

A 服务器环境: 1.8jdk, docker,ssh 外网访问地址 222.24.63.59 端口 8080:9064
8081:9066 内网: 172.19.100.17

B 服务器环境: 1.8jdk, docker,ssh 外网访问地址 222.24.63.59, 端口 8080:9065
8081:9067 内网: 172.19.100.20

在部署一个测试后我们进行对单点登录的部署.

单点登录项目: 采用 spring boot, spring cloud , redis , mysql。

项目概述:

注册: 需要用户的电话, 邮件, 用户名, 不能与数据库重复, 注册成功后会弹出输入激活码, 此时通过你所填的邮箱收取激活码并填入激活账号。

登录: 将输入的邮箱与密码对比与 mysql 数据库进行对比, 成功则将 key: tokenId , value: 用户数据存入 redis 中, 并将 tokenId 存入 cookie 中。

微服务:

LOGINWEB: 前置服务, 主要存放页面, 和做验证码的校验, 并且去发现 **LOGINSO** 服务完成业务处理。

LOGINSO: 业务微服务, 主要服务逻辑处理, 被前置任务所调用。

EUREKASERVER0-0: spring cloud 组件用于服务的注册与发现 (eureka)。

先分析一下构建 docker 的 dockerfile

Dockerfilejava 的内容如下:

```
FROM jdk1.8
```

```
ARG rest
```

```
MAINTAINER zlw
```

```
COPY $rest /user/local // rest 是参数在构建的时候传入表示将 xxxx.jar 复制到该容器中  
WORKDIR /user/local
```

```
EXPOSE $port //port 是参数, 表示将该容器的某各端口暴露, 这个要与你运行的 xxx.jar 的  
端口一致
```

```
ENV JAVA_HOME=/user/local/jdk1.8.0_171
```

```
ENV JRE_HOME=/user/local/jdk1.8.0_171/jre
```

```
ENV PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
```

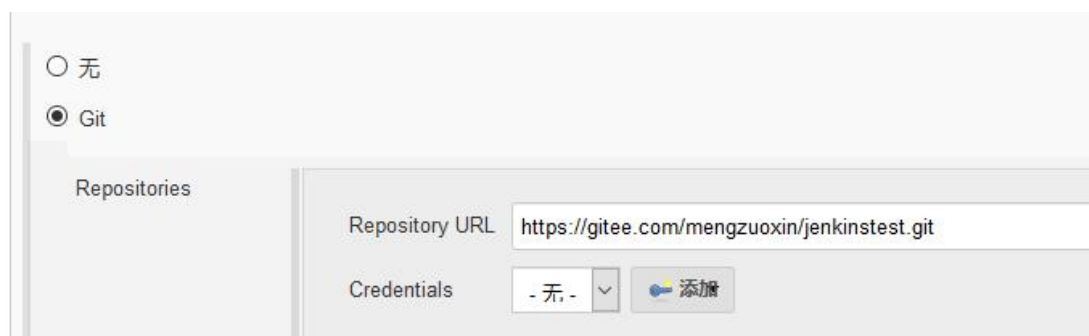
```
ENV CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib
```

```
ENV export JAVA_HOME JRE_HOME PATH CLASSPATH //java 环境变量
```

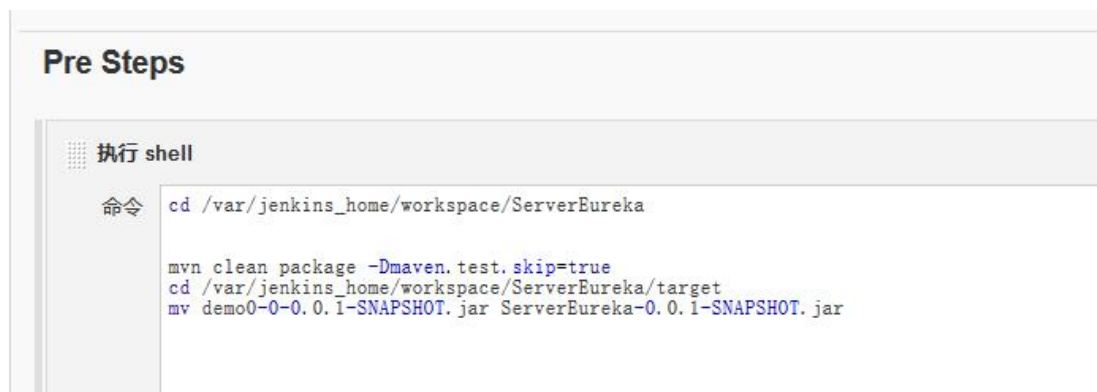
```
CMD [ "top" ]
```

先部署 **EUREKASERVER0-0:** 分别在 A 和 B 上部署端口为 8080

Git 地址:



shell 执行：



Shell 执行的位置为 jenkins 容器中。

`cd /var/jenkins_home/workspace/ServerEureka` //到该项目目录下

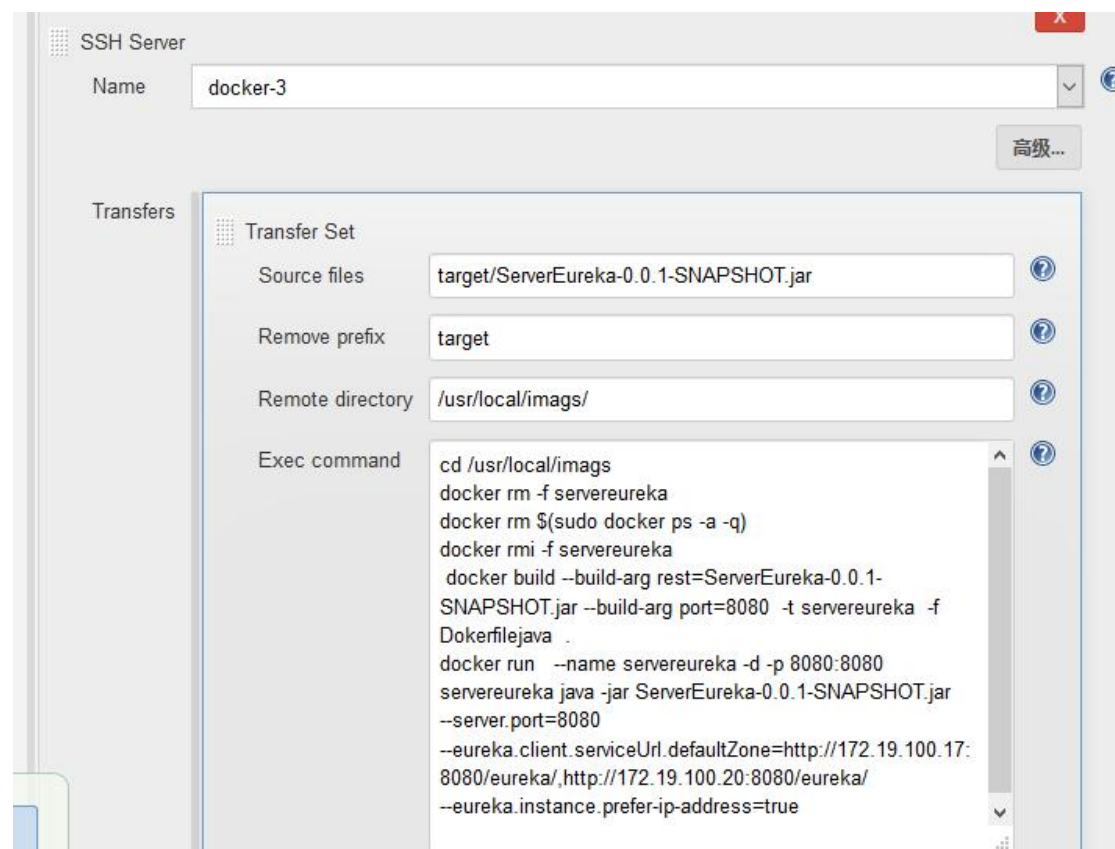
`mvn clean package -Dmaven.test.skip=true` //将 java 项目打成 jar 包

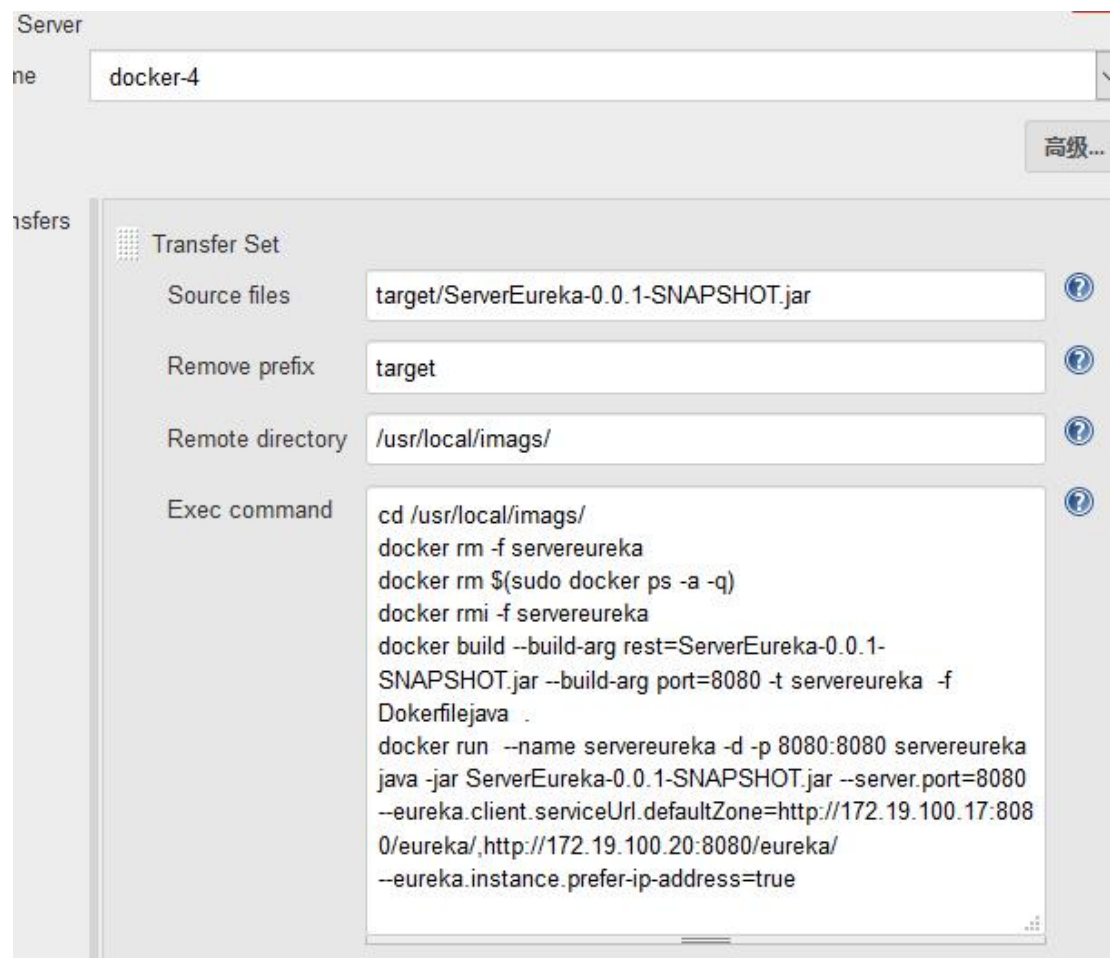
`cd /var/jenkins_home/workspace/ServerEureka/target`

`mv demo0-0-0.0.1-SNAPSHOT.jar ServerEureka-0.0.1-SNAPSHOT.jar` //做一个重命名（可选）

其中 `demo0-0-0.0.1-SNAPSHOT.jar` //这个名字由代码构建时取的名字

在生成 jar 后。





docker-3 和 docker-4shell 一样构建的 shell 为:

cd /usr/local/imags //到 A 虚拟机此目录下 (这个目录下存在 dockerfile)
 docker rm -f servereureka //去除以前创建的该容器, 否则会因为重命名启动失败

docker rm \$(sudo docker ps -a -q) //去除没有运行的容器, 否则会因为重命名启动失败

docker rmi -f servereureka //删除上次构建的容器, 否则会因为重命名多出来

docker build --build-arg rest=ServerEureka-0.0.1-SNAPSHOT.jar --build-arg port=8080 -t servereureka -f Dockerfilejava .//构建容器

docker run --name servereureka -d -p 8080:8080 servereureka java -jar ServerEureka-0.0.1-SNAPSHOT.jar --server.port=8080 --eureka.client.serviceUrl.defaultZone=http://172.19.100.17:8080/eureka/,http://172.19.100.20:8080/eureka/ --eureka.instance.prefer-ip-address=true //运行容器在 8080 , 自我注册和互相注册

部署成功:

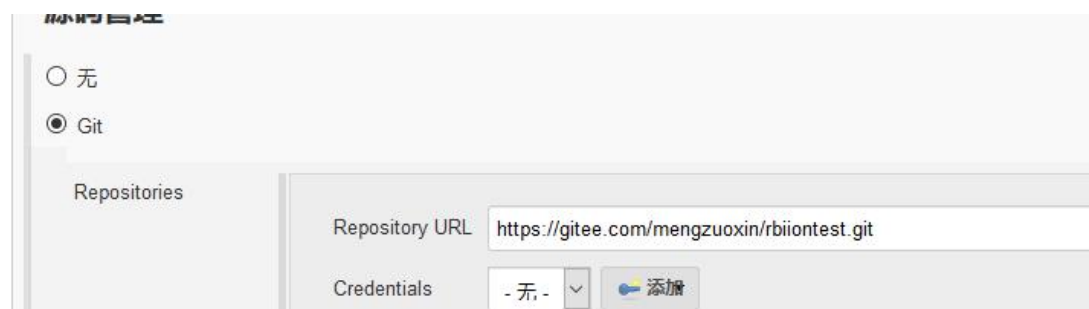


通过映射，可通过 <http://222.24.63.59:9065/>，<http://222.24.63.59:9064/> 进行访问相互注册以及自我注册

Application	AMIs	Availability Zones	Status
EUREKASERVER0-0	n/a (2)	(2)	UP (2) - a20b73913053:eurekaserver0-0:8080 , c518878846cb:eurekaserver0-0:8080

部署 LOGINSSO：在 A 和 B 上部署，端口随意，因为和 LOGINWEB 服务处理同一内网，LOGINWEB 可以调用。

GitHub:



Shell 执行:

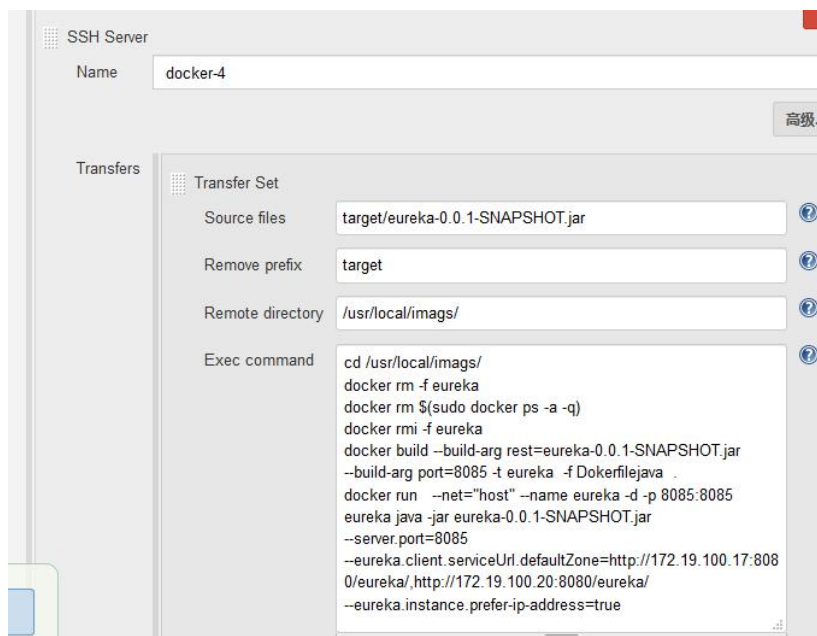
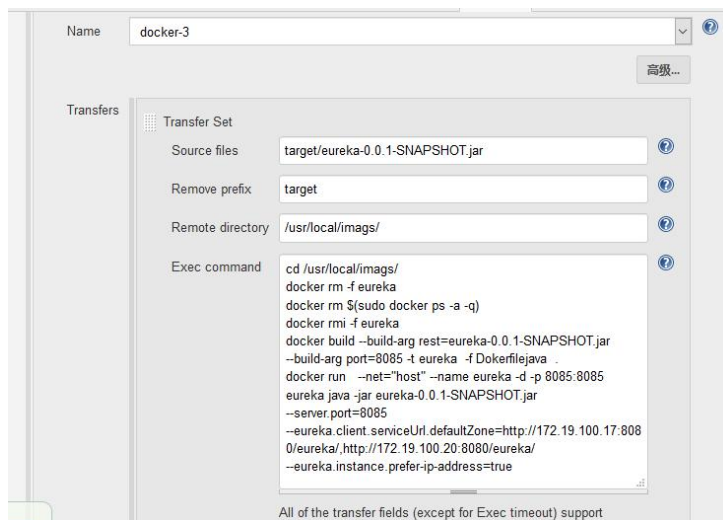
Pre Steps

执行 shell

```
命令 cd /var/jenkins_home/workspace/eureka

mvn clean package -Dmaven.test.skip=true
cd /var/jenkins_home/workspace/eureka/target
mv xin.menzuo.ttms-customer-0.0.1-SNAPSHOT.jar eureka-0.0.1-SNAPSHOT.jar
```

构建 shell:

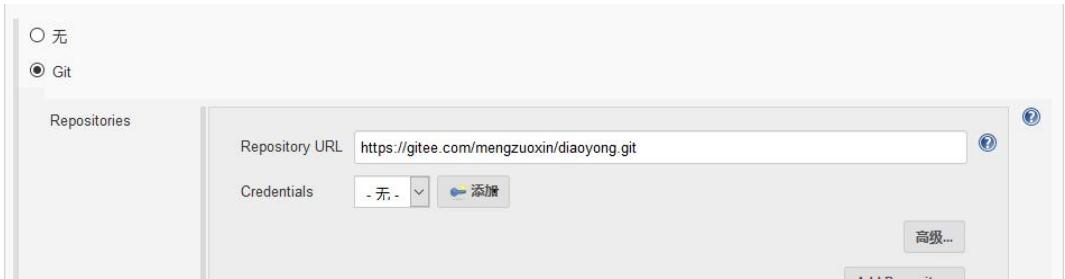


注意其中 `--net="host"` 表示容器运行采用主机的地址运行。这样 LOGINWEB 才能发现处于不同虚拟机的服务。


部署成功:

LOGINSO	n/a (2)	(2)	UP (2) - localhost:loginso:8085 , springcloud-2:loginso:8085
---------	------------	-----	--

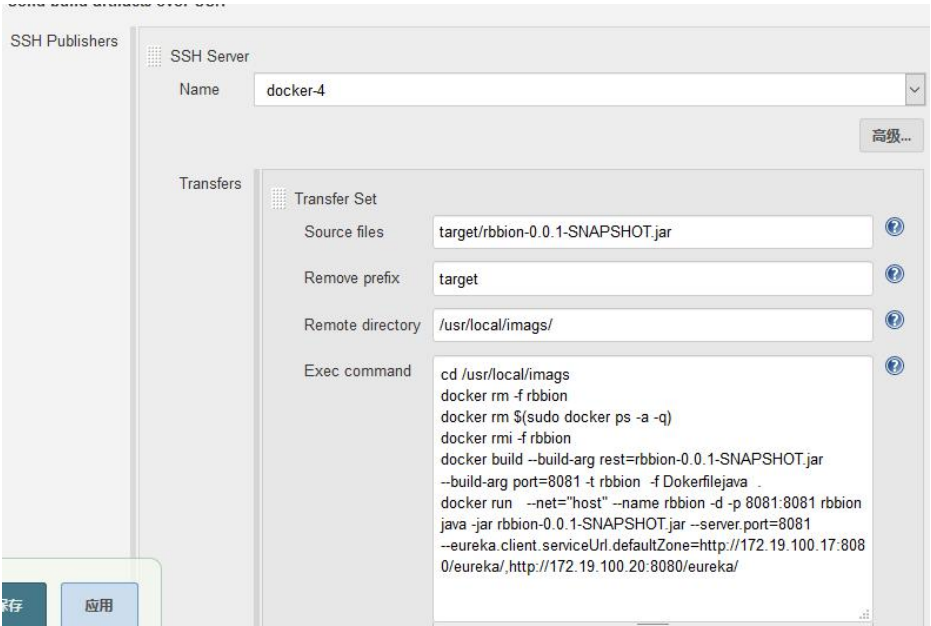
部署 LOGINWEB：在 B 服务器上部署端口为 8081。
Git 地址



执行 shell:



构建 shell:



部署成功:

LOGINWEB	n/a (1)	(1)	UP (1) - springcloud-2:loginweb:8081
----------	------------	-----	--------------------------------------

访问: <http://222.24.63.59:9067/login.html>

