

编译构建

# 最佳实践

文档版本

01

发布日期

2023-11-15



**版权所有 © 华为技术有限公司 2024。保留一切权利。**

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## **商标声明**



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## **注意**

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

# 安全声明

## 漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：

<https://www.huawei.com/cn/psirt/vul-response-process>

如企业客户须获取漏洞信息，请参见如下网址：

<https://securitybulletin.huawei.com/enterprise/cn/security-advisory>

目 录

1 使用 Maven 构建自定义镜像（内置执行机，图形化构建）.....1

2 使用 NPM 构建自定义镜像（内置执行机，图形化构建）..... 15

3 使用 Maven 构建上传软件包和推送镜像到 SWR（内置执行机，预置镜像，代码化构建）  
.....20

4 使用 Maven 构建执行多任务构建工程（内置执行机，预置镜像，代码化构建）.....25

5 基于私有依赖库使用 Maven 构建并上传软件包（内置执行机，预置镜像，图形化构建）  
.....31

6 HE2E DevOps 实践：构建应用部分..... 38

# 1 使用 Maven 构建自定义镜像（内置执行机，图形化构建）

## 应用场景

编译构建服务提供了大量构建工具、模板等，并通过缓存、私有依赖库、开源镜像站等实现开箱即用编译构建体验。但由于构建场景多样化，初次使用编译构建服务时，仍有可能会因设计不当或理解偏差、使用方式不当，导致上手过程存在一定困难。因此，编译构建针对常见的复杂构建场景提供完整的最佳实践方案，供初次使用编译构建服务或需要尝试复杂构建场景的用户使用。

本实践为您介绍如何使用编译构建服务完成Maven构建，使用构建包制作Docker镜像并推送到SWR仓库，同时对构建过程涉及的开源镜像站、私有依赖库、缓存的使用等进行简要说明。

本实践需要依赖使用的其它服务如下：

- SWR，即[容器镜像服务](#)。SWR镜像仓库用于存储用户上传的Docker镜像，可以在构建、部署或其他场景使用。
- [代码托管服务](#)，用于存储实践中项目所使用的代码。
- [制品仓库服务](#)，用于存储实践中使用的私有依赖包。

## 前提条件

- 已在容器镜像服务中[创建组织](#)。
- 需已具备CodeArts Artifact服务的操作权限，具体操作可参考[授权使用CodeArts Artifact服务](#)。
- 需已具备CodeArts Repo服务的操作权限，具体操作可参考[授权使用CodeArts Repo服务](#)。

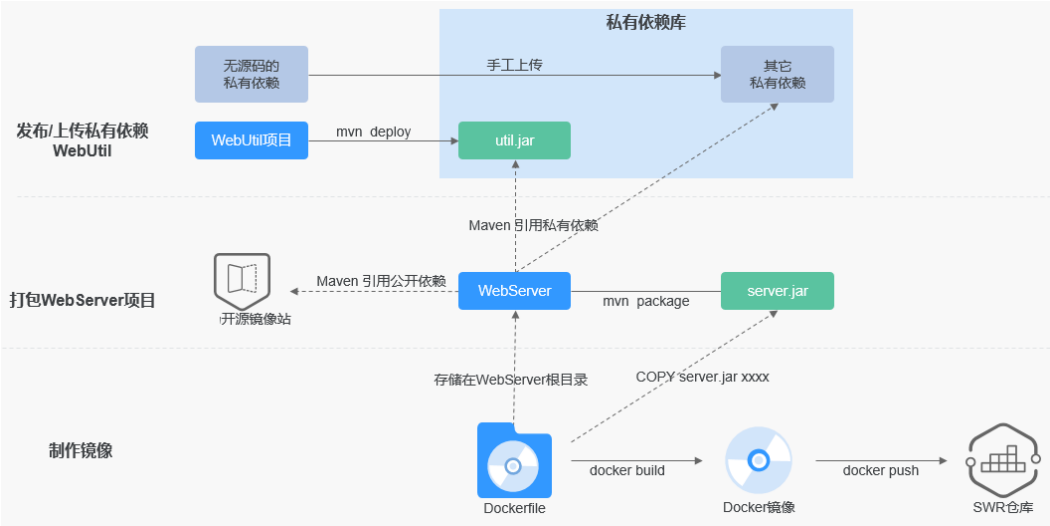
## 方案架构

本文演示项目涉及两个Maven构建工程、一个基础Docker镜像及一个Dockerfile。

- WebServer：此次构建主工程，期望使用CodeArts Build构建此工程，并使用得到的构建包制作Docker镜像，制作镜像所用Dockerfile存放于此项目根目录。
- WebUtil：WebServer依赖的自研工具包，在WebServer项目pom文件中引入，主要用于演示私有依赖库使用场景。

- 基础镜像：以此镜像为基础，在基础镜像中添加WebServer构建包制作Docker镜像。
- Dockerfile：用来制作镜像。

项目构建过程如下：




本文详细描述了从准备代码仓库到构建并制作镜像、推送镜像到SWR仓库的完整过程。详细的实践操作流程如下表所示。

流程	说明
新建项目	为本实践新建项目。
准备项目使用的代码	准备WebServer项目代码仓库和准备WebUtil代码仓库。
上传代码至代码仓	将准备好的代码仓上传至代码仓库。
新建私有依赖库	用于存储构建生成的依赖包。
新建私有依赖扩展点	新建连接私有依赖库的服务扩展点，新建后可使用私有依赖库中依赖包。
发布私有依赖到私有依赖仓库	本示例中假设WebServer使用了三种私有依赖。 <ul style="list-style-type: none"><li>● WebUtil：项目组自研公共组件。</li><li>● CommonUtil：合作伙伴提供，有“jar”包，有“pom”文件（CommonUtil中“pom”文件，不可使用WebServer的“pom”文件），可使用POM模式手动上传。</li><li>● MessageSDK：第三方消息推送平台提供，只有“jar”包，无“pom”文件，此时需要考虑可否通过其他途径获取“pom”文件，或者能否直接使用GAV模式上传。</li></ul>
制作镜像并推送至SWR仓库	通过执行构建任务，制作镜像并上传至SWR仓库。

流程	说明
查看构建结果	到SWR中查看上传的镜像。

新建项目

- 步骤1 使用华为云账号[登录华为云控制台页面](#)。
- 步骤2 单击页面左上角，在服务列表中选择“开发与运维 > 编译构建 CodeArts Build”。
- 步骤3 单击“立即使用”，进入编译构建服务首页。
- 步骤4 单击导航栏“首页”，在首页单击“新建项目”，选用“Scrum”项目模板。
- 步骤5 项目名称填写“build-bestpraticce”，其它保持默认即可。
- 步骤6 单击“确定”后，进入到“build-bestpraticce”项目下。
- 结束

准备 WebServer 代码

- 步骤1 在本地新建一个用于存放代码的目录“WebServer”并进入该目录。
- 步骤2 在“WebServer”目录下新建名称为“pom.xml”的文件，文件内容如下：

```
<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 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.xx.demo</groupId>
  <artifactId>server</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>server</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>


  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-jar-plugin</artifactId>
          <version>2.6</version>
          <configuration>
            <archive>
              <manifest>
                <addClasspath>true</addClasspath>
              </manifest>
            </archive>
          </configuration>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

```
<Main-Class>
  HelloWorld
</Main-Class>
</manifestEntries>
</archive>
</configuration>
</plugin>
</plugins>
</pluginManagement>
</build>
</project>
```

**步骤3** 在“WebServer”目录下新建“src\main\java”目录。

**步骤4** 在**步骤3**新建的目录中，新建名称为“HelloWorld.java”的文件，内容如下：

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

**步骤5** 在编译构建服务首页，单击右上角，选择“自定义构建环境”。

**步骤6** 单击“基于centos7包含各种常用工具的X86基础镜像”，即可获取该基础镜像对应的Dockerfile文件。

可使用DockerHub或SWR仓库中公开镜像，本例中使用CentOS作为基础镜像。

**步骤7** 通过**步骤2**中WebServer代码仓的pom.xml构建包的坐标确认构建包路径。

Maven构建包路径格式为“artifactId-version.packaging”，构建包默认生成于“./target”目录下。则最终构建包路径为“./target/server-1.0.jar”。

**步骤8** 使用**步骤7**中获取的构建包路径编写**步骤6**中获取的Dockerfile文件，内容如下。

```
FROM centos
MAINTAINER <devcloud@demo.com>
USER root
RUN mkdir /demo
COPY ./target/server-1.0.jar /demo/app.jar
```

----结束

## 准备 WebUtil 代码

**步骤1** 在本地新建目录“WebUtil”。

**步骤2** 在“WebUtil”目录下新建名称为“pom.xml”的文件，内容如下：

```
<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 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.xx.demo</groupId>
  <artifactId>util</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>util</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```



```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.6</version>
        <configuration>
          <archive>
            <manifest>
              <addClasspath>true</addClasspath>
            </manifest>
            <manifestEntries>
              <Main-Class>
                HelloWorld
              </Main-Class>
            </manifestEntries>
          </archive>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
</project>
```

**步骤3** 在“WebUtil”目录下新建“src\main\java”目录。

**步骤4** 在**步骤3**新建的目录下新建名称为“HelloWorld.java”的文件，内容如下：

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

----结束

## 上传代码至代码仓

**步骤1** 在导航栏选择“代码 > 代码托管”。

**步骤2** 进入代码托管页面，单击“新建仓库”，选择“普通仓库”。

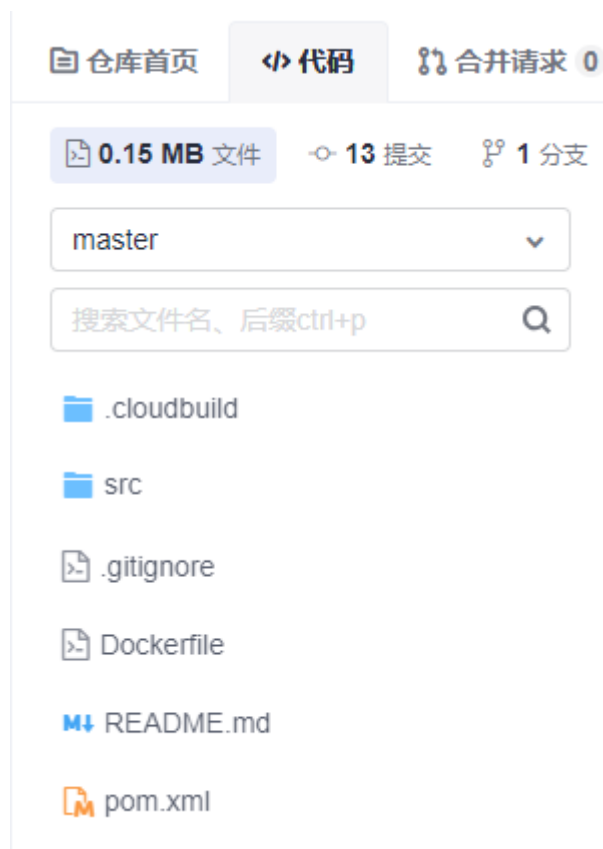
**步骤3** 单击“下一步”，“代码仓库名称”命名为“WebServer”，“选择gitignore”选择“Java”，其它参数保持默认即可。

**步骤4** 单击“确定”完成“WebServer”代码仓库的新建。

**步骤5** 重复**步骤2**至**步骤4**，完成“WebUtil”代码仓库的新建

**步骤6** 参考[上传代码至代码仓库](#)，将**准备WebServer代码**中**步骤2**、**步骤4**和**步骤8**中新建的文件上传到“WebServer”的代码仓库，将**准备WebUtil代码**中**步骤2**和**步骤4**新建的文件上传至“WebUtil”的代码仓库。

图 1-1 WebServer 代码仓目录



----结束

## 新建私有依赖库

**步骤1** 在导航栏选择“制品仓库 > 私有依赖库”。

**步骤2** 单击“新建”，在新建私有依赖库页面“仓库类型”选择“本地仓”，“仓库名称”为“Self-hosted-Repo01”，“制品类型”选择“Maven”，其它参数保持默认即可。

**步骤3** 单击“确定”。

----结束

## 新建私有依赖扩展点

**步骤1** 在导航栏选择“设置 > 通用设置 > 服务扩展点管理”。

**步骤2** 单击“新建服务扩展点”，选择“nexus repository”。

**步骤3** 在弹出的对话框中填写参数信息。

## 新建服务扩展点

\* 连接名称  
请输入接入点名称


\* 仓库地址

\* 用户名

\* 密码

确定 取消

- 连接名称：使用私有依赖扩展点的显示名称，本例中可命名为“私有依赖扩展点”。
- 仓库地址：私有仓库地址。

在导航栏选择“制品仓库 > 私有依赖库”，单击仓库地址旁的, 获取私有仓库地址。

## 详细信息

仓库名称	test5 (Release )
仓库类型	本地仓
仓库地址	https://ap-southeast-3_080b8db696000fe60f27c009b9105fe0_maven_...
相对路径	--

- 用户名：从私有依赖库下载的指定私有仓库配置文件中的用户名。
- 密码：从私有依赖库下载的指定私有仓库配置文件中的密码。

用户名和密码获取方式如下：

1. 在导航栏选择“制品仓库 > 私有依赖库”，
2. 单击“操作指导”。
3. 选择Maven依赖管理工具，单击“下载配置文件”。

## 选择依赖管理工具

☒ Maven ☐ Gradle

1.使用前请确保您已安装 JDK 及 Maven。

↓ 下载配置文件 新手指引

4. 获取文件中的用户名和密码。

```
<server>
  <id>release_ap-southeast-3_080b8db696000fe60f27c009b9105fe0_maven_1_12</id>
  <username>...</username>
  <password>...</password>
</server>
<server>
  <id>snapshot_ap-southeast-3_080b8db696000fe60f27c009b9105fe0_maven_2_12</id>
  <username>...</username>
  <password>...</password>
</server>
```

需要使用ID为仓库地址最后一个参数的用户名和密码。

----结束

## 发布自研工具包 WebUtil

对于自行研发的工具包（需要以一定频率编译发布依赖包），推荐使用编译构建服务提供的“Maven构建”构建并发布私有依赖到私有依赖库，此方式具备以下优势：

- 工具包版本迭代时，可以一键发布，避免版本迭代带来的重复的手动上传依赖操作。
- 可结合构建任务、流水线的定时构建、合并代码触发等功能实现自动化持续集成。
- 使用构建任务发布的内容由Maven自动生成，可有效避免手动操作导致的文件缺失、损坏，保证上传依赖的完整性、有效性。

配置方法如下：

**步骤1** **新建构建任务**，其中，代码源选择代码仓库“WebUtil”，构建模板选择“Maven”，并将任务名称命名为“发布WebUtil到私有依赖库”。

**步骤2** Maven模板预置了“Maven构建”和“上传软件包到软件发布库”构建步骤以及配套的默认构建命令。本示例需删除“上传软件包到软件发布库”构建步骤。

**步骤3** 配置“Maven构建”步骤。

1. 注释默认的“mvn package”命令，开放被注释的“mvn deploy”命令。



2. 检查构建命令。模板已给出默认参数配置，此处只需确认参数正确即可。
  - 默认命令要从根目录读取“pom”文件，本例中WebUtil项目“pom”文件在根目录，无需修改。
  - WebUtil项目要求使用jdk1.8编译、运行，确认工具版本选择“maven3.5.3-jdk8-open”。
  - 本次构建目标为发布私有依赖包，默认命令为“mvn deploy”，已经开放。

## 说明

- “mvn deploy”命令作用为将项目打包，并发布到指定的依赖仓库，可供其他项目直接引用，必要时（无需发布到私有仓库或私有仓库故障）也可选择使用“mvn install”命令，将项目打包并install到构建缓存，同样可以供其他项目直接引用（构建时需要使用缓存，缓存内容不保证数据持久性，如丢失需重新构建）。
  - 默认命令参数说明可参考[Maven构建默认命令含义](#)。
3. 展开“发布依赖包到私有依赖库”，选择“配置所有pom”，选择需要发布的私有依赖仓库。



检查缓存配置：

- 编译构建提供了构建缓存加速功能，可通过[缓存配置](#)选择是否使用缓存。
- 由于网络抖动、并发构建或其他极端情况，可能出现缓存内容异常导致构建异常，此时需要参考[缓存清理](#)步骤清理缓存。

**步骤4** 构建任务执行成功后，进入“制品仓库 > 私有依赖库”即可查看到上传的依赖包。

----结束

## 使用 POM 模式手动上传依赖包 CommonUtil

部分情况下，项目中使用的依赖是以SystemPath方式引入，或者拿到第三方提供的Jar包手动上传到企业自建的私有依赖库中（如本例中CommonUtil包），该类依赖无法从公开仓库下载，且不适合使用编译构建发布，此时需要在私有依赖库手动上传，上传时优先使用POM模式，操作步骤如下：

**步骤1** 选择“制品仓库 > 私有依赖库”，根据要上传的依赖包类型选择制品类型。

**步骤2** 获取“pom”文件。

- 方式一：从原始仓库下载“pom”文件。

部分依赖可能来自于CodeArts无法访问的第三方仓库，用户自己可以从仓库下载，此类场景下Maven仓库一般会同时提供jar包和“pom”文件，用户直接从原仓库下载“pom”文件即可。

- 方式二：从jar包中获取“pom”文件。

因为各种原因，部分依赖可能只能找到jar包，原始仓库、源码、“pom”文件等都无法找到，此时可以参考以下步骤尝试获取“pom”文件（以WebUtil包为例）。

- 解压util-1.0.jar，如无法解压需先更改后缀为支持的压缩包格式。
- 进入解压后目录，打开“META-INF/maven/{groupid}/{artifactid}”目录，此处为“META-INF/maven/com.xx.demo/util”，打开该“pom”文件，确认无误即可直接使用。

- c. 如果确认无法找到“pom”文件，则需要考虑是否可以[使用GAV模式上传](#)。

**步骤3** 单击右上角“上传制品”，选择“POM模式”，选择“pom”文件和“jar”文件上传即可。

#### 说明

以WebUtil为例，手动上传需要注意：

此处是WebServer项目依赖WebUtil项目，上传WebUtil项目时，必须使用WebUtil项目的“pom”文件，如果误操作上传了WebServer项目的“pom”文件与WebUtil项目jar包，会导致上传依赖坐标与预期不一致，导致依赖下载失败。

----结束

## 使用 GAV 模式上传三方依赖 MessageSDK

优先使用POM模式手工上传依赖，若始终无法找到“pom”文件，则需要考虑使用GAV模式上传，但此方式存在一定隐患，使用前需要注意评估。

使用GAV模式的场景及风险说明：

- 使用GAV模式上传时，私有仓库会根据输入的坐标信息自动生成“pom”文件，文件内容只包含依赖自身坐标定义。
- 以WebUtil为例，如果WebUtil项目本身依赖了工具包lib.jar，使用GAV模式上传WebUtil后，会导致最终WebServer构建无法下载lib.jar，导致构建包与预期不符。
- 若WebUtil项目本身无任何依赖（“pom”文件的节点为空），则可以使用此模式上传。

如您已认真阅读以上风险说明，确保上传依赖无上述隐患或接受该风险，可按如下步骤操作：

**步骤1** 选择“制品仓库 > 私有依赖库”，根据要上传的依赖包类型选择仓库类型。

**步骤2** 单击右上角“上传制品”，选择“GAV模式”，根据界面提示填写坐标信息，选择jar包上传即可。

----结束

## 制作镜像并推送至 SWR 仓库

**步骤1** [创建组织](#)。制作镜像并推送到SWR仓库时，需要指定SWR组织名。

**步骤2** [新建构建任务](#)，其中，“代码源”选择“Repo”，“代码仓库”选择“WebServer”，构建模板选择“Maven”，并将任务名称命名为“使用WebServer制作Docker镜像”。

**步骤3** Maven模板预置了“Maven构建”和“上传软件包到软件发布库”构建步骤以及配套的默认构建命令。本示例需删除“上传软件包到软件发布库”构建步骤。

**步骤4** 配置“Maven构建”构建步骤，确认构建命令、缓存配置正确。

1. 检查构建命令：模板已给出默认参数配置，此处只需确认参数正确即可。
  - 默认命令要从根目录读取“pom”文件，本例中WebServer项目“pom”文件在根目录，无需修改。
  - WebServer项目要求使用jdk1.8编译、运行，确认工具版本选择“maven3.5.3-jdk8-open”。

- 本次构建目标为打包，默认命令为“mvn package”，无需调整，默认参数说明可参考[Maven构建默认命令含义](#)。
2. 检查缓存配置：
- 编译构建提供了构建缓存加速功能，用户可通过[缓存配置](#)选择是否使用缓存。
  - 由于网络抖动、并发构建或其他极端情况，可能出现缓存内容异常导致构建异常，此时需要参考[缓存配置](#)步骤清理缓存。

### 📖 说明

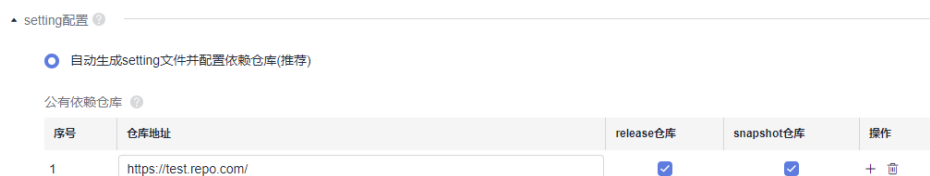
编译构建会自动配置华为开源镜像站作为开源依赖源，在使用编译构建时可自动下载，无需任何额外配置，开源镜像站已代理或同步的镜像源有：

- Maven2: <https://repo1.maven.org/maven2/>
- Jboss: <https://repository.jboss.org/nexus/content/repositories/releases/>
- Jcenter: <https://mvnrepository.com/repos/jcenter>
- Grails-core: <https://repo.grails.org/grails/core/>
- Grails-plugins: <https://repo.grails.org/grails/plugins/>
- Spring-release: <https://repo.spring.io/libs-release/>
- Spring-plugins: <https://repo.spring.io/plugins-release/>

### 步骤5 配置非CodeArts提供的公有依赖仓。

本例中假设WebServer引用了来自第三方仓库“<https://test.repo.com/>”的依赖lib.jar，需要在“Maven构建”中配置。

配置如下：



### 📖 说明

要使用此类仓库，该仓库必须满足以下条件：

- 仓库地址在公网（大陆）可直接访问。
- 依赖下载无需身份认证信息。

### 步骤6 配置私有依赖库。

在之前上传了WebUtil项目的构建包util-1.0.jar到私有依赖库，本例以该依赖为例，在“使用WebServer制作Docker镜像WebServer”任务中描述如何配置使用私有依赖库中的依赖。

1. 编辑本地代码目录WebServer中的“pom.xml”文件，在<dependencies></dependencies>中添加“util-1.0.jar”依赖。

```
<dependency>  
<groupId>com.xx.demo</groupId>
```



```
<artifactId>util</artifactId>
<version>1.0</version>
</dependency>
```

- 保存“pom.xml”后重新上传到“WebServer”代码仓库中。
- 在“Maven构建”步骤中选择[新建私有依赖扩展点](#)中已新建的私有依赖库扩展点。

**步骤7** 在“Maven构建”步骤后添加“制作镜像并推送到SWR仓库”构建步骤，并录入所需镜像信息。

- 镜像仓库：保持默认即可。
- 组织：填写[步骤1](#)中新建的组织名。
- 镜像名字：自定义，此处设置为“webserver”。
- 镜像标签：自定义，此处设置为“v1.1”。
- 工作目录：保持默认目录即可。
- Dockerfile路径：[上传代码至代码仓](#)中“Dockerfile”已存放于WebServer项目根目录，当前构建目录即为项目根目录，默认值“./Dockerfile”无需更改。

**步骤8** 保存并执行任务。

----结束

## 查看构建结果

**步骤1** 进入[容器镜像服务SWR](#)，选择对应region。



**步骤2** 单击导航栏“我的镜像”，选择在“制作镜像并推送到SWR仓库”构建步骤中填写的组织名，即可查看上传的镜像。



----结束

## Maven 构建默认命令含义

构建服务内置的默认构建命令为

```
# 功能：打包
# 参数说明：
# -Dmaven.test.skip=true：跳过单元测试
# -U：每次构建检查依赖更新，可避免缓存中快照版本依赖不更新问题，但会牺牲部分性能
# -e -X：打印调试信息，定位疑难构建问题时建议使用此参数构建
# -B：以batch模式运行，可避免日志打印时出现ArrayIndexOutOfBoundsException异常
# 使用场景：打包项目且不需要执行单元测试时使用
mvn package -Dmaven.test.skip=true -U -e -X -B
```

其中，各命令/参数含义为：



- mvn package：使用maven执行打包动作，此命令会在项目target目录下生成软件包，可根据需要自行调整目录。
- -Dmaven.test.skip=true：跳过单元测试，建议保留。
- -U：每次构建检查依赖更新，可避免缓存中快照版本依赖不更新问题，但会牺牲部分性能，建议保留。
- -e -X：打印调试信息，定位疑难构建问题时建议使用此参数构建。
- -B：以batch模式运行，可避免日志打印时出现ArrayIndexOutOfBoundsException异常

## 什么是构建缓存，缓存异常时怎么清理？

编译构建提供了构建缓存功能，构建时可将依赖缓存于用户私有存储空间，下次构建时直接使用，无需重复下载，可极大提高构建效率。

- 构建缓存配置

新建编译构建任务时，默认选择使用缓存加速构建，用户可在“Maven构建”中展开“缓存配置”选择是否使用缓存。

- 清理缓存

由于网络抖动、并发构建或其他极端情况，可能出现缓存内容异常导致构建异常，下面介绍异常缓存的清理过程。

执行缓存清理操作前，请务必仔细阅读以下缓存清理风险以及注意事项：

- 由于缓存目录为同租户共享，频繁清理缓存会概率性导致同租户用户构建异常（常表现为某文件不存在），故此操作只可在缓存异常时清理一次，正常后需要务必再次编辑任务，删除清理命令。
- 清理缓存时尽可能使用精确的文件路径，如：清理demo 1.0.0版本，请使用“rm -rf /path/com/xx/demo/1.0.0”，尽量避免删除目录层级过高，导致下次构建缓慢或因网络问题导致依赖异常。
- 出于安全考虑，缓存清理命令只可在“Maven构建”步骤执行，在其他步骤执行此命令会导致“目录不存在”或清理无效。

如您已认真阅读以上风险说明，确理解且接受该风险，可按如下步骤清理缓存。

a. 准备清理命令。

- 缓存清理命令格式为：rm -rf /repository/local/maven/{Group ID}/{Artifact ID}/{Version}。
- 其中，需要的参数分别对应依赖坐标中的GroupID、ArtifactID、version。

若依赖如下：

```
<dependency>
  <groupId>com.xx.devcloud</groupId>
  <artifactId>demo</artifactId>
  <version>1.0.9-SNAPSHOT</version>
</dependency>
```

那么，清理该依赖所需命令为：rm -rf /repository/local/maven/com/xx/devcloud/demo/1.0.9-SNAPSHOT。

b. 编辑构建任务，配置“Maven构建”步骤。

c. 找到mvn xxxx命令位置，在此命令之前新增一行，填入准备好的清理命令，保存任务。

- d. 重新执行构建任务。
- e. 成功后再次编辑任务，移除清理缓存命令。

# 2 使用 NPM 构建自定义镜像（内置执行机，图形化构建）

## 应用场景

本实践帮助您了解如何通过编译构建服务的内置执行机，并以图形化构建的方式编译 Node.js项目并上传软件包至软件发布库。

本实践需要依赖使用的其它服务如下：

- [代码托管服务](#)，用于存储实践中项目所使用的代码。
- [制品仓库服务](#)，用于存储实践中使用的私有依赖包。

## 前提条件

- 已在容器镜像服务中[创建组织](#)。
- 需已具备CodeArts Artifact服务的操作权限，具体操作可参考[授权使用CodeArts Artifact服务](#)。
- 需已具备CodeArts Repo服务的操作权限，具体操作可参考[授权使用CodeArts Repo服务](#)。


## 操作流程

表 2-1 操作流程

流程	说明
<a href="#">新建项目</a>	为本实践新建项目。
<a href="#">新建CodeArts Repo代码仓以及代码文件</a>	为本实践新建Repo代码仓和实践中使用的代码文件。
<a href="#">新建构建任务</a>	为本实践新建编译构建任务。
<a href="#">配置构建步骤并执行构建任务</a>	按照本实践场景配置构建步骤并执行构建任务。

流程	说明
<a href="#">查看并验证构建结果</a>	查看并验证构建结果。

新建项目

- 步骤1 使用华为云账号[登录华为云控制台页面](#)。
- 步骤2 单击页面左上角，在服务列表中选择“开发与运维 > 编译构建 CodeArts Build”。
- 步骤3 单击“立即使用”，进入编译构建服务首页。
- 步骤4 单击导航栏“首页”，在首页单击“新建项目”，选用“Scrum”项目模板。
- 步骤5 项目名称填写“build-bestpraticce”，其它保持默认即可。
- 步骤6 单击“确定”后，进入到“build-bestpraticce”项目下。
- 结束

新建 CodeArts Repo 代码仓以及代码文件

- 步骤1 在导航栏选择“代码 > 代码托管”。
- 步骤2 进入代码托管页面，单击“新建仓库”，选择“普通仓库”。
- 步骤3 单击“下一步”，在新建代码仓页面，“代码仓库名称”命名为“nodesource”，其他参数保持默认即可。
- 步骤4 单击“确定”，进入代码仓详情页。
- 步骤5 在代码仓详情页，单击“新建 > 新建文件”，在代码仓的根目录下新建名为“package.json”的文件，代码示例如下。

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.16.1"
  }
}
```

- 步骤6 参考[步骤5](#)，新建名为“server.js”的文件，代码示例如下。

```
'use strict';
const express =require('express');
// Constants
const PORT=8080;
const HOST='0.0.0.0';
// App
const app =express();
app.get('/',(req, res)=>{
```

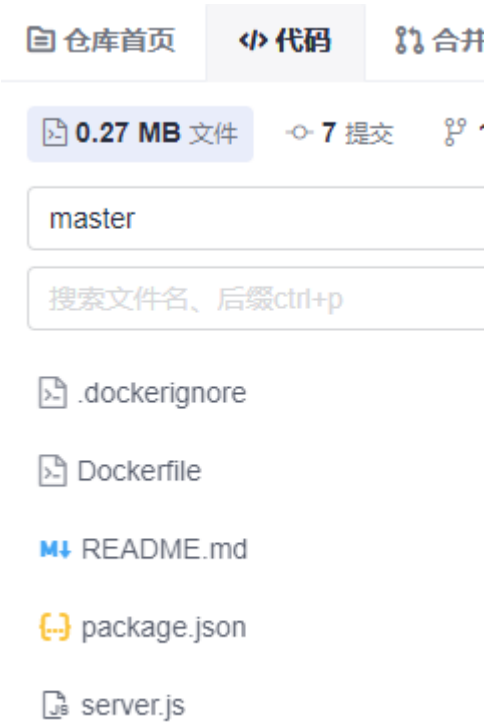
```
res.send('Hello world\n');
});
app.listen(PORT,HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

步骤7 参考步骤5，新建名为“.dockerignore”的文件，代码示例如下。

```
node_modules
npm-debug.log
```

代码文件新建完成后，文件目录如图2-1所示。

图 2-1 文件目录



----结束

新构建任务

- 步骤1 在导航栏选择“持续交付 > 编译构建”。
- 步骤2 单击“新建任务”，根据表2-2填写参数信息，单击“下一步”。

表 2-2 基本信息配置

参数名称	参数说明	是否必填
任务名称	自定义任务名称，例如：maven_yml_build。	是
代码源	选择“Repo”。	是

参数名称	参数说明	是否必填
代码仓	选择新建CodeArts Repo代码仓以及代码文件中新建的代码仓名称，选择“nodesource”。	是
默认分支	选择默认“master”即可。	是
任务描述	对该构建任务的描述。	否

步骤3 选择“npm”模板，单击“确定”，进入构建步骤配置页面。

----结束

配置构建步骤并执行构建任务

步骤1 配置“Npm构建”。

在命令编辑器里，npm run build命令前加“#”，新增zip -r ./nodeserver.zip ./命令，用来将代码打包成“nodeserver.zip”，如图2-2所示。其他参数保持默认即可。

图 2-2 命令示例

```
23 #npm run build
24 zip -r ./nodeserver.zip ./
25 #tar -zcvf demo.tar.gz ./**
```

步骤2 按图2-3所示配置“上传软件包到软件发布库”。

图 2-3 配置上传软件包到软件发布库

\* 步骤显示名称

上传软件包到软件发布库

\* 构建包路径 ?

./nodeserver.zip

发布版本号 ?

v0.0.1

包名 ?

nodeserver

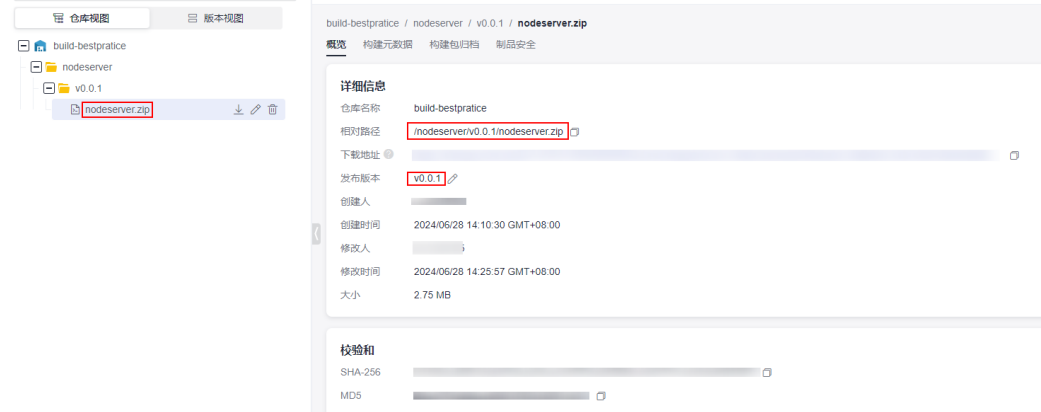
**步骤3** 配置完所有构建步骤，单击“保存并执行”，执行编译构建任务。

----结束

查看并验证构建结果

在导航栏选择“制品仓库 > 软件发布库”，查看上传的软件包，如图2-4所示。

图 2-4 查看上传的软件包



包名和发布版本与步骤2中配置的包名和发布版本号一致。

# 3 使用 Maven 构建上传软件包和推送镜像到 SWR（内置执行机，预置镜像，代码化构建）

## 应用场景

编译构建服务支持通过yaml文件配置构建脚本，用户可以将构建时需要配置的构建环境、构建参数、构建命令、构建步骤等操作，通过yaml语法编写成build.yml文件实现，并且将build.yml文件和被构建的代码一起存储到代码仓库。执行构建任务时，系统会以build.yml文件作为构建脚本执行构建任务，使构建过程可追溯、可还原，安全可靠。本实践以使用Maven构建为例，为您演示上传软件包至软件发布库和推送镜像到SWR。

本实践需要依赖使用的其它服务如下：

- SWR，即[容器镜像服务](#)。SWR镜像仓库用于存储用户上传的Docker镜像，可以在构建、部署或其他场景使用。
- [代码托管服务](#)，用于存储实践中项目所使用的代码。
- [制品仓库服务](#)，用于存储实践中使用的私有依赖包。

## 前提条件

- 已在容器镜像服务中[创建组织](#)。
- 需已具备CodeArts Artifact服务的操作权限，具体操作可参考[授权使用CodeArts Artifact服务](#)。
- 需已具备CodeArts Repo服务的操作权限，具体操作可参考[授权使用CodeArts Repo服务](#)。

## 约束与限制

代码化构建仅支持使用CodeArts Repo中的代码。




操作流程

表 3-1 操作流程

流程	说明
新建项目	为本实践新建项目。
新建CodeArts Repo代码仓	为本实践新建Repo代码仓用于存储代码文件。
新建build.yml文件	通过“build.yml”定义整个构建的流程。
新建Dockerfile文件	通过修改Dockerfile文件实现自定义镜像。
新建编译构建任务	新建本实践的编译构建任务。
查看并验证构建结果	查看并验证构建结果。

新建项目

- 步骤1 使用华为云账号[登录华为云控制台页面](#)。
- 步骤2 单击页面左上角，在服务列表中选择“开发与运维 > 编译构建 CodeArts Build”。
- 步骤3 单击“立即使用”，进入编译构建服务首页。
- 步骤4 单击导航栏“首页”，在首页单击“新建项目”，选用“Scrum”项目模板。
- 步骤5 项目名称填写“build-bestpraticce”，其它保持默认即可。
- 步骤6 单击“确定”后，进入到“build-bestpraticce”项目下。
- 结束

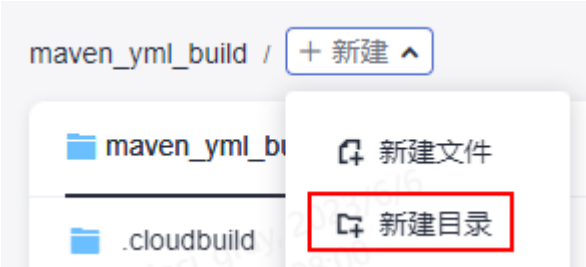
新建 CodeArts Repo 代码仓

- 步骤1 在导航栏选择“代码 > 代码托管”。
- 步骤2 单击“新建仓库”，选择“模板仓库”，单击“下一步”。
- 步骤3 选择“Java Maven Demo”模板，单击“下一步”。
- 步骤4 在新建代码仓页面，“代码仓库名称”命名为“maven\_yaml\_build”，其他参数保持默认即可。
- 步骤5 单击“确定”，进入代码仓详情页。
- 结束

新建 build.yml 文件

步骤1 在代码仓详情页，单击“新建 > 新建目录”，如图3-1所示。

图 3-1 新建目录



步骤2 在新建目录页面，根据表3-2填写参数信息，单击“确定”。

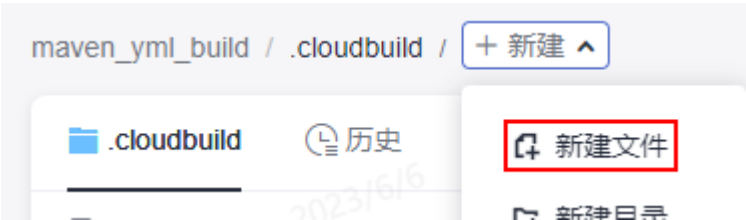
表 3-2 新建目录

参数名称	参数说明	是否必填
目录名称	填写目录名称，例如“.cloudbuild”。文件目录名称仅支持中文，英文字母，数字，单斜杠“/”，下划线“_”，中横线“-”和点“.”，字符长度范围为1~100。	是
提交信息	目录的备注信息，用于记录该文件夹文件的描述信息。字符长度范围1~2000。	是

步骤3 单击步骤2中新建的目录名称。

步骤4 单击“新建 > 新建文件”，如图3-2所示。

图 3-2 新建文件



步骤5 文件命名为“build.yml”，并将如下代码拷贝到文件中。

```
# This YAML is the default template and can be modified based on this
---
version: 2.0
steps:
  BUILD:
    - maven:
        image: cloudbuild@maven3.5.3-jdk8-open # 可以自定义镜像地址
        inputs:
          settings:
            public_repos:
              - https://mirrors.huawei.com/maven
```

```
cache: true # 是否开启缓存
command: mvn package -Dmaven.test.failure.ignore=true -U -e -X -B
- upload_artifact:
  inputs:
    path: "**/target/*.?ar"
- build_image:
  inputs:
    organization: codeci_gray # 组织名称
    image_name: maven_demo # 镜像名称
    image_tag: 1.0 # 镜像版本
    dockerfile_path: ./Dockerfile
```

步骤6 单击“确定”。

----结束

新建 Dockerfile 文件

步骤1 在根目录下，参考步骤4新建名为“Dockerfile”的文件。文件中代码如下：

```
FROM swr.cn-north-5.myhuaweicloud.com/codeci/special_base_image:centos7-base-1.0.2
MAINTAINER <devcloud@demo.com>
USER root
RUN mkdir /demo
COPY ./target/server-1.0.jar /demo/app.jar
```

其中server-1.0.jar为“pom.xml”文件中artifactId、packaging和version的参数值组合。

步骤2 单击“确定”。

----结束

新建编译构建任务

步骤1 在导航栏选择“持续交付 > 编译构建”。

步骤2 单击“新建任务”，根据表3-3填写参数信息。

表 3-3 基本信息配置

参数名称	参数说明	是否必填
任务名称	自定义任务名称，例如：maven_yml_build。	是
代码源	选择“Repo”。	是
代码仓	选择新建CodeArts Repo代码仓中创建的代码仓库名称“Repo01”。	是
默认分支	选择默认“master”即可。	是
任务描述	对该构建任务的描述。	否

步骤3 单击“下一步”，选择“空白构建模板”。

步骤4 单击“确定”，进入构建步骤配置页面。

步骤5 单击“代码化”页签，可查看到导入的构建脚本，如图3-3所示。

图 3-3 代码化页签



步骤6 单击页面右上角的“保存并执行”。

----结束

查看并验证构建结果

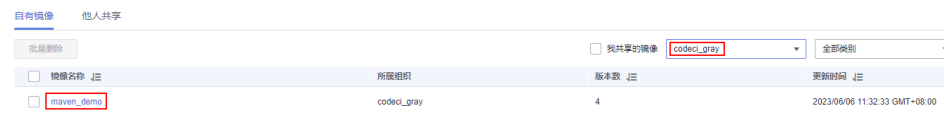
- 查看上传的软件包。
  - 选择页面导航栏“制品仓库 > 软件发布库”。
  - 在软件发布库查看发布的软件包。软件包与新建编译构建任务时的任务名称一致，如图3-4所示。

图 3-4 查看软件包



- 查看推送的镜像。
  - 进入容器镜像服务SWR。
  - 单击导航栏“我的镜像”，在组织中筛选新建build.yml文件时代码中填写的“组织名称”，如：codeci\_gray。
  - 在筛选结果中单击新建build.yml文件时代码中填写的“镜像名称”，如：maven\_demo，如图3-5所示。

图 3-5 筛选镜像



# 4 使用 Maven 构建执行多任务构建工程（内置执行机，预置镜像，代码化构建）

## 应用场景

在编译构建中，构建任务是构建的最小单元，适用于业务比较简单的场景，但是在有些复杂的构建场景下，构建任务可能并不能满足复杂的构建要求。例如，用户希望更模块化、更加细粒度的拆分构建任务，并按照构建任务之间的依赖顺序进行构建。

为此，编译构建服务支持使用BuildFlow将多个存在依赖关系的构建任务按照有向无环图（DAG）的方式组装起来，BuildFlow将会按照构建的依赖关系并发进行构建。

本实践为您演示构建任务Job3依赖于构建任务Job1和构建任务Job2的构建工程，实践中依赖使用[代码托管服务](#)，用于存储实践中项目所使用的代码。

## 约束与限制

- 使用BuildFlow构建仅支持使用CodeArts Repo中的代码。
- 需要您具备CodeArts Repo服务的操作权限，具体操作可参考[授权使用CodeArts Repo服务](#)。


## 操作流程

表 4-1 操作流程

流程	说明
<a href="#">新建项目</a>	为本实践新建项目。
<a href="#">新建CodeArts Repo代码仓</a>	为本实践新建Repo代码仓用于存储代码文件。
<a href="#">新建build.yml文件</a>	通过“build.yml”定义整个构建的流程。
<a href="#">新建build.yml中使用的子任务执行脚本</a>	新建整个构建过程中依赖的构建任务的执行脚本。
<a href="#">新建并执行编译构建任务</a>	新建BuildFlow编译构建任务并执行。

流程	说明
<a href="#">查看编译构建结果</a>	查看编译构建结果。

新建项目

- 步骤1 使用华为云账号[登录华为云控制台页面](#)。
- 步骤2 单击页面左上角，在服务列表中选择“开发与运维 > 编译构建 CodeArts Build”。
- 步骤3 单击“立即使用”，进入编译构建服务首页。
- 步骤4 单击导航栏“首页”，在首页单击“新建项目”，选用“Scrum”项目模板。
- 步骤5 项目名称填写“build-bestpratice”，其它保持默认即可。
- 步骤6 单击“确定”后，进入到“build-bestpratice”项目下。
- 结束

新建 CodeArts Repo 代码仓

- 步骤1 在页面导航栏选择“代码 > 代码托管”。
- 步骤2 单击“新建仓库”，选择“模板仓库”，单击“下一步”。
- 步骤3 选择“Java Maven Demo”模板，单击“下一步”。
- 步骤4 填写代码仓库名称为“Repo01”，其他参数保持默认即可。
- 步骤5 单击“确定”。
- 结束

新建 build.yml 文件

- 步骤1 在代码仓详情页，选择“新建 > 新建目录”。
- 步骤2 目录名称填写“.cloudbuild”，描述信息自定义即可。
- 步骤3 单击“确定”。
- 步骤4 在“.cloudbuild”目录下，选择“新建 > 新建文件”，文件名命名为“build.yml”，文件中代码内容如下。

```
version: 2.0 # 必须是2.0, 该版本号必填且唯一
params: # 构建参数, 可在构建过程中引用
  - name: condition_param
    value: 1
# envs配置为非必填项。
envs:
  - condition: condition_param == 0 # 主机规格与类型的判断条件, 不满足条件则不使用以下主机规格与类型
    resource:
      type: docker
      arch: ARM
  - condition: condition_param == 1 # 主机规格与类型的判断条件, 满足条件会使用以下主机规格与类型
    resource:
```

```
type: docker
arch: X86

buildflow:
  jobs: # 构建任务
    - job: Job3 # 子任务的名称，可自定义
      depends_on: # 定义job的依赖,此处表示Job3依赖Job1和Job2
        - Job1
        - Job2
      build_ref: .cloudbuild/build_job3.yml # 定义Job3在构建过程中需要运行的yaml构建脚本
    - job: Job1
      build_ref: .cloudbuild/build_job1.yml # 定义Job1在构建过程中需要运行的yaml构建脚本
    - job: Job2
      build_ref: .cloudbuild/build_job2.yml # 定义Job2在构建过程中需要运行的yaml构建脚本
```

“build.yml”定义了整个构建的流程，当前定义了3个构建任务，Job3依赖于Job1和Job2，即构建优先级Job1、Job2 > Job3，且Job1和Job2优先级相同会同步触发。  
“build\_ref”定义了构建子任务所需运行的构建脚本。

----结束

## 新建 build.yml 中使用的子任务执行脚本

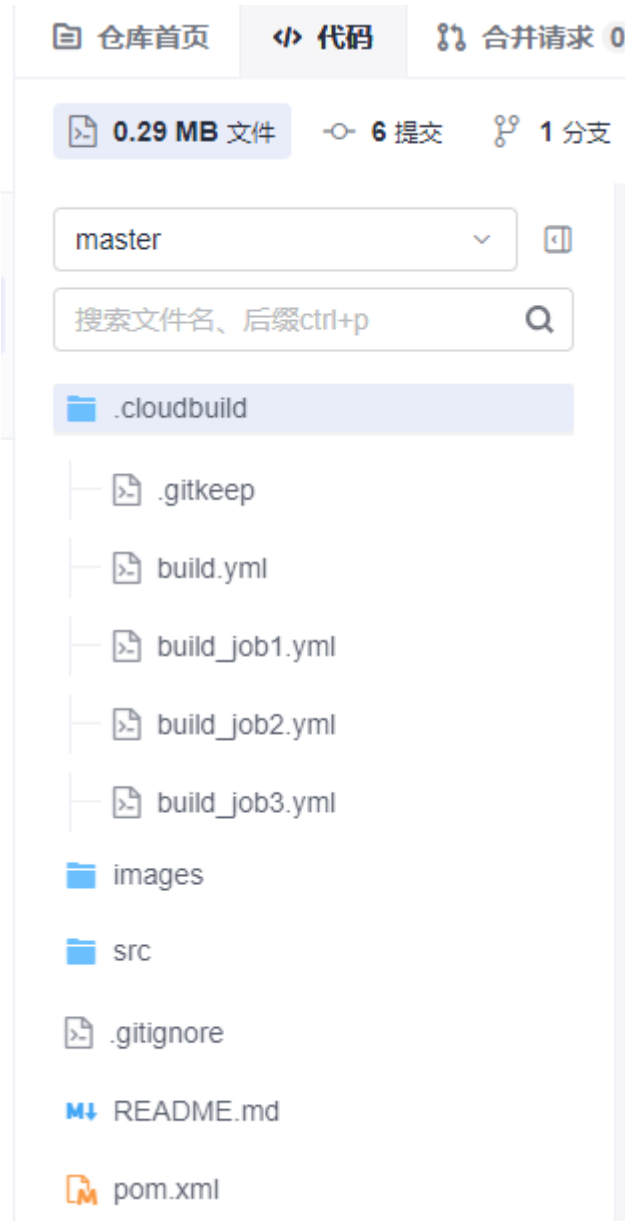
**步骤1** 在“.cloudbuild”目录下，选择“新建 > 新建文件”，文件名命名为“build\_job1.yml”，文件中代码示例如下。

```
version: 2.0
steps:
  BUILD:
    - maven:
        image: cloudbuild@maven3.5.3-jdk8-open # 使用的构建镜像，用户可以自定义镜像
        inputs:
          settings:
            public_repos:
              - https://mirrors.huawei.com/maven # 配置依赖仓
        cache: true # 是否开启缓存
        command: mvn package -Dmaven.test.failure.ignore=true -U -e -X -B # 执行命令
```

**步骤2** 参考**步骤1**新建“build\_job2.yml”和“build\_job3.yml”，代码示例保持一致即可。

**步骤3** 文件新建完成后，代码仓文件目录如下图所示。

图 4-1 文件目录



----结束

新建并执行编译构建任务

- 步骤1 在页面导航中选择“持续交付 > 编译构建”。
- 步骤2 单击“新建任务”，根据表4-2填写参数信息。



表 4-2 基本信息配置

参数名称	参数说明	是否必填
任务名称	自定义任务名称，例如：BuildFlow。	是
代码源	选择“Repo”。	是
代码仓	选择新建CodeArts Repo代码仓中新建的代码仓库名称“Repo01”。	是
默认分支	选择默认“master”即可。	是
任务描述	对该构建任务的描述。	否

- 步骤3 单击“下一步”，选择“Maven”模板。
- 步骤4 单击“确定”，进入构建步骤配置页面。
- 步骤5 单击“代码化”页签，会自动加载“Repo01”代码仓中的构建运行脚本。
- 步骤6 单击“保存并执行”，在弹出的窗口中单击“确定”即可跳转到构建任务运行页面。

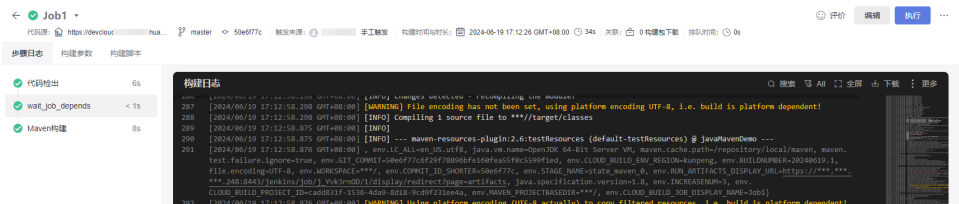
----结束

查看编译构建结果

“构建流程”页签中展示了当前构建任务运行的全量流程图，在构建任务未执行完成时，可以清晰的看到Job1和Job2是并行执行，Job3是等待Job1和Job2执行完成后才执行。

- 步骤1 单击“构建流程”页签中左侧菜单的“Job1”或右侧界面的绿色矩形图形的“Job1”，进入Job1构建任务的执行详情页面，可以查看Job1构建任务的构建日志，如图4-2所示。

图 4-2 查看构建结果



- 其中，
- “步骤日志”页签展示了当前构建任务运行顺序和资源调度情况。
  - “构建参数”页签展示了当前构建任务全局的参数信息。
  - “构建脚本”页签展示了当前构建任务执行的脚本内容。

**步骤2** 参考[步骤1](#)可查看“Job2”和“Job3”的构建任务执行详情。

----结束

# 5 基于私有依赖库使用 Maven 构建并上传软件包（内置执行机，预置镜像，图形化构建）

## 应用场景

当CodeArts Build提供的默认依赖库不满足业务要求时，用户可使用自己搭建的私有依赖库进行Maven构建。

本实践需要依赖使用的其它服务如下：

- [代码托管服务](#)，用于存储实践中项目所使用的代码。
- [制品仓库服务](#)，用于存储实践中使用的私有依赖包。

## 前提条件

- 需已具备CodeArts Artifact服务的操作权限，具体操作可参考[授权使用CodeArts Artifact服务](#)。
- 需已具备CodeArts Repo服务的操作权限，具体操作可参考[授权使用CodeArts Repo服务](#)。


## 操作流程

表 5-1 操作流程

流程	说明
<a href="#">新建项目</a>	为本实践新建项目。
<a href="#">新建私有依赖库</a>	新建本实践中使用的私有依赖库。
<a href="#">查询私有依赖仓库信息</a>	查询私有依赖库的id和url信息，用于配置在代码仓的“pom.xml”文件中。
<a href="#">上传settings.xml文件至编译构建</a>	上传“settings.xml”文件到编译构建服务的“文件管理”中。
<a href="#">新建CodeArts Repo代码仓</a>	新建本实践需要使用的代码仓。

流程	说明
配置Maven构建产物发布的私有依赖库地址	配置构建产物上传的私有依赖库的地址。
新建编译构建任务	新建本实践需要使用的编译构建任务。
配置构建步骤并执行构建任务	配置“下载文件管理的文件”和“Maven构建”步骤并执行构建任务。
查看编译构建结果	在私有依赖库中查看编译构建结果。

新建项目

- 步骤1 使用华为云账号[登录华为云控制台页面](#)。
- 步骤2 单击页面左上角，在服务列表中选择“开发与运维 > 编译构建 CodeArts Build”。
- 步骤3 单击“立即使用”，进入编译构建服务首页。
- 步骤4 单击导航栏“首页”，在首页单击“新建项目”，选用“Scrum”项目模板。
- 步骤5 项目名称填写“build-bestpraticce”，其它保持默认即可。
- 步骤6 单击“确定”后，进入到“build-bestpraticce”项目下。
- 结束

新建私有依赖库

- 步骤1 选择导航栏“制品仓库 > 私有依赖库”。
- 步骤2 单击“新建”，按照[如下表格](#)配置参数。

表 5-2 新建私有依赖库参数说明

参数名称	参数说明	是否必填
仓库类型	选择“本地仓”。	是
仓库名称	自定义仓库名称，例如“private_repository”。	是
制品类型	选择“Maven”。	是
归属项目	默认填写为“build-bestpraticce”，无需手动填写。	是
添加路径白名单	本实践不涉及，无需填写。	否

参数名称	参数说明	是否必填
版本策略	选择发布的版本，Release（功能稳定的发行版本）或者 Snapshot（功能不稳定、处于开发阶段中的快照版本）。本实践选择“Release”。	是
描述	自定义描述信息。最多200个字符。	否

步骤3 单击“确定”，进入到private\_repository依赖库的详情页面。

----结束

查询私有依赖仓库信息

- 步骤1 单击页面右上角“操作指导”。
- 步骤2 在弹出的窗口中保持默认选项，单击“下载配置文件”。
- 步骤3 在弹出的窗口中单击“下载”。

图 5-1 下载配置文件



步骤4 打开下载到本地的“settings.xml”文件，找到“<profile>”节点下定义的仓库信息<repository>中的“id”和“url”，并记录。

图 5-2 查看仓库的 id 和 url

```
</profile>
-->
<profile>
  <id>MyProfile</id>
  <repositories>
    <repository>
      <id>release_cn-north-4_f9e40463c23845438ca9efd3a7ec854e_maven_1_29</id>
      <url>https://devrepo.devcloud.cn-north-4.huaweicloud.com/artgalaxy/cn-north-4_f9e40463c23845438ca9efd3a7ec854e_maven_1_29/</url>
    </repository>
  </repositories>
  <enabled>true</enabled>
</profile>
<repositories>
  <repository>
    <id>release_cn-north-4_f9e40463c23845438ca9efd3a7ec854e_maven_1_29</id>
    <url>https://devrepo.devcloud.cn-north-4.huaweicloud.com/artgalaxy/cn-north-4_f9e40463c23845438ca9efd3a7ec854e_maven_1_29/</url>
  </repository>
</repositories>
<enabled>false</enabled>
</profile>
```

----结束

## 上传“settings.xml”文件至编译构建

步骤1 选择导航栏“持续交付 > 编译构建”。

步骤2 在编译构建任务列表页选择“更多 > 文件管理”。

步骤3 单击“上传文件”。

步骤4 在弹出的窗口中，上传[查询私有依赖仓库信息](#)中下载的“settings.xml”文件，勾选协议后单击“保存”。

----结束

## 新建 CodeArts Repo 代码仓

步骤1 选择导航栏“代码 > 代码托管”。

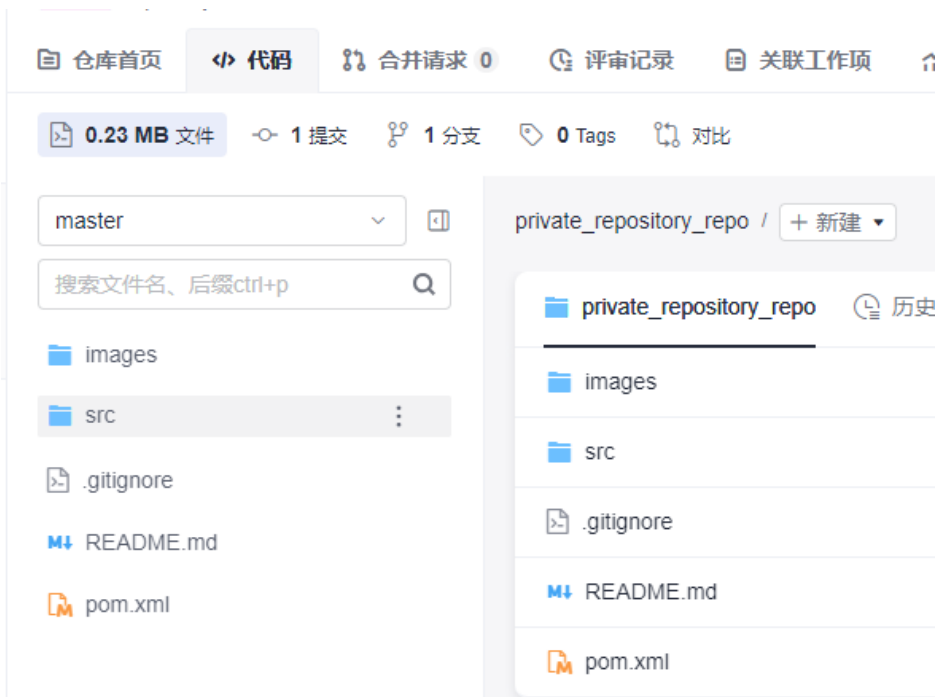
步骤2 单击“新建仓库”，选择“模板仓库”，单击“下一步”。

步骤3 选择“Java Maven Demo”模板，单击“下一步”。

步骤4 在按模板新建页面，“代码仓库名称”命名为“private\_repository\_repo”，其它参数保持默认即可。

步骤5 单击“确定”，新建后代码仓文件目录如[图5-3](#)所示。

图 5-3 文件目录



----结束

配置 Maven 构建产物发布的私有依赖库地址


- 步骤1 单击“pom.xml”文件，在右侧区域单击 进入到文件编辑模式。
- 步骤2 将如下代码示例粘贴到build标记下方。

图 5-4 pom 文件代码示例



其中“id”和“url”为步骤4中查看到的“id”和“url”。

- 步骤3 单击“确定”。

----结束

## 新建编译构建任务

**步骤1** 选择导航栏“持续交付 > 编译构建”。

**步骤2** 单击“新建任务”，按照如下参数说明配置参数，其它参数保持默认即可。

- 名称：自定义，例如“private\_repository\_task”。
- 代码源：选择“Repo”。
- 代码仓：选择新建CodeArts Repo代码仓中新建的代码仓“private\_repository\_repo”。

**步骤3** 单击“下一步”，选择“空白构建模板”。

**步骤4** 单击“确定”，进入到构建步骤配置页面。

----结束

## 配置构建步骤并执行构建任务

**步骤1** 单击“点击添加构建步骤”，添加“下载文件管理的文件”构建步骤，“步骤显示名称”和“工具版本”保持默认，“下载文件”选择[上传settings.xml文件至编译构建](#)中上传的文件“settings.xml”。

**步骤2** 单击“点击添加构建步骤”，添加“Maven构建”构建步骤，“命令”窗口中mvn package -Dmaven.test.skip=true -U -e -X -B命令前加“#”注释，删除mvn deploy -Dmaven.test.skip=true -U -e -X -B前的“#”，并将mvn deploy -Dmaven.test.skip=true -U -e -X -B改为mvn deploy -Dmaven.test.skip=true -s settings.xml -U -e -X -B，其它参数保持默认即可。

图 5-5 打包命令

```
# 使用场景：打包项目且不需要执行单元测试时使用
#mvn package -Dmaven.test.skip=true -U -e -X -B
```

图 5-6 发布依赖包命令

```
#功能：打包并发布依赖包到私有依赖库
#使用场景： 需要将当前项目构建结果发布到私有依赖仓库以供其它maven项目引用时使用
#注意事项： 此处上传的目标仓库为CodeArts私有依赖仓库，注意与软件发布仓库区分
mvn deploy -Dmaven.test.skip=true -s settings.xml -U -e -X -B
```

**步骤3** 单击“保存并执行”。

**步骤4** 在弹出的窗口中单击“确定”，等待构建任务执行完成。

----结束

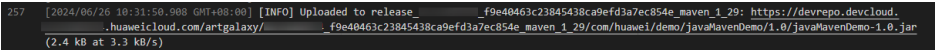
## 查看编译构建结果

**步骤1** 单击构建任务名称“private\_repository\_task”。

**步骤2** 在“构建历史”页签单击构建编号，查看步骤日志如下信息，其中“com/huawei/demo/javaMavenDemo/1.0”为构建产物在私有依赖库“private\_repository”中上传的路径。

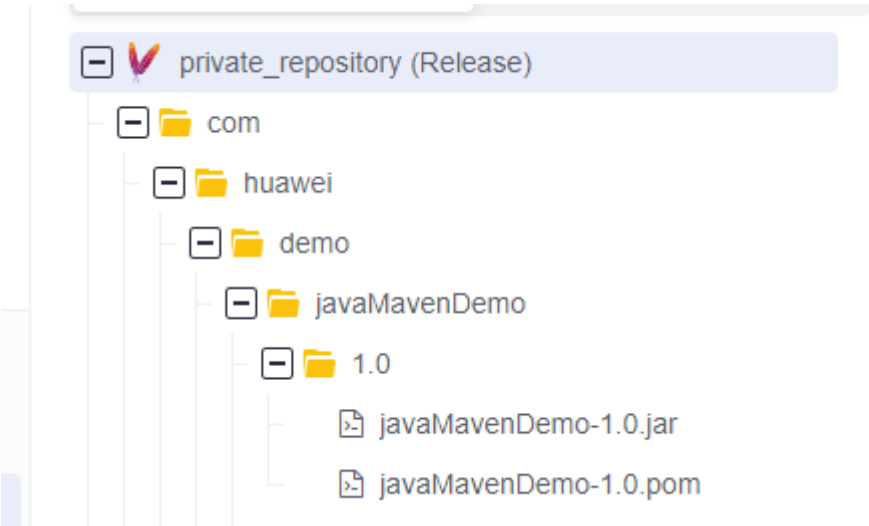


图 5-7 依赖包上传路径



- 步骤3** 选择导航栏“制品仓库 > 私有依赖库”。
- 步骤4** 展开“private\_repository”，打开“com/huawei/demo/javaMavenDemo/1.0”，可看到本实践上传的软件包。

图 5-8 查看上传的软件包



----结束

# 6 HE2E DevOps 实践：构建应用部分

本文以“[DevOps全流程示例项目](#)”为例，介绍如何在项目中配置构建任务，以及通过代码变更触发自动构建来实现持续集成。

开展实践前，需要完成[代码检查](#)。

## 预置任务简介

样例项目中预置了以下5个构建任务。

表 6-1 预置任务

预置任务	任务说明
phoenix-sample-ci	基本的构建任务。
phoenix-sample-ci-test	构建测试环境可用镜像的任务。
phoenix-sample-ci-worker	构建Worker功能镜像的任务。
phoenix-sample-ci-result	构建Result功能镜像的任务。
phoenix-sample-ci-vote	构建Vote功能镜像的任务。

本章节以任务“phoenix-sample-ci”为例进行讲解，此任务包含的步骤如下。


表 6-2 构建步骤

构建步骤	说明
制作Vote镜像并推送到SWR仓库	依据代码仓库中的“vote/Dockerfile”文件制作Vote功能镜像，并将镜像推送到容器镜像服务。

构建步骤	说明
制作Result镜像并推送到SWR仓库	依据代码仓库中的“result/Dockerfile”文件制作并推送Result功能镜像，并将镜像推送到容器镜像服务。
使用Maven安装Worker依赖包	使用Maven安装Worker功能所需的依赖。
制作Worker镜像并推送到SWR仓库	依据代码仓库中的“worker/Dockerfile”文件制作并推送Worker功能镜像，并将镜像推送到容器镜像服务。
生成Postgres and Redis Dockerfile	通过shell命令生成Dockerfile文件，用以制作Postgres（数据库）和Redis（缓存）镜像。
制作Postgres镜像并推送到SWR仓库	依据“生成Postgres and Redis Dockerfile”步骤中所生成的Dockerfile文件制作Postgres镜像，并将镜像推送到容器镜像服务。
制作Redis镜像并推送到SWR仓库	依据“生成Postgres and Redis Dockerfile”步骤中所生成的Dockerfile文件制作Redis镜像，并将镜像推送到容器镜像服务。
替换Docker-Compose部署文件镜像版本	为了将镜像部署到ECS时，能够可以拉取到正确的镜像，使用shell命令进行完成以下操作。 <ol style="list-style-type: none"><li>1. 使用sed命令，依次将文件“docker-compose-standalone.yml”中的参数替换为构建任务的参数“dockerServer”、“dockerOrg”、“BUILDNUMBER”进行替换。</li><li>2. 使用tar命令，将文件“docker-compose-standalone.yml”压缩为“docker-stack.tar.gz”，将部署所需文件进行打包，以便于后续步骤将该文件上传归档。</li></ol>
替换Kubernetes部署文件镜像版本	为了将镜像部署到CCE时，能够可以拉取到正确的镜像，使用shell命令进行完成以下操作。 <ol style="list-style-type: none"><li>1. 使用sed命令，将代码仓库中目录“kompose”下所有以“deployment”结尾的文件中的参数“docker-server”、“docker-org”，替换为构建任务的参数“dockerServer”、“dockerOrg”。</li><li>2. 使用sed命令，将代码仓库中“result-deployment.yaml”、“vote-deployment.yaml”、“worker-deployment.yaml”三个文件中的参数“image-version”用构建任务参数“BUILDNUMBER”进行替换。</li></ol>
上传Kubernetes部署文件到软件发布库	将“替换Kubernetes部署文件镜像版本”步骤中修改后的所有“.yaml”文件上传到软件发布库中归档。
上传docker-compose部署文件到软件发布库	将“替换Docker-Compose部署文件镜像版本”步骤中压缩好的“docker-stack.tar.gz”上传到软件发布库中归档。

配置 SWR 服务

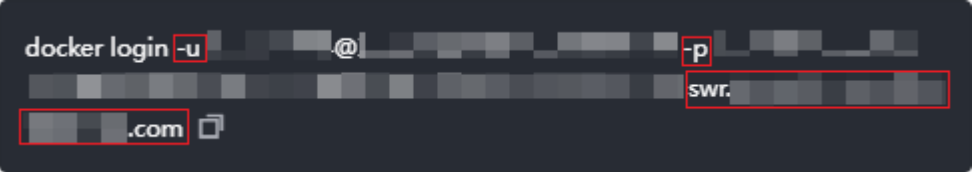
本文档使用SWR来保存环境镜像，在执行任务前需要获取SWR登录指令、创建组织。

- 步骤1** 在CodeArts中单击导航“控制台”。在控制台中单击左上角搜索并进入SWR服务。
- 请检查页面左上方的“区域”列表，请确保与编译构建任务所在区域相同。如果区域不同，请选择相同区域。
- 步骤2** 单击“登录指令”，页面弹框显示登录指令。

其中，

- “-u”之后的字符串为用户名。
- “-p”之后的字符串为密码。
- 最后的字符串为SWR服务器地址，此地址即为后续配置并执行任务中的参数“dockerServer”。

图 6-1 登录指令



说明

此处生成的登录指令为临时登录指令，有效期为24小时。如果需要长期有效的登录指令，请参见[获取长期有效登录指令](#)。

- 步骤3** 单击“创建组织”，在弹框中输入组织名称“phoenix”（如果页面提示“组织已存在”，请自定义其它名称），单击“确定”。
- 这里的组织名称，即为后续配置并执行任务中的参数“dockerOrg”。
- 结束

配置并执行任务


- 步骤1** 配置任务。
1. 进入“凤凰商城”项目，单击导航“持续交付 > 编译构建”。页面中显示样例项目内置的任务。
  2. 在列表中找到任务“phoenix-sample-ci”。单击图标，选择“编辑”。
  3. 选择“参数设置”页签，参照[表6-3](#)编辑参数值。

表 6-3 参数设置



参数名称	默认值
codeBranch	master。
dockerOrg	phoenix（即在SWR服务中创建的组织名称）。

参数名称	默认值
version	1.0.0
dockerServer	输入在SWR服务中获取的SWR服务器地址。

### 📖 说明

请务必确保参数“dockerOrg”、“dockerServer”的输入值是正确的，否则将导致任务失败。

**步骤2** 单击“保存并执行”，在弹框中单击“确定”，启动构建任务。

当页面中显示时，表示任务执行成功。请记录以“#”开头的字符串（例如 **#20230401.1**）。

如果构建失败，请根据失败步骤信息与日志中的报错信息排查。

**步骤3** 检查发布件。

- 单击导航“制品仓库 > 软件发布库”，进入软件发布库。
- 在与项目同名的仓库中，可以找到“docker-stack”、“phoenix-sample-ci”两个文件夹。
  - 在“docker-stack”文件夹中，可找到与**步骤2**中记录的字符串同名的文件夹，在此文件夹中可以找到发布件“docker-stack.tar.gz”。
  - 在文件夹“phoenix-sample-ci/1.0.0”中，可以找到归档的10个“.yaml”格式文件。
- 进入容器镜像服务，在导航中选择“组织管理”，单击与构建任务参数“dockerOrg”的值中同名的组织。  
选择“镜像”页签，可以在列表中找到5个镜像“redis”、“postgres”、“worker”、“result”、“vote”。
- 依次在列表中单击5个镜像的名称进入详情页。在“镜像版本”页签中查看镜像版本。
  - redis的镜像版本为“alpine”。
  - postgres的镜像版本为“9.4”。
  - worker、result、vote的镜像版本均与在**步骤2**中记录的字符串相同。


----结束

## 设置提交代码触发自动编译

通过以下配置，可实现代码变更后自动触发构建任务的执行，从而实现项目的持续集成。

**步骤1** 在任务“phoenix-sample-ci”的详情页，单击“编辑”。

**步骤2** 选择“执行计划”页签。

**步骤3** 打开“提交代码触发执行”开关, 保存任务。

由于在“参数设置”页签中“codeBranch”配置的默认值为“master”，因此本次设置的结果是当master有代码变更时自动触发构建。

**步骤4** 验证配置结果：修改项目代码并提交至master，即可查看构建任务是否自动执行。

----结束


## 设置定时执行任务

为了防止问题代码进入生产环境，确保应用总是处于可部署的状态，团队建议对应用进行持续不断的验证。

通过以下设置，可实现构建任务的定时执行。

**步骤1** 在任务“phoenix-sample-ci”的详情页，单击“编辑”。

**步骤2** 选择“执行计划”页签。

**步骤3** 打开“启用定时执行”开关 ，根据需要选择执行日与执行时间，保存任务。

本文档中勾选“全选”，执行时间为“12:00”（本文中默认时区，可以根据实际需要修改时区）。

**步骤4** 验证配置结果：根据配置时间查看构建任务是否自动执行，本节不再赘述。

----结束