ALIBABA CLOUD

阿里云

容器镜像服务

最佳实践

文档版本：20220419

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.使用触发器构建容器DevOps

通过容器镜像服务可以便捷地构建基于容器的DevOps开发环境。本文介绍如何使用触发器实现，镜像代码被修改后，自动触发镜像构建，并自动触发容器服务上应用的重新部署。

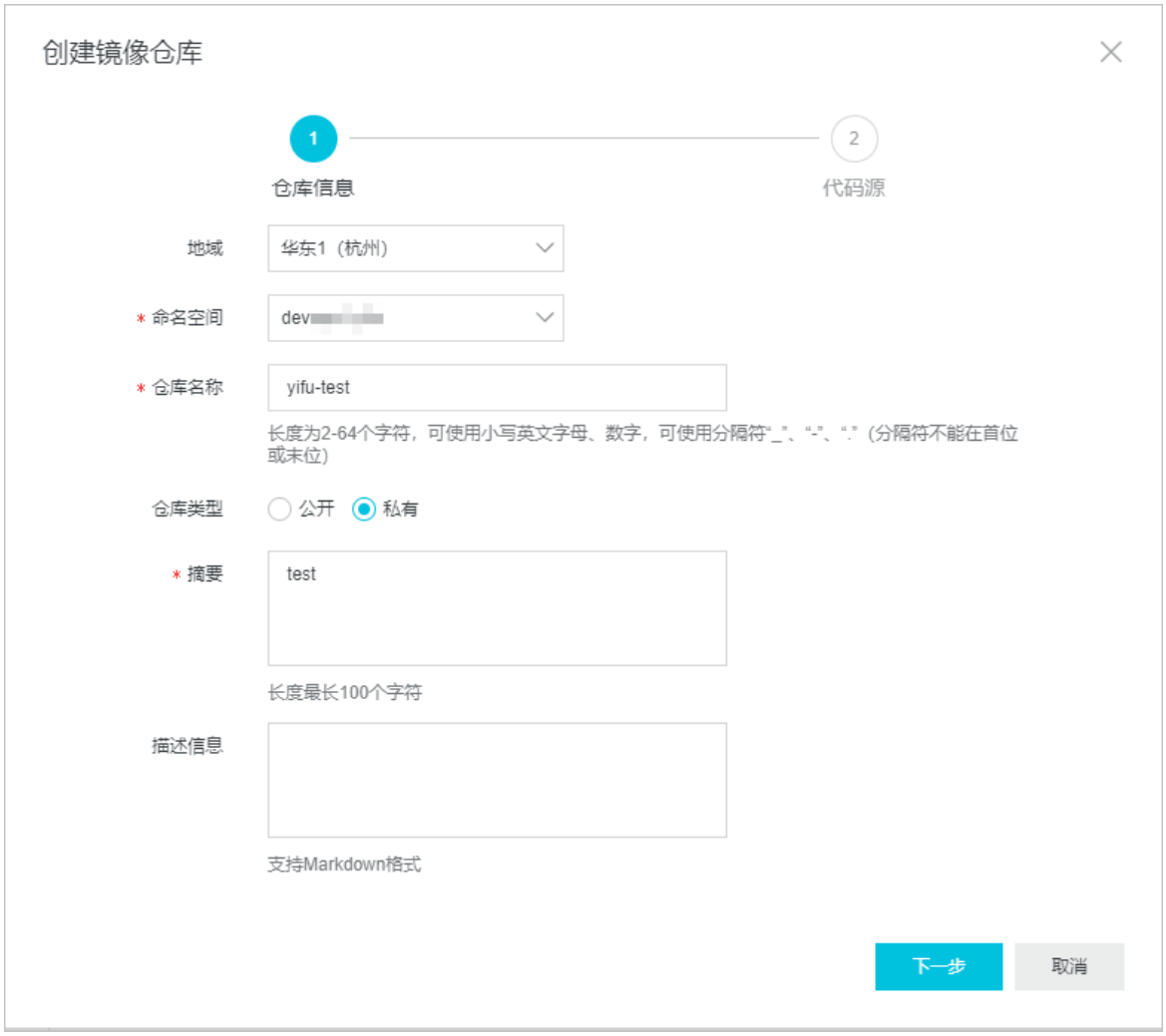
步骤一：创建仓库

1. 如果您还没有开通过阿里云Code，需要单击**绑定账号**去开通。

   > **说明** 默认情况下，如果您的容器镜像服务登录账户已经开通了阿里云Code，将会默认展示您在阿里云Code上的项目。

2. 基于阿里云Code上的项目，创建一个仓库。

   > **说明** 建议在构建设置上选择**代码变更时自动构建镜像**，这样当您在阿里云Code上进行代码修改时，将会触发仓库的自动构建，并将新的镜像推送至阿里云的Registry。



3. 将项目的master分支设置成latest的镜像版本。

   当您希望使用这个仓库镜像时，可以直接使用*reigstry.aliyuncs.com/**/dockertest*，无需指定Tag为latest版本，默认使用稳定的master分支构建稳定的latest镜像版本。

步骤二：构建仓库

在仓库的详情页，单击**立即构建**，仓库将使用新添加的两条构建规则进行构建。

当您在阿里云Code项目的test分支中修改并提交代码后，将触发仓库的第二条构建规则进行自动构建，产生新版本的镜像。

步骤三：绑定仓库触发器

仓库的触发器可以订阅新版本镜像产生的事件，建议可以先用 `http://requestb.in/` 生成一个request URL，绑定在仓库触发器上。当产生新的镜像后，您会看到触发器的访问记录，包括请求的时间、请求的参数以及请求得到的结果。其中请求的参数提供了当前仓库的相关信息。

步骤四：绑定容器服务触发器

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**多集群 > 应用中心**。
3. 在**应用**页面，单击目标应用区域。
4. 在应用资源详情页，单击**触发器**。
5. 在弹出的**触发器**面板中，单击**创建**，然后拷贝触发器URL。
6. 返回镜像信息页面，单击左侧操作栏中的**触发器**，新建一个触发器，并填入触发器的名称、URL、触发方式。

当阿里云Code上的代码被修改后，容器镜像将会自动构建，并自动触发容器服务上应用的重新部署。

2.在Dockerfile中使用多阶段构建打包Java应用

多阶段构建指在Dockerfile中使用多个FROM语句，每个FROM指令都可以使用不同的基础镜像，并且是一个独立的子构建阶段。使用多阶段构建打包Java应用具有构建安全、构建速度快、镜像文件体积小等优点，本文以Github上的Java Maven项目为例，结合阿里云容器镜像服务（ACR）的镜像构建服务，介绍如何进行多阶段构建。

前提条件

- 开通阿里云容器镜像服务。
- 请准备一个托管在Github、Gitlab、Bitbucket或者阿里云Code平台上的Java源代码仓库。

> **说明** 您可以拷贝并托管位于Github上的一个简单的Java Maven项目来体验整个流程。

背景信息

镜像构建的通用问题

镜像构建服务使用Dockerfile来帮助用户构建最终镜像，但在具体实践中，存在一些问题：

- Dockerfile编写有门槛

  开发者（尤其是Java）习惯了语言框架的编译便利性，不知道如何使用Dockerfile构建应用镜像。

- 镜像容易臃肿

  构建镜像时，开发者会将项目的编译、测试、打包构建流程编写在一个Dockerfile中。每条Dockerfile指令都会为镜像添加一个新的图层，从而导致镜像层次深，镜像文件体积特别大。

- 存在源码泄露风险

  打包镜像时，源代码容易被打包到镜像中，从而产生源代码泄漏的风险。

多阶段构建优势

针对Java这类的编译型语言，使用Dockerfile多阶段构建，具有以下优势：

- 保证构建镜像的安全性

  当您使用Dockerfile多阶段构建镜像时，需要在第一阶段选择合适的编译时基础镜像，进行代码拷贝、项目依赖下载、编译、测试、打包流程。在第二阶段选择合适的运行时基础镜像，拷贝基础阶段生成的运行时依赖文件。最终构建的镜像将不包含任何源代码信息。

- 优化镜像的层数和体积

  构建的镜像仅包含基础镜像和编译制品，镜像层数少，镜像文件体积小。

- 提升构建速度

  使用构建工具（Docker、Buildkit等），可以并发执行多个构建流程，缩短构建耗时。

步骤一：使用多阶段构建Dockerfile

以Java Maven项目为例，在Java Maven项目中新建Dockerfile文件，并在Dockerfile文件添加以下内容。

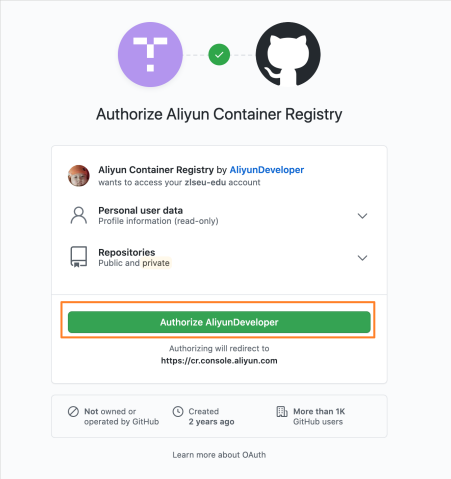
> **说明** 该Dockerfile文件使用了二阶段构建。
>
> - 第一阶段：选择Maven基础镜像（Gradle类型也可以选择相应Gradle基础镜像）完成项目编译，拷贝源代码到基础镜像并运行RUN命令，从而构建Jar包。
> - 第二阶段：拷贝第一阶段生成的Jar包到OpenJDK镜像中，设置CMD运行命令。

```
# 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"]
```

步骤二：绑定源代码仓库

在容器镜像服务控制台，绑定您托管的代码仓库，以下内容以GitHub为例。

1. 登录容器镜像服务控制台。
2. 在顶部菜单栏，选择所需地域。
3. 在左侧导航栏，选择**实例列表**。
4. 在**实例列表**页面单击个人版实例。
5. 在个人版实例管理页面左侧导航栏中选择**仓库管理 > 代码源**。
6. 在GitHub栏单击**绑定账号**，在弹出的对话框中单击**点击前往源代码仓库登录**，跳转到GitHub。
7. 在授权界面，单击**Authorize AliyunDeveloper**。



返回至**代码源**界面，GitHub栏中显示**已绑定**，表示绑定成功。

步骤三：新建镜像仓库

1. 登录容器镜像服务控制台。
2. 在顶部菜单栏，选择所需地域。
3. 在左侧导航栏，选择**实例列表**。
4. 在**实例列表**页面单击个人版实例。
5. 在个人版实例管理页面左侧导航栏中选择**仓库管理 > 镜像仓库**，然后单击**创建镜像仓库**。
6. 设置镜像仓库信息。

配置项	描述
地域	镜像仓库所在区域，目前阿里云容器镜像服务已经全球23个区域开服。
命名空间	仓库所属命名空间。一个镜像仓库必须且仅属于一个命名空间。一个命名空间下可以包含多个镜像仓库。
仓库名称	输入**仓库名称**。
仓库类型	仓库类型分为**公开**和**私有**。无论公开还是私有类型仓库，推送镜像都需要先进行登录。 ○ 公开：拉取镜像时可以免登录，直接通过网络拉取。 ○ 私有：必须要通过Docker客户端进行登录，才能拉取镜像。
摘要	简要描述信息。
描述信息	完整描述信息。

7. 单击**下一步**，设置代码源。
   - 代码源：将代码源设为**GitHub**，选择步骤二：绑定源代码仓库中绑定的源代码仓库。
   - 构建设置：
     a. **代码变更时自动构建镜像**：当分支有代码提交后会自动触发构建规则。
     b. **海外机器构建**：构建时会在海外机房构建，构建成功后推送到指定地域。
     c. **不使用缓存**：每次构建时会强制重新拉取基础依赖镜像，可能会增加构建时间。
8. 单击**创建镜像仓库**。
   选择**仓库管理 > 镜像仓库**，在**镜像仓库**界面查找到创建的仓库，表示创建镜像仓库成功。

步骤四：执行构建

1. 在左侧导航栏中选择**仓库管理 > 镜像仓库**，单击仓库名称或目标仓库**操作**列的**管理**，进入仓库详情页面。
2. 单击左侧导航栏中的**构建**，在**构建规则设置**区域的单击**添加规则**。
3. 设置构建规则。

配置项	描述
类型	定义了推送代码到托管仓库，触发构建规则的事件。目前有Branch和Tag两种类型的推送。
Branch/Tag	设置构建的代码分支。
Dockerfile目录	设置Dockerfile文件所在的目录。这里的目录指的是相对目录，以代码分支的根目录为父目录。
Dockerfile文件名	设置Dockerfile文件名，默认为Dockerfile。
镜像版本	设置镜像版本。

4. 单击**确认**，返回至构建页面。
5. 在**构建规则设置**区域中，找到创建的规则，单击目标规则对应**操作**列的**立即构建**。
   在**构建日志**区域中找到构建记录，当构建状态显示**成功**，表示构建成功。

执行结果

- 查看构建的镜像

  在个人版实例管理页面左侧导航栏中选择**仓库管理 > 镜像仓库**，单击仓库名称或目标仓库**操作**列的**管理**，进入仓库详情页面。单击**镜像版本**，查看构建的镜像。

- 查看镜像层数

  使用Docker客户端登录镜像仓库，执行以下命令，拉取构建的镜像并检查镜像层数。可以发现仅包含了第一阶段编译Jar包，大小增加有限，镜像层数控制在四层。

```
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"
  ]
}
```

- 运行镜像，查看运行结果

  使用Docker客户端登录镜像仓库，执行以下命令，运行镜像，查看运行结果。

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

3.十分钟迁移自建Harbor至容器镜像服务企业版

容器镜像服务企业版提供镜像极速导入和自定义域名等功能，可以短时间内迁移自建Harbor至容器镜像服务企业版。将自建Harbor迁移至容器镜像服务企业版后，减少了自行搭建及维护的运维管理成本，并提供云上专业稳定的托管服务及技术支持，实现了与容器服务ACK的无缝集成，帮助企业降低交付复杂度。本文将介绍如何将Harbor的后端存储数据、镜像等迁移到容器镜像服务企业版。

步骤一：迁移Harbor的后端存储数据

- 如果Harbor的后端存储为NAS，则需要先将NAS服务器上的数据迁移到阿里云OSS的Bucket中，详细介绍请参见NAS迁移至OSS教程。
- 如果Harbor的后端存储为阿里云OSS，请跳过此步骤。

步骤二：自定义OSS Bucket

容器镜像服务企业版创建实例时，支持自定义阿里云OSS Bucket作为企业版实例的后端存储。

1. 为账号添加RAM角色，并为该RAM角色授予对OSS Bucket的操作权限，详细介绍请参见配置使用自定义OSS Bucket时的RAM访问控制。
2. 创建企业版实例。

   创建企业版实例时，设置**实例存储**为**自定义**，并选择Bucket。详细介绍请参见创建企业版实例。

步骤三：导入镜像

1. 登录容器镜像服务控制台。
2. 在顶部菜单栏，选择所需地域。
3. 在左侧导航栏，选择**实例列表**。
4. 在**实例列表**页面单击目标企业版实例。
5. 在企业版实例管理页左侧导航栏中选中**实例管理 > 镜像导入**。
6. 在镜像导入页面单击**触发导入**。
7. 在**提示**对话框中选中**确认以上信息，触发导入任务**，然后单击**确定**。

> **说明** 在镜像导入页面，单击对应的镜像导入任务**操作**列中的**详情**，可查看任务进度。

步骤四：自定义域名

容器镜像服务企业版支持自定义域名功能，该功能允许为容器镜像服务企业版实例添加自定义域名和相应的SSL证书，从而使用自定义域名并通过HTTPS协议来访问实例。

自定义容器镜像服务企业版实例的域名为Harbor使用的域名，详细介绍请参见通过自定义域名访问容器镜像服务企业版实例。

4.从本地数据中心访问容器镜像服务企业版实例

您可以通过VPN网关、高速通道物理专线、智能接入网关打通本地数据中心和云上VPC，从而可以从本地数据中心访问容器镜像服务企业版实例，实现在本地数据中心推送和拉取镜像至企业版实例。本文将介绍如何从本地数据中心访问容器镜像服务企业版实例。

前提条件

- 云上VPC中的ECS能够正常访问容器镜像服务企业版实例。具体操作，请参见配置专有网络的访问控制。
- 已打通本地数据中心和云上VPC。具体操作，请参见连接本地IDC。

获取创建路由规则的IP

您需要获取OSS Bucket、容器镜像服务企业版实例和鉴权服务在VPC中的IP，根据获取的IP在本地数据中心创建路由规则。

1. 获取以下三种域名。
   - 获取OSS Bucket在VPC中的域名。关于OSS内网域名的更多信息，请参见OSS内网域名与VIP网段对照表。

     OSS Bucket在VPC中的域名为 `${InstanceId}-registry.oss-${RegionId}-internal.aliyuncs.com` 。

     > **说明** 如果您使用的是自定义OSS Bucket，则域名为 `${CustomizedOSSBucket}.oss-${RegionId}-internal.aliyuncs.com` 。

   - 获取容器镜像服务企业版实例在VPC中的域名。

     容器镜像服务企业版实例在VPC中的默认域名为 `${InstanceName}-registry-vpc.${RegionId}.cr.aliyuncs.com` 。

   - 获取鉴权服务在VPC中的域名。

     执行以下命令，获取鉴权服务在VPC中的域名。

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

     ```
     ]# curl -vv https://cr                                    g.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                du8yu1"
     <
     {"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. 获取配置路由规则的IP。

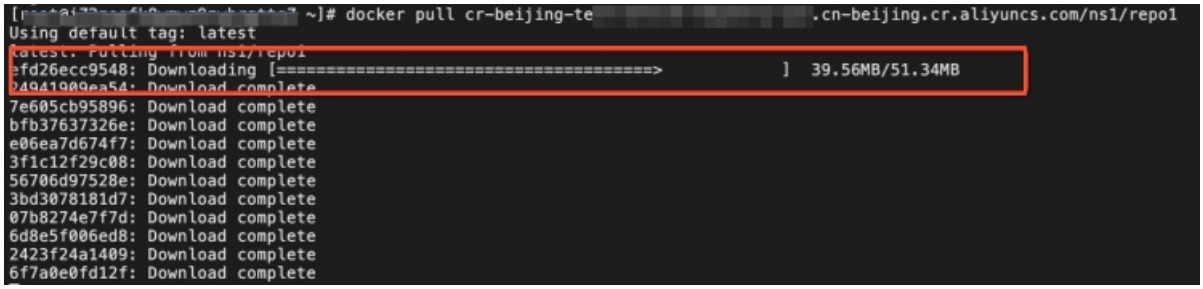
   在打通的VPC中的云上ECS分别Ping步骤获取的域名，返回结果即为配置路由规则的IP。

> **说明** 获取配置路由规则的IP后，您还需要配置路由规则。本地数据中心类型很多，您需要根据实际使用的本地数据中心创建路由规则，本文不再赘述。

验证从本地数据中心访问容器镜像服务企业版实例

使用 `docker login` 登录容器镜像仓库，然后执行 `docker pull` 命令，从本地数据中心拉取镜像。

> **说明** 关于推送和拉取镜像的详细介绍，请参见使用企业版实例推送和拉取镜像。



可以看到拉取镜像的进度条，说明可以从本地数据中心访问容器镜像企业版实例。

5.使用Jenkins实现镜像的CI/CD

Jenkins是一个持续集成工具。您可以使用Jenkins实现镜像的CI/CD，只要您在GitLab中提交源代码，容器镜像会自动使用源代码构建镜像，容器服务会自动拉取镜像部署应用，并自动发送事件通知到钉钉群。本文介绍如何使用Jenkins实现镜像的CI/CD。

前提条件

- 已安装Git、GitLab和Jenkins。

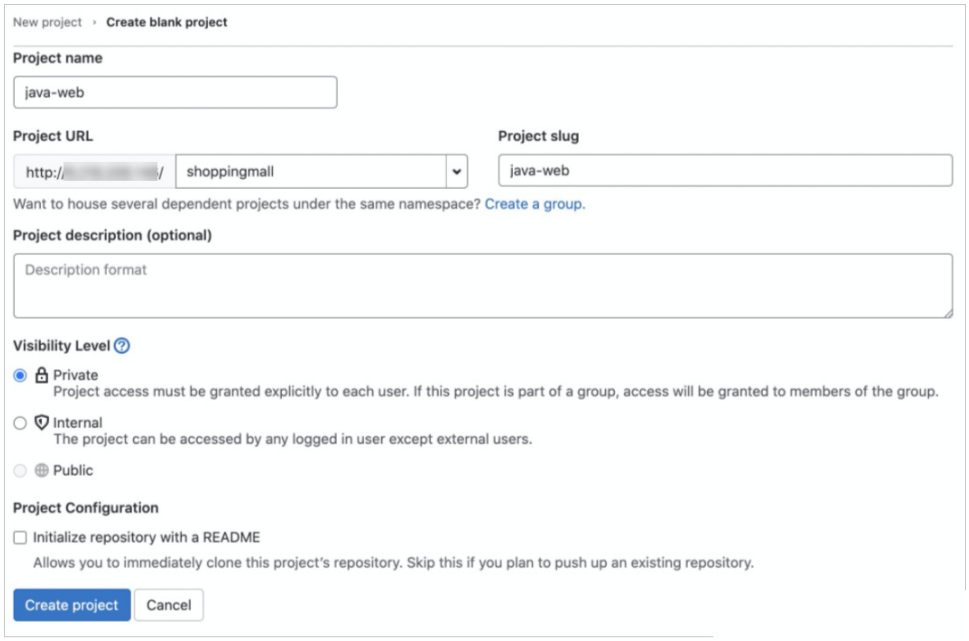
  > **说明** GitLab需要安装JDK8，部分插件不能在JDK11上运行。

- 已创建ACR高级版实例，并且开启公网访问。具体操作，请参见创建企业版实例和配置公网的访问控制。
- 已创建ACK集群，且与ACR实例位于同一地域。具体操作，请参见创建Kubernetes托管版集群。
- 已创建钉钉机器人，并记录下Webhook地址和加签密钥。具体操作，请参见步骤一：创建钉钉机器人。

使用Jenkins实现镜像的CI

只要您在GitLab中提交源代码，容器镜像会自动使用源代码构建镜像，然后您可以对镜像进行漏洞扫描，镜像扫描完成后，将自动发送事件通知到钉钉群。

1. 在GitLab中创建项目。
   i. 登录GitLab。
   ii. 在顶部导航栏中，选择**Projects > Your projects**。
   iii. 在**Projects**页面单击右上角的**New Project**，单击**Create blank project**。
   iv. 在**Create blank project**页面设置**Project name**、**Project URL**和**Project slug**，设置**Visbility Level**为**Private**，然后单击**Create project**。



   v. 使用以下内容，在本地创建*Dockerfile*、*pom.xml*、*DemoApplication.java*和*HelloController.java*文件。

- Dockerfile

```
FROM registry.cn-hangzhou.aliyuncs.com/public-toolbox/maven:3.8.3-openjdk-8-aliyun 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"]
```

- pom.xml

```
<?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>
    <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>
```

- DemoApplication.java

```
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);
    }
}
```

- HelloController.java

```
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").fo
rmat(new Date());
    }
}
```

vi. 执行以下命令，上传构建文件至GitLab。

```
cd java-web  #进入构建文件所在目录。
git remote set-url origin http://8.218.20*.***/shoppingmall/java-web.git
git push origin master
* [new branch]      master -> master
```

2. 在Jenkins配置构建镜像的流水线。
   i. 在Jenkins配置GitLab的SSH Key。
      a. 登录Jenkins。
      b. 在Jenkins左侧导航栏单击**系统管理**。
      c. 在**安全**区域单击**Manage Credentials**。
      d. 在**Stores scoped to Jenkins**区域单击**存储**列下的**Jenkins**，然后单击**全局凭据**。
      e. 在左侧导航栏单击**添加凭据**。
      f. 设置类型为**SSH Username with private key**，然后输入**描述**、**Username**、选中**Enter directly**，然后单击**确定**。

         在**全局凭据**页面会自动生成凭据ID，保存该凭据ID。

ii. 创建流水线。

a. 在Jenkins左侧导航栏单击**新建任务**。

b. 输入任务名称，选中**流水线**，单击**确定**。

c. 单击**构建触发器**页签，选中**Build when a change is pushed to GitLab**，然后选中**Push Events**。

在**Build when a change is pushed to GitLab**右侧记录Webhook的地址。

d. 单击**高级**，单击**Secret token**右下角的**Generate**。

Jenkins会生成一个Secret token，保存该Secret token。

e. 单击**流水线**页签，根据实际情况修改以下模板，然后将内容复制到文本框中，然后单击**保存**。

```
def git_auth_id = "6d5a2c06-f0a7-43c8-9b79-37b8c266****"  #凭据ID
def git_branch_name = "master" #分支名
def git_url = "git@172.16.1*.***:shoppingmall/java-web.git"   #GitLab仓库地址
def acr_url = "s*****-devsecops-registry.cn-hongkong.cr.aliyuncs.com"  #镜像仓库地址
def acr_username = "acr_test_*****@test.aliyunid.com"   #容器镜像用户名
def acr_password = "HelloWorld2021"   #容器镜像密码
def acr_namespace = "ns"    #命名空间
def acr_repo_name = "test"   #镜像仓库名称
def tag_version = "0.0.1"   #镜像版本
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. 添加Webhook地址到GitLab中。

i. 登录GitLab。

ii. 在**Projects**页面单击上文创建的Project名称。

iii. 在左侧导航栏选择**Settings > Webhooks**，输入Webhook地址和Secret token，取消选中**Enable SSL verfication**，然后单击**Add webhooks**。

4. 创建事件通知。

i. 登录容器镜像服务控制台。

ii. 在顶部菜单栏，选择所需地域。

iii. 在左侧导航栏，选择**实例列表**。

iv. 在**实例列表**页面单击目标企业版实例。

v. 在实例详情页面左侧导航栏选择**实例管理 > 事件通知** 。

vi. 在**事件规则**页签下单击**创建规则**。

vii. 在**事件范围**配置向导中设置**规则名称**，**事件类型**为**镜像扫描完成**，选中**扫描完成**，设置**生效范围**为**命名空间**，选择命名空间为ns，然后单击**下一步**。

viii. 在**事件通知**配置向导中设置**通知方式**为**钉钉**，输入钉钉的Webhook地址和加签密钥，然后单击**保存**。

5. 触发镜像构建。

执行以下命令，修改HelloController.java文件，提交文件到GitLab中，从而触发镜像构建。

```
vim java/com/example/demo/HelloController.java #根据实际情况在本地修改HelloController.java文件
git add . && git commit -m 'commit' && git push origin #提交文件到GitLab
```

稍等一段时间，在企业版实例详情页面左侧导航栏选择**仓库管理 > 镜像仓库**，在右侧页面单击目标仓库test，在**仓库管理**页面左侧导航栏单击**镜像版本**，可以看到**镜像版本**页面生成一个镜像。

6. 设置漏洞扫描。

i. 在镜像版本页面单击目标镜像右侧操作下的安全扫描。

ii. 在**ACR扫描引擎**页签下单击**触发扫描**。

待镜像扫描完成后，您可以在钉钉群里收到以下通知。

使用Jenkins实现镜像的CD

只要您在GitLab中提交源代码，容器镜像会自动用源代码构建镜像，镜像构建成功后，自动触发执行交付链，待交付链运行完成后，将自动发送HTTP请求至Jenkins，然后触发ACK的Deployment重新拉取镜像部署应用。

1. 创建应用。

i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，单击**集群**。

iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

iv. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。

v. 在**无状态**页面单击右上角的**使用YAML创建资源**。

vi. 在**创建**页面顶部选择**命名空间**，设置**示例模板**为**自定义**，然后复制以下内容到模板中，然后单击**创建**。

> **说明** 没有设置免密拉取镜像，您需要在容器镜像服务中开启公网访问，并设置镜像为公开镜像。具体操作，请参见配置公网的访问控制。

```
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
---
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
---
```

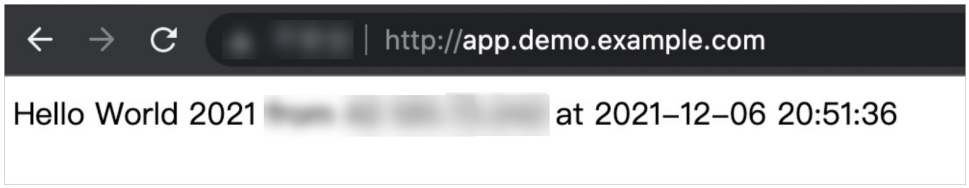
vii. 在**无状态**页面单击目标应用demo，单击**访问方式**页签。

在**访问方式**页签下获取外部端点地址。

viii. 在本地hosts绑定以下内容。

```
<外部端点地址> app.demo.example.com
```

ix. 在浏览器中输入*app.demo.example.com*。



看到以上页面，说明应用部署成功。

2. 创建触发器。

i. 在无状态页面单击目标应用demo的名称。

ii. 在应用详情页面单击**触发器**页签，单击**创建触发器**。

iii. 在**创建触发器**对话框设置**触发器行为**为**重新部署**，然后单击**确定**。

在**触发器**页签下获取触发器链接。

3. 创建流水线。

i. 登录Jenkins。

ii. 在左侧导航栏单击**新建任务**。

iii. 输入任务名称，单击**流水线**。

iv. 单击**构建触发器**页签，选中**Generic Webhook Trigger**。

在**Generic Webhook Trigger**下可以获取HTTP请求触发地址为 `JENKINS_URL/generic-webhook-trigger/invoke` ，本文设置 `Generic Webhook token` 为 `helloworld2021` ，因此Webhook地址为 `JENKINS_URL/generic-webhook-trigger/invoke?token=helloworld2021` 。

v. 单击**流水线**页签，根据实际情况修改以下模板中的Generic Webhook token和触发器链接，然后将内容复制到文本框中，然后单击**保存**。

```
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 $ref',
     token: 'helloworld2021',   #Generic Webhook token，请根据实际情况替换。
     tokenCredentialId: '',
     printContributedVariables: true,
     printPostContent: true,
     silentResponse: false,
     regexpFilterText: '$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 'redeploy to ACK or you can deoloy to other platforms use before m
essage'"
        sh "curl 'https://cs.console.aliyun.com/hook/trigger?token=g****'  #触发器链
接，请根据实际情况替换。
        sh "echo 'done'"
      }
    }
  }
}
```

4. 创建交付链。具体操作。请参见创建交付链。
5. 创建事件规则。
   i. 在企业版实例管理页面左侧导航栏中选择**实例管理 > 事件通知**。
   ii. 在**事件规则**页签下单击**创建规则**。
   iii. 在**事件范围**配置向导中设置**规则名称**，**事件类型**为**交付链处理完成**，选中**成功**，设置**生效范围**为**命名空间**，选择命名空间为ns，然后单击**下一步**。
   iv. 在**事件通知**配置向导中设置**通知方式**为**HTTP**，输入步骤3获取的Webhook地址，然后单击**保存**。
6. 执行以下命令，修改HelloController.java文件，提交代码到GitLab中，从而触发镜像构建。

```
vim java/com/example/demo/HelloController.java #根据实际情况在本地修改HelloController.java文件。
git add . && git commit -m 'add update' && git push origin   #提交文件到GitLab中。
```

镜像构建成功后，触发执行交付链，交付链执行完毕后，触发ACK的Deployment重新拉取镜像部署应用。

7. 执行以下命令，验证应用是否重新部署。

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

可以看到输出内容发生了变化， 说明应用重新部署成功。