Step by step procedure to create an end to end CI-CD pipeline using Jenkins and Docker

Problem Statement

You are a DevOps consultant in AchiStar Technologies. The company decided to implement DevOps to develop and deliver their products. Since it is an Agile organization, it follows Scrum methodology to develop the projects incrementally. You are working with multiple DevOps Engineers to Dockerize the Jenkins Pipeline. During the sprint planning, you agreed to take the lead on this project and plan on the requirements, system configurations, and track the efficiency. The tasks you are responsible for:

- Availability of the application and its versions in the GitHub.
- Track their versions every time a code is committed to the repository.
- Build the application in Docker and host it in Docker Hub.
- Pull the Docker image and run it again.

The company goal is to deliver the product frequently to the production with high-end quality.

You must use the following tools:

- Docker – To build the application in a Docker container and push it to Docker Hub
- Docker Hub – To store the Docker image
- GitHub – To store the application code and track its revisions
- Git – To connect and push files from local system to GitHub
- Linux (Ubuntu) – As a base operating system to start and execute the project
- Jenkins – To automate the deployment process during continuous integration

Following requirements should be met:

- Document the step-by-step process from the initial installation to the final stage.
- Track the versions of the code in the GitHub repository
- Availability of the application in the Docker Hub
- Track the build status of Jenkins for every increment of the project

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
- Maven Build
- Building Docker image with war deployed to tomcat container
- Pushing the image to Docker Hub
- Pulling the image and running the container

This document also gives steps to install the necessary software 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

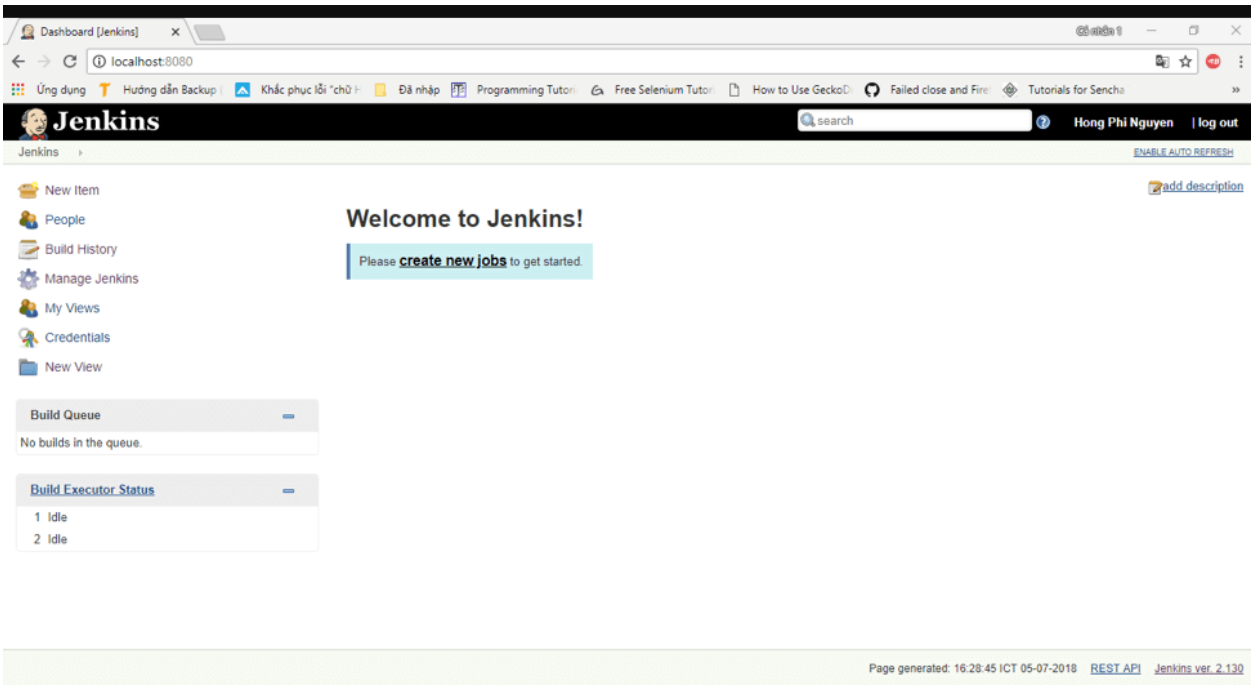
Step 4: Check if Jenkins is working properly

$ 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

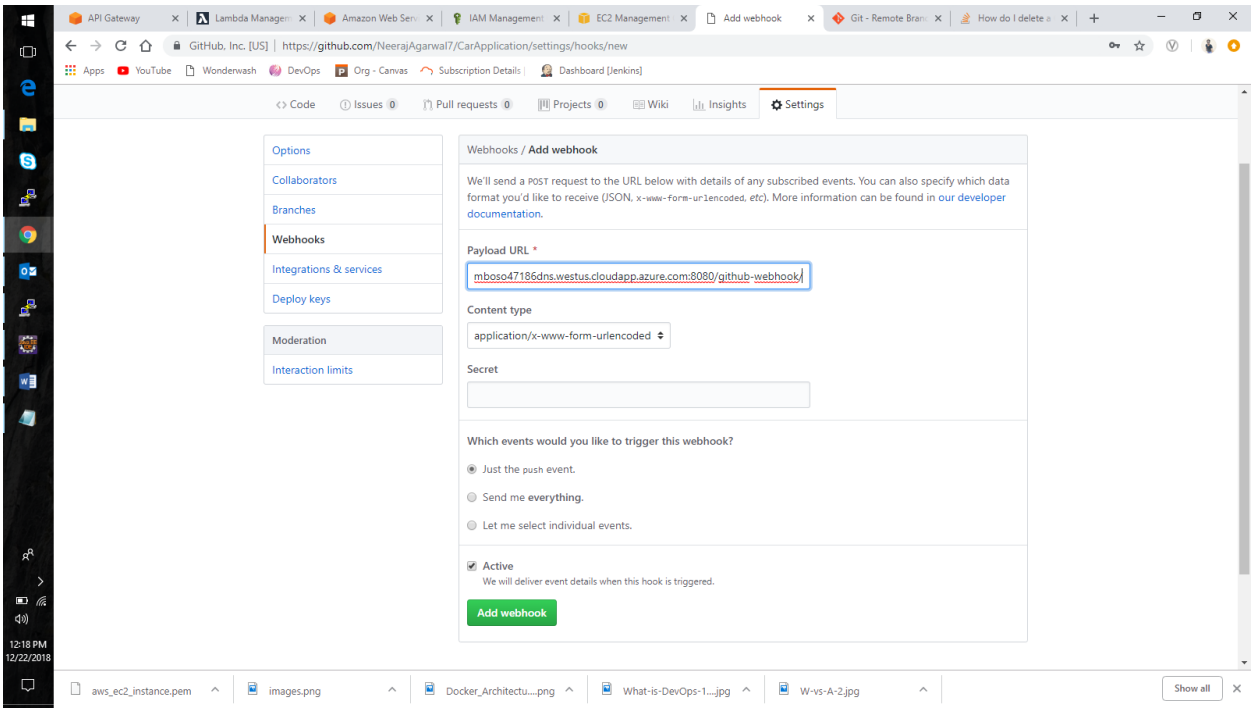
We are using GitHub as our source code management tool. The following steps describe how to get source code from GitHub and trigger build when code is pushed into that repository using webhooks.

- 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.

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 ('App Build') {
    sh "'${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore clean package"
  }
```

This step will build application and package it into a war file.

```
[INFO] --- maven-war-plugin:2.2:war (default-war) @ WebApp ---
[INFO] Packaging webapp
[INFO] Assembling webapp [WebApp] in [/var/lib/jenkins/workspace/devops-201/target/WebApp]
[INFO] Processing war project
[INFO] Copying webapp resources [/var/lib/jenkins/workspace/devops-201/src/main/webapp]
[INFO] Webapp assembled in [38 msecs]
[INFO] Building war: /var/lib/jenkins/workspace/devops-201/target/WebApp.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.300 s
[INFO] Finished at: 2019-04-15T06:54:54Z
[INFO] ------------------------------------------------------------------------
[Pipeline] }
[Pipeline] // stage
```

The resulting war file is stored in the location

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

With this the build 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.

Building Docker Image

This Step explains the procedure to build a custom application Docker image from a Docker file. Since this is a web application we will use tomcat as a base image for our application image. Before proceeding let us see how to install Docker onto our machine.

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>

Writing the Docker file

Note: The necessary configuration files referred in this section are available on git hub repository of this project.

We will build a custom Docker image for our application using tomcat image as a base. To customize it we will copy some configuration files and war files into the right locations inside the container file system.

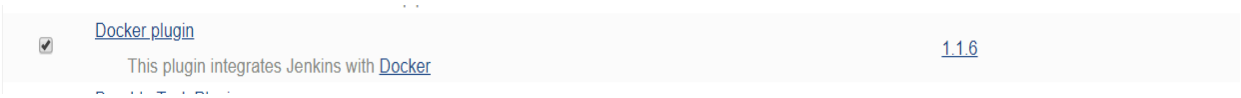
The Docker file used in this project is given below:

```
FROM tomcat

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
COPY target/WebApp.war /usr/local/tomcat/webapps
```

EXPOSE 8080
The COPY commands are copying the configuration files for setting up the tomcat user with manager permissions. This file can be edited to set up a username password combination for each user.

Building the Docker file

For our jenkins to be able to run docker commands smoothly we will install the Docker plugin. Go to Manage jenkins → Manage plugins and under available tab search for Docker. You should be able to find the plugin shown below



Now go ahead and create a stage in the pipeline as below.

```
stage ('Docker Image Build') {
        app =
        docker.build("neerajagarwal7/devops201:${BUILD_NUMBER}")
}
```

If everything checks out correct you should able to see the following output on the console.

```
+ docker build -t neerajagarwal7/devops-201:1-6 .
Sending build context to Docker daemon  143.9kB

Step 1/6 : FROM tomcat
 ---> 5a069ba3df4d
Step 2/6 : WORKDIR /usr/local/tomcat
 ---> Using cache
 ---> 2bfbc3af8213
Step 3/6 : COPY tomcat-users.xml /usr/local/tomcat/conf/tomcat-users.xml
 ---> Using cache
 ---> 3c774cb503c1
Step 4/6 : COPY context.xml /usr/local/tomcat/webapps/manager/META-INF/context.xml
 ---> Using cache
 ---> 77105e4b989f
Step 5/6 : COPY target/WebApp.war /usr/local/tomcat/webapps
 ---> ebf11bec3a8c
Step 6/6 : EXPOSE 8080
 ---> Running in f7f05e92dd14
Removing intermediate container f7f05e92dd14
 ---> 78f198428a80
Successfully built 78f198428a80
Successfully tagged neerajagarwal7/devops-201:1-6
[Pipeline] dockerFingerprintFrom
[Pipeline] }
[Pipeline] // stage
```

Note: Make sure your image name follows the format.

“username/repository_name:tag”

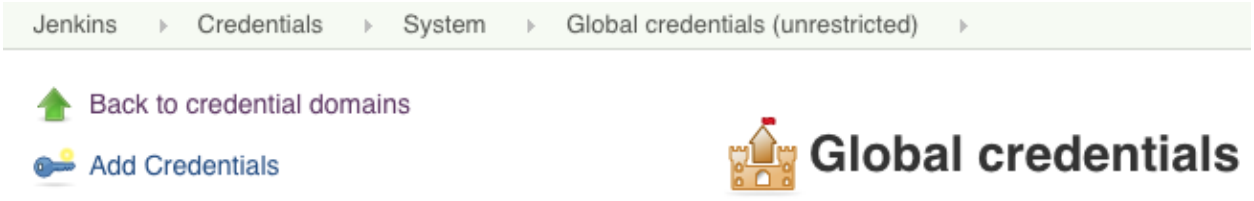
Pushing the Docker image to DockerHub

To push the docker image onto the dockerHub we will need to configure the dockerHub credentials into jenkins so that it can gain access to it.

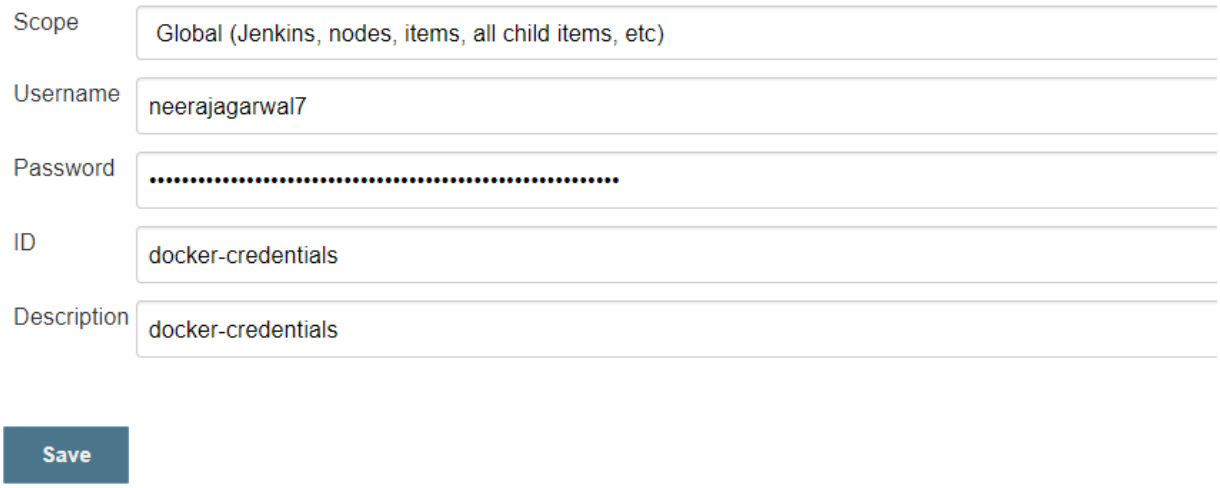
Creating DockerHub Credentials

Go to jenkins home, click on “credentials” and “(global)”.

Click on “Add Credentials” in left menu.



Put your credential and save it.



Pushing the image

The following pipeline script will push the image into the specified DockerHub repository with two tags, latest and build specified version number.

```
stage ('Push Docker Image') {
        docker.withRegistry('https://registry.hub.docker.com','docker-credentials') {
                app.push("1-${BUILD_NUMBER}")
                app.push("latest")
        }
}
```

This code will push the image in dockerHub with two different tags

```
[Pipeline] withDockerRegistry
$ docker login -u neerajagarwal7 -p ******** https://registry.hub.docker.com
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /var/lib/jenkins/workspace/devops-201@tmp/c8ffd506
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[Pipeline] {
[Pipeline] sh
+ docker tag neerajagarwal7/devops-201:1-6 registry.hub.docker.com/neerajagarwal7/devops-201:1-6
[Pipeline] sh
+ docker push registry.hub.docker.com/neerajagarwal7/devops-201:1-6
The push refers to repository [registry.hub.docker.com/neerajagarwal7/devops-201]
fb17848ad3f3: Preparing
```

```
604829a174eb: Layer already exists
fbb641a8b943: Layer already exists
fb17848ad3f3: Pushed
1-6: digest: sha256:38fc1ca84e18d99e53974c76451028d7d85afa303589a336eedb632e2584a522 size: 3248
[Pipeline] sh
```

```
fbb641a8b943: Layer already exists
604829a174eb: Layer already exists
latest: digest: sha256:38fc1ca84e18d99e53974c76451028d7d85afa303589a336eedb632e2584a522 size: 3248
[Pipeline] }
[Pipeline] // withDockerRegistry
```

Remember to provide the right credential ID as mentioned.

Running the Docker image

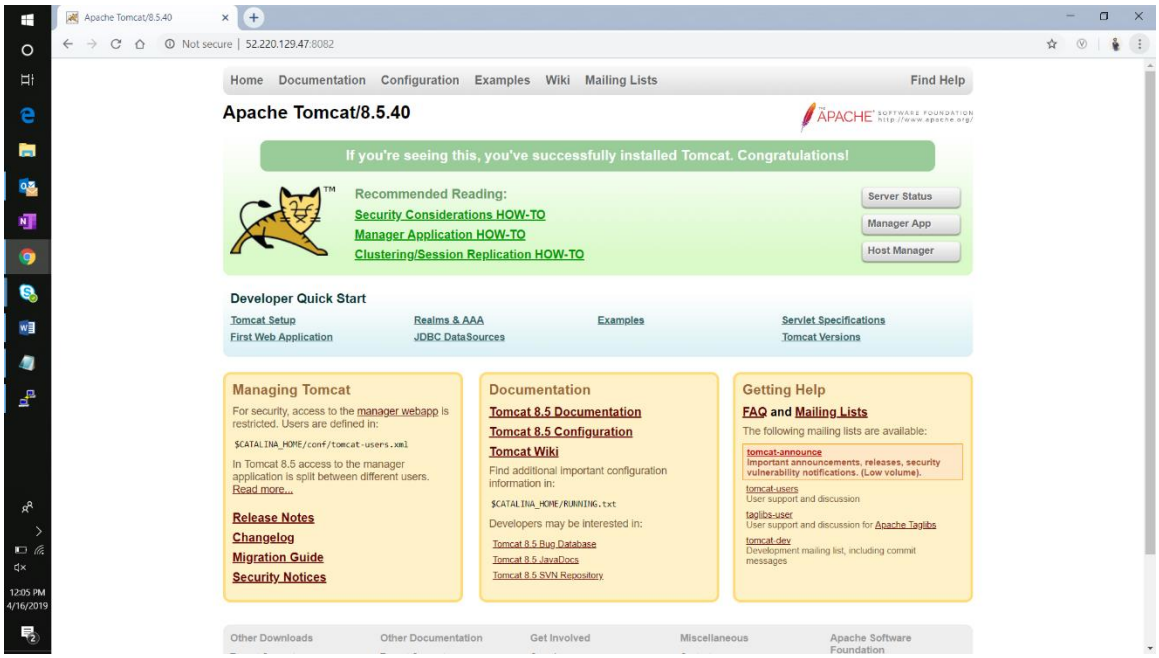
Since the image is available on our DockerHub repository and the last build image has the latest tag, all we need to do is pull the image and run it using the docker run command.

```
stage('Run Container') {
    sh "sudo docker run -p 8082:8080 -d neerajagarwal7/devops-201"
}
```

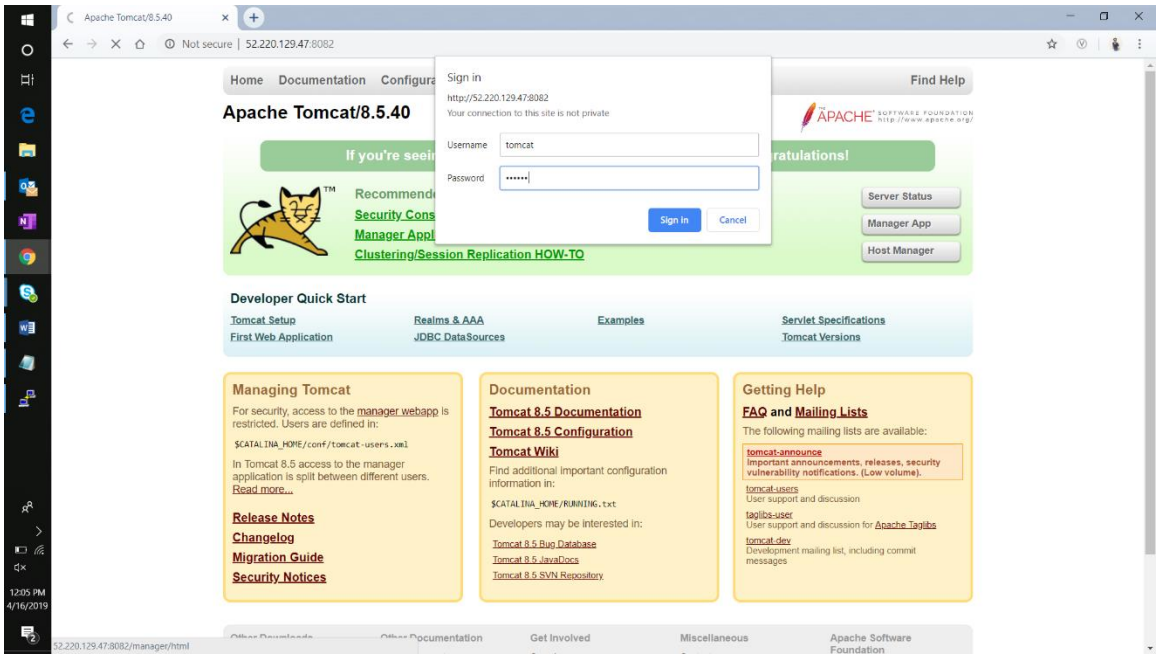
```
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Run Container)
[Pipeline] sh
+ sudo docker run -p 8082:8080 -d neerajagarwal7/devops-201
22ac7e32b1d72e505b586673069f71b77a6f899ca8b8ca48041c67f86a7488b4
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
```

Do a port mapping to some other port other than 8080. I have used port 8082 for this purpose

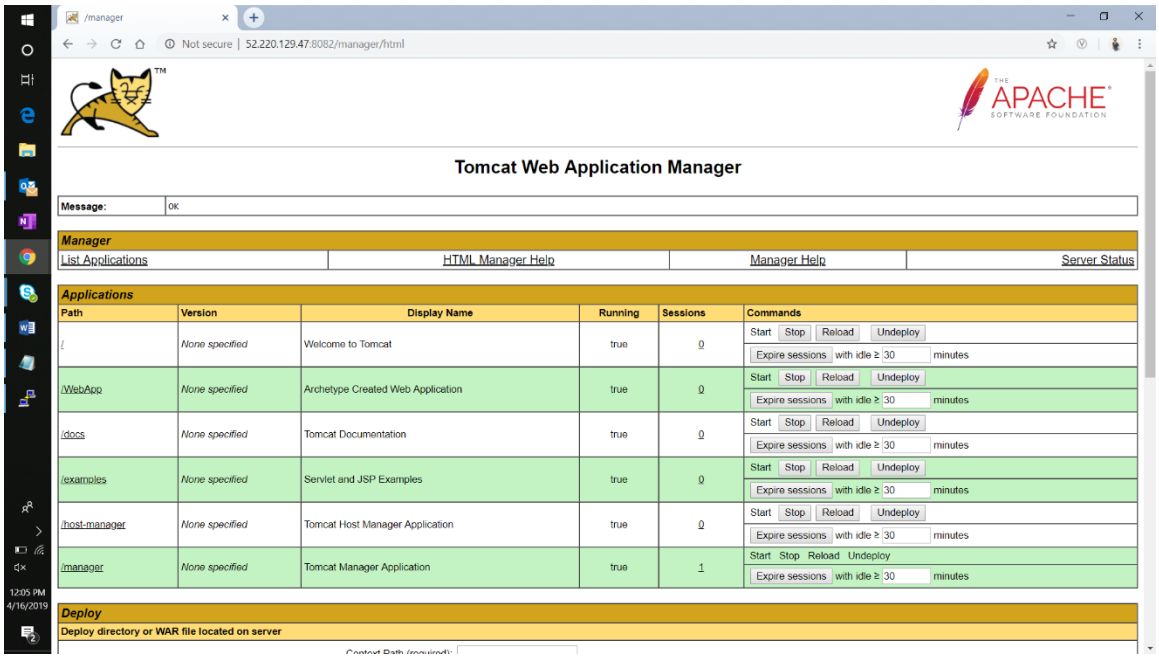
Open up your browser and open the tomcat webpage on port 8082



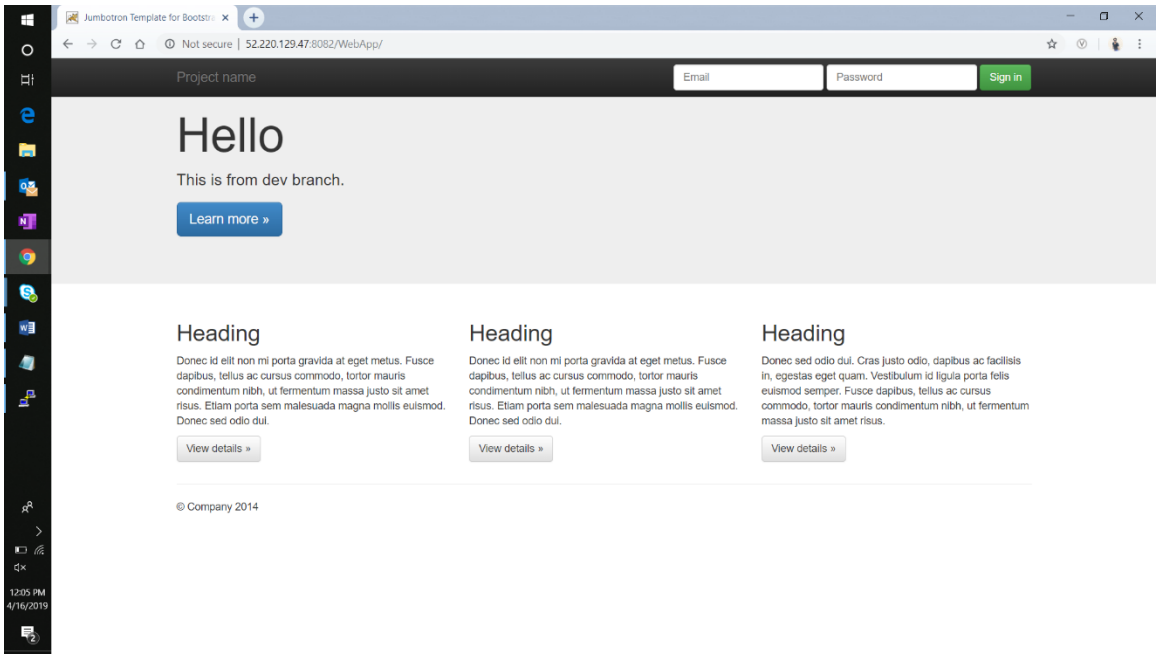
Tomcat Home page



Enter login credentials



Manager app where we can see our **WebApp** deployed



Application running on tomcat

Conclusion

This concludes the technical documentation for the solution to the project statement given.

As required I have used,

- Docker – To build the application in a Docker container and push it to Docker Hub
- Docker Hub – To store the Docker image
- GitHub – To store the application code and track its revisions
- Git – To connect and push files from local system to GitHub
- Linux (Ubuntu) – As a base operating system to start and execute the project
- Jenkins – To automate the deployment process during continuous integration

All the necessary files including application source code, Docker file, jenkins pipeline script, Configuration files etc can be found on my git hub repository.

https://github.com/NeerajAgarwal7/Devops-201-maven.git