**eclarative Pipelines in Jenkins**

**Introduction**

In this hands-on lab we will be looking at Declarative Pipelines. Declarative style pipelines limit what is available to the user with a more strict and pre-defined structure, making it an ideal choice for simpler continuous delivery pipelines.

We'll be building the Pipeline (Declarative) and ensuring that it has gone through the build stages as defined and provided a satisfactory Jenkins job result.

**The Scenario**

To lessen the reliance on your team having to know Groovy syntax for Scripted Jenkins Pipelines, and to simplify the Pipeline syntax for Pipelines, your company has opted to move to Declarative Pipelines in Jenkins.

In addition, your boss also wants see to if dependencies installed directly onto Jenkins servers (such as gcc compiler for compiling source code, and git for cloning/pulling down source code) can be cut down.

To reduce having to deal with these dependencies, you are thinking of running your whole Jenkins Pipeline job inside a Docker container which has gcc and git utilities pre-installed, and then saving any artifacts produced.

You have a rough idea of how you would tackle this: 1. Ensure that in your Declarative Pipeline you have a Docker agent which pulls down a Docker image and executes all steps within it. The Docker image with gcc and git pre-installed is named linuxacademycontent/jenkins\_pipelines. 2. Have three stages, at least: 1. A **Preparation** stage where you clone code from a Git repo: [*https://github.com/linuxacademy/content-pipelines-cje-labs.git*](https://github.com/linuxacademy/content-pipelines-cje-labs.git) 2. A **Build** stage where actual code is assembled, linked, and built, providing an executable binary. - Sample build command: gcc --std=c99 -o mario content-pipelines-cje-labs/lab1\_lab2/mario.c 3. An **Archiving** stage where you save or preserve the output of your **Build** stage. Hint: Use an archiveArtifacts step and the output (-o mario). mario is the file name.

**Note:** All steps should take place inside a Docker agent which has gcc and git pre-installed.

With the above in mind, you now have to come up with a Jenkins Declarative Pipeline to automate the task at hand.

**Logging In**

A Jenkins server will be made available. Once the lab interface provides details of the public ip address, please use the default Jenkins port in your browser to access it:

http://<Public\_IP\_of\_Instance>:8080/

You will be directly logged into the Jenkins instance, no password authentication is required.

**Jenkins UI**

Once you access the Jenkins landing page, click **New Item** on the left side of the screen On the next page, name the new item, something like **MyDeclarativePipeline**, then click on **Pipeline**. Then click **OK** to proceed with configuring the job.

**Configuring a Jenkins Pipeline Job (Declarative Pipeline)**

On the next page, scroll down to the *Pipeline* section and ensure that **Pipeline Script** is selected in the *Definition* drop-down. An input area should be visible. Copy the following body of text (which is basically in Groovy DSL format, a format understood by Jenkins) into the input area:

pipeline {

agent {

docker { image 'linuxacademycontent/jenkins\_pipelines' }

}

stages {

stage('fetch') {

steps {

sh 'git clone https://github.com/linuxacademy/content-pipelines-cje-labs.git'

}

}

stage('build'){

steps{

sh 'gcc --std=c99 -o mario content-pipelines-cje-labs/lab1\_lab2/mario.c'

}

}

stage('archive'){

steps{

archiveArtifacts artifacts: 'mario'

}

}

}

}

Leave everything else at the default settings and click on **Save**, which will take you to the control page for the job you just configured.

**Running/Building Your Jenkins Declarative Pipeline Job**

To kick off a build, click on **Build Now**. The build should start, and you should see a build number and a progress bar in the *Build History* box, on the left of your screen

Click on the build number under *Build History* (**#1** for example), then select **Console Output** from the dropdown to watch the job's progress in real time.

Once that's finished running, click back on our pipeline's name (in the menu bar at the top of the screen). We'll see that all of our stages completed, and a mario artifact.

**Conclusion**

Congratulations on completing this hands-on lab!