Hands-on Jenkins-03 : Java and Maven Jobs in Jenkins

Purpose of the this hands-on training is to learn how to install Java and Maven to Jenkins Server and configure Maven/Java Jobs.

Learning Outcomes

At the end of the this hands-on training, students will be able to;

- install and configure Maven,
- create Java and Maven jobs
- create job DSL

Outline

- Part 1 - Install Java, Maven and Git packages
- Part 2 - Maven Settings
- Part 3 - Creating Package Application - Free Style Maven Job
- Part 4 - Configuring Jenkins Pipeline with GitHub Webhook to Build the Java Code
- Part 5 - Configuring Jenkins Pipeline with GitHub Webhook to Build the a Java Maven Project
- Part 6 - Jenkins Job DSL

Part 1 - Install Java, Maven and Git packages

- Connect to the Jenkins Server
- Install Java

```
sudo yum update -y
sudo amazon-linux-extras install java-openjdk11 -y
sudo yum install java-devel
```

- Install Maven

```
sudo su
cd /opt
rm -rf maven
wget https://dlcdn.apache.org/maven/maven-3/3.8.4/binaries/apache-maven-3.8.4-
bin.tar.gz
tar -zxvf $(ls | grep apache-maven-*-bin.tar.gz)
rm -rf $(ls | grep apache-maven-*-bin.tar.gz)
```

```
sudo ln -s $(ls | grep apache-maven*) maven
echo 'export M2_HOME=/opt/maven' > /etc/profile.d/maven.sh
echo 'export PATH=${M2_HOME}/bin:${PATH}' >> /etc/profile.d/maven.sh
exit
source /etc/profile.d/maven.sh
```

- Install Git

```
sudo yum install git -y
```

Part 2 - Maven Settings

- Open Jenkins GUI on web browser
- Setting System Maven Path for default usage
- Go to `Manage Jenkins`
  - Select `Configure System`
  - Find `Environment variables` part,
  - Click `Add`
    - for `Name`, enter `PATH+EXTRA`
    - for `Value`, enter `/opt/maven/bin`
- Save
- Setting a specific Maven Release in Jenkins for usage
- Go to the `Global Tool Configuration`
- To the bottom, `Maven` section
  - Give a name such as `maven-3.8.4`
  - Select `install automatically`
  - `Install from Apache` version `3.8.4`
- Save

Part 3 - Creating Package Application - Free Style Maven Job

- Select `New Item`
- Enter name as `Package-Application`
- Select `Free Style Project`
- For Description : `This Job is packaging Java-Tomcat-Sample Project and creates a war file.`

- At `General Tab`, select Discard old builds, `Strategy` is `Log Rotation`, and for `Days to keep builds` enter `5` and `Max # of builds to keep` enter `3`.

- From `Source Code Management` part select `Git`

- Enter `https://github.com/Eser-U/java-tomcat-sample.git` for `Repository URL`.

- Go to the web browser and check the branch name of the git project `https://github.com/Eser-U/java-tomcat-sample.git`. Most of the time, deafult branch is `master` but there may be some exceptions. Enter the branch name (`main`) to the `Branch Specifier (blank for 'any')`.

- It is public repo, no need for `Credentials`.

- At `Build Environments` section, select `Delete workspace before build starts` and `Add timestamps to the Console Output` options.

- For `Build`, select `Invoke top-level Maven targets`

  - For `Maven Version`, select the pre-defined maven, `maven-3.8.4`
  - For `Goals`, write `clean package`
  - POM: `pom.xml`

- At `Post-build Actions` section,

  - Select `Archive the artifacts`
  - For `Files to archive`, write `**/*.war`

- Finally `Save` the job.

- Select `Package-Application`

- Click `Build Now` option.

- Observe the Console Output

Part 4 - Configuring Jenkins Pipeline with GitHub Webhook to Build the Java Code

- To build the `java` code with Jenkins pipeline using the `Jenkinsfile` and `GitHub Webhook`, we will leverage from the same job created in *** Hands-on-02 Part 2*** (named as `pipeline-with-jenkinsfile-and-webhook`). To accomplish this task, we need;

  - a java code to build

  - a java environment to run the build stages on the java code

  - a Jenkinsfile configured for an automated build on our repo

- Create a java file on the `pipeline-project` local repository(we have created in *** Hands-on-02 Part 2***), name it as `Hello.java`, add coding to print `Hello from Java` and save.

```
public class Hello {

    public static void main(String[] args) {
        System.out.println("Hello from Java");
    }
}
```

- Since the Jenkins Server is running on Java platform, we can leverage from the already available java environment.
- Update the `Jenkinsfile` with the following pipeline script, and explain the changes.

```
pipeline {
    agent any
    stages {
        stage('build') {
            steps {
                echo 'Compiling the java source code'
                sh 'javac Hello.java'
            }
        }
        stage('run') {
            steps {
                echo 'Running the compiled java code.'
                sh 'java Hello'
            }
        }
    }
}
```

- Commit and push the changes to the remote repo on GitHub.

```
git add .
git commit -m 'updated jenkinsfile and added Hello.java'
git push
```

- Observe the new built triggered with `git push` command on the Jenkins project page.
- Explain the role of java environment, `Jenkinsfile` and GitHub Webhook in this automation.

Part 5 - Configuring Jenkins Pipeline with GitHub Webhook to Build the a Java Maven Project

- To build the `java maven project` with Jenkins pipeline using the `Jenkinsfile` and `GitHub Webhook`. To accomplish this task, we need;
    - a java code to build

    - a java environment to run the build stages on the java code
    - a maven environment to run the build stages on the java code
    - a Jenkinsfile configured for an automated build on our repo

- Create a public project repository `jenkins-maven-project` on your GitHub account.
- Clone the `jenkins-maven-project` repository on local computer.
- Copy the files given within the hands-on folder hello-app and paste under the `jenkins-maven-project` GitHub repo folder.
- Go to your Github `jenkins-maven-project` repository page and click on `Settings`.
- Click on the `Webhooks` on the left hand menu, and then click on `Add webhook`.
- Copy the Jenkins URL from the AWS Management Console, paste it into `Payload URL` field, add `/github-webhook/` at the end of URL, and click on `Add webhook`.

```
http://ec2-54-144-151-76.compute-1.amazonaws.com:8080/github-webhook/
```

- Go to the Jenkins dashboard and click on `New Item` to create a pipeline.
- Enter `pipeline-with-jenkinsfile-and-webhook-for-maven-project` then select `Pipeline` and click `OK`.
- Enter `Simple pipeline configured with Jenkinsfile and GitHub Webhook for Maven project` in the description field.
- Put a checkmark on `GitHub Project` under `General` section, enter URL of the project repository.

```
https://github.com/<your_github_account_name>/jenkins-maven-project/
```

- Put a checkmark on `GitHub hook trigger for GITScm polling` under `Build Triggers` section.
- Go to the `Pipeline` section, and select `Pipeline script from SCM` in the `Definition` field.
- Select `Git` in the `SCM` field.
- Enter URL of the project repository, and let others be default.

```
https://github.com/<your_github_account_name>/jenkins-maven-project/
```

- Click `apply` and `save`. Note that the script `Jenkinsfile` should be placed under root folder of repo.
- Create a `Jenkinsfile` with the following pipeline script, and explain the script.

- For native structured Jenkins Server

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'mvn -f hello-app/pom.xml -B -DskipTests clean package'
            }
            post {
                success {
                    echo "Now Archiving the Artifacts....."
                    archiveArtifacts artifacts: '**/*.jar'
                }
            }
        }
        stage('Test') {
            steps {
                sh 'mvn -f hello-app/pom.xml test'
            }
            post {
                always {
                    junit 'hello-app/target/surefire-reports/*.xml'
                }
            }
        }
    }
}
```

- Commit and push the changes to the remote repo on GitHub.

```
git add .
git commit -m 'added jenkinsfile and maven project'
git push
```

- Observe the new built triggered with `git push` command on the Jenkins project page.

- Explain the role of the docker image of maven, `Jenkinsfile` and GitHub Webhook in this automation.

```
sudo -u jenkins /bin/bash # make a bash for jenkins user...
sudo su - jenkins
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

- Back to the job tab and show the `Last Successful Artifacts : single-module-project.jar`