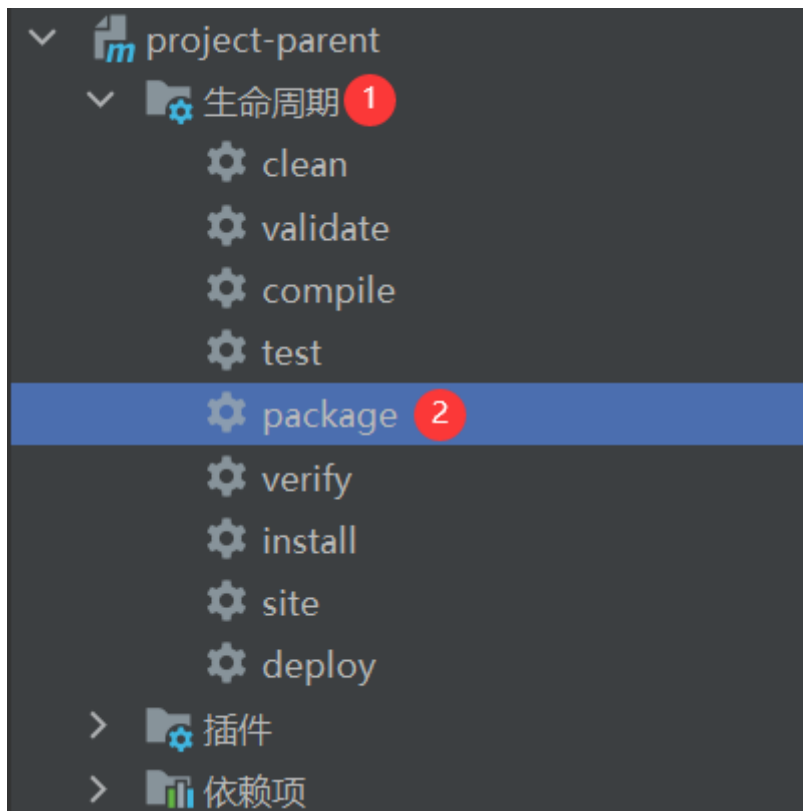


项目打包部署

1 测试打包

1.1 打包

可以通过parent模块对所有模块进行打包，使用生命周期package指令。



如果你想安装到maven仓库中可以使用install指令。

打包前注意检查子模块中，如果是spring-boot项目，是否正确配置了spring-boot-maven插件，配置示例如下：

```
1 <build>
2   <plugins>
3     <plugin>
4       <groupId>org.springframework.boot</groupId>
5       <artifactId>spring-boot-maven-plugin</artifactId>
6       <configuration>
7         <!-- 指定spring-boot程序入口类 -->
8         <mainClass>
9           com.zeroone.star.doc.DocApplication
10        </mainClass>
11      </configuration>
12    </plugin>
13  </plugins>
14 </build>
```

1.2 构建镜像

在模块根目录下新建Dockerfile文件，内容示例如下：

文档中的内容根据实际情况修改，主要修改第6行中的target目录下面的jar包名称。

```
1  # 该镜像需要依赖的基础镜像
2  FROM openjdk:8
3  # 指定维护者的名字
4  MAINTAINER 01star
5  # 将当前target目录下的jar包复制到docker容器中
6  ADD target/project-doc-1.0.0-SNAPSHOT.jar app.jar
7  # 定义JVM参数
8  ENV JAVA_OPTS="-Xms256m -Xmx256m"
9  ENV SPRING_ARGS=""
10 RUN echo "JAVA_OPTS=" $JAVA_OPTS
11 # 设置容器启动执行指令
12 CMD java $JAVA_OPTS -jar app.jar --logging.file.path=/tmp/logs/spring-boot
    $SPRING_ARGS
```

在当前模块的pom文件中，新增docker-maven插件，示例如下：

具体配置根据时间情况做微调整，主要修改内容：

- dockerHost：远程管理地址
- certPath：CA证书位置
- ports：端口映射
- JAVA_OPTS：JVM参数
- SPRING_ARGS：启动替换参数

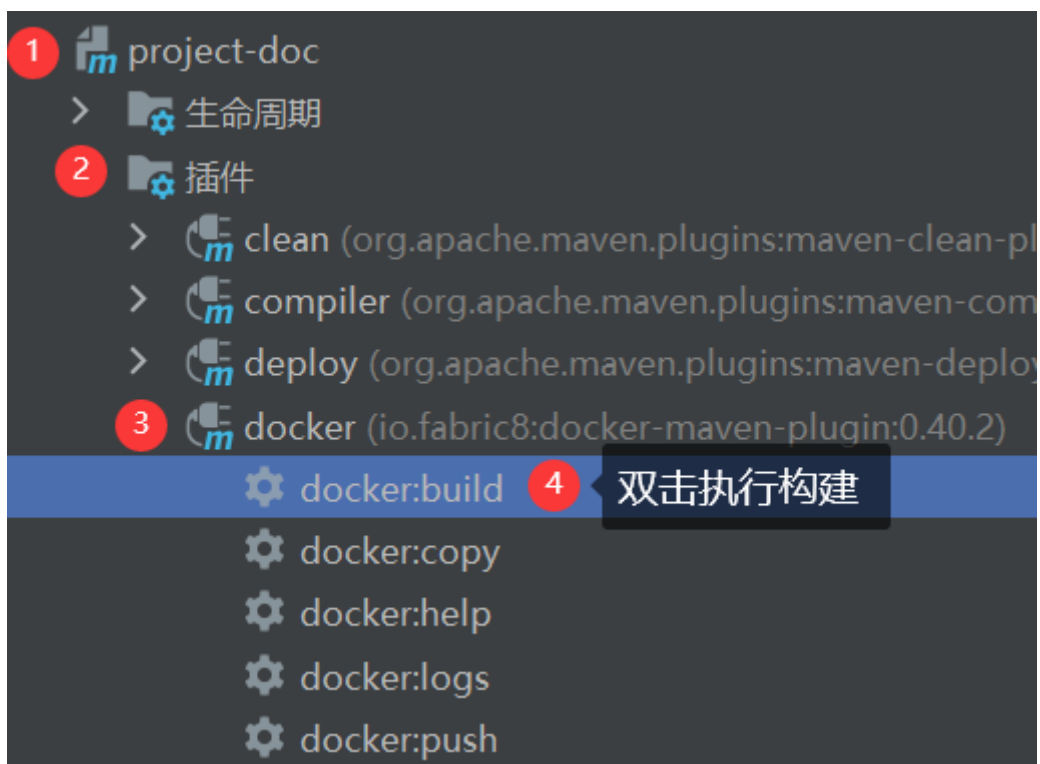
```
1  <plugin>
2    <groupId>io.fabric8</groupId>
3    <artifactId>docker-maven-plugin</artifactId>
4    <configuration>
5      <!-- Docker 远程管理地址 -->
6      <dockerHost>https://192.168.220.128:2375</dockerHost>
7      <!-- CA 证书位置 -->
8      <certPath>/home/docker-ca</certPath>
9      <images>
10        <image>
11          <!-- Docker 镜像名称定义 -->
12          <name>01star/${project.artifactId}:${project.version}</name>
13          <!-- 指定Dockerfile所在目录 -->
14          <build>
15            <contextDir>${project.basedir}</contextDir>
16          </build>
17          <!-- 别名用于容器命名 -->
18          <alias>${project.artifactId}</alias>
19          <!-- 容器run相关配置 -->
20          <run>
21            <!-- 配置运行时容器命名策略为:别名,如果不指定则默认为none,即使用
                随机分配名称 -->
22            <namingStrategy>alias</namingStrategy>
23            <!-- 端口映射 -->
24            <ports>
```

```

25         <port>9999:9999</port>
26     </ports>
27     <!-- 数据卷 -->
28     <volumes>
29         <bind>
30             <volume>/etc/localtime:/etc/localtime</volume>
31
32     <volume>/home/app/${project.artifactId}/logs:/tmp/logs</volume>
33         </bind>
34     </volumes>
35     <!-- 设置环境变量 -->
36     <env>
37         <!-- JVM参数 -->
38         <JAVA_OPTS>-Xms256m -Xmx256m</JAVA_OPTS>
39         <!-- 启动替换参数 -->
40         <SPRING_ARGS>
41             --spring.profiles.active=test --
42             spring.cloud.nacos.discovery.ip=192.168.220.128
43         </SPRING_ARGS>
44     </env>
45 </run>
46 </image>
47 </images>
48 </configuration>
49 </plugin>

```

执行镜像构建



成功构建结果如下图所示。

```
m project-doc [io.fabric8:docker-maven-plugin:0.40.2:build] x
✓ project-doc 11秒148毫秒 [INFO] --- docker-maven-plugin:0.40.2:build (default-cli) @ project-doc ---
[INFO] Building tar: F:\project-training\advanced-project\java-micro-services\pr
[INFO] DOCKER> [01star/project-doc:1.0.0-SNAPSHOT]: Created docker-build.tar in
[INFO] DOCKER> [01star/project-doc:1.0.0-SNAPSHOT]: Built image sha256:4699f
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

查看远程服务器镜像

```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID
01star/project-doc  1.0.0-SNAPSHOT     4699fabadada
sentinel-dashboard  1.8.4              93a4075da551
jenkinsci/blueocean latest             6aa7e6ae5876
redis               6.2.7              ae14d724f92e
seataio/seata-server 1.5.1              7ecfc0df35bf
nacos/nacos-server  v2.1.0             b0a4aba28604
apache/rocketmq      4.9.3              9a54e5b5b6c1
confluentinc/cp-kafka 7.0.1              5069d65bcc55
confluentinc/cp-zookeeper 7.0.1             3a7ea656f1af
apacherocketmq/rocketmq-dashboard latest            eae6c5db5d11
mysql               8.0.20             be0dbf01a0f3
java                8                  d23bdf5b1b1b
```

到此镜像构建成功，你可以使用docker命令创建容器了，当然也可以写个shell脚本来启动服务，示例如下。

```
1  #!/bin/bash
2  app_name='project-doc'
3  docker stop ${app_name}
4  echo '——stop container——'
5  docker rm ${app_name}
6  echo '——rm container——'
7  docker run -p 9999:9999 --name ${app_name} \
8  -v /etc/localtime:/etc/localtime \
9  -v /home/app/${app_name}/logs:/tmp/logs \
10 -e JAVA_OPTS="-Xms256m -Xmx256m" \
11 -e SPRING_ARGS="--spring.profiles.active=test --
   spring.cloud.nacos.discovery.ip=192.168.220.128" \
12 -d 01star/${app_name}:1.0.0-SNAPSHOT
13 echo '——start container——'
```

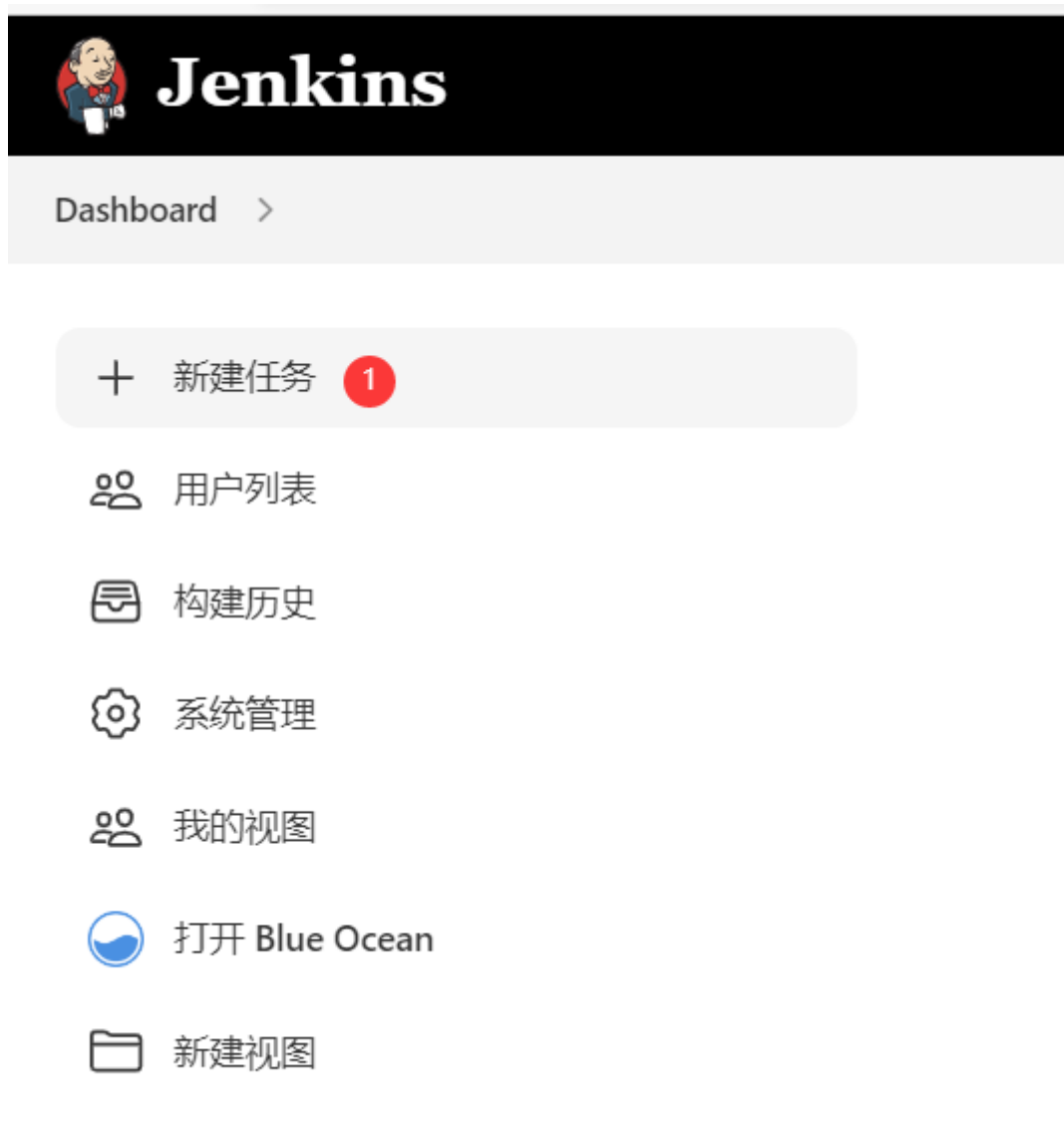
当然也可以自己写服务编排脚本。

2 集成到Jenkins中

Jenkins是开源CI&CD软件领导者，提供超过1000个插件来支持构建、部署、自动化，满足任何项目的需要。我们可以用Jenkins来构建和部署我们的项目，比如说从我们的代码仓库获取代码，然后将我们的代码打包成可执行的文件，之后通过远程的ssh工具执行脚本来运行我们的项目。

2.1 创建任务

选择创建任务




输入任务信息

输入一个任务名称

java-end


1 输入名称

» 必填项


**构建一个自由风格的软件项目**

2 选择自由构建


这是Jenkins的主要功能,Jenkins将会结合任何SCM和任何构建系统来构建你的项目, 甚至可以构建软件以外的系统。

**流水线**


精心地组织一个可以长期运行在多个节点上的任务。适用于构建流水线（更加正式地应当称为工作流），增加或者组织难以采用自由风格的任务类型。

**构建一个多配置项目**


适用于多配置项目,例如多环境测试,平台指定构建,等等。

**(0) 文件夹**

Creates a set of multibranch project subfolders by scanning for repositories.

**多分支流水线**

根据一个SCM仓库中检测到的分支创建一系列流水线。

**文件夹**

创建一个可以嵌套存储的容器。利用它可以进行分组。视图仅仅是一个过滤器，而文件夹则是一个独立的命名空间，因此你可以有多个相同名称的内容，只要它们在不同的文件夹里即可。

确定 3

确定创建

2.2 配置源码管理

任务描述自己按照情况书写即可，在源码管理中添加你的github仓库地址。

提示：如果GitHub 下载太缓慢，可以考虑将GitHub 仓库导入到Gitee上面使用Gitee 仓库来部署

Configuration

General

源码管理

构建触发器

构建环境

Build Steps

构建后操作

源码管理

无

Git

Repositories

Repository URL

git@github.com:

输入ssh地址

Credentials

anyuser (github ssh private key)

选择ssh key

这个Key应该是在安装的Jenkins已经配置了
如果没有配置那么点击下面的添加按钮去配置

+ 添加

高级...

Add Repository

Branches to build

指定分支 (为空时代表any)

*/master

在这里可以切换编译分支

Add Branch

源码库浏览器

(自动)

Additional Behaviours

新增

保存

应用

点击应用保存项目

2.3 配置maven构建

所有构建步骤都使用调用顶层maven目标，添加方式如下图所示：

Build Steps

增加构建步骤

1

Filter

Execute NodeJS script

Invoke Ant

Invoke Gradle script

Provide Configuration files

Run with timeout

Send files or execute commands over SSH

Set build status to "pending" on GitHub commit

执行 Windows 批处理命令

执行 shell

调用顶层 Maven 目标

2

2.3.1 将源码打包成jar包

选择maven版本，然后设置maven命令和指定父pom文件位置。

Build Steps

调用顶层 Maven 目标 ?

Maven 版本

maven3.6.3

1

选择maven版本

目标

clean package

2

执行清理和打包

POM ?

project-parent/pom.xml

3

设置父pom文件所在位置

属性 ?

保存

应用

4

保存配置

2.3.2 构建镜像和启动容器

这里我们以doc模块举例，操作如下图所示。

调用顶层 Maven 目标 ?

Maven 版本

maven3.6.3 1

目标

docker:stop docker:remove docker:build docker:start 2 停止容器、移除镜像、构建镜像、启动容器

POM ?

project-parent/project-doc/pom.xml 3 设置doc项目pom文件位置

属性 ?

保存 应用 4 保存设置

2.4 保存配置

所有操作执行完成后保存配置

构建后操作

增加构建后操作步骤 ▾

保存 应用

2.5 准备CA证书

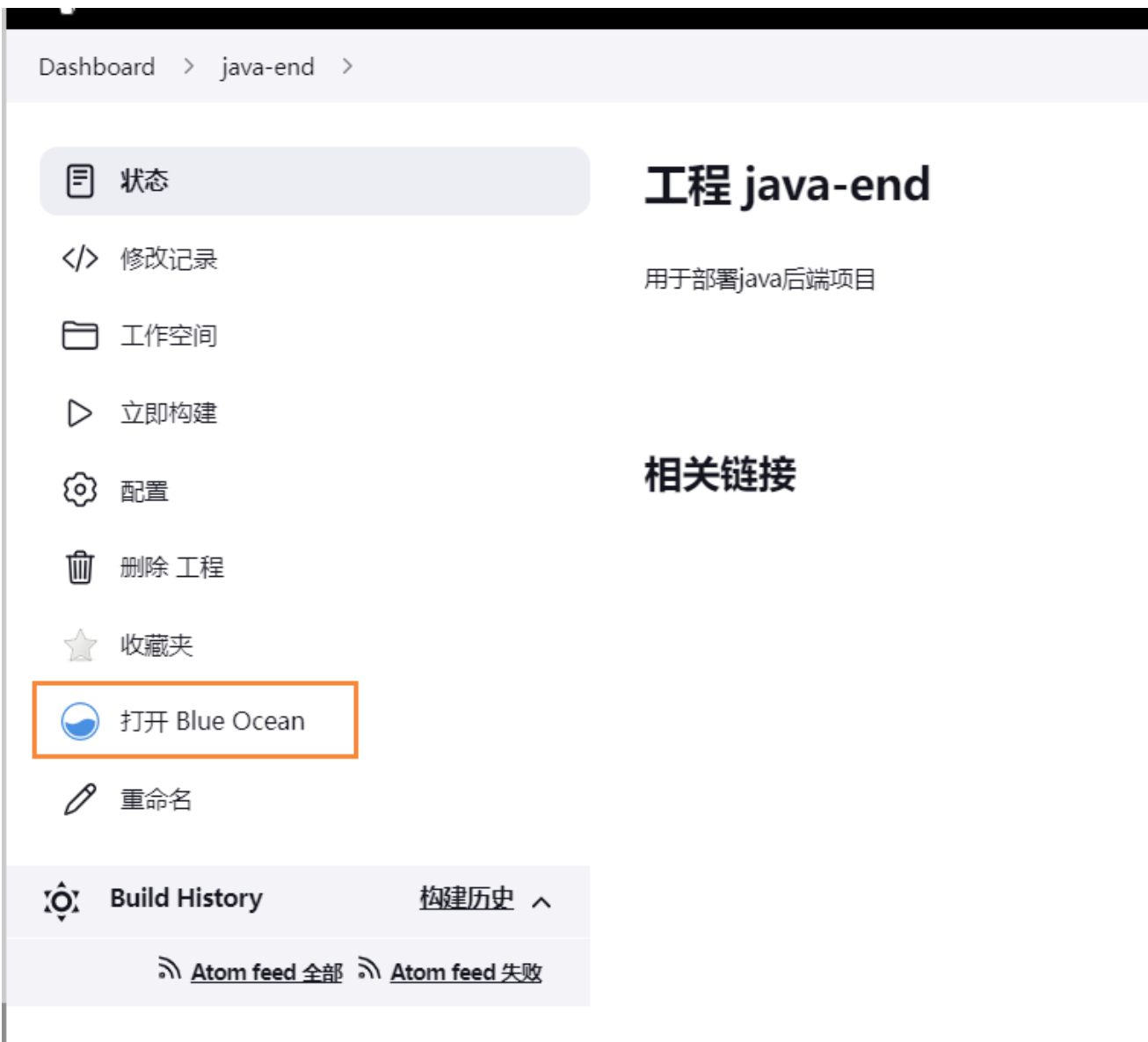
将客户端证书复制到Jenkins容器中，具体复制到那个目录就看你在书写docker-maven插件给的那个目录。

比如此时我要求的是/home/docker-ca，并且ca证书目前在宿主机的/home/docker-ca目录下面，我们使用cp命令把他复制过去。

```
1 docker cp /home/docker-ca/ jenkins:/home/docker-ca
```

2.6 执行任务

打开Blue Ocean



运行任务



2.7 查看日志

Jenkins

流水线配置管理

注册

java-end

活动分支Pull Requests

运行

Disable

状态	运行	提交	消息	持续时间	完成
	1	-	由用户 admin 启动	5s	-

点击正在执行的项目，来查看执行情况。

java-end 1

分支: - 1m 7s 没有修改

提交: - - 由用户 admin 启动

日志

```
1 Started by user admin
2 Running as SYSTEM
3 Building in workspace /var/jenkins_home/workspace/java-end
4 The recommended git tool is: NONE
5 using credential 7ed373b4-1a59-45ef-8562-7e697785e29f
6 Cloning the remote Git repository
7 Cloning repository git@github.com:
8 > git init /var/jenkins_home/workspace/jav
9 Fetching upstream changes from git@github.c
10 > git --version # timeout=10
11 > git --version # 'git version 2.36.2'
12 using GIT_SSH to set credentials github ssh private key
13 Verifying host key using known hosts file, will automati
14 > git fetch --tags --force --progress -- git@github.com:..
```

执行成功

```
900 [INFO] Finished at: 2023-03-16T19:39:44+08:00
901 [INFO] -----
902 [java-end] $ /var/jenkins_home/tools/hudson.tasks.Maven_MavenInstallation/maven3.6.3/bin/mvn -f project-parent/project-doc/pom.xml docker:stop docker:remove docker:build docker:start
903 [INFO] Scanning for projects...
904 [INFO] -----< com.zerone.star:project-doc >-----
905 [INFO] Building project-doc 1.0.0-SNAPSHOT
906 [INFO] -----[ jar ]-----
907 [INFO] --- docker-maven-plugin:0.40.2:stop (default-cli) @ project-doc ---
908 [INFO] --- docker-maven-plugin:0.40.2:remove (default-cli) @ project-doc ---
909 [INFO] --- docker-maven-plugin:0.40.2:build (default-cli) @ project-doc ---
910 [INFO] Building tar: /var/jenkins_home/workspace/java-end/project-parent/project-doc/target/docker/01star/project-doc/1.0.0-SNAPSHOT/tmp/docker-build.tar
911 [INFO] DOCKER> [01star/project-doc:1.0.0-SNAPSHOT] "project-doc": Created docker-build.tar in 676 milliseconds
912 [INFO] DOCKER> [01star/project-doc:1.0.0-SNAPSHOT] "project-doc": Built image sha256:41abd
913 [INFO] --- docker-maven-plugin:0.40.2:start (default-cli) @ project-doc ---
914 [INFO] DOCKER> [01star/project-doc:1.0.0-SNAPSHOT] "project-doc": Start container ec0cc03e2a69
915 [INFO] BUILD SUCCESS
916 [INFO] -----
917 [INFO] Total time: 14.730 s
918 [INFO] Finished at: 2023-03-16T19:40:00+08:00
919 [INFO] -----
920 Finished: SUCCESS
```

到你远程主机上去检查一下容器是否启动成功

```
[root@localhost workspace]# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
NAMES
ec0cc03e2a69   01star/project-doc:1.0.0-SNAPSHOT   "/bin/sh -c 'java $J..." 2 minutes ago  Up 2 minutes  0.0.0.0:9999->9999/tcp, :::9999->9999/tcp
```