

Alibaba Cloud Apsara Stack Enterprise

User Guide - Middleware and Enterprise
Applications

Version: 2001, Internal: V3.11.0

Issue: 20200513









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 due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall 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 the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. 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 bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, 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 contact Alibaba Cloud directly if you discover any errors in this document.

Document conventions

Style	Description	Example
	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.
	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.
	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.
	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 <code>cd /d C:/window</code> command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	<code>bae log list --instanceid Instance_ID</code>
[] or [a b]	This format is used for an optional value, where only one item can be selected.	<code>ipconfig [-all -t]</code>

Style	Description	Example
{ } or {a b}	This format is used for a required value, where only one item can be selected.	switch {active stand}

Contents

Legal disclaimer.....	I
Document conventions.....	I
1 Enterprise Distributed Application Service (EDAS).....	1
1.1 What is EDAS?.....	1
1.2 Quick start.....	2
1.2.1 Log on to the EDAS console.....	2
1.2.2 Deploy Java applications in ECS clusters.....	3
1.2.3 Deploy Spring Cloud applications to EDAS.....	8
1.2.4 Deploy Dubbo applications to EDAS.....	15
1.2.5 Deploy multi-language microservice-oriented applications.....	22
1.3 Application development.....	29
1.3.1 Use Spring Cloud to develop applications.....	29
1.3.1.1 Spring Cloud overview.....	29
1.3.1.2 Implement service registration and discovery.....	32
1.3.1.3 Implement load balancing.....	32
1.3.1.4 Implement configuration management.....	34
1.3.1.5 Build gateways based on Spring Cloud Gateway.....	38
1.3.1.6 Implement task scheduling.....	41
1.3.2 Use Dubbo to develop applications.....	44
1.3.2.1 Dubbo overview.....	44
1.3.2.2 Use Spring Boot to develop Dubbo applications.....	47
1.3.3 Develop applications in HSF.....	54
1.3.3.1 HSF overview.....	54
1.3.3.2 Configure the lightweight configuration center.....	56
1.3.3.3 Use Ali-Tomcat to develop applications.....	57
1.3.3.3.1 Ali-Tomcat overview.....	57
1.3.3.3.2 Install Ali-Tomcat and Pandora.....	57
1.3.3.3.3 Perform startup configuration for an IDE runtime environment.....	59
1.3.3.3.3.1 Configure the Eclipse development environment.....	59
1.3.3.3.3.2 Configure the IntelliJ IDEA development environment.....	59
1.3.3.3.4 Develop HSF applications (EDAS-SDK).....	60
1.3.3.3.4.1 Download demo projects.....	60
1.3.3.3.4.2 Define service interfaces.....	61
1.3.3.3.4.3 Implement services as a provider.....	61
1.3.3.3.4.4 Subscribe to services as a consumer.....	64
1.3.3.3.4.5 Use HSF features.....	67
1.3.3.3.4.6 Query services.....	71
1.3.3.3.5 Migrate Dubbo applications to HSF (not recommended).....	74
1.3.3.3.5.1 Precautions for developing Dubbo applications.....	74
1.3.3.3.5.2 Modify Dubbo application configurations.....	74

1.3.3.3.5.3 Convert the format of a package from JAR to WAR.....	76
1.3.3.3.5.4 Run programs.....	76
1.3.3.3.5.5 Compatibility between Dubbo and HSF.....	76
1.3.3.4 Use Pandora Boot to develop applications.....	79
1.3.3.4.1 Pandora Boot overview.....	80
1.3.3.4.2 Configure the local repository path and lightweight configuration center of EDAS.....	80
1.3.3.4.3 Develop HSF applications (Pandora Boot).....	82
1.3.3.4.3.1 Example of HSF application development.....	82
1.3.3.4.3.2 Advanced HSF features.....	87
1.3.3.4.3.3 Local debugging.....	91
1.3.3.4.3.4 Deploy applications to EDAS.....	93
1.3.3.4.4 Develop RESTful applications (not recommended).....	93
1.3.3.4.4.1 Terms.....	93
1.3.3.4.4.2 Service registration and discovery.....	94
1.3.3.4.4.3 Distributed tracing.....	102
1.3.3.4.5 Migrate Dubbo applications to HSF (not recommended).....	106
1.4 Deploy applications.....	108
1.4.1 Deploy applications in the console.....	108
1.4.1.1 Deploy web applications in ECS clusters.....	108
1.4.1.2 Deploy applications in Container Service Kubernetes clusters by using images.....	111
1.4.2 Use CLI to deploy applications.....	117
1.4.2.1 Use toolkit-maven-plugin to automatically deploy applications.....	117
1.4.2.2 Use CLI to deploy applications in EDAS.....	125
1.4.2.3 Use Alibaba Cloud Toolkit for Eclipse to deploy applications.....	127
1.4.2.4 Use Alibaba Cloud Toolkit for IntelliJ IDEA to deploy applications.....	130
1.4.3 Deploy applications in hybrid clouds.....	133
1.5 Console user guide.....	137
1.5.1 Overview page.....	137
1.5.2 Resource management.....	137
1.5.2.1 Import ECS instances.....	138
1.5.2.2 View a VPC.....	138
1.5.2.3 Manage clusters.....	139
1.5.2.3.1 Create an ECS cluster.....	139
1.5.2.4 Manage resource groups.....	141
1.5.3 Manage applications.....	142
1.5.3.1 Namespace.....	142
1.5.3.2 Lifecycle management for applications in ECS clusters.....	142
1.5.3.2.1 Publish an application.....	143
1.5.3.2.1.1 Create an empty application (applicable to ECS clusters).....	144
1.5.3.2.1.2 Deploy an application (applicable to ECS clusters).....	146
1.5.3.2.2 Manage applications.....	148
1.5.3.2.2.1 Scaling (applicable to ECS clusters).....	148
1.5.3.2.2.2 Create an application branch version.....	149

1.5.3.2.2.3 Upgrade the container version.....	150
1.5.3.2.2.4 Roll back an application.....	151
1.5.3.2.2.5 Delete an application.....	151
1.5.3.2.3 Application settings.....	152
1.5.3.2.3.1 Set JVM parameters.....	152
1.5.3.2.3.2 Configure Tomcat.....	153
1.5.3.2.3.3 Bind an SLB instance to EDAS.....	154
1.5.3.2.3.4 Set JVM -D startup parameters.....	157
1.5.3.3 Lifecycle management for Container Service Kubernetes applications.....	159
1.5.3.3.1 Container Service Kubernetes clusters.....	159
1.5.3.3.2 Prepare an application image (a Container Service Kubernetes cluster).....	159
1.5.3.3.3 Deploy an application (applicable to Container Service Kubernetes clusters).....	164
1.5.3.3.4 Scaling (applicable to Container Service Kubernetes clusters).....	167
1.5.3.4 Log management.....	167
1.5.3.5 Throttling and degradation (only applicable to HSF applications in ECS clusters).....	168
1.5.3.5.1 Throttling management.....	170
1.5.3.5.2 Degradation management.....	172
1.5.3.6 Container version management (only applicable to HSF applications in ECS clusters).....	174
1.5.4 Microservice management.....	174
1.5.4.1 Trace details.....	175
1.5.5 Batch operations.....	177
1.5.6 System management.....	179
1.5.6.1 Introduction to the EDAS account system.....	179
1.5.6.2 Manage RAM users.....	180
1.5.6.2.1 RAM user overview.....	180
1.5.6.2.2 Use a primary account for RAM user operations.....	181
1.5.6.3 Manage roles.....	182
1.5.6.4 View all permissions.....	182
1.6 FAQ.....	183
1.6.1 Known issues and solutions.....	183
1.6.2 Development FAQ.....	185
1.6.2.1 Ali-Tomcat FAQ.....	185
1.6.2.2 Lightweight configuration center FAQ.....	188
1.6.2.3 HSF FAQ.....	190
1.6.2.4 HSF error codes.....	192
1.6.2.5 Other development problems.....	199
1.6.3 Usage FAQ.....	199
1.6.3.1 Account management.....	199
1.6.3.2 Resource management.....	200
1.6.3.3 Application lifecycle.....	202

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

- Before logging on to the ASCM console, make sure that you have obtained the IP address or domain name of the ASCM console from the deployment personnel. The URL used to

access the ASCM console is in the following format: `https://[IP address or domain name of the ASCM console]`.

- We recommend that you use the Google Chrome browser.

Procedure

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

Obtain the username and password for logging 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 as prompted. Due to security concerns, 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 letters, lowercase letters, digits, and special characters such as 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, click **Products**, and select **EDAS**.
5. On the **EDAS** page, select an **organization** and a **region**, and then click **Go to 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**.

Table 1-1: 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	<p>Select the application runtime environment based on the application framework.</p> <ul style="list-style-type: none"> • For High-Speed Service Framework (HSF) applications, select EDAS-Container. • For Spring Cloud or Dubbo applications, <ul style="list-style-type: none"> - 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	<p>Click New. On the Instances page, select instances and click > to add the instances to the right-side section. Then, click OK.</p> <ul style="list-style-type: none">• 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 ApplicationDeploy 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.

Parameter	Description
File Uploading Method	<p>Select Upload WAR Package or WAR Package Location.</p> <ul style="list-style-type: none"> • 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. <div>  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. </div>
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 .
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.

- a. Run the main function of `ProviderApplication` in `nacos-service-provider` to start the application.
- b. 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.

- a. Run ConsumerApplication.java in eureka-service-provider to start the application.
- b. 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. [Configure SLB and access the application](#) 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
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.

Configuration item	Key	Default value	Description
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.

Configuration item	Key	Default value	Description
Cluster	spring.cloud.nacos. .discovery.cluster- name	DEFAULT	Set this item to the name of a Nacos Server cluster.
Endpoint	spring.cloud.nacos. discovery.endpoint	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:
 1. Decompress the downloaded Nacos Server package.
 2. 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.
 - a) Create a Maven project by using an integrated development environment (IDE), such as IntelliJ IDEA or Eclipse.
 - b) 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>
```

```
</dependencies>
```

2. Develop a Dubbo service provider.

All services in Dubbo are provided as interfaces.

- a) Create a package named `com.alibaba.edas` in `src/main/java`.
- b) 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);
}
```

- c) 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.

- a) Create a file named `provider.xml` in `src/main/resources` and open the file.
- b) 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">
```

- c) 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"/>
```

d) 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.

a) 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 ClassPathXmlApplicat
ionContext(new String[] {"provider.xml"});
        context.start();
        System.in.read();
    }
}
```

b) 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.

- a) Create a Maven project by using an integrated development environment (IDE), such as IntelliJ IDEA or Eclipse.
- b) 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.

- a) Create a package named com.alibaba.edas in src/main/java.
- b) 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.

- a) Create a file named `consumer.xml` in `src/main/resources` and open the file.
- b) 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">
```

- c) 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.

- a) 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 ClassPathXmlApplicat
ionContext(new String[] {"consumer.xml"});
        context.start();
        while (true) {
            try {
                TimeUnit.SECONDS.sleep(5);
                IHelloService demoService = (IHelloService)context.getBean("helloServi
ce");
                String result = demoService.sayHello("world");
                System.out.println(result);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

- b) 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

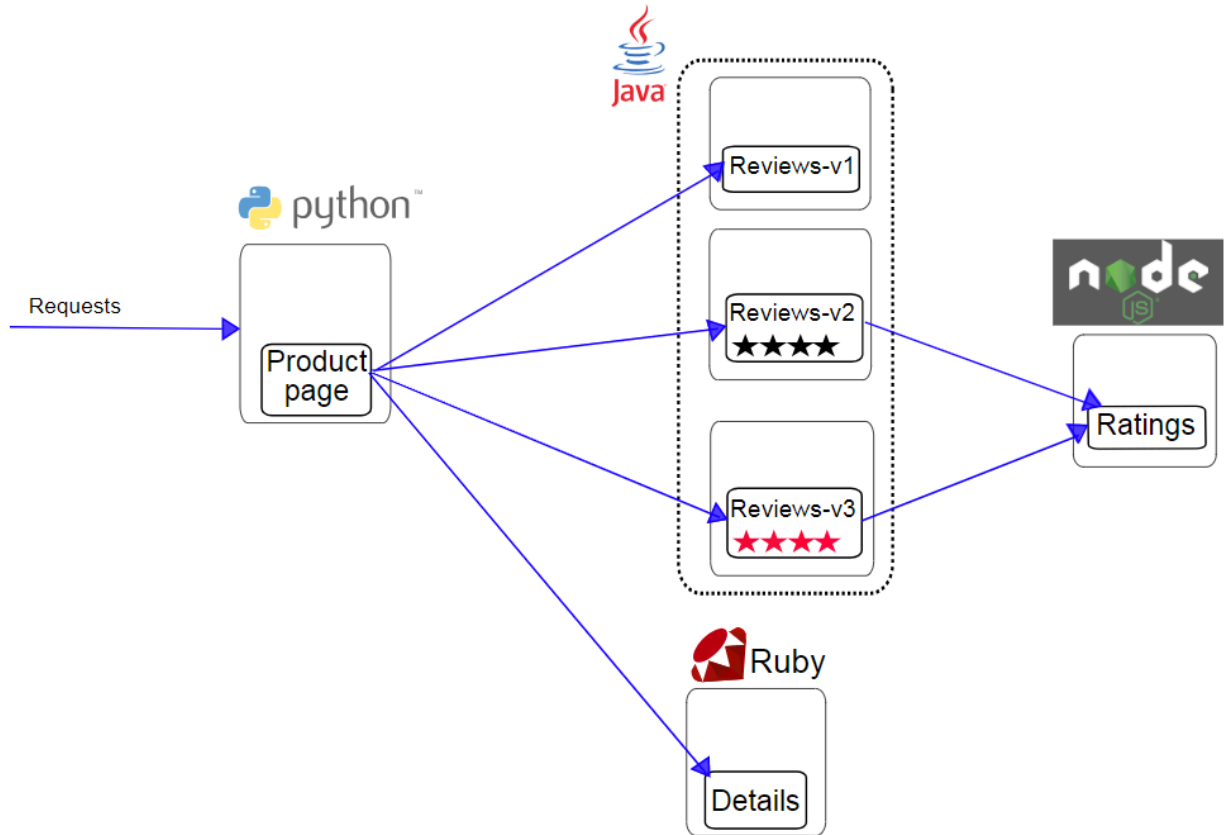
Application A provides the test service. Generally, we access the service by using the domain name or IP address, for example, `http://test.com:8080/` or `http://xx.xx.xx.xx:8080`. After the service mesh is used, the instance where a service is deployed can be abstracted into a service. The following 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.

a) 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.

b) 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. **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 compatibility between Spring Cloud features, other open-source implementation components, and EDAS.

Spring Cloud feature		Open source component	Compatibility with EDAS
Common features	Service registration and discovery	<ul style="list-style-type: none">Netflix EurekaConsul Discovery	Compatible, with an equivalent component
	Load balancing	Netflix Ribbon	Compatible
	Service call	<ul style="list-style-type: none">FeignRestTemplate	Compatible

Spring Cloud feature	Open source component	Compatibility with EDAS
Configuration management	<ul style="list-style-type: none"> Config Server Consul Config 	Compatible, with an equivalent component
Service gateway	<ul style="list-style-type: none"> Spring Cloud Gateway Netflix Zuul 	Compatible
Distributed tracing	Spring Cloud Sleuth	Compatible, with an equivalent component
Message-driven application development: Spring Cloud Stream	<ul style="list-style-type: none"> RabbitMQ Binder Kafka Binder 	Compatible, with an equivalent component
Message bus: Spring Cloud Bus	<ul style="list-style-type: none"> 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
			<ul style="list-style-type: none"> Nacos Registry Nacos Config
Greenwich	2.1.x	2.1.1.RELEASE	2.1.1.RELEASE
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 RibbonLoad  
BalancerClient of the LoadBalancerClient interface for load balancing.  
@Autowired  
private RestTemplate restTemplate;  
@LoadBalanced // Modify the built RestTemplate with this annotation to enable its  
load balancing function.  
@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.

- a. Use `@EnableFeignClients` to enable the functions of Feign.

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

- b. 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);  
}
```

- c. 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

EDAS integrates ACM into the console as a component to implement application configuration management. This topic describes how to connect your Spring Cloud applications to ACM, deploy the applications to EDAS, and use ACM to manage their configurations.

Prerequisites

- You have downloaded, started, and configured the lightweight configuration center.

To help you develop applications locally, EDAS provides the lightweight configuration center that contains the basic functions of the EDAS service registry. Applications that are developed in the lightweight configuration center can be deployed to off-premises EDAS, without the need to modify any code or configuration.

For more information about how to download, start, and configure the lightweight configuration center, see [Configure the lightweight configuration center](#). The latest version is recommended.

- Log on to the [lightweight configuration center console](#). In the left-side navigation pane, choose **Configuration List**. On the **Configuration List** page, click **Add**. On the **Create Configuration** page, enter the following information:
 - Group: DEFAULT_GROUP
 - DataId: acm-example.properties
 - Content: user.id=amctest



Note:

When setting the configuration management parameters, you must set Group to DEFAULT_GROUP. After local debugging is finished, log on to the EDAS console. In the left-side navigation pane, choose **Application Management > Configuration Management**. On the **Configuration Management** page, enter the configuration information based on this restriction. You do not need to modify the code. Therefore, you do not need to refer to the sample code in the Actions column and add such configuration items as AccessKeyId, AccessKeySecret, and ACM endpoint and namespace.

Context

ACM is a configuration management product of Alibaba Cloud, which is a commercial version of open source Nacos. Compared with other similar products, ACM offers certain advantages.

This topic describes key information for developing applications locally. For more information about Spring Cloud, download [acm-example](#).

Procedure

1. Create a Maven project named `acm-example`.
2. The following takes 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-acm</artifactId>
    <version>0.2.2.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>
```

If you want to use Spring Boot 1.x, use Spring Boot 1.5.x, Spring Cloud Edgware, and Spring Cloud Alibaba 0.1.2.RELEASE.



Note:

Spring Boot 1.x will expire in August 2019, so we recommend that you use a later version to develop applications.

3. Develop the startup class AcmExampleApplication of acm-example.

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

4. Create a simple controller and retrieve the value of UserId from the key user.id in the configuration file.

```
@RestController
public class EchoController {
    @Value("${user.id}")
    private String userId;
    @RequestMapping(value = "/")
    public String echo() {
        return userId;
    }
}
```

5. In the bootstrap.properties file, add the following configuration and specify the EDAS lightweight configuration center as the registry.

In the configuration, 127.0.0.1 is the address of the lightweight configuration center.

If your the lightweight configuration center is deployed on another instance, change

the address to the IP address of the instance. The default port of the lightweight configuration center is **8080** and cannot be changed.

```
spring.application.name=acm-example
server.port=18081
spring.cloud.alicloud.acm.server-list=127.0.0.1
spring.cloud.alicloud.acm.server-port=8080
```

If you have additional requirements, see the section "Reference configuration items" and add relevant configurations in the `bootstrap.properties` file.

Table 1-2: Reference configuration items

Configuration item	Key	Default value	Description
Extension	spring.cloud.alicloud.acm.file-extension	properties	The extension of the configuration file , which typically is properties or yaml.
Timeout	spring.cloud.alicloud.acm.timeout	3000	The timeout period for retrieving the configuration.
Refresh	spring.cloud.alicloud.acm.refresh-enabled	true	Specifies whether to refresh the Spring context when the configuration changes.
Endpoint	spring.cloud.alicloud.acm.endpoint	None	None
Namespace	spring.cloud.alicloud.acm.namespace	None	None
RAM role	spring.cloud.alicloud.acm.ram-role-name	None	None

6. Run the main function in `AcmExampleApplication` to enable the service.

Result

In the address bar of your browser, enter <http://127.0.0.1:18081/>. The value `acmtest` is returned, which indicates the value of `user.id` you configured in the lightweight configuration center.

What's next

Deploy applications to EDAS

ACM is designed for migrating applications from the development environment to EDAS. It allows you to directly deploy applications to EDAS without any code or configuration modifications. After the applications are deployed, you can manage the application configuration in the EDAS console. The configuration management function in the EDAS console is consistent with that in ACM.

1. In the `pom.xml` file of `acm-example`, add the following configuration. Then, run the `mvn clean package` command to package local programs into executable JAR packages.

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

2. Deploy applications according to the relevant documentation for the cluster type you want to deploy.

1.3.1.5 Build gateways based on Spring Cloud Gateway

This topic describes how to use ANS to build an application gateway from scratch based on Spring Cloud Gateway.

Prerequisites

- You have downloaded, started, and configured the lightweight configuration center.

To help you develop applications locally, EDAS provides the lightweight configuration center that contains the basic functions of the EDAS service registry. Applications that are developed in the lightweight configuration center can be deployed to off-premises EDAS, without the need to modify any code or configuration.

- You have downloaded [Maven](#) and set the environment variables. If you have installed Maven in your local instance, skip this step.

Procedure

1. Create a Maven project named `spring-cloud-example-ans-gateway`.

2. In the pom.xml file, add the dependencies of Spring Boot and Spring Cloud Finchley.

The following takes Spring Boot 2.0.6.RELEASE and Spring Cloud Finchley.SR1 as an example.

**Note:**

Spring Cloud Gateway is a component developed based on Spring Boot 2.0. If you use Spring Cloud Gateway as the gateway, select Spring Boot 2.0 or later. If you use Spring Boot 1.x, we recommend that you upgrade it to Spring Cloud 2.0.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.6.RELEASE</version>
  <relativePath/>
</parent>
<properties>
  <spring-cloud.version>Finchley.SR1</spring-cloud.version>
  <spring-cloud-alibaba-cloud.version>0.2.1.RELEASE</spring-cloud-alibaba-cloud.
version>
</properties>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-alibaba-dependencies</artifactId>
      <version>${spring-cloud-alibaba-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-alicloud-ans</artifactId>
    <exclusions>
      <!-- Spring Cloud Gateway uses Netty as its HTTP server. You must **exclude
** the dependency on spring-boot-starter-web. Otherwise, the gateway cannot be
started. -->
      <exclusion>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

```
</exclusion>
</exclusions>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-alicloud-context</artifactId>
</dependency>
</dependencies>
```

3. Develop the gateway startup class `AnsGatewayApplication`.

```
@SpringBootApplication
@EnableDiscoveryClient
<!-- Service registration and discovery must be enabled for the application. -->
public class AnsGatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(AnsGatewayApplication.class, args);
    }
}
```

4. In the `application.yaml` file, add the following configuration and specify the EDAS lightweight configuration center as the registry.

In the configuration, 127.0.0.1 is the address of the lightweight configuration center. If your lightweight configuration center is deployed on another instance, change the address to the IP address of the instance. The default port of the lightweight configuration center is **8080** and cannot be changed.

```
server:
  port: 15012
spring:
  application:
    name: spring-gateway-example
  cloud:
    gateway: # config the routes for gateway
    routes:
      - id: lb_service-provider
        uri: lb://service-provider
        predicates:
          - Path=/**
    alicloud:
      ans:
        server-list: 127.0.0.1
        server-port: 8080
```

5. Run the main function in the `AnsGatewayApplication` startup class to enable the service.
6. Log on to the lightweight configuration center console <http://127.0.0.1:8080>. In the left-side navigation pane, choose **Services** to view the list of service providers. `spring-gateway-example` exists in the list of service providers.
7. Create a service provider.

Result

1. Locally verify the result.

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

2. Verify the result in EDAS.

Deploy the gateway and service provider you just created to EDAS and access Spring Cloud Gateway to forward the request to the service provider. The result indicating a successful call is returned.

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.

- a) Log on to the EDAS console. In the **test** region, create a scheduled task group and record the group ID.
- b) 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.
- c) 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:

- a. Log on to the EDAS console. In the China (Hangzhou) region, create a **task group** and a **scheduled task**.
- b. Access the Security Management page, and retrieve the **AccessKeyId** and **AccessKeySecret**.
- c. In the `application.properties` file, configure the scheduled task.

d. 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.

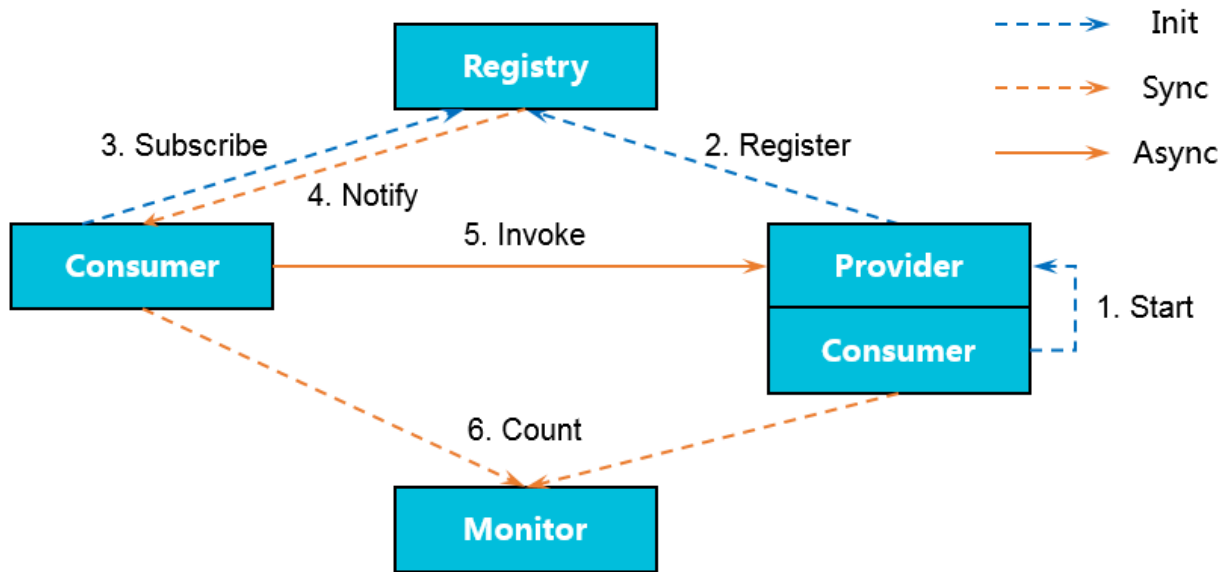
Dubbo architecture

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

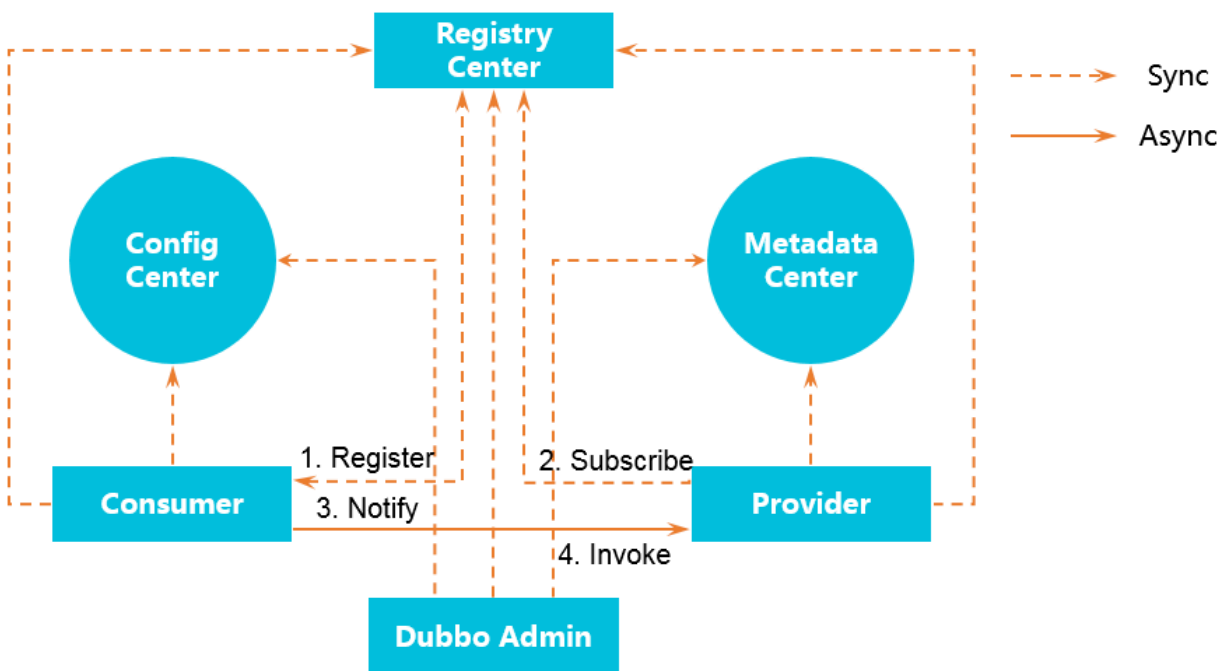
- Dubbo 2.6.x is widely used and will be maintained, but will not be upgraded with new features.
- Dubbo 2.7.x is the latest version of Apache Dubbo and will be upgraded with new features.

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
Registry	<ul style="list-style-type: none">• Nacos (recommended)• ZooKeeper (recommended)• etcd• Consul• Eureka	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• Nacos (recommended)• ZooKeeper (recommended)• Apollo	Nacos (recommended)	Add the dubbo-configcenter-nacos dependency to the application.
Metadata center	<ul style="list-style-type: none">• 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.
 1. Decompress the downloaded Nacos Server package.
 2. 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>
```

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

3. Develop a Dubbo service provider.

All services in Dubbo are provided as interfaces.

- a) Create a package named `com.alibaba.edas.boot` in `src/main/java`.
- b) 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);  
}
```

- c) 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.

- a) In `src/main/resources`, create a file named `application.properties` or `application.yml` and open it.
- b) In `application.properties` or `application.yml`, 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>
</dependencies>
```

```
<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.

- a) Create a package named com.alibaba.edas.boot in src/main/java.
- b) 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

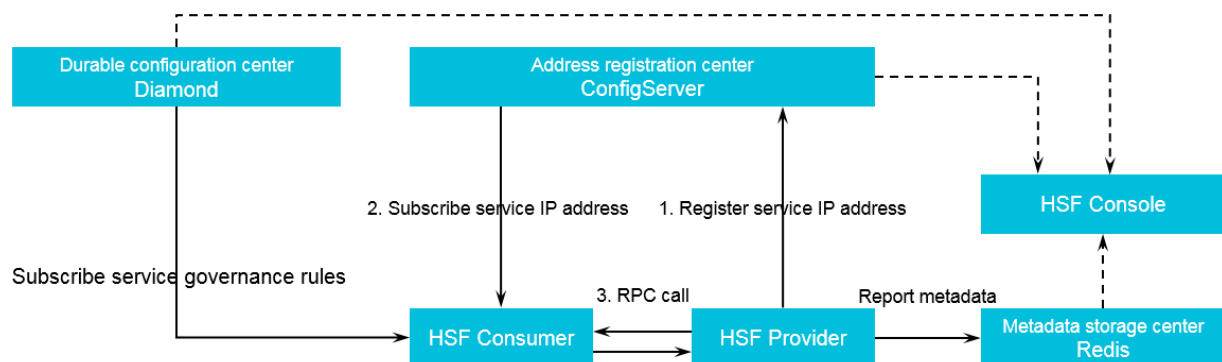
1.3.3.1 HSF overview

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.

Figure 1-1: 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.

1.3.3.3.3.1 Configure the Eclipse development environment

To configure Eclipse, you must download the Tomcat4E plug-in and save it to the directory of the Pandora container that you downloaded from [Install Ali-Tomcat and Pandora](#). After configuring Eclipse, you can directly publish and debug local code in Eclipse.

Procedure

1. Download the package of the [Tomcat4E plug-in](#) and decompress it to a local directory, such as d:\work\tomcat4e\. The package contains the following items:
2. Open Eclipse. From the toolbar, choose **Menu > Help**. On the page that appears, click **Install New Software**. In the dialog box that appears, choose **Add > Local**. Select the directory (d:\work\tomcat4e\) to which Tomcat4E is decompressed, and then click **OK**.
3. Click **Select All** and then **Next**. Tomcat4E is installed.
4. Restart Eclipse.
5. Configure the Eclipse project to activate Tomcat4E. Right-click the target Eclipse project and choose **Run As > Run Configurations** from the shortcut menu.
6. In the left-side navigation pane, click **Ali-Tomcat Webapp**, and click the **New launch configuration** icon at the top.
7. On the page that appears, click the **Ali-Tomcat** tab. In the Pandora (taobao-hsf.sar location) section, select **Use local taobao-hsf.sar**, and click **Browse**. Select the local path for Pandora, such as d:\work\tomcat\deploy\taobao-hsf.sar.
8. Click **Apply** or **Run**. Next time, you can directly start this project without the need to configure it again.

1.3.3.3.3.2 Configure the IntelliJ IDEA development environment

The configuration of an integrated development environment (IDEA) does not depend on any additional plug-ins. You can use the JVM startup parameter Dpandora.location. Now, the IntelliJ IDEA commercial edition, but not the community edition, is supported.

Procedure

1. From the menu or toolbar, choose **Run > Edit Configuration**.

2. Click **+** and choose **Tomcat Server > Local** to add the local Tomcat startup configuration.
3. Configure Ali-Tomcat: On the right of the page, click the **Server** tab. Click **Configure** on the right of the **Application server** field. On the page that appears, select the path of Ali-Tomcat that you downloaded from [Install Ali-Tomcat and Pandora](#), for example, d:\work\tomcat\.
4. Select the Ali-Tomcat instance you configured from the Application server drop-down list.
5. In the VM Options field, enter the JVM startup parameter that points to the Pandora path, such as -Dpandora.location=d:\work\tomcat\deploy\taobao-hsf.sar.
6. Click **Apply** or **OK**.

1.3.3.3.4 Develop HSF applications (EDAS-SDK)

1.3.3.3.4.1 Download demo projects

A demo project is provided for users as a code sample. This topic describes how to download the demo project.

Context



Note:

Use JDK 1.7 or later.

Procedure

1. Download the compressed demo project package.

All the following codes are available in the official demos. You can click [Here](#) to download official demos.

2. Decompress the downloaded package.

Three Maven projects are available: itemcenter-api, itemcenter, and detail. Among the demo projects:

- The itemcenter-api project provides the interface definition.
- The detail project is the service consumer application.
- The itemcenter project is a service provider application.

1.3.3.3.4.2 Define service interfaces

HSF services are implemented based on interfaces. After an interface is defined, the provider can implement a specific service through this interface. The consumer also subscribes to the service over this interface.

Context



Notice:

This topic only describes how to define a service interface. However, in actual application scenarios, you need to add routes in an interface and implement them through an instance because definition alone is inadequate.

Procedure

1. In the demo project, locate the itemcenter-api folder, and locate and open the ItemService.java file.

In the file, the service interface com.alibaba.edas.carshop.itemcenter.ItemService is defined and has the following content:

```
public interface ItemService {  
    public Item getItemById( long id );  
    public Item getItemByName( String name );  
}
```

The interface provides two methods: getItemById and getItemByName, indicating that the com.alibaba.edas.carshop.itemcenter.ItemService service provides the getItemById and getItemByName methods.

2. Define new service interfaces based on planning or actual implementation.

1.3.3.3.4.3 Implement services as a provider

The provider implements an interface to provide specific services. Besides code implementation, you must define the XML file used for service publishing because HSF is implemented based on the Spring framework.

1.3.3.3.4.3.1 Implement service interfaces by code

The sample code of itemcenter in the demo project is as follows:

```
package com.alibaba.edas.carshop.itemcenter;  
public class ItemServiceImpl implements ItemService {  
  
    @Override  
    public Item getItemById( long id ) {  
        Item car = new Item();  
        car.setItemId( 1 );  
        car.setItemName( "Mercedes Benz" );  
        return car;  
    }  
}
```

```
}  
@Override  
public Item getItemByName( String name ) {  
    Item car = new Item();  
    car.setItemId( 11 );  
    car.setItemName( "Mercedes Benz" );  
    return car;  
}  
}
```

1.3.3.3.4.3.2 Configure services

Context

[Implement service interfaces by code](#) The preceding example implements the service API `com.alibaba.edas.carshop.itemcenter.ItemService` and returns an `Item` object to both methods. After developing code, configure the required general Spring items and add Maven dependencies in the `web.xml` file. Then, use the `<hsf />` tag in the Spring configuration file to register and publish the service.

Procedure

1. Add the following Maven dependencies to the `pom.xml` file:

```
<dependencies>  
  <dependency>  
    <groupId>javax.servlet</groupId>  
    <artifactId>servlet-api</artifactId>  
    <version>2.5</version>  
    <scope>provided</scope>  
  </dependency>  
  <dependency>  
    <groupId>com.alibaba.edas.carshop</groupId>  
    <artifactId>itemcenter-api</artifactId>  
    <version>1.0.0-SNAPSHOT</version>  
  </dependency>  
  <dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-web</artifactId>  
    <version>2.5.6 (or later)</version>  
  </dependency>  
  <dependency>  
    <groupId>com.alibaba.edas</groupId>  
    <artifactId>edas-sdk</artifactId>  
    <version>1.5.0</version>  
  </dependency>  
</dependencies>
```

2. Add the HSF-specific Spring configurations. The content of the HSF configuration file (`/resources/hsf-provider-beans.xml`) of the demo project is as follows:

```
<? xml version="1.0" encoding="UTF-8"? >  
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:hsf="http://www.taobao.com/hsf"  
  xmlns="http://www.springframework.org/schema/beans"  
  xsi:schemaLocation="http://www.springframework.org/schema/beans  
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd  
    http://www.taobao.com/hsf
```



```
http://www.taobao.com/hsf/hsf.xsd" default-autowire="byName">
<!-- Define the implementation of the service. -->
<bean id="itemService" class="com.alibaba.edas.carshop.itemcenter.ItemServiceImpl" />
<!-- Use the hsf:provider tag to define a service provider. -->
<hsf:provider id="itemServiceProvider"
<!-- Use the interface property to indicate that the service is an implementation of the
class. -->
interface="com.alibaba.edas.carshop.itemcenter.ItemService"
<!-- The Spring object that is implemented by the service -->
ref="itemService"
<!-- The version of the published service, which is user-defined and is 1.0.0 by default
. -->
version="1.0.0"
</hsf:provider>
</beans>
```

1.3.3.3.4.3.3 Provider configuration properties

In addition to the properties shown in the preceding sample, you can use the following properties of the HSF provider configuration:

Property	Description
interface	A required string-type property. It is the interface for providing the target service.
version	An optional string-type property. It is the version of the target service. The default value is 1.0.0.
clientTimeout	This property applies to all methods in the interface. However, if the consumer sets a timeout period for a method by using the MethodSpecials property, the timeout period configured on the consumer prevails for the method. Other methods are not affected by this property and still use the timeout period configured on the provider.
serializeType	Optional. It indicates the serialization type. Its value is in string format and can be Hessian or Java. The default value is Hessian.
corePoolSize	This property is used to set part of the public thread pool as the core thread pool dedicated to this service.
maxPoolSize	This property is used to set part of the public thread pool as the maximum thread pool dedicated to this service.
enableTXC	This property enables the distributed transaction middleware GTS.
ref	A required ref-style property. It indicates the ID of the Spring bean that you want to publish as an HSF service.
methodSpecials	Optional. It is used to configure a timeout period (unit: ms) for each method. With this property, methods in an interface can apply to different timeout periods. This timeout property takes precedence over clientTimeout but defers to methodSpecials on the consumer.

Tag configuration example:

```
<bean id="impl" class="com.taobao.edas.service.impl.SimpleServiceImpl" />
<hsf:provider id="simpleService" interface="com.taobao.edas.service.SimpleService"
  ref="impl" version="1.0.1" clientTimeout="3000" enableTXC="true"
  serializeType="hessian">
  <hsf:methodSpecials>
    <hsf:methodSpecial name="sum" timeout="2000" />
  </hsf:methodSpecials>
</hsf:provider>
```

1.3.3.3.4.3.4 Publish services in the development environment

After coding and configuration, you can directly publish the service in Eclipse or IntelliJ IDEA.

Procedure

1. You can directly run the service by using Ali-Tomcat in Eclipse or IntelliJ IDEA. For more information, see [Startup configurations during IDE operation](#).
2. After the service runs properly, you can query the service you published in the configuration center. For more information, see [Query services](#).

1.3.3.3.4.3.5 Other JVM startup parameters

This topic describes additional JVM startup parameters.

The following table lists additional startup parameters in the service provider that change the behavior of HSF.

Property	Description
-Dhsf.server.port	Specifies a port bound to the HSF startup service. The default value is 12200.
-Dhsf.serializer	Specifies the serialization method of HSF, which is Hessian by default.
-Dhsf.server.max.poolsize	Specifies the maximum size of the thread pool of the HSF provider. The default value is 600.
-Dhsf.server.min.poolsize	Specifies minimum size of the thread pool of the HSF provider. The default value is 50.

1.3.3.3.4.4 Subscribe to services as a consumer

Service subscription for consumers is coded in two steps: First, use the <hsf:consumer/> tag in the Spring configuration file to define a bean. Second, retrieve the bean from the Spring context. In the demo project, detail shows a consumer-specific example.

1.3.3.3.4.4.1 Configure consumers

Similar to that of providers, the configuration file of consumers consists of the Maven dependency configuration and Spring configuration. The Maven dependency configuration of consumers is the same as that of providers. For more information, see [Configure services](#).

In addition to required Spring configurations, you must add the consumer definition in the Spring configuration file. Then, the HSF framework subscribes to the target services in the service center based on the configuration file. The content of the configuration file / resource/hsf-consumer-beans.xml and the meaning are as follows:

```
<? xml version="1.0" encoding="UTF-8"? >
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:hsf="http://www.taobao.com/hsf"
  xmlns="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.taobao.com/hsf
    http://www.taobao.com/hsf/hsf.xsd" default-autowire="byName">
  <!-- Example of service consumption -->
  <hsf:consumer
    <!-- The Bean ID that is used to retrieve the consumer object by code injection -->
    id="item"
    <!-- The name of the service, which corresponds to the service name of the service
    provider. HSF queries and subscribes to services according to the combined criteria of
    interface and version. -->
    interface="com.alibaba.edas.carshop.itemcenter.ItemService"
    <!-- The version that corresponds to the version of the service provider. HSF queries
    and subscribes to services according to the combined criteria of interface and version. --
  >
    version="1.0.0"
  </hsf:consumer>
</beans>
```

1.3.3.3.4.2 Use services as a consumer

This topic provides sample code for using services as a consumer.

The sample code in the demo project is as follows:

```
public class StartListener implements ServletContextListener{

  @Override
  public void contextInitialized( ServletContextEvent sce ) {
    ApplicationContext ctx = WebApplicationContextUtils.getWebApplicationContext( sce.
    getServletContext() );
    // Retrieve subscribed services according to the bean ID "item" in the Spring configurat
    ion.
    final ItemService itemService = ( ItemService ) ctx.getBean( "item" );
    .....
    // Call the getItemById method of ItemService.
    System.out.println( itemService.getItemById( 1111 ) );
    // Call the getItemByName method of ItemService.
    System.out.println( itemService.getItemByName( "myname is le" ) );
    .....
  }
}
```

```
}
```

1.3.3.3.4.3 Consumer configuration properties

This topic describes consumer configuration properties.

In addition to the properties in the sample code, such as interface and version, you can use other properties listed in the following table.

Property	Description
interface	A required string-type property. It is the interface for calling the target service.
version	An optional string-type property. It is the version of the target service. The default value is 1.0.0.
methodSpecials	Optional. It is used to configure the timeout period (unit: milliseconds) for each method separately. In this way, methods in an interface can apply different timeout periods. The timeout period specified by this property takes precedence over that of the provider.
target	This property is used in the unit testing environment and development environment where hsf.runmode is set to 0. In the runtime environment, this property is invalid, and the target service address pushed by the configuration center is used instead.
connectionNum	Optional. It is the maximum number of connections to the provider. The default value is 1. If you transmit a small amount of data and require a shorter delay, set this property to a larger value to improve TPS.
clientTimeout	It indicates that the consumer sets the same timeout period (unit : milliseconds) for all methods in an interface. Timeout settings are sorted in descending order of priority as follows: consumer MethodSpecial, consumer interface level, provider MethodSpecial, and provider interface level.
asynccallMethods	<p>An optional list-type property. It indicates that the asynchronously called method name list and asynchronous calls are required for calling the service.</p> <p>This property is an empty set by default, which indicates that all methods are called synchronously.</p>

Property	Description
maxWaitTimeForCsAddress	This property indicates the time during which the thread is blocked to wait for address push when a service is subscribed. Otherwise, the address may not be found due to an empty address when the service is called. If the address is not pushed before the blocking time expires, the thread no longer waits and proceeds with initialization.

Tag configuration example:

```
<hsf:consumer id="service" interface="com.taobao.edas.service.SimpleService"
  version="1.1.0" clientTimeout="3000"
  target="10.1.6.57:12200?_TIMEOUT=1000" maxWaitTimeForCsAddress="5000">
  <hsf:methodSpecials>
    <hsf:methodSpecial name="sum" timeout="2000" />
  </hsf:methodSpecials>
</hsf:consumer>
```

1.3.3.3.4.4 Consume services in the development environment

After coding and configuration, you can directly consume a service in Eclipse or IntelliJ IDEA.

Procedure

1. You can directly run the service by using Ali-Tomcat in Eclipse or IntelliJ IDEA. For more information, see [Startup configurations during IDE operation](#).
2. After the service starts running properly, query it in the configuration center. For more information, see the chapter about building a development environment.

1.3.3.3.4.5 Use HSF features

This topic describes the usage instructions and important notes for HSF features. You can download the demos for using all features in [demo](#).

1.3.3.3.4.5.1 Prerequisites

To use HSF features, you must add the following edas-sdk dependency to the pom.xml file:

```
<dependency>
  <groupId>com.alibaba.edas</groupId>
  <artifactId>edas-sdk</artifactId>
  <version>1.5.1</version>
</dependency>
```

1.3.3.3.4.5.2 Implicit parameter passing (currently, only string parameter passing is supported)

Implicit parameter passing is generally used to replace the interface mode for passing simple KV data and it is similar to cookies.

You can pass either a single parameter or multiple parameters in implicit mode.

- **Pass a single parameter**

- Service consumer:

```
RpcContext.getContext().setAttachment("key", "args test");
```

- Service provider:

```
String keyVal=RpcContext.getContext().getAttachment("key");
```

- **Pass multiple parameters**

- Service consumer:

```
Map<String,String> map=new HashMap<String,String>();  
map.put("param1", "param1 test");  
map.put("param2", "param2 test");  
map.put("param3", "param3 test");  
map.put("param4", "param4 test");  
map.put("param5", "param5 test");  
RpcContext rpcContext = RpcContext.getContext();  
rpcContext.setAttachments(map);
```

- Service provider:

```
Map<String,String> map=rpcContext.getAttachments();  
Set<String> set=map.keySet();  
for (String key : set) {  
    System.out.println("map value:"+map.get(key));  
}
```

1.3.3.3.4.5.3 Asynchronous calls

Asynchronous calls can be implemented by the Callback and Future methods.

- **Callback method**

If the Callback method is configured for the service consumer, you must configure a listener that implements HSFResponseCallback. After the result is returned, HSF calls the method in HSFResponseCallback.



Notice:

The listener of HSFResponseCallback must not be an internal class. Otherwise, an error is reported when the Pandora classloader is loaded.

XML configuration:

```
<hsf:consumer id="demoApi" interface="com.alibaba.demo.api.DemoApi"  
version="1.1.2">  
  <hsf:asyncallMethods>  
    <hsf:method name="ayncTest" type="callback"  
      listener="com.alibaba.ifree.hsf.consumer.AsynABTestCallbackHandler" />  
  </hsf:asyncallMethods>
```

```
</hsf:consumer>
```

The `AsynABTestCallbackHandler` class implements `HSFResponseCallback`. `DemoApi` has the `ayncTest` method .

Sample code:

```
public void onAppResponse(Object appResponse) {  
    //Retrieve the value after the asynchronous call.  
    String msg = (String)appResponse;  
    System.out.println("msg:"+msg);  
}
```



Note:

- Method names are used to identify methods. Therefore, repeatedly loaded methods are not differentiated. Methods that have the same name are set with the same call method.
- HSF calls cannot be initiated in a call. Otherwise, the I/O thread is suspended and cannot be recovered.

- **Future method**

If the Future method is configured for the service consumer, after a call is initiated, the return result is obtained through public static Object `getResponse(long timeout)` in `HSFResponseFuture`.

XML configuration:

```
<hsf:consumer id="demoApi" interface="com.alibaba.demo.api.DemoApi" version="1.1.2">  
    <hsf:asyncallMethods>  
        <hsf:method name="ayncTest" type="future" />  
    </hsf:asyncallMethods>  
</hsf:consumer>
```

Sample code:

- Asynchronous processing of a single call:

```
//Initiate a call.  
demoApi.ayncTest();  
// Process the service.  
...  
//Directly obtain the message. If the result is not required, you can skip this step.
```

```
String msg=(String) HSFResponseFuture.getResponse(3000);
```

- Concurrent processing of multiple calls:

To process multiple tasks concurrently, retrieve and store the future object and then reuse it after the call.

```
//Define a set.  
List<HSFFuture> futures = new ArrayList<HSFFuture>();
```

- Concurrent call within a method:

```
//Initiate a call.  
demoApi.asyncTest();  
//Retrieve the future object.  
HSFFuture future=HSFResponseFuture.getFuture();  
futures.add(future);  
//Continue calling other services in asynchronous call mode.  
HSFFuture future=HSFResponseFuture.getFuture();  
futures.add(future);  
  
// Process the service.  
...  
  
//Retrieve and process the data.  
for (HSFFuture hsfFuture : futures) {  
String msg=(String) hsfFuture.getResponse(3000);  
//Process corresponding data.  
...  
}
```

1.3.3.3.4.5.4 Generic calls

Generic calls can combine interfaces, methods, and parameters for RPCs, without depending on any service interface.

Procedure

1. Add the generic property to the service consumer configuration.

```
<hsf:consumer id="demoApi" interface="com.alibaba.demo.api.DemoApi" generic="true"/>
```



Note:

The property generic indicates generic parameters, the value true indicates that generic parameters are supported, and the value false indicates that generic parameters are not supported. The default value is false.

DemoApi method:

```
public String dealMsg(String msg);
```



```
public GenericTestDO dealGenericTestDO(GenericTestDO testDO);
```

2. Retrieve demoApi to enforce conversion to a generic service.

- a) Import the generic service interface.

```
import com.alibaba.dubbo.rpc.service.GenericService
```

- b) Retrieve generic objects.

```
//In a web project, you can enforce service conversion after injection using a Spring  
bean. This example is a unit test and therefore you must load the configuration file.  
ClassPathXmlApplicationContext consumerContext = new ClassPathXmlApplicat  
ionContext("hsf-generic-consumer-beans.xml");  
GenericService svc = (GenericService) consumerContext.getBean("demoApi");
```

3. Perform generic operations.

Object \$invoke(String methodName, String[] parameterTypes, Object[] args) throws
GenericException;



Note:

- methodName: The name of the method you want to call.
- parameterTypes: The type of the parameters of the method you want to call.
- args: The parameter value you want to transmit.

4. Make generic calls.

- String-type parameters

```
svc.$invoke("dealMsg", new String[] { "java.lang.String" }, new Object[] { "hello" })
```

- Object parameters

```
// Construct the entity object GenericTestDO, which has the ID and name properties.  
GenericTestDO genericTestDO = new GenericTestDO();  
genericTestDO.setld(1980l);  
genericTestDO.setName("genericTestDO-tst");  
// Use PojoUtils to generate the pojo description of the second-party package.  
Object comp = PojoUtils.generalize(genericTestDO);  
// Call the service in generic mode.  
svc.$invoke("dealGenericTestDO", new String[] { "com.alibaba.demo.generic.  
domain.GenericTestDO" }, new Object[] { comp });
```

1.3.3.3.4.6 Query services

Currently, EDAS supports the registration of Dubbo and HSF services. This topic only describes how to query HSF services. If your Dubbo services are published to the original registry (for example, ZooKeeper), you cannot query the services in the EDAS background.

1.3.3.3.4.6.1 Query HSF services in the development environment

During development and debugging, if your service is registered and discovered by using the lightweight configuration center, you can query the services provided or called by an application in the background of the lightweight configuration center.

Context

The following assumes that you start the EDAS configuration center on an ECS instance whose IP address is 192.168.1.100. To query HSF services, perform the following steps:

Procedure

1. Open your browser and enter `http://192.168.1.100:8080/` in the address bar to log on to the EDAS configuration center.
2. In the left-side navigation pane, choose **Services**. Set the **service name**, **service group name**, or **IP address** and then click **Search**.



Notice:

After the configuration center is started, the address of the first network interface card (NIC) is the service discovery address by default. If the ECS instance of the developer has multiple NICs, set the `SERVER_IP` variable in the startup script to explicitly bind an address.

3. View the service provider and service caller.
 - Providers tab page
 - In the search bar, enter the IP address, and click **Search** to query the services that are provided by the instance with the IP address you entered.
 - In the search bar, enter the service name or service group, and click Search to query which IP addresses provide the service.
 - Callers tab page
 - In the search bar, enter the IP address, and click **Search** to query the services that are called by the instance with the IP address you entered.
 - In the search bar, enter the service name or service group, and click Search to query which IP addresses call the service.

1.3.3.3.4.6.2 Query HSF services in the online environment

After developed services are packaged and deployed in the EDAS background and you confirm that applications start properly, you can query the corresponding service list in the EDAS background.

Context

To query HSF services in the online environment, perform the following steps:

Procedure

1. Log on to the EDAS console. In the left-side navigation pane, choose **Application Management**.
2. On the Applications page, click a deployed application to go to the Application Details page.
3. In the left-side navigation pane, choose **Services**. The Services page that appears has the **Published Services** and **Services Consumed** tabs. On the **Published Services** tab page, the service provider that you configured for the application is displayed. On the **Services Consumed** tab page, the service consumer that you configured for the application is displayed.



Notice:

If you log on to the console by using a sub-account, check whether the sub-account has the permission to view the **Services** tab page. In the left-side navigation pane, choose **Account Management > All Permissions**. On the Permissions page, click **Application Management**. On the page that appears, check whether the service appears in the list.

What's next

If the target service does not appear in the service list, perform the following steps to troubleshoot the problem:

- Check whether the service configurations in code are correct.
- Check whether the Tomcat process of the service is started and whether an error message is contained in the logs TOMCAT_HOME/logs/catalina.out and \$TOMCAT_HOME/logs/localhost.log. \$DATE_FORMAT.
- Check whether the software version is the latest. To view the software version, choose **Software Version** from the left-side navigation pane on the corresponding service information page. If the software version is not the latest, check whether the corresponding HSF group is created.
- Check whether the host of the ECS instance has special network binding. In normal cases, the online ECS instances are not bound to any hosts.
- Check whether the ECS instance network and ECS security group configuration have restrictions.

1.3.3.3.5 Migrate Dubbo applications to HSF (not recommended)

1.3.3.3.5.1 Precautions for developing Dubbo applications

This topic describes precautions for developing Dubbo applications.

1. A single Dubbo configuration file allows you to define multiple groups of consumers. However, EDAS allows you to specify only one group by using the group property.
2. In Dubbo, service consumers need to specify the version, for example, version = "1.0.0". In EDAS, service consumers may skip the version field, and the version is "1.0.0" by default.
3. Dubbo's RPC framework supports various protocols, such as RMI and Hessian. However, EDAS now only supports the Dubbo protocol, for example, <dubbo:protocol name="dubbo" port="20880">. Otherwise, an error like the following is reported: "com.alibaba.dubbo.config.ServiceConfig service [xx.xx.xxx] contain xx protocol, HSF not supported".
4. The methods for obtaining the RPC context information are different. Dubbo uses the method RpcContext.getContext() to obtain the RPC context information. HSF in EDAS uses the method com.taobao.hsf.util.RequestCtxUtil to obtain the RPC context information. After a Dubbo application is migrated to HSF, if HSF still calls RpcContext.getContext() to obtain the RPC context information, the error "Caused by: java.lang.UnsupportedOperationException: not support getInvocation method in HSF" is reported.

1.3.3.3.5.2 Modify Dubbo application configurations

You can migrate applications developed by using Dubbo to HSF by modifying the application configuration, configuring multiple registries, or converting JAR to WAR.

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](#).

Currently, you can configure Dubbo applications, including service providers and consumers, in either of the two methods in EDAS, namely, creating XML configuration files and adding annotations. This topic describes the two configuration methods with examples

- Configure a service producer in an XML file

```
<? xml version="1.0" encoding="UTF-8"? >  
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
  <dubbo:application name="edas-dubbo-demo-provider" ></dubbo:application>
  <bean id="demoProvider" class="com.alibaba.edas.dubbo.demo.provider.DemoProvider" ></bean>
  <dubbo:registry address="zookeeper://127.0.0.1:2181" ></dubbo:registry>
  <dubbo:protocol name="dubbo" port="20880" threadpool="cached"
    threads="100" ></dubbo:protocol>
  <dubbo:service delay="-1" interface="com.alibaba.edas.dubbo.demo.api.DemoApi"
    ref="demoProvider" version="1.0.0" group="dubbogroup" retries="3" timeout="3000"></dubbo:service>
</beans>
```



Note:

These parameters are optional: threadpool, threads, delay, version, retries, and timeout. Others are required. You can change the parameter locations in the XML file as needed.

- Configure a service consumer in an XML file

```
<? xml version="1.0" encoding="UTF-8"? >
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
  http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
  <dubbo:application name="edas-dubbo-consumer" />
  <dubbo:registry address="zookeeper://127.0.0.1:2181" />
  <dubbo:reference id="demoProviderApi"
    interface="com.alibaba.edas.dubbo.demo.api.DemoApi" version="1.0.0" group="dubbogroup" lazy="true" loadbalance="random">
    <!-- Define a method that does not wait for the return value -->
    <dubbo:method name="sayMsg" async="true" return="false" />
  </dubbo:reference>
  <bean id="demoConsumer" class="com.alibaba.edas.dubbo.demo.consumer.DemoConsumer"
    init-method="reviceMsg">
    <property name="demoApi" ref="demoProviderApi"></property>
  </bean>
</beans>
```



Note:

- These parameters are optional: version, group, lazy, loadbalance, async, and return. Others are required. You can change the parameter locations in the XML file as needed.

- The registry does not take effect in EDAS. All Dubbo services are automatically registered in the EDAS configuration center. You do not have to concern yourself with this.

1.3.3.3.5.3 Convert the format of a package from JAR to WAR

Currently, EDAS only supports web projects in WAR format. Therefore, if your project was released as a JAR package, you must convert it to WAR first.

Context



Note:

This topic uses a Maven project as an example.

Procedure

1. Convert packaging of the pom.xml file from JAR to WAR.
2. Add a web.xml file configuration, if it is unavailable.
3. Configure the web.xml file to load the configuration file.

1.3.3.3.5.4 Run programs

- Right-click to start Tomcat4E and then start a web project. This method is typically used in test environments, where the project directly runs in IDE. It does not require much configuration work. If you have multiple projects, make sure their Tomcat ports are unique. For more information about how to configure Tomcat4E, see [Install Ali-Tomcat and Pandora](#).
- Access the EDAS console to release the WAR project.

1.3.3.3.5.5 Compatibility between Dubbo and HSF

See the following table to check the compatibility between properties in Dubbo and HSF configuration files.

Feature	Dubbo parameter	Compatibil ity	Error message	Supported by EDAS
Timeout	timeout			Yes
Delayed exposure	delay			Yes

Feature	Dubbo parameter	Compatibility	Error message	Supported by EDAS
Thread model	dispatcher="all" threadpool="fixed" threads="100"			Yes
Echo testing				Yes
Delayed connection	lazy="true"	It is enabled by default.		Yes
Local call	protocol="injvm"			Yes
Implicit parameter passing				Yes
Concurrency control	actives="10" /executes="10"	The parameter is visible in the EDAS console. You do not need to configure it.		Yes
Connection control	accepts="10" / connections="2"	The parameter is visible in the EDAS console. You do not need to configure it.		Yes

Feature	Dubbo parameter	Compatibility	Error message	Supported by EDAS
Service degradation		The parameter is visible in the EDAS console. You do not need to configure it.		Yes
Cluster fault tolerance	retries/cluster	Retries are supported.	None	Partially supported
Load balancing	loadbalance	The default value is random.	None	Partially supported
Service grouping	group	Wildcard configuration is not supported.	<u>java.lang.IllegalStateException</u> : The consumption of multiple groups at the same time is not supported by HSF2.	Partially supported
Multi-version	version	Wildcard configuration is not supported.	[HSF-Consumer] cannot find the target address of the service you want to call.	Partially supported
Asynchronous calls	async="true" return="false"	The return parameter is invalid.	None	Partially supported
Check upon startup	check	Check upon startup is disabled by default in EDAS.	None	Check upon startup is disabled by default in EDAS.
Multi-protocol		Only the Dubbo protocol is supported.	com.alibaba.dubbo.config.ServiceConfig [com.alibaba.demo.api.DemoApi] RMI protocol is configured, which is not supported by HSF2.	Partially supported

Feature	Dubbo parameter	Compatibility	Error message	Supported by EDAS
Routing rule		The parameter is visible in the EDAS console. You do not need to configure it.		Yes
Configuration rule		The parameter is visible in the EDAS console. You do not need to configure it.		Yes
Multi-registry				Not supported
Group aggregation	group="aaa, bbb" merger="true"	Error	<u>java.lang.IllegalStateException:</u> The consumption of multiple groups at the same time is not supported by HSF2.	Not supported
Contextual information		Error	Caused by: <u>java.lang.UnsupportedOperationException:</u> not support getInvocation method in hsf2	Not supported

After checking the configuration compatibility, you can start to debug and publish the application according to preceding steps.

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>
      </repository>
    </repositories>
  </profile>
</profiles>
```

```
<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.

1.3.3.4.3.1 Example of HSF application development

This topic provides an example to demonstrate how to develop an HSF application based on Spring Cloud. To develop an HSF application, you must create a service provider and a service consumer.

1.3.3.4.3.1.1 Create a service provider

This topic describes how to create a service provider and provide services through an interface.

Procedure

1. Create a Spring Cloud project named sc-hsf-provider.
2. Add required dependencies to pom.xml.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.8.RELEASE</version>
  <relativePath/>
</parent>
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hsf</artifactId>
    <version>1.1</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-pandora</artifactId>
    <version>1.2</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>Dalston.SR4</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Although the HSF service framework is independent of the web environment, web-related features are required when EDAS is used to manage the lifecycle of applications. Therefore, you must add the spring-boot-starter-web dependency.

If you do not want to configure the parent of the project as spring-boot-starter-parent, you can add dependencyManagement and set scope=import as follows to manage dependency versions:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
      <version>1.5.8.RELEASE</version>
```

```
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>
```

3. Define a service interface, and create an interface class `com.aliware.edas.EchoService`.

```
public interface EchoService {
    String echo(String string);
}
```

The HSF service framework enables service communication over interfaces. When an interface is defined, providers implement and release specific services by using this interface, and consumers subscribe to and consume services also by using this interface.

The interface `com.aliware.edas.EchoService` provides an `echo` method, which also means that the service `com.aliware.edas.EchoService` provides an `echo` method.

4. Add the implementation class `EchoServiceImpl` of the service provider, and publish the service by using annotations.

```
@HSFProvider(serviceInterface = EchoService.class, serviceVersion = "1.0.0")
public class EchoServiceImpl implements EchoService {
    @Override
    public String echo(String string) {
        return string;
    }
}
```

In addition to the interface specified by `serviceInterface`, HSF also requires the version specified by `serviceVersion` to uniquely identify a service. In this example, the `serviceVersion` property in the `HSFProvider` annotation is set to "1.0.0". Then, the service you want to publish can be identified by the combination of `serviceInterface` `com.aliware.edas.EchoService` and `serviceVersion` `1.0.0`.

The configuration in the `HSFProvider` annotation has the highest priority. If it is not configured in the `HSFProvider` annotation, the global configuration of these properties is searched in the `resources/application.properties` file when the service is published. If neither is configured, the default values in the `HSFProvider` annotation are used.

5. Configure the application name and the listener port number in the `application.properties` file in the resources directory.

```
spring.application.name=hsf-provider
server.port=18081
spring.hsf.version=1.0.0
```

```
spring.hsf.timeout=3000
```

Best practices: We recommend that you configure both the **service version** and **service timeout** in the application.properties file .

6. Add the main function handler for starting the service.

```
@SpringBootApplication
public class HSFProviderApplication {
    public static void main(String[] args) {
        // Start Pandora Boot for loading the Pandora container.
        PandoraBootstrap.run(args);
        SpringApplication.run(ServerApplication.class, args);
        // Indicate that the service has been started, and a thread waiting time is set. This
        prevents the container from exiting due to users who exit after running the service
        code.
        PandoraBootstrap.markStartupAndWait();
    }
}
```

1.3.3.4.3.1.2 Create a service consumer

In this example, we create a service consumer that calls the service provider by using the interface provided by HSFProvider.

Procedure

1. Create a Spring Cloud project named sc-hsf-consumer.
2. Add required dependencies to pom.xml.

The Maven dependencies for HSFConsumer and HSFProvider are exactly the same.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.8.RELEASE</version>
  <relativePath/>
</parent>
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hsf</artifactId>
    <version>1.1</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-pandora</artifactId>
    <version>1.2</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>Dalston.SR4</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>
```

3. Copy the service interface `com.aliware.edas.EchoService`, including the package name, published by the service provider to a local instance.

```
public interface EchoService {
    String echo(String string);
}
```

4. Use annotations to inject the **service consumer** instance to the Spring context.

```
@Configuration
public class HsfConfig {
    @HSFConsumer(clientTimeout = 3000, serviceVersion = "1.0.0")
    private EchoService echoService;
}
```

Best practices: Configure `@HSFConsumer` once in the Config class, and then inject and use it in multiple steps through `@Autowired`. Usually, `HSFConsumer` is used in multiple places, but you do not have to mark each place where it is used with `@HSFConsumer`. You can write a unified Config class and directly inject it wherever it is needed through `@Autowired`.

5. To facilitate testing, an HTTP method of `/hsf-echo/*` is exposed through `SimpleController`. Calls to the HSF service provider are internally implemented in the API `/hsf-echo/*`.

```
@RestController
public class SimpleController {
    @Autowired
    private EchoService echoService;
    @RequestMapping(value = "/hsf-echo/{str}", method = RequestMethod.GET)
    public String echo(@PathVariable String str) {
        return echoService.echo(str);
    }
}
```

6. Configure the application name and the listener port number in the `application.properties` file in the resources directory.

```
spring.application.name=hsf-consumer
server.port=18082
spring.hsf.version=1.0.0
spring.hsf.timeout=1000
```

Best practices: We recommend that you configure both the **service version** and **service timeout** in the `application.properties` file .

7. Add the main function handler for starting the service.

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

1.3.3.4.3.2 Advanced HSF features

This topic describes how to use Pandora Boot to develop advanced HSF features, such as unit testing and asynchronous calls.

Context

Download the demo source code [sc-hsf-provider](#) and [sc-hsf-consumer](#).

Procedure

Unit testing

The implementation of spring-cloud-starter-hsf depends on Pandora Boot, and the unit testing of Pandora Boot is enabled through PandoraBootRunner and seamlessly integrated with SpringJUnit4ClassRunner.

1. Add the spring-boot-starter-test dependency in Maven.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

2. Compile the test code.

```
@RunWith(PandoraBootRunner.class)
@DelegateTo(SpringJUnit4ClassRunner.class)
// Add the test class. In this case, both the Spring Boot startup class and service test
// class are required.
@SpringBootTest(classes = {HSFProviderApplication.class, EchoServiceTest.class })
@Component
public class EchoServiceTest {
    /**
     * If you are using @HSFConsumer, you must add the service class to @SpringBoot
     * Test and use it to inject objects to prevent abnormal class conversion during generic
     * calls.
     */
    @HSFConsumer(generic = true)
    EchoService echoService;
    //Common calls
    @Test
    public void testInvoke() {
        TestCase.assertEquals("hello world", echoService.echo("hello world"));
    }
    //Generic calls
}
```

```

@Test
public void testGenericInvoke() {
    GenericService service = (GenericService) echoService;
    Object result = service.$invoke("echo", new String[] {"java.lang.String"}, new
Object[] {"hello world"});
    TestCase.assertEquals("hello world", result);
}
//Return the value Mock.
@Test
public void testMock() {
    EchoService mock = Mockito.mock(EchoService.class, AdditionalAnswers.
delegatesTo(echoService));
    Mockito.when(mock.echo("")).thenReturn("beta");
    TestCase.assertEquals("beta", mock.echo(""));
}
}

```

Asynchronous calls

HSF enables two types of asynchronous calls, Future and Callback.

3. To demonstrate asynchronous calls, you have to publish a new service:

com.aliware.edas.async.AsyncEchoService.

```

public interface AsyncEchoService {
    String future(String string);
    String callback(String string);
}

```

4. The service provider implements AsyncEchoService and uses annotations to publish it.

```

@HSFProvider(serviceInterface = AsyncEchoService.class, serviceVersion = "1.0.0")
public class AsyncEchoServiceImpl implements AsyncEchoService {
    @Override
    public String future(String string) {
        return string;
    }
    @Override
    public String callback(String string) {
        return string;
    }
}

```

As shown above, it makes no difference whether you choose common release or service provider-based release. The subsequent configurations and application startup processes are all the same. For more information, see the section about how to create a service provider in [HSF development](#).



Note:

The logic of asynchronous calls is modified on the service consumer rather than the service provider.

Future

5. To enable Future-type asynchronous calls for the service consumer, use annotations to inject service consumer instances into the Spring context, and configure the method name of asynchronous calls in futureMethods of @HSFConsumer.

In this example, the Future method of com.aliware.edas.async.AsyncEchoService is marked as Future-type asynchronous calls.

```
@Configuration
public class HsfConfig {
    @HSFConsumer(serviceVersion = "1.0.0", futureMethods = "future")
    private AsyncEchoService asyncEchoService;
}
```

6. After the method is marked as Future-type asynchronous calls, the actual return value of the method during synchronous execution is null, and the call result must be obtained through HSFResponseFuture.

TestAsyncController is used for demonstration. The sample code is as follows:

```
@RestController
public class TestAsyncController {
    @Autowired
    private AsyncEchoService asyncEchoService;
    @RequestMapping(value = "/hsf-future/{str}", method = RequestMethod.GET)
    public String testFuture(@PathVariable String str) {
        String str1 = asyncEchoService.future(str);
        String str2;
        try {
            HSFFuture hsfFuture = HSFResponseFuture.getFuture();
            str2 = (String) hsfFuture.getResponse(3000);
        } catch (Throwable t) {
            t.printStackTrace();
            str2 = "future-exception";
        }
        return str1 + " " + str2;
    }
}
```

Call /hsf-future/123. The str1 value is null, and the str2 value is 123, which is the actual return value.

7. If a series of operation return values are needed for service processing, see the following call method:

```
@RequestMapping(value = "/hsf-future-list/{str}", method = RequestMethod.GET)
public String testFutureList(@PathVariable String str) {
    try {
        int num = Integer.parseInt(str);
        List<String> params = new ArrayList<String>();
        for (int i = 1; i <= num; i++) {
            params.add(i + "");
        }
        List<HSFFuture> hsfFutures = new ArrayList<HSFFuture>();
        for (String param : params) {
            asyncEchoService.future(param);
        }
    }
}
```

```
hsfFutures.add(HSFResponseFuture.getFuture());
}
ArrayList<String> results = new ArrayList<String>();
for (HSFFuture hsfFuture : hsfFutures) {
    results.add((String) hsfFuture.getResponse(3000));
}
return Arrays.toString(results.toArray());
} catch (Throwable t) {
    return "exception";
}
}
```

Callback

8. To enable Callback-type asynchronous calls for the service consumer, create a class to implement HSFResponseCallback and use @Async for configuration.

```
@AsyncOn(interfaceName = AsyncEchoService.class,methodName = "callback")
public class AsyncEchoResponseListener implements HSFResponseCallback{
    @Override
    public void onAppException(Throwable t) {
        t.printStackTrace();
    }
    @Override
    public void onAppResponse(Object appResponse) {
        System.out.println(appResponse);
    }
    @Override
    public void onHSFException(HSFException hsfEx) {
        hsfEx.printStackTrace();
    }
}
```

AsyncEchoResponseListener implements HSFResponseCallback. Set interfaceName to AsyncEchoService.class and methodName to callback in @Async.

The Callback method of com.aliware.edas.async.AsyncEchoService is marked as Callback-type asynchronous calls.

9. Similarly, TestAsyncController is used for demonstration. The sample code is as follows:

```
@RequestMapping(value = "/hsf-callback/{str}", method = RequestMethod.GET)
public String testCallback(@PathVariable String str) {
    String timestamp = System.currentTimeMillis() + "";
    String str1 = asyncEchoService.callback(str);
    return str1 + " " + timestamp;
}
```

After the service consumer sets the Callback method to Callback-type asynchronous calls , the synchronous execution return value is actually null.

After the result is returned, HSF calls the method in AsyncEchoResponseListener, and the actual return value of the call can be obtained by using the onAppResponse method.

10. Use `CallbackInvocationContext` to transmit the contextual information of the call to the Callback method.

The call sample code is as follows:

```
CallbackInvocationContext.setContext(timestamp);
String str1 = asyncEchoService.callback(str);
CallbackInvocationContext.setContext(null);
```

The sample code of `AsyncEchoResponseListener` is as follows:

```
@Override
public void onAppResponse(Object appResponse) {
    Object timestamp = CallbackInvocationContext.getContext();
    System.out.println(timestamp + " " + appResponse);
}
```

The output in the console is `1513068791916 123`. This indicates that the `onAppResponse` method of `AsyncEchoResponseListener` has used `CallbackInvocationContext` to receive the timestamp content transmitted before the call.

1.3.3.4.3.3 Local debugging

After an HSF application is developed, you need to debug its code locally before publishing it.

Procedure

1. Start the lightweight configuration center.

The lightweight configuration center must be started for local code development and debugging, which includes a lightweight version of EDAS service registry for service providers. For more information about the lightweight configuration center, see [Lightweight configuration center](#).

2. Start the application.

You can locally start the application in either of the following two methods:

- Start the application in IDE

To start the application in IDE, configure the startup parameter `-Djmenv.tbsite.net=${IP}` in VM options and directly start the application by calling the main method. `{IP}` is the address of the instance on which the lightweight configuration center is started.

For example, if the lightweight configuration center is started on the current instance, **{IP}** is 127.0.0.1.

If you do not want to configure JVM parameters, you can modify the hosts file and bind **jmenv.tbsite.net** to the IP address of the instance on which the configuration center is started. For more information, see [Lightweight configuration center](#).

- Start the application by using FatJar
 - a. Add a FatJar packaging plug-in.

To package the pandora-boot project into a FatJar file by using Maven, add the following plug-in in pom.xml.

**Note:**

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

```
<build>
  <plugin>
    <groupId>com.taobao.pandora</groupId>
    <artifactId>pandora-boot-maven-plugin</artifactId>
    <version>2.1.7.8</version>
    <executions>
      <execution>
        <phase>package</phase>
        <goals>
          <goal>repackage</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
</build>
```

3. After adding the plug-in, run the mvn clean package command in the home directory of the project to create a package. The created FatJar file is placed in the target directory.
4. Start the application by running the following Java command:

```
java -Djmenv.tbsite.net=127.0.0.1 -Dpandora.location=/Users/${username}/.m2/
repository/com/taobao/pandora/taobao-hsf.sar/dev-SNAPSHOT/taobao-hsf.sar-dev
-SNAPSHOT.jar -jar sc-hsf-provider-0.0.1-SNAPSHOT.jar
```

**Note:**

The path specified by **-Dpandora.location** must be a full path placed before sc-hsf-provider-0.0.1-SNAPSHOT.jar.

Result

Start the service, call the service, and check whether the service can be called.

1.3.3.4.3.4 Deploy applications to EDAS

After the code for service discovery and calling is debugged on a local instance, it can be published to EDAS.

Procedure

1. When creating the applications, choose the container of the latest version.

For more information about how to create applications, see **Application management** in the User Guide.

2. In the directory of the project, run the mvn clean package command and pack it into a FatJar package. Add the FatJar package.

3. Upload the FatJar package in the target directory to deploy the application to EDAS.

For more information about how to deploy applications, see **Application management** in the User Guide.

1.3.3.4.4 Develop RESTful applications (not recommended)

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

1.3.3.4.4.1 Terms

This topic defines and explains terms you may encounter when developing applications based on Spring Cloud.

Pandora	Pandora is a lightweight container isolation service used in Alibaba, which is utilized in Aliware to isolate and load classes. Pandora is a required dependency for Spring Cloud for Aliware.
VIPServer	VIPServer is the server used for service registry for RESTful applications in EDAS, whose corresponding client is VIPClient.
Lightweight configuration center	The EDAS lightweight configuration center can run on a local instance and provides service discovery and configuration management features.
FatJar	FatJar (also known as the executable JARs) is an archive of compiled classes and dependent JARs for running code. You can run the <code>java -jar</code> command to use this program.
EagleEye	EagleEye is a distributed tracing system for Alibaba, developed based on Google Dapper.

EagleEye EagleEye processes trillions of distributed trace data each day, and implements instrumentation in various network calls through collection and analysis to acquire call relationships between systems in the same request. This helps sort out the application request portals and service call sources and dependencies, analyze system call bottlenecks, estimate the link capacities, and quickly locate exceptions.

HSF

HSF is a distributed service framework for Alibaba. HSF provides support for developers on distributed applications and unified publish and call methods . In this way, developers can easily develop distributed applications and provide or use public functional modules, without considering the technical details in distributed systems, such as remote communication, performance loss, call transparency, synchronous calls, and asynchronous calls.

1.3.3.4.4.2 Service registration and discovery

This topic describes how to implement service registration and discovery by using Spring Cloud.

1.3.3.4.4.2.1 Preparation

Before using Spring Cloud to implement service registration and discovery, you need to make the relevant preparations.

Configure the EDAS local repository path in Maven

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. For more information about how to configure Maven and the Maven local repository path, see [official documentation](#).

The local repository path of EDAS is <http://edas-public.oss-cn-hangzhou.aliyuncs.com/repository>.



Note:

You must use Maven 3.x or later, and add the local repository path of EDAS in the Maven configuration file settings.xml. [Download](#) the sample file.

Start the lightweight configuration center

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. For more information about the lightweight configuration center and its download URLs, see [Lightweight configuration center](#).

1.3.3.4.4.2.2 Implement service registration and discovery

This topic uses a simple example to describe how to discover and call services.

Context

The following two roles are involved in this process:

- Service provider: It provides a simple echo service and registers itself with the service registry.
- Service consumer: It calls services through RestTemplate, AsyncRestTemplate, and FeignClient.

Procedure

Service providers implement services and register them with the service registry.

1. Create a Spring Cloud project named sc-vip-client.
2. Add required dependencies to pom.xml.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.8.RELEASE</version>
  <relativePath/>
</parent>
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-vipclient</artifactId>
    <version>1.1</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-pandora</artifactId>
    <version>1.2</version>
  </dependency>
</dependencies>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Dalston.SR4</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

```
</dependencyManagement>
```

If you do not want to configure the parent of the project as spring-boot-starter-parent, you can add dependencyManagement and set scope=import as follows to manage dependencies:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
      <version>1.5.8.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

3. Add the code of the service provider, in which the annotation `@EnableDiscoveryClient` indicates that service registration and discovery are enabled for the application.

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

4. Create an EchoController to provide simple echo services.

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

5. Configure the application name and the listener port number in the application.properties file in resources.

```
spring.application.name=service-provider
server.port=18081
```

Service consumers call services through RestTemplate, AsyncRestTemplate, and FeignClient.

6. Create a Spring Cloud project named sc-vip-client.
7. Add required dependencies to pom.xml.

```
<parent>
  <groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-parent</artifactId>
<version>1.5.8.RELEASE</version>
<relativePath/>
</parent>
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-vipclient</artifactId>
    <version>1.1</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-pandora</artifactId>
    <version>1.2</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
  </dependency>
</dependencies>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Dalston.SR4</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

To demonstrate the use of FeignClient, an additional dependency of spring-cloud-starter-feign is added to the pom.xml file compared to that of the service provider.

8. Different from the service provider, in addition to service enabling and registration, two more configurations must be added to the service consumer for using RestTemplate, AsyncRestTemplate, and FeignClient.

a) Add the annotation @LoadBalanced to combine RestTemplate, AsyncRestTemplate, and service discovery.

b) Activate FeignClient by using the annotation @EnableFeignClients.

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class ConsumerApplication {
    @LoadBalanced
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
    @LoadBalanced
    @Bean
    public AsyncRestTemplate asyncRestTemplate(){
        return new AsyncRestTemplate();
    }
    public static void main(String[] args) {
        PandoraBootstrap.run(args);
    }
}
```

```

SpringApplication.run(ConsumerApplication.class, args);
PandoraBootstrap.markStartupAndWait();
}
}

```

9. Complete the configuration of FeignClient of EchoService before using it.

Configure the service name and the HTTP request corresponding to the method. The service name is service-provider configured in the sc-vip-server project. The code is as follows:

```

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

```

10. Create a Controller for call test.

- /echo-rest/* calls the service from the service provider through RestTemplate.
- /echo-async-rest/* calls the service from the service provider through AsyncRestTemplate.
- /echo-feign/* calls the service from the service provider through FeignClient.

```

@RestController
public class Controller {
    @Autowired
    private RestTemplate restTemplate;
    @Autowired
    private AsyncRestTemplate asyncRestTemplate;
    @Autowired
    private EchoService echoService;
    @RequestMapping(value = "/echo-rest/{str}", method = RequestMethod.GET)
    public String rest(@PathVariable String str) {
        return restTemplate.getForObject("http://service-provider/echo/" + str, String.class);
    }
    @RequestMapping(value = "/echo-async-rest/{str}", method = RequestMethod.GET)
    public String asyncRest(@PathVariable String str) throws Exception {
        ListenableFuture<ResponseEntity<String>> future = asyncRestTemplate.getForEntity("http://service-provider/echo/" + str, String.class);
        return future.get().getBody();
    }
    @RequestMapping(value = "/echo-feign/{str}", method = RequestMethod.GET)
    public String feign(@PathVariable String str) {
        return echoService.echo(str);
    }
}

```

11. Configure the application name and the listener port number.

```

spring.application.name=service-consumer
server.port=18082

```

1.3.3.4.4.2.3 Local debugging

After service discovery and calling are enabled, you need to debug the code on a local instance before deploying it to EDAS.

Procedure

1. Start the lightweight configuration center.

The lightweight configuration center must be started for local code development and debugging, which includes a lightweight version of EDAS service registry for service providers. For more information about the lightweight configuration center, see [Lightweight configuration center](#).

2. Start the application.

You can locally start the application in either of the following two methods:

- Start the application in IDE

To start the application in IDE, configure the startup parameter `-Dvipserver.server.port=8080` in VM options, and directly start the application by calling the main method.

If your lightweight configuration center and application are deployed on different instances, perform hosts binding. For more information, see [Lightweight configuration center](#).

- Start the application by using FatJar

a. Add a FatJar packaging plug-in.

To package the pandora-boot project into a FatJar file by using Maven, add the following plug-in in pom.xml. To avoid conflicts with other packaging plug-ins, do not add any other FatJar plug-ins to the plugin field in build.

```
<build>
  <plugin>
    <groupId>com.taobao.pandora</groupId>
    <artifactId>pandora-boot-maven-plugin</artifactId>
    <version>2.1.7.8</version>
    <executions>
      <execution>
        <phase>package</phase>
        <goals>
          <goal>repackage</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
```

```
</build>
```

- b. After adding the plug-in, run the `mvn clean package` command in the home directory of the project to create a package. The created FatJar file is placed in the target directory.
- c. Start the application by running the following Java command:

```
java -Dvipserver.server.port=8080 -Dpandora.location=/Users/{username}/.m2/repository/com/taobao/pandora/taobao-hsf.sar/dev-SNAPSHOT/taobao-hsf.sar-dev-SNAPSHOT.jar -jar sc-vip-server-0.0.1-SNAPSHOT.jar
```



Note:

The value of `-Dpandora.location` must be a full path placed before `sc-vip-server-0.0.1-SNAPSHOT.jar`.

3. Start the service, run the `curl` command to call the consumer and provider respectively. Each call is successful.

You can also paste the corresponding URL in the address bar of your browser and observe response for verification.

1.3.3.4.4.2.4 Deploy applications to EDAS

After the code for service discovery and calling is debugged on a local instance, it can be published to EDAS.

Procedure

1. When creating the applications, choose the container of the latest version.

For more information about how to create applications, see **Application management** in the User Guide.

2. In the directory of the project, run the `mvn clean package` command and pack it into a FatJar package. Add the FatJar package.

3. Upload the FatJar package in the target directory to deploy the application to EDAS.

For more information about how to deploy applications, see **Application management** in the User Guide.

1.3.3.4.4.2.5 Migrate from Eureka

An application connected with the Eureka service registry can be connected with the EDAS service registry in two steps.

Procedure

1. Modify the source code.

Add two lines in the main function. The original content of the main function is as follows:

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

The modified content of the main function is as follows:

```
public static void main(String[] args) {  
    PandoraBootstrap.run(args);  
    SpringApplication.run(ServerApplication.class, args);  
    PandoraBootstrap.markStartupAndWait();  
}
```

2. Modify the dependency of pom.xml.

Replace spring-cloud-starter-eureka with spring-cloud-starter-vipclient.

Before the replacement:

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

After the replacement:

```
<dependency>  
    <groupId>org.springframework.cloud</groupId>  
    <artifactId>spring-cloud-starter-vipclient</artifactId>  
    <version>1.1</version>  
</dependency>
```

1.3.3.4.4.2.6 FAQ

This topic describes possible problems during service registration and discovery based on Spring Cloud and their solutions.

1. What can I do if I cannot enable service discovery for AsyncRestTemplate?

AsyncRestTemplate provides service discovery only in Spring Cloud Dalston and later versions.

2. What can I do when FatJar packaging plug-ins conflict with other plug-ins?

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

3. Can taobao-hsf.sar be included during packaging?

Yes, but this is not recommended.

You can modify pandora-boot-maven-plugin and set excludeSar to false to automatically include taobao-hsf.sar during packaging.

```
<plugin>
<groupId>com.taobao.pandora</groupId>
<artifactId>pandora-boot-maven-plugin</artifactId>
<version>2.1.7.8</version>
<configuration>
<excludeSar>>false</excludeSar>
</configuration>
<executions>
<execution>
<phase>package</phase>
<goals>
<goal>repackage</goal>
</goals>
</execution>
</executions>
</plugin>
```

In this way, the package can be started even if the Pandora path is not configured.

```
java -jar -Dvipserver.server.port=8080 sc-vip-server-0.0.1-SNAPSHOT.jar
```

Restore the configuration to exclude the SAR package before deploying an application in EDAS.

1.3.3.4.4.3 Distributed tracing

To reduce development costs and improve development efficiency, EDAS provides EagleEye, a component for distributed tracing. After EagleEye instrumentation is implemented in the code, you can use the distributed tracing function of EDAS, without considering other processes such as log collection, analysis, and storage. This topic describes how to enable the distributed tracing function for your applications.

1.3.3.4.4.3.1 Access EagleEye

Before enabling distributed tracing, you must access EagleEye first.

Prerequisites

Before accessing EagleEye, add the local repository path of EDAS in the Maven configuration file.

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. For more information about how to configure Maven and the Maven local repository path, see [official documentation](#). [Download](#) the sample file.

The local repository path of EDAS is <http://edas-public.oss-cn-hangzhou.aliyuncs.com/repository>.



Note:

Maven 3.x or later is required.

Procedure

1. Add the following public dependency configuration in the pom.xml file:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-eagleeye</artifactId>
  <version>1.1</version>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-pandora</artifactId>
  <version>1.2</version>
</dependency>
```

2. Add two lines of code to the main function.

The original content of the main function is as follows:

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

The modified content of the main function is as follows:

```
public static void main(String[] args) {
    PandoraBootstrap.run(args);
    SpringApplication.run(ServerApplication.class, args);
    PandoraBootstrap.markStartupAndWait();
}
```

3. Add a FatJar packaging plug-in.

To package the pandora-boot project into FatJar by using Maven, add the following plug-ins in pom.xml.

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>2.1.7.8</version>
      <executions>
        <execution>
          <phase>package</phase>
```

```
<goals>
  <goal>repackage</goal>
</goals>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

Result

After finishing the preceding steps, you can use the EDAS distributed tracing function without setting up any collection and analysis systems.

1.3.3.4.4.3.2 Distributed tracing example

This topic provides an example to describe how to perform distributed tracing for applications.

Context

To demonstrate how to perform distributed tracing, two code demos [service1](#) and [service2](#) are used in this example.

service1 provides portals to three demonstration scenarios.

- /rest/ok for normal calls
- /rest/delay for calls with a large delay
- /rest/error for calls with errors

Procedure

Deploy applications

The collection and analysis functions of EagleEye are set up in EDAS. To demonstrate the trace query function, deploy the two applications service1 and service2 in EDAS.

1. When creating the applications, choose the container of the latest version.

For more information about how to create applications, see **Application management** in the User Guide .

2. Add a FatJar packaging plug-in, and run the mvn clean package command in the directory of the project to create a FatJar package.

3. Upload the FatJar application in the target directory to deploy the application to EDAS.

For more information about how to deploy applications, see **Application management** in the User Guide .

Call EagleEye

To view the trace information after the application is deployed, you have to know the call methods corresponding to the portals of three demonstration scenarios of service1.

4. You can run the `curl http://{ip:$port}/rest/ok` command. You can also use tools such as Postman or directly call the methods in your browser.

To observe the response, we recommend that you call the methods in script mode multiple times.

View traces

5. Log on to the EDAS console and access the application you deployed.
6. In the left-side navigation pane on the Application Details page, choose **Application Monitoring > Service Monitoring**.
7. On the Service Monitoring page, click the **RPC Services Provided** tab and then click **View Trace**.

For more information about how to use this function, see Service Monitoring.

For more information about service monitoring, see **Application management** in the User Guide.

- Trace details of a normal call

You can see how many times the service is called and that it takes 2 ms, 1 ms, and 0 ms respectively to call the service in steps 1, 2, and 3.

- Trace details of a call with a large delay

You can see that 453 ms, 353 ms, and 151 ms are respectively spent on calls with delay 1, delay 2, and delay 3.

By resting the pointer on the call with delay 3, you can see more details about the trace. The provider spends 150 ms processing the request, and the consumer receives a response 1 ms after the provider finishes processing the request.

- Trace details of a call with an error

You can see that the request with errors is `/rest/error3`, which helps you locate the problem.

Instrumentation of EagleEye for RestTemplate, AsyncRestTemplate, and FeignClient is demonstrated separately in `/echo-rest/{str}`, `/echo-async-rest/{str}`, and `/echo-feign/{str}` of service1. You can view the trace information in the same way after calling them.

1.3.3.4.4.3.3 FAQ

This topic describes possible problems during distributed tracing and their solutions.

Instrumentation

Currently, EagleEye of EDAS supports automatic tracking for requests of RestTemplate, AsyncRestTemplate, and FeignClient. **We will provide instrumentation of more components.**

AsyncRestTemplate

As AsyncRestTemplate must modify instrumentation during class instantiation, injection of the object eagleEyeAsyncRestTemplate, which supports service discovery by default, is required to enable distributed tracing.

```
@Autowired  
private AsyncRestTemplate eagleEyeAsyncRestTemplate;
```

FatJar packaging plug-in

To use Maven to package the pandora-boot project into FatJar, add pandora-boot-maven-plugin in pom.xml. To avoid conflicts with other packaging plug-ins, do not add configurations of any other FatJar plug-ins to the plugin field in build.

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 Deploy applications in Container Service Kubernetes clusters by using images

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 EDAS console, and then create and deploy applications in the cluster.

Prerequisites

Prerequisites

- Your Alibaba Cloud account has activated EDAS and Container Service for Kubernetes.
- You have granted the required permissions for Container Service for Kubernetes.
- You have prepared the application images (the Container Service Kubernetes cluster).

Context

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

Procedure

Step 1: Create a Container Service Kubernetes cluster.

1. Log on to the Container Service for Kubernetes console.
2. In the left-side navigation pane, choose **Clusters**. On the Cluster List page, click **Create Kubernetes Cluster**.

Container Service allows you to create three types of clusters, namely, **Kubernetes clusters**, **managed Kubernetes clusters**, and **multi-zone Kubernetes clusters**.

- Create Kubernetes clusters: Three of the instances that you buy and add must be used as master nodes. Applications cannot be deployed on these three instances. You can only deploy applications on other instances (workers).
- Create managed Kubernetes clusters: All the instances that you buy and add are workers and can be used to deploy applications.
- Create multi-zone Kubernetes clusters: Unlike Kubernetes clusters, in multi-zone Kubernetes clusters, nodes are deployed in different zones. When one zone becomes unavailable, services fail over to nodes in other zones. Three of the instances that you buy and add must be used as master nodes. Applications cannot be deployed on these three instances. You can only deploy applications on other instances (workers).

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

3. Log on to the EDAS console.
4. In the left-side navigation pane, choose **Resource Management > Clusters**.
5. 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 applications in the Container Service Kubernetes cluster.

6. In the left-side navigation pane, choose **Application Management**.

7. On the Applications page, set **Region** and **Namespace**, and then click **Create Application** in the upper-right corner.
8. 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.
9. Go to the **Application Configuration** page and configure an image. By default, **Image** is selected for **Deployment Method**. Select an image in **My Image**.
10. Set pods.
 - a) Set **Total 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 Total Pods to a maximum value of 50.
 - b) 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.
 - c) 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.

11.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.

12.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. The [billing method](#) of NAS is Pay-As-You-Go. Ensure that your account has a sufficient balance or is billed in Pay-As-You-Go mode.

- **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.

13.Set local storage.

You can map part of the file system of the host to the container as needed. Before using this function, read [hostpath](#) and consider the rationality of the solution.

Table 1-3: File types

Parameter	Value	Description
Default	Null string	The file is directly mounted , without checking the file type.
(New) File directory	DirectoryOrCreate	The file directory. A new directory is created if it does not exist.
File Directory	Directory	The file directory. Container startup fails if it does not exist.
(New) File	FileOrCreate	The file. A new file is created if it does not exist.
File	File	The file. Container startup fails if it does not exist.

Parameter	Value	Description
Socket	Socket	The standard UNIX Socket file. Container startup fails if it does not exist.
CharDevice	CharDevice	The character device file. Container startup fails if it does not exist.
BlockDevice	BlockDevice	The block storage device file. Container startup fails if it does not exist.

**Note:**

You do not need to concern yourself with the Value column in this step. However, the Value column may be used by APIs after the application is created.

14.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](#)

15. Then, click **Create**.

Result

It takes several minutes to create an application. You can trace the creation process based on the change record and change details. Kubernetes applications do not need to be deployed because they are immediately deployed upon creation. After the application is created, go to the Application Details page to check whether the pod status in the Instance Information section 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-beijing app: 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:

- a. Log on to the EDAS console.
 - b. 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**.
 - c. 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.

Type	Parameter	Required	Note
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.
	batch_wait_time	No	The waiting time (in minutes) between deployment batches . The default value is 0.

Type	Parameter	Required	Note
	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.
oss	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
	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-5f6a5455**** 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-beijing app: 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-beijing app: app_id: eb20dc8a-e6ee-4f6d-a36f-5f6a5455**** oss:  
region_id: cn-beijing 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> <accessKeySecret>1234567890</accessKeySecret></configurat  
ion></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 CLI to deploy applications in EDAS

The command line interface (CLI) was the most widely used type of user interface before graphical user interfaces (GUIs) become popular. CLIs usually do not support the use of a mouse. Instead, you enter instructions through 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 performing the steps in this tutorial, you must have done the following: [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 EDAS API. Make sure that you have activated EDAS and know how to use SDKs to call operations in EDAS. For more information about how to call operations, see [Developer Guide](#) . You can use Alibaba Cloud CLI to deploy all applications developed based on the HSF, Dubbo, or Spring Cloud framework in ECS or Swarm clusters in EDAS.

Procedure

1. Install 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:

- [MacOS](#)
- [Linux](#)
- [Windows \(64-bit\)](#)

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

2. Configure CLI

Before using Alibaba Cloud CLI, run the `aliyun configure` command to configure the AccessKey, region, and language for calling your Alibaba Cloud account.

You can create and view your AccessKey on the [Security Management](#) page, or obtain the AccessKey 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-hangzhouDefault output format [json]: jsonDefault Language [zh]: zh
```

3. Use CLI to create applications

Run the following script to create an application:

```
#!/bin/bash # Region for deployment REGION="cn-beijing" # ID of the ECS instance ECS_ID="i-2z*****b6" # ID of the VPC where the ECS instance is located VPC_ID="vpc-t*****c" # Name of a namespace (which is automatically created if it does not exists) NAMESPACE="myNamespace" # 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/null for i in `seq 300` do OUT=`aliyun edas ListClusterMembers --ClusterId $CLUSTER_ID | grep Eculd` && break sleep 1 done ECU_ID=`echo $OUT | sed -E 's/. *"Eculd": "([a-z0-9-]*)".*/\1/g` # Step 4: Create an application. APP_ID=`aliyun edas InsertApplication --ApplicationName $APP_NAME --BuildPackId 51 --EculInfo $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 CLI to deploy applications

Run the following code to use Alibaba Cloud CLI to deploy an application:

```
#!/bin/bash # ID of the application to be deployed (which must be created in advance) APP_ID="87a6*****4d1" # ID of the group to which the application belongs GROUP_ID="54b*****f27" # Name of the OSS bucket for uploading (the bucket must support public read) OSS_BUCKET="eda*****mo" # Installation package file (created 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:

- a) Log on to the EDAS console.
- b) 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**.
- c) 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.
 - a. In the toolbar, click the drop-down arrow of the Alibaba Cloud Toolkit icon. In the drop-down list, select **Alibaba Cloud Preference....**
 - b. In the **Preference (Filtered)** dialog box, choose **Accounts** from the left-side navigation pane.
 - c. 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.
 - a. In the **Preference (Filtered)** dialog box, choose **Appearance & Behavior > Endpoint** from the left-side navigation pane.
 - b. 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.

- a. 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.
- b. 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.

- a. On IntelliJ IDEA, click the Cloud Toolkit icon and select **Preferences** from the drop-down list.
- b. In the **Preferences** dialog box, choose **Appearance & Behavior > Endpoint** from the left-side navigation pane.
- c. 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.

a. 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.

b. 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 clouds

EDAS provides complete solutions for scaling, networking, and central management in hybrid clouds, allowing you to deploy applications in hybrid cloud environments. You can connect instances from Alibaba Cloud, on-premises IDCs, and other cloud service providers (CSPs) through leased lines, and add the instances to hybrid cloud (non-Alibaba Cloud) ECS clusters in EDAS. Then, you can deploy and manage HSF, Dubbo, and Spring Cloud applications in the EDAS console in a unified manner. 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 a physical connection to connect your on-premises IDC to Alibaba Cloud VPC.

- The instances in your on-premises IDC 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 or problems:

- The Alibaba Cloud traffic has a certain degree of volatility and you may face traffic spikes in special scenarios, such as flash sales. You can predict the traffic volumes in such scenarios, but deviations may exist. Since you need to buy ECS instances in advance, it is hard to control the number of needed ECS instances. Knowing when to add ECS instances is also a challenge.
- Some core business systems have high security requirements and you may want to deploy such applications in your own IDC. However, you cannot deploy and manage applications in different environments in a unified manner because instances from Alibaba Cloud, on-premises IDCs, and other CSPs cannot communicate with each other.
- Considering your business needs and availability requirements, you may want to deploy your applications on instances from multiple CSPs, that is, in multi-cloud mode. In this mode, manual processing is required because you cannot centrally manage these applications. This often leads to misoperations.
- Connect Alibaba Cloud to on-premises IDCs or to the clouds of other CSPs through Express Connect.
- Create a hybrid cloud cluster. Then, add ECS instances from Alibaba Cloud and instances from on-premises IDCs and other CSPs to the cluster.
- Deploy your applications to instances in this cluster.

In hybrid clouds, EDAS is used in the following scenarios:

- Manage applications deployed on instances in on-premises IDCs through Alibaba Cloud. After connecting your IDC to the Alibaba Cloud VPC through a leased line, you can manage your applications in the IDC by using Alibaba Cloud EDAS.
- Scale applications deployed on instances from Alibaba Cloud in or out. 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 instances from other CSPs. EDAS allows you to deploy applications to instances from CSPs other than Alibaba Cloud and manage these instances in a unified manner.

This topic describes how to use Alibaba Cloud to manage applications deployed on instances in on-premises IDCs. To deploy and manage instances from other CSPs, you only need to connect the target instances to the Alibaba Cloud VPC of EDAS through a leased line. Then, you can operate and manage these instances in the same way as instances in on-premises IDCs. 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.

- a) Log on to the EDAS console. For more information, see [Log on to the EDAS console](#).
- b) In the left-side navigation pane, choose **Resource Management** > **Clusters**.
- c) On the **Clusters** page, select the **region** and **namespace**, and click **Create Cluster**.
- d) In the **Create Cluster** dialog box, enter the cluster information and click **Create**.

Parameters for creating a cluster:

- **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:** 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:** 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 instances from on-premises IDCs and other CSPs, perform the following steps:

- a) On the Clusters page, click the name of the cluster you just created.
- b) On the Cluster Details page, click **Add ECS Instance**.
- c) In the **Add ECS Instance** dialog box, copy the command for installing EDAS Agent.
- d) Use the **root** account to log on to your **Alibaba Cloud ECS instance** or the **instance in the on-premises IDC**.
- e) Paste the EDAS Agent installation command and run it.

3. Open the required ports.

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

- 8182: This port is used to capture infrastructure 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 open the ports based on the instance type.

- ECS instances from Alibaba Cloud: Open the ports by referring to relevant documents.
- Instances from on-premises IDCs and other CSPs: Open the ports by referring to relevant solutions.

4. Check the cluster and instance statuses.

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

If the cluster status is **Normal**, the cluster is created. If the value of **Instances** is same as the number of instances you added, the instances are added successfully.

- b. Click the cluster name. On the Cluster Details page, check the values of **Instance Name** and **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 can only 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. See relevant topics to deploy applications.

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, view the instance running status. If Running Status/Time is **Running**, the application is published.

1.5 Console user guide

1.5.1 Overview page

The Overview page of the 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 deploying applications by using EDAS, import ECS instances to the specified cluster and install EDAS Agent.

Prerequisites

EDAS Agent must be installed on each target ECS instance. Before installing EDAS Agent, ensure that the RAM user is authorized. For the authorization procedure, see the Apsara Stack Console User Guide and read the **RAM management** topic

Procedure

1. [Log on to the EDAS console](#).
2. In the EDAS console, choose **Resource Management** > **ECS** from the left-side navigation pane.
3. On the **ECS** page, click **Import ECS** in the upper-right corner.
4. On the **Select Cluster and ECS** page, select a **namespace** and click **Select Cluster to Import**. In the instance list, select ECS instances and click **Next**.
5. On the **Ready to import** page, 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 and **confirm the new password**, and click **Next**.
6. On the **Import** page, view the import status.

On the Import page, the statuses of the imported instances become **Converting**. It may take 5 minutes. If you click "Click to return to the Cluster Details page" before the import is complete, the health check status shows **Converting** and the conversion progress is shown as a percentage. When the import is complete, the health check statuses become **Running**, indicating that the instances are successfully imported.

Result

Click **Click to return to the Cluster Details page** to go to the Cluster Details page. In the **ECS Instances and Applications** 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.

Table 1-4: 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**.

Table 1-5: 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.
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 EDAS resources, which can be ECS instances and SLB instances, but not VPCs. You can control account permissions through resource groups. You can grant resource group access permissions to the RAM users, and each RAM user has the permission to operate on all the resources in the specified group.

Typical scenarios

- Assume that your company publishes its application systems through EDAS. Department A is responsible for user-related applications and Department B for goods-related ones.
- The company registers an EDAS account (the primary account) to activate EDAS and creates one RAM user each for Departments A and B.
- Departments A and B have dedicated ECS and 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 ECS and SLB instances 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 ECS and SLB instances 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, choose **Resource Management > 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 **OK**.

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 Elastic Compute Service (ECS) instances.

1. On the **Resource Groups** page, locate the row that contains 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 **OK**.

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 with your primary account.
2. In the left-side navigation pane, choose **Account Management > Sub-accounts**.
3. Locate the row that contains 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 **OK**.

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 Namespace

With namespaces, you can completely isolate the 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 **Namespace**.
3. On the **Namespace** page, select a **region** and click **Create Namespace** in the upper-right corner.
4. In the **Create Namespace** dialog box, enter **Namespace Name**, **Namespace ID**, and **Description** (optional). Then, click **OK**.

Result

On the **Namespace** 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**.

Table 1-6: 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.

Parameter	Description
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	<p>Select the application runtime environment based on the application framework.</p> <ul style="list-style-type: none"> For High-Speed Service Framework (HSF) applications, select EDAS-Container. For Spring Cloud or Dubbo applications, <ul style="list-style-type: none"> 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 empty 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	<p>Select WAR or JAR.</p> <div> Note: Your deployment method has been determined and cannot be changed.</div> <p>The configuration processes for WAR package deployment and JAR package deployment are similar. Here, WAR package deployment is used as an example.</p>

Parameter	Description
File Uploading Method	<p>Select Upload WAR Package or WAR Package Location.</p> <ul style="list-style-type: none"> • 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. <div>  Note: The name of the application deployment package can only contain letters, digits, hyphens (-), and underscores (_). </div>
Version	<p>Enter a version number, for example, 1.1.0.</p> <div>  Note: We do not recommend that you use a timestamp as the version number. </div>
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 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 ECS instances, you need to manage those instances.

This topic briefly describes some simple management operations.

1.5.3.2.2.1 Scaling (applicable to ECS clusters)

If an application is overloaded, you can use the application scale-out function to manually scale out the application and share its load.

Procedure

Scale-out

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 **Application Scale Out** in the upper-right corner.
4. On the **Scale-Out Method** tab page of the **Purchase Instances** dialog box, select the **target group**, the ECS instance, and the target instance for scale-out, and 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 during scale-out, the added ECS instance automatically deploys, starts, and runs the application.
- If the application is stopped during scale-out, the added ECS instance automatically deploys but does not start or run the application.

Scale-in

5. On the Application Details page, click the **Instance Information** tab.
6. Scale in the instance on the **Instance Information** tab page.
 - If the ECS instance is running, click **Stop** and then **Delete**.
 - If the ECS instance is stopped, click **Delete**.

1.5.3.2.2.2 Create an application branch version

When you create an application, EDAS automatically creates an application group named "Default Group" for the application and adds the ECS instances of the application to this group. You can create subgroups under the default group and add some instances to the subgroups. If you deploy different versions of the application on the instances in the subgroups, 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:

- To perform an online test before publishing a new version
- A/B testing
- Canary deployment

Procedure

1. Create a subgroup.

a) [Log on to the EDAS console](#).

b) In the left-side navigation pane, choose **Application Management**. On the **Applications** page, click the name of the target application.

c) On the Application Details page, click the **Instance Information** tab. On the tab page that appears, click **Create Group** in the upper-right corner.

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

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

2. Add instances to a new group.

After a 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.

- a) On the **Instance Information** tab page of the Application Details page, select the instance whose group you want to change, and click **Change Group** on the right of the list.
- b) In the **Change Group** dialog box, select an option for **Target Group**.
- c) Click **Change Group**.



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 **Redeploy current instance for target group** to redeploy the deployment package on the instance using that in the group.
 - Select **Change group without redeployment** to add the instance without changing its deployment package.

3. Deploy the application in the new group.

- a) On the Application Details page, click **Deploy Application** in the upper-right corner.
- b) Based on [Deploy an application \(applicable to ECS clusters\)](#), set the **target publish group** as the new group, set the deployment parameters, and click **Deploy**.

Result

On the **Instance Information** tab page of the Application Details page, you can view the deployment package version and runtime status of the new group to check that the new application version is successfully 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 the EDAS container. You can upgrade the EDAS container to the specified version.

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 **Container Version** from the left-side navigation pane.
4. On the **Container Version** page, view the current container version 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 upgrade the container to the desired 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** function and select the target version.

Procedure

1. [Log on to the EDAS console](#).
2. In the left-side navigation pane, choose **Application Management**.
3. On the Applications page, click the name of the target application. On the Application Details page, click **Roll Back** in the upper-right corner.
4. Based on the name of the published WAR package and the publishing time that appear on the **Roll Back** page, select the target version and click **Roll Back**.



Note:

The rollback target option appears only when you have deployed a beta instance. Otherwise, all instances under the application are rolled back by default. **A maximum of five rollback versions appear.**

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 deleting 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**. On the **Applications** page, click the name of the target application.
3. Click **Instance Information**. On the **Instance Information** tab page, locate the row that contains the instance for the application, and click **Delete** in the **Actions** column.
4. Click **Delete Application**.

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

By **setting JVM parameters**, you can enable the container parameter setting when an application is started. Correctly setting JVM parameters helps reduce the overhead of GC and thus shorten the server response time and improve throughput. If container parameters are not set, JVM parameters are allocated by default.

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. On the **JVM Parameters** tab page of the **Application Settings** dialog box, click **Memory Configuration**, **Applications**, **GC Policy**, **Tool**, and **Custom** to set relevant parameters. Then, click **Save**.



Note:

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 setting, the message **Setting JVM successfully** appears in the upper-right corner.

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**.

Table 1-7: 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	<p>The access path of an application.</p> <ul style="list-style-type: none">• 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.

Name	Description
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 empty 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.

-Dhsf.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.

-Dhsf.server.max.poolsize

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

-Dhsf.server.min.poolsize

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

-Dhsf.client.localcall

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

-Dpandora.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.

-Dhsf.http.enable

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

-Dhsf.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.

-Dhsf.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.

-Dhsf.shuthook.wait

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

-Dhsf.publish.delayed

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

-Dhsf.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.

-DHsfBindHost

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.

-Dhsf.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.

-Dhsf.client.low.water.mark=32-Dhsf.client.high.water.mark=64-Dhsf.server.low.water.mark=32-Dhsf.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.

-Dhsf.generic.remove.class=true

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

-DdefaultHsfClientTimeout

Specifies the global time-out period of the consumer.

-Dhsf.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.

Table 1-8: Resources

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

Table 1-9: Environment variables

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

Table 1-10: 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 .

Table 1-11: Resources

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

Table 1-12: 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.

Environment variable key	Type	Description
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.

Table 1-13: 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
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.rpm -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_CONTAINER_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 mode, 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.rpm -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 environment variables.
RUN mkdir -p ${CATALINA_HOME}/deploy/
RUN wget http://edas-hz.oss-cn-hangzhou.aliyuncs.com/edas-plugins/edas.sar.
${EDAS_CONTAINER_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.

a) 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
```

b) 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
```

c) 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 `mxm`, `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="\n-Xmx3550m \n-Xms3550m \n-Xmn2g \n-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**.

Table 1-14: 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**.
 - a) 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.
- b) 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.

- c) 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.

- a) On the **Log Directories** page, select a path or folder and click **Bookmark Log Directory** in the upper-right corner of the page.
- b) 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

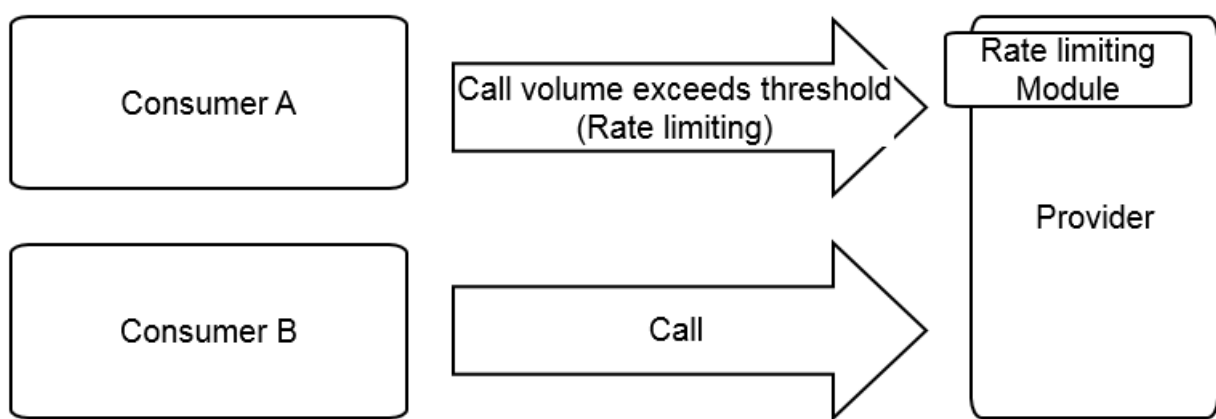
This function controls the traffic threshold or adjusts the traffic ratio. It controls traffic when front-end websites are dealing with heavy access traffic to prevent service unavailability that results from damage to backend core systems. By adjusting the traffic threshold, the

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.

Figure 1-2: Throttling



Degradation

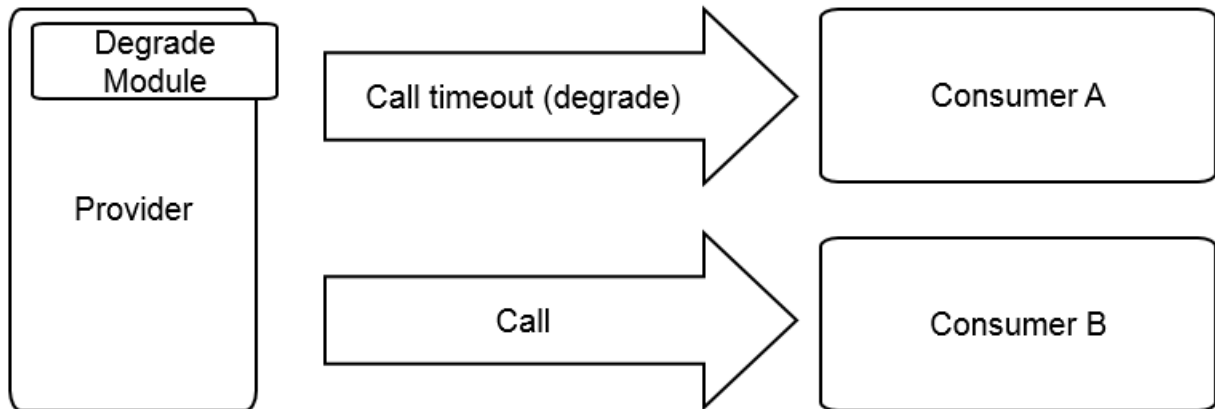
In EDAS, degradation refers to the reduction of the call priority of downstream non-core providers that have timed out to make sure the availability of core consumers.

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.

Figure 1-3: 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.
 - a) [Log on to the EDAS console](#).
 - b) In the left-side navigation pane, choose **Application Management**. On the **Applications** page, click a deployed provider application.
 - c) On the Application Details page, choose **Service Degradation > Rate Limiting Rules** from the left-side navigation pane.
 - d) 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**.

Table 1-15: 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.

Name	Description
Rate Limiting Granularity	Select QPS or Thread. <ul style="list-style-type: none">• 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.
 - a) [Log on to the EDAS console](#).
 - b) In the left-side navigation pane, choose **Application Management**. On the **Applications** page, select a deployed provider application.
 - c) 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**.

Table 1-16: 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.
 - a.** 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:
 - b.** 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.

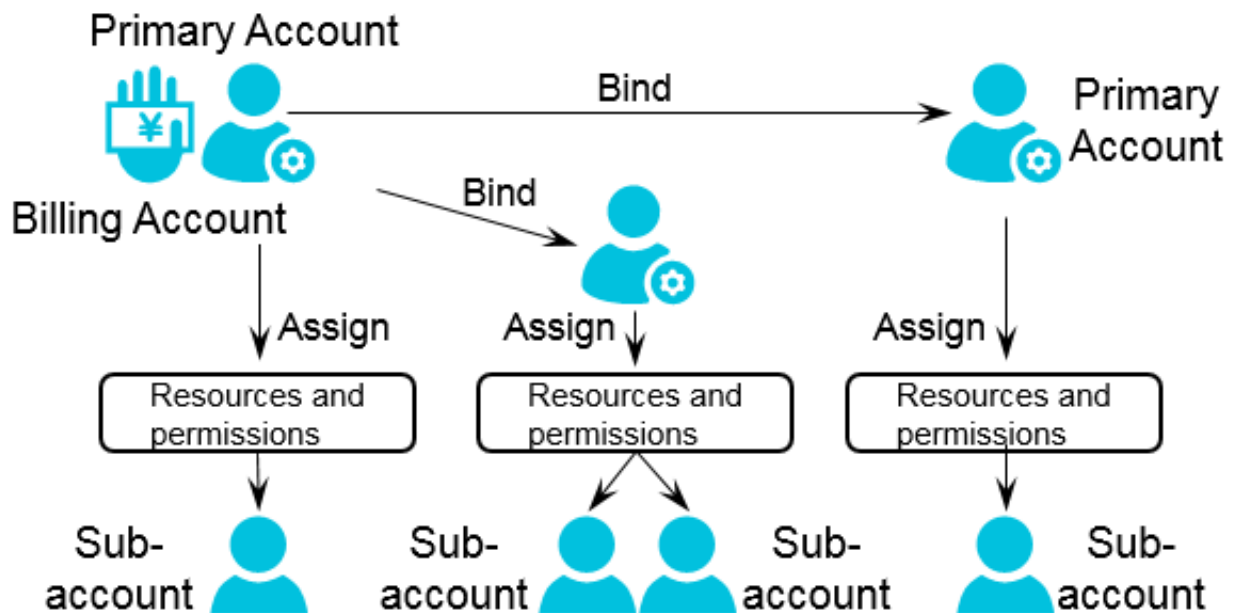
1.5.6 System management

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 you use your primary account to operate EDAS, you need to complete different types of jobs with different user identities, such as application administrator (with the permissions to create, start, stop, query, and delete applications) and operation administrator (with the permissions to list resources, check application monitoring data, and manage alarm rules, throttling rules, and degradation rules). You can allocate different roles and resources to the RAM users under the primary account to complete different types of jobs with different user identities. This primary account and RAM user permission model 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.

Primary account and RAM user relationship

- In the EDAS system, you can bind your primary account to a RAM user to avoid sharing your account key 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 a primary account is bound to a RAM user, their binding relationship is valid only within EDAS, and both are independent accounts in other environments.
- A primary account can be a primary account with RAM users or be a RAM user under another primary 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:

1. Reinstall EDAS Agent. For more information, see [Use the command script to manually install EDAS Agent](#).
2. 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	<p>java.lang.Exception: Service authentication failed</p> <div>  Note: This problem only occurs in the EDAS production environment. </div>	<ul style="list-style-type: none"> The AccessKeyId and AccessKeySecret used for installing EDAS Agent are incorrect or they became incorrect due to web-based installation or other reasons. <ol style="list-style-type: none"> Run <code>cat /home/admin/.spas_key/default</code>. Log on to the EDAS console. In the left-side navigation pane, choose Resource Management > ECS. On the Instances page, click Install Agent. 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. <ul style="list-style-type: none"> Run the <code>date</code> command to check whether the date is accurate.
Unknown host exception	Caused by: java.net.UnknownHostException: iZ25ax7xuf5Z	iZ25ax7xuf5Z indicates the current hostname. Check whether <code>/etc/hosts</code> contains the IP address and name of the current host. If not, configure them, for example, <code>192.168.1.10 iZ25ax7xuf5Z</code> .
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	<p>Address server connection failure jmenu.tbsite.net. Bind the domain. Add the following content to the hosts file to bind the domain name server address: <code>192.168.1.10 jmenu.tbsite.net</code>. Change 192.168.1.10 to the IP address of your lightweight configuration center. The path to the hosts file is as follows:</p> <ul style="list-style-type: none"> Windows: <code>C:\Windows\System32\drivers\etc\hosts</code> Linux: <code>/etc/hosts</code>

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 NullPointerException	Check whether the WAR package is normal. Run <code>jar xvf xxx.war</code> 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
Startup fails when startup.bat and startup.sh are executed.	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.
	Unable to start embedded Tomcat servlet container	<p>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:</p> <p>Windows:</p> <ol style="list-style-type: none"> 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 <code>javaw.exe</code>, appears. 3. Run <code>taskkill /PID 2720 /T /F</code>. 4. Start the lightweight configuration center again. <p>Linux:</p> <ol style="list-style-type: none"> 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.
	Tomcat connector in failed state	
	Caused by: java.net.UnknownHostException: iZ25ax7xuf5Z	iZ25ax7xuf5Z indicates the current hostname. Check whether the IP address and name of the current host are configured in <code>/etc/hosts</code> . If not, configure them, for example, <code>192.168.1.10 iZ25ax7xuf5Z</code> .

- **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:

1. 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.

2. 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.

3. Click **OK** to save the settings.
4. 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\configclient.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 adapter -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:

1. 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.
2. 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/dragon/bin/DragonAgent
```

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:

1. Check whether the group that corresponds to the service provider has been created. If not, authentication may fail.
2. 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:

1. HTTP-based web applications in EDAS use SLB for load balancing. You can configure SLB on the application configuration page of EDAS.
2. 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.