Title: Deployment of Spring Boot Application on Kubernetes Using Helm

---

Introduction

- Brief introduction to Spring Boot and Kubernetes.
- Importance of container orchestration and the role of Helm in simplifying deployments.
- Overview of what will be covered in the blog post.

Prerequisites

- Basic understanding of Spring Boot, Docker, Kubernetes, and Helm.
- Tools and technologies required:
  - Spring Boot application
  - Docker
  - Kubernetes cluster (Minikube or any cloud provider)
  - Helm
  - Jenkins for CI/CD (optional)

Step 1: Setting Up the Spring Boot Application

- Create a simple Spring Boot application.
- Add a simple REST controller that returns a "Hello World!" message.
- Test the application locally to ensure it's working.

1) mvn compile
2) mvn package

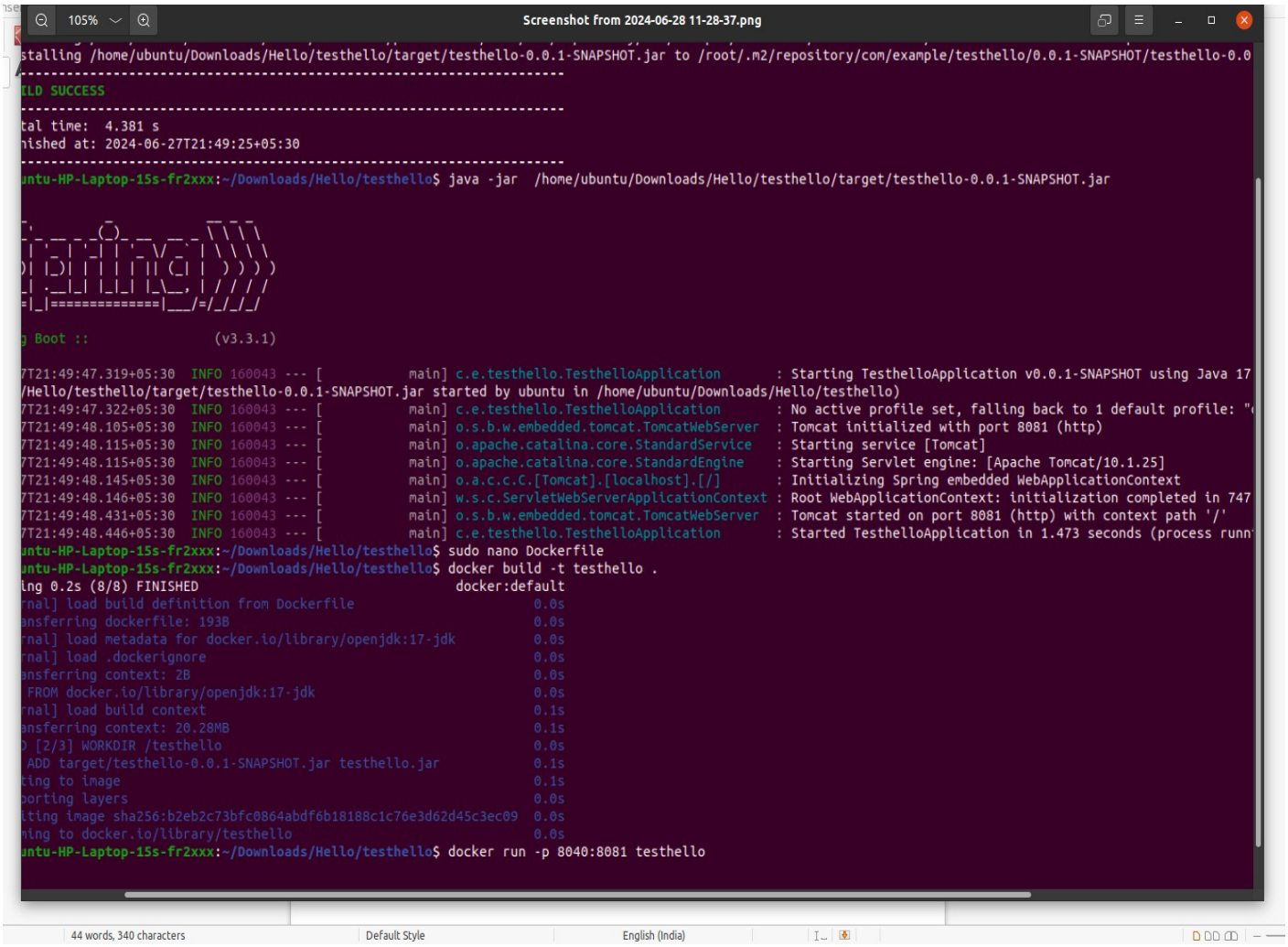
Step 2: Dockerizing the Spring Boot Application

- Create a Dockerfile for the Spring Boot application.
- Build the Docker image locally.
- Push the Docker image to DockerHub (or any container registry).
- Example Dockerfile

```
FROM openjdk:17-jdk

WORKDIR /testhello

ADD target/testhello-0.0.1-SNAPSHOT.jar testhello.jar

EXPOSE 8040

ENTRYPOINT ["java", "-jar", "testhello.jar"]
```

$ docker build -t testhello .

$ docker run -p 8040:8081 testhello

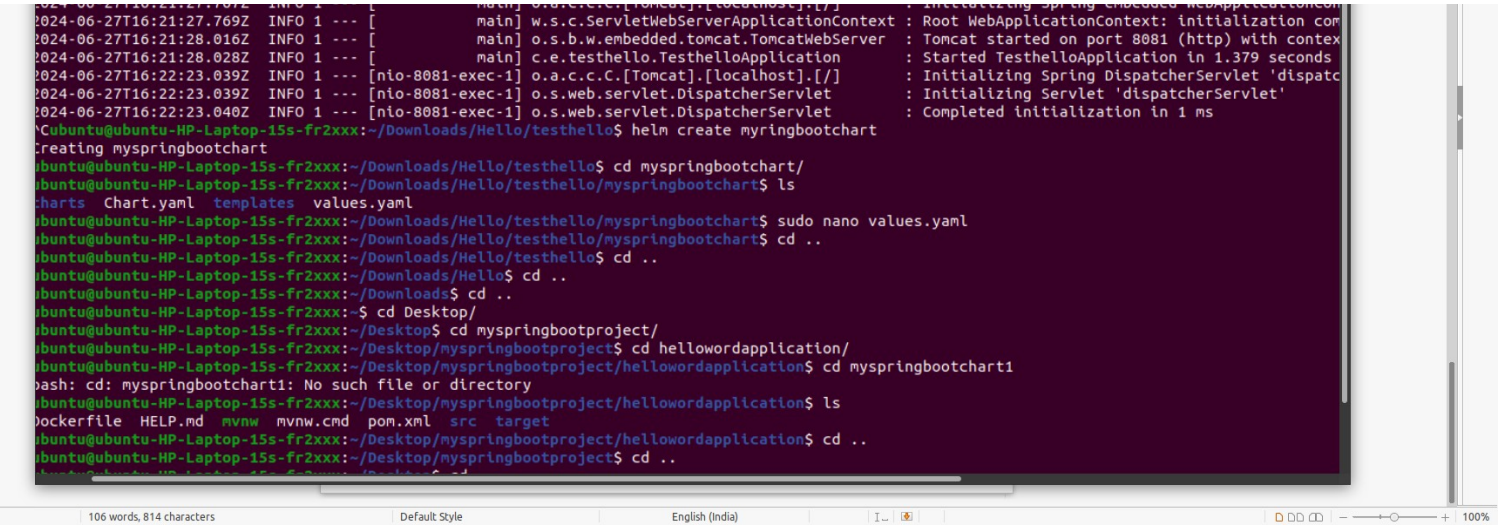


Step 3: Setting Up Kubernetes Cluster

- Set up a local Kubernetes cluster using Minikube or use a cloud provider (e.g., GKE, EKS, AKS).
- Ensure `kubectl` is configured to interact with the cluster.
- $ minikube start
- $ minikube status

Step 4: Creating Helm Charts

- Introduction to Helm and its components (Chart.yaml, values.yaml, templates, etc.).
- Create a Helm chart for the Spring Boot application.
- **$ helm create myspringbootchart**
    - Define the deployment and service templates.
    - Customize `values.yaml` for different environments (e.g., development, production).



<u>Edit this part:</u>

image:
  repository: pramila188/testhello
  tag: latest
  pullPolicy: IfNotPresent

service:
  type: NodePort
  port: 8081

Example Helm chart structure:

```
my-springboot-chart/
├── Chart.yaml
├── values.yaml
└── templates/
    ├── deployment.yaml
    ├── service.yaml
```

$ eval $(minikube docker -env)

again run docker build and run command

Step 5: Deploying the Application Using Helm

- Package the Helm chart.
- Install the Helm chart on the Kubernetes cluster.
- Verify the deployment and ensure the application is running.

$ helm install hellochart myspringbootchart

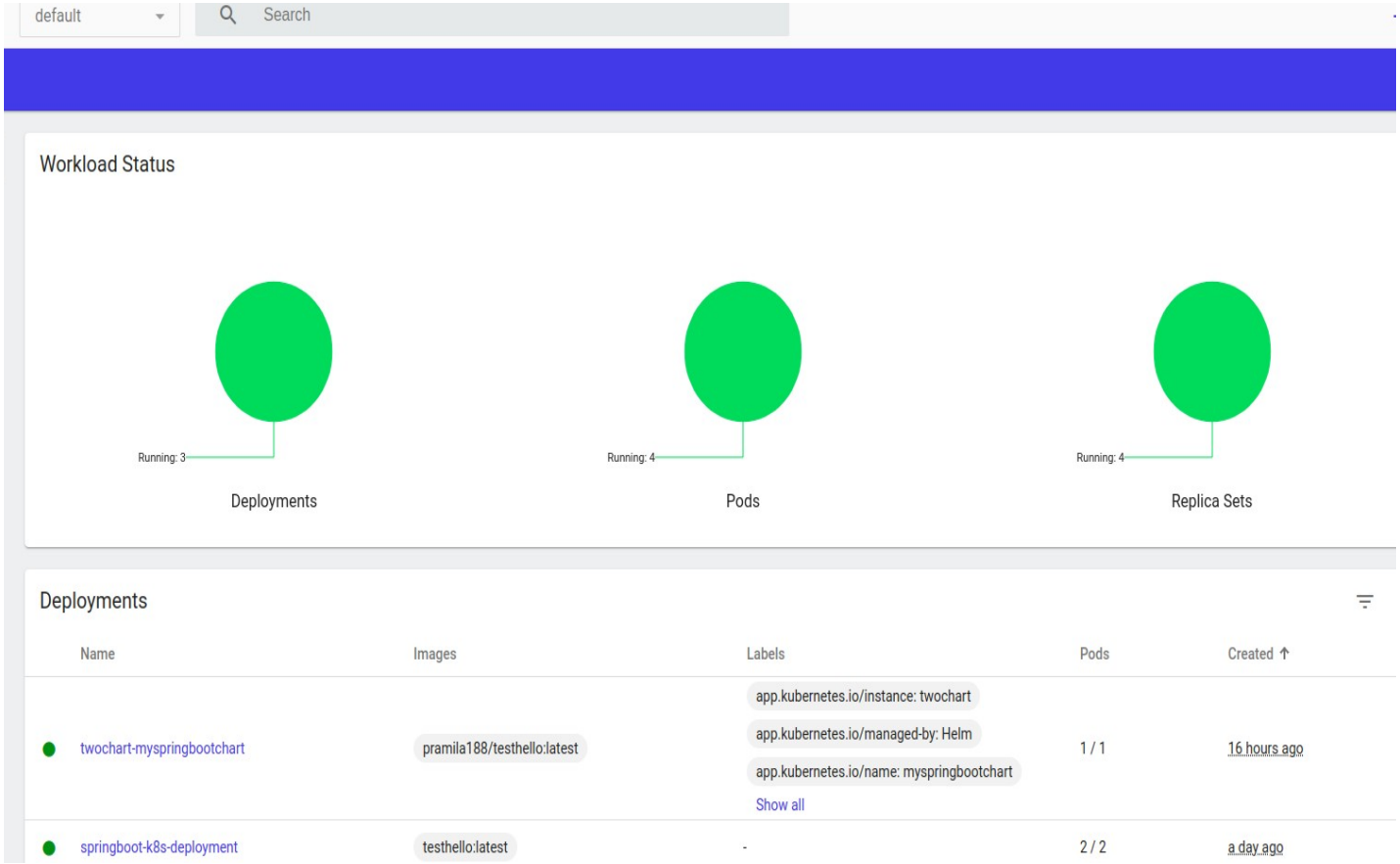
$ kubectl get pod

$ kubectl get svc

$ kubectl get deployment

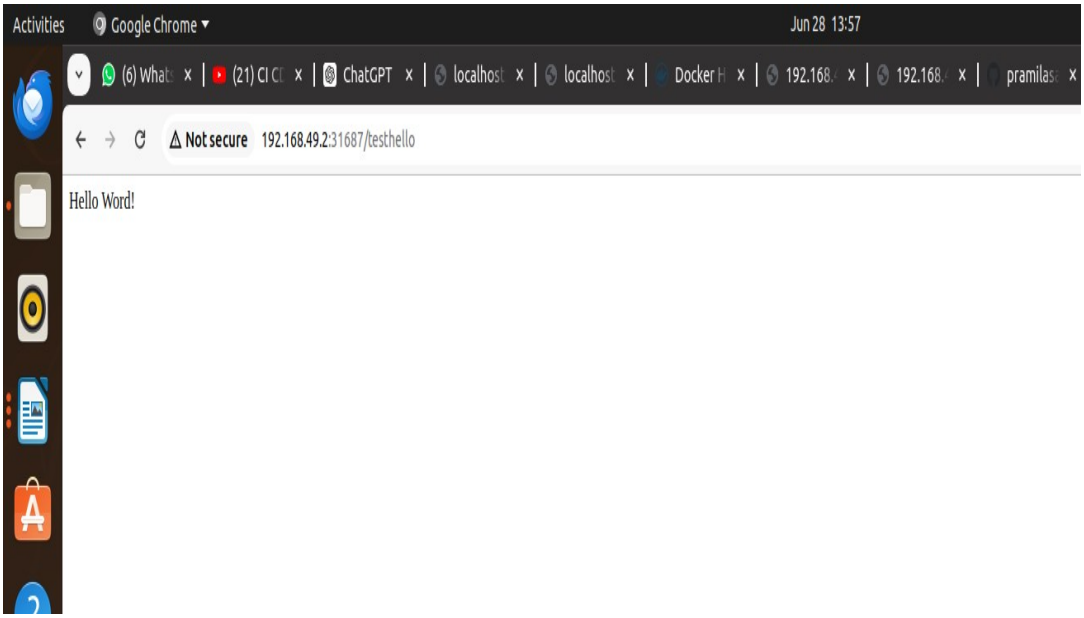
```
NAME: onechart
LAST DEPLOYED: Fri Jun 28 11:04:36 2024
NAMESPACE: test
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  http://chart-example.local/
ubuntu@ubuntu-HP-Laptop-15s-fr2xxx:~/Downloads/Hello/testhello$ kubectl get nodes
NAME       STATUS   ROLES           AGE     VERSION
minikube   Ready    control-plane   7d18h   v1.30.0
ubuntu@ubuntu-HP-Laptop-15s-fr2xxx:~/Downloads/Hello/testhello$ kubectl get pods
NAME                                          READY   STATUS             RESTARTS       AGE
hellochart-myspringbootchart1-d88f85574-pckg6   1/1     Running            10 (13h ago)   7d18h
springboot-k8s-deployment-68dbc55d54-75jkn      1/1     Running            2 (13h ago)    23h
springboot-k8s-deployment-68dbc55d54-ncmrn      1/1     Running            3 (94m ago)    23h
twochart-myspringbootchart-597b9779b9-f7cfp     0/1     ImagePullBackOff   0              6m35s
ubuntu@ubuntu-HP-Laptop-15s-fr2xxx:~/Downloads/Hello/testhello$ kubectl get pods --test
error: unknown flag: --test
See 'kubectl get --help' for usage.
ubuntu@ubuntu-HP-Laptop-15s-fr2xxx:~/Downloads/Hello/testhello$ helm list
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS        CHART
hellochart      default         1               2024-06-20 16:30:52.022648932 +0530 IST deployed      myspringbootchart1
onechart        default         1               2024-06-27 23:27:24.963134825 +0530 IST deployed      myspringbootchart-
twochart        default         1               2024-06-27 23:40:11.749108258 +0530 IST deployed      myspringbootchart-
ubuntu@ubuntu-HP-Laptop-15s-fr2xxx:~/Downloads/Hello/testhello$ kubectl get pod
NAME                                          READY   STATUS             RESTARTS       AGE
hellochart-myspringbootchart1-d88f85574-pckg6   1/1     Running            10 (13h ago)   7d18h
springboot-k8s-deployment-68dbc55d54-75jkn      1/1     Running            2 (13h ago)    23h
springboot-k8s-deployment-68dbc55d54-ncmrn      1/1     Running            3 (98m ago)    23h
twochart-myspringbootchart-597b9779b9-f7cfp     0/1     ImagePullBackOff   0              9m52s
ubuntu@ubuntu-HP-Laptop-15s-fr2xxx:~/Downloads/Hello/testhello$ kubectl get pod -A
NAMESPACE              NAME                                          READY   STATUS             RESTARTS       AGE
default                hellochart-myspringbootchart1-d88f85574-pckg6   1/1     Running            10 (13h ago)   7d18h
```

$ minikube dashboard



$ minikube service hellochart myspringbootchart –url

Final output:



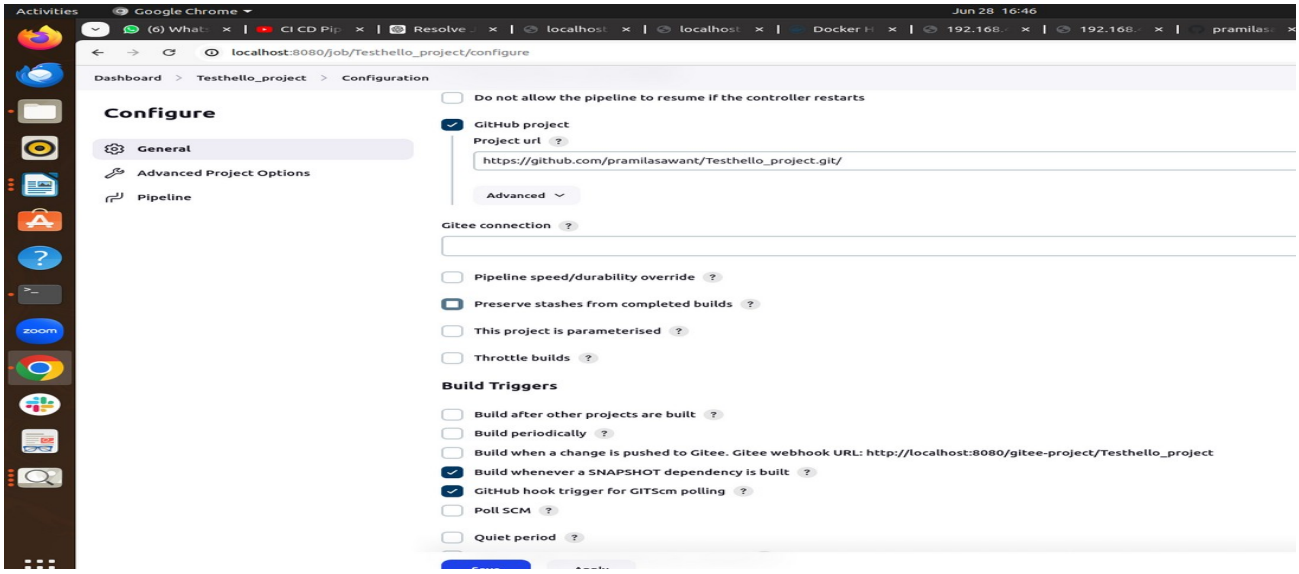
Push this project folder into new github repo.

Step 6: Setting Up Continuous Deployment with Jenkins (Optional)

- Introduction to Jenkins and its role in CI/CD.
- Create a Jenkins pipeline to automate the build, Docker image creation, and Helm deployment.

Open jenkins dashboard create new pipeline job with added plugin maven,Docker,kubernetes,kubernetes continous deployment etc...

create new pipeline job with adding project github url



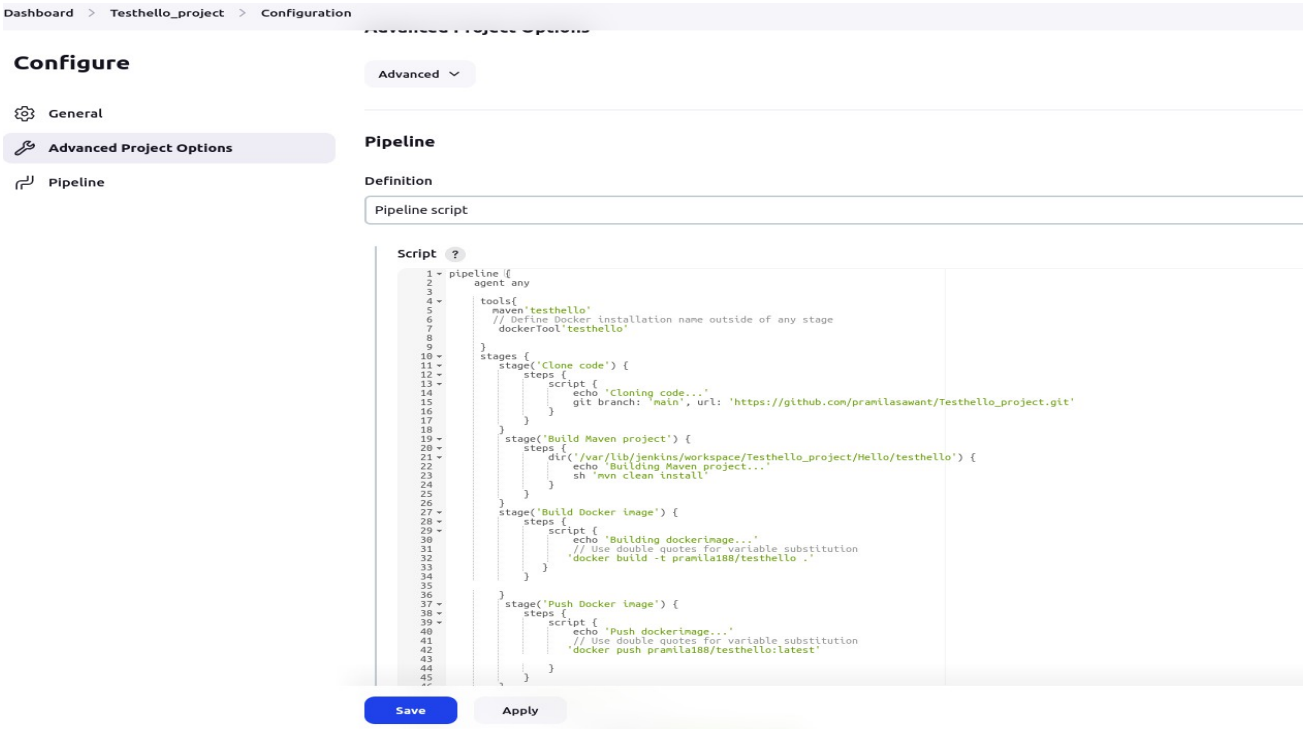
l

Step by step run the script:

1) Clone code

2) build maven project

3) build and push docker image into dockerhub



Pipeline script is as follows:

```
pipeline {
    agent any
     tools{
       maven'testhello'
       // Define Docker installation name outside of any stage
        dockerTool'testhello'
          }
     stages {
        stage('Clone code') {
            steps {
                script {
                    echo 'Cloning code...'
```

```
                git branch: 'main', url:
'https://github.com/pramilasawant/Testhello_project.git'
            }
        }
    }
    stage('Build Maven project') {
        steps {
            dir('/var/lib/jenkins/workspace/Testhello_project/Hello/testhello')
{
                echo 'Building Maven project...'
                sh 'mvn clean install'
            }
        }
    }
    stage('Build Docker image') {
        steps {
            script {
                echo 'Building dockerimage...'
                // Use double quotes for variable substitution
               'docker build -t pramila188/testhello .'
            }
        }
    }
    stage('Push Docker image') {
        steps {
            script {
```

```
                echo 'Push dockerimage...'
                // Use double quotes for variable substitution
               'docker push pramila188/testhello:latest'
                    }
            }
        }
        stage('Deploy on k8s') {
            steps {
                script {
                    kubernetesDeploy(
                        configs: 'deploymentservice.yaml',
                        kubeconfigId: 'k8sconfigpwd'
                    )
                }
            }
        }
    }
}
```

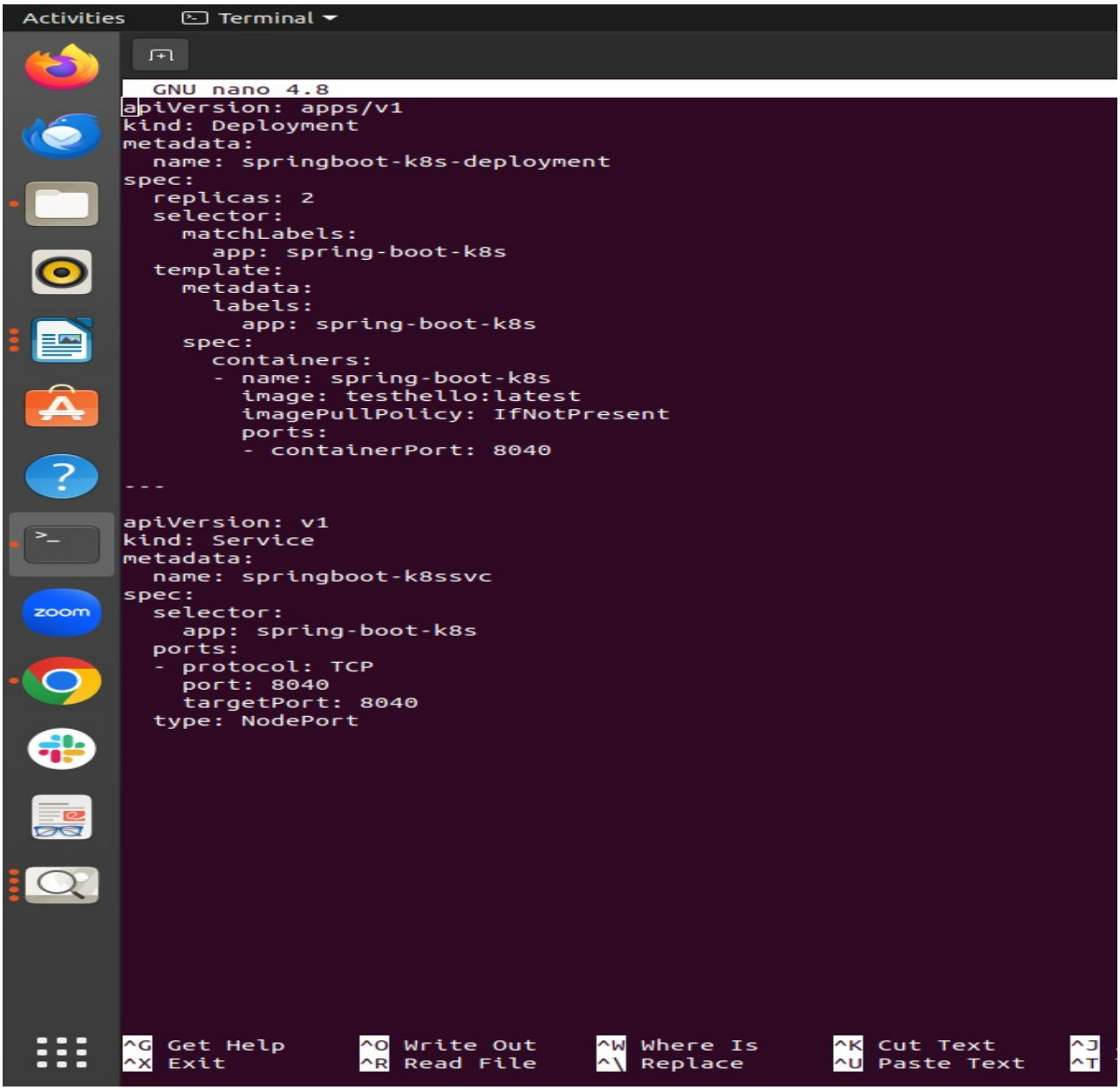
Open the terminal:

$ minikube start

$ minikube status

$ eval $(minikube docker -env)

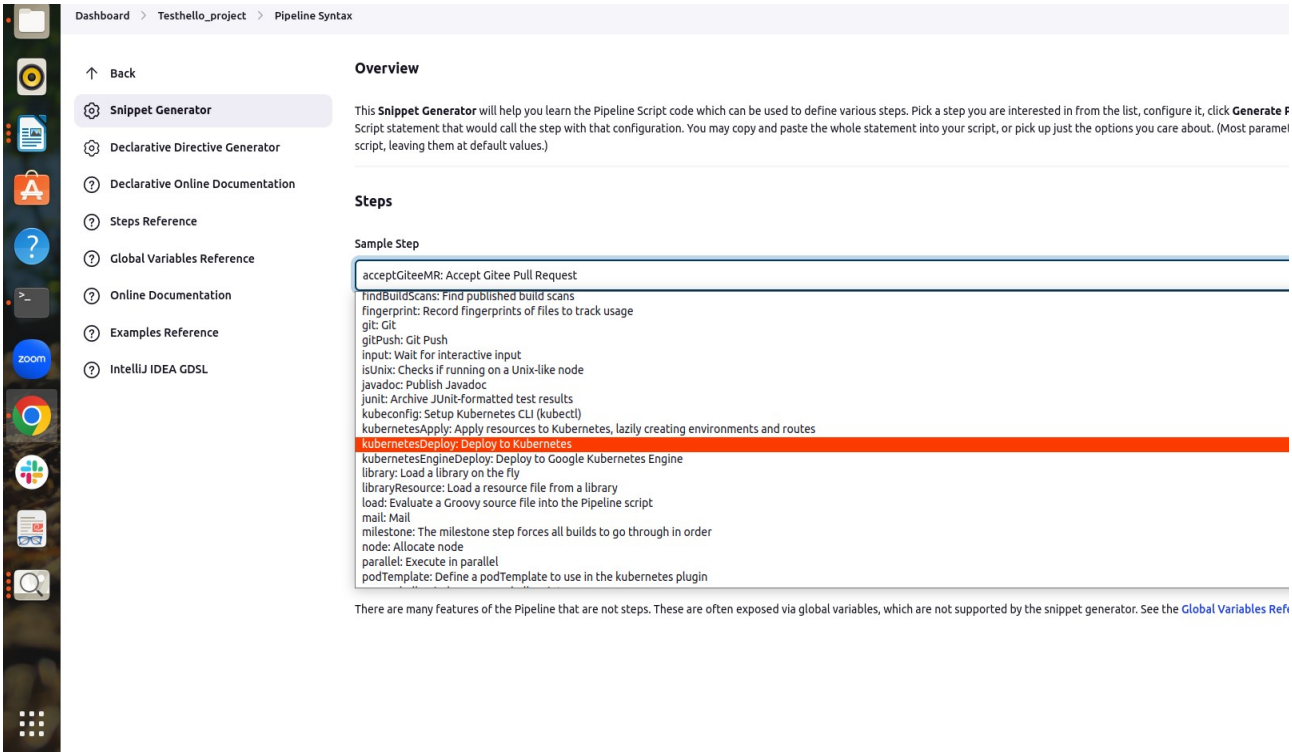
Create deploymentservice.yaml file in this project github repo.

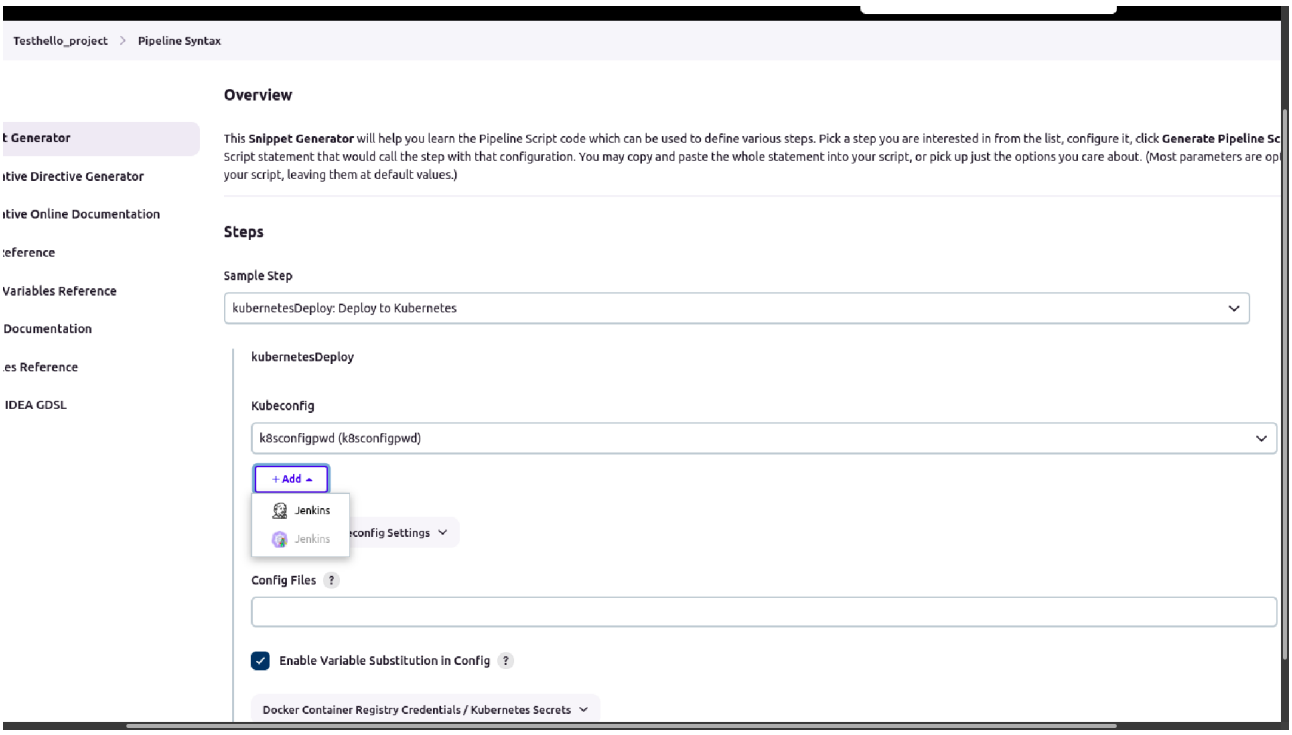


Open this pipeline job and click on pipeline syntax

click into kubernetes Deploy:Deploy to kubernetes

in kubeconfig section click on jenkins and select kubeconfigpwd





Add the Credentials:

Username: Kubernetes configuration(Kubeconfig)

Password: kubeconfigpwd or any something

click on Enter Directly

open the terminal and go to root directory

$ ls -la

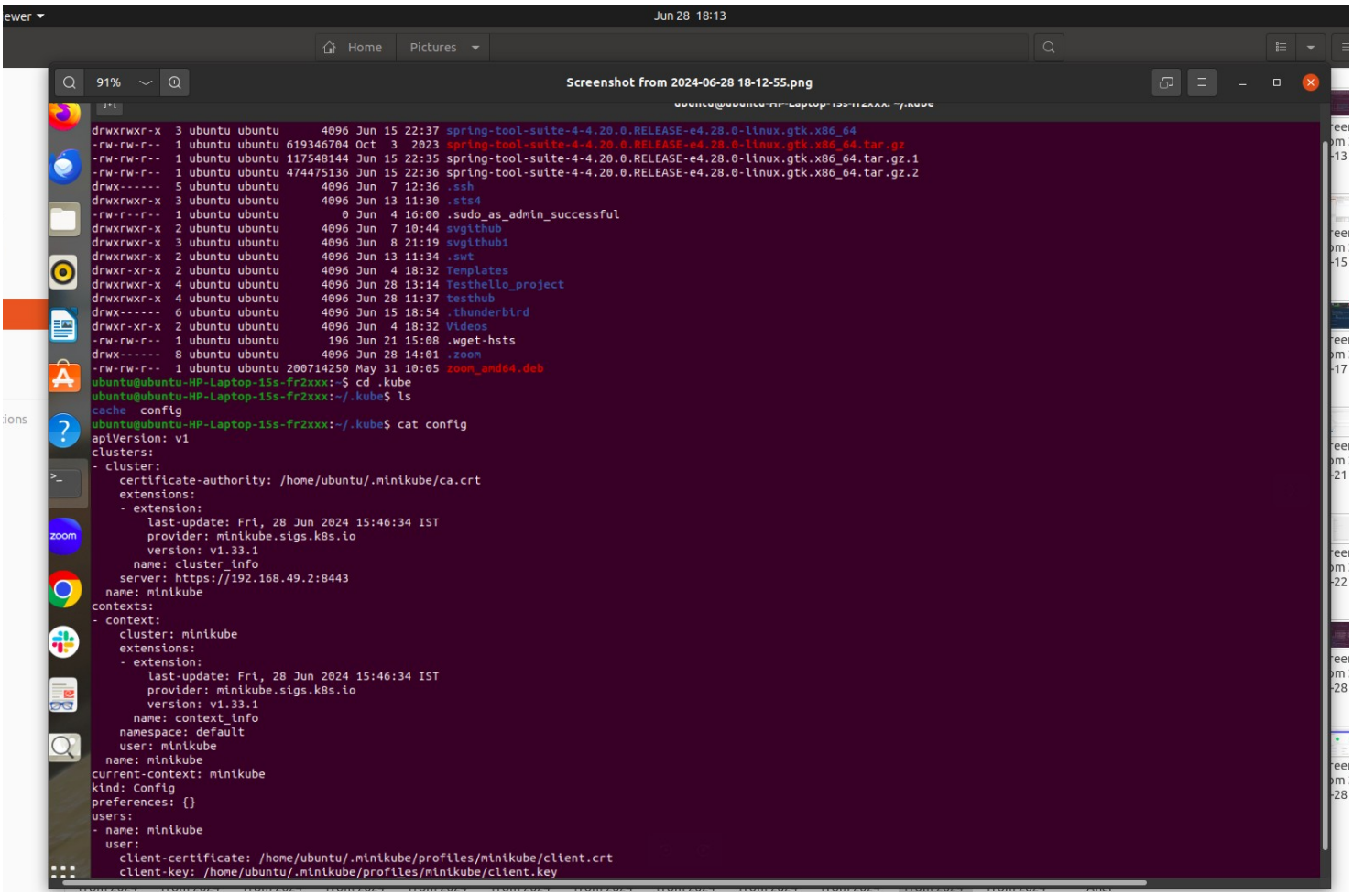
show all hidden file ....open the .kube

$ cd .kube

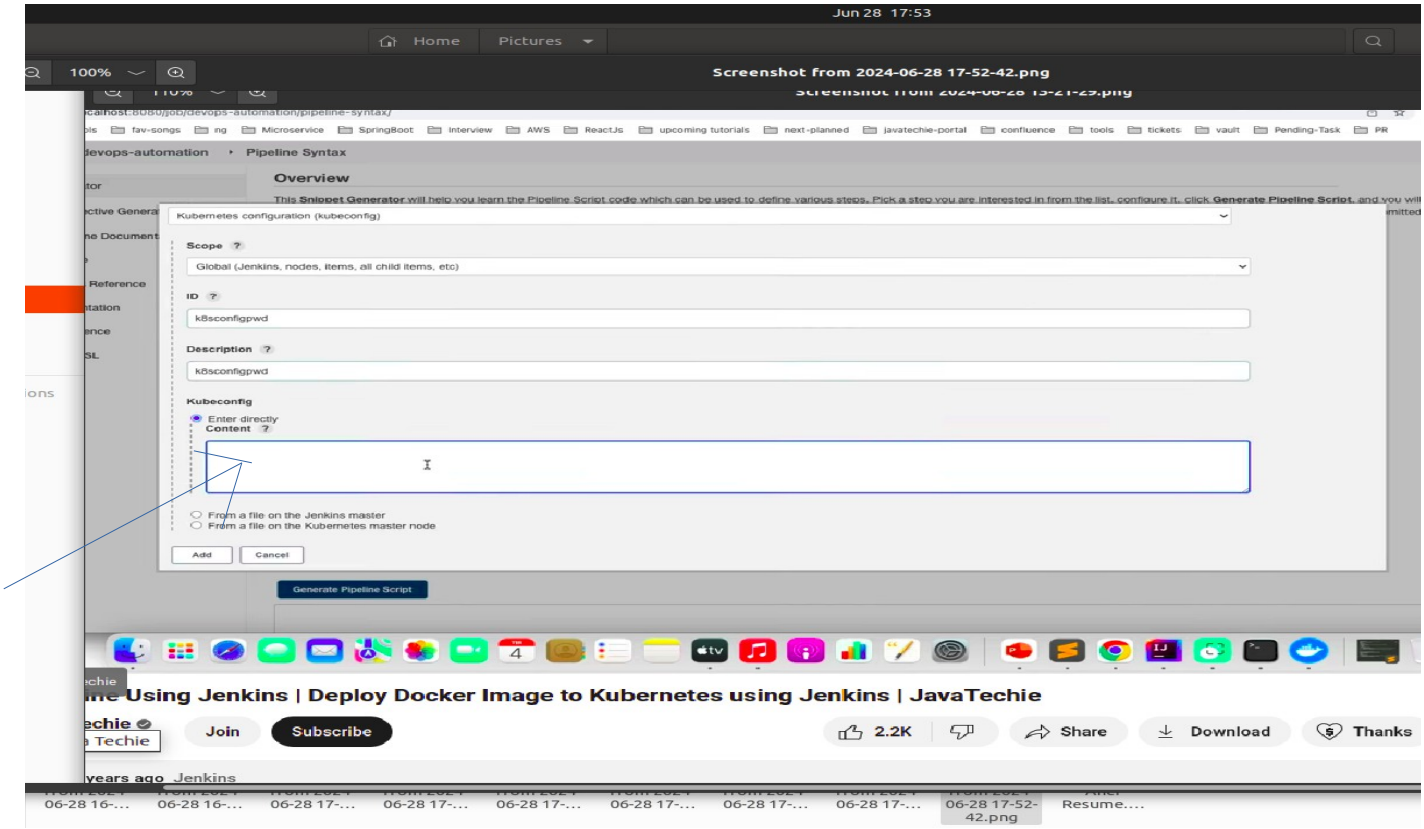
$ ls

open config using

$ cat config

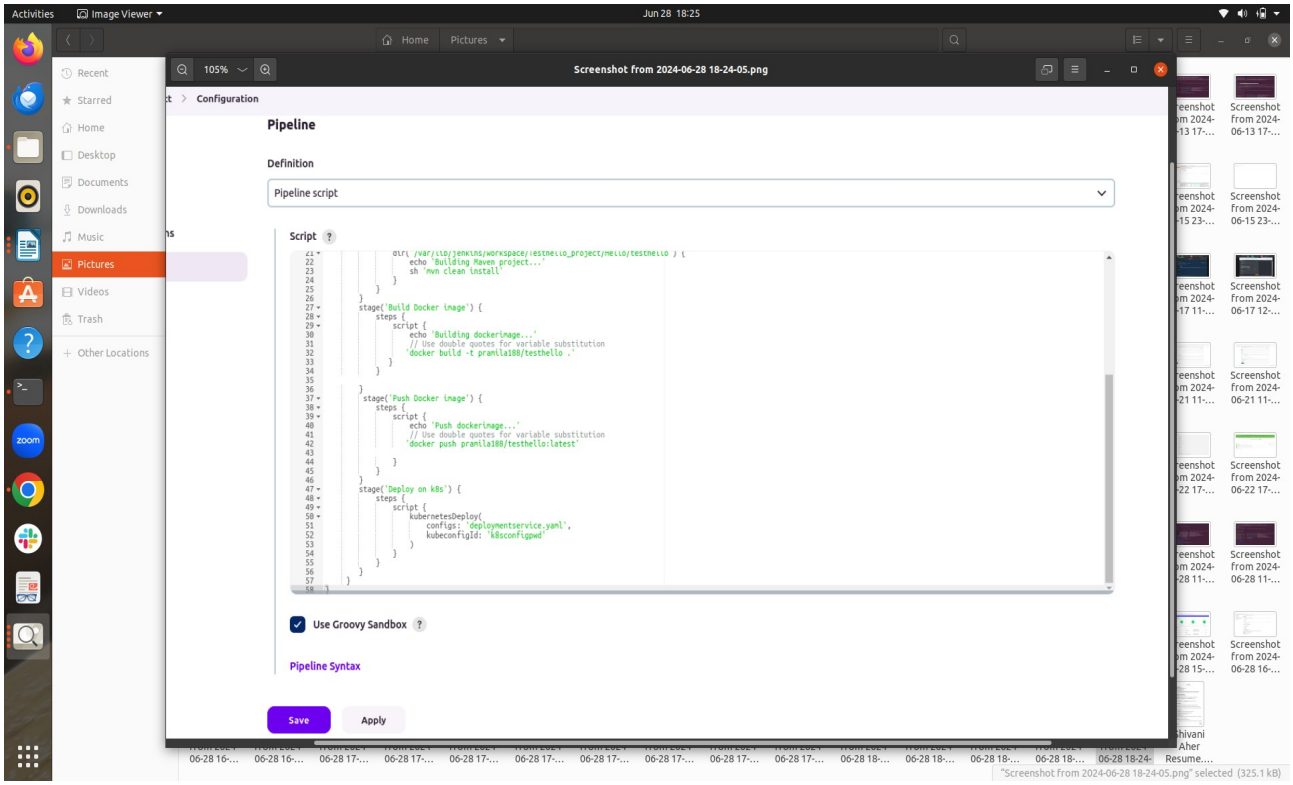


copy this config file and into highlighted box



click on Add and Generate pipeline script

copy this contetnt and paste into k8s deployment script



save the script and Build now

when build success open terminal

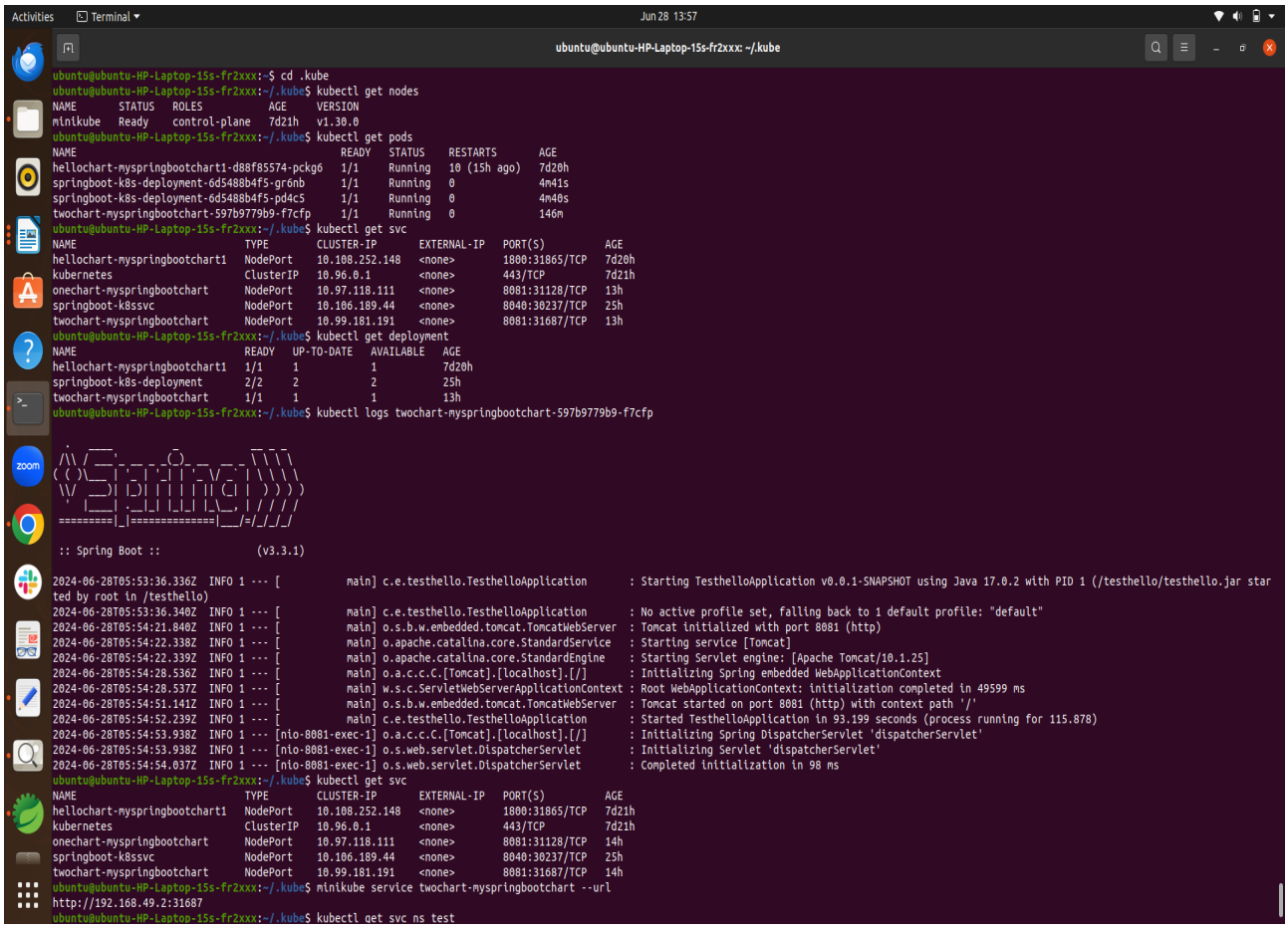
$ kubectl get pod

$ kubectl get node

$ kubectl get deployment

$ kubectl get svc

$ minikube service service name –url



open this url shows final output: