Jenkins 流水线部署应用集成 k8s 具体配置

目录

- 1 背景
- 2 安装 Jenkins
  - 2.1 配置要求
  - 2.2 安装 Jenkins
  - 2.3 配置 Jenkins
- 3 实现 Java 应用持续集成和持续发布
  - 3.1 Fork 和 clone GitHub 示例仓库
  - 3.2 在 Jenkins 中创建流水线
  - 3.3 安装 k8s 插件
  - 3.4 对接 k8s 集群
    - 3.4.1 申请 k8s 凭据
    - 3.4.2 配置 k8s 集群的对接
  - 3.5 k8s pod template 配置
  - 3.6 Jenkins pipeline 说明
  - 3.7 Jenkins pipeline 入门
    - 3.7.1 创建并运行 pipeline
    - 3.7.2 在 Slave 中运行 Pipeline
  - 3.8 完整 pipeline 示例
- 4 参考资料

1 背景

使用 Jenkins 和 Kubernetes 完成持续集成和持续部署的功能 。

2 安装 Jenkins

2.1 配置要求

macOS，Linux 或 Windows 操作系统的机器，配置要求如下：

- 最小 256MB 内存, 推荐 512MB 以上。
- 10GB 硬盘空间， 用于安装 Jenkins，Docker 镜像和容器。

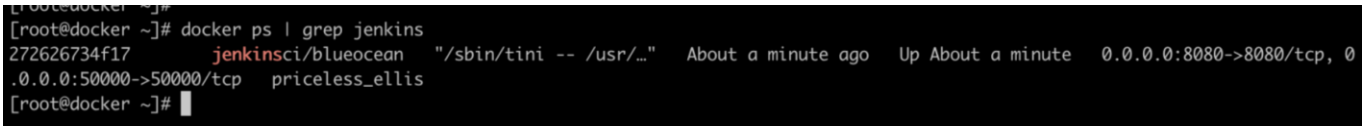
2.2 安装 Jenkins

docker 启动:

docker run -d -u root -v /data/demo/jenkins-home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -v "$HOME":/home -p 8080:8080 -p 50000:50000 jenkinsci/blueocean

- -d：d 指 daemon，后台启动
- -u：指定运行用户
- -v：-v /data/demo/jenkins-home:/var/jenkins_home 表示将容器里的 /var/jenkins_home 目录映射到宿主机的 /data/demo/jenkins-home 目录。
- -p：-p 8080:8080 对应 hostPort:containerPort

检查 jenkins 服务状态：
docker ps | grep jenkins



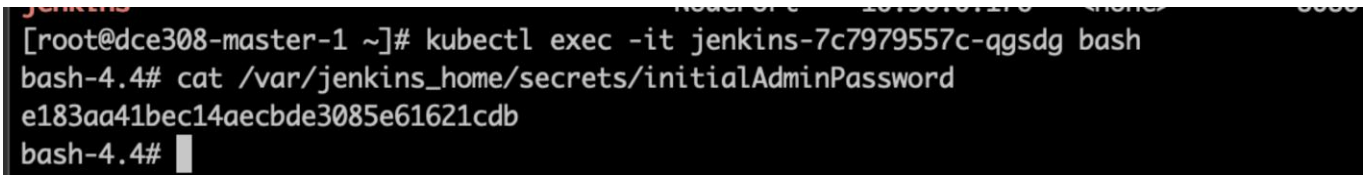
启动成功后，就可以通过 ip:8080 在浏览器上访问了。ip 为 docker 所在机器的 ip。

2.3 配置 Jenkins

Jenkins 启动成功后，需要执行一些快速的 "一次性" 步骤。当你第一次访问一个新的 Jenkins 实例时，要求你使用自动生成的密码对其进行解锁。密码为 jenkins 所在容器的 /var/jenkins_home/secrets/initialAdminPassword 的内容：

docker 方式：

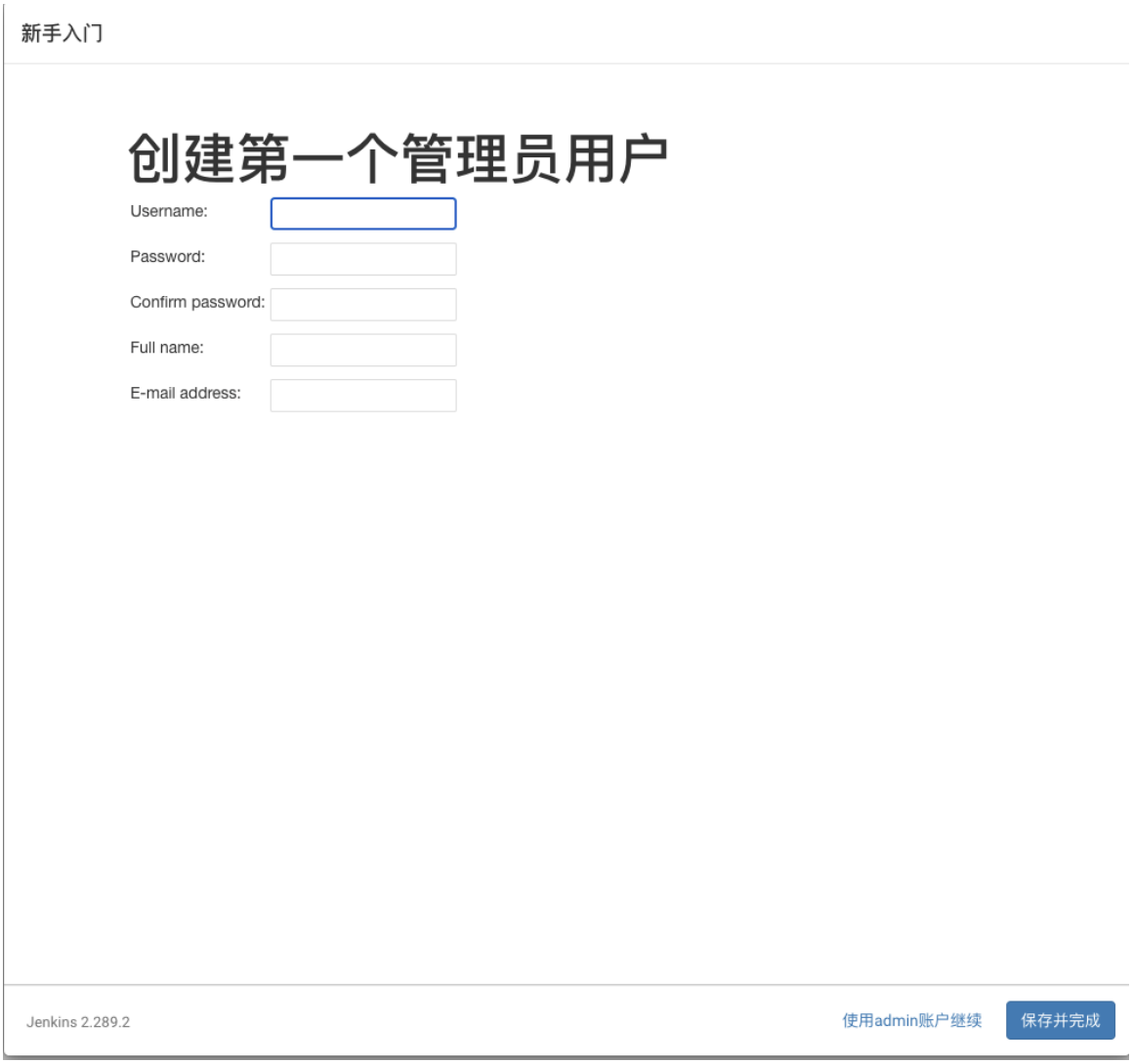
k8s 方式：



在 Unlock Jenkins 页面，粘贴该密码到 Administrator password 字段并点击 Continue。解锁 Jenkins 后，插件安装页面出现，点击 Install suggested plugins 即可。这个过程会耗时一段时间。


入门
解锁 Jenkins
为了确保管理员安全地安装 Jenkins，密码已写入到日志中（不知道在哪里？）该文件在服务器上：
/var/jenkins_home/secrets/initialAdminPassword
请从本地复制密码并粘贴到下面。
管理员密码
继续

创建第一个管理员用户，可以填写，也可以跳过。



这边我选择了 使用 admin 账户继续，点击保存并完成，就可以使用 Jenkins 了。

新手入门

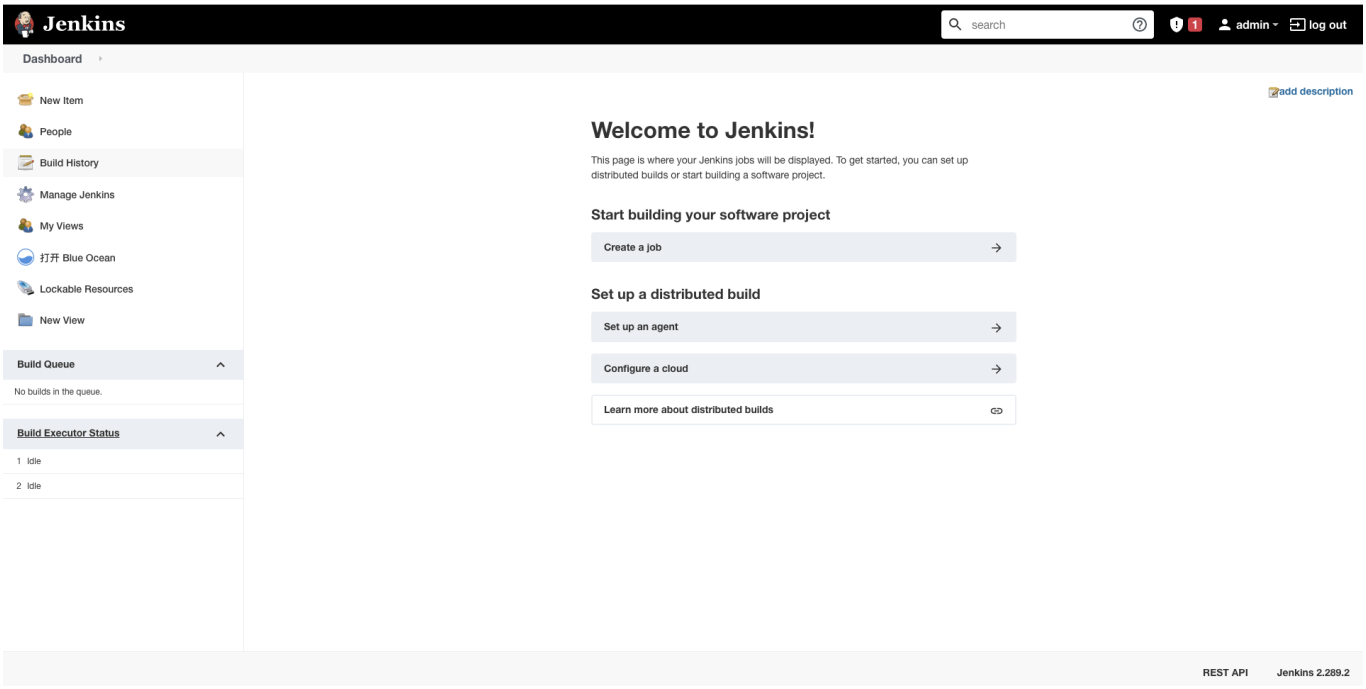
实例配置

Jenkins URL: http://172.29.4.36:8080/

Jenkins URL 用于给各种Jenkins资源提供绝对路径链接的根地址。 这意味着对于很多Jenkins特色是需要正确设置的，例如：邮件通知、PR状态更新以及提供给构建步骤的`BUILD_URL`环境变量。

推荐的默认值显示在**尚未保存**，如果可能的话这是根据当前请求生成的。 最佳实践是要设置这个值，用户可能会需要用到。这将会避免在分享或者查看链接时的困惑。

现在不要 保存并完成



3 实现 Java 应用持续集成和持续发布

3.1 Fork 和 clone GitHub 示例仓库

1. 登录 github 账号：https://github.com/

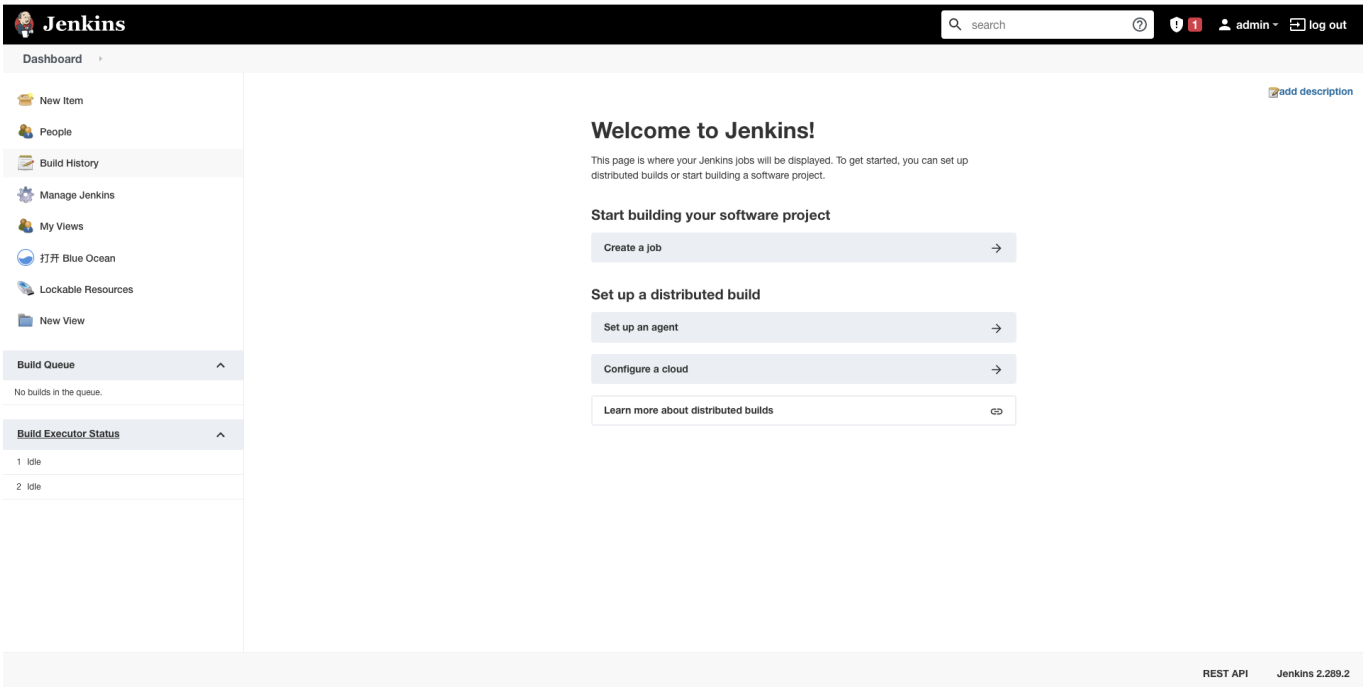
2. 上传 代码到 Github 代码仓库中 fork 到你的 github 仓库中，例 prometheus-test-demo，地址为 https://github.com/0820sdd/prometheus-test-demo/

3. 将你的 GitHub 账户中的 simple-java-maven-app 仓库 clone 到你的本地机器：
a. 打开一个终端/命令提示符，并且 cd 进入正确的目录路径:
macOS 系统路径为 /Users/<your-username>/Documents/GitHub/ Linux 系统路径为 /home/<your-username>/GitHub/ Windows 系统路径为 C:\Users\<your-username>\Documents\GitHub\（推荐使用 Git bash 命令行，而不是通常的 Microsoft 命令提示符）
b. 运行以下命令完成仓库的 clone：
git clone https://github.com/YOUR-GITHUB-ACCOUNT-NAME/simple-java-maven-app

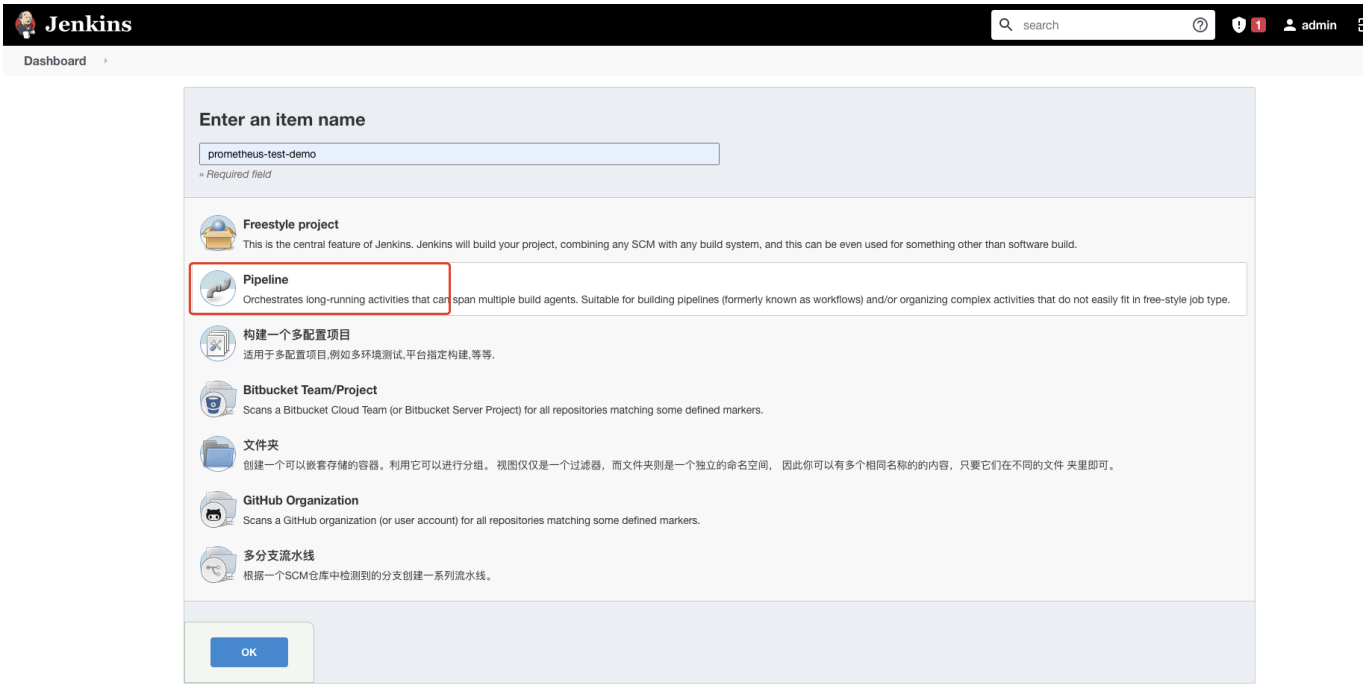
其中 YOUR-GITHUB-ACCOUNT-NAME 是你的 Github 账户的名称。

3.2 在 Jenkins 中创建流水线

1. 登录 Jenkins，点击页面 创建一个新任务。若是你无法看见以上内容，点击左上方的新建item。

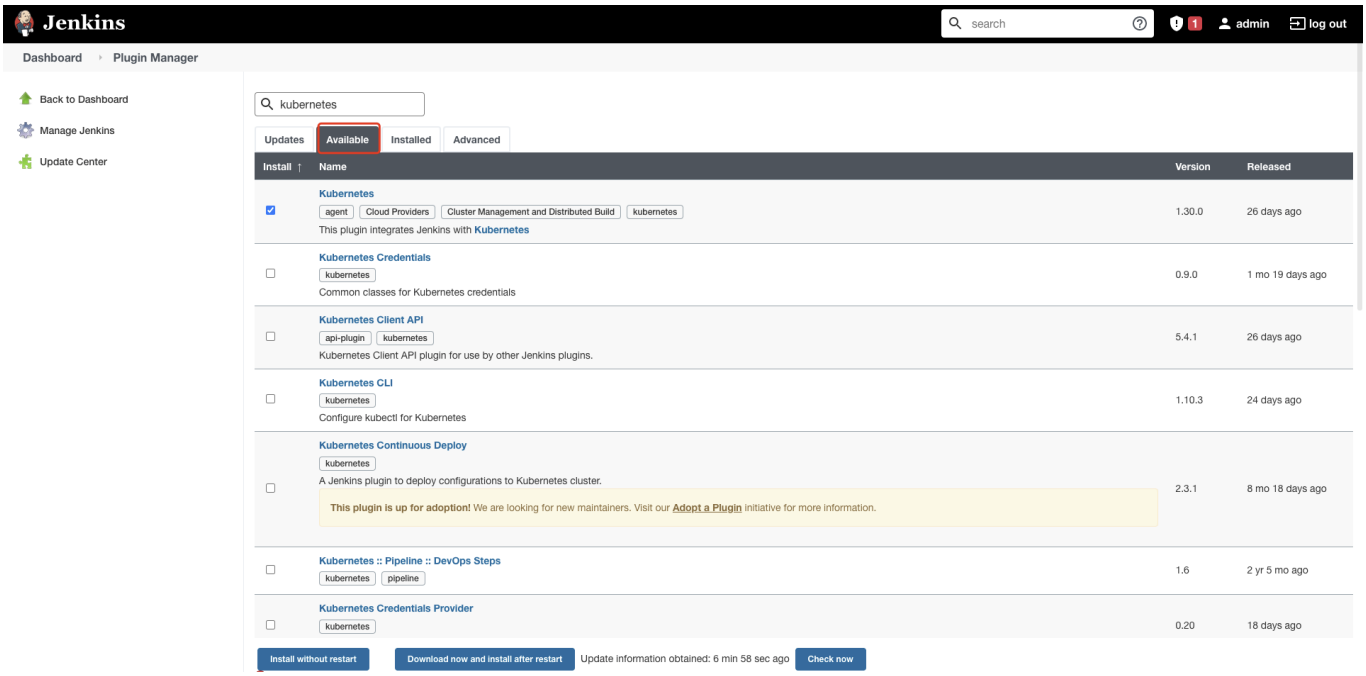


2. 为新的流水线项目指定名称（例如 prometheus-test-demo），选择 Pipeline，点击 OK。



3.3 安装 k8s 插件

登录 Jenkins，Manage Jenkins→ Manage Plugins → 搜索 kubernetes，选择 Kubernetes，点击 install，安装完成后重启 Jenkins 。

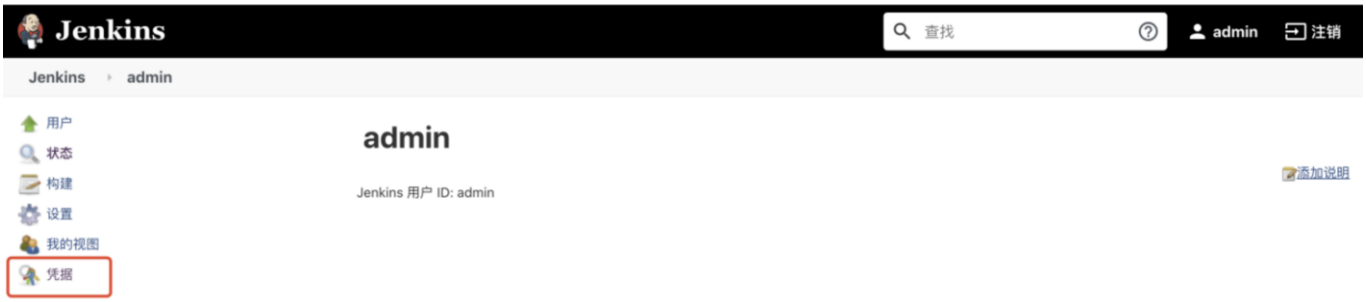


3.4 对接 k8s 集群

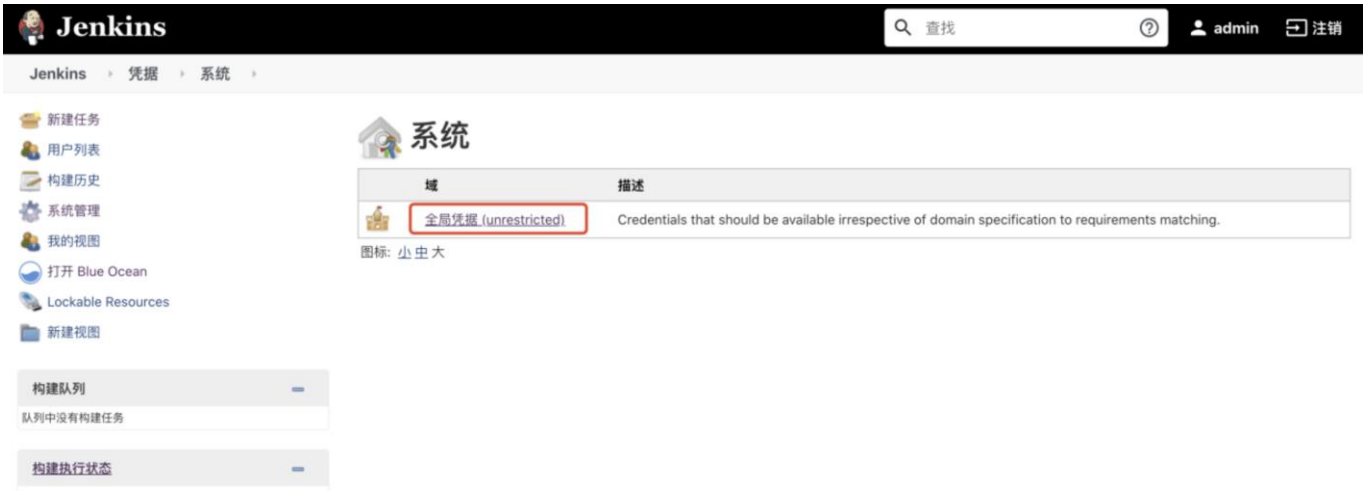
3.4.1 申请 k8s 凭据

因为 Jenkins 服务器在 kubernetes 集群之外，所以我们准备以下文件才能从外面连接到 kubernetes 集群。

登录 Jenkins，点击右上角「用户」 → 左下角「凭据」：



然后点击 Jenkins，选择 全局凭据（Unrestricted）

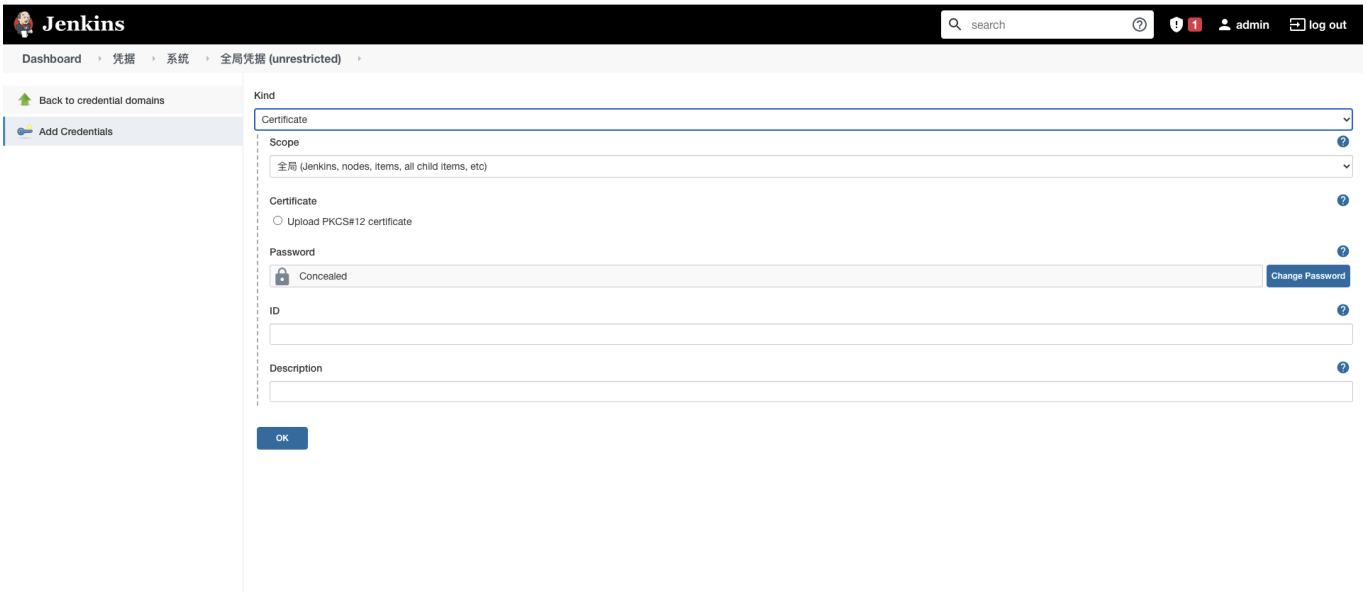


添加凭据，类型选择 Certificate

在 k8s 集群生成 cert.pfx，点击 Upload PKCS#12certificate，选择下面命令生成的 cert.pfx 文件，并输入生成 cert.pfx 文件时输入的密码。

```
# cat /root/.kube/config
apiVersion: v1
kind: Config
clusters:
  - cluster:
      certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0t...EUtLS0tLQo=
      server: https://127.0.0.1:16443
    name: kubernetes
contexts:
  - context:
      cluster: kubernetes
      user: k8s
```

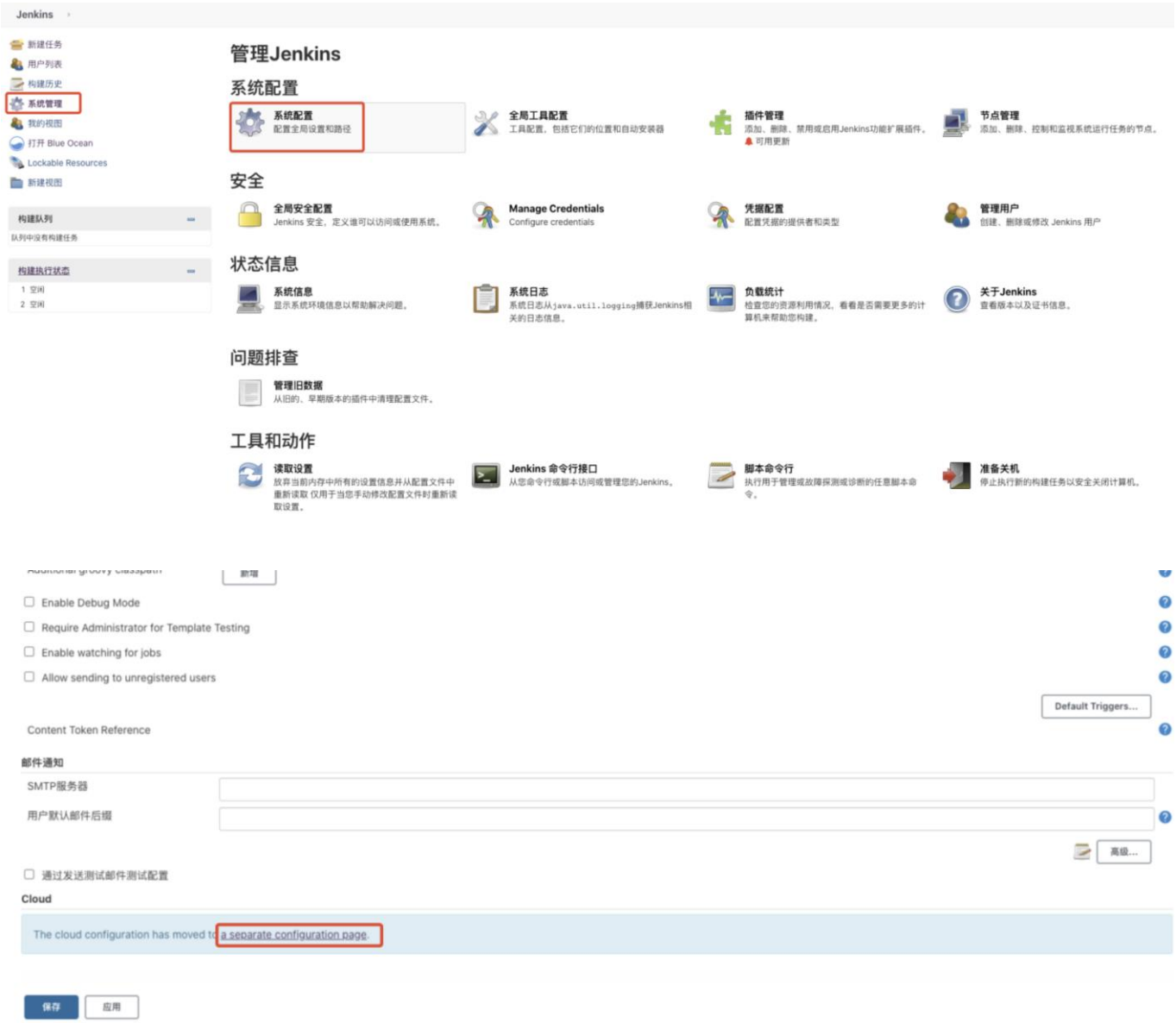
```
    name: kubelet-to-kubernetes
current-context: kubelet-to-kubernetes
users:
  - name: k8s
    user:
      client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJ...kQgQ0VSVElGSUNBVEUtLS0tLQo=
      client-key-data:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLR...S0tRU5EIFJTQSBQUklWQVRFIEtFWS0tLS0tCg==

# 下面将 certificate-authority-data client-certificate-data client-key-data 3 个证书对应的内容拷贝进行解压得到 ca.crt client.crt client.key 文件
# echo LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0t...EUtLS0tLQo= | base64 -d > ca.crt
# echo LS0tLS1CRUdJTiBDRVJUSUZJ...kQgQ0VSVElGSUNBVEUtLS0tLQo= | base64 -d > client.crt
# echo LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLR...S0tRU5EIFJTQSBQUklWQVRFIEtFWS0tLS0tCg== | base64 -d > client.key
# 然后根据如上内容生成客户端认账的 证书
# openssl pkcs12 -export -out cert.pfx -inkey client.key -in client.crt -certfile ca.crt
# 注意导出时输入密码，填入下面 Password 中
```



ID 可以不填，Describe 填写 k8s，做下备注。填写完毕，点击确定。

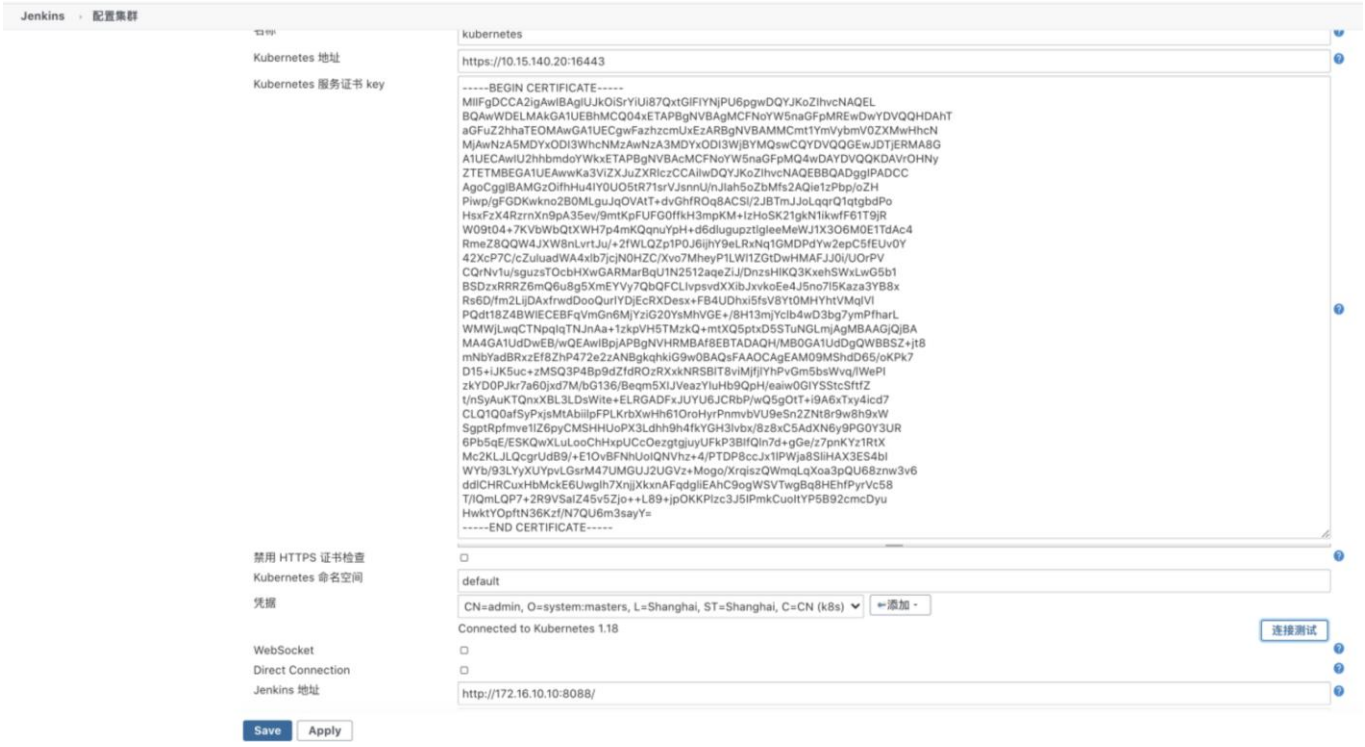
3.4.2 配置 k8s 集群的对接

登录 jenkins，点击 系统管理 → 系统配置 → 滑动到页面最下面



点击 a separate configuration page：

- Kubernetes 地址：kubernetes 服务地址，也就是 apiserver 的地址，一般是 master 节点 NodeIP+6443 端口；
- Kubernetes 服务证书 key：kube-ca.crt 文件的内容。
- 凭据：刚才创建的 certificate 凭据。
- Jenkins 地址：Agent 连接 Jenkins Master 的地址。



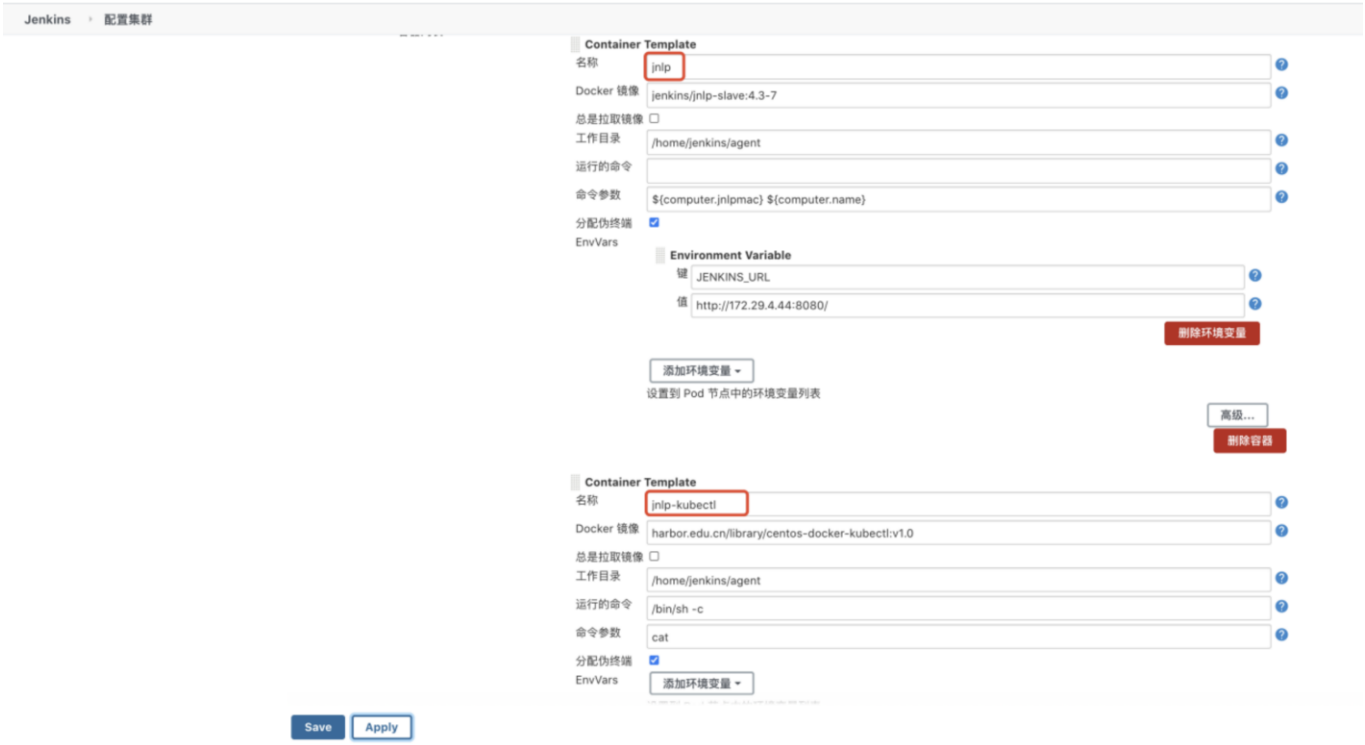
其他都使用默认配置，点击连接测试，连接测试成功，点击 Save 存储。

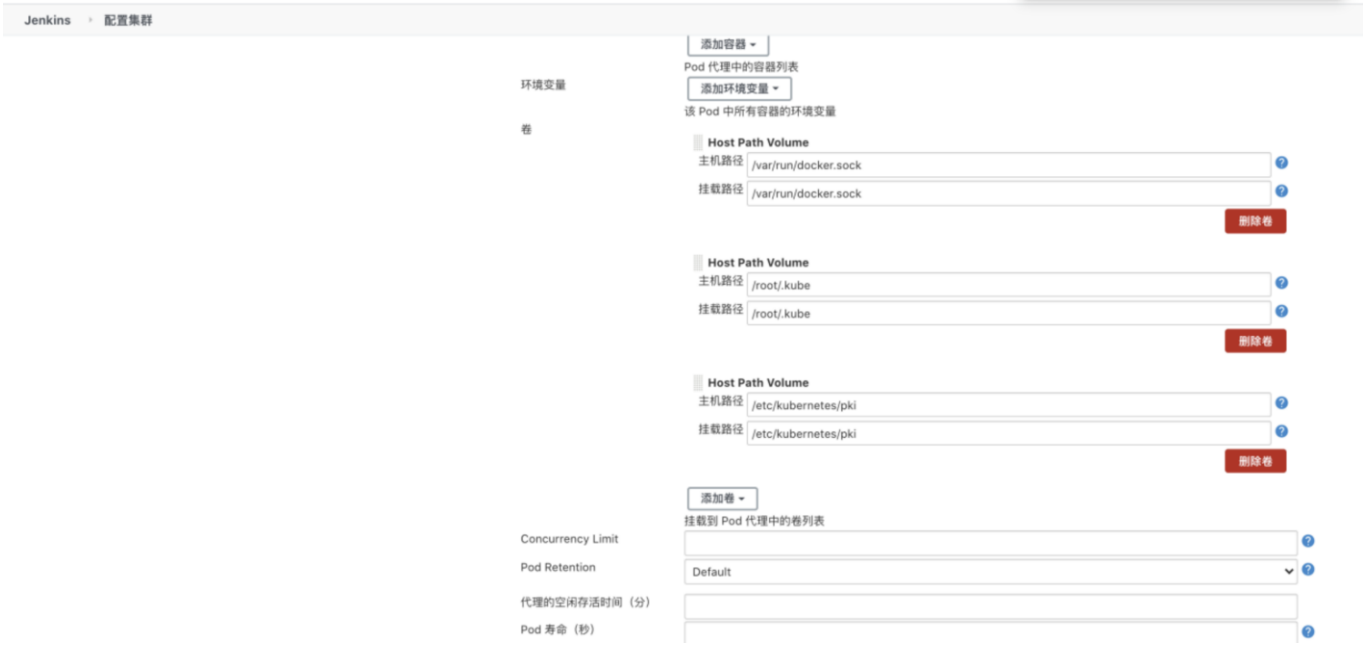
3.5 k8s pod template 配置

Jenkins 的 kubernetes-plugin 在执行构建时会在 kubernetes 集群中自动创建一个 Pod，并在 Pod 内部创建一个名为 jnlp 的容器，该容器会连接 Jenkins 并运行 Agent 程序，形成一个 Jenkins 的 Master 和 Slave 架构，然后 Slave 会执行构建脚本进行构建，但如果构建内容是要创建 Docker Image 就要实现 Docker In Docker 方案（在 Docker 里运行 Docker），如果要在集群集群内部进行部署操作可以使用 kubectl 执行命令，要解决 kubectl 的安装和权限分配问题。

为了方便配置一个 Pod Templates，在配置 kubernetes 连接内容的下面，这里的模板只是模板（与类一样使用时还要实例化过程），名称和标签列表不要以为是 Pod 的 name 和 label，这里的名称和标签列表只是 Jenkins 查找选择模板时使用的，Jenkins 自动创建 Pod 的 name 是项目名称+随机字母的组合，所以我们填写 jenkins-slave，命名空间填写对应的 namespace。

这边要注意，添加 2 个 container，第一个，Pod 内添加一个容器名称是 jnlp，Docker 镜像填写：jenkins/jnlp-slave，后面的使用默认的即可，然后在添加一个 container，容器名称是 jnlp-kubectl，是这个容器里面有 kubectl 的命令，镜像名称填写 harbor.edu.cn/library/centos-docker-kubectl:v1.0，下面增加了 Host Path Volume：/var/run/docker.sock、/root/.kube/、/etc/kubernetes/pki，这边便是为了 jenkins-slave 下有足够的权限可以执行 docker 及 kubectl 部署到 k8s 集群的权限，因为 jenkins-slave pod 有可能会被调度到任一 worker 节点，所以所有的 worker 节点上都必须有 /root/.kube/、/etc/kubernetes/pki，配置好之后点击保存。





3.6 Jenkins pipeline 说明

Pipeline，简单来说，就是一套运行在 Jenkins 上的工作流框架，将原来独立运行于单个或者多个节点的任务连接起来，实现单个任务难以完成的复杂流程编排和可视化的工作。

Jenkins Pipeline 有几个核心概念：

- Node：节点，一个 Node 就是一个 Jenkins 节点，Master 或者 Agent，是执行 Step 的具体运行环境，比如我们之前动态运行的 Jenkins Slave 就是一个 Node 节点
- Stage：阶段，一个 Pipeline 可以划分为若干个 Stage，每个 Stage 代表一组操作，比如：Build、Test、Deploy，Stage 是一个逻辑分组的概念，可以跨多个 Node
- Step：步骤，Step 是最基本的操作单元，可以是打印一句话，也可以是构建一个 Docker 镜像，由各类 Jenkins 插件提供，比如命令：sh 'make'，就相当于我们平时 shell 终端中执行 make 命令一样。

Pipeline 的使用：

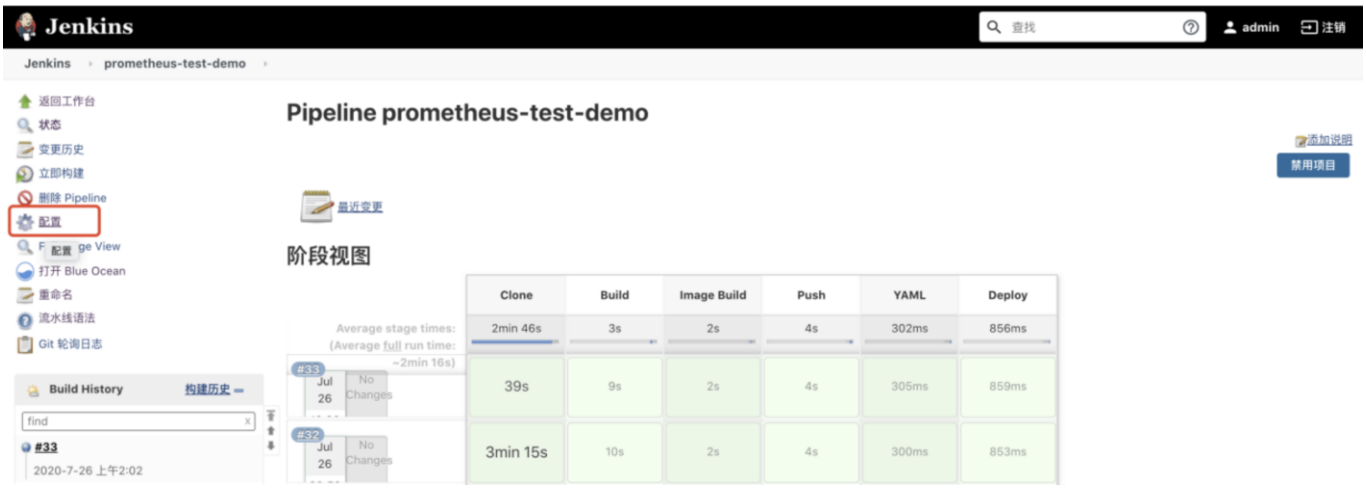
- Pipeline 脚本是由 Groovy 语言实现的
- Pipeline 支持两种语法：Declarative(声明式)和 Scripted Pipeline(脚本式)语法
- Pipeline 也有两种创建方法：可以直接在 Jenkins 的 Web UI 界面中输入脚本；也可以通过创建一个 Jenkinsfile 脚本文件放入项目源码库中
- 一般我们都推荐在 Jenkins 中直接从源代码控制(SCMD)中直接载入 Jenkinsfile Pipeline 这种方法，但是本次为了更直观的展示，我们在 Web UI 界面中输入脚本。
- 进入到 Jenkins 首页，点击项目 prometheus-test-demo，点击左侧配置
- 点击页面顶部的 **Pipeline** 选项卡，向下滚动到 **Pipeline** 部分。

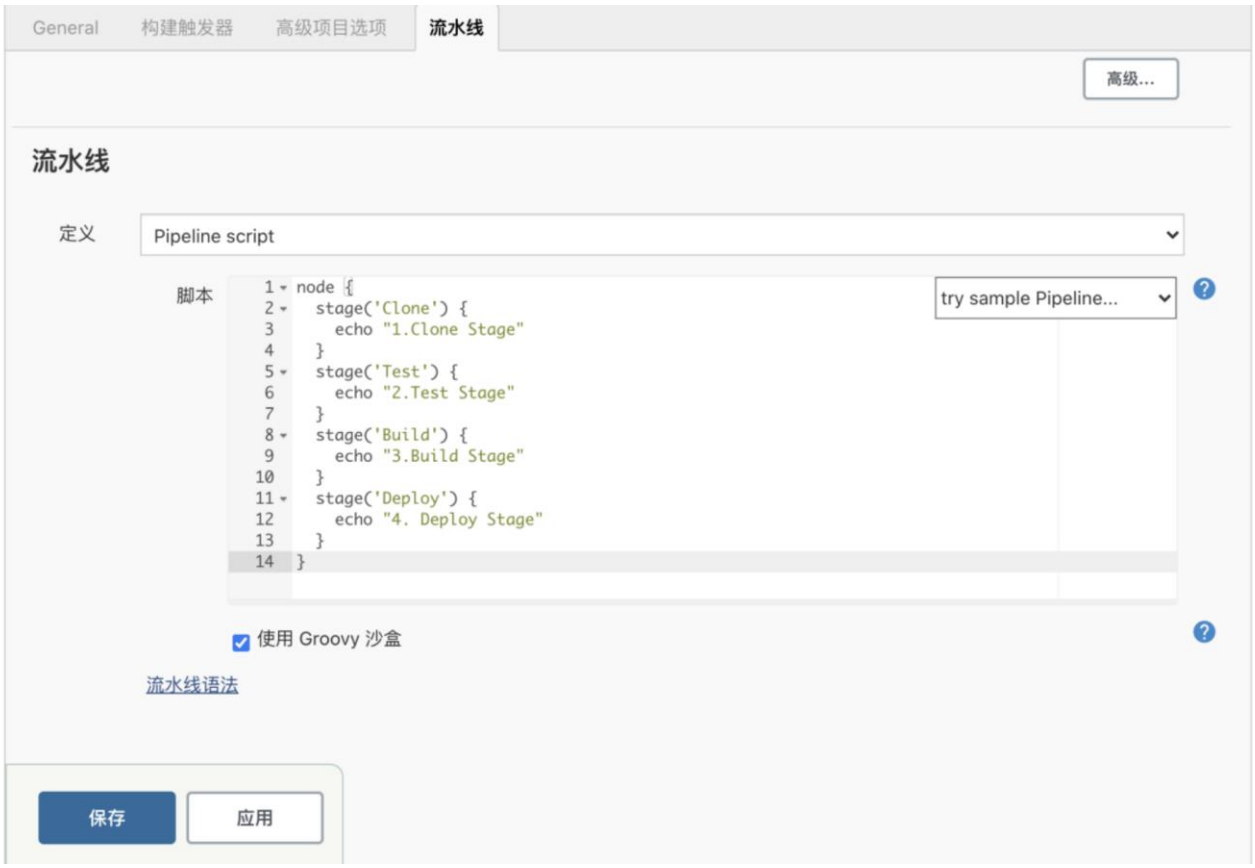
在 **定义** 域中，选择 **Pipeline script** 选项。

3.7 Jenkins pipeline 入门

3.7.1 创建并运行 pipeline

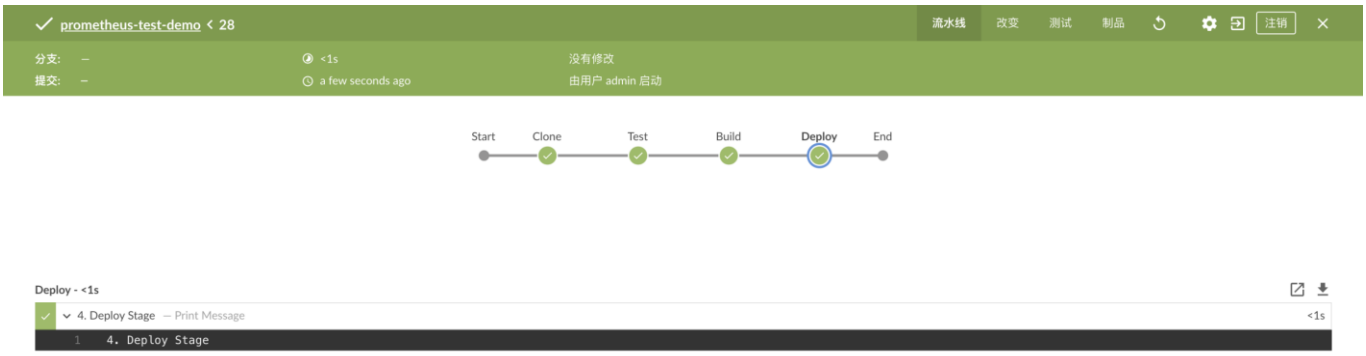
- 进入到 Jenkins 首页，点击项目 prometheus-test-demo，点击左侧配置
- 点击页面顶部的 **Pipeline** 选项卡，向下滚动到 **Pipeline** 部分。
- 在 **定义** 域中，选择 **Pipeline script** 选项。



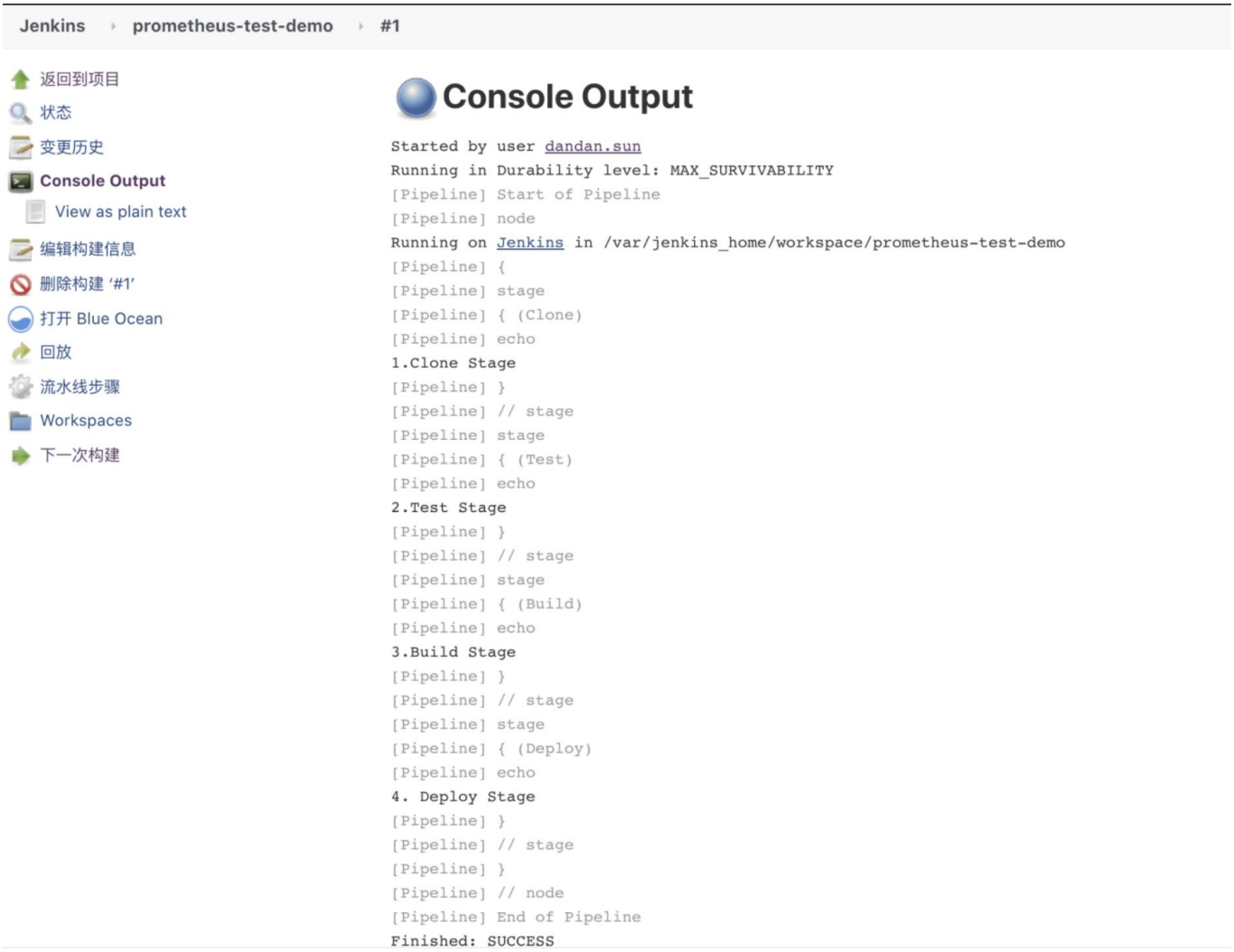



```
node {
  stage('Clone') {
    echo "1.Clone Stage"
  }
  stage('Test') {
    echo "2.Test Stage"
  }
  stage('Build') {
    echo "3.Build Stage"
  }
  stage('Deploy') {
    echo "4. Deploy Stage"
  }
}
```

点击保存，切换到 Jenkins 页面，点击左侧的 打开 Blue Ocean 进入 Jenkins 的 Blue Ocean 界面，进入到 相应的项目下，点击 运行 。



也可以在 Jenkins prometheus-test-demo 项目下，点击左侧菜单 立即构建，然后点击正在构建的任务，就可以看到 Console Output：



3.7.2 在 Slave 中运行 Pipeline

上面对 Jenkins 的 Pipeline 做了简单的测试，但是其并未在我们的 Slave 中运行，如果要在 Slave

中运行，其就要使用我们在对接 k8s 集群时 Pod Template 指定的标签列表 ，点击进 prometheus-test-demo 项目，点击左侧菜单 配置，进入到 pipeline scripts 部分，修改 pipeline scripts 如下：

```
node('slave') {
  stage('Clone') {
    echo "1.Clone Stage"
  }
  stage('Test') {
    echo "2.Test Stage"
  }
  stage('Build') {
    echo "3.Build Stage"
  }
  stage('Deploy') {
    echo "4. Deploy Stage"
  }
}
```

点击 立即构建，同时可以登录到 k8s 集群，使用 kubectl get po -w 可以看到 jenkins-slave pod 的生命周期，就是我们开始构建这个任务，选择了使用 jenkins slave，所以在执行过程中 jenkins-slave 就会自动创建，任务执行完成，jenkins-slave 对应的 pod 会自动回收：

```
[root@host-172-29-4-47 ~] eth0 = 172.29.4.47
# kubectl get pod -owide -w
NAME                                READY   STATUS    RESTARTS   AGE   IP             NODE          NOMINATED NODE   READINE
SS GATES
prometheus-test-demo-74877689f5-92d2h   1/1     Running   0          40m   10.93.217.142   172.29.4.55   <none>           <none>
jenkins-slave-dj3vc                 0/2     Pending   0          1s    <none>          <none>        <none>           <none>
jenkins-slave-dj3vc                 0/2     Pending   0          1s    <none>          172.29.4.57   <none>           <none>
jenkins-slave-dj3vc                 0/2     ContainerCreating   0          1s    <none>          172.29.4.57   <none>
   <none>
jenkins-slave-dj3vc                 0/2     ContainerCreating   0          3s    <none>          172.29.4.57   <none>
   <none>
jenkins-slave-dj3vc                 2/2     Running             0          3s    10.89.234.14    172.29.4.57   <none>
   <none>
```

在构建日志里我们也可以看到 jenkins 启动了 jenkins-slave-dj3vc pod 进行这个任务的执行，也和上面的 pod 名称对应起来了。

```
    path: "/etc/kubernetes/pki"
    name: "volume-2"
  - hostPath:
      path: "/root/.kube"
    name: "volume-1"
  - emptyDir:
      medium: ""
    name: "workspace-volume"

Running on jenkins-slave-dj3vc in /home/jenkins/agent/workspace/prometheus-test-demo
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Clone)
[Pipeline] echo
1.Clone Stage
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo
2.Test Stage
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] echo
3.Build Stage
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
4. Deploy Stage
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

3.8 完整 pipeline 示例

部署应用的流程如下：

- 拉取 Github 代码
- maven 打包
- 编写 Dockerfile
- 构建打包 Docker 镜像
- 推送 Docker 镜像到仓库
- 编写 Kubernetes YAML 文件
- 更改 YAML 文件中 Docker 镜像 TAG
- 利用 kubectl 工具部署应用

最终的 Pipeline 脚本如下：

Jenkinsfile

```
pipeline {
    agent none
    stages {
        stage('Clone Code') {
            agent {
                label 'master'
            }
            steps {
                echo "1.Git Clone Code"
                git url: "https://github.com/0820sdd/prometheus-test-demo.git"
            }
        }
        stage('Maven Build') {
            agent {
                docker {
                    image 'maven:latest'
                    args '-v /root/.m2:/root/.m2'
                }
            }
            steps {
                echo "2.Maven Build Stage"
                sh 'mvn -B clean package -Dmaven.test.skip=true'
            }
        }
        stage('Image Build') {
            agent {
                label 'master'
            }
            steps {
            echo "3.Image Build Stage"
            sh 'docker build -f Dockerfile --build-arg jar_name=target/prometheus-test-
demo-0.0.1-SNAPSHOT.jar -t prometheus-test-demo:${BUILD_ID} . '
            sh       'docker       tag               prometheus-test-demo:${BUILD_ID}
harbor.edu.cn/library/prometheus-test-demo:${BUILD_ID}'
            }
        }
        stage('Push') {
            agent {
                label 'master'
            }
            steps {
            echo "4.Push Docker Image Stage"
            sh "docker login --username=admin harbor.edu.cn -p Harbor12345"
            sh "docker push harbor.edu.cn/library/prometheus-test-demo:${BUILD_ID}"
            }
        }
```

```
    }
}


node('slave') {
    container('jnlp-kubectl') {

        stage('Clone YAML') {
        echo "5. Git Clone YAML To Slave"
        git url: "https://github.com/0820sdd/prometheus-test-demo.git"
        }

        stage('YAML') {
        echo "6. Change YAML File Stage"
        sh  'sed  -i  "s#{VERSION}#${BUILD_ID}#g"  ./jenkins/scripts/prometheus-test-
demo.yaml'
        }

        stage('Deploy') {
        echo "7. Deploy To K8s Stage"
        sh 'kubectl apply -f ./jenkins/scripts/prometheus-test-demo.yaml'
        }
    }
}
```

Code Block 3 Jenkinsfile

注意，prometheus-test-demo.yaml 可放在 github 的 prometheus-test-demo 仓库里，也可以另外新建一个仓库专门放 yaml 文件。

下面我们分解讲下上面过程的具体含义：

1. 克隆代码

Jenkinsfile

```
    stages {
        stage('Clone Code') {
            agent {
                label 'master'
            }
            steps {
                echo "1.Git Clone Code"
                git url: "https://github.com/0820sdd/prometheus-test-demo.git"
            }
```

```
}
```

这步就是从 github 上拉取代码，注意这边的 github 仓库仓库比如是 公开的，因为 private 的需要各种权限配置，jenkins 必须有一个公网 ip 或者是公网域名，但因资源问题，这部分暂时没有办法实现。注意，这边 agent 里面指定运行环境，选择了 master，即是这个步骤在 jenkins master 节点执行。

2. maven 打包

Jenkinsfile

```
stage('Maven Build') {
    agent {
        docker {
            image 'maven:latest'
            args '-v /root/.m2:/root/.m2'
        }
    }
    steps {
        echo "2.Maven Build Stage"
        sh 'mvn -B clean package -Dmaven.test.skip=true'
    }
}
```

maven 构建，我们指定了 maven 打包的 agent 是在 Jenkins 所在节点另起一个 docker 容器，容器的 image 为 maven:latest，并且使用 -v 参数把本地的 /root/.m2 挂载到 容器的 /root/.m2 目录下，下面 steps 的步骤即是在这个 maven 容器里面的具体操作：mvn -B clean package -Dmaven.test.skip=true。

3. 构建镜像

Jenkinsfile

```
stage('Image Build') {
    agent {
        label 'master'
    }
    steps {
    echo "3.Image Build Stage"
    sh 'docker build -f Dockerfile --build-arg jar_name=target/prometheus-test-demo-0.0.1-SNAPSHOT.jar -t prometheus-test-demo:${BUILD_ID} . '
    sh 'docker tag prometheus-test-demo:${BUILD_ID} harbor.edu.cn/library/prometheus-test-demo:${BUILD_ID}'
    }
}
```

maven 构建成功，下一步就是使用 maven build 生成的 prometheus-test-demo-0.0.1-SNAPSHOT.jar 包进行 docker build，docker build 的具体命令有 2 条 bash 命令 组成，第一步 docker build 使用 -f 指定了 Dockerfile 的文件，使用 --build-arg 参数指定了一些参数，比如上面指定了 jar_name 是 target/prometheus-test-demo-0.0.1-SNAPSHOT.jar，最后使用 -t 参数指定了 docker build 的 image 的名称及版本号。第二步就是 使用 docker tag 命令把上一步 docker build 完成的镜像 打 tag 为 harbor.edu.cn/library/prometheus-test-demo:${BUILD_ID}，这步打 tag 的步骤是为了上传到 harbor 镜像仓库，可以随时使用。

4. 推送镜像

Jenkinsfile

```
        stage('Push') {
            agent {
                label 'master'
            }
            steps {
            echo "4.Push Docker Image Stage"
            sh "docker login --username=admin harbor.edu.cn -p Harbor12345"
            sh "docker push harbor.edu.cn/library/prometheus-test-demo:${BUILD_ID}"
            }
        }
    }
}
```

镜像 build 完成，就可以使用 docker push 命令推送到 harbor.edu.cn 镜像仓库。

5. 拉取镜像

Jenkinsfile

```
        stage('Clone YAML') {
        echo "5. Git Clone YAML To Slave"
        git url: "https://github.com/0820sdd/prometheus-test-demo.git"
        }
```

现在镜像已经打包完成，并推送到了镜像仓库，后面我们所要做的就是拉取 k8s 编排文件，这一步和第一步的 拉取代码实际是一样的，只不过上面的拉取代码是为了 build image，这一步是为了进行部署到 k8s。

注意：这边指定了运行此步骤的节点是在 Jenkins 的 slave 节点下的 jnlp-kubectl container 下，这个 slave 是指在配置 对接 k8s 集群时，在 Pod Template 下指定的 标签列表的名称，

必须与这个名称一致，不然 jenkins 执行过程中就会报找不到对应的 label 。还有这边指定了 jnlp-kubectl container ，这是因为 jnlp-kubectl container 下有 kubectl 命令，且配置 对接 k8s 集群时，指定了把宿主机的 /root/.kube /etc/kubernetes/pki 目录分别挂载到 container 的 /root/.kube /etc/kubernetes/pki 目录下，这边就是 jnlp-kubectl container 可以访问 k8s 集群的原因。

6. 替换 YAML 文件变量

Jenkinsfile

```
        stage('YAML') {
        echo "6. Change YAML File Stage"
        sh  'sed  -i  "s#{VERSION}#${BUILD_ID}#g"  ./jenkins/scripts/prometheus-test-demo.yaml'
        }
```

yaml 文件拉取完毕，替换其中的变量。

7. 部署

Jenkinsfile

```
        stage('Deploy') {
        echo "7. Deploy To K8s Stage"
        sh 'kubectl apply -f ./jenkins/scripts/prometheus-test-demo.yaml'
        }
```

使用 kubectl 命令部署 prometheus-test-demo 应用到 k8s 集群。

8.

最终执行效果如下：


```
[root@host-172-29-4-47 ~] eth0 = 172.29.4.47
# kubectl get pod
NAME                                 READY   STATUS    RESTARTS   AGE
prometheus-test-demo-565bd8549-2p9jx   1/1     Running   0          46s
[root@host-172-29-4-47 ~] eth0 = 172.29.4.47
# kubectl get svc
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes           ClusterIP   10.64.0.1      <none>        443/TCP          14d
prometheus-test-demo   NodePort    10.64.120.85   <none>        8998:49097/TCP   3d15h
[root@host-172-29-4-47 ~] eth0 = 172.29.4.47
```

4 参考资料

https://www.jenkins.io/zh/doc/tutorials/build-a-java-app-with-maven/