---Fill in the steps to install python, django, docker, docker-compose--

DOCKER INSTALLATION:

SETTING UP DOCKER'S APT REPOSITORY

# Add Docker's official GPG key:

sudo apt-get update

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

INSTALLING DOCKER PACKAGES:

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

INCASE OF permission denied while trying to connect to the Docker daemon socket ERROR, RUN THE FOLLOWING COMMAND TO CREATE A NEW USER AND A GROUP

sudo usermod -aG docker $USER && newgrp docker

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Step 1: Creating Python Django Web Application

1. After proceeding with all the necessary installations, create a directory naming "project" in the desired location and enter the directory.

2. Enter the command <django-admin startproject mysite> and it will create "mysite" directory as referred in the command and also creates "manage.py" executable python file and "mysite" directory comprising the following ------insert snap of the files inside mysite directory-------

3. Run the command <python3 manage.py runserver 0.0.0.0:8000> to check whether the demo django application is running or not. In order to successfully open the django page, it is necessary to change the inbound rules of the current EC2 instance.

3. On the EC2 Dashboard, go to "Instances" and click on the desired instance ID and click on the "security groups" link under "security" tab. select the option "Edit inbound rules", click on "Add rule" option and provide port range as "8000" and save rules.

4. Copy the public IPV4 Address under instance ID. enter mysite directory and edit the file "settings.py" by passing the command nano settings.py. paste the copied public IPV4 address inside "ALLOWED\_HOSTS =[<IPV4 ADDRESS>]" and save the changes.

5. Open a new tab and paste the ip address by exposing the port 8000. http://<public ipv4 address >:8000 and hit enter. The demo django web application will look like this

-----insert demo django app-----

6. Go to the path project/mysite/ and run the command "python3 manage.py startapp hello\_world" to create the app directory "hello\_world" with necessary auto populated files where "urls.py" is manually created to deal with the URLpatterns of the web-application. -----insert helloworld url snap-------

7. Go to the path project/mysite/hello\_world/ and edit "views.py" file with the following code to deal with requests and responses regarding web-application ----insert hw views snap----

8.Go to the path project/mysite/mysite/ and edit "urls.py" with the following code and save the changes ---mysite urls snap-----

9. Now, run the server again in the main mysite directory with the command <python3 manage.py runserver 0.0.0.0:8000> and the output should look like this: ----insert helloworld snap.

10. Finally, test the entire web application code by running the tests.py file through command <python3 manage.py test> --T&o/p—

11. Code for the web application is located on “mysite” directory of given Github repository:

<https://github.com/SIDDHUGITTY/upgrad_project/tree/master/project>

Step 2: Containerizing Python Django Web Application

1. Firstly create a file named as "Dockerfile" without extension in the path project/mysite/ as follows: ---snap---

References:

FROM Create a new build stage from a base image.

ENV Set environment variables.

WORKDIR Change working directory.

ADD Add local or remote files and directories.

COPY Copy files and directories.

RUN Execute build commands.

2. create a file named as "docker-compose.yml" in the path project/mysite/ as follows: -----snap----

3. create "requirements.txt" to mention the required web servers as follows: ----snap----

4. Run "docker-compose build" command to build the image out of the application along with a tag enclosed and "docker-compose up" to run the container based on the docker image. ----docker snaps----

5. Run "docker login" command and provide username and password of the docker hub account and tag the image as follows:

docker tag <image name>:<tag> <repository name>:<new tag>

6. Now, run the command, docker push <latest docker image after tagging>:<tag>

------------------------------Version control system------------------------

1. Go to the desired directory to initialize the git and run command "git init" and run "git status" to view the unstaged files and folders on that particular initialized directory.

2. Inorder to stage, run "git add <directory/file name>" to selective staging or "git ." to stage all available directories and files.

3. Run "git commit -m <message regarding commit>" to commit and "git remote add origin <URL>" to connect local repository to the remote repository.

4. run "git push -u origin master" to push local repository to the master branch where '-u' refers to upstream and provide username & Personal Access Token to complete the push.

5. Switched from master branch to main branch can be done by the command "git branch -M main" and need to specify while pushing by the command "git push -u origin main".

Step 3: Set Up Minikube Kubernetes Cluster

1. MINIKUBE INSTALLATION:

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

2. KUBECTL INSTALLATION:

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl version –client

3. Run the command “minikube start” to start the minikube cluster in the current machine. In case of any startup issues, pass the command with additional argument “--driver=docker” or “—driver=hyperkit”. Docker desktop should be running in the background to start minikube cluster, but with the use case of virtual machine, docker desktop is not mandatory.

--------minikube start snap-----

Step 4: Deploy Python Django Web Application on Minikube Kubernetes

1. In Kubernetes, pods are the short lived, smallest deployable units of computing which consists of one or more containers with shared storage, network resources and a specification for how to run the containers.

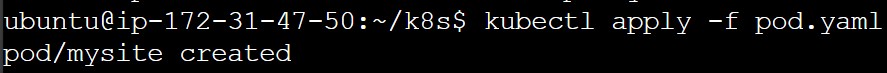
2. As the pod lifecycle is shorter in span, it is preferred to launch the deployments with replicaset object where it deals with the scalability factor of the pods running within the cluster.

3. Firstly, create a pod by writing YAML file naming it as “mysite” using vim editor and save the written YAML file. Run the command “kubectl apply -f mysite”. The definition for the pod is as follows:

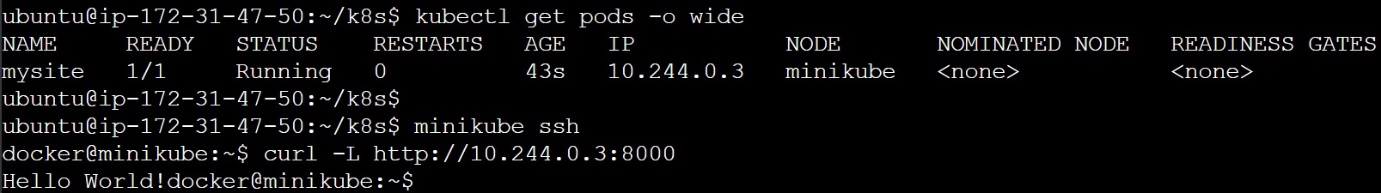


Here,

* Name of the pod is mentioned under metadata in the “name” section.
* Name of the container under “spec” section.
* Name of the docker image, from where the image is pulled and used to run the pod.
* Command to be executed to run the web application along with port number.



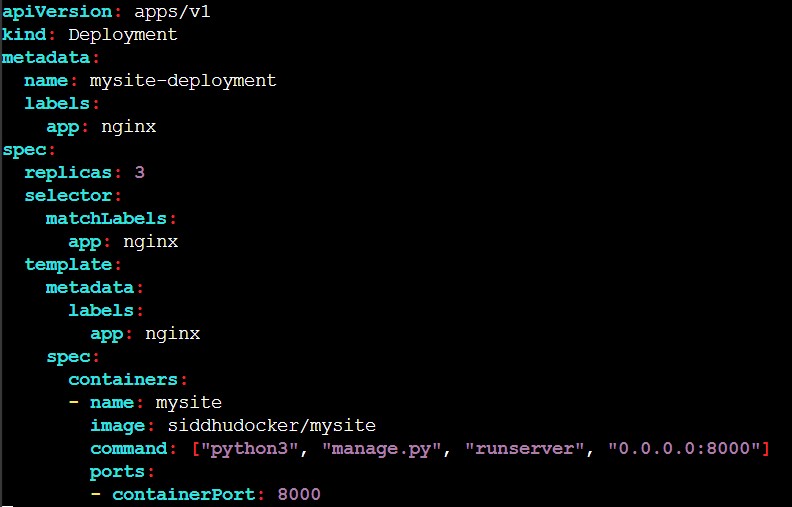
4. Run command “kubectl get pods” to view the running pods where the status should be “Running”. Test whether the web application is running properly in the minikube cluster by using minikube ssh command and curl along with designated port as 8000 in this case.



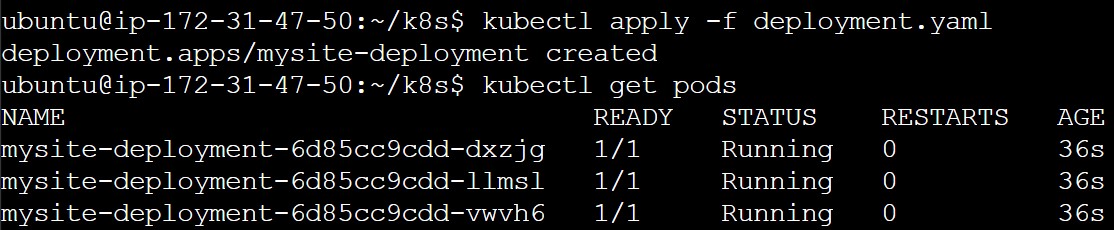
5. A pod can be deleted or may ran out of its lifecycle after some period of time. In order to meet the stability and scalability factor to the pods, replicasets were implemented while deploying an application in the server.

6. Pods will be deployed in the numbers as mentioned in the YAML file and a new one will be auto populated whenever any of the pods are down due to any aforesaid circumstances.

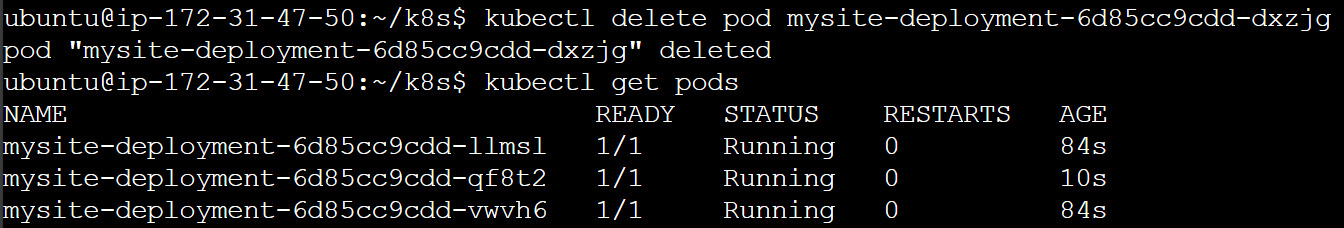
7. The definition for the deployment YAML file is as follows:



8. Apply the changes and view the number of replicas of deployment running in the Kubernetes cluster as follows:



9. Now delete a pod and run command “kubectl get pods” to view newly auto populated pod.



10. A web application has to be accessed by users outside Kubernetes cluster. In order to enable network access to the running pods, a service should be created.

11. Service YAML definition is as follows:



Here,

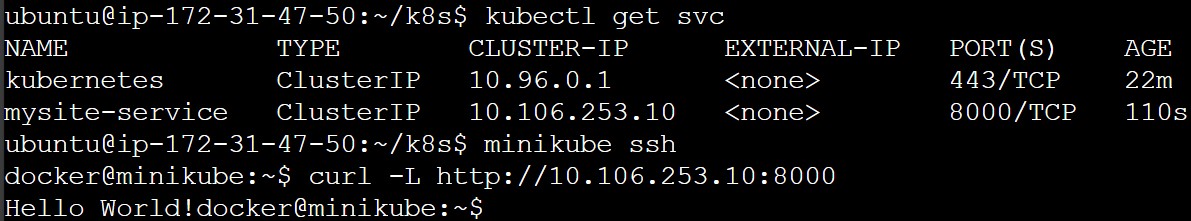
\* Name of the service is mysite-service under metadata section

\* Nginx app under selector of spec group refers to the deployment match selector where both names of the selector need to be matched to deploy a service.

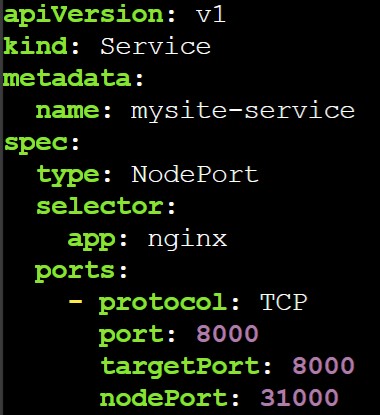
\* Target port is the port on which the service will send requests to.

12. Apply changes to the service YAML file and run service locally to test whether it is running properly.





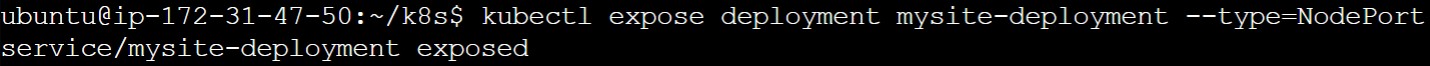
13. Nodeport exposes a service externally to the cluster by means of the target nodes IP address and the Nodeport. Provide nodeport number in the service definition to expose externally as follows:



14. Run the service with minikube IP binding with nodeport as mentioned in the definition



15. Expose the service with nodeport as follows:

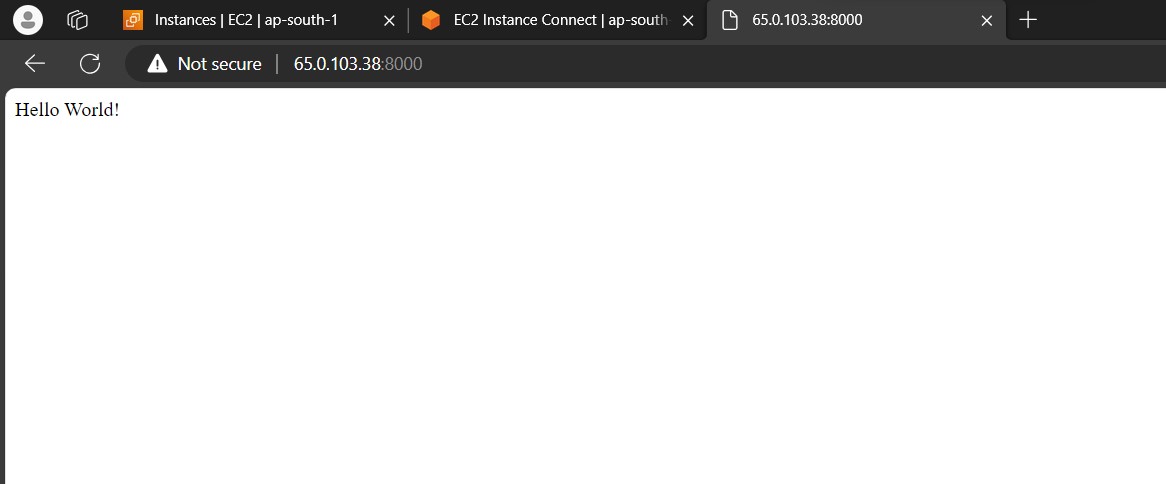


16. Port-forward the service to the designated ports as follows:



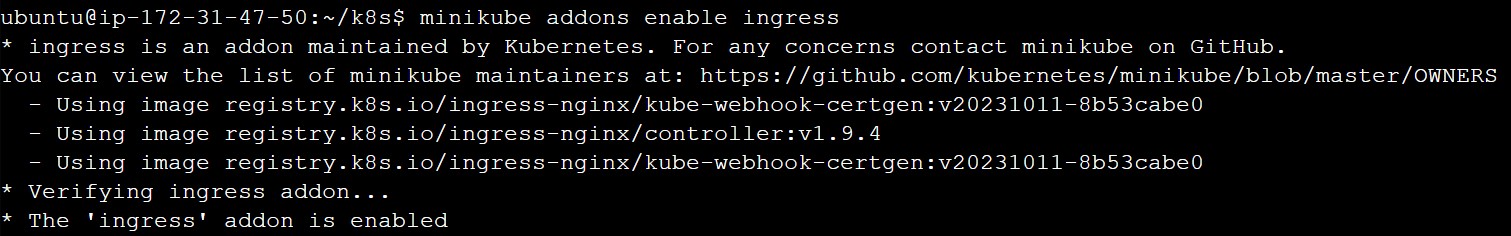
17. Run the service with public IP and port number 8000



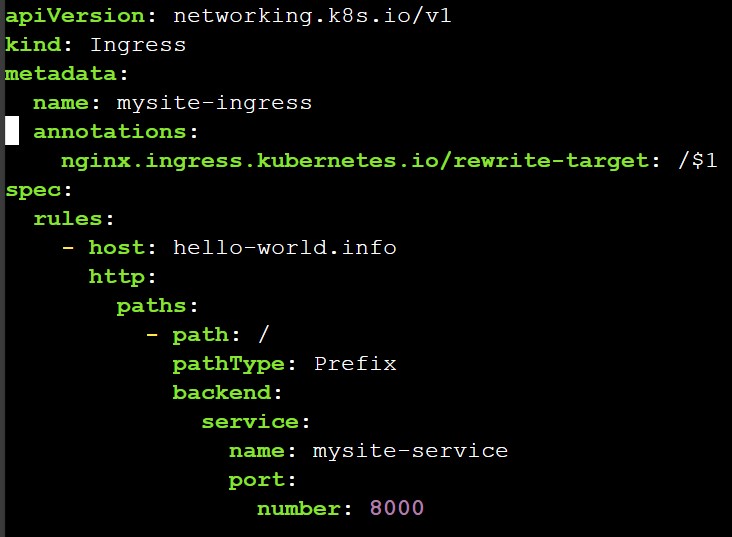


18. Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

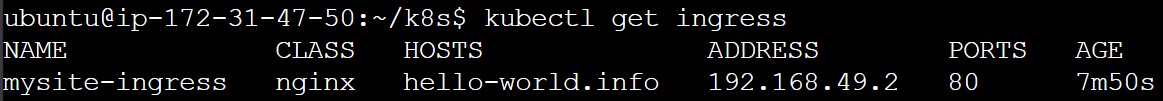
19. To implement ingress in kubernetes, firstly ingress addon should be enabled as follows



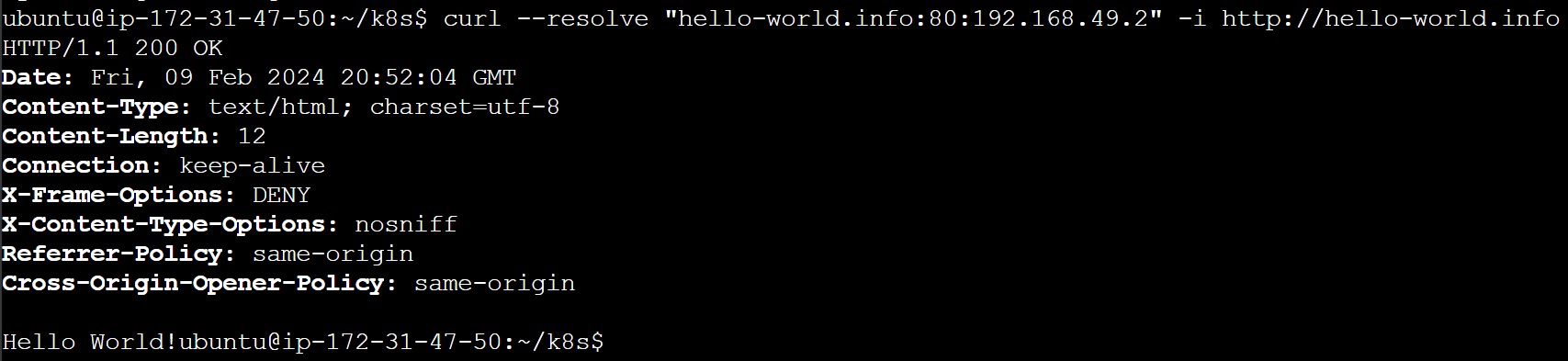
20. An ingress definition should be defined in YAML file in the following manner



21. Create the ingress by using the command “kubectl apply -f mysite-ingress”.

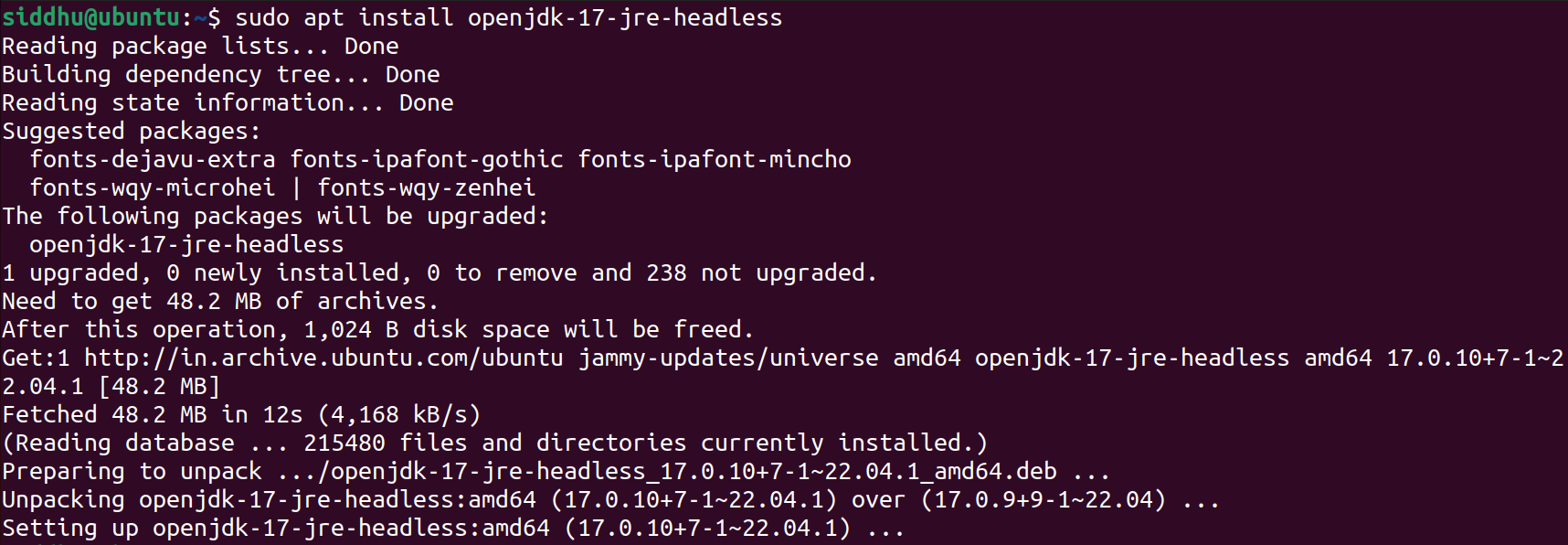


22. Run ingress using curl command



Step 5: Implement Jenkins CI/CD Pipeline

1. Firstly, install java as the prerequisite to Jenkins as follows



2. Install Jenkins from the documentation page

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

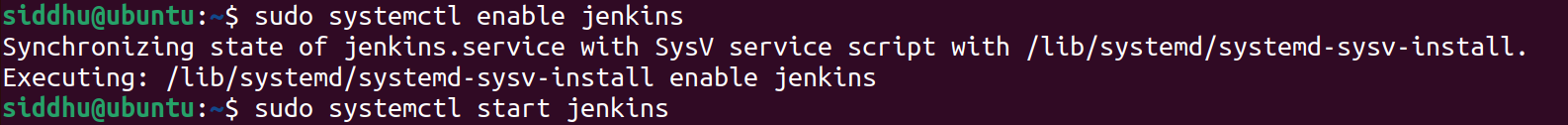
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

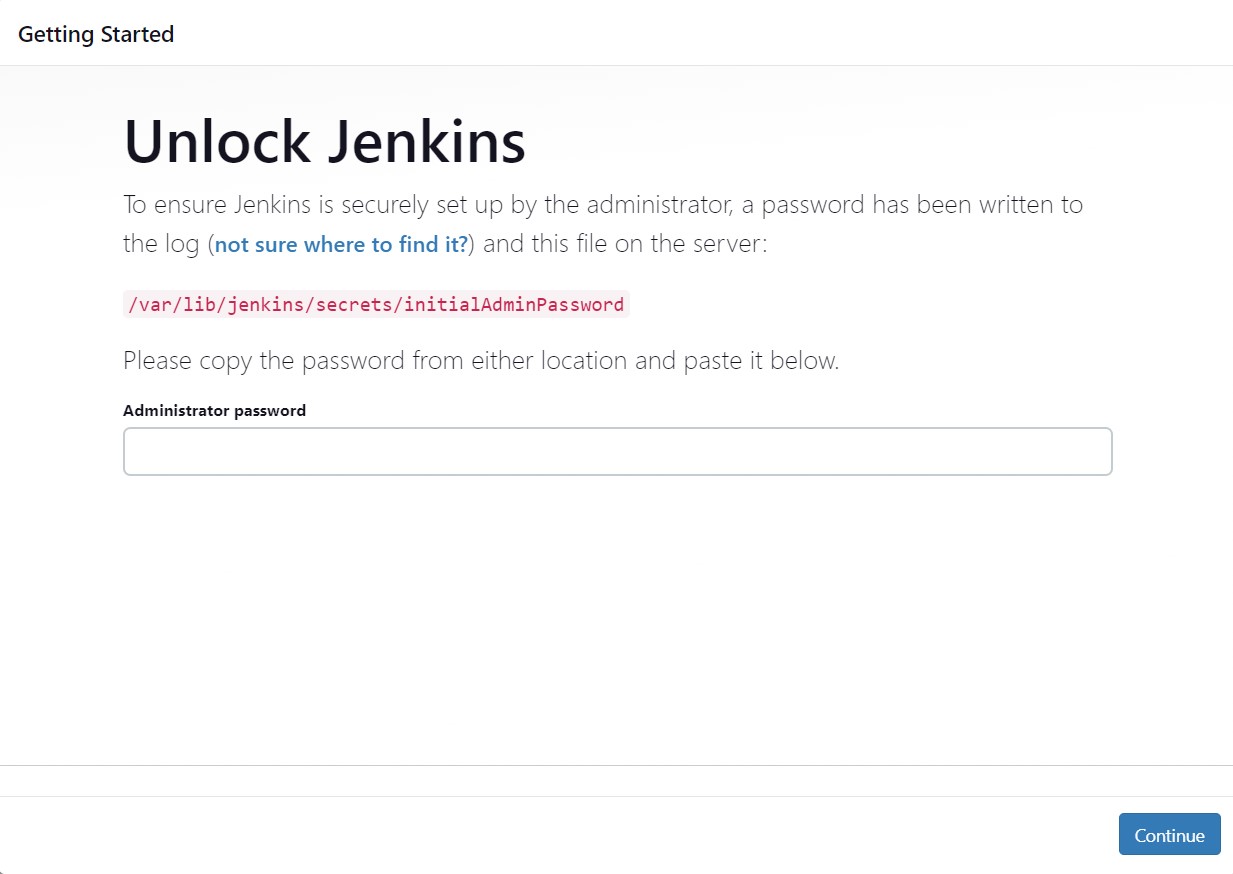
sudo apt-get install Jenkins

3. In order to build the Jenkins pipeline, firstly enable and start Jenkins server and to check Jenkins server’s running activity, run the status command.

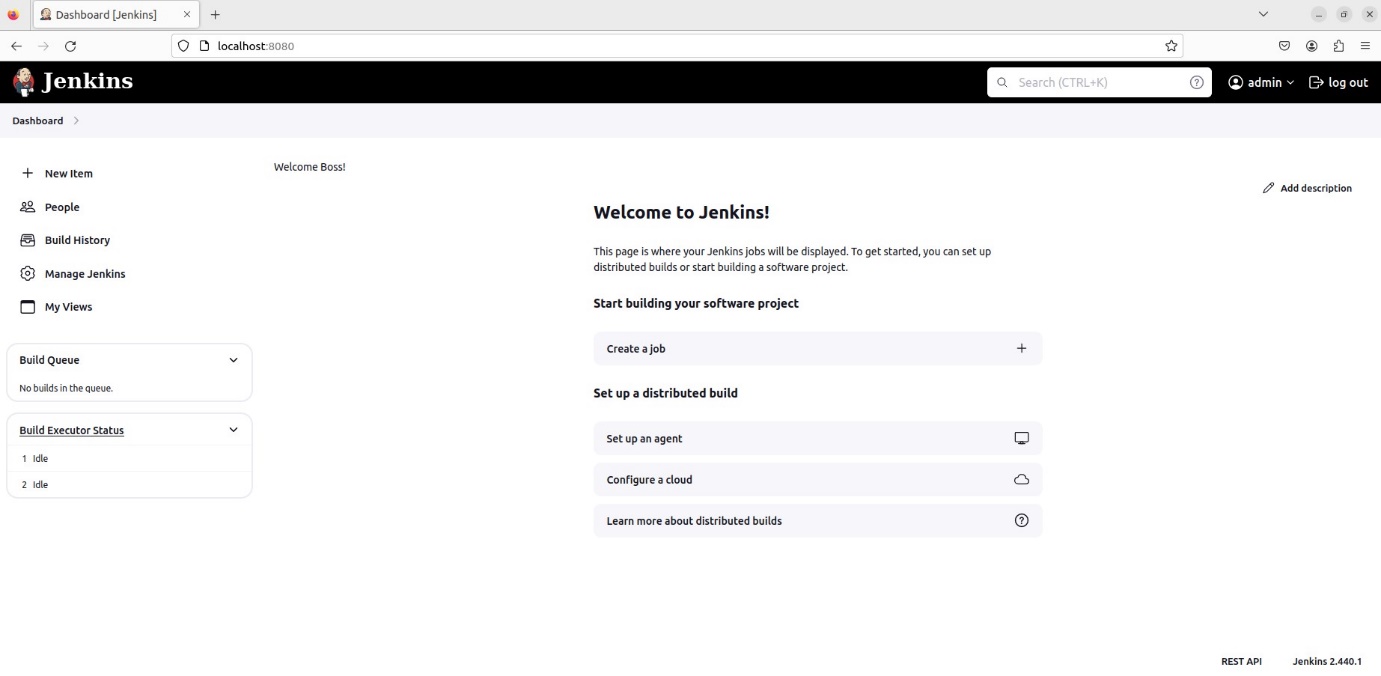




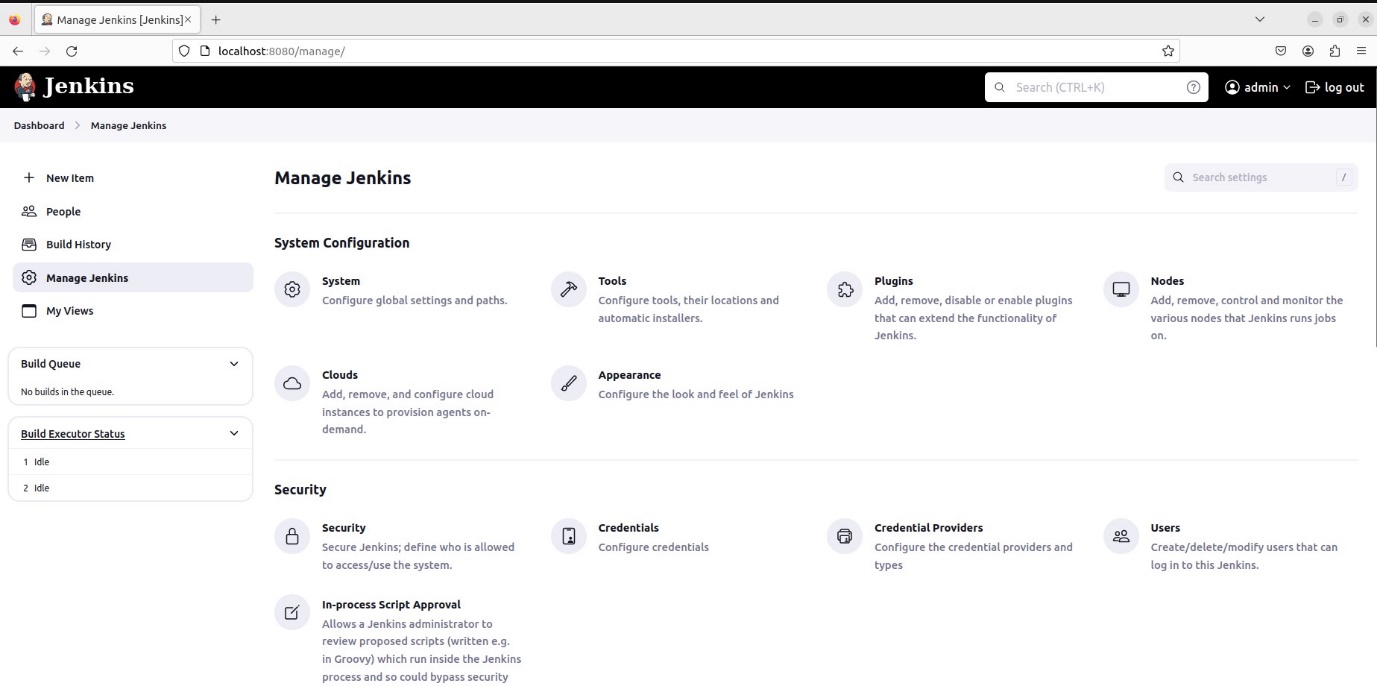
5. Go to Jenkins server via localhost with default Jenkins port number 8080 i.e., <https://localhost:8080> and login administrator password should be provided where it is located at the root path “/var/lib/Jenkins/secrets/initialAdminPassword”.



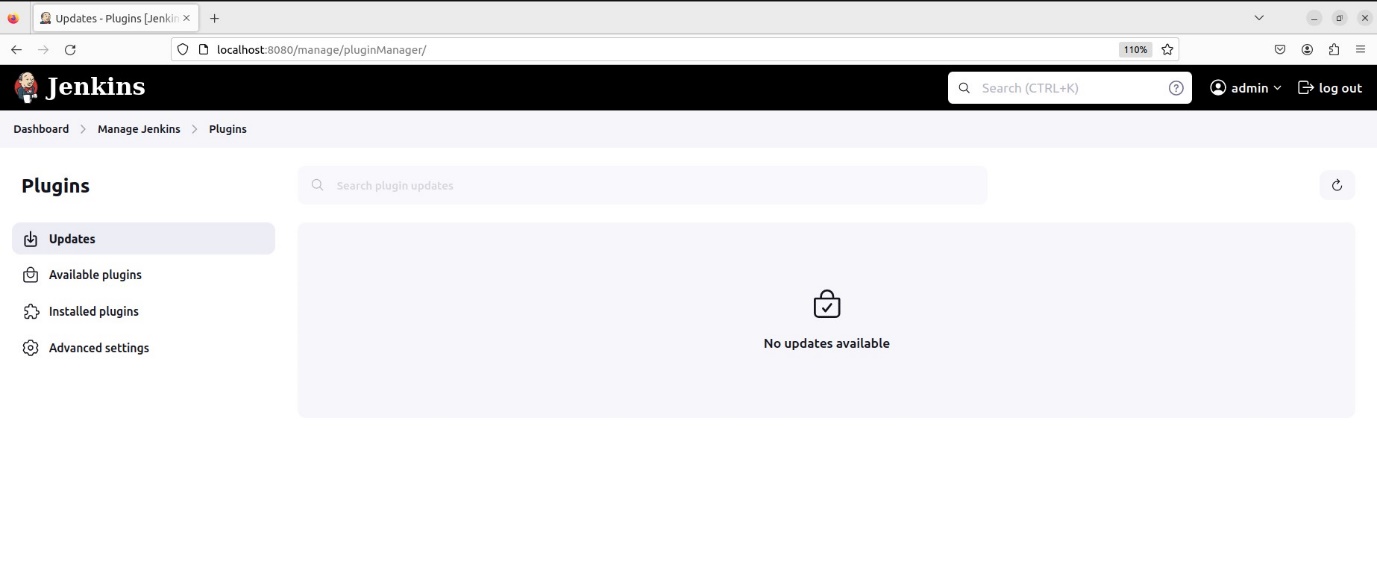
6. On successful logging, install the suggested plugins or choose custom installation of required plugins to work with and also assign username, password, email address as requested. After setting up, Jenkins server dashboard should look like this



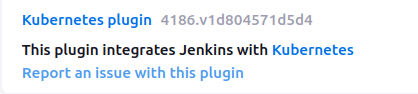
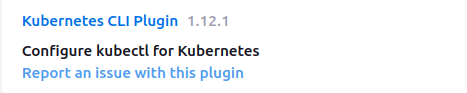
7. To install necessary plugins to work with, select manage Jenkins in dashboard page and go to plugins option under system configuration group.



8. Select updates if any plugin needs to be updated to the latest version, available plugins to install require plugins, installed plugins to view previously installed plugins, advanced plugins deals with unavailable plugins via downloading plugins and adding to the Jenkins server.

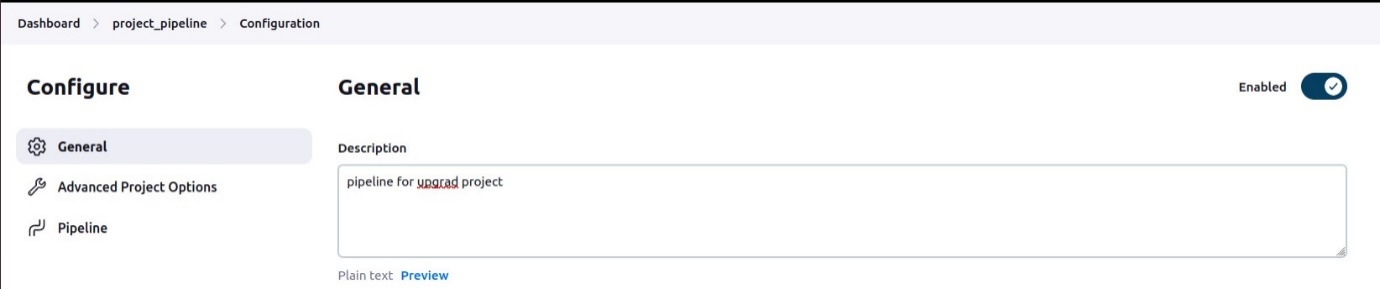


9. In this case, docker and Kubernetes plugins are needed to run the desired Jenkins pipeline. So, go to available plugins and search for docker plugin and Kubernetes plugin and install.

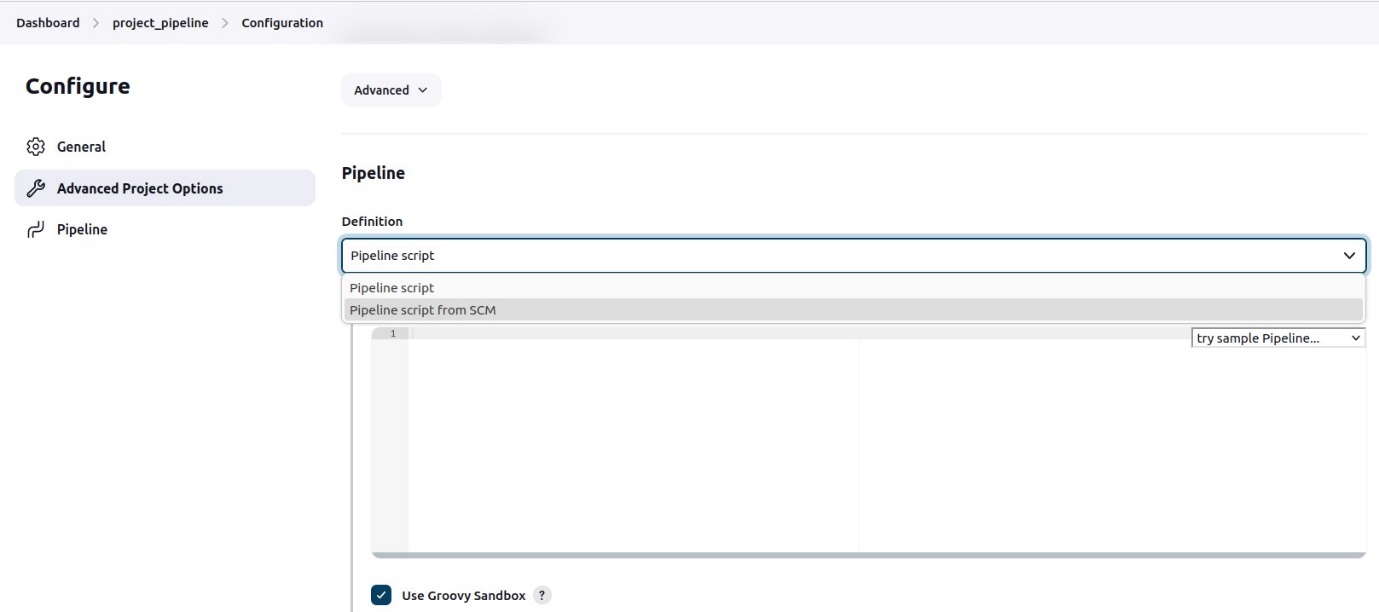


10. After installing plugins, go to the dashboard and select new item. Enter pipeline name and select pipeline option to start new pipeline. 

11. A configuration page will be opened after creating the pipeline and enter a description for pipeline under general tab description field.



12. Under pipeline section of advanced project options, select kind of definition “pipeline script” to write the Jenkins pipeline or “pipeline script from SCM” to draw Jenkinsfile from SCM like Github. For reference purposes, select the dropdown options to write custom pipeline script.



13. Pipeline script for the Jenkins pipeline is provided below: <https://github.com/SIDDHUGITTY/upgrad_project/blob/master/project/Jenkinsfile>

14. A Jenkins pipeline executes in a stage wise manner where each stage success depends on previous stage’s successful execution. The status of each stage can be viewed individually or collectively after completion of the build.

15. The first stage in the pipeline referred as “SCM checkout” draws the required web application source code from Github repository and deployment definitions as well.

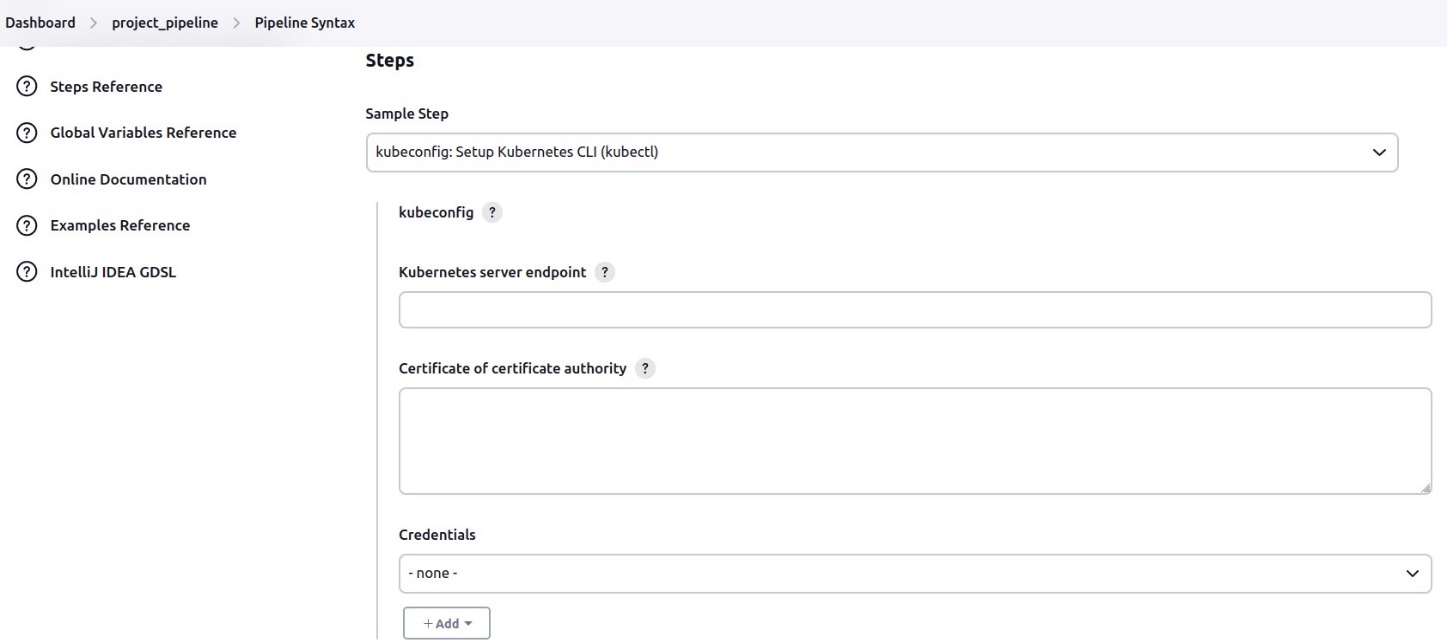
16. The second stage in the pipeline referred as “Test” is responsible to check the integrity of source code of web application and to present any errors as per the case.

17. The third stage in the pipeline referred as “Build” deals with building docker image and containerization of the built docker image.

18. The fourth stage in the pipeline referred as “Deploy” manages the deployment with assigned Kubernetes credentials such as Kubernetes server address, certificates like client key, client certificate, server CA certificate which are located in the cluster.

19. Run command “kubectl config view” to view the configuration details such as cluster name, paths of various certificates, Kubernetes server endpoint, version etc…

20. Select pipeline syntax option under pipeline section and choose sample step “kubeconfig: setup Kubernetes CLI (kubectl)”, enter Kubernetes server endpoint as in the kube config file located at ”.kube/config”.

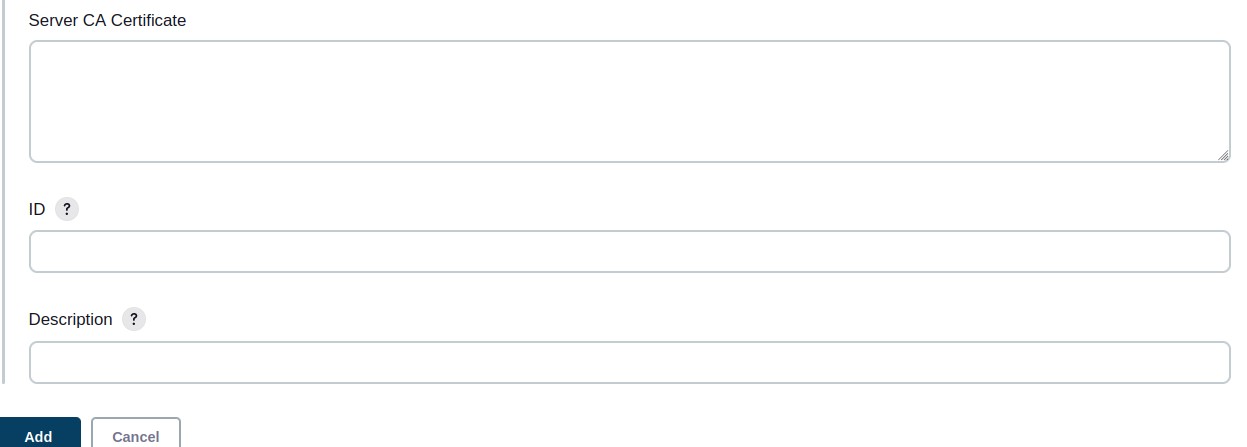


21. In order to not to disclose credentials in the pipeline script, certificates will be wrapped into an ID name and only the name will be displayed.

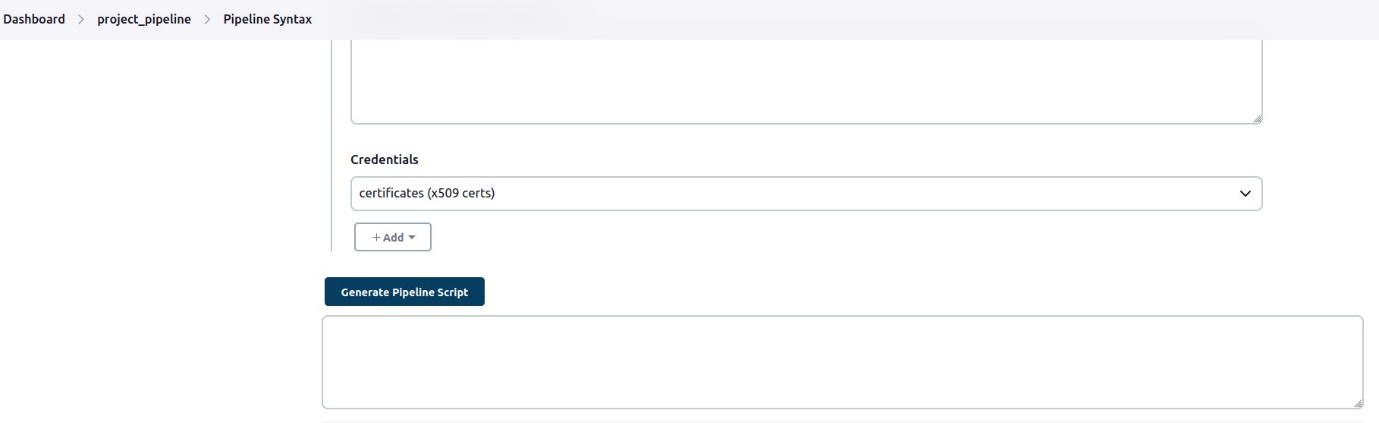
22. Select Jenkins in the drop-down menu under credentials group and provide necessary certificate details.



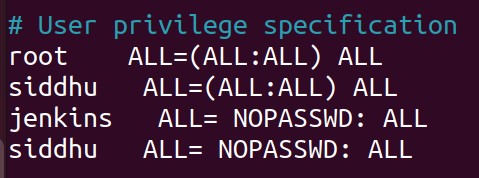
23. Select “X.509 client certificate” under kind and provide client key located at “.minikube/profiles/minikube/client.key”, client certificate located at “.minikube/profiles/minikube/client.crt”, server CA certificate located at “.minikube/ca.crt”, ID, description and click add.



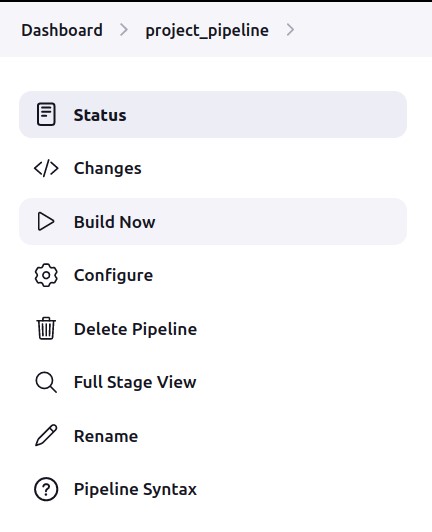
24. After adding credentials, again click on drop-down menu under credentials and select the saved credentials and click on “generate pipeline script”



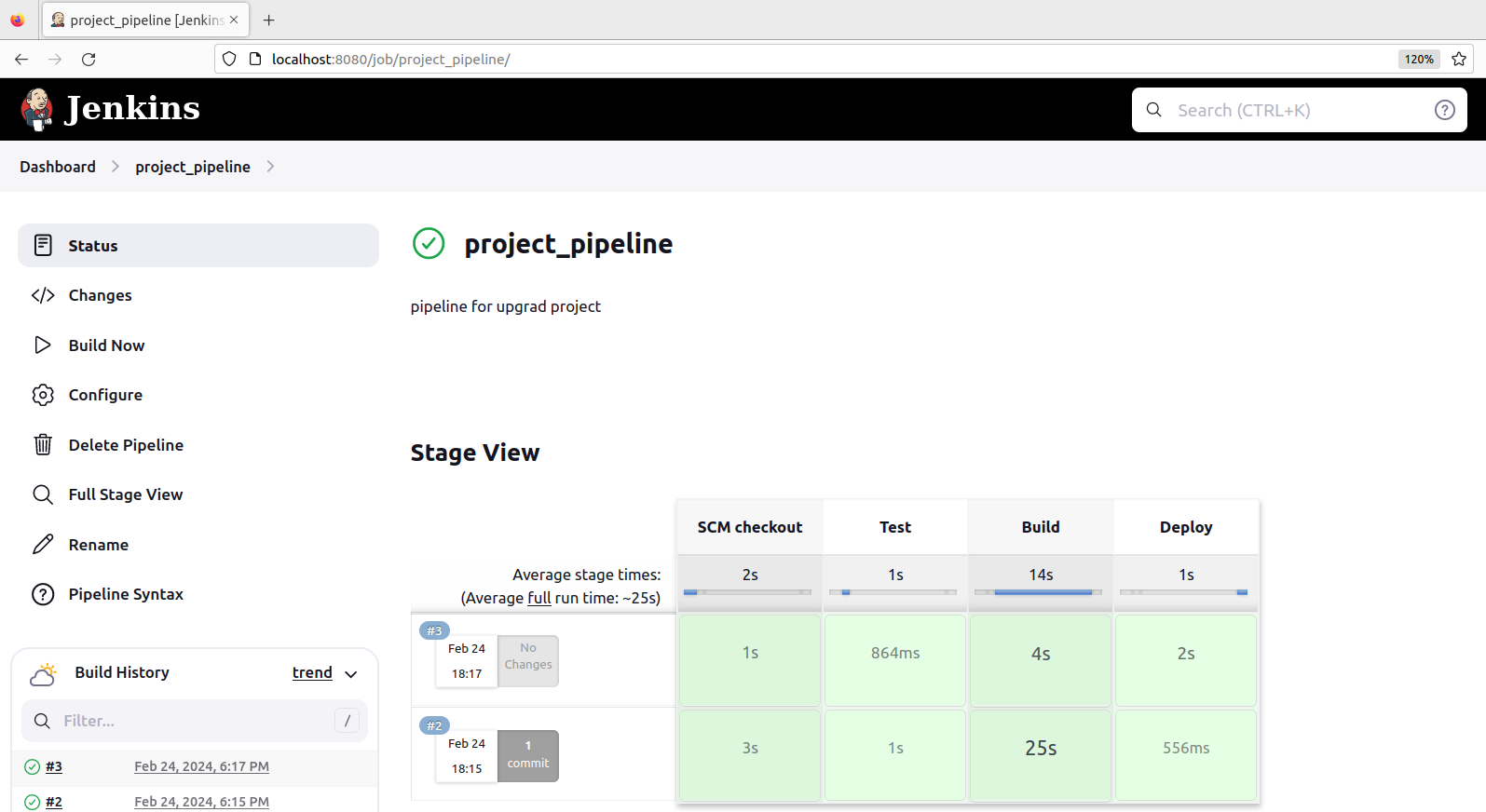
25. Before building the pipeline, add Jenkins to the sudoers list to avoid raising any authentication issues while executing the pipeline. In order to do so, run command “sudo nano /etc/sudoers” and enter the following under “user privilege specification”



26. Now, click on “Build Now” option on left side pane to build the Jenkins pipeline



27. After successfully building the Jenkins pipeline, the stage view should look similar like this:



STEP 6: MONITORING AND LOGGING

1. Prometheus is an open-source technology designed to provide monitoring and alerting functionality for cloud-native environments including Kubernetes. It can collect and store metrics as time-series data, recording information with timestamps.

2. Prometheus focuses on data acquisition and allows users select and aggregate in real time. The collected date will be utilised by Grafana for visualization purposes.

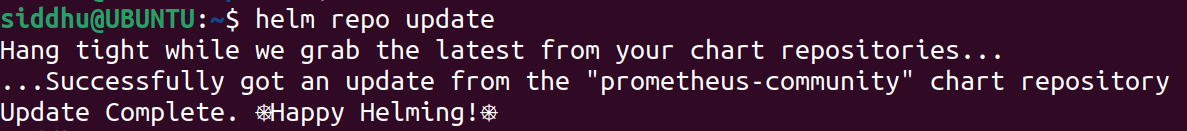
3. Firstly, install helm so that Prometheus can be installed with all the necessary packages. Helm can be installed from apt repository from official documentation page.



4. Add Prometheus repository from helm using the following command



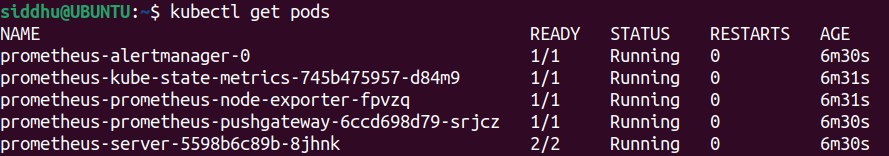
5. Update the existing helm repositories by performing “helm repo update” command.

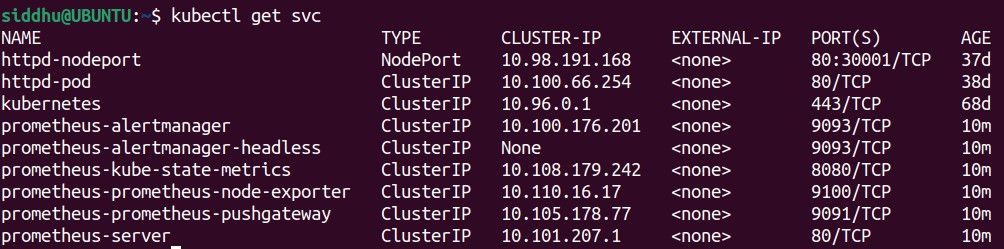


6. Install Prometheus from the following command

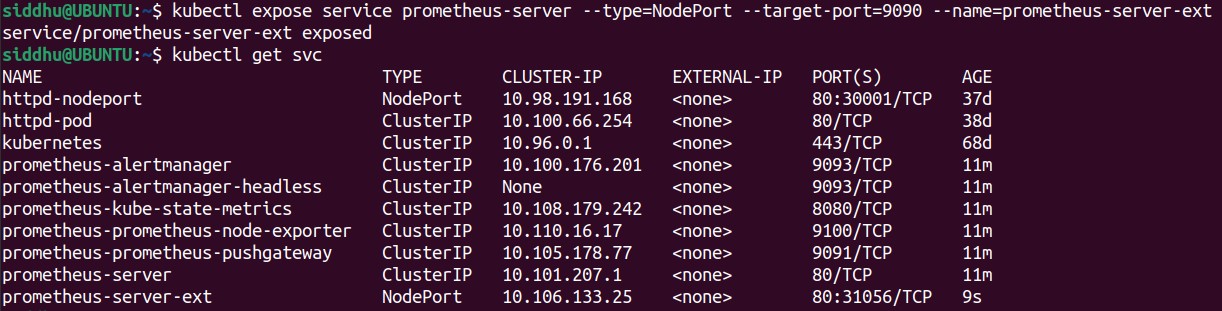


7. After installing Prometheus, pods and services will be up and running related to Prometheus controllers.

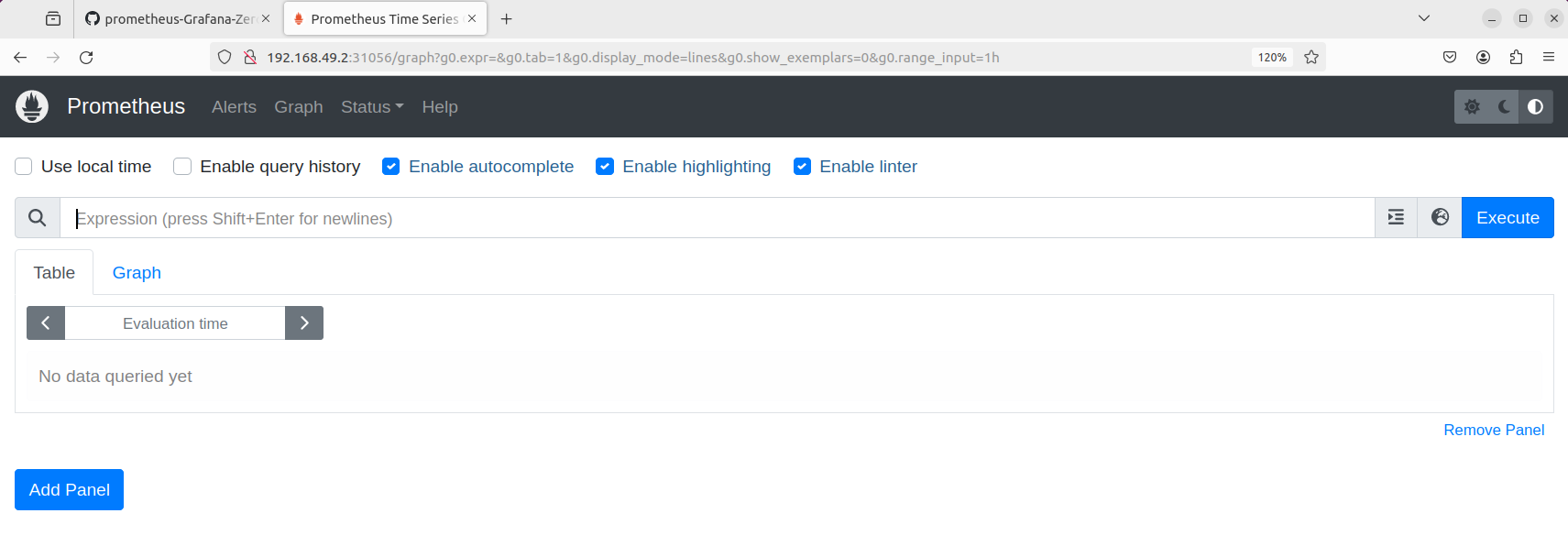




8. Now, expose the Prometheus-server service with nodeport and target port of 9090.



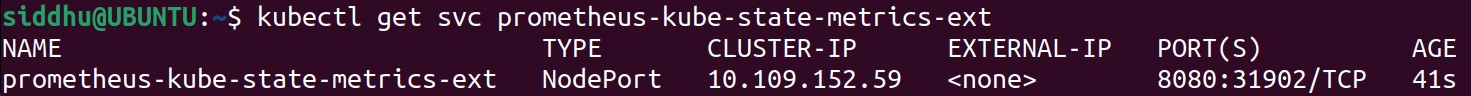
9. Newly created service with exposed port (Prometheus-server-ext) will be running along with minikube IP i.e., 192.168.49.2:31056 in this case.



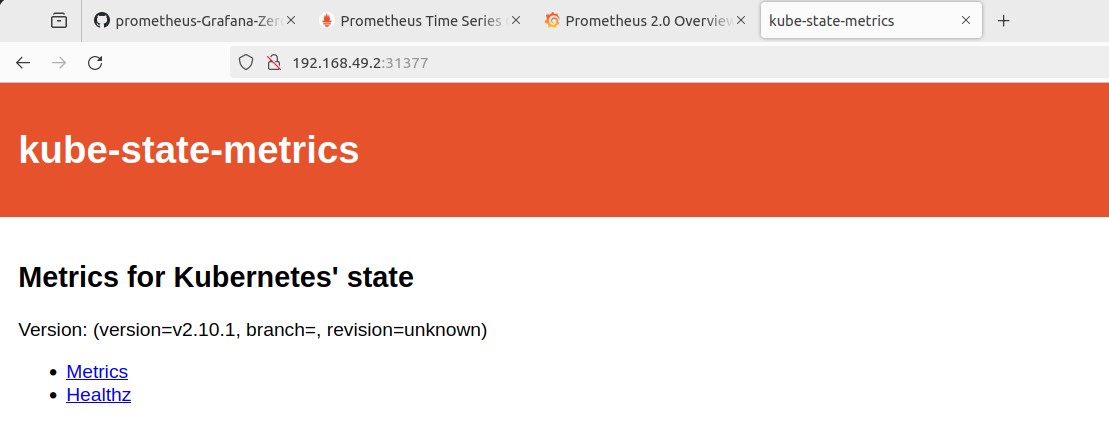
10. Here, expressions are determined by the metrics of the Kubernetes state. So, kube-state-metrics should also be exposed to access the expressions which can be used for executions.

11. kube-state-metrics will be exposed same as the Prometheus-server as follows

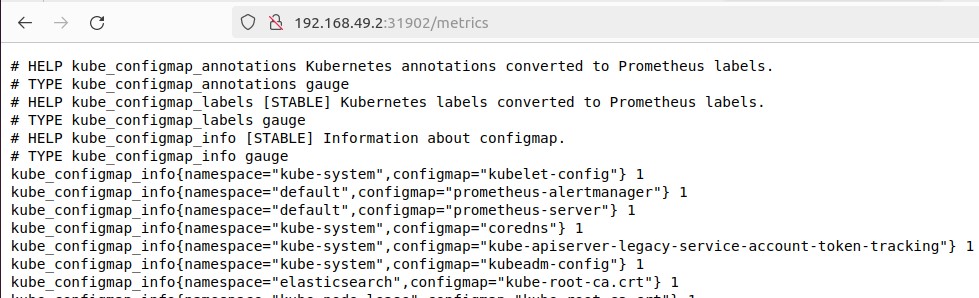




12. Newly exposed service will be running along with minikube i.e., 192.168.49.2:31377



13. select the “metrics” option to access the expressions where those expressions will be utilised to build the necessary Kubernetes objects and then can be used to feed that data to Grafana for visualisation.

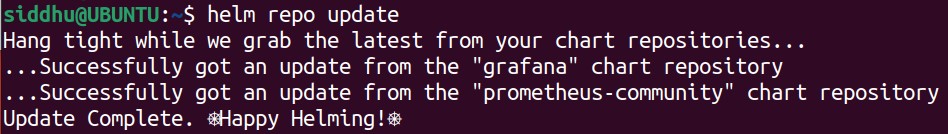


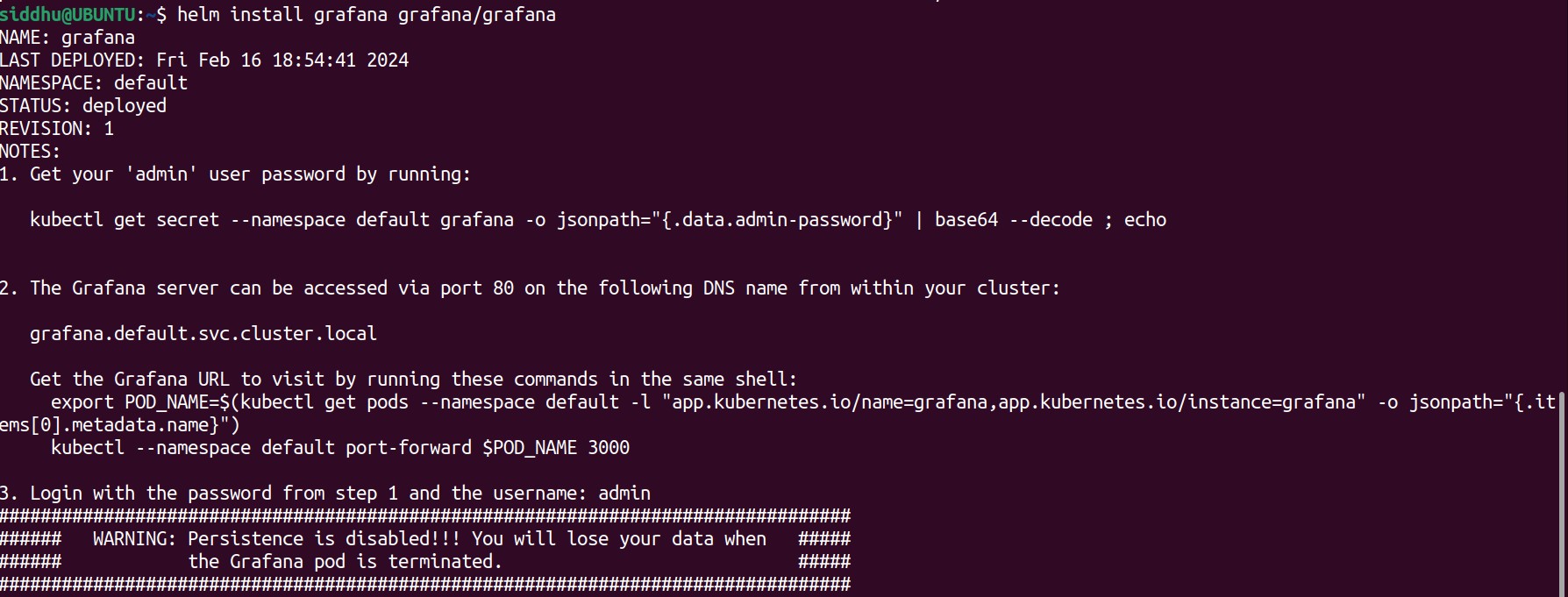
14. Grafana is an open-source software which enables users to query, visualise, alert and explore metrics, logs and traces. Grafana tools can turn time-series database into insightful graphs and visualization.

15. Add Grafana repository from helm using the following command

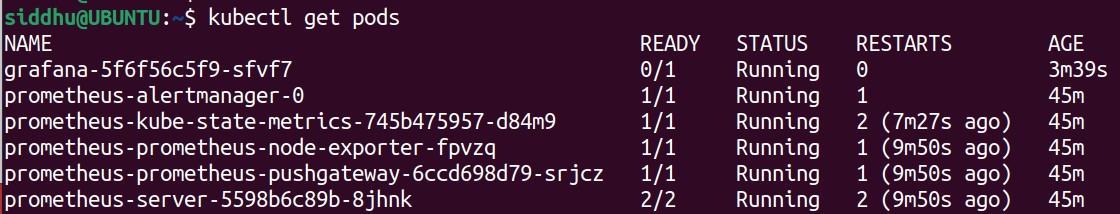


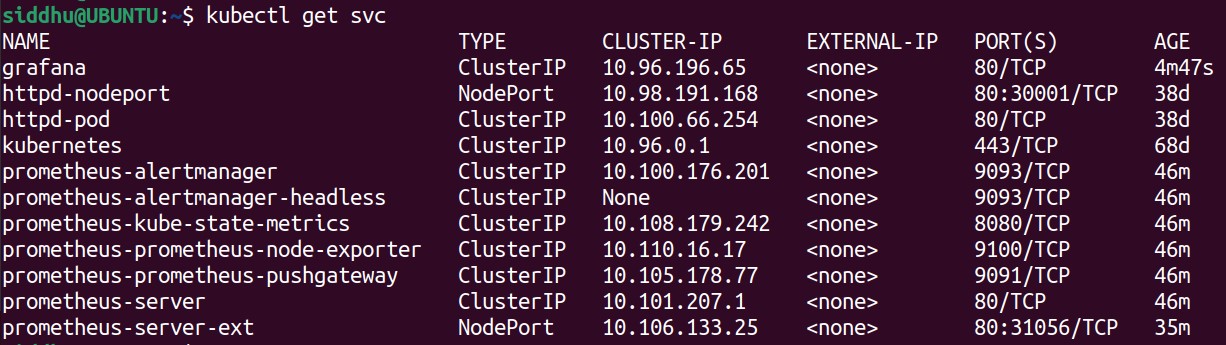
16. Update helm repository and install Grafana as follows



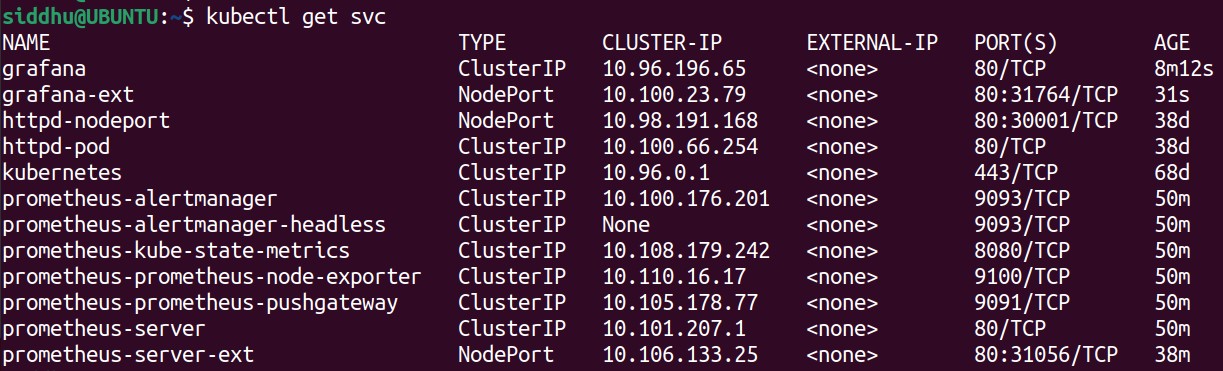


17. After installing Grafana, pods and services will be up and running related to Grafana controllers.

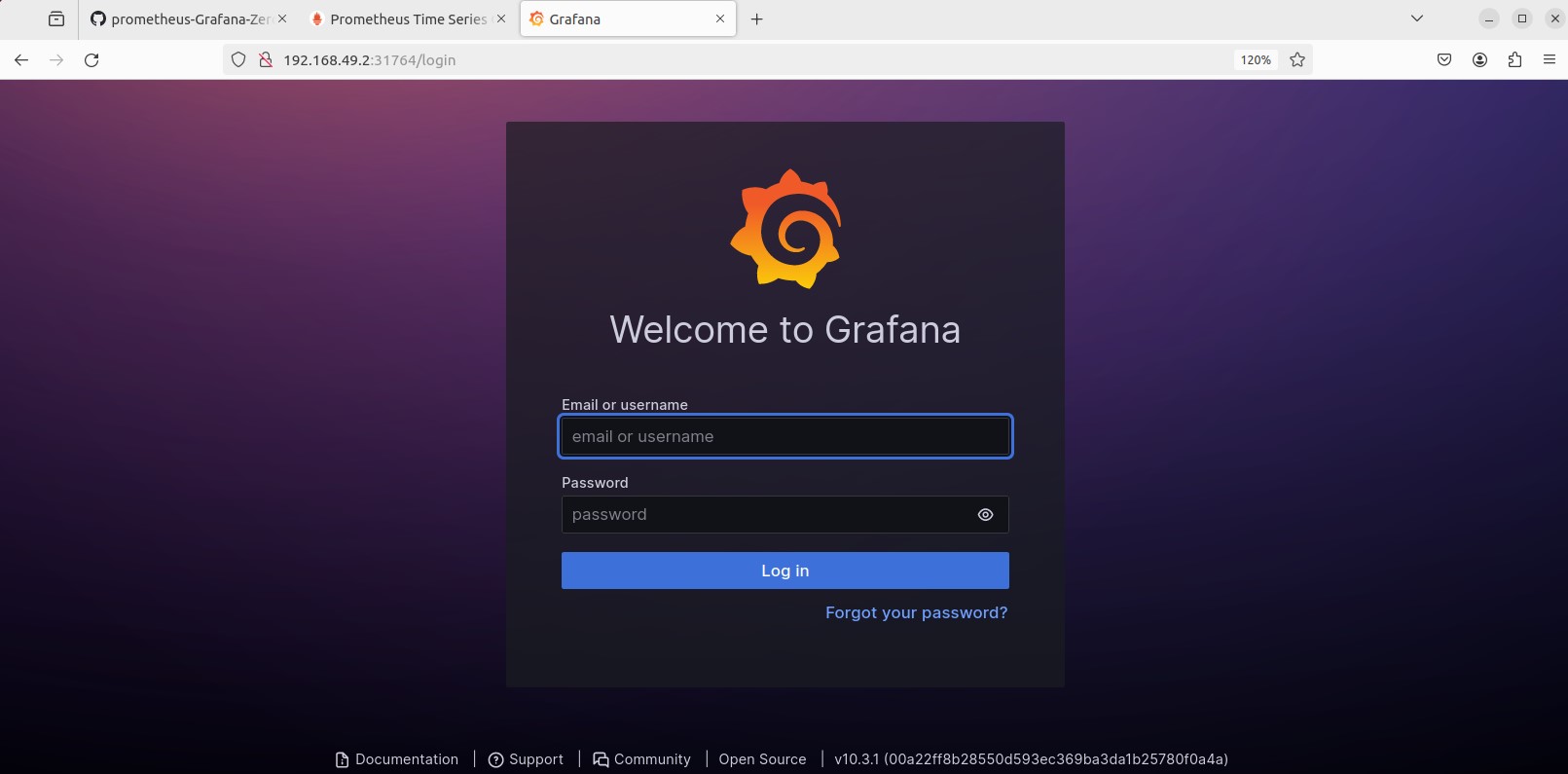




18. Now, expose the Grafana service with nodeport and target port of 3000.

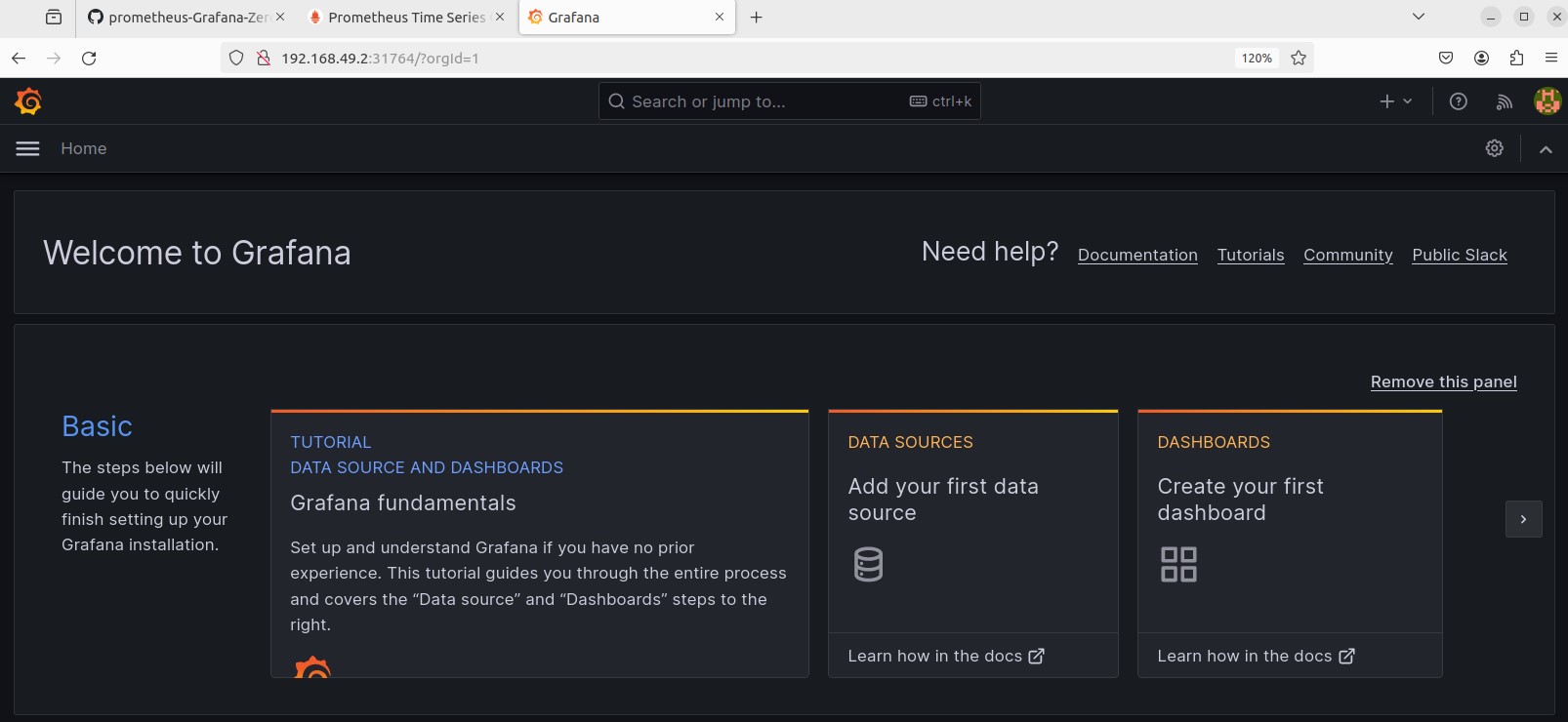
19. Newly created service with exposed port (grafana-ext) will be running along with minikube IP i.e., 192.168.49.2:31764 in this case.



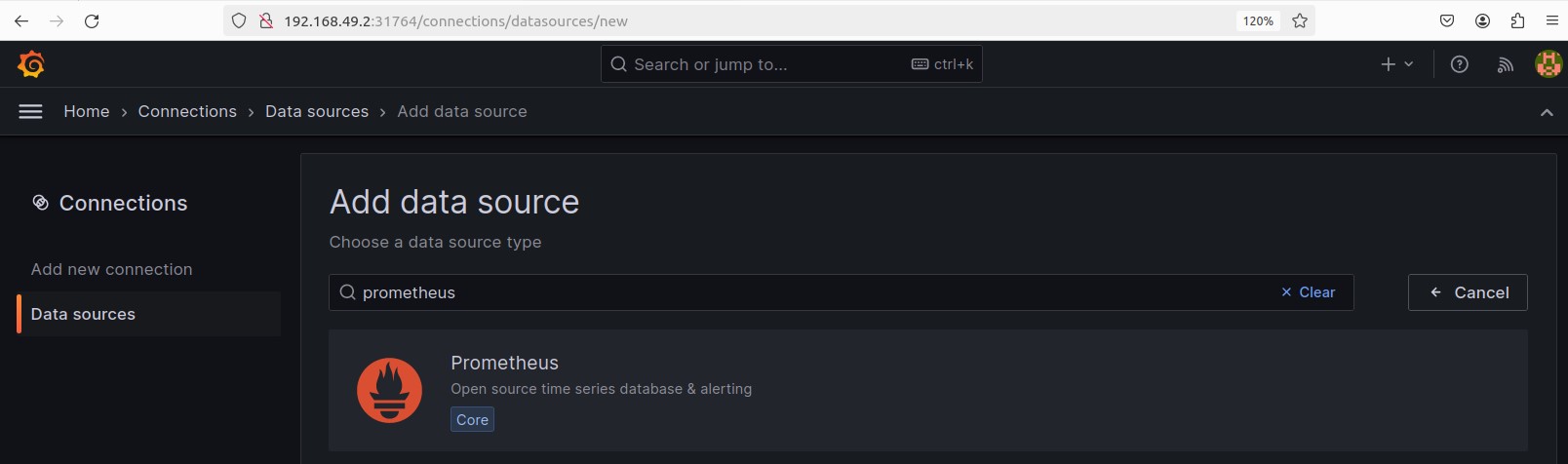
20. Grafana password can be obtained from initial installation verbose where default username is going to be “admin” and password can be viewed by running the following command



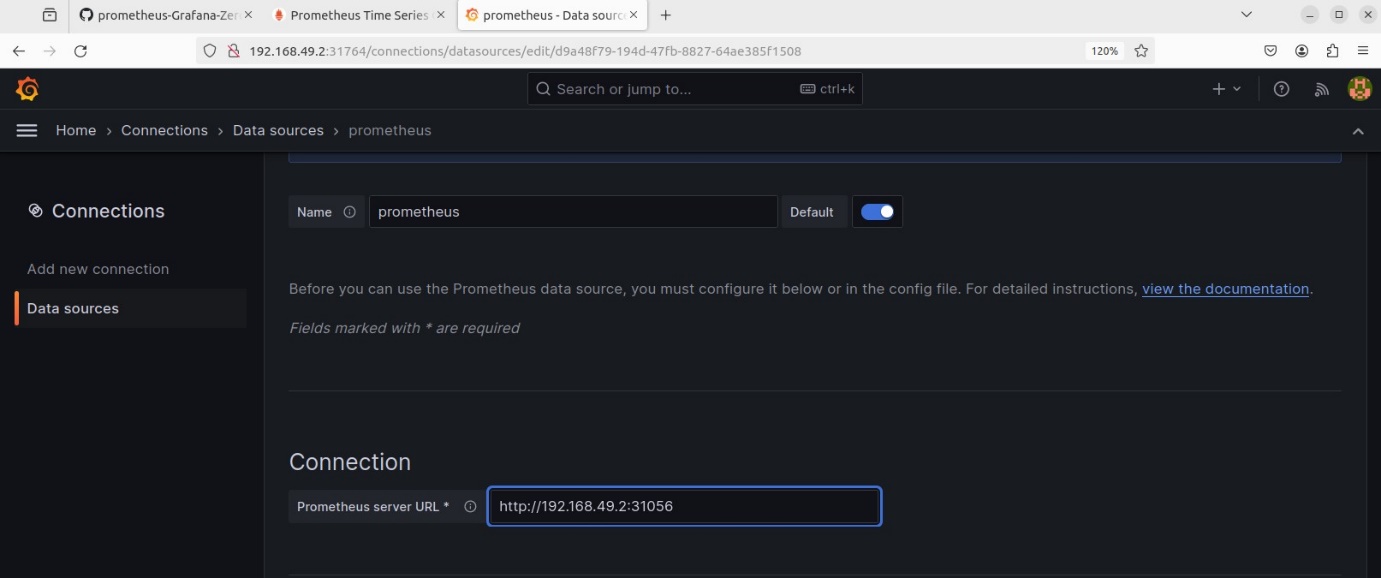
21. Enter into Grafana home page by providing log in credentials.



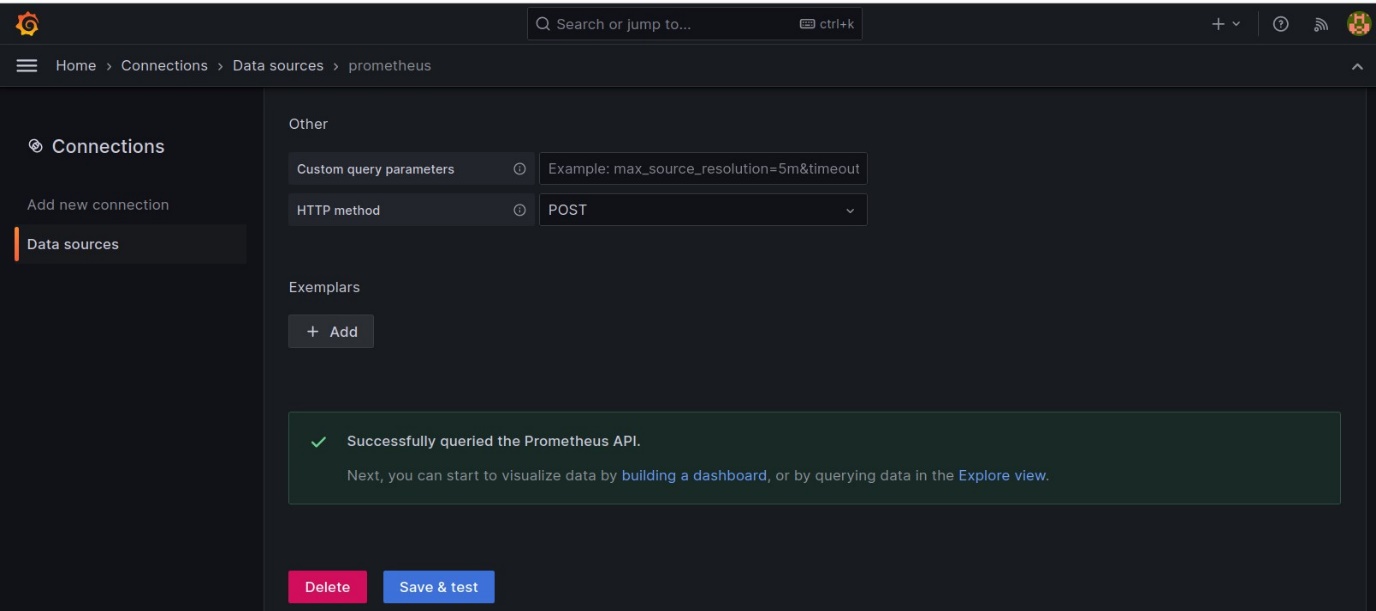
22. Select data sources option to assign data source where Grafana can obtain its inputs from monitoring systems like Prometheus. Here, search and select Prometheus as data source.



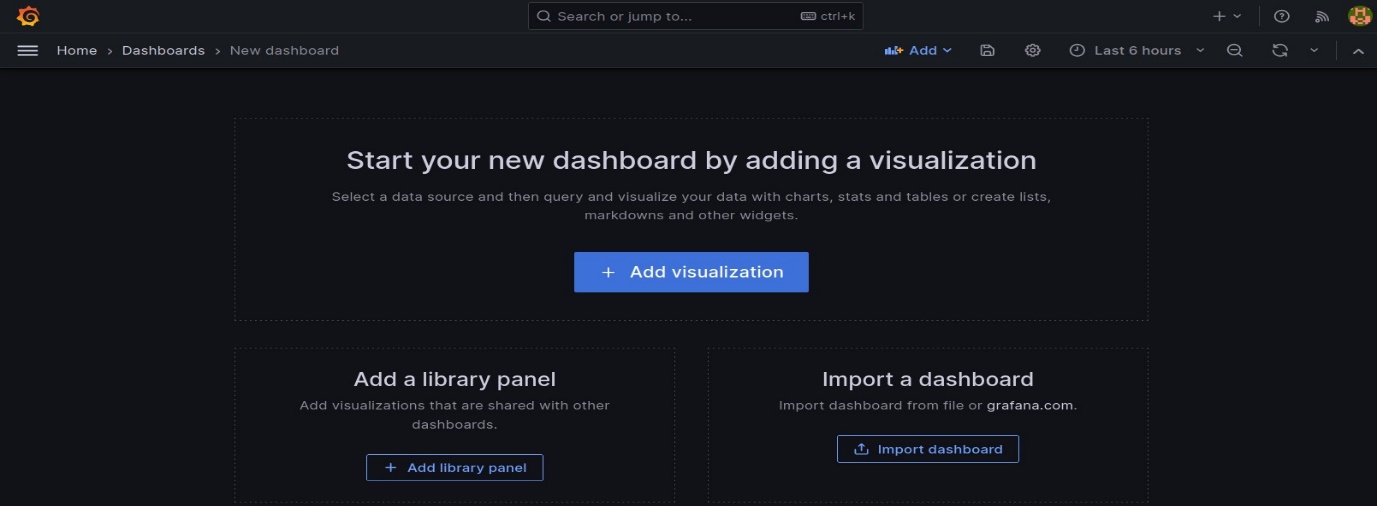
23. An optional name can be provided on name field and Prometheus server URL should be assigned to create connection between Prometheus and Grafana for monitoring data acquisition.



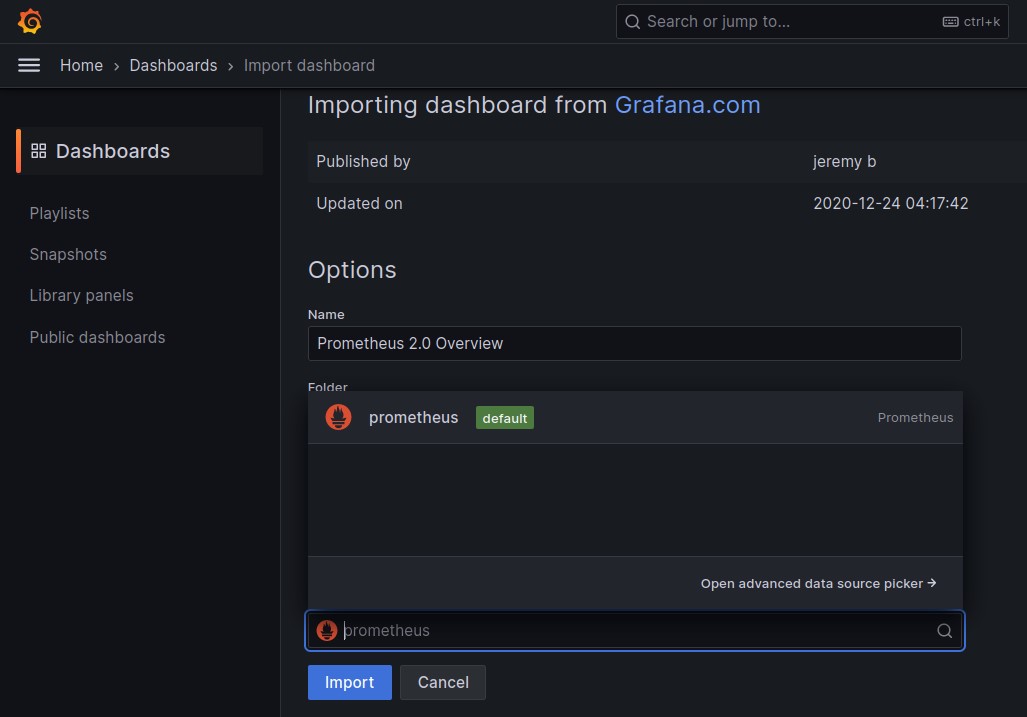
24. Click on “save & test” to check the connection integrity between Prometheus and Grafana.



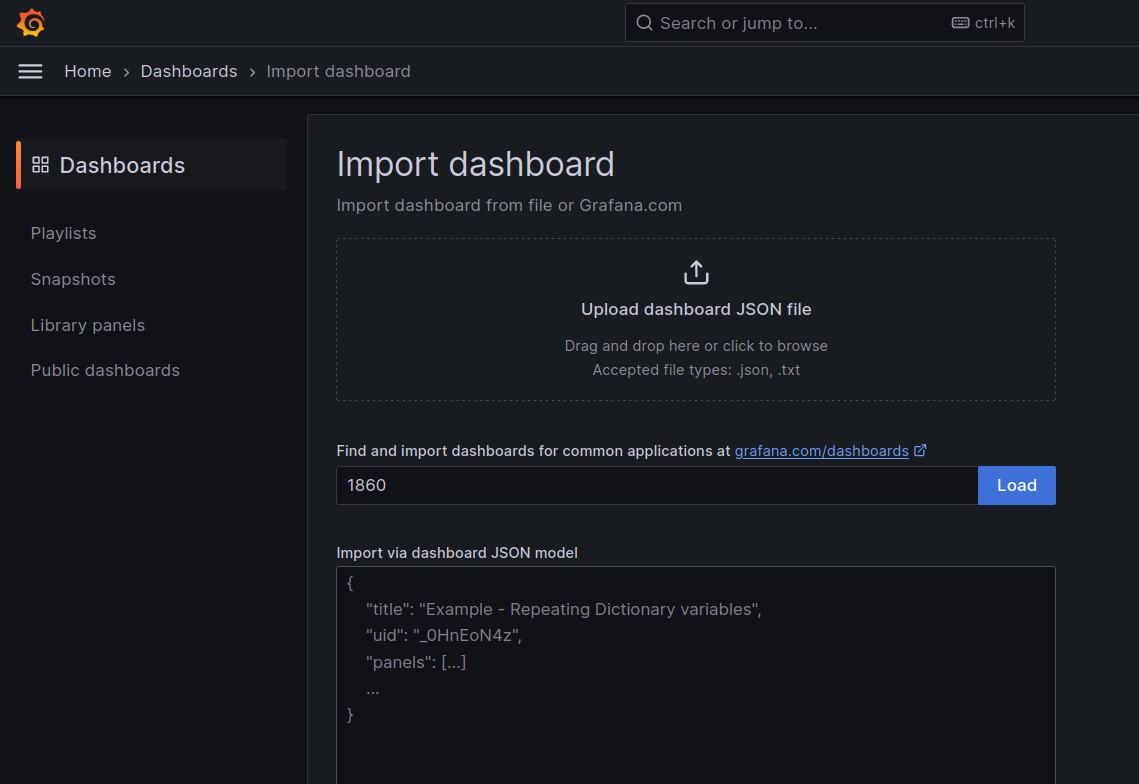
25. After establishing connection with data source, Go to Grafana home page and select dashboards option to access visualization options.



26. Select “import dashboard” and choose Prometheus as default data source.



27. Grafana dashboards can be imported from the provided link or dashboard ID alone can be provided. Here, an ID is provided.



28. After loading dashboard ID, a visual representation of metrics data is shown where it contains data regarding resource usage, status of deployments, services, running jobs if any and many more. User can customize visualization objects as per requirements to eliminate ambiguity.

