ALIBABA CLOUD

Alibaba Cloud

Apsara Stack Enterprise

User Guide - Middleware and Enterprise Applications

Product Version: 2006, Internal: V3.12.0

Document Version: 20200918

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Enterprise Distributed Application Service (EDAS)

1.1. What is EDAS?

Enterprise Distributed Application Service (EDAS) is a Platform as a Service (PaaS) platform for application hosting and microservice management, providing full-stack solutions such as application development, deployment, monitoring, and O&M. It supports Dubbo, Spring Cloud, and other microservice runtime environments, helping you easily migrate applications to the cloud.

Diverse application hosting environments

You can select instance-exclusive Elastic Compute Service (ECS) clusters, Container Service Kubernetes clusters, and user-created Kubernetes clusters based on your application systems and resource needs.

Abundant microservice frameworks

You can develop applications and services in the native Dubbo, native Spring Cloud, and High-Speed Service Framework (HSF) frameworks, and host the developed applications and services to EDAS.

- You can host Dubbo and Spring Cloud applications to EDAS by adding dependencies and modifying a few configurations. You have access to the features of EDAS, such as enterprise-level application hosting, service governance, monitoring and alerting, and application diagnosis, without having to build ZooKeeper, Eureka, and Consul. This lowers the costs of deployment and O&M.
- HSF is the distributed remote procedure call (RPC) framework that is widely used within Alibaba Group. It interconnects different service systems and decouples inter-system implementation dependencies. HSF unifies the service publishing and call methods for distributed applications to help you conveniently and quickly develop distributed applications. HSF provides or uses common functional modules, and frees developers from various complex technical details involved in distributed architectures, such as remote communication, serialization, performance loss, and the implementation of synchronous and asynchronous calls.

Comprehensive application management

You can perform end-to-end management, service governance, and microservice management for your applications in the EDAS console.

- Application lifecycle management

  EDAS provides end-to-end application management, allowing you to deploy, scale out, scale in, stop, and delete applications. Applications of all sizes can be managed in the EDAS console.

- Service governance

  EDAS integrates a wide variety of service governance components, such as auto scaling, throttling and degradation, and health check, to deal with unexpected traffic spikes and crashes caused by dependencies. This greatly improves platform stability.

- Microservice management

  EDAS provides the service topology, service report, and trace query features to help you manage every component and service in a distributed system.

Comprehensive monitoring and diagnosis

You can monitor the status of resources and services in applications in the EDAS console to promptly identify problems and quickly locate their causes through the logging and diagnosis components.

EDAS is connected to the Application Real-Time Monitoring Service (ARMS) to monitor the health status of application resources and services at the Infrastructure as a Service (IaaS) layer in real time, helping you quickly locate problems.

1.2. Quick start

This topic describes how to use EDAS to publish a simple web application that only contains a welcome page in Alibaba Cloud Virtual Private Cloud (VPC).

1.2.1. Log on to the EDAS console

This topic describes how to log on to the Enterprise Distributed Application Service (EDAS) console.

Prerequisites

- The domain name of the ASCM console is obtained from the deployment personnel before you log on to the ASCM console.
- A browser is available. We recommend that you use the Google Chrome browser.

Procedure

1. In the address bar, enter the URL used to log on to the ASCM console. Press the Enter key.
2. Enter your username and password.

   Obtain the username and password used to log on to the console from the operations administrator.

   > **Note** When you log on to the ASCM console for the first time, you must change the password of your username. For security reasons, your password must meet the minimum complexity requirements. The password must be 8 to 20 characters in length and must contain at least two of the following character types:
   >
   > - Uppercase or lowercase letters.
   > - Digits.
   > - Special characters. Special characters include exclamation points (!), at signs (@), number signs (#), dollar signs ($), and percent signs (%).

3. Click **Login** to go to the ASCM console homepage.
4. In the top navigation bar of the page, choose **Products** > **Middleware** > **Enterprise Distributed Application Service**.

5. On the **EDAS** page, select an **organization** and a **region**, and then click **EDAS**.

1.2.2. Deploy Java applications in ECS clusters

To help you get started quickly, Enterprise Distributed Application Service (EDAS) provides a Java web application demo that only contains a welcome page so that you can quickly learn how to publish the Java application on multiple Elastic Compute Service (ECS) instances. To use these ECS instances, you must create them on Alibaba Cloud and then deploy them in Alibaba Cloud Virtual Private Cloud (VPC) instances.

Prerequisites

- You have created a VPC instance, VSwitch, and security group.
- You have created an ECS cluster and added instances to the cluster.
- Before deploying an application, ensure that the RAM is authorized.

Create an application in an ECS cluster

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management > Applications**.
3. On the **Applications** page, click **Create Application** in the upper-right corner.
4. On the **Application Information** page, set the parameters of the application. Then, click **Next**.

Basic information and parameters of the application

Parameter	Description
Namespace	Select a namespace from the drop-down list.
Cluster Type	From the first drop-down list, select **ECS Cluster**. From the second drop-down list, select a specific cluster.
Application Name	Enter an application name, which must be 1 to 36 characters in length.
Deployment Method	Select **WAR** or **JAR** based on the application.
Application Runtime Environment	Select the application runtime environment based on the application framework. ○ For High-Speed Service Framework (HSF) applications, select **EDAS-Container**. ○ For Spring Cloud or Dubbo applications, ■ **WAR**: Select **Apache Tomcat**. ■ **JAR**: Select **Standard Java application runtime environment**.
Java Environment	Select **Open JDK 8**.
Application Description	Enter remarks for the application.

5. On the **Application Configuration** page, add an **instance**, set the deployment parameters, and then click **Create**.

Parameter	Description
Selected Instances	Click **New**. On the **Instances** page, select instances and click > to add the instances to the right-side section. Then, click **OK**. ○ **If no instances are selected**, click **Create an Empty Application**. Then, Scaling (applicable to ECS clusters), **add instances** or Deploy an application (applicable to ECS clusters) to complete the deployment. ○ **If instances are selected**, click **Create** to create an empty application that contains the instances. Then, you can click **Deploy Application**Deploy an application (applicable to ECS clusters) to publish the application.
Deploy Now	Select this option after instances are added. Set the deployment parameters in the lower section.
Deployment Method	Select **WAR** or **JAR**. The configuration processes for WAR package deployment and JAR package deployment are similar. Here, **WAR package deployment** is used as an example.
File Uploading Method	Select **Upload WAR Package** or **WAR Package Location**. ○ **Upload WAR Package**: Click Download Sample WAR Package. After the sample is downloaded, click **Select File** and select the WAR package. ○ **WAR Package Location**: Right-click Download Sample WAR Package and choose **Copy Link Address** from the shortcut menu. Copy and paste the address in the WAR package address bar. **Note** The name of the application deployment package can only contain letters, numbers, hyphens (-), and underscores (_). A JAR package can be uploaded only when the JAR package deployment method is selected. Otherwise, you can only deploy the application by using a WAR package.
Version	Enter a version number, for example, 1.1.0. We do not recommend that you use a timestamp as the version number.
(Optional) **Application Health Check**	Set a URL for application health check. The system checks the health of the application after EDAS Container has started or is running. Then, it performs a service routing task based on the health check result. In this example, the health check URL is set to `http://127.0.0.1:8080/healthCheck.html`.

Parameter	Description
Batch	Specify a number of deployment batches. Select an option from the drop-down list. The options are automatically generated based on the number of instances for the application. If you select two or more batches, you must set **Batch Wait Time**.
Batch Mode	Select **Automatic**.

After creating the application, go to the Change Details page. Click the **Basic Information** and **Instance Information** tabs. If the application status is **Normal**, the application is successfully deployed.

Update an application

The application has been deployed. You can update the application by deploying the application.

1. In the left-side navigation pane of the EDAS console, choose **Application Management** > **Applications**.
2. On the **Applications** page, click the name of the application for which you want to deploy.
3. On the **Instance Information** tab of the Application Details page, check whether any instances are available for the application. If no instances are available, click **Application Scale Out** to add at least one instance for the application. For more information, see Scaling (applicable to ECS clusters).
4. Click **Deploy Applications**. Configure the deployment parameters as prompted and click **Deploy**.
5. After the application is redeployed, the **Change Details** page appears, where you can view the deployment process and logs.
   After the deployment process is completed, if the status changes to **Execution Successful**, the deployment is successful.

Configure SLB and access the application

The application is created and published in a VPC. Therefore, the application does not have a public IP address unless otherwise specified. If your application is deployed on multiple ECS instances and you want to expose your application to external systems, we recommend that you configure a public Server Load Balancer (SLB). In this way, application access traffic can be distributed to ECS instances based on forwarding policies, which can enhance the service capability and availability of the application.

1. In the **Application Settings** section of the **Basic Information** page, click **Add** on the right of **SLB (Internet)**.

> **Note** If you have configured an SLB instance, the IP address and port number of the SLB instance are displayed. You can click Modify to go to the configuration page and modify the information of the SLB instance. You can also click Unbind to unbind the SLB instance.

2. In the **Bind SLB to Application** dialog box, select an SLB instance and set the listening port, virtual group, and forwarding policy. Set the SLB parameters, and click **Confirm change** to complete the configuration.
3. Copy the configured IP address and port number of the SLB instance such as 118.31.XXX.XXX:81, paste it in your browser address bar, and press Enter to go to the homepage of the application.

1.2.3. Deploy Spring Cloud applications to EDAS

You have developed a Spring Cloud application that depends on components such as Eureka, Consul, and ZooKeeper to implement service registration and discovery. To deploy the application in Enterprise Distributed Application Service (EDAS), you need to replace the dependencies and configurations of the service registration and discovery components with Spring Cloud Alibaba Nacos Discovery. In this case, you can deploy the application in EDAS and manage the application's microservices in EDAS without modifying any business code.

Background

Spring Cloud Alibaba Nacos Discovery implements the standard interfaces and specifications of Spring Cloud Registry, which are consistent with how Spring Cloud visits components such as Eureka, Consul, and ZooKeeper for service registration and discovery.

If you deploy applications developed by using the open-source Spring Cloud Alibaba Nacos Discovery in EDAS, you can enjoy the advantages and capabilities of the commercial EDAS Service Registry.

The commercial EDAS Service Registry has the following advantages over Nacos, Eureka, and Consul:

- Components are shared, which saves you the costs of deploying, operating, and maintaining Nacos, Eureka, or Consul.
- The links for calling service registration and discovery are encrypted to protect your services from being discovered by unauthorized applications.
- EDAS Service Registry is tightly integrated with other EDAS components to provide you with a complete set of microservice solutions, including environment isolation, smooth connection and disconnection, and phased release.

Prerequisites

You have downloaded the latest version of Nacos Server and started Nacos Server as follows:

1. Decompress the downloaded Nacos Server package.
2. Go to the `nacos/bin` directory and start Nacos Server.
   - For Linux, UNIX, or MacOS: Run the `sh startup.sh -m standalone` command.
   - For Windows: Double-click the `startup.cmd` file to run the file.

Step 1: Obtain a demo.

eureka-service-provider and eureka-service-consumer are the two demos provided by EDAS. They are Spring Cloud applications that have been connected to Eureka for registration and discovery. You need to download them to your local device for subsequent operations.

- eureka-service-provider
- eureka-service-consumer

Step 2: Perform operations on the provider application

To deploy the original application in EDAS, you must add the project object model (pom.xml) dependency to the provider application and specify the IP address of Nacos Server.

1. Add the pom.xml dependency.

   Open the *pom.xml* file of the provider application to replace `spring-cloud-starter-netflix-eureka-client` with `spring-cloud-starter-alibaba-nacos-discovery` and set the version of Nacos Server.

   Before the replacement:

   ```
   <dependency>
     <groupId>org.springframework.cloud</groupId>
     <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
   </dependency>
   ```

   After the replacement:

   ```
   <dependency>
     <groupId>com.alibaba.cloud</groupId>
     <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
     <version>2.1.0.RELEASE</version>
   </dependency>
   ```

   > **Note**
   > - In this example, Spring Cloud Greenwich is used, corresponding to `spring-cloud-starter-alibaba-nacos-discovery` of `2.1.0.RELEASE`.
   > - If you use Spring Cloud Finchley, the version of `spring-cloud-starter-alibaba-nacos-discovery` is `2.0.0.RELEASE`.
   > - If you use Spring Cloud Edgware, the version of `spring-cloud-starter-alibaba-nacos-discovery` is `1.5.0.RELEASE`.

2. Specify the IP address of Nacos Server.

Open `application.properties` in `src\main\resources` to specify the IP address of Nacos Server.

Before the modification:

```
spring.application.name=service-provider
server.port=18081
eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8761/eureka/
```

After the modification:

```
spring.application.name=service-provider
server.port=18081
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

Where, `127.0.0.1` is the IP address of Nacos Server. If your Nacos Server is deployed on another device, set the IP address to that of the corresponding device. If you have other requirements, see Reference configuration items to add the required configurations in the `application.properties` file.

3. Query the application service.
   i. Run the `main` function of `ProviderApplication` in `nacos-service-provider` to start the application.
   ii. Log on to the Nacos Server console at `http://127.0.0.1:8848/nacos`. In the left-side navigation pane, choose **Service Management > Services**. You can see `service-provider` in the list of services and query the details of the service in Details.

   > **Note** The default user name and password of the local Nacos Server console are `nacos`.

Step 3: Perform operations on the consumer application

To deploy the original application in EDAS, you must add the pom.xml dependency to the consumer application and specify the IP address of Nacos Server.

1. Add the pom.xml dependency.

   Open the `pom.xml` file of the consumer application to replace `spring-cloud-starter-netflix-eureka-server` with `spring-cloud-starter-alibaba-nacos-discovery` and set the version of Nacos Server.

   Before the replacement:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

After the replacement:

```
<dependency>
   <groupId>com.alibaba.cloud</groupId>
   <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
   <version>2.1.0.RELEASE</version>
</dependency>
```

> **Note**
>
> - In this example, Spring Cloud Greenwich is used, corresponding to `spring-cloud-starter-alibaba-nacos-discovery` of `2.1.0.RELEASE`.
> - If you use Spring Cloud Finchley, the version of `spring-cloud-starter-alibaba-nacos-discovery` is `2.0.0.RELEASE`.
> - If you use Spring Cloud Edgware, the version of `spring-cloud-starter-alibaba-nacos-discovery` is `1.5.0.RELEASE`.

2. Modify the settings.

   Open `application.properties` in `src\main\resources` to specify the IP address of Nacos Server.

   Before the modification:

```
spring.application.name=service-consumer
server.port=18082
eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8761/eureka/
```

   After the modification:

```
spring.application.name=service-consumer
server.port=18082
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

Where, `127.0.0.1` is the IP address of Nacos Server. If your Nacos Server is deployed on another device, set the IP address to that of the corresponding device. If you have other requirements, see Reference configuration items to add the required configurations in the `application.properties` file.

3. Query the application service.
   i. Run `ConsumerApplication.java` in `eureka-service-provider` to start the application.
   ii. Log on to the Nacos Server console at `http://127.0.0.1:8848/nacos`. In the left-side navigation pane, choose **Service Management** > **Services**. You can see `service-consumer` in the list of services and query the details of the service in Details.

> **Note** The default user name and password of the local Nacos Server console are `nacos`.

Step 4: View the call result.

Test the result of calling the provider's service by the consumer on the local device. Start the service, and run `IP + port / echo-rest / user-defined variable` or `IP + port / echo-feign / user-defined variable` to view the call result.

- For Linux, UNIX, or MacOS, run `curl http://127.0.0.1:18082/echo-rest/{user-defined variable}` or `curl http://127.0.0.1:18082/echo-feign/{user-defined variable}`.
- For Windows, enter `http://127.0.0.1:18082/echo-rest/{user-defined variable}` or `http://127.0.0.1:18082/echo-feign/{user-defined variable}` in the browser.

Step 5: Deploy the application to EDAS.

1. In the *pom.xml* file of the application, add the following configuration, and then run the **mvn clean package** command to compile the local program into an executable JAR package:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

2. See Deploy Java applications in ECS clusters to deploy the two applications whose dependency configurations are modified in Step 2: Perform operations on the provider application and Step 3: Perform operations on the consumer application to EDAS.

   > **Notice** The preceding applications are deployed using JAR packages. Therefore, **Application Runtime Environment** must be set to Standard Java application runtime environment.

   When you deploy the applications to EDAS, EDAS Service Registry automatically sets the IP address, port number, and other information such as namespace, AccessKey ID, AccessKey secret, and context-path of Nacos Server with a high priority. No additional configuration is required. You can retain or delete the original configurations.

Step 6: Verify the result

1. Deploy Java applications in ECS clusters for the consumer application to go to the homepage of the application.
2. Initiate a request on the homepage of the application. Then, log on to the EDAS console and go to the Application Details page of the consumer.
3. In the left-side navigation pane, choose **Application Monitoring > Overview** to overview the service call data. If call data is detected, the service call is successful.

Reference configuration items

Configuration item	Key	Default value	Description

Configuration item	Key	Default value	Description
IP Addresses	spring.cloud.nacos.discovery.server-addr	None	The IP address and port number of the server that Nacos Server listens to.
Service Name	spring.cloud.nacos.discovery.service	${spring.application.name}	The name of the current service.
Network Interface Name	spring.cloud.nacos.discovery.network-interface	None	The registered IP address is that of the corresponding network interface card (NIC) when no IP address is configured. If this item is not configured, the IP address of the first NIC is used by default.
Registered IP Address	spring.cloud.nacos.discovery.ip	None	This IP address is of the highest priority.
Registered Port	spring.cloud.nacos.discovery.port	-1	No configuration is required by default. The system automatically detects the port.
Namespace	spring.cloud.nacos.discovery.namespace	None	One of the common use cases is the isolation of registration in different environments, for example, the isolation of the resources (such as configurations and services) in development, test, and production environments.
Metadata	spring.cloud.nacos.discovery.metadata	None	This item is configured in the Map format. You can customize metadata information related to your services as needed.
Cluster	spring.cloud.nacos.discovery.cluster-name	DEFAULT	Set this item to the name of a Nacos Server cluster.

Configuration item	Key	Default value	Description
Endpoint	spring.cloud.nacos.discovery.enpoint	UTF-8	The domain name of a service in the region. The system dynamically retrieves the endpoint through this domain name. This configuration item is not required when an application is deployed to EDAS.
Enable Ribbon Integration	ribbon.nacos.enabled	true	You do not need to modify this item in most cases.

References

For more information on Spring Cloud Alibaba Nacos Discovery, see the open-source Spring Cloud Alibaba Nacos Discovery documentation.

1.2.4. Deploy Dubbo applications to EDAS

You can host Dubbo microservice-oriented applications to Enterprise Distributed Application Service (EDAS) and then use the shared components, enterprise-class security hardening, and comprehensive microservice solutions provided by EDAS. This reduces O&M costs and improves security and development efficiency. This topic describes how to develop a sample Dubbo microservice-oriented application in the local development environment through XML configuration, and deploy it in EDAS. The sample application contains a service provider and a service consumer.

Context

By hosting Dubbo applications to EDAS, you can focus on building the logic of Dubbo applications rather than creating and maintaining the registry, configuration center, and metadata center. Additionally, you can use EDAS capabilities such as auto scaling, throttling and degradation, monitoring, and microservice governance for various management purposes. The entire hosting process is completely transparent to you. It does not require you to learn anything, or increase your development costs.

Preparations

Before you start development, be sure to complete the following tasks:

- Download Maven and set the environment variables.
- Download the latest version of Nacos Server.
- Start Nacos Server as follows:
  (Optional)
  i. Decompress the downloaded Nacos Server package.
  ii. Go to the `nacos/bin` directory and start Nacos Server as follows:

- For Linux, UNIX, or MacOS: Run the `sh startup.sh -m standalone` command.
- For Windows: Double-click the `startup.cmd` file to run the file.

Version description

EDAS supports Dubbo 2.5.x, 2.6.x, and 2.7.x. We recommended that you use 2.7.x for better service governance. This topic takes the version 2.7.3 as an example to describe how to host Dubbo applications to EDAS.

Create a service provider

Create a provider application project in the local development environment, add dependencies, configure service registration and discovery, and specify Nacos as the registry.

1. Create a Maven project and add dependencies.
   i. Create a Maven project by using an integrated development environment (IDE), such as IntelliJ IDEA or Eclipse.
   ii. Add dubbo, dubbo-registry-nacos, and nacos-client to the `pom.xml` file.

```
<dependencies>

  <dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo</artifactId>
    <version>2.7.3</version>
  </dependency>

  <dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-registry-nacos</artifactId>
    <version>2.7.3</version>
  </dependency>

  <dependency>
    <groupId>com.alibaba.nacos</groupId>
    <artifactId>nacos-client</artifactId>
    <version>1.1.1</version>
  </dependency>
</dependencies>
```

2. Develop a Dubbo service provider. All services in Dubbo are provided as interfaces.
   i. Create a package named `com.alibaba.edas` in `src/main/java`.

ii. Create an interface named `IHelloService` that contains a `SayHello` method in `com.alibaba.edas`.

```
package com.alibaba.edas;

public interface IHelloService {
   String sayHello(String str);
}
```

iii. Create a class named `IHelloServiceImpl` in `com.alibaba.edas` to implement the interface.

```
package com.alibaba.edas;

public class IHelloServiceImpl implements IHelloService {
   public String sayHello(String str) {
      return "hello " + str;
   }
}
```

3. Configure the Dubbo service.

i. Create a file named `provider.xml` in `src/main/resources` and open the file.

ii. In `provider.xml`, add Spring-related XML namespace (xmlns) and XML schema instance (xmlns:xsi), as well as the Dubbo-related XML namespace (xmlns:dubbo) and XML schema instance (xsi:schemaLocation).

```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
xmlns="http://www.springframework.org/schema/beans"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd
">
```

iii. In `provider.xml`, expose the interface and implementation class as a Dubbo service.

```
<dubbo:application name="demo-provider"/>

<dubbo:protocol name="dubbo" port="28082"/>

<dubbo:service interface="com.alibaba.edas.IHelloService" ref="helloService"/>

<bean id="helloService" class="com.alibaba.edas.IHelloServiceImpl"/>
```

iv. In `provider.xml`, specify Nacos Server that starts locally as the registry.

```
<dubbo:registry address="nacos://127.0.0.1:8848" />
```

- `127.0.0.1` is the IP address of Nacos Server. If your Nacos Server is deployed on another machine, change the IP address to the corresponding one. When an application is deployed in EDAS, the registry address will be replaced with the address of the registry in EDAS. You do not need to make any changes.
- `8848` is the port number of Nacos Server, which cannot be changed.

4. Start the service.

i. Create the class Provider in `com.alibaba.edas` and load Spring context to the main function of Provider based on the following code to expose the configured Dubbo service.

```
package com.alibaba.edas;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Provider {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new Str
ing[] {"provider.xml"});
        context.start();
        System.in.read();
    }
}
```

ii. Execute the main function of Provider to start the service.

5. Log on to the Nacos console at `http://127.0.0.1:8848`. In the left-side navigation pane, click **Services** to view the list of providers. You can see that `com.alibaba.edas.IHelloService` is available in the list of providers. In addition, you can query **Service Group** and **Provider IP** of the service.

Create a service consumer

Create a consumer application project in the local development environment, add dependencies, and add the configuration to subscribe to the Dubbo service.

1. Create a Maven project and add dependencies.
   i. Create a Maven project by using an integrated development environment (IDE), such as IntelliJ IDEA or Eclipse.
   ii. Add dubbo, dubbo-registry-nacos, and nacos-client to the `pom.xml` file.

```
<dependencies>

  <dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo</artifactId>
    <version>2.7.3</version>
  </dependency>

  <dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-registry-nacos</artifactId>
    <version>2.7.3</version>
  </dependency>

  <dependency>
    <groupId>com.alibaba.nacos</groupId>
    <artifactId>nacos-client</artifactId>
    <version>1.1.1</version>
  </dependency>
</dependencies>
```

2. Develop a Dubbo service provider. All services in Dubbo are provided as interfaces.
   i. Create a package named `com.alibaba.edas` in `src/main/java`.

ii. Create an interface named `IHelloService` that contains a `SayHello` method in `com.alibaba.edas`.

> **Note** Generally, an interface is defined in an independent module. The provider and consumer reference the same module through Maven dependencies. In this topic, two identical interfaces are created for the provider and consumer for ease of description. However, we do not recommend this procedure in actual use.

```
package com.alibaba.edas;

public interface IHelloService {
    String sayHello(String str);
}
```

3. Configure the Dubbo service.

i. Create a file named `consumer.xml` in `src/main/resources` and open the file.

ii. In `consumer.xml`, add the Spring-related XML namespace (xmlns) and XML schema instance (xmlns:xsi), as well as the Dubbo-related XML namespace (xmlns:dubbo) and XML schema instance (xsi:schemaLocation).

```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
xmlns="http://www.springframework.org/schema/beans"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd
">
```

iii. Add the following configuration to `consumer.xml` to subscribe to the Dubbo service:

```
<dubbo:application name="demo-consumer"/>

<dubbo:registry address="nacos://127.0.0.1:8848"/>

<dubbo:reference id="helloService" interface="com.alibaba.edas.IHelloService"/>
```

4. Start and verify the Dubbo service.

i. Create the class Consumer in `com.alibaba.edas` and load Spring context to the main function of Consumer based on the following code to subscribe to and consume the Dubbo service:

```
package com.alibaba.edas;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.concurrent.TimeUnit;

public class Consumer {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new
String[] {"consumer.xml"});
        context.start();
        while (true) {
            try {
                TimeUnit.SECONDS.sleep(5);
                IHelloService demoService = (IHelloService)context.getBean("helloService");
                String result = demoService.sayHello("world");
                System.out.println(result);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

ii. Execute the main function of Consumer to start the Dubbo service.

5. Verify the creation result.
   After the Dubbo service is started, the console outputs `hello world` continuously, indicating successful service consumption.

   Log on to the Nacos console at `http://127.0.0.1:8848`. In the left-side navigation pane, click **Services**. On the **Services** page, select **Callers**.

   You can see that `com.alibaba.edas.IHelloService` is available in the list. In addition, you can query **Service Group** and **Caller IP** of the service.

Deploy the application to EDAS

You can deploy the application that uses local Nacos as the registry directly to EDAS without making any changes. This registry will be automatically replaced with the registry in EDAS.

Based on your actual needs, you can choose the cluster type (the ECS cluster or Container Service Kubernetes cluster) and deployment method (console or tools). For more information, see Deploy web applications in ECS clusters and Deploy applications in Container Service Kubernetes clusters by using images.

If you use the console for deployment, follow these steps in your local application before deploying it:

1. Add the following configuration of the packaging plug-in to the `pom.xml` file.
   - Provider

```
<build>
 <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
          <configuration>
            <classifier>spring-boot</classifier>
            <mainClass>com.alibaba.edas.Provider</mainClass>
          </configuration>
        </execution>
      </executions>
    </plugin>
 </plugins>
</build>
```

   - Consumer

```
<build>
 <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
          <configuration>
            <classifier>spring-boot</classifier>
            <mainClass>com.alibaba.edas.Consumer</mainClass>
          </configuration>
        </execution>
      </executions>
    </plugin>
 </plugins>
</build>
```

2. Run mvn clean package to package your local program into a JAR file.

After deploying the Dubbo microservice application to EDAS, you can use EDAS for microservice governance.

1.2.5. Deploy multi-language microservice-oriented applications

With the rapid development of languages such as Python and Node.js, more and more multi-language microservice-oriented applications have been developed. Enterprise Distributed Application Service (EDAS) supports the deployment of multi-language microservice-oriented applications through a service mesh, and provides service governance capabilities such as application hosting, service discovery, distributed tracing, and load balancing. This topic describes how to use EDAS to deploy an application that consists of microservice-oriented applications written in different languages by using an example.

Context

Applications have evolved from the original monolithic architecture to the current microservice architecture, which brings convenience and greatly increases the complexity of service deployment and O&M. Microservices can be developed in any language. After multi-language services are deployed, two methods can be used to provide capabilities such as distributed tracing, service discovery, and load balancing for an application that consists of microservices written in different languages: multi-language SDKs and service meshes. SDKs are invasive to applications, while service meshes are non-invasive and can also provide capabilities such as service discovery, load balancing, and distributed tracing. Therefore, EDAS uses service mesh to supports multiple languages.

A service mesh is an infrastructure that is used to implement communication between services. It is responsible for reliably delivering requests in the complex service topologies of modern cloud-native applications. Generally, a service mesh integrates a group of lightweight network agents with applications, without perceiving the applications.

Value and access cost of service mesh

- Value

  Currently, most services are deployed on multiple instances, which naturally require service discovery, load balancing, and distributed tracing. When deploying an application, you need to enter the name and port number of the service according to the code. The EDAS service mesh automatically registers services based on this information. When you use `http://service name:service port` to initiate an access request, the service mesh parses the service name from the request to complete service discovery, load balancing, and distributed tracing.

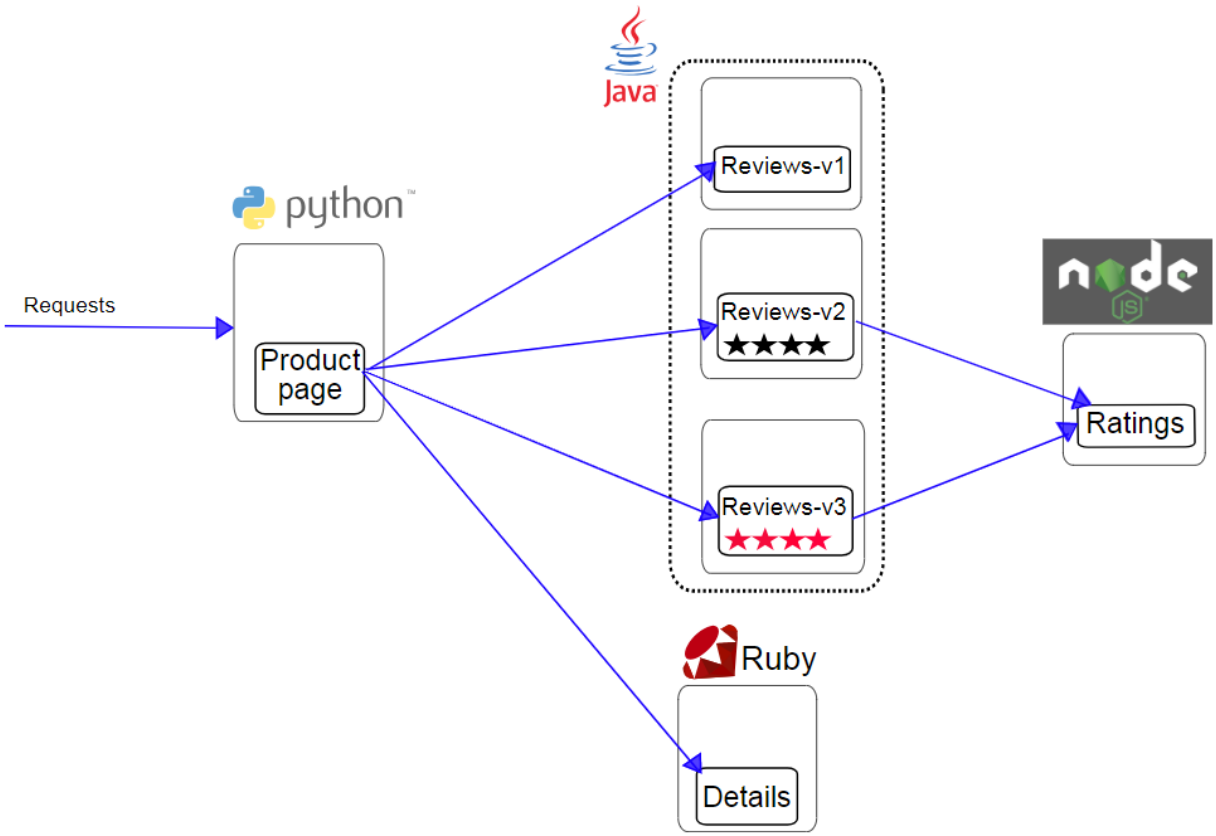
- Access cost

  Application A provides the test service. Generally, we access the service by using the domain name or IP address, for example, `http://test.com:8080/` or `http://xx.xx.xx.xx:8080`. After the service mesh is used, the instance where a service is deployed can be abstracted into a service. The following uses the test service as an example. Assume that the name and port number of the service are `my-test-service` and `8080`. In the service code, change the call syntax to `http://service name:service port`, such as `http://my-test-service:8080`. Then, enable service mesh when deploying my-test-service in image mode, and set the service name (*my-test-service*) and the port number (*9080*) to complete the access.

Example

BookInfo is a sample application that simulates a category of online bookstores and displays information about a book. The application page displays the description of a book, the details of the book (such as the ISBN and page number), and some reviews on the book.

BookInfo is a heterogeneous application that comprises several microservice-oriented applications written in different languages. These microservices constitute a sample of a representative service mesh: It consists of multiple services and languages. The Reviews service has multiple versions. The microservice architecture is as follows:



The Bookinfo application contains four independent services:

- Productpage: a Python service that calls the Details and Reviews services to generate a page. The Productpage also provides the sign-in and sign-out features.
- Details: a Ruby service that contains book information.
- Reviews: a Java service that contains reviews on the book. It also calls the Ratings service.
- Ratings: a Node.js service that contains rating information formed by book reviews. Three versions are available:
  - Version v1 does not call the Ratings service.
  - Version v2 calls the Ratings service and displays each rating as 1 to 5 black stars.
  - Version v3 calls the Ratings service and displays each rating as 1 to 5 red stars.

Prerequisites

Before deploying a multi-language microservice-oriented application in EDAS, complete the following tasks:

- Create an image of the sample application and upload it to the Alibaba Cloud image repository.

  Address for downloading the sample application: BookInfo Sample.

- Import a user-created Kubernetes cluster.
  - When creating a cluster, you must enable the **Internet access** feature, that is, checking **Use EIP Exposed API Server**.
  - Make sure that the Kubernetes version is 1.8.4 and no service mesh components are installed in the cluster.

> **Note** This topic describes how to deploy the BookInfo application as an example. Actually, you need to deploy your own application, which may be a microservice architecture. Therefore, you need to plan and develop your services as follows before deploying your application:
>
> - To deploy multiple services, ensure that the service name of each service is unique. This is because service names must be unique in the same namespace of EDAS to ensure that they can be called by other services.
> - If there are call relationships between multiple services you deployed, modify the call code in the following format for the caller service: `http://<service name of the provider>:<service port of the provider>` .

Step 1: Install service mesh for the Container Service Kubernetes cluster

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Resource Management** > **Clusters**.
3. On the **Clusters** page, select the **region** and **namespace** for the Container Service Kubernetes cluster, click the **Container Service K8s Cluster** tab, and then click the name of the Container Service Kubernetes cluster that you imported.
4. At the bottom of the Cluster Details page, click **Installing a service grid** in the **Service grid** section.
5. In the dialog box that appears, click **OK**.

   The dialog box displays **In Execution**, and then disappears. **In service grid installation** appears on the top of the Cluster Details page. Wait about 1 minute. When **In service grid installation** disappears, the installation is completed.
6. Click > on the right of the **Service grid** section to expand the section and view **Component version**, **Component health**, and **Tracking sample rate**.

Step 2: Deploy an application

You need to deploy the services in the sample scenario to EDAS as applications. The following describes how to deploy a single service.

> **Note** Currently, multi-language applications can only be deployed in image mode.

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management** > **Applications**. On the **Applications** page, click **Create Application** in the upper-right corner.
3. On the **Application Information** page, set the parameters of the application. Then, click **Next**.

   Basic parameters:

   - **Namespace**: Select a region from the left-side drop-down list. Select a namespace from the right-side drop-down list. If no namespace is selected, **Default** is selected.
   - **Cluster Type**: Select **Container Service K8S Cluster** from the left-side drop-down list and select a specific cluster from the right-side drop-down list.

- **K8S Namespace**: Internal system objects are allocated to different namespaces to form logically isolated projects, groups, or user groups. In this way, different groups can share resources of the whole cluster while being managed separately.
  - **default**: When an object is not set with a namespace, default is used.
  - **kube-system**: The namespace used by objects that are created by the system.
  - **kube-public**: The namespace that is automatically created by the system. It can be read by all users, including users that are not authenticated.
  - **istio-system**: The namespace that is automatically created by the system after service mesh is deployed.
- **Application Name**: Enter the name of application.
- **Application Description**: Enter the basic information of the application.

4. On the **Application Configuration** page, set **Deployment Method** to **Image**, and select the service image that you uploaded.
5. Set pods.

   Pods are the smallest units for deploying an application. An application can contain multiple pods. On a Server Load Balancer (SLB) instance, a request is randomly allocated to a pod for processing.

   i. Set **Pods**.

   When a pod fails to run or encounters a fault, it can automatically restart or services on the pod seamlessly fail over to other pods, ensuring a high availability for applications. For stateful applications that use persistent storage, instance data is retained when the applications are redeployed. For stateless applications, instance data is not retained when the applications are redeployed. You can set Pods to a maximum value of 50.

   ii. Set **Single Pod Resource Quota**.

   No quota is set by default. Therefore, both the CPU Cores and Memory values of a single pod are 0. To set the quota, enter a number.
6. (Optional)**Startup Command**, **Environment Variables**, **Persistent Storage**, **Local Storage**, and **Application Life Cycle Management** are optional. For more information, see the parameter description in Deploy an application (applicable to Container Service Kubernetes clusters).
7. Set service mesh.

   Service mesh parameters:

   - **Service grid**: It enables service mesh.
   - **Service name**: The service name provided by the application, which must be consistent with the service name in the application code to ensure that the service can be successfully registered and called. The **service names** of the four services in this demo are *Productpage*, *Details*, *Ratings*, and *Reviews*. To deploy your own service, enter the service name in the service code.
   - **Service Port**: The service port number provided by the application, which must be consistent with that in the application code to ensure that the service can be successfully registered and called. The **service port** numbers of the four services in this demo are *9080*. To deploy your own service, enter the service port number in the service code.
8. Then, click **Create**.

Creating an application may take up to several minutes. During the creation process, you can track the creation process based on the change record.

After the application is created in the Container Service Kubernetes cluster, the application is deployed. After the application is created, return to the Application Details page. If the pod status in the instance deployment information is **Running**, the application is successfully deployed.

Optional. Step 3: Enable access from the Internet

If a service needs to be accessed from the Internet, you must enable and set Internet access. In this sample, you need to set Internet for the main service Productpage.

1. In the **Application Settings** section of the Application Details page, enable **Public access**.
2. Set **Public access path**.

   Internet access path parameters:

   - **Service Name** is set during deployment and cannot be modified.
   - **Service Port** is also set during deployment and cannot be modified. In this sample, all the port numbers are **9080** and cannot be modified.
   - **Public IP** and **Public network ports** are automatically allocated by EDAS for a Container Service Kubernetes cluster through SLB when service mesh is installed for the cluster. They cannot be modified.
   - **Public access path**: When the service mesh is installed, the system allocates a public IP address and a port number for the cluster. Therefore, the services deployed in the cluster must be differentiated by paths. In this sample, only the main service needs to be accessed from the Internet. Therefore, the paths for the access, sign-in, and sign-out services of Productpage can be set as follows:
     - Access: /productpage
     - Sign-in: /login
     - Sign-out: /logout

     > **Note** To deploy your own service, enter the actual access path in the service code.

Verification

After deploying the four applications, you can access the main service. The page displays a book description, details (such as ISBN and number of pages), and reviews on the book. You can also log on to and log out of the page.

1. In the address bar of the browser, enter *http://<Internet IP address>:<Internet port number ><main service path>* such as *http://xxx.xxx.xxx.xxx:80/productpage*, and then click Enter.

   The page is accessible, and the **Book Details** and **Book Reviews** sections are properly displayed. This indicates that the main service Productpage, subservice Details, and subservice Reviews are normal.
2. Click **Sign in**. In the dialog box, enter the user name and password *admin*, and then click **Sign in**. The sign-in is successful.
3. After sign-in, click **Sign out**. The page can exit properly.

What to do next

After deploying a service, you can monitor the running of the service in the EDAS console. When an error occurs, you can use logs for diagnosis.

- Monitoring: You can use Tracing Analysis integrated into EDAS to monitor applications and view trace information on the Application Details page. For more information on how to use specific features such as Application Overview, Application Details, and API Calls), see Tracing Analysis documentation.
- Logs: You can view standard logs and service logs of Container Service by using the log management feature of EDAS. For more information, see Log management.

1.3. Application development

1.3.1. Use Spring Cloud to develop applications

1.3.1.1. Spring Cloud overview

Enterprise Distributed Application Service (EDAS) supports the native Spring Cloud microservice framework. You can deploy Spring Cloud applications to EDAS simply by adding dependencies and modifying configurations. Then you can use EDAS functions, such as enterprise-level application hosting, application governance, monitoring and alerting, and application diagnosis. This ensures zero code intrusion.

Introduction

Spring Cloud provides a series of standards and specifications to simplify application development. These standards and specifications cover service discovery, load balancing, circuit breakers, configuration management, message event triggering, and message bus. In addition, Spring Cloud provides implementation components for gateways, distributed tracing, security, distributed job scheduling, and distributed job coordination.

Currently, the most popular Spring Cloud implementation components in the industry include Spring Cloud Netflix, Spring Cloud Consul, Spring Cloud Gateway, and Spring Cloud Sleuth. Spring Cloud Alibaba, an open-source middleware recently developed by Alibaba, is also a very popular implementation component in the industry.

You can directly deploy and manage applications developed by using Spring Cloud components, such as Spring Cloud Netflix and Spring Cloud Consul, in EDAS. In addition, you can directly use the advanced monitoring functions provided by EDAS without modifying any code, enabling monitoring functions such as distributed tracing, monitoring and alerting, and application diagnosis.

To use more service governance functions in EDAS to manage your Spring Cloud applications, you need to replace your Spring Cloud components with those in Spring Cloud Alibaba or add Spring Cloud Alibaba components.

Compatibility

Currently, Enterprise Distributed Application Service (EDAS) supports Spring Cloud Greenwich, Spring Cloud Finchley, and Spring Cloud Edgware.

The following table lists the Spring Cloud features, open-source components, and compatibility with EDAS.

Spring Cloud feature		Open-source component	Compatibility with EDAS
Common features	Service registration and discovery	• Netflix Eureka • Consul Discovery	Compatible, with an equivalent component
	Load balancing	Netflix Ribbon	Compatible
	Service calls	• Feign • RestTemplate	Compatible
Configuration management		• Config Server • Consul Config	Compatible, with an equivalent component
Gateways		• Spring Cloud Gateway • Netflix Zuul	Compatible
Distributed tracing		Spring Cloud Sleuth	Compatible, with an equivalent component
Message-driven application development: Spring Cloud Stream		• RabbitMQ binder • Kafka binder	Compatible, with an equivalent component
Spring Cloud Bus		• RabbitMQ • Kafka	Compatible, with an equivalent component
Security		Spring Cloud Security	Compatible
Distributed job scheduling		Spring Cloud Task	Compatible
Distributed coordination		Spring Cloud Cluster	Compatible

Version mapping

The following table describes the mapping among Spring Cloud, Spring Boot, Spring Cloud Alibaba, and commercially available EDAS components.

Spring Cloud	Spring Boot	Spring Cloud Alibaba	Commercially available EDAS components • Nacos Registry • Nacos Config
Greenwich	2.1.x	2.1.1.RELEASE	2.1.1.RELEASE

Spring Cloud	Spring Boot	Spring Cloud Alibaba	Commercially available EDAS components • Nacos Registry • Nacos Config
Finchley	2.0.x	2.0.1.RELEASE	2.0.1.RELEASE
Edgware	1.5.x	1.5.1.RELEASE	1.5.1.RELEASE

1.3.1.2. Implement service registration and discovery

You can add basic dependencies and configurations to your Spring Cloud applications and then deploy them to Enterprise Distributed Application Service (EDAS) and use EDAS Service Registry to discover services.

For more information, see Deploy Spring Cloud applications to EDAS.

1.3.1.3. Implement load balancing

Spring Cloud uses the Ribbon component for load balancing. Ribbon mainly provides consumer-side software load balancing algorithms. In Spring Cloud, load balancing is implemented for RestTemplate and FeignClient through Ribbon.

Spring Cloud Alibaba ANS integrates the functions of Ribbon and AnsServerList implements the com.netflix.loadbalancer.ServerList interface provided by Ribbon.

This interface is generic and other similar service discovery components, such as Nacos, Eureka, Consul, and ZooKeeper, implement ServerList interfaces such as NacosServerList, DomainExtractingServerList, ConsulServerList, and ZookeeperServerList.

Implementing the com.netflix.loadbalancer.ServerList interface is equivalent to complying with the load balancing specifications of Spring Cloud. These specifications are generic. This means that no code modification is required to change the service discovery solution from Eureka, Consul, or ZooKeeper to Spring Cloud Alibaba, including RestTemplate, FeignClient, and the outdated AsyncRestTemplate.

The following describes how to implement load balancing for RestTemplate and FeignClient in your application.

This topic describes key information for developing applications locally. For more information about Spring Cloud, download service-provider and service-consumer.

The methods to implement load balancing for RestTemplate and FeignClient are different and thus described below separately.

RestTemplate

RestTemplate is a client provided by Spring Cloud to access RESTful services. It provides multiple ways to conveniently access remote HTTP services, greatly improving the writing efficiency of client-side code.

To use the load balancing feature of RestTemplate, you need to modify the code in your application based on the following example.

```
public class MyApp {
   // Inject the RestTemplate you built with the @LoadBalanced annotation.
   // This annotation adds the LoadBalancerInterceptor to RestTemplate.
   // Internally, LoadBalancerInterceptor uses the implementation class RibbonLoadBalancerClient of t
he LoadBalancerClient interface for load balancing.
   @Autowired
   private RestTemplate restTemplate;

   @LoadBalanced // Modify the built RestTemplate with this annotation to enable its load balancing f
unction.
   @Bean
   public RestTemplate restTemplate() {
      return new RestTemplate();
   }
   // RestTemplate internally calls services in load balancing mode.
   public void doSomething() {
      Foo foo = restTemplate.getForObject("http://service-provider/query", Foo.class);
      doWithFoo(foo);
   }
   ...
}
```

Feign

Feign is an HTTP client written in Java to simplify RESTful API calls. To enable load balancing on Feign, perform the following steps:

1. To enable load balancing on Feign, add the Ribbon dependency.

   ```
   <dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
   <version>{version}</version>
   </dependency>
   ```

2. Use @EnableFeignClients and @FeignClient to initiate a load balancing request.
   i. Use @EnableFeignClients to enable the functions of Feign.

      ```
      @SpringBootApplication
      @EnableFeignClients // Enable the functions of Feign.
      public class MyApplication {
      ...
      }
      ```

ii. Use @FeignClient to build FeignClient.

```
@FeignClient(name = "service-provider")
public interface EchoService {
@RequestMapping(value = "/echo/{str}", method = RequestMethod.GET)
String echo(@PathVariable("str") String str);
}
```

iii. Inject EchoService and call the echo method.

Calling the echo method is equivalent to initiating an HTTP request.

```
public class MyService {
@Autowired // Inject the EchoService you built with @FeignClient.
private EchoService echoService;
public void doSomething() {
  // This is equivalent to initiating an http://service-provider/echo/test request.
  echoService.echo("test");
}
...
}
```

Verify the result

After `service-consumer` and multiple `service-providers` are started, access the URL provided by the `service-consumer` to check whether load balancing is implemented.

- RestTemplate

  Access `/echo-rest/rest-test` multiple times and check whether the request is forwarded to different instances.

- Feign

  Access `/echo-feign/feign-test` multiple times and check whether the request is forwarded to different instances.

1.3.1.4. Implement configuration management

When you develop Spring Cloud applications, you can use Nacos (https://nacos.io) to manage application configurations locally. Nacos is an open-source version of Application Configuration Management (ACM). After you deploy an application to Enterprise Distributed Application Service (EDAS), you can manage and push application configurations by using the EDAS-integrated ACM.

You can follow the instructions in this topic to develop an application sample from scratch and use Spring Cloud Alibaba Nacos Config for configuration management. You can also directly download the demo nacos-config-example.

Note Spring Cloud Alibaba Nacos Config integrates Nacos and Spring Cloud frameworks and supports configuration injection specifications of Spring Cloud.

Preparations

Before you start development, make sure you have completed the following operations:

- Download Maven and set the environment variables.
- Download the latest version of Nacos Server.
- Start Nacos Server.
  i. Decompress the downloaded Nacos Server package.
  ii. Go to the `nacos/bin` directory and start Nacos Server:
    - For Linux, UNIX, or MacOS: Run the `sh startup.sh -m standalone` command.
    - For Windows: Double-click the `startup.cmd` file to run the file.
- Create a configuration in the local Nacos Server console.
  i. Log on to the local Nacos Server console, with nacos as both the default user name and password.
  ii. In the left-side navigation pane, click **Configurations**. On the **Configurations** page, click the Create Configuration icon + in the upper-right corner.
  iii. On the **Create Configuration** page, enter the following information and click **Publish**.
    - **Data ID**: nacos-config-example.properties
    - **Group**: DEFAULT_GROUP
    - **Configuration Body**: test.name=nacos-config-test

Implement configuration management by using Nacos Config

1. Create a Maven project named `nacos-config-example`.
2. Add dependencies to the `pom.xml` file.

   The following example uses Spring Boot 2.1.4.RELEASE and Spring Cloud Greenwich.SR1.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.4.RELEASE</version>
    <relativePath/>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
        <version>2.1.1.RELEASE</version>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Greenwich.SR1</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

This example uses Spring Cloud Greenwich, and the corresponding Spring Cloud Alibaba version is 2.1.1.RELEASE.

- If you are using Spring Cloud Finchley, the corresponding Spring Cloud Alibaba version is 2.0.1.RELEASE.
- If you are using Spring Cloud Edgware, the corresponding Spring Cloud Alibaba version is 1.5.1.RELEASE.

> **Note** The Spring Cloud Edgware release has reached the end of life. Therefore, we recommend that you do not use this release to develop applications.

3. Create a package named `com.aliware.edas` in `src/main/java`.
4. In the `com.aliware.edas` package, create a startup class named `NacosConfigExampleApplication` for `nacos-config-example`.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

5. In the `com.aliware.edas` package, create a simple controller named `EchoController`, automatically inject a property named `userName`, and add the `@Value` annotation to obtain the value of `test.name` from the configuration.

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.cloud.context.config.annotation.RefreshScope;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RefreshScope
public class EchoController {

    @Value("${test.name}")
    private String userName;

    @RequestMapping(value = "/")
    public String echo() {
        return userName;
    }
}
```

6. In the `src\main\resources` path, create a configuration file named `bootstrap.properties`, and add the following configuration in `bootstrap.properties` to specify the IP address of Nacos Server.

   `127.0.0.1:8848` is the IP address of Nacos Server, and `18081` is the service port.

   If your Nacos Server is deployed on another machine, change the IP address and service port to the actual ones. If you have other requirements, add configuration items in the `bootstrap.properties` file. For more information, see Reference configuration items.

   ```
   spring.application.name=nacos-config-example
   server.port=18081
   spring.cloud.nacos.config.server-addr=127.0.0.1:8848
   ```

7. Run the main function in `NacosConfigExampleApplication` to start the application.

Verify the result locally

Visit http://127.0.0.1:18081 in your browser. `nacos-config-test` is returned, which is the value you configured when you created a configuration in the local Nacos Server console, specifically, the value of `test.name`.

Deploy applications to EDAS

After you develop and test your application in the local development environment, you can package and deploy it to EDAS. You can choose to deploy your Spring Cloud application to an Elastic Compute Service (ECS) cluster or a Container Service Kubernetes cluster based on your needs. For more information about how to deploy an application, see Deploy web applications in ECS clusters and Deploy applications in Container Service Kubernetes clusters by using images.

ACM integrated in EDAS is the general availability (GA) version of Nacos. When you deploy an application to EDAS, EDAS sets the IP address and service port of Nacos Server, namespace, AccessKey ID, AccessKey secret, and context path by using a method with a higher priority.

Before you deploy an application, you need to add the same configuration on the Configuration Management page of the EDAS console as that on the local Nacos Server. Specifically, you can perform the following operations:

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management > Configuration Management**.
3. On the **Configurations** page, select a **region** and **namespace**, and click the Create Configuration icon + on the right side of the page.
4. On the **Create Configuration** page, set **Data ID**, **Group**, and **Configuration Content**, and click **Publish**.
   - **Data ID**: nacos-config-example.properties
   - **Group**: DEFAULT_GROUP
   - **Configuration Content**: test.name=nacos-config-test

We recommend that you use the console to deploy your application for the first time. If you choose to use JAR packages for initial deployment, be sure to select **Standard Java Application Runtime Environment** for **Application Runtime Environment** when you create the application.

Verify the result

1. After the deployment is completed, check the log to see whether the application is started.
2. Run the `curl http://<application instance IP address>:<service port>` command, for example, `curl http://192.168.0.34:8080`, to check whether the configuration content `nacos-config-test` is returned.
3. Change the previous configuration content to `nacos-config-test2` in the EDAS console, and then run the `curl http://<application instance IP address>:<service port>` command, for example, `curl http://192.168.0.34:8080`, to check whether the configuration content `nacos-config-test2` is returned.

Reference configuration items

If you have other requirements, see the following table and add configurations in the `bootstrap.properties` file.

Configuration item	key	Default value	Description
Server address	spring.cloud.nacos.config.server-addr	N/A	N/A

Configuration item	key	Default value	Description
Prefix of data ID	spring.cloud.nacos.config.prefix	${spring.application.name}	The prefix of the data ID.
Group	spring.cloud.nacos.config.group	DEFAULT_GROUP	
Suffix of data ID and content file format	spring.cloud.nacos.config.file-extension	properties	The suffix of the data ID, which is also the file format of the configuration body. The default value is properties, and yaml and yml are also supported.
Encoding mode of the configuration body	spring.cloud.nacos.config.encode	UTF-8	The encoding mode of the configuration.
Timeout period for retrieving the configuration	spring.cloud.nacos.config.timeout	3000	Unit: ms.
Configured namespace	spring.cloud.nacos.config.namespace		One of the common scenarios is the isolation of configurations in different environments, for example, the isolation of resources in development, test, and production environments.
Relative path	spring.cloud.nacos.config.context-path		The relative path of the server API.
Endpoint	spring.cloud.nacos.config.endpoint	UTF-8	The domain name of a service in the region. You can retrieve the server address based on this domain name.
Enable listener and auto-refresh	spring.cloud.nacos.config.refresh.enabled	true	The default value is true, and no modification is required.

For more information about configuration items, see the open-source Spring Cloud Alibaba Nacos Config.

1.3.1.5. Build service gateways

This topic describes how to use Nacos to build service gateways from scratch based on Spring Cloud Gateway and Spring Cloud Netflix Zuul.

Why do service gateways use the EDAS registry?

`Spring Cloud Alibaba Nacos Discovery` is the GA version of the open-source Nacos Server provided by the EDAS registry. It allows you to directly use the GA version of the EDAS registry.

The GA version of the EDAS registry has the following advantages over open-source Nacos, Eureka, and Consul:

- Components are shared, which saves the costs of deploying, operating, and maintaining Nacos, Eureka, or Consul.
- Links are encrypted for the calls to use service registration and discovery, which protects your services from being discovered by unauthorized applications.
- The EDAS registry is tightly integrated with other EDAS components to provide you with a complete set of microservice solutions, including environment isolation, graceful connection and disconnection, and canary release.

Preparations

- Download Maven and set the environment variables. Skip this step if you have installed Maven on your local instance.
- Download the latest version of Nacos Server. Skip this step if you have installed Nacos Server on your local instance.
- Start Nacos Server.

  Decompress the downloaded Nacos Server package and go to the nacos/bin directory.

  For Linux, UNIX, or MacOS, run `sh startup.sh -m standalone`. For Windows, run `cmd startup.cmd` or double-click `startup.cmd` to run the file.

- Demo application

  This topic describes key information for developing applications locally. For more information about Spring Cloud, download spring-cloud-gateway-nacos, spring-cloud-zuul-nacos, and nacos-service-provider.

Build service gateways based on Spring Cloud Gateway

This topic describes how to use Nacos to build a service gateway from scratch based on Spring Cloud Gateway.

1. Create a service gateway.
   i. Create a Maven project named `spring-cloud-gateway-nacos`.
   ii. Add the dependencies of Spring Boot and Spring Cloud in the `pom.xml` file.

   The following example uses Spring Boot 2.1.4.RELEASE and Spring Cloud Greenwich.SR1.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.4.RELEASE</version>
    <relativePath/>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
        <version>2.1.1.RELEASE</version>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Greenwich.SR1</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

iii. Develop a service gateway startup class named `GatewayApplication`.

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

iv. Add the following configuration in `application.yaml` to set the registry address to the address of Nacos Server.

`127.0.0.1:8848` is the address of Nacos Server. If your Nacos Server is deployed on another machine, change it to the actual one.

"routes" specifies the routing and forwarding rules of the gateway. In this example, all requests with the prefix `/provider1/` are routed to the backend service named `service-provider`.

```
  server:
port: 15012

  spring:
application:
    name: spring-cloud-gateway-nacos
cloud:
    gateway: # config the routes for gateway
      routes:
      -id: service-provider        # Forwards requests with the prefix /provider1/ to provider1.
        uri: lb://service-provider
        predicates:
        - Path=/provider1/**
        filters:
        - StripPrefix=1            # Indicates that the prefix /provider1 needs to be truncated.
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
```

v. Run the main function of the `GatewayApplication` startup class to start the gateway.

vi. Log on to the Nacos Server console at http://127.0.0.1:8848/nacos, with nacos as both the default user name and password. In the left-side navigation pane, choose **Service Management > Services**. You can see that spring-cloud-gateway-nacos is available in the service list. In addition, you can query the details of the service. This indicates that the gateway has been started and registered. Then, create a downstream service to verify the request forwarding feature of the gateway.

2. Create a service provider.

   Create a service provider application. For more information, see Host Spring Cloud applications to EDAS.

   Sample service provider:

```
@SpringBootApplication
@EnableDiscoveryClient
public class ProviderApplication {

  public static void main(String[] args) {

    SpringApplication.run(ProviderApplication, args);
  }

  @RestController
  public class EchoController {
    @RequestMapping(value = "/echo/{string}", method = RequestMethod.GET)
    public String echo(@PathVariable String string) {
      return string;
    }
  }
}
```

3. Verify the result.
   - Verify the result locally.

     Locally start the service gateway and service provider you created and access Spring Cloud Gateway to forward the request to the backend service. The result indicating a successful call is returned.

   - Verify the result in EDAS.

     Deploy your application to EDAS. For more information about how to deploy applications to EDAS, see Host Spring Cloud applications to EDAS. Then verify the result.

     The EDAS registry provides a GA version of Nacos Server. When you deploy an application to EDAS, EDAS sets the IP address and service port of Nacos Server, namespace, AccessKey ID, AccessKey secret, and context path by using a method with a higher priority. You do not need to make additional configurations. In addition, you can choose to retain or delete your original configuration.

Build service gateways based on Zuul

This topic describes how to use Nacos Server as the registry to build service gateways for applications from scratch based on Zuul.

1. Create a service gateway.
   i. Create a Maven project named `spring-cloud-zuul-nacos`.
   ii. Add the dependencies of Spring Boot, Spring Cloud, and Spring Cloud Alibaba in the `pom.xml` file.

   Add the dependencies of Spring Boot 2.1.4.RELEASE, Spring Cloud Greenwich.SR1, and Spring Cloud Alibaba 0.9.0.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.4.RELEASE</version>
  <relativePath/>
</parent>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
  </dependency>
  <dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    <version>2.1.1.RELEASE</version>
  </dependency>
</dependencies>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Greenwich.SR1</version>
      <type>pom</type>
      <scope>import</scope>
```

```
    </dependency>
  </dependencies>
</dependencyManagement>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

iii. Develop a service gateway startup class named `ZuulApplication`.

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

iv. Add the following configuration in `application.properties` to set the registry address to the address of Nacos Server.

`127.0.0.1:8848` is the address of Nacos Server. If your Nacos Server is deployed on another machine, change it to the actual one.

"routes" specifies the routing and forwarding rules of Zuul. In this example, all requests with the prefix `/provider1/` are routed to the backend service named `service-provider`.

```
spring.application.name=spring-cloud-zuul-nacos
server.port=18022

spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848

zuul.routes.opensource-provider1.path=/provider1/**
zuul.routes.opensource-provider1.serviceId=service-provider
```

v. Run the main function of `ZuulApplication` in spring-cloud-zuul-nacos to start the service.

vi. Log on to the Nacos Server console at http://127.0.0.1:8848/nacos with nacos as both the default user name and password. In the left-side navigation pane, choose Service Management > Services. You can see that spring-cloud-zuul-nacos is available in the service list. In addition, you can query the details of the service. This indicates that the gateway has been started and registered. Then, create a downstream service to verify the request forwarding feature of the gateway.

2. Create a service provider.For more information about how to create a service provider, see Host Spring Cloud applications to EDAS.The following example illustrates how to start a service provider:

```
@SpringBootApplication
@EnableDiscoveryClient
public class ProviderApplication {

    public static void main(String[] args) {

        SpringApplication.run(ProviderApplication, args);
    }

    @RestController
    public class EchoController {
        @RequestMapping(value = "/echo/{string}", method = RequestMethod.GET)
        public String echo(@PathVariable String string) {
            return string;
        }
    }
}
```

3. Verify the result.
   - Verify the result locally.

     Locally start the Zuul service gateway and service provider you created and forward the request to the backend service by using Spring Cloud Netflix Zuul. The result indicating a successful call is returned.
   - Verify the result in EDAS.

     Deploy your application to EDAS. For more information, see Host Spring Cloud applications to EDAS. Then verify the result.

     The EDAS registry provides a GA version of Nacos Server. When you deploy an application to EDAS, EDAS sets the IP address and service port of Nacos Server, namespace, AccessKey ID, AccessKey secret, and context path by using a method with a higher priority. You do not need to make additional configurations. In addition, you can choose to retain or delete your original configuration.

1.3.1.6. Implement task scheduling

EDAS integrates SchedulerX into the console as a component to implement distributed task scheduling. This topic describes how to use SchedulerX to schedule tasks in Spring Cloud applications, deploy the applications to EDAS in the *test* region, and realize task scheduling in *Simple Job Single-Server Edition* mode.

Context

SchedulerX is a distributed task scheduling product developed by Alibaba, which is accurate, highly reliable, and highly available. It allows you to run tasks on a schedule in seconds based on the Cron expression. It provides models for implementing distributed tasks, such as grid jobs.

Procedure

1. Create a Maven project named `scx-example`.
2. Take *Spring Boot 2.0.6.RELEASE* and *Spring Cloud Finchley.SR1* as an example. Add the following dependencies to the `pom.xml` file.

```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>2.0.6.RELEASE</version>
   <relativePath/>
</parent>
 <dependencies>
   <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-alicloud-schedulerx</artifactId>
      <version>0.2.1.RELEASE</version>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>
</dependencies>
<dependencyManagement>
   <dependencies>
      <dependency>
         <groupId>org.springframework.cloud</groupId>
         <artifactId>spring-cloud-dependencies</artifactId>
         <version>Finchley.SR1</version>
         <type>pom</type>
         <scope>import</scope>
      </dependency>
   </dependencies>
</dependencyManagement>
```

> **Note**
>
> - If you want to use *Spring Boot 1.x*, use *Spring Boot 1.5.x*, *Spring Cloud Edgware*, and **Spring Cloud Alibaba***0.1.1.RELEASE*.
> - *Spring Boot 1.x* will expire in **August 2019**, so we recommend that you use a later version of Spring Boot to develop applications.

3. Create the startup class `ScxApplication` for `scx-example`.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class ScxApplication {
    public static void main(String[] args) {
        SpringApplication.run(ScxApplication.class, args);
    }
}
```

4. Create a simple class `TestService` and inject the class to IoC of the test task through Spring.

```
import org.springframework.stereotype.Service;
@Service
public class TestService {
    public void test() {
        System.out.println("---------IOC Success--------");
    }
}
```

5. Create a simple class `SimpleTask` as the test task class and inject `TestService` to the class.

```
import com.alibaba.edas.schedulerx.ProcessResult;
import com.alibaba.edas.schedulerx.ScxSimpleJobContext;
import com.alibaba.edas.schedulerx.ScxSimpleJobProcessor;
import org.springframework.beans.factory.annotation.Autowired;
public class SimpleTask implements ScxSimpleJobProcessor {
    @Autowired
    private TestService testService;
    @Override
    public ProcessResult process(ScxSimpleJobContext context) {
        System.out.println("-----------Hello world---------------");
        testService.test();
        ProcessResult processResult = new ProcessResult(true);
        return processResult;
    }
}
```

6. Create a scheduled task and add a configuration.
    i. Log on to the EDAS console. In the test region, create a scheduled task group and record the group ID.

ii. In the task group that you created, configure the scheduled task as follows:
   - **Job Group**: Select the *ID of the group* you created in the *test* region.
   - **Job Processing Interface**: Enter the name of the class that implements the job interface. In this example, the value is *SimpleTask*, which is the same as the test task class in the application.
   - **Type**: Select *Simple Job Single-Server Edition*.
   - **Cron Expression**: `*0 * * * * ? *` is selected by default. It means that the task is run once every minute.
   - **Job Description**: None.
   - **Custom Parameters**: None.

iii. In the *src/main/resources* path of the local Maven project, create the `application.properties` file and add the following configuration to the file:

```
server.port=18033
# Configure the region (the **regionName** of the test region is *cn-test*) and the group ID (group-id) of the task.
spring.cloud.alicloud.scx.group-id=***
spring.cloud.alicloud.edas.namespace=cn-test
```

> **Note** In this topic, the test region is used and the test is performed in a public network environment. You can verify the deployment result both in on-premises and off-premises instances, without permission restrictions. If you want to deploy applications to other regions, for example, *China (Hangzhou)*, you need to perform the following steps in addition to creating a scheduled task and scheduling the task:
>
> i. Log on to the EDAS console. In the *China (Hangzhou)* region, create a task group and a scheduled task.
>
> ii. Access the Security Management page, and retrieve the **AccessKeyId** and **AccessKeySecret**.
>
> iii. In the `application.properties` file, configure the scheduled task.
>
> iv. In the `application.properties` file, add the **AccessKeyId** and **AccessKeySecret** of your Alibaba Cloud account.
>
> ```
> spring.cloud.alicloud.access-key=xxxxx
> spring.cloud.alicloud.secret-key=xxxxx
> ```

7. Run the main function in `ScxApplication` to start the service.

Result

Log on to the IntelliJ IDEA console and view the standard output. The following test information is printed periodically:

```
----------Hello world----------
---------IOC Success--------
```

What's next

After your application is deployed to EDAS, you can use SchedulerX to implement more task scheduling functions. For more information, see SchedulerX overview.

1.3.2. Use Dubbo to develop applications

1.3.2.1. Dubbo overview

Enterprise Distributed Application Service (EDAS) supports the Apache Dubbo microservice framework. With zero code intrusion, you can deploy Apache Dubbo microservices to EDAS simply by adding dependencies and modifying configurations. Then you have access to the features of EDAS, such as hosting of enterprise-level microservice-oriented applications, microservice governance, monitoring and alerting, and application diagnosis.

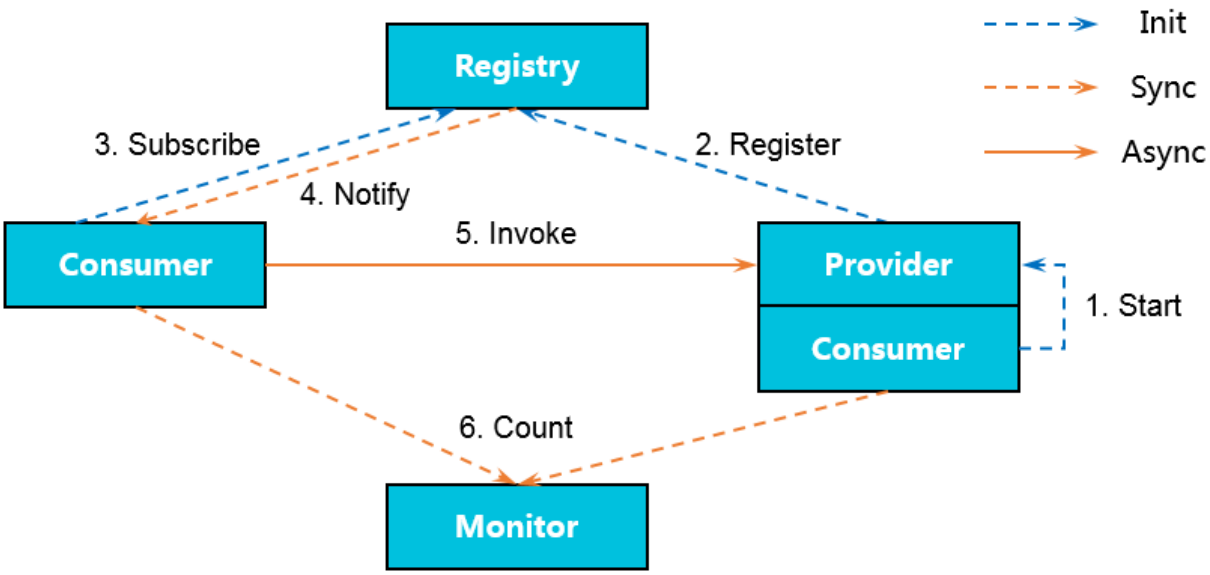
Dubbo architecture

There are two mainstream versions of open-source Apache Dubbo: 2.6.x and 2.7.x.

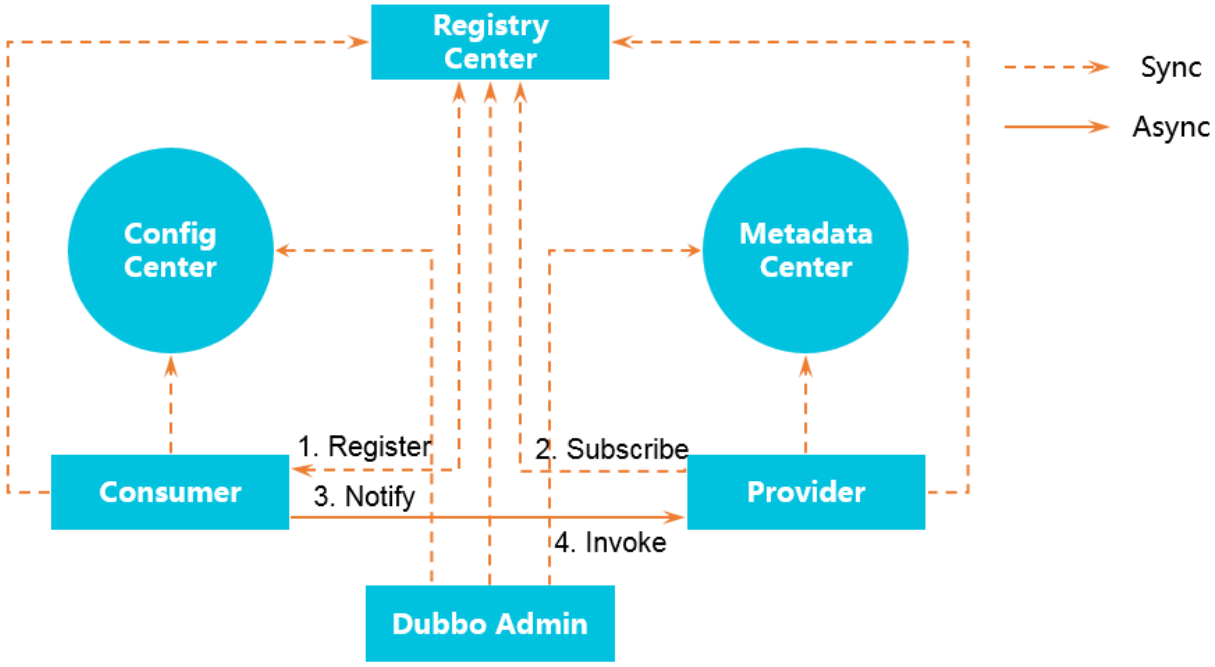
- Dubbo 2.6.x is widely used and will be maintained, but will not be upgraded with new features.
- Dubbo 2.7.x is the latest version of Apache Dubbo and will be upgraded with new features.

We recommend that you use Dubbo 2.7.x. If you are using Dubbo 2.6.x, we recommend that you migrate to Dubbo 2.7.x to use future new features.

Dubbo 2.6.x



Dubbo 2.7.x



The workflow of the Dubbo service framework is as follows:

1. During startup, the provider registers with the registry.
2. During startup, the consumer subscribes to services from the registry as needed.
3. The registry returns a list of provider addresses to the consumer. When the provider changes, the registry pushes changed data to the consumer.
4. The consumer selects a provider from the list of provider addresses based on the software load balancing algorithm.

Meaning of hosting Dubbo applications to EDAS

Hosting a Dubbo application to EDAS is hosting the registry, configuration center, and metadata center.

- Before hosting, you need to build and maintain the registry, configuration center, and metadata center. The registry is an open-source component such as ZooKeeper or Nacos. The configuration center and metadata center are included in Dubbo Admin.
- After hosting, EDAS provides Nacos (including the registry, configuration center, and metadata center) and the Dubbo service governance platform. You do not need to build or maintain these components or monitor their availability. You can use a microservice governance platform more powerful than the user-created Dubbo Admin.

Type	Open-source component	EDAS component	Hosting instruction

Type	Open-source component	EDAS component	Hosting instruction
Registry	• Nacos (recommended) • ZooKeeper (recommended) • etcd • Consul • Eureka	• Nacos (recommended) • EDAS registry	Nacos is the recommended registry. You only need to add the open-source `dubbo-nacos-registry` dependency to the application.
Configuration center	• Nacos (recommended) • ZooKeeper (recommended) • Apollo	Nacos (recommended)	Add the `dubbo-configcenter-nacos` dependency to the application.
Metadata center	• Nacos (recommended) • Redis (recommended) • ZooKeeper	Nacos (recommended)	Add the `dubbo-metadata-report-nacos` dependency to the application.

Benefits of hosting Dubbo applications to EDAS

By hosting Dubbo applications to EDAS, you only need to focus on building the logic of the Dubbo applications other than creating and maintaining the registry, configuration center, and metadata center. Also, you can take advantage of EDAS capabilities such as auto scaling, throttling, graceful service degradation, monitoring, and microservice governance for various management purposes. The entire hosting process is completely transparent to you. It does not require you to learn anything, or increase your development costs. Specific benefits of hosting are as follows:

- Costs: EDAS provides the service discovery and configuration management features, saving you from maintaining the middleware such as Eureka, ZooKeeper, and Consul.
- Deployment: EDAS provides flexible configuration of startup parameters, process visualization, graceful service connection and disconnection, and batch publishing, allowing you to configure, query, and manage your application deployment.
- Service governance: EDAS provides the service query, conditional routing, blacklist and whitelist, label-based routing, dynamic configuration, load balancing configuration, weight configuration, and centralized configuration management, allowing you to comprehensively govern your services.
- Auto scaling: EDAS provides the auto scaling feature, allowing you to dynamically scale your applications in or out based on traffic peaks and valleys.
- Throttling and degradation: EDAS provides throttling and graceful service degradation to ensure the high availability of your applications.
- Monitoring: EDAS integrates some monitoring features of Application Real-Time Monitoring Service (ARMS). In addition to instance information query, EDAS also provides advanced

monitoring features such as microservice trace query, service call topology query, and slow SQL query.

1.3.2.2. Use Spring Boot to develop Dubbo applications

Spring Boot simplifies the configuration and deployment of microservice-oriented applications. Nacos provides the service registration and discovery as well as configuration management features. Using Spring Boot and Nacos together can help you improve development efficiency. This topic describes how to use Spring Boot annotations to develop a sample Dubbo microservice-oriented application based on Nacos.

Prerequisites

Before using Spring Boot to develop microservice-oriented Dubbo applications, complete the following tasks:

- Download Maven and set the environment variables.
- Download the latest version of Nacos Server.
- Start Nacos Server.
  i. Decompress the downloaded Nacos Server package.
  ii. Go to the `nacos/bin` directory and start Nacos Server as follows:
    - For Linux, UNIX, or MacOS: Run the `sh startup.sh -m standalone` command.
    - For Windows: Double-click the `startup.cmd` file to run the file.

Sample project

You can follow the steps described in this topic to build the project. Alternatively, you can directly download the sample project used in this topic, or clone the project by running the Git command `git clone https://github.com/aliyun/alibabacloud-microservice-demo.git`.

This project contains many demos. The demo used in this topic can be found in `alibabacloud-microservice-demo/microservice-doc-demo/dubbo-samples-spring-boot`.

Create a service provider

1. Create a Maven project named `spring-boot-dubbo-provider`.
2. Add required dependencies to the `pom.xml` file.

   The following takes Spring Boot 2.0.6.RELEASE as an example.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
      <version>2.0.6.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-spring-boot-starter</artifactId>
    <version>2.7.3</version>
  </dependency>
  <dependency>
    <groupId>com.alibaba.nacos</groupId>
    <artifactId>nacos-client</artifactId>
    <version>1.1.1</version>
  </dependency>
</dependencies>
```

3. Develop a Dubbo service provider. All services in Dubbo are provided as interfaces.
   i. Create a package named `com.alibaba.edas.boot` in `src/main/java`.

ii. Create an interface named `IHelloService` that contains a `SayHello` method in `com.alibaba.edas.boot`.

```
package com.alibaba.edas.boot;
public interface IHelloService {
String sayHello(String str);
}
```

iii. Create a class named `IHelloServiceImpl` to implement the interface in `com.alibaba.edas.boot`.

```
package com.alibaba.edas.boot;
import com.alibaba.dubbo.config.annotation.Service;
@Service
public class IHelloServiceImpl implements IHelloService {
public String sayHello(String name) {
 return "Hello, " + name + " (from Dubbo with Spring Boot)";
 }
}
```

> **Note** In Dubbo, the service annotation is com.alibaba.dubbo.config.annotation.Service.

4. Configure the Dubbo service.

i. In `src/main/resources`, create a file named `application.properties` or `application.yaml` and open it.

ii. In `application.properties` or `application.yaml`, add the following configuration items.

```
# Base packages to scan Dubbo Components (e.g @Service , @Reference)
dubbo.scan.basePackages=com.alibaba.edas.boot
dubbo.application.name=dubbo-provider-demo
dubbo.registry.address=nacos://127.0.0.1:8848
```

> **Note**
>
> - You must specify values for the preceding three configuration items because they have no defaults.
> - The value of `dubbo.scan.basePackages` is the name of the package with code containing annotations `com.alibaba.dubbo.config.annotation.Service` and `com.alibaba.dubbo.config.annotation.Reference`. Separate multiple packages with commas (,).
> - The value of `dubbo.registry.address` must start with **nacos://**, followed by the IP address and port of Nacos Server. The IP address in the code example is a local address. If your Nacos Server is deployed on another machine, change it to the corresponding IP address.

5. Develop and start the Spring Boot main class `DubboProvider`.

```
package com.alibaba.edas.boot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DubboProvider {

    public static void main(String[] args) {
        SpringApplication.run(DubboProvider.class, args);
    }

}
```

6. Log on to the Nacos console at `http://127.0.0.1:8848`. In the left-side navigation pane, click **Services** to view the list of providers. You can see that `com.alibaba.edas.boot.IHelloService` is available in the list of providers. In addition, you can query Service Group and Provider IP of the service.

Create a service consumer

1. Create a Maven project named `spring-boot-dubbo-consumer`.

2. Add required dependencies to the `pom.xml` file. The following takes Spring Boot 2.0.6.RELEASE as an example.

```
  <dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
      <version>2.0.6.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-spring-boot-starter</artifactId>
    <version>2.7.3</version>
  </dependency>
  <dependency>
    <groupId>com.alibaba.nacos</groupId>
    <artifactId>nacos-client</artifactId>
    <version>1.1.1</version>
  </dependency>

</dependencies>
```

If you want to use Spring Boot 1.x, select Spring Boot 1.5.x. The corresponding com.alibaba.boot:dubbo-spring-boot-starter version is 0.1.0.

> **Note** Spring Boot 1.x has reached the end of life in August 2019. We recommend that you use a later version to develop applications.

3. Develop a Dubbo consumer.
   i. Create a package named `com.alibaba.edas.boot` in `src/main/java`.
   ii. Create an interface named `IHelloService` that contains a `SayHello` method in `com.alibaba.edas.boot`.

```
package com.alibaba.edas.boot;

public interface IHelloService {
 String sayHello(String str);
}
```

4. Develop a Dubbo service call.

   For example, you need to call a remote Dubbo service once in a controller. The code is as follows.

```
package com.alibaba.edas.boot;

import com.alibaba.dubbo.config.annotation.Reference;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
  public class DemoConsumerController {

    @Reference
    private IHelloService demoService;

    @RequestMapping("/sayHello/{name}")
    public String sayHello(@PathVariable String name) {
      return demoService.sayHello(name);
    }
  }
```

   Note The Reference annotation is com.alibaba.dubbo.config.annotation.Reference.

5. Add the following configuration items to the `application.properties or application.yaml` file.

```
dubbo.application.name=dubbo-consumer-demo
dubbo.registry.address=nacos://127.0.0.1:8848
```

> **Note**
>
> - You must specify values for the preceding two configuration items because they have no defaults.
> - The value of `dubbo.registry.address` must start with `nacos://`, followed by the IP address and port of Nacos Server. The IP address in the code example is a local address. If your Nacos Server is deployed on another machine, change it to the corresponding IP address.

6. Develop and start the Spring Boot main class `DubboConsumer`.

```
package com.alibaba.edas.boot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DubboConsumer {

    public static void main(String[] args) {
        SpringApplication.run(DubboConsumer.class, args);
    }

}
```

7. Log on to the Nacos console at `http://127.0.0.1:8848`. In the left-side navigation pane, click **Services**. On the Services page, click the **Callers** tab to view the list of callers. You can see that `com.alibaba.edas.boot.IHelloService` is available in the list. In addition, you can view the group and caller IP address of the service.

Verify the result

```
`curl http://localhost:8080/sayHello/EDAS`

`Hello, EDAS (from Dubbo with Spring Boot)`
```

Deploy applications to EDAS

You can deploy the application that uses local Nacos as the registry directly to Enterprise Distributed Application Service (EDAS) without making any changes. This registry will be automatically replaced with the registry in EDAS.

If you use the console for deployment, follow these steps in your local application before deploying it:

1. Add the following configuration of the packaging plug-in to the `pom.xml` file.

- Provider

```
<build>
 <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
          <configuration>
            <classifier>spring-boot</classifier>
            <mainClass>com.alibaba.edas.boot.DubboProvider</mainClass>
          </configuration>
        </execution>
      </executions>
    </plugin>
 </plugins>
</build>
```

- Consumer

```
<build>
 <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
          <configuration>
            <classifier>spring-boot</classifier>
            <mainClass>com.alibaba.edas.boot.DubboConsumer</mainClass>
          </configuration>
        </execution>
      </executions>
    </plugin>
 </plugins>
</build>
```

2. Run **mvn clean package** to package your local program into a JAR file.

1.3.3. Develop applications in HSF

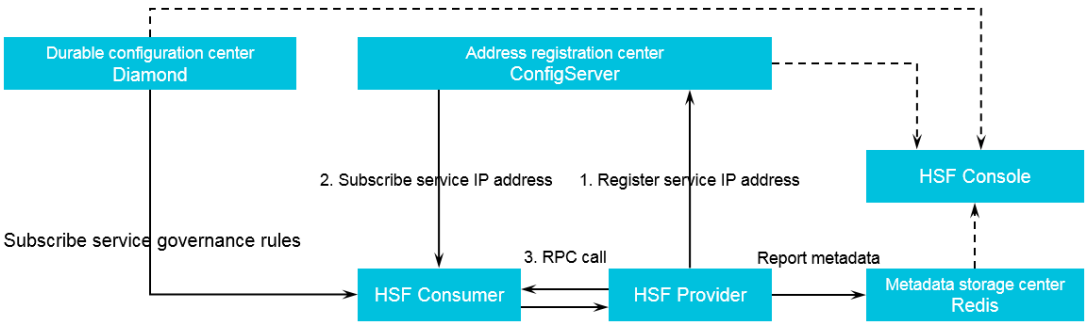
1.3.3.1. HSF overview

High-speed Service Framework (HSF) is a distributed RPC service framework widely used within the Alibaba Group. HSF connects different service systems, decoupling the implementation of the systems from each other. HSF unifies service publishing and call methods for distributed applications, helping you conveniently and quickly develop distributed applications. It provides shared function modules, which free developers from complex technical details in distributed systems, such as remote communication, serialization, performance loss, and synchronous or asynchronous calls.

HSF architecture

As a client-side RPC framework, HSF has no server cluster. All HSF service calls are point-to-point between consumers and providers, both of which can be regarded as HSF clients. However, HSF must work with the following external systems to implement the complete distributed service system.

HSF architecture



- Registry

  HSF depends on the registry for service discovery. Without the registry, HSF can only make simple point-to-point calls. The service provider cannot publish its service information to others. The service consumer may know which services to call, but cannot obtain information about the instances providing these services. In this case, the registry serves as a medium for the discovery of service information. The role of the registry is played by ConfigServer.

- Persistent configuration center

  The persistent configuration center is used to store the governance rules of HSF services. Upon startup, the HSF consumer subscribes to required service governance rules, such as routing rules, grouping rules, and weighting rules, from the persistent configuration center to intervene in the address selection logic of the call procedure based on the rules. The role of the persistent configuration center is played by DiamondServer.

- Metadata storage center

  Metadata refers to the methods, parameter structure, and other information related to HSF services. Metadata does not affect the call procedure of HSF. Therefore, the metadata storage center is optional. However, to ensure convenient service maintenance, upon startup, the HSF provider and consumer report the metadata to the metadata storage center for further maintenance. The role of the metadata storage center is played by Redis.

Functions

As a distributed RPC framework, HSF supports the following service call methods:

- Synchronous calls

  By default, an HSF consumer consumes services by synchronous calls, and the consumer's codes must synchronously wait for the returned results of calls.

- Asynchronous calls

  For a consumer that calls HSF services, it is not always necessary to synchronously wait for the returned results of calls. For such services, HSF supports asynchronous calls, so that consumers are not blocked synchronously in HSF call operations. You can make asynchronous HSF calls in either of the following two methods:

  - Future: The consumer obtains the returned results of calls by `HSFResponseFuture.getResponse(int timeout)` when needed.
  - Callback: The calls are made by the Callback method provided by HSF. After the specified HSF service is consumed and the results are returned, the HSF consumer calls `HSFResponseCallback` to obtain the call results through callback notifications.

- Generic calls

  For a typical HSF call, the HSF consumer has to perform a programming call with the API in the second-party package of the service to obtain the returned results. In contrast, a generic call initiates an HSF call and obtains returned results, independent of the second-party package of the service. For some platform-based products, generic calls can effectively reduce dependencies on second-party packages and realize lightweight system operation.

- HTTP calls

  HSF can expose services over HTTP. In this way, non-Java consumers can initiate service calls over HTTP.

- Trace filter extension

  HSF, designed with a built-in call filter, can actively find and instrument the user's call filter extension into HSF call traces, enhancing the convenience of HSF request extension.

Application development methods

Under HSF, you can use Ali-Tomcat and Pandora Boot to develop applications.

- **Ali-Tomcat**: Relying on Ali-Tomcat and Pandora, this method provides complete HSF functions, including service registration and discovery, implicit parameter passing, asynchronous calls, generic calls, and trace filter extension. In this method, applications must be deployed with WAR packages.
- **Pandora Boot**: Relying on Pandora, this method provides complete HSF functions, including service registration and discovery and asynchronous calls. Applications can be packaged and deployed as JAR packages that run independently.

1.3.3.2. Configure the lightweight configuration center

The lightweight configuration center allows developers to discover, register, and query services during development, debugging, and testing. Within a company, you generally only need to install the lightweight configuration center on one server and bind specific hosts on other development instances.

Prerequisites

Check that the environment requirements are met.

- Check that the environment variable JAVA_HOME points to JDK 1.6 or later.
- Check that port 8080 and port 9600 are not occupied.

  Port 8080 and port 9600 are occupied for starting the EDAS lightweight configuration center. We recommend that you use a dedicated ECS instance, for example, a test ECS instance, to start the EDAS lightweight configuration center.

Procedure

1. Download the installation package of the EDAS lightweight configuration center edas-config-center.zip and decompress it.
2. Go to the *edas-config-center* directory to start the configuration center.
   - For a Windows system, double-click `startup.bat`.
   - For a UNIX system, run the `sh startup.sh` command in the current directory.

3. On the local DNS server (or in the hosts file), point the jmenv.tbsite.net domain name to the IP address of the ECS instance that starts the EDAS lightweight configuration center.

   The path of the hosts file is as follows:

   - Windows system: *C:\Windows\System32\drivers\etc\hosts*
   - UNIX system: */etc/hosts*

   If you start the EDAS lightweight configuration center on the ECS instance whose IP address is 192.168.1.100, all developers only need to add the following line to the hosts files on their local instances:

   ```
   192.168.1.100 jmenv.tbsite.net
   ```

1.3.3.3. Use Ali-Tomcat to develop applications

1.3.3.3.1. Ali-Tomcat overview

Ali-Tomcat is a container on which EDAS depends to run services. It integrates such core functions as service publishing, subscription, and service call tracing. You can publish applications in this container in both development and runtime environments.

Pandora is a lightweight isolation container, namely taobao-hsf.sar. This container is used to isolate dependencies between applications and middleware products, as well as dependencies between middleware products. EDAS Pandora integrates plug-ins that implement service discovery, configuration push, service call tracing, and other middleware products. By using these plug-ins, you can monitor, process, trace, analyze, maintain, and manage EDAS applications.

> **Note** In EDAS, Ali-Tomcat is only available for HSF applications in WAR format.

1.3.3.3.2. Install Ali-Tomcat and Pandora

Ali-Tomcat and Pandora are containers on which EDAS depends to run services. They integrate such core functions as service publishing, subscription, and service call tracing. Applications must be published in such containers in both development and runtime environments.

Procedure

1. Download the Ali-Tomcat package and decompress the downloaded package to a directory, such as *d:\work\tomcat*.

   > **Note** Use JDK 1.7 or later.

2. Download the Pandora package, save it, and decompress the downloaded package to the deploy directory (*d:\work\tomcat\deploy*) where Ali-Tomcat is saved.

   The directory is structured as follows:

   - In a Linux system, run the tree -L 2 deploy/ command in the relevant path to view the directory structure.

```
d:\work\tomcat >  tree -L 2 deploy/

              deploy/

                   └──── taobao-hsf.sar

                        ├──── META-INF

                        ├──── lib

                        ├──── log.properties

                        ├──── plugins

                        ├──── sharedlib

                        └──── version.properties
```

- In a Windows system, directly navigate to the target path to view the directory structure.

1.3.3.3.3. Perform startup configuration for an IDE runtime environment

Startup configuration for an IDE runtime environment includes configuration of the Eclipse development environment and IntelliJ IDEA development environment.

1.3.3.3.4. Develop HSF applications (EDAS-SDK)

1.3.3.3.5. Migrate Dubbo applications to HSF (not recommended)

1.3.3.4. Use Pandora Boot to develop applications

Derived from Pandora, Pandora Booth is more lightweight.

- Based on Pandora and FatJar technologies, Pandora Boot allows you to directly start a Pandora environment in IDE, greatly improving the development and debugging efficiency.
- Pandora Boot deeply integrates with Spring Boot AutoConfigure, letting you enjoy the convenience of the Spring Boot framework.

Spring Boot users who need to use HSF and users who already use Pandora Boot can use Pandora Boot to develop EDAS applications.

1.3.3.4.1. Pandora Boot overview

Derived from Pandora, Pandora Booth is more lightweight.

- Based on Pandora and FatJar technologies, Pandora Boot allows you to directly start a Pandora environment in IDE, greatly improving the development and debugging efficiency.
- Pandora Boot deeply integrates with Spring Boot AutoConfigure, letting you enjoy the convenience of the Spring Boot framework.

Spring Boot users who need to use HSF and users who already use Pandora Boot can use Pandora Boot to develop EDAS applications.

1.3.3.4.2. Configure the local repository path and lightweight configuration center of EDAS

Before using Pandora Boot to develop HSF applications, you must configure the local repository path and lightweight configuration center of EDAS.

- Currently, third-party packages of Spring Cloud for Aliware are only released in the local repository of EDAS. You need to add the local repository path of EDAS in Maven.
- The lightweight configuration center must be started for local code development and debugging. The lightweight configuration center provides a lightweight version of EDAS service registry.

Configure the local repository path of EDAS in Maven

Note Maven 3.x or later is required. Add the local repository path of EDAS in the Maven configuration file `settings.xml`.

1. In the Maven configuration file, whose path is generally `~/.m2/settings.xml`, add the local repository path of EDAS. The following provides a configuration example:

```
<profiles>
 <profile>
   <id>nexus</id>
   <repositories>
     <repository>
       <id>central</id>
       <url>http://repo1.maven.org/maven2</url>
       <releases>
         <enabled>true</enabled>
       </releases>
       <snapshots>
         <enabled>true</enabled>
       </snapshots>
     </repository>
```

```
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>central</id>
        <url>http://repo1.maven.org/maven2</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>true</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>
  <profile>
    <id>edas.oss.repo</id>
    <repositories>
      <repository>
        <id>edas-oss-central</id>
        <name>taobao mirror central</name>
        <url>http://edas-public.oss-cn-hangzhou.aliyuncs.com/repository</url>
        <snapshots>
          <enabled>true</enabled>
        </snapshots>
        <releases>
          <enabled>true</enabled>
        </releases>
      </repository>
      </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>edas-oss-plugin-central</id>
        <url>http://edas-public.oss-cn-hangzhou.aliyuncs.com/repository</url>
        <snapshots>
          <enabled>true</enabled>
        </snapshots>
        <releases>
          <enabled>true</enabled>
        </releases>
      </pluginRepository>
    </pluginRepositories>
```

```
 </profile>
</profiles>
<activeProfiles>
 <activeProfile>nexus</activeProfile>
 <activeProfile>edas.oss.repo</activeProfile>
</activeProfiles>
```

2. In the CLI, run the `mvn help:effective-settings` command to check whether the settings are added.

   Pay attention to the following information during verification:

   - If no error exists, the file format of setting.xml is correct.
   - If edas.oss.repo is included in profiles, the local repository settings have been added to profiles.
   - If edas.oss.repo is included in activeProfiles, the edas.oss.repo local repository has been activated.

   > **Note** If no error is returned when you run the Maven packaging command in the CLI, but IDE still cannot download the dependencies, close IDE and start it again or search for a solution in the documentation for configuring Maven in IDE.

Configure the lightweight configuration center

For more information about how to configure the lightweight configuration center, see

1.3.3.4.3. Develop HSF applications (Pandora Boot)

Currently, EDAS fully supports Spring Cloud applications and they can be deployed directly in EDAS.

- The concept behind Spring Boot is "Build Anything". It helps resolve complicated XML configuration problems.
- The concept behind Spring Cloud is "Coordinate Anything". It helps simplify the development of distributed microservices by providing large-scale spring-cloud-starters featuring convenient component access.

EDAS also implements its own Spring Cloud Starter HSF, allowing you to develop HSF applications by using Spring Cloud.

This topic describes how to use Spring Cloud to develop HSF applications.

1.3.3.4.4. Develop RESTful applications (not recommended)

This topic describes how to develop RESTful applications in EDAS by using Spring Cloud.

1.3.3.4.5. Migrate Dubbo applications to HSF (not recommended)

You can migrate applications developed with Dubbo to HSF by adding Maven dependencies or adding or modifying the Maven packaging plug-in and modifying the configuration. However, we recommend that beginners do not use this method because EDAS already supports applications in the native Dubbo framework.

For more information about how to develop applications in the native Dubbo framework, see Use Spring Boot to develop Dubbo applications.

> **Note** This topic describes how to modify the configuration. The application development process is not described in detail in this topic. For more information about how to develop applications, download these Demos for converting Dubbo applications to HSF applications.

Add Maven dependencies

In the project configuration file `pom.xml`, add the `spring-cloud-starter-pandora` dependency.

```
<dependency>

  <groupId>org.springframework.cloud</groupId>

  <artifactId>spring-cloud-starter-pandora</artifactId>

  <version>1.3</version>

</dependency>
```

Add or modify the Maven packaging plug-in

In the project configuration file `pom.xml`, add or modify the Maven packaging plug-in.

> **Note** To avoid conflicts with other packaging plug-ins, do not add configurations of any other FatJar plug-ins to the plugin field in build.

```
<build>
  <plugins>
    <plugin>
      <groupId>com.taobao.pandora</groupId>
      <artifactId>pandora-boot-maven-plugin</artifactId>
      <version>2.1.9.1</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>repackage</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Modify the configuration

In the Spring Boot startup class, add the following two lines for loading Pandora:

```
import com.taobao.pandora.boot.PandoraBootstrap;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class ServerApplication {

    public static void main(String[] args) {

        PandoraBootstrap.run(args);

        SpringApplication.run(ServerApplication.class, args);

        PandoraBootstrap.markStartupAndWait();

    }

}
```

1.4. Deploy applications

1.4.1. Deploy applications in the console

1.4.1.1. Deploy web applications in ECS clusters

In an ECS cluster, an ECS instance can only deploy one application. This topic describes how to create a Java web application that only contains a welcome page, and use a WAR package to deploy, update, view, and manage the application in the Enterprise Distributed Application Service (EDAS) console.

Prerequisites

Prerequisites

- You have activated EDAS.
- You have created a VPC.
- (Optional) You have created a namespace.
- You have created an ECS cluster.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management** > **Applications**.
3. On the **Applications** page, select a **region** and a **namespace** (optional). Click **Create Application**.
4. On the **Application Information** page that appears, enter the application information. After setting the parameters, click **Create an Empty Application**. This creates an application without any instance. Click **Next**. On the Application Configuration page, set the required parameters.
   - **Namespace**: Select a **region** from the left-side drop-down list and select a **namespace** from the right-side drop-down list. If you do not select a namespace, the **default** one is used.
   - **Cluster Type**: Select **ECS Cluster** from the left-side drop-down list, and select a specific ECS cluster from the right-side drop-down list.
   - **Application Name**: The name of the application.
   - **Deployment Method**: After selecting an ECS cluster, you can deploy the application through a **WAR package** or a **JAR package**.
   - **Application Runtime Environment**:
     - **Deploy applications by using a WAR package**:
       - If you want to create a native Spring Cloud or Dubbo application, select **apache-tomcat**.
       - If you want to create an HSF application, select **EDAS-Container**.
     - **Deploy applications by using a JAR package**:
       - If you want to create a native Spring Cloud or Dubbo application, select **Default Environment**.
       - If you want to create an HSF application, select **EDAS-Container**.
   - **Java Environment**: Select **Open JDK 8** or **Open JDK 7**.
   - **Application Description**: Enter the basic information of the application. The maximum length of the description is 128 characters.
5. On the Application Configuration page, add an instance and configure the instance as instructed. After configuring the instance, click **Create**.
   - **Instance Source**: In ECS clusters, you can add an instance in any of the following three methods. If you do not select any instance, click **Create an Empty Application** to create an application that contains no instances. Then, add instances to the application through application scale-out and deploy the application.
     - **Select an instance from the cluster**: Click **Add** next to **Available Instances**. On the **Instances** page, select an idle instance from the cluster of the application, click > to add the instance to the Selected Instances area, and then click **OK**.

- **Create an instance based on the existing instance specifications:**
  a. Click **Host Selection** next to **Template Host**.
  b. In the **Template Host** dialog box, select any instance in the cluster and use it as the template. Click **Recycling Mode**, and then click **OK** in the lower-right corner.
  c. On the **Application Configuration** tab page, configure the **password** and **purchase quantity**. Then, select **ECS Service Terms | Image Service Terms**.
- **Create an instance from a template:**
  a. Click **Select Template** next to **Launch Template**.
  b. In the **Select Template to Be Launched** dialog box, select the template based on which the instance is created and the template version, select **Recycling Mode**, and then click **OK** in the lower-right corner.
  c. On the **Application Configuration** tab page, configure the **purchase quantity** of the instance, and select **ECS Service Terms | Image Service Terms**.

- **Deploy Now**: This option is available only after you have selected an instance. Turn on Deploy Now and configure the instance as instructed.
- **File Uploading Method**: Select **Upload WAR Package** or **WAR Package Location**.
  - **Upload WAR Package**: Click **Select File** and select the target WAR package.
  - **WAR Package Location**: Copy the storage path of the WAR package and paste the path to the WAR package location bar.

    > **Note** The name of the application deployment package can only contain letters, numbers, hyphens (-), and underscores (_). The JAR package can be uploaded only when the JAR package deployment method is selected. Otherwise, you can only deploy the application by using the WAR package.

- **Version**: Specify the version, for example, *1.1.0*. We recommend that you do not use the timestamp as the version number.
- **Application Health Check**: (Optional) Specify a URL for application health check. The system checks the health of the application after the container is started or is running. Then, it performs a service routing task based on the health check result. A sample URL is http://127.0.0.1:8080/_etc.html.
- **Batch**: Specify the number of batches. You can specify the number of batches and publish the application to the selected instances in batches only when two or more instances are selected.
- **Batch Mode**: Select **Automatic** or **Manual**. When you select Automatic, you need to specify **Batch Wait Time**, which is the interval between different application deployment batches.

Result

Wait several minutes until the application is created. After the application is created, you can view the application information on the Application Details page. On the Application Details page, click the **Instance Information** tab. On the Instance Information tab page, view the instance running status. If Running Status/Time is **Running**, the application is published.

1.4.1.2. Use an image to deploy an application in a Container Service Kubernetes cluster

You can deploy applications in Container Service Kubernetes clusters by using images. For this purpose, you must prepare images in advance, create a Container Service Kubernetes cluster in the Container Service for Kubernetes console, import the cluster to the Enterprise Distributed Application Service (EDAS) console, and then create and deploy applications in the cluster.

Prerequisites

Make sure the following prerequisites are met:

- You have granted the required permissions for Container Service for Kubernetes.
- You have prepared an application image for the Container Service Kubernetes cluster.

Context

Container Service for Kubernetes provides enterprise-level high-performance and flexible management for Kubernetes containerized applications throughout the application lifecycle. This service simplifies cluster creation and scale-out and integrates capabilities of Alibaba Cloud in virtualization, storage, networking, and security. Therefore, this service provides an improved runtime environment for Kubernetes containerized applications.

Step 1: Create a Container Service Kubernetes cluster

For more information, see *Container Service for Kubernetes User Guide.*

Step 2: Import the Container Service Kubernetes cluster to the EDAS console

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Resource Management** > **Clusters**.
3. On the **Clusters** page, click **Container Service K8S Cluster**. In the cluster list, locate the row that contains the Container Service Kubernetes cluster you created and click **Import** in the **Actions** column. In the **Import Kubernetes Cluster** dialog box, click **Import**.When the button in the **Actions** column of the target cluster changes to **Delete** and the cluster status is **Running**, the cluster is imported to EDAS.

Step 3: Create an application in the Container Service Kubernetes cluster

1. In the left-side navigation pane, choose **Application Management**.
2. On the Applications page, set **Region** and **Namespace**, and then click **Create Application** in the upper-right corner.
3. On the **Application Information** page, set the basic application information and parameters, and click **Next Step: Application Configurations**.
   - **Namespace**: Select a region from the left-side drop-down list. Select a namespace from the right-side drop-down list. If no namespace is set, **Default** is selected.
   - **Cluster Type**: Select **Container Service K8S Cluster** from the left-side drop-down list and select a specific cluster from the right-side drop-down list.
   - **K8S Namespace**: Internal system objects are allocated to different namespaces to form

logically isolated projects, groups, or user groups. In this way, different groups can share resources of the whole cluster while being managed separately.

- **default**: When the object is not set with a namespace, "default" is used.
- **kube-system**: The namespace used by objects that are created by the system.
- **kube-public**: The namespace that is automatically created by the system. It can be read by all users, including users that are not authenticated.

- **Application Name**: The name of the application.
- **Application Description**: The basic information of the application.

4. On the **Application Configuration** page, set **Deployment Method** to **Image** and configure the deployment parameters.
   - **Application Runtime Environment**: Select a value from the drop-down list. For HSF applications, select the corresponding **EDAS-Container** version. For Spring Cloud or Dubbo applications, select the **standard Java application runtime environment**.
   - **Java Environment**: Select **Open JDK 8** or **Open JDK 7** from the drop-down list.
   - **File Uploading Method**: Select **Upload JAR Package** from the drop-down list.
   - **Upload JAR Package**: Click **Select File**. In the dialog box that appears, select the JAR package file of the application.
   - **Version**: Enter the version of the application.
   - **Total Pods**: Enter the number of pods required by the application.
   - **Single Pod Resource Quota**: Set the CPU cores and memory for individual pods.
5. (Optional)Set the startup command and parameters.

   > **Note**
   >
   > If you do not know the CMD and ENTRYPOINT content of the original Dockerfile image, do not modify the custom startup command and parameters. Otherwise, you cannot create applications due to an incorrect custom command.

   - **Startup Command**: Enter the startup command. To run the `CMD ["/usr/sbin/sshd","-D"]` command, enter `/usr/sbin/sshd -D` in the text box.
   - **Startup Parameters**: Enter one parameter per line. For example, `args:["-c"; "while sleep 2"; "do echo date"; "done"]` contains four parameters. In this case, enter the parameters in four lines.
6. (Optional)Set environment variables.

   When creating the application, inject the environment variables you have entered to the container to be generated. This saves you from repeatedly adding common environment variables.

   If you are using a MySQL image, refer to the following environment variables:

   - MYSQL_ROOT_PASSWORD: (required) allows you to set a root password for MySQL.
   - MYSQL_USER and MYSQL_PASSWORD: (optional) allow you to add an account besides the root account and set a password.
   - MYSQL_DATABASE (optional): allows you to set the database that you want to create

when the container is generated.

If you are using another type of image, configure the environment variables as needed.

7. (Optional)Set persistent storage.

   In the Container Service Kubernetes cluster of Alibaba Cloud, the physical storage of the native volume object is not persistent. That is to say, the volume object is a temporary storage object and has the same lifecycle as the Kubernetes pods. You can use Network Attached Storage (NAS), a persistent storage service of Alibaba Cloud, to store instance data persistently. The instance data is retained when the instance is upgraded or migrated.

   Note: Before enabling persistent storage, ensure that you have activated NAS for your EDAS account.

   - **Storage Type**: The default value is NAS, which cannot be changed.
   - **Storage Service Type**: Currently, only SSD (Performance Type) is supported, which cannot be changed.
   - **Select NAS**:
     - **Buy New NAS**: Select a NAS mount directory and a local mount directory. A single region supports up to 10 NAS instances. Once you have 10, you cannot create any more. If you must create more instances, open a ticket.
     - **Use Existing NAS**: Select an existing NAS instance. You can create up to two mount points. Only compliant NAS instances appear in the drop-down list.
   - **Mount Directory**: Set the mount directory command.

8. (Optional)Configure the application lifecycle management.

   The Container Service Kubernetes cluster supports stateless applications and stateful applications.

   - Stateless: A stateless application supports multi-replica deployment. When a stateless application is redeployed, instance data is not retained. A stateless application can be either of the following applications:
     - A web application that does not retain instance data during upgrade or migration.
     - An application that can be scaled out to address changing service volumes.
   - Stateful: A stateful application stores data that requires persistent storage and retains instance data during upgrade or migration. A stateful application can be either of the following applications:
     - An application that frequently operates on containers through SSH.
     - An application that requires persistent data storage (such as applications using MySQL) or that supports inter-cluster election and service discovery, such as ZooKeeper and etcd.

   You can set lifecycle management for a stateful application as needed.

   Lifecycle management scripts:

- **Poststart script**: This is a container hook, which is triggered immediately after a container is created to notify the container of its creation. The hook does not pass any parameters to the corresponding hook handler. If the corresponding hook handler fails to run, the container is killed and the system determines whether to restart the container according to the restart policy of the container. For more information, see Container Lifecycle Hooks
- **PreStop script**: This is a container hook, which is triggered before a container is deleted. The corresponding hook handler must be completed before the container deletion request is sent to Docker daemon. Docker daemon sends an SGTERN semaphore to itself to delete the container, regardless of the hook handler execution result. For more information, see Container Lifecycle Hooks
- **Liveness script**: This is a container status probe, which monitors the health status of applications. If an application is unhealthy, the container is deleted and created again. For more information, see Pod Lifecycle
- **Readiness script**: This is a container status probe, which monitors whether applications have started successfully and are running properly. If an application is abnormal, the container status is updated. For more information, see Pod Lifecycle

9. Then, click **Create**.

Result

Creating an application takes several minutes. You can track the creation progress based on the change record and change details. The Container Service Kubernetes application does not need to be deployed because it has been deployed when it was created. After the application is created, go to the Instance Information tab of the Basic Information page to check whether the pod is **running**. If yes, the application is published.

1.4.2. Use CLI to deploy applications

1.4.2.1. Use toolkit-maven-plugin to automatically deploy applications

Previously, EDAS applications had to be deployed according to the step-by-step instructions in the console. To improve the developer experience, toolkit-maven-plugin has been provided for auto application deployment. You can use toolkit-maven-plugin to automatically deploy applications that are developed based on the HSF, Dubbo, or Spring Cloud framework in ECS or Swarm clusters.

Automatically deploy applications

1. Add the following plug-in dependencies to the pom.xml file in your packaged project.

```
<build>
  <plugins>
    <plugin>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>toolkit-maven-plugin</artifactId>
    <version>1.0.2</version>
    </plugin>
  </plugins>
</build>
```

2. Create a file named *.edas_config.yaml* in the root directory of the packaged project. If the packaged project is a Maven submodule, create the file in the submodule directory.

```
env:  region_id: cn-beijingapp:  app_id: eb20****-e6ee-4f6d-a36f-5f6a5455****  endpoint: xxxxx
```

In the preceding configuration, **region_id** indicates the ID of the region where the ECS instance that hosts the application is located. **app_id** indicates the ID of the application. **endpoint** indicates the point of presence (POP) of EDAS in Apsara Stack. The preceding parameter values are for reference only. Replace them with your actual application parameters. For example, to obtain an **endpoint**, contact EDAS Customer Services. For more information about configuration items, see More configuration items.

To obtain the values of these configuration items, perform the following steps:

i. Log on to the EDAS console.

ii. In the left-side navigation pane, choose **Application Management**. On the Applications page, locate the row that contains the target application and click the name of the application. On the Application Management page, click **Deploy Application**.

iii. On the **Deploy Application** page, click **Generate Maven Plug-in Configuration** to obtain the parameter values.

3. Create an account file and configure the AccessKey ID and AccessKey Secret in yaml format. Obtain the AccessKey ID and AccessKey Secret on the User Info page in the Alibaba Cloud console. We recommend that you use a RAM user that has been granted **application management** permissions to improve the application security. The following provides a configuration example:

```
access_key_id: abcaccess_key_secret: 1234567890
```

> **Note** In the preceding configuration, abc and 1234567890 are for reference only. Replace them with your actual AccessKey ID and AccessKey Secret. In this configuration, the AccessKey ID and AccessKey Secret are only used to generate request signatures and not for any other purposes, such as network transfers.

4. Go to the root directory (or the submodule directory if multiple Maven modules exist) and run the following packaging command:

```
mvn clean package toolkit:deploy -Daccess_key_file={account file path}
```

The preceding parameters are described as follows:

- **toolkit:deploy**: Use toolkit-maven-plugin to deploy the application after it is packaged successfully. The application is deployed only when this parameter is configured.
- **access_key_file**: The file of the Alibaba Cloud account. For more information about how to specify a key pair, see Account configuration.

5. After you run the preceding command, you have successfully deployed the application with toolkit-maven-plugin.

More configuration items

Configuration items for deploying applications are classified as follows:

- Basic environment variables (env)
- Application configuration items (app)
- Storage configuration items (oss)

The configuration items currently supported are listed in the following table.

Type	Parameter	Required	Note
env	region_id	Yes	The ID of the region where the application is located.
	endpoint	No	The POP of the application.
app	app_id	Yes	The ID of the application.
	package_version	No	The version of the deployment package. The default value is the string of the pom.xml file version plus the instance creation time, for example, "1.0 (2018-09-27 19:00:00)".
	desc	No	The deployment description.
	group_id	No	The ID of the group to which the application is deployed. The default value is All Groups.
	batch	No	The number of deployment batches. The default value is 1 and the maximum value is 5.

app Type	Parameter	Required	Note
	batch_wait_time	No	The waiting time (in minutes) between deployment batches. The default value is 0.
	stage_timeout	No	The timeout period (in minutes) for each change stage. The default value is 5. If batch_wait_time is set, it is automatically counted with this parameter during calculation. During runtime, if a stage waits for a time longer than this threshold value, the plug-in automatically exits.
	region_id	No	The ID of the region where the target bucket is located. The default value is the ID of the region where the application is located.
	bucket	No	The name of the target bucket. The default value is the free OSS bucket provided by EDAS. If OSS configuration items are specified, you must specify the bucket parameter. Otherwise, the instances use the free OSS bucket automatically allocated by EDAS.

Type	Parameter	Required	Note
oss	key	No	The custom path used to upload the application package to OSS. The instances use the free OSS bucket provided by EDAS by default. If you use a specified OSS bucket, specify the package storage path in this parameter and use the {region_id}, {app_id}, and {version} variables to set the path through parameters, for example, pkgs/petstore/{version}/store.war. The default value is {region_id}/{app_id}/{version}.
	access_key_id	No	The custom account ID that is used to upload the application package to OSS.
	access_key_secret	No	The custom account key that is used to upload the application package to OSS.

Configuration example 1: Specify the group and the deployment package version

Assume that you want to deploy application `eb20dc8a-e6ee-4f6d-a36f-5f6a545****` to group `06923bb9-8c5f-4508-94d8-517b692f****` in China (Beijing). The version of the deployment package is 1.2. In this case, the configuration is as follows:

```
env:  region_id: cn-beijingapp:  app_id: eb20dc8a-e6ee-4f6d-a36f-5f6a5455****  package_version: 1.2
group_id: 06923bb9-8c5f-4508-94d8-517b692f****
```

Configuration example 2: Specify an OSS bucket

Assume you want to deploy an application whose ID is `eb20dc8a-e6ee-4f6d-a36f-5f6a5455****` and upload the deployment package to your own bucket named `release-pkg` in China (Beijing). The file object name is my.war, the ID of the OSS account is `ABC`, and the key of the OSS account is `1234567890`. In this case, the configuration is as follows:

```
env:  region_id: cn-beijingapp:  app_id: eb20dc8a-e6ee-4f6d-a36f-5f6a5455****oss:  region_id: cn-beiji
ng  bucket: release-pkg  key: my.war   access_key_id: ABC  access_key_secret: 1234567890
```

Configuration file

- When no configuration file is specified, the plug-in uses the `.edas_config.yaml` file in the root directory of the packaged project as the configuration file by default. If the packaged project is a submodule of the Maven project, the configuration file is in the root directory of the submodule by default but not the root directory of the entire Maven project.
- You can also specify a configuration file by setting the `-Dedas_config=xxx` parameter.
- If the default configuration file exists but another configuration file is specified using the parameter, the plug-in uses the latter.

Account configurations and priorities

When using this plug-in to deploy applications, you must provide the AccessKey ID and AccessKey Secret of an Alibaba Cloud account for application deployment. Currently, the plug-in supports multiple configuration methods. If duplicate configurations exist, the configuration method with the higher priority overrides that with the lower priority. Configuration methods are listed as follows in descending order of priority:

- **Specify the AccessKey ID and AccessKey Secret in the CLI**: You can specify the AccessKey ID and AccessKey Secret in either of the following ways:
  - If you package the project by running Maven commands, specify both parameters with `-Daccess_key_id=xx -Daccess_key_secret=xx`.
  - When you configure this plug-in in the pom.xml file, configure both parameters as follows:

    ```
    <plugin>  <groupId>com.alibaba.cloud</groupId>  <artifactId>toolkit-maven-plugin</artifactId
    >  <version>1.0.2</version><configuration>  <accessKeyId>abc</accessKeyId>  <accessKeySe
    cret>1234567890</accessKeySecret></configuration></plugin>
    ```

- **Specify the account file in the CLI (recommended)**: When you package the project by running Maven commands, specify the account file in yaml format with `-Daccess_key_file={account file path}`. For example:

  ```
  access_key_id: abcaccess_key_secret: 1234567890
  ```

- **Use the default Alibaba Cloud account file**: If you choose not to specify an account in either of the preceding ways, the plug-in uses the Alibaba Cloud account you set previously to deploy the application.

  - **aliyuncli:** If you have used the latest Alibaba Cloud CLI and configured your Alibaba Cloud account, Alibaba Cloud generates the `.aliyuncli` directory in the current Home directory and creates the `credentials` file in the `.aliyuncli` directory to store your account information. Here, the MacOS system is used as an example. Assume that the system user is jack. Then, the following information is stored in the `/Users/jack/.aliyuncli/credentials` file:

    ```
    [default]aliyun_access_key_secret = 1234567890aliyun_access_key_id = abc
    ```

    This plug-in uses this account file as the account for deploying the application.

- **aliyun:** If you have used a legacy Alibaba Cloud CLI and configured the Alibaba Cloud account, the Alibaba Cloud CLI generates the `.aliyun` directory in the current Home directory and creates the `config.json` file in the `.aliyun` directory. Here, the MacOS system is used as an example. Assume that the system user is jack. Then, the following information is stored in the `/Users/jack/.aliyun/config.json` file:

  ```
  { "current": "", "profiles": [{ "name": "default", "mode": "AK", "access_key_id": "", "access_key_secret": "", "sts_token": "", "ram_role_name": "", "ram_role_arn": "", "ram_session_name": "", "private_key": "", "key_pair_name": "", "expired_seconds": 0, "verified": "", "region_id": "", "output_format": "json", "language": "en", "site": "", "retry_timeout": 0, "retry_count": 0 }, { "name": "", "mode": "AK", "access_key_id": "abc", "access_key_secret": "xxx", "sts_token": "", "ram_role_name": "", "ram_role_arn": "", "ram_session_name": "", "private_key": "", "key_pair_name": "", "expired_seconds": 0, "verified": "", "region_id": "cn-hangzhou", "output_format": "json", "language": "en", "site": "", "retry_timeout": 0, "retry_count": 0 }], "meta_path": ""}
  ```

- **System environment variables:** Then, the plug-in attempts to retrieve the values of access_key_id and access_key_secret from system environment variables. In other words, the plug-in retrieves the values from `System.getenv("access_key_id")` and `System.getenv("access_key_secret")` .

1.4.2.2. Use Alibaba Cloud CLI to deploy applications in EDAS

The command-line interface (CLI) was the most widely used before the graphical user interface (GUI) became popular. The CLI usually does not support the use of a mouse. Instead, you enter instructions by using a keyboard, and the computer receives and runs the instructions. By using the CLI, you can accurately control the system and efficiently and reliably perform complex operations.

Prerequisites

Before you perform the steps in this tutorial, you must have made the following preparation: Import ECS instances

Context

Alibaba Cloud CLI is an open-source tool built on the Go SDK provided by Alibaba Cloud. Alibaba Cloud CLI can directly call the Enterprise Distributed Application Service (EDAS) API. Make sure that you have activated EDAS and know how to use SDKs to call API operations in EDAS. For more information about how to call API operations, see *the related topic on how to obtain SDKs in Enterprise Distributed Application Service Developer Guide.* You can use Alibaba Cloud CLI to deploy all applications developed based on the High-speed Service Framework (HSF), Dubbo, or Spring Cloud framework in Elastic Compute Service (ECS) or Swarm clusters in EDAS.

Procedure

1. Install Alibaba Cloud CLI

   Alibaba Cloud CLI is available after you download and decompress it. It is supported on MacOS, Linux, and Windows (64-bit) clients. Download the appropriate installation package:

   - Mac
   - Linux
   - Windows (64 bit)

   After you decompress the installation package, move the *aliyun* file to the */usr/local/bin* directory or add it to the `$PATH` environment variable.

2. Configure Alibaba Cloud CLI

   Before you use Alibaba Cloud CLI, run the `aliyun configure` command to configure the AccessKey pair that you use to call API operations, the region where the cloud resource is located, and the language to use.

   You can create and view your AccessKey pair on the Security Management page, or obtain the AccessKey pair from your system administrator.

   ```
   $ aliyun configureConfiguring profile 'default' ...Aliyun Access Key ID [None]: <Your AccessKey ID>
   Aliyun Access Key Secret [None]: <Your AccessKey Secret>Default Region Id [None]: cn-hangzhou
   Default output format [json]: jsonDefault Language [zh]: zh
   ```

3. Use Alibaba Cloud CLI to create an application

   Run the following script to create an application:

```
#! /bin/bash # The region where you want to deploy the application. REGION="cn-beijing" # The ID of your ECS instance. ECS_ID="i-2z**********b6" # The ID of the VPC where your ECS instance is located. VPC_ID="vpc-t*********c" # The name of a namespace (which is automatically created if it does not exist). NAMESPACE="myNamespace" # The name of a cluster (which is automatically created). CLUSTER_NAME="myCluster" # Name of an application APP_NAME="myApp" # Step 1: Create a namespace. aliyun edas InsertOrUpdateRegion --RegionTag $REGION:$NAMESPACE --RegionName $NAMESPACE --region $REGION --endpoint "edas.cn-beijing.aliyuncs.com" >> /dev/null # Step 2: Create a cluster. CLUSTER_ID=`aliyun edas InsertCluster --ClusterName $CLUSTER_NAME --ClusterType 2 --NetworkMode 2 --VpcId $VPC_ID --logicalRegionId $REGION:$NAMESPACE --region $REGION --endpoint "edas.cn-beijing.aliyuncs.com" | sed -E 's/. *"ClusterId":"([a-z0-9-]*)".*/\1/g'` # Step 3: Convert the ECS instance (which takes some time). aliyun edas TransformClusterMember --InstanceIds $ECS_ID --TargetClusterId $CLUSTER_ID --Password Hello1234 >> /dev/nullfor i in `seq 300` do OUT=`aliyun edas ListClusterMembers --ClusterId $CLUSTER_ID | grep EcuId` && break sleep 1 done ECU_ID=`echo $OUT | sed -E 's/. *"EcuId":"([a-z0-9-]*)".*/\1/g'` # Step 4: Create an application. APP_ID=`aliyun edas InsertApplication --ApplicationName $APP_NAME --BuildPackId 51 --EcuInfo $ECU_ID --ClusterId $CLUSTER_ID --logicalRegionId $REGION:$NAMESPACE | sed -E 's/. *"AppId":"([a-z0-9-]*)".*/\1/g'` printf "An application is created by CLI, App ID:"$APP_ID"\n"
```

4. Use Alibaba Cloud CLI to deploy an application

   Use Alibaba Cloud CLI to deploy an application based on the following code:

```
#! /bin/bash # The ID of the target application that you want to deploy. You must create the ID in advance. APP_ID="87a6*********************4d1" # The ID of the application group. GROUP_ID="54b*********************f27" # The name of the OSS bucket to which the file is uploaded. The bucket must be in public-read mode. OSS_BUCKET="eda*****mo" # The installation package file, which is generated by your CI system. PACKAGE="hello-edas.war" # Step 1: Upload the deployment package to OSS. aliyun oss cp -f $PACKAGE oss://$OSS_BUCKET/$PACKAGE >> /dev/null PKG_URL=`aliyun oss sign oss://$OSS_BUCKET/$PACKAGE|head -1` # Step 2: Initiate a deployment request. CO_ID=`aliyun edas DeployApplication --AppId $APP_ID --PackageVersion $VERSION --DeployType url --WarUrl "${PKG_URL}" --GroupId $GROUP_ID | sed -E 's/. *"ChangeOrderId":"([a-z0-9-]*)".*/\1/g'` # Step 3: Wait until the application is deployed. for i in `seq 300` do STATUS=`aliyun edas GetChangeOrderInfo --ChangeOrderId $CO_ID | sed -E 's/. *"Status":(.).*/\1/g'` [ 2 = ${STATUS} ] && break sleep 1 done
```

   In the preceding configuration items, APP_ID and GROUP_ID are two configuration parameters of the application. All parameters in the preceding code are for reference only. Replace them with the actual values.

   To obtain the values of these configuration items, perform the following steps:

   i. Log on to the EDAS console.

   ii. In the left-side navigation pane, choose **Application Management**Applications. On the Applications page, click the target application name. On the Basic Information page, click **Deploy Application**.

iii. On the **Deploy Application** page, click **Generate Maven Plug-in Configuration** to retrieve the parameter values.

1.4.2.3. Use Alibaba Cloud Toolkit for Eclipse to deploy applications

Alibaba Cloud Toolkit (hereinafter referred to as "Cloud Toolkit") is a free IDE plug-in that helps users use Alibaba Cloud more efficiently. You only need to register or use an existing Alibaba Cloud account to download Cloud Toolkit for free. After the plug-in is downloaded, you can install it to Eclipse. You can use Cloud Toolkit to automatically deploy applications that are developed based on the HSF, Dubbo, or Spring Cloud framework in ECS or Swarm clusters. This topic describes how to install Cloud Toolkit to Eclipse and use Cloud Toolkit to deploy an application in EDAS.

Prerequisites

- You have downloaded and installed JDK 1.8 or later.
- You have downloaded and installed Eclipse IDE 4.5.0 (code: Mars) or later. The program must be suitable for Java EE developers.

Install Cloud Toolkit

1. Start Eclipse.
2. In the top navigation bar, choose **Help** > **Install New Software**.
3. In the **Available Software** dialog box, set **Work with** to the URL `http://toolkit.aliyun.com/eclipse/` of Cloud Toolkit for Eclipse.
4. In the Name section, select **Alibaba Cloud Toolkit Core** and **Alibaba Cloud Toolkit Deployment Tools**. In the **Details** section, clear **Connect all update sites during install to find required software**. Then, click **Next**.
5. Perform the subsequent steps as instructed on the Install page of Eclipse.

   > **Note** During the installation process, a dialog box indicating no digital signature may appear. In this case, click **Install anyway**.

6. After Cloud Toolkit is installed, restart Eclipse. Then, the Alibaba Cloud Toolkit icon appears in the toolbar.

Configure Cloud Toolkit

1. Start Eclipse.
2. Set the AccessKey ID and AccessKey Secret.
   i. In the toolbar, click the drop-down arrow of the Alibaba Cloud Toolkit icon. In the drop-down list, select **Alibaba Cloud Preference...**.
   ii. In the **Preference (Filtered)** dialog box, choose **Accounts** from the left-side navigation pane.

iii. On the **Accounts** page, set **Access Key ID** and **Access Key Secret**, and click **OK**.

> **Note**
>
> If you use the AccessKey ID and AccessKey Secret of a RAM user, make sure that the RAM user has the permission to deploy applications. For more information about how to grant permissions to RAM users, see RAM account authorization.

- If you already have an Alibaba Cloud account, on the **Accounts** page, click **Manage existing Account** to go to the logon page of Alibaba Cloud. After you log on to the system with an existing account, you are redirected to the Security Management page. On this page, obtain the **AccessKeyId** and **AccessKeySecret** of the account.
- If you do not have an Alibaba Cloud account, on the **Accounts** page, click **Sign up**. You are redirected to the Register account page of Alibaba Cloud. On this page, register an Alibaba Cloud account. Then, obtain the **AccessKeyId** and **AccessKeySecret** of the account.

3. Set an endpoint.
   i. In the **Preference (Filtered)** dialog box, choose **Appearance & Behavior** > **Endpoint** from the left-side navigation pane.
   ii. On the **Endpoint** page, set an endpoint and click **Apply and Close**.

> **Note** To obtain an endpoint, contact EDAS Customer Services.

Deploy applications to EDAS

Currently, you can use Cloud Toolkit to deploy applications to EDAS by using WAR or JAR packages.

1. In the **Package Explorer** left-side navigation pane of Eclipse, right-click your application project and choose **Alibaba Cloud** > **Deploy to EDAS** from the shortcut menu.
2. In the **Deploy to EDAS** dialog box, select **Region**, **Namespace**, **Application**, **Group**, and Deploy File as needed. Then, click **Deploy**.

   Parameters for deploying an application to EDAS:

   - **Region**: The region where the application is located.
   - **Namespace**: The namespace where the application is located.
   - **Application**: The name of the application.
   - **Group**: The group of the application.

   > **Note** If you have not created an application in EDAS, click **Create application on EDAS console** in the upper-right corner of the dialog box to go to the EDAS console and create an application. For more information about how to create an application, see Deploy Java applications in ECS clusters.

3. When the deployment process starts, the deployment logs are printed on the Console tab of Eclipse. You can view the deployment result based on the logs.

Stop Cloud Toolkit

If you want to stop Cloud Toolkit, end the **EDAS-deploy** process on the **Progress** tab.

1.4.2.4. Use Alibaba Cloud Toolkit for IntelliJ IDEA to deploy applications

Alibaba Cloud Toolkit for IntelliJ IDEA (hereinafter referred to as "Cloud Toolkit") is a free IDE plug-in that helps users use Alibaba Cloud more efficiently. You only need to register or use an existing Alibaba Cloud account to download Cloud Toolkit for free. After the plug-in is downloaded, you can install it to IntelliJ IDEA. You can use Cloud Toolkit to automatically deploy applications that are developed based on the HSF, Dubbo, or Spring Cloud framework in ECS or Swarm clusters. This topic describes how to install Cloud Toolkit in IntelliJ IDEA and how to use Cloud Toolkit to deploy an application in EDAS.

Prerequisites

- You have downloaded and installed JDK 1.8 or later.
- You have downloaded and installed IntelliJ IDEA (2018.3 or later).

> **Note** The official server of the JetBrains plug-in is deployed outside China. If you cannot download IntelliJ IDEA due to a slow network response, join the discussion group provided at the end of this topic to obtain the offline installation package for IntelliJ IDEA from Cloud Toolkit Customer Services.

Install Cloud Toolkit

1. Start IntelliJ IDEA.
2. Install Cloud Toolkit to IntelliJ IDEA.
   - **MacOS system**: On the **Preferences** page, choose **Plugins** from the left-side navigation pane. Search for **Alibaba Cloud Toolkit** and then click **Install**.
   - **Windows system**: Go to the **Plugins** page. Search for **Alibaba Cloud Toolkit** and then click **Install**.
3. After Cloud Toolkit is installed to IntelliJ IDEA, restart IntelliJ IDEA. The Alibaba Cloud Toolkit icon appears in the toolbar.

Configure Cloud Toolkit

After Alibaba Cloud Toolkit is installed, use the AccessKey ID and AccessKey Secret to configure the Cloud Toolkit account.

1. Start IntelliJ IDEA.
2. Set the AccessKey ID and AccessKey Secret.
   i. Click the Alibaba Cloud Toolkit icon and select **Preferences** from the drop-down list. On the Settings page, choose **Alibaba Cloud Toolkit** > **Accounts** from the left-side navigation pane.

ii. On the **Accounts** page, set **Access Key ID** and **Access Key Secret**, and click **OK**.

> **Note** If you use the AccessKey ID and AccessKey Secret of a RAM user, make sure that the RAM user has the permission to deploy applications. For more information about how to grant permissions to RAM users, see RAM account authorization.

- If you already have an Alibaba Cloud account, on the **Accounts** page, click **Get existing AK/SK** to go to the logon page of Alibaba Cloud. After you log on to the system with an existing account, you are redirected to the Security Management page. On this page, obtain the **AccessKeyId** and **AccessKeySecret** of the account.
- If you do not have an Alibaba Cloud account, on the **Accounts** page, click **Sign up**. You are redirected to the Register account page of Alibaba Cloud. On this page, register an Alibaba Cloud account. Then, obtain the **AccessKeyId** and **AccessKeySecret** of the account.

3. Set an endpoint.
   i. On IntelliJ IDEA, click the Cloud Toolkit icon and select **Preferences** from the drop-down list.
   ii. In the **Preferences** dialog box, choose **Appearance & Behavior** > **Endpoint** from the left-side navigation pane.
   iii. On the **Endpoint** page, set the endpoint of EDAS and click **Apply**.

> **Note** To obtain an endpoint, contact EDAS Customer Services.

Deploy applications to EDAS

Currently, you can use Cloud Toolkit to deploy applications to EDAS by using WAR or JAR packages.

1. On IntelliJ IDEA, click the Alibaba Cloud Toolkit icon and select **EDAS on Alibaba Cloud** from the drop-down list.
2. In the **Deploy to EDAS** dialog box, configure the application deployment parameters. Then, click **Apply** to save the configurations.
   i. In the Deploy to EDAS dialog box, select **Region**, **Namespace**, **Application**, and **Group** in the Application section as needed.
      - **Region**: The region where the application is located.
      - **Namespace**: The namespace where the application is located.
      - **Application**: The name of the application.
      - **Group**: The group of the application.
   ii. Set the build mode.
      - **Maven Build**: If this option is selected for building the application, the system adds a Maven task by default to build the deployment package.
      - **Upload File**: If this option is selected for building the application, upload the WAR package or JAR package, and then deploy the application.

> Note If you have not created an application in EDAS, click Create application on EDAS console in the upper-right corner of the dialog box to go to the EDAS console and create an application. For more information about how to create an application, see Deploy Java applications in ECS clusters.

3. Click Run to run the configurations you made in the preceding step. The deployment logs are printed on the Console tab of IntelliJ IDEA. You can view the deployment result based on the logs.

Manage Maven tasks

In Cloud Toolkit installed in IntelliJ IDEA, you can deploy Maven tasks. In the Deploy to EDAS dialog box, you can also add, delete, modify, or move Maven tasks in the **Before launch** section.

In the Select Maven Goal dialog box, click the folder icon on the right of the Working directory field and select all available modules for the current project. Enter the building command in the **Command line** field.

Deploy multi-module projects

Most Maven projects involve multiple modules. These modules can be separately developed and some of them may use the functions of other modules. This type of project is a multi-module project.

If your project is a Maven multi-module project and you want to deploy a submodule in the project, make sure that the last Maven task in the **Before launch** section in the **Deploy to EDAS** dialog box is built for the submodule. For more information about how to manage Maven tasks, see Manage Maven tasks.

For example, the CarShop project has the following submodules:

- carshop
  - itemcenter-api
  - itemcenter
  - detail

Itemcenter and detail are submodules and depend on the itemcenter-api module. In this case, how is the itemcenter submodule deployed? In the **Before launch** section of the Deploy to EDAS dialog box, add the following two Maven tasks:

1. Add a Maven task to run the `mvn clean install` command in the carshop parent project.
2. Add a Maven task to run the `mvn clean package` command in the itemcenter submodule.

1.4.3. Deploy applications in hybrid cloud clusters

Enterprise Distributed Application Service (EDAS) allows you to deploy applications in hybrid cloud clusters and provides complete solutions for scaling, networking, and central management in such clusters. You can connect instances from Alibaba Cloud and machines from on-premises data centers and other cloud service providers (CSPs) by using Express Connect circuits, and add these instances or machines to hybrid cloud (non-Alibaba Cloud) Elastic Compute Service (ECS) clusters in EDAS. Then, you can deploy High-speed Service Framework (HSF), Dubbo, and Spring Cloud applications in a unified manner and manage these applications in a centralized manner in the EDAS console. EDAS supports the auto scaling of ECS instances in Alibaba Cloud.

Prerequisites

- You have created a VPC.
- You have activated Express Connect.
- You have applied for an Express Connect circuit to connect your on-premises data centers to Alibaba Cloud VPC.
- Make sure the machines in your on-premises data centers meet the following requirements:
  - Operating system: CentOS 7
  - Docker: not supported
  - Hardware: no special requirements for CPU and memory

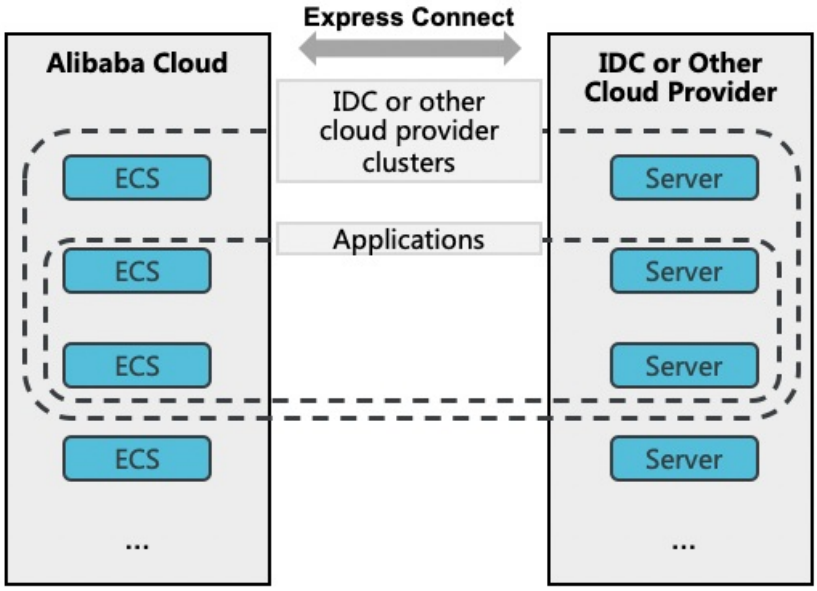
Context

Your application system may have the following requirements and problems:

- The Alibaba Cloud traffic has a certain degree of volatility and you may face traffic spikes in special scenarios. You can predict the traffic volumes in these scenarios, but deviations may exist. It is challenging to know when to add ECS instances and how many ECS instances are needed. In addition, you must purchase ECS instances in advance.
- Some core business systems have high security requirements and you may want to deploy such applications in your own on-premises data center. However, you cannot centrally manage applications that are deployed in different environments. This is because instances from Alibaba Cloud, machines from on-premises data centers, and machines from other CSPs cannot communicate with each other.
- Based on your business needs and availability requirements, you may want to deploy your applications to machines from multiple CSPs. This is the multi-cloud mode. In this mode, manual processing is required since you cannot centrally manage these applications. This often leads to misoperations.

You can deploy applications in a hybrid cloud environment by using the architecture shown in Hybrid Cloud Cluster Architecture.

Hybrid Cloud Cluster Architecture



- Connect Alibaba Cloud to on-premises data centers or to the clouds of other CSPs over Express Connect.
- Create a hybrid cloud cluster. Then, add ECS instances from Alibaba Cloud and machines from on-premises data centers or other CSPs to this cluster.
- Deploy your applications to instances or machines in this cluster.

In an hybrid cloud environment, EDAS is used in the following scenarios:

- Manage applications in on-premises data centers by using Alibaba Cloud. After you connect your on-premises data center to Alibaba Cloud VPC over an Express Connect circuit, you can manage the applications in your on-premises data center by using EDAS.
- Scale in or out applications deployed on instances from Alibaba Cloud. EDAS supports auto scaling and helps you automatically purchase and release instances in Alibaba Cloud. You only need to associate EDAS with your billing account and do not need to buy instances in advance.
- Deploy and manage machines from other CSPs. EDAS allows you to deploy applications to machines from CSPs other than Alibaba Cloud and manage these machines in a centralized manner.

This topic describes how to manage machines from on-premises data centers by using EDAS in Alibaba Cloud. To deploy and manage machines from other CSPs, you only need to connect the target machines to VPCs in EDAS over Express Connect circuits. Then, you can manage these machines in the same way as Alibaba Cloud manages machines from on-premises data centers that have been connected to Alibaba Cloud. For more information about how to scale out applications deployed on ECS instances from Alibaba Cloud, see Scaling (applicable to ECS clusters).

> **Note** Currently, only EDAS Professional Edition and EDAS Enterprise Platinum Edition allow you to deploy applications in hybrid cloud environments.

Procedure

1. Create a cluster.
   i. Log on to the EDAS console. For more information, see Log on to the EDAS console.

ii. In the left-side navigation pane, choose **Resources > Clusters**.

iii. On the **Clusters** page, select a **region** and **namespace**, and click **Create Cluster** in the upper-right corner of the page.

iv. In the **Create Cluster** dialog box, enter the cluster information and click **Create**.

Parameter field description:

- **Cluster Name**: Enter a name for the cluster. The name can contain only letters, numbers, underscores (_), and periods (.) and must be 1 to 64 characters in length.
- **Cluster**: Select **Non-Alibaba Cloud**.
- **Cluster Type**: The default value is ECS, which cannot be changed.
- **Network Type**: The default value is VPC, which cannot be changed.
- **VPC Network**: From the drop-down list, select the VPC where you want to create the cluster.
- **Namespace**: The namespace you selected for the hybrid cluster on the Clusters page, which cannot be edited.

After the cluster is created, **Created successfully** appears in the upper-right corner of the page, and the cluster appears in the cluster list.

2. Add instances to the cluster.

To add ECS instances from Alibaba Cloud and machines from on-premises data centers and other CSPs, perform the following steps:

i. On the Clusters page, click the name of the cluster you just created.

ii. On the Cluster Details page, click **Add an existing ECS**.

iii. In the **Add ECS Instance** dialog box, copy the command for installing EDAS Agent.

iv. Use the **root** account to log on to your **Alibaba Cloud ECS instance** or the **machine in the on-premises data center**.

v. Paste the EDAS Agent installation command and run it.

3. Enable the required ports.

To ensure that your applications in the hybrid cloud cluster can use EDAS normally, you must enable the following ports after you add the instances:

- 8182: This port is used to capture basic monitoring and trace monitoring logs.
- 12200 to 12300: These ports are used for remote procedure calls (RPCs).
- 65000 to 65535: These are web ports.

You must enable the ports based on the instance type.

- ECS instances from Alibaba Cloud: Enable the ports. For more information, see relevant documentation.
- Machines from on-premises data centers or other CSPs: Enable the related ports. For more information, see the corresponding solutions.

4. Check the cluster and instance statuses.

i. Return to the **Clusters** page. In the cluster list, find the cluster you just created and check the values of **Status** and **Instances**.

   If the cluster status is **Normal**, this cluster has been created. If the value of **Instances** is the same as the number of instances that you added, the instance has been added.

ii. Click the name of the target cluster. On the Cluster Details page, check the value of **Instance ID/Name** for the added instance in the ECS Instance section and check **Cluster Status** in the Cluster Information section.

   If the cluster status is **Running**, the instance is running properly.

5. Deploy an application.

   Currently, the hybrid cloud cluster type must be ECS cluster. Therefore, you can deploy applications only in hybrid cloud ECS clusters.

   The method for deploying applications in hybrid cloud clusters is the same as that for deploying applications in ECS clusters. For more information, see the Deploy an application topic.

Result

Wait several minutes until the application is created. After the application is created, you can view the application information on the Basic Information page. On the Basic Information page, click the **Instance Information** tab. Check the running status of the instance. If Status is **Normal**, the application is published.

1.5. Console user guide

1.5.1. Overview page

The Overview page of the Enterprise Distributed Application Service (EDAS) console displays the subscription type, runtime status, and number of application instances under the current account, allowing you to intuitively know the resource status of the account.

- **Applications**: the number of applications that you publish in EDAS.
- **Application Instances**: the number of instances on which your applications are deployed.
- **Services**: the number of services included in your applications.
- **Deployments in the Last 7 Days**: the number of times applications were deployed during the past seven days.

1.5.2. Resource management

This topic describes EDAS resources and how to use and manage the resources.

In the EDAS console, you can view and use resources, such as ECS and Server Load Balancer (SLB) instances. The EDAS resource management function allows you to use the resources by application. EDAS also supports resource group management. When EDAS is used by multiple users or departments, the permissions to use resources can be controlled by using a primary Alibaba account and its RAM users.

1.5.2.1. Import ECS instances

Before you deploy applications by using Enterprise Distributed Application Service (EDAS), you need to import Elastic Compute Service (ECS) instances to a specified cluster and install EDAS Agent.

Prerequisites

EDAS Agent must be installed on each target ECS instance. Before you install EDAS Agent, ensure that the RAM user is authorized. For information about how to perform authorization, see the "RAM" topic in *ASCM Console User Guide .*

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Resources** > **ECS**.
3. On the **ECS** page, click **Import ECS** in the upper-right corner.
4. In the **Select Cluster and ECS** step, select a namespace from the **Namespace** drop-down list and select a cluster from the **Select Cluster to Import** drop-down list. In the instance list, select an ECS instance and click **Next**.
5. In the **Ready to Import** step, select **I agree to convert the above instances, and fully understand that the data in the original systems will be lost after conversion.** Then, enter a **new password** for the root user, **confirm the new password**, and click **Confirm and Import**.
6. In the **Import** step, view the import progress.In the Import step, the import progress of the ECS instance is **Converting now**. This conversion might take 5 minutes. If you click "Click to return to the Cluster Details page" before the import is completed, the health check status shows **Converting** and the conversion progress is shown as a percentage. When the health check status changes to **Running**, the instance is imported.

Result

Click **Click to return to the Cluster Details page** to go to the Cluster Details page. In the **ECS Instance** section, view the import status and progress.

1.5.2.2. View a VPC

VPCs are created in the VPC console. After synchronizing resources in the EDAS console, you can view VPC information.

Context

VPCs are virtual private clouds that allow custom isolation settings. You can define the custom VPC topology and IP address. VPCs are suitable for customers with high cybersecurity requirements and network management capability.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Resource Management** > **VPC** to go to the **VPC** page and view the VPC information and status.

Instance information

Name	Description
VPC ID	The ID that is automatically generated when a VPC is created.
Name	The name that you set when creating a VPC.
CIDR	VPC statuses include Running and Stopped. Expired VPCs do not appear.
Status	The status of an SLB instance, which may be Running or Stopped. Expired SLB instances do not appear.
ECS Instance	The number of ECS instances created in this VPC. Click the number to go to the ECS page, where you can view all the ECS instances in this VPC.

What's next

In the VPC, ECS instances are isolated from the EDAS server. You need to install a log collector to collect ECS instance information. Locate the row that contains the target instance, and click **Install Log Collector** in the **Actions** column.

1.5.2.3. Manage clusters

A cluster is a set of ECS instances necessary to deploy applications. Cluster management mainly includes creating clusters, viewing clusters, and managing cluster hosts.

1.5.2.3.1. Create an ECS cluster

Create a cluster before publishing applications.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Resource Management** > **Clusters**.
3. On the **Clusters** page, click **EDAS Cluster**. On the **EDAS Cluster** tab page, click **Create Cluster** in the upper-right corner.
4. In the **Create Cluster** dialog box, set the cluster parameters and click **Create**.

Cluster parameters

Name	Description
Cluster Name	Enter a name for the cluster. The name can only contain letters, numbers, underscores (_), and periods (.), with a length up to 64 characters.
Cluster	The options are **Alibaba Cloud** and **Non-Alibaba Cloud**. Select **Alibaba Cloud** in this case. Select **Non-Alibaba Cloud** when creating a hybrid cloud cluster.
Cluster Type	Currently, only ECS clusters are supported.

Name	Description
Network Type	Only VPC is supported.
VPC	Select a specific VPC.
Namespace	A namespace has been selected on the **Clusters** page, so this parameter cannot be set here.

Result

After the cluster is created, the message **Cluster created successfully** appears in the upper-right corner of the page, and the cluster appears in the cluster list and is in the **Normal** state.

What's next

Add ECS instances after the cluster is created.

1. On the Cluster Details page, click **Add ECS Instance** in the upper-right corner.
2. On the **Add ECS Instance** page, click **Import ECS** or **From Existing Cluster** to add ECS instances.
   - **Import ECS**: See Import ECS instances.
   - **From Existing Cluster**: In the current region, select a **namespace** and **source cluster**. In the ECS instance list, select ECS instances and click > to add them to the field on the right. Then, click **Next**. The subsequent procedure is the same as that for importing ECS instances.
3. After ECS instances are added, return to the Cluster Details page to view the health status of the ECS instances. The ECS instances are successfully added if the health status is **Normal**.

1.5.2.4. Manage resource groups

Resource groups are groups of Enterprise Distributed Application Service (EDAS) resources. You can use resource groups to control account permissions. You can grant resource group access permissions to Resource Access Management (RAM) users, and each RAM user has the permission to operate on all the resources in the specified group.

Typical scenarios

- A company uses EDAS to create business applications. Department A is responsible for user-related applications and Department B for goods-related ones.
- The company registers an EDAS account (the Apsara Stack tenant account) to activate EDAS and creates two RAM users for Departments A and B.
- Departments A and B have dedicated Elastic Compute Service (ECS) and Server Load Balancer (SLB) instances for deploying user-related applications and goods-related applications, respectively.
- You have created two resource groups in EDAS and bound them to the resources of Departments A and B, respectively. Then, you grant the RAM users of Departments A and B the permissions to access the two resource groups, respectively.

- Department A uses its RAM user only to operate the resources in the authorized resource group. Department B does the same for its resource group. There is no conflict between Departments A and B during resource management.

Create a resource group

1. In the left-side navigation pane of the console, choose **Resources** > **Resource Groups**.
2. On the **Resource Groups** page, click **Create Resource Group** in the upper-right corner.
3. In the **Create Resource Group** dialog box, enter **Resource Group Name** and **Resource Group Description**, and click **Confirm**.

After the resource group is created, you can edit or delete it as needed.

Bind resources to resource groups

You can bind ECS instances, SLB instances, and clusters to resource groups. The procedures for binding different types of resources are similar. This topic describes how to bind ECS instances.

1. On the **Resource Groups** page, find the target resource group, and click **Bind ECS** in the Actions column.
2. In the **Bind ECS** dialog box, select one or more ECS instances and click **Confirm**.

Grant RAM users the permissions to access resource groups

You can grant RAM users the permissions to access specified resource groups.

1. Log on to the EDAS console by using your Apsara Stack tenant account.
2. In the left-side navigation pane, choose **System Management** > **Sub-Accounts**.
3. Find the target user and click **Resource Group Permission** in the Actions column.
4. In the **Resource Group Permission** dialog box, select a resource group and click **Confirm**.

1.5.3. Manage applications

In the EDAS console, you can perform application lifecycle management, O&M, monitoring, and service governance.

1.5.3.1. Namespaces

You can use namespaces to isolate resources in different environments and use the same account to centrally manage them.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose Application Management**Namespaces**.
3. On the **Namespaces** page, select a **region** and click **Create Namespace** in the upper-right corner.
4. In the **Create Namespace** dialog box, enter **Namespace Name** and **Namespace ID**, select whether to enable **Allow Remote Debugging**, and click **Create**.

Result

On the **Namespaces** page, view the created namespace.

1.5.3.2. Lifecycle management for applications in ECS clusters

Applications are the basic units for EDAS management. A single application contains a group of instances on which the same application is deployed. EDAS provides a comprehensive application lifecycle management mechanism, covering the entire process from application publishing to operation, including application creation, deployment, startup, rollback, scaling, stop, and deletion.

Application lifecycle management includes application publishing, management, and configuration.

- Application publishing includes application creation, deployment, start, and stop.
- Application management includes application rollback, scale-out, scale-in, and deletion and instance reset and deletion.
- Application configuration includes container, JVM parameter, SLB, and health check configuration.

> **Note**
>
> - You can deploy, scale out, roll back, reset, and configure an application no matter if the application is running or stopped.
> - After the parameters of the Tomcat container and JVM are set and saved, the related configuration files are modified. The changes take effect only after you restart the application.

1.5.3.2.1. Publish an application

This topic describes how to publish an application in the EDAS console, helping you quickly familiarize yourself with EDAS operations and application publishing.

Note

- If your EDAS service is deployed in Sugon, you can create an application in the Apsara Stack console or EDAS console.
  - If you create an application in the Apsara Stack console, your RAM user is authorized by default.
  - If you create an application in the EDAS console, you need to authorize your RAM user manually. For more information, see Use a primary account for RAM user operation.

  If required authorization is not performed in the EDAS console, an exception may occur when you manage applications in the Apsara Stack console.
- Applications must be managed in the EDAS console.
- If you publish an HSF application, create a service group before starting the application. Otherwise, application publishing may fail due to failed authentication. In the EDAS console, choose **Service Market** > **Service Groups** from the left-side navigation pane. On the page that appears, click **Create Service Group** in the upper-right corner to create a service group. The service group name must be globally unique. After the service group is created, restart the application to allow the service group to take effect.

1.5.3.2.1.1. Create an empty application (applicable to ECS clusters)

You can create an empty application during the planning phase and then deploy packages on the application subsequently.

Prerequisites

An Elastic Compute Service (ECS) cluster has been created. For more information, see Create an ECS cluster.

Context

You can create an empty application in either of the following two states:

- Empty application without instances: an empty application that is configured only with basic information, including a region, namespace, cluster, application name, deployment method, and runtime environment.
- Empty application with instances: an empty application that is configured with basic information (including a region, namespace, cluster, application name, deployment method, and runtime environment) and with ECS instances added.

This topic describes how to create an empty application with ECS instances.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management** > **Applications**.
3. On the **Applications** page, click **Create Application** in the upper-right corner.

4. On the **Application Information** page, set the parameters of the application. Then, click **Next**.

   ### Basic information and parameters of the application

Parameter	Description
Namespace	Select a namespace from the drop-down list.
Cluster Type	From the first drop-down list, select **ECS Cluster**. From the second drop-down list, select a specific cluster.
Application Name	Enter an application name, which must be 1 to 36 characters in length.
Deployment Method	Select **WAR** or **JAR** based on the application.
Application Runtime Environment	Select the application runtime environment based on the application framework. ○ For High-Speed Service Framework (HSF) applications, select **EDAS-Container**. ○ For Spring Cloud or Dubbo applications, ■ **WAR**: Select **Apache Tomcat**. ■ **JAR**: Select **Standard Java application runtime environment**.
Java Environment	Select **Open JDK 8**.
Application Description	Enter remarks for the application.

5. (Optional)At the bottom of the page, click **Create an Empty Application** to create an empty application without instances.
6. On the **Application Configuration** page, click **Add** to the right of **Selected Instances**.

   > **Note** If no instances are added, you can click **Create an Empty Application** to create an empty application without instances.

7. In the **Instances** dialog box, select an ECS instance and click **>** to add the instance to the right-side section. Then, click **OK**.
8. Return to the **Application Configuration** page and click **Create**.
   Wait several minutes until the application is created.

Result

Return to the Application Details page to view the statuses of the application and instances.

- An application without instances is an application that contains basic information, including the application name, ID, namespace, and deployment package type, but it does not contain instance information.
- An application with instances is an application that contains basic information (including the application name, ID, namespace, and deployment package type) and also contains instance information and status.

What's next

You can deploy the application after it is created. For more information, see Deploy an application (applicable to ECS clusters).

1.5.3.2.1.2. Deploy an application (applicable to ECS clusters)

You can deploy an empty application after it is created. After the application is deployed, you can upgrade the application by redeploying the application.

Prerequisites

- You have created an empty application. For more information, see Create an application (applicable to ECS clusters).
- You have created a Server Load Balancer (SLB) instance. For more information, see *SLB User Guide* .

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management** > **Applications**.
3. On the **Applications** page, click the name of the created empty application.
4. On the Application Details page, click **Deploy Application** in the upper-right corner.
5. On the **Deploy Application** page, set deployment parameters and click **Deploy**.

   Deployment parameters:

Parameter	Description
Deployment Method	Select **WAR** or **JAR**. **Note** Your deployment method has been determined and cannot be changed. The configuration processes for WAR package deployment and JAR package deployment are similar. Here, **WAR package deployment** is used as an example.
File Uploading Method	Select **Upload WAR Package** or **WAR Package Location**. ○ **Upload WAR Package**: Click **Select File** to the right of **Upload WAR Package**, and select a local WAR package for uploading. ○ **WAR Package Location**: Enter the path of the WAR package. **Note** The name of the application deployment package can only contain letters, digits, hyphens (-), and underscores (_).

Parameter	Description
Version	Enter a version number, for example, 1.1.0. Note We do not recommend that you use a timestamp as the version number.
Group	Select the instance group where the application is deployed.
Batch	Specify a number of deployment batches. Select an option from the drop-down list. The options are automatically generated based on the number of instances for the application. If you select two or more batches, you must set **Batch Wait Time**.
Batch Mode	Select **Automatic**.
Java Environment (optional)	Select the runtime environment of the application from the drop-down list.

Go to the **Change Details** page to view the task progress and logs of the application deployment.

Result

1. On the Application Details page, check whether the deployment package is of the new version.
2. Click the **Instance Information** tab to check whether **Running Status** of the ECS instance is **Normal** and whether **Change Status** is **Successful**.

1.5.3.2.2. Manage applications

You can manage a published application in the Enterprise Distributed Application Service (EDAS) console. This includes viewing application information, upgrading, starting, stopping, scaling out, and scaling in the application, creating branch versions, upgrading container versions, and rolling back and deleting the application. If the application is deployed on Elastic Compute Service (ECS) instances, you need to manage these ECS instances.

This topic briefly describes some simple management operations.

1.5.3.2.2.1. Scale out and scale in applications (ECS clusters)

When the traffic is heavy and the application load is high, you can add Elastic Compute Service (ECS) instances to scale out an application and share the load. When the traffic is low and the load of an instance is low, you can remove the instance to scale in the application and reduce the cost.

Scale out an application

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management** > **Applications**.
3. On the **Applications** page, select a **region** and **namespace**, and then click the name of the target application.
4. On the Basic Information page, click **Scale Out** in the upper-right corner.
5. On the **Scale-Out Method** tab of the **Add an Instance** dialog box, select **Target Group** and the target ECS instance, and then click **Scale Out**.

> **Note**
>
> The runtime status of the added ECS instance depends on the runtime status of the application on the instance.
>
> - If the application is running, the added instance automatically deploys, starts, and runs the application.
> - If the application is stopped, the added instance deploys the application but does not start or run the application.

Scale in an application

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management** > **Applications**.
3. On the **Applications** page, select a **region** and **namespace**, and then click the name of the target application.
4. On the Basic Information page, click the **Instance Information** tab.
5. On the **Instance Information** tab, delete the target instance.
   - If the ECS instance is in the **Normal** state, click **Stop** and then **Delete**.
   - If the ECS instance is in the **Stop** state, click **Delete**.

1.5.3.2.2.2. Create branch versions of an application

When you create an application, Enterprise Distributed Application Service (EDAS) automatically creates an application group named "Default Group" for the application and adds the Elastic Compute Service (ECS) instances of the application to this group. You can create new groups under the default group and add some instances to the new groups. If you deploy different versions of the application on the instances in the new groups, these versions of the application are the branch versions of the application.

Context

You can create branch versions if you have the following requirements for your application:

- Online test before you publish a new version
- A/B testing
- Canary release

Procedure

1. Create a group.

i. Log on to the EDAS console.

ii. In the left-side navigation pane, choose **Application Management > Applications**.

iii. On the **Applications** page, click the name of the created empty application.

iv. On the Basic Information page, click the **Instance Information** tab. On the tab that appears, click **Create Group** in the upper-right corner.

v. In the **Create Group** dialog box, enter a **group name** and click **Create**.

After the group is created, the message `Group created successfully` appears in the upper-right corner of the page.

2. Add instances to the new group.

After the group is created, you can add instances to the new group in two ways: **scale out** and **change group**. For more information about application scale-out methods, see Scaling (applicable to ECS clusters). This topic describes how to add instances from the default group to the new group by changing the group.

i. On the **Instance Information** tab of the Basic Information page, find the target instance, and click **Change Group** in the Actions column.

ii. In the **Change Group** dialog box, set **Target Group**.

iii. Click **Confirm**.

> Note
>
> - If no application is deployed in the new group while an application deployment package has been deployed on the added instance, this deployment package is deployed in the group.
> - If an instance is added to an existing group rather than a new group, the versions of the deployment package in the group and on the instance are different. When the system displays the following messages, select the appropriate option as needed:
>   - Select **Re-deploy the current instance using the target group's version** to redeploy the deployment package on the instance using that in the group.
>   - Select **Change Group Only Without Re-deployment** to add the instance without changing its deployment package.

3. Deploy the application in the new group.

i. On the Basic Information page, click **Deploy Application** in the upper-right corner.

ii. Set **Group** to the target new group, set the deployment parameters, and click **Deploy**.

Result

On the **Instance Information** tab of the Basic Information page, you can view the deployment package version and running status of the new group to check whether the new application version is published.

1.5.3.2.2.3. Upgrade the container version

WAR and JAR packages are used for application deployment. The deployment involves an application runtime environment and EDAS Container. You can upgrade EDAS container to the specified version.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management > Applications**. On the **Applications** page, click the target application name.
3. In the left-side navigation pane of the Basic Information page, choose **Container Version**.
4. On the **Container Version** page, view the current version of the EDAS container for the application.

   The current version is marked with a tick (√) in the **Actions** column. The **Actions** column also displays the availability status of other versions.
5. Click the corresponding button in the **Actions** column to update the container to the target version.

1.5.3.2.2.4. Roll back an application

To roll back a published application to an earlier version, you can use the **application rollback** feature and select the target version.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management > Applications**.
3. On the **Applications** page, click the target application name.
4. On the Basic Information page, click **Roll Back** in the upper-right corner.
5. On the **Roll Back** page, select **Deployment Package Version** and **Group** for the rollback, set **Batch** and **Batch Mode**, and then click **Roll Back**.

   > **Note** You can view a maximum of five deployment package versions for rollback.

1.5.3.2.2.5. Delete an application

After an application is deleted, all information related to the application is deleted, all instances under the application are released, and all deployment packages and container files on the instances are deleted.

Prerequisites

Before you delete an application, be sure to save the logs, WAR packages, and configurations of all instances in the application.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management > Applications**.

3. On the **Applications** page, click the target application name.
4. On the **Basic Information** page of the application, click the **Instance Information** tab.
5. On the **Instance Information** tab, find the instance and click **Stop** in the **Actions** column.
6. After the instance status changes to **Stop**, click **Delete**.
   After the application is deleted, the message `Deleted successfully` appears in the upper-right corner of the page.

1.5.3.2.3. Application settings

On the Application Settings page, you can set the JVM parameters, Tomcat, SLB, and health check of applications.

1.5.3.2.3.1. Set JVM parameters

You can set JVM parameters to enable the container parameter setting when an application is started in Enterprise Distributed Application Service (EDAS). Correctly, setting JVM parameters helps reduce the overhead of garbage collection (GC) and shorten the server response time and improve throughput.

Context

- If no container parameters are set, JVM parameters are allocated by default.
- The configured JVM parameters take effect only after you manually restart the application.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management > Applications**.
3. On the **Applications** page, select a **region** and **namespace**, and then click the name of the target application.
4. In the **Application Settings** section of the **Basic Information** tab, click **Settings**, and select **JVM** from the drop-down list.
5. In the **Application Settings** dialog box, click to expand **Memory Configuration**, **Application**, **GC Policy**, **Tool**, and **Custom** respectively to set related parameters and then click **Save**.

   > **Note**
   > - You can move the pointer over **i** to the right of a specific parameter to learn the meaning of this parameter.
   > - The JVM parameter settings are written in the *bin/setenv.sh* file in the container directory. To apply the settings, restart the application.

Result

After the settings are completed, the message `JVM parameters successfully configured` appears in the upper-right corner of the page.

1.5.3.2.3.2. Configure Tomcat

EDAS supports Tomcat container parameter settings. You can configure settings such as the port number, application access path, and the number of connections in the connection pool of the Tomcat container in the EDAS console.

Prerequisites

Note

- After setting Tomcat container parameters, restart the container to apply the parameter settings.
- Tomcat container configuration is supported by EDAS Agent 2.8.0 and later.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management**. On the **Applications** page, click the name of the target application.
3. On the Application Details page, click **Settings** on the right of the **Application Settings** section.
4. In the **Application Setting** dialog box, click the **Tomcat** tab and set Tomcat parameters. Then, click **Save**.

Tomcat configuration description

Name	Description
Application Port	The port range is (1024, 65535). The admin authority is needed for container configuration and the root authority is required to operate on ports with numbers less than 1024. Therefore, enter a port number greater than 1024. If this parameter is not set, the default value 8080 is used.
Tomcat Context	The access path of an application. ◦ If you select **Package Name**, you do not need to set a custom path. The default value is the WAR package name. ◦ If you select **Root**, you do not need to set a custom path. The default value is a slash (/). ◦ If you select **Custom**, enter a custom path below, namely, the access path. If this parameter is not set, the default value is the WAR package name.
Maximum Threads	The maximum number of connections in a connection pool. The parameter is maxThreads. The default value is 400. We recommend that this parameter be set under professional guidance.
Tomcat encoding	Select an encoding format for Tomcat: UTF-8, ISO-8859-1, GBK, or GB2312. The default format is ISO-8859-1. Select useBodyEncodingForURI as needed.

> **Note** Click **Advanced Settings** to configure the full text of server.xml. The application groups use the application configuration after the advanced settings are enabled.

1.5.3.2.3.3. Bind an SLB instance to EDAS

In an Elastic Compute Service (ECS) cluster, you can bind an application to a Server Load Balancer (SLB) instance to implement load balancing.

Scenarios

Dedicated SLB instance for an application

You have an application that provides order query and contains multiple ECS instances. You want the application to provide a public IP address for external access. In this case, you can bind an SLB instance to the application to achieve this purpose.

The following diagram shows the simple mapping between the SLB instance and the application in the preceding scenario.

Dedicated listening port for an application to distribute traffic

You have application A that provides order query and application B that provides user logon. Both applications are accessed using the same public IP address and are bound to the same domain name. You can distribute traffic by binding different listening ports of the same SLB instance to the two applications.

The following diagram shows the simple mapping between the SLB instance and the application in the preceding scenario.

An SLB instance shared by different domain names

In the Internet scenario, port 80 is selected by default to provide the HTTP service externally. If you want to use an SLB instance to distribute traffic, a common solution is to use domain names to route data to different applications. Assume that u.domain.com is the domain name that is bound to the user application, and o.domain.com is the domain name that is bound to the order application.

The following diagram shows the simple mapping between the SLB instance and the application in the preceding scenario.

Set SLB rules in an application group

For example, in the flash sales scenario, the number of visits to the URL (o.domain.com/orders/queryitem) that queries product information in the order system is significantly higher than that to URLs that provide other services and functions. We hope to transfer the high service traffic of the same type to a separate group of instances, which provide the order query service while instances in other groups provide other services.

The following diagram shows the topology.

Prerequisites

- You have Create an application (applicable to ECS clusters) in the EDAS console.
- If you do not have an SLB instance, go to the SLB console to create an SLB instance.
- To configure forwarding rules for a deployed group, you have set an application group for the

application.

Bind an SLB instance to an application in the EDAS console

1. In the **Application Settings** section of the **Basic Information** page, click **Add** on the right of **SLB (Internet)**.

   > **Note** If you have configured an SLB instance, the IP address and port number of the SLB instance are displayed. You can click Modify to go to the configuration page and modify the information of the SLB instance. You can also click Unbind to unbind the SLB instance.

2. In the **Bind SLB to Application** dialog box, select the SLB instance, and then click **Next**.
3. On the **Select configuration listener** tab, select the configuration listener protocol and listener port, and then click **Next**.
   - You can select an existing listener from **Select an existing listening port**.
   - You can also create a listener in **Add a new listening port**. For example, set the listener protocol to HTTP and the frontend port number to 82.
4. On the Configuring virtual grouping and forwarding policies tab, set the bound server group. Click **Next**.
   - You can choose **Default server group** to bind all the servers under this application to the default server group of the SLB instance.
   - You can also select a virtual server group from the **Existing virtual server group** list.
   - You can also enter a virtual server group name in **New virtual server group** to create a virtual server group as the bound server group.
5. On the Confirm change SLB tab, view the change information of the SLB instance, and click **Confirm change** to complete the configuration.

Results

Copy the configured IP address and port number of the SLB instance such as 118.31.XXX.XXX:81, paste it in your browser address bar, and press Enter to go to the homepage of the application.

If the IP address and port number do not appear on the right side of the SLB instance, the binding failed. Go to Change Logs to view the change details, and troubleshoot and fix the failure based on the change logs.

FAQ

After I bound an SLB instance with forwarding rules to an application Group and then unbound the SLB instance from the application Group, the application cannot be accessed through SLB traffic. What can I do if the HTTP Code 503 error is reported?

Cause: You entered the forwarding rules and bound the SLB instance to the application group, and then unbound the SLB instance in the EDAS console. In this case, EDAS did not delete the forwarding rules of the SLB instance. Since you unbound the SLB instance from the application group in the EDAS console, the servers in the virtual SLB group were also unbound. As a result, the SLB traffic forwarding failed and the HTTP 503 error was reported.

Solution: Manually delete the forwarding rules in the SLB console.

Why does an application bound to an SLB instance fail to be accessed through the IP address of

the SLB instance after traffic management is enabled?

Cause: In this scenario, traffic management was enabled and the application was bound to an SLB instance (over HTTP). At this time, there is a limitation. When the SLB instance used HTTP to detect whether the backend nodes were alive, the message HEAD/HTTP/1.0 was sent, and Tengine responded with HTTP 400, which caused the failure of SLB listener health check. As a result, the error code 502 (bad gateway) was reported when you accessed the application.

Solution: Disable traffic management on the Application Information page of the EDAS console for applications that do not require traffic management. This will uninstall Tengine, modify the application configurations, and restart the application. To retain this function, you can select Code 4xx returned by health check in Health Check of the SLB instance.

1.5.3.2.3.4. Set JVM -D startup parameters

This topic describes how to set JVM -D startup parameters when developing High-Speed Service Framework (HSF) applications.

-D `hsf.server.port`

Specifies the port for starting HSF services. The default value is 12200. Use another port than the default port if you start multiple HSF providers locally.

-D `hsf.server.max.poolsize`

Specifies the maximum size of the thread pool of the HSF provider. The default value is `720`.

-D `hsf.server.min.poolsize`

Specifies the minimum size of the thread pool of the HSF provider. The default value is `60`.

-D `hsf.client.localcall`

Enables or disables the precedence of calling local HSF clients. The default value is `true`.

-D `pandora.qos.port`

Specifies the Pandora monitoring port. The default value is `12201`. Use another port than the default port if you start multiple HSF providers locally.

-D `hsf.http.enable`

Specifies whether to enable the HTTP port. The default value is `true`.

-D `hsf.http.port`

Specifies the HTTP port used by the HSF application to provide services externally. The default value is `12220`. Use another port than the default port if you start multiple HSF providers locally.

-D `hsf.run.mode`

Specifies whether the HSF consumer performs a targeted call, that is, bypassing Config Server. The value `1` indicates that a targeted call is disallowed, and the value `0` indicates a targeted call is allowed. The default value is `1`. Do not set this parameter to `0` unless necessary.

-D `hsf.shuthook.wait`

The wait time for gracefully disconnecting an HSF application, in ms. The default value is `10000`.

-D `hsf.publish.delayed`

Specifies whether to delay publishing all services. The default value is false, indicating not to delay service publishing.

-D `hsf.server.ip`

Specifies the IP address to be bound. By default, the IP address of the first network interface controller (NIC) is bound when multiple NICs exist.

-D `HsfBindHost`

Specifies the host to be bound. By default, the HSF server binds the IP address of the first NIC and reports it to the address registry when multiple NICs exist. If you set this parameter to `-DHsfBindHost=0.0.0.0`, the HSF server port is bound to all NICs of the local device.

-D `hsf.publish.interval=400`

Specifies the time interval between the publishing of two services. HSF services are instantly exposed when being published. You can set this parameter to mitigate the burden on starting applications during service exposure. The default value is 400, in ms.

-D `hsf.client.low.water.mark=32` -D `hsf.client.high.water.mark=64` -D `hsf.server.low.water.mark=32` -D `hsf.server.high.water.mark=64`

Specifies the write buffer limit for each channel of the consumer or provider.

- The unit is KB. When the consumer exceeds the upper limit, the channel forbids writing new requests and returns an error. Writing is resumed when the write buffer drops below the lower limit.
- When the provider exceeds the upper limit, the channel forbids writing new responses, and the consumer times out because no response is received. Writing is resumed when the write buffer drops below the lower limit.
- The upper and lower limits must be set as a pair, and the upper limit must be greater than the lower limit.

-D `hsf.generic.remove.class=true`

Retrieves the result of a generic call, without output of the `class` field.

-D `defaultHsfClientTimeout`

Specifies the global time-out period of the consumer.

-D `hsf.invocation.timeout.sensitive`

Determines whether the HSF call duration includes the time consumption logic such as connection creation and address selection. The default value of `hsf.invocation.timeout.sensitive` is false.

1.5.3.3. Lifecycle management for Container Service Kubernetes applications

1.5.3.3.1. Container Service Kubernetes clusters

Kubernetes is a popular orchestration technology for open source containers. Kubernetes-published applications have unique management advantages. For more information, see the Kubernetes official documentation.

A Container Service Kubernetes cluster is a Kubernetes cluster that is provided by Alibaba Cloud and has passed the CNCF standardized test. It runs stably and integrates other Alibaba Cloud services, such as SLB and Network Attached Storage (NAS). After creating a Kubernetes cluster in Container Service and importing it to EDAS, you can deploy applications to the Container Service Kubernetes cluster in EDAS.

1.5.3.3.2. Prepare an application image (a Container Service Kubernetes cluster)

EDAS allows you to deploy RPC applications (HSF) in Container Service Kubernetes clusters by using custom images (Dockerfile).

Observe the following specifications and limits when creating a custom image by using a Dockerfile:

Tenant and encryption information

The tenant and encryption information is used for user authentication and credential encryption of EDAS applications.

Resources

Resource type	Resource name	Description
Secret	edas-certs	An encryption dictionary that stores EDAS tenant information.

Environment variables

Environment variable key	Type	Description
tenantId	String	The ID of an EDAS tenant.
accessKey	String	The AccessKeyId for authentication.

Environment variable key	Type	Description
secretKey	String	The AccessKeySecret for authentication.

Local files

Path	Type	Description
/home/admin/.spas_key/default	File	The authentication information of an EDAS tenant, including the preceding environment variable information.

Service information

The service information includes the EDAS domain and port to be connected during runtime.

Resources

Resource type	Resource name	Description
ConfigMap	edas-envs	EDAS service information

Environment variables

Environment variable key	Type	Description
EDAS_ADDRESS_SERVER_DOMAIN	String	The service domain or IP address of the configuration center.
EDAS_ADDRESS_SERVER_PORT	String	The service port of the configuration center.
EDAS_CONFIGSERVER_CLIENT_PORT	String	The port of ConfigServer.

(Mandatory) Environment variables during application runtime

The following environment variables are provided during EDAS deployment to ensure the proper running of applications. For this reason, do not overwrite the current configuration.

Environment variables

Environment variable key	Type	Description
POD_IP	String	The IP address of a pod.
EDAS_APP_ID	String	The ID of an EDAS application.
EDAS_ECC_ID	String	EDAS ECC ID

Environment variable key	Type	Description
EDAS_PROJECT_NAME	String	Same as EDAS_APP_ID and used for trace parsing.
EDAS_JM_CONTAINER_ID	String	Same as EDAS_ECC_ID and used for trace parsing.
EDAS_CATALINA_OPTS	String	The CATALINA_OPTS parameter required during middleware runtime.
CATALINA_OPTS	String	The default startup parameter of Tomcat, which is the same as EDAS_CATALINA_OPTS.

Procedure

1. Define a standard Dockerfile.

   A standard Dockerfile defines the EDAS application runtime environment, including the definitions of download, installation, JDK startup, Tomcat, and WAR and JAR packages.

   By modifying the Dockerfile, you can replace the JDK version, modify the Tomcat configuration, change the runtime environment, and make other changes.

   The following example shows how to define an EDAS application.

   > **Note** The example will be occasionally updated to incorporate the latest EDAS features.

   - Sample Dockerfile that uses Tomcat and a WAR package

```
FROM centos:7
MAINTAINER EDAS development team <edas-dev@list.alibaba-inc.com>
# Install and package the required software.
RUN yum -y install wget unzip
# Prepare JDK and Tomcat system variables.
ENV JAVA_HOME /usr/java/latest
ENV CATALINA_HOME /home/admin/taobao-tomcat
ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/bin
# Set the EDAS-Container version.
ENV EDAS_CONTAINER_VERSION V3.5.0
LABEL pandora V3.5.0
# Download and install JDK 8.
RUN wget http://edas-hz.oss-cn-hangzhou.aliyuncs.com/agent/prod/files/jdk-8u65-linux-x64.r
pm -O /tmp/jdk-8u65-linux-x64.rpm && \
   yum -y install /tmp/jdk-8u65-linux-x64.rpm && \
   rm -rf /tmp/jdk-8u65-linux-x64.rpm
# Download and install Ali-Tomcat 7.0.85 to the /home/admin/taobao-tomcat.
RUN wget http://edas-hz.oss-cn-hangzhou.aliyuncs.com/edas-container/7.0.85/taobao-tomcat
-production-7.0.85.tar.gz -O /tmp/taobao-tomcat.tar.gz && \
   mkdir -p ${CATALINA_HOME} && \
   tar -xvf /tmp/taobao-tomcat.tar.gz -C ${CATALINA_HOME} && \
   mv ${CATALINA_HOME}/taobao-tomcat-production-7.0.59.3/* ${CATALINA_HOME}/ && \
   rm -rf /tmp/taobao-tomcat.tar.gz ${CATALINA_HOME}/taobao-tomcat-production-7.0.59.3 && \
   chmod +x ${CATALINA_HOME}/bin/*sh
# Download and install an EDAS container based on environment variables.
RUN wget http://edas-hz.oss-cn-hangzhou.aliyuncs.com/edas-plugins/edas.sar. ${EDAS_CONT
AINER_VERSION}/taobao-hsf.tgz -O /tmp/taobao-hsf.tgz && \
   tar -xvf /tmp/taobao-hsf.tgz -C ${CATALINA_HOME}/deploy/ && \
   rm -rf /tmp/taobao-hsf.tgz
# Downloads and deploys the EDAS demo WAR package.
RUN wget http://edas.oss-cn-hangzhou.aliyuncs.com/demo/hello-edas.war -O /tmp/ROOT.war
&& \
   unzip /tmp/ROOT.war -d ${CATALINA_HOME}/deploy/ROOT/ && \
   rm -rf /tmp/ROOT.war
# Set the Tomcat installation directory as the container startup directory, start Tomcat in run m
ode, and output the catalina log in the standard CLI.
WORKDIR $CATALINA_HOME
CMD ["catalina.sh", "run"]
```

- Sample Dockerfile that uses a JAR package

```
FROM centos:7
MAINTAINER EDAS development team <edas-dev@list.alibaba-inc.com>
# Install and package the required software.
RUN yum -y install wget unzip
# Prepare JDK and Tomcat system variables.
ENV JAVA_HOME /usr/java/latest
ENV CATALINA_HOME /home/admin/taobao-tomcat
ENV PATH $PATH:$JAVA_HOME/bin
# Set the EDAS-Container version.
ENV EDAS_CONTAINER_VERSION V3.5.0
LABEL pandora V3.5.0
# Download and install JDK 8.
RUN wget http://edas-hz.oss-cn-hangzhou.aliyuncs.com/agent/prod/files/jdk-8u65-linux-x64.r
pm -O /tmp/jdk-8u65-linux-x64.rpm && \
   yum -y install /tmp/jdk-8u65-linux-x64.rpm && \
   rm -rf /tmp/jdk-8u65-linux-x64.rpm
# Download and install an EDAS container to /home/admin/taobao-tomcat based on environme
nt variables.
RUN mkdir -p ${CATALINA_HOME}/deploy/
RUN wget http://edas-hz.oss-cn-hangzhou.aliyuncs.com/edas-plugins/edas.sar. ${EDAS_CONT
AINER_VERSION}/taobao-hsf.tgz -O /tmp/taobao-hsf.tgz && \
   tar -xvf /tmp/taobao-hsf.tgz -C ${CATALINA_HOME}/deploy/ && \
   rm -rf /tmp/taobao-hsf.tgz
# Download and deploy the EDAS demo JAR package.
RUN mkdir -p /home/admin/app/ && wget http://edas.oss-cn-hangzhou.aliyuncs.com/demoapp
/fatjar-test-case-provider-0.0.1-SNAPSHOT.jar -O /home/admin/app/provider.jar
# Include the startup command in the startup script start.sh.
RUN echo '$JAVA_HOME/bin/java -jar $CATALINA_OPTS -Djava.security.egd=file:/dev/./urandom
-Dcatalina.logs=$CATALINA_HOME/logs -Dpandora.location=$CATALINA_HOME/deploy/taobao-
hsf.sar  "/home/admin/app/provider.jar"   --server.context-path=/ --server.port=8080 --server.
tomcat.uri-encoding=ISO-8859-1 --server.tomcat.max-threads=400' > /home/admin/start.sh &&
chmod +x /home/admin/start.sh
WORKDIR $CATALINA_HOME
CMD ["/bin/bash", "/home/admin/start.sh"]
```

2. Customize settings in the Dockerfile.

   The following describes how to customize settings in the standard Dockerfile prepared previously.

i. Upgrade JDK.

Change the download and installation methods in the standard Dockerfile. The following uses JDK 8 as an example.

```
# Download and install JDK 8.
RUN wget http://edas-hz.oss-cn-hangzhou.aliyuncs.com/agent/prod/files/jdk-7u80-linux-x64
.rpm -O /tmp/jdk-7u80-linux-x64.rpm && \
   yum -y install /tmp/jdk-7u80-linux-x64.rpm && \
   rm -rf /tmp/jdk-7u80-linux-x64.rpm
```

ii. Upgrade EDAS Java Container.

When using a WAR package and Tomcat, upgrade the EDAS container to use new middleware features or fix known bugs. The upgrade procedure is as follows:

a. Locate the latest version (3.X.X) of the EDAS container.
b. Replace the version in the Dockerfile, such as 3.5.0.
c. Recreate and publish an application image.

```
# Prepare ENV
ENV EDAS_CONTAINER_VERSION V3.5.0
```

iii. Add the EDAS runtime environment to Tomcat startup parameters.

See (Mandatory) Environment variables during application runtime. EDAS provides the JVM environment variable `EDAS_CATALINA_OPTS`, which contains the minimum parameters required during runtime. Tomcat provides the custom JVM parameter configuration option `JAVA_OPTS` for setting xmx, xms, and other parameters.

```
# Set the JVM parameters of the EDAS application.
ENV CATALINA_OPTS ${EDAS_CATALINA_OPTS}
# Set the JVM parameters.
ENV JAVA_OPTS="\
   -Xmx3550m \
   -Xms3550m \
   -Xmn2g \
   -Xss128k"
```

1.5.3.3.3. Deploy an application (applicable to Container Service Kubernetes clusters)

You can deploy applications in a Container Service Kubernetes cluster.

Prerequisites

- Prepare an application image (a Container Service Kubernetes cluster) is complete, and the image has been pushed to the container image repository.

- The Container Service Kubernetes cluster has been imported to EDAS.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management**. On the **Applications** page, click **Create Application** in the upper-right corner.
3. On the **Application Information** page, set the parameters of the application. Then, click **Next Step: Application Configurations**.

Basic parameters

Name	Description
Namespace	Select a namespace from the drop-down list.
Deploy Cluster	Select a **Container Service Kubernetes cluster** from the drop-down list.
Application Type	The application type is determined by the cluster where the application is deployed. If you select a Container Service Kubernetes cluster, the application type is Kubernetes application. This parameter cannot be set manually.
Application Name	Enter a descriptive application name.
Application Description	Enter remarks for the application.

4. On the **Application Configuration** page, select **Image** for **Deployment Method**, select an image and a version, and click **Select**.
5. Set **Total Pods** and **Single Pod Resource Quota** (**CPU cores** and **Memory**).
6. Drag the slider on the right of **Advanced Setting** to the right to set advanced parameters. Then, click **Next Step: Application Access Settings**.
   i. (Optional)Set the startup command and parameters.

   > **Note** If you do not know the CMD and ENTRYPOINT content of the original Dockerfile image, do not modify the custom startup command and parameters. Otherwise, you cannot create applications due to an incorrect custom command.

   - **Startup Command**: Enter the content in `[ "" ]`. For example, set Startup Command to `/usr/sbin/sshd -D` for **CMD ["/usr/sbin/sshd" ," -D"]**.
   - **Startup Parameters**: Enter one parameter per line. For example, `args:[ "-c" ; "while sleep 2" ; "do echo date" ; "done" ]` contains four parameters. In this case, enter the parameters in four lines.

   ii. (Optional)Set environment variables.

   When creating the application, inject the environment variables you have entered to the container to be generated. This saves you from repeatedly adding common environment variables.

iii. (Optional)(Applicable to stateful applications) Set the application lifecycle management script.

Lifecycle management scripts:

- **PreStop script**: This is a container hook, which is triggered before a container is deleted. The corresponding hook handler must be completed before the container deletion request is sent to Docker daemon. Docker daemon sends an SGTERN semaphore to itself to delete the container, regardless of the hook handler execution result. For more information, see Container Lifecycle Hooks
- **Liveness script**: This is a container status probe, which monitors the health status of applications. If an application is unhealthy, the container is deleted and created again. For more information, see Pod Lifecycle
- **Readiness script**: This is a container status probe, which monitors whether applications have started successfully and are running properly. If an application is abnormal, the container status is updated. For more information, see Pod Lifecycle
- **Poststart script**: This is a container hook, which is triggered immediately after a container is created to notify the container of its creation. The hook does not pass any parameters to the corresponding hook handler. If the corresponding hook handler fails to run, the container is killed and the system determines whether to restart the container according to the restart policy of the container. For more information, see Container Lifecycle Hooks

7. On the **Application Access Settings** page, set SLB and click **Create**.

SLB corresponds to TCP/UDP settings. You can configure multiple port mappings for multi-port listening.

- Intranet SLB: This option ensures that all the nodes in a VPC can access the application.
- Public-facing SLB: After you enable this option, the system buys a public-facing SLB instance for the application to ensure that the application is accessible from the Internet.

SLB parameters:

- **SLB Port**: This parameter indicates the frontend port of the internal network or public-facing SLB instance, which is used to access the application. For example, NGINX uses port 80 by default.
- **Container Port**: This is the port that listens to processes. It is generally defined by the program. For example, the web service uses port 80 or 8080 by default, while the MySQL service uses port 3306 by default. The container port can be the same as the port used by the SLB instance.
- **Network Protocol**: You can select TCP or UDP.

Result

Return to the **Applications** page and check whether the created application is running properly.

1.5.3.3.4. Scaling (applicable to Container Service Kubernetes clusters)

Compared with common applications, Kubernetes applications feature much greater scalability due to the advantages of Kubernetes in container orchestration.

Procedure

1. Log on to the EDAS console and choose **Application Management** from the left-side navigation pane.
2. On the **Application Management** page, click the target Container Service Kubernetes application.
3. On the Application Details page, click **Application Scaling** in the upper-right corner.
4. In the **Application Scaling** dialog box, set **Total Application Pods** and click **OK**.

Result

A message that indicates successful operation appears after scaling is complete. Return to the Application Details page and click **Instance Information** to view the instance information and runtime status after scaling.

1.5.3.4. Log management

The EDAS console provides the runtime log function, allowing you to view the runtime logs of applications without having to log on to the ECS instance. When an exception occurs in an application, you can check logs to troubleshoot the problem.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Application Management**. On the **Applications** page, click the name of the target application.
3. On the Application Details page, choose **Log Management** > **Log Directories**from the left-side navigation pane.

   By default, the **Log Directories** page contains two log paths: the log path of the Tomcat container (such as */home/admin/taobao-tomcat-production-2.0.59.4/logs*) and the log path of EDAS Agent (such as */home/admin/edas-agent/logs*). Tomcat The path to container logs varies depending on the actual version.
4. Click the log folder or path to show all log files in the folder.

   > **Note** Only readable files but not folders are displayed.

5. Double-click a log file to view log details.
   - Select an instance from the **ECS Instance ID/Name/IP** drop-down list to view its real-time logs.
   - Click **Enable Real-time Additions** in the lower-right corner of the page to ensure that the latest additions to the file have been added (similar to the `tail -f`).
6. (Optional)Bookmark a log path.
   i. On the **Log Directories** page, select a path or folder and click **Bookmark Log Directory** in the upper-right corner of the page.

ii. In the Add Application Log Path dialog box, enter an **application log path** and click **Add**.

> **Notice**
>
> - The path must be in the /home/admin directory and contain "log" or "logs".
> - The file name must end with a slash (/) to indicate that it is a folder.

To cancel the bookmark status, click the name of a **folder** in the selected directory and click **Remove Directory from Bookmark** in the upper-right corner of the page. When a path is removed from favorites, it is no longer displayed on the logs page. This operation does not delete or change any files on the server.

1.5.3.5. Throttling and degradation (only applicable to HSF applications in ECS clusters)

Throttling and degradation are mainly used to solve slow system response or breakdown due to excessive burden on backend core services. These features are generally used in high-traffic scenarios, such as flash sales, shopping sprees, major promotions, and empty box scam protection.

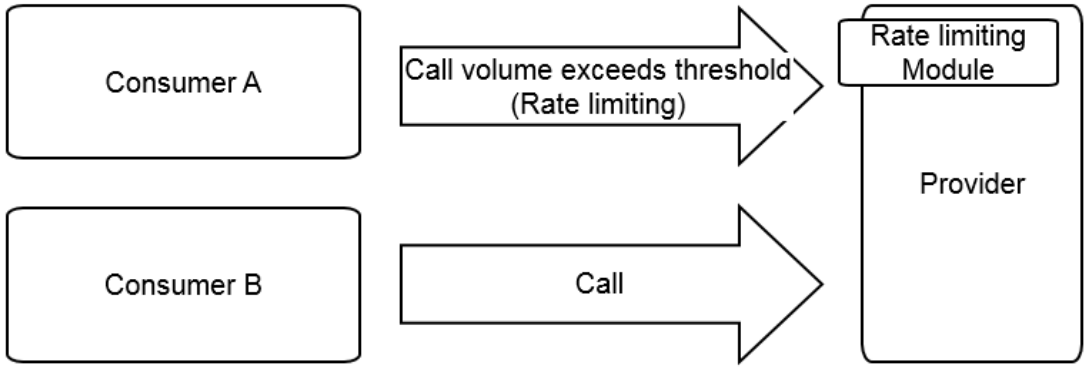
Throttling

This function controls the traffic threshold or adjusts the traffic ratio. It controls traffic when front-end websites are dealing with heavy access traffic to prevent service unavailability that results from damage to backend core systems. By adjusting the traffic threshold, the throttling function controls the maximum traffic volume of the system to make sure secure and stable system operation.

Principles

After the throttling code is configured for a provider and a throttling policy is configured in EDAS, the provider has the throttling function. When a consumer calls the provider, all access requests are calculated by the throttling module. If the call volume of the consumer exceeds the preset threshold in a specific period, the throttling policy is triggered.

Throttling



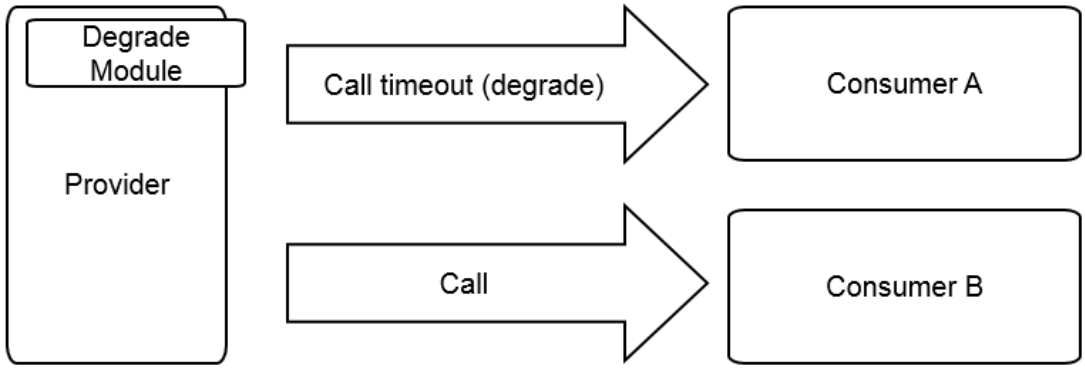
Degradation

In EDAS, degradation refers to the reduction of the call priority of downstream non-core providers that have timed out to make sure the availability of core consumers.

Principles

After degradation code is configured for a consumer and a degradation policy is configured in EDAS, the consumer has the degradation function. When the consumer calls a provider, if the response time of the provider exceeds the preset threshold, the degradation policy is triggered, as shown in the following figure.

Degradation



1.5.3.5.1. Throttling management

One application provides multiple services. EDAS allows you to configure throttling rules for the services, ensuring service stability and rejecting traffic that exceeds the service capabilities. EDAS allows you to configure throttling rules based on the QPS and threads to ensure the optimal operation stability of application systems during traffic peaks.

Context

- **HSF rate limiting:** When the traffic during a traffic spike exceeds the upper threshold defined by the throttling rules, the BlockException error occurs for some consumers. Based on the set threshold, the same number of services as the set threshold are successfully called within 1s.
- **HTTP rate limiting:** When a traffic spike occurs, some consumers are redirected to an error page. During actual access, the Taobao homepage appears. Based on the set threshold, some requests can be successfully sent to the services.

> **Notice** Throttling rules apply only to providers but cannot be configured for consumers. Before configuration, make sure that the application serves as the provider.

Procedure

1. Write the throttling rule code.
   i. Log on to the EDAS console.
   ii. In the left-side navigation pane, choose **Application Management**. On the **Applications** page, click a deployed provider application.
   iii. On the Application Details page, choose **Service Degradation** > **Rate Limiting Rules** from the left-side navigation pane.
   iv. On the **Rate Limiting Rules** page, click **Application Configuration Guide** in the upper-right corner. Write throttling code based on the example.
2. Add the throttling rule code to the application and then compile the code and Publish an

application.

3. Return to the EDAS console. In the left-side navigation pane, choose **Service Degradation** > **Rate Limiting Rules** . On the **Rate Limiting Rules** page, click **Add Rate Limiting Rules** in the upper-right corner.
4. On the **Add Rate Limiting Rules** page, set the throttling rule parameters and then click **OK**.

Throttling rule parameters

Name	Description
Rate Limiting Type	Select HSF Rate Limiting or HTTP Rate Limiting based on the access type of the application.
Interface	Select the interface to which the throttling rule applies from the listed interfaces as needed.
Method	Select a specific method or all methods to which the throttling rule applies after all methods of the selected interface are automatically loaded.
Application	Select the application to which the throttling rule applies from the application list as needed. The application list includes all applications that may access the current application, excluding the current application itself.
Rate Limiting Granularity	Select QPS or Thread. ○ QPS indicates limiting the number of requests per second. ○ Thread indicates limiting the number of threads. The QPS value is typically proportional to the number of threads. However, the QPS of a thread is generally greater than 1 because a thread keeps sending requests and the response time is dozens of milliseconds.
Rate Limiting Threshold	Throttling is triggered when the set threshold is exceeded.

What's next

On the **Rate Limiting Rules** page, locate the row that contains the target rule, and click **Edit**, **Stop**, **Enable**, or **Delete** on the right.

1.5.3.5.2. Degradation management

Each application calls multiple external services. Service degradation can be configured to pinpoint and block poor services. This feature ensures the stable operation of your application and prevents the functionality of your application from being compromised by dependency on poor services.

Context

EDAS allows you to configure degradation rules based on the response time, preventing your application from depending on poor services during traffic peaks. The consumer who triggers a degradation rule will not initiate an actual remote call within the specified time window and returns the DegradeException error. After the time window ends, the original remote service call is restored.

Note The degradation rules apply only to consumers and cannot be configured for providers. Before configuration, make sure that the application serves as a consumer.

Procedure

1. Write the degradation rule code.
   i. Log on to the EDAS console.
   ii. In the left-side navigation pane, choose **Application Management**. On the **Applications** page, select a deployed provider application.
   iii. On the Application Details page, choose **Service Degradation** > **Degradation Rules** from the left-side navigation pane. Click **Application Configuration Guide** in the upper-right corner. Write degradation rule code based on the example.
2. Add the degradation rule code to the application and then compile the code and Publish an application.
3. Return to the EDAS console. In the left-side navigation pane, choose **Service Degradation** > **Degradation Rules** . On the **Degradation Rules** page, click **Add Degradation Rules** in the upper-right corner.
4. On the **Add Degradation Rules** page, set degradation rule parameters and click **OK**.

Degradation rule parameters

Name	Description
Degradation Type	Select HSF Degradation and HTTP Degradation as needed.
Interface	All interfaces that the consumer is consuming are listed. Select the interface to be degraded as needed.
Method	All methods are automatically loaded based on the selected interface. You can select whether to degrade all methods or a specific method as needed.
RT Threshold	The threshold of the service response time that triggers degradation, in ms. If this threshold is exceeded, the selected interface or method is degraded.
Time Window	The rule execution duration after degradation is triggered.

What's next

On the **Degradation Rules** page, locate the row that contains the target rule, and click **Edit**, **Stop**, **Enable**, or **Delete** on the right.

1.5.3.6. Container version management (only applicable to HSF applications in ECS clusters)

EDAS allows you to view container versions and historical publishing details and perform upgrade and downgrade.

Context

An EDAS container consists of Ali-Tomcat, Pandora, and custom Pandora plug-ins. In addition to the support for existing Apache Tomcat core functions, EDAS provides a class isolation mechanism, QoS, and Tomcat-Monitor. Highly custom plug-ins are added to EDAS containers to implement complex and advanced functions, such as container monitoring, service monitoring, and tracing. Applications deployed by using EDAS must run in EDAS containers.

You must select a container version when creating an application in EDAS. EDAS containers are maintained and published by the EDAS development team. Choose **Application Management** > **Container Version** to view the container publishing history and the description of each publishing operation. Generally, a container of a later version is superior to a container of an earlier version in terms of stability and function variety.

EDAS container publishing does not affect deployed applications. Once a new container is available, you can immediately upgrade your container to the latest version.

Procedure

1. In the left-side navigation pane, choose **Application Management** to go to the Applications page.
2. Click the name of the target application to go to the Application Details page.
3. In the left-side navigation pane, choose **Container Version** to go to the Container Version page.
4. Locate the row that contains the target container version and click **Upgrade to This Version** or **Downgrade to This Version** on the right to upgrade or downgrade the container in one click.

1.5.4. Microservice management

Microservice management is an important function of EDAS. It allows you to view services in applications and inter-service traces.

Microservice management provides the following main functions:

- Trace query

  By setting filter criteria, you can accurately locate services with poor performance or exceptions.

- Trace details

  Based on the trace query results, you can view details of slow or abnormal services and reorganize their dependencies. This information allows you to identify frequent failures, performance bottlenecks, strong dependencies, and other problems. You can also evaluate service capacities based on trace call ratios and peak QPS.

- Service topology

  The service topology intuitively presents the call between services and relevant performance data.

1.5.4.1. Trace details

On the Trace Details page, you can query the details about a trace based on the TraceId in the selected region.

Prerequisites

The Trace Details page shows traces for which remote methods are called. It does not display local methods that are called.

Trace details are used to locate the elapsed time and exceptions in each step during a distributed call. Local calls are not the focus of traces. We recommend that you view service logs to check the elapsed time and exceptions for local calls. For example, the Trace Details page does not display the process where the local logic methodA() calls localMethodB() and localMethodC(). Therefore, sometimes the elapsed time on a parent node is greater than the total elapsed time on all subnodes.

You can search trace details on the Trace Details page. A more typical scenario is checking the slow or abnormal services in trace query results. The following uses an example to describe how to view details of a trace through trace query.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Microservice Management > Trace Details**.
3. On the Trace Details page, view trace details.
4. On the Trace Query Result page, locate the most time-consuming HSF method, database request, or other remote calls.
   - For database, Redis, MQ, or other simple calls, identify the cause of slow access to these nodes and check whether slow SQL or network congestion occurs.
   - For an HSF method, further analyze the reason why the method consumes so much time.
5. Confirm the elapsed time on a local method. Place the pointer over the timeline of the method. A pop-up window appears, showing the time it takes the consumer to send the request, the time it takes the provider to process the request, and the time it takes the consumer to receive the response.
   - If the time it takes the provider to process the request is long, analyze the service.
   - Otherwise, analyze the cause by using the method for analyzing call timeout.
6. Check whether the total elapsed time on subnodes is close to the elapsed time on this method.
   - If the time difference is small, most of the time is consumed by network calls. In this case, reduce network calls as much as possible to shorten the elapsed time on each method. The FOR statement cyclically calls the same method. The methods should be called in one batch to retrieve the response whenever possible.
   - If the time difference is large (for example, the elapsed time on the parent node is 607 ms while the total elapsed time on the subnodes does not reach 100 ms), the time is consumed on the service logic of the provider, rather than the request of the remote call.
7. Locate the time-consuming call. Inspect time-consuming calls by viewing the timelines of nodes to first locate the call initiated before the excessive time consumption. This is the local logic, for which further troubleshooting is required.

i. After locating the time-consuming logic, review the code or add a log method to the code to locate the specific error.

If the code does not consume so much time, perform the following step:

ii. Check whether GC occurred at that time. Therefore, the gc.log file is important.

8. Locate the timeout error. A timeout error occurred. Perform the following steps to evaluate the time. The time is divided into three parts:
   - Consumer sends request (0 ms): indicates the elapsed time from when the consumer sends a request to its receipt by the provider, including the time for serialization, network transfer, and deserialization. If this process takes a long time, check whether consumer GC is triggered. A lot of time is consumed if the serialization or deserialization object is large, the network is under a high transmission load, or provider GC occurs.
   - Provider processes request (10,077 ms): indicates the elapsed time from the receipt of the request by the provider to its response to the consumer. During this period, the provider processes the request, and the time consumed by other operations are not included.
   - Consumer receives the response (3,002 ms): indicates the elapsed time from when the provider sends the response to the receipt of the response by the consumer. With the 3-second timeout period, the provider directly returns a timeout error if the operation times out, but the provider continues processing the request. If this process consumes a lot of time, perform troubleshooting by using the same method as that for the consumer sending the request.

1.5.5. Batch operations

In the EDAS console, you can run machine commands to perform batch operations on the ECS instances with EDAS Agent installed.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **Batch Operations > Machine Commands**.
3. On the **Batch Operations** page, select a region and namespace.
4. In the **Machine Commands** section, click **By Clusters**, **By Applications**, or **By Instances** to determine the operation level.
5. > **Note** This topic describes operations at the cluster level. The procedures at the other two levels are similar.

   Click **Add** next to **Select Cluster**. In the **Select Cluster** dialog box, select a cluster (or search for the target cluster by performing a keyword search for its name) in the field on the left. Click > to add the cluster to the Selected field on the right. Then click **OK**.
6. Enter a command in the **Command** field.
7. (Optional) Select an operation range.
   - **Skip this step** if all the selected items are **ECS clusters**, **common applications**, or **common single-server instances**. The system uses the admin account to log on to instances and run commands.
   - If the selected items **include Swarm or Kubernetes clusters**, **Docker or Kubernetes**

applications, or **Docker single-server instances**, select **Execute in Host**, **Execute in Docker Container**, or **Execute in Host and Docker Container** (or select the three options). The system uses the admin account to log on to the host and run commands, and uses the root account to log on to the Docker container and run commands.

8. Click **Run**.

Result

- View operation results and details

  You are redirected to the **View Details** page after commands are executed. The **View Details** page includes the Overview, Basic Information, and Details tabs.

  - The Overview tab page shows the comprehensive analysis results of the command execution for batch operations, the number of successful and failed execution instances, and the time consumption.
  - The Basic Information tab page shows the batch operator, operating time, and executed commands.
  - The Details tab page shows the IP addresses and statuses (successful or failed) of the ECS and Docker instances for batch operations, and the command execution details.

    The Execution Details section shows the detailed command execution processes on instances. If command execution fails, an error message that indicates the cause is returned.

  In this case, select the instance and click **Retry**. You can rerun the command on the selected instance.

- View operation records

  On the **Batch Operations** page, view the batch operation record in the lower section. The record contains the operator name, creation time, end time, commands, and status (indicated by the execution results).

  - If the current account is the primary account, you can view all the batch commands that are executed by the primary account and all its RAM users.
  - If the current account is a RAM user, you can view only the batch commands that are executed by this RAM user.

  The entries in the operation record are sorted in descending order by time. You can sort the entries by operator name, creation time, or end time.

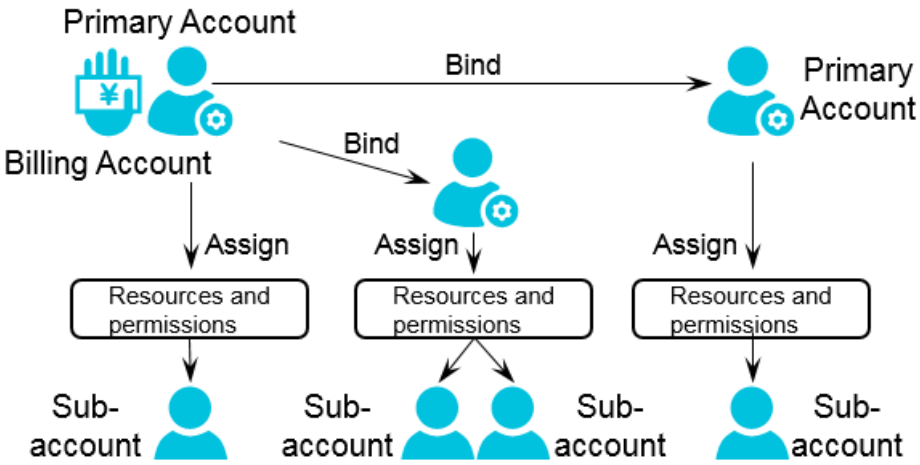
  Click **View** in the Details column to go to the Details page.

1.5.6. System management

1.5.6.1. Introduction to the EDAS account system

EDAS provides a comprehensive primary and RAM user management system. A primary account can assign permissions and resources to multiple RAM users as needed in accordance with the minimum permission principle. This lowers the risks to enterprise information security and reduces the work burden on the primary account.

EDAS account system



1.5.6.2. Manage RAM users

Resource Access Management (RAM) user management consists of RAM user overview and the Apsara Stack tenant account's operations on the RAM users.

1.5.6.2.1. RAM user overview

When using Enterprise Distributed Application Service (EDAS), you often need to complete different types of tasks as different roles. You can allocate different roles and resources to RAM users under an Apsara Stack tenant account to complete different types of jobs with different user identities. This permission model between the Apsara Stack tenant account and RAM user works in a similar way to the system and common user model in a Linux operating system, where system users can grant or revoke permissions to or from common users.

Relationship between an Apsara Stack tenant account and a RAM user:

- In the EDAS system, you can bind your Apsara Stack tenant account to a RAM user to avoid sharing your AccessKey pair with other users, and assign minimum permissions to the RAM user to complete different types of jobs with different user identities for effective enterprise management.
- When an Apsara Stack tenant account is bound to a RAM user, their binding relationship is valid only within EDAS, and both are independent accounts in other environments.
- An Apsara Stack tenant account can be an Apsara Stack tenant account with RAM users or be a RAM user under another Apsara Stack tenant account.

1.5.6.2.2. Use a primary account for RAM user operations

You can use a primary account for RAM user operations, such as Manage Role, Authorize Application, Authorize Resource Group, and Unbind. The procedures for these operations are similar. The following describes how to manage roles in detail and how to perform the other three operations briefly.

Context

A primary account can assign a role to a RAM user to grant the role-associated permissions to this sub-account.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **System Management** > **Sub-Account**.
3. Locate the row that contains the target RAM user, and click **Manage Roles** in the Actions column.
4. Select the target role and click **OK**.

   After the preceding settings, the role name appears in the Role field for the RAM user on the Sub-Accounts page.

> **Note**
>
> - **Authorize an application**
>
>   A primary account can assign an application to a RAM user to grant the application ownership to this RAM user.
>
>   Application authorization only grants the application ownership to the RAM user. To grant application operation permissions (to start or delete the application, for example) to the RAM user, assign a role to the RAM user. Therefore, application authorization is typically followed by role authorization.
> - **Authorize a resource group**
>
>   A primary account can assign a resource group to a RAM user, allowing the RAM user to use resources in the resource group. For the definition of a resource group, see Resource management.
> - **Unbind**
>
>   Through the unbinding operation, you can release the binding relationship between a RAM user and the primary account. The relationships with the assigned role, application, and resource group are also released. If you have not bought the EDAS service for the RAM user, you cannot log on to the EDAS console by using this RAM user after unbinding.

1.5.6.3. Manage roles

A primary account can define different operation permissions for its RAM users by creating different roles.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **System Management** > **Roles**.
3. Click **Create Role** in the upper-right corner of the page.
4. Enter a role name, add the permissions in the left-side field to the right, and click **OK**.

   After a role is added, you can perform actions on this role, such as **View Permissions**, **Manage Permissions**, and **Delete**.

1.5.6.4. View all permissions

You can list all permissions of the EDAS system in the console.

Procedure

1. Log on to the EDAS console.
2. In the left-side navigation pane, choose **System Management > All Permissions**.
3. Click a level to view the details of permissions at this level.

1.6. FAQ

This topic describes the common problems and solutions during product development and use.

1.6.1. Known issues and solutions

The features of this version have some known issues. If you encounter these issues, resolve them by following the methods.

- **Why can I not open the application monitoring data page?**

  Cause: This problem occurs if you do not purchase a certificate. In this case, the Application Real-time Monitoring Service (ARMS) page uses a self-signed or invalid certificate under HTTPS. Therefore, the access is directly blocked by the browser.

  Solution: Open the blocked ARMS page on a separate tab, and then open it again in Enterprise Distributed Application Service (EDAS).

- **Why is there no monitoring data after an application is upgraded?**

  Cause: This problem occurs when an application is upgraded to a later version. Since the EDAS monitoring solution has changed in version 3.9.0, you must manually restart the application to enable monitoring.

  Solution: You need to restart the application to view the basic monitoring data and service monitoring data again. If you cannot restart the application, remotely access the Elastic Compute Service (ECS) instance and manually perform the following steps:

  i. Reinstall EDAS Agent. For more information, see Use the command script to manually install EDAS Agent.

  ii. After switching to the admin user, run the `edas refresh-apm` command.

- **What can I do if the HTTP Code 503 error is reported? That is, after I bound a Server Load Balancer (SLB) instance with forwarding rules to an application group and then unbound the SLB instance from the application group, the application cannot be accessed through this SLB.**

  Cause: You entered the forwarding rules and bound the SLB instance to the application group, and then unbound the SLB instance in the EDAS console. In this case, EDAS did not delete the forwarding rules of the SLB instance. Since you unbound the SLB instance from the application group in the EDAS console, the servers in the virtual SLB group were also unbound. As a result, the SLB traffic forwarding failed and the HTTP 503 error was reported.

  Solution: Manually delete the forwarding rules in the SLB console.

- **Why does an application bound to an SLB instance fail to be accessed through the IP address of the SLB instance after traffic management is enabled?**

Cause: In this scenario, traffic management was enabled and the application was bound to an SLB instance (over HTTP). At this time, there is a limitation. When the SLB instance used HTTP to detect whether the backend ECS instances were alive, the message HEAD/HTTP/1.0 was sent, and Tengine responded with HTTP 400, which caused failure of SLB listener health check. As a result, the error code 502 (bad gateway) was reported when you accessed the application.

Solution: Disable traffic management on the Application Information page of the EDAS console for applications that do not require traffic management. This will uninstall Tengine, modify the application configurations, and restart the application. To retain this function, you can select Code 4xx returned by health check in Health Check of the SLB instance.

- **What can I do if a task is stuck and no longer scheduled due to a change order lag?**

  When a change order lags, restart both EDAS Enterprise Asset Management (EAM) containers and try again.

- **What can I do if the content of the mount script is cleared but the last content remains?**

  We recommend that you change the script to a null statement `echo ""` for bypass.

- **If you cancel HTTP rules and click Save when setting throttling, the instance reports the error "execution failed".**

  Generally, an error message is reported when you cancel HTTP rules for the first time. However, you can successfully cancel the HTTP rules by saving the configuration on the current page again.

- **After you create an application as a RAM user, the error "no permission" is reported when you delete the application. However, the application is deleted successfully.**

  This error message may confuse you, but you only need to cancel the error dialog box. The application can be deleted successfully, and this error will not appear later.

- **Why can I use a RAM user to view ECS instances in the cluster list, but cannot view the corresponding ECS instances on the ECS page or in the application scale-out list?**

  All ECS instances in the cluster are directly displayed in the cluster list. However, in the ECS instance list and during application scale-out, the system strictly checks whether an ECS instance is granted to a RAM user. If the ECS instance is not granted to the RAM user, the user cannot use the ECS instance.

1.6.2. Development FAQ

The development FAQ covers Ali-Tomcat, lightweight configuration center, HSF, HSF error codes, and other development problems.

1.6.2.1. Ali-Tomcat FAQ

This topic describes the problems frequently encountered during the Ali-Tomcat development process and their solutions.

- **Problem locating procedure**

  Ali-Tomcat may fail to start due to various errors. Check the catalina.out and localhost.log files to locate the error. If you use the Tomcat4E plug-in, you can view the detailed problem description in the Eclipse console.

- **How do I distinguish an EDAS error from a code error when an exception occurs?**

  Check whether the last part of the error stack contains the code itself. Example: Caused by: com.yourcompany.yourpack.

Problem	Error message	Solution
Service authentication failure	java.lang.Exception: Service authentication failed Note This problem only occurs in the EDAS production environment.	• The AccessKeyId and AccessKeySecret used for installing EDAS Agent are incorrect or they became incorrect due to web-based installation or other reasons. i. Run `cat /home/admin/.spas_key/default` . ii. Log on to the EDAS console. In the left-side navigation pane, choose **Resource Management > ECS** . On the **Instances** page, click **Install Agent**. iii. On the page that appears, check whether AccessKeyId and AccessKeySecret are set to the preceding values (case-sensitive). Web-based installation may cause case inconsistency. • The ECS instance has a delay of more than 30s. Adjust the time of the ECS instance. ◦ Run the `date` command to check whether the date is accurate.
Unknown host exception	Caused by: java.net.UnknownHostException: iZ25ax7xuf5Z	**iZ25ax7xuf5Z** indicates the current hostname. Check whether */etc/hosts* contains the IP address and name of the current host. If not, configure them, for example, `192.168.1.10 iZ25ax7xuf5Z` .
Port in use	Caused by: java.net.BindException: Address already in use: JVM_Bind	The port is in use. The troubleshooting method is the same as the method for troubleshooting port conflict in the lightweight configuration center.
com.ali.unit.rule.Router initialization failure	SEVERE: Context initialization failed java.lang.NoClassDefFoundError: Could not initialize class com.ali.unit.rule.Router	Address server connection failure jmenv.tbsite.net. Bind the domain. Add the following content to the hosts file to bind the domain name server address: `192.168.1.10 jmenv.tbsite.net` . Change 192.168.1.10 to the IP address of your lightweight configuration center. The path to the hosts file is as follows: • Windows: *C:\Windows\System32\drivers\etc\hosts* • Linux: */etc/hosts*

Problem	Error message	Solution
QoS port binding exception, resulting in a Pandora startup failure	Cannot start pandora qos due to qos port bind exception	The QoS port is in use. The troubleshooting method is the same as the method for troubleshooting port conflict in the lightweight configuration center.
Insufficient JVM memory	java.lang.OutOfMemoryError	Set the memory size. For more information about the solution, search **JVM memory settings** on the Internet.
A null pointer exception during WAR package deployment	deployWAR NullPointException	Check whether the WAR package is normal. Run jar xvf xxx.war to check whether the WAR package can be decompressed properly.
com.taobao.diamond.client.impl.DiamondEnvRepo initialization failure	Could not initialize class com.taobao.diamond.client.impl.DiamondEnvRepo	If DiamondServer data on the address server is empty, check whether the address server is correctly configured and is running stably. Access http://jmenv.tbsite.net:8080/diamond-server/diamond and check whether a response is properly returned.

1.6.2.2. Lightweight configuration center FAQ

This topic describes the common problems related to the lightweight configuration center and their solutions.

Problem	Error message	Solution
	Java version not supported, must be 1.6 or 1.6+	Check whether Java is properly installed. If Java is not installed, install Java 1.6 or a later version.

<table>
<tr><th>Problem</th><th>Error message</th><th>Solution</th></tr>
<tr><td rowspan="3">Startup fails when startup.bat and startup.sh are executed.</td><td>Unable to start embedded Tomcat servlet container</td><td rowspan="2">Check whether port 8080 is in use. If the port is used by another application, stop the application and run the startup script. Perform the following operations:
Windows:
1. Open the CMD window and run <code>netstat -aon|findstr “8080”</code>. Record the last column of numbers in the queried data, that is, the process ID (PID), such as 2720.
2. Run <code>tasklist|findstr “2720”</code>. The application that corresponds to the current PID, such as javaw.exe, appears.
3. Run <code>taskkill /PID 2720 /T /F</code>.
4. Start the lightweight configuration center again.
Linux:
1. Run <code>netstat -antp|grep 8080</code>. The PID of the process that uses port 8080 appears, for example "2720".
2. Run <code>kill -9 2720</code>.
3. Start the lightweight configuration center again.</td></tr>
<tr><td>Tomcat connector in failed state</td></tr>
<tr><td>Caused by: java.net.UnknownHostException: iZ25ax7xuf5Z</td><td>iZ25ax7xuf5Z indicates the current hostname. Check whether the IP address and name of the current host are configured in /etc/hosts. If not, configure them, for example, <code>192.168.1.10 iZ25ax7xuf5Z.</code></td></tr>
</table>

- **How do I specify the startup IP address for instances with multiple NICs?**

  In the startup script startup.bat or startup.sh, add the startup parameter -Daddress.server.ip={accessible IP address}.

- **How do I customize service publishing IP addresses?**

  In some cases, a service must be published on a vNIC or a non-physical IP address (for example, the EIP of an ECS instance) associated with the local host. If the virtual IP address is specified by using -Dhsf.server.ip, an error may occur when the service is started and the service cannot be published. This is because the virtual IP address cannot be found on the NIC of the local host during publishing.

  To solve this problem, EDAS provides the service IP address customization function for the provider that allows the provider to publish a service in the configuration center without specifying any IP address. After the service is successfully published, modify the IP address and then publish the service again. The consumer does not need to make any changes.

  Perform the following operations:

  i. After the service is published, find it in **Configuration List** and click **Update** on the right of the service.

You can also find the published service on the **Services** tab page.

ii. On the **Edit Configuration** page, modify the IP address in the Content field.

> **Notice** Do not modify the content after the IP address unless necessary. Otherwise, a service call error may occur.

iii. Click **OK** to save the settings.

iv. Restart the service. The service with the new IP address is registered again to enable the modification to take effect.

After modification, the consumer does not need to make any changes and can call the service in the normal way. You can query logs in *{user.home}\logs\configclient\config-client.log* to check the real IP address that is called by the consumer. Check the content next to the keyword **[Data-received]** in the logs to view the complete information about the called service.

1.6.2.3. HSF FAQ

- **Locate and solve HSF problems**

  HSF problems are logged in */home/admin/logs/hsf/hsf.log*. If any HSF problem occurs, check this file to locate the error. HSF errors have corresponding error codes. You can use these error codes to find the appropriate solution.

- **Set the timeout period for an HSF service**

  Use the HSF tags methodSpecials and clientTimeout to configure the timeout period.

  - **methodSpecials**: sets the timeout period (unit: ms) for a single method.
  - **clientTimeout**: sets the general timeout period (unit: ms) for all methods in the interface.

  The timeout period settings are sorted in descending order of priority as follows:

  Consumer methodSpecials > Consumer clientTimeout > Provider methodSpecials > Provider clientTimeout

  An example of the Consumer tag settings is as follows:

```
<hsf:consumer id="service" interface="com.taobao.edas.service.SimpleService"
version="1.1.0" group="test1" clientTimeout="3000"
target="10.1.6.57:12200? _TIMEOUT=1000" maxWaitTimeForCsAddress="5000">
<hsf:methodSpecials>
  <hsf:methodSpecial name="sum" timeout="2000" ></hsf:methodSpecial>
</hsf:methodSpecials>
</hsf:consumer>
```

- **HSF invalid call is removed**

  Error message:

  invalid call is removed because of connection closed

  Causes:

- Transient network disconnection: After the provider and consumer establish a connection, the consumer initiates a call request. An error is returned if the provider is still processing this request within the timeout period of the consumer and the consumer is disconnected due to network and other problems.
- Provider restart: After the consumer initiates a request, it waits for a response from the provider. If the consumer is restarted at this time, the socket is disconnected and the consumer receives an operating system connection closed callback. In this case, an error is returned.

Solution

If the service is idempotent, retry the service. Check the HSF provider network. This problem is often caused by network disconnection (transient disconnection).

- **Binding an IP address and port fails upon HSF startup**

  Problem: An error is returned when HSF is started. The error message is as follows:

  Java.net.BindException: Can't assign requested address

  Cause: The current IP address and port cannot be obtained.

  Solution: Set the following JVM parameter:

  -Dhsf.server.ip=IP address of your local network adpater -Dhsf.server.port=12200

- **Keep user logs from being overwritten**

  Problem: After EDAS is used, the log4j log cannot be generated.

  Cause: The log4j log is overwritten and thus cannot be generated.

  Solution: Set the JVM parameter Dlog4j.defaultInitOverride to false to generate user logs.

- **HSF Others**

  Error message: The following error is reported during startup:

  java.lang.IllegalArgumentException: HSFApiConsumerBean.ServiceMetadata.ifClazz is null.

  Solution: The class for the interface cannot be loaded. Check that the interface class is loaded to class path.

  Error message: failure to connect 10.10.1.1

  Solution:

  Check whether the HSF services are in the same VPC and the same region. If not, they cannot be connected.

  Check whether the HSF services are in the same security group. If not, enable port 12200.

  Run `telnet 10.10.1.1 12200` to check whether the port can be connected. If the port cannot be connected, check the firewall settings of the ECS instance with the IP address 10.10.1.1.

1.6.2.4. HSF error codes

Error code: HSF-0001

Error message:

HSFServiceAddressNotFoundException: This error message is returned when the address of the target service to be called is not found.

Description:

The target service to be called is xxxx, which is in the xxxx group.

Solution:

1. In the case of name mismatch, check whether the service name, version, and group (case-sensitive, without leading or trailing spaces) are set consistently for the provider and consumer.
2. Check whether an error is reported when the Tomcat container is started. Go to the Tomcat installation directory and check whether /logs/catalina.out localhost.log. 2016-07-01 (current date) contains any errors. If yes, fix the errors.
3. No service group is created. Log on to the EDAS console. In the left-side navigation pane, choose **Service Marketplace** > **Service Groups** to check whether a service group is created for the application. Example:

   ```
   <hsf:provider
         id="sampleServiceProvider" interface="com.alibaba.edas.SampleService" ref="target"
         version="for-test" group="your-namespace" ></hsf:provider>
   ```

   The corresponding **group named** *your-namespace* must exist in the service group list.
4. In the case of failed authentication, go to the ECS instance that corresponds to the **provider** and check whether */home/admin/configclient/logs/config.client.log* contains the spas-authentication-failed error. If this error exists:
   - No service group is created.
   - The AccessKeyId and AccessKeySecret used for installing EDAS Agent are incorrect or they became incorrect due to web-based installation or other reasons.
     a. Run `cat /home/admin/.spas_key/default` .
     b. Log on to the EDAS console. In the left-side navigation pane, choose **Resource Management** > **ECS** and click **Install Agent**.
     c. On the page that appears, check whether AccessKeyId and AccessKeySecret are set to the preceding values (case-sensitive). Web-based installation may cause case inconsistency.
     d. The IP address of the provider cannot be pinged. If multiple NICs exist, publish the IP address that is inaccessible from the consumer. Use -Dhsf.server.ip to specify the IP address of the provider.
5. The service call is initiated too early. A call is initiated before ConfigServer pushes the address, resulting in an error. Add maxWaitTimeForCsAddress to the consumer configuration file. For more information, see *Developer Guide*.
6. In the case of a data push error, contact a developer for troubleshooting.

Error code: HSF-0002

Error message:

Consumer error: HSFTimeOutException

Solution:

- Check whether the network of the ECS instance is healthy. Check whether the IP address of the provider can be pinged.
- If the processing time of the provider is greater than 3s, find the service execution timeout logs in hsf.log of the provider to locate the specific class and method:
  - A serialization error has occurred for the provider. Check the codes. The stream type, files, and oversized objects may cause a serialization error, and they cannot be transferred.
  - The code performance is inadequate. Optimize the code.
  - The logic of the provider is complex, and service processing requires more than 3s. Modify the timeout period. (See the *Developer Guide*.)
- Timeout occurs occasionally, and GC occurs for both the provider and consumer. Check the GC logs of the provider and consumer. GC that requires a long time may result in timeout. For more information about troubleshooting methods, search **Java GC optimization** on the Internet.
- The consumer is heavily loaded and fails to send the request, resulting in timeout. Add more instances for the consumer.

Error code: HSF-0003

Error message:

Consumer error: java.io.FileNotFoundException: /home/admin/logs/hsf.log (The specified path is not found.)

Solution: The default HSF log path cannot be found or is under access control. Load -DHSF.LOG.PATH=xxx during startup to modify the default path.

Error code: HSF-0005

Error message:

Startup error:

java.lang.IllegalArgumentException: This error message is returned when the object to be published as a service is not configured. The service name is com.taobao.hsf.jar.test.HelloWorldService:1.0.zhouli.

Solution:

The target attribute is missing from the bean of the provider. Check the configuration file.

The implementation class of the service specified by target does not exist. Check the configuration file.

Error code: HSF-0007

Error message:

java.lang.IllegalArgumentException: This error message is returned during startup when the serialization type is not supported.

Solution: The serializeType or preferSerializeType attribute is incorrectly configured for the bean of the provider. Check the configuration file. We recommend that you use Hessian or Hessian 2.0.

Error code: HSF-0008

Error message: java.lang.IllegalArgumentException, which is returned when the service type specified by ProviderBean is not [com.taobao.hsf.jar.test.HelloWorldServiceImpl].

Solution: serviceInterface configured for the bean of the provider is not an interface. serviceInterface must be set to an interface name. Check the configuration file.

Error code: HSF-0009

Error message: java.lang.IllegalArgumentException, which is returned when the real service object [com.taobao.hsf.jar.test.HelloWorldServiceImpl@10f0a3e8] does not implement the specified interface [com.taobao.hsf.jar.test.HelloWorldService].

Solution: No interface is implemented for the bean specified by target of the provider. Check that the corresponding interface is implemented in the interface class.

Error code: HSF-0014

Error message: java.lang.IllegalArgumentException, which is returned when the interface class specified by ProviderBean does not contain [com.taobao.hsf.jar.test.HelloWorldService1].

Solution: The serviceInterface attribute of the provider is incorrectly configured, and the specified interface does not exist.

Error code: HSF-0016

Error message:

Startup error: Failed to start the HSF provider.

Solution:

- Check whether port 12200 is already occupied. A server binding failure may cause a startup failure.
- If multiple NICs and an instance with a public network IP address exist, specify the local IP address by using -Dhsf.server.ip.

Error code: HSF-0017

Error message:

Startup error: java.lang.RuntimeException: [ThreadPool Manager] Thread pool allocated failed for service [com.taobao.hsf.jar.test.HelloWorldService:1.0.zhouli]: balance [600] require [800]

Solution: The allocated thread pool is insufficient. By default, the maximum thread pool size of HSF is 600. You can set the JVM parameter -Dhsf.server.max.poolsize=xxx to modify the default global maximum thread pool size.

Error code: HSF-0020

Error message:

WARN taobao.hsf - HSF service: com.taobao.hsf.jar.test.HelloWorldService:1.0.zhouli, which is returned when initialization is repeated.

Solution: In one HSF process, a service is uniquely identified by the service name and version. Services with the same name and version but of different groups cannot be published or subscribed to in a single process. Check the configuration file. For example, the service com.taobao.hsf.jar.test.HelloWorldService cannot be published or subscribed to in a single process if the following two configurations exist in the configuration file:

com.taobao.hsf.jar.test.HelloWorldService 1.0 groupA

com.taobao.hsf.jar.test.HelloWorldService 1.0 groupB

Error code: HSF-0021

Error message:

Startup error:

java.lang.IllegalArgumentException, which is returned when the interface class specified by ProviderBean does not contain [com.taobao.hsf.jar.test.HelloWorldService1].

java.lang.IllegalArgumentException: This error message is returned when the interface class specified by ConsumerBean does not contain [com.taobao.hsf.jar.test.HelloWorldService1].

Solution: The serviceInterface attribute of HSFSpringProviderBean is incorrectly configured, the specified interface does not exist (HSF-0014), or the interface specified by the interfaceName field in HSFSpringConsumerBean does not exist (HSF-0021).

Error code: HSF-0027

Error message: [HSF-Provider] HSF thread pool is full

Solution:

The processing speed of a service on the HSF provider is too slow, and requests from the client cannot be processed in time. As a result, the thread pool of the HSF provider for service execution reaches the maximum value. By default, HSF dumps the */home/admin/logs/hsf/HSF_JStack.log* file (default path). View the **HSFBizProcessor-xxx** thread stack information about the file and analyze the performance bottleneck.

The maximum number of threads of HSF is 600 by default. To increase the number, change the value of the -Dhsf.server.max.poolsize=xxx JVM parameter.

Error code: HSF-0030

Error message: [HSF-Provider] cannot find the method to be called.

Solution:

- The method is not provided by the provider. Log on to the EDAS console. In the left-side navigation pane, choose **Application Management** and **click the name of the application that corresponds to the service provider** to go to the Application Details page. In the left-side navigation pane, choose **Services** and check whether the corresponding service is successfully published.
- An earlier version and a later version coexist. The wrong version of a service is called. View the details of the service by using the preceding method.
- The interfaces of the provider and the consumer are inconsistent. For example, the provider provides java.lang.Double, whereas the consumer uses double to call the provider.
- Inconsistent interface classes are loaded for the provider and consumer. Check whether the MD5 values in the interface-contained JAR packages of the provider and consumer are

consistent.

Error code: HSF-0031

Error message: [HSF-Provider] takes xxx ms to execute the xxx method of the xxx HSF service. The time approximates the timeout period.

Solution: The provider prints this log when **the timeout period minus the actual time elapsed is less than 100 ms**. The timeout period is 3s by default.

- If the timeout period is short, for example, less than 100 ms, this log is printed in each call, and you can ignore it.
- If this log is still printed for a long timeout period, it indicates service execution is slow. Analyze the performance bottleneck in service execution.

Error code: HSF-0032

Error message: please check log on server side that unknown server error happens.

Solution: An uncaptured error occurs when the provider processes a request. Check the hsf.log file of the provider.

Error code: HSF-0033

Error message: Serialization error during serialize response.

Solution:

An error occurs when the provider returns data during serialization. Check the hsf.log file of the provider.

If the log file contains "must implement java.io.Serializable", implement a serializable interface on the DO.

Error code: HSF-0038

Error message: Multiple NICs are configured for the HSF provider, and the HSF provider is bound to an incorrect IP address.

Solution: Add -Dhsf.server.ip=xxx.xxx.xx.xx to the JVM startup parameters to specify the desired IP address.

Error code: HSF-0035

Error message: RPCProtocolTemplateComponent invalid address.

Solution: A TCP connection cannot be established between the current instance and the corresponding address. Check whether the corresponding remote address and port can be connected.

1.6.2.5. Other development problems

- Q: How do I develop an HSF application by using a framework other than Spring?

  A: We recommend that you use Spring to develop HSF applications. If you use another framework, you can develop applications by using LightAPI. For more information, see the *Developer Guide*.

- Q: Can I access the services in a production environment directly from a development

environment?

A: No. The production environment is isolated for security.

- Q: Does EDAS provide APIs? What functions do they have?

  A: EDAS provides APIs to implement resource query, application lifecycle management, and account management.

- Q: Does EDAS support other languages in addition to Java?

  A: HSF is developed in Java by default. HSF clients are also available in C++ and PHP, allowing you to access the backend HSF services provided by Java.

1.6.3. Usage FAQ

Common problems during development are related to accounts, resources, application lifecycle, and monitoring and alarms.

1.6.3.1. Account management

- Q: Can I create multiple RAM users?

  A: Yes.

- Q: Who can grant application operation permissions for RAM users?

  EDAS allows you to grant application operation permissions to RAM users only by using the primary account.

1.6.3.2. Resource management

- Q: Why doesn't the a prompt appear after EDAS Agent installation and EDAS Agent version is not displayed?

  A: Perform the following steps to troubleshoot the problem:

  i. Log on to the ECS instance and check */home/admin/edas-agent/logs/agent.log*. If UnAuthorizedException exists, check whether:

     - The AccessKeyId and AccessKeySecret used for installing EDAS Agent are incorrect or they became incorrect due to web-based installation or other reasons.

       a. Run `cat /home/admin/.spas_key/default` .

       b. Log on to the EDAS console. In the left-side navigation pane, choose **Resource Management > ECS** . On the **Instances** page, click **Install Agent**.

       c. On the page that appears, check whether AccessKeyId and AccessKeySecret are set to the preceding values (case-sensitive). Web-based installation may cause case inconsistency.

     - The region script used for installation is incorrect.

  ii. Check */home/admin/edas-agent/logs/std.log*. If "Java not found" or other error messages exist, run java —version to check whether the Java version is 1.7. If the version is Java 1.5, run `rpm -e corresponding installed RPM name` to remove it and reinstall EDAS Agent.

- Q: Why is the status Unknown or Abnormal after EDAS Agent is installed?

A: Check the std.log and agent.log files in the */home/admin/edas-agent/logs* directory of the ECS instance.

- std.log is the log of EDAS Agent installation.
- agent.log is the runtime log of EDAS Agent.

The possible causes are as follows:

- If "*Permission denied*" or "*Not such file*" is found in those logs, the possible cause is the lack of required file and directory permissions. In this case, check whether the admin account has permissions for all files in the /home/admin directory, and reinstall EDAS Agent.
- Check whether the ECS hostname is the same as that in the /etc/hosts file. If not, modify the name and restart EDAS Agent.

```
/home/admin/edas-agent/bin/shutdown.sh
/home/admin/edas-agent/bin/startup.sh
```

- Q: Which version of Java is EDAS using? Can I choose another version?

  A: EDAS provides Java 7 and Java 8. Java 7 is used by default. You can select a Java version when installing EDAS Agent. Run the following command to select a Java version:

  ```
  install.sh -ak -sk [-java <7(default)|8>]
  ```

- Q: What can happen if the heartbeat process of EDAS Agent stops?

  A: If no application is installed on that ECS instance, no services are affected. If an application is installed on that ECS instance, the *real-time status* of the ECS instance in the ECS instance list of the application (which appears in the lower part of the page after you select the application on the **Application Management** page and go to the **Basic Information** page) changes to **Agent Abnormal**. Any commands for the ECS instance, such as deploy, start, and stop, are ineffective.

  Log on to the ECS instance and run `sudo -u admin /home/admin/edas-agent/bin/startup.sh` to start EDAS Agent. Troubleshoot the EDAS Agent crash as follows:

  Check whether error messages are logged in */home/admin/edas-agent/logs/agent.log*.

  Check whether the system memory is sufficient. If the system memory is insufficient, the OOM Killer may be triggered. For more information, search for Linux OOM Killer on the Internet. If the OOM Killer is triggered, we recommend that you check the system memory usage and adjust memory allocation.

- Q: What should I do if the Ali-Tomcat container suddenly exits?

  A: Log on to the EDAS console to start the corresponding application. Troubleshoot the crash of Ali-Tomcat as follows:

  - Check whether error messages are logged in */home/admin/tomcat (installation directory)/logs/catalina.out*.
  - Check whether the system memory is sufficient. If the system memory is insufficient, the OOM Killer may be triggered. For more information, search for Linux OOM Killer on the Internet. If the OOM Killer is triggered, we recommend that you check the system memory usage and adjust the memory allocation policy.

- Q: Why doesn't EDAS Agent start after the system is restarted?

A: Currently, EDAS Agent of the CentOS 6.5 version supports automatic startup. Testing is not performed in other systems for the moment. If EDAS Agent is not started, run the following program:

```
sudo -u admin /home/admin/edas-agent/bin/startup.sh
/usr/alisys/dragoon/bin/DragoonAgent
```

1.6.3.3. Application lifecycle

- Q: Does EDAS Agent automatically restart after the target ECS instance is restarted?

  A: Yes. EDAS Agent automatically restarts after you restart the target ECS instance, but your Tomcat does not.

- Q: Why cannot I start EDAS Agent?

  A: Run the following command on the ECS instance where the EDAS console is deployed to check whether the instance is reachable:

  ```
  ping edas-internal.console.aliyun.com
  ```

  Then, check whether the security token file is correctly set:

  ```
  cat /home/admin/.spas_key/default
  ```

- Q: Can I deploy multiple applications on the same ECS instance in EDAS?

  A: EDAS allows you to deploy only one application on a single ECS instance.

- Q: Can I set the URL of an application deployment package to any address?

  A: Ensure that the application deployment package can be downloaded from this URL.

- Q: Why does an application operation (such as starting, stopping, or deploying an application) fail?

  A: Check whether EDAS Agent runs properly on the ECS instance with the failed operation. An application operation usually fails because EDAS Agent is not running properly.

- Q: Why don't the ECS instances under my account appear in the instance selection dialog box when I create an application?

  A: Check whether EDAS Agent is correctly installed on your ECS instances. Install EDAS Agent based on the procedure described in Resource management > ECS instance list > Install EDAS Agent.

  > **Notice** Be sure to select the correct region when installing EDAS Agent.

- Q: Why is the ECS instance status in the EDAS console "Unknown"?

  A: EDAS Agent reports heartbeat data periodically to the EDAS console. If EDAS Agent stops reporting heartbeat data, the ECS instance is set to the Unknown state after a certain time. This problem is typically caused by the stopping of EDAS Agent.

- Q: Why doesn't the service list appear while services can be called normally?

A: APIs have generics, but the generics do not have a specific type, which results in a failure to resolve the service list. In this case, modify the corresponding code.

- Q: What should I do when a service appears as Normal in the service list but I cannot call it?

  A:

  i. Check whether the group that corresponds to the service provider has been created. If not, authentication may fail.

  ii. Check */home/admin/logs/hsf/hsf.log* to determine the error code, and query HSF FAQ based on the error code.

- Q: Can I restore an application after I delete it?

  A: No. Application deletion is irrevocable. All application data is cleared after the application is deleted.

- Q: How do I perform batch or beta publishing?

  A:

  - If an application has multiple ECS instances, select batch publishing and set the number of batches to a value greater than 1 to publish the application in batches.
  - If an application has multiple ECS instances, set some of these instances to beta instances. You can separately publish the application to the beta instances. Only beta instances are updated during publishing. Other instances are not updated.

- Q: How do I share cluster sessions after deploying my application on multiple ECS instances?

  A: Currently, EDAS does not support distributed session management. You can use a distributed cache system (such as OCS and Redis) to manage distributed sessions.

- Q: How do I set the health check URL?

  A: When an application is published, the provided WAR file is automatically deployed in the Tomcat directory. Therefore, the WAR package name is added to the health check URL by default, and files in the WAR package must return the 200-400 HTTP codes. For example, assume a WAR package is named *order.war* and includes the file *index.jsp*. The health check URL can be set to http://127.0.0.1:8080/order/index.jsp.

- Q: Can I use SLB for load balancing after deploying my application on multiple ECS instances?

  A:

  i. HTTP-based web applications in EDAS use SLB for load balancing. You can configure SLB on the application configuration page of EDAS.

  ii. Load balancing does not need to be considered for applications that belong to RPC providers of EDAS. EDAS natively supports loading balancing for RPC providers.

2.API Gateway

2.1. What is API Gateway?

API Gateway provides a comprehensive suite of API hosting services that help you share capabilities, services, and data with partners in the form of APIs.

- API Gateway provides multiple security mechanisms to secure APIs and reduce the risks arising from open APIs. These mechanisms include protection against replay attacks, request encryption, identity authentication, permission management, and throttling.
- API Gateway provides API lifecycle management that allows you to define, publish, and unpublish APIs. This improves API management and iteration efficiency.

API Gateway allows enterprises to reuse and share their capabilities with each other so that they can focus on their core business.

API Gateway

2.2. Log on to the API Gateway console

This topic describes how to log on to the API Gateway console.

Prerequisites

- The domain name of the ASCM console is obtained from the deployment personnel before you log on to the ASCM console.
- A browser is available. We recommend that you use the Google Chrome browser.

Procedure

1. In the address bar, enter the URL used to log on to the ASCM console. Press the Enter key.
2. Enter your username and password.

   Obtain the username and password used to log on to the console from the operations administrator.

> **Note** When you log on to the ASCM console for the first time, you must change the password of your username. For security reasons, your password must meet the minimum complexity requirements. The password must be 8 to 20 characters in length and must contain at least two of the following character types:
>
> - Uppercase or lowercase letters.
> - Digits.
> - Special characters. Special characters include exclamation points (!), at signs (@), number signs (#), dollar signs ($), and percent signs (%).

3. Click **Login** to go to the ASCM console homepage.
4. In the top navigation bar, choose **Products** > **Application Services** > **API Gateway**.

2.3. Quick start

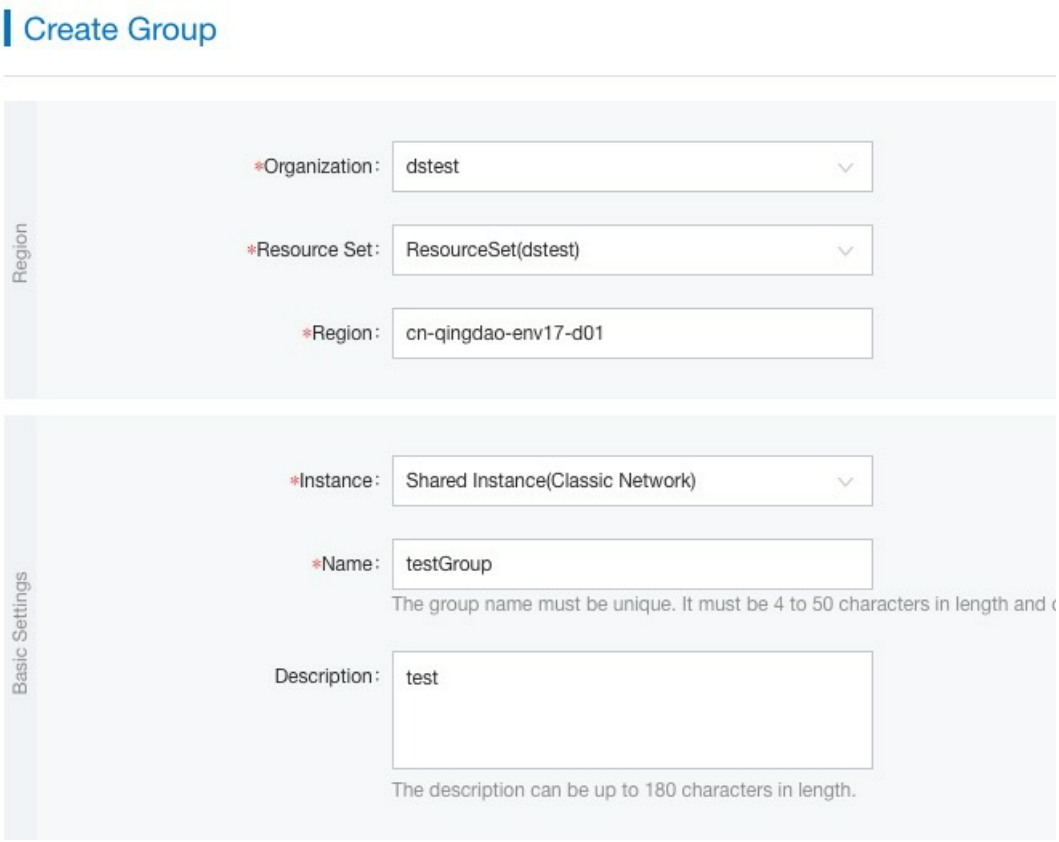
2.3.1. Create an API with HTTP as the backend service

This topic describes how to create and publish an API with HTTP as the backend service in API Gateway. This topic also describes how to call the API by using the AppKey and AppSecret pair of an app, with Security Certification set to Alibaba Cloud APP. You need to perform the following operations in sequence: **create an API group**, **create an API**, **create an app and an API authorization**, **debug the API**, and **call the API**.

Create an API group

APIs are managed in API groups. Before you create an API, you must create an API group.

1. **Create a group**: In the left-side navigation pane of the API Gateway console, choose **Publish APIs** > **API Groups**. On the Group List page, click **Create Group** in the upper-right corner. On the Create Group page, specify **Organization**, **Resource Set**, and **Region**, set Name to testAppkeyGroup, and click **Submit**. After you specify Organization, Instance is automatically set to Shared Instance(Classic Network).

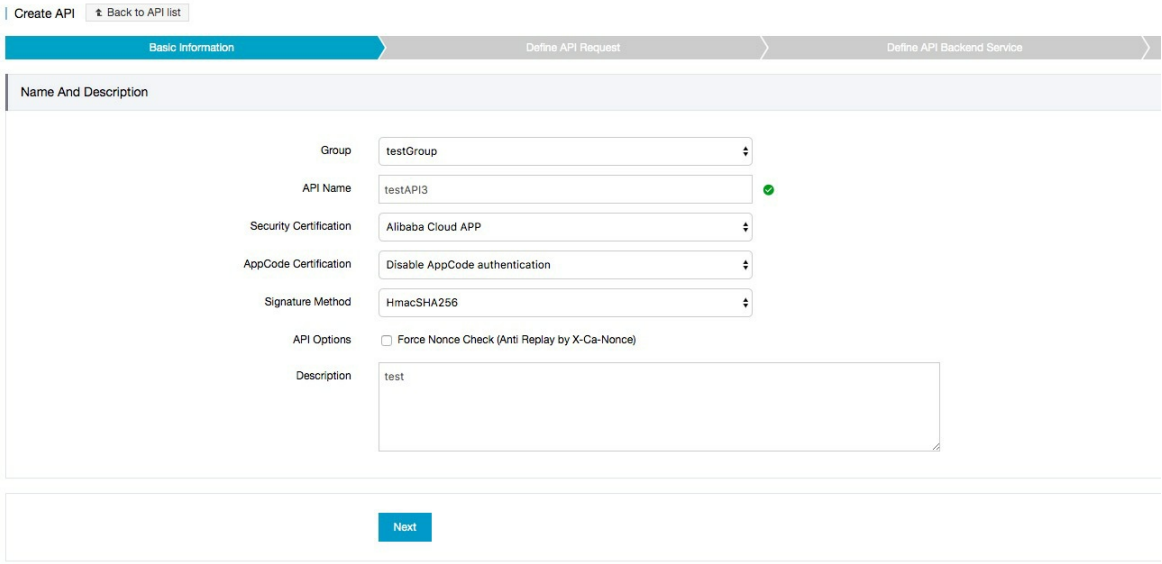


2. **View group information:** In the Submitted message, click Back to Console. On the **Group List** page, click **Refresh** in the upper-right corner. The group you created is displayed. You can click the group name to go to the Group Details page and perform operations such as binding a domain name and modifying basic information. A second-level domain is automatically created for the API group. It can be used in Apsara Stack to call all APIs under this group. In this example, the domain name is used for tests.

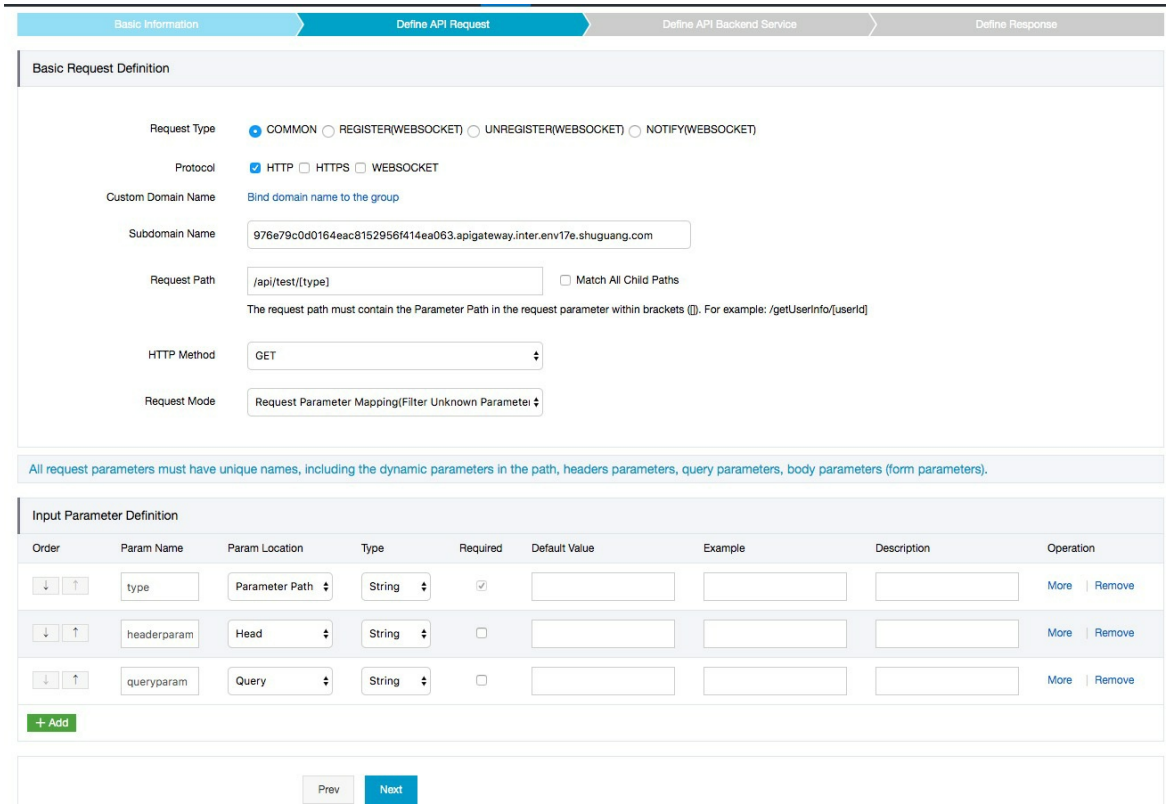
Create an API

In the left-side navigation pane of the API Gateway console, choose **Publish APIs > APIs**. On the API List page, click **Create API** in the upper-right corner. On the Create API page, perform the following steps:

1. **Specify basic information.** In this step, specify basic information, including Group, API Name, Security Certification, and Description. In this example, set Group to testAppkeyGroup, Security Certification to Alibaba Cloud APP, and AppCode Certification to Disable AppCode authentication, set other parameters as required, and click Next.

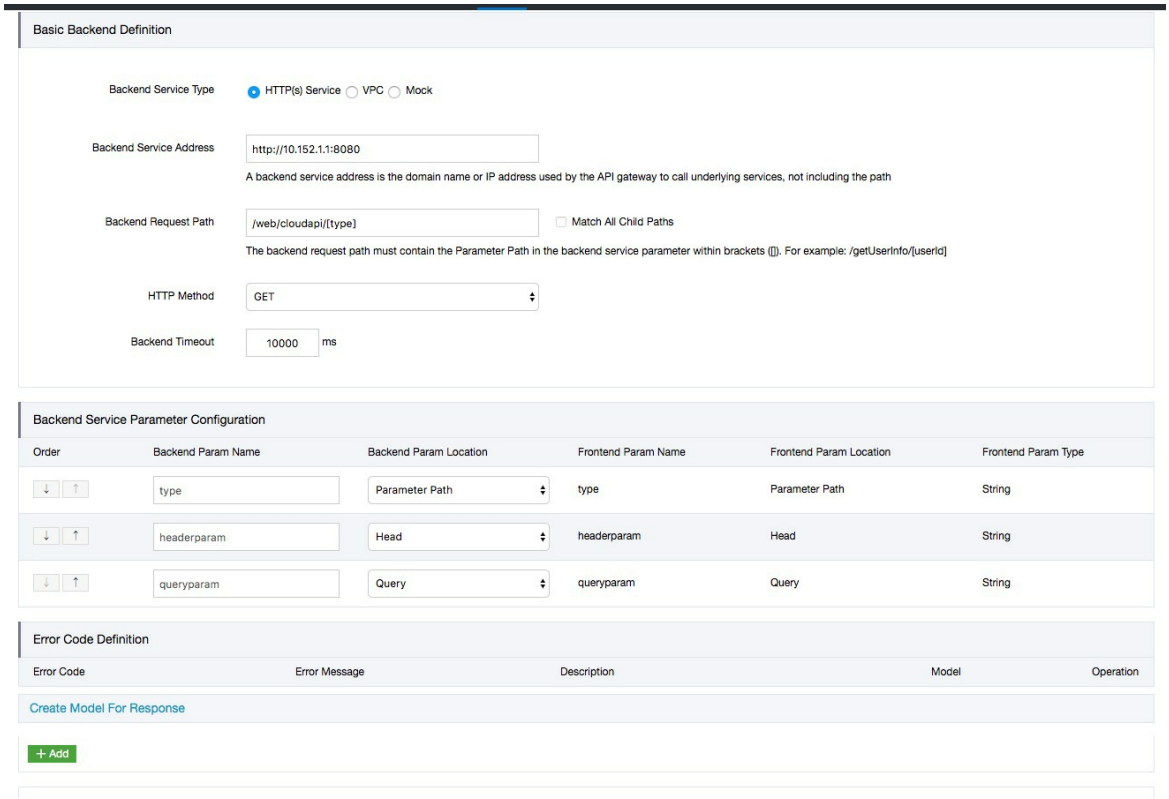


2. **Define an API request.** In this step, define how a client, such as a browser, a mobile app, or a business system, sends a request for the API. The parameters that need to be specified in this step include Request Type, Protocol, Request Path, HTTP Method, Request Mode, and those in the Input Parameter Definition section. Then click Next. In this example, enter /web/cloudapi in the Request Path field and configure a path parameter, a query parameter, and a header parameter in the Input Parameter Definition section.

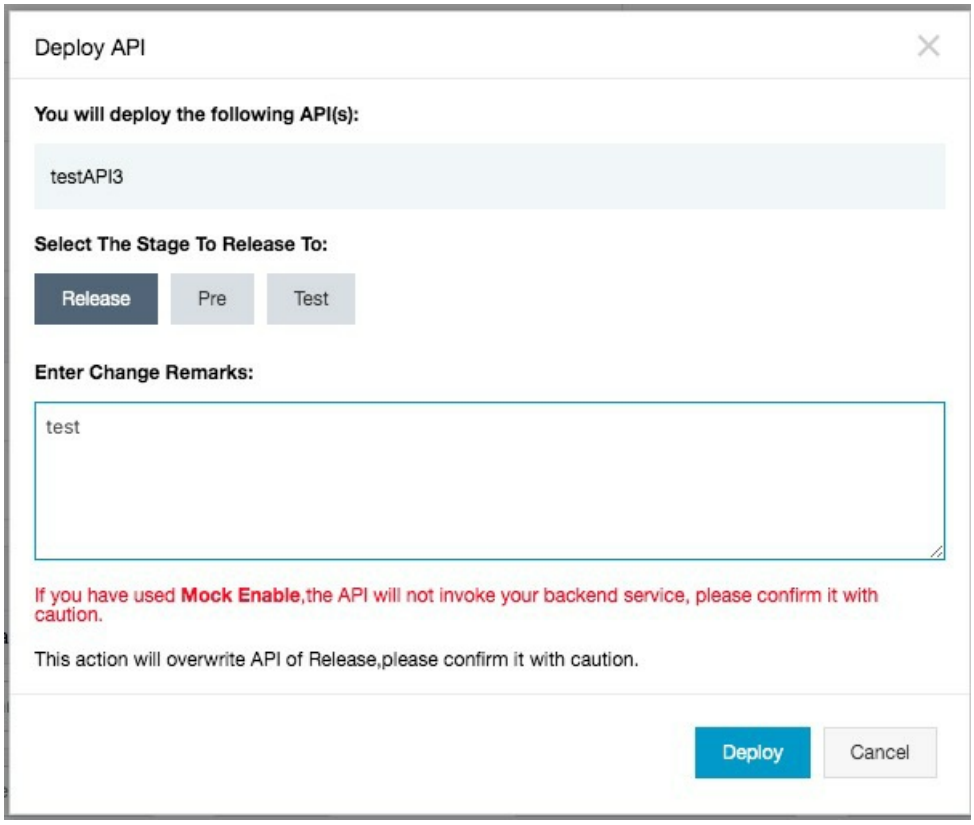


3. **Specify API backend service information.** In this step, configure a backend service type and a backend service address of the API and the mappings between request and response parameters. In this example, set Backend Service Type to HTTP(s) Service and Backend Service Address to an address that you can use to access API Gateway. For information about other backend service types, see API Gateway documentation. Set other parameters

such as Backend Request Path as prompted, and click Next.



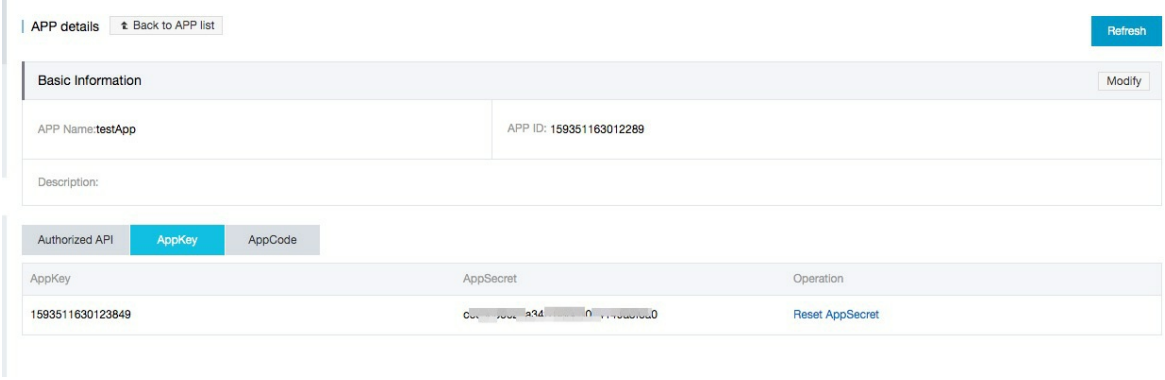
4. **Define return results.** In this step, configure response information to generate API documentation. The documentation helps API callers better understand APIs. You can set parameters such as ContentType of Response, Sample of Returned Results, and Sample of Returned Failure. The configurations in this step are not involved in this example. Click **Create**.
5. **Publish the API.** API Gateway provides three environments to which you can publish an API: Release, Pre, and Test. All configurations you perform on an API can take effect only after you publish the API to a required environment. In this example, click Deploy in the message that indicates successful API creation. Alternatively, find the created API on the API List page and click Deploy in the Operation column. In the Deploy API dialog box, set Select The Stage To Release To to Release, specify Enter Change Remarks, and click Deploy.



Create an app and an API authorization.

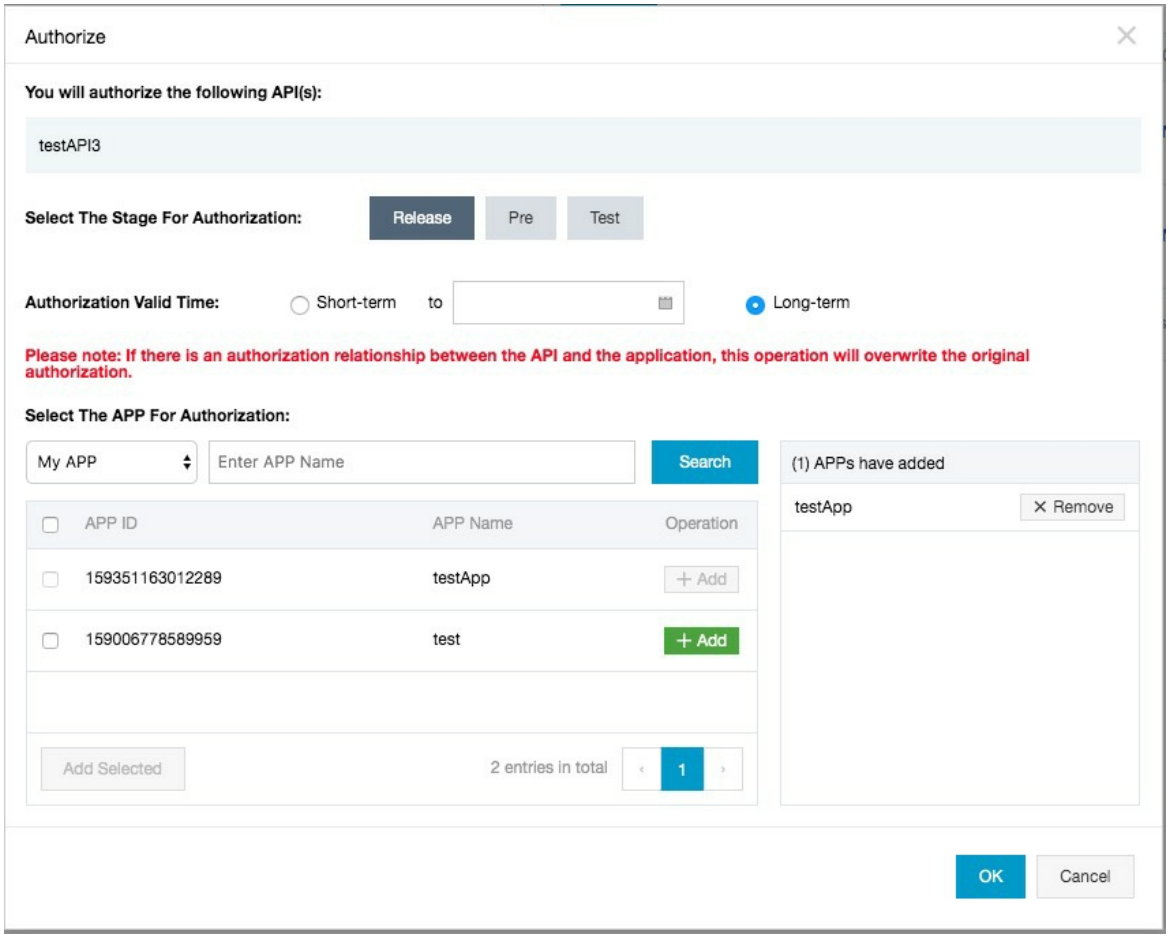
Apps are the identities that you use to call APIs. In Step 1 of the "Create an API" section, Security Certification is set to Alibaba Cloud APP. Therefore, after you publish the API, you must create an app and authorize the app to call the API.

1. **Create an app:** In the left-side navigation pane of the API Gateway console, click **Consume APIs** and then **APPs**. On the APP List page, click Create APP in the upper-right corner to create an app. Then click the name of the created app to go to the APP details page, as shown in the following figure. Two authentication modes are provided: an AppKey and AppSecret pair and AppCode. Each app has an AppKey and AppSecret pair. It works in a way similar to an account and password pair. When you call an API, you must pass in the AppKey as an input parameter. AppSecret is used to calculate the signature string. API Gateway authenticates the key pair to verify your identity.



2. **Authorize an API:** On the **API List** page, find the created API and click **Authorize** in the Operation column. In the Authorize dialog box, set Select The Stage For Authorization to the environment in which you published the API. In this example, set this parameter to Release.

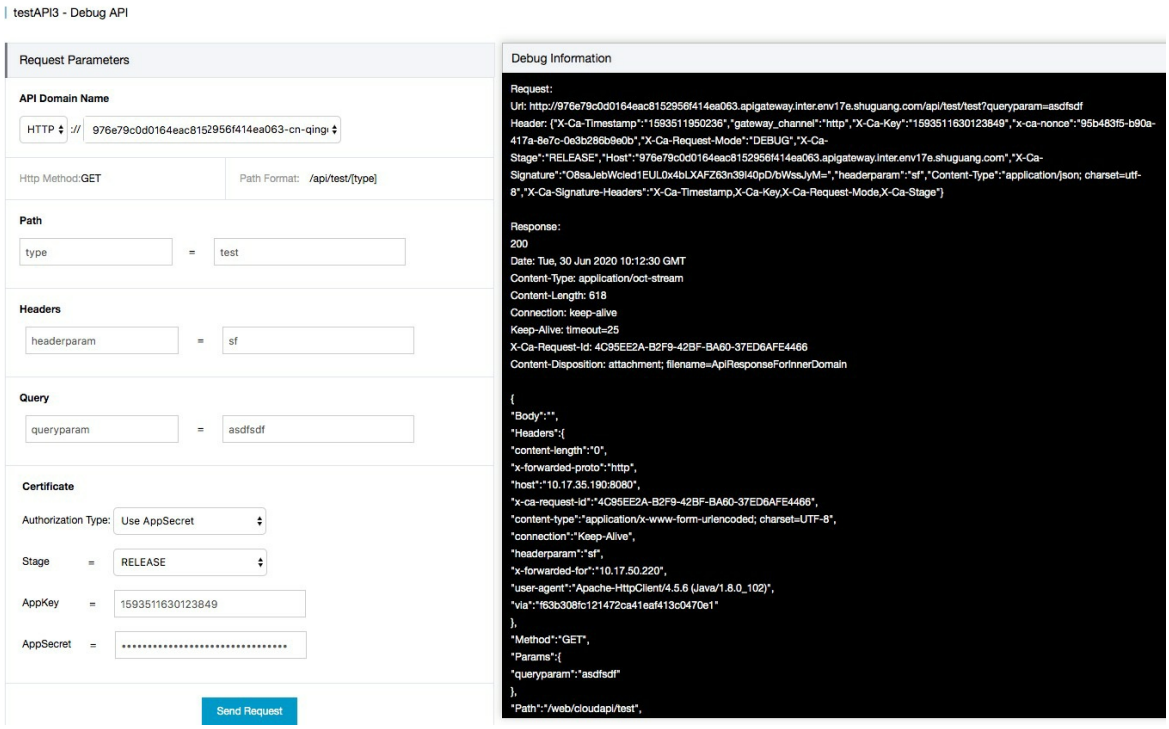
Click **Search** to search for the created app and click **+Add** in the Operation column to add this app to the pane on the right. Then click **OK**. A success message is displayed.



Debug an API

API Gateway supports online debugging. We recommend that you use this feature to check whether an API is correctly configured before you call this API at the client side.

1. In the left-side navigation pane of the API Gateway console, click **Consume APIs** and then **APPs**. On the APP List page, click the name of the app that has been authorized to call the created API. On the APP details page, click **Authorized API**, find the target API, and click **Debug API** in the Operation column. On the API debugging page, if you have defined input parameters for this API, you can enter different values for the input parameters to check whether the API is correctly configured.



Only the APIs that are published to a required environment and can be called by authorized apps are displayed after you click **Authorized API**.

Call an API

After you debug and publish an API to a Release environment, you can use SDKs for API Gateway to call the API in your business system.

1. In the left-side navigation pane of the API Gateway console, click **Consume APIs** and then **Authorized APIs SDK**. On the Authorized APIs SDK Auto-Generation page, find the target app and click the required programming language in the Authorized APIs SDK Auto-Generation column to download the relevant SDK package. The SDK package contains the API documentation and the SDK for the created API. For information about how to use the SDK, see the Readme file in the SDK package.Only the SDKs for APIs that are published to a Release environment are supported.

2.4. Call an API

2.4.1. Manage applications

2.4.1.1. Create an app

Apps are the identities that you use to call APIs. You can own multiple apps. Your apps can be authorized to call different APIs based on your business requirements. User accounts cannot be authorized to call APIs. In the API Gateway console, you can create, modify, or delete apps, view the details of apps, manage key pairs, and view the APIs that can be called by authorized apps.

Each app has an **AppKey** and **AppSecret** pair. It works in a way similar to an account and password pair. When you call an API, you must pass in the **AppKey** as an input parameter. **AppSecret** is used to calculate the signature string. API Gateway authenticates the key pair to verify your identity. An app must be authorized to call an API. Both authorization and authentication are intended for apps.

You can create apps on the **APP List** page in the API Gateway console.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, click **Consume APIs** and then **APPs**.
3. On the APP List page, click **Create APP** in the upper-right corner.The app name must be globally unique. It must be 4 to 26 characters in length and can contain only letters, digits, and underscores (_). It must start with a letter.

   After an app is created, the system automatically assigns an **AppKey** and **AppSecret** pair to the app. You must use the **AppSecret** to calculate a signature string. When you call an API, you must include the signature string in the request. API Gateway verifies your identity based on the signature string.

   On the **APP List** page, click the app name to go to the APP details page that displays the **AppKey** and **AppSecret** information. If the key pair is missing, you can reset it.

2.4.1.2. View app details

You can view details of created apps.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, click **Consume APIs** and then **APPs**.
3. On the APP List page, click the name of the app that you want to view.On the APP details page, you can view basic app information. You can also click **AppKey** or **Authorized API** to view key pair information and APIs that can be called by authorized apps.

2.4.1.3. Edit an app

You can edit a created app.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, click **Consume APIs** and then **APPs**.
3. On the APP List page, find the target app and click **Edit** in the Operation column.
4. In the Modify APP dialog box, modify app information and click **OK**.

2.4.1.4. Delete an app

You can delete a created app.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, click **Consume APIs** and then **APPs**.
3. On the APP List page, find the target app and click **Delete** in the Operation column.
4. In the Confirm Deletion message, click **OK**.

2.4.2. View created APIs

You can view created APIs in the API Gateway console.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs** > **APIs**.

2.4.3. Authorize an application

Authorization is the process of authorizing an application to call an API. Your applications must be authorized before they can call APIs.

You must provide your application IDs to the API provider for authorization. After authorization, you can view the APIs that your applications have been authorized to call in the API Gateway console.

The APIs that your applications have been authorized to call are displayed in the **Callable APIs** section on the application details page.

After the API provider authorizes your applications to call APIs, you do not need to and cannot authorize your applications.

2.4.4. Encrypt a signature

When you call an API, you must construct a signature string and add the calculated signature string to the request header. API Gateway uses symmetric encryption to verify the identity of the request sender.

- Add the calculated signature string to the request header.
- Organize the request parameters into a string-to-sign based on Request signatures. Then, use the algorithm provided in the SDK sample to calculate the signature. The result is the calculated signature string.
- Both HTTP and HTTPS requests must be signed.

For more information about how to construct a string-to-sign, see Request signatures. Replace the AppKey and AppSecret in the SDK sample with your own AppKey and AppSecret. Then, construct a string-to-sign based on Request signatures. After creating the string-to-sign, you can use it to initiate a request.

2.4.5. Request signatures

Endpoint

- Each API belongs to an API group, and each API group has a unique endpoint. An endpoint is an independent domain name that is bound to an API group by the API provider. API Gateway uses endpoints to locate API groups.
- An endpoint must be in the *www.[Independent domain name].com/[Path]?[HTTPMethod]* format.
- API Gateway locates a unique API group by endpoint, and locates a unique API in the group through the combination of Path and HTTPMethod.
- After you purchase an API, you can obtain the API documentation from the **Purchased APIs** list in the API Gateway console. If you have not purchased an API, you must obtain authorization from the API provider for your applications to call the API. After authorization, you can obtain the API documentation from the **Callable APIs** list on the application details page.

System-level header parameters

- (Required) X-Ca-Key: AppKey.
- (Required) X-Ca-Signature: the signature string.
- (Optional) X-Ca-Timestamp: the timestamp passed in by the API caller. This value is a UNIX timestamp representing the number of milliseconds that have elapsed since January 1, 1970 00:00:00 UTC. The timestamp is valid for 15 minutes by default.
- (Optional) X-Ca-Nonce: the UUID generated by the API caller. To prevent replay attacks, you must specify both the X-Ca-Nonce header and the X-Ca-Timestamp header.
- (Optional) Content-MD5: When the request body is not a form, you can calculate the MD5 value of the request body. Then, you can send the value to API Gateway for MD5 verification.
- (Optional) X-Ca-Stage: the stage of the API. Valid values: TEST, PRE, and RELEASE. Default value: RELEASE. If the API that you intend to call has not been published to the release environment, you must specify the value of this parameter. Otherwise, a URL error will be reported.

Signature validation

Construct the signature calculation strings

```
String stringToSign=
HTTPMethod + "\n" +
Accept + "\n" +            // We recommend that you specify the Accept header in the request. If the req
uest header is not set, some HTTP clients will use the default value */*, causing signature verification
to fail.
Content-MD5 + "\n"
Content-Type + "\n" +
Date + "\n" +
Headers +
Url
```

An HTTP method must be uppercase, such as POST.

If Accept, Content-MD5, Content-Type, and Date are empty, add a line break `\n` after each of them . If Headers is empty, `\n` is not required.

Content-MD5

Content-MD5 indicates the MD5 value of the request body. The value is calculated as follows:

```
String content-MD5 = Base64.encodeBase64(MD5(bodyStream.getbytes("UTF-8")));
```

bodyStream indicates a byte array.

Headers

Headers indicates the string constructed by the keys and values of the header parameters that are used for Headers signature calculation. We recommend that you use the parameters starting with X-Ca and custom header parameters for signature calculation.

> **Notice** The following parameters are not used for Headers signature calculation: X-Ca-Signature, X-Ca-Signature-Headers, Accept, Content-MD5, Content-Type, and Date.

Headers construction method:

Sort the header keys used for Headers signature calculation **in alphabetical order**. Construct the string based on the following rules: If the value of a header parameter is empty, use `HeaderKey + ":" + "\n"` for signature calculation. The key and colon (:) must be retained.

```
String headers =
HeaderKey1 + ":" + HeaderValue1 + "\n"\+
HeaderKey2 + ":" + HeaderValue2 + "\n"\+
...
HeaderKeyN + ":" + HeaderValueN + "\n"
```

The keys of the header parameters used for Headers signature calculation must be separated with commas (,), and placed in the request headers. The key is X-Ca-Signature-Headers.

Url

Url indicates the Form parameter in Path + Query + Body. For Query + Form, sort keys specified by Key in alphabetical order and construct the string based on the following rules: If Query or Form is empty, no question marks `?` are required for `Url = Path`. If Value of a parameter is empty, only Key is used for signature calculation and an equal sign (=) is not required.

```
String url =
Path +
"?" +
Key1 + "=" + Value1 +
"&" + Key2 + "=" + Value2 +
...
"&" + KeyN + "=" + ValueN
```

> **Notice** Note: The Query parameter or the Form parameter may have multiple values specified by Value. If both parameters have multiple values, only the first value of each parameter is used for signature calculation.

Signature calculation

```
Mac hmacSha256 = Mac.getInstance("HmacSHA256");
byte[] keyBytes = secret.getBytes("UTF-8");
hmacSha256.init(new SecretKeySpec(keyBytes, 0, keyBytes.length, "HmacSHA256"));
String sign = new String(Base64.encodeBase64(Sha256.doFinal(stringToSign.getBytes("UTF-8")),"UTF-8"));
```

`secret` indicates the AppSecret.

Signature passing

Add the calculated signature to the request header. The key is X-Ca-Signature.

Signature troubleshooting

If signature verification fails, API Gateway places the returned stringToSign value in the HTTP response header and sends the response to the client. The key is X-Ca-Error-Message. Compare the stringToSign value calculated by the client with the one returned by the server.

If the stringToSign values from the client and server are the same, check the AppSecret used for signature calculation.

HTTP headers do not support line breaks. Line breaks in stringToSign values are filtered out. Ignore the line breaks when you make a comparison.

Signature demo

For more information about the Java demo of signature calculation, see .

2.4.6. API call examples

You can edit an HTTP or HTTPS request to call an API. The API Gateway console provides API call examples of multiple programming languages for you to test the call.

Part 1: Request

Request URL

When you call an API over an internal network, the second-level domain of the API group to which this API belongs is used by default. To view a second-level domain, choose **Publish APIs** > **API Groups** in the left-side navigation pane of the API Gateway console. Click the name of the target group to go to the Group Details page. If this group is bound with an independent domain, you can use this independent domain to initiate an access request.

```
http://e710888d3ccb4638a723ff8d03837095-cn-qingdao.aliapi.com/demo/post
```

Request method

`POST`

Request body

```
FormParam1=FormParamValue1&FormParam2=FormParamValue2  //HTTP request body
```

Request header

```
Host: e710888d3ccb4638a723ff8d03837095-cn-qingdao.aliapi.com
Date: Mon, 22 Aug 2016 11:21:04 GMT
User-Agent: Apache-HttpClient/4.1.2 (java 1.6)
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
// The request body type. Set the request body type based on the actual request you want to make.
Accept: application/json
// The response body type. Some APIs can return data in the appropriate format based on the specifie
d response type. We recommend that you manually specify the request header. If the request header i
s not specified, some HTTP clients will use the default value */*, which causes a signature error.
X-Ca-Request-Mode: debug
// Specifies whether to enable the debug mode. This parameter is not case-sensitive. If it is not specif
ied, the debug mode is disabled. Enable this mode in the API debugging phase.
X-Ca-Version: 1
// The API version number. Currently, all APIs support only version 1. You can leave this request heade
r unspecified. The default version number is 1.
X-Ca-Signature-Headers: X-Ca-Request-Mode,X-Ca-Version,X-Ca-Stage,X-Ca-Key,X-Ca-Timestamp
// The custom request headers involved in signature calculation. The server reads the request header
s based on this configuration to sign the request. This configuration does not include the Content-Typ
e, Accept, Content-MD5, and Date request headers, which are already included in the basic signature s
tructure. For more information about the signature, see Request signatures.
X-Ca-Stage: RELEASE
// The stage of the API. Valid values: TEST, PRE, and RELEASE. This parameter is not case-sensitive. Th
e API provider can select the stage to which the API is published. The API can be called only after it is
published to the specified stage. Otherwise, the system will prompt that the API cannot be found or th
at the request URL is invalid.
X-Ca-Key: 60022326
// The AppKey of the request. You must obtain the AppKey in the API Gateway console. Apps can call
APIs only after they have been authorized.
X-Ca-Timestamp: 1471864864235
// The request timestamp. This value is a UNIX timestamp that represents the number of milliseconds t
hat have elapsed since January 1, 1970 00:00:00 UTC. The timestamp is valid for 15 minutes by default.
X-Ca-Nonce:b931bc77-645a-4299-b24b-f3669be577ac
// The unique ID of the request. AppKey, API, and Nonce must be unique within the last 15 minutes. To
prevent replay attacks, you must specify both the X-Ca-Nonce header and the X-Ca-Timestamp header
.
X-Ca-Signature: FJleSrCYPGCU7dMlLTG+UD3Bc5Elh3TV3CWHtSKh1Ys=
// The request signature.
CustomHeader: CustomHeaderValue
// The custom request headers. CustomHeaderValue is used as an example. You can configure multipl
e custom request headers in requests based on the definition of the API that is being called.
```

Part 2: Response

Status code

```
400  // The status code of the response. If the value is greater than or equal to 200 but less than 300, t
he call succeeded. If the value is greater than or equal to 400 but less than 500, a client-side error has
occurred. If the value is greater than 500, a server-side error has occurred.
```

Response header

```
X-Ca-Request-Id: 7AD052CB-EE8B-4DFD-BBAF-EFB340E0A5AF
// The unique ID of the request. When API Gateway receives a request, it generates a request ID and r
eturns the request ID to the client in the X-Ca-Request-Id header. We recommend that you record the r
equest ID in both the client and backend server for troubleshooting and tracking.
X-Ca-Error-Message: Invalid Url
// The error message returned by API Gateway. If a request fails, API Gateway returns the error mess
age to the client in the X-Ca-Error-Message header.
X-Ca-Debug-Info: {"ServiceLatency":0,"TotalLatency":2}
// The message returned only when the debug mode is enabled. The message is used only for referen
ce at the debugging stage.
```

Regardless of whether you call an API by using HTTP or HTTPS, the request must include the signature information. For information about how to calculate and deliver an encrypted signature, see Request signatures.

2.5. APIs

2.5.1. Manage groups

2.5.1.1. Create an API group

You can create an API group in the API Gateway console.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs** > **API Groups**.
3. On the Group List page, click **Create Group** in the upper-right corner.
4. On the Create Group page, specify **Organization**, **Resource Set**, and **Region** in the Region section. Then specify **Name** and **Description** in the Basic Settings section and click **Submit**.The group name must be unique. It must be 4 to 50 characters in length and can contain only letters, digits, and underscores (_). It must start with a letter.

2.5.1.2. Manage domain names

In Apsara Stack, you can use the second-level domain of a group to directly call an API that belongs to this group. You can also bind your domain name to the group so that you can use your domain name to call APIs that belong to the group.

Context

If you want to use your domain name to directly call APIs that belong to a group, you must bind the domain name to the group and add a DNS record to your domain name. The domain name must be resolved to the second-level domain of the group or the IP address that corresponds to the second-level domain.

Bind an independent domain

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the Group List page, find the target group and click **Bind Domain** in the **Operation** column.
4. In the Bind Domain Name dialog box, specify **Domain Name** and click **OK**.

Delete an independent domain

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the **Group List** page, find the target group and click its name to go to the **Group Details** page.
4. In the **Custom Domain Name** section, click **Delete Domain** in the Operation column.
5. In the Confirm Deletion message, click **OK**.

2.5.1.3. Manage certificates

To use HTTPS on an independent domain, you must upload an SSL certificate.

Context

To perform HTTPS API calls, you must use a domain name that supports HTTPS and set Protocol to HTTPS in the Basic Request Definition section when you define an API request.

Upload a certificate

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the **Group List** page, click the name of the target group to go to the **Group Details** page.
4. In the **Custom Domain Name** section, click **Create Certificate** in the **SSL Certificate** column.
5. In the **Create Certificate** dialog box, specify **Certificate Name**, **Certificate Content**, and **Private Key**, and click **OK**.

Delete a certificate

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the **Group List** page, click the name of the target group to go to the **Group Details** page.

4. In the **Custom Domain Name** section, click **Delete Certificate** in the **Operation** column.
5. In the Confirm Deletion message, click **OK**.

2.5.1.4. Delete an API group

You can delete a created API group.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs** > **API Groups**.
3. On the Group List page, find the target group and click **Delete** in the Operation column.
4. In the Delete Group message, click **Delete**.

> **Note** Before you delete a group, you must delete APIs that belong to this group.

2.5.1.5. Manage environments

To understand environment management, you must be familiar with two concepts: environment and environment variable.

- An environmentis an API group configuration. You can configure several environments for a group. APIs that have not been published are considered defined APIs. An API can provide external services only after it is published to an environment.
- Environment variablesare environment-specific variables that you can create and manage. For example, you can create an environment variable named `Path` in the Release environment. The value of this variable is `/stage/release`.

When you define an API request, you can set `Backend Service Address` to http(s):// `#Path#`. #Path# indicates a variable named `Path`.

When you publish the API to the Release environment, the value of `#Path#` is `/stage/release`.

When you publish the API to another environment that does not have the environment variable `#Path#`, the variable value cannot be obtained and the API cannot be called.

Environment variables allow backend services to run in different runtime environments. You can access various backend services by configuring the same API definition but different backend service endpoints and paths across different environments. When you use environment variables, consider the following limits:

- Variable names are case-sensitive.
- If you configure a variable in the API definition, you must configure the name and value of the variable for the environment to which the API is published. Otherwise, no value is assigned to the variable and the API cannot be called.

Create an environment variable

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs** > **API Groups**.

3. On the Group List page, find the target group and click **View Stages** in the Operation column.
4. On the Stage Management page, click **Add Variable** in the upper-right corner. In the Add Variable dialog box, specify *Name* and *Value* and click **Add**.

> **Notice** The variable names for the Release, Pre, and Test environments must be the same. However, the variable values for the three environments can be different. When an API is published to a specified environment, the variable value will be automatically replaced.

Delete an environment variable

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the Group List page, find the target group and click **View Stages** in the Operation column.
4. On the Stage Management page, select the runtime environment, find the target variable, and click **Delete** in the Operation column.
5. In the Confirm Deletion message, click **OK**.

2.5.2. Create an API

2.5.2.1. Overview

Creating an API is the process of defining the API in the API Gateway console. When creating an API, you must define the basic information, back-end service information, API request information, and response information of the API.

- API Gateway enables you to configure verification rules for input parameters. API Gateway can be configured to pre-verify and forward API requests that contain valid parameters.
- API Gateway enables you to configure mappings between front-end and back-end parameters. API Gateway can map a front-end parameter at one location to a back-end parameter at a different location. For example, you can configure API Gateway to map a **Query** parameter in an API request to a **Header** parameter in a back-end service request. In this way, you can encapsulate your back-end services into standard API operations.
- API Gateway enables you to configure constant and system parameters. These parameters are not visible to your users. API Gateway can add these parameters to requests based on your business requirements before sending the requests to your back-end services. If you want API Gateway to attach the keyword **apigateway** to each request that API Gateway forwards to your back-end services, you can configure **aligateway** as a constant parameter and specify where it is received.

2.5.2.2. Create an API

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, click **Create API** in the upper-right corner.

4. Specify basic information of the API and click **Next**.

Parameter	Description
Group	The basic management unit of APIs. Before you create an API, you must create an API group. When you select a group, a region is selected for the API.
API Name	The name of the API to be created.
Security Certification	The authentication mode of API requests. Valid values: *Alibaba Cloud APP* and *No Certification*. ○ *Alibaba Cloud APP*: This mode requires the requester to pass the app authentication to call an API. ○ *No Certification*: This mode allows all users who know the request definition of an API to initiate a request. API Gateway directly forwards the request to your backend service without the need to verify the identity of a requester.
Signature Method	The algorithm that is used to sign API requests. Valid values: ○ HmacSHA256 ○ HmacSHA1 and HmacSHA256: If you set this parameter to this value, both the algorithms are supported.
Description	The description of the API.

5. Define an API request. In this step, define how users call your API, with the following parameters specified: Request Type, Protocol, Request Path, HTTP Method, and Request Mode.

Parameter	Description
Request Type	The request type. Only the COMMON request type is supported. Valid values: COMMON, REGISTER(WEBSOCKET), UNREGISTER(WEBSOCKET), and NOTIFY(WEBSOCKET). ○ *COMMON*: indicates common HTTP or HTTPS requests. ○ *REGISTER(WEBSOCKET)*: indicates the bidirectional control signaling to register devices. It is sent from the client to the server. ○ *UNREGISTER(WEBSOCKET)*: indicates the bidirectional control signaling to deregister devices. It is sent from the client to the server. After devices are deregistered, server-to-client notifications are no longer received. ○ *NOTIFY(WEBSOCKET)*: After receiving the registration signaling sent from the client, the backend service records the device ID and sends a server-to-client notification to API Gateway. Then, API Gateway sends the notification to the device. If the device is online, API Gateway sends the server-to-client notification to the device.
Protocol	The supported protocol. Valid values: *HTTP*, *HTTPS*, and *WEBSOCKET*.

Parameter	Description
Request Path	The API request path. The request path can be different from the actual backend service path. You must specify a valid and semantically accurate path as the request path. You can configure dynamic parameters in the request path. This requires that you specify path parameters in the request. In addition, the path parameters can be mapped to query and header parameters that are received by the backend service.
HTTP Method	The HTTP request method. Valid values: *PUT*, *GET*, *POST*, *PATCH*, *DELETE*, and *HEAD*.
Request Mode	The request mode. Valid values: *Request Parameter Mapping(Filter Unknown Parameters)*, Request Parameter Mapping(Passthrough Unknown Parameters), and *Request Parameter Passthrough*. ○ Request Parameter Mapping(Filter Unknown Parameters): You must configure request and response data mappings for query, path, and body form parameters. API Gateway transparently passes only the configured parameters to the backend service. Other parameters are filtered out. ○ Request Parameter Mapping(Passthrough Unknown Parameters): API Gateway maps and verifies only configured request parameters and transparently passes unknown parameters in a request to the backend service. ○ Request Parameter Passthrough: You do not need to configure query and body form parameters, but must configure path parameters in the Input Parameter Definition section. All parameters sent from the client are transparently passed by API Gateway to the backend service.

6. Define request parameters.In this step, define the request parameters of your API. You can specify different request parameters for different parameter paths. You can select Head, Query, Body, or Parameter Path from the Param Location drop-down list. When you configure a dynamic path parameter, you must provide a description of this dynamic parameter in the Input Parameter Definition section. The following data types are supported: String, Int, and Boolean.
   - Note that the names of all parameters must be unique.
   - You can use the shortcut keys in the Order column to adjust the parameter order.
   - To delete a parameter that is no longer required, you can click **Remove** in the Operation column that corresponds to the parameter.
7. Configure parameter verification rules.To configure verification rules of a parameter, you can click **More** in the Operation column that corresponds to the parameter. For example, you can specify Max Length and Enumeration. API Gateway pre-verifies requests based on the verification rules. Requests with invalid parameters are not sent to your backend service. This significantly reduces the workload on your backend service.
8. Configure the backend service and click **Next**.In this step, define mappings between request and response parameters, and specify the API configurations of your backend service. Backend service configurations include Backend Service Address, Backend Request Path, Backend Timeout, and configurations in the Backend Service Parameter Configuration,

Constant Parameter, and System Parameter sections. After receiving a request, API Gateway converts the format of the request into the format that is required by your backend service based on the backend service configuration. Then, API Gateway forwards the request to your backend service.

> **Note** You can configure the following parameters: dynamic path parameters, header parameters, query parameters, body parameters (non-binary), constant parameters, and system parameters. Each parameter name must be globally unique. For example, you cannot specify a header parameter and a query parameter that have the same name.

i. Specify related parameters in the Basic Backend Definition section.

Parameter	Description
Backend Service Type	■ HTTP(s) Service: This option is selected by default. It indicates that API Gateway accesses the backend service over HTTP or HTTPS. If API Gateway can directly communicate with the backend service, select this option. ■ VPC: If the backend service is deployed in a VPC, select this option. ■ Mock: If you want to simulate expected return results, select this option.
VPC ID	The ID of the VPC where your backend service is deployed. This parameter is required when Backend Service Type is set to VPC.
Backend Service Address	The host of the backend service. ■ If Backend Service Type is HTTP(s) Service, set this parameter to a domain name or a value in the `http(s)://host:port` format. ■ If Backend Service Type is VPC, set this parameter to a value in the `http://ip:port` format.
Backend Request Path	The actual request path of your API on your backend server. If you want to receive dynamic parameters in the backend path, you must specify the locations and names of the corresponding request parameters to declare parameter mappings.
HTTP Method	The HTTP request method. Valid values: *PUT*, *GET*, *POST*, *PATCH*, *DELETE*, and *HEAD*.
Backend Timeout	The response time for API Gateway to access the backend service after API Gateway receives an API request. The response time starts from the time when API Gateway sends an API request to the backend service and ends at the time when API Gateway receives a response returned by the backend service. The response time cannot exceed 30s. If API Gateway does not receive a response from the backend service within 30s, API Gateway stops accessing the backend service and returns an error message.

ii. Configure parameters in the Backend Service Parameter Configuration section.

API Gateway can set up mappings between request and response parameters, including name mappings and parameter location mappings. API Gateway can map a path, header, query, or body request parameter to a response parameter at a different location. This way, you can package your backend service into a standardized and professional API form. This part declares the mappings between request and response parameters.

> Note The request and response parameters must be globally unique.

iii. Configure constant parameters in the Constant Parameter section.

If you want API Gateway to attach the apigateway tag to each request that API Gateway forwards to your backend service, you can configure this tag as a constant parameter. Constant parameters are not visible to your users. After API Gateway receives requests, it automatically adds constant parameters to the specified locations and then forwards the requests to your backend service.

iv. Configure system parameters in the System Parameter section.

By default, API Gateway does not send its system parameters to your backend service. If you require the system parameters, you can configure the related locations and names. The following table lists the system parameters.

Parameter	Description
CaClientIp	The IP address of the client that sends a request.
CaDomain	The domain name from which a request is sent.
CaRequestHandleTime	The time when a request is sent. It must be in GMT.
CaAppId	The ID of the app that sends a request.
CaRequestId	The unique ID of the request.
CaApiName	The name of the API.
CaHttpSchema	The protocol that is used to call an API. The protocol can be HTTP or HTTPS.
CaProxy	The proxy (AliCloudApiGateway).

9. Define responses and click **Create**.In this step, specify ContentType of Response, Sample of Returned Results, and Sample of Returned Failure, and add configurations in the Error Code Definition section. API Gateway does not parse responses, but forwards the responses to API requesters.

2.5.2.3. Security authentication

The security authentication methods that are supported by API Gateway include Alibaba Cloud applications and none.

- Alibaba cloud applications: An application must be authorized by the API provider to call an API. An API caller must provide an AppKey and encrypted signature. Otherwise, the API request validation will fail. For more information about the signature method, see Encrypt a signature.
- None: The API can be called without authorization after it is published. The AppKey and encrypted signature are not required when you make an API request.

2.5.2.4. Configure a network protocol

HTTPS domain names are not supported in the API Gateway console. To use an HTTPS domain name, you can call the API operations of API Gateway.

To configure a network protocol, perform the following operations: Find the target API on the API List page in the API Gateway console, and click Manage in the Operation column. On the **API Definition** page, click **Edit** in the upper-right corner. On the page that appears, specify Protocol in the **Define API Request** step.

Valid values of Protocol:

- HTTP
- HTTPS
- WEBSOCKET

2.5.2.5. Configure a request body

You can configure a request body when the HTTP method is POST, PUT, or PATCH. You can use the following methods to configure the request body. The methods are mutually exclusive.

- Form-based request body: Add a request parameter in the Input Parameter Definition section of the Define API Request step on the Create API page, and select Body from the Param Location drop-down list. The configured request body can only be used to transmit form data.
- Non-form-based request body: If the body content to be transmitted is in the JSON or XML format, select Non-Form data, such as JSON, Binary data in the Request Body section of the Define API Request step on the Create API page. The size of a request body cannot exceed 8 MB.

2.5.2.6. Configure an API in Mock mode

In most cases, business partners can work in combination to develop a project. The project development process is hindered due to the interdependence among business partners. Misunderstandings can also arise and affect the development progress or even cause severe delays to the project. The Mock mode is used to simulate the predetermined API responses in the project development process. This reduces misunderstandings and improves development efficiency.

API Gateway provides a simple configuration process of an API in Mock mode.

Configure an API in Mock mode

Log on to the API Gateway console. In the left-side navigation pane, choose **Publish APIs** > **APIs**. On the API List page, find the target API and click **Manage** in the Operation column. On the API Definition page, click **Edit** in the upper-right corner.

On the page that appears, configure the Mock mode in the **Define API Backend Service** step.

1. Set **Backend Service Type** to **Mock**.
2. Specify Mock Result in the Mock Configuration section.

   Enter your responses as the Mock-based response body. Responses can be in the JSON, XML, or text format. Example:

   ```
   {
   "result": {
      "title": " Mock test for API Gateway",
   }
   }
   ```

   Save the settings and then publish the API to the Test or Release environment for testing.
3. Specify **HTTP Status Code** based on HTTP status code specifications. Enter 200 to indicate a successful API request.
4. Specify **Mock Header**. You can click +Add Item to add a Mock response header based on your business requirements.

2.5.2.7. Return the Content-Type header

The value of the Content-Type header is only used to generate API documentation. It does not affect responses returned by the back-end service. The Content-Type header is returned by the back-end service.

2.5.3. API management

2.5.3.1. View and modify an API

You can view and modify an API as required.

> **Note** If you modify an API that is published, the modifications are not immediately applied. You must republish the modified API to synchronize the changes to the Release environment.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the target API.
   - Click **Manage** in the Operation column. On the API Definition page, you can view the information of the API.
   - Click **Edit** in the upper-right corner to edit the API as required.

The procedure of creating an API is similar to that of modifying an API. For more information about how to create an API, see Create an API. If you want to cancel the modifications before the modifications are submitted, click Cancel Edit in the upper-right corner of the edit page.

2.5.3.2. Publish an API

After you create an API, you must publish the API to the Test, Pre, or Release environment before it can be called.

- When you use a second-level or independent domain to access an API that is published to a specified environment, you must specify the environment in the request header.
- If you publish an API that already has a running version in the Test or Release environment, the running version is automatically overwritten by the new version within 15s. However, all historical versions and definitions are recorded. This allows you to roll the API back to an earlier version.
- You can unpublish an API in the Test or Release environment. The plugin binding relationship or the app authorization relationship is retained after you unpublish an API. These relationships take effect again if the API is republished. You can also perform related operations to remove the authorization or unbind a required plugin.

Step 1: Publish the API

After the test is complete, you can publish the API.

API Gateway allows you to manage different versions of APIs in the Test or Release environment. You can publish or unpublish the API, and switch the version of the API. The version switch takes effect in real time.

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the target API and click **Deploy** in the Operation column.
4. In the Deploy API dialog box, specify Enter Change Remarks and click **Deploy**.

Step 2: Test the API

To simulate API requests, you can create an app and authorize the app to call your API.

You can compile code based on actual scenarios, or use the SDK samples provided by API Gateway to call your API.

You can publish the API to the Test or Release environment. If no independent domain is bound to the group to which the API belongs, you can test or call the API by using a second-level domain. When you make an API request, set the X-Ca-Stage header to TEST, PRE, or RELEASE to specify the environment of the API. If you do not specify the header, the API will be invoked to the Release environment.

2.5.3.3. Authorize an app

You must authorize an app before it can call an API. After you publish an API to the Release environment, you must authorize apps to call the API. You can grant or revoke the authorization of an app to call an API. API Gateway verifies the authorization relationship.

> Note
>
> - You can authorize one or more apps to call one or more APIs.
> - If an API is published to both the Test and Release environments and an app is authorized to call the API in the Test environment, the app can call only the API in the Test environment.
> - You can find an app based on its ID.
> - If you want to revoke the authorization of an app to call an API, go to the Authorization page of the API. Then select the required app and click Revoke Authorization in the lower-left corner.

An app indicates the identity of a requester. Before testing or calling an API, you or your users must create an app that is used as the identity of a requester. Then, you must authorize the app to call the API.

> Note Authorizations are environment-specific. If you want to use an app to call an API in both the Test and Release environments, you must authorize the app in both environments. Otherwise, errors may occur due to the inconsistency between the authorized environment and the requested environment.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the target API and click **Authorize** in the Operation column.
4. In the Authorize dialog box, specify Select The Stage For Authorization and Select The APP For Authorization.**My APP** is automatically selected from the drop-down list on the left. Click **Search**. Apps created under your account appear. If you want to authorize an app created under a different account, select **APP ID** from the drop-down list on the left, enter the app ID in the search bar, and click **Search**.To view the ID of an app, click **Consume APIs** and then **APPs** in the left-side navigation pane. On the APP List page, click the name of the target app to go to the APP details page.
5. Select an app to be authorized and click +Add in the Operation column to add this app to the right pane. Alternatively, you can select multiple apps to be authorized at a time and click Add Selected in the lower-left corner of the page to add these apps to the right pane.
6. Click **OK** to complete the authorization.
7. Click Manage in the Operation column that corresponds to the target API. On the API Definition page, click **Authorization** in the left-side navigation pane to view the authorized apps.

2.5.3.4. Revoke an authorization

You can revoke the authorization of an app to call an API.

Procedure

1. Log on to the API Gateway console.

2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, click the name of the target API for which you want to revoke the authorization. On the API Definition page, click **Authorization** in the left-side navigation pane.
4. Select target apps and click **Revoke Authorization** in the lower-left corner.
5. In the Confirm authorization revocation message, click **OK**.

2.5.3.5. Unpublish an API

You can unpublish an API.

You can unpublish an API in the Test or Release environment. The binding or authorization relationships of policies, keys, and apps are retained after you unpublish an API. These relationships will take effect again if the API is republished. To remove these relationships, you must delete the API.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the target API and click **Undeploy** in the Operation column.
4. In the Undeploy api message, click **OK**.

2.5.3.6. View the version history of an API

You can view the version history of an API, including the version number, description, environment, publish time, and specific definition of each version.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the target API and click **Manage** in the Operation column to go to the API Definition page.
4. Click **Deployment History** in the left-side navigation pane. You can view the version history of this API.
5. On the Deployment History page, find the target version and click **View** in the Operation column.

2.5.3.7. Change the version of an API

When you view the version history of an API, you can select a different version to switch the API to this version. The selected version then replaces the previous version and takes effect in the specified environment.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.

3. On the API List page, find the target API and click **Manage** in the Operation column to go to the API Definition page.
4. Click **Deployment History** in the left-side navigation pane.
5. Find the target version and click **Switch to this version** in the Operation column.
6. In the API Version Switch dialog box, enter the description and click **Switch**.

2.5.4. Plugin management

2.5.4.1. Use parameters and conditional expressions

In an `access control plugin`, `throttling plugin`, `backend routing plugin`, or `error code mapping plugin`, you can obtain parameters from requests, responses, and system context. Then, you can use conditional expressions to evaluate these parameters. This topic describes how to define parameters and write conditional expressions.

1. Define parameters

1. Definition methodBefore you use a conditional expression, you must explicitly define all the parameters required in this conditional expression in the parameters field. Example:

   ```
   ---
   parameters:
    method: "Method"
    appId: "System:CaAppId"
    action: "Query:action"
    userId: "Token:UserId"
   ```

   The parameters specified in `parameters` are key-value pairs of the string type.

   - `key` indicates the name of a variable to be used in a conditional expression. The name must be unique and must conform to the following regular expression: `[a-zA-Z_][a-zA-Z0-9]+`.
   - `value` indicates the location of a parameter. It is specified in the `{location}` or `{location}:{name}` format.
     - `location` indicates the location of a parameter. For more information, see the following table.
     - `name` indicates the name of a parameter, which is used to locate the parameter at a specific location. For example, `Query:q1` indicates the first value of the query string named q1.

2. Parameter locationsBefore you use a conditional expression, you must define the parameters that are required in this conditional expression. The following table describes parameters at specific locations that can be used by various plugins.

Location	Included in	Description

Location	Included in	Description
Method	Request	The HTTP request method, in uppercase, such as GET or POST.
Path	Request	The complete HTTP request path, such as `/path/to/query`.
StatusCode	Response	The HTTP status code in a backend response, such as 200 or 400.
ErrorCode	Response	Error codes.
Header	Request/Response	Use `Header:{Name}` to obtain the first value of the HTTP header that is specified by `{Name}`.
Query	Request	Use `Query:{Name}` to obtain the first value of the query string that is specified by `{Name}`.
Form	Request	Use `Form:{Name}` to obtain the first value of the form that is specified by `{Name}`.
Host	Request	Use `Host:{Name}` to obtain the template parameters of the matched wildcard domain names.
Parameter	Request	Use `Parameter:{Name}` to obtain the first value of the custom API parameter that is specified by {Name}.
BodyJsonField	Response	Use `BodyJsonField:{JPath}` to obtain the JSON string in the body of an API request or a backend response in JSONPath mode.
System	Request/Response	Use `System:{Name}` to obtain the value of the system parameter that is specified by `{Name}`.

Location	Included in	Description
Token	Request/Response	If JWT is used for authentication, use `Token:{Name}` to obtain the value of the parameter that is specified by `{Name}` in a token.

Rules for use:

- You can use the following plugins at the request phase: `access control plugin`, `throttling plugin`, and `backend routing plugin`. These plugins support only the parameters at the following locations: `Method`, `Path`, `Header`, `Query`, `Form`, `Parameter`, `System`, and `Token`.
- You can also use the `error code mapping plugin` at the response phase. This plugin supports only the parameters at the following locations: `StatusCode`, `ErrorCode`, `Header`, `BodyJsonField`, `System`, and `Token`.
- Parameters at the `Method`, `Path`, `StatusCode`, and `ErrorCode` locations are defined in the {location} format.
- If you use parameters at the `Header` location in a plugin at the request phase, headers from client requests are read. If you use these parameters at the response phase, headers from backend responses are read.
- Parameters at the `Parameter` location are available only for plugins at the request phase. A `frontend parameter`, instead of a `backend parameter`, is used to search for the parameter with the same name in the API definition. If no parameter with the same name exists, a null value is returned.
- A complete request path is returned from `Path`. If you require a parameter at the Path location, use the corresponding parameter at the `Parameter` location.
- Parameters at the `BodyJsonField` location are available only for the `error code mapping plugin`. Obtain the JSON string in the body of a backend response in `JSONPath` mode. For more information, see Usage notes of JSONPath.
- If `JWT` is used for authentication, use `Token:{CliamName}` to obtain the value of the parameter specified by `{CliamName}` in a token. For more information, see the plugin documentation.

3. Usage notes of JSONPathJSONPath is available only for the `error code mapping` plugin at the `BodyJsonField` location. It is used to extract the `JSON` string in the body of a backend response. For more information about JSONPath, see the JSONPath overview documentation.

   Example: When you use the expression `code:"BodyJsonField:$.result_code"`, you can obtain the value of `result_code` from the following body. `code:ok` is parsed from the following body.

```
{ "result_code": "ok", "message": ... }
```

4. System parameters

Parameter	Description	Value
CaClientIp	The IP address of the request client.	Example value: 37.78.3.3.
CaDomain	The full domain name in a request, with a Host header.	Example value: api.foo.com.
CaAppId	The ID of the application that sends the request.	Example value: 49382332.
CaAppKey	The key of the application that sends the request.	Example value: 12983883923.
CaRequestId	The unique ID of the request generated by API Gateway.	Example value: CCE4DEE6-26EF-46CB-B5EB-327A9FE20ED1.
CaApiName	The API name.	Example value: TestAPI.
CaHttpSchema	The protocol used by the client to call operations.	Valid values: http, https, and ws.
CaClientUa	The UserAgent header of the client.	Used to transparently pass values uploaded by the client.
CaStage	The running environment of API Gateway.	Valid values: TEST, PRE, and RELEASE.

2. Write conditional expressions

You can use conditional expressions in plugins or other scenarios to evaluate parameters in a wide variety of scenarios.

1. Basic syntax
   - Conditional expressions are similar to SQL statements. Example: `$A > 100 and '$B = 'B'` .
   - An expression is in the following format: `{Parameter}{Operator}{Parameter}` . In the preceding example, you can specify a `variable` or a `constant` for `$A > 100` .
   - A `variable` starts with $ and references a parameter defined in the context. For example, `q1:"Query:q1"` is defined in `parameters` . You can use the variable `$q1` in your expression. The value of this variable is the value of the `q1` query parameter in the request.
   - A `constant` can be a `string` , `number` , or `Boolean value` . Examples: `"Hello",' foo', 100, -1, 0.1, and true` . For more information, see Value types and evaluation rules.
   - The following `operators` are supported:

- `=` and `==` : equal to.
- `<>` and `!=` : not equal to.
- `>` , `>=` , `<` , and `<=` : comparison.
- `like` and `!like` : check whether a specific string matches a specified pattern. The percent sign `%` is used as a wildcard in the evaluation. Example: `$Query like 'Prefix%'` .
- `in_cidr` and `!in_cidr` : specify the mask of an IP address. Example: `$ClientIp in_cidr '47.89.0.0/24'` .

- You can use `null` to check whether a parameter is empty. Example: `$A == null` or `$A != null` .
- You can use the operators `and` , `or` , and `xor` to combine different expressions in a right-to-left order by default.
- You can use parentheses `()` to specify the priority of conditional expressions.
- You can use `!()` to perform the logical negation operation on the enclosed expression. For example, the result of `!(1=1)` is false.
- The following built-in functions are used for evaluation in some special scenarios:
  - `Random()` : generates a parameter of the floating-point number type. The parameter value ranges from 0 to 1. This parameter is used in scenarios where random input is required, such as blue-green release.
  - `Timestamp()` : returns a UNIX timestamp representing the number of milliseconds that have elapsed since the epoch time January 1, 1970, 00:00:00 UTC.
  - `TimeOfDay()` : returns the number of milliseconds from the current time to 00:00 of the current day in GMT.

2. Value types and evaluation rules
   - The following value types are supported in expressions:
     - `STRING` : The value can be a string. Single quotation marks (' ') or double quotation marks (" ") can be used to enclose a string. Examples: `"Hello"` and `'Hello'` .
     - `NUMBER` : The value can be an integer or a floating-point number. Examples: `1001` , `-1` , `0.1` , and `-100.0` .
     - `BOOLEAN` : The value can be a Boolean value. Valid values: `true` and `false` .
   - For the operator types `equal to` , `not equal to` , and `comparison` , the following evaluation rules apply:
     - `STRING` type: uses the string order for evaluation. Examples:
       - `'123' > '10000'` : The result is true.
       - `'A123' > 'A120'` : The result is true.
       - `'' <'a'` : The result is true.

- `NUMBER` type: uses numerical values for evaluation. Examples:
  - `123 > 1000` : The result is false.
  - `100.0 == 100` : The result is true.
- `BOOLEAN` type: For Boolean values, true is greater than false. Examples:
  - `true == true` : The result is true.
  - `false == false` : The result is true.
  - `true > false` : The result is true.

- For the operator types `equal to` , `not equal to` , and `comparison` , if the value types before and after an operator are different, the following evaluation rules apply:
  - Assume that a value before an operator is of the `STRING` type and that after the operator is of the `NUMBER` type. If the value type before the operator can be changed to `NUMBER` , use numerical values for evaluation. Otherwise, use the string order for evaluation. Examples:
    - `'100' == 100.0` : The result is true.
    - `'-100' > 0` : The result is false.
  - Assume that a value before an operator is of the `STRING` type and that after the operator is of the `BOOLEAN` type. If the value type before the operator can be changed to `BOOLEAN` and the value is not case-sensitive, use `BOOLEAN` values for evaluation. Otherwise, except for the evaluation result of `!=` , all the other evaluation results are `false` . Examples:
    - `'True' == true` : The result is true.
    - `'False' == false` : The result is true.
    - `'bad' == false` : The result is false.
    - `'bad' != false` : The result is true. If the value before the operator is not `true` or `false` , only the result for `!=` is true.
    - `'bad' != true` : The result is true.
    - `'0' > false` : The result is false.
    - `'0' <= false` : The result is false.
  - Assume that a value before an operator is of the `NUMBER` type and that after the operator is of the `BOOLEAN` type. The result is false.
- The `null` value is used to check whether a parameter is empty. For the operator types `equal to` , `not equal to` , and `comparison` , the following evaluation rules apply:
  - If the `$A` parameter is empty, the result of `$A == null` is true, and the result of `$A !=`

`null` is false.

- If the empty string `''` is not equal to `null`, the result of `'' == null` is false, and the result of `'' == ''` is true.
- For the `comparison` operator type, if the value on either side of the operator is `null`, the result is false.

- `like` and `!like` operators are used to match the prefix, suffix, and inclusion of a string. The following evaluation rules apply:
  - In an expression, the value after the operator must be a constant of the `STRING` type. Example: `$Path like '/users/%'`.
  - The `'%'` wildcard character in the value after the operator is used to match the prefix, suffix, or inclusion of a string. Examples:
    - Prefix matching: `$Path like '/users/%'` and `$Path !like '/admin/%'`
    - Suffix matching: `$q1 like '%search'` and `$q1 !like '%.do'`
    - Inclusion relation matching: `$ErrorCode like '%400%'` and `$ErrorCode !like '%200%'`
  - If the value type before an operator is not `NUMBER` or `BOOLEAN`, change the type to `STRING` and then perform the evaluation.
  - If the value before an operator is `null`, the result is `false`.
- `in_cidr` and `!in_cidr` operators are used to identify the mask of a CIDR block. The following evaluation rules apply:
  - The value after an operator must be a constant of the `STRING` type and must be an IPv4 or IPv6 CIDR block. Examples:
    - `$ClientIP in_cidr '10.0.0.0/8'`
    - `$ClientIP !in_cidr '0:0:0:0:0:FFFF::/96'`
  - If the value type before an operator is `STRING`, the value is considered an IPv4 CIDR block for evaluation.
  - If the value type before an operator is `NUMBER` or `BOOLEAN` or the value is empty, the result is `false`.
  - The `System:CaClientIp` parameter specifies the IP address of the client, which is used for evaluation.

3. Use cases

- The following expression indicates that the probability is less than 5%:

```
Random() < 0.05
```

- The following expression indicates that the requested API is published to the Test environment:

```
parameters:
  stage: "System:CaStage"
```

```
$CaStage='TEST'
```

- The following expression indicates that the custom parameter UserName is set to Admin and the source IP address is `47.47.74.0/24`:

```
parameters:
 UserName: "Token:UserName"
 ClientIp: "System:CaClientIp"
```

```
$UserName = 'Admin' and $CaClientIp in_cidr '47.47.74.0/24'
```

- The following expression indicates that the AppId parameter is set to 1001, 1098, or 2011, and the protocol that is used by the API request is HTTPS:

```
pameters:
 CaAppId: "System:CaAppId"
 HttpSchema: "System:CaHttpSchema"
```

```
$CaHttpScheme = 'HTTPS' and ($CaAppId = 1001 or $CaAppId = 1098 or $CaAppId = 2011)
```

- The following expression indicates that the JSON string in a body contains result_code that is not ok when StatusCode in a response is 200:

```
parameters:
 StatusCode: "StatusCode"
 ResultCode: "BodyJsonField:$.result_code"
```

```
$StatusCode = 200 and ($ResultCode <> null and $ResultCode <> 'ok')
```

4. Limits

- A maximum of 16 parameters can be specified in a plugin.
- A conditional expression can contain a maximum of 512 characters.
- The size of a request or response body specified by BodyJsonField cannot exceed 16 KB. Otherwise, the settings will not take effect.

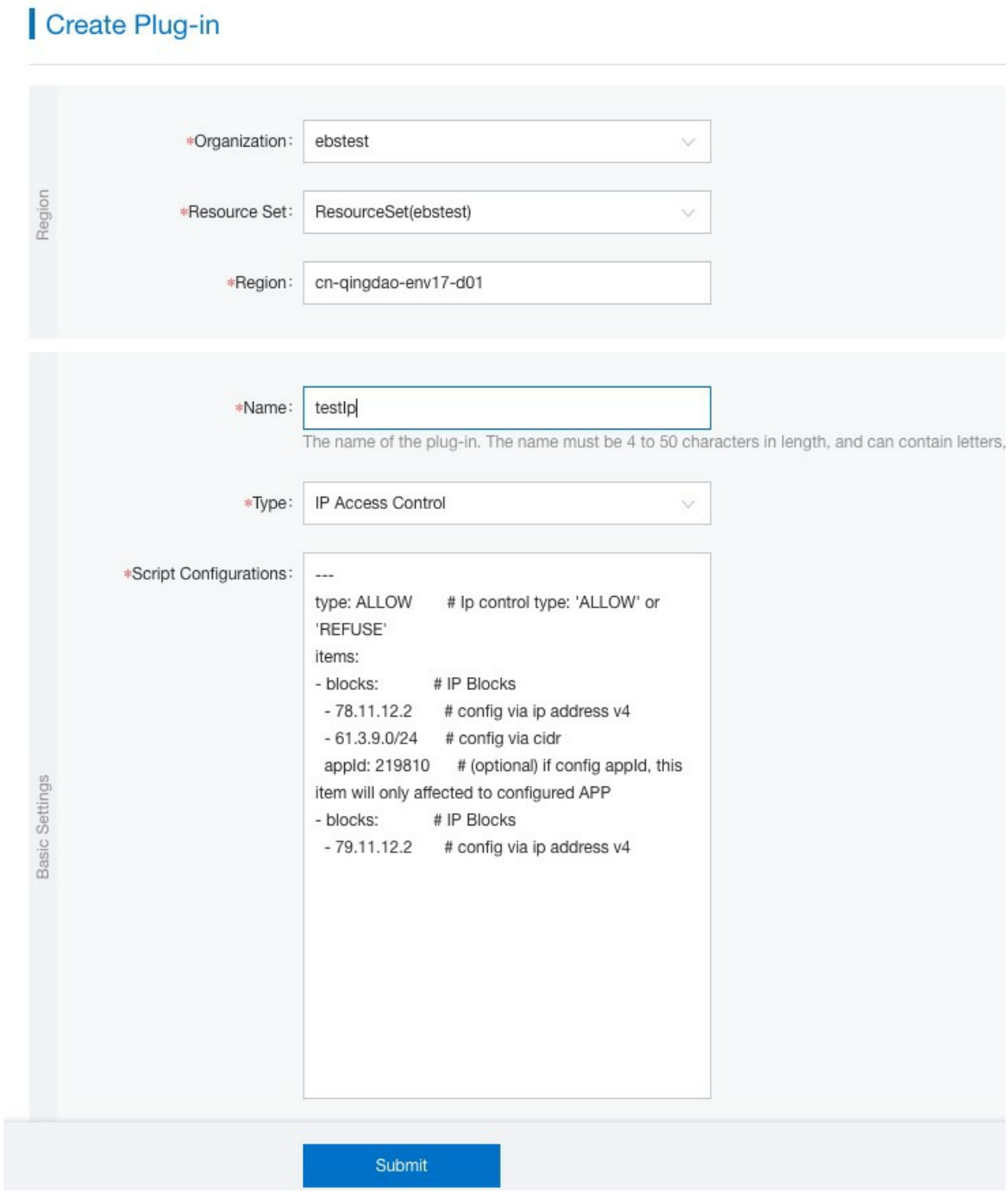
2.5.4.2. Create a plugin

2.5.4.2.1. Create an IP address-based access control plugin

IP address-based access control helps API providers configure an IP address whitelist or blacklist for API calls. This topic describes how to create an IP address-based access control plugin.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > Plugin.**
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create plugin page, specify **Organization**, **Resource Set**, **Region**, and **Name**, and set Type to **IP Access Control**. A plugin definition template in the YAML format is automatically loaded in the **Script Configurations** field. Modify template content.



Parameter	Description

Parameter	Description
type	○ ALLOW: You can configure a whitelist to allow the API requests that meet specific requirements. The following types of whitelists are supported: ■ You can configure a whitelist that includes only IP addresses. In this case, only API requests from the IP addresses in the whitelist are allowed. ■ You can configure a whitelist that contains apps and their IP addresses. In this case, each app can send API requests only from its IP addresses in the whitelist. ○ REFUSE: You can configure an IP address blacklist. API Gateway rejects all API requests from the IP addresses in the blacklist.

Script template of the IP address-based access control plugin

```
---
type: ALLOW      # The type of access control. You can set this parameter to ALLOW to apply a whi
telist or to REFUSE to apply a blacklist.
items:
- blocks:       # The IP address segment.
  - 78.11.12.2    # Specifies an IP address.
  - 61.3.9.0/24   # Specifies a CIDR block.
  appId: 219810   # Optional. If you specify this parameter, this IP address-based access control po
licy applies only to the app specified by this parameter.
- blocks:       # The IP address segment.
  - 79.11.12.2    # Specifies an IP address.
```

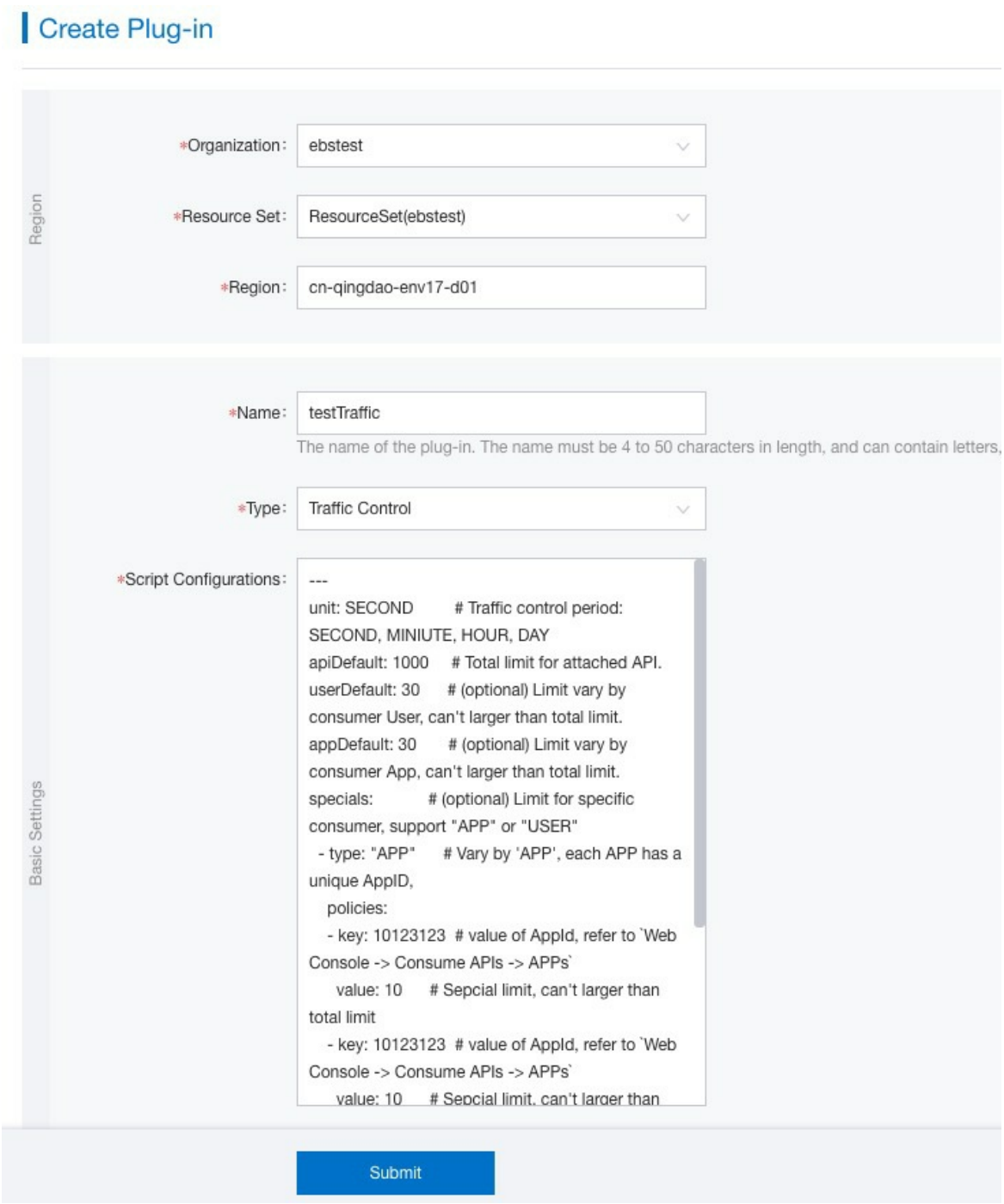
4. Click **Submit**.

2.5.4.2.2. Create a throttling plugin

You can use a throttling plugin to limit the number of API requests. A throttling plugin helps prevent a backend service from being overwhelmed by a large number of API requests. This topic describes how to create a throttling plugin.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > Plugin**.
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create plugin page, specify **Organization**, **Resource Set**, and **Region**. Then specify **Name**, set Type to **Traffic Control**, and modify configurations in the Script Configurations field.



Parameter	Description
unit	The unit of time. Valid values: SECOND, MINUTE, HOUR, and DAY.
apiDefault	The default API-level throttling threshold. It indicates the maximum number of times that an API bound with a throttling policy can be called within a specific unit of time. This parameter is set based on the backend service capability. This parameter is required.

Parameter	Description
userDefault	The default user-level throttling threshold. It indicates the maximum number of times that each user can call an API that is bound with a throttling policy within a specific unit of time. The user-level throttling threshold cannot be greater than the API-level throttling threshold. This parameter is optional.
appDefault	The default app-level throttling threshold. It indicates the maximum number of times that each app can call an API that is bound with a throttling policy within a specific unit of time. The app-level throttling threshold cannot be greater than the user-level throttling threshold. This parameter is optional.
specials	The special throttling settings. This parameter is optional. You can set throttling thresholds for special apps or users in a throttling policy. After this parameter is specified, the special throttling settings prevail for special apps or users.

Script template

```
---
unit: SECOND        # The unit of time. Valid values: SECOND, MINUTE, HOUR, and DAY.
apiDefault: 1000    # The default API-level throttling threshold.
userDefault: 30     # Optional. The default user-level throttling threshold. If you set this threshold to 0, user-level throttling is not performed. The user-level throttling threshold cannot be greater than the API-level throttling threshold.
appDefault: 30      # Optional. The default app-level throttling threshold. If you set this threshold to 0, app-level throttling is not performed. The app-level throttling threshold cannot be greater than the user-level throttling threshold.
specials:           # Optional. The special throttling settings. You can set throttling thresholds for special apps or users in a throttling policy.
 - type: "APP"      # The special throttling type. The value APP indicates that throttling is performed for special apps based on their AppKeys.
   policies:
   - key: 10123123  # The app ID. You can obtain the ID of an app from the app details page. To go to this page, click Consume APIs and then APPs in the left-side navigation pane of the API Gateway console and click the name of the app.
     value: 10      # The special throttling threshold for the app. This threshold cannot be greater than the user-level throttling threshold in the throttling policy.
   - key: 10123121  # The app ID.
     value: 10      # The special throttling threshold for the app. This threshold cannot be greater than the user-level throttling threshold in the throttling policy.
 - type: "USER"     # The special throttling type. The value USER indicates that throttling is performed for special Apsara Stack tenant accounts.
   policies:
   - key: 123455    # The ID of an Apsara Stack tenant account. You can move the pointer over the profile picture in the upper-right corner of the Alibaba Cloud Management Console to obtain the ID.
     value: 100     # The special throttling threshold for the Apsara Stack tenant account. This threshold cannot be greater than the API-level throttling threshold in the throttling policy.
```

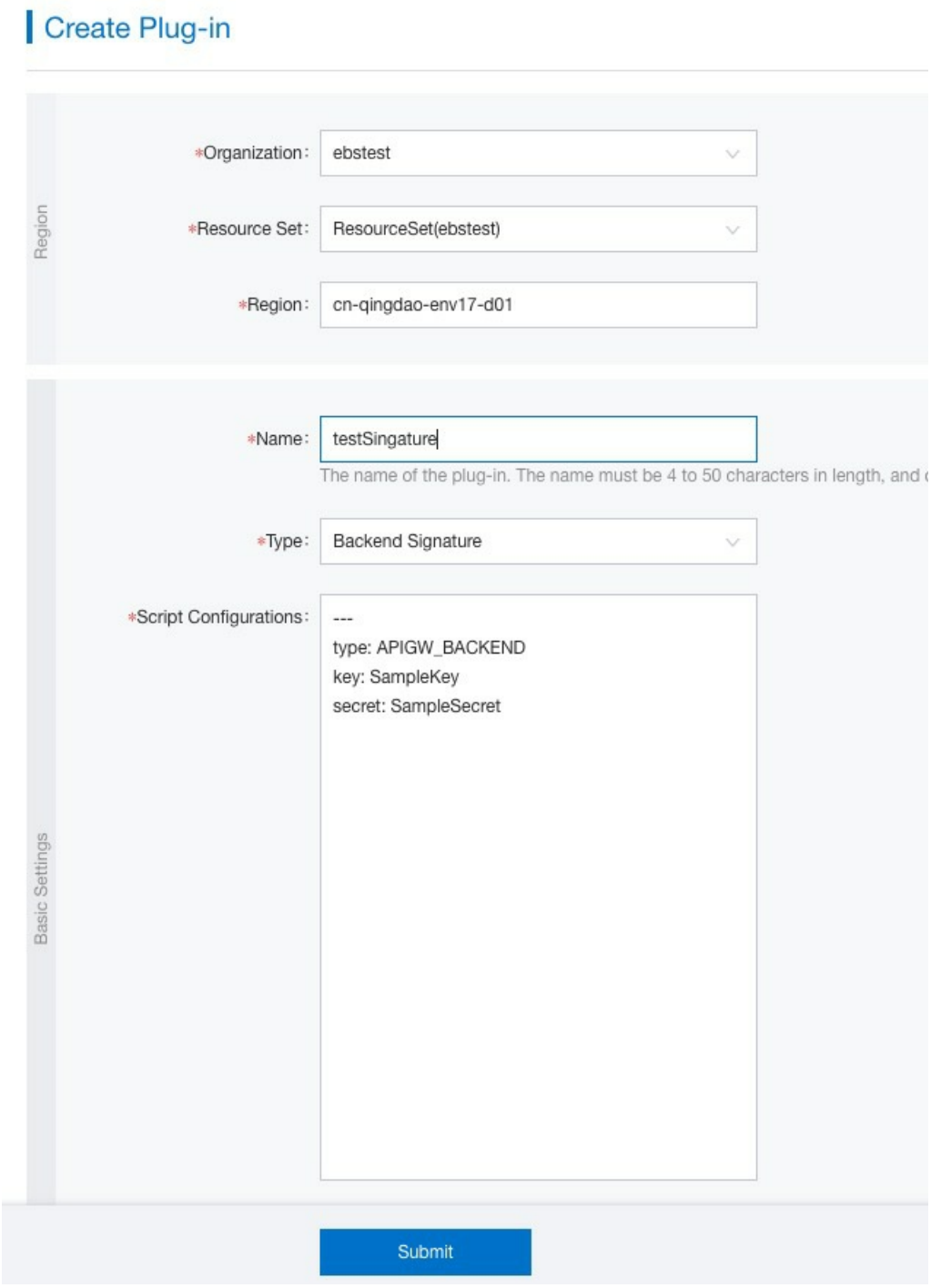
4. Click **Submit**.

2.5.4.2.3. Create a backend signature plugin

A backend signature plugin is used for signature verification between API Gateway and your backend service. A backend signature is a key-secret pair that you create and issue to API Gateway. It works in a way similar to an account and password pair. When API Gateway sends a request to your backend service, API Gateway uses the backend signature to calculate a signature string and pass it to your backend service. Your backend service obtains the signature string and authenticates API Gateway by using symmetric calculation. Perform the following steps to create a backend signature plugin:

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > Plugin.**
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create plugin page, specify **Organization**, **Resource Set**, **Region**, and **Name**. Set Type to **Backend Signature**.



Configure the plugin parameters as required.

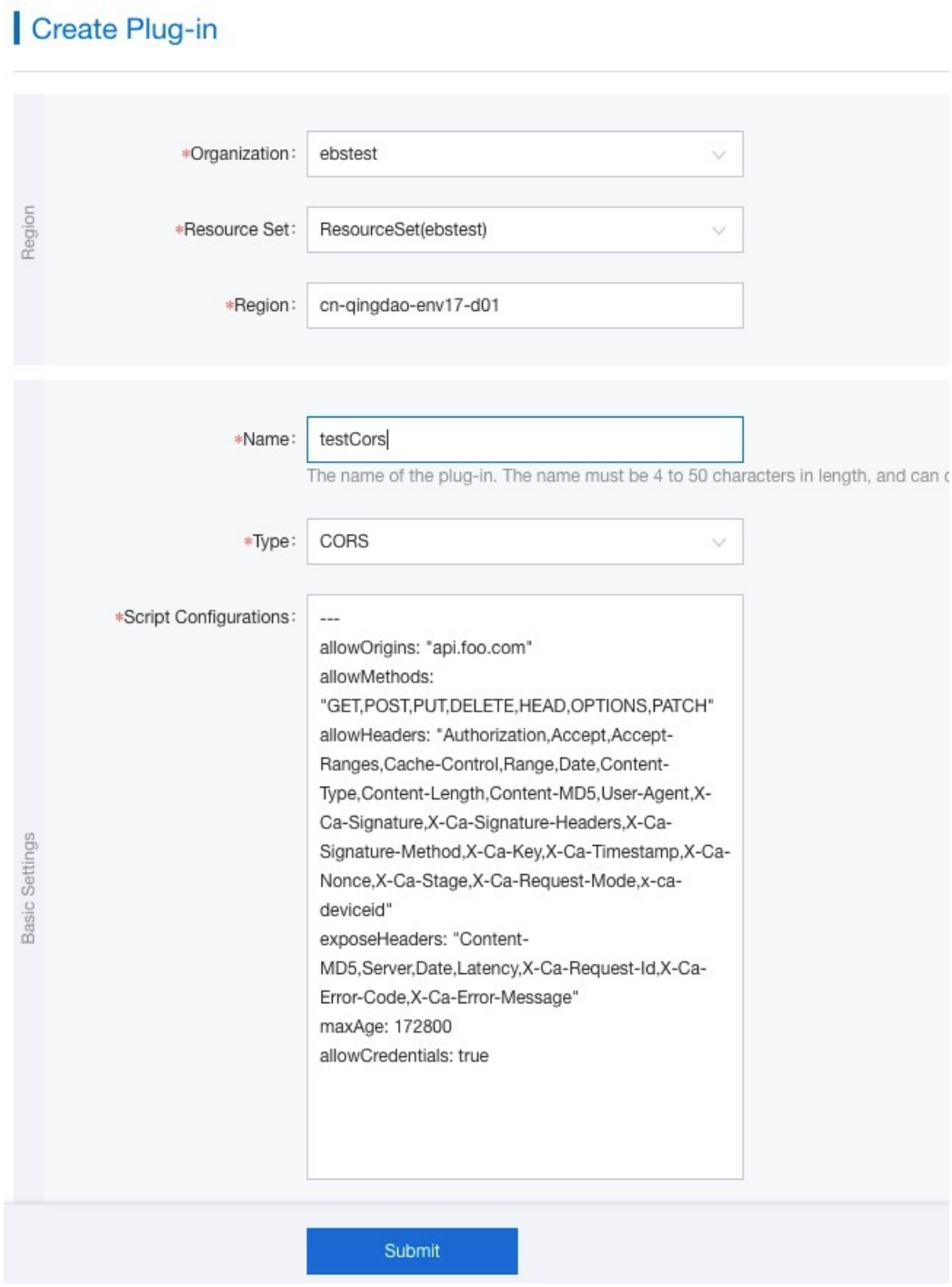
4. Click **Submit**.

2.5.4.2.4. Create a CORS plugin

This topic describes how to create a cross-origin resource sharing (CORS) plugin. If a resource requests another resource from a different domain or port of a different server, the former resource initiates a cross-domain HTTP request. For security purposes, the browser blocks the request and reports an error message. In this case, you need to use a CORS plugin to troubleshoot the issue.

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs** > **Plugin**.
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create plugin page, specify **Organization**, **Resource Set**, **Region**, and **Name**. Set Type to **CORS**.



Cross-domain access template

```
---
allowOrigins: api.foo.com,api2.foo.com   # The allowed origins. Separate origins with commas (,). Default value: *.
allowMethods: GET,POST,PUT               # The allowed HTTP methods. Separate methods with commas (,).
allowHeaders: X-Ca-RequestId             # The allowed request headers. Separate headers with commas (,).
exposeHeaders: X-RC1,X-RC2               # The headers that can be exposed to the XMLHttpRequest object. Separate headers with commas (,).
allowCredentials: true                   # Controls whether cookies are allowed.
maxAge: 172800
```

Configure the plugin parameters as required.

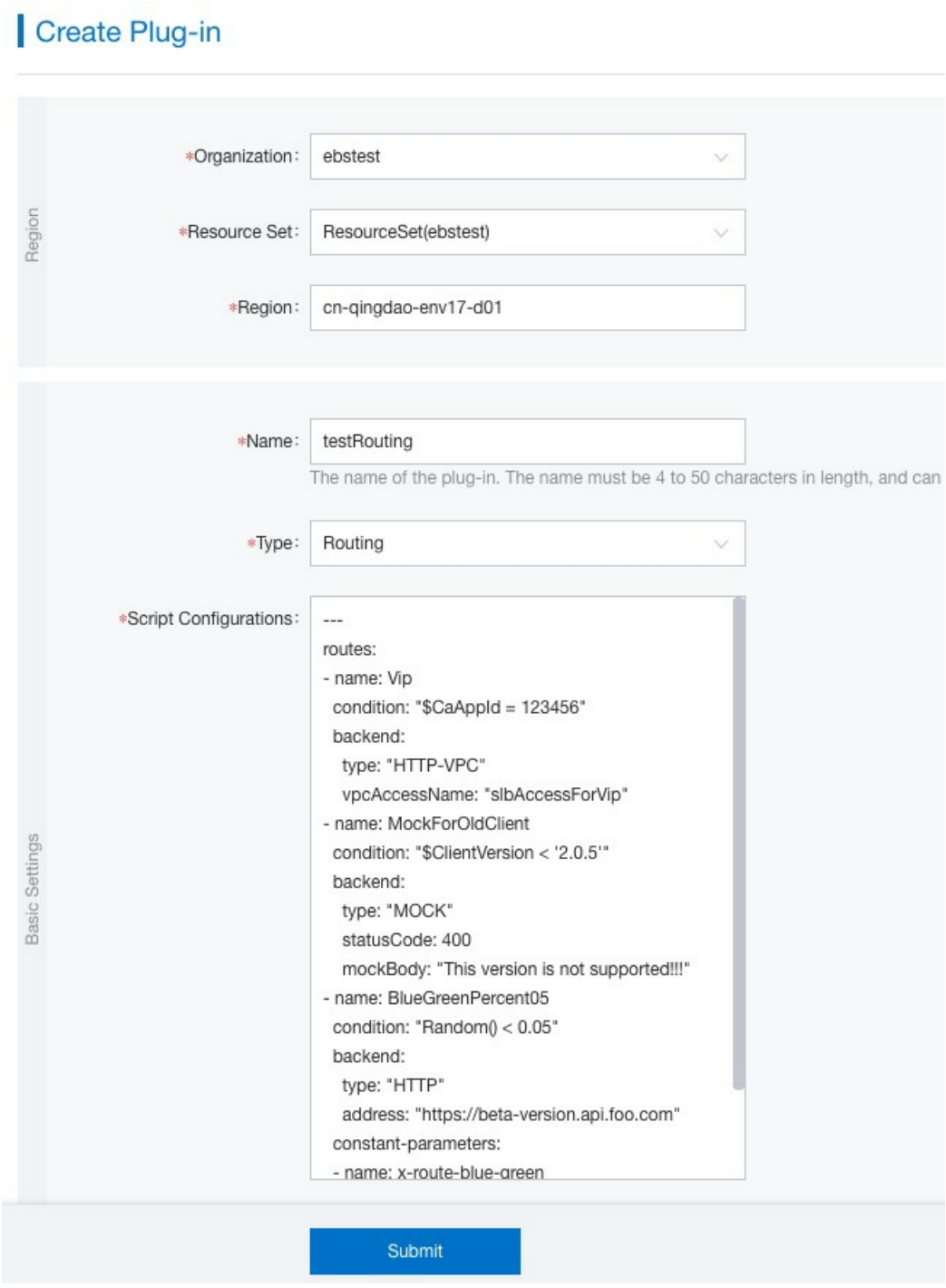
4. Click **Submit**.

2.5.4.2.5. Create a backend routing plugin

A backend routing plugin is used to route API requests to different backend services by changing the backend service type, backend service address, backend request path, and response parameters based on request and system parameters in API requests. Backend routing plugins can be used for multi-tenant routing and blue-green release. They can also be used to distinguish between different environments.

Create a backend routing plugin

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs** > **Plugin**.
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create plugin page, specify **Organization**, **Resource Set**, **Region**, and **Name**. Set Type to **Routing**.



4. Modify configurations in the Script Configurations field and click **Submit**.

Configurations

Configuration template

1. You can configure a backend routing plugin in the JSON or YAML format because these two formats have the same schema. You can use the yaml to json tool to convert the plugin configuration format. The following example describes a plugin configuration template in the YAML format:

```
---
routes:
# Responses that are no longer supported are returned to clients of an earlier version. ClientVers
ion is a custom parameter in the API.
- name: MockForOldClient
 condition: "$ClientVersion < '2.0.5'"
 backend:
  type: "MOCK"
  statusCode: 400
  mockBody: "This version is not supported!!!"
# Blue-green release scenarios: Five percent of requests are routed to the backend of a blue-gre
en release.
- name: BlueGreenPercent05
 condition: "Random() < 0.05"
 backend:   type: "HTTP"
  address: "https://beta-version.api.foo.com"
 constant-parameters:
 - name: x-route-blue-green
  location: header
  value: "route-blue-green"
```

The template has a root object routes that contains multiple route objects. Each route object is used to specify a routing rule. Each routing rule consists of the following parts:

- name: the name of the routing rule. The name must be unique within each plugin and can contain only letters and digits. If an API request hits the rule, an HTTP header X-Ca-Routing-Name that contains the name of the rule is added to the request before the request is routed to your backend service.
- condition: the conditional expression of the routing rule. If an API request meets the condition, the request hits the routing rule. The backend routing plugin checks the routing rules based on the order in which they are configured. The API request is routed to your backend service in the first routing rule that the request hits. After this occurs, the plugin does not check the remaining routing rules. If you configure multiple routing rules, make sure that they are configured in the order that meets your service expectations.
- backend: the description of your backend service. The description must be consistent with the Swagger specification files for API Gateway. The backend configurations for an API in a backend routing plugin override the original backend configurations in the API. If the backend configurations are incomplete after being overridden, the X-Ca-Error-Code: I504RB error is reported to the client. If this error is returned, check whether your backend configurations are complete.
- constant-parameters: the constant parameters that you can customize in the routing rule. Constant parameters are included in an API request before the request is routed to your backend service. These parameters are used in the business logic of your backend service. A constant parameter can be a query or header parameter.

Conditional expressions

Basic syntax

- The syntax of conditional expressions in backend routing plugins are similar to that of SQL statements. The basic format is $A = 'A' and '$B = 'B'.
- Each parameter starts with $. You can reference the request parameters that are defined in an API to which a plugin is bound. The request mode of the API can be set to Request Parameter Mapping(Filter Unknown Parameters), Request Parameter Mapping(Passthrough Unknown Parameters), or Request Parameter Passthrough. If you define a request parameter named query1 when you configure an API, you can use $query1 to reference this parameter in conditional expressions.
- The following constant parameter types are supported:
  - STRING: the string data type. Single or double quotation marks can be used to enclose a string, for example, "Hello".
  - INTEGER: the integer data type, for example, 1001 and -1.
  - NUMBER: the floating point data type, for example, 0.1 and 100.0.
  - BOOLEAN: the Boolean data type. Valid values: true and false.
- You can use and and or operators to connect different expressions.
- You can use parentheses () to specify the priority of conditional expressions.
- As a built-in function, Random() generates a NUMBER-type parameter that returns a random number in the range of [0, 1).
- You can use $CaAppId to reference system parameters of the current request. You can reference system parameters without the need to define them in an API. However, if you have defined a parameter in the API with the same name as a system parameter, the value of the system parameter is overwritten by that of the defined parameter. The following system parameters apply to backend routing plugins:
  - CaStage: the environment to which the requested API is published. Valid values: RELEASE, PRE, and TEST.
  - CaDomain: the domain name of the API group to which the requested API belongs.
  - CaRequestHandleTime: the time in UTC at which the current request is received.
  - CaAppId: the value of the AppId parameter in the current request.
  - CaAppKey: the value of the AppKey parameter in the current request.
  - CaClientIp: the IP address of the client from which the current request is sent.
  - CaApiName: the name of the requested API.
  - CaHttpScheme: the protocol used by the current request. Valid values: HTTP, HTTPS, and WS.
  - CaClientUa: the UserAgent field uploaded from the client.
- If you use an unknown parameter in a conditional expression, such as $UnknonwParameter = 1, the result of the expression is false.

Conditional expression examples

- The following expression indicates that the probability is less than 5%:

```
Random() < 0.05
```

- The following expression indicates that the requested API is published to the Test environment:

```
$CaStage = 'TEST'
```

- The following expression indicates that the custom parameter UserName is set to Admin and the source IP address is 47.47.74.77.

```
$UserName = 'Admin' and $CaClientIp = '47.47.74.77'
```

- The following expression indicates that the AppId parameter is set to 1001, 1098, or 2011, and the protocol that is used by the API request is HTTPS:

```
$CaHttpScheme = 'HTTPS' and ($CaAppId = 1001 or $CaAppId = 1098 or $CaAppId = 2011)
```

Backend configuration and overriding rules

The structure of a backend service is consistent with the Swagger definitions imported into API Gateway. The following examples describe the supported backend service types and configuration samples. The backend configurations in a backend routing plugin override the backend configurations in an API that is bound to the plugin. If you do not need to change the backend service type, specify only the parameters whose values you want to change.

- HTTP

```
---
backend:
 type: HTTP
 address: "http://10.10.100.2:8000"
 path: "/users/{userId}"
 method: GET
 timeout: 7000
```

- HTTP-VPC

```
---
backend:
 type: HTTP-VPC
 vpcId: vpc-xxxx
 vpcInstance: 172.168.1.1
 vpcInstancePort: 80
 path: "/users/{userId}"
 method: GET
 timeout: 10000
```

- MOCK

```
---
backend:
 type: MOCK
 mockResult: "mock resul sample"
 mockStatusCode: 200
 mockHeaders:
  - name: server
    value: mock
  - name: proxy
    value: GW
```

Limits

- The metadata of a backend routing plugin can be a maximum of 16,384 bytes in size. If this limit is exceeded, the InvalidPluginData.TooLarge error is reported.
- A maximum of 16 routing rules can be configured in a backend routing plugin. If this limit is exceeded, the InvalidPluginData.TooManyRoutes error is reported.
- The size of a single conditional expression cannot exceed 512 bytes. If this limit is exceeded, the InvalidPluginData.ConditionTooLong error is reported.
- Configuration updates in a plugin are synchronized in real time to all the APIs bound to the plugin. An interval of at least 45s is required between two updates. If you update a plugin twice within less than 45s, the InvalidPluginData.UpdateTooBusy error is reported.

Typical scenarios

- Configure multi-tenant routing. Different backend service addresses are allocated based on the AppId settings. Assume that users whose app ID is 10098 or 10099 are VIP customers. API requests from these two users are required to be routed to an independent server cluster.

```
---
-routes:
# If the AppId value for an API caller is 10098 or 10099, requests to the API are routed to an indepen
dent address.
# In this example, the VPC access name is set to slbAddressForVip.
- name: http1
 condition: "$CaAppId = 10098 or $CaAppId = 10099"
 backend:
  type: "HTTP"
  address: "https://test-env.foo.com"
```

- Configure routing based on environment settings (Test, Pre, and Release). All requests for the APIs that are published to the same environment are required to be routed to the same server.

```
---
routes:
# Route all requests for APIs that are published to the Test environment to the test server on the In
ternet.
- name: Vip
  condition: "$CaStage = 'TEST'"
  backend:
    type: "HTTP"
    address: "https://test-env.foo.com"
```

- Five percent of requests are required to be directed to a group of beta servers to perform a blue-green release.

```
---
routes:
# Blue-green release scenarios: Five percent of requests are routed to the backend of a blue-green
release.
- name: BlueGreenPercent05
  condition: "Random() < 0.05"
  backend:
    type: "HTTP"
    address: "https://beta-version.api.foo.com"
```

2.5.4.2.6. Create a caching plugin

You can bind a caching plugin to an API to cache the responses from your backend service. This reduces the load on the backend service and shortens the response time.

1. Usage notes

- Caching plugins can cache only the responses to API requests that use the GET method.
- When you configure a caching plugin, you can use the following parameters to sort responses in a cache:
  - varyByApp: controls whether to match and serve cached responses based on the app IDs of API callers.
  - varyByParameters: controls whether to match and serve cached responses based on the values of specific parameters. The plugin uses the same request parameters of APIs that are bound to the plugin to sort the responses to API requests.
  - varyByHeaders: controls whether to match and serve cached responses based on different request headers. For example, match and serve cached responses based on the `Accept` or `Accept-Language` header.
- API Gateway provides each user with 5 MB of cache space in each region. Caches are cleared after expiration. If a cache reaches its space limit, no more responses are stored in the cache.
- If `Cache-Control` is specified in a response from your backend service, the response is stored

in a cache based on the specified cache policy. If Cache-Control is not specified in a response, after the response expires, the response is stored in a cache based on the default cache policy and is stored for the period of time specified by the duration parameter.

- A response can be stored in a cache for a maximum of 48 hours (172,800 seconds) after it expires. Configurations made after the 48 hours are invalid.
- API Gateway determines how to process the `Cache-Control` headers of client requests based on the clientCacheControl settings. By default, API Gateway does not process the Cache-Control headers. You can set clientCacheControl to the following modes:
  - off: API Gateway ignores the `Cache-Control` headers of all client requests.
  - all: API Gateway processes the `Cache-Control` headers of all client requests.
  - app: API Gateway processes only the `Cache-Control` headers of client requests whose `app IDs` are included in the configured `apps` list.
- By default, API Gateway caches only the `Content-Type`, `Content-Encoding`, and `Content-Language` headers in responses. If you need to cache more headers, add the headers in the `cacheableHeaders` parameter of the caching plugin.

2. Configurations

You can configure a caching plugin in the JSON or YAML format because these two formats have the same schema. You can use the yaml to json tool to convert the plugin configuration format. The following example describes a plugin configuration template in the YAML format:

```
---
varyByApp: false    # Controls whether to match and serve cached responses based on the app IDs of API callers. Default value: false.
varyByParameters:   # Controls whether to match and serve cached responses based on the values of specific parameters.
- userId            # The name of a backend parameter. If the backend parameter is mapped to a parameter with a different name, set this parameter to the mapped parameter name.
varyByHeaders:      # Controls whether to match and serve cached responses based on different request headers.
- Accept            # Cached responses are matched and served based on the Accept header.
clientCacheControl: # API Gateway determines how to process the Cache-Control headers of client requests based on the clientCacheControl settings.
  mode: "app"       # Valid values: off, all, and apps. Default value: off. off indicates that API Gateway ignores the Cache-Control headers of all client requests. all indicates that API Gateway processes the Cache-Control headers of all client requests. apps indicates that API Gateway processes only the Cache-Control headers of client requests whose app IDs are included in the configured apps list.
  apps:             # A list of app IDs. If mode is set to app, API Gateway processes only the Cache-Control headers of client requests whose app IDs are in this list.
  - 1992323         # A sample app ID. It is not an AppKey.
  - 1239922         # A sample app ID. It is not an AppKey.
cacheableHeaders:   # The cacheable response headers. By default, API Gateway caches only the Content-Type and Content-Length headers of backend responses.
- X-Customer-Token  # The name of the cacheable response header.
duration: 3600      # The default grace period, in seconds.
```

3. Working mechanism

If an API request hits the cache of an API, the `X-Ca-Caching: true` header is included in the response to the API request.

4. Limits

- The metadata of a caching plugin can be a maximum of 16,380 bytes in size.
- A response body that exceeds 128 KB in size cannot be cached.
- Each user has a maximum of 30 MB of total cache space in each region.

2.5.4.2.7. JWT authentication plugin

RFC 7519-compliant JSON Web Token (JWT) is a simple method used by API Gateway to authenticate requests. `API Gateway` hosts the `public JSON Web Keys (JWKs)` of users and uses these JWKs to sign and authenticate JWTs in requests. Then, API Gateway forwards `claims` to backend services as backend parameters. This simplifies the development of backend applications.

Compared to the `OpenID Connect` feature, the `JWT authentication plugin` can implement the functions of this feature and bring the following benefits:

- You do not need to configure an additional `authorization API`. `JWTs` can be generated and distributed in multiple ways. API Gateway is only responsible for `JWT` authentication by using `public JWKs`.
- `JWKs` without `kid` specified are supported.
- Multiple `JWKs` can be configured.
- You can read token information from the `header` of a request or a `query` parameter.
- If you want to transmit a `JWT` in an Authorization header, such as `Authorization bearer {token}`, you can set `parameter` to Authorization and `parameterLocation` to header, so the token information can be correctly read.
- The `jti` claim-based anti-replay check is supported if you set `preventJtiReplay` to true.
- Requests that do not include tokens can be forwarded to backend services without verification if you set `bypassEmptyToken` to true.
- The verification on the `exp` setting for tokens can be skipped if you set `ignoreExpirationCheck` to true.

If you configure a `JWT authentication plugin` and bind it to an `API` for which the `OpenID Connect` feature is configured, the JWT authentication plugin takes effect in place of the `OpenID Connect` feature.

1. Obtain a JWK

RFC 7517-compliant JWK is used to sign and authenticate JWTs. If you want to configure a `JWT authentication plugin`, you must generate a valid `JWK` manually or by using an online `JWK generator` such as mkjwk.org. The following example shows a valid `JWK`. In the JWK example, the private key is used to sign the token, and the public key is configured in the `JWT authentication plugin` to authenticate the signature.

```
{
 "kty": "RSA",
 "e": "AQAB",
 "kid": "O9fpdhrViq2zaaaBEWZITz",
 "use": "sig",
 "alg": "RS256",
 "n": "qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBs
R-BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACO
x4Hfr-9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh
2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYd
P5oYQ"
}
```

*The preceding JWK is in the JSON format. If you want to configure a JWT authentication plugin in the YAML format, you must use a JWK in the YAML format.**

- For a `JWT authentication plugin`, you only need to configure a `public key`. Keep your `private key` safe. The following table lists the signature algorithms supported by the JWT authentication plugin.

Signature algorithm	Supported `alg` setting
RSASSA-PKCS1-V1_5 with SHA-2	RS256, RS384, RS512
Elliptic Curve (ECDSA) with SHA-2	ES256, ES384, ES512
HMAC using SHA-2	HS256, HS384, HS512

When you configure a key of the HS256, HS384, or HS512 type, the key value is base64url encoded. If the signature is invalid, check whether your key is in the same format as the key used to generate the token.

2. Plugin configurations

You can configure a JWT authentication plugin in the JSON or YAML format because these two formats have the same schema. You can use the `yaml to json` tool to convert the plugin configuration format. The following example describes a plugin configuration template in the YAML format:

```
---
parameter: X-Token          # The parameter from which the JWT is read. It corresponds to a parameter in an API request.
parameterLocation: header   # The location from which the JWT is read. Valid values: query and header. This parameter is optional if Request Mode for the bound API is set to Request Parameter Mapping(Filter Unknown Parameters) or Request Parameter Mapping(Passthrough Unknown Parameters). This parameter is required if Request Mode for the bound API is set to Request Parameter Passthrough.
preventJtiReplay: false     # Controls whether to enable the anti-replay check for jti. Default value: false.
bypassEmptyToken: false     # Controls whether to forward requests that do not include tokens to backend services without verification.
ignoreExpirationCheck: false # Controls whether to ignore the verification of the exp setting.
claimParameters:            # The claims to be converted into parameters. API Gateway maps JWT claims to backend parameters.
- claimName: aud            # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud      # The name of the backend parameter, to which the JWT claim is mapped.
  location: header          # The location of the backend parameter, to which the JWT claim is mapped. Valid values: query, header, path, and formData.
- claimName: userId         # The name of the JWT claim, which can be public or private.
  parameterName: userId     # The name of the backend parameter, to which the JWT claim is mapped.
  location: query           # The location of the backend parameter, to which the JWT claim is mapped. Vali
```

```
location: query        # The location of the backend parameter, to which the JWT claim is mapped. Valid values: query, header, path, and formData.
#
# Public key in the JWK
jwk:
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
#
# You can configure multiple JWKs and use them together with the jwk field.
# If multiple JWKs are configured, kid is required. If the JWT does not include kid, the consistency check on kid fails.
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz  # If only one JWK is configured, kid is optional. If the JWT includes kid, API Gateway checks the consistency of kid.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
- kid: 10fpdhrViq2zaaaBEWZITz  # If only one JWK is configured, kid is optional. If the JWT includes kid, API Gateway checks the consistency of kid.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v...
```

- The `JWT authentication plugin` retrieves JWTs based on the settings of `parameter` and `parameterLocation`. For example, if `parameter` is set to X-Token and `parameterLocation` is set to header, the JWT is read from the `X-Token` header.
- If the parameter configured in an API has the same name as the parameter specified by `parameter`, do not specify `parameterLocation`. Otherwise, an error is reported when the API is called.

- If you want to transmit a token in an Authorization header, such as `Authorization bearer {token}`, you can set `parameter` to Authorization and `parameterLocation` to header, so the token information can be correctly read.
- If `preventJtiReplay` is set to true, the JWT authentication plugin uses `jti` in `claims` to perform an anti-replay check.
- If `bypassEmptyToken` is set to true and a token is not included in a request, API Gateway skips the check and directly forwards the request to a backend service.
- If `ignoreExpirationCheck` is set to true, API Gateway skips the verification of the `exp` setting. Otherwise, API Gateway checks whether a token expires.
- If API Gateway is required to forward `claims` in tokens to backend services, you can set `tokenParameters` to configure the following parameters to be forwarded:
  - `claimName`: the name of the claim in a token, which can be `kid`.
  - `parameterName`: the name of the parameter forwarded to a backend service.
  - `location`: the location of the parameter forwarded to a backend service. Valid values: `header`, `query`, `path`, and `formData`.
    - If this parameter is set to `path`, the backend path must contain a parameter with the same name, such as `/path/{userId}`.
    - If this parameter is set to `formData`, the body of a received request in a backend service must be of the `Form` type.
- You can configure only one key in the `jwk` field. You can also configure multiple keys in the `jwks` field.
  - You can configure only one key with `kid` not specified.
  - You can configure multiple keys with `kid` specified. `kid` must be unique.

3. Verification rules

- A JWT authentication plugin obtains tokens based on the settings of `parameter` and `parameterToken`. If API Gateway is required to forward requests to backend services even when tokens are not included in the requests, set `bypassEmptyToken` to true.
- If you want to configure multiple keys, abide by the following principles:
  - Preferentially select a key whose ID is the same as the value of `kid` in a token for signature and authentication.
  - You can configure only one key with `kid` not specified. If there is no key whose ID is the same as the value of `kid` in a token, use the key with `kid` not specified for signature and authentication.
  - If all the configured keys have specified `kid` settings, and the token in a request does not contain `kid` or no keys match `kid`, an `A403JK` error is reported.
- If a token contains `iat`, `nbf`, and `exp`, the JWT authentication plugin verifies the validity of their time formats.

- By default, API Gateway verifies the setting of `exp`. If you want to skip the verification, set `ignoreExpirationCheck` to true.
- `tokenParameters` is configured to extract the required parameters from the `claims` of a token. These parameters are forwarded to backend services.

4. Configuration examples

4.1 Configure a single JWK

```
---
parameter: X-Token       # The parameter from which the JWT is read. It corresponds to a parameter in an API request.
parameterLocation: header  # The location from which the JWT is read. Valid values: query and header. This parameter is optional if Request Mode for the bound API is set to Request Parameter Mapping(Filter Unknown Parameters) or Request Parameter Mapping(Passthrough Unknown Parameters). This parameter is required if Request Mode for the bound API is set to Request Parameter Passthrough.
claimParameters:         # The claims to be converted into parameters. API Gateway maps JWT claims to backend parameters.
- claimName: aud         # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud    # The name of the backend parameter, to which the JWT claim is mapped.
  location: header       # The location of the backend parameter, to which the JWT claim is mapped. Valid values: query, header, path, and formData.
- claimName: userId      # The name of the JWT claim, which can be public or private.
  parameterName: userId   # The name of the backend parameter, to which the JWT claim is mapped.
  location: query        # The location of the backend parameter, to which the JWT claim is mapped. Valid values: query, header, path, and formData.
preventJtiReplay: false   # Controls whether to enable the anti-replay check for jti. Default value: false.
#
# Public key in the JWK
jwk:
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
```

4.2. Configure multiple JWKs

```
---
parameter: Authorization   # The parameter from which the token is obtained.
parameterLocation: header  # The location from which the token is obtained.
claimParameters:           # The claims to be converted into parameters. API Gateway maps JWT claims to backend parameters.
- claimName: aud           # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud     # The name of the backend parameter, to which the JWT claim is mapped.
  location: header         # The location of the backend parameter, to which the JWT claim is mapped. Valid values: query, header, path, and formData.
- claimName: userId        # The name of the JWT claim, which can be public or private.
  parameterName: userId    # The name of the backend parameter, to which the JWT claim is mapped.
  location: query          # The location of the backend parameter, to which the JWT claim is mapped. Valid values: query, header, path, and formData.
preventJtiReplay: true     # Controls whether to enable the anti-replay check for jti. Default value: false.
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz  # kid must be set to different values for different JWKs.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
- kid: 10fpdhrViq2zaaaBEWZITz  # kid must be set to different values for different JWKs.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v...
```

5. Error codes

Status	Code	Message	Description
400	I400JR	JWT required	No JWT-related parameters are found.

Status	Code	Message	Description
403	S403JI	Claim `jti` is required when `preventJtiReplay:true`	No valid `jti` claims are included in the request when preventJtiReplay is set to true in a `JWT authentication plugin`.
403	S403JU	Claim `jti` in JWT is used	The `jti` claim that is included in the request has been used when preventJtiReplay is set to true in a `JWT authentication plugin`.
403	A403JT	Invalid JWT: ${Reason}	The `JWT` that is read from the request is invalid.
400	I400JD	JWT Deserialize Failed: `${Token}`	The `JWT` that is read from the request fails to be parsed.
403	A403JK	No matching JWK, `kid:${kid}` not found	No `JWK` matches `kid` configured in the `JWT` included in the request.
403	A403JE	JWT is expired at `${Date}`	The `JWT` that is read from the request expires.
400	I400JP	Invalid JWT plugin config: ${JWT}	The `JWT authentication plugin` is incorrectly configured.

If an HTTP response message includes an unexpected response code specified by `ErrorCode` in the `X-Ca-Error-Code` header, such as `A403JT` or `I400JD`, you can visit the jwt.io website to check the token validity and format.

6. Limits

- The metadata of a JWT authentication plugin can contain a maximum of 16,380 characters.

- You can configure a maximum of 16 parameters to be forwarded. Both the `claimName` and `parameterName` parameters cannot exceed 32 characters in length. Only the following regular expression is supported: [A-Za-z0-9-_].
- `alg` can be set to RS256, RS384, RS512, ES256, ES384, ES512, HS256, HS384, or HS512 for JWKs.

2.5.4.2.8. Access control plugin

1. Overview

In an access control plugin, you can define conditions based on the request parameters or context of an API to which the plugin is bound. This allows you to determine whether to deliver an API request to a backend service. For information about how to define parameters and use conditional expressions, see Use parameters and conditional expressions.

2. Configurations

Assume that the API request path is `/{userId}/...`. JWT authentication is enabled for APIs. Two claims, userId and userType, are available in the JWT. The following plugin verification conditions apply:

- If userType is set to admin, requests in all paths are allowed.
- If userType is set to user, only the requests in the same /{userId} path are allowed.

```
---
#
# Assume that the API request path is /{userId}/... in this example.
# JWT authentication is enabled for APIs. Two claims, userId and userType, are available in the JWT.
# The following plugin verification conditions apply:
# - If userType is set to admin, requests in all paths are allowed.
# - If userType is set to user, only the requests in the same /{userId} path are allowed.
parameters:
  userId: "Token:userId"
  userType: "Token:userType"
  pathUserId: "path:userId"
#
# Rules are defined based on the preceding parameters. For each API request, the plugin checks the r
ules in sequence. If a condition in a rule is met, the result is true and the action that is specified by ifT
rue is performed. If a condition in a rule is not met, the result is false and the action that is specified b
y ifFalse is performed.
# The action ALLOW indicates that the request is allowed. The action DENY indicates that the request
is denied and an error code is returned to the client. After the ALLOW or DENY action is performed, the
plugin does not check the remaining conditions.
# If neither the ALLOW nor DENY action is performed, the plugin proceeds to check the next condition.
rules:
  - name: admin
    condition: "$userType = 'admin'"
    ifTrue: "ALLOW"
  - name: user
    condition: "$userId = $pathUserId"
    ifFalse: "DENY"
    statusCode: 403
    errorMessage: "Path not match ${userId} vs /${pathUserId}"
    responseHeaders:
      Content-Type: application/xml
    responseBody:
      <Reason>Path not match ${userId} vs /${pathUserId}</Reason>
```

3. Relevant errors

Error code	HTTP status code	Message	Description

Error code	HTTP status code	Message	Description
A403AC	403	Access Control Forbidden by ${RuleName}	The error message returned because the request is rejected by the access control plugin that is bound to the API.

4. Limits

- A maximum of 16 parameters can be specified in an access control plugin.
- Each conditional expression can contain a maximum of 512 characters.
- The metadata of an access control plugin can contain a maximum of 16,380 characters.
- A maximum of 16 `rules` can be configured in each access control plugin.

2.5.4.2.9. Error code mapping plugin

An `error code mapping plugin` is used to map backend error responses to expected error responses based on mapping rules that are defined by clients.

1. Overview

An error code mapping plugin is used to map backend error responses to expected error responses based on mapping rules that are defined by clients.

2. Quick start

The following example shows an error response that is returned by a backend service. The HTTP status code is 200, but the response body contains an error message in a JSON string.

```
HTTP 200 OK
Content-Type:application/json

{"req_msg_id":"d02afa56394f4588832bed46614e1772","result_code":"ROLE_NOT_EXISTS"}
```

- Assume that clients want to receive an HTTP status code other than 200 but do not want to modify backend configurations. For example, clients expect the following error response:

```
HTTP 404
X-Ca-Error-Message: Role Not Exists, ResultId=d02afa56394f4588832bed46614e1772
```

In this case, you can use the following sample to configure an error code mapping plugin and bind the plugin to relevant APIs:

```
---
# The parameters that are involved in mapping.
parameters:
  statusCode: "StatusCode"
  resultCode: "BodyJsonField:$.result_code"
  resultId: "BodyJsonField:$.req_msg_id"
# The mapping condition.
errorCondition: "$statusCode = 200 and $resultCode <> 'OK'"
# The parameter in an error response that is used to specify the error code and hit mapping rules.
errorCode: "resultCode"
# Mapping rules.
mappings:
  - code: "ROLE_NOT_EXISTS"
    statusCode: 404
    errorMessage: "Role Not Exists, RequestId=${resultId}"
  - code: "INVALID_PARAMETER"
    statusCode: 400
    errorMessage: "Invalid Parameter, RequestId=${resultId}"
# Optional. The default mapping rule.
defaultMapping:
  statusCode: 500
  errorMessage: "Unknown Error, ${resultCode}, RequestId=${resultId}"
```

In this example, the HTTP status code and the `result_code` parameter in an error response are used to define the mapping condition. If the HTTP status code of an error response is 200 and the value of the `result_code` parameter is not `'OK'`, the mapping starts. The `result_code` parameter is used to define the mapping rules. If the value of the result_code parameter is `ROLE_NOT_EXISTS`, the original HTTP status code is mapped to 404. If the value of the result_code parameter is `INVALID_PARAMETER`, the original HTTP status code is mapped to 400. If the value of the result_code parameter is neither of the preceding values, the original HTTP status code is mapped to 500.

3. Plugin configurations and mapping rules

3.1 Plugin configurations

You can configure an error code mapping plugin in the `JSON` or `YAML` format. The following parameters can be specified:

- `parameters`: required. The parameters that are involved in mapping. These parameters are specified as key-value pairs in the `map` format. For information about how to define parameters and write conditional expressions, see Use parameters and conditional expressions.

- `errorCondition`: required. The condition under which a response is considered an error response. If the result of the conditional expression is `true`, the mapping starts.
- `errorCode`: optional. The parameter that is used to specify the error code in an error response and hit mapping rules. The error code that is specified by this parameter is compared with the value of the `code` parameter in the mapping rules specified by `mappings`.
- `mappings`: required. The mapping rules. API Gateway reconstructs error responses based on the setting of `errorCode` or `errorCondition`. A mapping rule may contain the following parameters:
  - `code`: optional. The value of this parameter must be unique among all mapping rules. If the error code of an error response is the same as the value of the `code` parameter in the current mapping rule, the error response is mapped based on the current mapping rule.
  - `condition`: optional. The condition under which an error response needs to be mapped based on the current mapping rule. If the result of the conditional expression is `true`, the error response is mapped based on the current mapping rule.
  - `statusCode`: required. The HTTP status code that replaces the original HTTP status code of an error response if the error response needs to be mapped based on the current mapping rule.
  - `errorMessage`: optional. The error message that is returned to the client after mapping. The value of this parameter is obtained from the parameters in the original backend error response and is also stored in the `errorMessage` parameter in error logs. In the error response after mapping, this parameter is displayed as the value of the `X-Ca-Error-Message` header.
  - `responseHeaders`: optional. The response headers that are included in an error response after mapping if the current mapping rule is hit. This parameter is specified as key-value pairs in the map format.
  - `responseBody`: optional. The response body that overwrites the original response body of an error response if the error response needs to be mapped based on the current mapping rule.
- `defaultMapping`: optional. The default mapping rule. If all the rules that are defined in `mappings` are not hit by an error response, the error response is mapped based on this default mapping rule.
  - `statusCode`: required. The HTTP status code that replaces the original HTTP status code of an error response if the error response needs to be mapped based on the current mapping rule.

- `errorMessage`: optional. The error message that is returned to the client after mapping. The value of this parameter is obtained from the parameters in the original backend error response and is also stored in the `errorMessage` parameter in error logs. In the error response after mapping, this parameter is displayed as the value of the `X-Ca-Error-Message` header.
- `responseHeaders`: optional. The response headers that are included in an error response after mapping if the current mapping rule is hit. This parameter is specified as key-value pairs in the `map` format.
- `responseBody`: optional. The response body that overwrites the original response body of an error response if the error response needs to be mapped based on the current mapping rule.

Take note of the following points when you configure an error code mapping plugin:

- The parameters that are used to write conditional expressions in `mappingCondition` and `mappings[].condition` must be defined in `parameters`. Otherwise, the plugin does not work and reports an error. For information about how to define parameters and write conditional expressions, see Use parameters and conditional expressions.
- The value of the `errorCode` parameter must be the name of a parameter that is defined in `parameters`.
- When you configure a mapping rule specified by `mappings`, you must specify `code` or `condition`. When you specify `code`, the value of this parameter must be unique among all mapping rules. When you specify `condition`, you must write conditional expressions in the order that meets your requirements. This is because the order of conditions determines their priorities.
- You can specify `errorMessage` and `responseBody` in a format similar to `"${Code}: ${Message}"` and obtain the parameter values from those specified in `parameters`.
- You can specify `responseHeaders` in the `${Message}` format.
- If you do not specify `responseBody`, the body of an error response returned to the client after mapping is the same as that of the original error response.
- You can use the `responseHeaders` parameter to specify headers and their settings to replace corresponding headers in a backend error response. If you specify the value of a header as `''`, this header will be deleted after mapping. If you do not specify this parameter, the headers of the error response returned to the client after mapping are the same as those of the original error response.
- If you do not specify `defaultMapping`, the error code mapping does not take effect. The original error response from your backend service is returned to the client.

3.2 Parameters involved in mapping

As described in the following code, you must specify the parameters that are involved in mapping as key-value pairs in `parameters`. Each key is the name of a parameter. Each value is specified in the `Location:Name` format. This format indicates that the value of a parameter is obtained from a specific location in the response or system context.

```
---
# The parameters that are involved in mapping.
parameters:
 statusCode: "StatusCode"
 resultCode: "BodyJsonField:$.result_code"
 resultId: "BodyJsonField:$.req_msg_id"
```

An error code mapping plugin supports the parameters at specific locations in the following table.

Location	Included in	Description
StatusCode	Response	The HTTP status code in a backend error response, such as `200` or `400`.
ErrorCode	Response	The error code of a system error response in API Gateway.
ErrorMessage	Response	The system error message in API Gateway.
Header	Response	Use `Header:{Name}` to obtain the first value of the HTTP header that is specified by `{Name}`.
BodyJsonField	Response*	Use `BodyJsonField:{JPath}` to obtain the JSON string in the body of an API request or a backend response in `JSONPath` mode.
System	Response	Use `System:{Name}` to obtain the value of the system parameter that is specified by `{Name}`.
Token	Response	If `JWT` is used with `OAuth2` for authentication, use `Token:{Name}` to obtain the value of the parameter that is specified by `{Name}` in a token.

- `ErrorCode` and `ErrorMessage` are used to return system error codes and detailed system error information in API Gateway. For more information, see Error codes.

- `BodyJsonField` can be used to obtain the JSON string in the body of a backend response. However, if the size of the response body exceeds 15,360 bytes, the string obtained is `null`.

3.3 Working mechanism

The following operations describe how an error code mapping plugin works:

- i. Step 1: Based on the list of parameters that are defined in `parameters`, the plugin obtains the values of the parameters from a backend error response and the system context.
- i. Step 2: The plugin uses the parameters and obtained values to execute the conditional expression that is written in `errorCondition`. If the result is `true`, go to the next step. If the result is `false`, the process ends.
- i. Step 3: If `errorCode` is specified, the plugin obtains the value of `errorCode`. Then the plugin checks whether there is a mapping rule that indicates that the errorCode setting is the same as the setting of `code`. The mapping rule is specified by `mappings`.
- i. Step 4: If no mapping rule meets requirements, the plugin executes in sequence the conditional expressions that are written in `condition` in mapping rules.
- i. Step 5: If a mapping rule is hit in Step 3 or Step 4, the original error response is mapped based on the mapping rule. Otherwise, the original error response is mapped based on the default mapping rule.

3.4 Mapping of system error codes and error logs

- In API Gateway, system errors may occur in processes such as check, verification, throttling, and plugin operations. For more information, see Error codes. You can use `ErrorCode` as a location to obtain information in a system error response. For example, clients support only HTTP status code 200 and want to map HTTP status code 429 that is returned by API Gateway to HTTP status code 200.
- For a system error response, the values that are obtained from locations such as `StatusCode`, `Header`, and `BodyJsonField` are all `null`. When you define a mapping condition for an error code mapping plugin, note that for a backend error response, the value that is obtained from the `ErrorCode` location is `OK`.
- The error code of a system error response is specified by the `X-Ca-Error-Code` header and by the `errorCode` parameter in error logs. This value cannot be overwritten by an `error code mapping plugin`.
- The `statusCode` parameter in error logs records the value of the HTTP status code that is sent from API Gateway to the client. This value can be overwritten by an `error code mapping plugin`.

4. Configuration examples

4.1 Use the error codes in error responses for mapping

Mapping

```
---
# The parameters that are involved in mapping.
parameters:
  statusCode: "StatusCode"
  resultCode: "BodyJsonField:$.result_code"
  resultId: "BodyJsonField:$.req_msg_id"
# The mapping condition.
errorCondition: "$statusCode = 200 and $resultCode <> 'OK'"
# The parameter in an error response that is used to specify the error code and hit mapping rules.
errorCode: "resultCode"
# Mapping rules.
mappings:
  - code: "ROLE_NOT_EXISTS"
    statusCode: 404
    errorMessage: "Role Not Exists, RequestId=${resultId}"
  - code: "INVALID_PARAMETER"
    statusCode: 400
    errorMessage: "Invalid Parameter, RequestId=${resultId}"
# Optional. The default mapping rule.
defaultMapping:
  statusCode: 500
  errorMessage: "Unknown Error, ${resultCode}, RequestId=${resultId}"
```

5. Limits

- A maximum of 16 parameters can be specified in an error code mapping plugin.
- A single conditional expression can contain a maximum of 512 characters.
- If you use the `BodyJsonField` location to obtain the JSON string in the body of an error response, the size of the response body cannot exceed 16,380 bytes. If the size of the response body exceeds this limit, the obtained string is null.
- The metadata of an error code mapping plugin can contain a maximum of `16,380` characters.
- For an error code mapping plugin, you can configure a maximum of 20 mapping rules by using the `condition` parameter defined in `mappings` .

2.5.4.3. Bind a plugin to an API

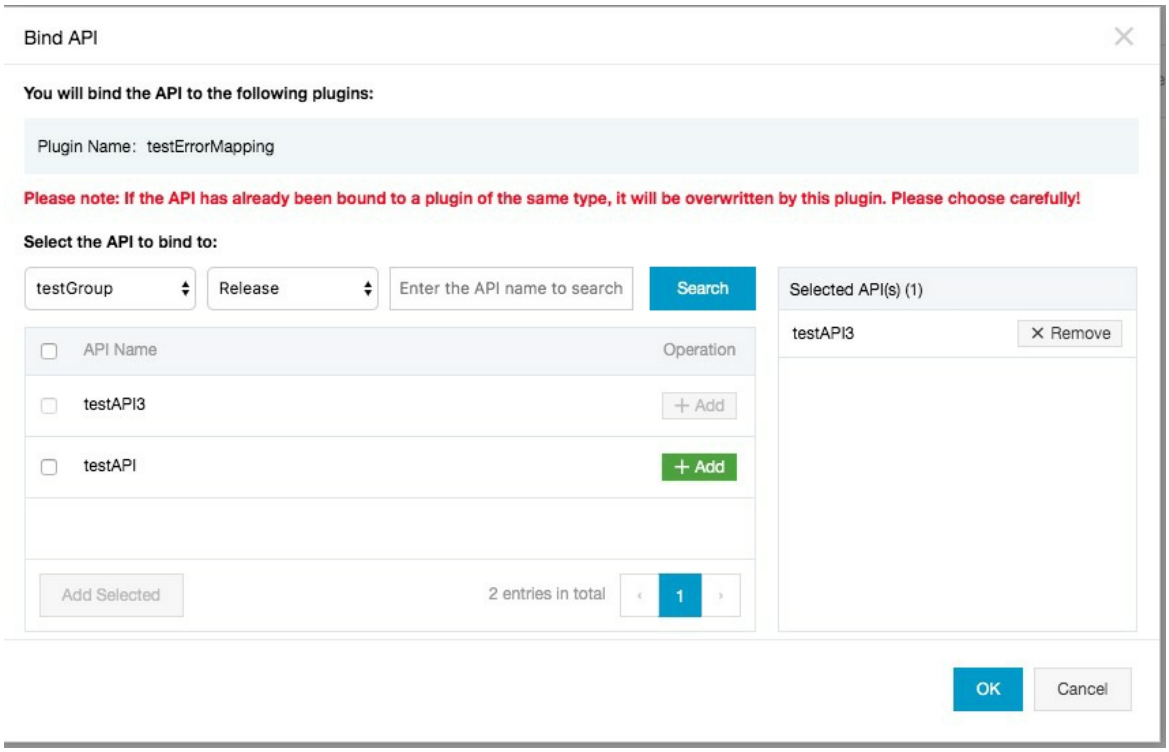
After you create a plugin, you must bind the plugin to an API for the plugin to take effect.

Context

You can bind a plugin to multiple APIs. The plugin will individually take effect on each API. For each type of plugin, you can bind only one plugin of such type to an API. If you bind two plugins of the same type to an API, the new plugin will replace the previous one and take effect.

Procedure

1. Log on to the API Gateway console
2. In the left-side navigation pane, choose Publish APIs > **Plugin**.
3. On the Plugins list page, find the target plugin and click **Bind API** in the Operation column.
4. Select the publish environment and the group of the APIs to which you want to bind a plugin.
5. To bind a plugin to one API, find the target API and click +Add in the Operation column to add the API to the right pane. To bind a plugin to multiple APIs, select the target APIs and click Add Selected in the lower-left corner to add these APIs to the right pane. Then, click **OK**.



2.5.4.4. Delete a plugin

You can delete existing plugins.

Procedure

1. Log on to the API Gateway console
2. In the left-side navigation pane, choose Publish APIs > **Plugin**.
3. On the Plugins list page, find the target plugin and click **Delete** in the Operation column.
4. In the Confirm Deletion message, click **OK**.

2.5.4.5. Unbind a plugin

You can unbind plugins from the APIs to which they are bound.

Procedure

1. Log on to the API Gateway console
2. In the left-side navigation pane, choose Publish APIs > **Plugin**.
3. On the Plugins list page, click the name of the target plugin to go to the **Create Plugin** page.
4. Click Bound API List. The bound APIs are displayed. Find the target APIs one at a time and click **Unbind** in the Operation column.
5. In the Confirm Unbind message, click **OK**.

2.6. Manage monitoring

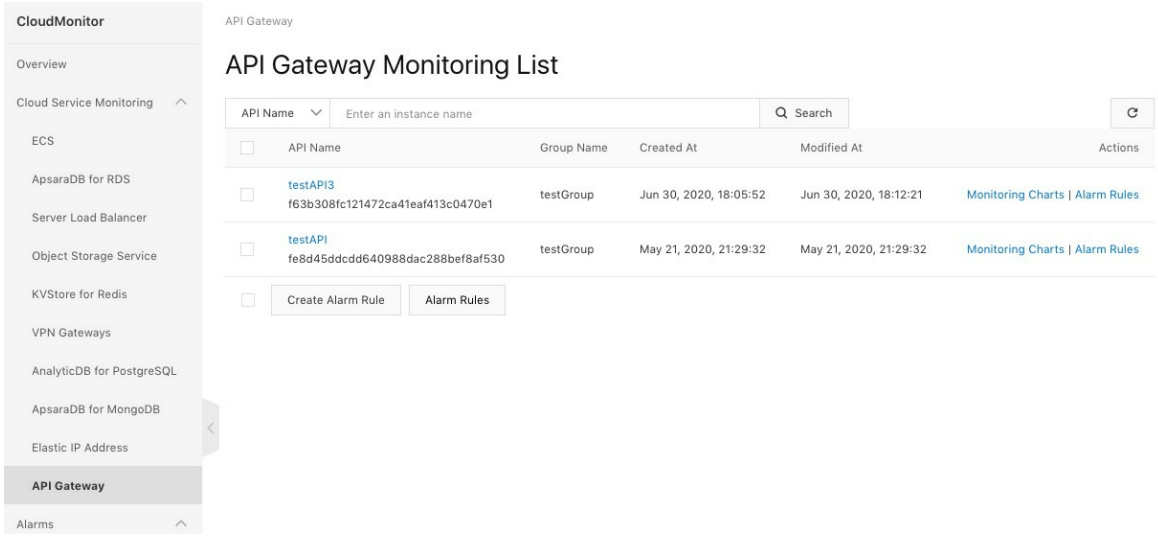
2.6.1. View monitoring information and configure alerts

API Gateway works with Cloud Monitor to provide visualized real-time monitoring and alerting features. You can use these features to obtain statistical data about your APIs in multiple dimensions, such as the number of API calls, traffic, backend response time, and error distribution. You can view data in different units of time.

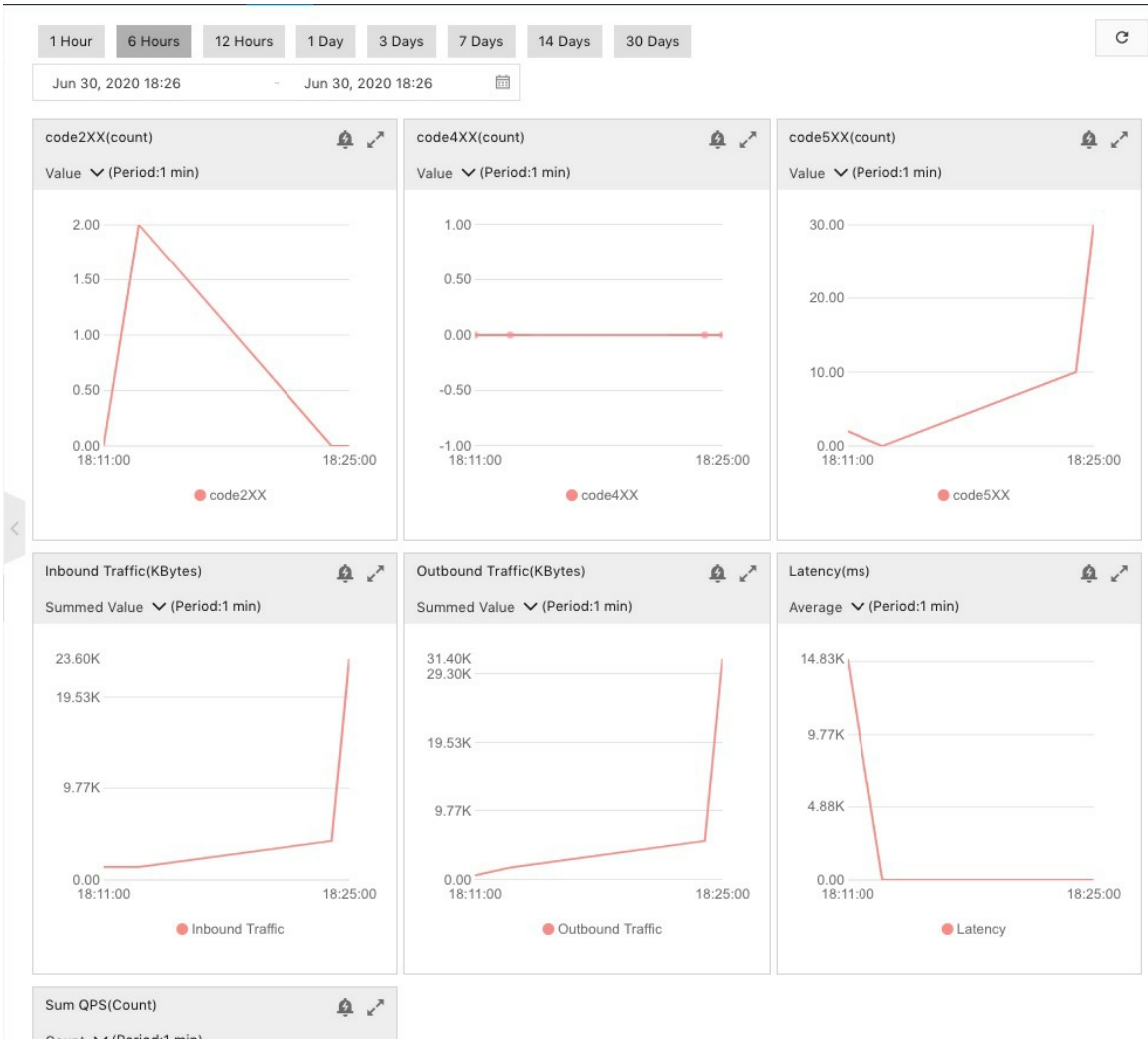
View monitoring information about API calls

Perform the following steps to view data of API calls under your Apsara Stack tenant account in the Cloud Monitor console.

1. In the top navigation bar of the ASCM console, choose **Products** > **Monitoring and O&M** > **CloudMonitor**.
2. On the page that appears, choose Cloud Service Monitoring > **API Gateway** in the left-side navigation pane.



3. On the API Gateway Monitoring List page, find the target API and click **Monitoring Charts** in the Actions column.



Descriptions of monitoring charts:

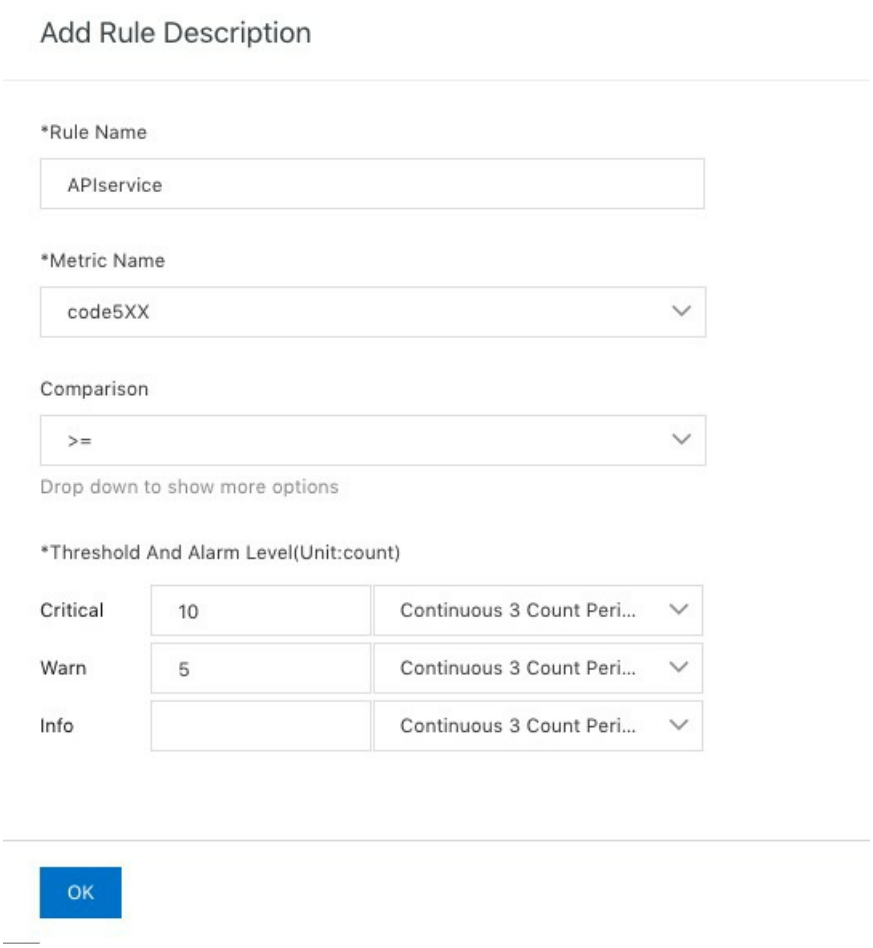
- code2XX(count): shows the number of requests with the 2XX HTTP status code returned. The 2XX HTTP status code, such as 200, indicates that the request succeeded at the backend.
- code4XX(count): shows the number of requests with the 4XX HTTP status code returned. The 4XX HTTP status code, such as 404, indicates a client error.
- code5XX(count): shows the number of requests with the 5XX HTTP status code returned. The 5XX HTTP status code, such as 500, indicates a server error.
- Inbound Traffic(KBytes): shows the size of API requests.
- Outbound Traffic(KBytes): shows the size of API responses sent.
- Latency(ms): shows the response time of your backend service. The latency in API Gateway ranges from 3 ms to 5 ms, which is excluded from the response time.
- Sum QPS(Count): shows the total number of API requests.

Configure API alert rules

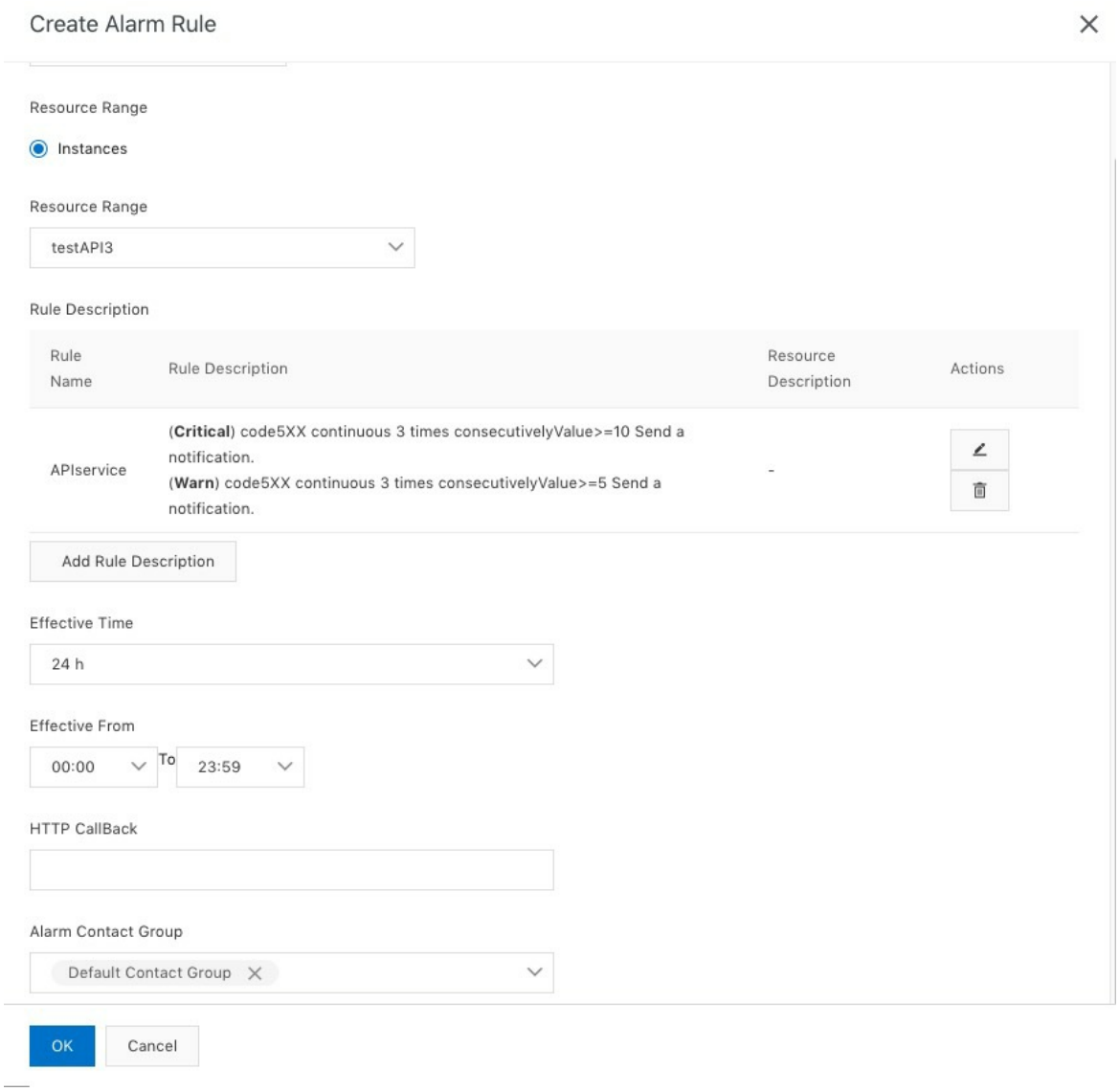
You can configure API alert rules in the Cloud Monitor console to achieve real-time API alerting.

1. On the API Gateway Monitoring List page, find the target API.
2. Click **Alarm Rules** in the Actions column.

3. On the Alarm Rules page, click **Create Alarm Rule** in the upper-right corner. In the Create Alarm Rule pane, set Product to API Gateway and Resource Range to the required API.
4. Click **Add Rule Description** and specify Rule Name, Metric Name, Comparison, and Threshold And Alarm Level. Then click **OK**.



5. Specify **Alarm Contact Group** and click **OK**.



Note To monitor the service status of APIs, we recommend that you monitor the 5XX HTTP status code.

2.6.2. View statistical information on the dashboard of API Gateway

You can view monitoring information about API calls in the Cloud Monitor console. The API Gateway console provides an overview page for statistics of API calls. You can also view statistical information about API calls on the dashboard page.

1. Log on to the API Gateway console. In the left-side navigation pane, click **Instances**. On the Instance list page, click **View Dashboard** in the upper-right corner.
2. On the page that appears, select different time granularities to view specific information about the API groups and API calls under your account. By default, this page displays a summary of API calls by domain name. The summarized information includes the number of requests, return code, and the latency to call a backend service.

3. To view the API calls of a specific domain name, click **domain List** in the left-side navigation pane. On the page that appears, click the name of the required domain name to go to the domain Detail page. You can view all API calls under this domain name on this page.

> **Note** On the dashboard page, you can view only the information about API groups and API calls under your account. Even user root cannot view data on the dashboard page.

2.7. Advanced usage

2.7.1. Business parameters of custom logs

API Gateway provides the Hack mode, which allows you to record the request and response parameters of API calls in logs.

Procedure

1. Log on to the Apsara Stack console.
2. In the left-side navigation pane, choose **Compute, Storage & Networking** > **API Gateway**.
3. Click the **Groups** tab.
4. Click the management icon in the Actions column corresponding to a group and choose **Change Group** from the shortcut menu.
5. In the **Description** field, add the following content:

```
logConf:reqBody=1024,reqHeaders,reqQuery,respHeaders,respBody=1024
```

> **Note**
> - The content must be in a separate line. Otherwise, the configuration fails.
> - You can also modify the content that follows logConf:reqBody=1024. For example, you can remove respHeaders to omit the response header and its content in the logs.
> - reqBody = 1024 indicates that the log only records up to 1,024 characters from the request body. Extra characters are truncated.

6. Log on to the Log Service console and check whether the description change has been recorded.

apiUid:73c226d7169148489a71c5d33813b3a5
appId:157555089414951
appName:integration_root
clientIp:172.31.224.26
clientNonce:c3de6af9-17be-49a4-b516-12d7d7535101
domain:ad41bfca1175433389bc6fa1669e2472.apigateway.inter.env11b.shuguang.com
errorCode:
errorMessage:
exception:
httpMethod:GET
initialRequestId:
instanceId:
path:/testIpControl
providerAliUid:1663875445751523
region:cn-qingdao-env11-d01
requestBody:
requestHandleTime:2020-01-03T07:52:14Z
requestHeaders:{"X-Ca-Key":"157555089415157","X-Ca-Stage":"PRE","X-Forwarded-Proto":"http","Host":"ad41bfca1175433389bc6fa1669e2472.apigateway.inter.env11b.shuguang.com","Date":"Fri, 03 Jan 2020 07:52:14 GMT","X-Ca-Signature-Headers":"X-Ca-Key,X-Ca-Nonce,X-Ca-Timestamp","X-Ca-Nonce":"c3de6af9-17be-49a4-b516-12d7d7535101","X-Ca-Timestamp":"1578037934480","X-Ca-Signature-Method":"HmacSHA256","X-Forwarded-For":"172.31.224.26","X-Ca-Signature":"CiBQR3ezw5GgGZQGzZT8l5rn3V4C2TncCKUVnn0se5c=","eagleeye-rpcid":"0.1","X-Real-IP":"172.31.224.26","accept-encoding":"gzip","user-agent":"unirest-java/1.3.11"}
requestId:F88B9A2C-1191-4BC7-98AE-D5304BFC88DA
requestProtocol:http
requestQueryString:
requestSize:554
responseBody:{
"Body":"",
"Headers":{
"date":"Fri, 03 Jan 2020 07:52:14 GMT",
"host":"172.31.224.26:8080",
"x-ca-request-id":"F88B9A2C-1191-4BC7-98AE-D5304BFC88DA",
"connection":"Keep-Alive",
"accept-encoding":"gzip",
"x-ca-stage":"PRE",
"user-agent":"unirest-java/1.3.11",
"via":"73c226d7169148489a71c5d33813b3a5"
},
"Method":"GET",
"Params":{},
"Path":"/web/cloudapi",
"RequestURL":"http://172.31.224.26:8080/web/cloudapi"
}
responseHeaders:{"Transfer-Encoding":"chunked","Date":"Fri, 03 Jan 2020 07:52:14 GMT","Content-Type":"application/json"}
responseSize:220
serviceLatency:1
statusCode:200

2.7.2. Configure Log Service logs for API Gateway

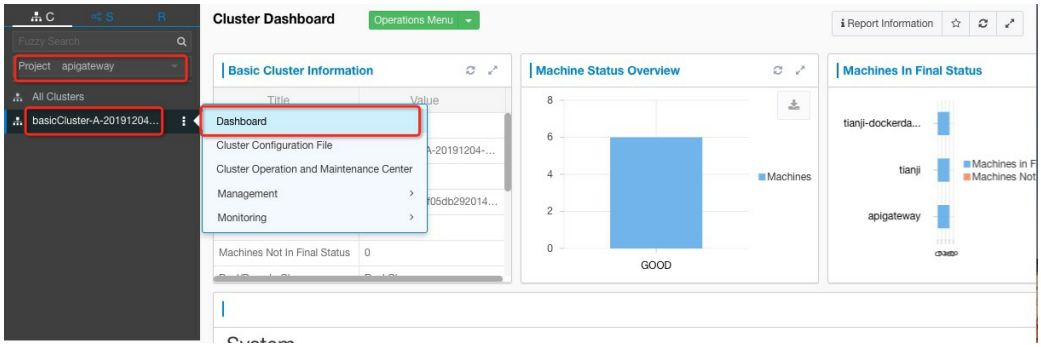
2.7.2.1. Initialize the default Log Service configuration of API Gateway

By default, API call logs in API Gateway are synchronized to Log Service in Apsara Infrastructure Management Framework. However, your account can be activated only after you log on to the Log Service console from Apsara Infrastructure Management Framework.

Procedure

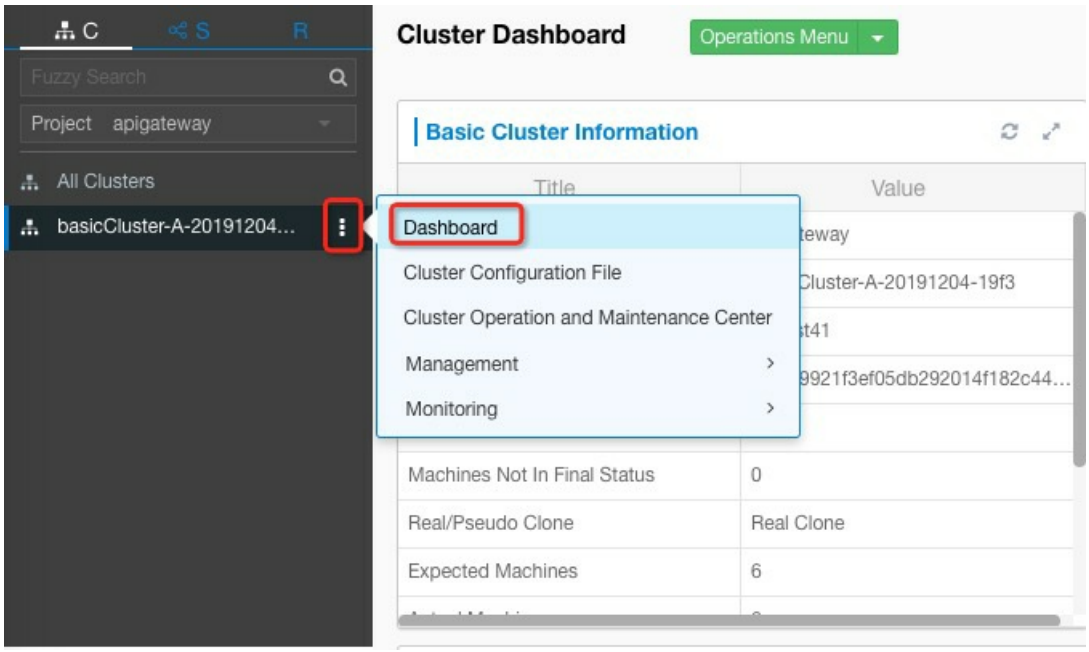
1. Log on to the Apsara Infrastructure Management Framework console.
2. In the left-side navigation pane, click Tools, Operation Tools, and then Machine Tools. On the Machine Tools page, click Go. On the page that appears, click the **C** tab in the left-side navigation pane. Then select *apigateway* from the **Project** drop-down list.

   Cluster O&M page

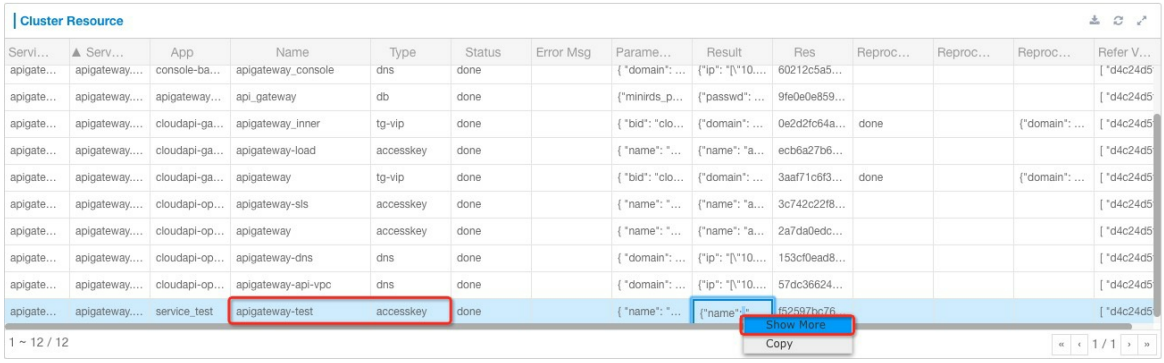


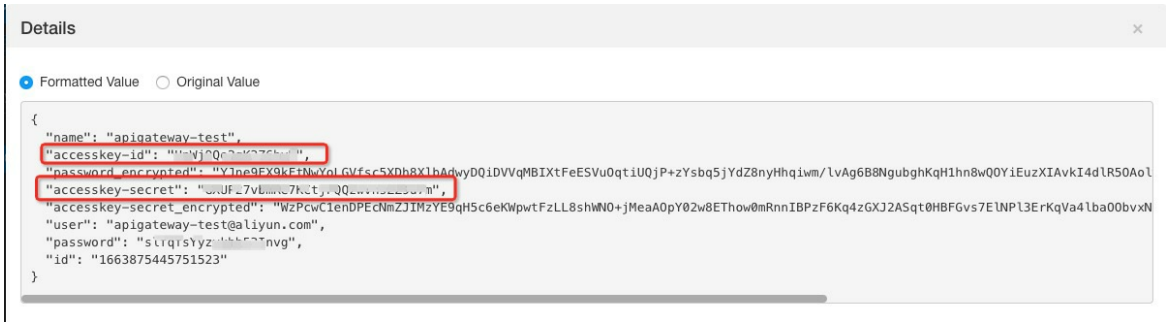
3. Move the pointer over the ⋮ icon next to one of the filtered clusters, and select

Dashboard from the shortcut menu.



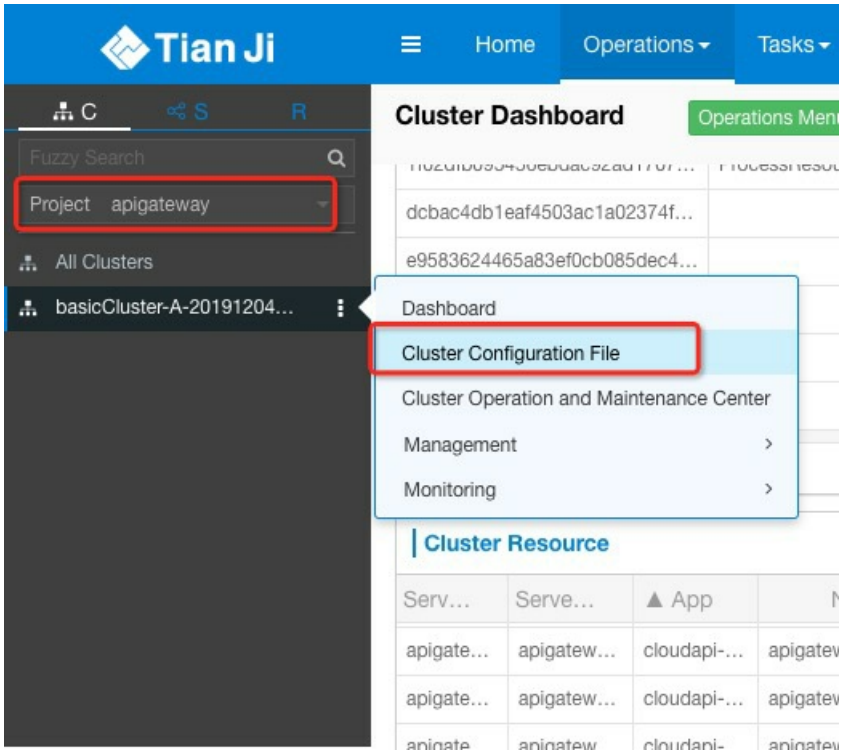
4. In the Cluster Resource section of the page that appears, find the service with Name set to apigateway-sls and Type set to accesskey. Right-click the value in the Result column and select **Show More** from the shortcut menu to view the values of accesskey-id and accesskey-secret.



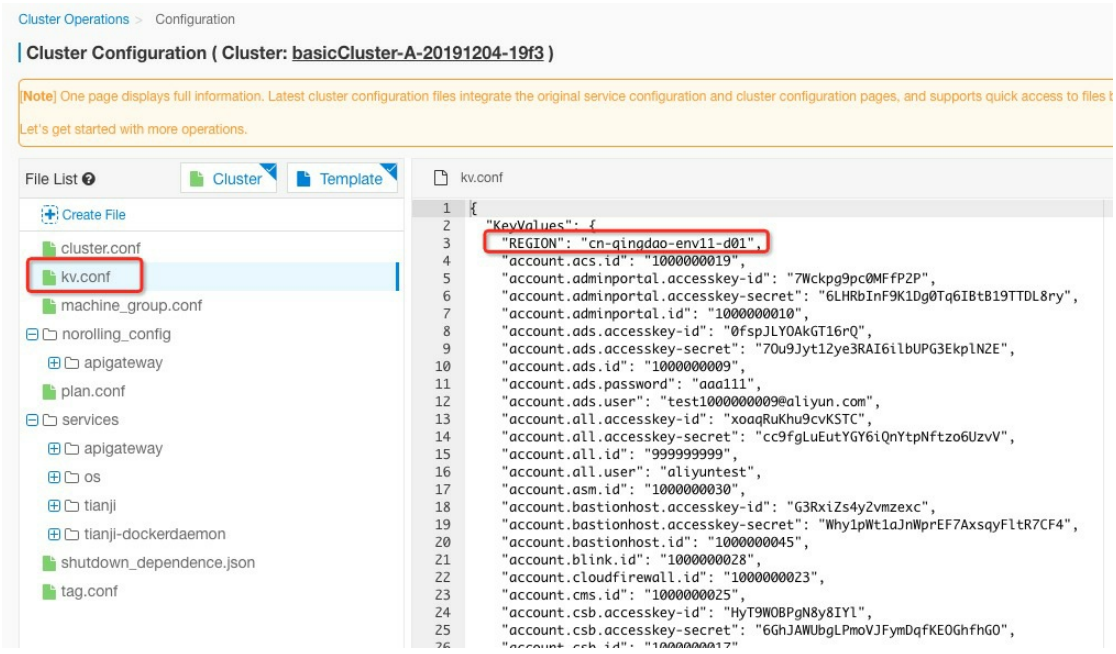


5. Use the obtained accesskey-id and accesskey-secret values to log on to the Log Service console from Apsara Infrastructure Management Framework. The URL for the Log Service console in Apsara Infrastructure Management Framework is http://portal.${region}.sls.${internet-domain}. You can obtain the values of region and internet-domain from the kv.conf file in the Apsara Infrastructure Management Framework console.

i. Move the pointer over the More icon next to the apigateway cluster and select **Cluster Configuration File** from the shortcut menu.



ii. Click the **kv.conf** file to view the values of region and internet-domain.



Note After you log on to the Log Service console, Log Service is automatically configured for API Gateway. This operation takes several minutes.

2.7.2.2. Configure API Gateway to deliver logs to your Log Service project

If you do not want to use Log Service and a fixed account in Apsara Infrastructure Management Framework, you can create a Log Service project in the Apsara Stack console through Log Service. Then, configure API Gateway to deliver logs to your Log Service project.

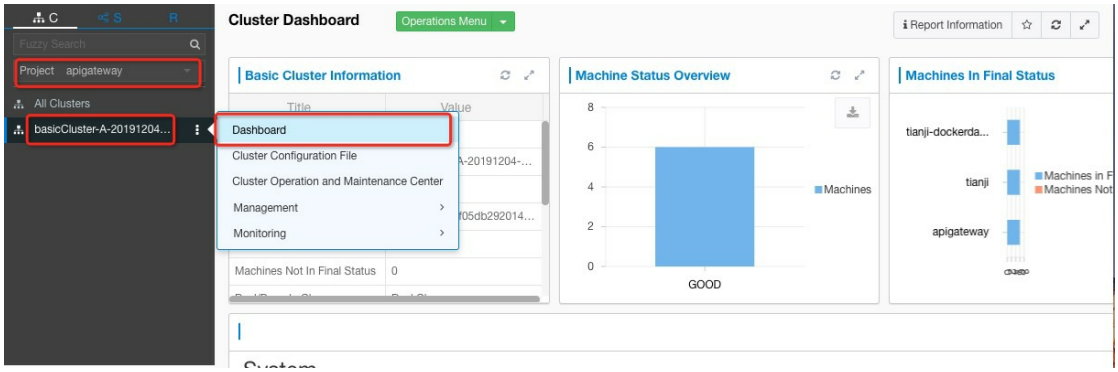
Context

You must first modify the configurations on machines where API Gateway resides. Then, create a Log Service project and configure the Logstores and machine groups. The procedure is as follows:

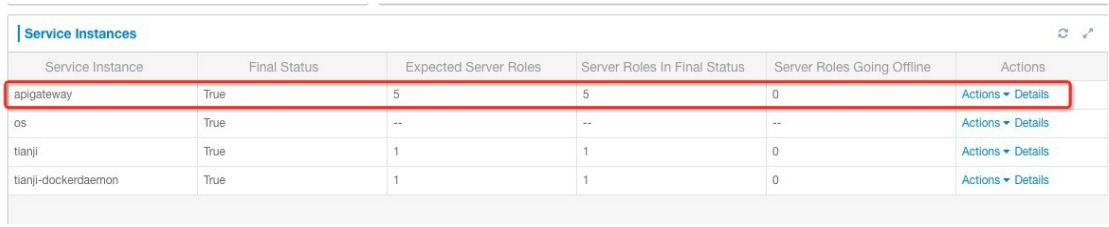
1. Modify the configurations on machines where API Gateway resides

You can use the Hack method to manually modify the Logtail configurations of API Gateway and change the log delivery method.

1. First, find the machines where API Gateway resides.
   i. Log on to the Apsara Infrastructure Management Framework console. In the left-side navigation pane, click the C tab and select apigateway from the Project drop-down list. Place the pointer over the More icon next to one of the filtered clusters and choose Dashboard from the shortcut menu.

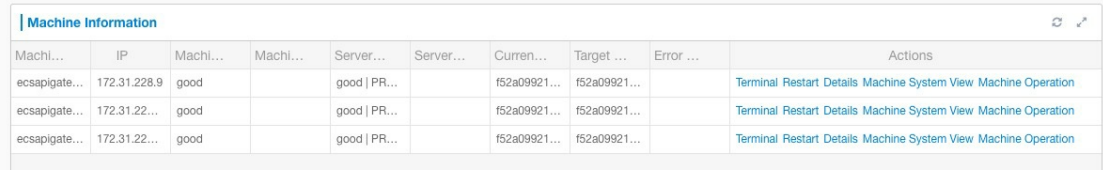


   ii. On the dashboard page, go to **Service Instance List**, and find apigateway in the Service Instance column.



iii. Click **Details** in the Actions column corresponding to the apigateway service instance. In the **Server Role List** section, find ApigatewayLite# in the Server Role column.

iv. Click **Details** in the Actions column corresponding to the ApigatewayLite# server role. Then, find the **Machine Information** section.



> **Note** Perform steps 2, 3, and 4 on each machine.

2. Log on to the gateway terminal and go to the /alidata/settings directory to create the init_ilogtail.sh script. Note that the name cannot be changed. The script content is as follows:

```
#! /bin/bash
wget data.cn-qingdao-env4b-d01.sls-pub.env4b.shuguang.com/logtail.sh
sudo sh logtail.sh install
```

> **Note** In the preceding content, data.cn-qingdao-env4b-d01.sls-pub.env4b.shuguang.com needs to be replaced with the value of the sls_data.endpoint variable in the sls-backend-server service. You can find the value from the Registration Vars of Services report in Apsara Infrastructure Management Framework.

3. Add execution permissions to the init_ilogtail.sh file.

```
sudo chmod +x init_ilogtail.sh
```

4. Access the gateway container (the container in the server role starting with ApigatewayLite) from Apsara Infrastructure Management Framework, and then run the following commands:

```
cd /home/admin/bin
sudo sh config_ilogtail.sh MANUAL
```

If the following information is displayed, the installation is successful.

```
install logtail success
```

2. Configure logs in the Log Service console

1. Log on to the Apsara Stack console.
2. In the left-side navigation pane, choose **Compute, Storage & Networking > Log Service**.
3. Click **Create Project** and set the parameters.
4. Click **Go to Console**. Set **Region** and **Department**, and then click **SLS**. You will be directed to the Log Service console.
5. In the Log Service console, click the project you created.
6. On the Logstores page, create a Logstore.
7. In the left-side navigation pane, choose LogHub - Collect > Logtail Config. The Logtail Configurations page appears.
8. Click **Create**. The configuration page appears. select **text**
9. Configure the data source as follows, and then click **Next**.

Parameter	Value
Configuration Name	gateway_request_log
Log Path	/alidata/www/logs/java/cloudapi-gateway/logs/request_user.log
Mode	Delimiter Mode
Log Sample	2019-08-15 14:25:05|CC7C526B-C915-44F1-93A2-F44DBBE35177|b2909c9fa66146f19baf2bd8f4709ab8|integration_root|e4032f87ace14cc6965cf5352a9637a6|RELEASE|0619f7763b004fc3a16a41dd712ce8d7|biz1_anonymous|10.4.21.241|||b2909c9fa66146f19baf2bd8f4709ab8.apigateway.env4b.shuguang.com|POST|/biz1/anonymous|403|A403JT:Invalid JWT: deserialized JWT failed||1453964555641148|cn-qingdao-env4b-d01|2019-08-15T06:25:05Z|614|0|0|A403JT|http|||cf802da1-54d0-49b8-b77c-2b20b172d90a||{"X-JWT-Token":"bad jwt token"}|a=%21111||||||
Delimiter	Vertical Line

10. Click **Next** to go to the **Apply to Machine Group** page. If no machine groups are available, create a machine group. Enter a machine group name and enter all IP addresses of Docker

containers.

11. Select the newly created machine group and click **Apply to Machine Group**. Use the default settings until the Logstore is created.
12. After the configuration is complete, check whether the configuration takes effect. In the left-side navigation pane of the Log Service project, choose **LogHub - Collect** > **Logtail Machine**. On the Machine Groups page, find the created machine group and click **Status** in the Actions column to check whether the Logtail heartbeat of the machine group is normal.

> **Note** :
> - The heartbeat check takes several minutes.
> - If the heartbeat is normal, you can query logs.

2.7.3. Cross-user VPC authorization

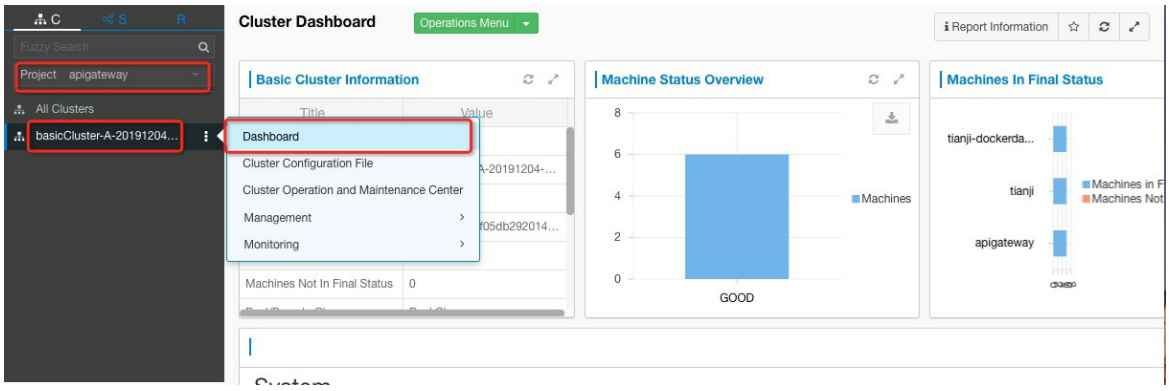
If your backend service resides in a VPC, you must configure the backend service address through VPC authorization. By default, a VPC owner must be the same user as an API owner. Starting from Apsara Stack V3.8.1, API Gateway provides two internal APIs that allow VPC owners to authorize their VPCs to other users.

2.7.3.1. User authorization across VPCs

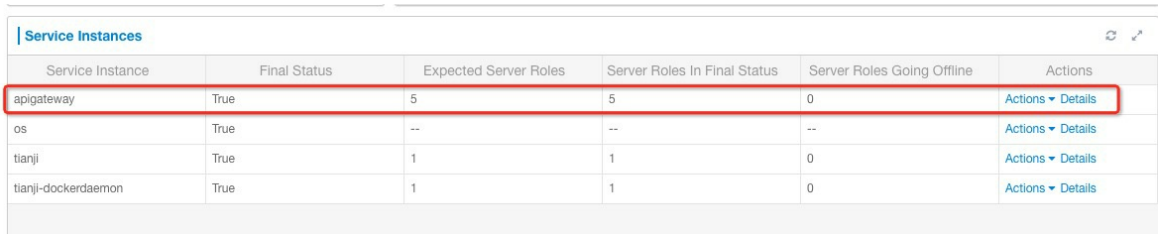
APIs can be used across multiple VPCs. For security reasons, VPC owners must call APIs to explicitly authorize access to VPCs before API providers can use the VPCs. The OpenAPI component of API Gateway has a built-in Aliyun CLI tool. You can use this tool to authorize access to VPCs. The following steps describe how to call the API:

Procedure

1. Log on to the Apsara Infrastructure Management Framework console. In the left-side navigation pane, click the C tab and select apigateway from the Project drop-down list. Place the pointer over the More icon next to one of the filtered clusters and choose Dashboard from the shortcut menu.

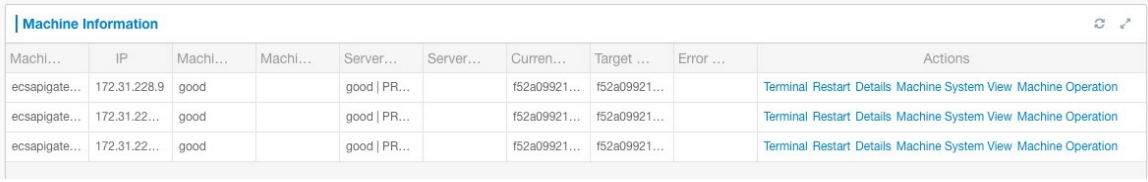


2. On the dashboard page, go to **Service Instance List**, and find apigateway in the Service Instance column.



3. Click **Details** in the Actions column corresponding to the apigateway service instance. In the **Server Role List** section, find ApigatewayLite# in the Server Role column.

4. Click **Details** in the Actions column corresponding to the ApigatewayLite# server role. Then, find the **Machine Information** section.



5. Click Terminal in the Actions column corresponding to a machine in the server role to access the container.
6. Configure the AccessKey pair used to call the CLI tool. Run the following commands:

```
aliyun configure -- profile vpctest //Add the AccessKey pair configuration. vpctest is the profile n
ame, which can be customized. After you press Enter, configure AccessKeyId and AccessSecret as
prompted.
aliyun configure list //View the config configuration to check whether the preceding profile has b
een added.
```

7. Run the following command to perform authorization:

```
aliyun cloudapi AuthorizeVpc --VpcId vpc-tb5mfcwx3s4zqctzw**** --TargetUserId 147546214349****
--endpoint apigateway.cn-qingdao-env11-d01.inter.env11b.shuguang.com --force
--profile vpctest
```

> **Note**
>
> - Log on to the Apsara Infrastructure Management Framework console. In the top navigation bar, choose **Reports** > **System Reports**. On the System Reports page, click Registration Vars of Services. On the Registration Vars of Services report, right-click the value in the Service Registration column corresponding to the apigateway service and choose Show More from the shortcut menu. The value of the apigateway.openapi.endpoint variable must be used as the endpoint in the preceding command. The profile value also needs to be replaced based on actual needs.
> - This command authorizes the user whose ID is 1475462143497330 to use the VPC with the ID vpc-tb5mfcwx3s4zqctzw19w2. You can replace the parameter values as needed.

Success Operation Sample

```
[root@docker010011102023 /]
#aliyun cloudapi AuthorizeVpc --VpcId vpc-tb5mfcwx3s4zqctzw19w2 --TargetUserId 1475462143497330 --endpoint apigateway.cn-qingdao-env
11-d01.inter.env11b.shuguang.com --force --profile vpctest
{"RequestId":"8E64BC51-60D7-4D85-B5F0-3E3F12C4B6D2"}
```

2.7.3.2. Configure APIs

After an app is authorized to call an API, the API owner must configure the API and define the API backend service because you cannot select the VPC ID of another user in the Apsara Stack Cloud Management (ASCM) console.

Context

When you configure an API, take note of the following points:

- Backend Service Type cannot be set to VPC.
- The backend service address must be in the http(s)://{Backend service IP address}.{vpcId}.gateway.vpc:{port} format. The content in {} can be substituted as required. Example:

  ```
  http://192.168.XX.XX.vpc-tb5mfcwx3s4zqctzw****.gateway.vpc:8080
  ```

Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose Publish APIs > **APIs**.
3. On the API List page, find the target API and perform the following operations:
   - Click the name of the API to go to the **API Definition** page. You can view information of the API.
   - Click **Edit** in the upper-right corner, modify configurations as required, and then click **Save**.

     The procedure of modifying an API is similar to that of creating an API. For more information about how to create an API, see Create an API. If you want to cancel the modifications before they are submitted, click Cancel Edit in the upper-right corner of the edit page.

2.7.4. Call an API over HTTPS

Context

API Gateway locates a unique API group by domain name, and locates a unique API in the API group by using Path and HTTPMethod. API Gateway assigns a second-level domain for each API group. You can use the domain name to call APIs that belong to the API group. The second-level domain supports only access over HTTP. You can also use a custom domain to call APIs. This topic describes how to call APIs by using a second-level domain or by using a custom domain.

Use a second-level domain to call APIs over HTTPS

To use a second-level domain to call APIs over HTTPS, you must perform the following steps to configure a wildcard domain name certificate in Apsara Stack. You must prepare the certificate yourself.

1. Prepare configuration files for a second-level domain.Modify configurations in the following code: Replace *.wildcard.com with your wildcard domain name *.apigateway.${internet-domain}. You can obtain the value of the ${internet-domain} variable in the kv.conf configuration file for Apsara Infrastructure Management Framework. For example, if the domain name that you use to provide external services is abc.alibaba.com, replace *.wildcard.com with *.alibaba.com. Save the modified code to a wildcard.conf file. Set the name of the public key file in the certificate to wildcard.crt. Set the name of the private key file in the certificate to wildcard.key.

```
server {
    listen          443 http2 ssl;
    server_name     *.wildcard.com;

    limit_req       zone=perserver_req burst=100;

    ssl_protocols   TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers     ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA:AES128-GCM-SHA256:
AES128-SHA256:AES128-SHA:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES256-SHA:AES256-GCM-SHA384:AES256-SHA256:AES256-SHA:ECDHE-RSA-AES128-SHA256:!
aNULL:! eNULL:! RC4:! EXPORT:! DES:! 3DES:! MD5:! DSS:! PKS;

    ssl_certificate     /home/admin/cai/certs/wildcard.crt;
    ssl_certificate_key /home/admin/cai/certs/wildcard.key;

    ssl_session_cache   shared:SSL:10m;
    ssl_session_timeout 10m;

    location / {
        proxy_pass http://gateway_upstream;
    }
}
```

2. Place the files at specified locations.Copy and paste the wildcard.conf file to the /alidata/sites/conf directory of the host for apigateway.ApiGatewayLite#. Copy and paste the wildcard.crt and wildcard.key files to the /alidata/sites/certs directory of the host for apigateway.ApiGatewayLite#.
3. Activate the certificate.Run `/home/admin/cai/bin/nginxctl reload` in the apigateway.ApiGatewayLite# SR container.

> **Note** You must perform Step 2 and Step 3 on all machines under the apigateway.ApiGatewayLite# SR container.

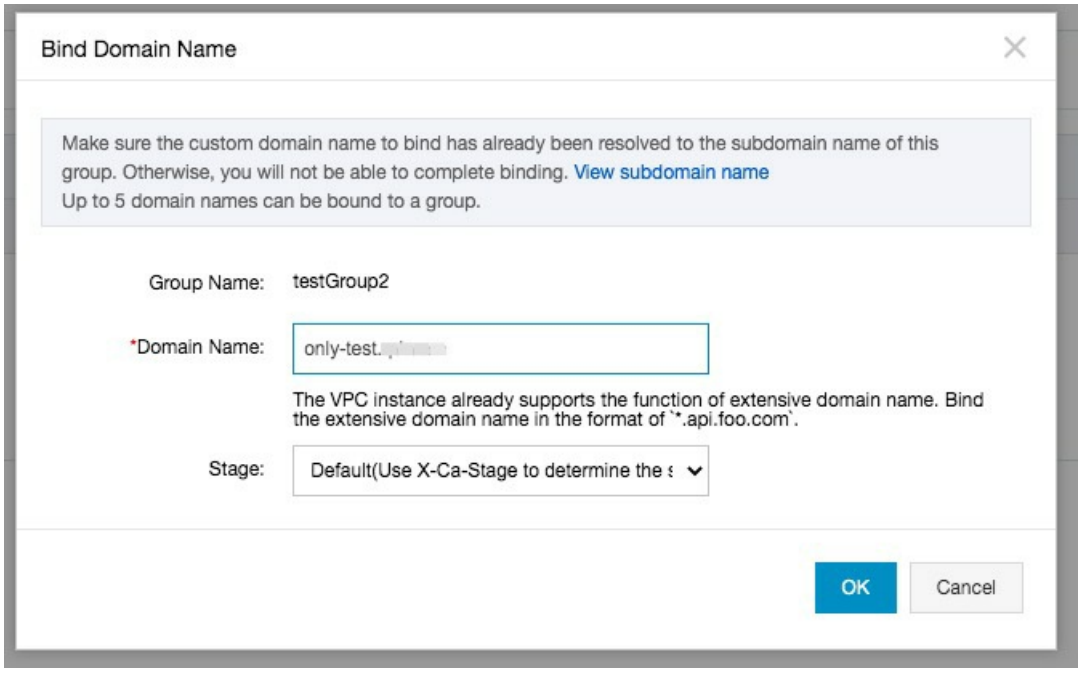
Use a custom domain to call APIs over HTTPS

Perform the following steps to use a custom domain to call APIs over HTTPS:

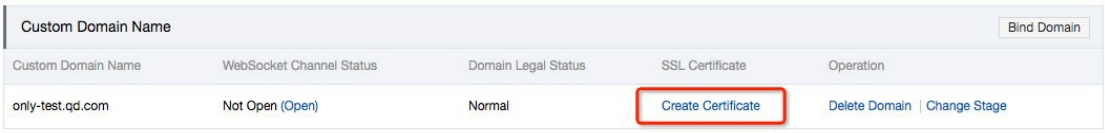
1. Bind a custom domain to a specific API group.
    i. Log on to the API Gateway console. In the left-side navigation pane, choose Publish APIs > API Groups. On the Group List page, click the name of the target group to go to the Group Details page.

ii. In the Custom Domain Name section, click **Bind Domain**.

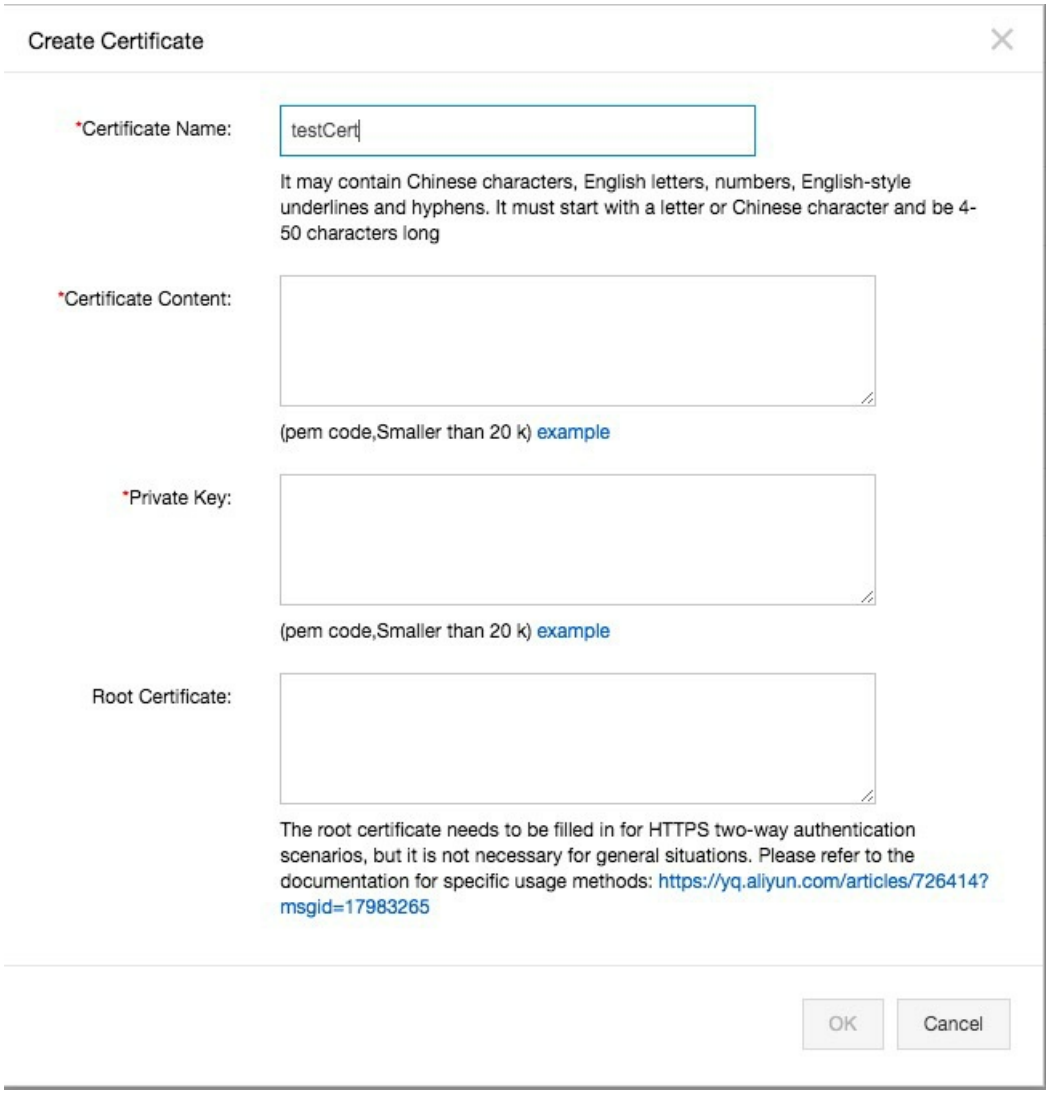
iii. In the Bind Domain Name dialog box, specify Domain Name and click **OK**.



2. Resolve the domain name.To access API Gateway by using this domain name, you must resolve the domain name in Apsara Stack to the virtual IP address of API Gateway.Run the ping [Second-level domain] command to obtain the virtual IP address of API Gateway.
3. Upload a domain name certificate.After a domain name is bound to an API group, you can use the domain name to call all the APIs that belong to this API group over HTTP. If you want to call APIs over HTTPS, you must upload an SSL certificate for the domain name. You must prepare the certificate yourself and upload the certificate to API Gateway.
   i. In the **Custom Domain Name** section of the **Group Details** page, find the target domain name and click **Create Certificate** in the SSL Certificate column.



ii. In the Create Certificate dialog box, specify **Certificate Name**, **Certificate Content**, and **Private Key**. Then click **OK**.



Note If the certificate is a self-signed certificate, ignore certificate verification when you call APIs.

2.8. FAQ

2.8.1. How do I obtain error information?

API Gateway returns a response to the client after it receives a request.

You must check the response headers that start with X-Ca. Take note of the following points:

```
// The unique ID of the request. When API Gateway receives a request, it generates a request ID and r
eturns the request ID to the client in the X-Ca-Request-Id header. We recommend that you record the r
equest ID in both the client and your backend service for troubleshooting and tracking.
X-Ca-Request-Id: 7AD052CB-EE8B-4DFD-BBAF-EFB340E0A5AF

// The error message returned by API Gateway. If a request fails, API Gateway returns the error mess
age to the client in the X-Ca-Error-Message header.
X-Ca-Error-Message: Invalid Parameter Required `field1`

// The error code of a system error in API Gateway. If a request is blocked by API Gateway due to an er
ror, API Gateway returns the corresponding error code in the X-Ca-Error-Code header. Instances of the
classic network type do not have this header.
X-Ca-Error-Code: I400MP
```

The `X-Ca-Error-Code` and `X-Ca-Error-Message` `headers` help you identify the error cause. The `X-Ca-Request-Id` header helps you query request logs in Log Service. You can also provide the request ID included in the X-Ca-Request-Id header for technical support personnel to check log information and resolve issues.

For more information about `X-Ca-Error-Code`, see Error codes.

2.8.2. Error codes

- If the client receives a response in which the `X-Ca-Error-Code` header is not empty, the header is returned by API Gateway. An error code is six characters in length. For more information, see the following table. `X-Ca-Error-Message` indicates detailed information about an error message.
- If the `X-Ca-Error-Code` header is empty, the HTTP error code is generated by your backend service. API Gateway transparently transmits the error message from your backend service.

Error code	HTTP status code	Error message	Description
I400HD	400	Invalid Header `${HeaderName}` ${Reason}	The error message returned because the HTTP request header is invalid.
I400MH	400	Header `${HeaderName}` is Required	The error message returned because the HTTP request header is missing.
I400BD	400	Invalid Body: ${Reason}	The error message returned because the HTTP request body is invalid.

I400PA	400	Invalid Request Path `${Reason}`	The error message returned because the HTTP request path is invalid.
I405UM	405	Unsupported Method `${Reason}`	The error message returned because the HTTP request method is not supported.
I400RU	400	Invalid Request Uri `${Reason}`	The error message returned because the HTTP request URL is invalid.
I403PT	403	Invalid protocol ${Protocol} unsupported	The error message returned because a protocol that is not supported in API configurations is used. Check the API configurations.
I413RL	413	Request body too Large	The error message returned because the request body is too large.
I413UL	413	Request URL too Large	The error message returned because the request URL is too long.
I400CT	400	Invalid Content-Type: `${Reason}`	The error message returned because the Content-Type setting is invalid.
I404DO	404	Invalid Domain `${DomainName}`	The error message returned because the domain name is unknown.
I410GG	410	Group's instance invalid	The error message returned because an invalid instance is requested. The group may not belong to the current instance.
I400SG	400	Invalid Stage	The error message returned because an unknown environment is requested.

I404NF	404	API not found ${Reason}	The error message returned because the corresponding API is not found based on the Path and Method settings of the request.
X400PM	400	Invalid plugin meta ${PluginName} ${Reason}	The error message returned because the metadata of the plugin is invalid. Submit a ticket to contact customer service.
X500ED	500	Expired api definition	The error message returned because the specified API metadata definition is invalid. Submit a ticket to contact customer service.
X500AM	500	Invalid Api Meta, try deploy again or contact us via ticket	The error message returned because the specified API metadata definition is invalid. Submit a ticket to contact customer service.
X403DG	403	Bad Domain or Group: ${Reason}	The error message returned because the grouped data is invalid. Submit a ticket to contact customer service.
B451DO	451	Unavailable Domain for Legal Reasons	The error message returned because the domain name does not comply with the requirements of relevant laws and regulations.
B451GO	451	Unavailable Group for Legal Reasons	The error message returned because the group does not comply with the requirements of relevant laws and regulations.

B403OD	403	Provider Account Overdue	The error message returned because the API provider has overdue payments.
A400AC	400	Invalid AppCode ${Reason}	The error message returned because the corresponding AppCode is not found when you perform an authorization in AppCode mode.
A400IK	400	Invalid AppKey	The error message returned because the corresponding AppKey setting is not found when you perform an authorization by using a key-secret pair.
A403IS	403	Invalid Signature, Server StringToSign:`${StringToSign}`	The error message returned because the signature is invalid. For more information, see Request signatures.
A403EP	403	App authorization expired	The error message returned because the authorization expired.
A403PR	403	Plugin Authorization Needed	The error message returned because plugin authorization is not performed.
A400MA	400	Need authorization, `X-Ca-Key` or `Authorization: APPCODE ...` is required	The error message returned because authorization is not performed in AppCode mode or by using a key-secret pair.
I400I5	400	Invalid Content-MD5 ${Reason}	The error message returned because Content-MD5 is invalid.

I400NC	400	X-Ca-Nonce is required	The error message returned because the X-Ca-Nonce header is not provided after you select Force Nonce Check (Anti Replay by X-Ca-Nonce).
S403NU	403	Nonce Used	The error message returned because a replay attack is detected. The X-Ca-Nonce header in the request is repeated.
S403TE	403	X-Ca-Timestamp is expired	The error message returned because the timestamp specified by the X-Ca-Timestamp header expired.
I400MP	400	Parameter `${ParameterName}` is required	The error message returned because the required parameter is not specified in the API configuration.
I400IP	400	Invalid parameter `${ParameterName}` ${Reason}	The error message returned because the value of the parameter that is specified in the API configuration is invalid.
I400JR	400	JWT required	The error message returned because no JWT-related parameters are found.
S403JI	403	Claim `jti` is required when `preventJtiReplay:true`	The error message returned because no valid jti claim is included in the request when preventJtiReplay is set to true in a JWT authentication plugin.

S403JU	403	Claim `jti` in JWT is used	The error message returned because the jti claim that is included in the request is used when preventJtiReplay is set to true in a JWT authentication plugin.
I400JD	400	JWT Deserialize Failed: `${Token}`	The error message returned because the JWT that is read from the request failed to be parsed.
A403JT	403	Invalid JWT: ${Reason}	The error message returned because the JWT that is included in the request is invalid.
A403JK	403	No matching JWK, `${kid}` not found	The error message returned because no JWK matches kid configured in the JWT included in the request.
A403JE	403	JWT is expired at `${Date}`	The error message returned because the JWT that is read from the request expired.
I400JP	400	Invalid JWT plugin config: ${JWT}	The error message returned because the JWT authentication plugin is incorrectly configured.
A403OL	403	OAuth2 Login failed: ${Reason}	
A403OU	403	OAuth2 Get User Info failed: ${Reason}	
A401OT	401	Invalid OAuth2 Access Token	
A401OM	401	OAuth2 Access Token is required	

T429ID	429	Throttled by INNER DOMAIN Flow Control, ${Domain} is a test domain, only 1000 requests per day	The error message returned because the number of requests initiated has exceeded the upper limit allowed for a default second-level domain. To increase the quota, use your own domain name.
T429IN	429	Throttled by INSTANCE Flow Control	The error message returned because throttling is performed for the current instance.
T429GR	429	Throttled by GROUP Flow Control	The error message returned because throttling is performed for the current group.
T429PA	429	Throttled by API Flow Control	The error message returned because the default API-level throttling policy defined in the throttling plugin is used.
T429PR	429	Throttled by PLUGIN Flow Control	The error message returned because the special throttling policy defined in the throttling plugin is used.
T429UP	429	Throttled by Usage Plan Flow Control	The error message returned because throttling is performed for the usage plan.
T429SR	429	Throttled by SERVER Flow Control	
T429MR	429	Too Many Requests, throttle by `${Description}`	

A403IP	403	Access denied by IP Control Policy	The error message returned because access is denied by the IP address-based access control plugin.
A403IN	403	Access from internet is disabled ${Reason}	The error message returned because you are not allowed to call APIs or access API groups over the Internet.
A403VN	403	Access from invalid VPC is disabled	The error message returned because access over a VPC is denied.
A403AC	403	Access Control Forbidden by ${RuleName}	The error message returned because access is denied by the access control plugin.
A403CO	403	Cross origin resource forbidden ${Domain}	The error message returned because access is denied by the CORS plugin.
I404CO	404	Cross origin resource not found ${Method} - ${Path}	The error message returned because the API definition is not found based on the Path and Method settings of the request that is pre-checked by the CORS plugin.
I404CH	404	Content not cached, with `Cache-Control:only-if-cached`	
I404NR	404	${Resource} not found	
I404SR	404	Stage route missing: ${Reason}	
B403MO	403	Api Market Subscription overdue	The error message returned because the API provider has overdue payments.

B403MQ	403	Api Market Subscription quota exhausted	The error message returned because the API quota you purchased in Alibaba Cloud Marketplace has been exhausted.
B403ME	403	Api Market Subscription expired	The error message returned because the API subscription relationship expired.
B403MI	403	Api Market Subscription invalid	The error message returned because the API marketplace subscription relationship is invalid.
D504RE	504	Backend domain `${Domain}` resolve failed	The error message returned because the domain name failed to be resolved at the backend.
D504IL	504	Backend domain `${Domain}` resolve to illegal address `${Address}`	The error message returned because the domain name resolution results are invalid at the backend.
D504CO	504	Backend service connect failed `${Reason}`	The error message returned because the backend connection failed. Check the security group configurations, the startup status of the backend server, and firewall configurations.
D504CS	504	Backend http ssl connect failed `${Reason}`	The error message returned because the backend connection over HTTPS failed. Check whether the backend protocol matches the port.
D504TO	504	Backend service request timeout	The error message returned because the backend request timed out.

X504VE	504	Backend service vpc mapped failed	The error message returned because the VPC mapping at the backend is invalid. Submit a ticket to contact customer service.
D503BB	503	Backend circuit breaker busy	The error message returned because the API is protected by its circuit breaker.
D503CB	503	Backend circuit breaker open, ${Reason}	The error message returned because the circuit breaker is open for the API. Check the backend performance of the API.
I508LD	508	Loop Detected	The error message returned because loopback call is detected.
I404DD	404	Device id ${DeviceId} not found	The error message returned because the device ID is not found when you call APIs over WebSocket.
A403FC	403	Function Compute AssumeRole failed ${RequestId}:${Reason}	The error message returned because an authorization error occurs when Function Compute serves as the backend service.
D502FC	502	Function Compute response invalid: ${Reason}	The error message returned because responses from the backend service of the Function Compute type are invalid.
X500ER	500	Service Internal Error	The error message returned because an internal server error occurred. Submit a ticket to contact customer service.

X503BZ	503	Service Busy	The error message returned because the service is busy in API Gateway. Try again later or submit a ticket to contact customer service.
X504TO	504	Service timeout	The error message returned because the service processing timed out in API Gateway.

Some error codes may change with version updates or the addition of new features.