DevOps – Laboratory 5

Automation and Jenkins

This lab will introduce you to Automation which is a major component of any DevOps pipeline. You will learn how to use the Jenkins automation tool to compile and test your code as soon as you push changes to GitHub.

# Getting Started

**You will need to launch an AWS EC2 Instance on which you will complete this Lab.**

**For this lab you will need an instance with more resource that the previous labs. During the setup you should select the t2.large instance size.**

**Use the guidance in Lab 1 and 3 to launch an instance with ports 9000 and 8080 open and 12GB of storage space.**

**When you have successfully connected to your new VM via SSH you are ready to continue with this Lab.**

**PLEASE REMEMBER TO STOP OR TERMINATE YOUR VM AT THE END OF THIS LAB SO THAT YOU DO NOT USE UNNECCESARY CREDIT!!**

# Important Information

This Lab documents will contain commands that you should run in the bash terminal (both locally and on your VM). This code will be provided in boxes as shown below:

|  |
| --- |
| This is a sample command that fits on one line  This is a much longer sample command that covers multiple lines but should still be copied as a single command |

As you can see, some commands will be short and clearly displayed on a single line. This can be copied directly into the Terminal. Other commands such as the second one are much longer and do not fit on one line. If you are going to copy and paste these commands make sure to copy all lines and paste as a single command. Each new command is separated by a blank line so that each command is clearly distinguished.

**If you are unfamiliar with using the Terminal or Bash it is highly recommended that you complete the following Tutorial:**

[**https://ubuntu.com/tutorials/command-line-for-beginners**](https://ubuntu.com/tutorials/command-line-for-beginners)

This tutorial is written for Ubuntu but as the Terminal we are using (GitBash for Windows or Terminal for MacOS/Linux) also uses Bash, the commands are the same and the tutorial is very relevant. The Virtual Machines you create on AWS will utilise the Ubuntu Operating system and therefore this tutorial will also be useful when interacting with your VMS.

Section 7 of this Tutorial covers SuperUser which is not applicable to Windows Systems but will be important on your VMs so reading is recommended.

The following Cheat Sheet is also helpful for remembering Bash commands and will be a good point of reference throughout this Module:

<https://cheatography.com/davechild/cheat-sheets/linux-command-line/pdf/>

**If you encounter any issued during this process, please contact the Module Tutor.**

# Installing Jenkins

1. Connect to your Build Server using SSH
2. Install Docker using any of the methods we have utilised so far
3. Jenkins needs Java so we now need to install the correct Java Packages using the commands below:

sudo apt update

sudo apt install -y openjdk-17-jre

sudo apt install -y default-jdk

1. Now we are ready to install Jenkins. Run the following commands to install Jenkins directly on your EC2 Instance:

curl -fsSL <https://pkg.jenkins.io/debian/jenkins.io-2023.key> | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install -y fontconfig

sudo apt-get install -y jenkins

1. Jenkins has created its own user profile (different from the ubuntu user you log in as) so we need to add this to the docker group so that Jenkins can use Docker:

sudo usermod -a -G docker jenkins

sudo adduser jenkins docker

1. The Jenkins application is running on port 8080 as specified earlier so you can connect to it through your web browser by entering the following in the address bar, substituting your Public IP Address:

ip\_address:8080

You will be directed to a screen like this:

A screenshot of a social media post

Description automatically generated

Figure : Jenkins Unlock Screen

1. To get initial admin password run:

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

1. Copy the 32 character code into the Jenkins window and click continue.
2. On the next screen, click on the ‘Install Suggested Plugins’ icon and wait while Jenkins installs various plugins, some of which will be helpful later.
3. You should now be asked to create an Admin account. Follow the instructions on the screen to do this and click continue.
4. On the ‘Instance Configuration’ page just click ‘Save and Finish’ then click ‘Start Using Jenkins’
5. Go to Manage Jenkins > Plugins. Select the Available Plugins option and search for and install the following plugins using the ‘install without restart’ option:
   1. Groovy
   2. Docker Pipeline
   3. SSH Agent
   4. SSH Pipeline Steps
6. Restart Jenkins by entering the following in the address bar of your browser. You will be taken to a page with an option to restart Jenkins:
   1. YOUR\_IP\_ADDRESS:8080/restart
7. Jenkins is now ready to use as normal. When it restarts, log in with the admin details you created earlier. If you restart your VM, Jenkins will start running again automatically and will have saved all of your settings.

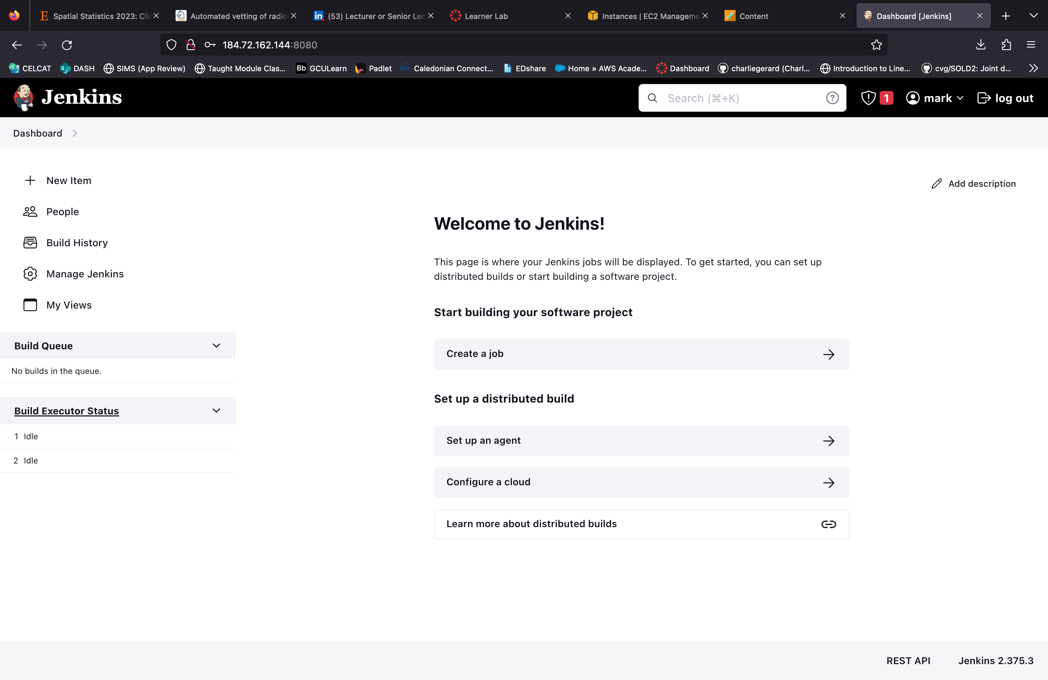


Figure : Jenkins Main Screen

# Building a java Project with Jenkins

Now that Jenkins is running it is time to use it to automate components of a Devops pipeline.

1. Create a new Git repository containing the below code in a file called HelloWorld.java and push it to a remote repository.

|  |
| --- |
| /\* This is a simple Java program.  FileName : "HelloWorld.java". \*/  class HelloWorld  {  // Your program begins with a call to main().  // Prints "Hello, World" to the terminal window.  public static void main(String args[])  {  System.out.println("Hello, World");  }  } |

Now you can create a Jenkins job to build this code.

1. Go back to Jenkins in the web browser and click on the ‘New Item’ button.
2. Call this job ‘job-01’ and select a Freestyle project as the base. Enter the name, select the project type and click OK at the bottom of the page.
3. Enter a description in the box provided and then move to the ‘Source Code Management’ tab.
4. Select the Git option and you will see a window like this:

A screenshot of a cell phone

Description automatically generated

Figure : Jenkins Source Code Management Option Page

1. Enter the HTTP URL of your git repository. Check your branch name matches what you expect. **It might need to be changed to ./main** rather than Master

Jenkins has to be able to authenticate the connection with GitHub or it will be unable to access the code you store there. To do this you first need to acquire an Access Token from GitHub.

1. Go to the GitHub website, click on the Profile icon at the top right and select Settings from the drop-down menu. Then select Developer Settings from the left of the screen and then Personal Access Tokens. **Use the Classic Token style.**
2. Click ‘Generate a new Token,’ give it the name Jenkins and set the Expiration to 7 Days as you will only need this particular token this week. Under Scopes check the ‘repo’ box and then Generate Token at the bottom of the page.
3. Copy the token to the box below for safe keeping

|  |
| --- |
|  |

1. Back in Jenkins, beside Credentials click the Add button and select Jenkins from the drop-down menu.
2. Enter your GitHub username in the Username field, paste the Token from above into the Password field and enter GitHub in the ID field. **Ensure that you do not copy any extra spaces or the Token will not work!**
3. Now scroll down to the ‘Build’ section, click ‘Add build step’ and select ‘Execute Shell’
4. In the text box, enter the commands to build and run your project as follows:

|  |
| --- |
| javac HelloWorld.java  java HelloWorld |

1. That is all of the configuration we require at present so click ‘Save’ at the bottom of the screen.
2. Click ‘Build Now’ on the left of the screen to run the build process we just configured.

You should now see the completed build in the Build History as shown below:

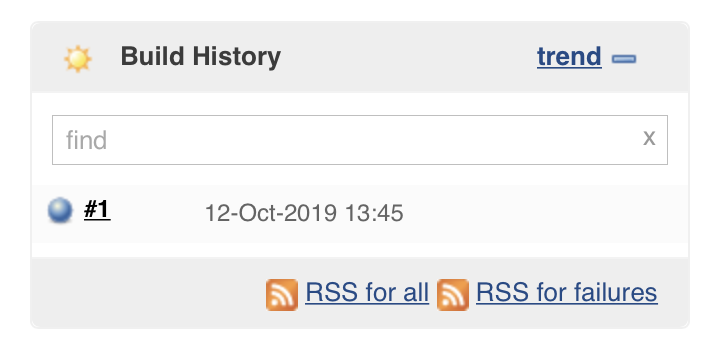


Figure : Jenkins Build History

The blue circle indicates that the build was successful. If this is red then the build has failed - check all of your settings again and if required speak to the lab supervisor.

1. Click on the build and then select ‘Console Output’ on the left of the screen. Read through this and try to understand the process that Jenkins has worked through.

# Automating your Jenkins Build

Now that you have tested your Jenkins project, you can configure Jenkins to run every time it detects a change to your remote repository.

1. Click ‘Back to Project’ and then ‘Configure’ to get back to the job settings.
2. Under ‘Build Triggers’ check the ‘Poll SCM’ box.

The Schedule box accepts input in the form:

Minute Hour DayOfMonth Month DayOfWeek

1. For now, we will have Jenkins check our repository for changes every minute by entering:

\* \* \* \* \*

1. Now save the changes.
2. Head back to your VM terminal and add a new print line under the ‘Hello World’ message.
3. Commit the changes and push them to your remote repository.
4. Go back to Jenkins, refresh the page and have a look at the ‘Build History’.

You should now see two builds because Jenkins has automatically checked for changes and carried out a build when you pushed your new code.

# Automated Testing

Now that your code is being built by Jenkins every time you commit changes it is time to add in another stage which tests your code as well.

For this Lab you will implement Static Code Analysis using a tool called SonarQube.

Fortunately, there is a Docker image available that will take care of the SonarQube server.

1. You can pull and run as follows:

|  |
| --- |
| docker run -d --rm --name sonarqube-container -p 9000:9000 sonarqube |

1. If you run the docker ps command we will see that now there is now a container running for SonarQube.
2. You can access SonarQube in the same was a Jenkins in our browser but by changing the port from 8080 to 9000. However, when on the university network, outbound requests to port 9000 are blocked. If you are on the university network, we need to start up a new virtual machine to access this port. **If you are working from home, please ignore the next section in the box and access the SonarQube console using a browser.**

**ONLY NEEDED FOR ACCESSING SONARQUBE ON CAMPUS**

1. In AWS, launch another EC2 instance. Here we need to configure it differently. We are going to give it an appropriate name and **select Windows as our operating system** and select a **t2.Medium instance**. We select **vockey** as our key pair. We then launch the instance
2. Once the instance is running, select the instance and click connect. We want to connect using RDP. Download the Remote Desktop file and open it to start the session
3. You need to provide a user name and password. The user name is Administrator and the password we get by clicking the get password button then by uploading our pem file.
4. Enter the username and password and wait for windows to load up. Once loaded, open Microsoft edge and visit the site  
   public\_ip\_address:9000
5. Once you have accessed SonarQube, click on ‘Log in’ and use admin for the username and password.

This server performs the actual code analysis but there is another component which must be installed in the Jenkins container which scans the code and sends the result to the server for processing.

1. Enter the following commands to download and unpack the SonarQube Scanner application:

|  |
| --- |
| cd /var/lib/jenkins  sudo mkdir sonarqube && cd sonarqube  sudo wget <https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-3.3.0.1492-linux.zip>  sudo apt install unzip  sudo unzip sonar-scanner-cli-3.3.0.1492-linux.zip  cd /var/lib/jenkins  sudo rm /var/lib/jenkins/sonarqube/sonar-scanner-3.3.0.1492-linux/jre/bin/java    sudo ln -s /usr/lib/jvm/java-11-openjdk-amd64/bin/java /var/lib/jenkins/sonarqube/sonar-scanner-3.3.0.1492-linux/jre/bin/java  sudo chown -R jenkins:jenkins sonarqube |

The final three commands above ensures that the SonarQube Scanner is using the same version of Java that Jenkins is using to build your code and the scanner will not work without this.

1. Head back to the Jenkins UI and from the main dashboard (accessible by clicking the top left Jenkins icon) select ‘Manage Jenkins’ -> ‘Plugins’ -> ‘Available’ and enter SonarQube in the ‘Find’ box at the top of the screen.
2. Select the SonarQube Scanner plugin as shown below and click ‘Install without restart’.

A screenshot of a social media post

Description automatically generated

Figure : SonarQube Plugin

You now need to connect Jenkins to our SonarQube Server.

1. Go to your SonarQube UI in your web browser and select the ‘Administration’ tab at the top of the page.

Click the ‘Configuration’ tab at the top left and select ‘Webhooks’ from the dropdown menu.

Click the ‘Create’ button and enter Jenkins for the name.

For the URL enter the following using your VM IP address:

http://ip\_address:8080

Click ‘create’ to finish this stage.

You also need to get an Authentication Token from SonarQube for Jenkins.

1. Click on the A box at the top right of the SonarQube UI and select ‘My Account’ from the dropdown menu.

Click on the ‘security’ tab and under ‘Generate Tokens’, enter the name Jenkins, select ‘User Token’ and click ‘Generate’.

You should now see a window like this:

A screenshot of a cell phone

Description automatically generated

Figure : SonarQube Authentication Token

1. Copy this token to the box below as you will need it in the next step and it cannot be recovered.

|  |
| --- |
|  |

1. Head back to Jenkins and go to ‘Jenkins’ -> ‘Manage Jenkins’ -> ‘System’

Find the ‘SonarQube Servers’ section and click ‘Add SonarQube’.

Use the name SonarQube and for the URL use (replacing your Public IP address):

http://ip\_address:9000

To add our authentication token select ‘Add’ -> ‘Jenkins’ and change ‘Kind’ to ‘Secret text’

In some instances, the add button does not respond when clicking. In this case go to the bottom of the of the screen and click save then go back into ‘Jenkins’ -> ‘Manage Jenkins’ -> ‘System’ and it should now work

Enter your Authentication Token in the ‘Secret’ box and ‘SonarQube’ in the ‘Description’ box and click ‘Add’.

I

Now select ‘SonarQube’ from the dropdown menu to use the Authentication Token.

Click ‘Save’ at the bottom of the screen to complete this step.

Finally, you have to configure the SonarQube Scanner plugin.

1. Go to ‘Jenkins’ -> ‘Manage Jenkins’ -> ‘Tools’ and find the ‘SonarQube Scanner installations’ section.

Click ‘Add SonarQube Scanner’ and uncheck the ‘Install Automatically’ box. Fill out the remaining fields as follows:

**Name:** SonarQube

**SONAR\_RUNNER\_HOME:** /var/lib/jenkins/sonarqube/sonar-scanner-3.3.0.1492-linux

Now click ‘Save’ to complete the setup.

You are now ready to add the SonarQube static analysis to our build process.

1. First you need to create a file called sonar-project.properties inside your github repository and add the following two lines of text to it:

sonar.projectKey=java-jenkins-sonar

sonar.sources=.

The project key just gives the project a name and the sources tells SonarQube where to look for files. In this case you have entered. so that it uses the local directory.

The benefit of this is that you can now add the properties file to version control along with your code.

1. You can add and commit this file but **do not push it** to the remote repository yet.
2. Go back to the main Jenkins Dashboard, click on our ‘job-01’ job and select ‘Configure’.

Under the ‘Build’ section where you have the commands from the last setup. From the dropdown menu, add a ‘Execute SonarQube Scanner’ stage and drag it to the top so that it is executed before the build steps you added earlier.

You only need to add one thing here which is the ‘Path to project properties’ which is the file we just created. As this file is included in your Git repository, it will be in the current directory (from the perspective of the Jenkins container) and you can therefore enter ./sonar-project.properties in the box.

Click ‘Save’ which will take you back to the ‘job-01’ project page.

1. Now you can push your latest changes to the remote repository and watch the Jenkins UI to watch the build being triggered.

This will take a bit longer than previously because SonarQube will run its analysis.

When it is finished, click on the SonarQube option on the left of the page which will take you to the SonarQube UI where you should see a screen like this:

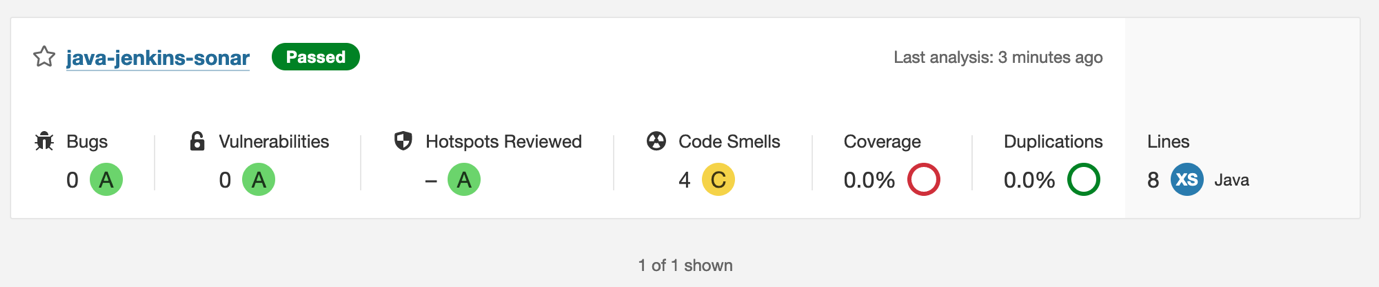


Figure : SonarQube Analysis results

Explore the SonarQube page to see what sort of tests have been carried out.

Congratulations!! You have completed the fourth lab in this module. You can now configure Jenkins to automatically build and test your java project when you push changes to your remote GitHub repository.

**PLEASE REMEMBER TO STOP OR TERMINATE YOUR VM AT THE END OF THIS LAB SO THAT YOU DO NOT USE UNNECCESARY CREDIT!!**