CI/CD Principles and Practices

Lecture: What Is CI/CD

In this lesson, we will be showing how software was developed in the past and see the issues that it caused. Teams were separated, and there was a lack of effective collaboration. This is where Continuous Integration/Continuous Delivery (CI/CD)comes in. Using CI/CD, development cycle times are shortened, and teams work together more closely. Code commits happen often, and code testing is done on each commit to ensure that the build is always in a state in which it could be deployed.

Lecture: Continuous Deployment

In this lesson, we will take the next logical step in the CI/CD process, and move into Continuous Deployment. This is the process of taking the tested code and deploying it into production environments in an automated fashion. We will talk about smoke testing as well as checkpoints, which are a part of Jenkins Pipelines, that allow for a manual confirmation prior to the automated deployment.

Lecture: Test Driven Development

In this last lesson in this section, we will be looking at Test Driven Development (TDD). We will also mention some of the different types of tests that are performed during the software development cycle. This is the last lesson in the concepts portion of this course, but it is one of the most important as it speaks to the process that needs to be followed in order to get the most out of Jenkins.

Jenkins Installation

Lecture: Running Jenkins War File

In this lesson, we will be running Jenkins from a war file. The link to this file is available from the official documentation, and this is one of the ways in which you can get Jenkins running on your system. This is especially useful if you are not wanting to install Jenkins as a service, and just need to test functionality on a local system.

The commands used in this lesson:

```
sudo apt install -y openjdk-8-jre

wget http://mirrors.jenkins.io/war-stable/latest/
jenkins.war

java -jar jenkins.war --httpPort=80 --prefix=/dashboard
```

Lecture: The Installation Wizard

In this lesson we will be installing Jenkins. We will also be talking about the steps that are required to get the initial setup complete so that we can start using it. The installation wizard, also called the initial setup wizard, can be used to select and install an initial set of plugins, either from a curated list that is provided or from a base set of available plugins that you can select. Once we have this step completed, we can move on to explore the functionality of Jenkins and building some software.

Commands used in this lesson:

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key |
sudo apt-key add -

sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable
binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update
sudo apt-get install -y openjdk-8-jre jenkins
```

Lecture: Alternate Configurations

In this lesson, we will not only be talking about some of the places that Jenkins can be installed, but we will be looking specifically at creating a Jenkins installation in Docker. The concepts that are presented in this lesson can be found here:

```
https://jenkins.io/doc/book/installing/
```

The commands that are used in this lesson to create the docker instance of Jenkins are:

```
sudo apt install -y docker.io
sudo usermod -aG docker cloud_user
```

Log out and then back in to apply the group change, then continue:

```
docker network create jenkins
docker volume create jenkins-docker-certs
docker volume create jenkins-data
docker container 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
docker container run --name jenkins-blueocean --rm --detach
\
  --network jenkins --env DOCKER_HOST=tcp://docker:2376 \
  --env DOCKER_CERT_PATH=/certs/client --env
DOCKER_TLS_VERIFY=1 \
  --volume jenkins-data:/var/jenkins_home \
  --volume jenkins-docker-certs:/certs/client:ro \
  --publish 8080:8080 --publish 50000:50000 jenkinsci/
blueocean
```

For the unlock password, run:

```
docker logs jenkins-blueocean
```

Jenkins Graphical User Interface

Lecture: Exploring the GUI

In this lesson, we will be taking a look at the default Jenkins interface. This is a quick lesson on basic navigation that will get you familiar with the parts of the interface, and where you can locate the information that you need about your builds. We will dig a bit deeper into parts of the interface as we progress in this section.

Lecture: System Configuration

In this lesson, we will be taking a look at the system configuration menu in Jenkins. The functionality in this menu will serve as the base for all of the menus that are available in Jenkins. This will help us as we move forward, since it will show how to use item descriptions like an internal help system, and how to get to the settings that are sometimes not readily visible in the menus.

Lecture: Job Configuration

In this lesson, we will look at the job configuration for a freestyle job. This is a quick overview of the options that are in the configuration menu, to get us familiar with them was we move forward and begin to create jobs.

Jenkins Plugins

Lecture: Managing Plugins

In this lesson, we will be looking at using the plugin manager to determine which plugins are installed and enabled. We will see where in the configuration menu the plugin manager is located, and we will see the impact of disabling a plugin on the available configuration for the Jenkins server.

Lecture: Updating Plugins

In this lesson, we will look at the process of updating a plugin via the plugin manager. We will see how a managed plugin can be downgraded to a previously installed version and also take a look behind the scenes to see what is going on in the plugins folder when a plugin is upgraded to a newer version. We will also be looking at the documentation for a plugin to do some detective work on what the plugin might be called in the file system as sometimes the display name and the file names do not match.

Lecture: Adding Plugins

In this lesson, we will be looking at the process of adding plugins. We have seen how to use the available plugins tab to install plugins in the normal way; now, we will look at the *Advanced* tab and how to leverage the Jenkins plugin manager to resolve dependencies for non-standard plugins that are provided as .hpi files. We will also see how to manually install a plugin directly into the Jenkins masters file system, and how this can cause dependency resolution errors that require manual intervention to correct.

Freestyle Jobs

Lecture: Basic Job Structure

In this lesson, we will walk through the most basic freestyle job. We'll look at what happens with the project workspace as well as where logs for each build are located. We will also see where files that a build creates get placed and how we can leverage workspace management to ensure that we have clean builds without overlapping artifacts.

Lecture: Parameters

In this lesson, we will be looking at the ways we can leverage parameters in our jobs. Parameters can be thought of as command-line arguments for our builds. We will see how the values of parameters are made available to our builds.

Code used in this lesson:

```
#!/bin/bash

mapfile -t <remote.txt
echo "Hello" $name " This is ${MAPFILE[0]}"
```

Lecture: Notifications

In this lesson, we will look at how to configure notifications and where to enable them in your builds. Since we are using cloud playground servers, setting up actual mail relays will not be possible, but we will look at the extended mail notification plugin and the standard email plugin as well as how to provide the ability to leverage alternative communication channels such as instant messaging and SMS.

Agents and Distributed Builds

Lecture: Setting up a Build Agent

In this lesson, we will be setting up a remote build agent using an Ubuntu server on our cloud playground. This is the first step in distributing a build, and once we have the build agent setup completed, we will move forward with creating distributed builds.

In the process of setting up the build agent the following commands are used.

setting up the jenkins user:

```
ls -l /var/lib
sudo mkdir /var/lib/jenkins

ls -l /var/lib | grep jenkins
sudo useradd -d /var/lib/jenkins jenkins
sudo chown jenkins:jenkins /var/lib/jenkins
```

Generating and setting ssh keys

```
ssh-keygen

sudo mkdir /var/lib/jenkins/.ssh
cat ./.ssh/id_rsa_pub
sudo vim /var/lib/jenkins/.ssh/auhorized_keys
```

installing java

```
sudo apt-install openjdk-8-jre-headless
```

getting the private key for the user

```
cat ./.ssh/id_rsa
```

correcting the known hosts issue once we have ssh'd from master to agent

```
sudo cp ./.ssh/known_hosts /var/lib/jenkins/.ssh
```

Lecture: Distributing a Build

In this lesson, we will learn how to force our projects to use the newly created build agent. We will see how to explicitly ensure that the project builds on a specific agent, as well as how to ensure that the projects do not build on the master. Only builds that need to build on the master should build there, otherwise your builds should take place on your build agents.

Lecture: Monitoring Build Agents

In this lesson, we will be taking a quick look at how to leverage plugins to monitor your Jenkins installations. In this case, we will be looking at the monitoring plugin that used Java melody to create monitoring pages that can be used to provide data on the running processes in your Jenkins build environments. This can allow you to do performance tuning on the environments and make adjustments as needed.

Source Code Management, Build Tools, and Test Reports

Lecture: Leveraging SCM in Builds

In this lesson, we will see how to add Source Code Management (SCM) into our builds. We will see how to add a repository and talk about where files are placed when Jenkins clones the repository to the local workspace. Once we have done this, and the build is working when manually run, we will go the next step and configure a webhook in GitHub that will automatically trigger a build when changes are committed to the SCM repository. This is another of the fundamental processes in CI/CD that make it possible to have the code in a ready state.

Lecture: Configuring Build Tools

In this lesson, we talk about build tools and how they are configured in Jenkins. While we will be talking about build tools, in general, we will not be talking about any specifics of any build tool as that is something that is dependant on the build tool that you are using. What I will be talking about, though, is what Jenkins can and cannot configure for you and then following up with how you will need to resolve dependencies for your build environments.

Lecture: Testing and Test Reports

In this lesson, we will be working with the project from the last lesson and taking it a step further by configuring the build to process the Junit test reports, allowing Jenkins to process the report and indicate the correct status for the build. This allows us to include testing as part of our process and validate that the code is in a deployable state. This is one of the key pillars of the CI/CD process.

Upstream, Downstream, and Triggers

Lecture: Artifacts and Fingerprints

In this lesson, we will discuss artifacts, which are files that are produced by builds, and we will talk about fingerprints, which are pieces of information about an artifact. We will see where they are stored on the server, and how a fingerprint is tied to an artifact by its MD5 hash value. We will also see what happens when a build is deleted on both the master and the build agent.

Lecture: Linking Jobs

In this lesson, we will be creating two jobs and then leveraging the

Parameterized Trigger plugin to pass information from one job to another, then start the build on the second job. We will discuss environment variable scope and describe 'upstream' and 'downstream' jobs. This is a fundamental concept in a pipeline, as some jobs that create dependencies for downstream jobs might need to produce an up-to-date final result that reflects the upstream changes in the finished product.

Lecture: Automating Jobs

In this lesson, we will be looking at the use of a Jenkinsfile to automate the build process. The addition of the Jenkinsfile to the code repository for a project allows you to instruct Jenkins to scan the repository and discover the configuration for the build. This means that your project can leverage source code management to maintain its own configuration and benefit from change tracking. The project that will be used to demonstrate this concept is the multibranch pipeline project. We have not discussed pipelines yet but we will keep it simple and easy to follow.

This is the Git repository used in this lesson: https://github.com/linuxacademy/content-cje-prebuild

Jenkins on the Command Line

Lecture: Using the Jenkins API

In this lesson, we will be taking a look at how to interface with Jenkins from the terminal. In this case, we will be using the Jenkins API to run a build and modify a build config from the terminal.

Commands used in this lesson:

```
curl -u username:apiToken http://jenkinsMasterURL/job/
APIJOB/build
curl -u username:apiToken http://jenkinsMasterURL/job/
APIJOB/config.xml
curl -X POST -u username:apiToken "http://jenkins/job/
APIJOB/config.xml" -d "@apiconfig.xml"
```

Lecture: Using the Jenkins CLI

In this lesson, we will be looking at the Jenkins-CLI. This is provided as a `jar` file that is located in the `jnlpJar` directory on the Jenkins Master. Much like the API, the documentation for the CLI is accessed by adding /cli to the end of the Jenkins master URL. The CLI also uses the API token that can be generated for your user from the user configuration section, under your user account on the master.