

Alibaba Cloud

Container Registry Best Practices

Document Version: 20220505

Legal disclaimer

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

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).
5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions









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

Table of Contents


1. Build an image for a Java application by using a Dockerfile wi...	05
2. Migrate images from a self-managed Harbor instance to Conta...	11
3. Access a Container Registry Enterprise Edition instance from a...	13
4. Use Jenkins to create a CI/CD pipeline of images	15

1. Build an image for a Java application by using a Dockerfile with multi-stage builds

The multi-stage builds feature of Docker allows you to specify multiple FROM statements in your Dockerfile. Each FROM statement can use a different base image, and each FROM statement begins a new build stage. Multi-stage builds for Java applications bring benefits such as high security, fast building, and small image size. This topic describes how to build an image for a Java application by using Container Registry and multi-stage builds. In this example, an image is built for a Java application that is developed with Maven and uses GitHub to manage its source code repository.

Prerequisites

- The [Container Registry](#) service is activated.
- A Java application is created and its source code is hosted in a repository on [GitHub](#), [GitLab](#), [Bitbucket](#), or [Alibaba Cloud Code](#).

 **Note** You can use [this Java application that is developed with Maven and uses GitHub to manage its source code repository](#) to experience multi-stage builds.

Context

Common issues in Docker image building

The image building service of Container Registry uses a Dockerfile to build the final image of an application. During this process, you may encounter the following issues:

- It is difficult to write a Dockerfile.

When you get used to using the powerful frameworks of programming languages, especially Java, to conveniently build applications, you may find it difficult to write Dockerfiles to build application images.

- The final image may be large in size.

When you build an image, you may include the application compilation, test, and packaging processes in the same Dockerfile. Each command in the Dockerfile creates a layer of the image, which complicates the structure of the image and enlarges the image size.

- The source code may be leaked.

You may package the source code of your application in the final image, which may lead to code leakage.

Benefits of multi-stage builds

When you use multi-stage builds in a Dockerfile to build images for applications developed with compilation languages such as Java, you can enjoy the following benefits:

- The final image is built in a secure way.

In the first stage of image building, you must specify an appropriate base image. Then, you need to copy source code to the base image, download application dependencies, compile the source code, test the application, and package the application. In the second stage, you must specify another appropriate base image and copy runtime dependency files generated in the first stage to the base image. This way, the final image does not contain the source code.

- The final image has fewer layers and a smaller size.


The final image contains only a base image and compiled artifacts, which occupy few layers and require a small storage size.

- The final image is built at a fast speed.

You can use build tools such as Docker or Buildkit to concurrently run multiple build processes, which improves the build speed.

Step 1: Create a Dockerfile with multi-stage builds

In this example, a Java application that is developed with Maven and uses GitHub to manage its source code repository is used. Create a Dockerfile in the Java project and add the following configuration to the Dockerfile:

-  **Note** The Dockerfile includes two build stages, as shown in the following configuration.
- First stage: Specify the Maven base image, copy source code to the base image, and then run the RUN command to create a JAR package. If the Java application is developed with Gradle, you can specify a Gradle base image in this stage.
 - Second stage: Copy the JAR package generated in the first stage to the OpenJDK image and run the CMD command to generate the final image.

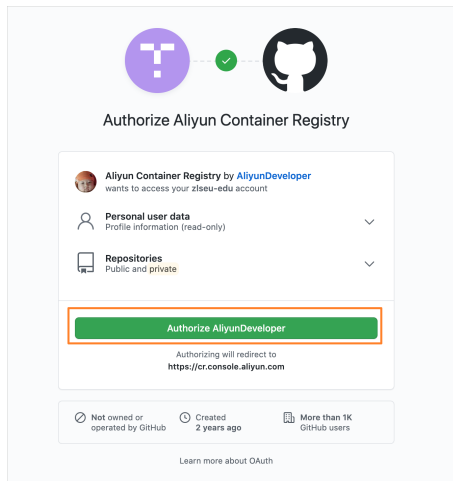
```
# First stage: complete build environment
FROM maven:3.5.0-jdk-8-alpine AS builder
# add pom.xml and source code
ADD ./pom.xml pom.xml
ADD ./src src/
# package jar
RUN mvn clean package
# Second stage: minimal runtime environment
FROM openjdk:8-jre-alpine
# copy jar from the first stage
COPY --from=builder target/my-app-1.0-SNAPSHOT.jar my-app-1.0-SNAPSHOT.jar
EXPOSE 8080
CMD ["java", "-jar", "my-app-1.0-SNAPSHOT.jar"]
```

Step 2: Authorize Container Registry to access the source code repository

Log on to the Container Registry console and authorize Container Registry to access the source code repository. In this example, Container Registry is authorized to access source code repositories on GitHub.

- 1.
- 2.

- 3.
- 4.
- 5.
6. On the management page of the Container Registry Personal Edition instance, choose **Repositories > Code Source** in the left-side navigation pane.
7. Click **Bind Account** for the GitHub service. In the GitHub dialog box, click **Go to the source code repository to bind account**. On the page that appears, enter your username and password to log on to GitHub.
8. On the authorization page, click **Authorize AliyunDeveloper**.



Go to the **Code Source** page. Check whether the status of the GitHub service appears as **Bound**.

Step 3: Create an image repository

- 1.
- 2.
- 3.
- 4.
5. On the management page of the Container Registry Personal Edition instance, choose **Repositories > Repositories** in the left-side navigation pane. On the Repositories page, click **Create Repository**.
6. Set the parameter as required.

Create Repository

Repository Info

Region: China (Shanghai)

Namespaces: devops1-yifu

Repository Name:

Repository Type: ☐ Public ☒ Private

Summary:

Description:

Next Cancel

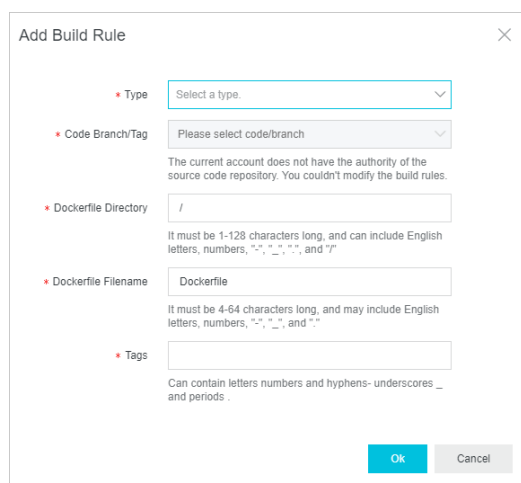
Parameter	Description
Region	The region where the image repository resides. Container Registry supports 23 regions around the world.
Namespaces	The namespace to which the image repository belongs. An image repository belongs to only one namespace, whereas a namespace can contain multiple image repositories.
Repository Name	The name of the image repository.
Repository Type	The type of the image repository. Valid values: Public and Private . You can push images to an image repository only after you log on to the image repository, regardless of the repository type. <ul style="list-style-type: none">Public: You can pull images from the image repository without the need to log on to the image repository.Private: You can pull images from the image repository only after you log on to the image repository on the Docker client.
Summary	The brief description of the image repository.
Description	The detailed description of the image repository.

- Click **Next**. In the Code Source step, specify the code source and build settings.
 - Code Source: the service or platform where the source code repository resides. Click the **Git Hub** tab and select the source code repository that you have authorized Container Registry to access in [Step 2: Authorize Container Registry to access the source code repository](#).
 - Build Settings: the mode in which images are built.
 - Automatically Build Images When Code Changes**: If you select this option, an image is automatically built when code is committed from a branch.
 - Build With Servers Deployed Outside Mainland China**: If you select this option, images are built in a data center outside mainland China and then pushed to the image repository.
 - Build Without Cache**: If you select this option, the system pulls the dependent base image for every image to be built. This may slow down the build process.
- Click **Create Repository**.

On the **Repositories** page, check whether the image repository that you have created appears.

Step 4: Build an image

1. On the management page of the Container Registry Personal Edition instance, choose **Repositories > Repositories** in the left-side navigation pane. On the **Repositories** page, click the name of the repository or click **Manage** in the **Actions** column to go to the repository details page.
2. In the left-side navigation pane, click **Build**. On the page that appears, click **Add Build Rule** in the **Build Rules** section.
3. In the Add Build Rule dialog box, set the parameters as required.



Parameter	Description
Type	The type of content pushed to the source code repository that will trigger the build rule. Valid values: Branch and Tag.
Branch/Tag	The code branch or tag that will trigger the build rule.
Dockerfile Directory	The directory where the Dockerfile resides. The specified directory is a relative directory, with the root directory of the code branch as its parent directory.
Dockerfile Filename	The name of the Dockerfile. Default value: Dockerfile.
Image Tag	The tag of the image to be built.

4. Click **Confirm**.
5. In the **Build Rules** section, find the created rule and click **Build** in the **Actions** column. After you start the build, a build record is generated in the **Build Log** section. When the status of the build record becomes **Successful**, the image is built.

Result

- Check whether the image is built

On the management page of the Container Registry Personal Edition instance, choose **Repositories > Repositories** in the left-side navigation pane. On the Repositories page, click the name of the repository or click **Manage** in the **Actions** column to go to the repository details page. In the left-side navigation pane, click **Tags**. On the Tags page, find the image that has been built.

- Check the number of layers in the image

Log on to the image repository on the Docker client. Run the relevant commands to pull the image and query the number of layers in the image. Only the JAR package compiled in the first stage is added to the image, which occupies small storage space, and the image contains four layers.

```
docker pull registry.cn-hangzhou.aliyuncs.com/docker-builder/java-multi-stage:master
docker inspect registry.cn-hangzhou.aliyuncs.com/docker-builder/java-multi-stage:master |
jq -s '[0][0].RootFS
{
  "Type": "layers",
  "Layers": [
    "sha256:f1b5933fe4b5f49bbe8258745cf396afe07e625bdab3168e364daf7c956b6b81",
    "sha256:9b9b7f3d56a01e3d9076874990c62e7a516cc4032f784f421574d06b18ef9aa4",
    "sha256:edd61588d12669e2d71a0de2aab96add3304bf565730e1e6144ec3c3fac339e4",
    "sha256:2be89a7ccd49878fb5f09d6dfa5811ce09fc1972f0a987bbb5a006992aa3dff3"
  ]
}
```

- Run the image and check the result

Run the following command on the Docker client to run the image and check the result:

```
docker run -ti registry.cn-hangzhou.aliyuncs.com/docker-builder/java-multi-stage:master
Hello World!
```

2. Migrate images from a self-managed Harbor instance to Container Registry Enterprise Edition in 10 minutes

Container Registry Enterprise Edition allows you to migrate images from a self-managed Harbor instance to a Container Registry Enterprise Edition instance. You can also customize a domain name for the Container Registry Enterprise Edition instance. This reduces your costs on creating and maintaining image repositories. Container Registry Enterprise Edition provides professional and stable cloud-based image management and technical support. This service is integrated with Container Service for Kubernetes (ACK) to simplify the application delivery process for enterprises. This topic describes how to migrate the backend data and images of a self-managed Harbor instance to Container Registry Enterprise Edition.

Step 1: Migrate backend data of the Harbor instance

- If your Harbor instance uses Apsara File Storage NAS as the backend storage, you must migrate data from NAS to an Object Storage Service (OSS) bucket. For more information, see [Migrate data from NAS to OSS](#).
- Skip this step if the backend data of the Harbor instance is stored in OSS.

Step 2: Select an OSS bucket


When you create a Container Registry Enterprise Edition instance, you can select an existing OSS bucket as the backend storage of the instance.

1. Attach a RAM role to the account and grant the RAM role the permission to manage the OSS bucket. For more information, see [Use RAM to grant permissions to access custom OSS buckets](#).
2. Create a Container Registry Enterprise Edition instance.

When you create a Container Registry Enterprise Edition instance, set **Instance Storage** to **Custom** and select a bucket. For more information, see [Create a Container Registry Enterprise Edition instance](#).

Step 3: Import images

- 1.
- 2.
- 3.
- 4.
- 5.
6. On the management page of the Container Registry Enterprise Edition instance, choose **Instances** > **Image Import** in the left-side navigation pane,
7. On the Image Import page, click **Trigger Task**.
8. In the Tips dialog box, select **Confirm to import** and click **Confirm**.

 **Note** On the Image Import page, find the created task and click **Details** in the **Actions** column to view the progress.

Step 4: Bind a custom domain name to the instance

You can bind a custom domain name that has a Secure Sockets Layer (SSL) certificate to a Container Registry Enterprise Edition instance. After you perform this operation, you can use the custom domain name to access the instance over HTTPS.

We recommend that you set the custom domain name of the Container Registry Enterprise Edition instance to the domain name of the self-managed Harbor instance. For more information, see [Use a custom domain name to access a Container Registry Enterprise Edition instance](#).

3. Access a Container Registry Enterprise Edition instance from a data center

You can connect a data center to a virtual private cloud (VPC) in Alibaba Cloud by using VPN Gateway, an Express Connect circuit, and Smart Access Gateway. This allows servers in the data center to access Container Registry Enterprise Edition instances. The servers in the data center can push images to and pull images from Container Registry Enterprise Edition instances. This topic describes how to access a Container Registry Enterprise Edition instance from a data center.

Prerequisites

- Elastic Compute Service (ECS) instances in the VPC can access the Container Registry Enterprise Edition instance. For more information, see [Configure access over VPCs](#).
- The data center is connected to the VPC. For more information, see [Connect a data center to a VPC](#).


Obtain the IP addresses that are used to create routing rules

You need to obtain the IP addresses of the Object Storage Service (OSS) bucket that is used as the backend storage, Container Registry Enterprise Edition instance, and authentication service in the VPC. You can create routing rules in the data center based on the obtained IP addresses.

1. Obtain the following three domain names:

- The domain name of the OSS bucket in the VPC.

The domain name of an OSS bucket in a VPC is in the format of `${InstanceId}-registry.oss-${RegionId}-internal.aliyuncs.com`.

 **Note** If you use a custom OSS bucket, the domain name is in the format of `${CustomizedOSSBucket}.oss-${RegionId}-internal.aliyuncs.com`.

- The domain name of the Container Registry Enterprise Edition instance in the VPC.

The default domain name of a Container Registry Enterprise Edition instance in a VPC is in the format of `${InstanceName}-registry-vpc.${RegionId}.cr.aliyuncs.com`.

- The domain name of the authentication service in the VPC.

Run the following command to obtain the domain name of the authentication service in the VPC:

```
curl -vv https://${InstanceName}-registry-vpc.${RegionId}.cr.aliyuncs.com/v2/
```

```

$ curl -vv https://cr-beijing-test-online-registry-vpc.cn-beijing.cr.aliyuncs.com/v2/
* About to connect() to cr-beijing-test-online-registry-vpc.cn-beijing.cr.aliyuncs.com port 443 (#0)
* Trying 172.17.179.206...
* Connected to cr-beijing-test-online-registry-vpc.cn-beijing.cr.aliyuncs.com (172.17.179.206) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* CAfile: /etc/pki/tls/certs/ca-bundle.crt
* CApath: none
* SSL connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate:
*  subject: CN=*.cr.aliyuncs.com, O="Alibaba (China) Technology Co., Ltd.", L=HangZhou, ST=ZheJiang, C=CN
*   start date: Dec 14 06:26:02 2020 GMT
*   expire date: Jan 15 06:26:02 2022 GMT
*   common name: *.cr.aliyuncs.com
*   issuer: CN=GlobalSign Organization Validation CA - SHA256 - G2, O=GlobalSign nv-sa, C=BE
* GET /v2/ HTTP/1.1
* User-Agent: curl/7.29.0
* Host: cr-beijing-test-online-registry-vpc.cn-beijing.cr.aliyuncs.com
* Accept: */*
*
< HTTP/1.1 401 Unauthorized
< Date: Tue, 02 Feb 2021 06:34:36 GMT
< Content-Type: application/json; charset=utf-8
< Content-Length: 87
< Connection: keep-alive
< Docker-Distribution-Api-Version: registry/2.0
< Www-Authenticate: Bearer realm="https://dockerauth-ee-vpc.cn-beijing.aliyuncs.com/auth", service="registry.aliyuncs.com:cn-beijing:chin
{"errors":[{"code":"UNAUTHORIZED","message":"authentication required","detail":null}]}
* Connection #0 to host cr-beijing-test-online-registry-vpc.cn-beijing.cr.aliyuncs.com left intact

```

2. Obtain the IP addresses that are used to create routing rules.

On an ECS instance in the VPC, ping the domain names that you obtained in Step to obtain the IP addresses.

Note After you obtain the IP addresses, you can create routing rules based on these IP addresses. The method of creating routing rules varies with the data center type. Create routing rules based on the type of your data center.

Verify the access to the Container Registry Enterprise Edition instance from the data center

Run the `docker login` command to log on to an image repository in Container Registry. Then, run the `docker pull` command to pull an image to the data center.

Note For more information about how to push and pull images, see [Use a Container Registry Enterprise Edition instance to push and pull images](#).

```

[root@i373-6f0-m0-sub-00000000 ~]# docker pull cr-beijing-test-online-registry-vpc.cn-beijing.cr.aliyuncs.com/ns1/repol1
Using default tag: latest
latest: pulling from ns1/repol1
3fd26ecc9548: Downloading [=====] 39.56MB/51.34MB
24941900ea54: Download complete
7e605cb95896: Download complete
bfb37637326e: Download complete
e06ea7d674f7: Download complete
3f1c12f29c08: Download complete
56706d97528e: Download complete
3bd3078181d7: Download complete
07b8274e7f7d: Download complete
6d8e5f006ed8: Download complete
2423f24a1409: Download complete
6f7a0e0fd12f: Download complete

```


If the progress bar for pulling the image is displayed, the access configuration is valid and takes effect.

4. Use Jenkins to create a CI/CD pipeline of images

Jenkins is an automation tool that is used for continuous integration (CI). You can use Jenkins to create a continuous integration or continuous delivery (CI/CD) pipeline of images. After you commit source code to GitLab, Container Registry automatically uses the source code to build an image. Then, Container Service for Kubernetes (ACK) pulls the image to deploy an application and Container Registry sends an event notification to the DingTalk group. This topic describes how to use Jenkins to create a CI/CD pipeline of images.

Prerequisites

- Git, GitLab, and Jenkins are installed.

 **Note** JDK8 instead of JDK11 is installed on GitLab. Some GitLab plug-ins cannot run on JDK11.

- An Advanced Edition instance of Container Registry Enterprise Edition is created. The access over the Internet feature is enabled for the instance. For more information, see [Create a Container Registry Enterprise Edition instance](#) and [Configure access over the Internet](#).
- An ACK cluster that resides in the same region as the Advanced Edition instance of Container Registry Enterprise Edition is created. For more information, see [Create an ACK managed cluster](#).
- A DingTalk chatbot is created. The webhook URL and secret token of the DingTalk chatbot are recorded. For more information, see [Step 1: Create a DingTalk chatbot](#).

Use Jenkins to create a CI pipeline of images

After you commit source code to GitLab, Container Registry automatically uses the source code to build an image. After you scan the image, Container Registry automatically sends an event notification to the DingTalk group.

1. Create a project in GitLab.
 - i. Log on to GitLab.
 - ii. In the top navigation bar of the GitLab console, choose **Projects > Your projects**.
 - iii. On the **Your projects** page, click **New Project** in the upper-right corner and then click **Create blank project**.

- iv. On the **Create blank project** page, configure the **Project name**, **Project URL**, and **Project slug** parameters, select **Private** for **Visibility Level**, and then click **Create project**.

New project > Create blank project

Project name
java-web

Project URL
http://shoppingmall

Project slug
java-web

Want to house several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)
Description format

Visibility Level

☒ **Private**
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

☐ **Internal**
The project can be accessed by any logged in user except external users.

☐ **Public**

Project Configuration

☐ **Initialize repository with a README**
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project **Cancel**

- v. Create *Dockerfile*, *pom.xml*, *DemoApplication.java*, and *HelloController.java* files on your on-premises computer.

■ **Create a Dockerfile**

```
FROM registry.cn-hangzhou.aliyuncs.com/public-toolbox/maven:3.8.3-openjdk-8-alinyun AS build
COPY src /home/app/src
COPY pom.xml /home/app
RUN ["/usr/local/bin/mvn-entrypoint.sh", "mvn", "-f", "/home/app/pom.xml", "clean", "package", "-Dmaven.test.skip=true"]
FROM registry.cn-hangzhou.aliyuncs.com/public-toolbox/openjdk:8-jdk-alpine
COPY --from=build /home/app/target/demo-0.0.1-SNAPSHOT.jar /usr/local/lib/demo-0.0.1-SNAPSHOT.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/usr/local/lib/demo-0.0.1-SNAPSHOT.jar"]
```

■ **Create a pom.xml file**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```



```
<version>0.0.1-SNAPSHOT</version>
<name>demo</name>
<description>Demo project for Spring Boot</description>
<properties>
  <java.version>1.8</java.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-freemarker</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.13.2</version>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <excludes>
          <exclude>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
          </exclude>
        </excludes>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

- Create a DemoApplication.java file

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import java.util.TimeZone;
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        TimeZone.setDefault(TimeZone.getTimeZone("Asia/Shanghai"));
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

- Create a HelloController.java file

```
package com.example.demo;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import javax.servlet.http.HttpServletRequest;
import java.text.SimpleDateFormat;
import java.util.Date;
@RestController
@Slf4j
public class HelloController {
    @RequestMapping("/hello", "/")
    public String hello(HttpServletRequest request) {
        return "Hello World at " + new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());
    }
}
```

- vi. Run the following command to upload the build file to GitLab:

```
cd java-web # Go to the directory where the build file resides.
git remote set-url origin http://8.218.20*.***/shoppingmall/java-web.git
git push origin master
* [new branch]      master -> master
```

2. Create a pipeline to build images in Jenkins.

- i. Configure the Secure Shell (SSH) key pair of GitLab in Jenkins.
 - a. Log on to Jenkins.
 - b. In the left-side navigation pane of the Jenkins dashboard, click **Manage Jenkins**.
 - c. In the **Security** section, click **Manage Credentials**.
 - d. In the **Stores scoped to Jenkins** section, click **Jenkins** in the **Store** column, and then click **Global credentials**.
 - e. In the left-side navigation pane, click **Add Credentials**.

- f. Set the Kind parameter to **SSH Username with private key**, enter values for the **Description** and **Username** parameters, select **Enter directly**, and then click **OK**.

On the **Global credentials** page, the ID of the credential is automatically generated. Record the ID.

- ii. Create a pipeline.

- a. In the left-side navigation pane of the Jenkins dashboard, click **New Item**.
- b. Enter a name for the pipeline, select **Pipeline**, and then click **OK**.
- c. Click the **Build Triggers** tab, select **Build when a change is pushed to GitLab**, and then select **Push Events**.

Record the webhook URL to the right of **Build when a change is pushed to GitLab**.

- d. Click **Advanced** and then click **Generate** in the lower-right corner of **Secret token**.

Record the secret token that is generated by Jenkins.

- e. Click the **Pipeline** tab, and replace the values of the parameters in the following template with your actual values. Copy the content that you modified to the code editor, and click **Save**.

```
def git_auth_id = "6d5a2c06-f0a7-43c8-9b79-37b8c266****" # The ID of the credential.
def git_branch_name = "master" # The name of the branch.
def git_url = "git@172.16.1*.*:*:shoppingmall/java-web.git" # The address of the GitLab repository.
def acr_url = "s****-devsecops-registry.cn-hongkong.cr.aliyuncs.com" # The endpoint of the image repository.
def acr_username = "acr_test_*****@test.aliyunid.com" # The username of the container image.
def acr_password = "HelloWorld2021" # The password of the container image.
def acr_namespace = "ns" # The name of the namespace.
def acr_repo_name = "test" # The name of the image repository.
def tag_version = "0.0.1" # The tag of the container image.
node {
    stage('checkout git repo') {
        checkout([class: 'GitSCM', branches: [[name: "*/${git_branch_name}"]],
        extensions: [], userRemoteConfigs: [[credentialsId: "${git_auth_id}", url: "${git_url}"]]])
    }
    stage('build image') {
        sh "sudo docker build -t java-web:${tag_version} ."
        sh "sudo docker tag java-web:${tag_version} ${acr_url}/${acr_namespace}/${acr_repo_name}:${tag_version}"
    }
    stage('push image') {
        sh "sudo docker login --username=${acr_username} --password=${acr_password} ${acr_url}"
        sh "sudo docker push ${acr_url}/${acr_namespace}/${acr_repo_name}:${tag_version}"
    }
}
```

3. Add the webhook URL to GitLab.

- i. Log on to GitLab.
 - ii. On the **Projects** page, click the name of the project that you created.
 - iii. In the left-side navigation pane, choose **Settings > Webhooks**. Enter the webhook URL and secret token, clear **Enable SSL verification**, and then click **Add webhooks**.
4. Create an event notification rule.
 - i.
 - ii.
 - iii.
 - iv.
 - v.
 - vi. In the left-side navigation pane of the instance details page, choose **Instances > Event Notification**.
 - vii. On the **Event Rules** tab, click **Create Rule**.
 - viii. In the **Event Scope** step of the Create Rule wizard, enter a value for the **Rule Name** parameter, select **The image is scanned** for the **Event Type** parameter, select **Scan Completed**, set **Effective Scope** to **Namespace**, select ns as the name of the namespace, and then click **Next**.
 - ix. In the **Event Notification** step, set **Notification Method** to **DingTalk**, enter the webhook URL and secret token of the DingTalk chatbot, and then click **Save**.
5. Trigger image build.

Run the following command and replace the value of the parameter in the HelloController.java file with your actual value. Commit the file to GitLab to trigger image build.

```
vim java/com/example/demo/HelloController.java # Replace the value of the parameter in  
the HelloController.java file with your actual value.  
git add . && git commit -m 'commit' && git push origin # Commit the file to GitLab.
```

Wait a while. In the left-side navigation pane of the details page of the Container Registry Enterprise Edition instance, choose **Repository > Repositories**. On the right side of the page that appears, click the test repository. In the left-side navigation pane of the management page of the repository, click **Tags**. An image is generated on the **Tags** page.

6. Configure security scan.
 - i. On the **Tags** page, click **Security Scan** in the **Actions** column of the image.

- ii. On the **ACR Scan Engine** tab, click **Trigger Scan**.


After the image is scanned, Container Registry sends the following notification to the DingTalk group.



Use Jenkins to create a CD pipeline of images

After you commit source code in GitLab, Container Registry automatically uses the source code to build an image. After the image is built, the delivery chain is automatically triggered. After the delivery chain is executed, an HTTP request is automatically sent to Jenkins. Then, A Deployment in ACK is triggered to pull the image again to deploy the application.

1. Create an application.
 - i.
 - ii.
 - iii.
 - iv.
 - v. On the **Deployments** page, click **Create from YAML** in the upper-right corner.
 - vi. On the **Create** page, select a value for **Namespace**, select **Custom** from the **Sample Template** drop-down list, copy the following content to the template, and then click **Create**.

 **Note** If you do not enable the pulling images without secrets feature in ACK and Container Registry, you must enable the access over the Internet feature in Container Registry and set the image type to public. For more information, see [Configure access over the Internet](#).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
```

```

    app: demo
    name: demo
spec:
  replicas: 3
  minReadySeconds: 5
  progressDeadlineSeconds: 60
  revisionHistoryLimit: 5
  selector:
    matchLabels:
      app: demo
  strategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      annotations:
        prometheus.io/port: "9797"
        prometheus.io/scrape: "true"
      creationTimestamp: null
      labels:
        app: demo
    spec:
      containers:
      - image: s*****-devsecops-registry.cn-hongkong.cr.aliyuncs.com/ns/test:0.0.1
        imagePullPolicy: Always
        name: demo
        ports:
        - containerPort: 8080
          name: http
          protocol: TCP
        readinessProbe:
          initialDelaySeconds: 5
          tcpSocket:
            port: 8080
          timeoutSeconds: 5
        resources:
          limits:
            cpu: "2"
            memory: 512Mi
          requests:
            cpu: 100m
            memory: 64Mi
    status: {}
---
apiVersion: v1
kind: Service
metadata:
  name: demo-svc
spec:
  selector:
    app: demo
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080

```

```
    port: 80
    targetPort: 8080
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: demo
  labels:
    app: demo
spec:
  rules:
  - host: app.demo.example.com
    http:
      paths:
      - backend:
          serviceName: demo-svc
          servicePort: 80
---
```

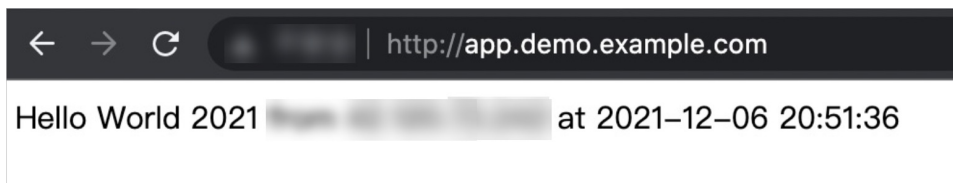
- vii. On the **Deployments** page, click the name of the application demo. Click the **Access Method** tab.

Obtain the external endpoint on the **Access Method** tab.

- viii. Enter the following content for the local host.

```
<The external endpoint> app.demo.example.com
```

- ix. Enter *app.demo.example.com* in the address bar of the browser.



If a page appears as shown in the preceding figure, the application is deployed.

2. Create an application trigger.
 - i. On the Deployments page, click the name of the application demo.
 - ii. On the details page of the application, click the **Triggers** tab. Then, click **Create Trigger**.
 - iii. In the **Create Trigger** dialog box, set **Action** to **Redeploy** and click **OK**.
Obtain the webhook URL on the **Triggers** tab.
3. Create a pipeline.
 - i. Log on to Jenkins.
 - ii. In the left-side navigation pane, click **New Item**.
 - iii. Enter a name for the pipeline and click **Pipeline**.

iv. Click the **Build Trigger** tab and select **Generic Webhook Trigger**.

In the **Generic Webhook Trigger** section, you can obtain the webhook URL of the HTTP request, which is `JENKINS_URL/generic-webhook-trigger/invoke`. In this example, the generic webhook token is set to `helloworld2021`. Therefore, the webhook URL is `JENKINS_URL/generic-webhook-trigger/invoke?token=helloworld2021`.

v. Click the **Pipeline** tab, and replace the generic webhook token and webhook URL in the following template with your actual values. Copy the content that you modified to the code editor, and click **Save**.

```
pipeline {
  agent any
  triggers {
    GenericTrigger(
      genericVariables: [
        [key: 'InstanceId', value: '$.data.InstanceId'],
        [key: 'RepoNamespaceName', value: '$.data.RepoNamespaceName'],
        [key: 'RepoName', value: '$.data.RepoName'],
        [key: 'Tag', value: '$.data.Tag']
      ],
      causeString: 'Triggered on $ref',
      token: 'helloworld2021', # The generic webhook token. Replace the token with your actual value.
      tokenCredentialId: '',
      printContributedVariables: true,
      printPostContent: true,
      silentResponse: false,
      regexpFilterText: '$ref'
    )
  }
  stages {
    stage('Some step') {
      steps {
        sh "echo 'will print post content'"
        sh "echo $InstanceId"
        sh "echo $RepoNamespaceName"
        sh "echo $RepoName"
        sh "echo $Tag"
        sh "echo 'redploy to ACK or you can deoloy to other platforms use before message'"
        sh "curl 'https://cs.console.aliyun.com/hook/trigger?token=g*****' # Replace the webhook URL with your actual value."
        sh "echo 'done'"
      }
    }
  }
}
```

4. Create a delivery chain. For more information, see [Create a delivery chain](#).

5. Create an event notification rule.

- i. In the left-side navigation pane of the management page of the Container Registry Enterprise Edition instance, choose **Instances > Event Notification**.

- ii. On the **Event Rules** tab, click **Create Rule**.
 - iii. In the **Event Scope** step of the Create Rule wizard, enter a value for the **Rule Name** parameter, select **The delivery chain is processed** for **Event Type**, select **Success**, set the **Effective Scope** parameter to **Namespace**, select **ns** from the Namespace drop-down list, and then click **Next**.
 - iv. In the **Event Notification** step, set the **Notification Method** parameter to **HTTP**, enter the webhook URL that you obtained in **Step 3**, and then click **Save**.
6. Run the following command and replace the value of the parameter in the `HelloController.java` file with your actual value. Commit the code to GitLab to trigger image build.

```
vim java/com/example/demo/HelloController.java # Replace the value of the parameter in
the HelloController.java file with your actual value.
git add . && git commit -m 'add update' && git push origin # Commit the file to GitLa
b.
```

After the image is built, the delivery chain is triggered. After the delivery chain is executed, a Deployment in ACK is triggered to pull the image again to deploy the application.

7. Run the following command to check whether the application is redeployed.

```
curl app.demo.example.com
```

If the command output changes, the application is redeployed.