7、IDEA中使用Docker插件构建boot单体项目为镜像，并自动部署到远程docker服务所在机器中，并运行出容器

Docker7~20.rar 16 MB

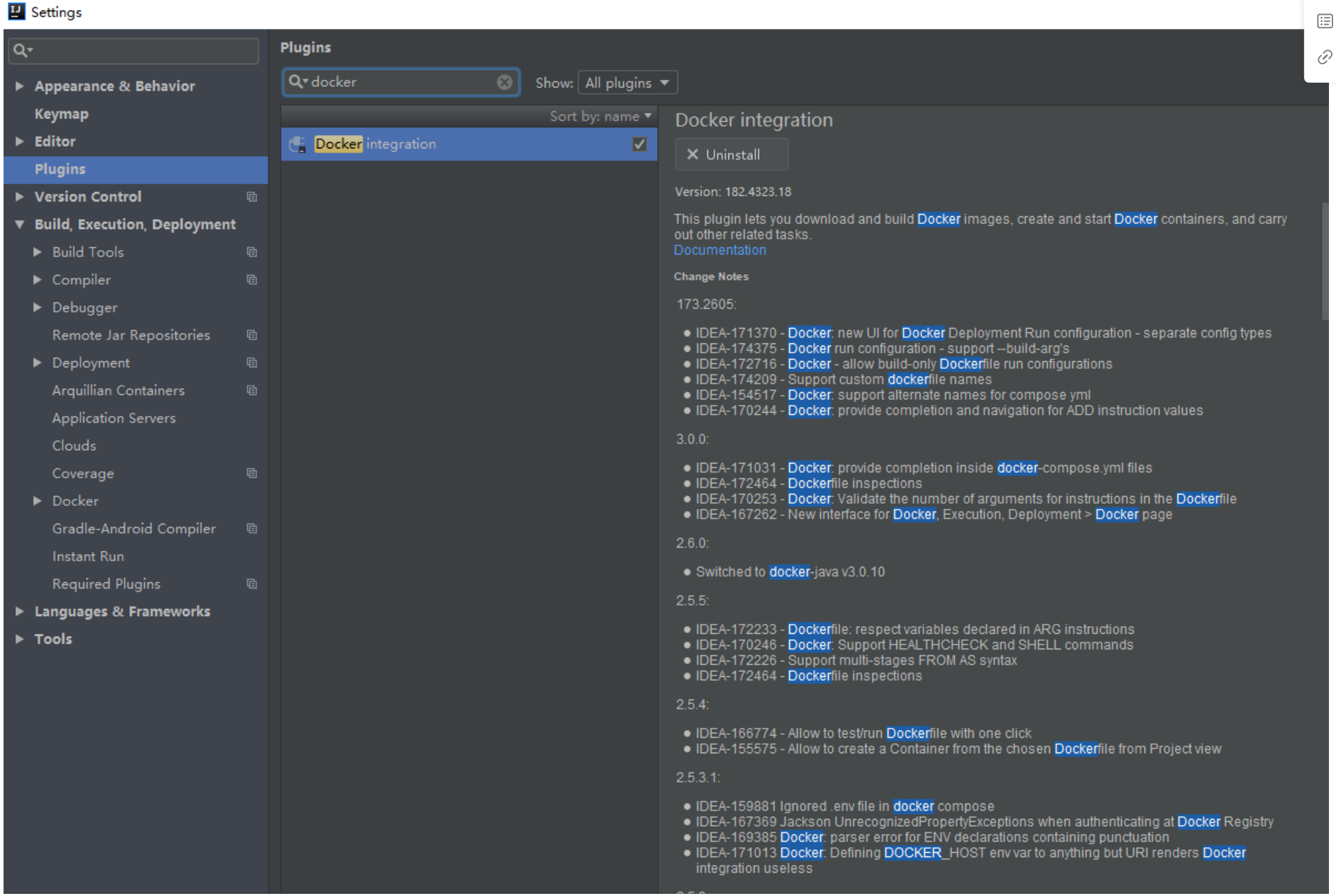
0、参考

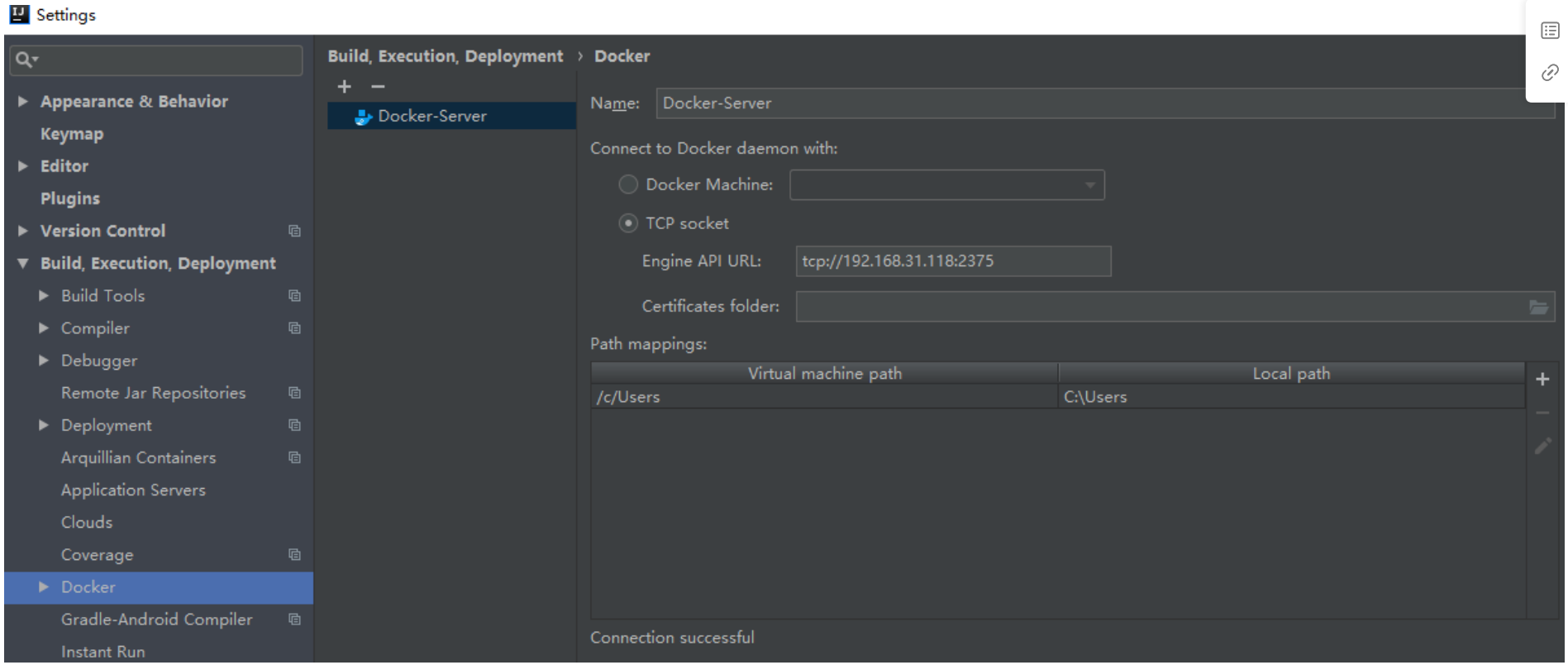
- https://www.cnblogs.com/hsz-csy/p/9488469.html
- https://www.cnblogs.com/alan6/p/11876567.html
- https://blog.csdn.net/qq_35976271/article/details/100287663
- https://blog.csdn.net/qq_36850813/article/details/92835885

springboot-demo-remote-docker.rar 58 KB

1、idea安装docker客户端插件并连接远程linux中的docker服务器

- **目的是在idea中通过docker客户端插件构建镜像到linux的docker服务器中**
- File-settings或ctrl+alt+s


Settings
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Build Tools
Compiler
Debugger
Remote Jar Repositories
Deployment
Arquillian Containers
Application Servers
Clouds
Coverage
Docker
Gradle-Android Compiler
Instant Run
Required Plugins
Languages & Frameworks
Tools
Plugins
docker
Show: All plugins
Sort by: name
Docker integration
Docker integration
Uninstall
Version: 182.4323.18
This plugin lets you download and build Docker images, create and start Docker containers, and carry out other related tasks.
Documentation
Change Notes
173.2605:
IDEA-171370 - Docker: new UI for Docker Deployment Run configuration - separate config types
IDEA-174375 - Docker run configuration - support --build-arg's
IDEA-172716 - Docker - allow build-only Dockerfile run configurations
IDEA-174209 - Support custom dockerfile names
IDEA-154517 - Docker: support alternate names for compose yml
IDEA-170244 - Docker: provide completion and navigation for ADD instruction values
3.0.0:
IDEA-171031 - Docker: provide completion inside docker-compose.yml files
IDEA-172464 - Dockerfile inspections
IDEA-170253 - Docker: Validate the number of arguments for instructions in the Dockerfile
IDEA-167262 - New interface for Docker, Execution, Deployment > Docker page
2.6.0:
Switched to docker-java v3.0.10
2.5.5:
IDEA-172233 - Dockerfile: respect variables declared in ARG instructions
IDEA-170246 - Docker: Support HEALTHCHECK and SHELL commands
IDEA-172226 - Support multi-stages FROM AS syntax
IDEA-172464 - Dockerfile inspections
2.5.4:
IDEA-166774 - Allow to test/run Dockerfile with one click
IDEA-155575 - Allow to create a Container from the chosen Dockerfile from Project view
2.5.3.1:
IDEA-159881 Ignored .env file in docker compose
IDEA-167369 Jackson UnrecognizedPropertyExceptions when authenticating at Docker Registry
IDEA-169385 Docker: parser error for ENV declarations containing punctuation
IDEA-171013 Docker: Defining DOCKER_HOST env var to anything but URI renders Docker integration useless



2、docker服务开启远程访问2375端口

- 在linux环境下,vim打开 /usr/lib/systemd/system/docker.service文件，找到 ExecStart，在此行的最后面添加监听端口 -H tcp://0.0.0.0:2375，表示打开2375端口，支持远程连接docker：

```
[Service]
ExecStart=/usr/bin/dockerd -H fd:// --container=/run/containerd.sock -H tcp://0.0.0.0:2375
```

- 重启docker服务

```
systemctl daemon-reload
systemctl start docker
```

3、拉取docker镜像的加速配置（配置国内的镜像仓库）

```
[root@MiWiFi-R4A-srv tmp]# vim /etc/docker/daemon.json
```

```
"registry-mirrors": ["https://docker.mirrors.ustc.edu.cn/","https://hub-mirror.c.163.com","https://registry.docker-cn.com"],
"insecure-registries": ["10.0.0.12:5000"]
```

4、构建前注意事项

- Dockerfile中的ADD后面的jar文件名必须与项目打包后的target/xxx.jar名字保持一致
- Dockerfile中我们添加了一个VOLUME指向“/tmp”的内容，因为这是Spring Boot应用程序默认为Tomcat创建工作目录的地方。效果是在主机“/var/lib/docker”下创建一个临时文件，并将其链接到“/tmp”下的容器。对于我们在此处编写的简单应用程序，此步骤是可选的，但如果需要在文件系统中实际编写，则对于其他Spring Boot应用程序可能是必需的。

5、项目中通过pom.xml文件中来配置docker客户端插件docker-maven-plugin（使用pom文件形式引入docker镜像构建插件并进行构建镜像，和1一样只是以配置文件形式安装插件）

```
<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
 <java.version>1.8</java.version>
 <dockerfile-version>1.0.0或1.2.2</dockerfile-version>
</properties>

<build>
 <plugins>
  <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
  <!--
    https://github.com/spotify/docker-maven-plugin
    https://blog.csdn.net/aixiaoyang168/article/details/77453974
  -->
  <!--docker客户端插件构建boot项目为镜像，部署到远程linux中-->
  <plugin>
        <groupId>com.spotify</groupId>
        <artifactId>docker-maven-plugin</artifactId>
```

```
        <artifactId>docker-maven-plugin</artifactId>
        <version>${dockerfile-version}</version>
        <configuration>
          <!--指定生成的镜像名：就是项目的artifactId值-->
          <imageName>${project.artifactId}</imageName>
          <!--指定标签 这里指定的是镜像的版本，我们默认版本是latest-->
          <imageTags>
            <imageTag>latest</imageTag>
          </imageTags>
          <!--指定Dockerfile文件的位置：这里指定的是项目根目录/。
                也可以自定义指定位置,如：${project.basedir}/src/main/docker
          -->
          <dockerDirectory>${project.basedir}</dockerDirectory>
          <!--是否跳过构建镜像的步骤-->
          <skipDockerBuild>false</skipDockerBuild>
          <!--指定docker镜像要部署到的docker服务(2375)所在的机器地址，可以配置到配置文件中动态取值-->
          <dockerHost>http://192.168.31.118:2375</dockerHost>
          <!-- 构建项目时需要将哪些文件进行打包构建镜像 -->
          <resources>
            <resource>
              <!--这里目的是找到directory目录中include包含的jar，然后构建成镜像-->
              <targetPath>/</targetPath>
              <!--jar 包所在的路径 ： 此处路径是（ 项目名/target 目录 ）-->
              <directory>${project.build.directory}</directory>
              <!-- jar包名字 ： 这里project.build.finalName是项目名-版本，必须对应的是 Dockerfile中ADD中构建的jar
                  名字
               -->
              <include>${project.build.finalName}.jar</include>
            </resource>
          </resources>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

- pom中也可以配置**dockerfile-maven-plugin插件**构建推送镜像到harbor仓库中

Dockerfile Maven的特性

默认，构建Dockerfile的过程包含在mvn package阶段；

默认，为Dockerfile打标签的过程包含在mvn package阶段；

默认，发布Dockerfile的过程包含在mvn deploy阶段；

也可以直接陆续执行：

```
mvn dockerfile:build #构建镜像
mvn dockerfile:tag #为镜像打标签
mvn dockerfile:push #发布（推送）镜像
#完整的命令（从gitlab中使用git pull拉取变更的项目后，使用编译，打成jar包，安装jar包到本地，构建镜像，打标签，推送镜像到harbor镜像仓库，跳过test）
mvn clean install dockerfile:build dockerfile:tag dockerfile:push -Dmaven.test.skip=true
#运行镜像，创建出容器
docker-compose up -d 镜像名  或  docker-compose up --build -d 镜像名
#查看运行的容器的最后100行日志
docker logs -f --tail 100 容器名
```

说明：

其中，${dockerfile-maven-version}为Dockerfile Maven插件的版本，当前为1.4.0；

${project.version}为Docker项目的版本；

${project.build.finalName}.jar为Docker项目构建生成的组件，JAR包形式；

构建Docker项目时，直接执行mvn deploy即可构建并发布Dockerfile文件到Maven本地库spotify/foobar。

```
<project>
    <properties>
        <!-- 推荐使用Harbor -->
        <docker.registry.url>harbor.scmsafe.com</docker.registry.url>
        <docker.registry.host>https://${docker.registry.url}</docker.registry.host>
        <docker.username>coral</docker.username>
        <docker.password>coral123456</docker.password>
        <docker.namespace>coral</docker.namespace>
        <docker.plugin.version>1.4.13</docker.plugin.version>
    </properties>

    <plugin>
        <groupId>com.spotify</groupId>
        <artifactId>dockerfile-maven-plugin</artifactId>
```

```
        <version>${docker.plugin.version}</version>
        <configuration>
            <skip>true</skip>
            <username>${docker.username}</username>
            <password>${docker.password}</password>
            <repository>${docker.registry.url}/${docker.namespace}/${project.artifactId}</repository>
            <tag>${project.version}</tag>
            <useMavenSettingsForAuth>true</useMavenSettingsForAuth>
            <buildArgs>
                <JAR_FILE>target/${project.build.finalName}.jar</JAR_FILE>
            </buildArgs>
        </configuration>
        <!--子服务添加如下配置，运行 mvn deploy 命令便会自动构建推送镜像-->
        <!--<executions>
            <execution>
                <id>default</id>
                <goals>
                    <goal>build</goal>
                    <goal>push</goal>
                </goals>
            </execution>
        </executions>
        -->
    </plugin>


    <distributionManagement>
        <repository>
          <id>nexus-release</id>
          <url>http://47.94.41.79:8081/repository/maven-releases/</url>
        </repository>
        <snapshotRepository>
          <id>nexus-snapshots</id>
          <url>http://47.94.41.79:8081/repository/maven-snapshots/</url>
        </snapshotRepository>
      </distributionManagement>

      <repositories>
        <repository>
          <id>aliyun-repos</id>
```

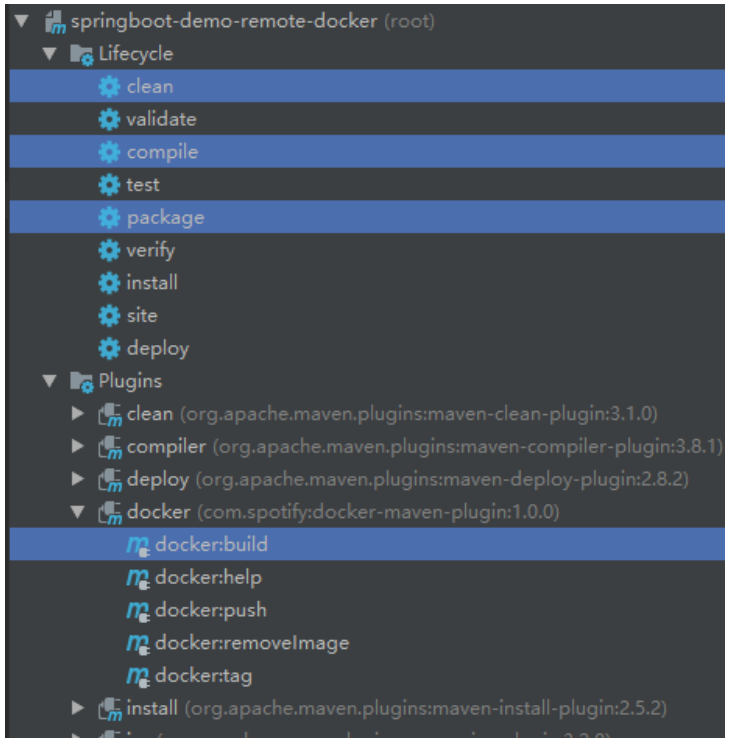
```
            <url>https://maven.aliyun.com/nexus/content/groups/public/</url>
            <snapshots>
              <enabled>false</enabled>
            </snapshots>
          </repository>
          <repository>
            <id>nexus-public</id>
            <url>http://47.94.41.79:8081/repository/maven-public/</url>
            <releases>
              <enabled>true</enabled>
            </releases>
            <snapshots>
              <enabled>true</enabled>
            </snapshots>
          </repository>
        </repositories>

        <pluginRepositories>
          <pluginRepository>
            <id>aliyun-plugin</id>
            <url>https://maven.aliyun.com/nexus/content/groups/public/</url>
            <snapshots>
              <enabled>false</enabled>
            </snapshots>
          </pluginRepository>
        </pluginRepositories>
</project>
```

- 查看顶级项目的pom.4.0.0.xml的内容

```
${M2_HOME}/lib/maven-model-builder-3.0.3.jar
用7-zip打开jar（或使用jar工具）
org/apache/maven/model，找到pom-4.0.0.xml
//内容
<project>
  ...
  <build>
    <directory>${project.basedir}/target</directory>
    <outputDirectory>${project.build.directory}/classes</outputDirectory>
    <finalName>${project.artifactId}-${project.version}</finalName>
    <testOutputDirectory>${project.build.directory}/test-classes</testOutputDirectory>
    <sourceDirectory>${project.basedir}/src/main/java</sourceDirectory>
    <scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>
    <testSourceDirectory>${project.basedir}/src/test/java</testSourceDirectory>
    <resources>
      <resource>
        <directory>${project.basedir}/src/main/resources</directory>
      </resource>
    </resources>
    <testResources>
      <testResource>
        <directory>${project.basedir}/src/test/resources</directory>
      </testResource>
    </testResources>
    ...
  </build>
  ...
</project>
```

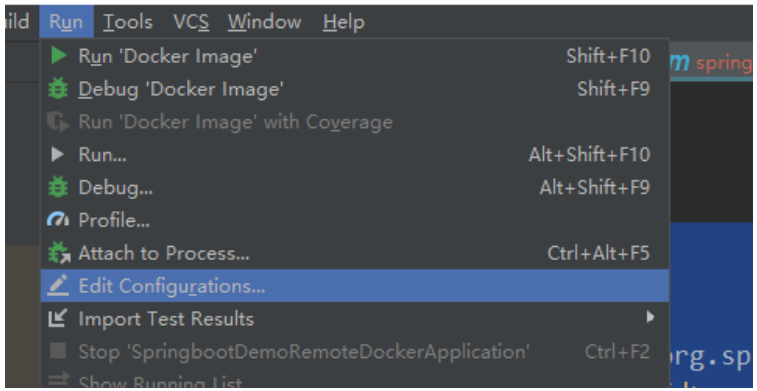
6、使用pom.xml中的docker客户端插件构建镜像（使用pom文件形式安装插件并进行构建镜像，和1一样只是以配置文件形式引入插件并构建镜像）

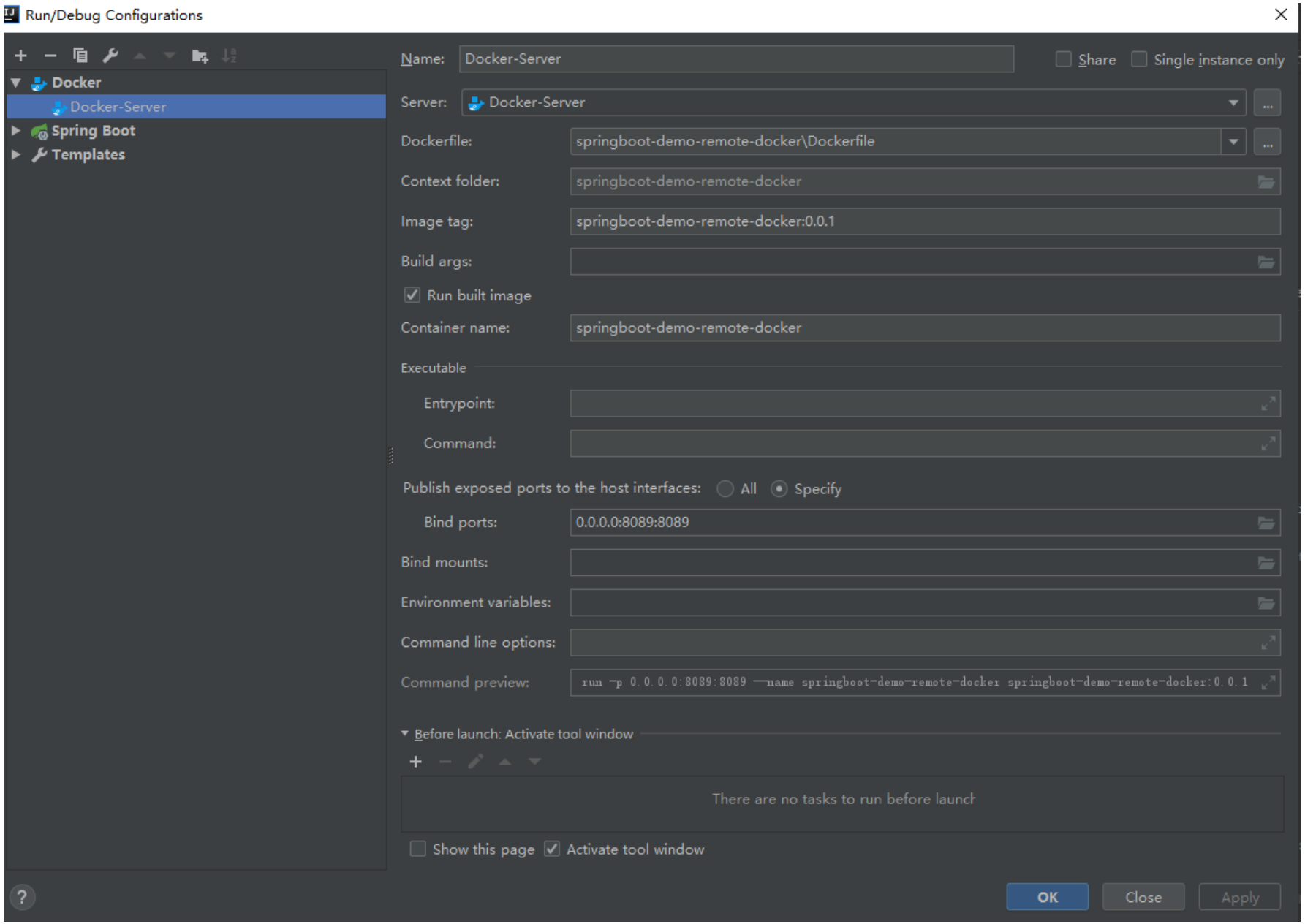


7、idea中使用已经安装的docker客户端插件来构建部署镜像到远程docker所在服务器中。（1或者56选一种即可，这里使用1来配置和构建镜像）

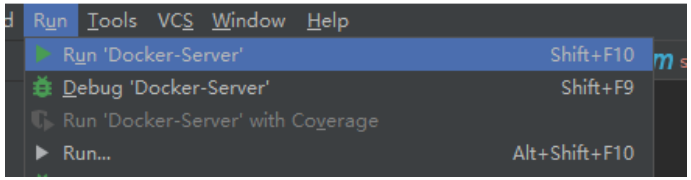
（1）第1-2步已经安装docker客户端插件及配置

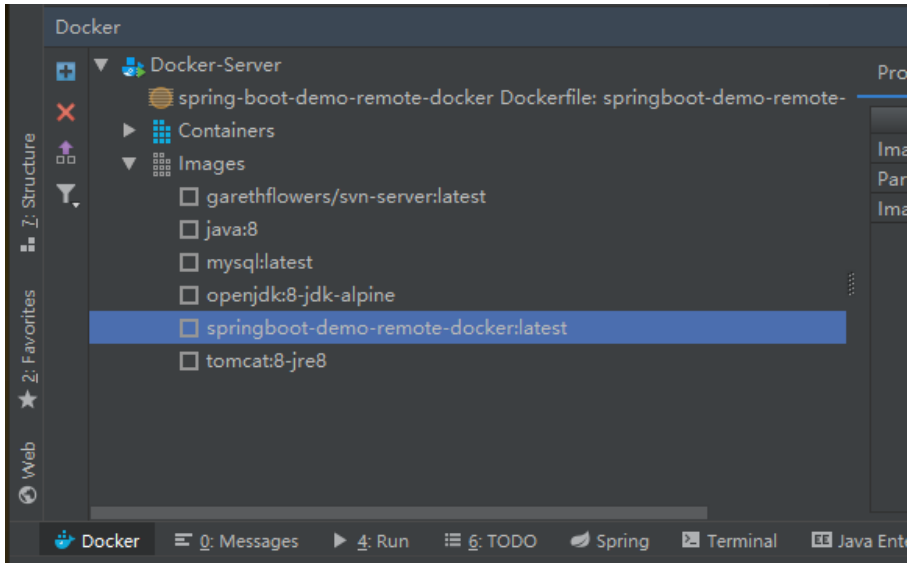
（2）配置docker客户端插件及构建项目为镜像到linux过程的参数（镜像名，版本号，映射端口）





（3）以可视化docker客户端插件构建部署镜像到**到远程docker所在服务器中**





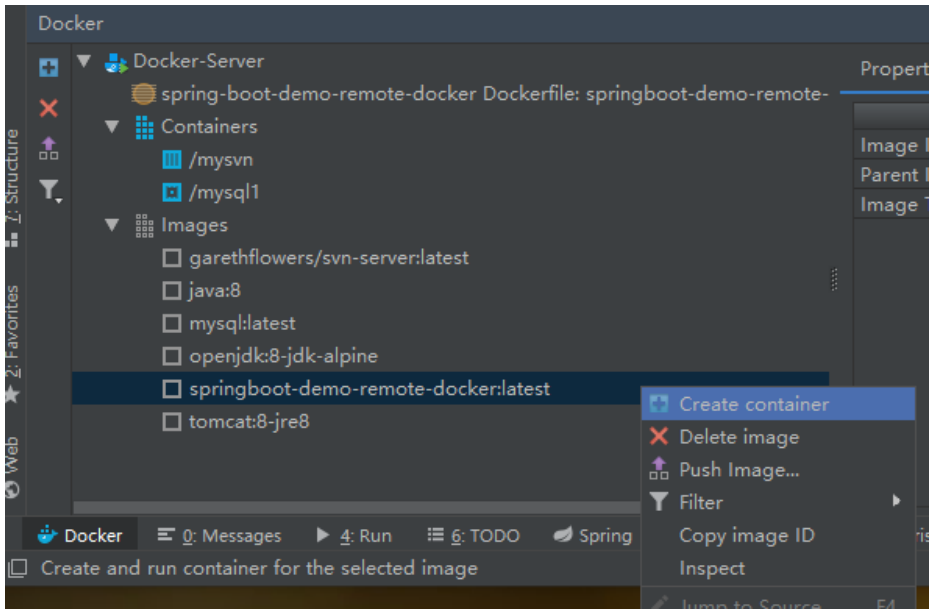
（4）以命令行的方式构建部署镜像**到远程docker所在服务器**

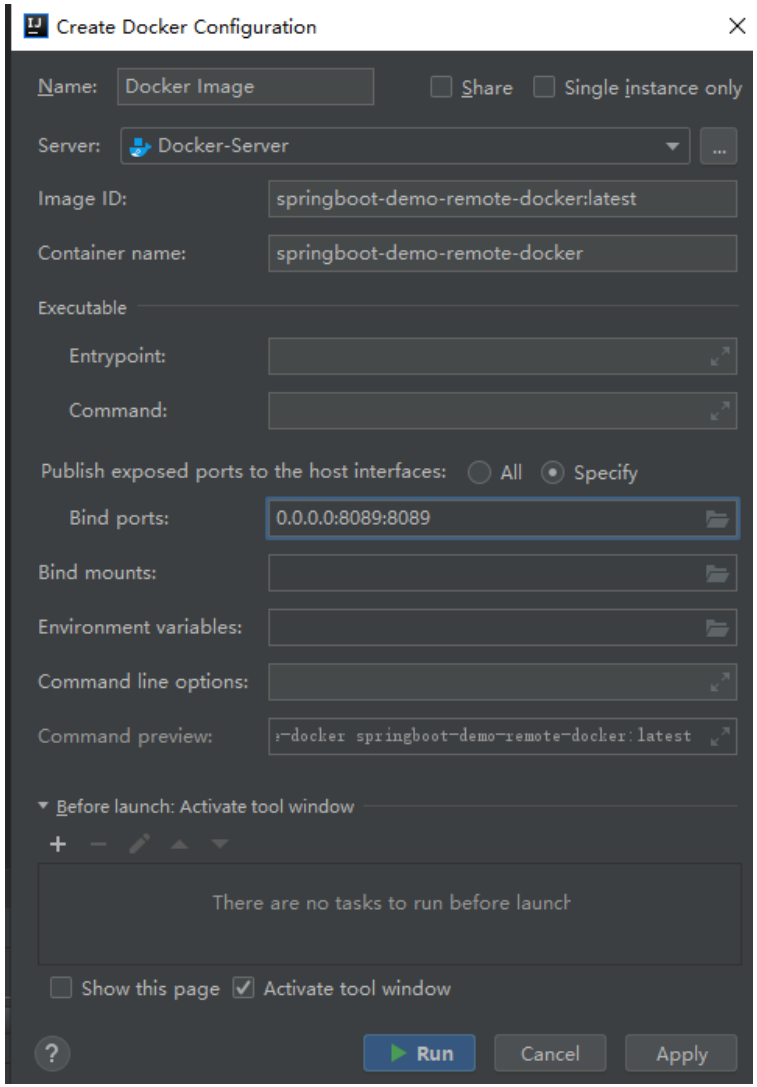
- Maven中-DskipTests和-Dmaven.test.skip=true的区别

```
mvn clean package -DskipTests docker:build
```

8、构建镜像后，可以创建并运行出一个容器

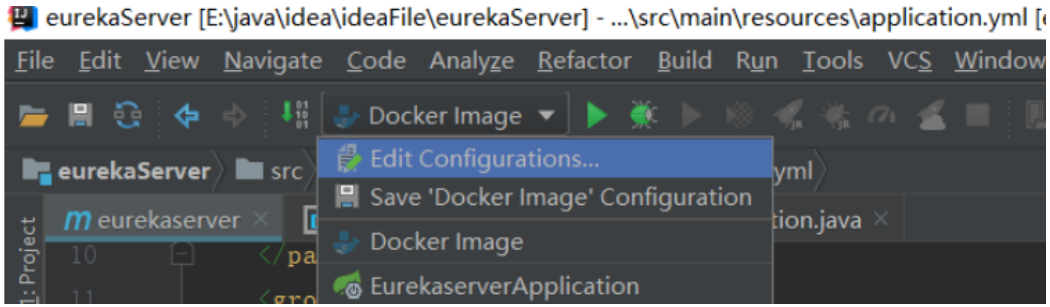
- （1-1）xshell连接到docker所在linux服务器，docker images命令可以看到刚刚IDEA中构建部署过来的镜像
- （1-2）docker服务器中可以运行出容器
- （2-1）idea中也能看到刚刚构建的镜像
- （2-2）idea中也能创建出一个容器（配置容器的信息）并运行
- 而且当项目根目录打包时，每个服务的镜像都会上传到远程docker服务所在主机并自动重启容器


Docker
Docker-Server
spring-boot-demo-remote-docker Dockerfile: springboot-demo-remote-
Containers
/mysvn
/mysql1
Images
garethflowers/svn-server:latest
java:8
mysql:latest
openjdk:8-jdk-alpine
springboot-demo-remote-docker:latest
tomcat:8-jre8
Create container
Delete image
Push Image...
Filter
Copy image ID
Inspect
Docker
0: Messages
4: Run
6: TODO
Spring
Create and run container for the selected image
1: Structure
2: Favorites
Web

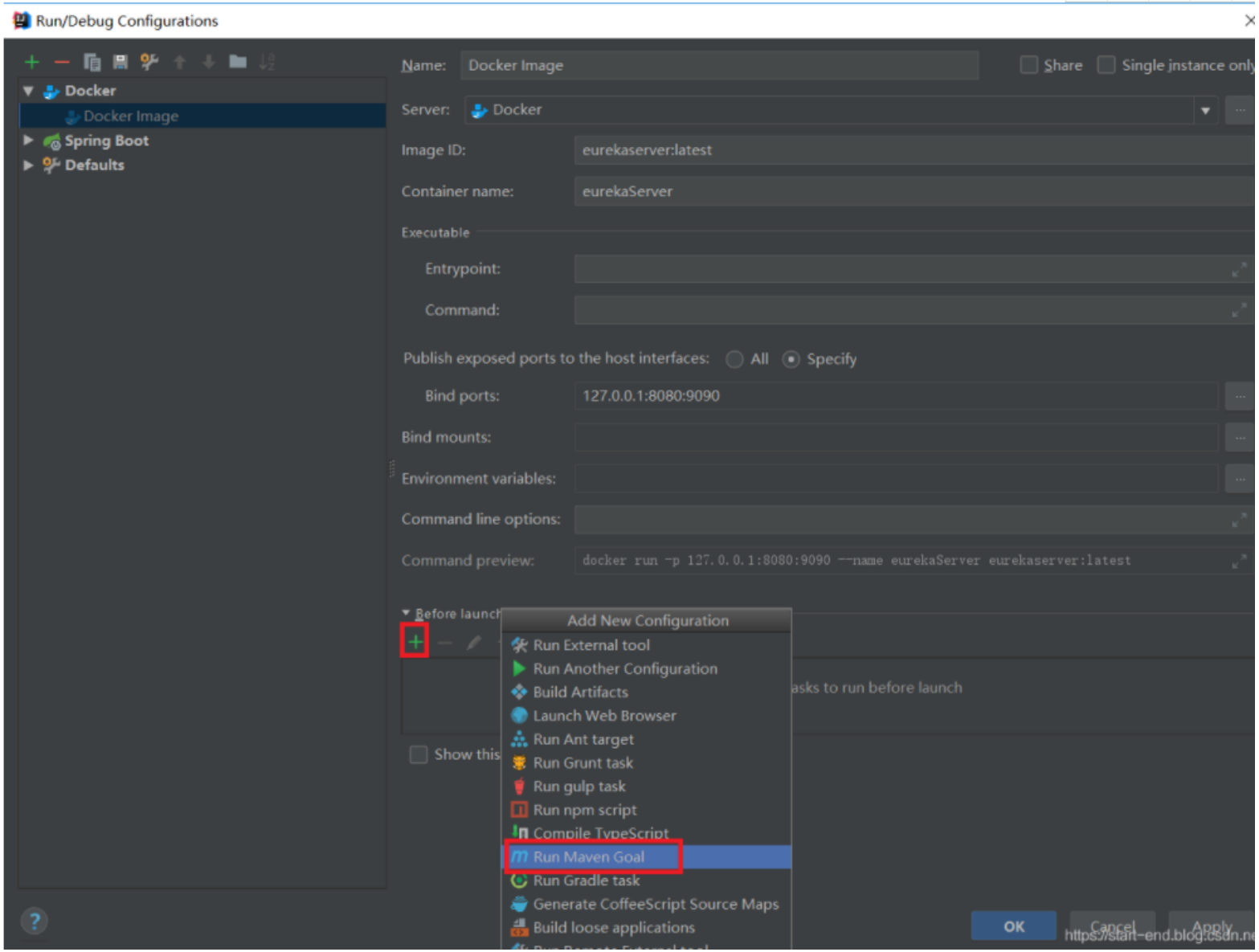


（2-3）如果我们修改了项目，再使用docker插件启动项目，会发现在启动依然还是修改前的项目，因为我们只是启动了容器，并没有将修改后的项目重新打包并生成docker的images。如果我们想在启动时直接用运行打包并启动项目可以按照下面的操作

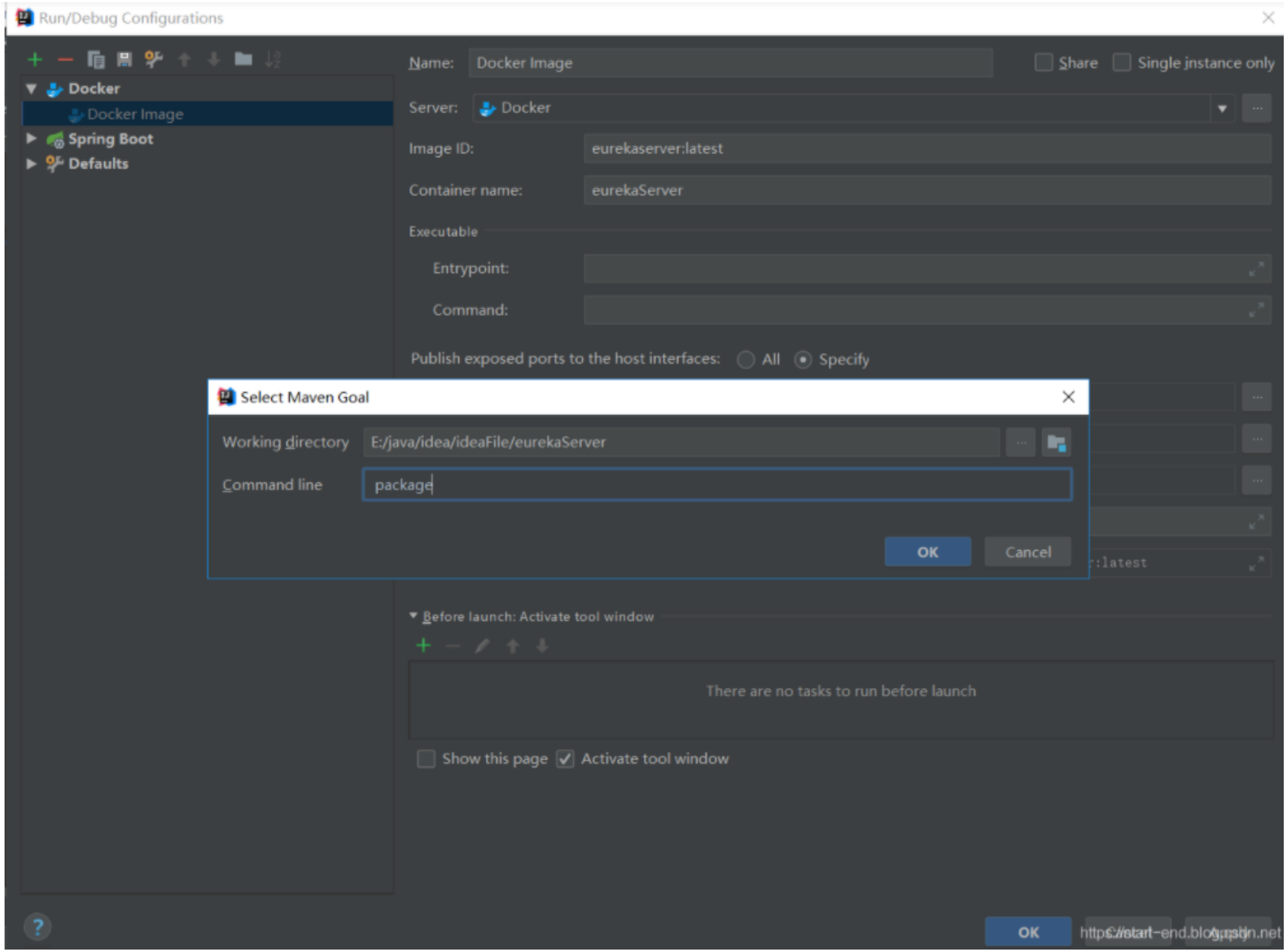
- 我们修改前面我们创建的Docker Images 的配置



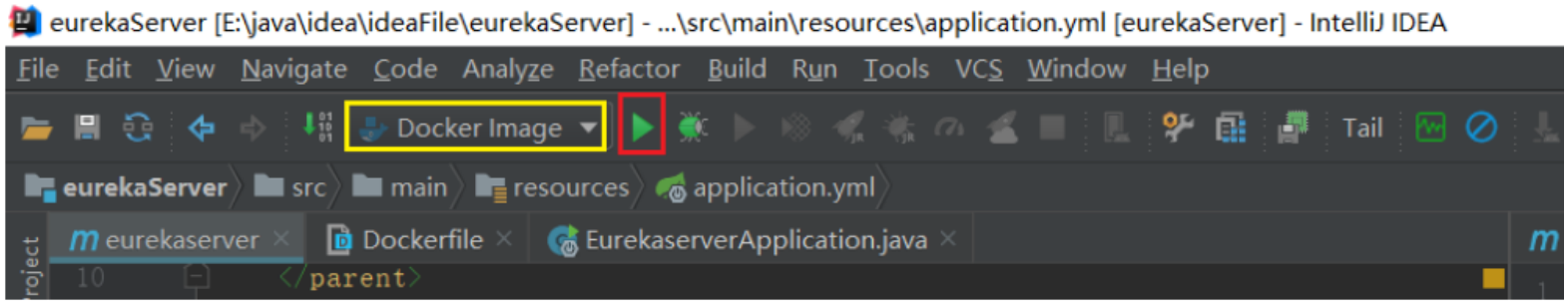
- 在配置启动项中找到Before launch：Activate tool window，在这里我们新增一个Run Maven Gold



- 我们在这里的Command line 中添加一个命令 package。配置后保存。



- 以后我们启动项目时都会执行maven的package命令，自动将我们打包并把项目生成docker镜像文件启动。
- 以后，如果我们修改项目后，都可以使用下图所示去启动。它会自动打包创建docker镜像并启动项目。



- 如果我们只需要启动项目，到docker插件窗口，启动对应项目的容器就可以了

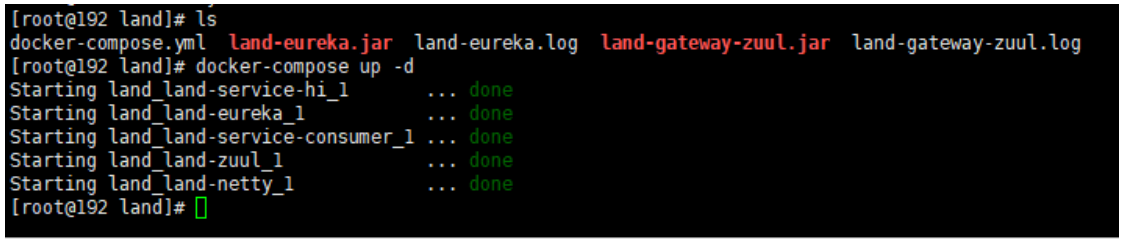
（2-4）当多服务时，可以使用上面手动创建容器的方法，让远程docker服务所在主机启动容器。也可以待各个服务镜像上传到远程主机后，使用docker-compose完成服务编排，然后同时启动各个微服务容器。

1）在远程docker服务所在主机编辑docker-compose.yml文件

```
version: '2'
services:
land-eureka:
image: land/land-eureka:latest
container_name: land-eureka
ports:
- '8761:8761'
land-zuul:
image: land/land-zuul:latest
container_name: land-zuul
ports:
- '9000:9000'
land-netty:
image: land/land-netty:latest
container_name: land-netty
ports:
- '8000:8000'
land-service-hi:
image: land/land-service-hi:latest
container_name: land-service-hi
ports:
- '8800:8800'
land-service-consumer:
image: land/land-service-consumer:latest
container_name: land-service-consumer
ports:
- '8801:8801'
```

2）在docker-compose.yml文件目录下运行docker-compose命令，启动容器

```
docker-compose up -d
```



9、访问测试

http://192.168.31.118:8089/demo/hello

boot项目的端口是8089，项目路径/demo

```yaml
server:
  port: 8089
  servlet:
    context-path: /demo
```

10、出现的问题及处理

（1）、构建项目镜像过程需要pull拉取jdk时失败的解决

```
[root@demaxiya usr]# vi /etc/resolv.conf
# Generated by NetworkManager
nameserver 192.168.33.1
nameserver 8.8.8.8
~
~
~
```

```
-bash: nameserver: 未找到命令
-bash: nameserver: 未找到命令
[root@demaxiya usr]# service docker restart
Redirecting to /bin/systemctl restart docker.service
[root@demaxiya usr]#
[root@demaxiya usr]#
[root@demaxiya usr]#
[root@demaxiya usr]# docker pull docker.io/java
Using default tag: latest
Trying to pull repository docker.io/library/java ...
latest: Pulling from docker.io/library/java
5040bd298390: Pull complete
fce5728aad85: Pull complete
76610ec20bf5: Pull complete
60170fec2151: Pull complete
e98f73de8f0d: Pull complete
11f7af24ed9c: Pull complete
49e2d6393f32: Pull complete
bb9cdec9c7f3: Pull complete
Digest: sha256:5d6cc05636c16abb3e49f2117fd6aa4bddb8f0d775075972a8b9b03f754ea9f2
Status: Downloaded newer image for docker.io/java:latest
[root@demaxiya usr]#
就绪                    ssh2: AES-256-CTR    37,  22  37 行, 107 列  VT100    大
```

（2）pull时io超时

```
yum -y install bind-utils
vim /etc/hosts 将answer中的ip和域名映射关系配置进去再重新pull
```

（3）stat /var/lib/docker/tmp/docker-builderXXXXXX: no such file or directory

Dockerfile中的ADD后面的jar文件名必须与项目打包后的target/xxx.jar名字保持一致

(4)Dockfile文件内容

```
# 基础jdk镜像
FROM java:8
# 作者信息
MAINTAINER "GaoMing.Han 1534834526@qq.com"
# 添加一个存储空间
VOLUME /tmp
# 暴露8089端口（外部访问时linux内网暴露的端口）
EXPOSE 8089
# 添加变量，如果使用dockerfile-maven-plugin，则会自动替换这里的变量内容（此处的jar文件名必须和package后的target中的文件名保持一致）
ARG JAR_FILE=target/springboot-demo-remote-docker-0.0.1-SNAPSHOT.jar
# 往容器中添加jar包
ADD ${JAR_FILE} app.jar
# 启动镜像自动运行程序
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/urandom","-jar","/app.jar"]
```

（5）在执行Dockerfile时出现错误Gethttps://registry-1.docker.io/v2/

```
//配置远程镜像仓库加速器,参考笔记4
vim /etc/docker/daemon.json
{
"registry-mirrors":["https://6kx4zyno.mirror.aliyuncs.com"]
}
```

```
systemctl daemon-reload
systemctl restart docker
重新拉取pull
```

其它文章

https://www.jianshu.com/p/c435ea4c0cc0