resource Management**.

Step 3 Click **Create Now**.

Step 4 Choose **ECSs** under **Compute**. Then click **Manage Resource**.

1. Select the ECS created in **Preparing Resources**.
2. Click **OK**.

Step 5 Install the VM agent on the managed ECS. For details, see **Installing a VM Agent**.

----End

Creating an Application

Step 1 Click < in the upper left corner to return to the **Environment Management** page.

Step 2 Choose **Application Management** > **Create Application** and configure the application by referring to the following table.

Parameter	Description
Name	Enter an application name, for example, **zip-demo**.
Enterprise Project	Specify **Enterprise Project**. Enterprise projects let you manage cloud resources and users by project. For details, see **Enabling the Enterprise Project Function**.

Step 3 Click **OK**.

----End

10.3 Deploying a Compressed Package Component

Step 1 Log in to ServiceStage.

Step 2 Choose **Application Management**. The application list is displayed.

Step 3 Choose **More** > **Create Component** in the **Operation** column of the application created in **Creating an Application**, for example, **zip-demo**.

Step 4 In the **Basic Information** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Component Name	Enter a component name, for example, **comp-zip**.
Component Version	Click **Generate**.
Application	Select the application created in **Creating an Application**, for example, **zip-demo**.
Environment	Select the environment created in **Creating an Environment**, for example, **test-env**.

Step 5 In the **Component Package** area, set the following mandatory parameters. Retain the default values for other parameters.

Parameter	Description
Stack	Select **Java**.
Software Package	Select **Compressed package**.
Upload Method	1. Select **OBS**. 2. Click **Software Package** and select **demoJavaExample.zip** uploaded in **Obtaining and Uploading the Software Package**. 3. Click **OK**.

Step 6 Click **Next**.

Step 7 In the **Resources** area, select the ECS that has been managed in **Creating an Environment**.

Step 8 Click **Create and Deploy** and wait until the component is deployed.

Step 9 Click **Overview** and confirm that **Status** is **Running**.

----End