Cloud Service Engine

Getting Started

Issue 01

Date 2024-02-29

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 Exclusive ServiceComb Engine

1.1 Creating a ServiceComb Engine

This section describes how to create a ServiceComb engine.

Prerequisites

- A ServiceComb engine runs on a VPC. Before creating a ServiceComb engine, ensure that VPCs and subnets are available.

  You have created a VPC. For details, see **Creating a VPC**.
- The login account has the permission to create a ServiceComb engine. For details about how to authorize and bind account permissions, see **Creating a User and Granting Permissions**.

Procedure

Step 1 Go to the **Buy Exclusive ServiceComb Engine** page.

Step 2 Set parameters according to the following table. Parameters marked with an asterisk (*) are mandatory.

Parameter	Description
* Billing Mode	Billing mode. Currently, pay-per-use is supported.
*Enterprise Project	Project where the microservice engine locates. You can search for and select an enterprise project in the drop-down list.
*Instances	Specifications of the microservice engine.
*Engine Type	Microservice engine type. If the engine type is cluster, the engine is deployed in cluster mode and supports host-level DR.
*Name	Enter a ServiceComb engine name, for example, cse-lhy-nodelete.
*AZ	Select an available zone (AZ).

Parameter	Description
*Network	Select the created VPC and subnet. You can search for and select a VPC and subnet from the drop-down list.
Description	Click ✎ and enter the engine description, for example, create a microservice engine.
Tags	Tags are used to identify cloud resources. When you have multiple cloud resources of the same type, you can use tags to classify them based on usage, owner, or environment. Click ⊕ **Add Tag**. In the **Add Tag** dialog box, enter a tag key and value. For details about tag naming rules, see **Managing Tags**. In the **Add Tag** dialog box, you can click ⊕ **Add Tag** to add multiple tags at a time, or click ⊖ next to a tag to delete the tag.
*Security Authentication	The exclusive ServiceComb engine with security authentication enabled provides the system management function using the role-based access control (RBAC) through the microservice engine console. ● Select **Enable security authentication**: 1. Determine whether to enable **Authenticate Programming Interface**. After it is enabled, you need to add the corresponding account and password to the microservice configuration file. Otherwise, the service cannot be registered with the engine. After it is disabled, you can register the service with the engine without configuring the account and password in the microservice configuration file, which improves the efficiency. You are advised to disable this function when accessing the service in a VPC. 2. Enter and confirm the password of user **root**. Keep the password secure. ● Select **Disable security authentication**: Disable security authentication. You can enable it after the instance is created.

Step 3 Click **Buy**. The page for confirming the engine information is displayed.

Step 4 Click **Submit** and wait until the engine is created.

NOTE

- It takes about 31 minutes to create an engine.
- After the microservice engine is created, its status is **Available**. For details about how to view the microservice engine status, see **Viewing ServiceComb Engine Information**.
- If the microservice engine fails to be created, view the failure cause on the **Operation** page and rectify the fault. Then, you can perform the following operations:
  - In the **Microservice Engine Information** area, click **Retry** to create a microservice engine again.
  - If the retry fails, delete the microservice engine that fails to be created. For details, see **Deleting an Exclusive ServiceComb Engine**.

----End

1.2 Connecting to ServiceComb Engines

1.2.1 Connecting Spring Cloud Applications to ServiceComb Engines Using SpringCloudHuawei SDK

This section uses a demo to demonstrate how to use ServiceComb engines.

NOTE

This section describes how to connect a provider service and a consumer service to a ServiceComb engine.

Prerequisites

- You have created a ServiceComb engine. For details, see **Creating a ServiceComb Engine**.
- You have downloaded the **demo source code** from GitHub to the local host and decompressed it.
- The Java JDK and Maven have been installed on the local host for compilation, building, and packaging, and the Maven central library can be accessed.

Procedure

Step 1 Logging In to the CSE Console

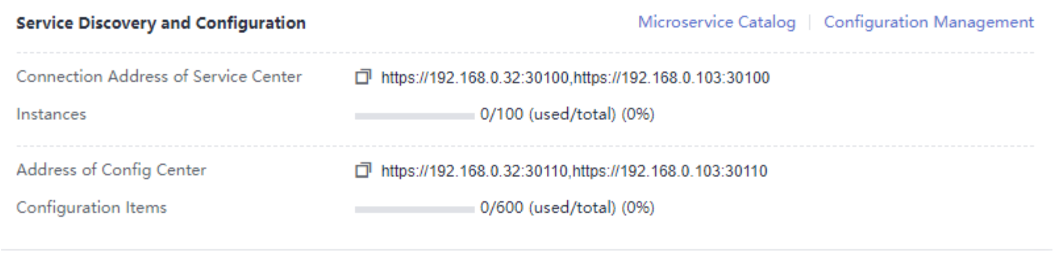
1. Log in to the **management console**.
2. Click and select a region.
3. Click in the upper left corner and select **Cloud Service Engine** in the service list. The CSE console is displayed.

Step 2 Choose **Exclusive ServiceComb Engine**.

Step 3 Click the ServiceComb engine created in **Prerequisites**.

Step 4 Obtain the service center address and configuration center address of the ServiceComb engine.

In the **Service Discovery and Configuration** area, you can view the service center address and configuration center address of the microservice engine.



Service Discovery and Configuration

Connection Address of Service Center	https://192.168.0.32:30100,https://192.168.0.103:30100
Instances	0/100 (used/total) (0%)
Address of Config Center	https://192.168.0.32:30110,https://192.168.0.103:30110
Configuration Items	0/600 (used/total) (0%)

Step 5 Change the addresses of the registry center and configuration center in the demo.

1. In the directory of the demo source code downloaded to the local host, find the **\basic\consumer\src\main\resources\bootstrap.yml** and **\basic\provider\src\main\resources\bootstrap.yml** files.
2. Add the service center address and configuration center address of the ServiceComb engine to the project configuration file (**\basic\consumer\src\main\resources\bootstrap.yml** is used as an example).

```
spring:
  application:
    name: basic-consumer
  cloud:
    servicecomb:
      discovery:
        enabled: true
        watch: false
        # Registry center address
        address: https://192.168.0.210:30100,https://192.168.0.246:30100
        appName: basic-application
        serviceName: ${spring.application.name}
        version: 0.0.1
        healthCheckInterval: 30
      config:
        # Configuration center address
        serverType: kie
        serverAddr: https://192.168.0.210:30110,https://192.168.0.246:30110
```

NOTE

In the ServiceStage deployment scenario, the service registry center and configuration center addresses are automatically injected during the deployment. You do not need to manually add them.

Step 6 Pack the demo source code into a JAR package.

1. In the root directory of the demo source code, open the Command Prompt and run the **mvn clean package** command to package and compile the project.
2. After the compilation is successful, two JAR packages are generated, as shown in **Table 1-1**.

Table 1-1 Software packages

Directory Where the Software Package Is Located	Package Name	Description
basic\consumer\target	basic-consumer-1.0-SNAPSHOT.jar	Service consumer
basic\provider\target	basic-provider-1.0-SNAPSHOT.jar	Service provider

Step 7 Deploy the application.

- Method 1: Deploy provider and consumer on the ECS node in the VPC where the ServiceComb engine is located.
  a. Create an ECS node in the VPC where the engine instance is located and log in to the ECS node. For details, see **Purchasing and Logging In to a Linux ECS**.
  b. Install JRE to provide a running environment for services.
  c. Upload the JAR package generated in **Step 6** to the ECS node.
  d. Run the **java -jar** *{JAR package}* command to run the generated JAR package.
- Method 2: Deploy provider and consumer on ServiceStage.
  a. Upload the JAR package generated in **Step 6** to OBS.
  b. Create a CCE cluster in the same VPC as the ServiceComb engine instance. For details, see **Creating a Kubernetes Cluster**.
  c. Create a ServiceStage environment in the VPC where the engine instance is located, and manage the ServiceComb engine and CCE resource. For details, see **Creating an Environment**.
  d. Deploy provider and consumer. For details, see **Creating and Deploying a Component**.

Step 8 Confirm the deployment results.

1. **Optional:** On the CSE console, choose **Exclusive ServiceComb Engine** and click the ServiceComb engine created in **Prerequisites**.
2. Choose **Microservice Catalog** > **Microservice List** to view the number of **basic-consumer** and **basic-provider** microservice instances.
   – If **Instances** is not **0**, the demo has been connected to the microservice engine.
   – If **Instances** is **0** or the **basic-consumer** and **basic-provider** services cannot be found, the demo fails to be connected to the microservice engine.

----End

2 Registry/Configuration Center

2.1 Creating a Nacos Engine

This section describes how to create a Nacos engine.

NOTE

Nacos engines are supported only in CN-Hong Kong and AP-Singapore.

Prerequisites

A Nacos engine runs on a VPC. Before creating a Nacos engine, ensure that VPCs and subnets are available.

You have created a VPC. For details, see **Creating a VPC**.

Procedure

Step 1 Go to **Buy Registry/Configuration Center Instance**.

Step 2 In the left navigation pane, choose **Registry/Configuration Center**.

Step 3 Click **Buy Registry/Configuration Center Instance**.

Step 4 Set parameters according to the following table. Parameters marked with an asterisk (*) are mandatory.

Parameter	Description
* Billing Mode	Billing mode. Currently, pay-per-use is supported.
*Enterprise Project	Project where the Nacos engine locates. You can search for and select an enterprise project in the drop-down list.
*Name	Enter the name of the Nacos engine.

Parameter	Description
*Registry/Configuration Center Instance	Select **Nacos**. **NOTE** By default, the Nacos engine is deployed in multiple AZs on three nodes to provide AZ-level DR.
*Instances	Select the required capacity specifications.
Version	Only the latest version can be created.
*Network	Select the created VPC and subnet. You can search for and select a VPC and subnet from the drop-down list. A VPC enables you to provision logically isolated, configurable, and manageable virtual networks for your engine.
Tags	Tags are used to identify cloud resources. When you have multiple cloud resources of the same type, you can use tags to classify them based on usage, owner, or environment. Click ⊕ **Add Tag**. In the **Add Tag** dialog box, enter a tag key and value. For details about tag naming rules, see **Managing Tags**. In the **Add Tag** dialog box, you can click ⊕ **Add Tag** to add multiple tags at a time, or click ⊖ next to a tag to delete the tag.

Step 5 Click **Buy Now**. When the status becomes **Available**, the engine is created.

----End

2.2 Connecting to Nacos Engines

2.2.1 Connecting Spring Cloud Applications to Nacos Engines

This section uses a demo to demonstrate how to connect microservice applications to Nacos engines.

This section describes how to connect a provider service and a consumer service to a Nacos engine.

NOTE

Nacos engines are supported only in CN-Hong Kong and AP-Singapore.

Prerequisites

- You have created a Nacos engine. For details, see **Creating a Nacos Engine**.
- You have downloaded the **demo source code** from GitHub to the local host and decompressed it.
- The Java JDK and Maven have been installed on the local host for compilation, building, and packaging, and the Maven central library can be accessed.

Procedure

Step 1 Log in to CSE.

1. Log in to the **management console**.
2. Click and select a region.
3. Click in the upper left corner and select **Cloud Service Engine** in the service list. The CSE console is displayed.

Step 2 Obtain the service center address of the exclusive Nacos engine.

1. In the left navigation pane, choose **Registry/Configuration Center** and click the Nacos engine instance.
2. In the **Connection Information** area on the **Basic Information** page, obtain the service center address.

Connection Information

Private IP	[redacted]	Private Port	8848,9848
Virtual Private Cloud	vpc-[redacted]	Subnet	lbcdzw

Step 3 Change the configuration center address, service center address, and microservice name in the demo.

1. Configure the Nacos configuration center in **bootstrap.properties**.

   Find the **nacos-examples-master\nacos-spring-cloud-example\nacos-spring-cloud-discovery-example\nacos-spring-cloud-consumer-example\src\main\resources** file in the demo source code directory downloaded to the local host, add the **bootstrap.properties** file, and configure the Naocs configuration center.

```
spring.cloud.nacos.config.server-addr= XXX.nacos.cse.com:8848 //Address of the Nacos configuration center
spring.cloud.nacos.config.prefix= example //Prefix of the configuration file name
spring.cloud.nacos.config.file-extension= properties //Extension of the configuration file name
spring.cloud.nacos.config.group= XXX //Group to which the configuration file belongs. If this parameter is left blank, the default value DEFAULT_GROUP  is used.
spring.cloud.nacos.config.namespace= XXX  //ID of the namespace to which the configuration file belongs. If this parameter is left blank, the default value public is used.
```

2. Configure the Nacos service center address and microservice name in the **application.properties** file.

   – Find the **nacos-examples-master\nacos-spring-cloud-example\nacos-spring-cloud-discovery-example\nacos-spring-cloud-consumer-example\src\main\resources\application.properties** file in the demo source code directory downloaded to the local host and configure the consumer service.

```
server.port=8080
spring.application.name= service-consumer //Microservice name
spring.cloud.nacos.discovery.server-addr= XXX.nacos.cse.com:8848 //Nacos service center address
spring.cloud.nacos.discovery.group= XXX  //Group to which the microservice belongs. If this parameter is left blank, the default value DEFAULT_GROUP is used.
spring.cloud.nacos.discovery.namespace= XXX  //ID of the namespace to which the microservice belongs. If this parameter is left blank, the default value public is used.
spring.cloud.nacos.discovery.cluster-name= XXX  //Name of the cluster to which the microservice belongs. If this parameter is left blank, the default value DEFAULT is used.
```

   – Find the **nacos-examples-master\nacos-spring-cloud-example\nacos-spring-cloud-discovery-example\nacos-spring-cloud-provider-example**

\src\main\resources\application.properties file in the demo source code directory downloaded to the local host and configure the provider service.

```
server.port=8070
spring.application.name= service-provider //Microservice name
spring.cloud.nacos.discovery.server-addr= XXX.nacos.cse.com:8848 //Nacos service center address
spring.cloud.nacos.discovery.group= XXX  //Group to which the microservice belongs. If this parameter is left blank, the default value DEFAULT_GROUP is used.
spring.cloud.nacos.discovery.namespace= XXX  //ID of the namespace to which the microservice belongs. If this parameter is left blank, the default value public is used.
spring.cloud.nacos.discovery.cluster-name= XXX  //Name of the cluster to which the microservice belongs. If this parameter is left blank, the default value DEFAULT is used.
```

Step 4 Pack the demo source code into a JAR package.

1. In the root directory of the demo source code, open the Command Prompt and run the **mvn clean package** command to package and compile the project.
2. After the compilation is successful, two JAR packages are generated, as shown in **Table 2-1**.

Table 2-1 Software packages

Directory Where the Software Package Is Located	Package Name	Description
\nacos-spring-cloud-consumer-example\target	nacos-spring-cloud-consumer-example-0.2.0-SNAPSHOT.jar	Service consumer
\nacos-spring-cloud-provider-example\target	nacos-spring-cloud-provider-example-0.2.0-SNAPSHOT.jar	Service provider

Step 5 Deploy the Spring Cloud application.

Deploy provider and consumer on the ECS node in the VPC where the Nacos engine is located.

1. Create an ECS node in the VPC where the engine instance is located and log in to the ECS node. For details, see **Purchasing and Logging In to a Linux ECS**.
2. Install JRE to provide a running environment for services.
3. Upload the JAR package generated in **Step 4** to the ECS node.
4. Run the **java -jar** *{JAR package}* command to run the generated JAR package.

Step 6 Confirm the deployment results.

1. **Optional:** On the CSE console, choose **Registry/Configuration Center** and click the Nacos engine created in **Prerequisites**.
2. Choose **Service Management** and check the number of instances of microservices **service-consumer** and **service-provider**.
   – If **Instances** is not **0**, the demo has been connected to the Nacos engine.
   – If **Instances** is **0** or the **service-consumer** and **service-provider** services cannot be found, the demo fails to be connected to the Nacos engine.

----End

2.2.2 Connecting Spring Cloud Eureka Applications to Nacos Engines

This section uses a demo to demonstrate how to connect Spring Cloud Eureka applications to Nacos engines.

This section describes how to connect a provider service and a consumer service to a Nacos engine.

NOTE

Nacos engines are supported only in CN-Hong Kong and AP-Singapore.

Prerequisites

- You have created a Nacos engine. For details, see **Creating a Nacos Engine**.
- You have downloaded the **demo source code** from GitHub to the local host and decompressed it.
- The Java JDK and Maven have been installed on the local host for compilation, building, and packaging, and the Maven central library can be accessed.

Constraints

- Nacos is compatible with Eureka APIs on the Eureka server side and saves and updates client instance information registered on the service side. Therefore, if you use only Eureka as the registry center, many features of Nacos, such as namespace and configuration management, cannot be used.
- Eureka is used as the client and can be viewed only on the **Service Management** page of Nacos. Eureka services are displayed using the default attributes of Nacos.
  - Default namespace: **public**
  - Default group: **DEFAULT_GROUP**
- The value of **Protection Threshold** for creating a service on the **Service Management** page of Nacos is a feature of Nacos and does not apply to the Eureka service.

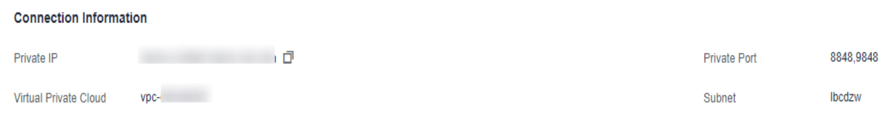
Procedure

Step 1 Log in to CSE.

1. Log in to the **management console**.
2. Click and select a region.
3. Click in the upper left corner and select **Cloud Service Engine** in the service list. The CSE console is displayed.

Step 2 Obtain the service center address of the exclusive Nacos engine.

1. In the left navigation pane, choose **Registry/Configuration Center** and click the Nacos engine instance.
2. In the **Connection Information** area on the **Basic Information** page, obtain the service center address.


Connection Information

Private IP	[redacted]	Private Port	8848,9848
Virtual Private Cloud	vpc-[redacted]	Subnet	lbcdzw

Step 3 Change the registry center address and microservice name in the demo.

1. Configure the Nacos service center address and microservice name in the **application.properties** file.
   - Find the **eureka-demo-master\eureka-consumer\src\main\resources\application.properties** file in the demo source code directory downloaded to the local host and configure the consumer service.
     ```
     server.port=9001
     spring.application.name= eureka-client-consumer //Microservice name
     eureka.client.serviceUrl.defaultZone= XXX.nacos.cse.com:8848/nacos/eureka //Service center address of Eureka
     eureka.instance.lease-renewal-interval-in-seconds=15 //Service heartbeat update interval
     eureka.client.registry-fetch-interval-seconds=15 //Interval for pulling the registry center. It is recommended that the value be the same as the heartbeat interval.
     ```
   - Find the **eureka-demo-master\eureka-provider\src\main\resources\application.properties** file in the demo source code directory downloaded to the local host and configure the provider service.
     ```
     server.port=9000
     spring.application.name= eureka-client-provider //Microservice name
     eureka.client.serviceUrl.defaultZone= XXX.nacos.cse.com:8848/nacos/eureka //Service center address of Eureka
     eureka.instance.lease-renewal-interval-in-seconds=15 //Service heartbeat update interval
     eureka.client.registry-fetch-interval-seconds=15 //Interval for pulling the registry center. It is recommended that the value be the same as the heartbeat interval.
     ```

Step 4 Pack the demo source code into a JAR package.

1. In the root directory of the demo source code, open the Command Prompt and run the **mvn clean package** command to package and compile the project.
2. After the compilation is successful, two JAR packages are generated, as shown in **Table 2-2**.

Table 2-2 Software packages

Directory Where the Software Package Is Located	Package Name	Description
\eureka-consumer\target	eureka-client-consumer-1.0.0-SNAPSHOT.jar	Service consumer
\eureka-provider\target	eureka-client-provider-1.0.0-SNAPSHOT.jar	Service provider

Step 5 Deploy the Spring Cloud application.

Deploy provider and consumer on the ECS node in the VPC where the Nacos engine is located.

1. Create an ECS node in the VPC where the engine instance is located and log in to the ECS node. For details, see **Purchasing and Logging In to a Linux ECS**.

2. Install JRE to provide a running environment for services.
3. Upload the JAR package generated in **Step 4** to the ECS node.
4. Run the **java -jar** *{JAR package}* command to run the generated JAR package.

Step 6 Confirm the deployment results.

1. **Optional:** On the CSE console, choose **Registry/Configuration Center** and click the Nacos engine created in **Prerequisites**.
2. Choose **Service Management** and check the number of instances of microservices **eureka-client-consumer** and **eureka-client-provider**.
   - If **Instances** is not **0**, the demo has been connected to the Nacos engine.
   - If **Instances** is **0** or the **eureka-client-consumer** and **eureka-client-provider** services cannot be found, the demo fails to be connected to the Nacos engine.

----End