**Building a Jenkins Pipeline**

Jenkins Pipeline allows you to treat all of your build logic as code. You can also integrate your build into a larger automated process that includes things like automated deployment. Pipelines are especially useful as you transition from doing CI to doing CI+CD.

In this hands-on lab, you will build a simple pipeline for a sample application. You will gain a basic familiarity with how a Jenkins Pipeline is created and how to implement simple build logic within a pipeline.

**The Scenario**

Your team is building a train scheduling app. Continuous integration has already been implemented in Jenkins, but they want to move toward doing automated deployments as well. You have been asked to build a Jenkins Pipeline project called train-schedule to build the train-schedule app. The pipeline project needs to build the code, run the automated tests, and archive the deployable zip artifact that is created by the build.

To do this, you will need to:

* Fork the source repo.
* Create a Jenkinsfile for the app.
* Create a stage to execute the gradle build.
* Archive the *dist/trainSchedule.zip* artifact.
* Create new multibranch pipeline project in Jenkins called train-schedule.
* Run the build successfully in Jenkins.

Feel free to look at the *example-solution* branch of the git repo for an example of the source code changes needed for the solution.

Some useful links:

* Jenkins Pipelines documentation: <https://jenkins.io/doc/book/pipeline/>

**Build the Master Branch of the train-schedule Pipeline Project**

Create a personal fork of the sample source code repository, which you can find [here](https://github.com/linuxacademy/cicd-pipeline-train-schedule-pipelines).

Click **Create new file**. Name the file **Jenkinsfile**.

Enter:

pipeline { agent any stages { stage ('Build') { steps { echo 'Running build automation' sh './gradlew build --no-daemon' archiveArtifacts artifacts: 'dist/trainSchedule.zip' } } } }

Click **Commit new file** at the bottom.

Before we can test this in Jenkins, we need to get an API key from GitHub so Jenkins can pull down this source code from GitHub. To do this:

* Click your avatar in the top right corner
* Click **Settings**.
* Click **Developer settings**.
* Click **Personal access tokens**.
* Click **Generate new token**.
* *Token description*: **jenkins**
* Check the box next to **admin:repo\_hook**.
* **Generate token**

Copy the generated token to the clipboard.

**Create a New Multibranch Pipeline Project in Jenkins**

Navigate to the Jenkins server by entering the public IP address listed on the lab page, followed by :8080, in a browser.

In Jenkins, click **New Item**.

For the item name, type train-schedule, and select **Multibranch Pipeline**. Click **OK**.

Under *Branch Sources*, click **Add source**, and select **Git**.

Under *Credentials*, click **Add**, and select **Jenkins**.

* *Username*: Enter your GitHub username.
* *Password*: Paste in the API key you copied before.
* *ID*: **github\_key**
* *Description*: **GitHub key**
* Click **Add**.

In the new *Git* section under *Branch Sources*, click the *Credentials* dropdown and select the credential you just created. For *Project Repository*, copy and paste in the URL of your personal fork of the source code repository. Click **Save**.

The initial build will take several minutes to complete.

**Conclusion**

Congratulations on completing this lab!

GUIDE

**Distributing a Jenkins Build**

**Introduction**

In this hands-on lab we will be creating a build agent on a second server, and then creating a build that runs only on that server. This will demonstrate the use of labels for the build.

**The Scenario**

Your company has decided that they need to scale out their Jenkins installation, and that running projects on the master is no longer desirable. You have been tasked with configuring a build agent on a second server. Log into the Jenkins master web interface at port :8080 using these credentials: *username:* **student** *password:* **OmgPassword!**.

It should have the following configuration:

* The node name should be worker1.
* It should connect via SSH with keys.
* The root directory should be /var/lib/jenkins.
* The remote user should be jenkins.
* The remote should have 2 executors.
* All builds should be configured to run on the agent by default.
* Ensure that the build is running on the agent by issuing a hostname command.
* Verify that the agent builds are able to be archived and fingerprinted.

**Configure SSH on the Remote Agent**

Log into the master via SSH, using the credentials provided on the hands-on lab overview page. Then from the master node, use SSH to log into the worker node.

On the worker node, create the directory for the user's home:

[cloud\_user@worker ]$ sudo mkdir /var/lib/jenkins

Add the user, assigning the home directory:

[cloud\_user@worker ]$ sudo useradd -d /var/lib/jenkins jenkins

Make the user the owner of their home directory:

[cloud\_user@worker ]$ sudo chown -R jenkins:jenkins /var/lib/jenkins

Create an .ssh directory for the jenkins user:

[cloud\_user@worker ]$ sudo mkdir /var/lib/jenkins/.ssh

Run ssh-keygen. Hit **Enter** to accept defaults until it completes.

Copy the contents of ~/.ssh/id\_rsa.pub to the file /var/lib/jenkins/.ssh/authorized\_keys

[cloud\_user@worker ]$ cat ~/.ssh/id\_rsa.pub # Copy the output [cloud\_user@worker ]$ sudo vim /var/lib/jenkins/.ssh/authorized\_keys # Paste the output of cat and save

We need the contents of id\_rsa, as this is the private key that we will paste into Jenkins:

[cloud\_user@worker ]$ cat ~/.ssh/id\_rsa

Copy that command's output and exit the worker node.

Create an .ssh directory on the master in the jenkins directory:

[cloud\_user@master ]$ sudo mkdir /var/lib/jenkins/.ssh

Copy the known\_hosts entry over from the .ssh directory in /home/cloud\_user to the jenkins user's .ssh directory:

[cloud\_user@master ]$ sudo cp ~/.ssh/known\_hosts /var/lib/jenkins/.ssh

**Set up the Node on the Jenkins Master**

Log into the Jenkins master using these credentials: *username:* **student** *password:* **OmgPassword!**

In the Jenkins master, go to **Manage Jenkins** and select **Manage Nodes and Clouds**. On this page, click **New Node**, and fill out the following web form with these values:

* *Name:* **worker1**
* *#of executors:* **2**
* *Remote root directory:* **/var/lib/jenkins**
* *Labels:* **linux**
* *Host*: Use the public IP of the worker node here.
* *Credentials:* Click the **Add** dropdown and select **Jenkins**. This will pop up another form we need to fill out:
  + *Kind:* **SSH Username with private key**
  + *Username:* **jenkins**
  + *Private Key:* Click **Enter directly** and paste in the key that we copied right before we exited from the worker node SSH session.
  + *Description:* **jssh**
  + Click **Add**.
* We'll land back on the first form we were filling out. Finish up the *Credentials* section by clicking the dropdown (it currently says *-none-*) and choosing **jenkins (jssh)**.

Leave any fields alone that weren't mentioned here, and click **Save**.

On the *Nodes* page we are redirected to, click on **worker1**, then **See log for more details**. We're looking to see if there was a successful SSH authentication. If there's one shown in the log, and we see an entry saying *Agent successfully connected and online*, we're good to go.

**Test a Remote Build**

We need to configure the master to use labels, to ensure default builds run on the remote.

On the master node, get back into the main Jenkins screen, then in the left-hand menu click **Manage Jenkins**. Now click on select **Configure System**, and then we can fill out a couple of form values:

* On the *Labels* line, enter **master**.
* Set *Usage* to **Only build jobs with label expressions matching this node**.
* Click **Save**.

Back out on the main Jenkins screen, click **New Item**. Enter an item name of **test**, click on **Freestyle project**, then click **OK**.

On the next screen, scroll down to the *Build* section, and choose **Execute shell** from the *Add build step*\* dropdown.

In the *Command* box, type this:

hostname > location.txt

Now, in *Post-build Actions*, click the **Add post-build action** dropdown and select **Archive the artifacts**. Paste **location.txt** in the *Files to archive* box. Click on **Advanced** and check the box next to *Fingerprint all archived artifacts*.

Click **Save**.

Run the build by clicking **Build Now** in the left-hand menu, then click on the build number to get into the build itself. If we watch the Console Output, we'll see the build process happening.

Go back to the **test** environment (using the breadcrumb navigation at the top of the screen), and we will see **location.txt** listed in the *Last Successful Artifacts* section. Open the **view** link that's next to it, in a new tab or window.

**Conclusion**

If we see **jslave** in that window, we know the build executed on the worker node, not the master. We're done! Congratulations!

# Building In Jenkins

## Introduction

In this lab, you will create a simple shell build in a folder on a prepped server with Jenkins installed to help you get comfortable with the interface and navigation in the freestyle build type. Once you have run the build, you will then check the output and verify that the build is progressing as intended.

## Solution

Access the Jenkins environment by copying the public IP address provided with the lab credentials. Then, in a browser window, navigate to that IP address using:

http://<public IP address>:8080

When prompted, log in to Jenkins with the username student and the password OmgPassword!

### Create a Folder Named Test

1. Once logged in to Jenkins, click **New Item** in the left pane.
2. In the Enter an item name field, type "Test".
3. From the list of available item types, select **Folder**.
4. Click **OK**.
5. Click **Save** to accept the default configurations for the folder.

### Create and Run a User Test Job That Saves Its Output in a File Named user\_test.txt

1. In the Test folder you just created, click the **create new jobs** link.

**Note:** You could also click **New Item** in the left pane.

1. In the Enter an item name field, type "user\_test".
2. From the list of available item types, make sure that **Freestyle project** is selected.
3. Click **OK**.
4. In the configuration for the project, scroll down to the Build section and click the **Add build step** drop-down arrow.
5. From the drop-down menu, select **Execute shell**.
6. In the Command text box, add the code containing the uname and whoami commands required to output the desired information to the requested file name:

uname -a && whoami > user\_test.txt

1. Click **Save**.
2. In the user\_test project, click **Build Now** in the left pane.
3. Verify that the build completes in the Build History pane.
4. Once completed, click **Workspace** in the project pane.
5. Click the **view** link to the right of the user\_test.txt file.
6. Verify that the output in the file displays jenkins.
7. Click the back button in the browser window.
8. Click on the drop-down arrow next to the build in the Build History pane and click **Console Output**.
9. In the Console Output, view the output in the file and verify the following:
   * The username of the user who ran the build (which should be student, the username with which you are signed in).
   * The version of Linux that the build was running on.

## Conclusion

Congratulations — you've completed this hands-on lab!

# Implementing Automated Deployment Through a Jenkins Pipeline

## Introduction

Jenkins Pipeline is a powerful tool for implementing continuous delivery. In order to fully utilize Jenkins Pipeline, you will need to implement an automated deployment. This learning activity will guide you through the process of deploying code as part of a Jenkins pipeline. After completing this exercise, you will have a basic familarity with what automated deployment using Jenkins Pipeline looks like.

## Solution

1. In a new browser tab, log in to the Jenkins instance using the Jenkins server public credentials provided:

<PUBLIC\_IP\_ADDRESS>:8080

1. Using the same credentials, open a terminal window and log in to the Jenkins server using SSH to retrieve the temporary admin password:

ssh cloud\_user@<PUBLIC\_IP\_ADDRESS>

1. Retrieve the temporary admin password:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

1. Copy the temporary admin password and paste it into the Administrator password field in the new browser tab.
2. Click **Continue**.

## Deploy the App to the Staging Server Via the Jenkins Pipeline

### Getting started

1. On the Create First Admin User form, provide the following information:
   * Username: **jenkins**
   * Password: random
   * Confirm password: random
   * Full name: **jenkins**
   * Email address: [**noreply@linuxacademy.com**](mailto:noreply@linuxacademy.com)
2. Click **Save and Continue** > **Start using Jenkins**.

### Configure staging and production Servers for the Publish Over SSH Plugin

1. On the lab page, copy the staging server public IP address.
2. In the Jenkins tab in your browser, click **Manage Jenkins** and then click **Configure System**.
3. Scroll to the bottom of this page to find the Publish over SSH section.
4. In SSH Servers, click **Add**.
5. Add the following SSH Server values, replacing <STAGING\_SERVER\_PUBLIC\_IP\_ADDRESS> with the IP address copied earlier:
   * Name: **staging**
   * Hostname: <STAGING\_SERVER\_PUBLIC\_IP\_ADDRESS>
   * Remote Directory: **/**
6. Click **Add**.
7. Return to the lab page and copy the production server public IP address.
8. Add the following SSH Server values, replacing <PRODUCTION\_SERVER\_PUBLIC\_IP\_ADDRESS> with the IP address copied earlier:
   * Name: **production**
   * Hostname: <PRODUCTION\_SERVER\_PUBLIC\_IP\_ADDRESS>
   * Remote Directory: **/**
9. Click **Save**.

### Set Up Jenkins Credentials

1. From the left menu, click **Credentials**.
2. In Stores scoped to Jenkins, click **global**.
3. On the left, click **Add Credentials**.
4. Add the following values:
   * Username: **deploy**
   * Password: **jenkins**
   * ID: **webserver\_login**
   * Description: **Webserver Login**
5. Click **OK** to save our changes.

### Set up the Jenkins Project

1. On the top menu, click **Jenkins** to return to the main page.
2. From the left menu, click **New Item**.
3. Enter the item name "train-schedule".
4. Select **Multibranch Pipeline** and click **OK**.
5. Navigate to the [train-schedule Git repo](https://github.com/linuxacademy/cicd-pipeline-train-schedule-cd) and click **Fork** to fork the repo to your account.
6. From the GitHub top menu, click the avatar icon and click **Settings** > **Developer settings** > **personal access tokens** > **Generate new token**.
7. In Token description, enter "Jenkins".
8. Select **admin:repo\_hook**. and click **Generate token**.
9. Copy the generated API token to the clipboard.
10. Return to the Jenkins page and select the Branch sources tab.
11. In Branch Sources, click **Add source** and select **GitHub**.
12. In Credentials, click Add and select **Jenkins**.
13. Set the following values:

\* \*Username\*: Your GitHub username \* \*Password\*: Generated API token copied earlier \* \*ID\*: \*\*github\_key\*\* \* \*Description\*: \*\*GitHub Key\*\*

1. Click **Add**.
2. In Credentials, select the newly created GitHub Key credential.
3. In Owner, enter your GitGub username.
4. In Repository, select **cicd-pipeline-train-schedule-cd**.
5. Click **Save**.
6. From the top menu, click **train-schedule** and then click **master** to view the initial build in the master branch.

### Create a Stage in the Jenkinsfile and Run the Build

1. From the GitHub personal fork, open the Jenkinsfile and click the pencil icon to edit the file's contents.
2. Delete the file contents.
3. In a new browser tab, access the solution Jenkinsfile on the [Github example-solution branch](https://github.com/linuxacademy/cicd-pipeline-train-schedule-cd/blob/example-solution/Jenkinsfile) and copy the DeployToStaging stage text.
4. Paste the text into the Jenkinsfile in our GitHub fork.
5. Click **Commit changes**.
6. Return to the Jenkins Branch master page.
7. On the left menu, click **Build Now**.
8. To test our deployment, copy the staging server public IP address again from the lab page.
9. Open a new browser tab and paste the IP address, specifying the port 3000. Our train schedule application should load successfully.

## Deploy the App to the Production Server Via the Jenkins Pipeline

1. From the GitHub personal fork, open the Jenkinsfile and click the pencil icon to edit the file's contents.
2. In a new browser tab, access the solution Jenkinsfile on the [Github example-solution branch](https://github.com/linuxacademy/cicd-pipeline-train-schedule-cd/blob/example-solution/Jenkinsfile) and copy the DeployToProduction stage text.
3. Paste the text into the Jenkinsfile in our GitHub fork.
4. Click **Commit changes**.
5. Return to the Jenkins Branch master page.
6. On the left menu, click **Build Now**.
7. Hover over DeployToProduction and click **Proceed**.
8. To test our deployment, copy the production server public IP address again from the lab page.
9. Open a new browser tab and paste the IP address, specifying the port 3000. Our train schedule application should load successfully.

## Conclusion

Congratulations — you've completed this hands-on lab!