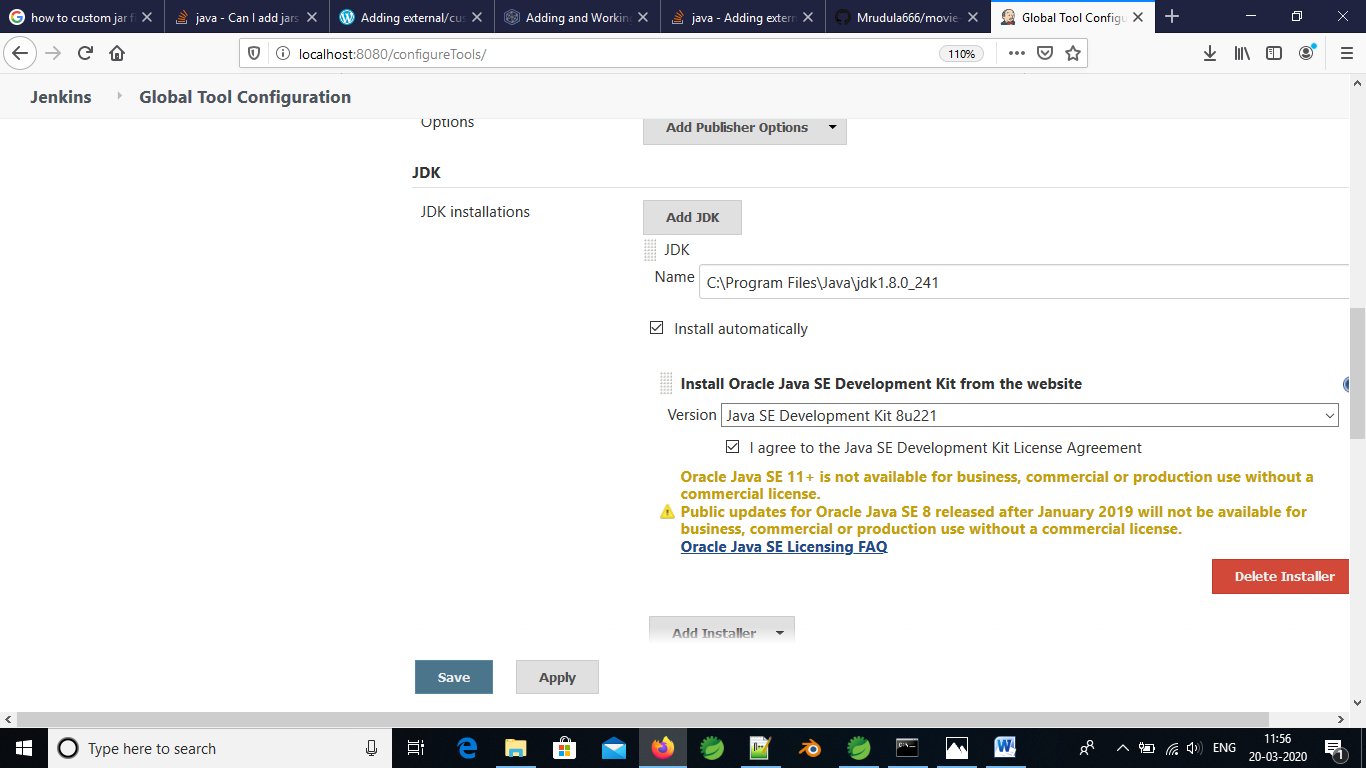
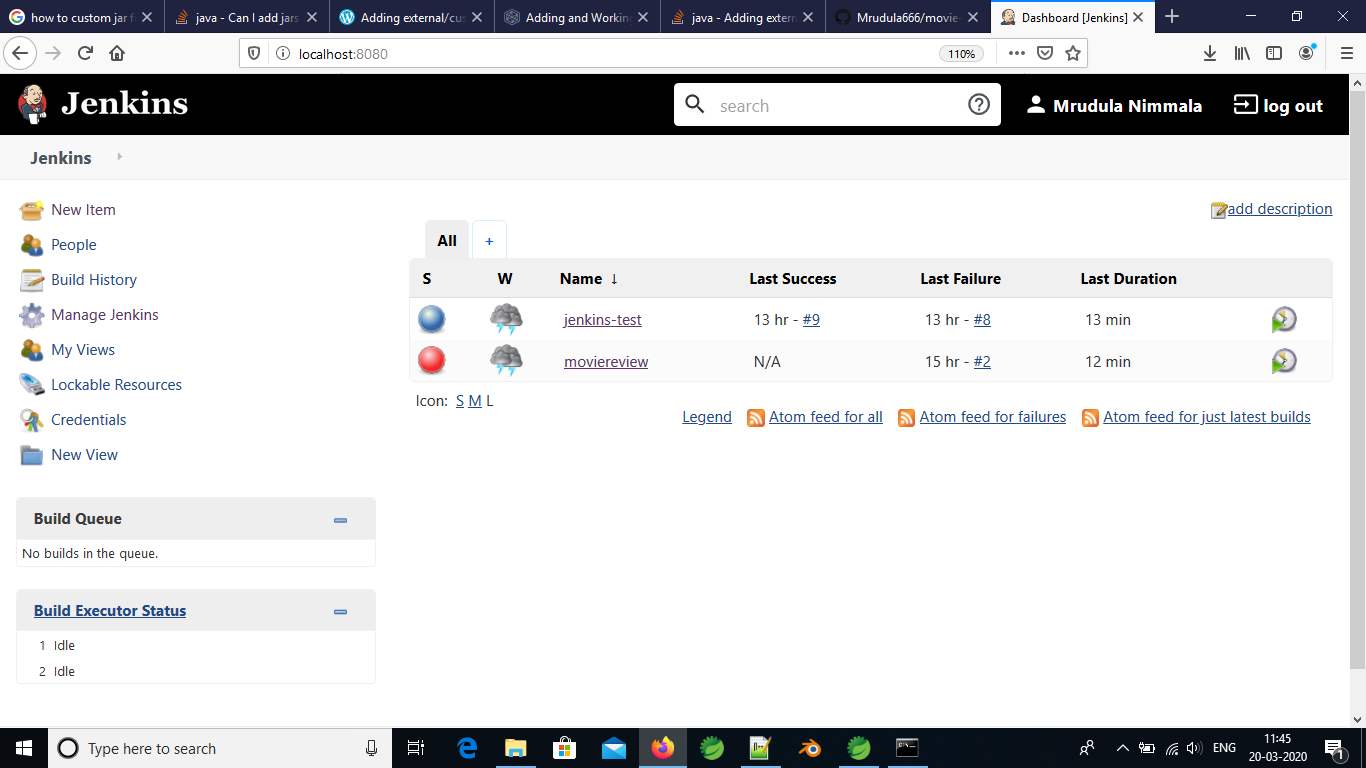
1. I have implemented the movie-catalog-service and uploaded in git.

url for git:

<https://github.com/Mrudula666/movie-catalog-service.git>

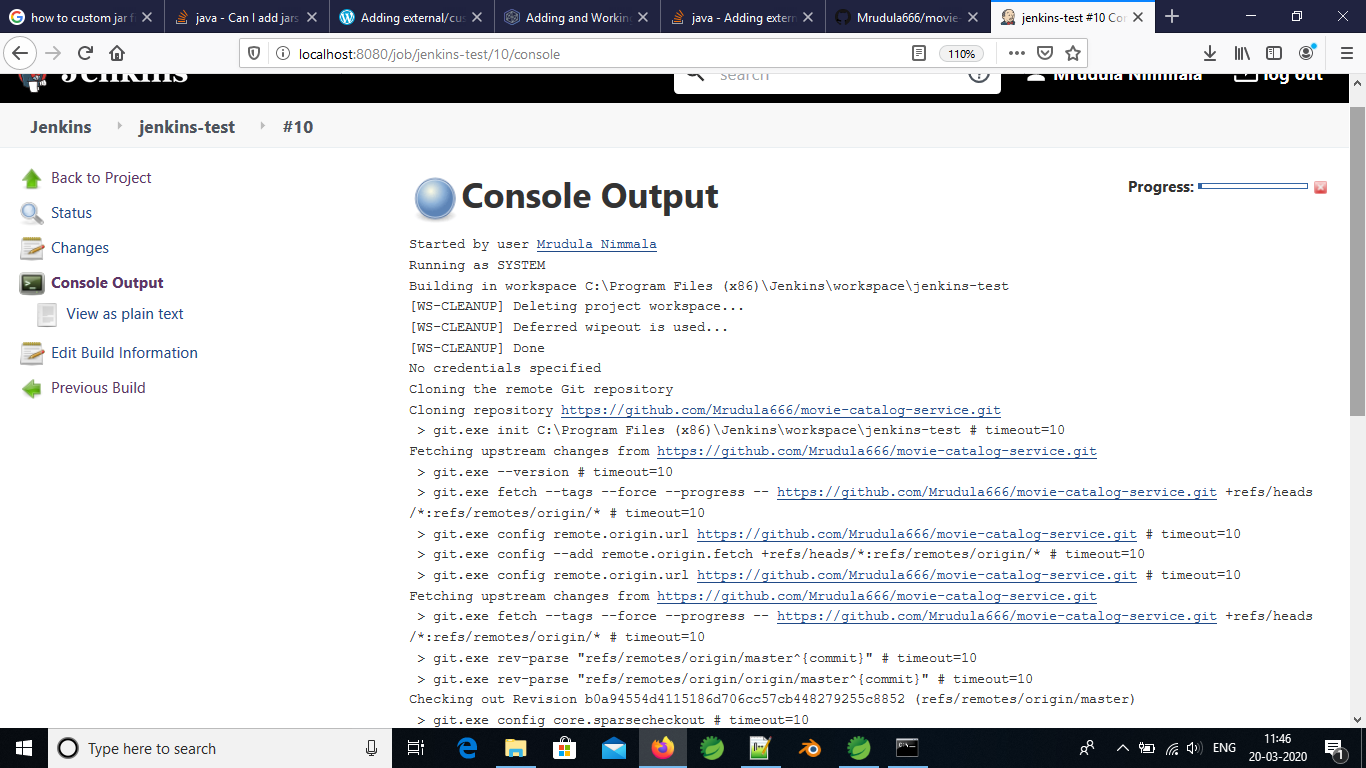
1. Configured global settings in Jenkins for jdk installations.
   1. 
2. Uploaded the maven project in Jenkins.

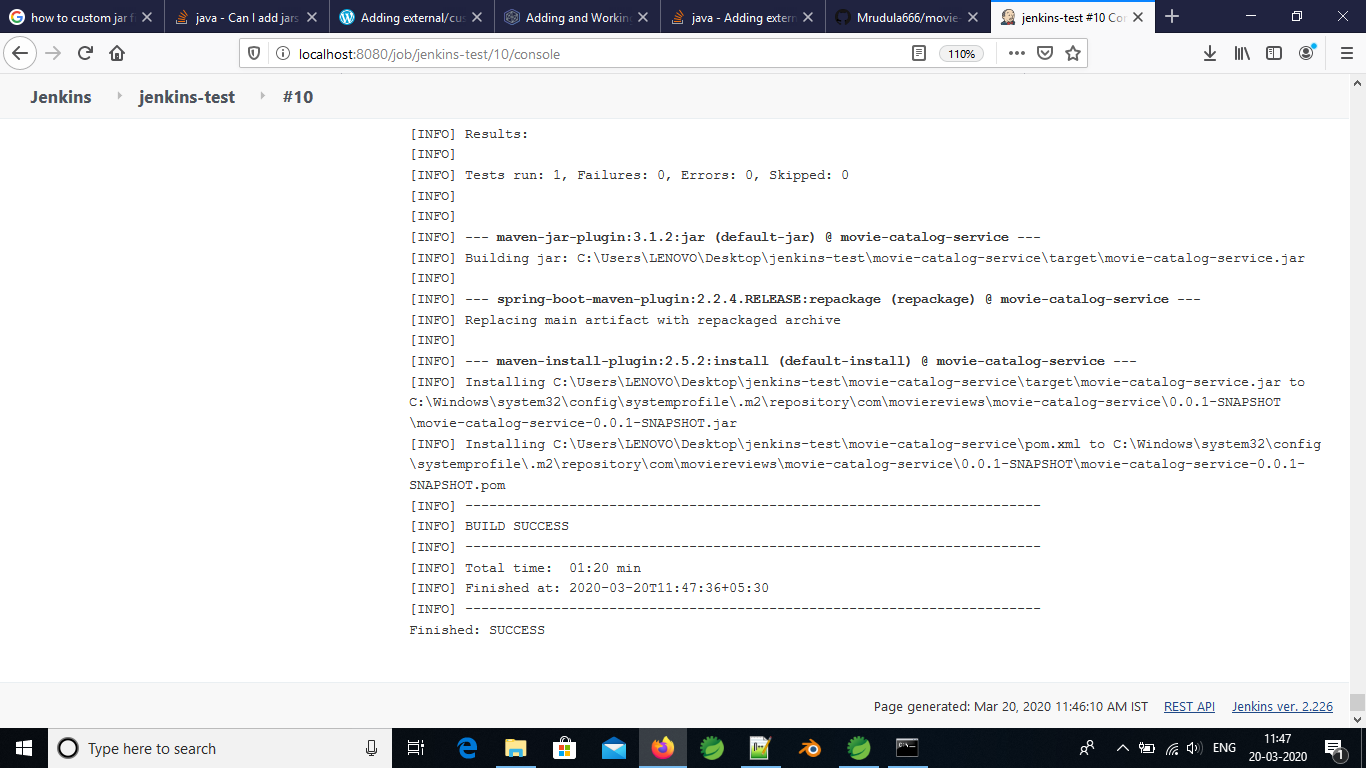


1. Configured the git url and the maven settings in Jenkins from manage Jenkins option



1. Run the maven command with following console output

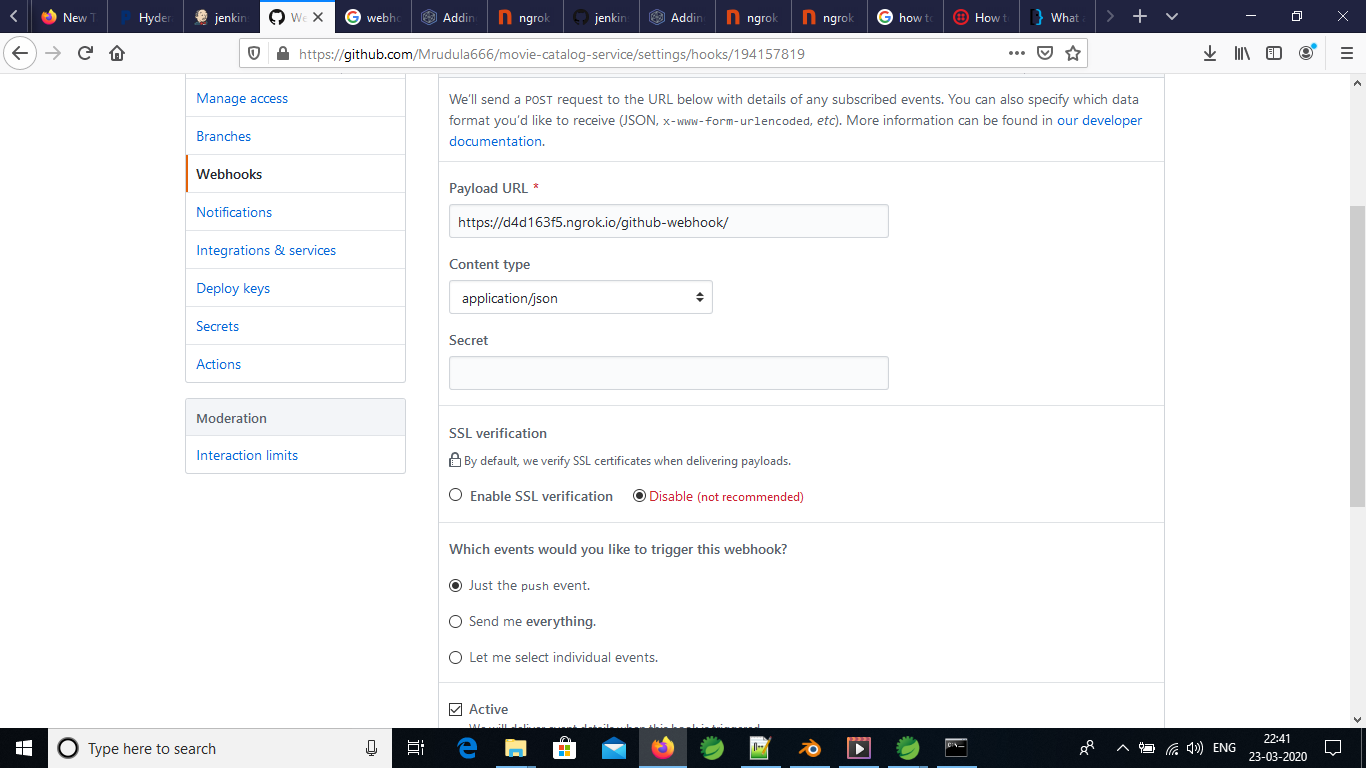




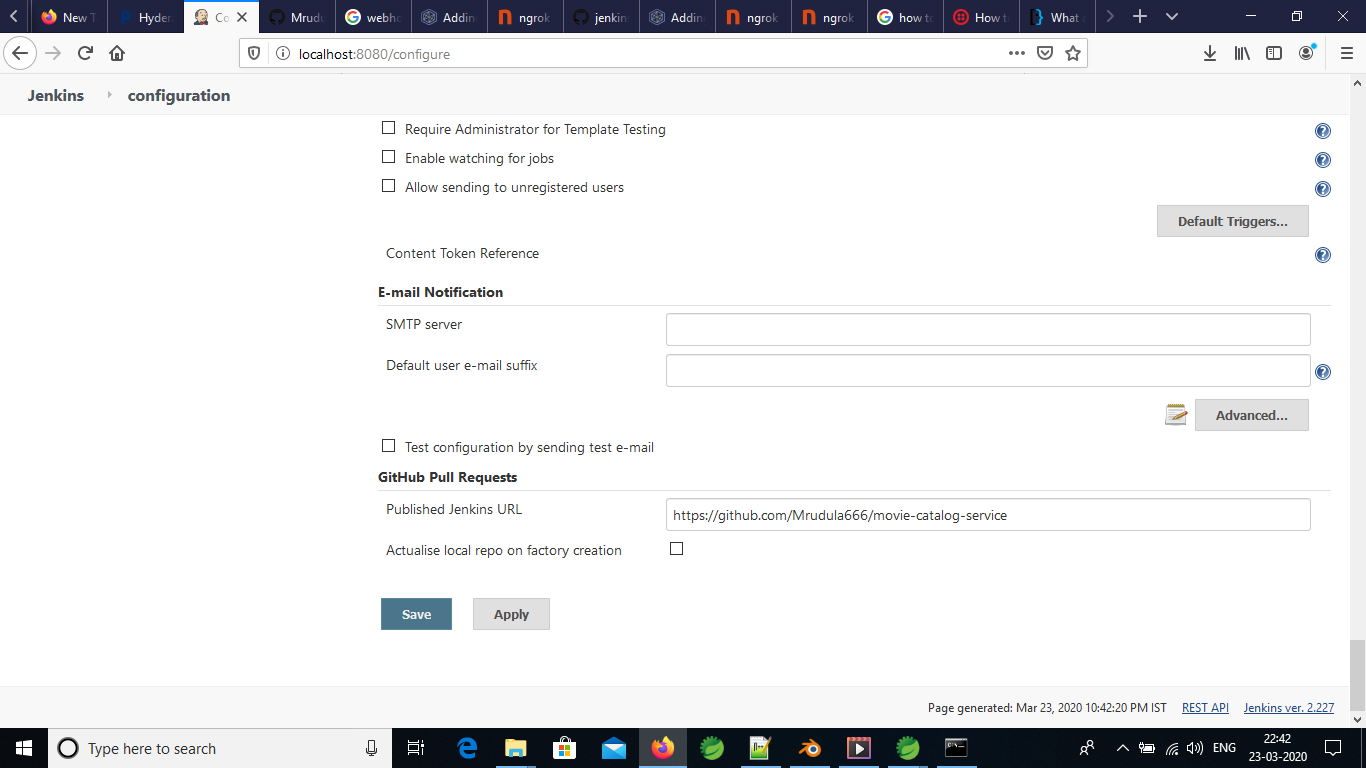
Date: 23-march-2020

**To configure the automatic build trigger for each commit.**

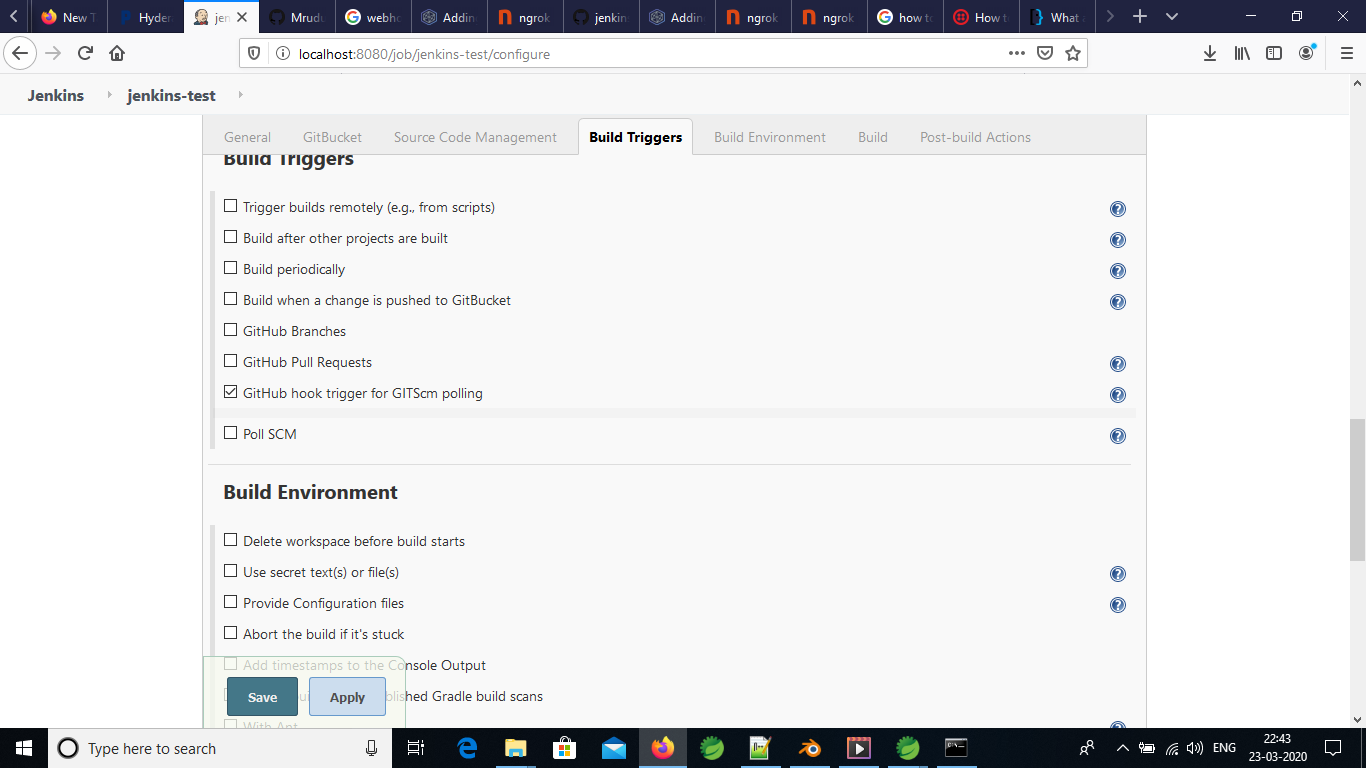
1. To configure the webhook we need to convert the localhost to public network.
2. To do this, we have to install ngrok in the local system and set the path in environment variables.
3. Added the webhook in github as show below.



1. Configure the git pull requests in Jenkins with git repository



1. Modify the manage configure section in job to mark “GitHub hook trigger for GITScm polling” as below.



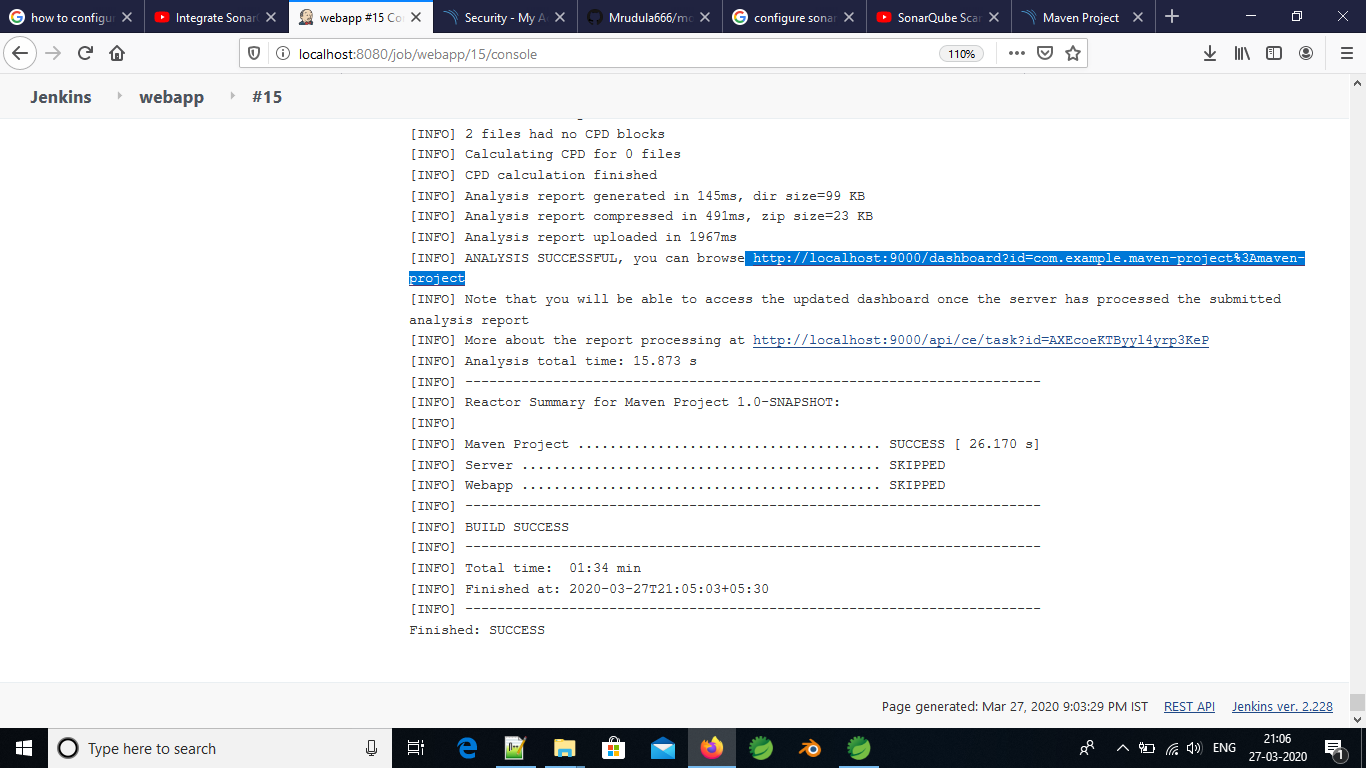
1. Refresh the Jenkins page.

Date: 27-march-2020

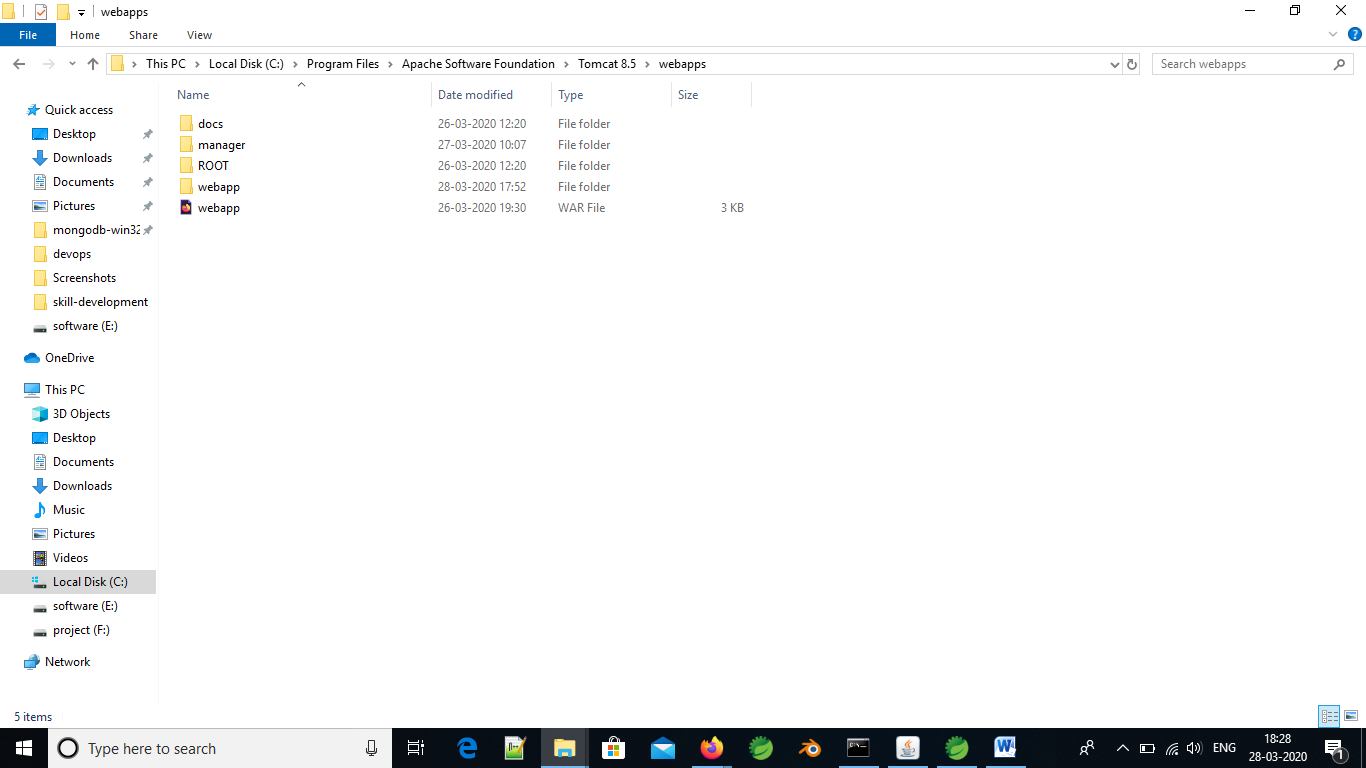
Task: Run the Jenkins created war in tomcat server locally

1. At Jenkins, set the goal in job as clean package where the war file is created at

C:\Program Files (x86)\Jenkins\workspace\webapp\webapp\target\webapp.war



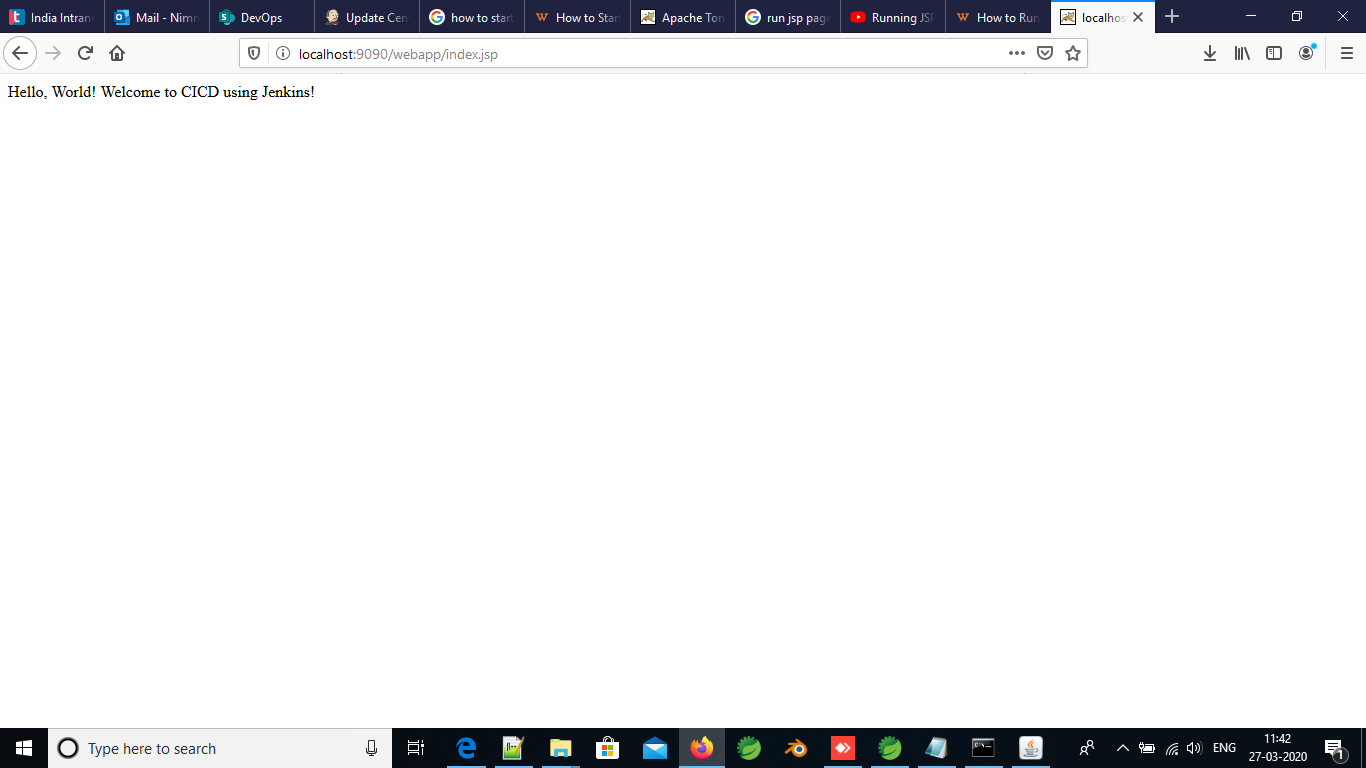
1. Copied the webapp.war file at C:\Program Files\Apache Software Foundation\Tomcat 8.5\webapps location.



1. Started the tomcat server

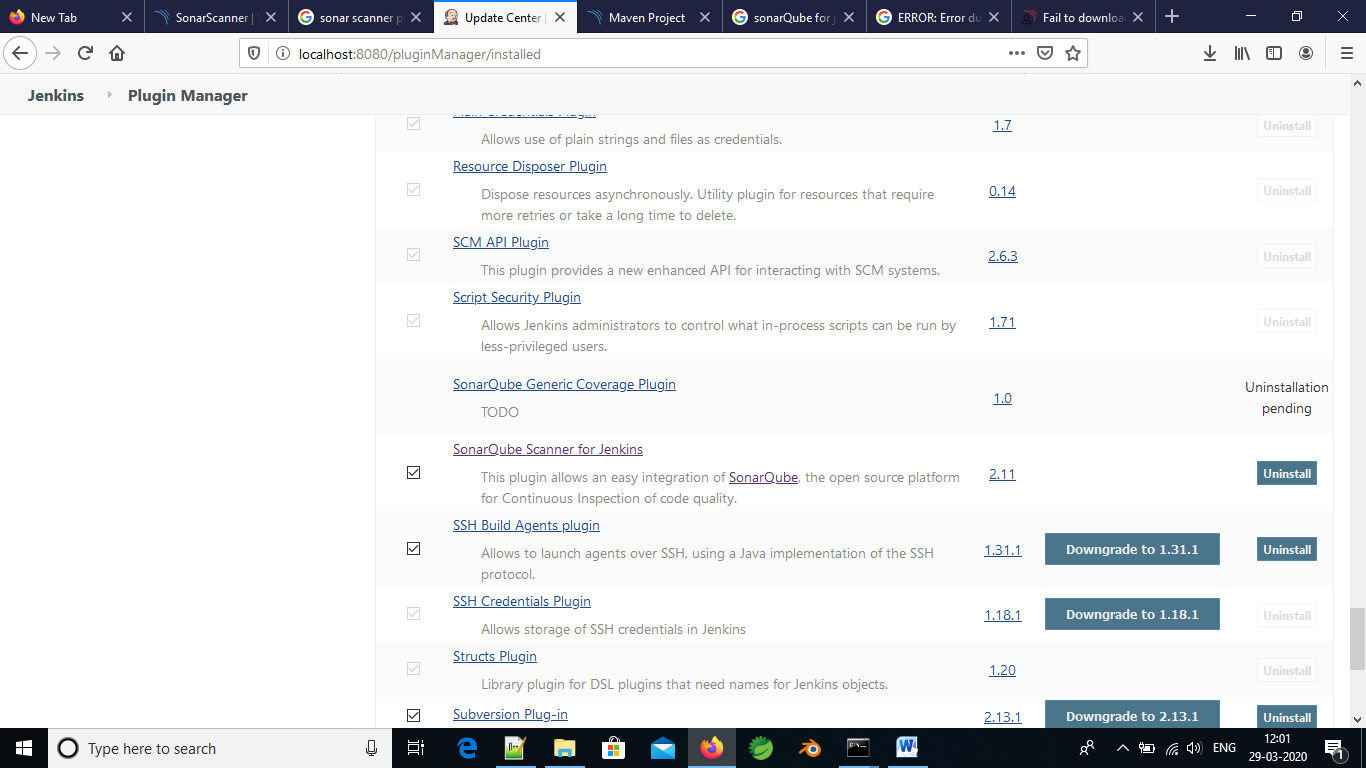


1. Ran the server at 9090 with the url <http://localhost:9090/webapp>

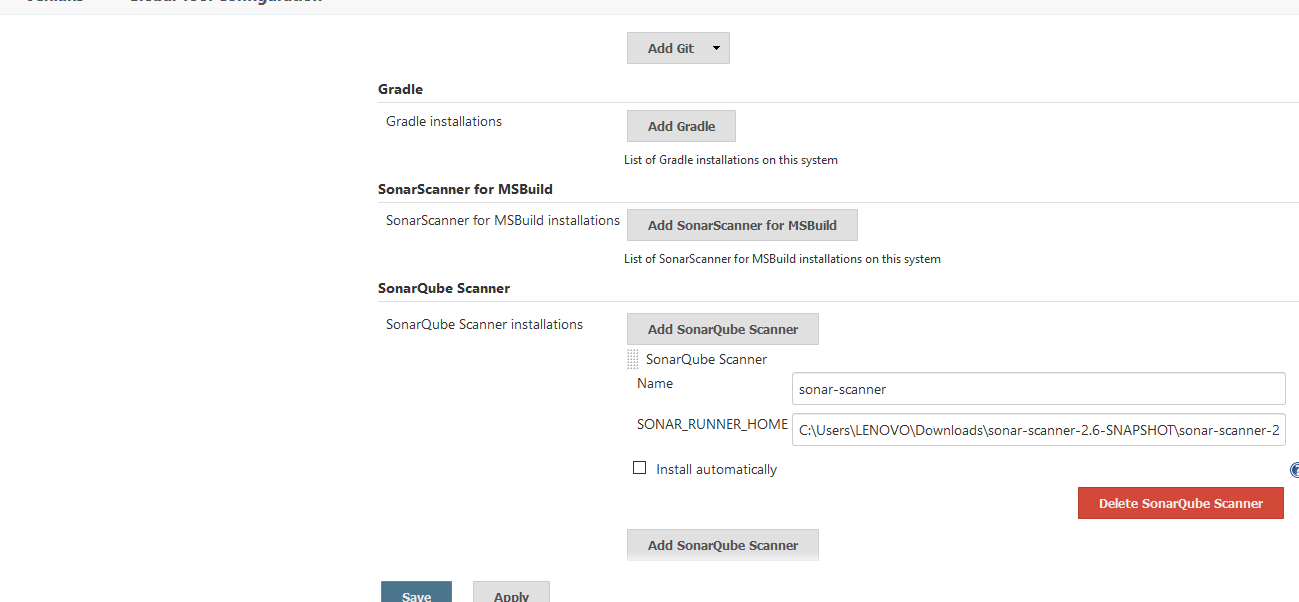


Task: Run the sonarQube in Jenkins

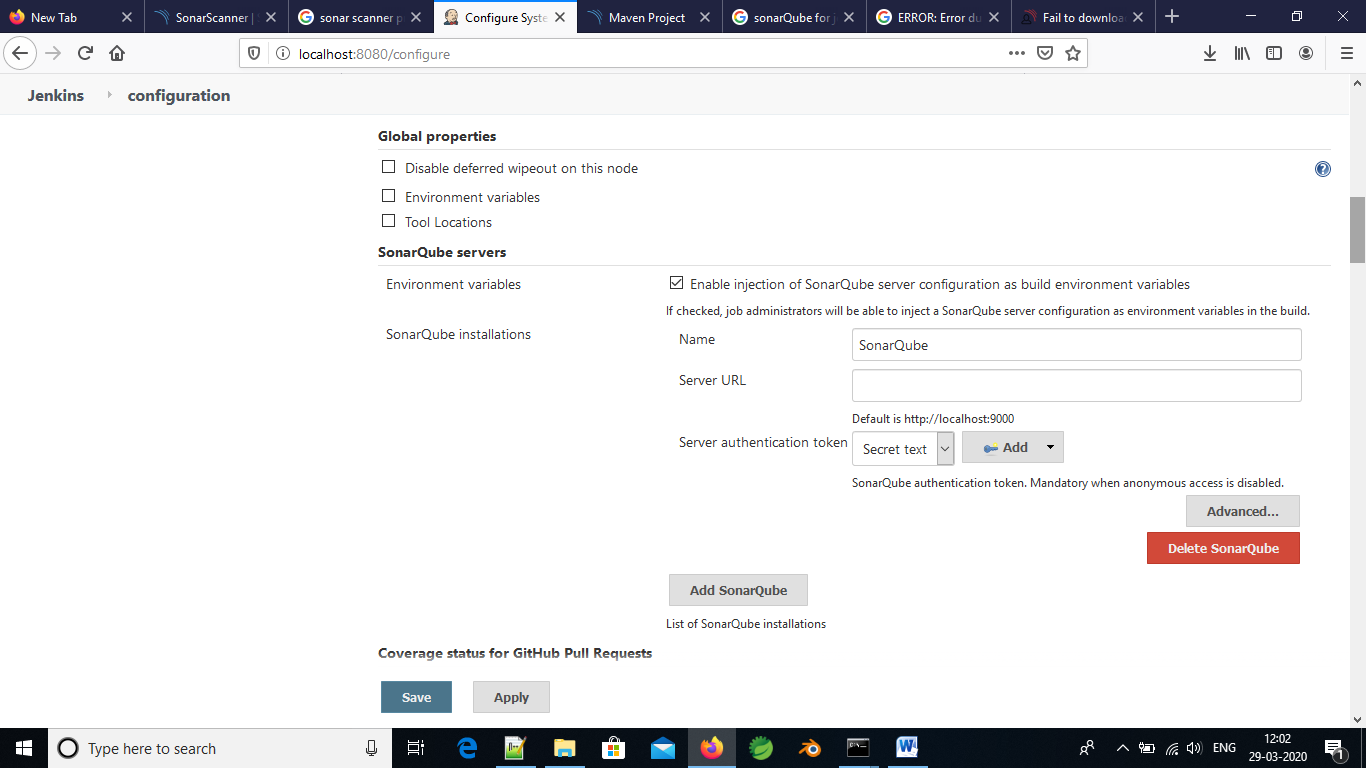
1. Install the sonarQube Scanner in Jenkins at manage Jenkins



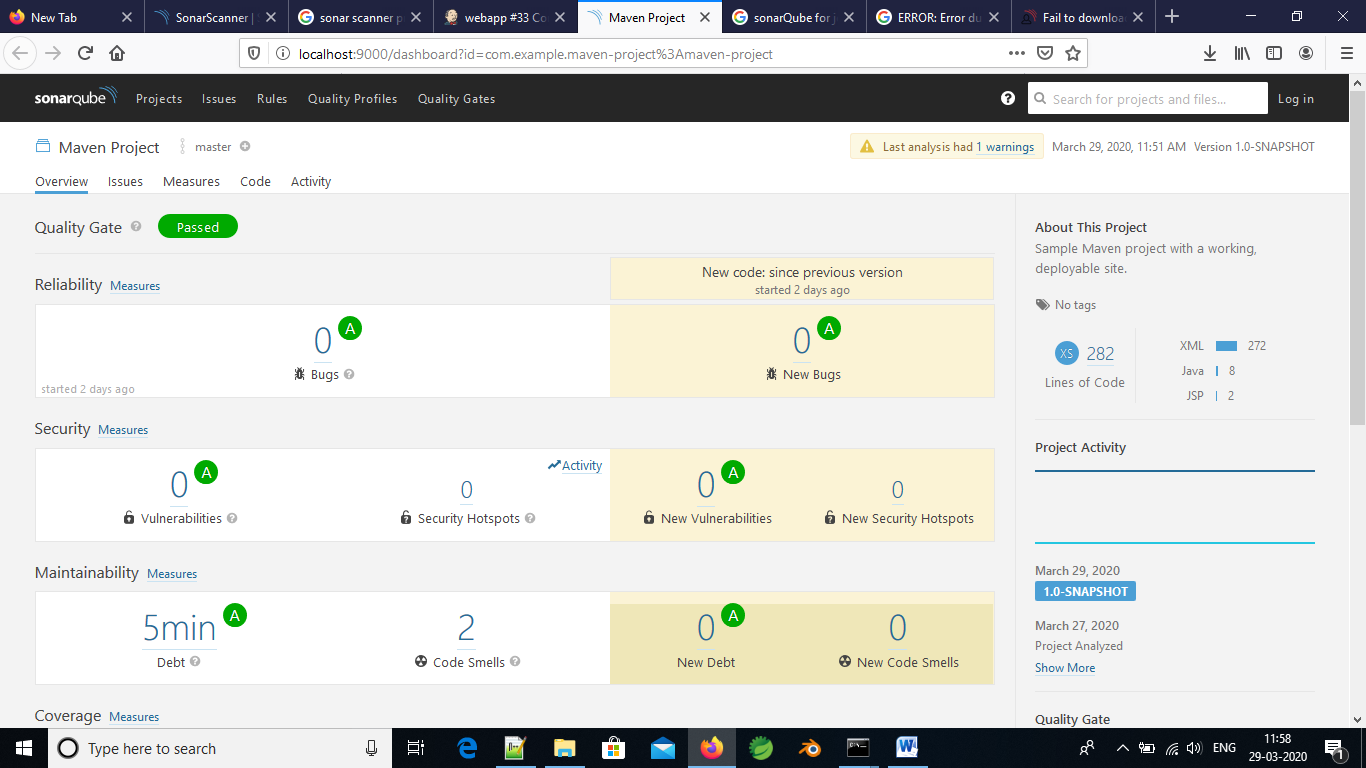
1. Configure the Jenkins in global tool configuration



1. Configure the sonarQube server installation in configure option



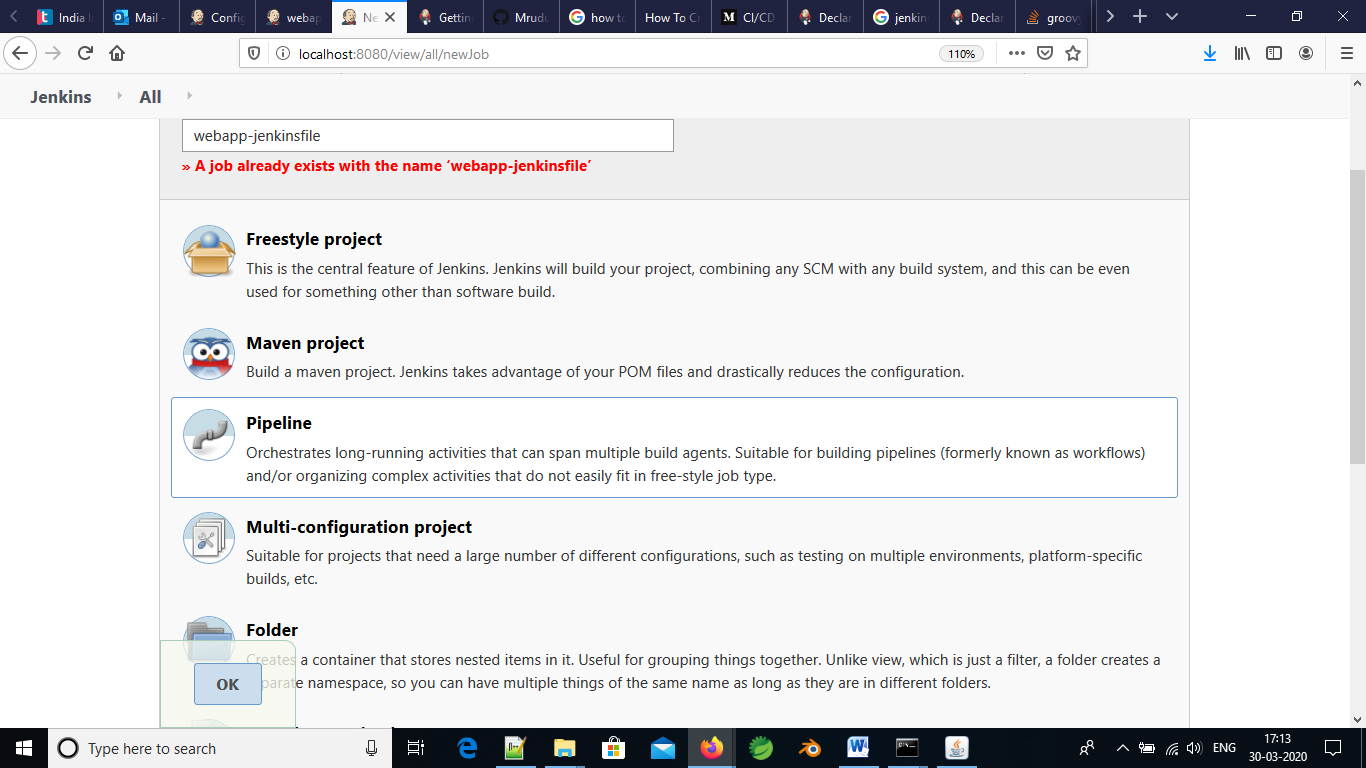
1. Build the job.
2. See the results in <http://localhost:9000>



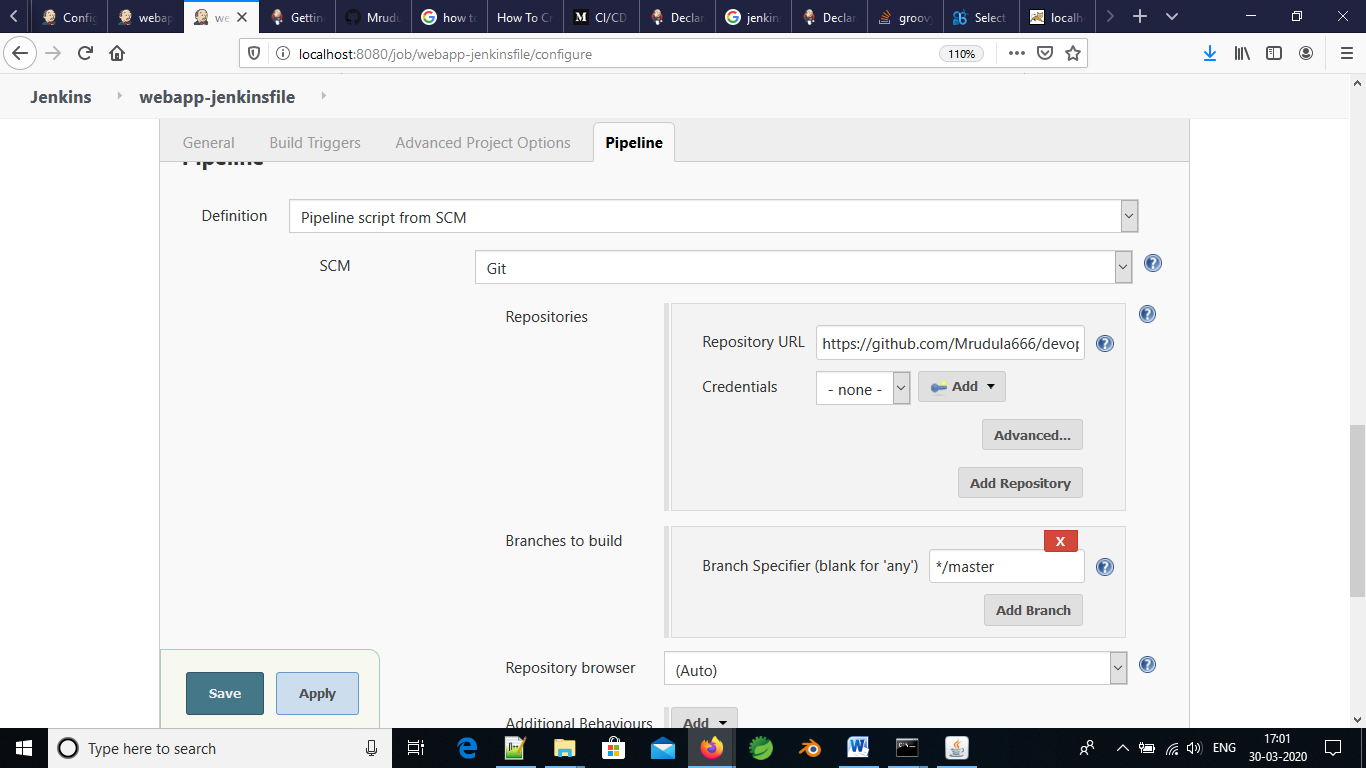
Date: 30/03/2020

Task: Run the war file that is created by the Jenkins job using the Jenkins pipeline

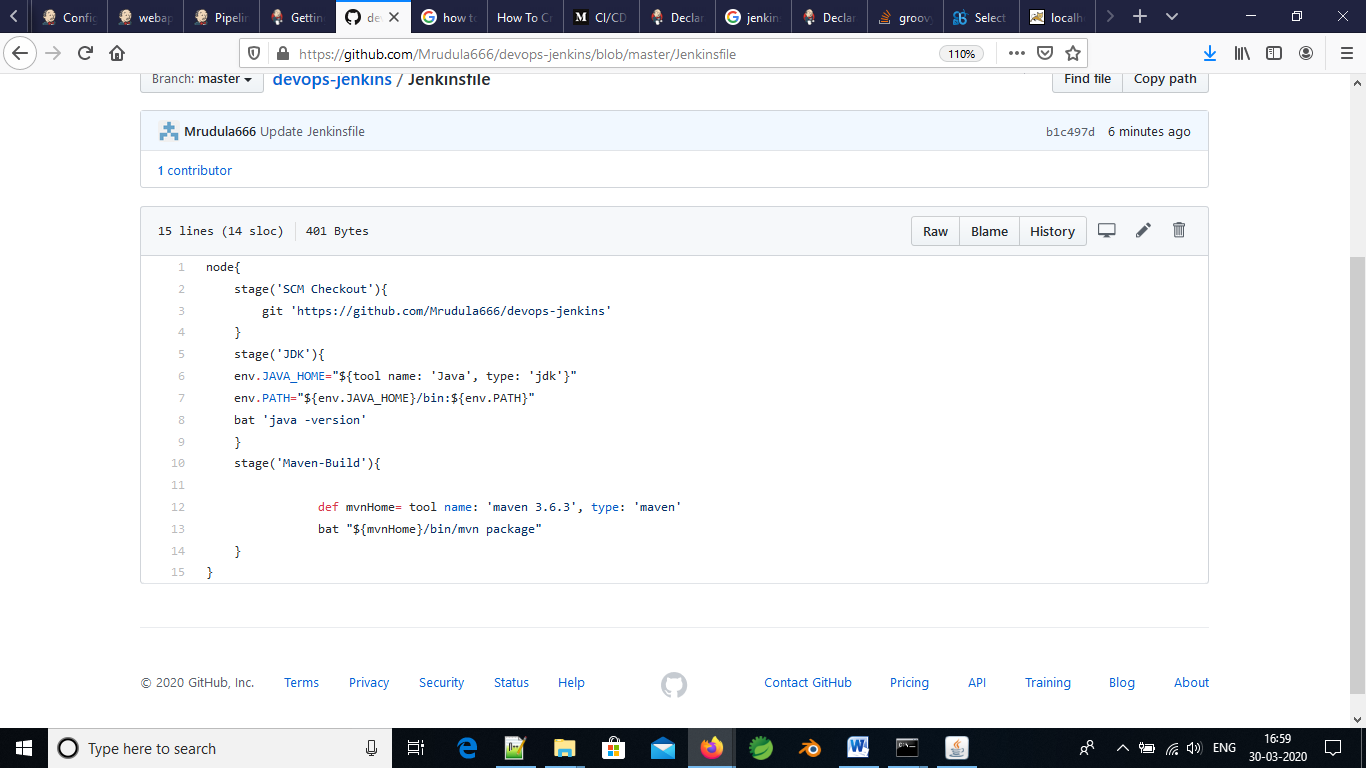
1. Create a new **Pipeline Job**



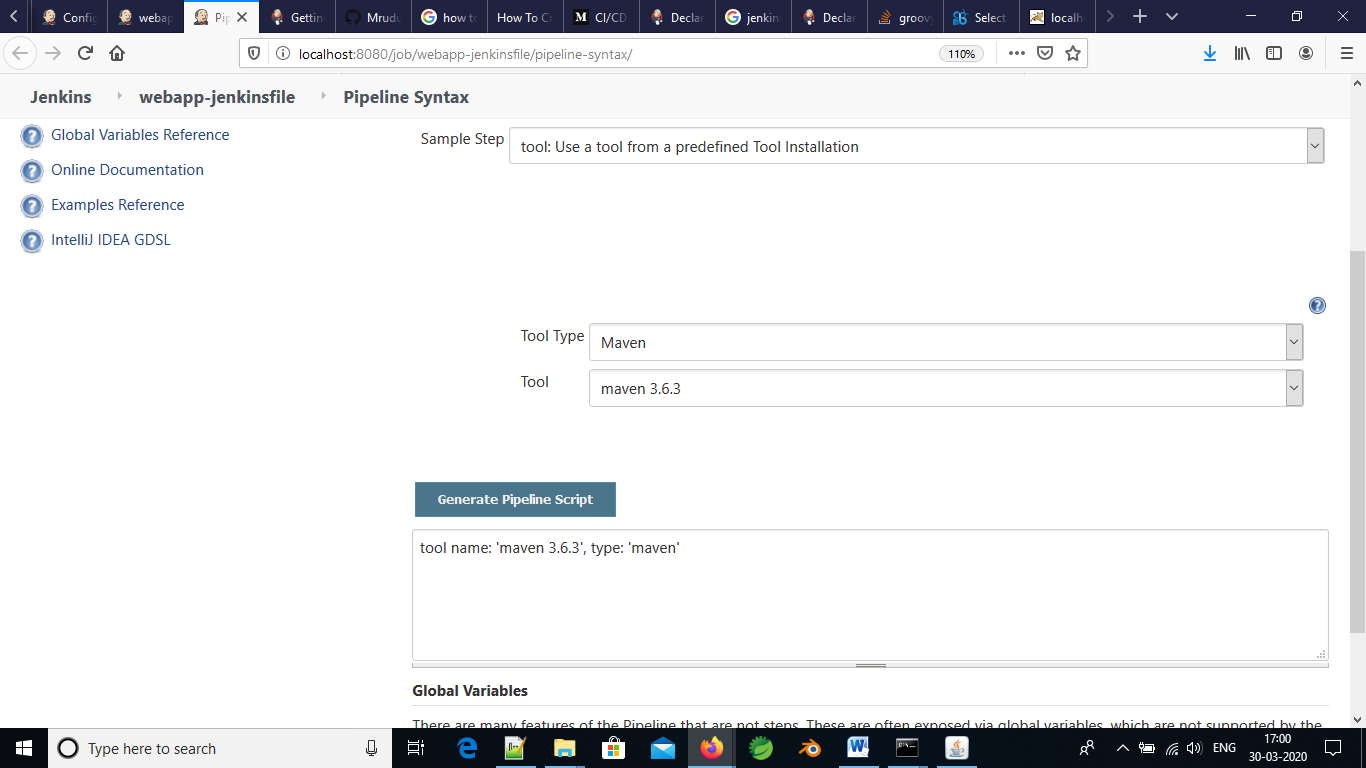
1. In the job go to **PIPELINE** section, where in the Definition section select **Pipeline script from SCM.** Give the essential git repository details in the SCM section



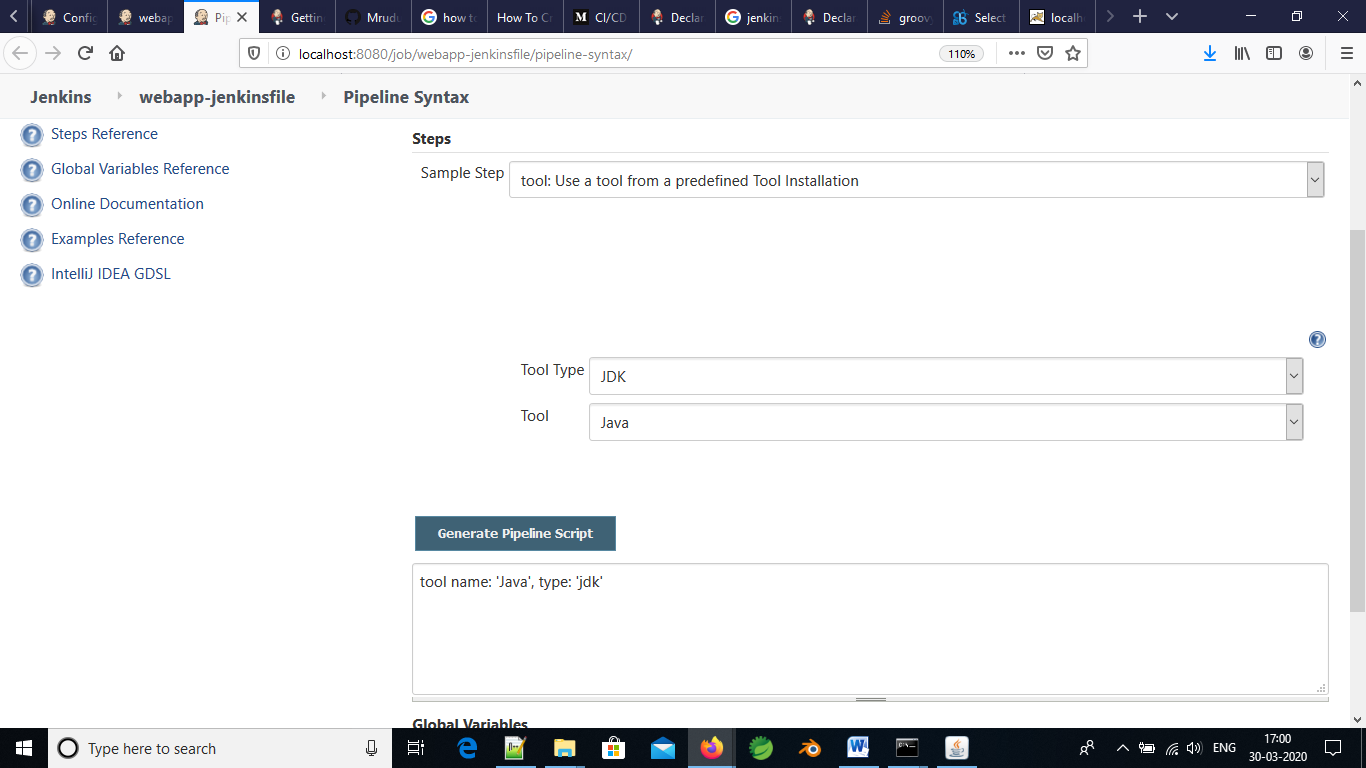
1. In git repository, create Jenkinsfile that has the configurations of the jdk and maven details. And also git url as follows



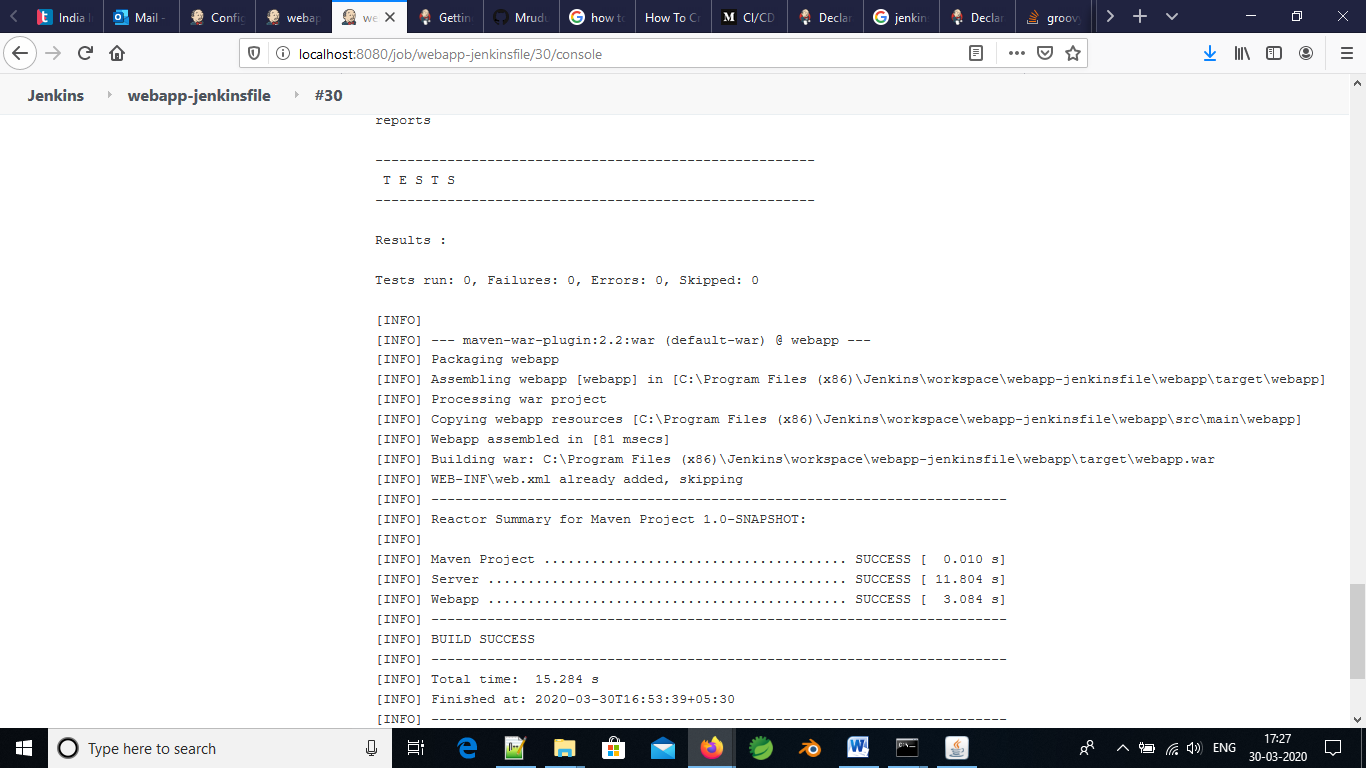
1. For maven version and jdk versions, go to the **pipeline syntax** the mention the tools needed and press **generate pipeline script** as shown below.
2. For maven:



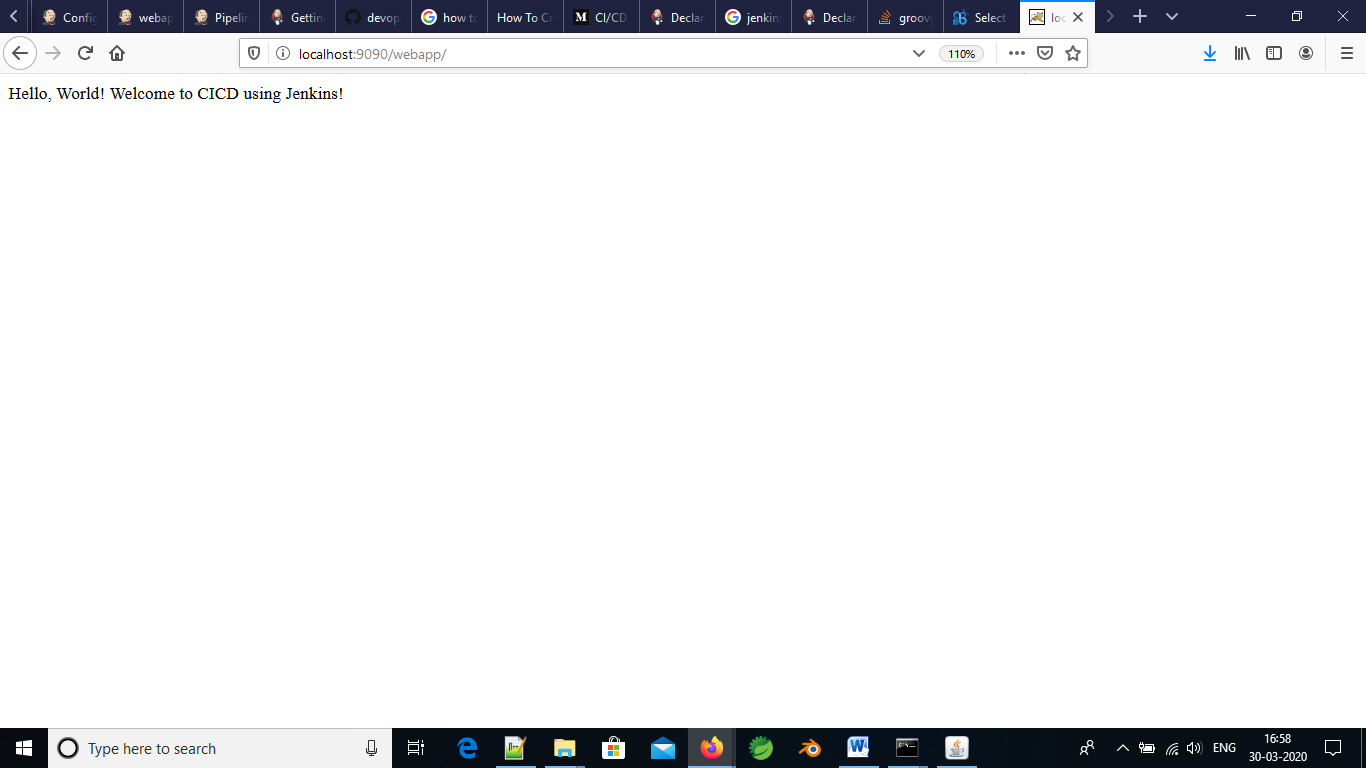
1. For Java tool:



1. Now build the job, and webapp.war file is created



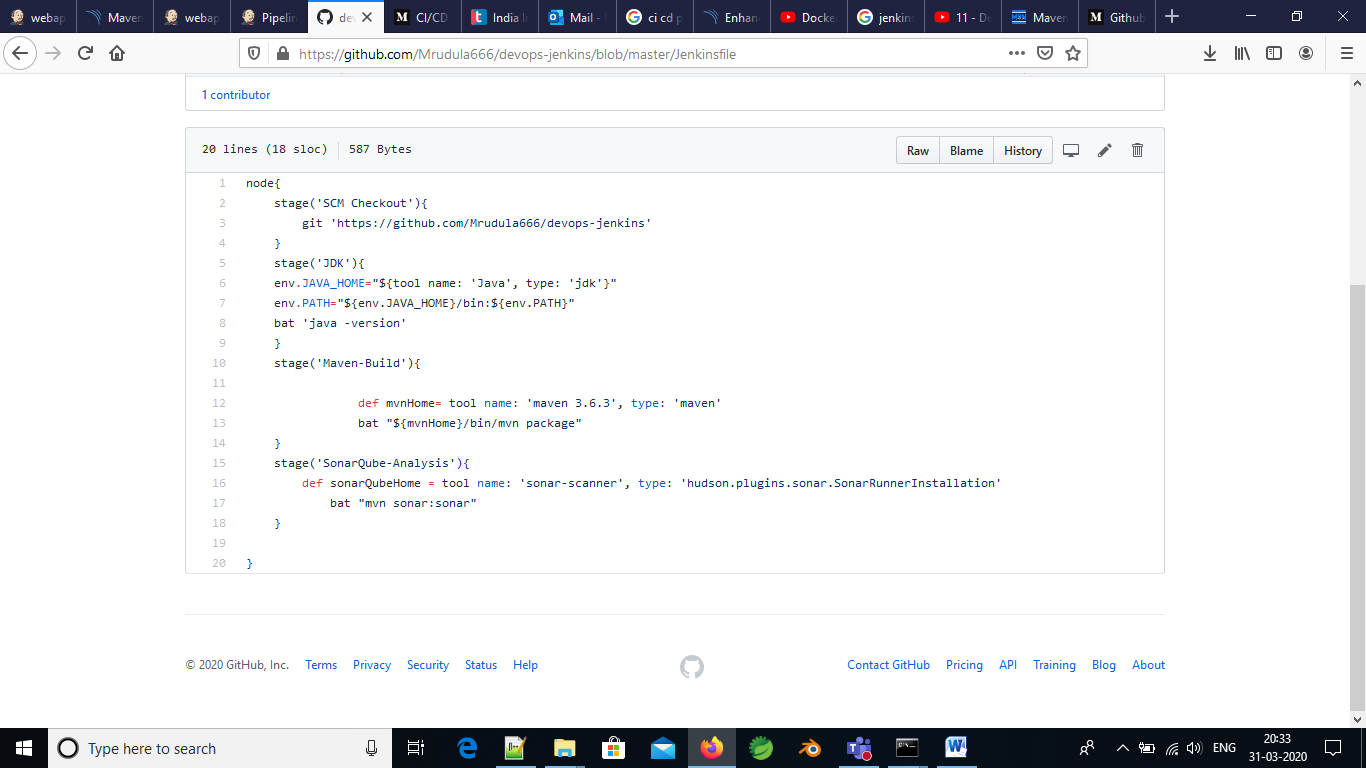
1. Start the tomcat server and run the war file as <http://localhost:9090/webapp> and see the below result



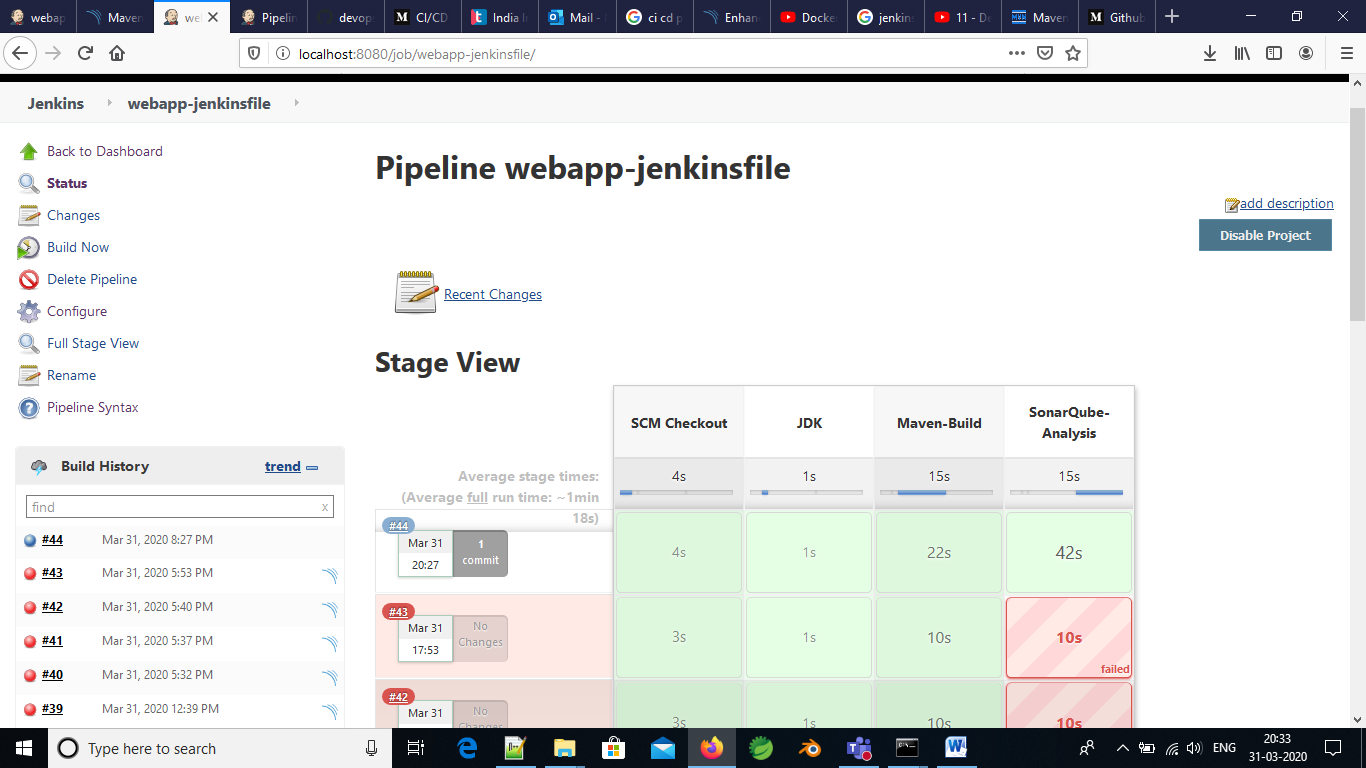
Date:31/March/2020

Aim: Adding the sonarQube server using pipeline script for above job in Jenkins

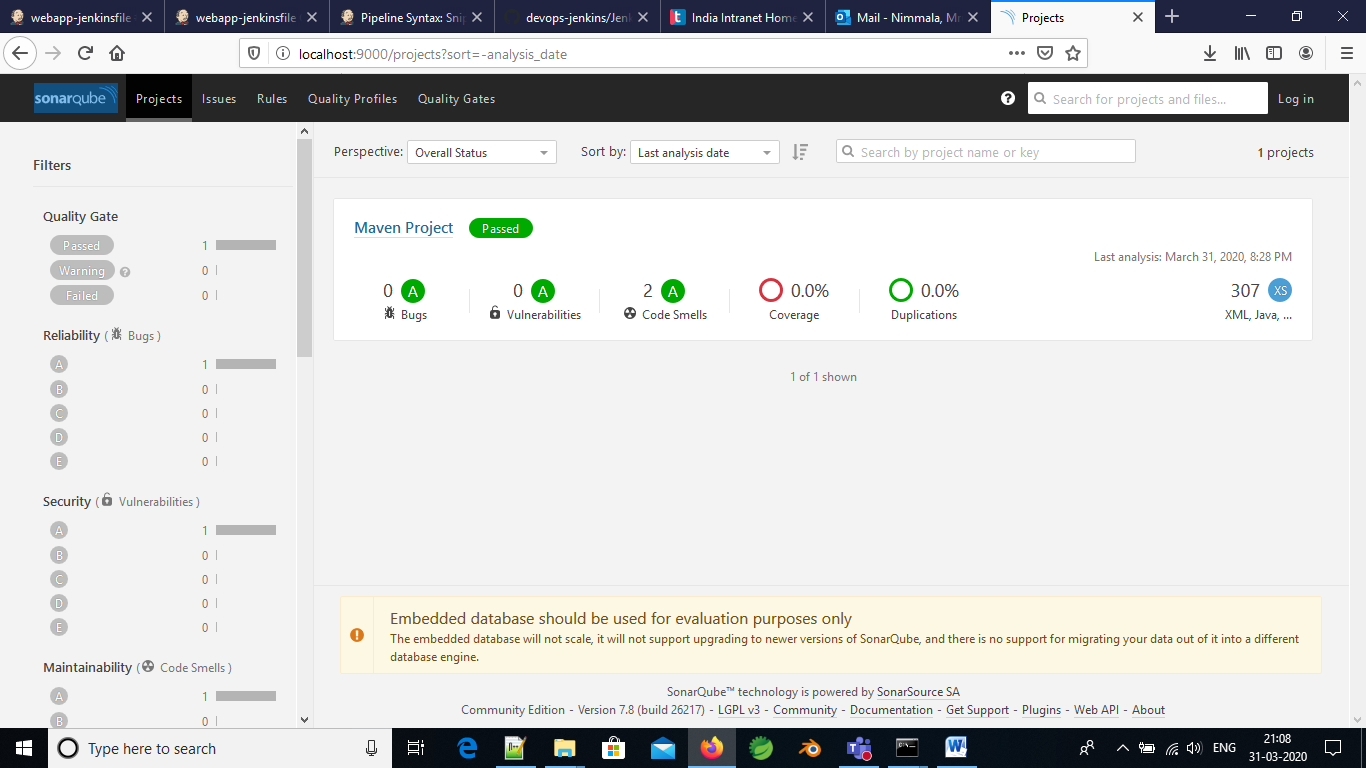
1. Just add the following script in Jenkins file and build the job.



1. Build the Jenkins job



1. See the results in <http://localhost:9000>

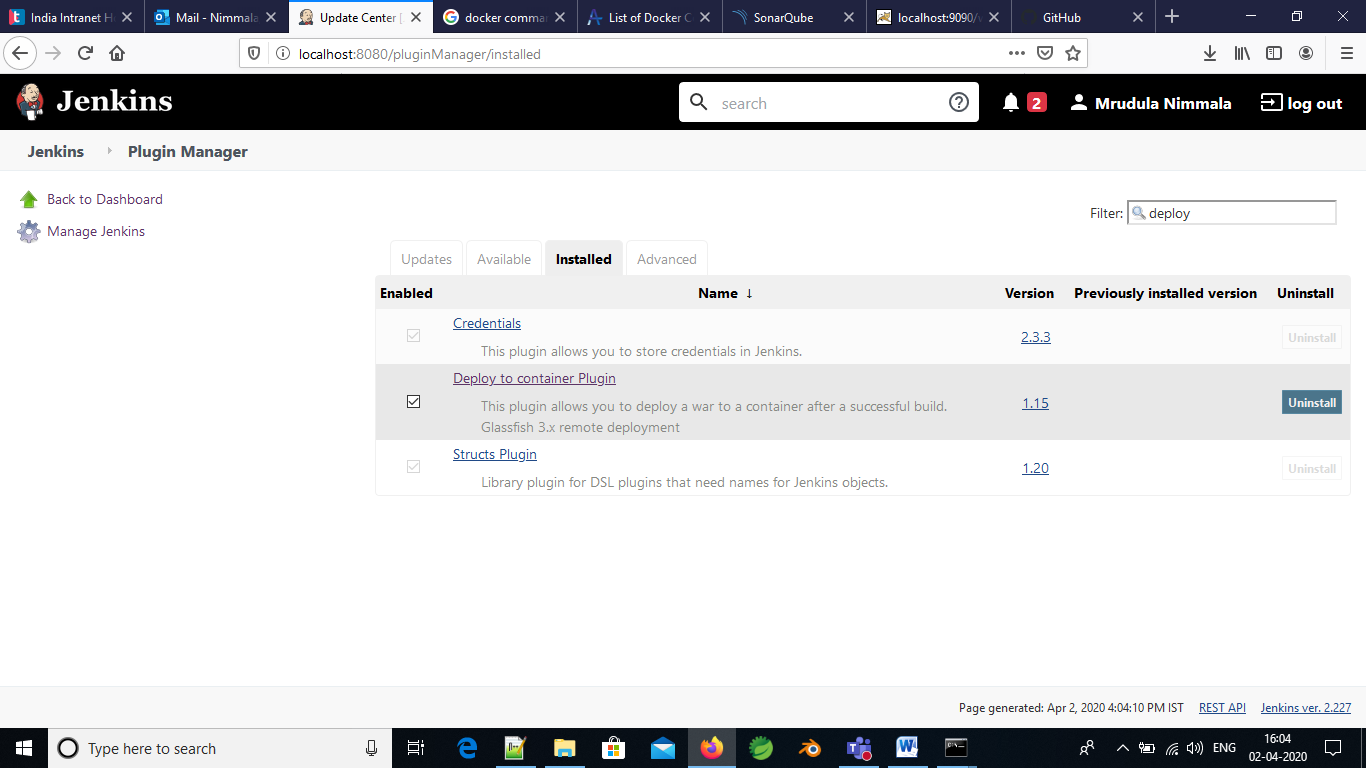


1. As we can see the there are 2 code smells in the project.

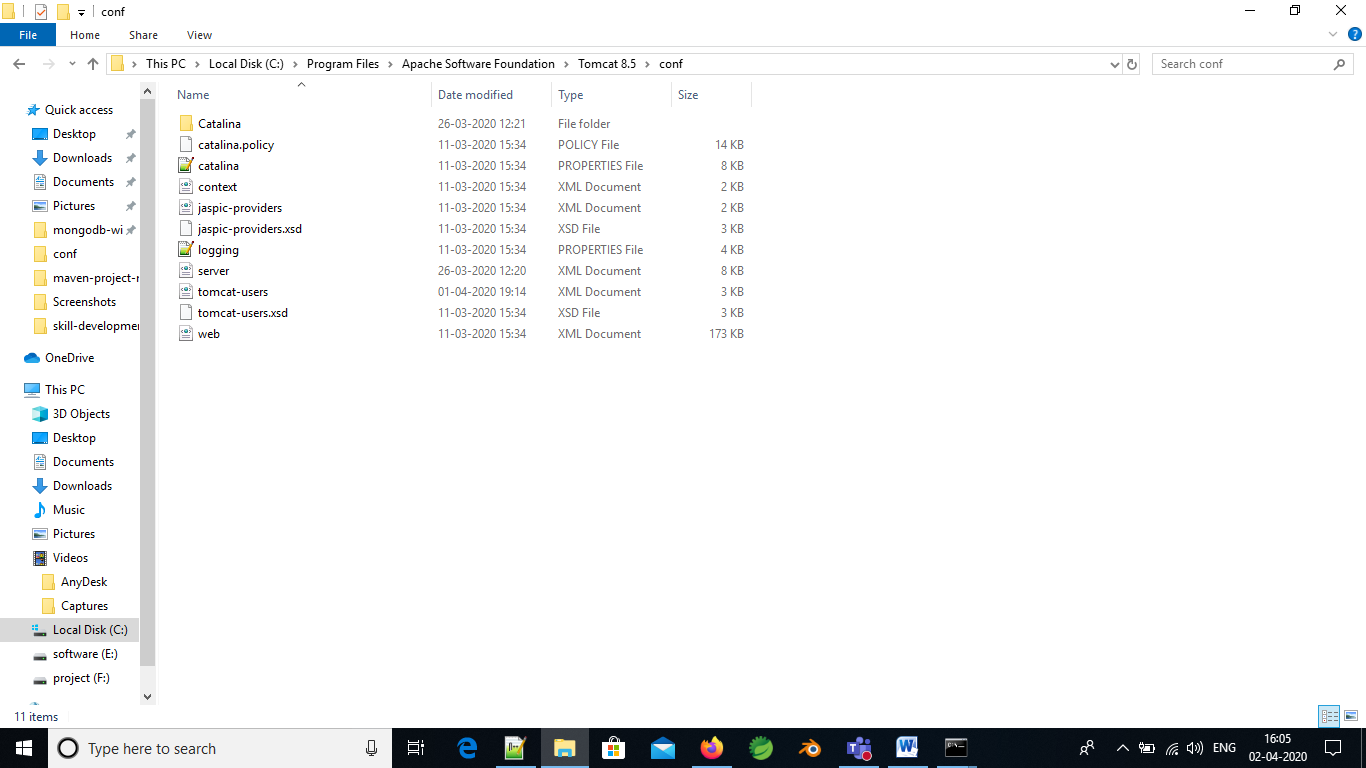
Date: 02-March-2020

Aim: Deployed the Tomcat server in Jenkins.

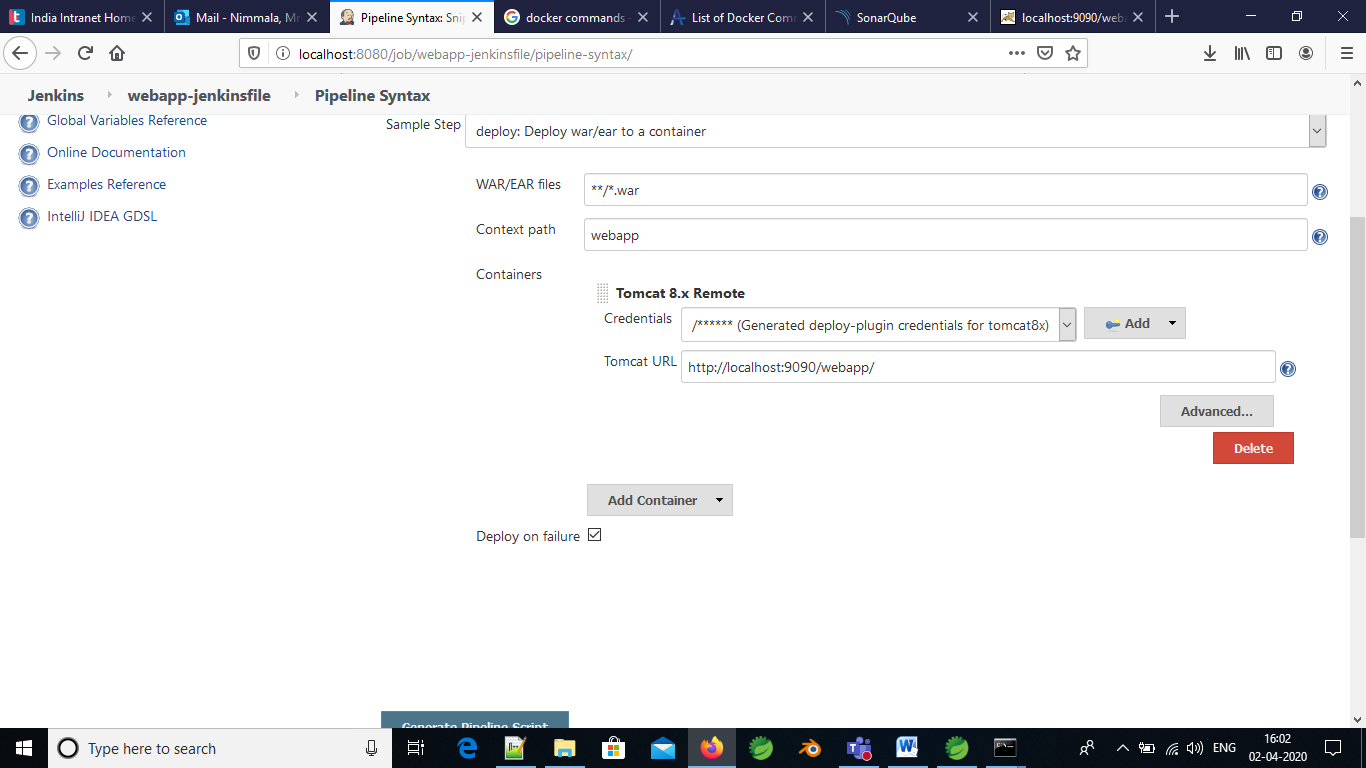
1. To add the tomcat server in Jenkins job Configure **Deploy to Container** in the manage Jenkins option



1. Add manager role, user name and password in the tomcat-users.xml in the location C:\Program Files\Apache Software Foundation\Tomcat 8.5\conf\tomcat-users.xml like below



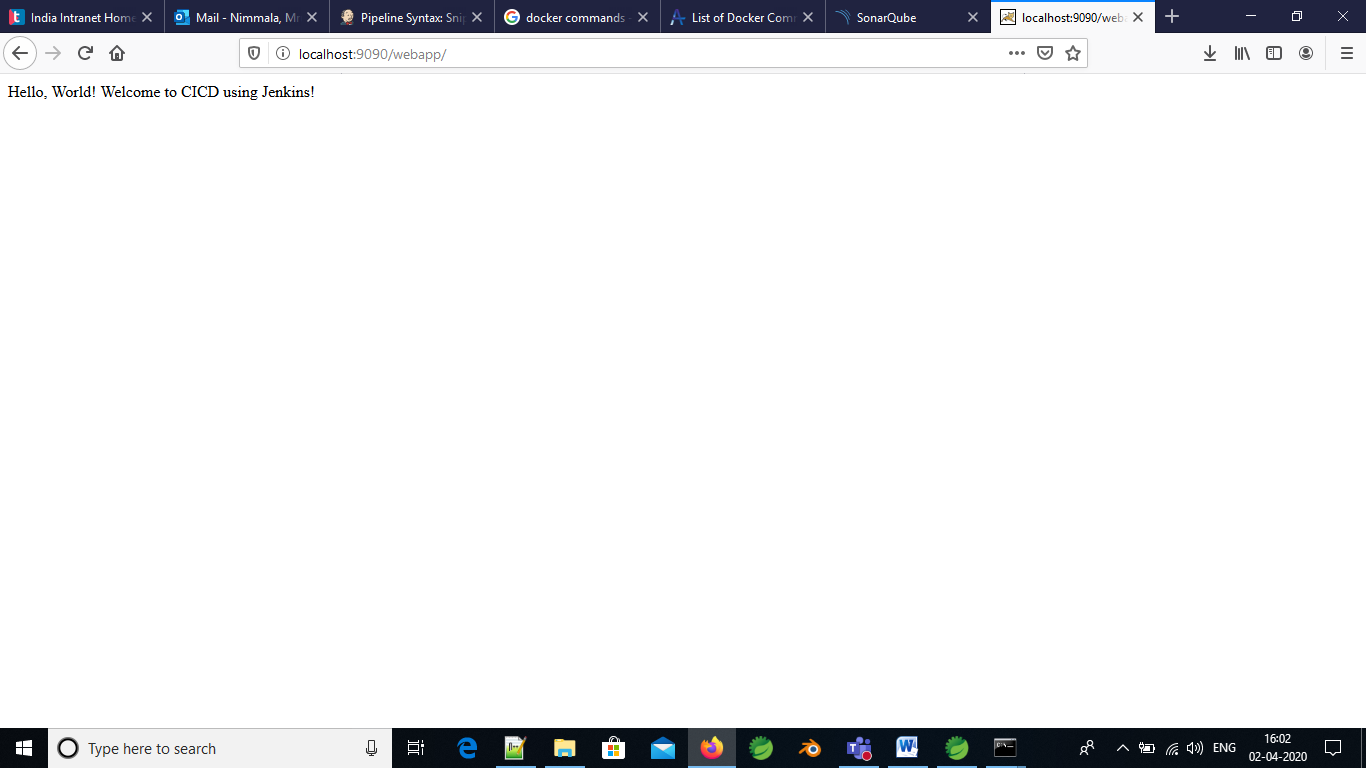
1. Generate the pipeline syntax script in the job



1. Add it in the Jenkinsfile in github link. The Script is **deploy adapters: [tomcat8(credentialsId: '7de1de9d-8e53-441a-8a2d-c06b708e9dcd', path: '', url: 'http://localhost:9090/webapp/')], contextPath: 'webapp', onFailure: false, war: '\*\*/\*.war'**



1. Build the job. See the result.



Date: 20-April-2020

AIM: Selenium Testing in Maven project

Selenium is a testing automation tool that tests the front end of the application. This is mostly for automating the integration testing.

1. To integrate selenium with maven we have to download the zip from <http://www.seleniumhq.org>
2. This zip file contains the all the jar files essential for the selenium sever
3. Add the below dependencies for selenium in pom.xml as shown below:

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.5.3</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.8</version>

<scope>test</scope>

</dependency>

1. Build the selenium configuration in maven using configure-buildpath as shown below



1. Create a class that tests the project web-site as shown below in the java test folder of maven.

**public** **class** DemoClassForSeleniumTest {

**public** **static** **void** main(String[] args) {

System.*setProperty*("webdriver.chrome.driver",

"C:\\Users\\LENOVO\\Downloads\\chromedriver\_win32(2)\\chromedriver.exe");

WebDriver driver = **new** ChromeDriver();

driver.get("http://amazon.in");

driver.manage().window().maximize();

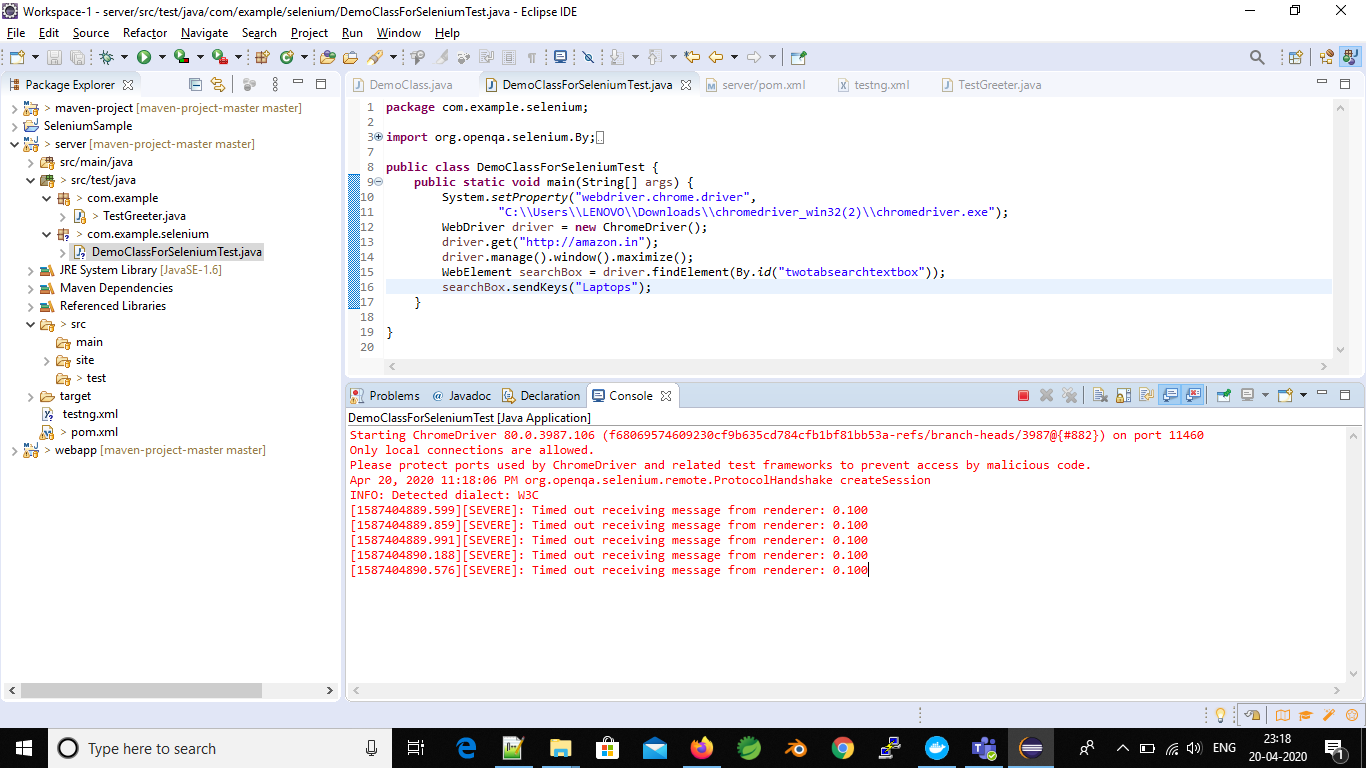
WebElement searchBox = driver.findElement(By.*id*("twotabsearchtextbox"));

searchBox.sendKeys("Laptops");

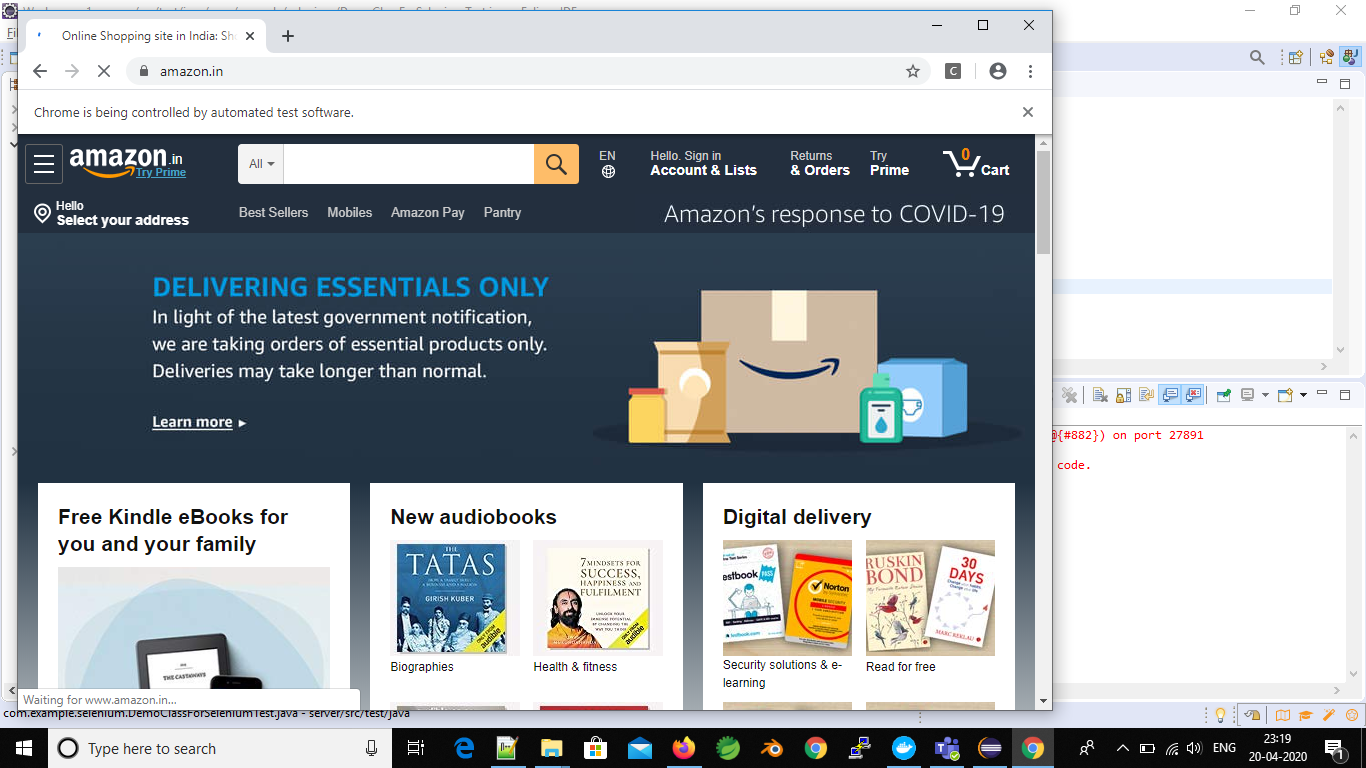
}

}

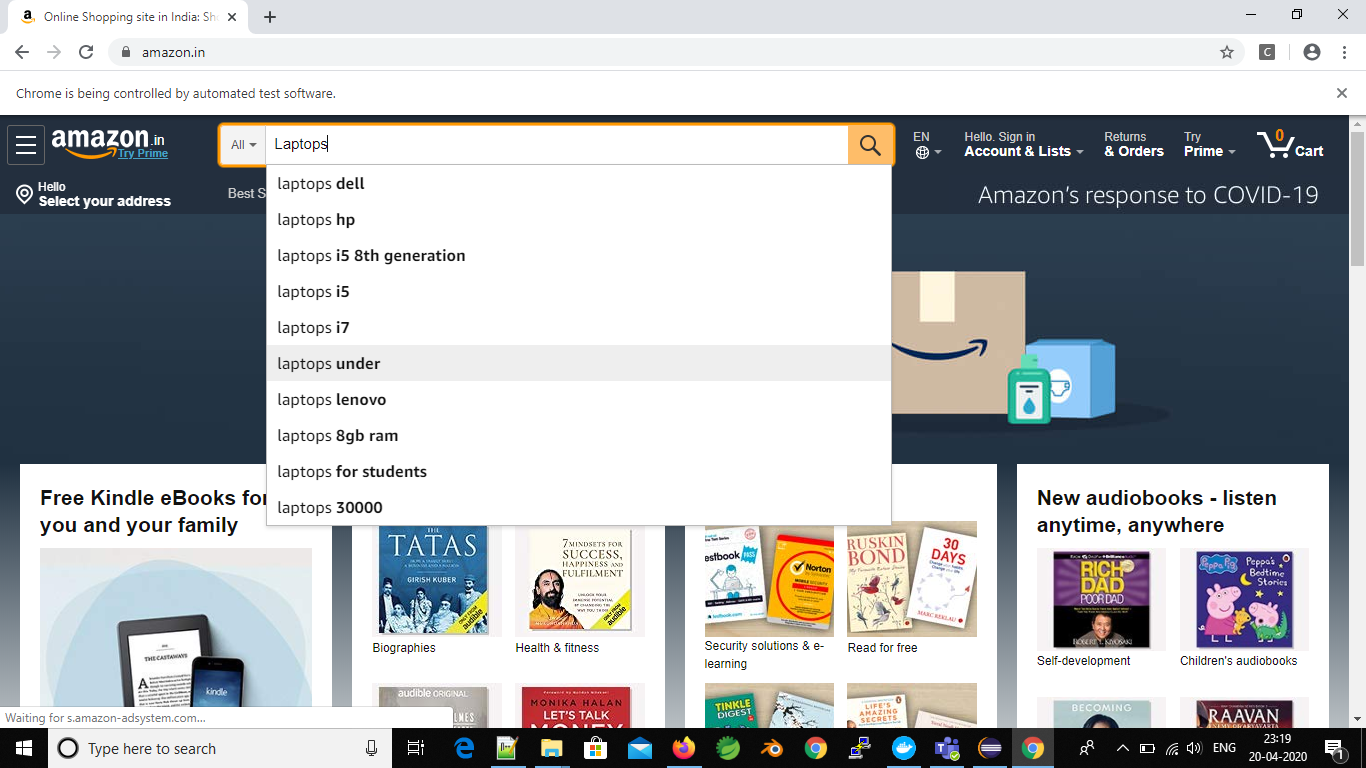
1. In the above code we have set the browser properties. So we have to download the third- party browser for testing the web-site. We have tested the amazon.in website and the search bar of the web-site.
2. Run the above class to test the browser. And the result is as shown below.



1. The tested web-site is.



1. Search box that need to be tested is:



Kubernetes Assessment:

Learning Objectives:-

Create a yaml file containing the pod spec for the nginx pod.

Create the pod.

Your company is getting ready to launch a new website, and they need you to set up an nginx web server in their Kubernetes cluster.

The nginx server will need to be accessible via network in the future, so you will need to expose port 80 as a containerPort for the nginx container.

Your team has also asked you to ensure that nginx runs in quiet mode for the time being to cut down on unnecessary log output.

You can do this by setting the command to nginx and passing the following arg to the container:

-g daemon off; -q. As this nginx server belongs to the Web team, you will need to create it in the team's web namespace.

To summarize:

Use the nginx container image.

The container needs a containerPort of 80.

Set the command to nginx

Pass in the -g daemon off; -q args to run nginx in quiet mode.

Create the pod in the web namespace.

Once the pod is created, you should be able to find it with kubectl get pods -n web.

Once the pod is created, you can get more information about its current status with kubectl describe pod nginx -n web.

apiVersion: v1  
kind: Pod  
metadata:  
 name: nginx-pod  
 labels:  
 app: nginx  
spec:  
 template:  
 *#pod template starts* spec:  
 containers:  
 - name: nginx  
 image: nginx:1.15.4  
 ports:  
 - containerPort: 80  
 args : []  
 restartPolicy: OnFailure

Solution 2:

Learning Objectives:-

Create a ConfigMap called `candy-service-config` to store the container's configuration data.

Create a Kubernetes secret called `db-password` to store the database password.

Create the pod for the candy-service application according to the provided specification.

Problem Statement:-

Your company is nearing completion of their new mobile app, a candy-themed game. This application has some backend infrastructure the company plans to run using Kubernetes.

They want to begin deploying one of their backend services to the cluster and have asked you to create a pod definition that meets the specifications required by the software.

Create a pod definition in /home/cloud\_user/candy-service-pod.yml, and then create a pod in the cluster using this definition to make sure it works.

The specifications are as follows:

The current image for the container is linuxacademycontent/candy-service:1. You do not need a custom command or args.

There is some configuration data the container will need:

candy.peppermint.power=100000000

candy.nougat-armor.strength=10

It will expect to find this data in a file at /etc/candy-service/candy.cfg.

Store the configuration data in a ConfigMap called candy-service-config and provide it to the container as a mounted volume.

The container will need to run with the file system group with the id 2000. You will need to set this using the securityContext.

The container should expect to use 64MiB of memory and 250m CPU (use resource requests).

The container should be limited to 128MiB of memory and 500m CPU (use resource limits).

The container needs access to a database password in order to authenticate with a backend database server. The password is Kub3rn3t3sRul3s!.

It should be stored in a secure fashion (as a Kubernetes secret called db-password) and passed to the container as an environment variable called DB\_PASSWORD.

Good luck!

Creating ConfigMap: Kubectl create configmap candy-service-config --from-file=/etc/candy-service/

Creating the secret: kubectl create secret generic db-password --from-file=./username.txt --from-file=./password.txt

Config-service-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: candy-service-pod

spec:

securityContext:

runAsUser: 1000

runAsGroup: 3000

fsGroup: **2000**

containers:

- name: candy-service

image: **linuxacademycontent/candy-service:1**

volumeMounts:

- name: **candy-service-config**

mountPath: "/etc/candy-service/"

readOnly: true

volumes:

# You set volumes at the Pod level, then mount them into containers inside that Pod

- name: candy-service-config

configMap:

# Provide the name of the ConfigMap you want to mount.

name: candy-service-config