

通过Jenkins与Docker构建CI/CD基础架构

前言

提到容器平台，最早接触的便是LXC（Linux Container），是2010年刚刚接触虚拟化平台的时候，当时开源解决方案是xen的天下(后来KVM才后来者居上)，且性能各方面都不弱，价值当时还不是移动互联网时代，业务量远远没有那么大，大部分公司都是物理机部署应用，用虚拟化平台的公司也是寥寥无几，可想而知，没有业务，没有场景，那就没有技术的用武之地了，所以，LXC生而伟大而用不逢时，Docker之所以能够青出于蓝而胜于蓝，取得如此大的成功的原因还是归咎于移动互联网带来的流量大爆炸，普通基于物理机，虚拟机甚至云主机(虽然弹性伸缩应该是云主机的特性，但是当时国内看起来根本没有做到的，比起AWS来，差距之大，只能意会)的业务架构已经不能满足目前的应用场景了。

关于Docker，在2013年的时候就开始接触了，当时在一家做私有云解决方案的公司里面以Openstack/Cloudstack私有云管理平台+VMWare/Xen/KVM等虚拟化平台在各大行业进行云平台的推广与实施，Docker当时对我们而言就是个实验室里面的Demo产品，并没有作过多的关注。

后来于2014年在腾讯游戏任职业务运维，中心有一个部门就已经专门研究Docker技术，我也跟着凑了一把热闹，只是当时比较火的还是云主机，而且腾讯云当时才刚刚起步，加之当时维护的更多的还是端游业务，考虑到游戏的稳定性，当时内部使用虚拟机的场景都不是非常多，更不用提Docker容器技术的大规模应用了，本着业务运维以业务稳定为第一原则，加之业务运维的职责不在于基础架构的研究，所以就没再深入了。

直到2015年，一个运维朋友问我后面什么比较火，我就随口一答：Docker以后肯定会流行的，你去看看吧，于是他就去了DaoCloud，Docker在后面的几年真的就火起来了，一时间Docker正在以星星之火可以燎原之势在整个中国掀起了一阵风，国内比较知名的DaoCloud,灵雀云等创业公司发展得非常迅猛，且资本不断进入，大公司也纷纷布局，如阿里云，腾讯云等，更不用说现在还有更多运维公司进场想分一杯羹了。

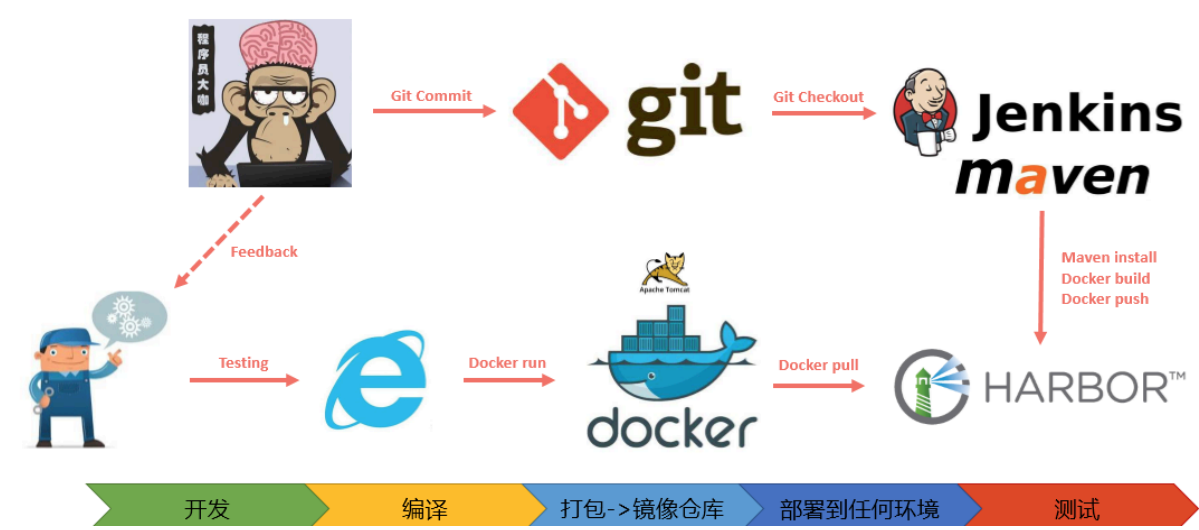
2016年，我跟另外一个朋友选择了开发自动化运维平台作为创业方向，现在想来，应该抓着Docker一起做的，由于自动化运维平台更多还是聚焦在运维层面的工作，如果需要实现持续集成/持续发布等工作支撑还需要做大量的设计，且成本不菲，算不得真正的DevOps整套解决方案，而有了Docker，这一切变得就简单多了，也变得顺理成章了许多。

当然，创业方向虽然不是容器云方向，但是在做运维平台项目的时候，我自己都会持续关注Docker发展以及客户的Docker诉求，如今Docker+K8S已经成为了一个运维的技能标配，也是更多企业选择业务架构设计的底层架构，作为一个技术人，我们不得不去正视Docker在未来的更大的发展，加之本人从2018年开始转做区块链架构设计与开发工作，接触到的Hyperledger Fabric在进行模块启动的时候也是通过Docker实现，更加坚定了将Docker技术研究彻底的决心。

由于时间有限，概念性的东西大家自己有不清楚的可以自行翻阅相关资料，本文主要将这几天学习到的东西进行整理并通过实战分享，在今后的学习与工作中，我会更多的把重心放在区块链架构与Docker的整合之上，将最好的技术整合在一起，为业务提供最好的架构设计与服务支撑。

一. 部署环境

本文描述的就是一个测试环境(操作系统: CentOS7U5 X64), 测试代码用的是JAVA的开源博客系统 solo(<https://github.com/b3log/solo>), 通过域名的方式进行角色的划分, 大家在自己的实际环境中根据实际用途进行不同主机不同角色的划分。



角色	IP地址	访问域名	部署服务
Git版本控制器	172.16.222.180	git.brucefeng.com	Git
Docker主机	172.16.222.180	docker01.brucefeng.com	Docker docker-compose
Docker仓库注册服务器	172.16.222.180	reg.brucefeng.com	Docker主机服务+Harbor
Jenkins服务器(Master)	172.16.222.180	jenkins.brucefeng.com	JDK Tomcat Jenkins
Jenkins客户端(Slave)	172.16.222.180	docker01.brucefeng.com	JDK Tomcat Maven

需要在服务器与自己用于访问域名的机器上面都进行/etc/hosts的解析配置

二. 安装Git服务器

1.概念简述

Git是一个开源的分布式版本控制系统,是Linus Torvalds(Linux之父)为了帮助管理Linux内核开发而开发的一个开放源码的版本控制软件。

2. 安装Git

(1) 安装git并创建用户

```
# yum install git -y ; useradd git ; echo git123|passwd --stdin git
```

注意: 类似安装之前的yum源配置等基础运维操作本文都不会赘述, 下文如此。

(2) 创建仓库

```
# su - git # 切换至git用户
$ mkdir solo.git ; cd solo.git
$ git --bare init #初始化仓库
```

此时可以通过git clone命令访问这个仓库了，目前没有代码数据

```
# git clone git@git.brucefeng.com:/home/git/solo.git
```

3. 将项目传至私有仓库

将solo项目从<https://github.com/b3log/solo>获取到之后提交至刚刚创建的git仓库(私有仓库)

(1) 从github.com拉取代码

```
$ git clone https://github.com/b3log/solo.git
```

(2) 添加至私有仓库

```
$ cd solo
$ git remote remove origin
$ git remote add origin git@git.brucefeng.com:/home/git/solo.git
```

(3) 提交至私有仓库

```
$ git add .
$ git commit -m "All Solo Files To Local Git Server"
$ git push origin master
```

至此，我们已经将solo项目成功传至我们创建的本地git仓库中solo.git

三.安装Docker CE

CentOS安装Docker的官方文档

```
https://docs.docker.com/install/linux/docker-ce/centos/
```

1.安装依赖组件

```
# yum install -y yum-utils \
    device-mapper-persistent-data \
    lvm2
```

2.添加docker专用yum源

```
# yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
```

3.安装Docker CE

```
# yum install docker-ce
```

自己根据需要可以再配置一个Docker加速器(国内镜像源)，Daocloud跟阿里云都有，自己决定即可。

4.启动与停止命令

```
# systemctl start docker #启动docker
# systemctl stop docker  #停止docker
```

5.安装docker-compose

Compose 是一个用户定义和运行多个容器的 Docker 应用程序。在 Compose 中你可以使用 YAML 文件来配置你的应用服务。然后，只需要一个简单的命令，就可以创建并启动你配置的所有服务。

使用 Compose 基本会有如下三步流程：

- 在 Dockfile 中定义你的应用环境，使其可以在任何地方复制。
- 在 docker-compose.yml 中定义组成应用程序的服务，以便它们可以在隔离的环境中一起运行。
- 运行dcoker-compose up，Compose 将启动并运行整个应用程序。

(1)直接下载

```
# curl -L
https://github.com/docker/compose/releases/download/1.18.0/docker-compose-
`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

(2) 添加可执行权限

```
# chmod +x /usr/local/bin/docker-compose
# ln -s /usr/local/bin/docker-compose /usr/bin/
```

(3) 查看版本

```
# docker-compose version
```

返回结果

```
docker-compose version 1.18.0, build 8dd22a9
docker-py version: 2.6.1
CPython version: 2.7.13
OpenSSL version: OpenSSL 1.0.1t  3 May 201
```

或者通过pip工具进行安装

```
pip install -U -i https://pypi.tuna.tsinghua.edu.cn/simple docker-compose
```

5. 拉取所需基础镜像

```
# docker pull centos:7
```

6. 创建Tomcat镜像

(1) 创建Dockerfile文件 Dockerfile-tomcat-85

```
FROM centos:7
MAINTAINER git.brucefeng.com

ENV TOMCAT_VERSION=8.5.32
ENV TOMCAT_MIRROR=http://mirrors.shu.edu.cn/apache/tomcat/tomcat-8/
ENV JDK_MIRROR=http://172.16.222.182:8888/jdk-8u181-linux-x64.tar.gz
ENV JAVA_HOME /usr/local/jdk

RUN yum install wget curl unzip iproute net-tools -y && \
    yum clean all && \
    rm -rf /var/cache/yum/*

RUN wget ${TOMCAT_MIRROR}/v${TOMCAT_VERSION}/bin/apache-tomcat-
${TOMCAT_VERSION}.tar.gz && \
    tar xzf apache-tomcat-${TOMCAT_VERSION}.tar.gz && \
    mv apache-tomcat-${TOMCAT_VERSION} /usr/local/tomcat && \
    rm -rf apache-tomcat-${TOMCAT_VERSION}.tar.gz
/usr/local/tomcat/webapps/* && \
    mkdir /usr/local/tomcat/webapps/ROOT && \
    echo "${TOMCAT_VERSION} INSTALL DONE" >
/usr/local/tomcat/webapps/ROOT/status.html && \
    sed -i '1a JAVA_OPTS="-Djava.security.egd=file:/dev/urandom"'
/usr/local/tomcat/bin/catalina.sh && \
    ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

RUN wget ${JDK_MIRROR} && \
    tar xzf jdk-8u181-linux-x64.tar.gz && \
    mv jdk1.8.0_181 /usr/local/jdk && \
    rm -rf jdk-8u181-linux-x64.tar.gz
```

```
ENV PATH $PATH:/usr/local/tomcat/bin
ENV PATH $PATH:/usr/local/jdk/bin
WORKDIR /usr/local/tomcat
EXPOSE 8080
CMD ["catalina.sh","run"]
```

(2) 构建docker镜像

镜像名: tomcat-85

```
docker build -t tomcat-85 -f Dockfile-tomcat-85 .
```

(3) 创建容器进行测试

```
docker container run -d --name=tomcat-1 -p 8888:8080 tomcat-85
```

这一步务必做好，在后面Jenkins的Pipeline中会用到

四.配置Harbor服务器

以上我们通过docker pull命令都是从公网拉取的镜像文件，为了方便企业内部开发需要，我们有必要搭建一套Docker的私有仓库管理常用镜像。

VMWare Harbor是我们常用的也是比较优秀的Docker私有仓库开源解决方案，项目地址为：<https://github.com/vmware/harbor/>，关于项目介绍以及特性相关内容可自行查阅。

Harbor的安装有多种方式，在线安装|离线安装|OVA安装，为了避免由网络稳定性带来的问题，我们采用离线安装的方式进行Harbor的安装（所需的Docker与Docker-Compose上文已经安装完毕，此处不再写安装配置）

1.下载离线安装包

```
https://github.com/vmware/harbor/releases
```

选择 Harbor offline installer

```
https://storage.googleapis.com/harbor-releases/release-1.5.0/harbor-offline-installer-v1.5.2.tgz
```

2. 自签TLS证书

Harbor可以通过两种方式进行部署，HTTP与HTTPS，本文直接通过HTTPS的方式进行部署，大家可以参考官方文档的配置：https://github.com/vmware/harbor/blob/master/docs/configure_https.md

(1) 创建CA证书

```
# mkdir /tmp/ssl ; cd /tmp/ssl #创建一个ssl目录,存放证书文件
# openssl req -newkey rsa:4096 -nodes -sha256 -keyout ca.key -x509 -days
365 -out ca.crt
```

设置这两处即可

```
Country Name (2 letter code) [XX]:CN
Common Name (eg, your name or your server's hostname) []:brucefeng.com
```

生成文件

```
ca.crt  ca.key
```

(2) 生成证书签名

```
# openssl req -newkey rsa:4096 -nodes -sha256 \
-keyout brucefeng.com.key \
-out brucefeng.com.csr
```

注意:

```
Country Name (2 letter code) [XX]:CN
Common Name (eg, your name or your server's hostname) []:brucefeng.com
```

生成文件

```
brucefeng.com.csr  brucefeng.com.key
```

(3) 生成注册证书

```
# openssl x509 -req -days 365 -in brucefeng.com.csr -CA ca.crt -CAkey
ca.key -CAcreateserial -out brucefeng.com.crt
```

返回结果

```
Signature ok
subject=/C=CN/L=Default City/O=Default Company Ltd
Getting CA Private Key
```

全部的生成文件

```
brucefeng.com.crt  brucefeng.com.csr  brucefeng.com.key  ca.crt  ca.key  
ca.srl
```

3. Harbor安装与配置

(1) 解压文件

```
# tar zxvf harbor-offline-installer-v1.5.2.tgz  
# mv /tmp/ssl harbor/ #将ssl目录拷贝进harbor目录下
```

(2) 修改配置

```
# cd harbor  
# vim harbor.cfg
```

需要修改的参数如下

```
hostname = reg.brucefeng.com  
ui_url_protocol = https  
ssl_cert = ./ssl/reg.brucefeng.com.crt  
ssl_cert_key = ./ssl/reg.brucefeng.com.key  
harbor_admin_password = 123456
```

注意: 本人发现harbor的安装脚本有BUG,hostname必须是直接修改,不能注释后修改,否则报错!

如下为**错误配置**

```
# hostname = reg.mydomain.com  
hostname = reg.brucefeng.com
```

(3) 生成用于安装的配置文件

```
#./prepare
```

(4) 安装并启动Harbor

```
#./install.sh
```

(5) 查看运行状态


```
# docker-compose ps
```

返回结果

Name	Command	State
Ports		

harbor-adminserver	/harbor/start.sh	Up
harbor-db	/usr/local/bin/docker-entr ...	Up 3306/tcp
harbor-jobservice	/harbor/start.sh	Up
harbor-log	/bin/sh -c /usr/local/bin/ ...	Up
127.0.0.1:1514->10514/tcp		
harbor-ui	/harbor/start.sh	Up
nginx	nginx -g daemon off;	Up 0.0.0.0:443-
>443/tcp, 0.0.0.0:4443->4443/tcp, 0.0.0.0:80->80/tcp		
redis	docker-entrypoint.sh redis ...	Up 6379/tcp
registry	/entrypoint.sh serve /etc/ ...	Up 5000/tcp

4. 测试网页登录

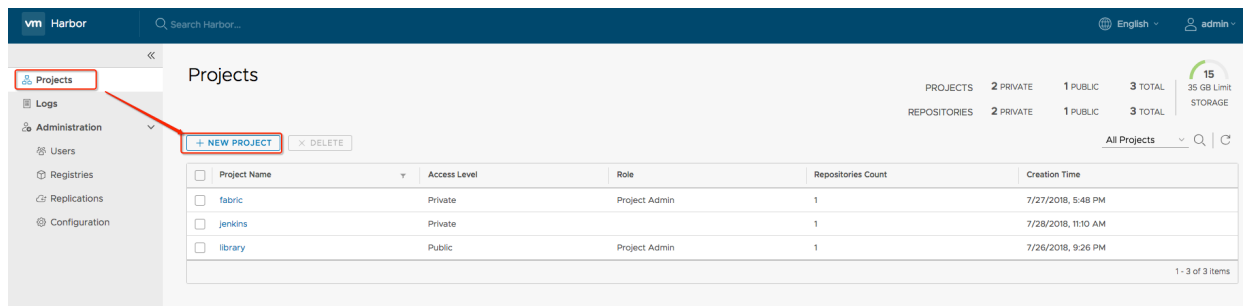
访问链接: <https://reg.brucefeng.com/harbor/projects>

用户名: admin

密码: 123456

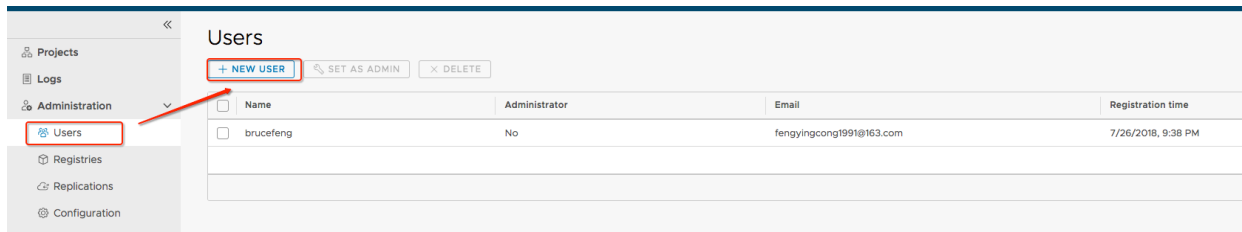
密码是我们在配置文件harbor.cfg中配置的harbor_admin_password参数值

(1) 创建项目



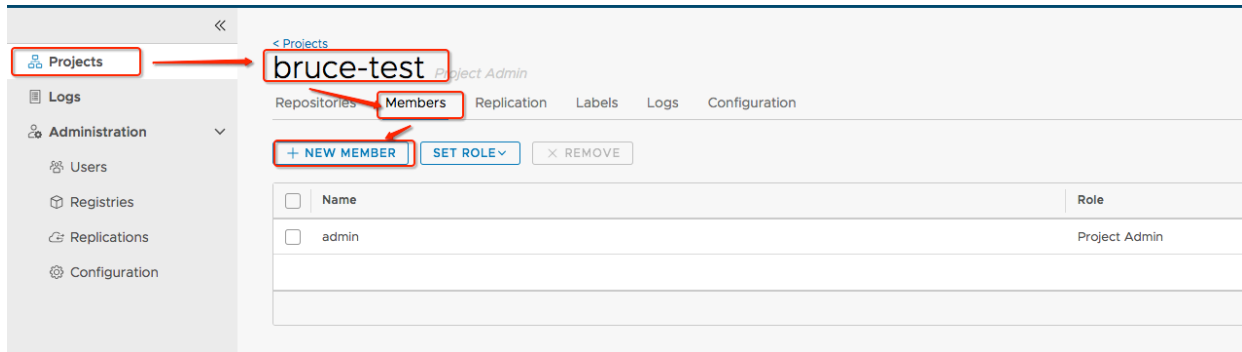
创建新项目名称: bruce-test

(2) 创建用户



创建新用户为:bruce-test-user

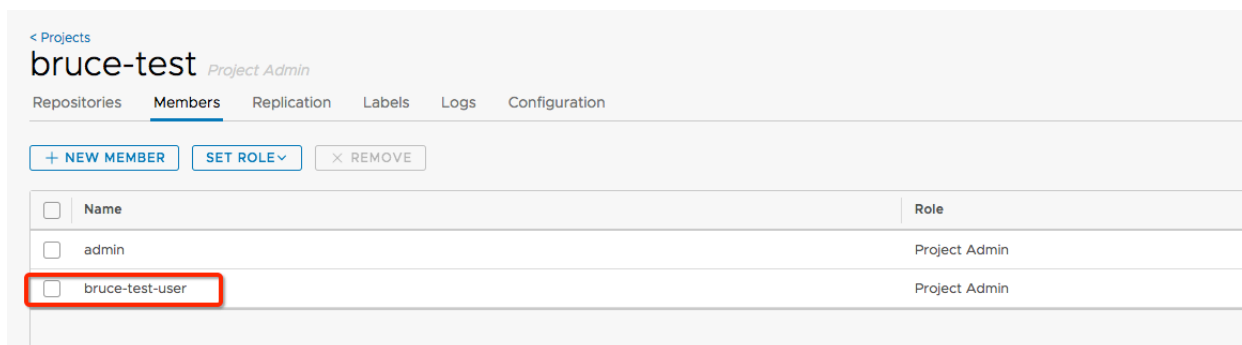
(3) 关联项目与用户



填写刚刚创建的新用户:

Name:bruce-test-user

Role角色:Project Admin



5.为Docker主机添加信任

(1) 拷贝证书文件

```
# mkdir /etc/docker/certs.d/reg.brucefeng.com/  
# cp /tmp/ssl/reg.brucefeng.com.crt /etc/docker/certs.d/reg.brucefeng.com/
```

(2) 测试登录

此处用的测试账户仍未之前创建的brucefeng，项目按照之前创建的几个项目进行操作

```
$ docker login -u brucefeng -p password(自定义的密码) reg.brucefeng.com
```

返回结果

Login Succeeded

认证成功，可以通过命令行进行数据交互。

6. 上传下载镜像

以Harbor上的fabric项目为例进行镜像的上传与下载操作

(1) 将本地镜像打标签

命令格式

```
docker tag SOURCE_IMAGE[:TAG] reg.brucefeng.com/fabric/IMAGE[:TAG]
```

本文以centos7镜像为例测试

```
# docker tag docker.io/centos:7 reg.brucefeng.com/fabric/centos:v1.0
```

查看当前的centos7镜像

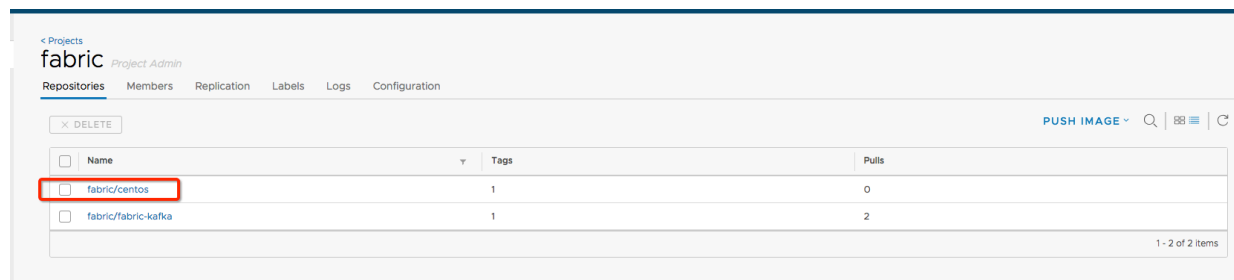
```
# docker image ls |grep centos
```

返回结果

```
docker.io/centos          7
49f7960eb7e4             7 weeks ago             200 MB
reg.brucefeng.com/fabric/centos  v1.0
49f7960eb7e4             7 weeks ago             200 MB
```

(2) 推送(上传)镜像

```
# docker push reg.brucefeng.com/fabric/centos:v1.0
```



< Projects < Repositories
fabric/centos

Info Images

SCAN COPY DIGEST +ADD LABELS X DELETE

<input type="checkbox"/>	Tag	Size	Pull Command	Author	Creation Time	Docker Version	Labels
<input type="checkbox"/>	v1.0	71.24MB			6/5/2018, 6:19 AM	17.06.2-ce	

1 - 1 of 1 items

(3) 拉取(下载)镜像

- 删除本地镜像

```
# docker image rm reg.brucefeng.com/fabric/centos:v1.0
```

- 从Harbor上拉取镜像

```
# docker pull reg.brucefeng.com/fabric/centos:v1.0
```

五.Jenkins Master/Slave配置

Jenkins下载地址:<https://jenkins.io/download/>

本文通过Jenkins.war包启动Jenkins服务，而Slave需要通过maven工具进行代码构建，所以此处需要配置JDK|Tomcat|Maven环境

1. 依赖环境安装

```
# tar zxf jdk-8u181-linux-x64.tar.gz
# tar zxf apache-tomcat-8.5.32.tar.gz
# tar zxf apache-maven-3.5.4-bin.tar.gz
# mv jdk1.8.0_181 /usr/local/jdk
# mv apache-tomcat-8.5.32 /usr/local/tomcat
# mv apache-maven-3.5.4 /usr/local/maven3.5
```

2.环境变量配置

```
# vim /etc/profile
```

添加如下信息

```
JAVA_HOME=/usr/local/jdk
PATH=$PATH:$JAVA_HOME/bin
export JAVA_HOME PATH
```

```
# source /etc/profile
# java -version    #检查环境变量是否配置成功，查看java版本
```

返回结果

```
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

3. 启动Jenkins

(1) 部署Jenkins包

```
# rm -rf /usr/local/tomcat/webapps/ROOT
# unzip jenkins.war -d /usr/local/tomcat/webapps/ROOT
# cd /usr/local/tomcat/bin ; ./startup.sh
```

(2) 获取Jenkins初始化密码

```
# tailf ../logs/catalina.out    #查看日志信息
# cat /root/.jenkins/secrets/initialAdminPassword //查看Jenkins初始化密码
```

4. 初始化 Jenkins

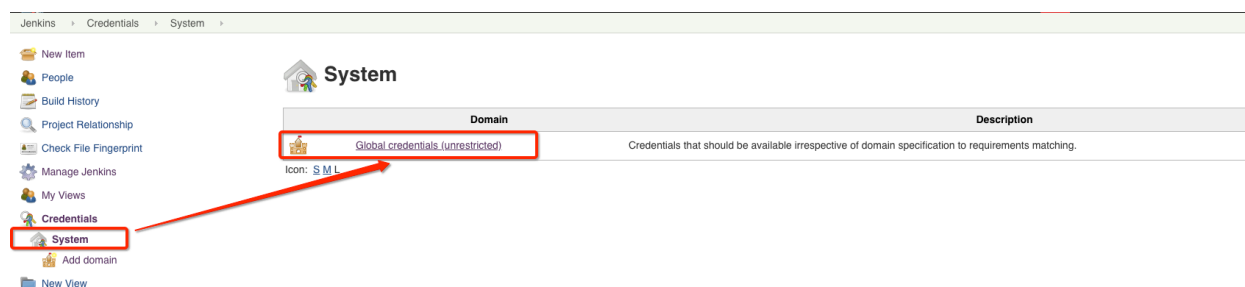
登录地址: <http://jenkins.brucefeng.com:8080>

按照默认的方式进行安装即可，此处不深入。

5. Jenkins Slave配置

Master/Slave相当于Server和agent的概念。Master提供web接口让用户来管理job和slave，job可以运行在master本机或者被分配到slave上运行。一个master可以关联多个slave用来为不同的job或相同的job的不同配置来服务。

(1) 创建认证方法



点击**Add credentials** 创建新的凭据

The screenshot shows the 'Add Credentials' dialog in Jenkins. The 'Kind' is set to 'SSH Username with private key'. The 'Scope' is 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' is 'root'. The 'Private Key' is set to 'Enter directly', and the 'Key' field contains the text '输入ssh的密钥信息'. The 'Passphrase' field is empty. The 'ID' field is empty. The 'Description' is 'Docker-SSH-KEY'. There is an 'OK' button at the bottom.

可以选择多种认证方式，一般我们可以采用ssh的用户名跟密码或者密钥登录，此处我选择密钥登录。

(2) 新建Slave节点

Jenkins-Manage Jenkins-Manage Node-New Node

The screenshot shows the 'New Node' configuration page in Jenkins. The 'Name' is 'docker01.brucefeng.com'. The 'Description' is 'Permanent Slave'. The '# of executors' is '1'. The 'Remote root directory' is '/var/jenkins_home'. The 'Labels' is 'docker01.brucefeng.com'. The 'Usage' is 'Only build jobs with label expressions matching this node'. The 'Launch method' is 'Launch slave agents via SSH'. The 'Host' is 'docker01.brucefeng.com'. The 'Credentials' is 'root (Docker-SSH-KEY)'. The 'Host Key Verification Strategy' is 'Known hosts file Verification Strategy'. The 'Port' is '22'. The 'JavaPath' is '/usr/local/jdk/bin/java'. The 'JVM Options' is empty. The 'Prefix Start Slave Command' is empty. The 'Suffix Start Slave Command' is empty. The 'Connection Timeout in Seconds' is empty. The 'Maximum Number of Retries' is '0'. The 'Seconds To Wait Between Retries' is '0'. The 'Availability' is 'Keep this agent online as much as possible'. There is a 'Save' button at the bottom.

注意配置项

- Labels: docker01.brucefeng.com #用于标识Slave的标签
- Remote root directory: /var/jenkins_home #用于指定通过jenkins构建的项目路径
- JavaPath: /usr/local/jdk/bin/java #并非JAVA_HOME的配置项

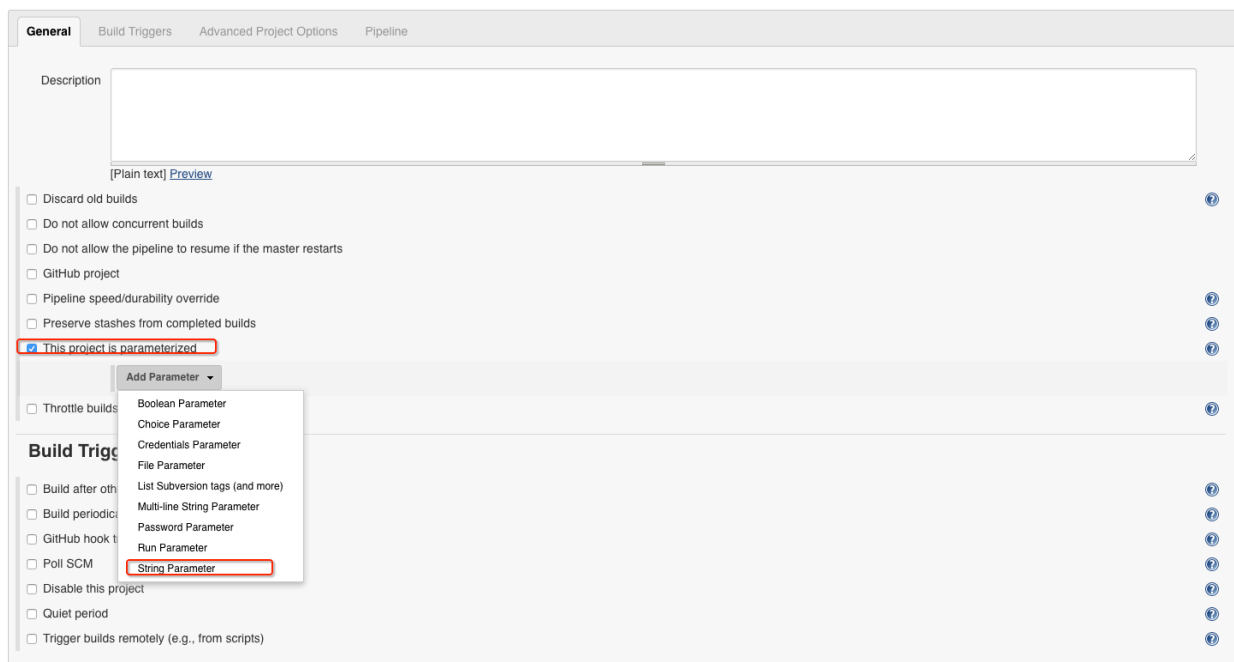
6.配置Docker主机与Git服务器免密登录

通过ssh-keygen与ssh-copy-id密码实现

六.通过Jenkins创建项目

1.新建任务

New Item 类型选择Pipeline,进行参数化设置



选择字符串参数



定义参数名为Tag，用于传入项目代码版本号

2.定义Pipeline

关于Pipeline就是一条任务流水线，便于分阶段执行与排错



```
1 node ("docker01.brucefeng.com") { //指定Slave的标签
2   // 拉取代码，$Tag引用用户交互输入的tag
3   stage("Git Checkout"){
4     checkout([$class: 'GitSCM', branches: [[name: '$Tag']], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemote
5   ]
6   }
7   // 代码编译
8   stage("Maven Build")
9   {
10    sh """
11      export JAVA_HOME=/usr/local/jdk
12      .../usr/local/maven3.5/bin/mvn clean package -Dmaven.test.skip=true
13    """
14  }
15 }
```

脚本内容如下

```
node ("docker01.brucefeng.com") { //指定Slave的标签
```

```

// 拉取代码, $Tag引用用户交互输入的tag
stage('Git Checkout'){
    checkout([$class: 'GitSCM', branches: [[name: '$Tag']],
doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [],
userRemoteConfigs: [[url: 'git@git.brucefeng.com:/home/git/solo.git']]])
}
// 代码编译
stage('Mavin Build')
{
sh '''
    export JAVA_HOME=/usr/local/jdk
    /usr/local/maven3.5/bin/mvn clean package -Dmaven.test.skip=true
'''

}

// 项目打包到镜像并推送至镜像仓库
stage('Build and Push Image'){
    sh '''
REPOSIROTY=reg.brucefeng.com/jenkins/solo:$Tag
cat > Dockerfile << EOF
FROM reg.brucefeng.com/library/tomcat-85:latest
RUN rm -rf /usr/local/tomcat/webapps/ROOT
COPY target/*.war /usr/local/tomcat/webapps/ROOT.war
CMD ["catalina.sh","run"]
EOF
docker build -t $REPOSIROTY .
docker login -u brucefeng -p Cpic@1234 reg.brucefeng.com
docker push $REPOSIROTY
'''
}

//根据$Tag作为镜像版本号
stage('Deploy to Docker'){
sh '''
REPOSIROTY=reg.brucefeng.com/jenkins/solo:$Tag
docker rm -f blog-solo|true
docker image rm $REPOSIROTY |true
docker login -u brucefeng -p Cpic@1234 reg.brucefeng.com
docker container run -d --name blog-solo -p 88:8080 $REPOSIROTY
'''
}
}

```

3.模拟更新代码

```
# vim solo/src/main/resources/latke.properties
```

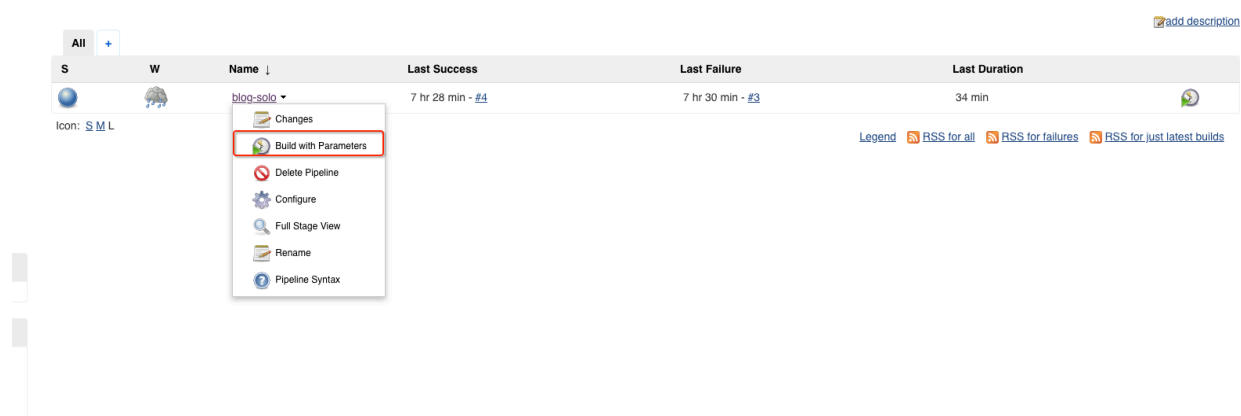

修改内容

```
# Browser visit domain name
serverHost=docker01.brucefeng.com
# Browser visit port, 80 as usual, THIS IS NOT SERVER LISTEN PORT!
serverPort=88
```

4.提交代码

```
# git add .
# git commit -m "modify latke.properties"
# git tag 1.0.1
# git push origin 1.0.1
```

5. 发布测试



The screenshot shows the Jenkins pipeline interface. At the top right, there is a link to "add description". Below it is a table with columns: S, W, Name, Last Success, Last Failure, and Last Duration. The table contains one row for the pipeline named "blog-solo". The "Last Success" column shows "7 hr 28 min - #4" and the "Last Failure" column shows "7 hr 30 min - #3". The "Last Duration" column shows "34 min". To the left of the table, there is a section for "Icon: S M L". A context menu is open over the "blog-solo" pipeline, showing options: Changes, Build with Parameters (highlighted with a red box), Delete Pipeline, Configure, Full Stage View, Rename, and Pipeline Syntax. To the right of the table, there is a "Legend" section with links for "RSS for all", "RSS for failures", and "RSS for just latest builds".

选择Build with Patameters

Pipeline blog-solo

This build requires parameters:

Tag	1.0.1
代码版本号	

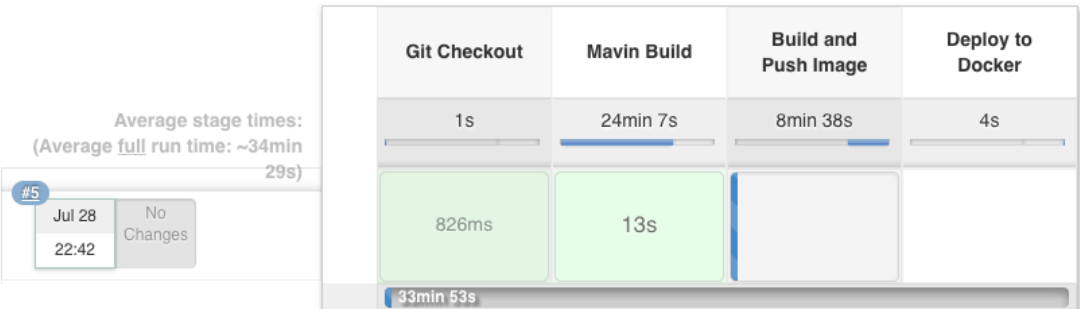
Build

填写Tag:1.0.1

Pipeline blog-solo

 [Recent Changes](#)

Stage View



6.登录访问

发布成功返回日志

```
Digest:
sha256:dc45052ad607de3757d330c389071bcbd985dd9981aba1c4f8196d05265c62a8
Status: Downloaded newer image for reg.brucefeng.com/jenkins/solo:1.0.0
c639e3bd71ca3935486c9f49bb8ba414031dc85927664f2c0d5d8d7b5eef4066
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

通过浏览器访问<http://docker01.brucefeng.com:88>



Welcome to Solo

Email:

Username:

Password:

Input password again:

Next