Linux Academy

Hands-on Lab

Configure a Jenkins Pipeline

Contents

Set up a Docker Jenkins Master 1

Forking a GitHub Repo 2

Generate SSH Keys for the Containerized Jenkins Master 2

Configure and Run the Pipeline 3

Lab Connection Information

- Labs may take up to five minutes to build
- The IP address of your server is located on the Live! Lab page
- Username: linuxacademy
- Password: 123456
- Root Password: 123456

Related Courses

Certified Jenkins Engineer - 2017

Related Videos

The Jenkinsfile

Configuring the Running a Pipeline

Need Help?

Linux Academy Community

... and you can always send in a support ticket on our website to talk to an instructor!

In this lab, we set up a Jenkins pipeline using GitHub and a Jenkinsfile. We also learn how to archive artifacts of our builds.

Set up a Docker Jenkins Master

Log in to the server using the credentials provided on the Hands-on Lab page.

Add the Docker repository:

```
[linuxacademy@ip] sudo yum-config-manager --add-repo https://download.
docker.com/linux/centos/docker-ce.repo
[linuxacademy@ip] sudo yum clean all
```

Install the version of Docker CE we want to download:

```
[linuxacademy@ip] yum install -y --setopt=obsoletes=0 \
docker-ce-17.03.1.ce-1.el7.centos \
docker-ce-selinux-17.03.1.ce-1.el7.centos
```

You will be prompted to confirm GPG keys.

Start the Docker services:

```
[linuxacademy@ip] sudo systemctl start docker
```

We now want to pull down the Jenkins Docker image:

```
[linuxacademy@ip] sudo docker pull jenkins:2.19.4
```

Create a directory for the Jenkins container to share on the host:

```
[linuxacademy@ip] sudo mkdir /var/jenkins_home
```

Run the container:

```
[linuxacademy@ip] sudo docker run -d -u root -p 8080:8080 -p 50000:50000
-v /var/jenkins_home:/var/jenkins_home jenkins:2.19.4
```

`-d` denotes that we are running the task in the background, while the `-p` flag shows which ports we are using. The `-v` maps the `/var/jenkins_home` directories between the two hosts.

To confirm the process is successful, navigate to your web browser and input the IP address of our host, followed by `:8080` to access the "Getting Started" Jenkins page. Copy the directory path provided, so we can determine the administrative password, then `cat` the file from your terminal to view:

```
[linuxacademy@ip] sudo cat /var/jenkins_home/secrets/
initialAdminPassword
```

Use that password to log in to Jenkins from your web browser. **Install suggested plugins**, then wait for the installation process to finish. We are now prompted to create our first admin user. Set the credentials to whatever you wish, but be sure to remember the username and password. **Save and Finish**. **Start using Jenkins**.

Forking a GitHub Repo

In your web browser, access and log in to github.com. Navigate to the repository for this lab, located at the following URL: https://github.com/linuxacademy/content-jenkins-pipeline

Press the **Fork** button in the upper right corner to fork it to your account.

Generate SSH Keys for the Containerized Jenkins Master

Return to your lab server. We need to generate an SSH key. Use `docker ps` to retrieve your container ID:

```
[linuxacademy@ip] sudo docker ps
```

Now use `docker exec` to generate the key, inputting your container ID in the defined place:

```
[linuxacademy@ip] sudo docker exec -it <CONTAINTERID> bash
```

This drop us into a Bash shell inside of the Docker container. Generate the SSH key, using the defaults if you so desire:

```
root@dockerip: cd ~
root@dockerip: ssh-keygen
```

Copy your public key to add to GitHub:

```
root@dockerip: cat .ssh/id_rsa.pub
```

Return to your fork of the Jenkins Pipeline project on GitHub and press **Settings**, then **Deploy keys**. Press

Add deploy key. Copy your public key into the text box. Give the key a title. Check *Allow write access*. **Add key**.

Press **Integrations & services** on the left menu. Under **Services,** select **Add service**, then find *Jenkins (Git plugin)*. For the **Jenkins URL** input the server's IP address, including `http://` and the port 8080, then press **Add Service**.

Configure and Run the Pipeline

Press **New Item** on the Jenkins Dashboard. **Enter an item name** of *Lab Pipeline* and select *Pipeline* from the list of options. Press **OK**.

Check *GitHub project* and input the URL to your fork of the Jenkins Pipeline repository.

Under **Build Triggers**, select *Poll SCM*, not inputting anything into the text box.

Scroll down to the **Pipeline** section, and for **Definition** set it to *Pipeline script from SCM*. Set **SCM** to *Git* and input the **Repository URL**, which is the Clone with SSH link found on your GitHub fork, under the green **Clone or Download** button. For **Credentials**, press **Add**, **Jenkins**, then set the **Kind** to *Username with private key*. Set the **Username** to *root* and the **Private Key** to *From the Jenkins master ~/.ssh*. **Add**. Select *root* under the credentials dropdown.

Leave **Branches to build** set on the *master* branch, and change the **Repository browser** to *githubweb*, copying in the URL to your fork of the project. **Save** the pipeline.

From your workstation computer, clone the git repository using the HTTPS or SSH via the **Clone or download** button:

```
[user@workstation] git clone <URL>
```

Make a new file under the repository directory using your preferred text editor. The file should be called `Jenkinsfile`. Add the following:

```
pipeline {
    agent any
    stages {
        stage('build') {
            steps {
                sh 'javac -d . src/*.java'
                sh 'echo Main-Class: Rectangulator > MANIFEST.MF'
                sh 'jar -cvmf MANIFEST.MF rectangle.jar *.class'
            }
        }
    }
}
```

Save and exit the file.

Commit the changes to the Jenkins Pipeline repository:

```
[user@workstation] git add Jenkinsfile
[user@workstation] git commit -m "Added Jenkinsfile with a build step"
[user@workstation] git push origin master
```

Return to the Jenkins Dashboard. Pipelines require users to trigger the first build, so select your project page and press **Build Now**.

Under **Build History** click the number of the build (**#1**) to ensure that the build was successful by viewing the **Console Output**.

Return to your workstation and open the `Jenkinsfile` again. We want to execute the `rectangle.jar` script:

```
pipeline {
    agent any
    stages {
        stage('build') {
            steps {
                sh 'javac -d . src/*.java'
                sh 'echo Main-Class: Rectangulator > MANIFEST.MF'
                sh 'jar -cvmf MANIFEST.MF rectangle.jar *.class'
            }
        }
        stage('run') {
            steps {
                sh 'java -jar rectangle.jar 7 9'
            }
        }
    }
}
```

Save and exit the file.

Add and commit the file:

```
[user@workstation] git commit -am "Added build step"
[user@workstation] git push origin master
```

Return to your Jenkins Dashboard. A new build has started.

We now want to add an archival step to our `Jenkinsfile`. Once more, open the file, and add an additional step:

```
pipeline {
        agent any
        stages {
            stage('build') {
                steps {
                    sh 'javac -d . src/*.java'
                    sh 'echo Main-Class: Rectangulator > MANIFEST.MF'
                    sh 'jar -cvmf MANIFEST.MF rectangle.jar *.class'
                }
            }
            stage('run') {
                steps {
                    sh 'java -jar rectangle.jar 7 9'
                }
            }
        }
        post {
            success {
                archiveArtifacts artifacts: 'rectangle.jar', fingerprint: 
    true
            }
        }
    }
```

Save and exit the file.

Commit the changes:

```
[user@workstation] git commit -am "Added post step with archive"
[user@workstation] git push origin master
```

Return to the Jenkins Dashboard. A third build has been triggered. Press the **#3** under **Build History**, then **Console Output** to view and ensure that the build was successful. Should we return to the general build overview, we can see the archived `rectangle.jar` file under **Build Artifacts**.

This lab is now complete!