Step by step procedure to create an end to end CI-CD pipeline using pipeline script

Abstract

This document demonstrates the step by step procedure to create an end to end CI-CD pipeline with the following devops steps

- Source code checkout
- Quality gateway analysis
- Build
- Artifactory upload and download
- Deploying to remote tomcat and mysql container
- JIRA to track bugs
- Email notification
- Load Balancing

This documents gives steps to install the necessary softwares for each step and configure them.

Contents

Installing Jenkins

Prerequisites:

Java 8 or above installed and JAVA_HOME environment variable set to the right path.

Installing Jenkins:

Step 1: import GPG keys of the Jenkins repository.

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

Step 2: Add Jenkins repository to the system.

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

Step 3 : Update the apt package list and install the latest version of Jenkins

$ sudo apt update

$ sudo apt install Jenkins

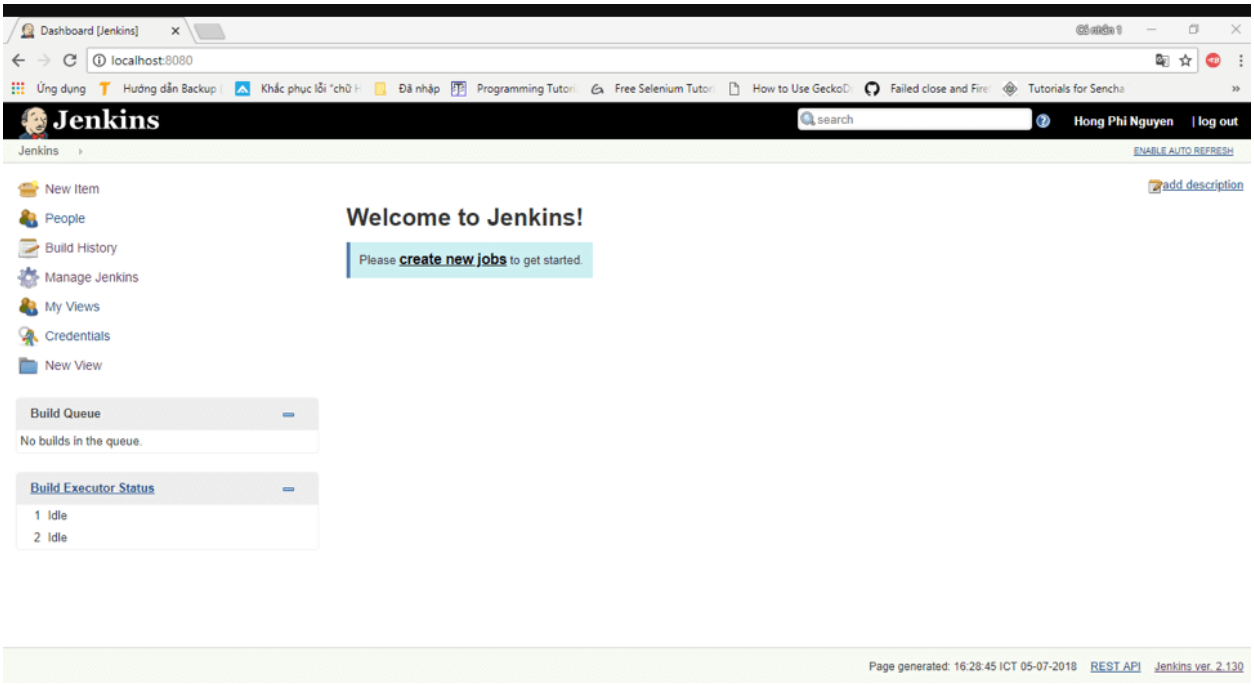
Step 4 : Check if Jenkins is working properly

$ systemctl status Jenkins

Step 5 : Adjust firewall to allow the port 8080 for Jenkins

$ sudo ufw allow 8080

Open up your browser and register as an admin. Select suggested plugins when prompted. Upon successful completion you will get the following screen.



Go ahead and create a pipeline project with relevant name to build the pipeline.

Source Code Checkout

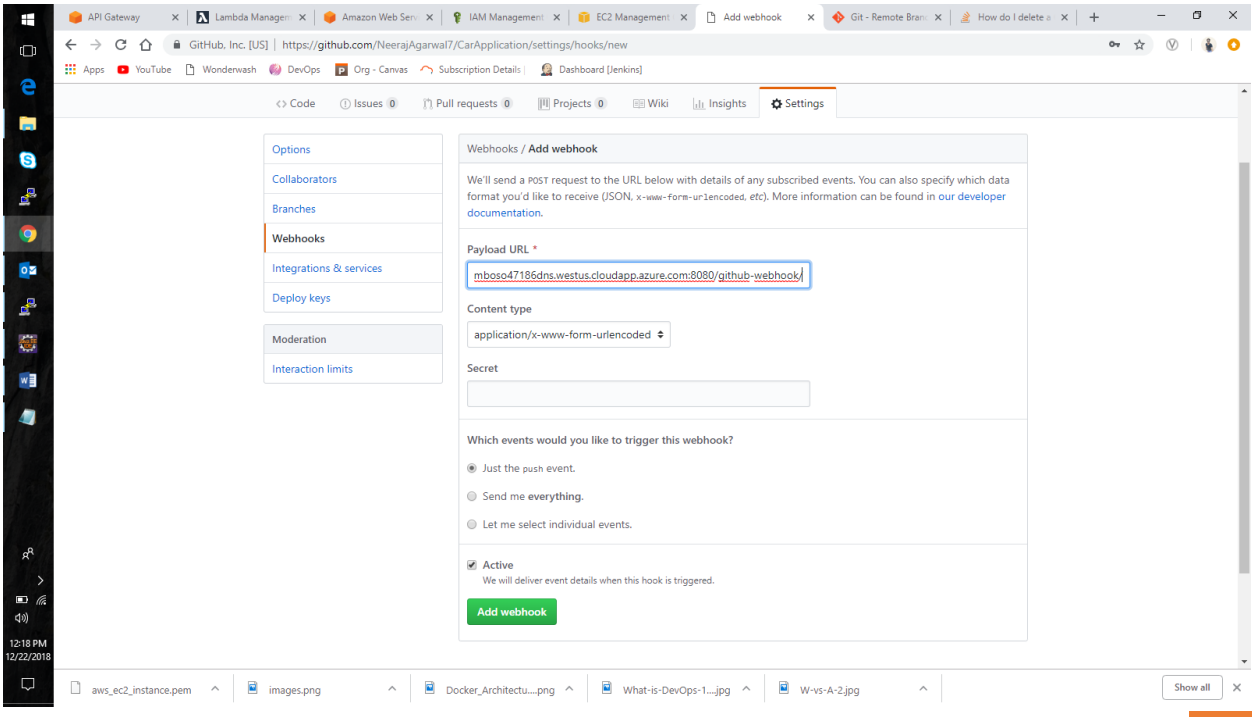
We are using GitHub as our source code management tool. The following steps describe how to get source code from GitHub and trigger build when code is pushed into that repository using webhooks.

- Go to **Manage Jenkins → Manage Plugins → Available**
- Select the following plugins and click on install without restart
    - Git plugin
    - Github plugin

Restart Jenkins after installation if required.

- Login to your github account and open the project you want to configure
- Go to settings and click on **Webhooks → Add webhook**
- Enter the payload url as

http://<Jenkins URL>/github-webhook/



- Click on add webhook
- In your Jenkins job go to **configuration** and under build triggers select **GitHub hook trigger for GITScm polling** option.
- Now we have configured our Jenkins job to trigger build when code is pushed to github.

In your pipeline script create a stage “Source code Checkout” and enter the following script

```
stage('Preparation') {
    git ‘<GitHub Repository URL>’
}
```

This will get the source code for the project from the github url specified. And will trigger a build every time code is pushed into github.

Quality gate analysis

We are using sonarqube for checking code quality and create a quality gate for the project.

Installing Sonarqube

Step 1 : create a sonarqube user

$ sudo adduser --system --no-create-home --group --disabled-login sonarqube

Step 2 : create the directory that will hold the SonarQube files

$ sudo mkdir /opt/sonarqube

Step 3 : update the permissions so that the sonarqube user will be able to read and write files in this directory

$ sudo chown -R sonarqube:sonarqube /opt/sonarqube

Step 4 : SonarQube releases are packaged in a zipped format, so install the unzip utility using your package manager so you can extract the distribution files

$ sudo apt-get install unzip

Step 5 : change the current working directory to the SonarQube installation directory

$ cd /opt/sonarqube

Step 6 : download the sonarqube file

**$sudo wget **
https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-7.5.zip

Step 7 : Then unzip the file

$ sudo unzip sonarqube-7.5.zip

Installing MySQL

We need to create a database and credentials that SonarQube will use.

Step 1 : we need to install MySQL, a database management system, to store and manage the data

```
$ sudo apt-get install mysql-server
```

Step 2 : To secure the installation, we can run a simple security script that will ask whether we want to modify some insecure defaults. Begin the script by typing

```
$ mysql_secure_installation
```

You will be asked to enter the password you set for the MySQL root account. Next, you will be asked if you want to configure the VALIDATE PASSWORD PLUGIN. Answer y for yes, or anything else to continue without enabling.

If you enabled password validation, you'll be shown a password strength for the existing root password, and asked you if you want to change that password. If you are happy with your current password, enter n for "no" at the prompt.

For the rest of the questions, you should press Y and hit the Enter key at each prompt. This will remove some anonymous users and the test database, disable remote root logins, and load these new rules so that MySQL immediately respects the changes we have made.

At this point, your database system is now set up and we can move on.

Step 3 : Log in to the MySQL server as the root user

$ mysql -u root –p

Step 4 : Then create the SonarQube database

mysql> CREATE DATABASE sonarqube;

Step 5 : Now create the credentials that SonarQube will use to access the database.

mysql> CREATE USER sonarqube@'localhost' IDENTIFIED BY 'some_secure_password';

mysql> GRANT ALL ON sonarqube.* to sonarqube@'localhost';

mysql> FLUSH PRIVILEGES;

mysql> EXIT;

Configuring the SonarQube Server

Step 1 : Start by opening the SonarQube configuration file

$ sudo nano sonarqube-7.5/conf/sonar.properties

First, change the username and password that SonarQube will use to access the database to the username and password you created for MySQL

...

sonar.jdbc.username=sonarqube

sonar.jdbc.password=some_secure_password

...

Next, tell SonarQube to use MySQL as the database driver

...

sonar.jdbc.url=jdbc:mysql://localhost:3306/sonarqube?useUnicode=true&characterEncoding=utf8&rewriteBatchedStatements=true&useConfigs=maxPerformance&useSSL=false

...

Finally, tell SonarQube to run in server mode and only listen to the local address

...

sonar.web.host=127.0.0.1

sonar.web.javaAdditionalOpts=-server

Step 2 : Create the service file as follows

$ sudo nano /etc/systemd/system/sonarqube.service

[Unit]

Description=SonarQube service

After=syslog.target network.target

[Service]

Type=forking

ExecStart=/opt/sonarqube/sonarqube-7.5/bin/linux-x86-64/sonar.sh start

ExecStop=/opt/sonarqube/sonarqube-7.5/bin/linux-x86-64/sonar.sh stop

User=sonarqube

Group=sonarqube

Restart=always

[Install]

WantedBy=multi-user.target

Script to perform sonarqube code quality analysis

Step 1 : Go to plugin-manager of Jenkins to install “Sonarqube Plugin”

Step 2 : Go to Global system configurations and set up the Sonarqube server details(URL and Token Number)

Step 3 : Configure maven home in global configuration settings.

Step 3 : In Jenkins job configuration, go to pre build option and select “execute shell” and write the following shell script

```
stage(Code Quality){
    def mvn_version="Maven"
    mvn sonar:sonar \
    -Dsonar.host.url= <URL of the sonarqube sever> \
    -Dsonar.login= <Token number>
}
```

OR

```
stage('Code Qaulity') {
    def mvn_version="Maven"
    withEnv( ["PATH+MAVEN=${tool mvn_version}/bin"] ){
        withSonarQubeEnv('mySonar'){
            sh 'mvn sonar:sonar'
    }
}
```

Script to check Sonarqube quality gate

The following script waits for quality gate results from sonarqube server and succeeds only when the code passes the quality gateway. The gateway has default parameters set but you can create your own gateway parameters according to the requirements.

```
stage("SonarQube Quality Gate") {
    withSonarQubeEnv('mySonar'){
        timeout(time: 1, unit: 'HOURS') {
            def qg = waitForQualityGate()
            if (qg.status != 'OK') {
                error "Pipeline aborted due to quality gate failure: ${qg.status}"
            }
        }
    }
}
```

Building the application

Since we are usually building java applications using maven, we will use maven to build our application and package it into war file.

The following script builds the application as well as run all unit test cases defined in the program.

```
stage('Build') {
    withEnv( ["PATH+MAVEN=${tool mvn_version}/bin"] ){
        sh 'mvn clean install'
    }
}
```

The resulting war file is stored in the location

/var/lib/jenkins/workspace/<JOB_NAME>/target

With this the build step step is complete.

If you face any errors during the build then check if java is installed in the system and JAVA_HOME is set correctly. Try manually installing maven and setting the MAVEN_HOME environment variable in the system and in Jenkins configuration.

Artifactory

After the build is complete we upload the war file into an artfactory for version controlling, rollbacks etc… The Artifactory used here is Jfrog artifactory.

Installing Jfrog Artifactory

Step 1 : Go to the Jfrog Artifactory official website and browse to download section.

https://jfrog.com/open-source/

Step 2 : Download the zip file fro either wget or WinSCP into your VM and unzip it.

Step 3 : After unzipping file navigate to bin folder and run the 'installService' shell script

$sh installService.sh

Step 4 : Now run the artifactory using the following command

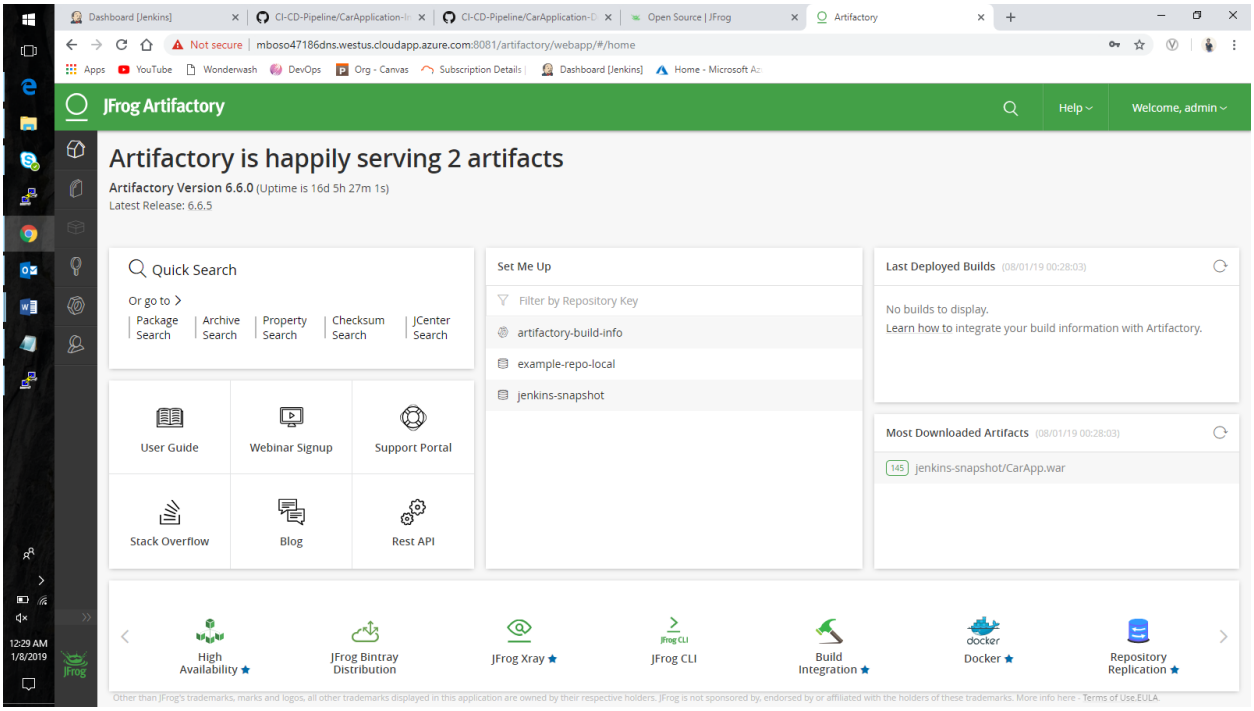
$sudo service artifactory start

Check if artifactory is up and running

$sudo service artifactory status

NOTE: Artifactory by default runs on 8081 port. So make sure the port is free and allowed by the firewall and the server.

Go to your browser and enter the DNS name of your VM followed by the port. If everything is successful you will see the artifactory home page. The default credentials are username:"admin" and password:"password"



If you notice I have created a Repository called “Jenkins-snapshot” which I will be using to store the artifacts. You can also create a repository or use the existing ones.

Script to upload and download artifacts

Before writing the script download the artifactory plugin and configure the artifactory credentials to be used in the global configurations.

Upload:

```
def server =Artifactory.server 'my_artifactory'
   stage('upload to artifactory'){
      def uploadSpec = """{
    "files": [
        {
              "pattern":"<JENKINS_HOME>/workspace/<JOB_NAME>/
     target/*.war",
          "target": "jenkins-snapshot"
        }
      ]
      }"""
   server.upload(uploadSpec)
   }
```

This script will fetch the war file from the Jenkins source and store into the "jenkins-snapshot" repository. You can open your artifactory in the browser and navigate to your repository to check if file has been uploaded or not.

If file is not uploaded then check if you have provided the right credentials or if the repository name is misspelled.

Download:

```
stage('downloading artifact')
   {
      def downloadSpec="""{
      "files":[
      {
        "pattern":"jenkins-snapshot/<FILE_NAME>.war",
         "target":"/var/lib/jenkins/warfiles/"
      }
      ]
      }"""
   server.download(downloadSpec)
}
```

I have created a folder “warfiles” in my Jenkins home to store the war files downloaded you can choose any location of your choice.

With this we have completed the artifactory upload and download of artifacts.

Deployment

For the deployment process we are creating tomcat server and mysql database as Docker containers. Usually these containers reside on another VM. Therefore we have to send the war file from one VM to another using scp.

Before commencing with the step Setup the two VMs to be able to send files between then via scp command without and password or passphrase.

Generating SSH key to setup scp connection

- To generate the SSH key, run the 'ssh-keygen' linux command in the Virtual Machine where we have our Jenkins.
- After generating the key send the ssh public key to the remote VM by executing the command
  scp id_rsa.pub <username>@<IP address>:"/File_Path"
- In the remote VM generate the ssh key and copy the contents of the received file to the file "authorized_keys".
- In the Jenkins VM copy the .ssh folder inside /var/lib/Jenkins to so that Jenkins can use the private key to make connection. Also provide the ownership of that folder to Jenkins using chown command.

Installing Docker

Step 1 : Install packages to allow apt to use a repository over HTTPS

```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
```

Step 2 : Add Docker's official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add –
```

Step 3 : Use the following command to set up the stable repository

```
$ sudo add-apt-repository \
   "deb [arch=amd64]https://download.docker.com/linux/ubuntu\
   $(lsb_release -cs) \
   stable"
```

Step 4 : Install the latest version of Docker CE, or install a specific version

```
$ sudo apt-get install docker-ce
        Or
$ sudo apt-get install docker-ce=<VERSION>
```

Docker file to create tomcat image

Note : Have a stand alone tomcat running in the VM to get its configuration files.

Before creating this image copy two required configuration files "context.xml" and "tomcat-users.xml" into the folder where dockerfile is to be created.

Configure these files according to your requirements (Username and Password for manager app etc…).

You can find these files in these locations

- $CATALINA_HOME/conf/tomcat-users.xml
- $CATALINA_HOME /webapps/manager/META-INF/context.xml

Once you are done with this open up your favorite editor and paste the following code

```
FROM tomcat

MAINTAINER <usern-ame>

RUN apt-get update && apt-get -y upgrade

WORKDIR /usr/local/tomcat

COPY tomcat-users.xml /usr/local/tomcat/conf/tomcat-users.xml

COPY context.xml /usr/local/tomcat/webapps/manager/META-INF/context.xml

EXPOSE 8080
```

Save the file under the name **Dockerfile**

Build the docker file to create a custom tomcat image by running the following command.

$docker build .

This will create a tomcat custom image you can modify the file to give the image a name/tag of your choice. The above file will create an image without a name or tag.

Note down the image id of the image created to be used later.

Docker compose file to create containers

Once the tomcat image is created we can use this to create containers. We are going to write a docker-compose file in YAML format to create tomcat container with volume mounting and mysql container with root password to be used for deploying our application.

Open up your favorite editor an type the following code

```
db:
  image: mysql
  container_name: <MqSQL container name>
  environment:
   MYSQL_ROOT_PASSWORD: <Your password>

web:
```

```
image: <tomcat image id>
container_name: <tomcat container name>
ports:
    - "8081:8080"
volumes:
    - /opt/tomcat/webapps:/usr/local/tomcat/webapps
links:
    - db
```

Make sure you change the volume mount source path according to your desired folder where you want to keep your war files.

If you notice I have used 8081 host port to access the container's 8080 port on which the tomcat server runs. You can use any port available.

Save the file under the name **docker-compose.yml**

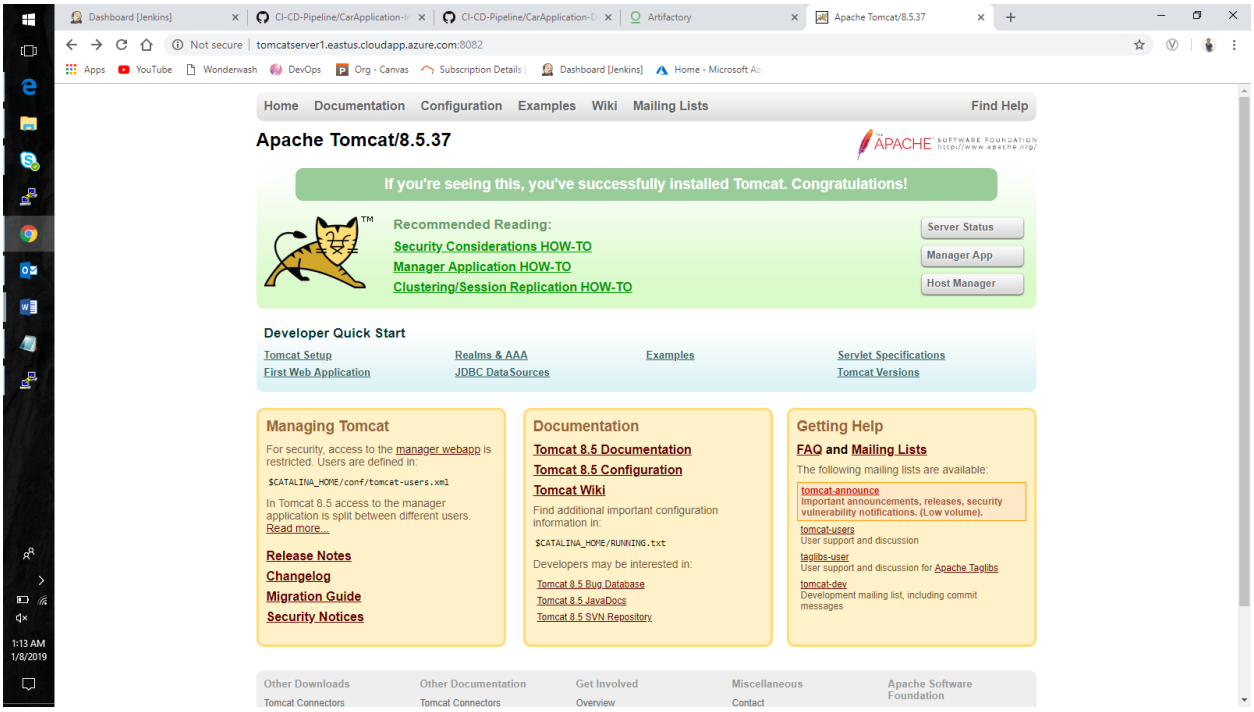
Run the docker compose file by running the command

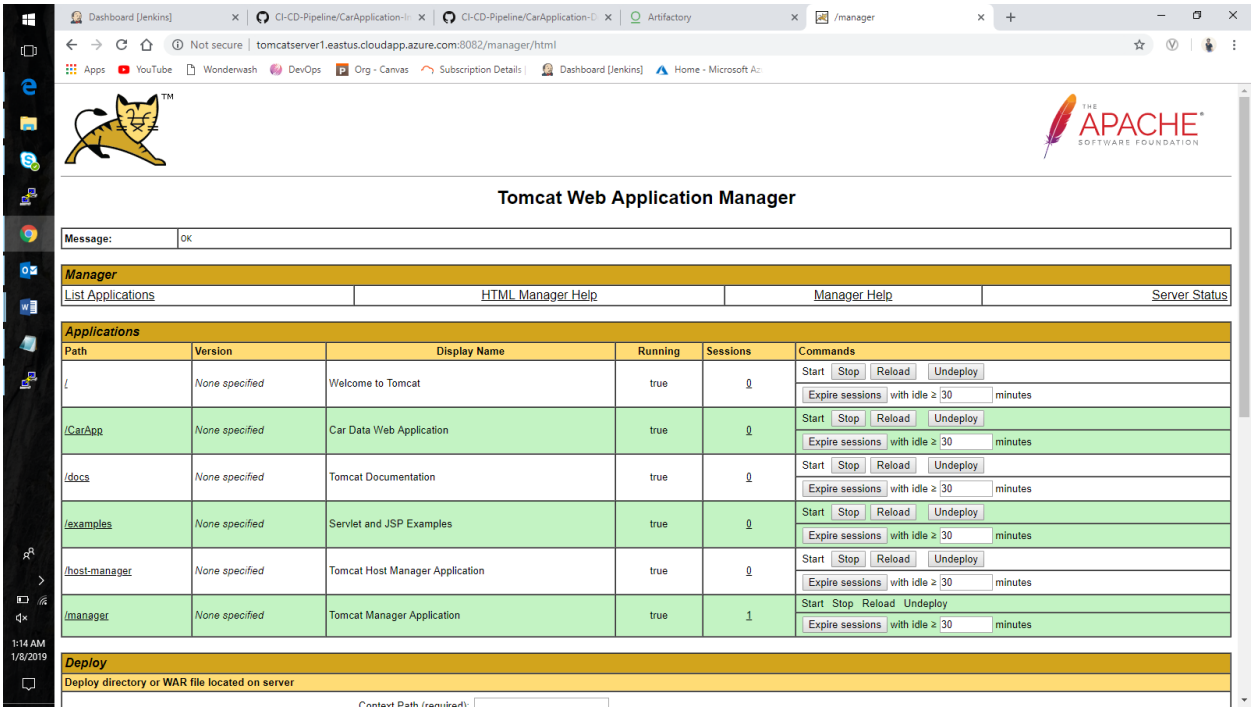
docker-compose up

In your projects configuration files make sure you change the mysql url from localhost to containers ip address. To get the IP address of the mysql container run the **docker inspect** command on the container.

Also change the root password to the one specified by you in the docker-compose file.

Now as soon Jenkins sends the war file to the stand alone tomcat of the vm, it will automatically deploy to the container as well.





As you can see the “CarApp” application is deployed into the VM which I have used to build my pipeline.

JIRA Integration with Jenkins

We are using JIRA software by Attlasian to track bugs or issues in our pipeline. I have used jira for thr following use cases.

Previous Build	Current Build	Action
Success	Failure	Create and assign issue
Failure	Failure	Comment on issue
Failure	Success	Comment and transition of an issue

JIRA automatically provides a key to the issues created and cannot be specified by the user. This can be a problem when we perform operations on tickets dynamically. To tackle this problem I have used an approach where I store the issue id generated into a file and use the same whenever required.

Prerequisites

- **Creating Jira Project**

Before beginning with the procedure you need have an Atlassian account with JIRA software (You can get free one week trail account). On this account create a project with relevant name. To do so…

Go to **JIRA software → Projects → Create project → Classic Project**

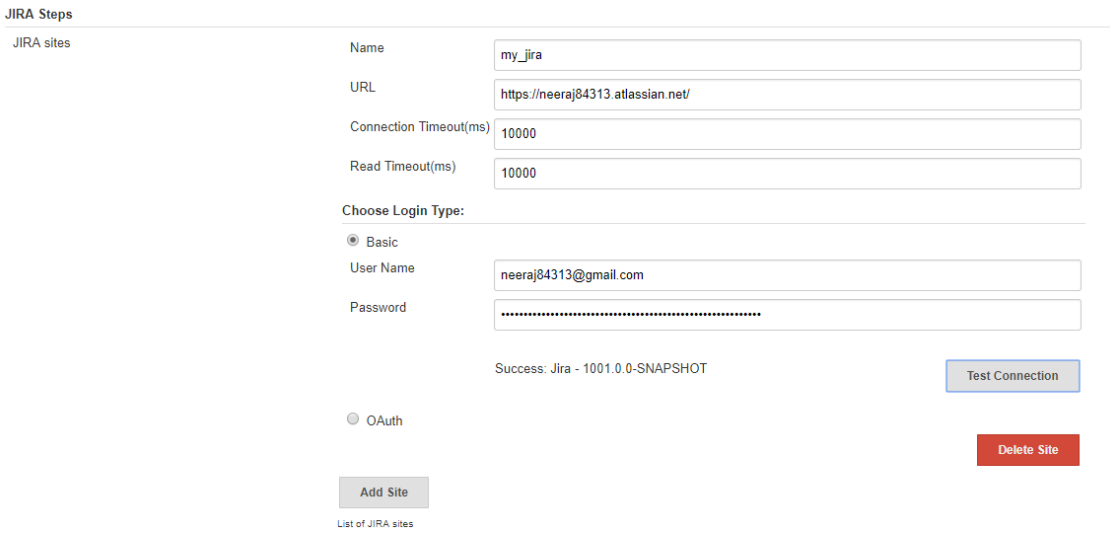
And fill out all necessary data required to create the project.

- **Configuring JIRA plugin in Jenkins**
  Install the following plugins in your Jenkins
  - JIRA plugin
  - JIRA Pipeline Steps
  - Jira Integration for Jenkins

  To configure the JIRA settings in your Jenkins

Go to **Manage Jenkins → Configure System →JIRA Steps**

And provide necessary information



Creating a ticket

Depending upon the use cases and type of pipeline script you can write the following scripts inside try/catch blocks, or as a stage in a pipeline or as post build steps.

Script for creating an issue and assigning it to someone.

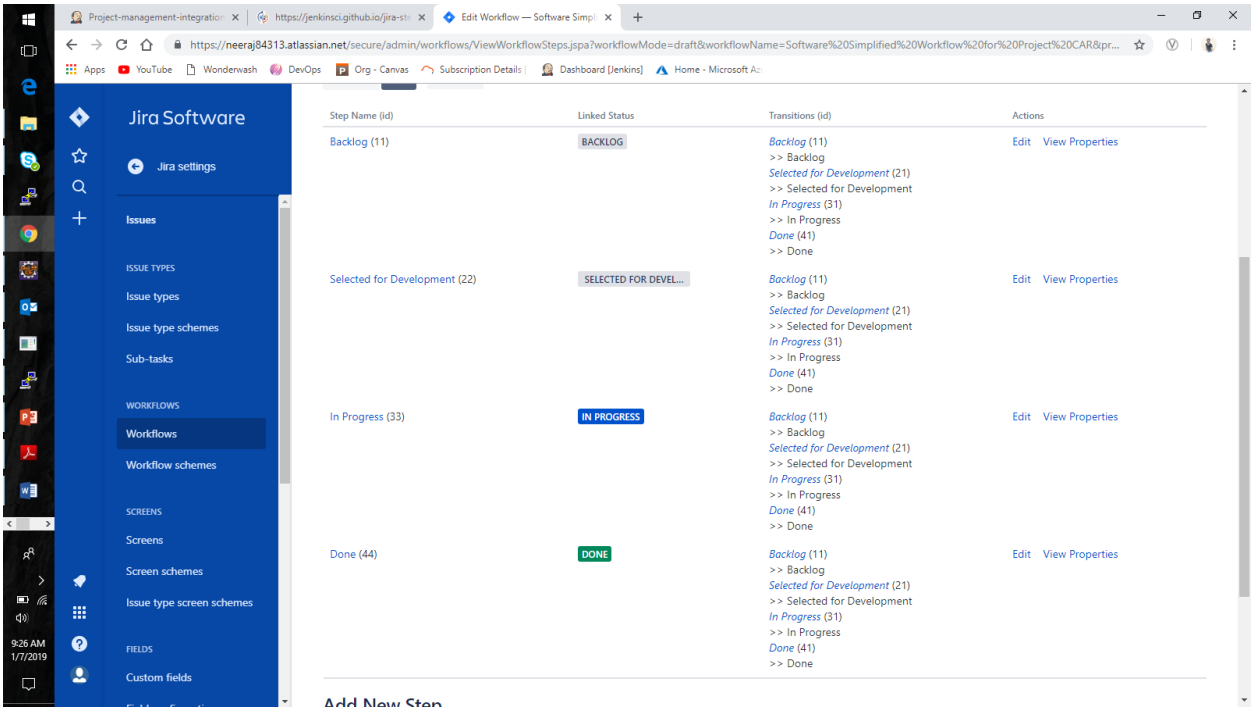
```
withEnv(['JIRA_SITE=<Local JIRA name>']) {
        def testIssue = [fields: [ project: [key: '<Project key>'],
                                summary: 'Build Fail',
                                description: 'Build has failed for application.',
                                issuetype: [name: 'Bug']]]

                response = jiraNewIssue issue: testIssue

                echo response.successful.toString()
                echo response.data.toString()
jiraAssignIssue idOrKey: <issue ID> userName: '<assignee>'
}
```

Adding Comment and transit an issue to another state

You can add comments to any jira ticket at any point of your pipeline depending upon the use case of your project. To transit an issue to a state you need to get the **transition ID** of that state. To find all transition ids of your project

Go to **JIRA → Project settings → Workflows → Edit workflow**

You must get something like this



If you notice you can see transition ids written besides all possible transitions.

With this information in hand you are now ready to perform required operations on a JIRA ticket.

- **Script to add comment**

```
withEnv(['JIRA_SITE=<Local JIRA name>']) {
    jiraAddComment idOrKey:<Issue ID>, comment: '<comment>'
}
```

- **Script to transit an issue**

```
withEnv(['JIRA_SITE=<Local JIRA name>']) {
        def transitionInput =
              [
                 transition: [
                    id: '<Transition ID>'
                 ]
              ]
        jiraTransitionIssue idOrKey:<Issue ID>, input: transitionInput
}
```

Final script according to our use case

```
stage('JIRA') {
       withEnv(['JIRA_SITE=my_jira']) {
          if(currentBuild.result == 'FAILURE'){
             if(currentBuild.previousBuild.result!='FAILURE'){
               <Script to create issue>
               def issue_id=response.data.key
               jiraAssignIssue idOrKey: issue_id userName: '<assignee>'
                       sh 'echo $CAR_ISSUE > /var/lib/jenkins/jira-ticket.txt'
               }
               else if(currentBuild.previousBuild.result=='FAILURE'){
```

```
                def issue=readFile '/var/lib/jenkins/jira-ticket.txt'
                issue=issue.trim()
                jiraAddComment idOrKey:issue, comment: 'Build is still
Failing'
            }
        }
        else if(currentBuild.previousBuild.result=='FAILURE'){
            <Script to transit an issue>
             def issue=readFile '/var/lib/jenkins/jira-ticket.txt'
                    issue=issue.trim()
            jiraTransitionIssue idOrKey:issue, input: transitionInput
            jiraAddComment idOrKey:issue, comment: 'Build is
successful'
        }
    }
  }
```

If the script is written correctly the Jenkins will be able automatically create issues in jira and perform operation on it.

Note : The file jira-ticket.txt was previously created in the Jenkins home so that Jenkins can easily read and write from the file. Make sure no text is to be entered into the file after creation manually.

Sending Emails from Jenkins

Even though JIRA sends an email every time there is some activity to the assignee and the manager, we may need to send email notifications to other people or during some other stage or use case of our pipeline

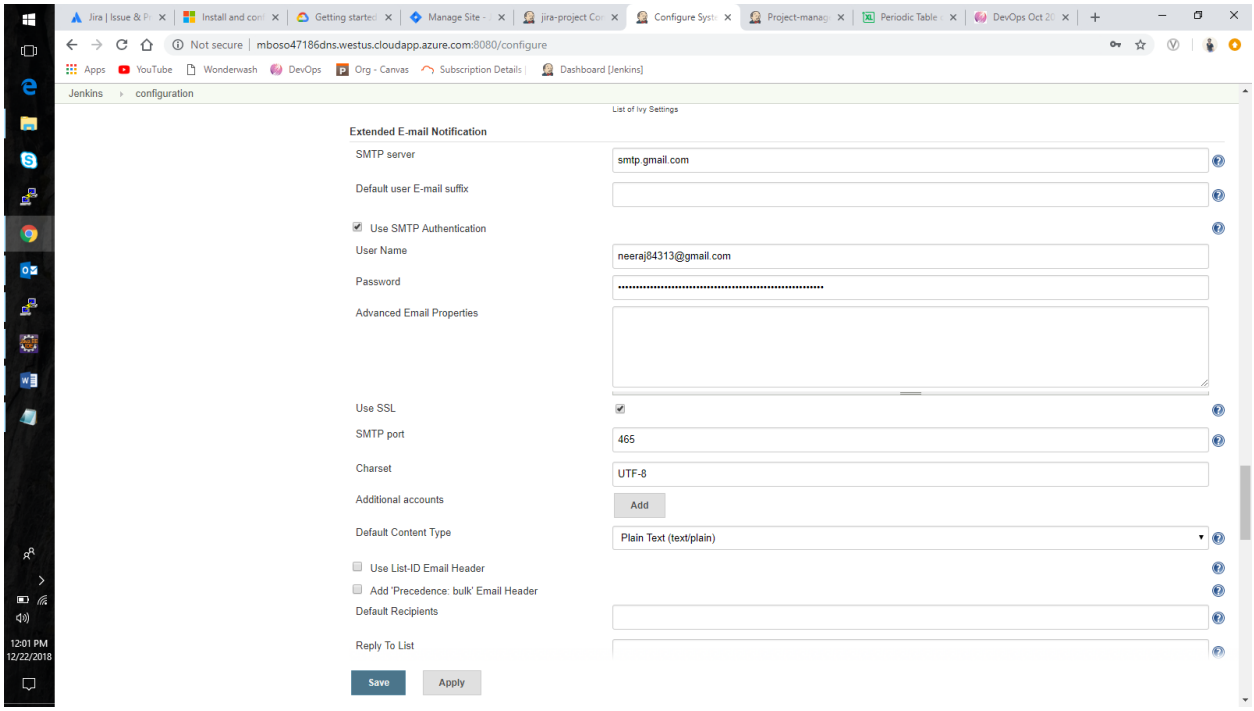
Prerequisites

- Go to **Manage Jenkins → Manage Plugins → Available**
- Select the following plugins and click on install without restart
    - Email Extension Plugin
    - Git plugin
    - Github plugin
    - Mailer plugin

Restart Jenkins after installation if required.

Configuring Email service

- Go to **Manage Jenkins → Configure System**
- Under the **Extended E-mail Notification** section enter all the details and credentials of the sender. After completion it should look something like this :



- Enter the same credentials and information provided above in the **E-mail Notification** section
  Note : ensure you have checked the use SSL checkbox
- Go to your gmail account and under account settings → privacy, turn on **Less secure app access**
  Note : for this to work turn off 2-step verification process
- Select the **Test configuration by sending test e-mail** checkbox to send a test email to check if the configurations and correct

Whenever you want to send an email just add the following lines of code in your script

emailext body: "<SOME_TEXT_FOR_BODY>", subject: '<SUBJECT_OF_EMAIL>', to: '<RECIEVER_URL>'

Nginx Load Balancing

If you wish to run multiple tomcat servers and deploy the application in all of the containers then install nginx server to perform load balancing between the containers.

Installing Nginx

Step 1 : Update local package index

$ sudo apt-get update

Step 2 : Use the apt package management suite to complete the installation.

$ sudo apt-get install nginx

On Ubuntu 16.04, Nginx is configured to start running upon installation.

Step 3 : Allow firewall for nginx connections

$ sudo ufw allow 'Nginx HTTP'

Verify the change using

$ **Sudo ufw status**

Step 4 :

Go to your browser and enter your host URL, it should take you to Nginx's default loading page.

Create tomcat containers

Step 1 : Create two containers from the image downloaded previously

**$ sudo docker run -d -p 8082:8080 -v **
**/opt/tomcat/webapps:/usr/local/tomcat/webapps --name tomcat1 **
<container_id>
**$ sudo docker run -d -p 8083:8080 -v **
**/opt/tomcat/webapps:/usr/local/tomcat/webapps --name tomcat2 **
<container_id>
And so on…

Notice the tag –p 8082:8080, this maps the containers 8080 port to the 8082 port of the host. Make sure the host ports (8082 and 8083) is free and if any service is running then stop it.
Verify if the containers are up and running

$ sudo docker ps

Step 2 : Configure firewall to allow the ports

$ sudo ufw allow 8082
$ sudo ufw allow 8083

Configure Nginx to perform load balancing

Step 1 : Go to the nginx directory

$ cd /etc/nginx/sites-enabled

Step 2 : Create a file with relevant name to configure nginx and add the following lines of code

```
upstream backend {
server <Server URL>:8082;
server <Server URL>:8083;
…
}

server {
listen 80;
server_name <Server URL>;
location / {
proxy_pass http://backend;
}
}
```

With the above lines of code we grouped the host ports under upstream backend. Every time the host receives the request from browser our nginx server redirects it to one of the containers.

You can verify this by going to your browser and typing

http://<Server URL>/

You will see the tomcat home page displayed

To verify whether nginx was able to map both the containers, execute the following command :

$ netstat -lntp | grep "docker-proxy"

You should be able to see something like this :

```
root@ubuntuaf02513a:/etc/nginx/sites-enabled# vi default
root@ubuntuaf02513a:/etc/nginx/sites-enabled# netstat -lntp | grep "docker-proxy"
tcp6       0      0 :::8082                 :::*                    LISTEN      32484/docker-proxy
tcp6       0      0 :::8083                 :::*                    LISTEN      674/docker-proxy
root@ubuntuaf02513a:/etc/nginx/sites-enabled#
```

As we can see both the ports configured in the nginx file are up and running.

Testing the Nginx Load Balancing

Step 1 : Stop one of the docker containers.

$ sudo docker ps

This will give all running docker containers. Copy the CONTAINER_ID of the server you want to stop

$ sudo docket stop < CONTAINER_ID >

Verify that the container is not running by running the command

$ sudo docker ps

Step 2 : Try to access tomcat

Go to your browser and typing

http://<Server URL>/

You will see the tomcat home page displayed. Nginx automatically redirects to the container which is up and running instead of the server which we stopped earlier.

You can also check the same by running the command

$ netstat -lntp | grep "docker-proxy"

You may notice that only one docker-proxy is running.

Conclusion

With this we finish an end to end pipeline with multiple devops stages such as source code checkout, quality assessment, build, artifactory, deployment, email notification and load balancing.

This has been a generalized document for creating a pipeline and does not contain the actual code used by me to create a pipeline for deploying a “CarApp application”. For the source code please refer the following GitHub link of the project

https://github.com/NeerajAgarwal7/CarApplication.git

For the pipeline script used to create the end to end pipeline please refer to the following GitHub link

https://github.com/NeerajAgarwal7/CI-CD-Pipeline.git