DevOps Certification

Training Certification Project –01

Insure Me - Insurance Domain

The company's goal is to deliver the product updates frequently to production with High quality & Reliability. They also want to accelerate software delivery speed, quality and reduce feedback time between developers and testers.

Following are the problems the company is facing at the moment

✓ Building Complex builds is difficult
✓ Manual efforts to test various components/modules of the project
✓ Incremental builds are difficult to manage, test and deploy
✓ Creation of infrastructure and configure it manually is very time consuming
✓ Continuous manual monitoring of the application is quite challenging.

In order to implement a POC, you are requested to develop a Mavenaised microservice using spring boot and in memory h2 database.

1. A microservice which exposes below mentioned endpoints as APIs and uses in memory h2 database to store the data.

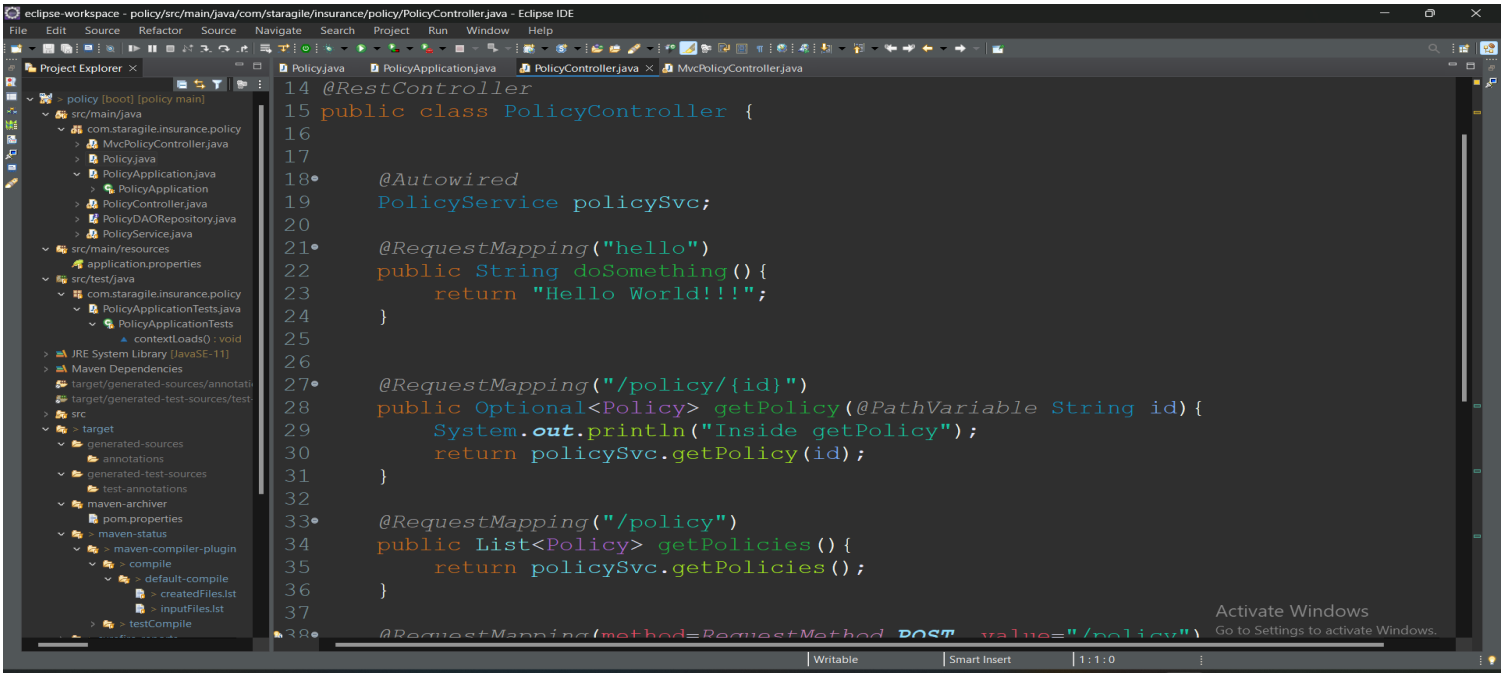
a. /createPolicy (HTTP Method : POST) (Request Body : JSON)
b. /updatePolicy/{policy id} (HTTP Method : PUT) (Request Body : JSON)
c. /viewPolicy/{policy id} (HTTP Method : GET) (No Request Body)
d. /deletePolicy/{policy id} (HTTP Method : DELETE) (No Request Body)

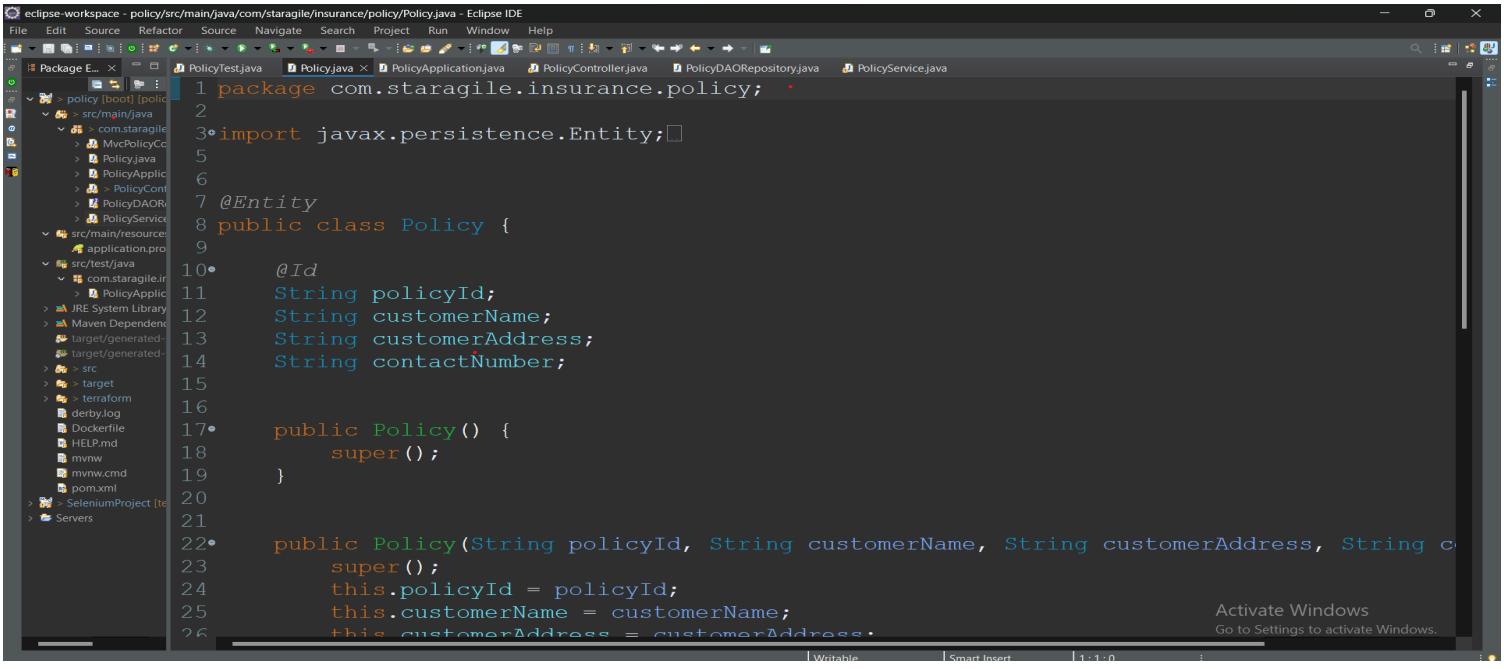
- Here I wrote the code with Junit test cases & added maven dependencies for the application using eclipse IDE and pushed it to the github repository.

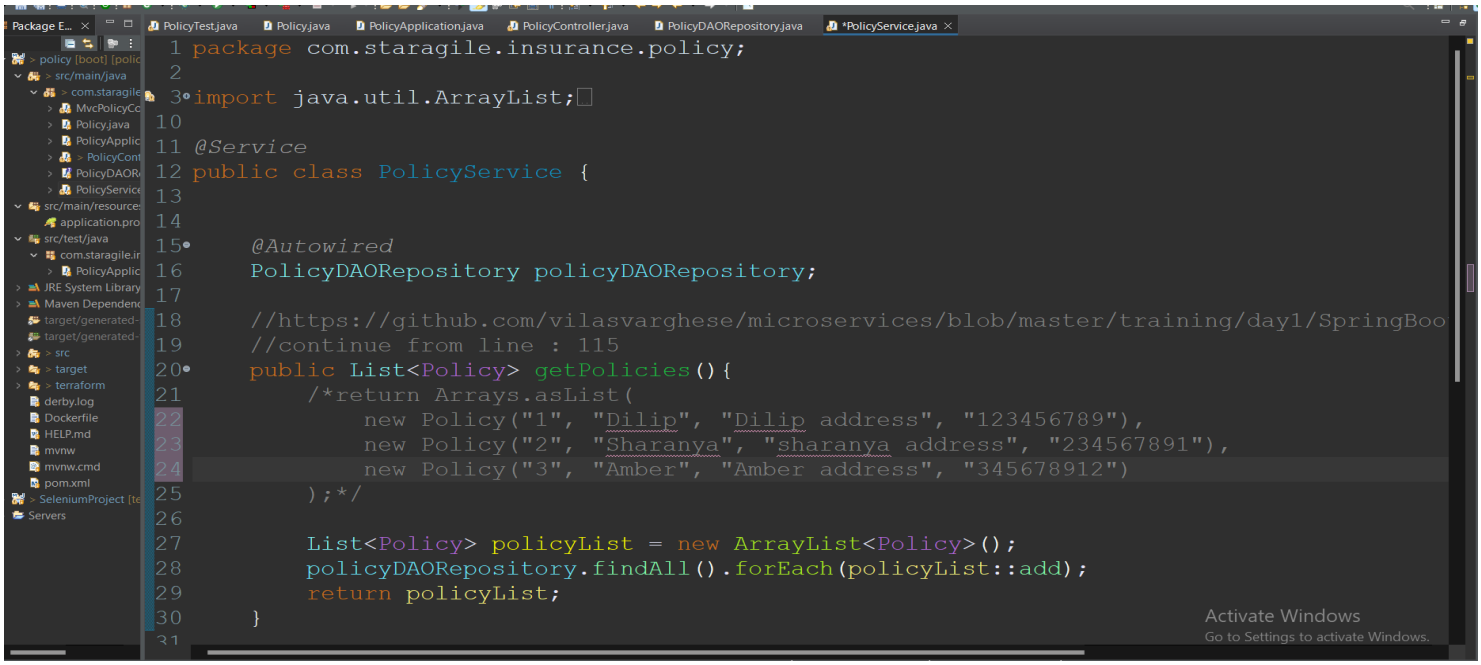
- https://github.com/Dilipkumar-M/Insureme

1. A microservice which exposes endpoints as APIs and uses an in memory h2 database to store the data.

a. /createPolicy (HTTP Method : POST) (Request Body : JSON)
b. /updatePolicy/{policy id} (HTTP Method : PUT) (Request Body : JSON)
c. /viewPolicy/{policy id} (HTTP Method : GET) (No Request Body)
d. /deletePolicy/{policy id} (HTTP Method : DELETE) (No Request Body





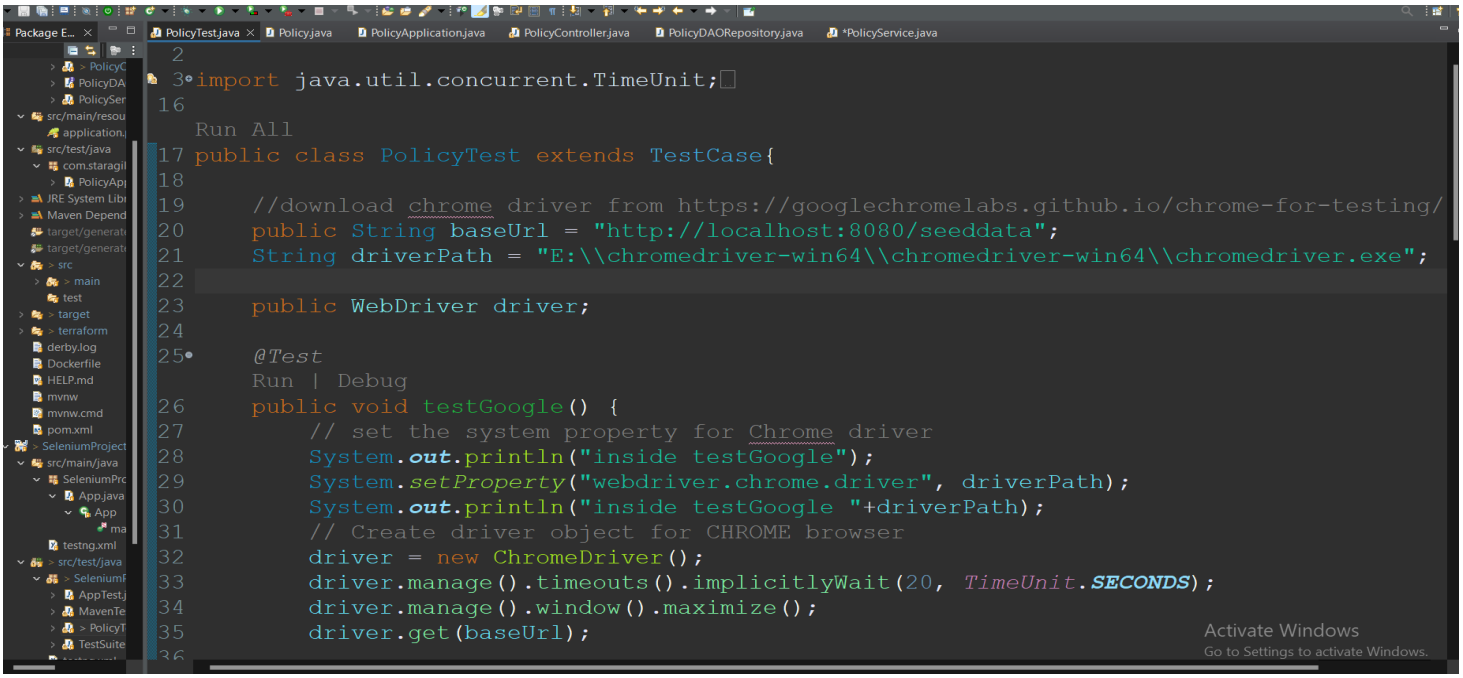


```java
package com.staragile.insurance.policy;

import java.util.ArrayList;

@Service
public class PolicyService {


    @Autowired
    PolicyDAORepository policyDAORepository;

    //https://github.com/vilasvarghese/microservices/blob/master/training/day1/SpringBoo
    //continue from line : 115
    public List<Policy> getPolicies(){
        /*return Arrays.asList(
            new Policy("1", "Dilip", "Dilip address", "123456789"),
            new Policy("2", "Sharanya", "sharanya address", "234567891"),
            new Policy("3", "Amber", "Amber address", "345678912")
        );*/

        List<Policy> policyList = new ArrayList<Policy>();
        policyDAORepository.findAll().forEach(policyList::add);
        return policyList;
    }
```

Here I wrote the code for following endpoints mentioned in the question 1).

2. Write necessary Junit test cases.

In eclipse choose file→java project→new project→policytest→new policytest.java



```java
import java.util.concurrent.TimeUnit;

Run All
public class PolicyTest extends TestCase{

    //download chrome driver from https://googlechromelabs.github.io/chrome-for-testing/
    public String baseUrl = "http://localhost:8080/seeddata";
    String driverPath = "E:\\chromedriver-win64\\chromedriver-win64\\chromedriver.exe";

    public WebDriver driver;

    @Test
    Run | Debug
    public void testGoogle() {
        // set the system property for Chrome driver
        System.out.println("inside testGoogle");
        System.setProperty("webdriver.chrome.driver", driverPath);
        System.out.println("inside testGoogle "+driverPath);
        // Create driver object for CHROME browser
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        driver.get(baseUrl);
```

3. Generate HTML reports using TestNG.

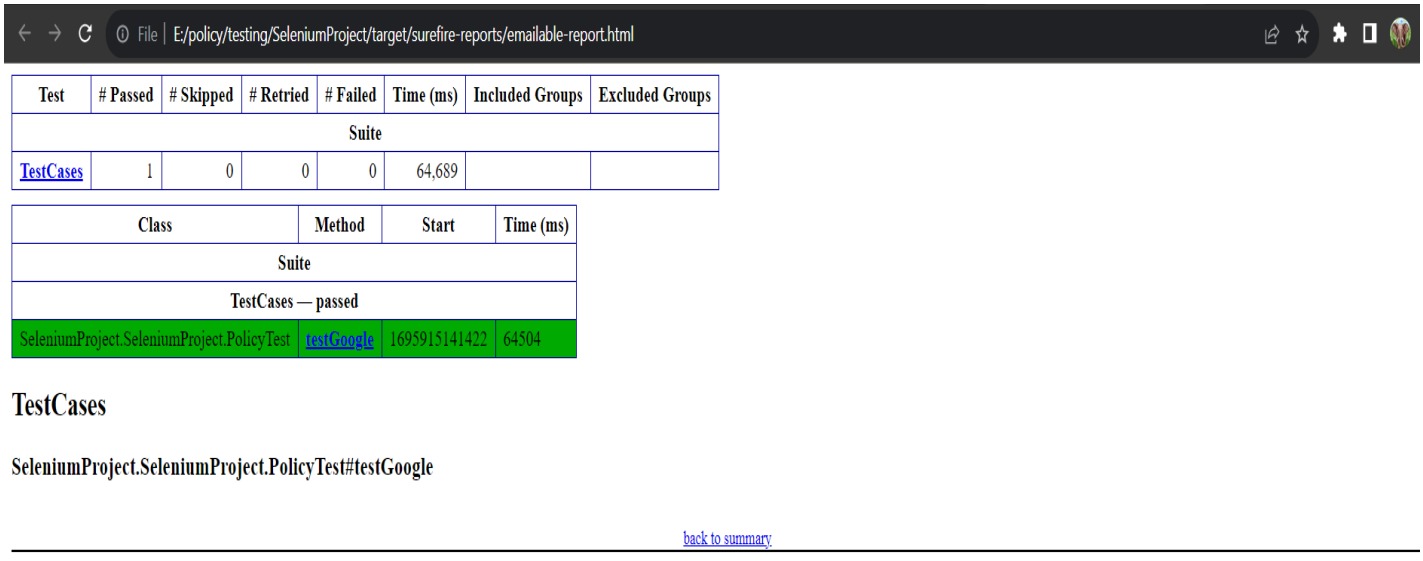
After file→java project→new project→policytest→new policytest.java run the application as TestNG application before that make sure you're installing the chromedriver.exe in the local machine and set the path in policytest.java.Download it from given link below:

https://googlechromelabs.github.io/chrome-for-testing/

/ersion: 118.0.5993.54 (r1192594)

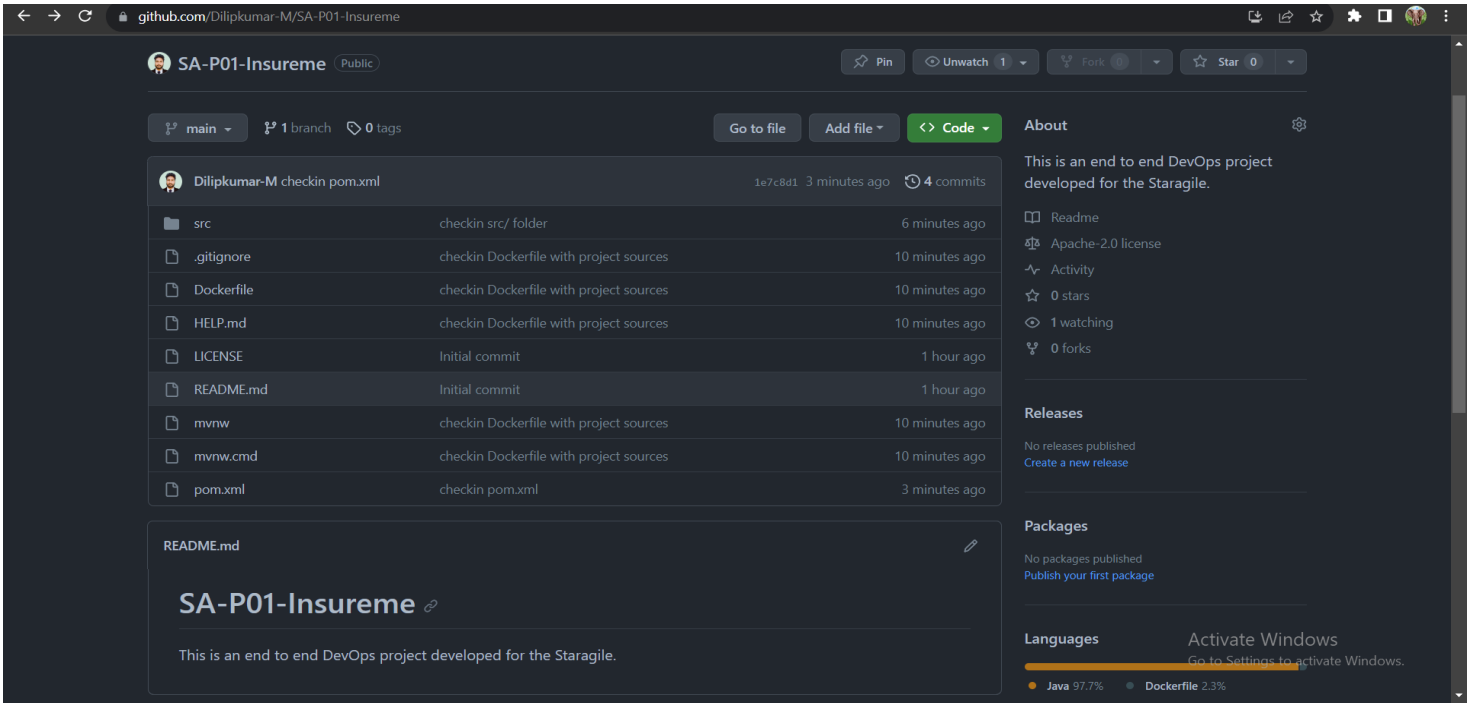
Binary	Platform	URL	HTTP status
chrome	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/linux64/chrome-linux64.zip	200
chrome	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/mac-arm64/chrome-mac-arm64.zip	200
chrome	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/mac-x64/chrome-mac-x64.zip	200
chrome	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/win32/chrome-win32.zip	200
chrome	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/win64/chrome-win64.zip	200
chromedriver	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/linux64/chromedriver-linux64.zip	200
chromedriver	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/mac-arm64/chromedriver-mac-arm64.zip	200
chromedriver	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/mac-x64/chromedriver-mac-x64.zip	200
chromedriver	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/win32/chromedriver-win32.zip	200
chromedriver	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/win64/chromedriver-win64.zip	200
chrome-headless-shell	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/linux64/chrome-headless-shell-linux64.zip	200
chrome-headless-shell	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/mac-arm64/chrome-headless-shell-mac-arm64.zip	200
chrome-headless-shell	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/mac-x64/chrome-headless-shell-mac-x64.zip	200
chrome-headless-shell	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/win32/chrome-headless-shell-win32.zip	200
chrome-headless-shell	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/118.0.5993.54/win64/chrome-headless-shell-win64.zip	200

Get the html reports after running the application



4. Push your code into your GitHub Repository

```
Dilip@Dilipkumar MINGW64 /e/SA-P01-Insureme (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.04 KiB | 1.04 MiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Dilipkumar-M/SA-P01-Insureme.git
   1b77598..1e7c8d1  main -> main

Dilip@Dilipkumar MINGW64 /e/SA-P01-Insureme (main)
$
```



Here I pushed the application source code into my Github repository repo link is mentioned below:

- ❖ https://github.com/Dilipkumar-M/Insureme

2. Implementing CI/CD in the aws instance.

Here I implemented Continuous Integration & Continuous Deployment of the application using DevOps tools like,Jenkins,Git,Docker,aws,Maven,Java.all these tools are configured with terraform and launched the Jenkins-instance from the local machine through aws configure using the accesskey and the secret key from the IAM security policies,and configured jenkins permissions to access the docker and the push image to dockerhub.

Files which are needed to start terraform:-

- **Compute.tf**

```
resource "aws_instance" "jenkins-instance" {
  ami            = var.instance_ami   #ubuntu ami for N.virginia region.
  instance_type   = var.instance_type
  key_name       = var.keyname
  #user_data = file("install_jenkins.sh")
  associate_public_ip_address = true
  tags = {
    Name = "Jenkins-Instance"
```

- **main.tf**

```
provider "aws" {
  region                  = "${var.region}"
  shared_credentials_files = ["~/.aws/credentials"]
}
```

- **terraform.tfvars**

```
region = "us-east-1"
instance_type = "t2.medium"
instance_ami = "ami-032df771421fcffbd"#"ami-053b0d53c279acc90"
keyname = "awsLinux"
```

- **variables.tf**

```
variable "region" {
```

```
  default = "us-east-1"
}
variable "instance_type" {

}
variable "instance_ami" {

}
variable "keyname" {
  default = "awsLinux"
}
```

Follow the commands to start the aws instance from the local machine:

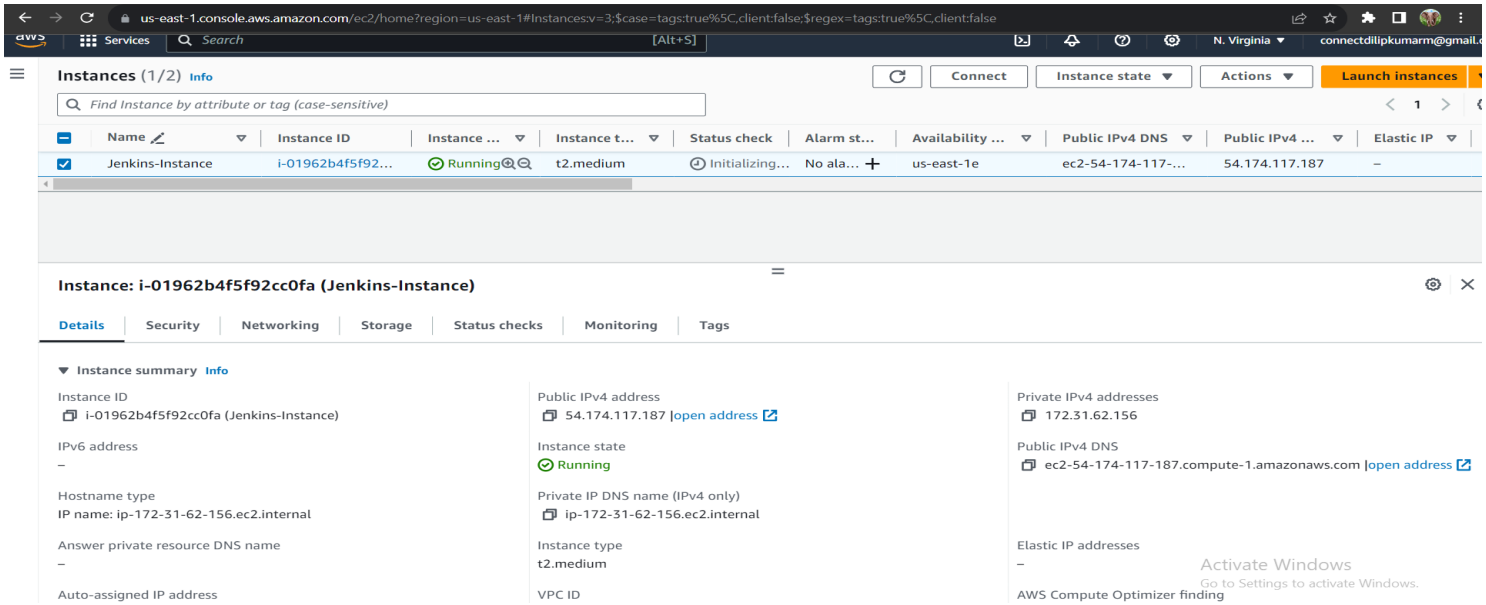
In, terraform folder→terraform init→terraform plan→terraform apply→terraform destroy.

```
Dilip@Dilipkumar MINGW64 /e/policy/terraform (main)
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.19.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

Dilip@Dilipkumar MINGW64 /e/policy/terraform (main)
```

```
Dilip@Dilipkumar MINGW64 /e/policy/terraform (main)
$ terraform plan

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.jenkins-instance will be created
  + resource "aws_instance" "jenkins-instance" {
      + ami                                  = "ami-053b0d53c279acc90"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = true
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```

```
MINGW64:/e/policy/terraform

Dilip@Dilipkumar MINGW64 /e/policy/terraform (main)
$ terraform apply

Terraform used the selected providers to generate the following e
xecution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.jenkins-instance will be created
  + resource "aws_instance" "jenkins-instance" {
      + ami                                  = "ami-053b0d53c279a
cc90"
      + arn                                  = (known after apply
)
      + associate_public_ip_address          = true
      + availability_zone                    = (known after apply
)
      + cpu_core_count                       = (known after apply
)
      + cpu_threads_per_core                 = (known after apply
)
      + disable_api_stop                     = (known after apply
)
      + disable_api_termination              = (known after apply
```

```
      + ebs_block_device {
          + delete_on_termination = true
          + device_name           = "/dev/sda1"
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + snapshot_id           = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = 20
          + volume_type           = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.jenkins-instance: Creating...
aws_instance.jenkins-instance: Still creating... [10s elapsed]
aws_instance.jenkins-instance: Still creating... [20s elapsed]
aws_instance.jenkins-instance: Still creating... [30s elapsed]
aws_instance.jenkins-instance: Creation complete after 38s [id=i-
008a79e9a117f867a]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```



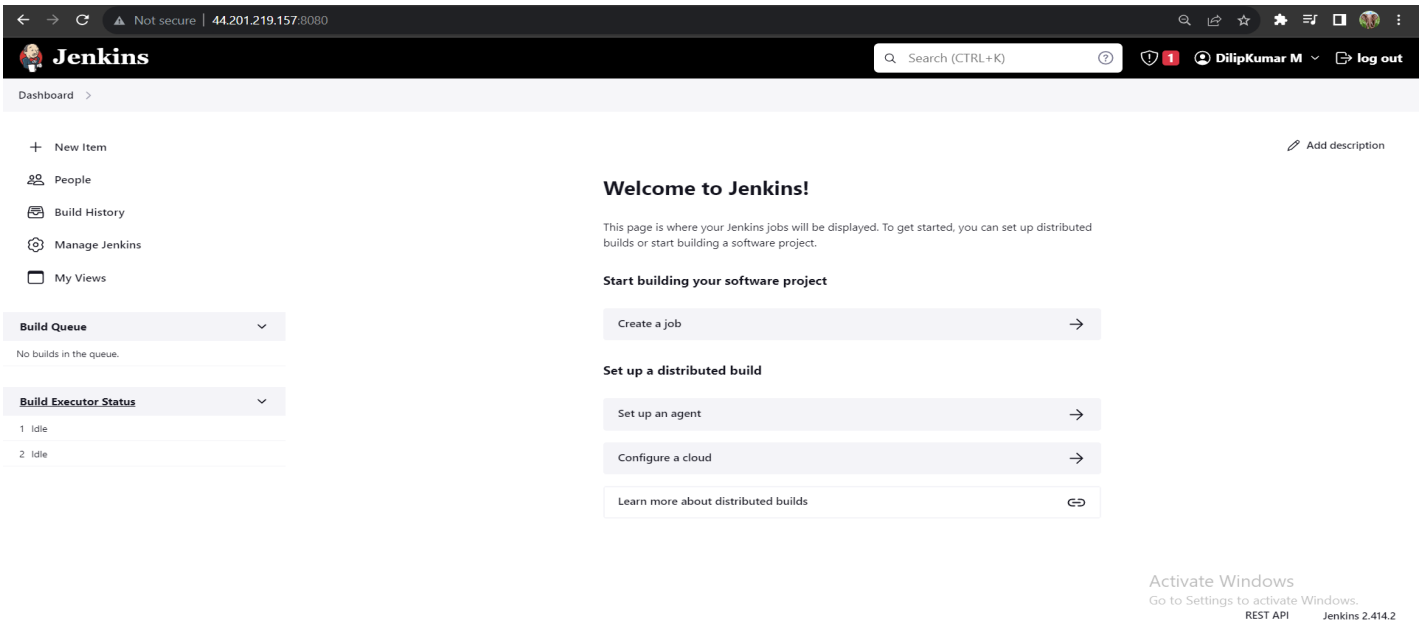
Above image shows how terraform created the jenkins instance which includes all the tools mentioned.
After initialization of the Jenkins-instance with terraform from the local machine, connect it to run the job of CICD with jenkins and docker to produce images from the container.and push it to the docker hub.

Here are the commands given in the form of shell script to the terraform folder to execute within the aws Virtual machine to install all the required Devops tools for the initial process.

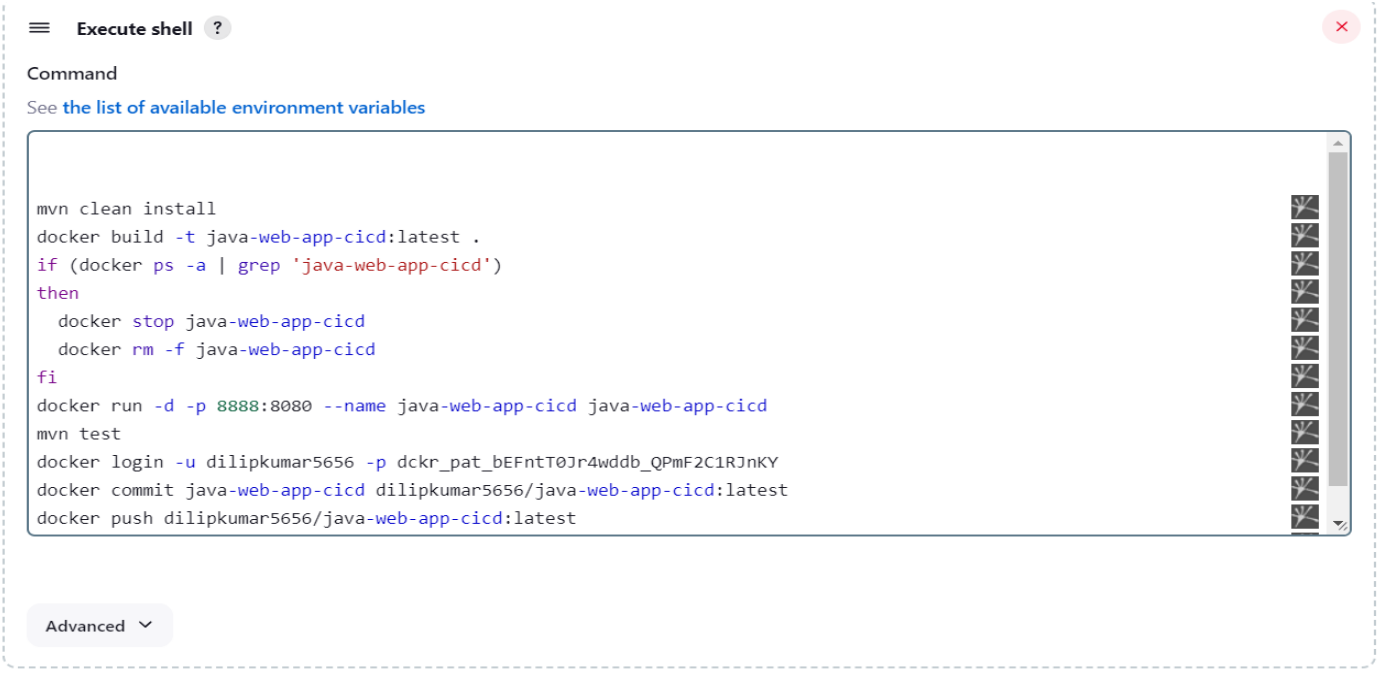
```
#!/bin/bash
sudo apt update -y && apt upgrade -y

echo "Install Java JDK 8"
sudo apt remove -y java
sudo apt install default-jdk -y

echo "Install Maven"
sudo apt install maven -y

echo "Install git"
sudo apt install -y git

echo "Install Docker engine"
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

#sudo usermod -a -G docker jenkins
#sudo service docker start
#sudo chkconfig docker on

echo "Install Jenkins"
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian-stable binary/ | sudo tee  /etc/apt/sources.list.d/jenkins.list >
/dev/null
sudo apt-get update -y
sudo apt-get install fontconfig -y
```

sudo apt-get install jenkins -y

After running this script the tools were automatically installed in the respective virtual machine that have we chosen



After accessing the jenkins server choose the Freestyle job and provide the github project configure git access to the jenkins to run continuous integration,and change give permissions to the jenkins to configure docker user to it and run the below commands,and also add git SCM webhooks to the job.

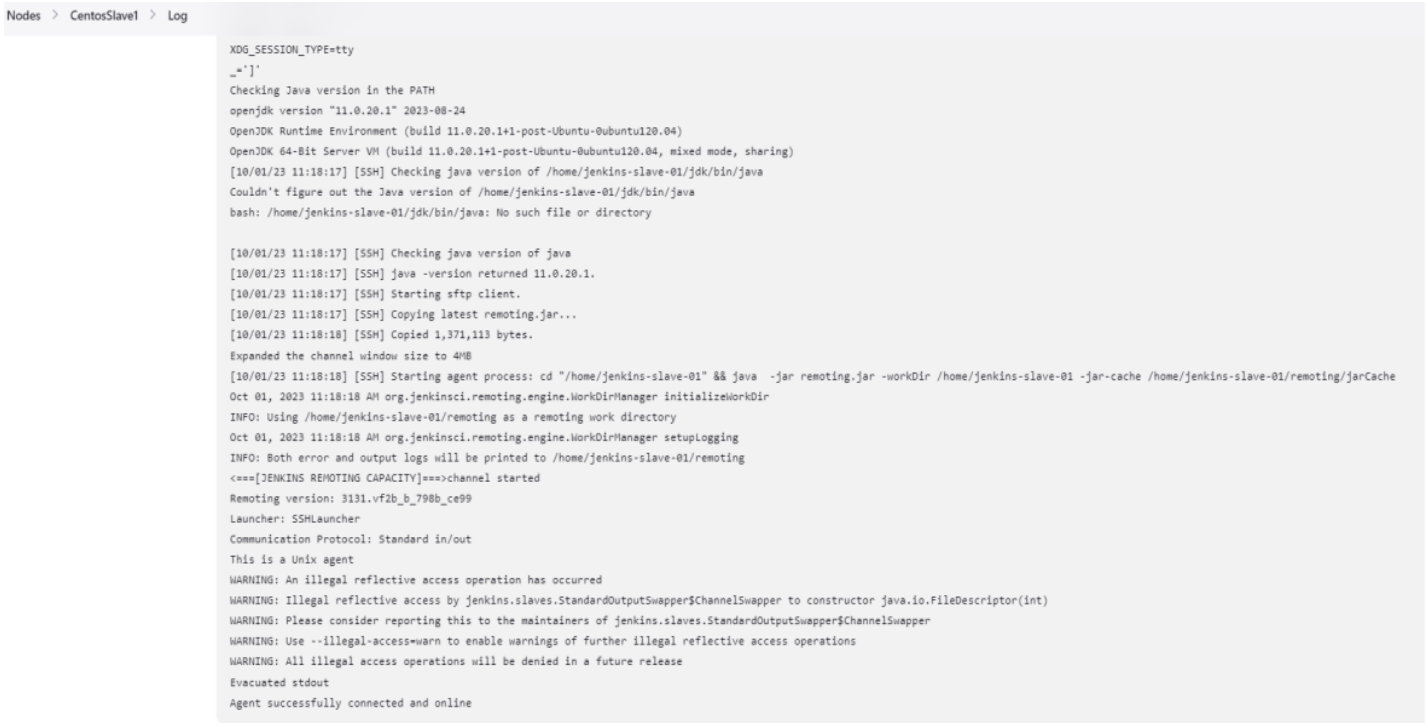


After adding the webhooks to the jenkins job this webhooks alerts the developer with notifications whenever the source code of the project is deployed by users or the developers,then builds the CI/CD.

Commands as script in ShellExecution to do a job:

```
mvn clean install
docker build -t java-web-app-cicd:latest .
if (docker ps -a | grep 'java-web-app-cicd')
then
  docker stop java-web-app-cicd
  docker rm -f java-web-app-cicd
fi
docker run -d -p 8888:8080 --name java-web-app-cicd java-web-app-cicd
mvn test
docker login -u dilipkumar5656 -pdckr_pat_1foZ87ZAbTpjBKZgR9t242fAwlc
docker commit java-web-app-cicd dilipkumar5656/java-web-app-cicd:latest
docker push dilipkumar5656/java-web-app-cicd:latest
```

Add slave node centos



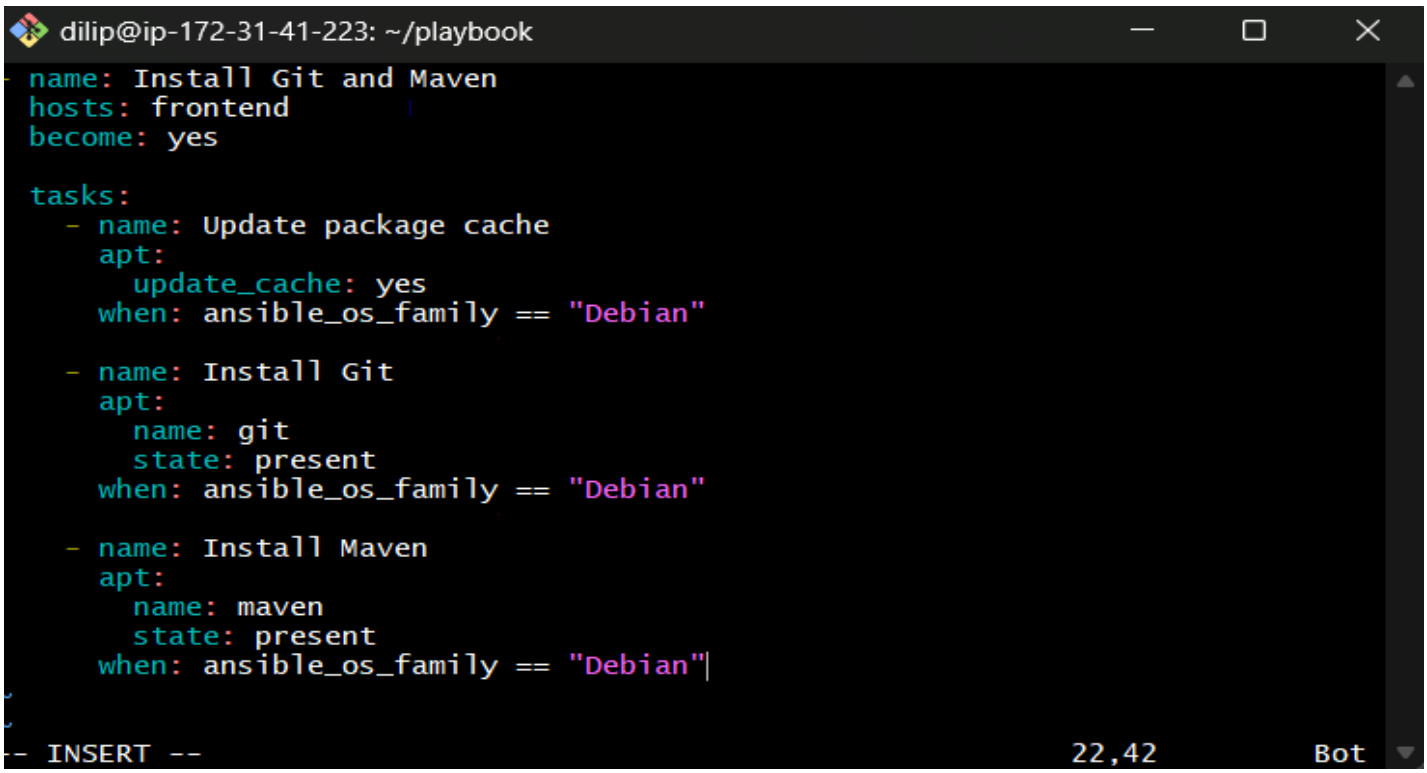
3. Ansible Configuration

Installing ansible and configuring both machines and adding inventory files.


```
dilip@ip-172-31-41-223:/etc/ansible$ ansible all -m ping
[WARNING]: Ansible is being run in a world writable directory (/etc/ansible),
ignoring it as an ansible.cfg source. For more information see
https://docs.ansible.com/ansible/devel/reference_appendices/config.html#cfg-in-
world-writable-dir
172.31.43.56 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Connected successfully

Write a playbook to install some tools in the Frontend server.

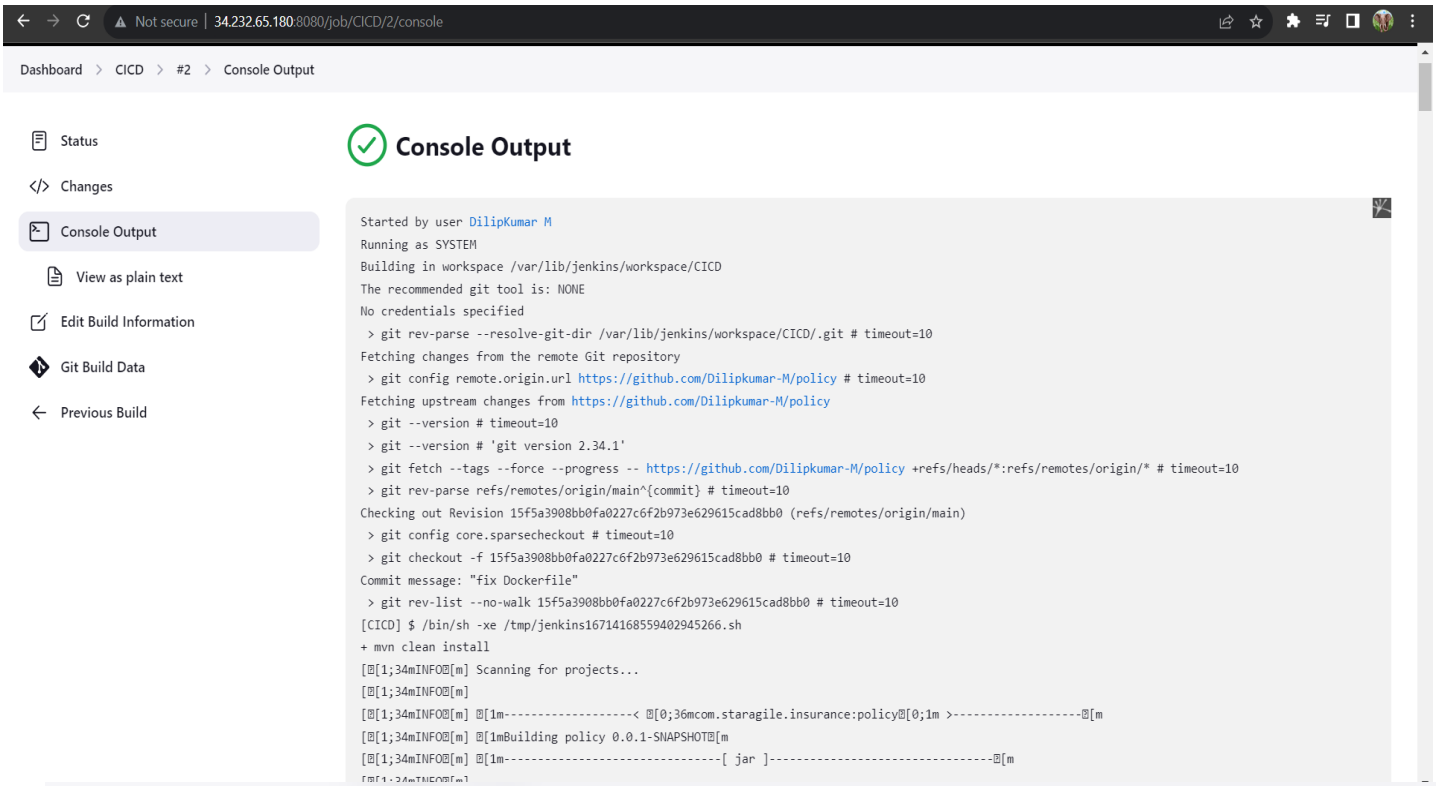


```
dilip@ip-172-31-41-223:~/playbook$ ansible-playbook playbook1.yaml -kK
SSH password:
BECOME password[defaults to SSH password]:

PLAY [Install Git and Maven] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [172.31.43.56]

TASK [Update package cache] ****************************************************
changed: [172.31.43.56]

TASK [Install Git] *************************************************************
ok: [172.31.43.56]

TASK [Install Maven] ***********************************************************
changed: [172.31.43.56]

PLAY RECAP *********************************************************************
172.31.43.56               : ok=4    changed=2    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0
```

Docker installation in frontend server using scripts

```
dilip@ip-172-31-41-223:/home/ubuntu$ sudo snap install docker
docker 20.10.24 from Canonical✓ installed
dilip@ip-172-31-41-223:/home/ubuntu$ docker --version
Docker version 20.10.24, build 297e128
```

```
Docker version 20.10.24, build 297e128
dilip@ip-172-31-41-223:/home/ubuntu$ docker run hello-world
docker: Got permission denied while trying to connect to the Docker daemon soc
: connect: permission denied.
See 'docker run --help'.
dilip@ip-172-31-41-223:/home/ubuntu$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:4f53e2564790c8e7856ec08e384732aa38dc43c52f02952483e3f003afbf23c
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
```

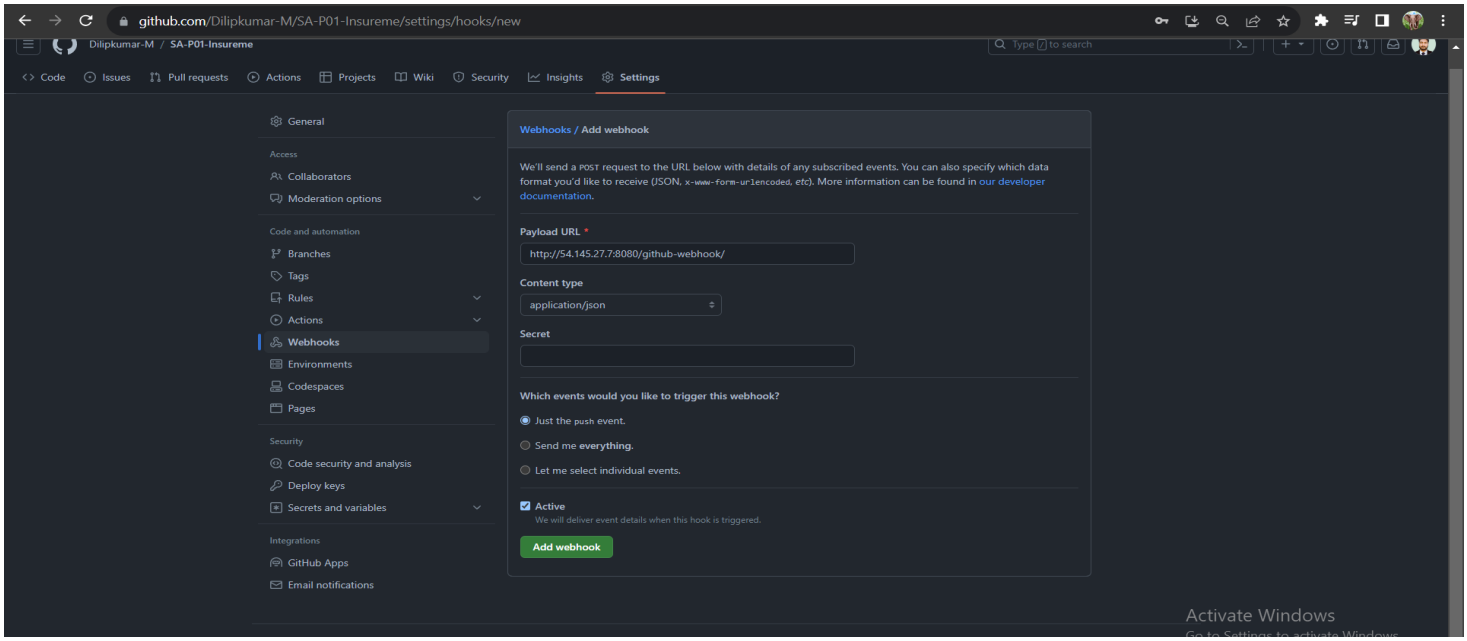


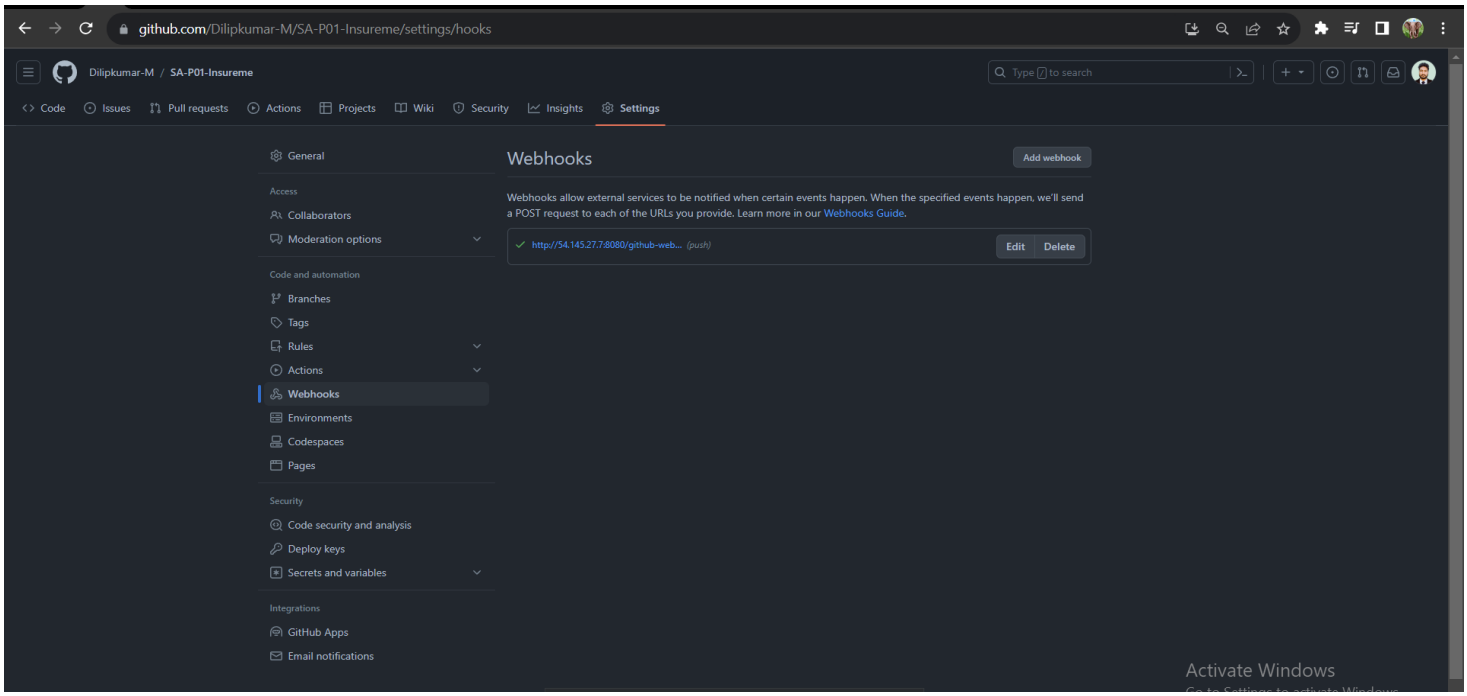

```
4e34ded7dafa: Preparing
805f571012db: Preparing
8d2c3e6dd4a7: Preparing
bc2af3cd1567: Preparing
3736f2313843: Preparing
6be690267e47: Preparing
13a34b6fff78: Preparing
9c1b6dd6c1e6: Preparing
6be690267e47: Waiting
13a34b6fff78: Waiting
9c1b6dd6c1e6: Waiting
bc2af3cd1567: Waiting
3736f2313843: Waiting
8d2c3e6dd4a7: Mounted from library/maven
805f571012db: Mounted from library/maven
bc2af3cd1567: Mounted from library/maven
3736f2313843: Mounted from library/maven
6be690267e47: Mounted from library/maven
13a34b6fff78: Mounted from library/maven
6b532a0850da: Pushed
4e34ded7dafa: Pushed
9c1b6dd6c1e6: Mounted from library/maven
f177b4bbb761: Pushed
latest: digest: sha256:2045b905637059c2641642b9c9adf28ef3cca98ebfe76908ba485e99ca2a8df7 size: 2415
Finished: SUCCESS
```

Give permissions to the jenkins slave to configure with docker to push the images and restart both jenkins and docker.

sudo usermod -aG docker jenkins-centos
sudo systemctl restart docker
sudo systemctl restart jenkins

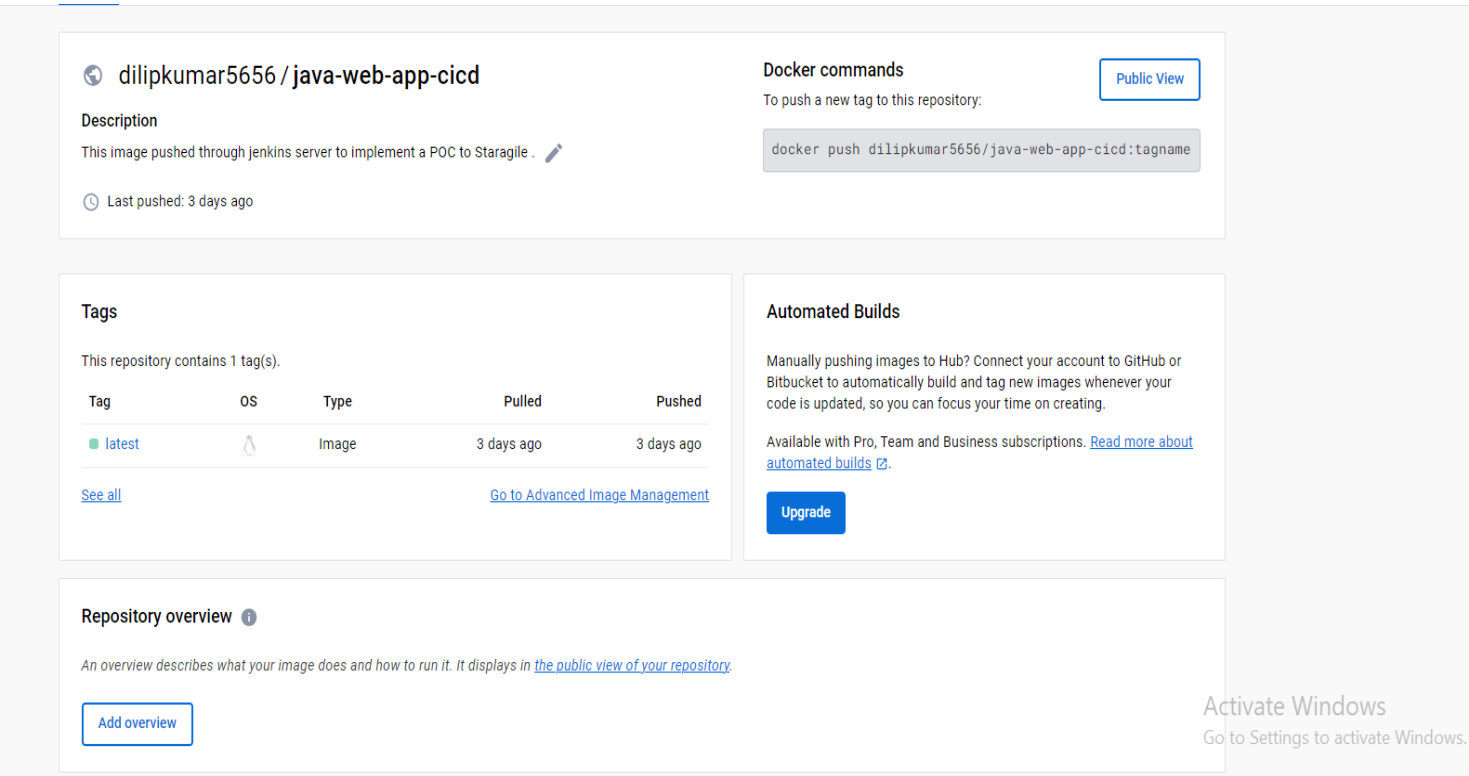
Webhook





After all these process the jenkins pushes automatically the image into the docker hub

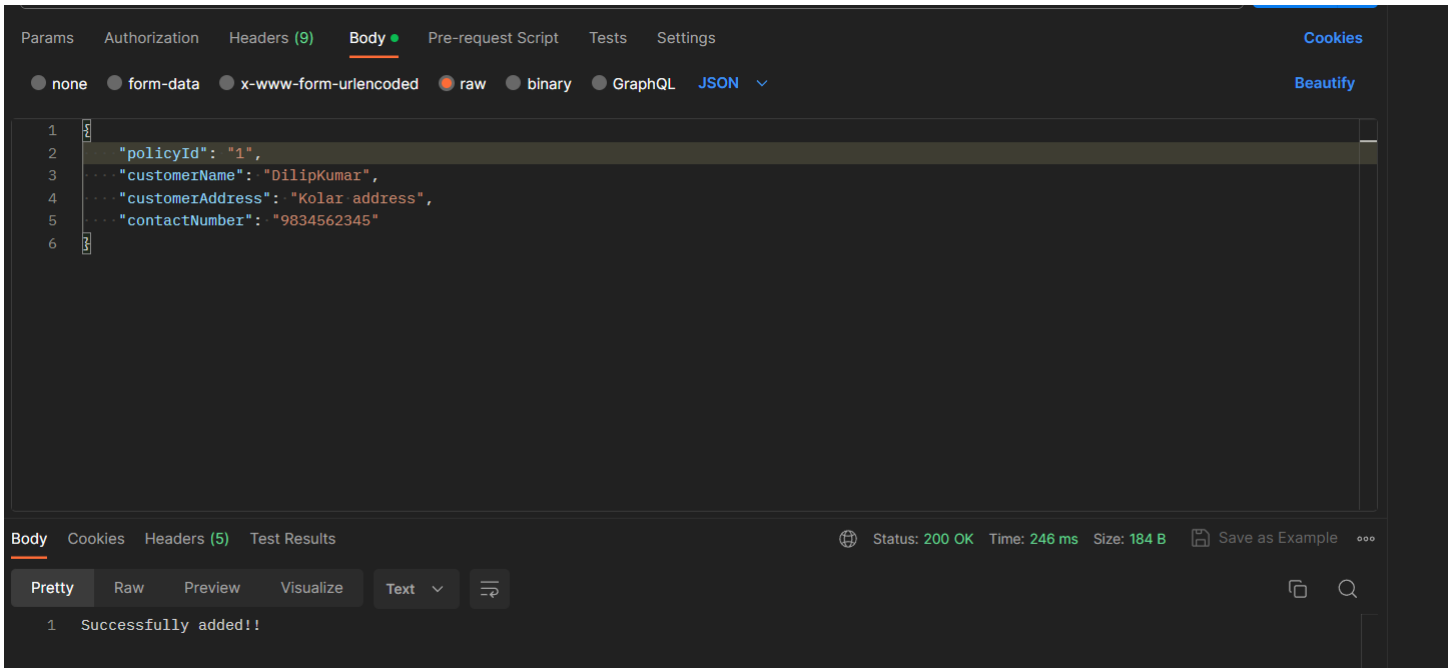
BUILD SUCCESSFUL:



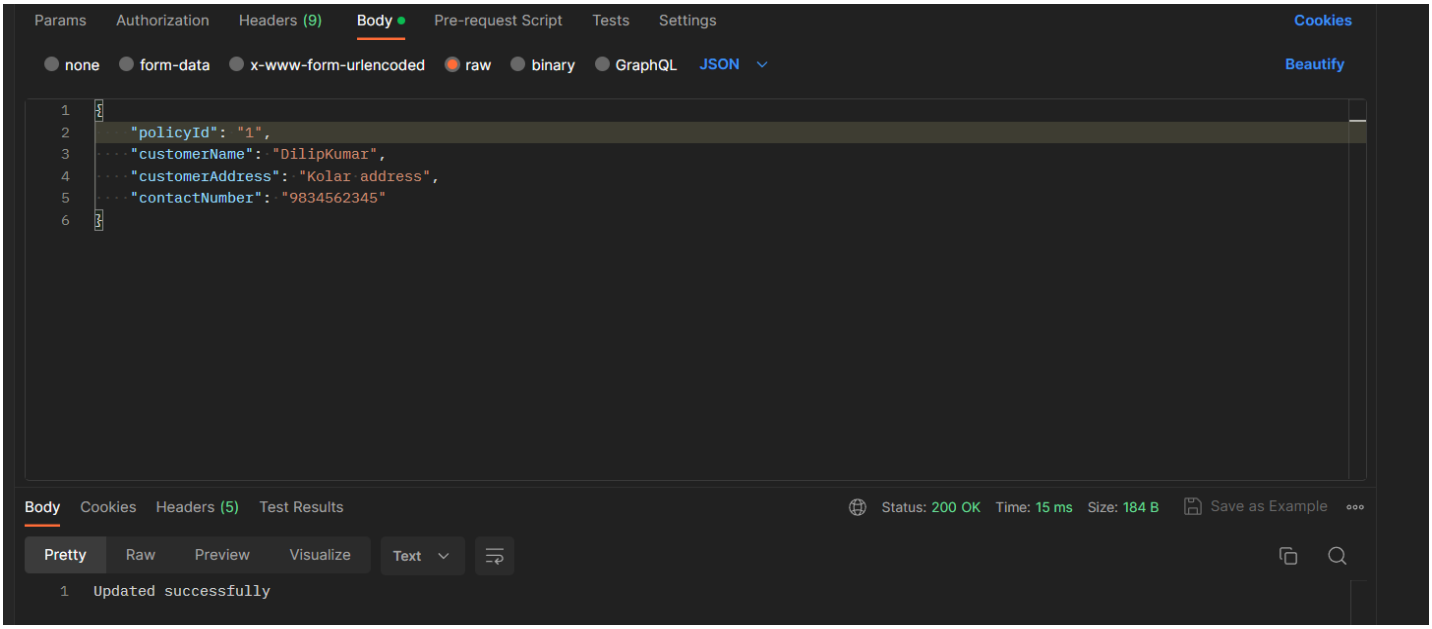
OUTPUT:

http://35.153.140.136:8888/getallpolicy

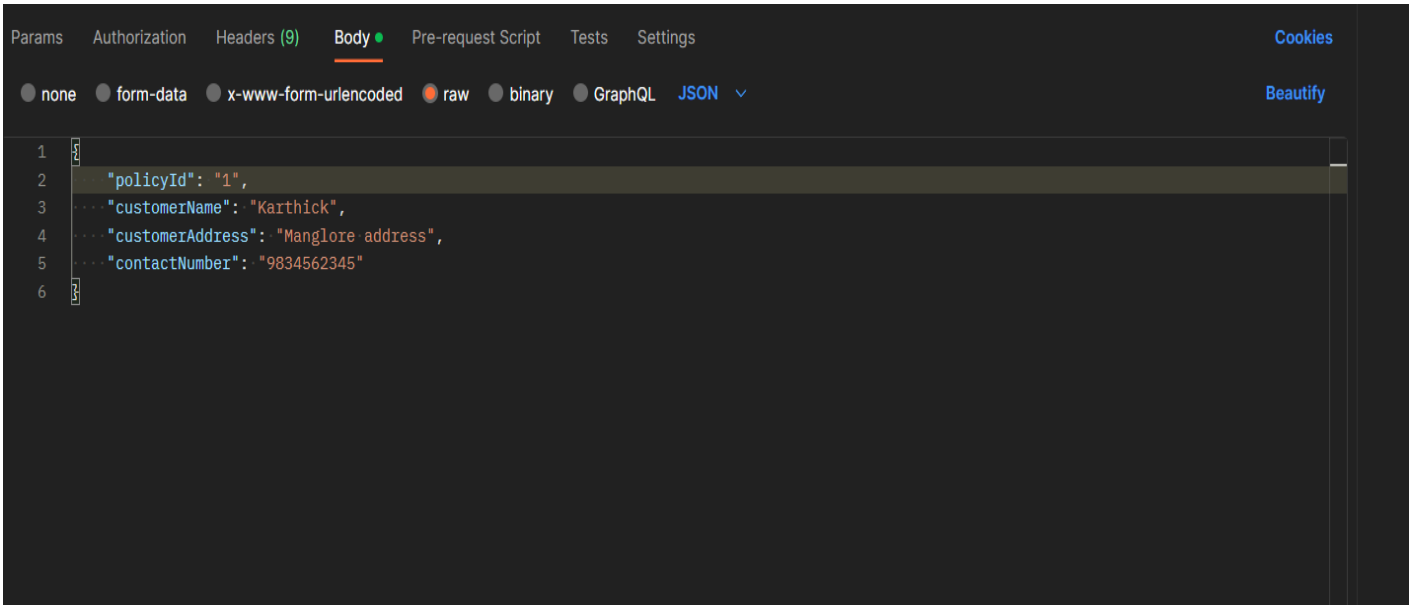
- **To show the database i used the postman tool**
- **Create: - Post endpoint**



- **Update : Put endpoint**

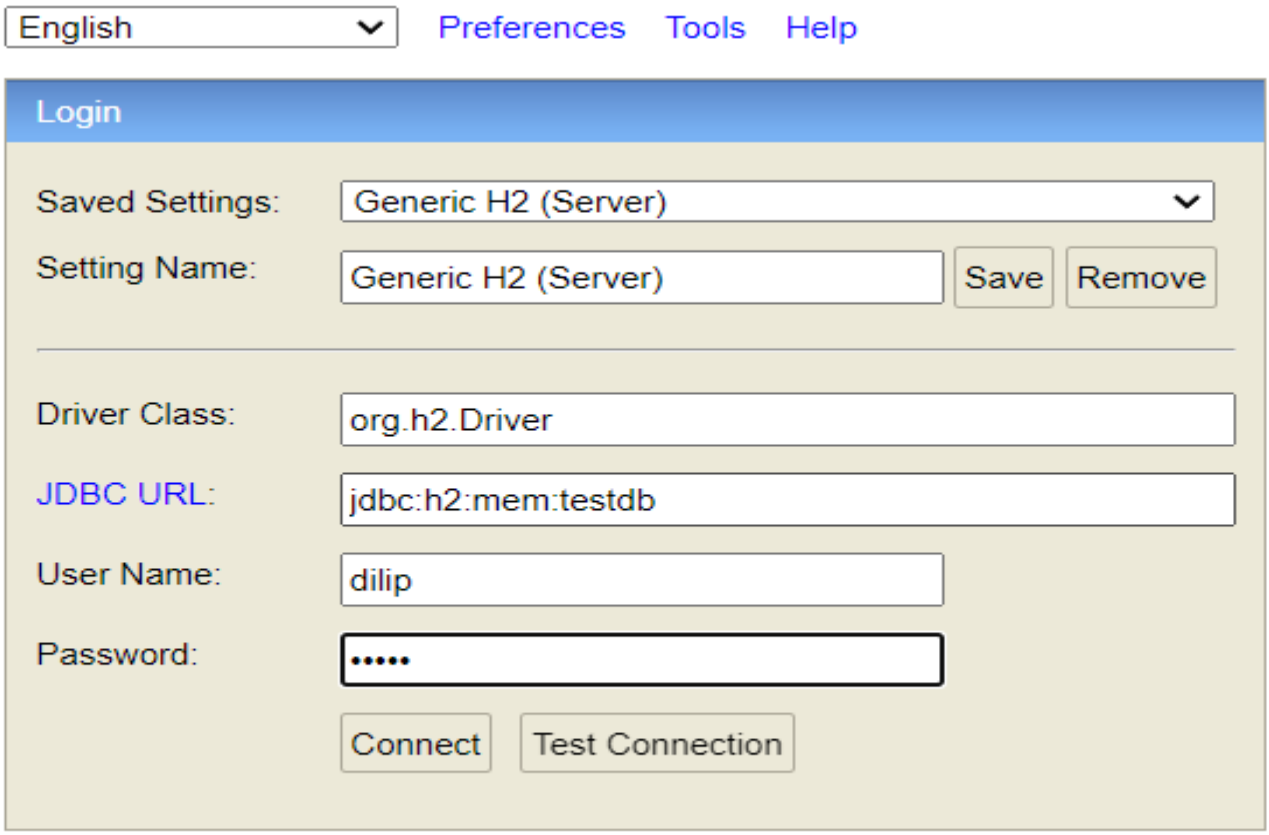


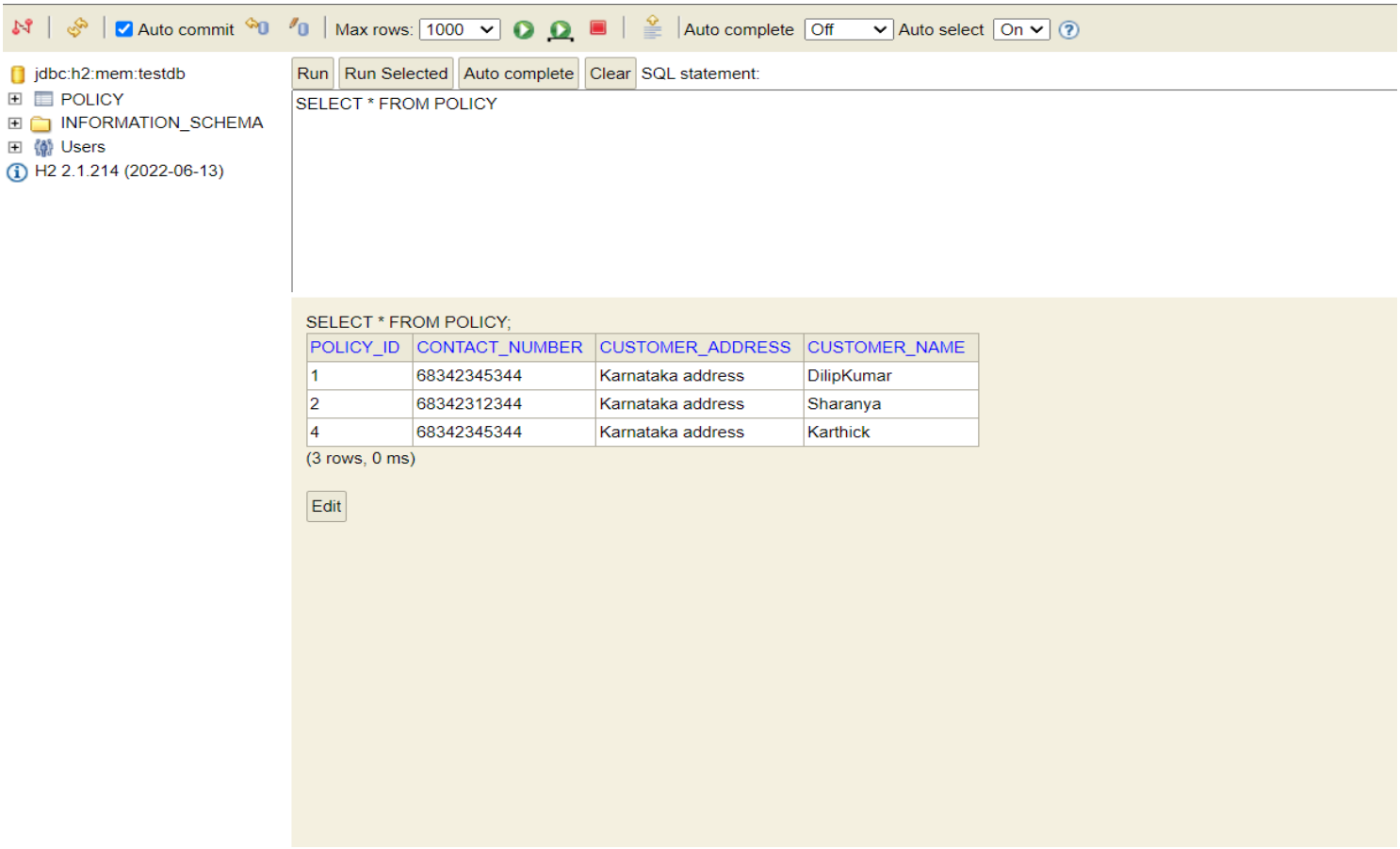
Delete : Delete endpoint



H2-Console database output:

http://*35.153.140.136*:8888/h2-console/login.do?jsessionid=bcba9df8165133f1dbf22b5e67cb6990





Submitted by

DilipKumar M
DevOps Certification batch

To

StarAgile training institute
Bangalore

Thankyou