Cloud Service Engine

Getting Started

Issue 01

Date 2024-09-26

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 Connecting Spring Cloud Applications to ServiceComb Engines Using Spring Cloud Huawei SDK

This section uses a demo to demonstrate how to connect a Spring Cloud application to a ServiceComb engine Using Spring Cloud Huawei SDK.

1. **Preparations**

   You need to register with Huawei Cloud and complete real-name authentication, top up your account, grant permissions, create a VPC and subnet, obtain the demo package, and prepare the local host for compilation, building, and packaging.
2. **Creating a ServiceComb Engine**

   You can select engine specifications, AZs, and networks when creating an engine.
3. **Connecting Spring Cloud Applications to ServiceComb Engines**

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

Preparations

1. Sign up for a HUAWEI ID and complete real-name authentication.

   Skip this step if you already have a HUAWEI ID, or perform the following steps to create one.

   a. Log in to the **Huawei Cloud official website** and click **Sign Up**.

   b. Sign up for a HUAWEI ID. For details, see **Signing up for a HUAWEI ID and Enabling Huawei Cloud Services**.

      After your ID is created, the system redirects you to your personal information page.

   c. Complete real-name authentication. For details, see **Individual Real-Name Authentication**.

      NOTE

      Real-name authentication is required only when you buy or use resources in the Chinese mainland.

2. Top up your account.

   Ensure that your account has sufficient balance.

   – For details about ServiceComb engine prices, see **CSE Pricing Details**.
   – For details about top-up, see **Top-Up and Repayment**.

3. Grant permissions.

   You must have the required permissions to create dependent resources and ServiceComb engines. For details, see **Creating a User and Granting Permissions**.

4. Create a VPC and subnet.

   ServiceComb engines run in a VPC and need to be bound to a subnet. You must have a VPC and subnet to create ServiceComb engines. For details, see **Creating a VPC**. Skip this step if you already have a VPC and subnet.

5. 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.

6. Download the **demo source code** from GitHub to the local host and decompress it.

   NOTE

   Spring Cloud Huawei has been integrated in the configuration file of this demo. For details, see **Connecting Spring Cloud Applications to ServiceComb Engines**.

Creating a ServiceComb Engine

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. For details, see **Creating a ServiceComb Engine**.

Table 1-1 Creating a ServiceComb engine

Parameter	Description
*Billing Mode	Billing mode. Currently, **Pay-per-use** is supported. Pay-per-use is postpaid. You use ServiceComb engines and then pay as billed for your usage duration.
*Enterprise Project	Select the enterprise project where the ServiceComb engine locates. Enterprise projects let you manage cloud resources and users by project. **default** is selected by default.
*Instances	Select the engine specifications. In this example, select 100 microservice instances.
*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. The name contains 3 to 64 characters, including letters, digits, and hyphens (-), and starts with a letter but cannot end with a hyphen (-). For example, **cse-test**.

Parameter	Description
*AZ	Select one or three AZs. In this example, select one AZ to provide host-level DR.
*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 ServiceComb engine.
*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. In this example, 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 a microservice 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

Connecting Spring Cloud Applications to ServiceComb Engines

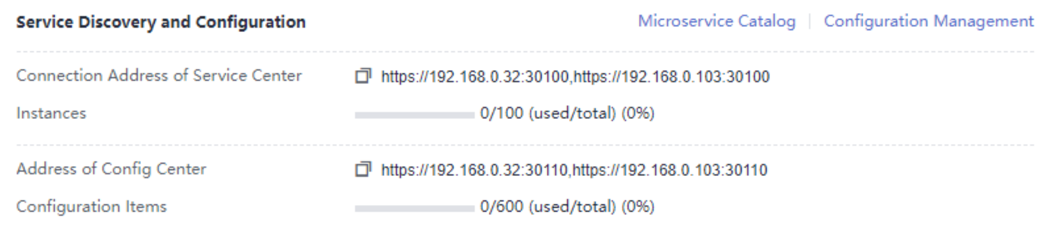
Step 1 Log in to **CSE**.

Step 2 Choose **Exclusive ServiceComb Engine**.

Step 3 Click the ServiceComb engine created in **Creating a ServiceComb Engine**.

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
Microservice Catalog | Configuration Management

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 1-2**.

Table 1-2 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.

Deploy provider and consumer on ServiceStage.

1. Upload the JAR package generated in **Step 6** to OBS.
2. Create a CCE cluster in the same VPC as the ServiceComb engine instance. For details, see **Creating a Kubernetes Cluster**.

3. 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**.
4. 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 **Creating a ServiceComb Engine**.
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 ServiceComb 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 Connecting Spring Cloud Applications to Nacos Engines Using Spring Cloud SDK

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

1. **Preparations**

   You need to register with Huawei Cloud and complete real-name authentication, top up your account, grant permissions, create a VPC and subnet, obtain the demo package, and prepare the local host for compilation, building, and packaging.
2. **Creating a Registry/Configuration Center**

   You can select engine specifications, AZs, and networks when creating an engine.
3. **Connecting Spring Cloud Applications to Nacos Engines**

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

Preparations

1. Sign up for a HUAWEI ID and complete real-name authentication.

   Skip this step if you already have a HUAWEI ID, or perform the following steps to create one.

   a. Log in to the **Huawei Cloud official website** and click **Sign Up**.

   b. Sign up for a HUAWEI ID. For details, see **Signing up for a HUAWEI ID and Enabling Huawei Cloud Services**.

      After your ID is created, the system redirects you to your personal information page.

   c. Complete real-name authentication. For details, see **Individual Real-Name Authentication**.

      NOTE

      Real-name authentication is required only when you buy or use resources in the Chinese mainland.
2. Top up your account.

Ensure that your account has sufficient balance.

- For details about registry/configuration center prices, see **CSE Pricing Details**.
- For details about top-up, see **Top-Up and Repayment**.

3. Grant permissions.

   You must have the required permissions to create dependent resources and Nacos engines. For details, see **Creating a User and Granting Permissions**.

   NOTE

   To create a registry/configuration center, you must have the CSE FullAccess and DNS FullAccess permissions.

4. Create a VPC and subnet.

   Nacos engines run in a VPC and need to be bound to a subnet. You must have a VPC and subnet to create Nacos engines. For details, see **Creating a VPC**. Skip this step if you already have a VPC and subnet.

5. 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.

6. Download the **demo source code** from GitHub to the local host and decompress it.

Creating a Registry/Configuration Center

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

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. For details, see **Creating a Registry/Configuration Center**.

Table 2-1 Creating a registry/configuration center

Parameter	Description
*Billing Mode	Billing mode. Currently, **Pay-per-use** is supported. Pay-per-use is postpaid. You use registry/configuration centers and then pay as billed for your usage duration.
*Enterprise Project	Select the enterprise project where the Nacos engine is located. Enterprise projects let you manage cloud resources and users by project. **default** is selected by default.
*Name	Enter a Nacos engine name. The name contains 3 to 64 characters, including letters, digits, and hyphens (-), and starts with a letter but cannot end with a hyphen (-). For example, **nacos-test**.

Parameter	Description
*Registry/Configuration Center Instance	The registry/configuration center supports Nacos engines. **NOTE** Cluster nodes in the registry/configuration center are evenly distributed to different AZs. A failure of a single node does not affect external services. The registry/configuration center does not support AZ-level DR but provides host-level DR.
*Instances	Select the required capacity specifications. In this example, the number of instances is 500, and the number of capacity units is 10.
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.

Step 5 Click **Buy Now**. The registry/configuration center starts to be created. When the status is **Available**, the registry/configuration center is created.

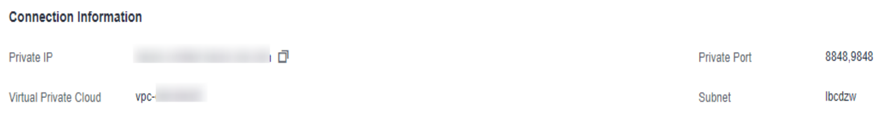
----End

Connecting Spring Cloud Applications to Nacos Engines

Step 1 Log in to **CSE**.

Step 2 Obtain the registry and discovery address of a 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 Nacos 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 2-2**.

Table 2-2 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 Spring Cloud applications.

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 **Creating a Registry/Configuration Center**.
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

3 Getting Started with Common Practices

This section describes common CSE practices.

Practice	Description
Hosting a Java Chassis Application	Spring Boot and Spring Cloud are widely used to build microservice applications. The main purpose of using ServiceComb engines to host Spring Cloud applications is to replace open-source components with highly reliable commercial middleware to better manage and maintain the application system. The reconstruction process should minimize the impact on service logic.

Practice	Description
Hosting a Java Chassis Application	**Java chassis** is an open-source microservice development framework managed by the Apache Software Foundation. It was first donated by CSE. Till now, hundreds of developers have contributed to the project. It has the following functions: ● Flexible and high-performance RPC implementation. Based on open APIs, Java chassis provides unified description of different RPC development modes, standardizing microservice API management and retaining flexible usage habits of developers. Based on Reactive, Java chassis implements efficient communication protocols such as REST and Highway, and is compatible with traditional communication protocols such as Servlet. ● Rich service governance capabilities and unified governance responsibility chain. Common microservice governance capabilities, such as load balancing, rate limiting, and fault isolation, can be used out of the box. In addition, a unified governance responsibility chain is provided to simplify the development of new governance functions.