Cloud/DevOps Curriculum Series for Online Learning

Lab 4 – CI/CD with Jenkins on AWS with data-intensive web apps

Technical Contacts: Dr. Prasad Calyam (calyamp@missouri.edu); Dr. Songjie Wang (wangso@missouri.edu)

Release 1: May, 2021

Contents

1 Purpose of the lab

In this lab, you will learn how to deploy a Jenkins server on a VM instance on the Amazon AWS cloud platform using AWS EC2 services, and use the Jenkins server to automate the integration and deployment of an example web application, the KnowCOVID-19 application that was developed in the VIMAN lab at the University of Missouri.

2 References to guide lab work

Please use the links below to learn more about the Jenkins tool, and to help you understand the lab contents.

- Jenkins – *https://www.jenkins.io/*
- Docker – *https://docs.docker.com*
- Docker Hub – *https://hub.docker.com/*
- KnowCOVID-19 – *https://sites.google.com/view/knowCOVID-19*
- AWS essential concepts:
  - Amazon EC2 Linux Instances – *http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html*
  - Connecting to your Instance – (ordered by priority)
    - Linux SSH/SCP Client - *http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html*
    - Windows -
      - Putty – *http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html*
      - Transfer files from Windows to AWS Instance using PSCP – *http://the.earth.li/ sgtatham/putty/0.62/htmldoc/Chapter5.htmlpscp*

3 Overview

In this lab, we will learn about about continuous integration (CI) and continuous deployment (CD), and Jenkins, the leading open source automation server for adopting CI/CD.

What is Continuous Integration? Integration is the development practice that requires developers to integrate code into a shared repository several times a day each check and then verified by an automated build teams to detect problems early.

A continuous integration system usually involves a tool that keeps a mandatory version control system. Whenever a change to version control system is detected, the system would automatically build and test your application. If a built or test has certain problem, the system immediately notifies the developers to fix the issue right away.

Jenkins automation Jenkins is a continuous integration and pute server. It's used to manually periodically or automatically build software development projects. Jenkins provides hundreds of plugins to support building, deploying and automating any project. Jenkins offers a simple way to set up a continuous integration or continuous delivery (CI/CD) environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks. While Jenkins doesn't eliminate the need to create scripts for individual steps, it does give you a faster and more robust way to integrate your entire chain of build, test, and deployment tools than you can easily build yourself.

Why has Jenkin been so widely used?

Firstly it's very easy to use. The UI is simple, intuitive, and visually appealing. Overall, Jenkins as a whole has a relatively low learning curve. You can get started with Jenkin's in just a few minutes.

Also it has great extensibility. It's extremely flexible and easy to adapt to your own purposes. You can find thousands of open source plugins are available these plugins provide various functionalities, such as Virtion control systems, code quality metrics, build notifiers, UI customization, and etc.

4 How does Jenkins work?

4.1 Jenkins' Distributed Architecture

Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.

4.1.1 Master:

Your main Jenkins server is the Master. The Master's job is to handle:

- Scheduling build jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master instance of Jenkins can also execute build jobs directly.

4.1.2 Slave:

Slaves are executables that run on remote machines and are set up to build projects for a master.

- Jenkins runs a separate program called "slave agent" on slaves.
- When slaves are registered to a master, a master starts distributing loads to slaves.
- Node is used to refer to all machines that are part of Jenkins grid, slaves and master
- You can configure a project to always run on a particular Slave machine or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

4.2 Some important Jenkins' terminologies

4.2.1 Job/Project

These two terms are used interchangeably. They all refer to runnable tasks that are controlled/monitored by Jenkins.

4.2.2 Executor

Executor is a separate stream of builds to be run on a node in parallel. A Node can have one or more executors.

4.2.3 Build

A build is a result of one of the projects.

4.2.4 Plugin

A Plugin, like plugins on any other system, is a piece of software that extends the core functionality of the core Jenkins server.

5 Jenkins installation overview

Jenkins is typically run as a standalone application in its own process with the built-in Java servlet container/application server (Jetty). Jenkins can be installed on different platforms (such as Linux, Windows, etc) and set-up in different ways. It can be installed as a jar file deployed using Java, as a repository in Linux environments, as a war file deployed to a Servlet such as Apache Tomcat, as a Docker container either locally or on a public or private cloud, or as a SAAS hosted by the companies like Cloudbees, Azure, etc.

In this lab, we will deploy a Jenkins server using Docker container image that was built with Jenkins on a Linux VM instance hosted in the Amazon AWS cloud. Dockerized Jenkins can be downloaded as read-only "images" (or Docker

images), and run in Docker as a container.

The minimum hardware requirements for the lab server installation:

- 256 MB of RAM
- 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

Recommended hardware configuration for a small team:

- 4 GB+ of RAM
- 50 GB+ of drive space

6 AWS VM instance preparation

6.1 AWS EC2 overview

Figure 2 shows the instance architecture to be configured in this Lab. Using your AWS account, you will launch a virtual instance created in a new 'Volume' from an Amazon EBS-backed instance snapshot (called 'Root'), in order to access your reserved infrastructure resources over the Internet; you will need to create key pairs and secure it through a security group; all the infrastructure will be created in a specific zone.

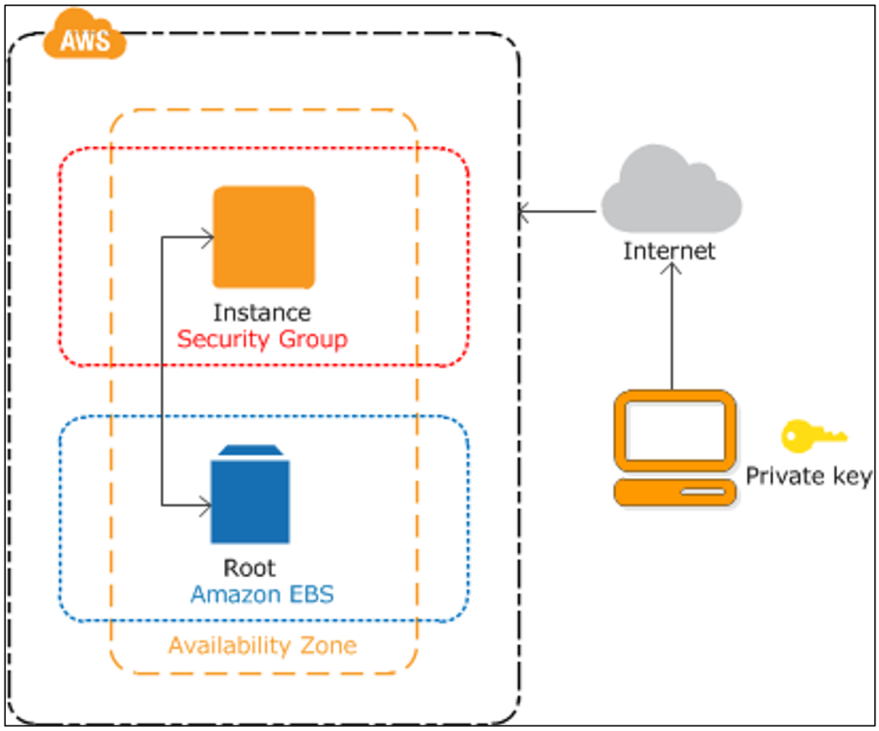

Figure 1: Overview of an AWS EC2 Instance

6.2 Create an EC2 VM instance

6.2.1 Access AWS EC2 Service

- Login to your Amazon AWS cloud account,
- then click 'Services'. Make sure to select the US East (N. Virginia) region in the top-right part of your screen and select EC2 service (Elastic Compute Cloud).

6.2.2 Manage Key Pairs

- In left menu, select "Key Pairs" under "Network Security" option.
- Click on "Create Key Pair" and create a key pair called 'key-ec2' and store it in a safe location, you will need this key for the future labs.

6.2.3 Manage Security Group

- Select "Security Groups" from the left menu, name a Security Group 'SG_EC2', add description and a SSH rule with 'anywhere' option selected in source field.
- Example of Security Group creation.

6.2.4 Launch an Instance

- In left menu, launch a new instance in the 'Instances' option Click on the 'Launch Instance' button and select the first Image of 'Ubuntu Server 20.04 LTS' on the list
- Select the t2.medium instance.
- Check to see that there are 7 steps in launching an instance and you are in the second step at the moment. Click on "Next: ..." button at the bottom right corner. Keep default values in the next configuration windows and continue until you get to the 'Add tags' option. Add 'Key' and 'Value' as shown in figure below and click on 'Next: Configure Security Group'.
- Select the 'Security Group' created previously and click on 'Review and Launch'.
- Once you click 'launch' you will be prompted to choose the key pair 'key-ec2' created previously. Acknowledge and click on "Launch Instances" button
- In a short time your new instance will be deployed and ready to be used. In the "Launch Status" page (the one you are currently in), click on "View Instances" button. Check the "Instance State" column. It should say running. If not, you can wait for a few seconds. It should not take long.

7 Jenkins installation using Docker image

Since we are installing and running the Jenkins server on a Docker container, we need to install Docker on the VM instance first. The requirement and steps to install Docker and Jenkins using Docker image can be accessed at: *https://www.jenkins.io/doc/book/installing/docker/*.

As a jar file deployed using Java. As a repository in Linux environments. As a war file deployed to a Servlet such as Apache Tomcat. As a Docker container either locally or on a public or private cloud. As a SAAS hosted by the companies like Cloudbees, Azure, etc.v

7.1 Login to the Jenkins server VM

Amazon AWS EC2 machine using Ubuntu as machine image requires you to login to the VM instance using the user "ubuntu". Login to the VM with the following command from your directory that has your SSH private key:

```
$ ssh -i Your_Key ubuntu@public_IP
```

7.2 Install Docker

Install Docker on your VM instance by following Docker installation guide at: *https://docs.docker.com/engine/install/*

Note

Note: The containers that we will be running will take 10GB of space. Make sure your VM instance has enough space available (20GB as we recommended), and then continue to setup the lab environment.

We will use the following commands to install Docker engine on the VM instance:

```
$ sudo su
$ apt-get update
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sh get-docker.sh
```

7.3 Install Jenkins from Docker Image

There are several Docker images of Jenkins available. The recommended Docker image to use is the Official jenkins/-jenkins image (accessible from the Docker Hub repository at *https://hub.docker.com/_/jenkins?tab=tagspage=1ordering=last_updated*). From the terminal on your VM instance, run the following:

7.3.1 Create a bridge network in Docker using the following docker network create command:

```
$ docker network create jenkins
```

7.3.2 In order to execute Docker commands inside Jenkins nodes, download and run the docker:dind Docker image using the following docker run command:

```
$ docker run \
  --name jenkins-docker \ ❶
  --privileged ❷\
  --network jenkins ❸\
  --network-alias docker ❹\
  --env DOCKER_TLS_CERTDIR=/certs ❺\
  --volume jenkins-docker-certs:/certs/client ❻\
  --volume jenkins-data:/var/jenkins_home ❼\
  --publish 2376:2376 ❽\
  docker:dind ❾\
  --storage-driver overlay2 ❿
```

❶ Specifies the Docker container name to use for running the image. By default, Docker will generate a unique name for the container.
❷ Running Docker in Docker currently requires privileged access to function properly. This requirement may be relaxed with newer Linux kernel versions.
❸ This corresponds with the network created in the earlier step.
❹ Makes the Docker in Docker container available as the hostname docker within the jenkins network.
❺ Enables the use of TLS in the Docker server. Due to the use of a privileged container, this is recommended, though it requires the use of the shared volume described below. This environment variable controls the root directory where Docker TLS certificates are managed.
❻ Maps the /certs/client directory inside the container to a Docker volume named jenkins−docker−certs as created above.
❼ Maps the /var/jenkins_home directory inside the container to the Docker volume named jenkins−data. This will allow for other Docker containers controlled by this Docker container's Docker daemon to mount data from Jenkins.
❽ (Optional) Exposes the Docker daemon port on the host machine. This is useful for executing docker commands on the host machine to control this inner Docker daemon.
❾ The docker:dind image itself. This image can be downloaded before running by using the command: docker image pull docker:dind.
❿ The storage driver for the Docker volume. See "Docker storage drivers" for supported options.

You can also copy the command from here:

```
$ docker run --name jenkins-docker --rm --detach \
  --privileged --network jenkins --network-alias docker \
  --env DOCKER_TLS_CERTDIR=/certs \
  --volume jenkins-docker-certs:/certs/client \
  --volume jenkins-data:/var/jenkins_home \
  --publish 2376:2376 docker:dind --storage-driver overlay2
```

7.3.3 Customise official Jenkins Docker image, by executing below two steps:

- Create a Dockerfile in a customized dir:

```
$ mkdir jenkins
$ cd jenkins
$ nano Dockerfile
```

In the Dockerfile, add the following:

```
FROM jenkins/jenkins:lts-jdk11
USER root
RUN apt-get update && apt-get install -y apt-transport-https \
       ca-certificates curl gnupg2 \
       software-properties-common
RUN curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -
RUN apt-key fingerprint 0EBFCD88
RUN add-apt-repository \
       "deb [arch=amd64] https://download.docker.com/linux/debian \
         $(lsb_release -cs) stable"
RUN apt-get update && apt-get install -y docker-ce-cli
USER jenkins
RUN jenkins-plugin-cli --plugins "blueocean:1.24.6 docker-workflow:1.26"
```

- Build a new docker customized Jenkins Docker image

```
$ docker build -t Your_Image_Name:version .
```

Example: *$ docker build -t my_jenkins:v1 .*

Note

Do not forget about the "." at the end of the docker build command. Also keep in mind that the process described above will automatically download the official Jenkins Docker image if this hasn't been done before.

7.3.4 Run the newly built Docker image for Jenkins

Replace the "Your_Image_Name:version" with your customized image name and version.

```
$ docker run --name jenkins-lab --rm --detach \
  --network jenkins --env DOCKER_HOST=tcp://docker:2376 \
  --env DOCKER_CERT_PATH=/certs/client --env DOCKER_TLS_VERIFY=1 \
  --publish 8080:8080 --publish 50000:50000 \
  --volume jenkins-data:/var/jenkins_home \
  --volume jenkins-docker-certs:/certs/client:ro \
  Your_Image_Name:version
```

7.4 Post-installation setup wizard

Now you have both the Jenkins container and the dind container running, we will now setup the Jenkins through its web portal by following the steps below. This setup wizard takes you through a few quick ”one-off” steps to unlock Jenkins, customize it with plugins and create the first administrator user through which you can continue accessing Jenkins.

7.4.1 Unlocking Jenkins

When you first access a new Jenkins instance, you are asked to unlock it using an automatically-generated password.

- Browse to http://Host_VM_IP:8080 (Replace the Host_VM_IP with the public IP (IPv4) of the AWS EC2 instance). If you configured the Jenkins with a different port, change 8080 to the actual port you are using.

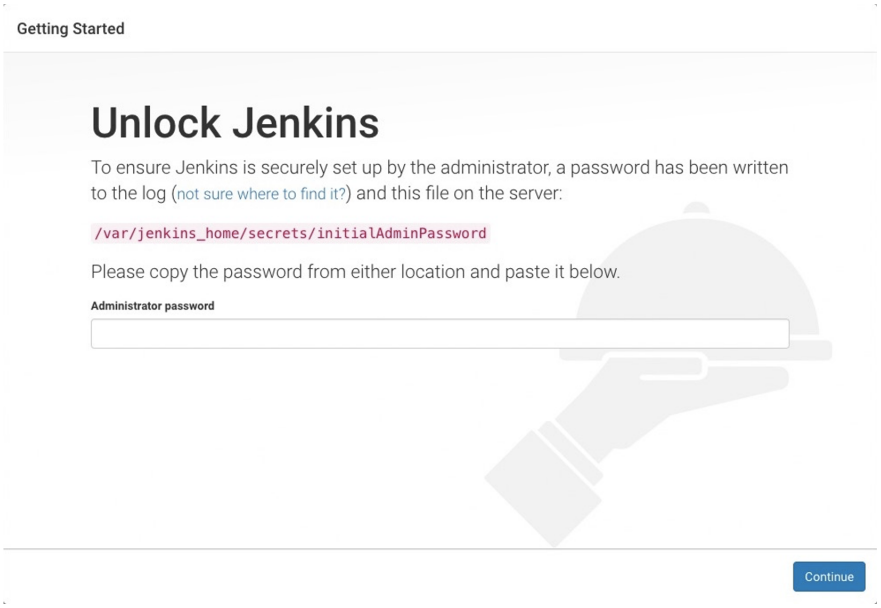

Figure 2: Overview of an AWS Instance

- From the Jenkins console log output, copy the automatically-generated alphanumeric password (between the 2 sets of asterisks).

Note

Run the command:

```
$ docker exec ${CONTAINER\_NAME} cat /var/jenkins_home/secrets/initialAdminPassword
```

to print the password in the console without having to exec into the container. Replace the CONTAINER_ID or CONTAINER_NAME with your actual ID or name.
For example:

```
$ docker exec jenkins-lab cat /var/jenkins_home/secrets/initialAdminPassword
```

- On the Unlock Jenkins page, paste this password into the Administrator password field and click Continue.

Note

You can always access the Jenkins console log from the Docker logs (above).
The Jenkins console log indicates the location (in the Jenkins home directory) where this password can also be obtained. This password must be entered in the setup wizard on new Jenkins installations before you can access Jenkins's main UI. This password also serves as the default admininstrator account's password (with username "admin") if you happen to skip the subsequent user-creation step in the setup wizard.

7.5 Customizing Jenkins with plugins

After unlocking Jenkins, the Customize Jenkins page appears. Here you can install any number of useful plugins as part of your initial setup.
Click one of the two options shown:

- Install suggested plugins - to install the recommended set of plugins, which are based on most common use cases.
- Select plugins to install - to choose which set of plugins to initially install. When you first access the plugin selection page, the suggested plugins are selected by default.

Note

If you are not sure what plugins you need, choose Install suggested plugins. You can install (or remove) additional Jenkins plugins at a later point in time via the Manage Jenkins >Manage Plugins page in Jenkins.

7.6 Creating the first administrator user

After customizing Jenkins with plugins, Jenkins asks you to create your first administrator user.

7.6.1 When the Create First Admin User page appears, specify the details for your administrator user in the respective fields and click Save and Finish.

7.6.2 When the Jenkins is ready page appears, click Start using Jenkins

Note

This page may indicate Jenkins is almost ready! instead and if so, click Restart.
If the page does not automatically refresh after a minute, use your web browser to refresh the page manually.

7.6.3 If required, log in to Jenkins with the credentials of the user you just created and you are ready to start using Jenkins!

8 Jenkins UI Overview

Once you finish setting up the Jenkins plugin and user, we are at the main landing page.

8.1 Jenkin's web interface

Go to the Jenkins web URL, then type admin and the password you set in the last lecture to login into Jenkin web interface.

On the left hand side there are some menu items, as well as build queue and build executor status.

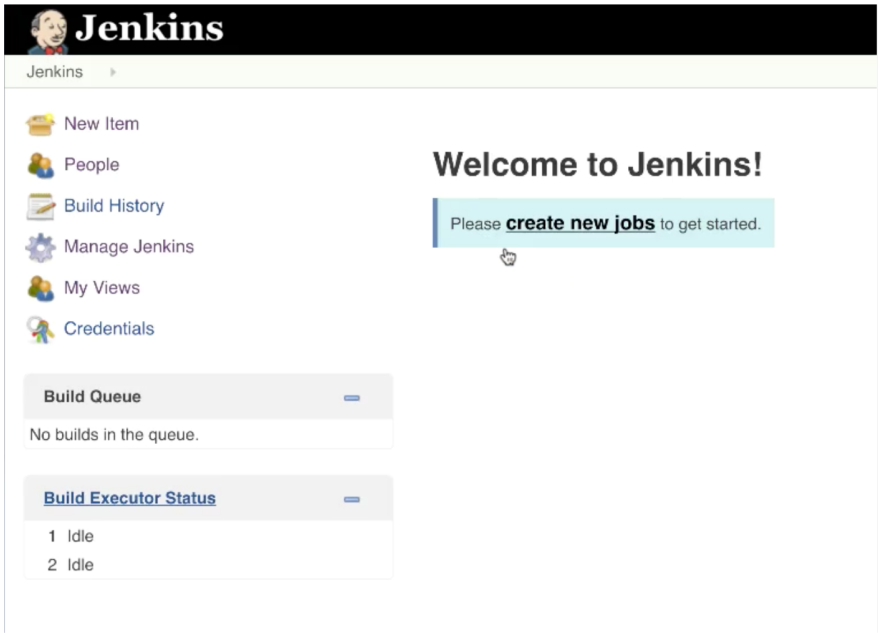

Figure 3: Jenkins landing page

- New Item: the item refers to a job or a project. It helps us to create a new job.
- People: this menu item manages users within Jenkins.
- Build History: shows us a history of builds in this Jenkins instance. It includes the builds that were executed not only on the master node but also on the slave nodes.
- Manage Jenkins: This configure System item can help us to configure global settings and paths, such as the path to Maven and Java, security settings, system information and system logs, etc.
- My views: This displays a private view which provides a possibility for every user to have its own filter of Jenkins Jobs.
- Credentials: which is responsible for managing all the global credentials in Jenkins.
- The Build Queue status here shows all the builds that have been triggered.
- Next on the Build Executor section displays the current build executors on this Jenkins instance.

9 Create Your First Simple Jenkins Job

Now We will start creating our first Jenkins job. We are now at the landing page, there aren't any jobs created, you have a nice little inviting message to create a new job. Let's create a job now!

9.1 Create a job

- From the landing page, click on create new jobs. Give the new job a meaningful name, e.g. in this demo we use "first-job". Try to avoid using spaces in the name as it will be used to create links and directories, which with spaces can be a problem.
- Next it is to choose the type of project we want to create. For the first job, we will create a Freestyle project. Freestyle build jobs are general-purpose build jobs, which provides a maximum of flexibility so that you can tailor it to your needs. Select that item, and then click on the "OK" button.

9.2 Job configuration

After you select the type of the project, you will be redirected to the job configuration page, where you can see a list of sections at the top of the configuration. We describe each of these sections as follows:

- **General** Next let's input the project description. The project description will go on the project home page. It provides an overview of the build job's goals and context. Just type this is our first Jenkins job. Now we are at the job configuration page.

  As you see, the job configuration page has been divided into several sections. The first section is some basic information such as the name and description of the project which we have already filled in. And some other options about how and where the build job should be executed.

  The other options are more technical. You can disable old builds. Build jobs can consume a lot of disk space, especially if you store the build artifacts. Even without artifacts, keeping a record of every build job consumes additional disk space and memory. The Discard Old Builds option lets you limit the number of builds you record in the build history. You can either tell Jenkins to only keep recent builds or to keep no more than a specified number of builds.

  You also have the option to disable the build. A disabled build will not be executed until you enable it again. Using this option when you create a new build job is quite rare. But this option is very handy if you want to temporarily suspend a build during maintenance work or major refactoring.

- **Source Control Management** section, which defines the source of the code which will be built by our job. Jenkins supports CVS and SVN out of box. Our first Jenkins job won't checkout any source code. So just leave None as default.

- The next section is the **Build Trigger** section. We can define multiple triggers for this job. If we don't specify any triggers, the job can only be triggered manually. In our first job, we will trigger this job manually.

- And next is the **Build** section, where we define the actual build steps. Builds steps are the basic building blocks for the Jenkins free style builds process. Builds steps tell Jenkins exactly, how we want our build project. We can define as many steps as we want within a job. Here we click add build step. Then we select execute shell. If you are on a Windows system, you probably want to select "Execute Windows batch command". Then we just echo hello Jenkins which should print out hello Jenkins message for us.

- The last section is the **Post Build** section where we can define some post build processing steps. These steps will execute whether the build is successful or not, usually we use the post build jobs to define our notifications to send out emails to the development team. Jenkins can also collect information out of the build such as archiving the artifacts, recording javadoc and test results. We won't do any post build actions in this demo. So let's leave it empty. Now we are done with configuring our first job, we can click on the "save" button. Great! After saving the configuration for our first job, let's go back to our landing page. The new job appears on our landing page.

10 Add GitHub Support

In the previous section we created and triggered a simple Jenkins job by running a shell script. But the Jenkins job doesn't do much besides printing out the hello Jenkins message. Now we will set up a more practical maven based built this Jenkin's job would check out the source code from github, compile the code, run the test, and package the application. The project we're going to build as a very simple maven project. We need to install git on our VM instance and then install git plugins on Jenkins.

10.1 Git installation

You VM instance should already have git installed. Check the git installation with the following command:

```
$ git --version
```

If it says git can not be found, then install git with the following command:

```
$ sudo apt-get install git
```

10.2 Configure Git Plugin

On your Jenkins landing page, click Manage Jenkins, then click Manage Plulgins. Click on the installed tab and type ”github” in the filter and look for ”GitHub plugin” in the shown lists. If you see the check mark in front of the ”GitHub plugin”, then you already have github linked in Jenkins. If not, then you need to install the plugin as follows:

Click the available tab. Go to the filter search and type in Github. Locate the ”github plugin” and then click the check box next to it. There are two buttons to install the plugins: install without restart, and download now and install it after restart. We will choose the ”install without restart”. Click on install without restart and Jenkins will start the installation process. Once installation is done, you should see that github plugin is installed, and you can see the version of it, which in our example is 1.33.1.

11 Add Java/Maven Support

11.1 Java installation

Your VM instance should have Java installed. Download and install Java jdk1.8 on the VM instance using the following commands (skip ”sudo” if you are running with root user):

```
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk
$ java -version
```

If you see output similar to below, your VM should have Java jdk installed correctly.

```
openjdk version "1.8.0_242"
OpenJDK Runtime Environment (build 1.8.0_242-b09)
OpenJDK 64-Bit Server VM (build 25.242-b09, mixed mode)
```

11.2 Maven installation

Next, we will download and install Maven on the VM instance using the following commands (skip ”sudo” if you are running with root user):

```
$ wget
  https://apache.osuosl.org/maven/maven-3/3.8.1/binaries/apache-maven-3.8.1-bin.zip
$ sudo apt-get install unzip
$ unzip apache-maven-3.8.1-bin.zip
$ cd apache-maven-3.8.1
$ export PATH=$PATH:/root/apache-maven-3.8.1/bin
```

11.3 Java/Maven/Git configure – In progress

11.4 Adding Docker container support – In progress

12 Homework Problems

1) Describe pod, deployment, service, and what they are used for.
2) Describe the different between ClusterIP, NodePort and Load Balancing, and what they are best suited for.
3) Describe the different between port, target port and node port, and what they are best used for.
4) Describe the difference between Ingress and port-forwarding, and what they are best used for.

13 Lab Report

1) Submit the required screen shots in the lab steps
2) Short assay answers to the homework problems