**Lab: Parameterized Build and Triggers**

**Pre-Requisites: -**

1. Git Repository with the code to be deployed should be available. The git repo to be used in training: <https://github.com/LovesCloud/java-tomcat-demo-app> Please fork the repo in your GitHub account.
2. Install following Plugins
   1. Hudson Post Build Task
   2. Paramerized Trigger Plugin
   3. Deploy to Container Plugin
3. **AWS CLI** is required to be installed in Server where Job will be run. For training , aws cli is already installed in slaves machine.If you want to run job in Jenkins master, please do install aws cli on it first.

*Note: AWS cli is required because we are saving and retrieving artifacts in and from AWS S3.*

1. Deployment server i.e. Tomcat Server should be set up. For Lab, We have the tomcat server ready, Tomcat Server IP is available with Trainer.To know the settings done in Tomcat server , Refer section [**SetUp Done at Tomcat Server End**](#_toc132)
2. Create Tomcat deployment user in your Jenkins following the steps mentioned in [**Create Tomcat Deployment User**](#_toc118)
3. [Configure GitHub Webhook for Jenkins](#_Configure_GitHub_Webhook) on your forked repo. follow the steps mentioned in [**Configure GitHub Webhook for Jenkins**](#_Configure_GitHub_Webhook_1)

***Steps To Follow:***

***Step A:*** *Create a job for building the project*

1. Click on **New Item**
2. Enter **Name** such as <yourname>\_buildjob
3. Select **Maven project**
4. Click **OK**
5. Under **General**  Tab, Select **Restrict where this project should be run** checkbox.
6. Enter **Label Expression**  as you slave name
7. Under Source Code Management section, Select **Git** radio button
   1. Enter Repository URL - <Git repo>

*(Note: Git Repository URL is the one forked as part of prerequisites)*

1. Under Build Triggers section, Select **GitHub hook trigger for GITScm polling** checkbox
2. Under Build section
   1. Enter **Root POM**-pom.xml
   2. Enter **Goals and options**-clean package
3. Under Post-build Actions, Click **Add post-build action** drop down and select Post build task.
4. Post build task section gets added.
5. Enter **Log text** as BUILD SUCCESS
6. Enter **Script** as *aws s3 cp target/\*.war s3://jenkinslabs-artifacts/$JOB\_NAME/$BUILD\_NUMBER/*
7. Click **Save**

***Step B:*** *Create a job for deploying the application on Tomcat Server*

1. In Jenkins, Click on **New Item**
2. Enter **Name** such as <yourname>\_deployjob
3. Select **Freestyle project**
4. Click **OK**
5. Under General, Select **This project is parameterized** checkbox
   1. Select **Add Parameter**-String Parameter
   2. Enter **Name-**DEPLOY\_VERSION
   3. Enter **Default value**-0
   4. Enter **Description**-To deploy latest war file built in build job
   5. Select **Trim the string** checkbox
   6. Repeat step a.
   7. Enter **Name-**BUILD\_JOB\_NAME
   8. Enter **Default value**-0
   9. Enter **Description**- This is the build job name.
   10. Select **Trim the string** checkbox
6. Under Build section, Select **Add build step**- Execute Shell
7. Enter **Command as** aws s3 cp s3://jenkinslabs-artifacts/$BUILD\_JOB\_NAME/$DEPLOY\_VERSION/demopipeline.war .

Note: Please do not miss the space and dot after war. Please take the complete command highlighted in yellow.

1. Under Post-build Actions, Select **Add post-build action**- Deploy war/ear to a container.
   1. Enter WAR/EAR files: demopipeline.war
   2. Context path: <yourname>
   3. **Containers** field:
      1. Click **Add Containers** dropdown: Select Tomcat 7.x
      2. **Credentials**: select **tomcatserveruser** user;
      3. Enter **Tomcat URL**: http://<Deployment Server IP>:8080
2. Click **Save**

**Step C:** *Modify the build job*

1. From the Jenkins Home Page, click on the build job link created in Step A.
2. From Left Panel, Click **Configure**
3. Scroll down
4. Under Post-build Actions, Select **Add post-build action**- Trigger parameterized build on other projects
   1. Enter **Build Triggers->Projects to build**-<Name of the Deploy job>
   2. Select **Trigger when build is-**Stable
   3. Select **Add Parameters**-Predefined parameters
      1. Enter **Parameters-**DEPLOY\_VERSION=${BUILD\_NUMBER}

BUILD\_JOB\_NAME=${JOB\_NAME}

1. Click **Save**

**Step D:** *Triggering the Deployment automatically.*

1. On Jenkins end, Open the Build Job’s Details Page
2. On GitHub end, as you are the owner of the new forked Git repo, edit any file and commit the changes.
3. Open Jenkins again, Observe the new triggered build On Left Navigation Panel in Build Job details page.
4. Follow the same steps to see the log as we did in previous labs.
5. Click on the job name (triggered after the successful completion of build job) present at the end of the page;
6. View the Console Output of the latest build executed in this deploy job
7. It should display the Finished status as Success
8. Verify the deployed application by following the steps mention in **Step E.**

**Step E:** *Verifying the deployed application*

1. Open any browser
2. http://<Deployment Server IP>:8080/<Context path>

Note: the <Context path> was set in the deploy job in **Step B->8->b**, please take from there.

1. Hit Enter;

# **Configure GitHub Webhook for Jenkins**

1. Open GitHub
2. Navigate to Git Repo;
3. Navigate to Settings of repository
4. Click Webhook
5. Click Add Webhook
6. Enter Payload URL-http://<Public IP of Jenkins Server>:8080/github-webhook/
7. Click Save Webhook

# **Create Tomcat Deployment User**

1. Login to Jenkins;
2. On the Left Navigation panel, Clcik on  **Credentials** link
3. Under **Stores Cpies to Jenkins** section,Click on **Global** link
4. Click **Add Credentials**
5. Keep **Kind** as Username with password
6. Enter **Username**  as deployer
7. Enter **Password**  as deployer
8. Enter **ID**  as tomcatserveruser
9. Enter **Description**  as tomcatserveruser
10. Click **OK**

# **SetUp Done at Tomcat Server End**

1. Ssh the tomcat server;
2. Type command: sudo su
3. Type command: cd /opt/tomcat/conf
4. Type command: vim tomcat-users.xml
5. Press i
6. Use down arrow till second last line in the file
7. Add the code

<role rolename="manager-script"/>

<user username="tomcat" password="tomcat" roles="manager-script"/>

<user username="deployer" password="deployer" roles="manager-script"/>

1. Press escape
2. Press :wq!
3. Type command: cd /opt/tomcat/webapps/manager/META-INF
4. Type command: vim context.xml
5. Press i
6. Use down arrow and take the cursor to the Value section;
7. comment Valve section by prefixing the code with “<!--” and suffixing the code with “-->”

For Eg:-

<Context antiResourceLocking="false" privileged="true" >

<!--

<Valve className="org.apache.catalina.valves.RemoteAddrValve"

allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />

-->

<Manager sessionAttributeValueClassNameFilter ="java\.lang\.(?:Boolean|Integer|Long|Number|String)|org\.apache\.catalina\.filters\.CsrfPreventionFilter\$LruCache(?:\$1)?|java\.util\.(?:Linked)?HashMap"/>

</Context>

1. Press escape
2. Press :wq!

If you would like to change the port where Tomcat should run, then follow these steps: -

1. To change the port of Tomcat Server.
2. Type command: cd /opt/tomcat/conf
3. Type command: vim server.xml
4. Press i
5. Replace the port 8080 with 9090
6. Press escape
7. Press :wq!