编译构建

最佳实践

文档版本	01
发布日期	2023-11-15

华为技术有限公司

商标声明


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 图形化构建

1.1 使用 Maven 构建包制作 Docker 镜像

1.1.1 背景信息

编译构建服务提供了大量构建步骤、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。

本文旨在演示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、缓存的使用等作简要说明。

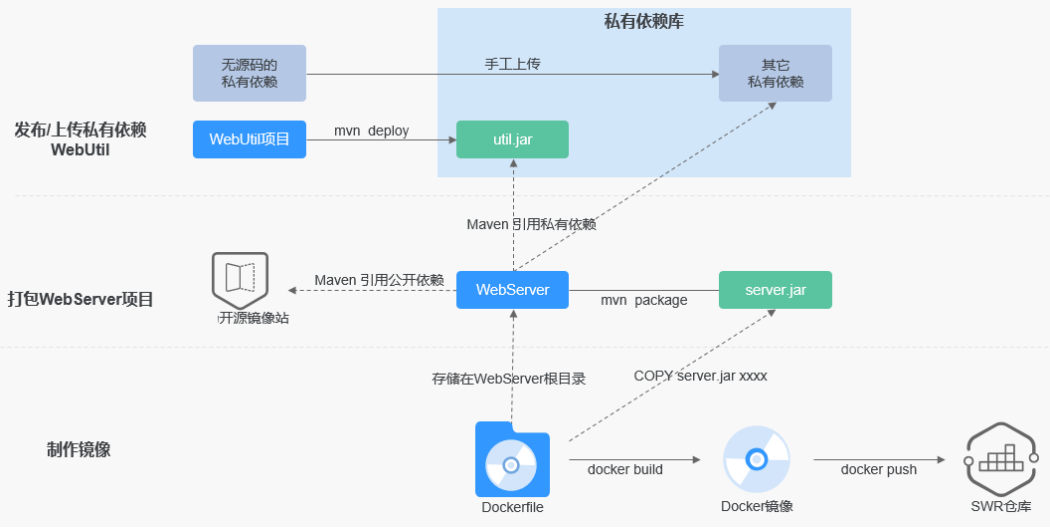
SWR，即**容器镜像服务**。SWR镜像仓库用于存储用户上传的Docker镜像，可以在构建、部署或其他场景使用。

1.1.2 项目说明

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

- WebServer项目：此次构建主项目，期望使用CodeArts Build构建此项目，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项目：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。大致分为以下步骤，可根据熟悉程度选择阅读：

- **构建准备**
- **发布私有依赖到私有依赖库**
- **打包并制作、推送镜像**
- **查看构建结果**

1.1.3 前提准备

如果是初次使用编译构建服务，请先**新建项目**，然后开始本示例。

准备 WebServer 项目代码仓库

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

步骤3 新建“src\main\java”目录。

步骤4 在**步骤3**创建的目录中，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 在编译构建首页，单击右上角  ，选择“自定义构建环境”。

步骤6 进入“自定义构建环境”页面，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。

本例中使用CentOS作为基础镜像。

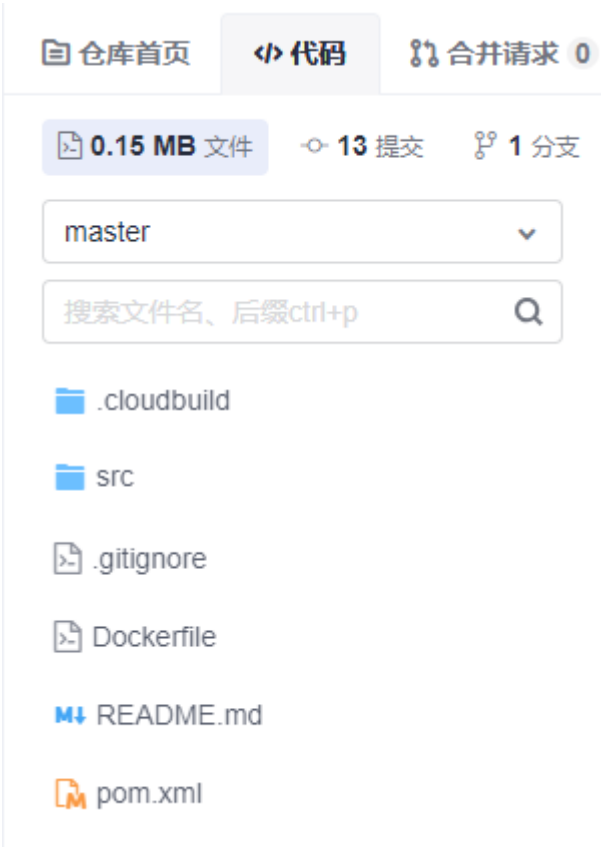
步骤7 查看**步骤2**中WebServer项目pom.xml中构建包的坐标定义。

Maven构建包名格式为：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6**中获取的构建包路径编写Dockerfile，内容如下：

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

步骤9 在导航栏选择“服务 > 代码托管”，参考**创建代码仓库**创建名为“WebServer”的代码仓库，然后将**步骤2**、**步骤3**和**步骤7**中创建的文件**上传代码至代码仓库**，代码上传完后，可进入代码仓库查看内容。



----结束

准备 WebUtil 项目代码仓库

步骤1 在本地新建一个用于存放代码的目录“WebUtil”。

步骤2 在步骤1创建的目录下新建名称为“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 本地新建“src\main\java”目录。

步骤4 在**步骤3**创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!");
 }
}
```

步骤5 在导航栏选择“服务 > 代码托管”，参考**创建代码仓库**创建名为“WebUtil”的代码仓库，然后将**步骤2**和**步骤4**创建的文件**上传代码至代码仓库**，代码上传完后，可进入代码仓库查看内容。

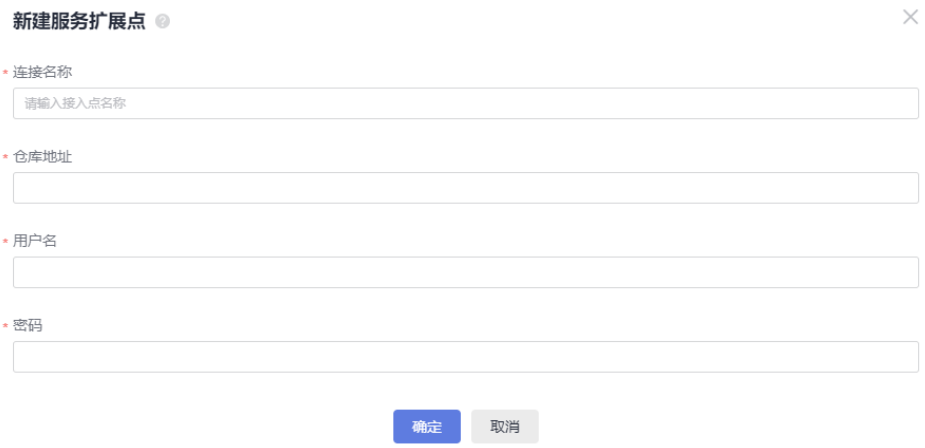
----结束

准备私有依赖扩展点

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。

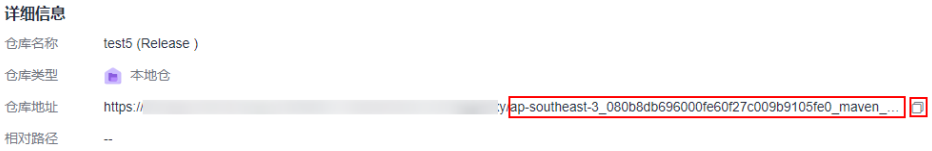


- 连接名称：使用私有依赖扩展点的显示名称，本例中可命名为“私有依赖扩展点”。
- 仓库地址：私有仓库地址。
- 用户名：从私有依赖库下载的指定私有仓库配置文件中的用户名。

- 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。

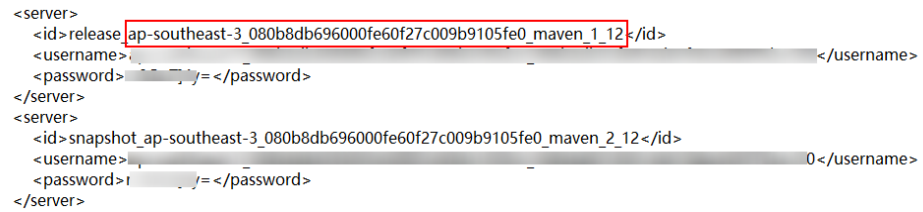
参数信息来源如下：

1. 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。



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”。

4. 获取文件中的用户名和密码。



需要使用ID为**仓库地址**最后一个参数的用户名和密码。

步骤4 请到**容器镜像服务SWR创建组织**，制作镜像并推送到SWR仓库时，需要指定SWR组织名。

----结束

1.1.4 发布私有依赖到私有依赖库

本节介绍如何将所需私有依赖发布到私有依赖库，构建前请务必仔细阅读以下注意事项，以避免因依赖使用不当导致构建异常。

说明

私有依赖库与软件发布库属于两个概念，使用时要务必注意区分，避免出现将依赖上传到软件发布库，构建时无法下载此类场景。

- 软件发布库主要用于归档用以部署或其他用途的软件包。
- 私有依赖库主要用于存储供其他项目使用的工具包等，如：WebUtil.jar。

此例中假设WebServer使用了三种私有依赖：

- WebUtil：项目组自研公共组件，本例使用构建任务发布到私有依赖库。
- CommonUtil：合作伙伴提供，有jar包，有“pom”文件（CommonUtil项目“pom”文件，不可使用WebServer项目的“pom”文件），此时可使用POM模式手动上传。
- 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。

前提准备

新用户首次使用软件开发生产线服务时，需要前往私有依赖库首页初始化私有依赖库，详情可参考文档**创建私有依赖库**。

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，源码源选择**前提准备**中创建代码仓库“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”。

Maven模板预置了“Maven构建”和“上传软件包到软件发布库”步骤以及配套的默认构建命令，多数场景下，直接使用即可完成构建并将生成的软件包上传到软件发布库。

步骤2 删除“上传软件包到软件发布库”构建步骤。

说明

本例主要介绍将项目依赖的工具包发布到私有依赖库，因此不需要“上传软件包到软件发布库”构建步骤，如需要归档软件包到发布库，也可以选择保留该步骤，构建包默认生成于“./target”目录下，一般会自动生成。

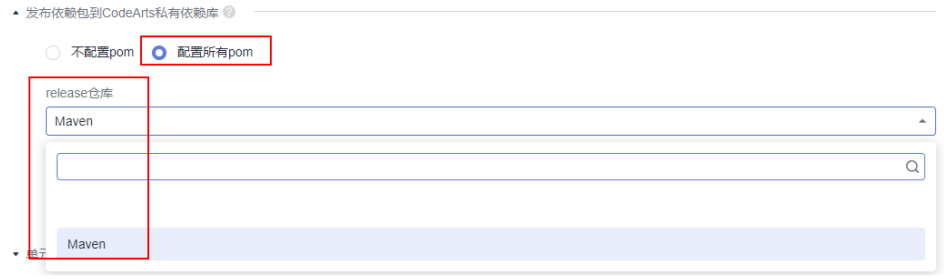
步骤3 配置“Maven构建”步骤。

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）。

  a. 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。
  b. 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

1.1.5 打包并制作、推送镜像

步骤1 **新建构建任务**，其中，源码源选择**前提准备**中创建的代码仓库“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”。

Maven模板预置了“Maven构建”和“上传软件包到软件发布库”步骤以及配套的默认构建命令，多数场景下，直接使用即可完成构建并将生成的软件包上传到软件发布库。

步骤2 删除“上传软件包到软件发布库”构建步骤。

说明

本例主要介绍将项目打包制作并推动镜像，因此不需要“上传软件包到软件发布库”构建步骤，如需要归档软件包到发布库，也可以选择保留该步骤，构建包默认生成于“./target”目录下，一般会自动生成。

步骤3 配置“Maven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**缓存清理**步骤清理缓存。

说明

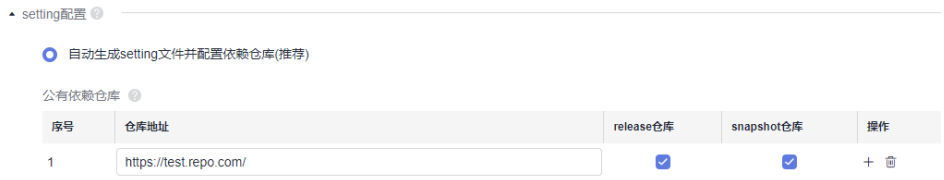
编译构建会自动配置华为开源镜像站作为开源依赖源，在使用编译构建时可自动下载，无需任何额外配置，开源镜像站已代理或同步的镜像源有：

- 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4 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5 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>
```

2. 保存“pom.xml”后重新上传到“WebServer”代码仓库中。
3. 在“Maven构建”步骤中选择**前提准备**中已创建的私有依赖库扩展点。

步骤6 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**前提准备**中创建的组织名。
- 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：**前提准备**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。

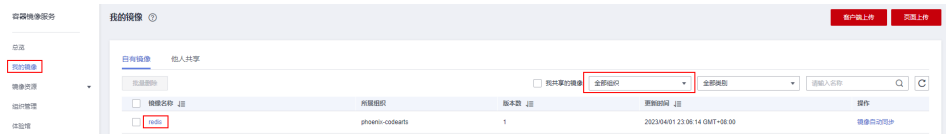
步骤7 保存并执行任务，执行成功后，即可**查看构建结果**。

----结束

1.1.6 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

1.1.7 疑问解答

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，确保理解且接受该风险，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。

Maven 构建默认命令含义是什么?

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

1.2 使用 Node.js 构建包制作 Docker 镜像

目标

本文旨在帮助用户了解如何使用软件开发生产线编译构建服务打包Node.js项目及制作构建包的Docker镜像。

前提准备

了解Docker的基本概念、环境安装和基本操作，关于如何在各个操作系统平台安装docker，请参考**Docker官方文档**。

如果是初次使用编译构建服务，请先**新建项目**，然后开始本示例。

步骤1 在本地创建一个用于存放代码的目录“nodesource”并进入该目录。

步骤2 在“nodesource”目录新建名称为“package.json”的文件，内容如下:

```
{
"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3 新建名为“server.js”的文件，内容如下:

```
'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4 本地验证。代码准备好后，可以在本地先编译，然后将项目运行起来，看结果是否正常。

1. 打开一个命令行，cd到nodesource所在目录，输入“npm install”完成依赖安装，再输入“node server.js”运行项目。


```
[root@localhost node_source]# ls -la
total 8
drwxr-xr-x. 2 root root  43 Jul 18 14:43 .
drwxr-xr-x. 5 root root  68 Jul 17 11:45 ..
-rw-r--r--. 1 root root 265 Jul 17 11:25 package.json
-rw-r--r--. 1 root root 277 Jul 17 11:27 server.js
[root@localhost node_source]# npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN docker_web_app@1.0.0 No repository field.
npm WARN docker_web_app@1.0.0 No license field.

added 50 packages from 37 contributors in 2.026s
[root@localhost node_source]# node server.js
Running on http://0.0.0.0:8080
```

2. 然后，打开浏览器，在浏览器中输入“*本机IP*:8080”，如果出现下图结果，则表示服务运行正常。

步骤5 在“nodesource”目录下，新建“Dockerfile”文件，内容如下：

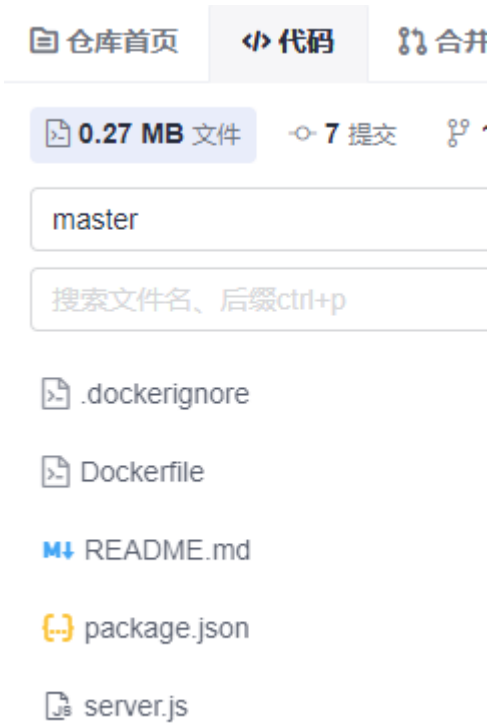
```
#use the latest LTS (long term support) version 12 of node available from the Docker Hub
FROM node:12
# Create app directory
WORKDIR /usr/src/app
# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@6+)
COPY package*.json ./
RUN npm install
# If you are building your code for production
# RUN npm ci --only=production
# Bundle app source
COPY . .
EXPOSE 8080
CMD ["node","server.js"]
```

步骤6 新建“.dockerignore”文件，内容如下：

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

步骤7 上传代码至代码托管服务。

在导航栏选择“服务 > 代码托管”，参考**创建代码仓库**创建名为“nodesource”的代码仓库，然后**上传代码至代码仓库**，代码上传完后，可进入代码仓库查看内容。



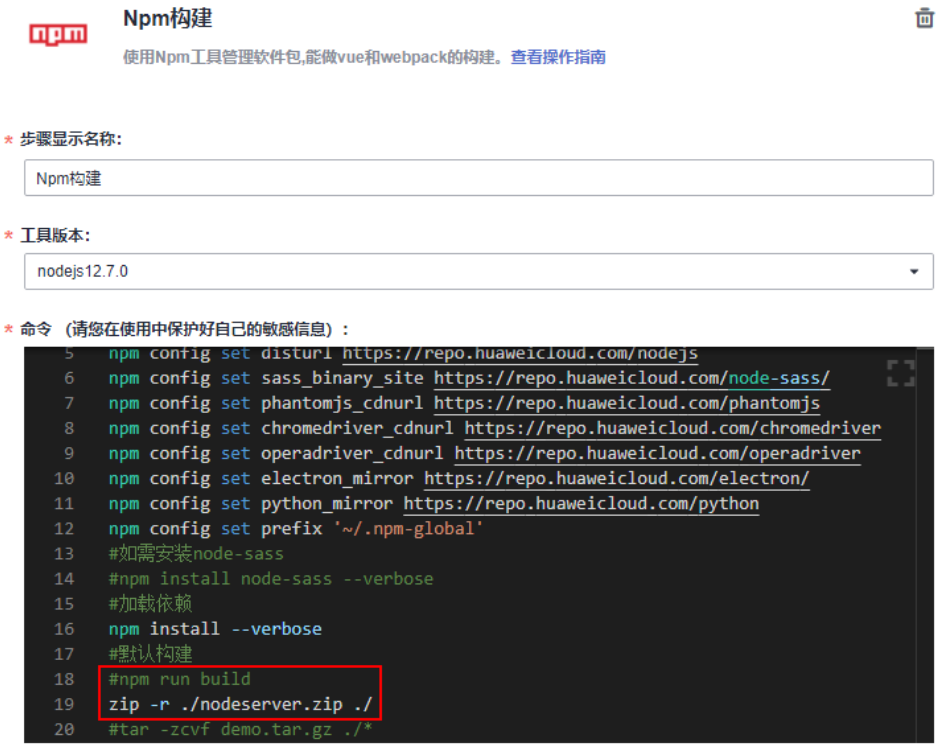
----结束

打包并制作、推送镜像

步骤1 **新建构建任务**，其中，源码源选择**前提准备**中创建好的代码仓库“nodesource”，构建模板选择“npm”，并将任务名称命名为“nodesource-build”。

步骤2 配置构建步骤。

1. 配置“Npm构建”，在命令编辑器里，注释掉“npm run build”，输入“zip -r ./nodeserver.zip ./”，用来将代码打包成“nodeserver.zip”。

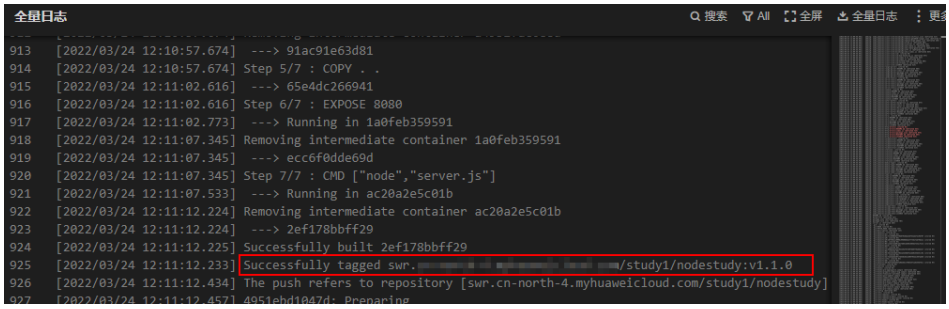


2. 配置“上传软件包到软件发布库”，按如下图填写参数。
   - 构建包路径：待上传的软件包路径。
   - 发布版本号：软件包的版本。
   - 包名：软件包名称。

3. 在“上传软件包到软件发布库”构建步骤后添加“制作镜像并推送到SWR仓库”构建步骤，并按如下图填写参数。

步骤3 配置完所有构建步骤，单击“新建并执行”，执行编译构建任务。

任务执行成功后，从日志里可以查看到新的镜像地址。



----结束

查看并验证构建结果

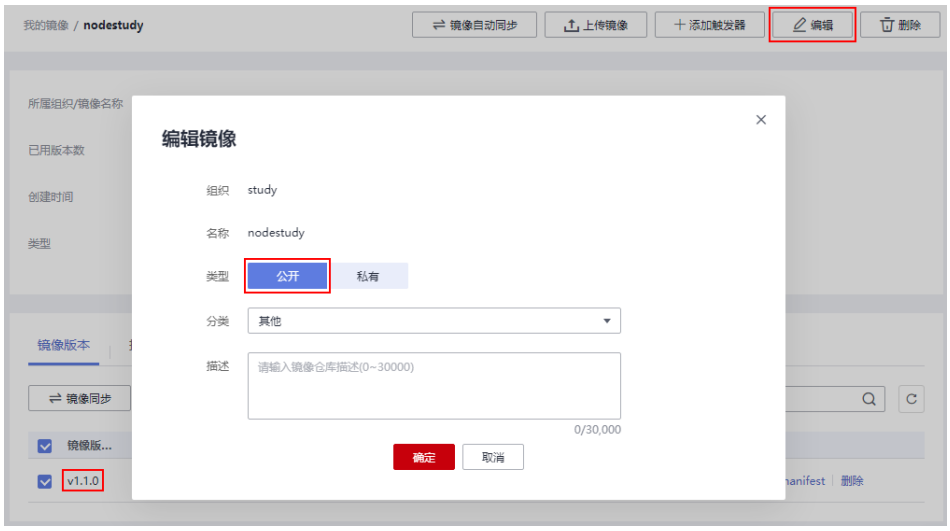
步骤1 在导航栏选择“制品仓库 > 软件发布库”，查看上传的软件包，名称与构建任务名称一致。

步骤2 获取镜像下载指令并将镜像设置为公开镜像。

1. 进入**容器镜像服务SWR**，单击左侧导航栏“我的镜像”，搜索并单击刚制作好的“nodestudy”镜像。

2. 进入镜像详情页，单击“下载指令”列的，获取完整的镜像下载指令（*docker pull + 镜像地址*）。

3. 选中镜像版本“v1.1.0”，单击右上角的“编辑”，在弹框中选择“公开”，然后单击“确定”。



步骤3 验证镜像。

1. 找一个安装了Docker的主机，在命令行中输入上一步获取的镜像下载指令。

```
[root@localhost nodejs]# docker pull swr.                    m/study/nodestudy:v1.1.0
v1.1.0: Pulling from study/nodestudy
a4d8138d0f6b: Already exists
dbdc36973392: Already exists
f59d6d019dd5: Already exists
aaef3e026258: Already exists
6e454d3b6c28: Already exists
c717a7c205aa: Already exists
e8566b4564fe: Already exists
04239395be03: Already exists
27ace7d95321: Already exists
ad27193056cd: Pull complete
98878eca7158: Pull complete
6341743603fc: Pull complete
423da2eb0660: Pull complete
Digest: sha256:355cb860206f1489587a1061967f5e1a371d76a227b67e11bef10526ce52bfc1
Status: Downloaded newer image for swr.c                     m/study/nodestudy:v1.1.0
```

2. 执行命令“docker run -p 28080:8080 -d *镜像地址*”启动镜像。
   其中，“-p 28080:8080”表示把镜像中的8080端口映射到本机的28080端口。

```
[root@localhost nodejs]# docker run -p 28080:8080 -d swr.c                    m/study/nodestudy:v1.1.0
d01dfc340d036841205fa1779160154168f8b457f699538e56ba4cd212adbaf8
[root@localhost nodejs]#
```

3. 在浏览器中输入“http://ip:port”，出现如下界面，说明镜像制作成功。
   其中ip为主机IP地址。

----结束

1.3 使用 Dockerfile 制作 Docker 镜像

背景信息

编译构建默认提供大量构建步骤、模板等，如果已有模板无法满足用户的使用需求，如缺少必要的依赖包、工具等，用户可以根据实际需要自定义Docker镜像。通过手动编写Dockerfile文件的方式，加入项目需要的依赖和工具。

- Dockerfile是由一系列命令和参数构成的脚本，从FROM命令开始，紧接着跟随各种方法、命令和参数。这些命令应用于基础镜像并最终创建一个新的镜像。它们简化了从头到尾的流程并极大的简化了部署工作。了解更多请参见Docker官网。
- 编译构建提供四种基础镜像：基于centos7包含各种常用工具的X86、ARM基础镜像和基于ubuntu18包含各种常用工具的X86、ARM基础镜像，可基于该基础镜像制作Dockerfile文件。

本示例以Maven构建为例，为您具体介绍如何使用Dockerfile定制一个简单的容器镜像并推送到SWR仓库。

前提准备

- **准备组织**

  在制作镜像并推送到SWR仓库时，需要指定SWR组织名，请提前在容器镜像服务**创建组织**。组织的约束与限制参考容器镜像服务的**约束与限制**。
- **准备项目代码**

  在代码托管服务基于Java Maven Demo模板创建代码仓库，请参见**按模板新建仓库**。
- **准备Dockerfile文件**

  基于编译构建基础镜像制作Dockerfile文件，本例使用centos7作为基础镜像。

  a. 在编译构建首页，单击右上角  ，选择“自定义构建环境”。

  b. 进入“自定义构建环境”页面，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。

  c. 获取构建包路径。

  Maven构建包名格式为：artifactId-version.packaging，构建包默认生成于“./target”目录。

  i. 导航栏选择“代码 > 代码托管”。

  ii. 单击代码仓库名称，进入代码托管“代码”页。

  iii. 查看代码仓库pom.xml文件坐标定义，如下图所示，则最终构建包路径为：./target/javaMavenDemo-1.0.jar。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>com       .demo</groupId>
  <artifactId>javaMavenDemo</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>maven_demo</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>
```

d. 使用**第3步**获取的构建包路径编写**第2步**下载的Dockerfile文件，在基础镜像中添加Maven构建包，添加如下命令：

```
COPY ./target/javaMavenDemo-1.0.jar /demo/app.jar
```

该命令表示将构建包复制到镜像的“demo”目录，并将构建包命名为app.jar。

e. 编写完成后，将Dockerfile文件及制作镜像过程中需要的其他文件上传到代码仓库根目录，具体上传操作请参考**上传代码至代码仓库**。

操作步骤

步骤1 选择“持续交付 > 编译构建”。

步骤2 单击“新建任务”，进入配置“基本信息”页面，填写构建任务基本信息。

表 1-1 基本信息

参数项	描述
任务名称	任务的名称。
归属项目	任务所属项目。
源码源	Repo：从代码托管拉取代码进行构建，请选择**前提准备**中创建的源码仓库及分支。
任务描述	对任务进行描述。

步骤3 单击“下一步”，进入“构建模板”页面。

步骤4 选择“Maven构建”模板，单击“下一步”。

步骤5 添加“制作镜像并推送到SWR仓库”构建步骤。

“Maven构建”构建步骤参数保持默认即可，“制作镜像并推送到SWR仓库”构建步骤参数配置说明如下：

参数项	说明
步骤显示名称	构建步骤的名称，可自定义修改。
工具版本	选择工具版本，使用默认版本即可。
镜像仓库	编译构建服务已经默认给出了各区域对应的SWR仓库地址，用户无需更改。 **说明** 支持推送到用户自定义镜像仓库。
授权用户	当前用户。请确保当前用户对组织内所有镜像享有编辑或管理权限，详见**授权管理**。
组织	SWR仓库组织名，请填写**准备工作**中创建好的组织名。
镜像名字	制作完成后的镜像名称，可自定义。
镜像标签	用来标记镜像的版本，可自定义。通过“镜像名:标签”可以唯一指定镜像。

参数项	说明
工作目录	docker build命令中的“上下文路径”参数，该路径是CodeArts Repo代码仓库根目录的相对路径。 上下文路径，指的是docker在构建镜像时，docker build命令将该路径下的所有内容打包给容器引擎帮助构建镜像。
Dockerfile路径	Dockerfile文件所在路径，请填写相对于工作目录的路径，如：工作目录为根目录， 且Dockerfile文件在根目录下，则此处填写为“./Dockerfile”。
添加构建元数据到镜像	将本次构建信息添加到镜像中，镜像制作完成后可以通过docker inspect命令查看镜像元数据。

步骤6 配置完构建步骤，单击“新建”，开始执行构建任务。

步骤7 执行成功后，进入**容器镜像服务**。

步骤8 单击导航栏“我的镜像”，选择“制作镜像并推送到SWR仓库”构建步骤中填写的组织，即可查看刚构建并上传的镜像。

----结束

2 代码化构建

2.1 使用 CMake 构建上传软件包

背景说明

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，安全可信。本节以使用Cmake构建为例。

前提条件

已有可用项目，如果没有，请**新建项目**。

新建代码仓

步骤1 使用华为云账号登录软件开发生产线首页。

步骤2 单击需要创建代码仓的项目名称。

步骤3 在页面左侧目录中选择“代码 > 代码托管”。

步骤4 进入代码托管页面，单击“新建仓库”，选择“普通仓库”。

步骤5 根据表2-1填写参数，单击“确定”。

表 2-1 新建代码仓

参数名称	参数说明
代码仓库名称	自定义代码仓名称。例如：CppDemo_build。 • 以数字、字母或者“_”开头。 • 可包含“.”和“-”。 • 不能以“.git”、“.atom”或者“.”结尾。
描述	对代码仓的描述。
选择gitignore	根据编程语言选择“.gitignore”，例如：Java。
权限设置	勾选全部。 • 允许项目内人员访问仓库：选择后会自动将项目中的项目经理设为仓库管理员，开发人员设为仓库普通成员。当项目新增这两个角色时，也会自动同步到已经存在的仓库中。 • 允许生成README文件：可以通过编辑README文件，记录项目的架构、编写目的等信息，相当于对整个仓库的一种注释。 • 自动创建代码检查任务（免费）：仓库创建完成后在代码检查任务列表中，可看到对应仓库的检查任务。
是否公开	设置为“私有”。 • 私有：仓库仅对仓库成员可见，仓库成员可访问仓库或者提交代码。 • 公开只读：仓库对所有访客公开只读，但不出现在访客的仓库列表及搜索中，您可以选择开源许可证作为备注。

----结束

创建 build.yml 文件

步骤1 在页面导航中选择“代码 > 代码托管”。

步骤2 单击新建代码仓中创建的代码仓名称。

步骤3 单击“新建 > 新建目录”，如图2-1所示。

图 2-1 新建目录

步骤4 根据表2-2填写参数信息，单击“确定”。

表 2-2 新建目录

参数名称	参数说明
目录名称	可自定义，例如“.cloudbuild”。
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。

步骤5 单击步骤4创建的目录名称。

步骤6 单击“新建 > 新建文件”，如图2-2所示。

图 2-2 新建文件

步骤7 文件命名为“build.yml”，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:
  PRE_BUILD:
  - checkout:
      name: "checkout"
      inputs:
        scm: "codehub"
        url: "git@codehub.devcloud.cn-north-4.huaweicloud.com:cszl00001/cppDemo.git"
        branch: "master"
        lfs: false
        submodule: false
  BUILD:
  - cmake:
      name: "Cmake构建"
      inputs:
        command: |
          #新建build目录 切换到build目录、
          mkdir build && cd build
          # 生成Unix 平台的makefiles文件并执行构建
          cmake -G 'Unix Makefiles' ../ && make -j
  - upload_artifact:
      inputs:
        path: "build/*"
        version: 2.1
        name: packageName
```

步骤8 单击“确定”。

----结束

创建 CMakeLists.txt 文件

步骤1 在根目录下，参考步骤6和步骤7，创建名为“CMakeLists.txt”的文件。文件中代码如下：

```
cmake_minimum_required (VERSION 2.5)
```



```
project (HolleWorld)
AUX_SOURCE_DIRECTORY(. DIR_SRCS)

add_executable(bin ${DIR_SRCS})
```

步骤2 单击“确定”。

----结束

创建 helloworld.cpp 文件

步骤1 在根目录下，参考**步骤6**和**步骤7**，创建名为“helloworld.cpp”的文件。文件中代码如下：

```
#include <iostream>
int main()
{
    std::cout << "Hello World !"  << std::endl;
    return 0;
}
```

步骤2 单击“确定”。

----结束

创建构建任务

步骤1 在页面导航中选择“持续交付 > 编译构建”，如**图2-3**所示。

图 2-3 编译构建首页

步骤2 单击“新建任务”。

步骤3 根据**表2-3**填写参数信息。

表 2-3 基本信息配置

参数名称	参数说明
任务名称	自定义任务名称，例如：CppDemo_build。
源码源	选择“Repo”。
源码仓库	选择**新建代码仓**中创建的代码仓库名称。
分支	选择**新建代码仓**中创建的分支，若没有创建，选择默认“master”即可。
任务描述	对该构建任务的描述。

步骤4 单击“下一步”。

步骤5 选择“空白构建模板”，单击“下一步”。

步骤6 单击“代码化”页签，可查看到导入的构建脚本，如**图2-4**所示。

图 2-4 代码化页签

步骤7 单击页面右上角的“新建并执行”。

----结束

查看并验证构建结果

步骤1 选择页面导航栏“制品仓库 > 软件发布库”。

步骤2 在软件发布库查看发布的软件包。软件包名称与**创建构建任务**时的任务名称一致。

----结束

2.2 使用 Maven 构建上传软件包

场景概述

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，安全可信。本节以使用Maven构建为例。

前提条件

已有可用项目，如果没有，请**新建项目**。

新建代码仓

步骤1 使用华为云账号登录软件开发生产线首页。

步骤2 单击需要创建代码仓的项目名称。

步骤3 在页面左侧目录中选择“代码 > 代码托管”，如**图2-5**所示。

图 2-5 代码托管

步骤4 单击“普通新建”。

步骤5 根据**表2-4**填写参数，单击“确定”。

表 2-4 新建代码仓

参数名称	参数说明
代码仓库名称	自定义代码仓名称。例如：maven_yml_build。 • 以数字、字母或者“_”开头。 • 可包含“.”和“-”。 • 不能以“.git”、“.atom”或者“.”结尾。
描述	对代码仓的描述。
选择gitignore	根据编程语言选择“.gitignore”，例如：Java。

参数名称	参数说明
权限设置	勾选全部。 • 允许项目内人员访问仓库：选择后会自动将项目中的项目经理设为仓库管理员，开发人员设为仓库普通成员。当项目新增这两个角色时，也会自动同步到已经存在的仓库中。 • 允许生成README文件：可以通过编辑README文件，记录项目的架构、编写目的等信息，相当于对整个仓库的一种注释。 • 自动创建代码检查任务（免费）：仓库创建完成后在代码检查任务列表中，可看到对应仓库的检查任务。
是否公开	设置为“私有”。 • 私有：仓库仅对仓库成员可见，仓库成员可访问仓库或者提交代码。 • 公开只读：仓库对所有访客公开只读，但不出现在访客的仓库列表及搜索中，您可以选择开源许可证作为备注。

----结束

创建 build.yml 文件

步骤1 在页面导航中选择“代码 > 代码托管”。

步骤2 单击**新建代码仓**中创建的代码仓名称。

步骤3 单击“新建 > 新建目录”，如**图2-6**所示。

图 2-6 新建目录

步骤4 根据**表2-5**填写参数信息，单击“确定”。

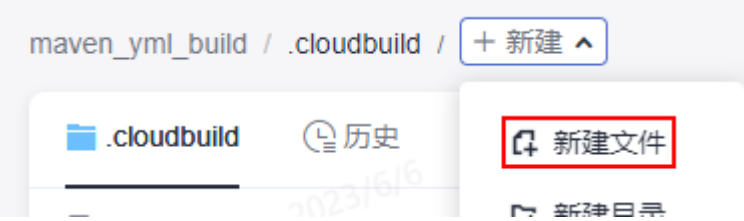
表 2-5 新建目录

参数名称	参数说明
目录名称	可自定义，例如“.cloudbuild”。
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。

步骤5 单击**步骤4**创建的目录名称。

步骤6 单击“新建 > 新建文件”，如**图2-7**所示。

图 2-7 新建文件



步骤7 文件命名为“build.yml”，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8 单击“确定”。

----结束

创建 Java 文件

步骤1 参考**步骤4**，创建名为“src/main/java”的目录。

步骤2 在“src/main/java”的目录下，参考**步骤6**和**步骤7**，创建名为“HelloWorld.java”的文件。文件中代码如下：

```
/**
 * Hello world
 *
 */

public class HelloWorld {

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

}
```

步骤3 单击“确定”。

----结束

创建 Dockerfile 文件

步骤1 在根目录下，参考**步骤6**和**步骤7**，创建名为“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-in
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

创建 pom.xml 文件

步骤1 在根目录下，参考**步骤6**和**步骤7**，创建名为“pom.xml”的文件。文件中代码如下：

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

步骤2 单击“确定”。

----结束

创建构建任务

步骤1 在页面导航中选择“持续交付 > 编译构建”，如图2-8所示。

图 2-8 编译构建首页

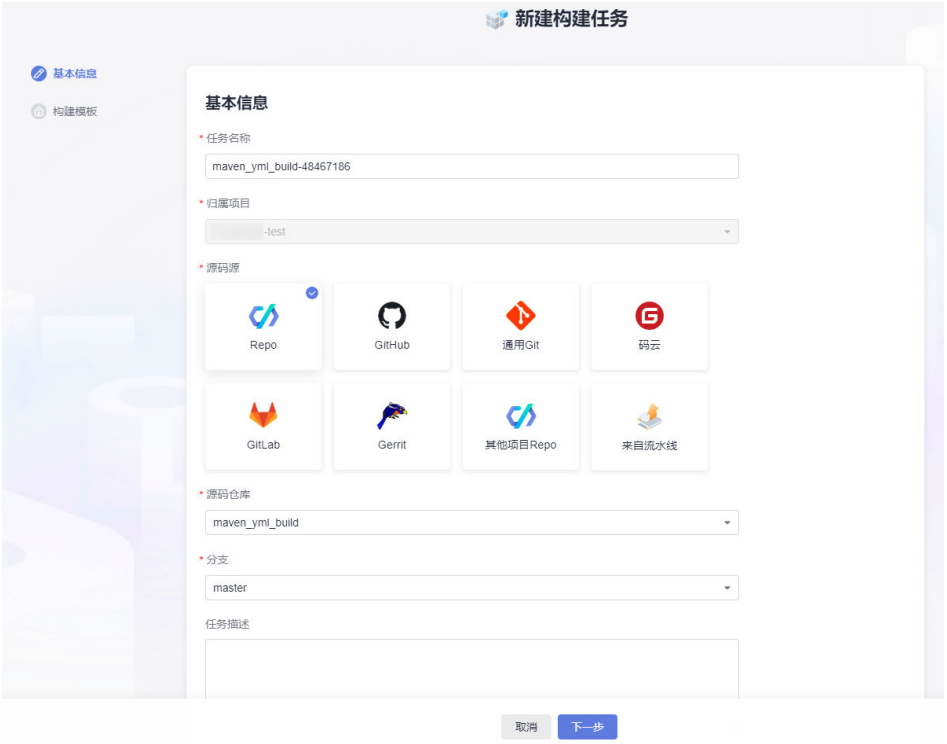
步骤2 单击“新建任务”。

步骤3 根据表2-6填写参数信息，如图2-9所示。

表 2-6 基本信息配置

参数名称	参数说明
任务名称	自定义任务名称，例如：maven_yml_build。
源码源	选择“Repo”。
源码仓库	选择**新建代码仓**中创建的代码仓库名称。
分支	选择**新建代码仓**中创建的分支，若没有创建，选择默认“master”即可。
任务描述	对该构建任务的描述。

图 2-9 创建构建任务



步骤4 单击“下一步”。

步骤5 选择“空白构建模板”，单击“下一步”。

步骤6 单击“代码化”页签，可查看到导入的构建脚本，如图2-10所示。

图 2-10 代码化页签

步骤7 单击页面右上角的“新建并执行”。

----结束

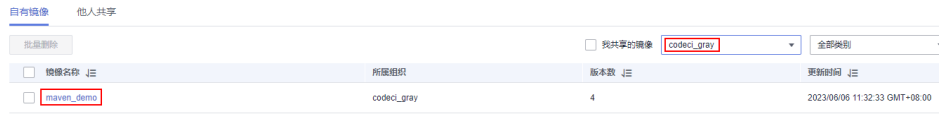
查看并验证构建结果

- 查看上传的软件包。
  a. 选择页面导航栏“制品仓库 > 软件发布库”。
  b. 在软件发布库查看发布的软件包。软件包与**创建构建任务**时的任务名称一致，如图2-11所示。

图 2-11 查看软件包

- 查看推送的镜像。
  a. 进入**容器镜像服务SWR**。
  b. 单击导航栏“我的镜像”，在组织中筛选**创建build.yml文件**时代码中填写的“组织名称”，如：codeci_gray。
  c. 在筛选结果中单击**创建build.yml文件**时代码中填写的“镜像名称”，如：maven_demo，如**图2-12**所示。

图 2-12 筛选镜像



2.3 使用 NPM 构建上传软件包

场景概述

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，安全可信。本节以使用Npm构建为例。

前提条件

已有可用项目，如果没有，请**新建项目**。

新建代码仓

步骤1 使用华为云账号登录软件开发生产线首页。

步骤2 单击需要创建代码仓的项目名称。

步骤3 在页面左侧目录中选择“代码 > 代码托管”。

步骤4 进入代码托管页面，单击“新建仓库”，选择“普通仓库”。

步骤5 根据表2-7填写参数，单击“确定”。

表 2-7 新建代码仓

参数名称	参数说明
代码仓库名称	自定义代码仓名称。例如：npm_yml_build。 • 以数字、字母或者“_”开头。 • 可包含“.”和“-”。 • 不能以“.git”、“.atom”或者“.”结尾。
描述	对代码仓的描述。
选择gitignore	根据编程语言选择“.gitignore”，例如：Java。
权限设置	勾选全部。 • 允许项目内人员访问仓库：选择后会自动将项目中的项目经理设为仓库管理员，开发人员设为仓库普通成员。当项目新增这两个角色时，也会自动同步到已经存在的仓库中。 • 允许生成README文件：可以通过编辑README文件，记录项目的架构、编写目的等信息，相当于对整个仓库的一种注释。 • 自动创建代码检查任务（免费）：仓库创建完成后在代码检查任务列表中，可看到对应仓库的检查任务。
是否公开	设置为“私有”。 • 私有：仓库仅对仓库成员可见，仓库成员可访问仓库或者提交代码。 • 公开只读：仓库对所有访客公开只读，但不出现在访客的仓库列表及搜索中，您可以选择开源许可证作为备注。

----结束

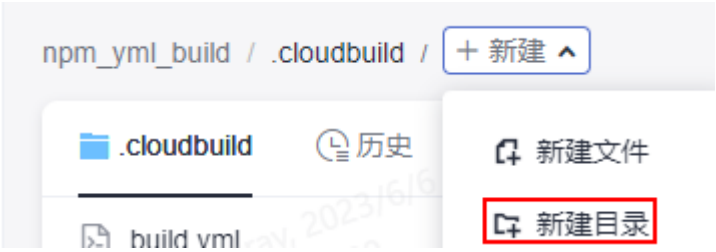
创建 build.yml 文件

步骤1 在页面导航中选择“代码 > 代码托管”。

步骤2 单击**新建代码仓**中创建的代码仓名称。

步骤3 单击“新建 > 新建目录”，如**图2-13**所示。

图 2-13 新建目录



步骤4 根据**表2-8**填写参数信息，单击“确定”。

表 2-8 新建目录

参数名称	参数说明
目录名称	可自定义，例如“.cloudbuild”。
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。

步骤5 单击**步骤4**创建的目录名称。

步骤6 单击“新建 > 新建文件”，如**图2-14**所示。

图 2-14 新建文件

步骤7 文件命名为“build.yml”，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:
    - npm:
        inputs:
          #check:
            #project_dir: .
          command: |
            export PATH=$PATH:~/.npm-global/bin
            #设置缓存目录
            npm config set cache /npmcache
            npm config set registry http://mirrors.tools.huawei.com/npm/
            npm config set disturl http://mirrors.tools.huawei.com/nodejs
            npm config set sass_binary_site http://mirrors.tools.huawei.com/node-sass/
            npm config set phantomjs_cdnurl http://mirrors.tools.huawei.com/phantomjs
```

```
        npm config set chromedriver_cdnurl http://mirrors.tools.huawei.com/chromedriver
        npm config set operadriver_cdnurl http://mirrors.tools.huawei.com/operadriver
        npm config set electron_mirror http://mirrors.tools.huawei.com/electron/
        npm config set python_mirror http://mirrors.tools.huawei.com/python
        npm config set prefix '~/.npm-global'
        npm install --verbose
        zip -r ./nodeserver.zip ./
  - upload_artifact:
      inputs:
        path: "./nodeserver.zip"
```

步骤8 单击“确定”。

----结束

创建 package.json 文件

步骤1 在根目录下，参考**步骤6**和**步骤7**，创建名为“package.json”的文件。文件中代码如下：

```
{
"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2 单击“确定”。

----结束

创建 server.js 文件

步骤1 在根目录下，参考**步骤6**和**步骤7**，创建名为“server.js”的文件。文件中代码如下：

```
'use strict';
const express =require('express');
// Constants
const PORT=8080;
const HOST='127.0.0.1';
// 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2 单击“确定”。

----结束

创建构建任务

步骤1 在页面导航中选择“持续交付 > 编译构建”，如**图2-15**所示。

图 2-15 编译构建首页

步骤2 单击“新建任务”。

步骤3 根据表2-9填写参数信息。

表 2-9 基本信息配置

参数名称	参数说明
任务名称	自定义任务名称，例如：npm_yml_build。
源码源	选择“Repo”。
源码仓库	选择新建代码仓中创建的代码仓库名称。
分支	选择新建代码仓中创建的分支，若没有创建，选择默认“master”即可。
任务描述	对该构建任务的描述。

步骤4 单击“下一步”。

步骤5 选择“空白构建模板”，单击“下一步”。

步骤6 单击“代码化”页签，可查看到导入的构建脚本，如图2-16所示。

图 2-16 代码化页签

步骤7 单击页面右上角的“新建并执行”。

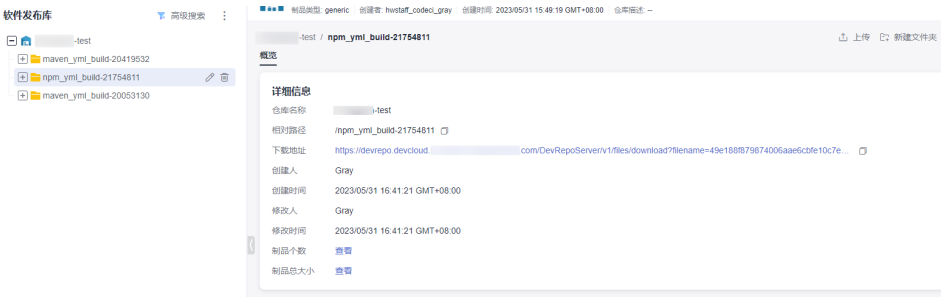
----结束

查看并验证构建结果

步骤1 选择页面导航栏“制品仓库 > 软件发布库”。

步骤2 在软件发布库查看发布的软件包。软件包与**创建构建任务**时的任务名称一致，如**图 2-17**所示。

图 2-17 查看软件包



----结束