ServiceStage

Development Guide

Issue	01
Date	2021-04-07

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

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

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

Contents

1 Java Chassis

1.1 Overview

Apache ServiceComb Java chassis can directly access the microservice engine without modifying the code and use the service governance function provided by the microservice engine. For details about how to use Java chassis to develop the first microservice, see **Developing and Deploying a ServiceComb Java Microservice**

NOTE

For details about getting started, design selection, application development, and security of ServiceComb Java chassis microservices, see **ServiceComb Java Chassis Developers Guide**.

1.2 Developer Guide

1.2.1 Accessing Applications to a Microservice Engine

For **ServiceComb Java chassis** 2.1.3 and later versions, you can directly use the open-source software package to access the microservice engine. If ServiceComb Java chassis earlier than 2.1.3 accesses a professional microservice engine, you need to import the **foundation-auth** package.

Prerequisite

- You have obtained the AK/SK. For details, see **Obtaining the AK/SK and Project Name**.
- You have obtained the project name. For details about how to obtain the project name, see **Obtaining the AK/SK and Project Name**.

NOTE

The AK/SK or project name can be deleted or modified. If the AK/SK or project name is changed, the microservice needs to be upgraded. Otherwise, service registration and discovery may fail. For example, if the project name is changed, the "Project ID or name does not exist" error may be generated. AK/SK information needs to be configured for only professional engines.

Access Guidance

- 2.1.3 and later

To access a microservice engine, ensure that the following dependencies are introduced to the project:

```
<dependency>
   <groupId>org.apache.servicecomb</groupId>
   <artifactId>solution-basic</artifactId>
</dependency>
<dependency>
   <groupId>org.apache.servicecomb</groupId>
   <artifactId>servicestage-environment</artifactId>
</dependency>
```

Alternatively, ensure that the following dependencies are introduced to the project:

```
<dependency>
   <groupId>org.apache.servicecomb</groupId>
   <artifactId>servicestage</artifactId>
</dependency>
<dependency>
   <groupId>org.apache.servicecomb</groupId>
   <artifactId>servicestage-environment</artifactId>
</dependency>
```

The **servicestage-environment** software package is optional. This software package provides the environment variable mapping function. After this software package is used, you can use ServiceStage to deploy applications without modifying the application name, microservice name, microservice version, registration center address, configuration center address, and project name. The default configurations in the **microservice.yaml** file are overwritten by environment variables, containing the following **mapping.yaml** file:

```
CAS_APPLICATION_NAME:
  - servicecomb.service.application
CAS_COMPONENT_NAME:
  - servicecomb.service.name
CAS_INSTANCE_VERSION:
  - servicecomb.service.version
PAAS_CSE_ENDPOINT:
  - servicecomb.service.registry.address
  - servicecomb.config.client.serverUri
PAAS_CSE_SC_ENDPOINT:
  - servicecomb.service.registry.address
PAAS_CSE_CC_ENDPOINT:
  - servicecomb.config.client.serverUri
PAAS_PROJECT_NAME:
  - servicecomb.credentials.project
```

- Versions earlier than 2.1.3

To access a microservice engine, ensure that the following dependencies are introduced to the project:

```
<dependency>
    <groupId>com.huawei.paas.cse</groupId>
    <artifactId>foundation-auth</artifactId>
</dependency>
```

Table 1-1 shows the relationship between the software package version Java chassis version.

Table 1-1 Version mapping between Java chassis and **foundation-auth** package

Java Chassis Version	foundation-auth Version
2.1.0~2.1.2	3.1.0
2.0.1~2.0.2	3.0.0
1.3.0~1.3.1	2.5.0

To use the **foundation-auth** software package, configure the following Mavan repository:

```
<repositories>
    <repository>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
        <id>huaweicloudsdk-releases</id>
        <name>huaweicloudsdk</name>
        <url>https://repo.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
    </repository>
</repositories>
```

1.2.2 Using AK/SK Authentication

Concept

The professional microservice engine requires AK/SK authentication. This topic describes how to configure AK/SK.

Example Configuration

AK/SK authentication depends on the AK/SK configured by users. ServiceComb provides plaintext configuration by default and allows users to customize the encryption storage scheme. The customized encryption storage requires version 2.1.3 or later.

1. Add the following configuration in plaintext to the **microservice.yaml** file:

```
servicecomb:
  credentials:
    accessKey: yourak
    secretKey: yoursk
    project: yourprojectname
    akskCustomCipher: default
```


2. Implement API **org.apache.servicecomb.foundation.auth.Cipher**, which contains two methods:
   - String name()

     Name definition of **servicecomb.credentials.akskCustomCipher**, which needs to be added to the configuration file.
   - char[] decode(char[] encrypted)

     Decrypt the API, which is used after secretKey is decrypted.

   The implementation class must be declared as **Bean**. For example:

   ```
   @Component
   public class MyCipher implements Cipher
   ```

   Add the following configuration to the **microservice.yaml** file:

   ```
   servicecomb:
     credentials:
       accessKey: yourak
       secretKey: yoursk  #SK information in ciphertext.
       project: yourprojectname
       akskCustomCipher: youciphername  #returned name of the name() method in the
   added class.
   ```

1.2.3 Using RBAC Authentication

Concept

An exclusive microservice engine provides two authentication modes: no authentication and RBAC authentication. To use RBAC authentication, ServiceComb Java chassis must be of version 2.1.3 or later. Earlier versions do not support RBAC authentication.

After RBAC authentication is enabled for a microservice engine, all called APIs can be called only after a token is obtained. For details about the authentication process, see **RBAC**.

Example Configuration

To use RBAC, obtain the username and password from the microservice engine and then add the following configuration to the configuration file:

```
servicecomb:
  credentials:
    rbac.enabled: true
    acount:
      name: youracountname # Username obtained from the microservice engine.
      password: yourpassword # Password obtained from the microservice engine.
    cipher: default # Returned name of the name() method in the implementation class.
```

cipher specifies the name of the algorithm used to encrypt the **password**. By default, the password is stored in plaintext. Similar to the encryption scheme of AK/SK authentication, API **org.apache.servicecomb.foundation.auth.Cipher** can be used to encrypt passwords for storage.

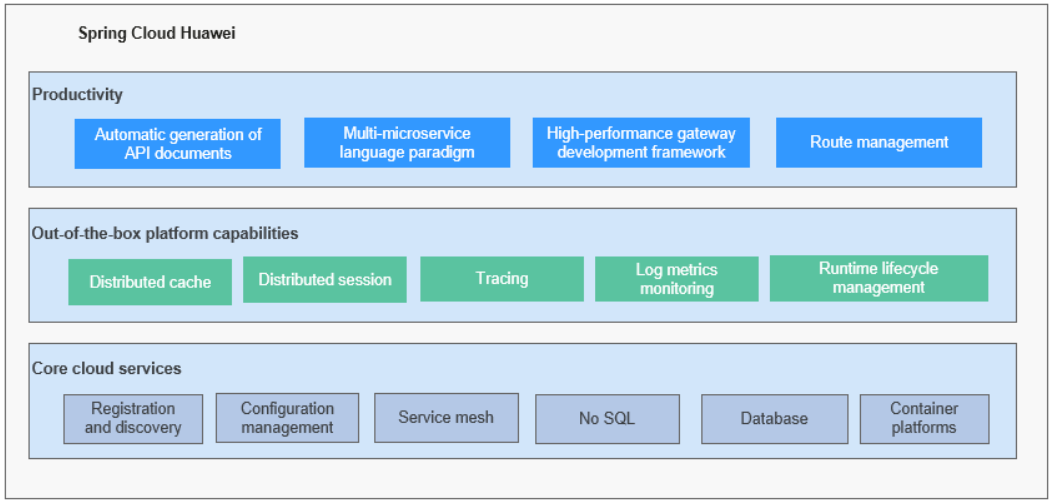
2 Spring Cloud Huawei

2.1 Overview

You can easily connect Spring Cloud applications to microservice engines without modifying code.

The advantages of accessing ServiceComb are as follows:

1. Enable you to focus on the development of service systems so that you do not pay attention to middleware reliability evaluation and cluster deployment.
2. Ensure quick service delivery and agile development. You can use ServiceStage to dynamically adjust resource usage based on the service scale to reduce service risks.
3. Route management: Traffic can be distributed to different microservice instances based on configurations to implement dark launch.
4. Service mesh: PHP, Python, Go, and .NET Core applications can be connected to the registry center through service mesh and achieve unified governance together with Spring Cloud applications.



This section describes the principle and process of accessing Spring Cloud applications to microservice engines.

spring-cloud-huawei is used for access. The starter mechanism of Spring Boot is used for Spring Cloud extension. To connect your applications to the cloud, you only need to import corresponding starter dependencies and make simple configurations, but do not need to change any code. **spring-cloud-huawei** is released in the Maven central repository. You can log in to the GitHub website to view the **latest release version**.

Version is introduced. You are advised to use dependencyManagement to manage dependencies in a unified manner.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.huaweicloud</groupId>
      <artifactId>spring-cloud-huawei-dependencies</artifactId>
      <version>${project.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Version mapping:

Spring Cloud Huawei supports all Spring Cloud series.

The Spring Cloud official website does not maintain the Edgware and Finchley versions, and the corresponding Spring Cloud Huawei does not maintain these two versions. In this case, only basic service access is provided. Bug fixing and security issue resolving will not be supported.

Spring Cloud Hoxton (preferred) and Spring Cloud Huawei mapping Spring Cloud Greenwich are recommended. In addition, new features are preferentially updated in the corresponding Spring Cloud Hoxton version. For Spring Cloud Greenwich, only basic bug fixing and security issue resolving are provided.

2.2 Using the Distributed Registry Center

Spring Cloud Extension

Service Provider Interface (SPI) is a built-in JDK mechanism for service discovery and can be regarded as a type of specifications. You only need to define APIs in specified directories and import standard JAR packages. This method is suitable for third-party developers to implement extension.

However, SPI itself has some problems. For example, during JDK SPI implementation, all extension points will be instantiated at a time, which increases the application startup time and wastes resources.

Spring uses the SPI mechanism for extension. Different from other frameworks, Spring emphasizes openness or scalability. In the Spring framework, many components are integrated from external systems. Various starters are a collection of such components. For example, you can use Consul, Eureka, ZooKeeper, and ETCD for registry and discovery.

Spring Cloud depends on Spring Boot, and Spring Boot depends on Spring. Spring Cloud scalability comes from **@EnableAutoConfiguration** in Spring Boot. The loading procedure is as follows:

1. During the startup of Spring Boot, scan all **META-INF/spring.factories** files in JAR packages in the class path.
2. Read the specified configuration and automatically create a bean based on the condition in the configuration.
3. Put the bean into Spring Context. In this way, it is injected to the container and can be used anywhere.

Spring Cloud Huawei provides multi-registry centers.

Getting Started

spring-cloud-huawei supports service registry and discovery in Spring Cloud mode. You only need to modify some configuration files. The procedure is as follows:

Step 1 Add the following dependencies.

```
<dependencies>
  <dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>spring-cloud-starter-huawei-servicecomb-discovery</artifactId>
  </dependency>
</dependencies>
```

Step 2 Create a project or module and define a configuration file.

Define configurations by setting the **application.yml** file.

```
server:
  port: 8080
spring:
  application:
    name: price
```



```
cloud:
  servicecomb:
    discovery:
      enabled: true
      address: https://cse.cn-xxxx-2.myhuaweicloud.com
      serviceName: price
      version: 0.0.1
      healthCheckInterval: 30
    credentials:
      enabled: true
      accessKey: your ak
      secretKey: your sk
      akskCustomCipher: default
      project: cn-xxxx-2
```

Step 3 Add annotations for start classes. Example:

NOTE

Note: In the case of Eureka access, if the **@EnableEurekaClient** annotation is used, delete it.

```
@SpringBootApplication
@EnableDiscoveryClient
public class Application {
  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }
}
```

----End

Running the Demo

The following shows how to implement service registry and discovery based on a project demo. **spring-cloud-huawei-sample/discovery-demo** provides three projects and is used to simulate the scenario where Spring Cloud projects are migrated to the registry center of Apache ServiceComb.

- **price-provider**: Price service for service providers; RESTful API.
- **order-consumer**: Order service for service consumers; RESTful API; invoking **price-provider** to obtain price information through RestTemplate.
- **product-consumer-feign**: Product service for service consumers; RESTful API; invoking **price-provider** to obtain price information through Feign.

Run the demo according to the following procedure:

Step 1 Download **the local lightweight microservice engine**, decompress it, and run **start.bat**. Then you can access the following address to go to the console.

http://localhost:30106

Step 2 Respectively run the main function to start the three demo services.

1. Configure the actuator in the **application.properties**.

   ```
   management.endpoints.web.exposure.include=*
   management.endpoint.health.show-details=always
   management.security.enabled=false
   ```

2. Check the service running status through the actuator.

   http://127.0.0.1:8080/actuator/service-registry

If the service is normal, **UP** is displayed.

3. To change the service status to **DOWN**, run the following command:
   ```
   curl -i -H "Content-Type: application/json" -X POST -d '{"status":"DOWN"}' http://localhost:8080/actuator/service-registry
   ```

Step 3 Access the service registry center and check the instance running status. The address is as follows:

http://127.0.0.1:30103

Step 4 Use the curl, browser, or Postman to verify the invocation.

Address of **product-consumer-feign**:

http://127.0.0.1:8089/product?id=11

Address of **order-consumer**:

http://127.0.0.1:8088/order?id=11

----End

Configuration Items

The following table lists the configuration items in the **application.yaml** file.

Configuration Item	Key	Default Value
Enabling ServiceComb service discovery	spring.cloud.servicecomb.discovery.enabled	true
Enabling the WebSocket watch mechanism	spring.cloud.servicecomb.discovery.watch	true (Set this parameter to false for professional engines.)
Registry center address. You can use commas (,) to separate multiple addresses.	spring.cloud.servicecomb.discovery.address	-
Service name	spring.cloud.servicecomb.discovery.serviceName	If no service name exists, use the Spring Cloud application name.
Application name	spring.cloud.servicecomb.discovery.appName	default
Environment name	server.env	""
Version	spring.cloud.servicecomb.discovery.version	-

Configuration Item	Key	Default Value
Enabling health check	spring.cloud.servicecomb.discovery.healthCheck	true
Health check interval	spring.cloud.servicecomb.discovery.healthCheckInterval	10s
DNS-resolved IP address (avoiding repeated DNS resolution)	spring.cloud.servicecomb.discovery.autoDiscovery	false
Cross-app invoking	spring.cloud.servicecomb.discovery.allowCrossApp	false
Name (AZ affinity. After this parameter is configured, instances in the same AZ are preferentially selected.)	spring.cloud.servicecomb.discovery.datacenter.name	-
Region (AZ affinity. After this parameter is configured, instances in the same AZ are preferentially selected.)	spring.cloud.servicecomb.discovery.datacenter.region	-
AZ (AZ affinity. After this parameter is configured, instances in the same AZ are preferentially selected.)	spring.cloud.servicecomb.discovery.datacenter.availableZone	-

To interconnect with the professional engine, you need to configure the AK/SK for IAM authentication. The AK/SK configuration items in the **application.yaml** file are described as follows:

Configuration Item	Key	Default Value
Authentication enablement	spring.cloud.servicecomb.credentials.enabled	false
Tenant AK	spring.cloud.servicecomb.credentials.accessKey	-
Tenant SK	spring.cloud.servicecomb.credentials.secretKey	-
Cipher (set it to **default**)	spring.cloud.servicecomb.credentials.akskCustomCipher	default
Region	spring.cloud.servicecomb.credentials.project	""

Spring Cloud Huawei can automatically obtain the AK/SK. During the configuration, set **spring.cloud.servicecomb.credentials.enabled** to **true**. Then, the AK/SK can be automatically injected without being configured.

2.3 Using the Distributed Configuration Center

Spring Cloud Huawei supports two configuration centers:

- Use the distributed configuration center provided by CSE.
  a. Download the **local lightweight microservice engine**, which contains the configuration center.
  b. Connect to the CSE engine.
- Use the **servicecomb-kie** distributed configuration center to support the long polling push model.

  Set up the environment based on the Quick Start in the **readme** file. Such a distributed configuration center will be rolled out on HUAWEI CLOUD

Getting Started

Step 1 Add the following dependencies.

```
<dependencies>
  <dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>spring-cloud-starter-huawei-config</artifactId>
  </dependency>
</dependencies>
```

Step 2 Create a project or module and define a configuration file.

The **bootstrap.yml** configuration file is used. In Spring Cloud, the startup priority of the **bootstrap.yml** configuration file is higher than that of the **application.yml** configuration file. When the configuration information is obtained from the configuration center, the related information needs to be defined in the **bootstrap.yml** configuration file.

```
spring:
  application:
    name: price
  cloud:
    servicecomb:
      config:
        serverAddr: https://cse.cn-east-3.myhuaweicloud.com
        watch:
          delay: 10000
      credentials:
        enabled: true
        accessKey: your ak
        secretKey: your sk
        akskCustomCipher: default
```

Step 3 Obtain the value configured in the configuration center from the **@Value** annotation.

```
@Value("${server.port}")
```

----End

Running the Demo

The following shows how to access the distributed configuration center based on a project demo. **spring-cloud-huawei-sample/config-demo** is used to simulate the scenario where ServiceStage is used to implement distributed configuration in a Spring Cloud project.

Step 1 Create related configurations on ServiceStage, for example, **dd** in the demo. For details, see **Global Configuration**.

Step 2 Modify the **bootstrap.yml** file and start the service.

```
spring:
  application:
    name: price
  cloud:
    servicecomb:
      config:
        serverAddr: https://cse.cn-east-3.myhuaweicloud.com
        watch:
          delay: 10000
      credentials:
        enabled: true
        accessKey: your ak
        secretKey: your sk
        akskCustomCipher: default
```

Step 3 Implement verification.

Access the following address (the value of **dd** configured in the configuration center will then be displayed):

http://127.0.0.1:8080/price?id=11

----End

Configuration Items

The following table lists the configuration items in the **bootstrap.yaml** file.

Configuration Item	Key	Default Value
ServiceComb start	spring.cloud.servicecomb.config.enabled	true
Service name	spring.cloud.servicecomb.discovery.serviceName	If no service name exists, use **spring.application.name**.
Application name	spring.cloud.servicecomb.discovery.appName	default
Environment name	server.env	""
Version	spring.cloud.servicecomb.discovery.version	-

Configuration Item	Key	Default Value
Configuration center address. You can use commas (,) to separate multiple addresses.	spring.cloud.servicecomb.config.serverAddr	""
Configuration center type. (The value can be kie, indicating the access kie. It is connected to config-center by default.)	spring.cloud.servicecomb.config.serverType	-
long polling (supported by only kie).	spring.cloud.servicecomb.config.enableLongPolling	true
Default polling configuration period	spring.cloud.servicecomb.config.watch.delay	10*1000 ms
Waiting time for establishing a long polling	spring.cloud.servicecomb.config.watch.pollingWaitTimeInSeconds	20 s

To interconnect with the professional engine, you need to configure the AK/SK for IAM authentication. The AK/SK configuration items in the **bootstrap.yaml** file are described as follows:

Configuration Item	Key	Default Value
Authentication enablement	spring.cloud.servicecomb.credentials.enabled	false
Tenant AK	spring.cloud.servicecomb.credentials.accessKey	-
Tenant SK	spring.cloud.servicecomb.credentials.secretKey	-
Cipher (set it to **default**)	spring.cloud.servicecomb.credentials.akskCustomCipher	default
Region	spring.cloud.servicecomb.credentials.project	""

2.4 Accessing Gateways

Spring Cloud supports gateways such as Netflix Zuul and Spring Cloud gateways. To connect them to the service registry and discovery center of Cloud Service Engine (CSE), you only need to import JAR packages and make simple configuration, but do not need to modify any code.

Getting Started

You can connect a gateway to the registry center in the same way as a common Spring Cloud application. No additional configuration is required. For details, see **Using the Distributed Registry Center**.

Running the Demo

The following shows how to access gateway services based on a project demo. **spring-cloud-huawei-sample/gateway-demo** is used to simulate the scenario where ServiceStage is used to migrate Spring Cloud and Zuul gateway services to the registry center of Apache ServiceComb.

- **backend-service**: Backend service.
- **gateway-service**: Spring Cloud gateway service.
- **zuul-service**: Netflix Zuul gateway service.

Run the demo according to the following procedure:

1. Download **the local lightweight microservice engine**, decompress it, and run **start.bat**. Then you can access the following address to go to the console.

   http://localhost:30106
2. Respectively run the main function to start the three demo services.
3. Access the following addresses:

   Access the service through the Spring Cloud gateway: http://127.0.0.1:8082/backend-service/hello

   Access the service through the Netflix Zuul gateway: http://127.0.0.1:8083/backend-service/hello

   Check whether the expected result is returned.

You are advised to use ServiceComb edge service gateways. Based on performance tests, Edge service gateways support higher concurrent traffic and provide better performance than Spring Cloud and Netflix Zuul gateways.

2.5 Registering Contracts

Reasons for registering contracts:

1. View and export swagger API documents that are automatically generated (without writing any code or making any configuration).
2. Enjoy better performance when interconnecting with edge service gateways compared with Spring Cloud gateways. For details, see Performance Tests.
3. Enjoy the **performance advantages** brought by the reactive thread model when interconnecting with Java chassis microservice applications.

Prerequisite

You have added the **spring-cloud-starter-huawei-servicecomb-discovery** dependency.

Getting Started

For a Spring Cloud application, you only need to add the following dependencies.

1. Add the following dependencies.

```
<dependencies>
  <dependency>
     <groupId>com.huaweicloud</groupId>
     <artifactId>spring-cloud-starter-huawei-swagger</artifactId>
  </dependency>
</dependencies>
```

   NOTE

   Note: If you only need API management but do not need to connect Service Comb Java-chassis applications or edge service gateways, you can implement asynchronous contract registry through **spring.cloud.servicecomb.swagger.enableJavaChassisAdapter:false** (default value: **true**) to accelerate application startup.

2. After the contract registry succeeds, the registered swagger contract information will be printed on the SDK side. Then go to the Cloud Service Engine (CSE) console, choose **Service Catalog**, and view the service contract in the registered microservice details.

Running the Demo

The following shows how to use a contract during interconnection with a Java-chassis application and an edge service gateway based on a project demo. **spring-cloud-huawei-sample/swagger-demo** is used to simulate the scenario where a Spring Cloud application is connected to a Java-chassis application. The following three services are involved:

- **springcloud-provider**: Spring Cloud provider service.
- **javachassis-consumer**: Java-chassis consumer service.
- **edge-service**: Edge service gateway service.

Run the demo according to the following procedure:

1. Download **the local lightweight microservice engine**, decompress it, and run **start.bat**.Then you can access **http://localhost:30106** to go to the console.
2. Respectively run the main function to start the three demo services.
3. Access **http://localhost:30106/#/cse/service/list** to go to the local CSE console. Then, go to the microservice details page and click the **Schema** tab to view the contract information.
4. Access the following addresses:

   http://localhost:18080/rest/swagger-provider/hello?name=123 (when the edge service gateway is used to access services)

   http://localhost:8082/consumer/hello?name=123 (when the Java chassis consumer service is connected to the Spring Cloud service)

   Check whether the expected result is returned.

Configuration Items

The following table lists the configuration items in the **application.yaml** file.

Configuration Item	Key	Default Value
Calling with Java chassis. If this parameter is set to **false**, the contract is registered asynchronously to speed up the startup.	spring.cloud.servicecomb.swagger.enableJavaChassisAdapter	true

2.6 Implementing Route Management

For route management, flexible route definition rules are provided, which can be used to implement dark launch easily.

Prerequisite

The **local lightweight microservice engine** is downloaded.

Getting Started

Step 1 Add the following dependencies for route management.

```
<dependency>
  <groupId>com.huaweicloud</groupId>
  <artifactId>spring-cloud-starter-huawei-router</artifactId>
</dependency>
```

Step 2 Create a project or module and define a configuration file.

You can configure routing rules by setting the **application.yml** file. For details, see **Route Management Rules**.

```
servicecomb:
  routeRule:
    canary-provider: |
      - precedence: 1 #Priority.
        route: #Routing rule.
          - weight: 80
            tags:
              version: 0.0.1
          - weight: 20
            tags:
              version: 0.0.2
```

Step 3 Start services. The traffic is then allocated based on the configuration.

----End

Running the Demo

The following shows how to implement dark launch based on a project demo. **spring-cloud-huawei-sample/canary-demo** is used to simulate the scenario where ServiceStage is used to implement dark launch in a Spring Cloud project.

Step 1 Respectively modify the **application.yml** files of the **canary-consumer**, **canary-provider**, and **canary-provider-beta** projects. For the **canary-consumer** project, you also need to configure the **bootstrap.yaml** file for connecting to the configuration center.

The following lists the configuration items in the **application.yaml** file.

```
server:
  port: 8091
spring:
  application:
    name: consumer
  cloud:
    servicecomb:
      discovery:
        enabled: true #Enable service discovery.
        address: https://cse.cn-north-4.myhuaweicloud.com:443  #Registry center address.
        version: 0.0.2 #Service version.
        healthCheckInterval: 30 #Heartbeat interval.
        autoDiscovery: true # Avoid domain name resolution in the same Virtual Private Cloud (VPC).
      credentials:
        enabled: true # Enable IAM authentication.
        accessKey: yourak # Access Key ID (AK).
        secretKey: yoursk # Secret Access Key (SK).
        project: cn-north-4 # Region.
        akskCustomCipher: default
```

The **bootstrap.yaml** file is used to specify the configuration center address.

```
spring:
  application:
    name: consumer
  cloud:
    servicecomb:
      config:
        serverAddr: https://cse.cn-north-4.myhuaweicloud.com # Configuration center address.
        watch:
          delay: 10000
      credentials:
        enable: true # Enable IAM authentication.
        accessKey: yourak # AK.
        secretKey: yoursk # SK.
        akskCustomCipher: default # Region.
        project: cn-north-4
```

For the **canary-consumer** project, you can also configure **dark launch rules** in the **application.yaml** file and change routes to achieve expected traffic allocation. In addition, dark launch supports both Feign and Hystrix, and no additional reference or configuration is required. Example:

```
servicecomb:
  routeRule:
    canary-provider: |
      - precedence: 2 #Priority.
        route: #Routing rule.
          - weight: 100
            tags:
              version: 0.0.1
```

Step 2 Respectively start the **canary-consumer**, **canary-provider**, and **canary-provider-beta** services.

Step 3 Implement verification.

Access the following address:

http://127.0.0.1:8091/canary?id=11

The traffic is allocated as expected.

----End

The following table lists the configuration items in the **application.yaml** file.

Configuration Item	Key	Default Value
Header key to be transparently transmitted. The specified header can be transparently transmitted to all downstream microservices for route management. It is of the array type.	servicecomb.router.header	[]
Route rules setting for a specified calling service	servicecomb.routeRule.[provider]	-

2.7 Route Management Rules

Scenario

Through route management, you can perform dark launch to smoothly upgrade versions.

Precautions

1. You need to access microservices to the configuration center before using the dark launch function.
2. You can customize dark launch rules by setting the **microservice.yaml** file. Example:

```
servicecomb:
    routeRule:
      provider: | #Service name.
      - precedence: 2 #Priority.
          match:        #Matching policy.
            headers:          #Header matching.
              region:
                regex: 'regoin[0-9]*'
                caseInsensitive: false # Whether the value is case-sensitive. The default value is false, indicating that the value is case-sensitive.
              type:
                exact: gray
        route: #Routing rule.
          - weight: 100 #Weight value.
            tags:
```



```
        version: {version1}
        app: {appId}
  - precedence: 1
    route:
      - weight: 20
        tags:
          version: 0.01
          tags: tag
      - weight: 80
        tags:
          version: 0.02
```

Rule Description

- **match** specifies the requests to be matched. The matching condition is **header**.
- **header** supports regular expressions for accurate matching.
- If **match** is not defined, any request can be matched.
- The forwarding weight is defined in **routeRule.{targetServiceName}.route** and is configured by **weight**. The value of **weight** indicates the percentage. The sum of the values must be equal to 100. If the sum is smaller than 100, the value in the latest version is calculated.
- **tags** specifies a service group and is defined in **routeRule.{targetServiceName}.route**. The configuration content includes **version** and **app**.
- **caseInsensitive** determines whether a value is case-sensitive. **false** (default): indicates that the condition is case-sensitive. **true**: indicates that the value is not case-insensitive.
- A larger priority value indicates a higher priority.

Matching Process

In the preceding example, all requests for accessing the provider service are first matched with the rule with the priority of 2. If the header contains the key which is **region**, implement case-sensitive matching based on the regular expression rule. If the header contains the key which is **type**, implement accurate matching based on the string. If the matching succeeds, 100% traffic is allocated to the instances corresponding to the versions and tags based on the route configuration. If the matching fails, the rule with the priority of 1 is used. Different rules cannot have the same priority.

Troubleshooting

If rule configurations are incorrect or no rule is matched, dark launch rules are skipped. In this case, it can be regarded that no dark launch configuration exists.

If a rule is matched but no instance can be found based on versions or tags, remaining traffic is automatically forwarded to the latest version.

Load Balancing Algorithm

Like Nginx, the smooth weighted round robin algorithm is used to evenly distribute traffic to service instances.

Instance Tags Configuration

For microservice instances, you can specify **tags** by setting the **microservice.yaml** file or through APIs of the service center.

```
instance_description:
  properties:
    tags:
      tag_key: tag_value
```

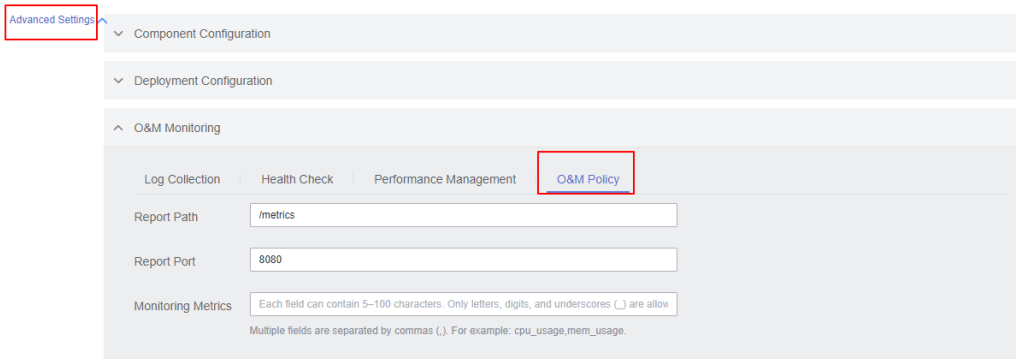
3 Go Chassis

3.1 Monitoring Metrics

Metrics are important monitoring data for service running. Go-chassis collects metrics in Promethues exporter mode and exports them. In addition to system metrics, such as QPS, latency, requests, CPU, and memory, the system also supports collection of user-defined service metrics, such as the number of login users, online users, and transactions.

Step 1 Enable the metric function. For details, see **Metrics**.

Step 2 When deploying a Go-chassis application component in a container on ServiceStage, configure parameters in **O&M Policy**. **Report Path** is the path configured in step 1. If the path is not configured, the default path is **/metrics**. **Report Port** is the listening port of the application, as shown in the following figure.



Step 3 After the configuration and deployment are complete, you can view detailed metrics on AOM. On the AOM page, choose **View Management** > **Metric Monitoring**.

Step 4 In the **Select Metrics** area, select and view the desired metrics.

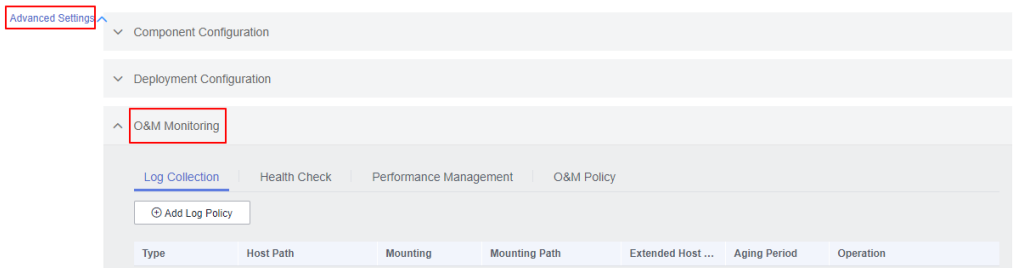
In addition to viewing service metrics, you can view instance metrics.

----End

3.2 Collecting and Viewing Logs

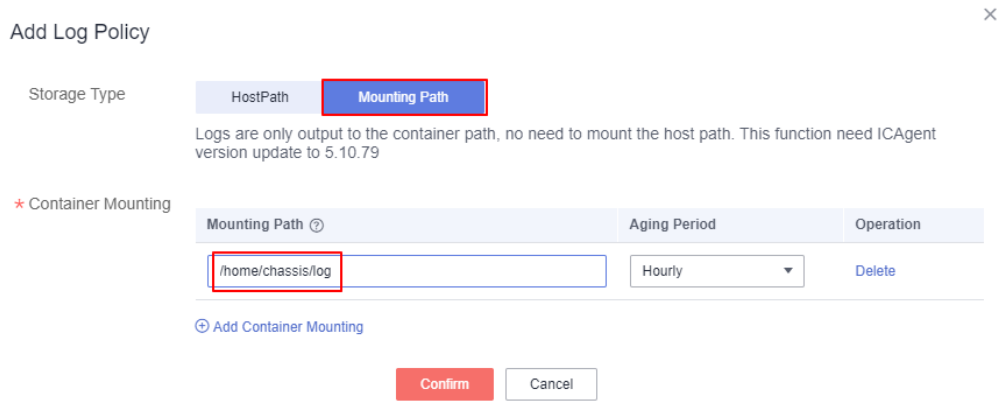
When you use the ServiceStage to deploy a Go-chassis application component, you can view related run logs on the **Application Operations Management** page.

Step 1 During application component deployment, expand **Advanced Settings**, click **O&M Monitoring**, click **Log Collection**, and click **Add Log Policy** , as shown in the following figure.

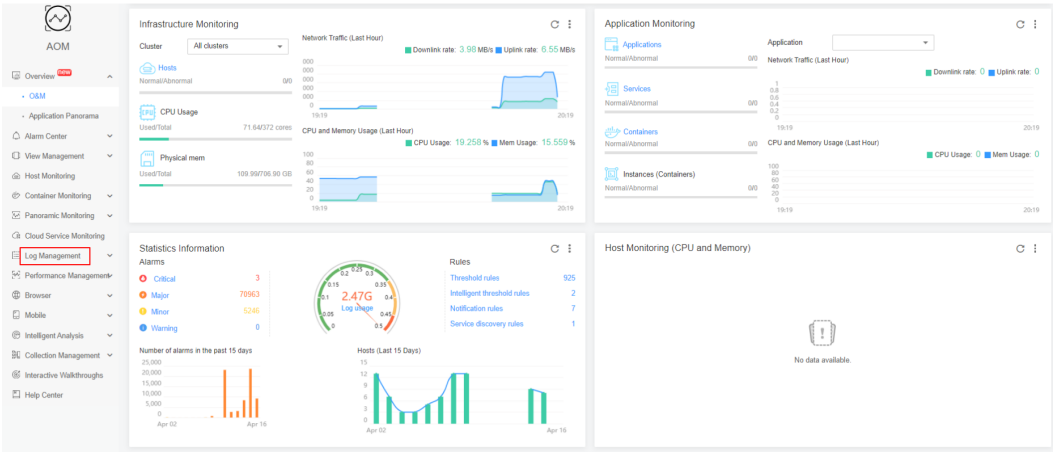


Step 2 In the **Add Log Policy** dialog box, enter the path where the running log locates in **Mounting Path**.

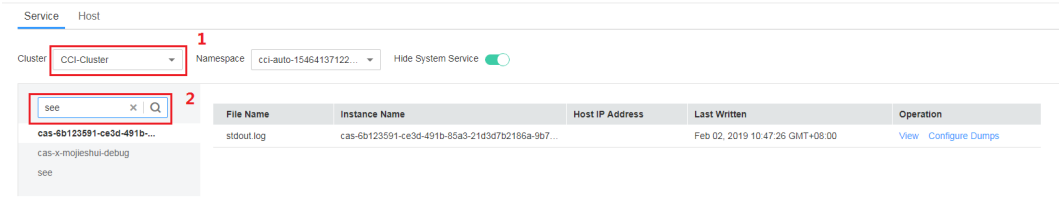
As shown in the following figure, the application component runs in **/home/chassis**, and the log path is set to **log/chassis.log**.



Step 3 Access the **Application Operations Management** service, choose **Log Management**



Step 4 Select the cluster where the application component locates, enter an application component name to search for the application component, and click **View** next to **mesher.log** to view the application, as shown in the following figure.



----End

3.3 Locating Circuit Breaker

If an API on the server times out, a large number of concurrent requests are sent from the client, or the server returns an error (caused by a network error or an error user-defined status code), circuit breaker occurs.

If circuit breaker occurs, the API on the server is isolated from that on the client by network. Then, the client does not access the API. You can analyze client and server logs to locate faults and find the root causes.

Locating Method

Step 1 The API on the server is isolated. Check the server logs to find out what happened when the API was called.

Step 2 Check the client logs.

- View the logs related to the Call got Error to check the error information returned by the transport layer during the calling, such as the network error, timeout, and error user-defined status code.
- If **max concurrency** is displayed, concurrent client requests exceed the threshold.

----End

Alternatively, check metrics in **Monitoring Metrics**.

Possible Causes

- A deadlock occurs on the server.
- A large number of concurrent requests are sent from the client.
- The server response times out.
- The status code returned by the server complies with the error code defined in **Transport**.

4 Mesher

4.1 Overview

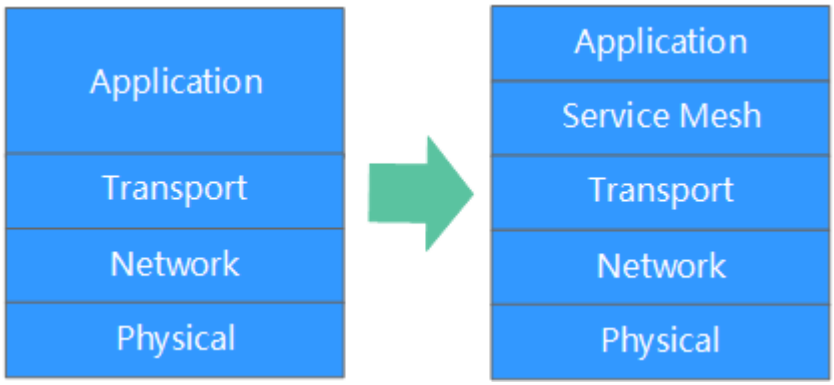
What Is Mesher?

Mesher provides the Service Mesh, which is a lightweight proxy service that runs together with microservices in Sidecar mode.

Service mesh is defined by William Morgan.

A Service Mesh is a dedicated infrastructure layer for handling service-to-service communication. It's responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application. In practice, the Service Mesh is typically implemented as an array of lightweight network proxies that are deployed alongside application code, without the application needing to be aware.

The concept of the Service Mesh as a separate layer is tied to the rise of the cloud native application. In the cloud native model, a single application might consist of hundreds of services; each service might have thousands of instances; and each of those instances might be in a constantly-changing state. Not only is service communication in this world incredibly complex, it's a pervasive and fundamental part of runtime behavior. Managing it is vital to ensuring end-to-end performance and reliability.



The Service Mesh is a networking model that sits at a layer of abstraction above TCP/IP. It assumes that the underlying L3/L4 network is present and capable of delivering bytes from point to point. (It also assumes that this network, as with every other aspect of the environment, is unreliable; the Service Mesh must therefore also be capable of handling network failures.)

In some ways, the Service Mesh is analogous to TCP/IP. Just as the TCP stack abstracts the mechanics of reliably delivering bytes between network endpoints, the Service Mesh abstracts the mechanics of reliably delivering requests between services. Like TCP, the Service Mesh does not care about the actual payload or how it is encoded. The application has a high-level goal ("send something from A to B"), and the job of the Service Mesh, like that of TCP, is to accomplish this goal while handling any failures along the way.

Unlike TCP, the Service Mesh has a significant goal beyond "just make it work": it provides a uniform, application-wide point for introducing visibility and control into the application runtime. The explicit goal of the Service Mesh is to move service communication out of the realm of the invisible, implied infrastructure, and into the role of a first-class member of the ecosystem—where it can be monitored, managed and controlled.

Why Do I Use Mesher?

- Service codes do not need to be reconstructed.
- Existing applications can be accessed.
- Common applications quickly become cloud-native.
- Service codes do no need to be modified.

Basic Implementation Principle

Mesher is the proxy of Layer 7 protocols. It runs in Sidecar mode in a pod where applications reside, and shares network and storage resources with the pod.

1. Applications in the Pod use Mesher as the HTTP proxy to automatically discover other services.
2. Instead of applications in the Pod, Mesher registers with the registration center to be discovered by other services.

The network request process of a consumer and provider using Service Mesh is as follows:

- **Scenario 1**. Only the consumer uses Mesher in Sidecar mode.

  In this scenario, the provider needs to be available for service registration and discovery, or uses the Java or Go chassis development framework. Otherwise, the AccountService cannot be discovered.

  The network request process between applications is as follows:

  Store web -> Mesher -> Account service
- **Scenario 2**. Both the consumer and provider use Mesher in Sidecar mode.

  In this scenario, the microservice development framework is not required.

  The network request process between applications is as follows:

  Store web -> Mesher -> Mesher -> Account service
- **Scenario 3**. Only the provider uses Mesher in Sidecar mode.

  In this scenario, the consumer needs to use the Java or Go chassis development framework.

  The network request process between applications is as follows:

  Store web -> Mesher -> Account service

Precautions

Configurations need to be modified accordingly after applications are deployed on the cloud. For example, when Mesher is not used, StoreWeb uses http://IP:port/ to access AccountService. After Mesher is used, StoreWeb uses http://AccountService:port/ to access AccountService, which is explained in detail in the following.

4.2 Admin API Definition

4.2.1 Overview

Service Mesh Admin APIs provide routing query, health check, monitoring, and other functions.

Configuration

Edit **chassis.yaml** to enable admin API server listening.

```
cse:
  protocols:
    grpc:
      listenAddress: 127.0.0.1:40101
    http:
      listenAddress: 127.0.0.1:30101
    rest-admin:
      listenAddress: 127.0.0.1:30102 # listen addr use to adminAPI
```

Modify the **mesher.yaml** file to enable Admin APIs.

```
admin:
  enable: true #Whether to enable APIs.
  goRuntimeMetrics : true   #Whether to enable monitoring during go running and export metric data
```


4.2.2 Health Check

Business Function

This API is used to check the health status of Mesher.

URI

GET /v1/mesher/health

Request

None

Response

Response Parameter

Table 4-1 describes the response parameters.

Table 4-1 Parameter description

Parameter	Type	Description
Health	health	Health status

Table 4-2 Health parameter description

Parameter	Type	Description
serviceName	string	Microservice name.
version	string	Microservice version.
status	string	The status can be green (healthy) or red (unhealthy). After being disconnected from the registration center, the status changes to red.
connectedConfigCenterClient	bool	Whether to connect to the configuration center.
connectedMonitoring	bool	Whether to connect to the CSE dashboard.

Response Example

```
{
   "serviceName": "cart",
   "version": "0.1",
   "status": "green",
```



```
  "connectedConfigCenterClient": false,
  "connectedMonitoring": false
}
```

Status Code

Table 4-3 describes the status code.

Table 4-3 Status code

Code	Description
200	Request succeeded.
500	Internal error.

4.2.3 Mesher Version

Business Function

This API is used to check the version of Mesher.

URI

GET /v1/mesher/version

Request

None

Response

Response Parameter

Table 4-4 describes the response parameters.

Table 4-4 Parameter description

Parameter	Type	Description
text	text	Version number

Response Example

```
1.0
```

Status Code

Table 4-5 describes the status code.

Table 4-5 Status code

Code	Description
200	Request succeeded.
500	Internal error.

4.2.4 Routing Information

Business Function

This API is used to query the routing information about a service.

URI

GET /v1/mesher/routeRule/{serviceName}

Table 4-6 describes the parameters.

Table 4-6 Parameter description

Parameter	Location	Type	Mandatory or Not	Description
serviceName	path	String	No	Microservice name, used for checking how microservice consumers to perform target microservice diversion.

Request

None

Response

Response Parameter

Table 4-7 Parameter description

Parameter	Type	Description
Route information	text	Complete routing information

Response Example

```
- precedence: 2
  route:
  - tags:
      app: HelloWorld
      version: "1.2"
    weight: 80
  - tags:
      app: HelloWorld
      version: "1.3"
    weight: 20
  match:
    refer: vmall-with-special-header
    source: vmall
    sourceTags:
      version: v2
    httpHeaders:
      X-Age:
        exact: "18"
      cookie:
        regex: ^(.*?;)?(user=jason)(;.*)?$
- precedence: 1
  route:
  - tags:
      version: "1.0"
    weight: 100
  match:
    refer: ""
    source: ""
    sourceTags: {}
    httpHeaders: {}
```

Status Code

Table 4-8 describes the status code.

Table 4-8 Status code

Code	Description
200	Request succeeded.
400	Request error.
500	Internal error.

4.2.5 Metrics

Business Function

standard format of open-source Prometheus. You can use Promethues to collect data and connect to the Grafana for monitoring.

URI

GET /v1/mesher/metrics

Request

None

Response

Response Parameter

Table 4-9 describes the response parameters.

Table 4-9 Parameter description

Parameter	Type	Description
text	text	Metrics in Prometheus format.

Response Example

```
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 28
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.9"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 3.891488e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.63252264e+08
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash
table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.454785e+06
# HELP Server_request_latency_seconds request_latency_seconds
# TYPE Server_request_latency_seconds summary
Server_request_latency_seconds{AppId="CSE",version="0.1",quantile="0.5"} 0.000658564
Server_request_latency_seconds{AppId="CSE",version="0.1",quantile="0.9"} 0.001710824
Server_request_latency_seconds{AppId="CSE",version="0.1",quantile="0.99"} 0.001710824
Server_request_latency_seconds_sum{AppId="CSE",version="0.1"} 0.004386012
Server_request_latency_seconds_count{AppId="CSE",version="0.1"} 5
# HELP Server_requests_total requests_total
# TYPE Server_requests_total counter
Server_requests_total{AppId="CSE",version="0.1"} 5
# HELP Server_succeses_total succeses_total
# TYPE Server_succeses_total counter
Server_succeses_total{AppId="CSE",version="0.1"} 5
# HELP mesher_request_latency_seconds request_latency_seconds
# TYPE mesher_request_latency_seconds summary
mesher_request_latency_seconds{AppId="default",version="0.1",quantile="0.5"} 0.000658564
mesher_request_latency_seconds{AppId="default",version="0.1",quantile="0.9"} 0.001710824
mesher_request_latency_seconds{AppId="default",version="0.1",quantile="0.99"} 0.001710824
mesher_request_latency_seconds_sum{AppId="default",version="0.1"} 0.004386012
mesher_request_latency_seconds_count{AppId="default",version="0.1"} 5
# HELP mesher_5xx_errors 5xx_errors
# TYPE mesher_5xx_errors counter
mesher_5xx_errors{AppId="default",version="0.1"} 2
# HELP mesher_failures_total failures_total
```



```
# TYPE mesher_failures_total counter
mesher_failures_total{AppId="default",version="0.1"} 2
# HELP mesher_requests_total requests_total
# TYPE mesher_requests_total counter
mesher_requests_total{AppId="default",version="0.1"} 5
# HELP mesher_services_monitored services_monitored
# TYPE mesher_services_monitored gauge
mesher_services_monitored{AppId="default",version="0.1"} 1
# HELP mesher_start_time_seconds start_time_seconds
# TYPE mesher_start_time_seconds gauge
mesher_start_time_seconds{AppId="default",version="0.1"} 1.507637815e+09
# HELP mesher_succeses_total succeses_total
# TYPE mesher_succeses_total counter
mesher_succeses_total{AppId="default",version="0.1"} 5
```

Status Code

Table 4-10 describes the status code.

Table 4-10 Status code

Code	Description
200	Request succeeded.
500	Internal error.

4.3 Accessing CSE Through Multi-protocol

4.3.1 gRPC

Procedure

Step 1 Edit the **chassis.yaml** file to enable the gRPC proxy. By default, the proxy is enabled.

```
cse:
  protocols:
    grpc:
      listenAddress: 127.0.0.1:40101
    http:
      listenAddress: 127.0.0.1:30101
```

Step 2 Add a customized dialer to adapt to application code.

Assume that the previous code is:

```
conn,err:=grpc.Dial("10.0.1.1:50051",grpc.WithInsecure(),)
```

The code needs to be modified to:

```
 //target address is consist of the provider name(in that case "Server") and provider port
conn, err := grpc.Dial("Server:50051",
    grpc.WithInsecure(),
    grpc.WithDialer(func(addr string, time time.Duration) (net.Conn, error) {
        //127.0.0.1:40101 is local grpc proxy address
```



```
        return net.DialTimeout("tcp", "127.0.0.1:40101", time)
}))
```

Step 3 Deploy Mesher and applications.

----End

4.4 Accessing Mesher Through Multi-language

4.4.1 Accessing Mesher Through PHP

For details, see **Accessing Existing Applications**.

4.4.2 Accessing Mesher Through .NET

For details, see **Accessing Mesher Through .NET Core**.

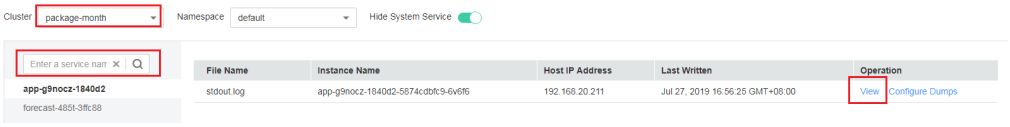
4.5 O&M

4.5.1 Viewing Logs

After selecting **Mesher** and deploying application components using ServiceStage, you can view Mesher running logs in **Application Operations Management**.

Step 1 Click the service list and click **Application Operations Management**. On the page that is displayed, choose **Log Management**.

Step 2 Select the cluster where the application component locates, enter an application component name to search for the application component, and click **View** next to **mesher.log** to view the application, as shown in the following figure.



----End

5 Configuration Information About the Middleware Bound to Applications

During application creation (such as web and mobile applications), middleware (such as DCS and RDS) can be bound. You can obtain the configuration information about the middleware bound to applications by using the following environment variables:

Environment Variables of Distributed Sessions

Distributed sessions are stable and reliable session storage based on Distributed Cache Service (DCS), supporting automatic injection for mainstream web containers, such as tomcat context, node.js express-session, and PHP session handler.

Table 5-1 describes the environment variables of distributed sessions.

Table 5-1 Environment variables of DCS sessions

Environment Variable	Description
DISTRIBUTED_SESSION_CLUSTER	Whether the instance is in cluster mode. Value: true or false.
DISTRIBUTED_SESSION_TYPE	Storage type of a distributed session instance. Currently, only Redis is supported.
DISTRIBUTED_SESSION_VERSION	Version of a distributed session instance.
DISTRIBUTED_SESSION_NAME	Name of a distributed session instance.
DISTRIBUTED_SESSION_HOST	IP address for connecting to a distributed session instance.
DISTRIBUTED_SESSION_PORT	Port for connecting to a distributed session instance.

Environment Variable	Description
DISTRIBUTED_SESSION_PASSWORD	Password for connecting to a distributed session instance.

Environment Variables of DCS

Distributed Cache Service (DCS) for Redis is an online, distributed, in-memory cache service compatible with Redis and Memcached. It is reliable, scalable, usable out of the box, and easy to manage, meeting your requirements for high read/write performance and fast data access.

Table 5-2 describes the environment variables of DCS.

Table 5-2 Environment variables of DCS

Environment Variable	Remarks
DISTRIBUTED_CACHE_CLUSTER	Whether the instance is in cluster mode. Value: true or false.
DISTRIBUTED_CACHE_TYPE	Storage type of a distributed cache instance. Currently, only Redis is supported.
DISTRIBUTED_CACHE_VERSION	Version of a DCS instance.
DISTRIBUTED_CACHE_NAME	Name of a DCS instance.
DISTRIBUTED_CACHE_HOST	IP address for connecting to a DCS instance.
DISTRIBUTED_CACHE_PORT	Port for connecting to a DCS instance.
DISTRIBUTED_CACHE_PASSWORD	Password for connecting to a DCS instance.

Environment Variables of RDS

Relational Database Service (RDS) for MySQL is a cloud-based web service that is reliable, scalable, easy to manage, and out of the box.

Table 5-3 describes the environment variables of RDS.

Table 5-3 Environment variables of RDS

Environment Variable	Description
RELATIONAL_DATABASE_NAME	Name of an RDS instance.

Environment Variable	Description
RELATIONAL_DATABASE_CONNECTION_TYPE	Connection type of an RDS instance. Value: JNDI/ SPRING_CLOUD_CONNECTOR.
RELATIONAL_DATABASE_JNDI_NAME	JNDI name of an RDS instance. This variable is used if the connection type is JNDI.
RELATIONAL_DATABASE_DB_NAME	Database name of an RDS instance.
RELATIONAL_DATABASE_DB_USER	Database user of an RDS instance.
RELATIONAL_DATABASE_DB_TYPE	Database type of an RDS instance. Currently, only MySQL is supported.
RELATIONAL_DATABASE_VERSION	Database version of an RDS instance.
RELATIONAL_DATABASE_HOST	Database IP address of an RDS instance.
RELATIONAL_DATABASE_PORT	Database port of an RDS instance.
RELATIONAL_DATABASE_PASSWORD	Database password of an RDS instance.

6 servicestage-maven-plugin Usage Guide

6.1 Tool Introduction

Function

- Uploads software packages to the HUAWEI CLOUD software repository.
- Uses the software package to associate with the application component (Tomcat or Java running environment) for upgrade.

Scenario

- After creating a Java or Tomcat application component on ServiceStage, you can use the Maven plug-in to upgrade the component by replacing the original application program package with the JAR or WAR package compiled by the local project.
- Currently, the Maven plug-in supports the CCE and CCI deployment systems.

Prerequisite

Tomcat or Java application component is created. For details, see **Creating a Web Component**.

Configuring the Tool

1. Download the **servicestage-maven-plugin** package, copy and decompress it to your local Maven repository.

   After being decompressed, its directory structure is as follows:

   ```
   ├──com.huawei.servicestage
      ├──servicestage-client
      ├──servicestage-maven-plugin
   ```

2. Create a **servicestage.yaml** file in the root directory of the project and configure the following parameters in the file:

   ```
   iam:
    region: xxx
   ```



```
  username: xxx
  password: xxx
  domain: xxx
swr:
  namespace: xxx
  repository: xxx
  packagename: xxx
  version: xxx
application:
  id: xxx
component:
  id: xxx
instance:
  id: xxx
mavenPlugin:
  pluginSkip: false
  binaryName: xxx.war
  updateAppTimeout: 60
```

The following lists the parameters.

Parameter	Mandatory or Not	Description	Remarks
iam.domain	Yes	Tenant for an IAM login.	Account for logging in to HUAWEI CLOUD. Note: The value cannot be an email address and must be an account.
iam.username	Yes	Username for an IAM login.	Username for logging in to HUAWEI CLOUD. If there is no username, the username is the account name. Note: The value cannot be an email address or a mobile number.
iam.password	Yes	Password for an IAM login.	Password for logging in to HUAWEI CLOUD.
iam.region	Yes	Region where the current user is located.	After logging in to HUAWEI CLOUD, you can see **region=xxx** in the address bar, for example, cn-north-1 for Beijing1.

Parameter	Mandatory or Not	Description	Remarks
application.id	Yes	Application ID of ServiceStage.	Log in to ServiceStage, choose **Application Management** > **Application List**, select and click an application, and select a component to view the component details. The first 32-bit UUID in the address bar is your desired (for example, */apps/applications/info/{UUID}/{xxx}/{xxx}).
component.id	Yes	Component ID of a ServiceStage application	Log in to ServiceStage, choose **Application Management** > **Application List**, select and click an application, and select a component to view the component details. The second 32-bit UUID in the address bar is your desired (for example, */apps/applications/info/{xxx}/{UUID}/{xxx}).
instance.id	Yes	Instance ID of a ServiceStage component	Log in to ServiceStage, choose **Application Management** > **Application List**, select and click an application, and select a component to view the component details. The third 32-bit UUID in the address bar is your desired (for example, */apps/applications/info/{xxx}/{xxx}/{UUID}).

Parameter	Mandatory or Not	Description	Remarks
mavenPlugin.binaryName	Yes	Name of the binary file to be released (with the file name extension).	Generally, the package is in the target directory of the project, for example, test.jar and test.war..
swr.namespace	No	Name of the organization that archives the software package.	Log in to ServiceStage and choose **Software Center** > **Software Repository** to view the organization name. By default, there is an organization.
swr.repository	No	Name of the repository where the software package is archived.	Log in to ServiceStage and choose **Software Center** > **Software Repository** to view the repository name. The default value is truncated from the application name.
swr.packagename	No	Name of the archived software package.	Log in to ServiceStage and choose **Software Center** > **Software Repository**. The default value is truncated from the instance name.
swr.version	No	Version of the repository where the software package is archived.	Log in to ServiceStage and choose **Software Center** > **Software Repository** to view the repository version. The default value is 1.0.
mavenPlugin.pluginSkip	No	Indicates whether to skip the ServiceStage plug-in during compilation.	The default value is false.
mavenPlugin.updateAppTimeout	No	Timeout interval for upgrading an application using the plug-in.	The default value is 60*10s. For example, if the value is 3, the timeout interval is 3*10s.

3. Add the plug-in dependency to the pom file. The following is an example:

```
<build>
   <plugins>
      <plugin>
         <groupId>com.huawei.servicestage</groupId>
         <artifactId>servicestage-maven-plugin</artifactId>
         <version>1.0</version>
         <executions>
            <execution>
               <phase>package</phase>
               <goals>
                  <goal>deploy</goal>
               </goals>
            </execution>
         </executions>
      </plugin>
   </plugins>
</build>
```

6.2 User Guide

Instructions

To use the Maven plug-in, log in to ServiceStage, **create an application component**, and release the JAR or WAR package of a module in the project following **2** and **3**.

To release a module, modify the configuration of the module.

- Scenario 1: One project has only one module.

```
├──Myproject
   ├──src...
   ├──pom.xml      Add the ServiceStage plug-in configuration to the pom.xml file.
   └──servicestage.yaml   Configure parameters required by the ServiceStage plug-in.
```

- Scenario 2: A project has multiple modules. If you want to upgrade a module, for example, module-B, configure the **servicestage.yaml** file of module-B. If multiple modules need to be upgraded, configure the **servicestage.yaml** file of the corresponding modules.

```
├──Myproject
   ├──module-A
      ├──src...
      └──pom.xml

   ├──module-B
      ├──src...
      ├──pom.xml      Add the ServiceStage plug-in configuration to the pom.xml file.
      └──servicestage.yaml   Configure parameters required by the ServiceStage plug-in.

   └──module-C
      ├──src...
      └──pom.xml
└──pom.xml
```

Maven Compilation Commands

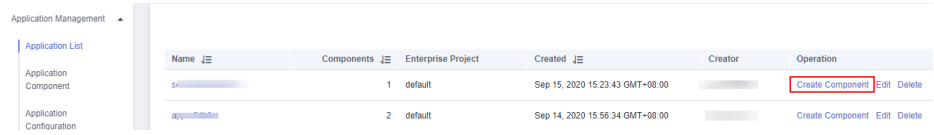
The servicestage-maven-plugin allows the mvn compilation command to transfer parameters through environment variables. During the use, the environment variables of the mvn compilation command are preferentially used, and then the variables in the **servicestage.yaml** file are used. Command example:


```
mvn clean package -Diam.domain=xxx -Diam.username=xxx  -DmavenPlugin.pluginSkip=false
```

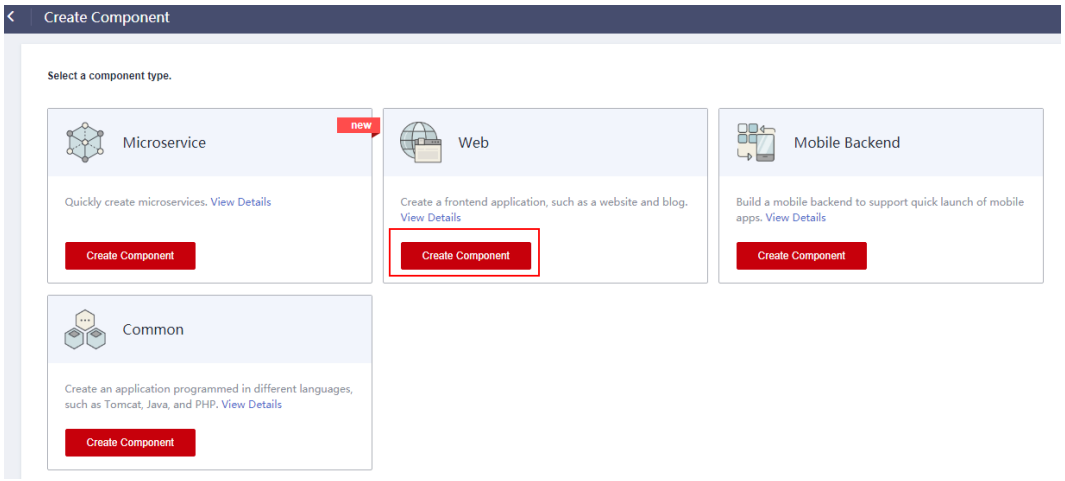
Example

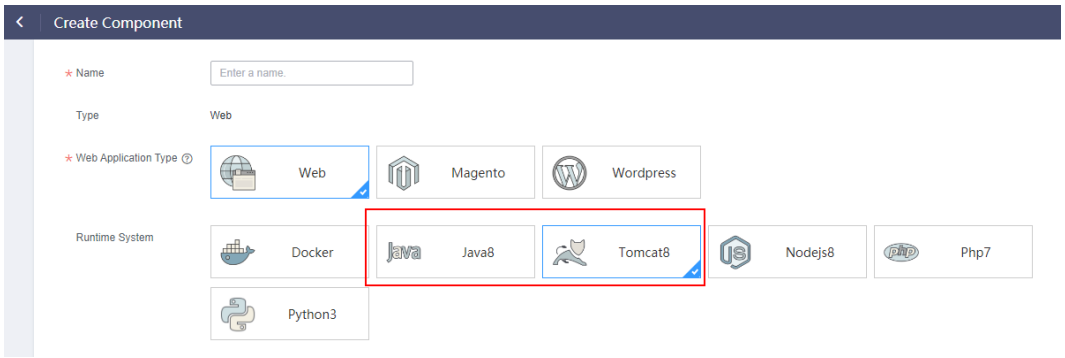
Step 1 Log in to ServiceStage, choose **Application Management** > **Application List**, and click **Create Application**. If an application has been created, click the created application.

Step 2 After the application is created, click **Create Component**. (If a component has been created, skip this step.)



Step 3 Create a web component and set **Runtime System** to **Tomcat8** or **Java8**. Set this parameter based on the site requirements. For example, if the JAR package is used, set this parameter to Java8.





Step 4 After the component is created and deployed, click the component in the **Environment** view to view the component details.

Step 5 Find the application ID, component ID, and instance ID in the address bar. For example:

https://console.huaweicloud.com/servicestage/?region=cn-south-1#/apps/applications/info/6d8065ea-45b1-44b2-a661-58fb1abb5d0a/0936177c-733c-433e-

a633-25e6973d4153/948da446-9ffb-44f0-b4c4-17c614378294/test-maven/overview

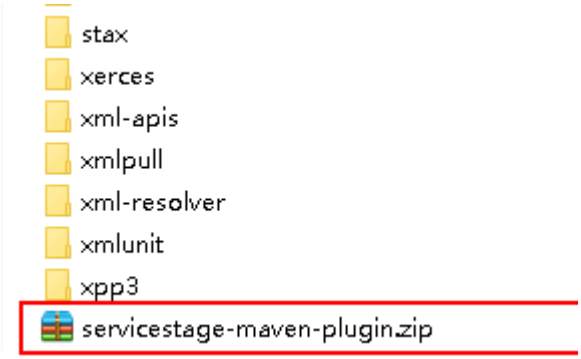
The application ID, component ID, and instance ID are as follows:

application ID: 6d8065ea-xxxx

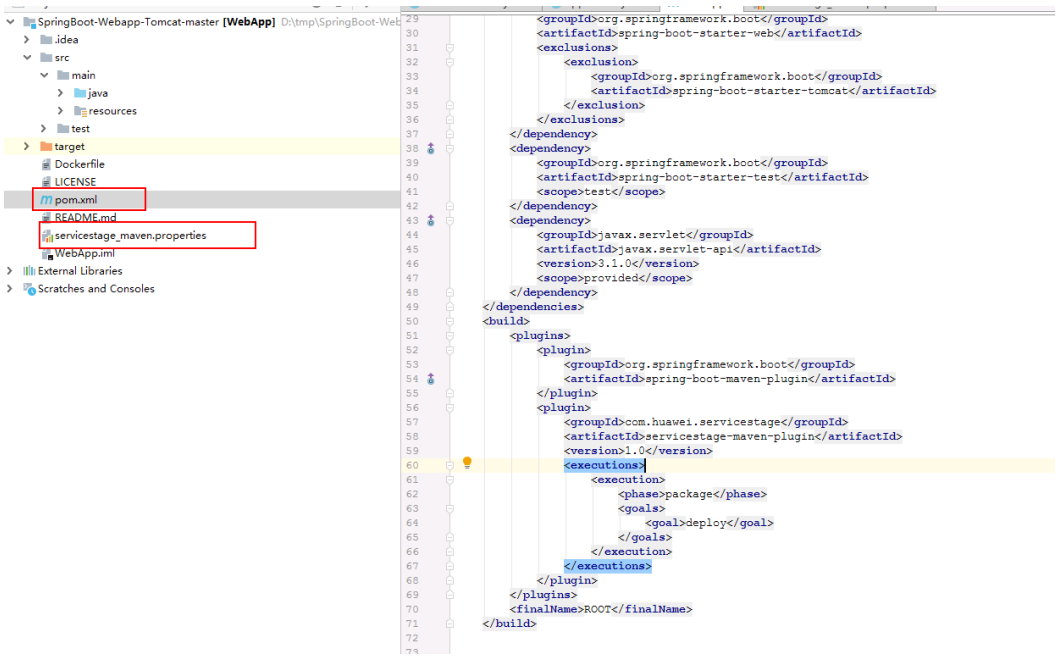
component ID: 0936177c-xxxxx

instance ID: 948da446-xxxx

Step 6 Download the compressed package on which the **Maven plug-in** depends and decompress the package to the local Maven repository.

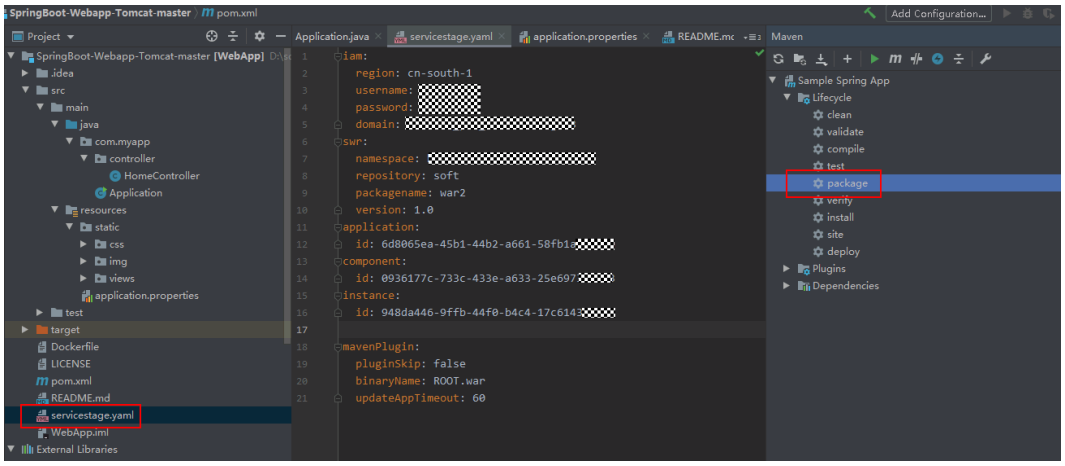


Step 7 For details about how to set parameters for a local project, see the tool configuration description in the **Tool Introduction**.



Step 8 Compile and package the code.

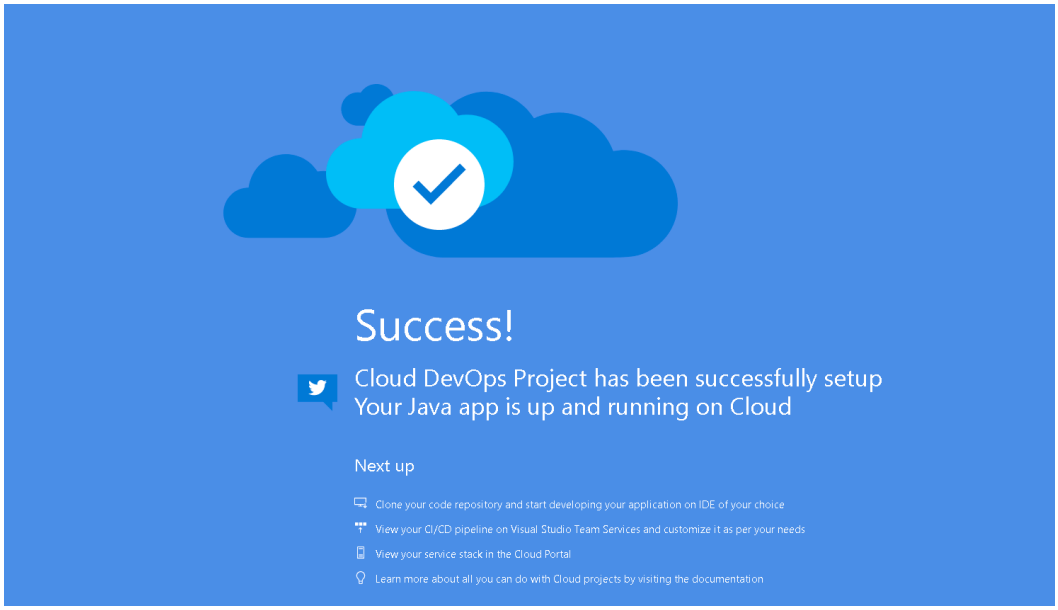
When the **mvn** command is used to compile a software package, the tool upgrades the software package compiled for the project to the specified application.



Step 9 Check the application component upgrade result.

After the upgrade is complete, access the application and compare the results before and after the upgrade.

After the upgrade is complete, access the application components, as shown in the following figure.



----End

7 Checking and Configuring the AKSK Authentication Mode

7.1 Checking Whether the AK/SK Need to Be Configured

Step 1 Check whether you use the professional CSE engine.

- If yes, go to the next step.
- If not, no further action is required.

Step 2 Check whether you deploy a microservice application based on container deployment provided by ServiceStage.

- If yes, go to the next step.
- If not, no further action is required.

Step 3 Check whether your framework and usage comply with the description in the following table.

- If not, no further action is required.
- If yes, configure the AK/SK by referring to **Configuring the AK/SK for Microservice Applications**.

Table 7-1 Framework and usage

Framework	Usage
– Java Chassis – Go Chassis – Spring Cloud Huawei	Use the frameworks to develop microservices.

Framework	Usage
Mesher	Create Mesher applications on ServiceStage.

----End

7.2 Configuring the AK/SK for Microservice Applications

When the professional CSE engine is used, you need to configure the AK/SK.

7.2.1 Java Chassis

Method 1

Add the following configurations to the **microservice.yml** file of the microservice. If they are configured, skip this step. For details about how to obtain the AK/SK and project name, see **Obtaining the AK/SK and Project Name**.

```
cse:
  credentials:
    accessKey: ak  # Set this parameter based on the actual value.
    secretKey: sk  # Set this parameter based on the actual value.
    project: project name  # Set this parameter based on the actual value.
    akskCustomCipher: default
```

Method 2

Add the following environment variables for the microservice. For details about how to obtain the AK/SK and project name, see **Obtaining the AK/SK and Project Name**.

Table 7-2 Environment variables

Environment Variable	Description
cse_credentials_accessKey	Set the AK based on the actual value.
cse_credentials_secretKey	Set the SK based on the actual value.

7.2.2 Go Chassis

Method 1

Add the following configurations to the **chassis.yaml** or **auth.yaml** file of the microservice. If they are configured, skip this step. For details about how to obtain the AK/SK and project name, see **Obtaining the AK/SK and Project Name**.

```
cse:
  credentials:
```



```
accessKey: ak  # Set this parameter based on the actual value.
secretKey: sk  # Set this parameter based on the actual value.
project: project name  # Set this parameter based on the actual value.
akskCustomCipher: default
```

Method 2

Add the following environment variables for the microservice. For details about how to obtain the AK/SK and project name, see **Obtaining the AK/SK and Project Name**.

Table 7-3 Environment variables

Environment Variable	Description
cse_credentials_accessKey	Set the AK based on the actual value.
cse_credentials_secretKey	Set the SK based on the actual value.

7.2.3 Spring Cloud Huawei

Add the following configurations to the **application.yml** file of the microservice. If they are configured, skip this step. For details about how to obtain the AK/SK and project name, see **Obtaining the AK/SK and Project Name**.

```
spring:
  cloud:
    servicecomb:
      credentials:
        enabled: true
        accessKey: ak  # Set this parameter based on the actual value.
        secretKey: sk  # Set this parameter based on the actual value.
        project: project name  # Set this parameter based on the actual value.
        akskCustomCipher: default
```

7.2.4 Mesher

Perform the following steps to create a secret named **mesher-secret**. For details about how to obtain the AK/SK and project name, see **Obtaining the AK/SK and Project Name**.

Step 1 Log in to ServiceStage and choose **Application Management** > **Application Configuration**. Then, click the **Secret** tab. On the page that is displayed, click **Create**.

Step 2 Set **Creation Mode** to **Visualization**, **Name** to **mesher-secret**, and **Cluster** and **Namespace** to the cluster and namespace where the application is to be deployed.

Step 3 Set **Secret Type** to **Opaque**.

Step 4 Set **Secret Data** according to the following table.

Table 7-4 Secret data

Key	Value
cse_credentials_accessKey	Set the AK based on the actual value and encode it using Base64.
cse_credentials_secretKey	Set the SK based on the actual value and encode it using Base64.

Step 5 Click **Create**.

----End

7.3 Appendix

7.3.1 Obtaining the AK/SK and Project Name

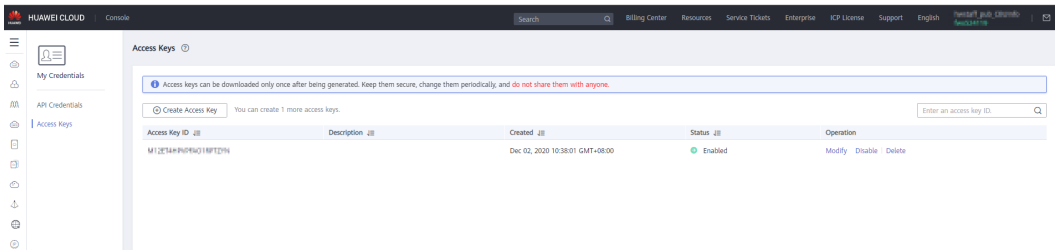
Obtaining the AK/SK

Step 1 Log in to the management console.

Step 2 Hover the mouse pointer over the username and choose **My Credentials** from the drop-down list.

Step 3 In the navigation pane, choose **Access Keys**.

View the AK/SK in the **Access Key ID** column.



----End

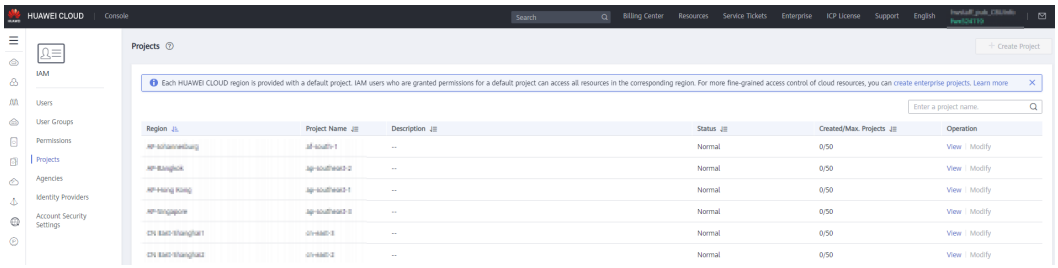
Obtaining the Project Name

Step 1 Log in to the management console.

Step 2 Hover the mouse pointer over the username and choose **Identity and Access Management** from the drop-down list.

Step 3 In the navigation pane, choose **Projects**.

View the project name in the **Project Name** column.



----End

