Enterprise Distributed Application Service (EDAS)

Quick Start

Quick Start

Preparation

Activate EDAS

When using the EDAS service for the first time, follow these steps to activate it.

Log on to the EDAS console.

On the homepage of EDAS, click **Buy Now**.

On the sales page of EDAS, select the configuration specifications based on the information in Product Series.The options are **EDAS - Pay-As-You-Go** and **EDAS - Subscription Billing**.

- **EDAS - Pay-As-You-Go**: In this billing method, EDAS services are charged according to the number of instances with applications deployed. Two instances are provided for free and each additional instance has a charge of 1 RMB per day.
- **EDAS - Subscription Billing**:
  - **Current Environment**: The default value is **Public Cloud** and cannot be modified.
  - **Series**: Select Professional Edition or Platinum Edition as needed.For the functional differences between these series, see Product Series.
  - **Number of Application Instances**: Select a number of instances for application deployment as needed.We recommend that you select one to five instances when you use EDAS for the first time. Later, you can scale up this service based on your growing business needs.
  - **Subscription Validity Period**: Select a subscription validity period for the EDAS service.

Log on to your Alibaba Cloud account and finish purchasing the EDAS service as prompted. For EDAS pricing information, see Pricing.

After EDAS is activated, click **Console** to go to the EDAS console.

Note: When you use EDAS for the first time, the **Security Authorization Tip** dialog box is displayed after you log on to the console. Click **Authorize Now**. On the **Cloud Resource Access Authorization** page that is displayed, click **Agree to Authorize**.

Create resources

To deploy an application in EDAS, you must create the required resources.

To help simplify resource utilization, the following describes the basic concepts of EDAS resources.

Basic concepts

Resources	Definition	Scenarios
Elastic Compute Service (ECS)	ECS is a computing service provided by Alibaba Cloud that has elastic processing capabilities.	The applications that you publish in EDAS are deployed on ECS instances.
EDAS Agent	EDAS Agent is a Daemon program provided by EDAS for communication between applications and EDAS service clusters and between applications and the EDAS console.	EDAS Agent is installed on an ECS instance, so you can use and manage the ECS instance in EDAS.
Clusters	In EDAS, a cluster is a collection of ECS instances used to deploy applications.	To meet demanding security requirements, you must specify a namespace and a VPC network when creating a cluster.
Namespace	A namespace is an isolated resource environment built in a region.	To meet demanding resource security requirements, you must create a namespace.
Virtual Private Cloud (VPC)	A VPC network is an isolated network environment built in Alibaba Cloud.	To meet demanding network security requirements, we recommend that you create a VPC network and add ECS instances.
Server Load Balancer (SLB)	SLB is a load balancing service provided by Alibaba Cloud for distributing traffic	SLB instances are used to achieve load balancing among ECS instances.Your

	among multiple ECS instances.	application is accessible on the Internet through the IP address and port of an Internet SLB instance.

By default, all of the preceding resources must be created.The exceptions to this rule are described at the beginning of the relevant sections.

Note： Chrome browser is recommended for operations in EDAS console.

Create VPCs

To create an ECS cluster and publish a common application without special network security requirements, you can skip this step and use a classic network.

Log on to the **VPC console** to create a VPC network.For detailed instructions, see **Deploy a VPC**.

Note: If you have not activated the VPC service, click **Activate VPC Service**.On the VPC sales page, read and agree to the **VPC Activation Agreement** and then click **Open Now**.

Synchronize the VPC network to the EDAS console.

Log on to the **EDAS console**. In the left-side navigation pane, choose **Resource Management** > **VPC**.

On the **VPC** page, select the region where the created VPC is located, for example, East China 1. Then, click **Synchronize VPC** in the upper-right corner.

Verify that the created VPC network has been synchronized to the EDAS console.

Create ECS instances

Log on to the **EDAS console**.

In the left-side navigation pane, choose **Resource Management** > **ECS**.

Select a region and namespace (optional) for the ECS instance to be created. Then, click **Create Instance** in the upper-right corner.

On the ECS purchase page, select ECS type and configuration and pay for it. See Create ECS instances for details.

The application configuration specifications are as follows:

Cluster type	Configuration requirement	Attach data disk	Notes
ECS cluster	CPU≥1 vCPU Memory size ≥1 GB	Supported	- JVMs are multi-thread applications that may cause problems when they run out of memory.Therefore, avoid applying the minimum configurations to the ECS instances added to the cluster created in the production environment. - When all target ECS instances are added to the cluster, set the swap partition.This

			prevents the log process and other processes from using a lot of memory resources and thus from causing running faults in the operating system.
Swarm Cluster	CPU≥1 vCPU Memory size ≥1 GB	Not supported. Plan disk space in advance.	- JVMs are multi-thread applications that may cause problems when they run out of memory.Therefore, avoid applying the minimum configurations to the ECS instances added to the cluster created in the productio

			n environment. - When all target ECS instances are added to the cluster, set the swap partition.This prevents the log process and other processes from using a lot of memory resources and thus from causing running faults in the operating system.
Container Service Kubernetes cluster	CPU≥1 vCPU Memory size ≥ 2 GB	Supported	- When all target ECS instances are added to the cluster, set the swap partition.This prevents the log

			process and other processes from using a lot of memory resources and thus from causing running faults in the operating system.

Note: After you create the ECS instance, it is added to the default namespace and cluster.

Create namespaces

You can skip this step if you have no special resource security requirements.In this case, you can use the default namespace of a region.

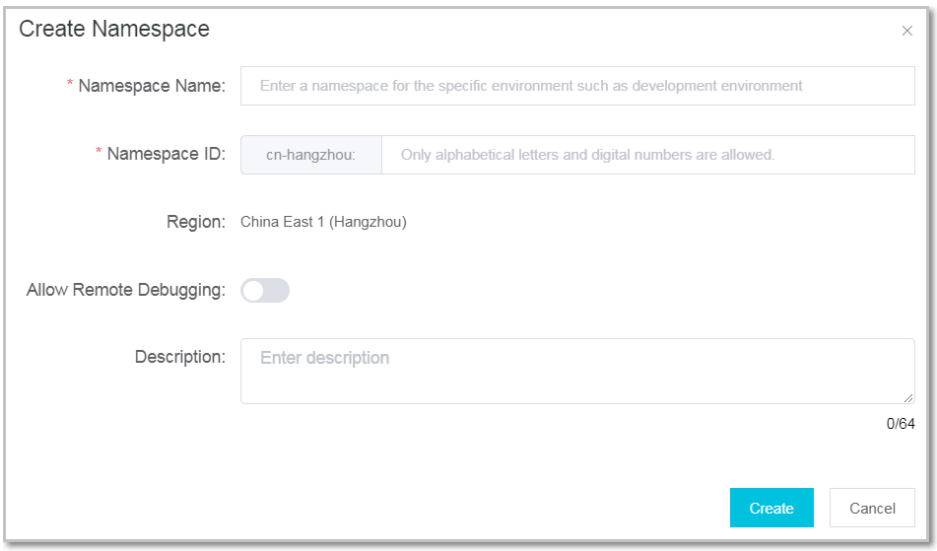
Log on to the EDAS console.

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

On the **Namespace** page, select **Region** and click **Create Namespace** in the upper-right corner.

In the **Create Namespace** dialog box, set **Namespace Name** (required), **Namespace ID** (required), and **Namespace Type** (to **Normal Namespace**), and enter descriptions as needed.Click **Confirm**.

Note: The prefix of a namespace ID is determined by the specified region (target region) and cannot be modified. Instead, only the custom part can be modified.



Create clusters

Create clusters

In the left-side navigation pane, choose **Resource Management** > **Cluster**.

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

In the **Create Cluster** dialog box, set the cluster parameters and click **Create** to save the settings.

Configuration	Description
Cluster Name	Name of a cluster, which contains a maximum of 64 characters consisting of letters, digits, underscores (_), hyphens (-), and periods (.).
Cluster Type	Select **ECS** for common applications and **Swarm** for Docker applications.Note: A Container Service Kubernetes cluster must be created in Container Service and then synchronized to the EDAS console. For more information, see **Create Container Service Kubernetes clusters**.
Network Type	If **Cluster Type** is set to **ECS**, **Network Type** can be **VPC** or **Classic Network**. If **Cluster Type** is set to **Swarm**, **Network Type** can only be **VPC** because Swarm clusters in EDAS do not support classic networks.
VPC	Select the VPC you created.
Namespace	This parameter is selected on the **Clusters** page and cannot be modified here.If no namespace is

	configured, the default namespace of the region is selected by default and displayed as the region name, for example *East China 1*.

Add ECS instances

You can add ECS instances to a cluster on the EDAS in either of the following ways:

Directly import instances to the cluster without image conversion.

Re-install the system with the EDAS official image. After re-installation, all data in the instance will be deleted and a new password for logging on to the instance must be configured.

You cannot import an instance directly to a cluster in any of the following conditions:

- The ECS instance was created before December 1, 2017.
- The instance of classic network type is imported into a cluster of classic network type.
- The instance is not running, for example, the instance is in Stopped, Starting, or Stopping status.
- If you create a Windows instance or another instance that does not support simple shell commands, it cannot be imported to EDAS and converted successfully.
- The instance is not imported from an ECS cluster.

On the **Clusters** page, select a region, such as *East China 1*, and click the name of the created cluster.

On the **Cluster Details** page, click **Add ECS Instance** in the upper-right corner.

On the **Select Cluster and ECS** page, select an instance addition method and ECS instance in the instance list, and then click **Next**.

- **Import ECS**: You cannot modify the namespace and the imported cluster. Also, ECS instances are imported from the default namespace and cluster under this region.
- **From Existing Cluster**: A namespace and a cluster are selected from the region.Then, the ECS instance is moved from the left pane to the right pane on the page.

If no instances meet the necessary conditions, click **Create ECS Instance** in the upper-right corner of the page. This takes you to the ECS purchase page on the Alibaba Cloud official website, where you can purchase and create a new ECS instance.For more information, see Create ECS instances.

On the **Ready to Import** Page, view the information of the selected instance.

- If the ECS instance can be imported directly, click **Confirm and Import**.
- If the instance needs to be converted, 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 root user logon after the conversion.Click **Confirm and Import**.

On the **Import** tab page, view the import progress of the instance.

- If the ECS instance supports direct import, you can view its import progress on the **Import** tab page. If the message **Instance transfer succeeded** is displayed, the ECS instance has been imported successfully.Click **Click to return to the Cluster Details page**. When the ECS instance status changes to **Running**, the ECS instance is imported to the cluster successfully.
- For an ECS instance that needs to be converted before being imported, the import progress of the ECS instance displayed on the **Import** page is **Converting now. This may take 5 minutes**.If you click **Click to return to the Cluster Details page** before the import is completed, the health check status **Converting** and the conversion progress in percentage are displayed. When the import is completed, the health check status **Running** is displayed.

Synchronize SLB instances to EDAS

Log on to the SLB console to create an SLB instance.For detailed instructions, see **Create SLB instances**.

Synchronize the SLB instance to the EDAS console.

Log on to the EDAS console. In the left-side navigation pane, choose **Resource Management** > **SLB**.

On the **SLB** page, select the region where the created SLB instance is located, such as East China 1. Then, click **Synchronize SLB** in the upper-right corner.

Verify that the created SLB instance has been synchronized to the EDAS console.

Deploy Spring Cloud Applications to EDAS

Taking a Spring Cloud application that contains a service provider and a service consumer as an example, this topic describes how to develop, including add dependencies and required configuration items, and test an application locally and then deploy it to EDAS to implement service registration and discovery, as well as consumer calls to providers.

> Even if you know nothing about Spring Cloud and only the basics about Spring and Maven, after going through this article, you will learn how to implement service registration and discovery for Spring Cloud applications, as well as consumer calls to providers through Spring Cloud Alibaba Nacos Discovery.
>
> If you are familiar with service registration components in Spring Cloud such as Eureka, Consul, and ZooKeeper, but have not used the service registration component Nacos Discovery of Spring Cloud Alibaba, you only need to replace the dependencies and configuration items of these service registration components with Spring Cloud Alibaba Nacos Discovery, without modifying any code.
>
> Spring Cloud Alibaba Nacos Discovery also implements the standard interfaces and specifications of Spring Cloud Registry, which are basically the same as the methods that you use to access service registration and discovery in Spring Cloud.
>
> If you are familiar with how to use the open source Spring Cloud Alibaba Nacos Discovery component to register and discover services for Spring Cloud applications, you can directly deploy the applications to EDAS to use the commercial version of service registration and discovery capabilities provided by EDAS. For more information about how to deploy applications to EDAS, see Deploy an application to EDAS.

Spring Cloud Alibaba Nacos Discovery is the commercial version of the open source Nacos Server provided by the EDAS service registry. It allows you to directly use the business edition of service registry provided by EDAS.

The commercial EDAS service registry has the following advantages over Nacos, Eureka, and Consul:

- Components sharing, which saves you the costs of deploying, operating, and maintaining Nacos, Eureka, or Consul.
- Your services are protected from being discovered by unauthorized applications by link encryption for the calls to use service registration and discovery.
- The 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 canary deployment.

Local development

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

Preparations

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

- Download Maven and set environment variables.
- Download the latest version of Nacos Server.
- To start Nacos Server, follow these steps.
    i. Decompress the downloaded Nacos Server package.
    ii. Go to the nacos/bin directory and start Nacos Server.
        - For Linux/UNIX/Mac: Run the sh startup.sh -m standalone command.
        - For Windows: Double-click the startup.cmd file to run it.

Create a service provider

Create a local service provider application project, add dependencies, enable service registration and discovery, and specify Nacos Server as the registry.

Procedure

Create a Maven project named Nacos-service-provider.

Add dependencies to the pom.xml file.

Take Spring Boot 2.1.4.RELEASE and Spring Cloud Greenwich.SR1 as an example. The dependencies are as follows:

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

<dependencies>
<dependency>
<groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
<version>2.1.0.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>Greenwich.SR1</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>

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

This example uses Spring Cloud Greenwich, and the corresponding Spring Cloud Alibaba version is 2.1.0.RELEASE.

- If you are using Spring Cloud Finchley, the corresponding Spring Cloud Alibaba version is 2.0.0.RELEASE.
- If you are using Spring Cloud Edgware, the corresponding Spring Cloud Alibaba version is 1.5.0.RELEASE.

Note: The Spring Cloud Edgware release will reach end-of-life in August 2019. We do not recommend that you use this release to develop applications.

Create a package named com.aliware.edas in src\main\java.

In the com.aliware.edas package, create a startup class named ProviderApplication for the service provider, and add the following code:

The @EnableDiscoveryClient annotation indicates that the service registration and discovery feature must be enabled for the application.

```
package com.aliware.edas;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class ProviderApplication {
```

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

In the com.aliware.edas package, create EchoController, and specify {/echo/{String} as the URL mapping and GET as the HTTP method. Retrieve the method parameter from the URL, and echo the received parameter.

```
package com.aliware.edas;

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

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

In src\main\resources, create a file named application.properties and add the following configuration to application.properties to specify the Nacos Server address.

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. If you have other requirements, refer to Configuration item reference for adding configuration items to the application.properties file.

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

Verify the result

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

Log on to the Nacos Server console at http://127.0.0.1:8848/nacos while noting that both the default username and password of the local Nacos console are “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.

Create a service consumer

This section demonstrates the service registration function and explains how Nacos service discovery works with the RestTemplate and FeignClient clients.

Procedure

Create a Maven project named nacos-service-consumer.

Add dependencies to the pom.xml file.

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

<dependencies>
<dependency>
<groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
<version>2.1.0.RELEASE</version>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
</dependencies>

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

<build>
<plugins>
<plugin>
```

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

In src\main\java, create a package named com.aliware.edas.

In the com.aliware.edas package, set RestTemplate and FeignClient.

Create an interface class named EchoService in 'com.aliware.edas', add the @FeignClient annotation, and configure the corresponding HTTP URL and HTTP method.

```
package com.aliware.edas;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

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

Create a startup class named ConsumerApplication in the com.aliware.edas package and add related configuration items.

i. Use the @EnableDiscoveryClient annotation to enable service registration and discovery.
ii. Use the @EnableFeignClients annotation to activate FeignClient.
iii. Add the @LoadBalanced annotation to integrate RestTemplate with service discovery.

```
package com.aliware.edas;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
```

```
public class ConsumerApplication {

@LoadBalanced
@Bean
public RestTemplate restTemplate() {
return new RestTemplate();
}

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

Create a class named TestController in the com.aliware.edas package to demonstrate and verify the service discovery feature.

```
package com.aliware.edas;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class TestController {

@Autowired
private RestTemplate restTemplate;
@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-feign/{str}", method = RequestMethod.GET)
public String feign(@PathVariable String str) {
return echoService.echo(str);
}

}
```

In src\main\resources, create a file named application.properties and add the following configuration to specify the address of Nacos Server.

127.0.0.1:8848 is the IP address of Nacos Server. If your Nacos Server is deployed on another machine, change the IP address to the corresponding one. If you have other requirements, refer to Configuration item reference for adding configuration items to the

application.properties file.

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

Verify the result

Run the main function of ConsumerApplication in nacos-service-consumer to start the application.

Log on to the Nacos Server console that starts locally at http://127.0.0.1:8848/nacos. Note that both the default username and password of the local Nacos console are "nacos". In the left-side navigation pane, choose **Service Management** > **Services**. You can see service-consumer in the list and query the details of the service in Details.

Test the result locally

Test the result of service call to the provider made by the consumer in a local environment.

For Linux/UNIX/Mac: Run curl http://127.0.0.1:18082/echo-rest/rest-rest and curl http://127.0.0.1:18082/echo-feign/feign-rest.

```
➜  ~ curl http://127.0.0.1:18082/echo-rest/rest-test
rest-test%
➜  ~ curl http://127.0.0.1:18082/echo-feign/feign-test
feign-test%
```

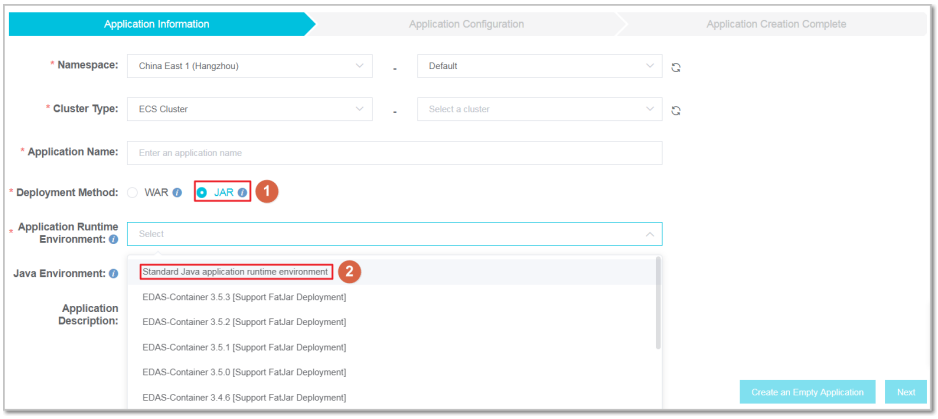
For Windows: In the address bar of your browser, enter http://127.0.0.1:18082/echo-rest/rest-rest and http://127.0.0.1:18082/echo-feign/feign-rest.

Deploy an application to EDAS

After developing and testing your application locally, you can package and deploy it to EDAS. You can choose to deploy your Spring Cloud application to an ECS cluster, Container Service Kubernetes cluster, or EDAS Serverless cluster based on your needs. For detailed steps to deploy an application, see Application deployment overview.

Note: We recommend that you use the console for your initial deployment. If you choose to use JAR packages for this purpose, make sure to select **Standard Java application runtime environment** for

Application Runtime Environment when creating an application.



The EDAS service registry provides a GA release of Nacos Server. When you deploy an application to EDAS, EDAS sets the IP address and service port of Nacos Server, as well as other information such as the namespace, access-key, secret-key, and context-path with higher priority. You do not need to make any additional configurations, and you can choose to retain or delete your original configuration.

Verify the result

After the deployment is completed, in the left-side navigation pane of the EDAS console, choose **Microservice Management** > **Service Query**. On the **Service Query** page, select **Region** and **Namespace** , and then search for your deployed application by entering service-provider and service-consumer.

Configuration item reference

Configuration item	Key	Default value	Description
Server address	spring.cloud.nacos.discovery.server-addr	None	The IP address and port of the server that Nacos Server listens to.
Service name	spring.cloud.nacos.discovery.service	${spring.application.name}	The name of the current service.
Network interface name	sspring.cloud.nacos.discovery.network-interface	None	The registered IP address is the IP address of the corresponding network interface when an IP is not configured. If this item is not configured, the IP address of the first network interface is used by default.

Registered IP address	spring.cloud.nacos.discovery.ip	None	Highest priority
Registered port	spring.cloud.nacos.discovery.port	-1	No configuration is required by default. The system automatically detects the port.
Namespace	spring.cloud.nacos.discovery.namespace	None	One of the common use cases is the isolation of registration in different environments, for example, the isolation of the resources (such as configurations and services) in development, test, and production environments.
Metadata	spring.cloud.nacos.discovery.metadata	None	This item is configured in the Map format. You can customize metadata information related to your services as needed.
Cluster	spring.cloud.nacos.discovery.cluster-name	DEFAULT	Set it to the name of a Nacos cluster.
Endpoint	spring.cloud.nacos.discovery.endpoint	UTF-8	The domain name of the entry of a service in the region. You can dynamically retrieve the server address 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.

For more information about Spring Cloud Alibaba Nacos Discovery, see Spring Cloud Alibaba Nacos Discovery documentation.

FAQ

Use other versions

This example uses Spring Cloud Greenwich, and the corresponding Spring Cloud Alibaba version is 2.1.0.RELEASE. Spring Cloud Finchley must be used with Spring Cloud Alibaba 2.0.0.RELEASE, and Spring Cloud Edgware with Spring Cloud Alibaba 1.5.0.RELEASE.

Note: The Spring Cloud Edgware release will reach end-of-life in August 2019. We do not recommend that you use this release to develop applications.

Migrate from ANS

The EDAS registry configures the data structures of ANS and Nacos in a way that the two are compatible on servers. In the same namespace and when a group is not set up in Nacos, the Nacos client and the ANS client can discover each other' s registration services.

Deploy Dubbo applications to EDAS

You can host Dubbo microservice applications in EDAS and take advantage of shared components, enterprise-class security hardening, and full microservice solutions provided by EDAS to reduce O&M costs and improve security and development efficiency. This topic describes how to develop a Dubbo microservice sample application on-premises through XML configuration items and deploy it to EDAS. The sample application contains a service provider and a service consumer.

Why host applications in EDAS

By hosting Dubbo applications in EDAS, you can focus on building the logic of Dubbo applications rather than concerning yourself with creating and maintaining the registry, configuration center, and metadata center. Also, you can take advantage of EDAS capabilities such as elastic scaling, rate limiting and degradation, monitoring, and microservice governance for various management purposes. The entire hosting process is completely transparent to you, does not require you to learn anything, and does not increase your development costs.

Preparations

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

Download Maven and set environment variables.

Download the latest version of Nacos Server.

To start Nacos Server, follow these steps.

i. Decompress the downloaded Nacos Server package.
ii. Go to the nacos/bin directory and start Nacos Server.
- For Linux/UNIX/Mac: Run the sh startup.sh -m standalone command.
- For Windows: Double-click the startup.cmd file to run it.

Create a service provider

Create a provider application project on-premises, add dependencies, configure service registration and discovery, and specify Nacos as the registry.

Create a Maven project and add dependencies.

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

Add **dubbo**, **dubbo-registry-nacos**, and **nacos-client** to the pom.xml file.

Note: Dubbo 2.7.3 or later is required.

```
<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>
```

Develop a Dubbo service provider.

All services in Dubbo are provided as interfaces.

In src/main/java, create a package named com.alibaba.edas.

In com.alibaba.edas, create an interface named IHelloService that contains a SayHello method.

```
package com.alibaba.edas;

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

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;
}
}
```

Configure the Dubbo service.

In src/main/resources, create a file named provider.xml and open it.

In provider.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">
```

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"/>
```

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

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

i. *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 to EDAS, the registry address will be replaced with the address of the registry in EDAS. You do not need to make any changes.
ii. *8848* is the port number of Nacos Server, which cannot be changed.

Start the service.

In com.alibaba.edas, create the Provider class and load Spring context to the main function of Provider based on the following code to expose the configured Dubbo service.

```
package com.alibaba.edas;

import org.springframework.context.support.ClassPathXmlApplicationContext;

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

Execute the main function of Provider to start the service.

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

Create a service consumer

Create a consumer application project locally, add dependencies, and configure it to subscribe to the Dubbo service.

Create a Maven project and add dependencies.

The procedure is the same as that for creating a provider. For more information, see the procedure for creating a provider.

Develop the Dubbo service.

All services in Dubbo are provided as interfaces.

In src/main/java, create a package named com.alibaba.edas.

In com.alibaba.edas, create an interface named IHelloService, which contains a SayHello method.

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);
}
```

Configure the Dubbo service.

In src/main/resources, create a file named consumer.xml and open it.

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">
```

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"/>
```

Start and verify the service.

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

```
package com.alibaba.edas;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.concurrent.TimeUnit;

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

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

Verify the creation result.

i. After Dubbo 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. Then, in the left-side navigation pane, choose **Services**. On the **Services** page, select **Callers**.

You can see com.alibaba.edas.IHelloService in the list and query the **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 type of cluster to deploy the application to, which is mainly ECS cluster or Container Service Kubernetes cluster, as well as the deployment method, which can be in the console or with tools. For more information, see Application deployment overview.

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

Add the following packaging plug-in configuration 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>
```

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

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

More information

You can also use Spring Boot to develop Dubbo applications. For more information, see Use Spring Boot to develop Dubbo applications.

If you are familiar with Dubbo and only want to try hosting applications in EDAS, you can use Alibaba Cloud Toolkit to create a demo sample and deploy it to EDAS.

If you are using edas-dubbo-extension, see Host Dubbo applications in EDAS with EDAS-Dubbo-extension. With edas-dubbo-extension, you are unable to use related capabilities provided by EDAS, such as Dubbo service governance. Therefore, we recommend that you migrate to Nacos instead.

Quickly create a Demo microservice-oriented application