Jenkins

Continuous Delivery

> Continuous Delivery (CD) is a DevOps practice that is used to deploy an application quickly while maintaining a high quality with an automated approach. It is about the way application package is deployed in the Web Server or in the Application Server in environment such as dev, test or staging. Deployment of an application can be done using shell script, batch file, or plugins available in Jenkins. Approach of automated deployment in case of Continuous Delivery and Continuous Deployment will be always same most of the time. In the case of Continuous Delivery, the application package is always production ready

Pre-requisites

- Below steps assume that, you have a Jenkins Server Up and Running on one of the EC2 instance.

Apache Tomcat on Amazon Linux:

```
sudo hostnamectl set-hostname tomcat.example.com
sudo yum install java-1.8.0 -y
cd /opt/
sudo wget https://archive.apache.org/dist/tomcat/tomcat-9/v9.0.35/bin/apache-
tomcat-9.0.35.tar.gz
sudo tar -zvxf apache-tomcat-9.0.35.tar.gz
----------------------------
x -  Extract files
v - Verbose, print the file names as they are extracted one by one
z - The file is a “gzipped” file
f - Use the following tar archive for the operation
----------------------------
sudo ls -ltr /opt/apache-tomcat-9.0.35/bin
sudo cd /opt/apache-tomcat-9.0.35/bin
```

- To Start Apache Tomcat : Run the `./startup.sh` file in `/opt/apache-tomcat-9.0.35/bin`
- We can make the scripts executable and then create a symbolic link for this scripts.

```
sudo chmod +x /opt/apache-tomcat-9.0.35/bin/startup.sh
sudo chmod +x /opt/apache-tomcat-9.0.35/bin/shutdown.sh
```

- Create symbolic link to these file so that tomcat server start and stop can be executed from any directory.

```
echo $PATH
sudo ln -s /opt/apache-tomcat-9.0.35/bin/startup.sh /usr/bin/tomcatup
sudo ln -s /opt/apache-tomcat-9.0.35/bin/shutdown.sh /usr/bin/tomcatdown
tomcatup
netstat -nltp | grep 8080
```

> If you want to run Apache Tomcat on same Machine where Jenkins is Installed, then change the port of Apache Tomcat in : `/opt/apache-tomcat-9.0.35/conf/server.xml` file to `8090` as below,

```
<Connector port="8090" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

- If above changes are made, execute the command `tomcatdown` and `tomcatup`.
- Create an empty repo and clone it, add project files into the local git folder and commit -> push the local repo to remote github repo using Git Bash.
- Verify the files are available in your github repository

Tomcat War file deployment Configs

- To have access to the dashboard the admin user needs the manager-gui role. Later, we will need to deploy a WAR file using Maven, for this, we need the `manager-script` role too.
- In order for Tomcat to accept remote deployments, we have to add a user with the role `manager-script. To do so, edit the file` ../conf/tomcat-users.xml` and add the following lines:
- In this case : add below in file `/opt/apache-tomcat-9.0.35/conf/tomcat-users.xml`

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="admin" roles="manager-gui, manager-script"/>
<user username="deployer" password="deployer" roles="manager-script" />
```

- Edit the RemoteAddrValve under this file `/opt/apache-tomcat-9.0.35/webapps/manager/META-INF/context.xml` to allow all.
- `Before`

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
 allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
```

- `After`

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
 allow=".*" />
```

- Restart the tomcat server using `tomcatdown` and `tomcatup`

Jenkins Plugin installation

- To install the Plugin `Deploy to container` navigate to `Manage Jenkins` > `Manage Plugins`, search `Deploy to container` under `Available` tab.

Jenkins Job to deploy war file

- Click on **New Item** then enter an item name, select **Freestyle project**.
- Select the GitHub project checkbox and set the Project URL to point to your GitHub Repository. `https://github.com/YourUserName/`
- Under Source Code Management Section : Provide the Github Repository URL of the Maven Project, keep the branch as `master`.
- Go to Jenkins Project -> Configure -> Under Build Environment Build Step > Select `Invoke top-level Maven targets` from dropdown > select the `Maven Version` that is configured > Enter `clean install`
- Under `Post-build Actions`, from the `Add post-build action` dropdown button select the option `Deploy war/ear to a container`
- Enter details of the War file that will be created as:
    - For `WAR/EAR files` you can use wild cards, e.g. `**/*.war`.
    - The `context path` is the context path part of the URL under which your application will be published in Tomcat.
    - Select the appropriate Tomcat version from the Container dropdown box (note that you can also deploy to Glassfish or JBoss using this Jenkins plugin).
    - Under the `Credentials`, `Add` username and password value that is entered in the `tomcat-users.xml` file.
    - The Tomcat URL is the base URL through which your Tomcat instance can be reached (e.g http://172.31.67.85:8080)

> Make Sure network is open on specific port.

- `Save` the changes and `Build Now`.
- Once Jenkins Job is build, if there is a Success for deploy, verify the deployment files on Tomcat Server under `webapps` path.
- Make some changes in the code on the github configured branch in the Jenkins Job and build the Job again to verify the Artifact Deployment on Tomcat Path.

Artifacts Archive

- Go to `Jenkins dashboard` -> `Jenkins project or build job` -> `Post-build Actions` -> `Add post-build action` -> `Archive the artifacts`:
- Enter details for options in `Archive the artifacts` section:
  - For `Files to archive` enter the Path of the `.war` file like : `java-tomcat-sample/target/*.war`
- `Save` the changes and `Build Now`.
- Check the directories as below to validate above information:

```
ls /var/lib/jenkins/jobs
ls /var/lib/jenkins/jobs/<JOB_NAME>
ls /var/lib/jenkins/jobs/<JOB_NAME>/builds/<BUILD_NUMBER>
ls /var/lib/jenkins/workspace/<JOB_NAME>
```

- If you check the directory structure, there will be archive directory present under the subsequent build number for which the job is executed with Post build action as `Archive the artifacts`
- Once build is successfull , lets add webhook to the github project.

Jenkins Environment Variables:

- To view all the environment variables simply append `env-vars.html` to your Jenkins Server's URL.
- Create a simple free style job to display the value of the environment variables that are set for a Jenkins Job:
- Under Build Section > Add build step > Execute shell , add below commands:

```
echo "BUILD_NUMBER" :: $BUILD_NUMBER
echo "BUILD_ID" :: $BUILD_ID
echo "BUILD_DISPLAY_NAME" :: $BUILD_DISPLAY_NAME
echo "JOB_NAME" :: $JOB_NAME
echo "EXECUTOR_NUMBER" :: $EXECUTOR_NUMBER
echo "NODE_NAME" :: $NODE_NAME
echo "NODE_LABELS" :: $NODE_LABELS
echo "WORKSPACE" :: $WORKSPACE
```

```
echo "JENKINS_HOME" :: $JENKINS_HOME
echo "JENKINS_URL" :: $JENKINS_URL
echo "BUILD_URL" ::$BUILD_URL
echo "JOB_URL" :: $JOB_URL
echo "Below output is all the environment variable in Jenkins"
printenv
```

- The `printenv` command prints all the Jenkins Environment Variables set for that specific Build.

Jenkins Github Webhook

- Integrate jenkins with github so automatically CICD works when any commit is made to the repo Go to `Jenkins` > `Manage Jenkins` > `Configure System` > `Add a Github Server` > Enter URL : `http://public-ip:8080/github-webhook/`
- Lets add a webhook in Github to point to Jenkins URL
- In `Github Repository > Go to Repository > Settings > Go to webhook and addnew webhook > Specify http://public-ip:8080/github-webhook/`
- Go to Jenkins Project, Select the `GitHub hook trigger for GITScm polling` checkbox under Build Triggers tab > `Save`
- For Webhook to work, open port 8080 in security group.
- Now if we make some changes to some file in Github, this Jenkins Project should be triggered.
- Jenkins receives a Github Payload similar to this Github Push Webhook Event Payload

Jenkins Build with Jenkinsfile

- Navigate to Provide a name for your new item and select Pipeline
- **Jenkinsfile**

```
pipeline {
    agent any
    parameters {
        string(name: 'myParameter', defaultValue: 'myVal', description: 'Enter
Parameter value?')
    }
    stages {
        stage('Build') {
            steps {
                echo 'Building..'
                echo "Running ${env.BUILD_ID} on ${env.JENKINS_URL}"
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
                echo "${params.myParameter} is value retrieved!"
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
```

```
            }
        }
    }
}
```

Jenkins pipeline-syntax

- Jenkins has a built-in **Snippet Generator** utility that is helpful for creating bits of code for individual steps, discovering new steps provided by plugins, or experimenting with different parameters for a particular step.
- The Snippet Generator is dynamically populated with a list of the steps available to the Jenkins instance. The number of steps available is dependent on the plugins installed which explicitly expose steps for use in Pipeline.
- To generate a **step snippet** with the **Snippet Generator**:
    - Navigate to the Pipeline Syntax link from a configured Pipeline, or at `${YOUR_JENKINS_URL}/pipeline-syntax`.
    - Select the desired step in the Sample Step dropdown menu
    - Use the dynamically populated area below the Sample Step dropdown to configure the selected step.
    - Click Generate Pipeline Script to create a snippet of Pipeline which can be copied and pasted into a Pipeline.
- The code for a Jenkinsfile should be available in Github Repo
- Click the Add Source button, select git choose the type of repository you want to use and fill in the details.
- Click the Save button and watch your first Pipeline run!

```
pipeline {
    agent { docker { image 'python:3.5.1' } }
    stages {
        stage('build') {
            steps {
                sh 'python --version'
                sh 'echo Hello Jenkins'
            }
        }
    }
}
```

- Since above `Jenkinsfile` contains a stage with docker agent, we need to install docker on the Jenkins Node.

```
```

```
#install docker
sudo yum install docker -y
sudo systemctl start docker

#add jenkins user to docker and wheel group
sudo usermod -aG wheel jenkins
sudo usermod -aG docker jenkins

#Restart jenkins
sudo systemctl restart jenkins
```

- Click on **Build Now** to build the jenkinsfile project

Configuring Credentials in Jenkinsfile

- Navigate to `Jenkins Home page > Credentials > System > Add Credentials`.
- Select Scope as 'Global' **Global** - When credentials are to be added for a Pipeline project/item. **System** - When credentials are to be added for a Jenkins itself to interact with system administration functions., such as email authentication, agent connection, etc. This option applies the scope of the credential to a single object only.
- Types of credentials:
    - **Secret text** - a token such as an API token (e.g. a GitHub personal access token)
    - **Username and password** - which could be a colon separated string in the format username:password

Audit Trail Plugin – an overview and usage

- Manage Jenkins > Manage Plugins > Install the `Audit Trail` Plugin.
- Go to Manage Jenkins > Configure Systems > Audit Trail > Add Logger > Select Log
- Provide the Log Location as `/var/log/jenkins/audit-%g.log`, provide Log File Size as `50` and Log File Count `10`.
- After executing some build job for some Jenkins Project, check the content of the audit file as below.

```
ls -ltr /var/log/jenkins/


> Audit Trail Plugin keeps a log of users who performed particular Jenkins
operations, such as configuring jobs.
This plugin adds an Audit Trail section in the main Jenkins configuration page.
Here you can configure log location and settings (file size and number of rotating
log files), and a URI pattern for requests to be logged. The default options
select most actions with significant effect such as creating/configuring/deleting
jobs and views or delete/save-forever/start a build. The log is written to disk as
configured and recent entries can also be viewed in the Manage / System Log
section.
```