Implementing End to End CI/CD Pipeline

Project Overview:

In this project, We are implementing End to End CI/CD Pipeline using tools like:

GitHub: For storing the source code of the application

Jenkins: To implement the Continuous Integration

SonarQube: For testing the source code and Static Analysis

Maven: To build the source code of the application

Docker: To containerization the application and push it to the DockerHub

Minikube: To create Kubernetes Cluster

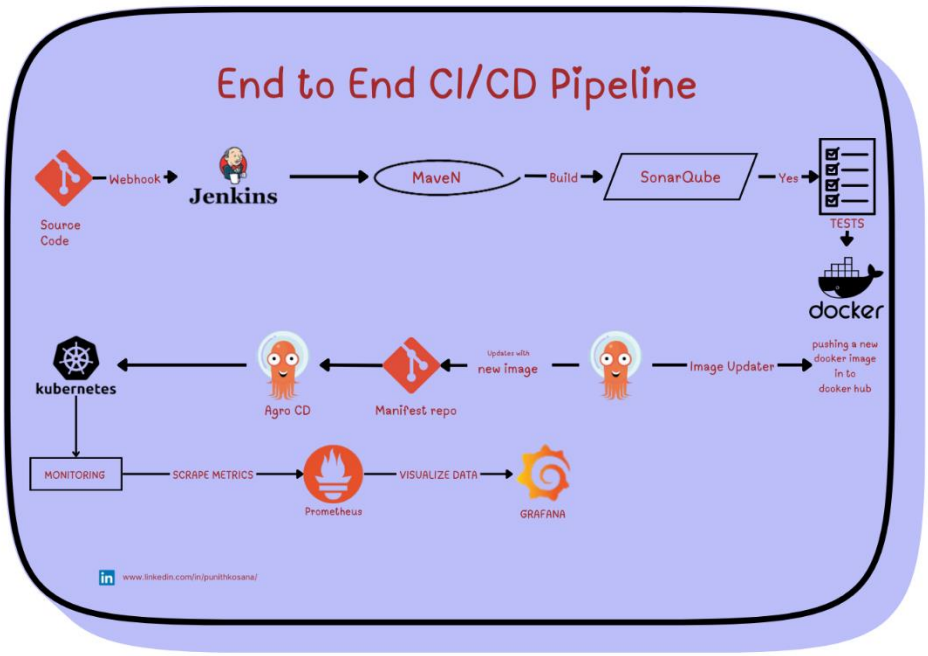
ArgoCD: To implement Continuous Delivery

Prometheus: To scrape the metrics from the cluster

Grafana: To visualize the metrics and create dashboards

GitHub Link: https://github.com/PunithKosana/CICD-Pipeline.git

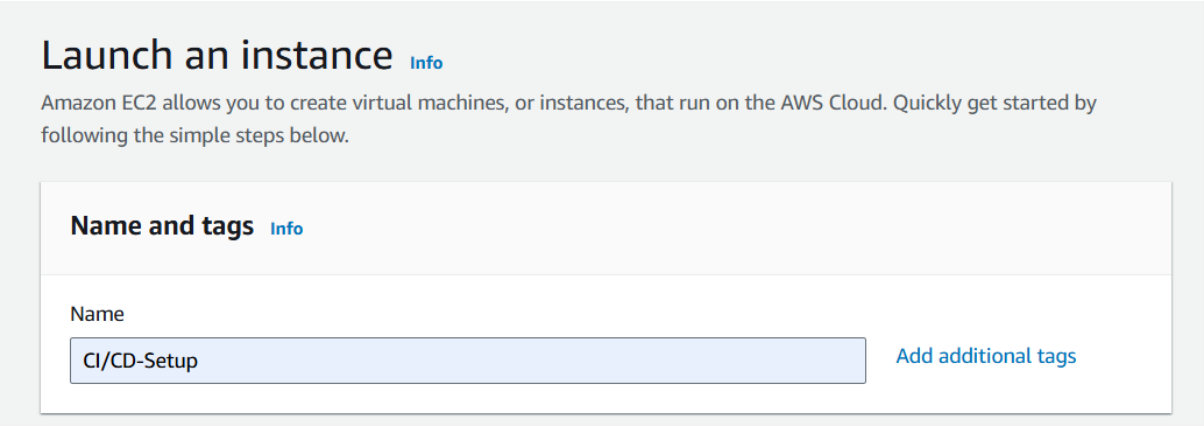
Workflow:



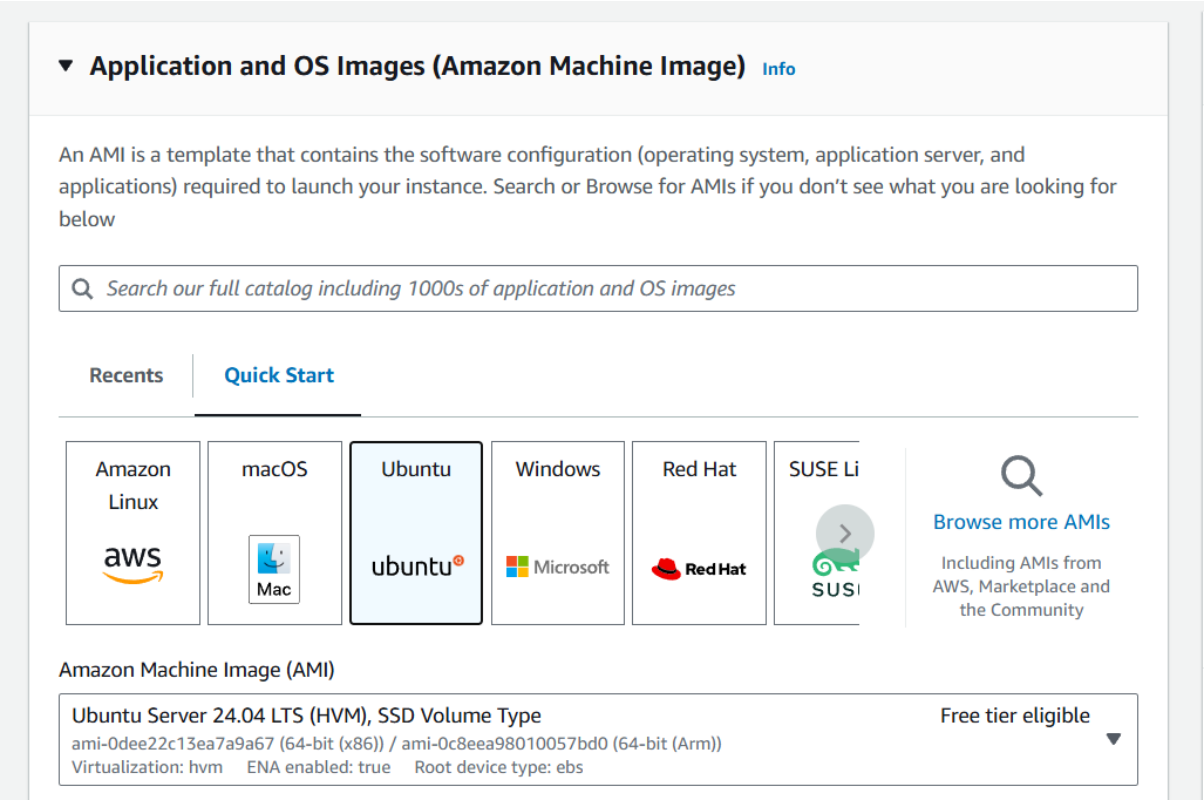
Steps:

EC2 Setup:

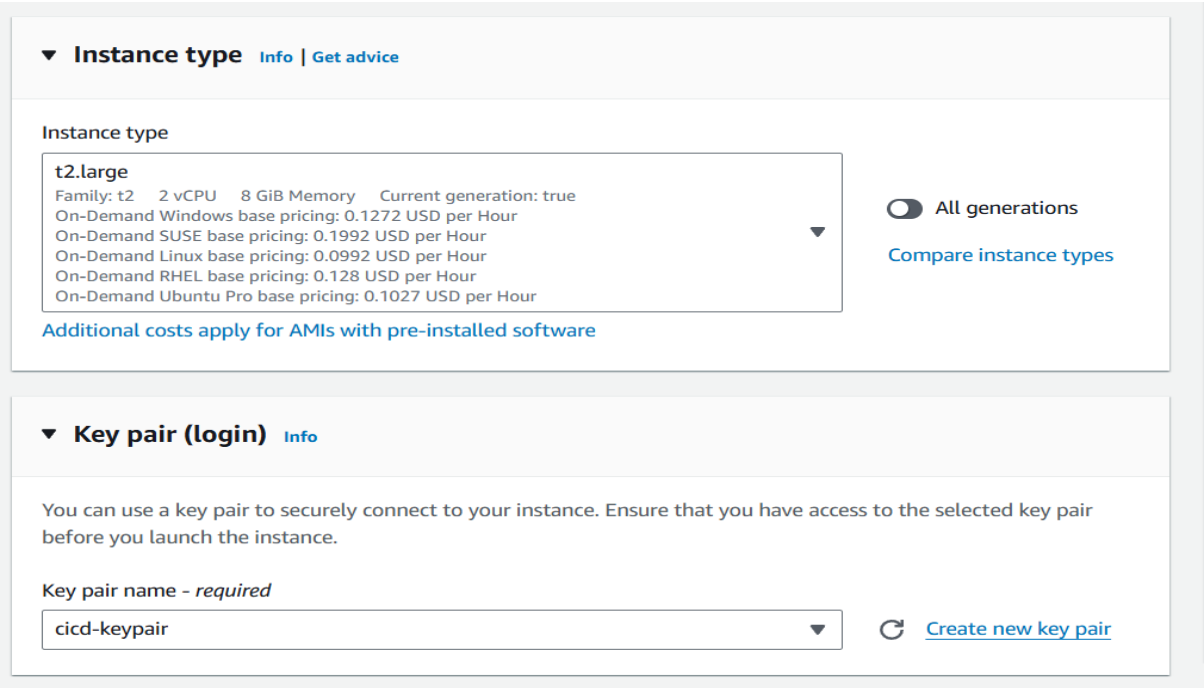
Go to AWS, Launch an EC2 and Name it



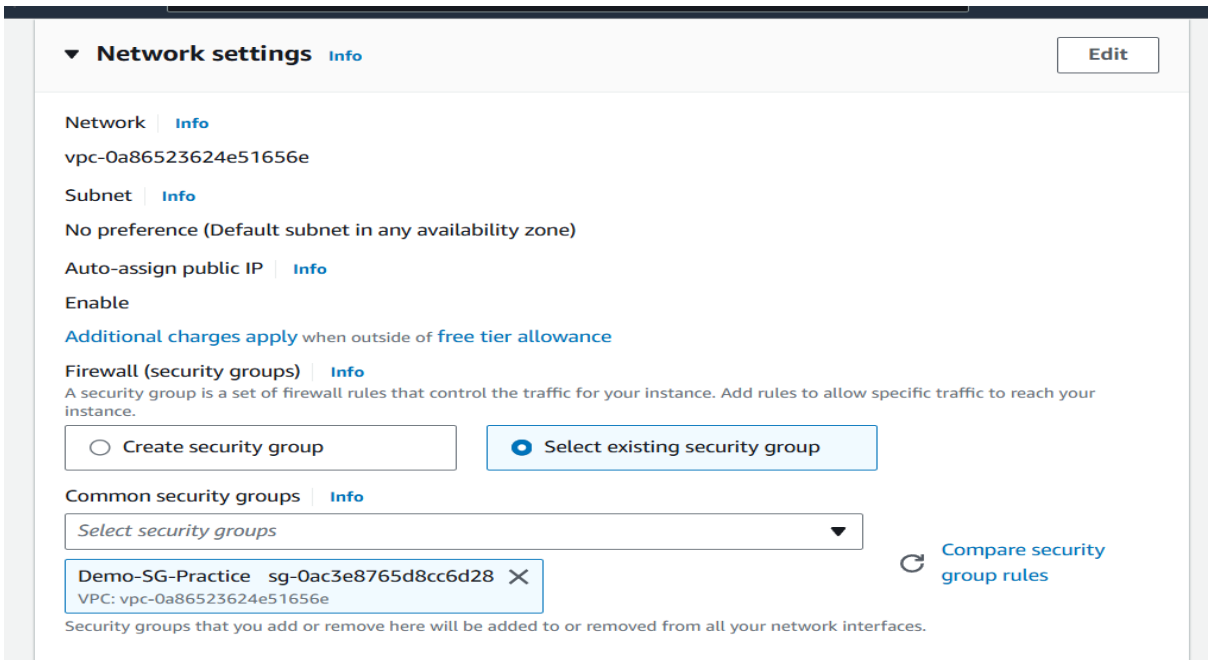
Select the AMI image as Ubuntu



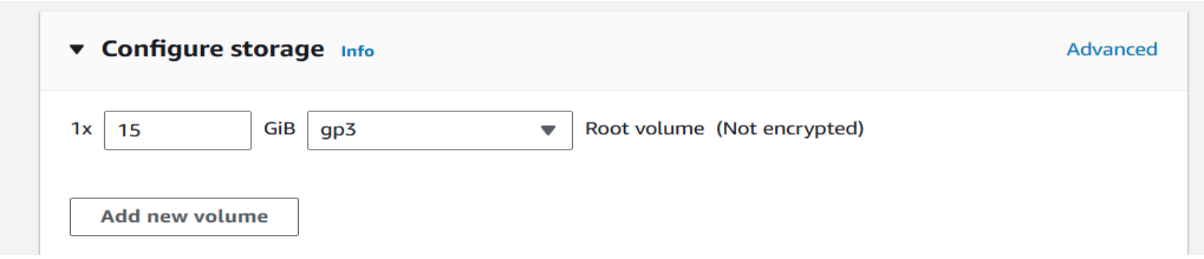
Select the instance type and create a key-pair



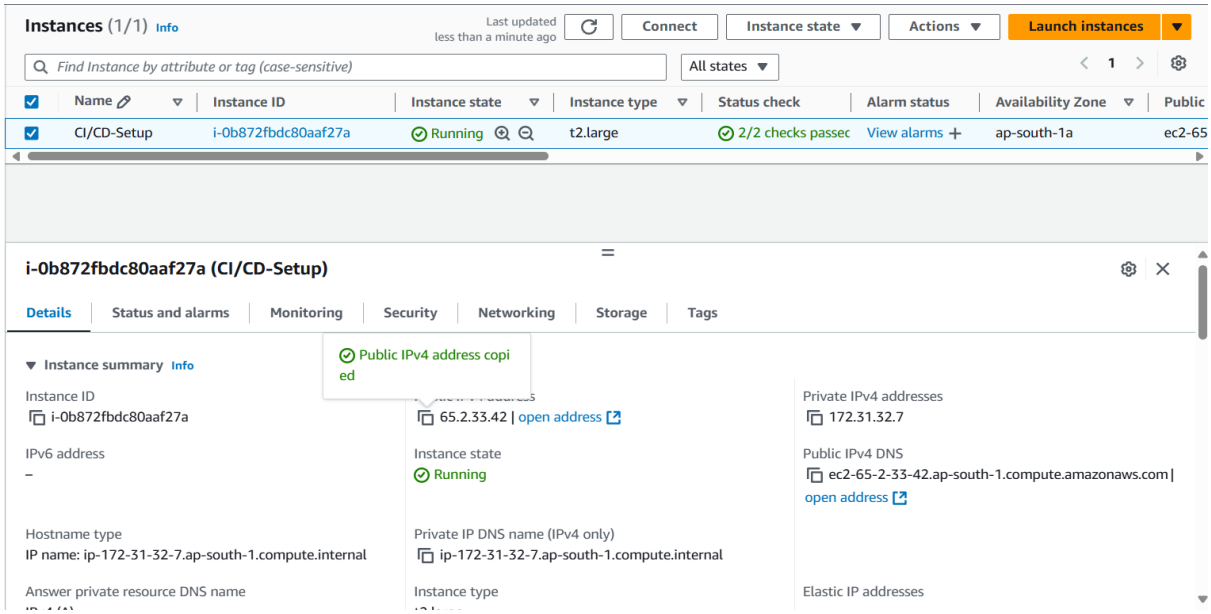
Create or Select the existing security group. Make sure to allow all traffic on inbound rules



Configure storage and launch the instance



EC2 is running successfully and copy the Public_IPAddress



Connect to local system via ssh

```
C:\Users\Punith\Downloads>ssh -i cicd-keypair.pem ubuntu@65.2.33.42
```

Check the connection and username

```
ubuntu@ip-172-31-32-7:~$ whoami
ubuntu
ubuntu@ip-172-31-32-7:~$
```

Configuring Continuous Integration and Implementation:

Install Java

```
ubuntu@ip-172-31-32-7:~$ sudo apt install openjdk-17-jre -y
```

Check the java version installed

```
ubuntu@ip-172-31-32-7:~$ java --version
openjdk 17.0.13 2024-10-15
OpenJDK Runtime Environment (build 17.0.13+11-Ubuntu-2ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.13+11-Ubuntu-2ubuntu124.04, mixed mode, sharing)
ubuntu@ip-172-31-32-7:~$
```

Install Jenkins from https://www.jenkins.io/doc/book/installing/linux/

```
ubuntu@ip-172-31-32-7:~$ curl -fsSL https://pkg.jenkins.io/debian/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 binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins -y |
```

Update the packages and install docker

```
ubuntu@ip-172-31-32-7:~$ sudo apt update
sudo apt install docker.io -y
```

Go to root. Add ubuntu and Jenkins users to docker group to give full permissions to access from either of the user

```
root@ip-172-31-32-7:~# usermod -aG docker ubuntu
root@ip-172-31-32-7:~# usermod -aG docker jenkins
```

Restart the docker service

```
root@ip-172-31-32-7:~# systemctl restart docker
```

Install unzip packages

```
root@ip-172-31-32-7:~# apt install unzip
```

Create a user for sonarqube

```
root@ip-172-31-32-7:~# adduser sonarqube
```

Switch to the sonarqube user

```
root@ip-172-31-32-7:~# su - sonarqube
```

Download the sonarqube zip and unzip the packages downloaded

```
sonarqube@ip-172-31-32-7:~$ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.4.0.54424.zip
unzip *
```

List the packages

```
sonarqube@ip-172-31-32-7:~$ ls
sonarqube-9.4.0.54424  sonarqube-9.4.0.54424.zip
sonarqube@ip-172-31-32-7:~$
```

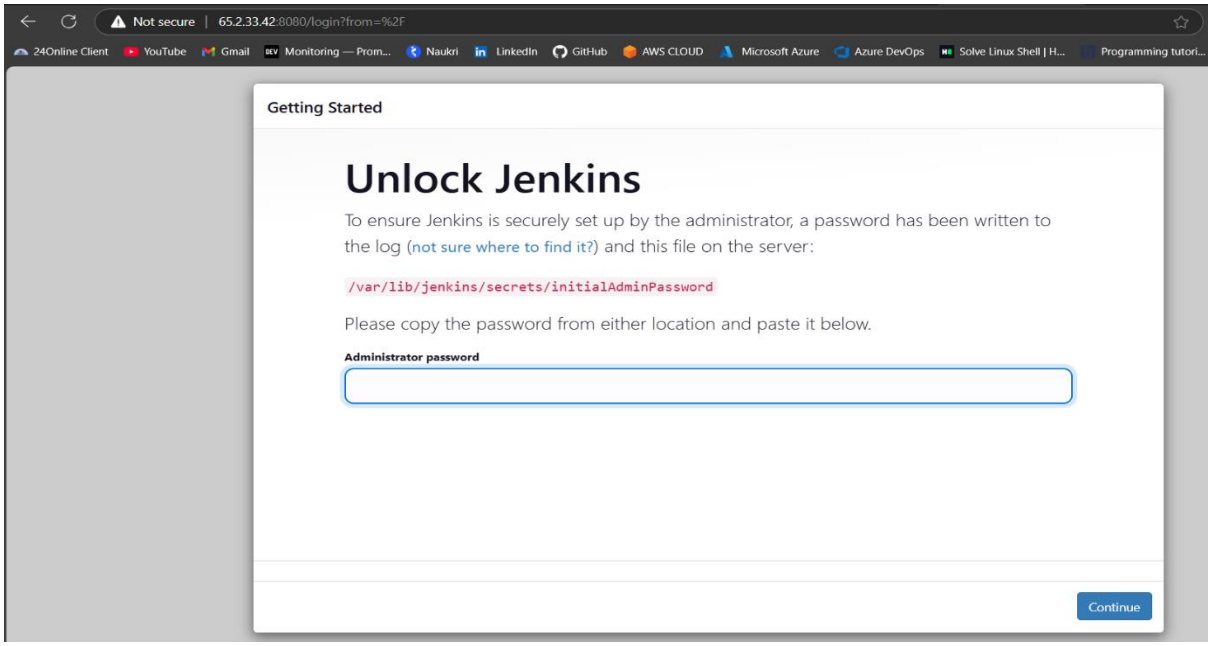
Change directory path to the linux distro directory

```
sonarqube@ip-172-31-32-7:~$ cd sonarqube-9.4.0.54424/bin/linux-x86-64/
```

Start the Sonarqube

```
sonarqube@ip-172-31-32-7:~/sonarqube-9.4.0.54424/bin/linux-x86-64$ ls
lib  sonar.sh  wrapper
sonarqube@ip-172-31-32-7:~/sonarqube-9.4.0.54424/bin/linux-x86-64$ ./sonar.sh start
Starting SonarQube...
Started SonarQube.
sonarqube@ip-172-31-32-7:~/sonarqube-9.4.0.54424/bin/linux-x86-64$
```

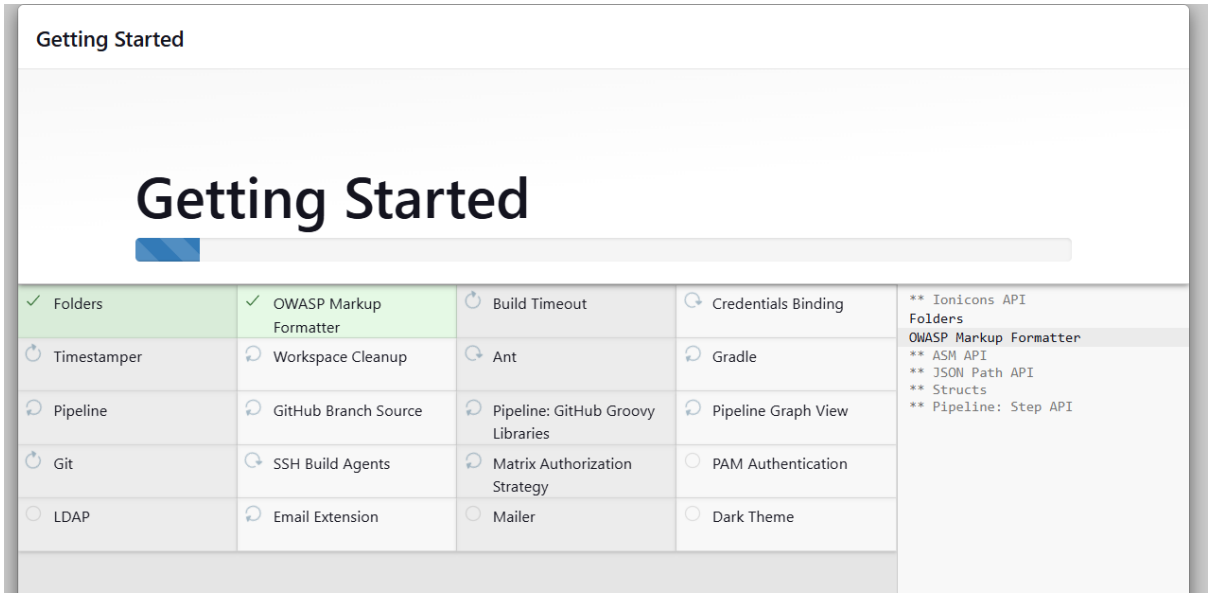
Access Jenkins by “Public_IPAddress:8080”.



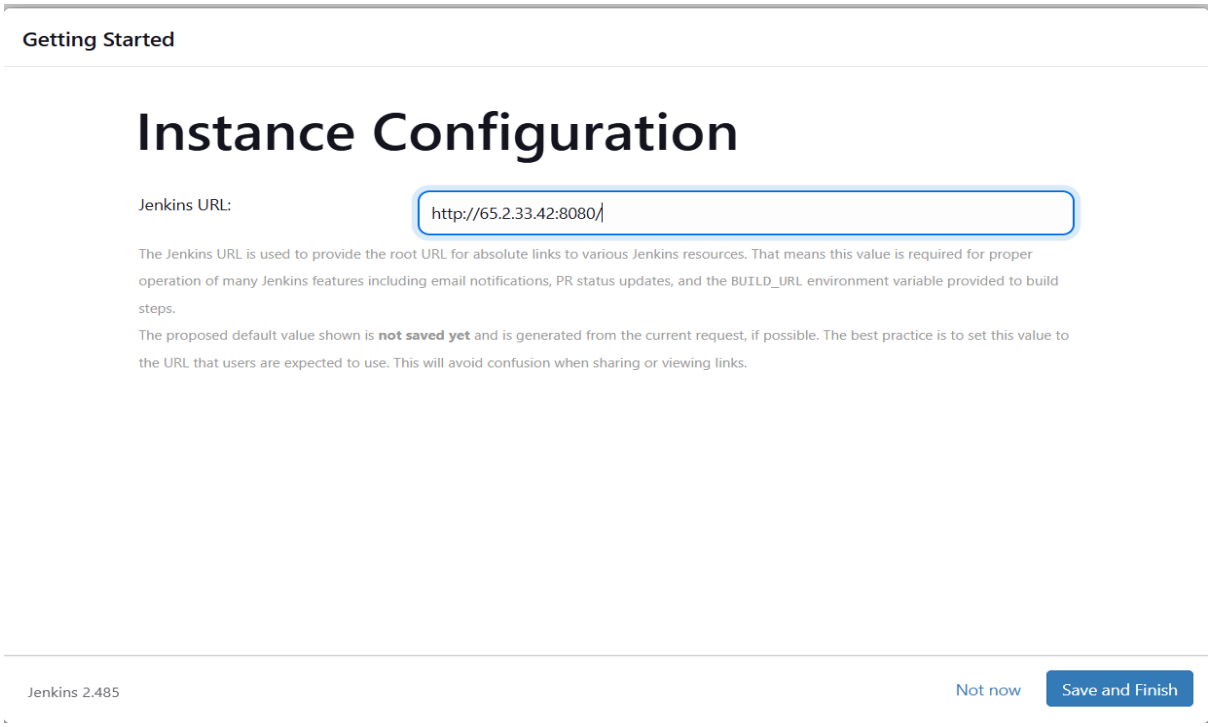
Get the admin_password

```
root@ip-172-31-32-7:~# cat /var/lib/jenkins/secrets/initialAdminPassword
a8dee1e12f254dda84dc7467b3564c06
root@ip-172-31-32-7:~#
```

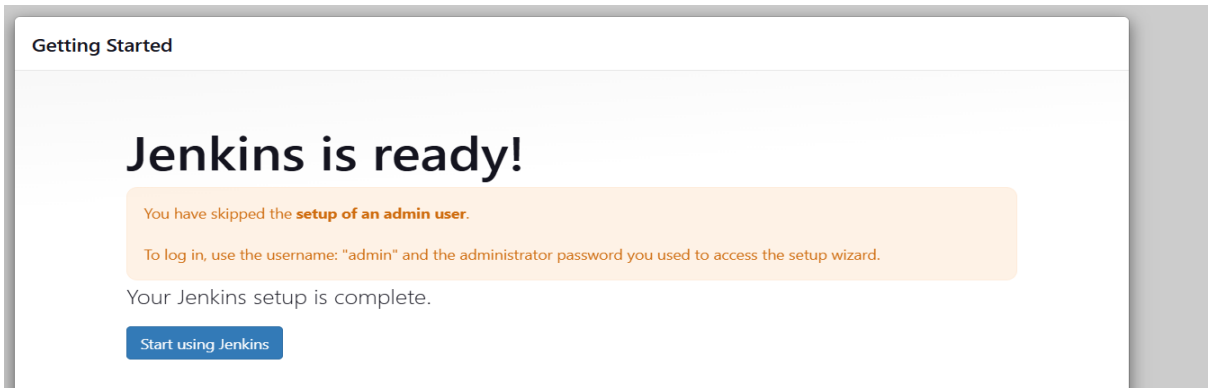
Install the suggested plugings and the plugins get pre-installed



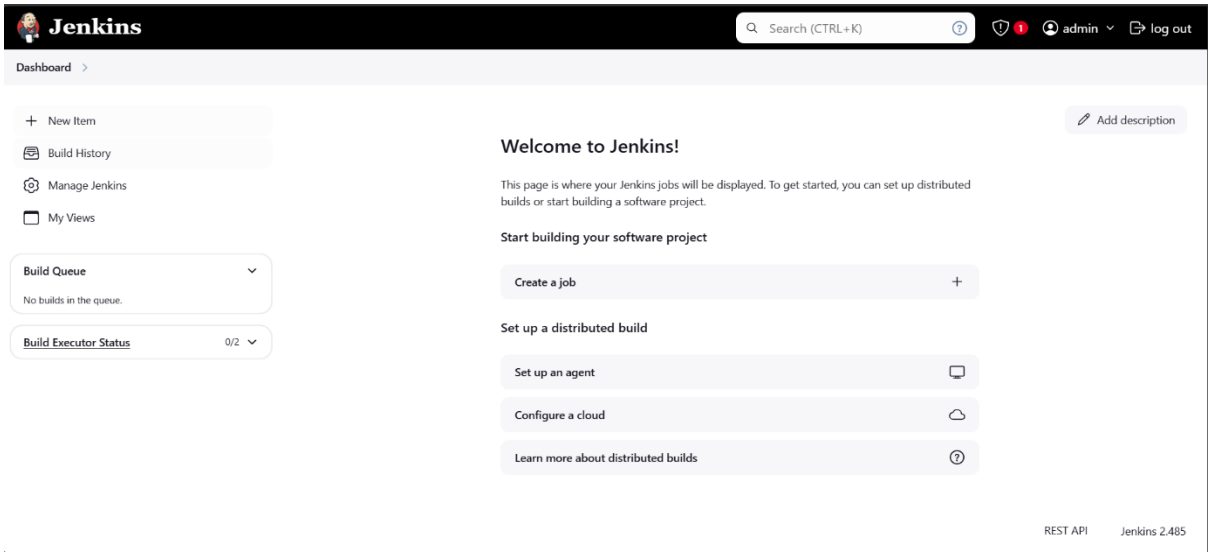
Configure the user details or continue as admin. Configure the Jenkins URL and save



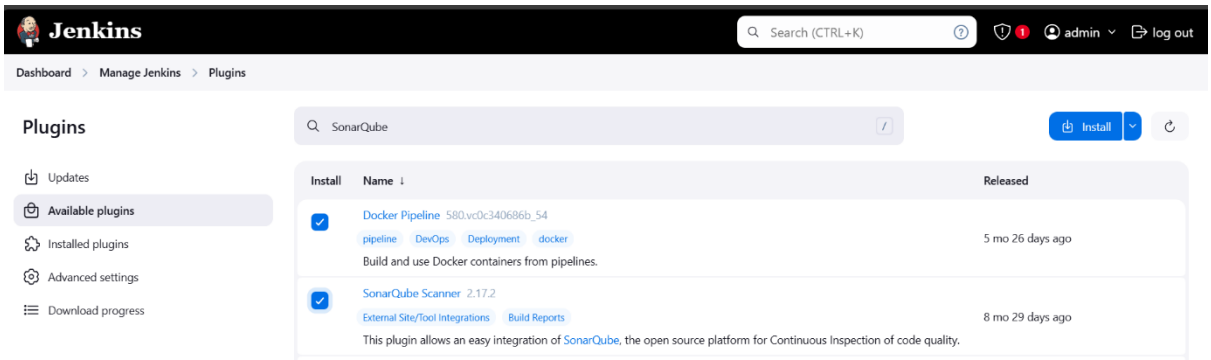
Start using the jenkins



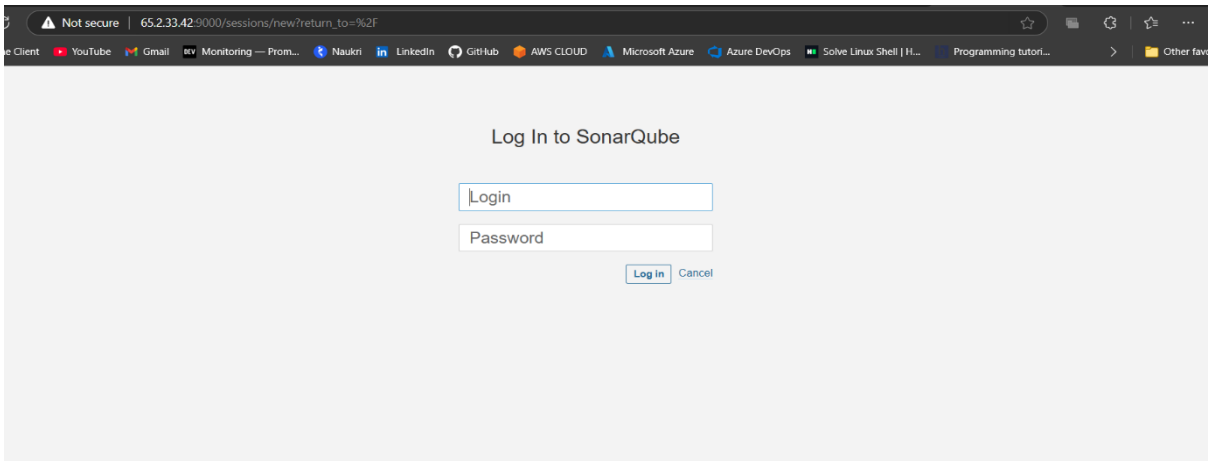
This is how Jenkins dashboard looks like



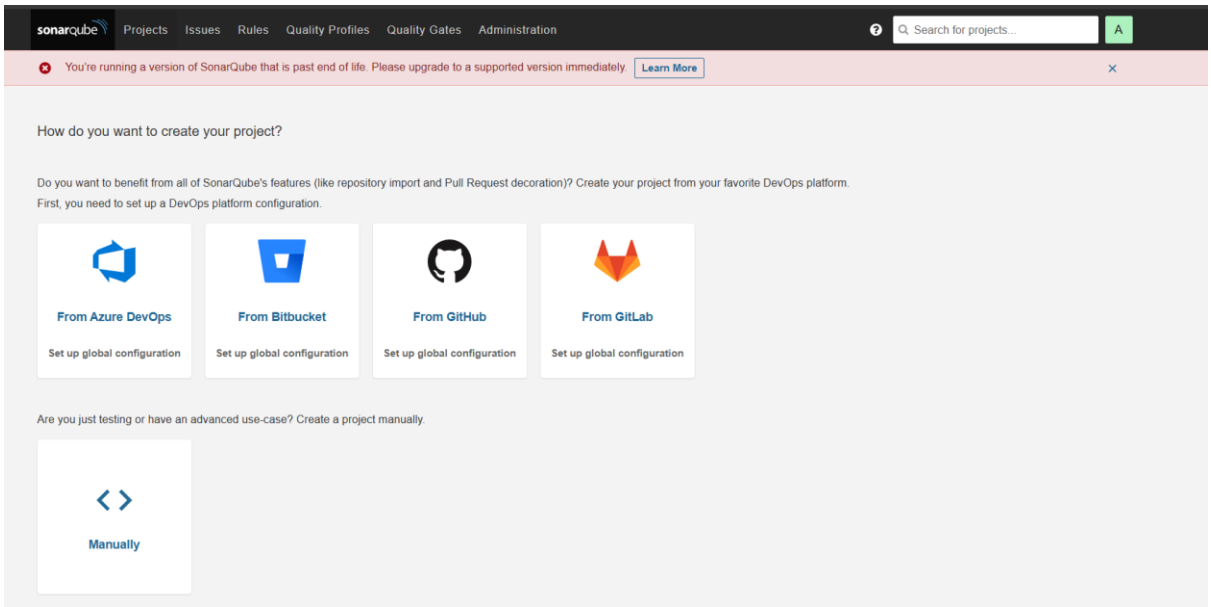
Go to Manage Jenkins > Plugins > Available Plugins and Install Docker Pipeline and SonarQube Scanner Plugins



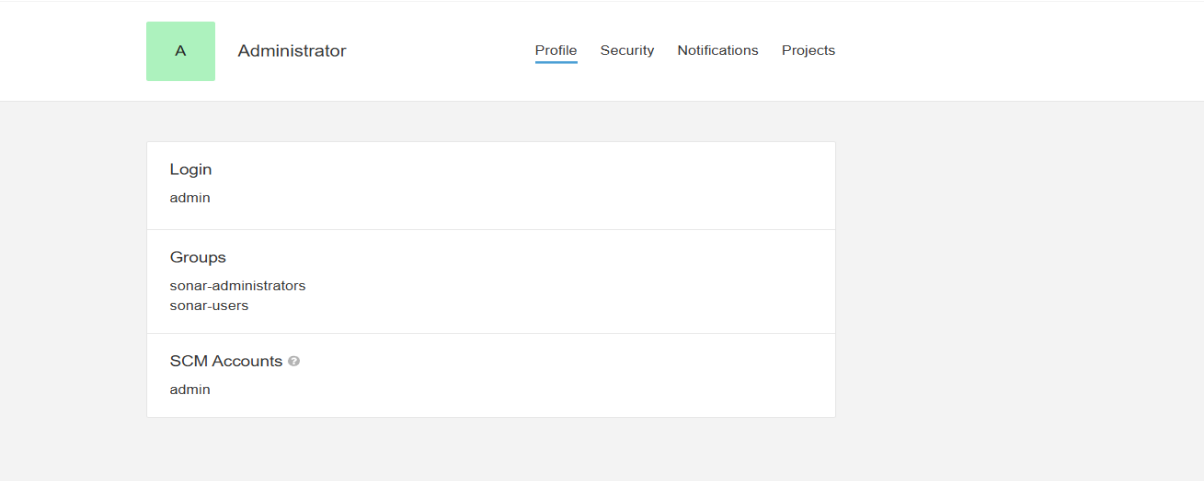
Now, Access the SonarQube on “Public_IPAddress:9000” as SonarQube uses port 9000 as default. Use ‘admin’ as username and password and update the new password and login



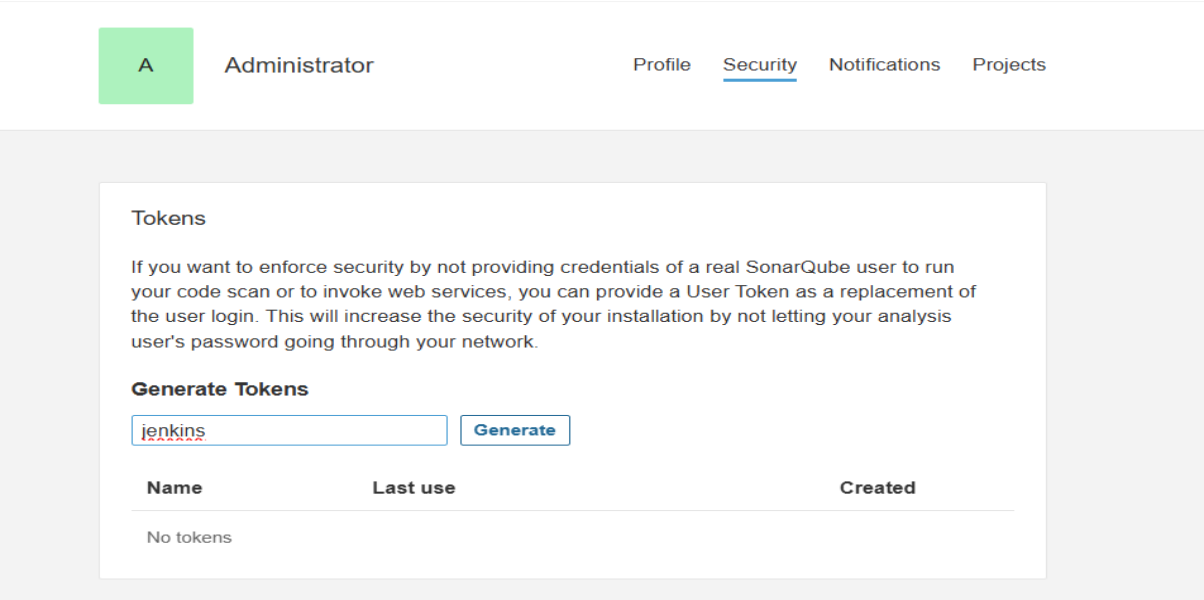
This is how the SonarQube dashboard looks like



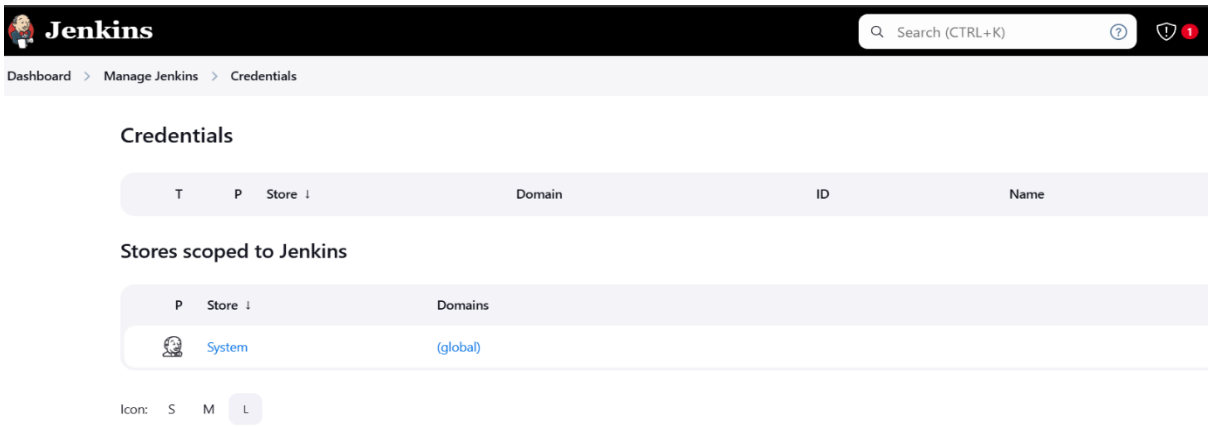
Go to Account > Security



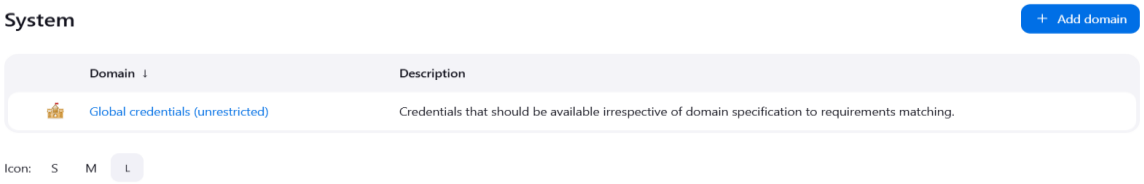
Name the token and generate it. Copy the token and let's connect to jenknis



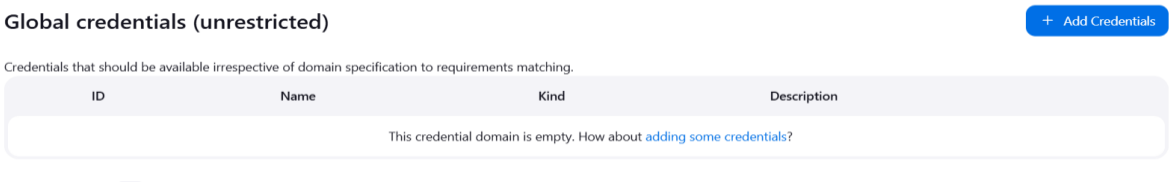
Go to Manage Jenkins > Credentials > System



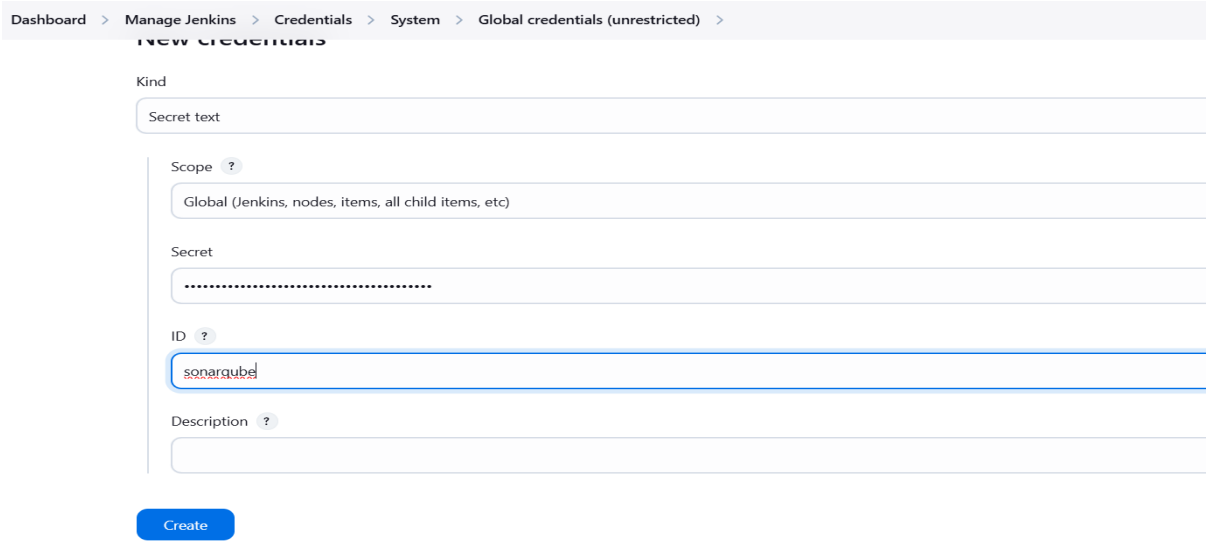
Go to Global credentials



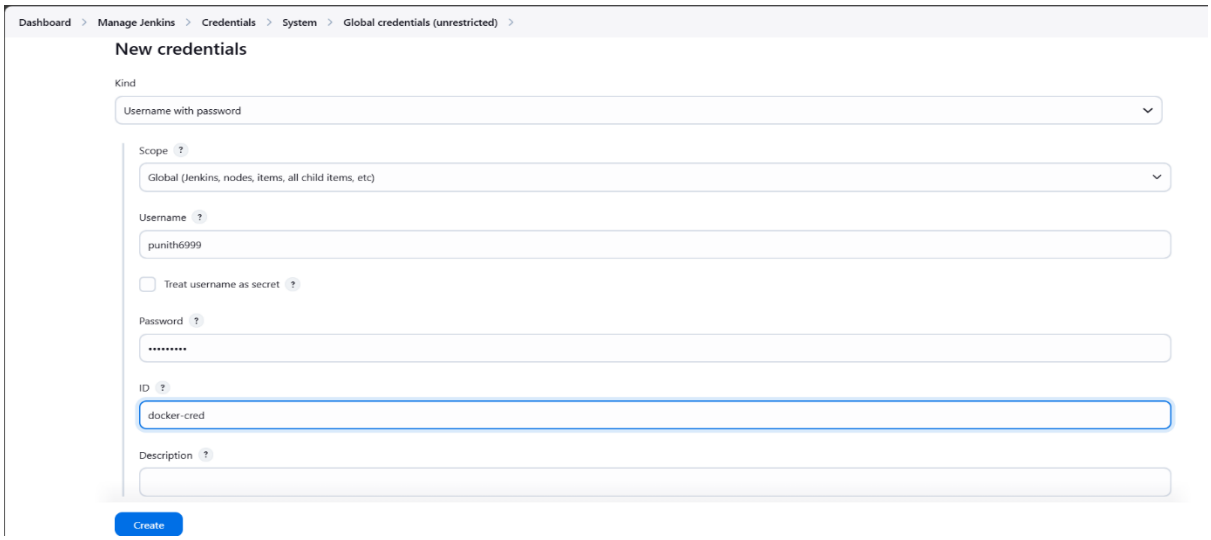
Add new credentials



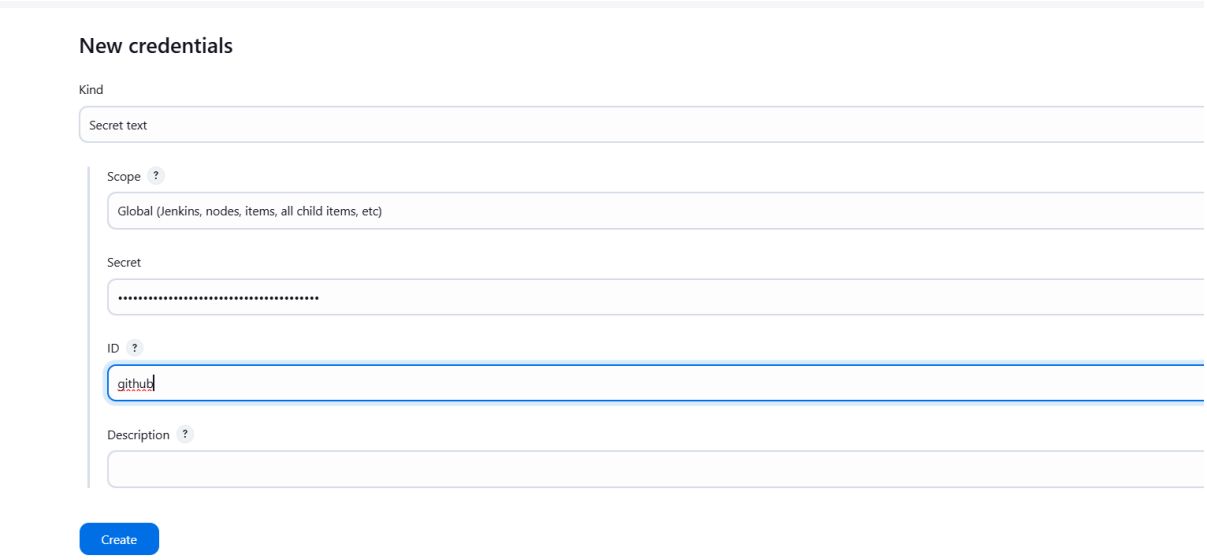
Select Secret text and paste the sonarqube token here and create it



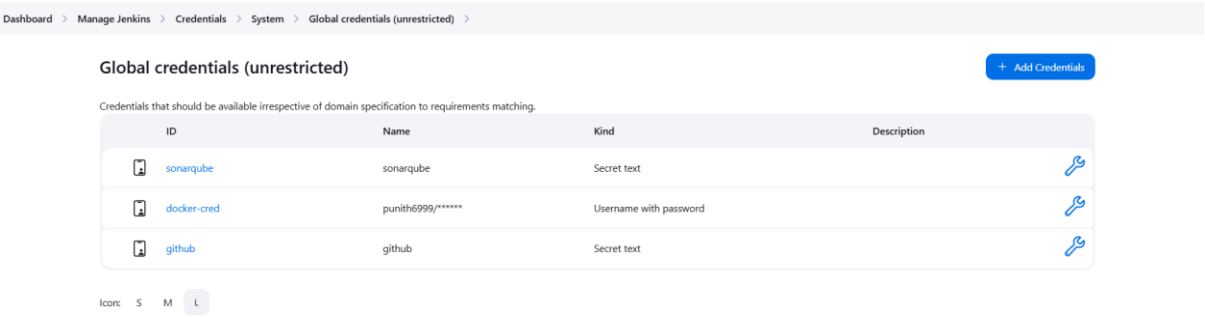
Also, add DockerHub credentials and create it



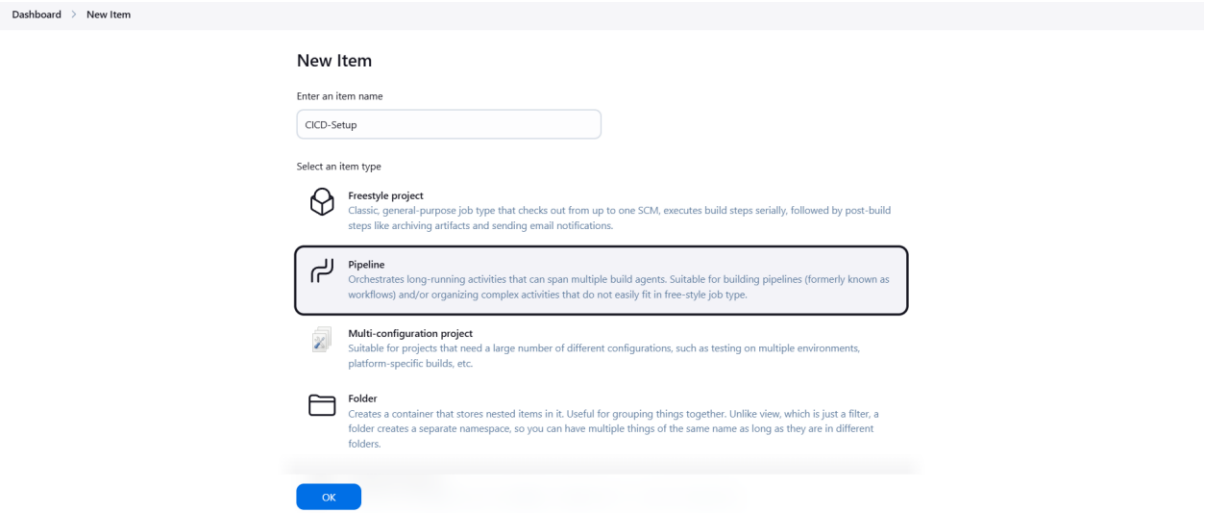
To generate GitHub token, Go to Setting > Developer Settings > Personal access tokens > Tokens(Classic). Generate the token and add it here



The Jenkins configuration is almost done. Now, let's create the job



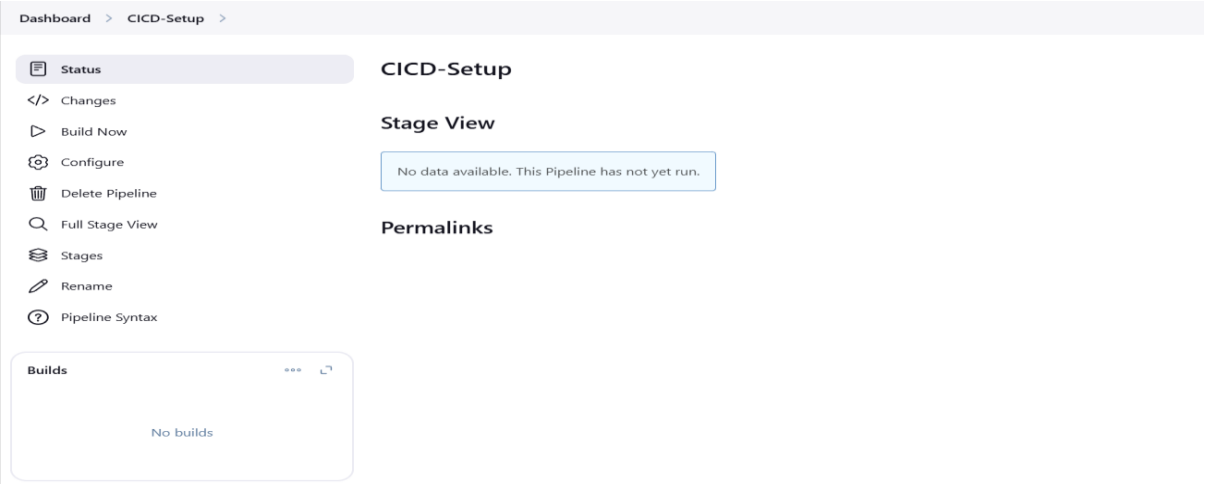
Name the job and select Pipeline as item type



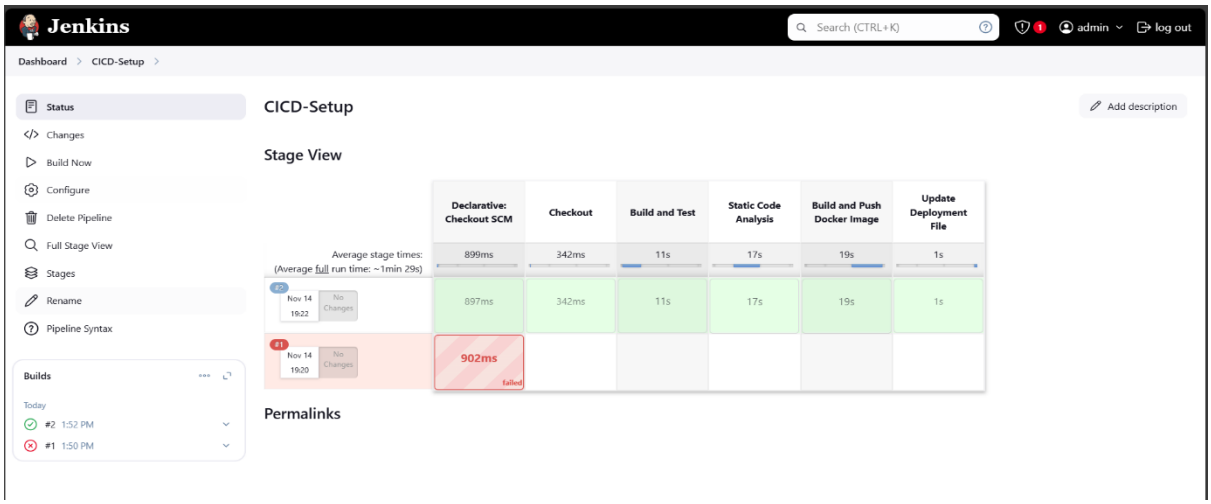
In Pipeline, Select ‘Pipeline script from SCM’ . Choose Git and Enter the GitHub URL

Make sure the branch is correctly configured and Give the path of the Jenkinsfile from the repository. Apply and Save

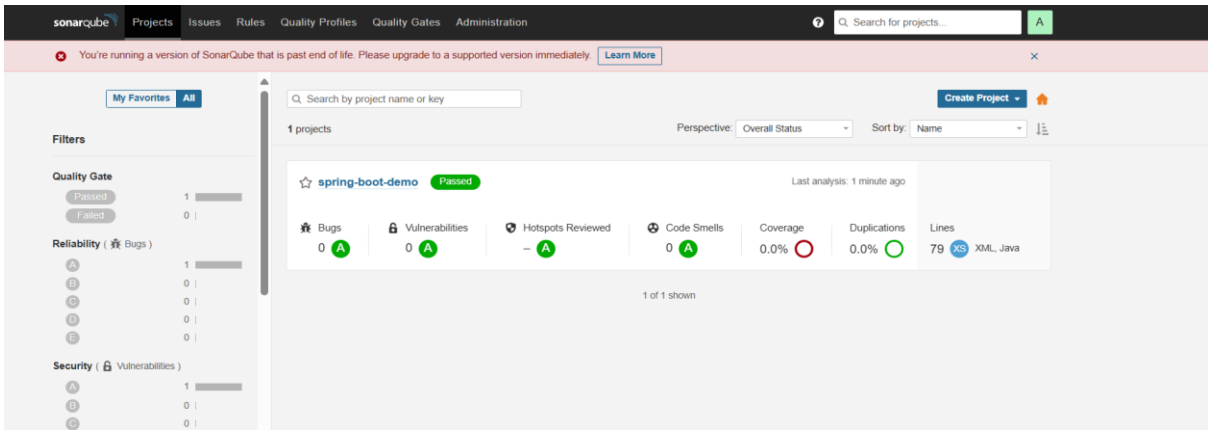
Schedule the Build. Select Build Now



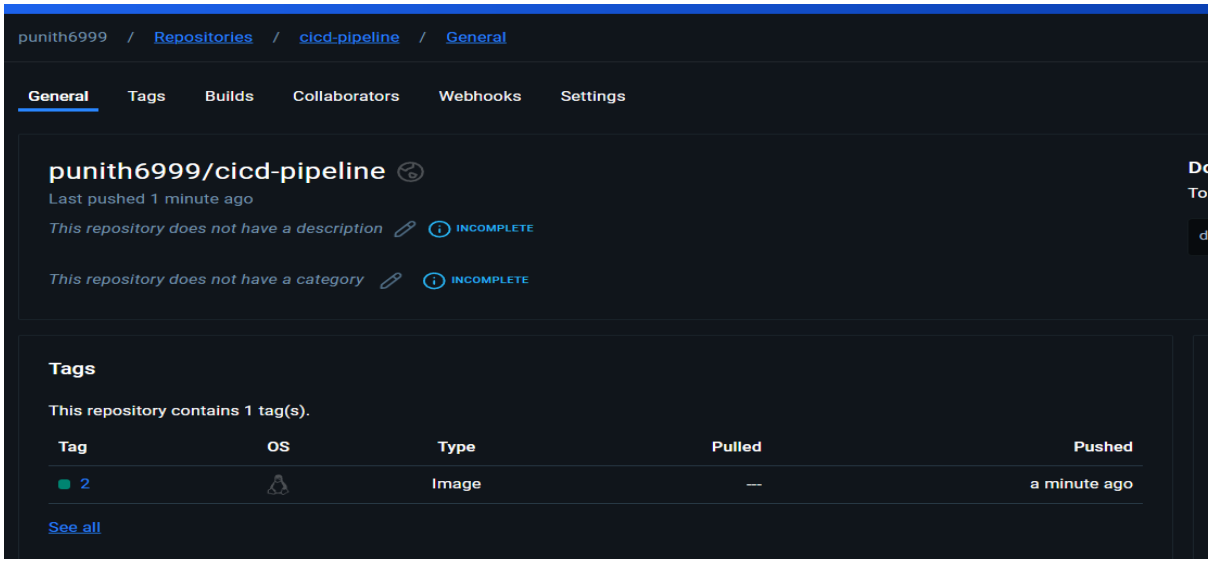
After many attempts, the build is successfully and all the stages executed as expected



Refresh the SonarQube page. We can see the project got created and tested successfully



The image is also pushed successfully to the DockerHub



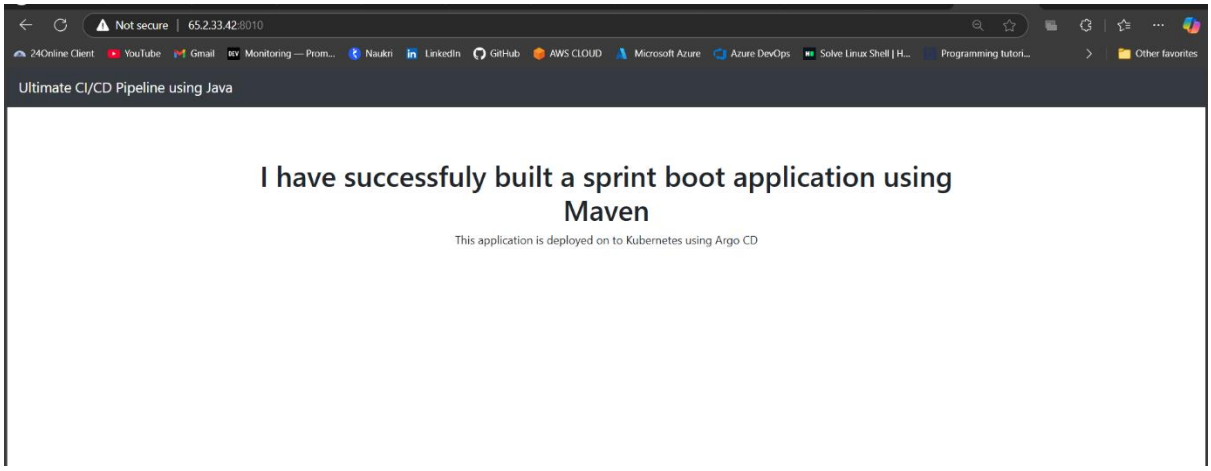
The images are also created on to the host system

```
root@ip-172-31-32-7:~# docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
punith6999/cicd-pipeline      2         5c09997e7d9f   5 minutes ago    170MB
punith6999/sprint-boot-java   v1        3fb9145e2467   19 months ago    913MB
root@ip-172-31-32-7:~#
```

Run the container of the image created

```
root@ip-172-31-32-7:~# docker run -d -p 8010:8080 -t punith6999/cicd-pipeline:2
fb694ba0bc20d27fc464d8cc6a6cf69153157ed320a664d3fcc470fb67d0fbd0
root@ip-172-31-32-7:~#
```

Access the application on 'Public_IPAddress:8010'



Creating Kubernetes Cluster and Deploying using ArgoCD:

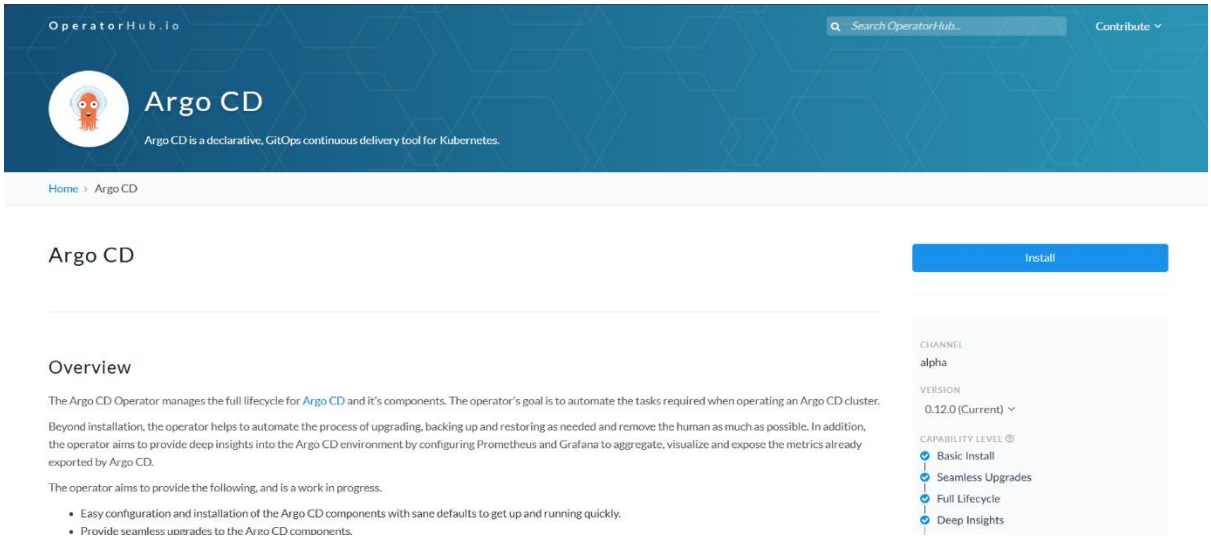
On my local machine (WSL Ubuntu), I have installed minikube to create Kubernetes cluster. Refer README file for the installation steps

```
punith@punith:~$ minikube start
```

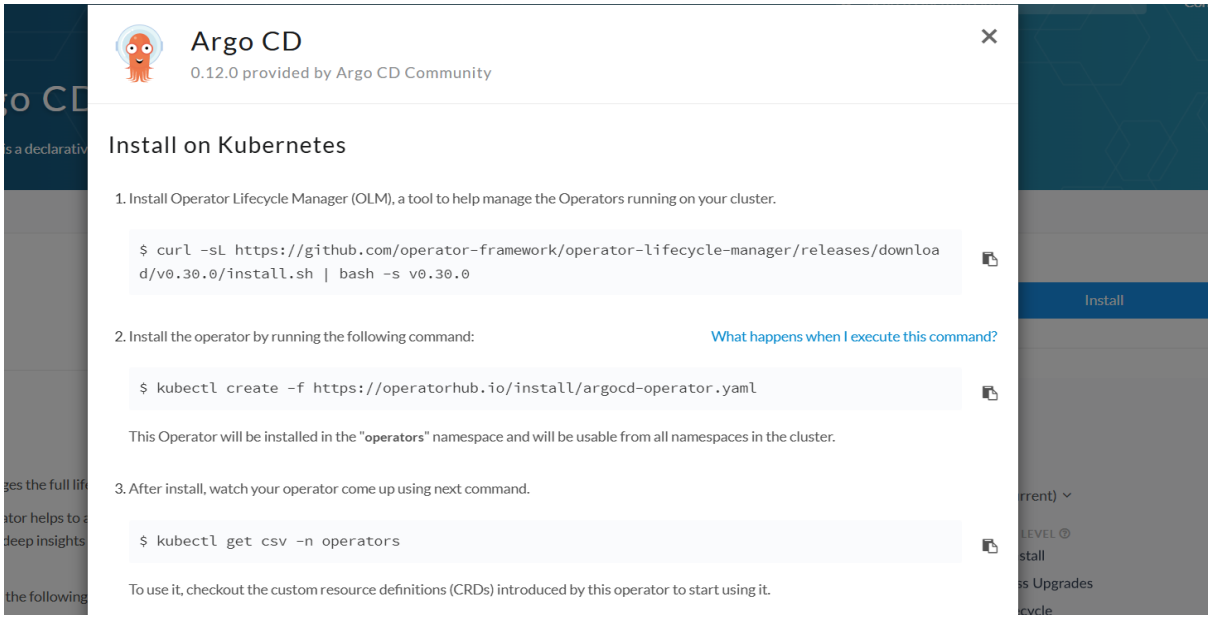
The K8 node is configured and running successfully

```
punith@punith:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   59d   v1.31.0
punith@punith:~$
```

Go to https://operatorhub.io/operator/argocd-operator to Install the K8 operator



Apply these commands to create a separator namespace for the ArgoCD operator



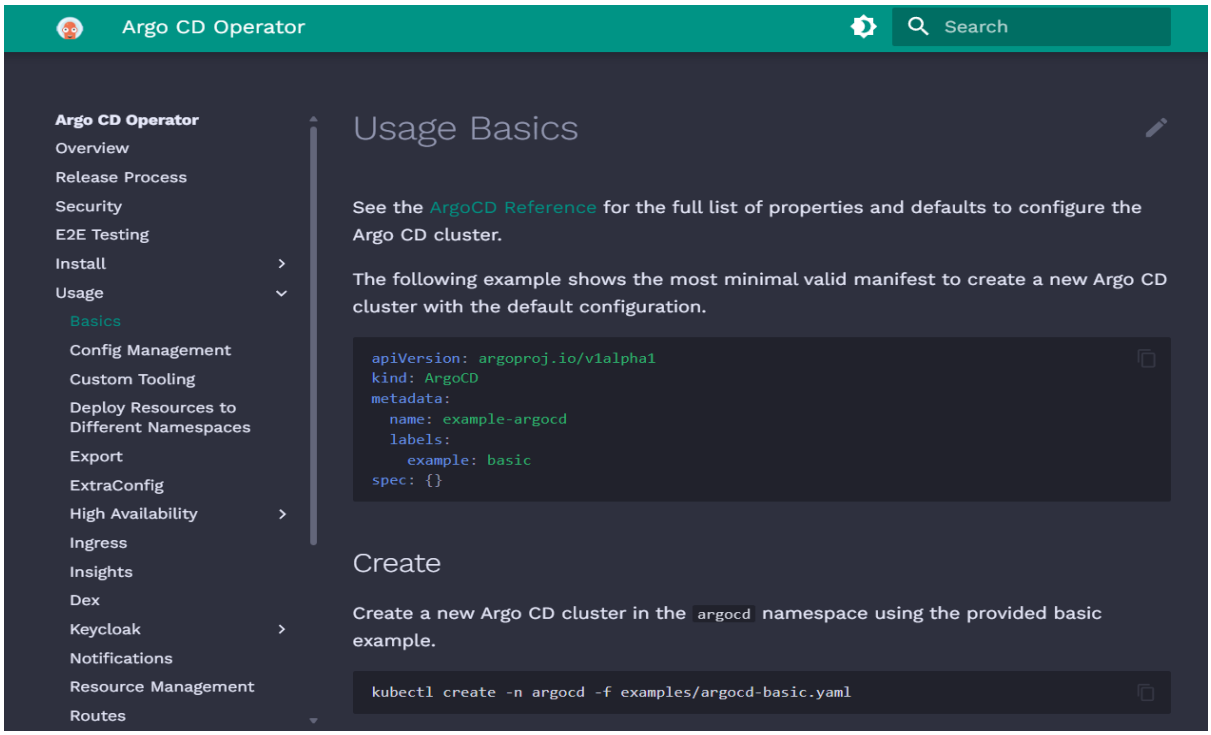
The namespace is created and it is Active

```
punith@punith:~$ kubectl get ns
NAME                STATUS   AGE
default             Active   5m4s
kube-node-lease     Active   5m4s
kube-public         Active   5m4s
kube-system         Active   5m4s
olm                 Active   2m49s
operators           Active   2m49s
punith@punith:~$
```

The ArgoCD operator is also configured successfully

```
punith@punith:~$ kubectl get csv -n operators
NAME                      DISPLAY   VERSION   REPLACES                  PHASE
argocd-operator.v0.12.0   Argo CD   0.12.0    argocd-operator.v0.11.0   Succeeded
punith@punith:~$
```

Use this basic reference script to create ArgoCD cluster on default namespace



Create a .yaml file and save the script

```
punith@punith:~$ vim argocd.yaml
punith@punith:~$ cat argocd.yaml
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec: {}
punith@punith:~$
```

Apply the configurations

```
punith@punith:~$ kubectl apply -f argocd.yaml
argocd.argoproj.io/example-argocd created
punith@punith:~$
```

The ArgoCD pods are created and running successfully

```
punith@punith:~$ kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
example-argocd-application-controller-0       1/1     Running   0          25s
example-argocd-redis-6545fd6d6c-kb4qk         1/1     Running   0          26s
example-argocd-repo-server-869d5757c7-7glgm   1/1     Running   0          26s
example-argocd-server-76bb84cddc-kr7dm        1/1     Running   0          25s
punith@punith:~$
```

List the K8 services

```
punith@punith:~$ kubectl get svc
NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)            AGE
example-argocd-metrics        ClusterIP   10.104.172.40   <none>        8082/TCP           43s
example-argocd-redis          ClusterIP   10.104.202.21   <none>        6379/TCP           43s
example-argocd-repo-server    ClusterIP   10.110.34.119   <none>        8081/TCP,8084/TCP  43s
example-argocd-server         ClusterIP   10.103.208.52   <none>        80/TCP,443/TCP     43s
example-argocd-server-metrics ClusterIP   10.111.50.207   <none>        8083/TCP           43s
kubernetes                    ClusterIP   10.96.0.1       <none>        443/TCP            51m
punith@punith:~$
```

In the ‘example-argocd-server’ , Change the Service type from ClusterIP to NodePort and save it

```
    targetPort: 8080
  selector:
    app.kubernetes.io/name: example-argocd-server
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
:wq
```

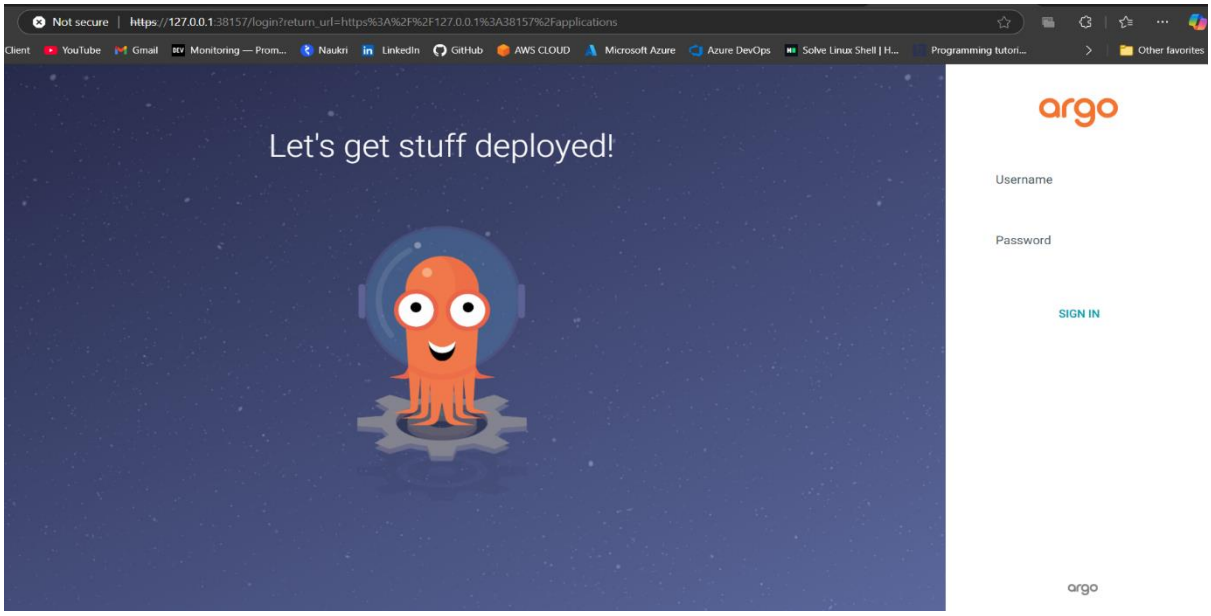
The service type is changed successfully

```
punith@punith:~$ kubectl edit svc example-argocd-server
service/example-argocd-server edited
punith@punith:~$ kubectl get svc
NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
example-argocd-metrics        ClusterIP   10.104.172.40   <none>        8082/TCP                     112s
example-argocd-redis          ClusterIP   10.104.202.21   <none>        6379/TCP                     112s
example-argocd-repo-server    ClusterIP   10.110.34.119   <none>        8081/TCP,8084/TCP            112s
example-argocd-server         NodePort    10.103.208.52   <none>        80:32338/TCP,443:30796/TCP   112s
example-argocd-server-metrics ClusterIP   10.111.50.207   <none>        8083/TCP                     112s
kubernetes                    ClusterIP   10.96.0.1       <none>        443/TCP                      52m
punith@punith:~$
```

Start the minikube service to access the ArgoCD

```
punith@punith:~$ minikube service example-argocd-server
|-----------|-----------------------|-------------|---------------------------|
| NAMESPACE |         NAME          | TARGET PORT |            URL            |
|-----------|-----------------------|-------------|---------------------------|
| default   | example-argocd-server | http/80     | http://192.168.58.2:32338 |
|           |                       | https/443   | http://192.168.58.2:30796 |
|-----------|-----------------------|-------------|---------------------------|
🏃  Starting tunnel for service example-argocd-server.
|-----------|-----------------------|-------------|------------------------|
| NAMESPACE |         NAME          | TARGET PORT |          URL           |
|-----------|-----------------------|-------------|------------------------|
| default   | example-argocd-server |             | http://127.0.0.1:45809 |
|           |                       |             | http://127.0.0.1:38157 |
|-----------|-----------------------|-------------|------------------------|
[default example-argocd-server  http://127.0.0.1:45809
http://127.0.0.1:38157]
❗  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

Access ArgoCD on the URL generated by the minikube service



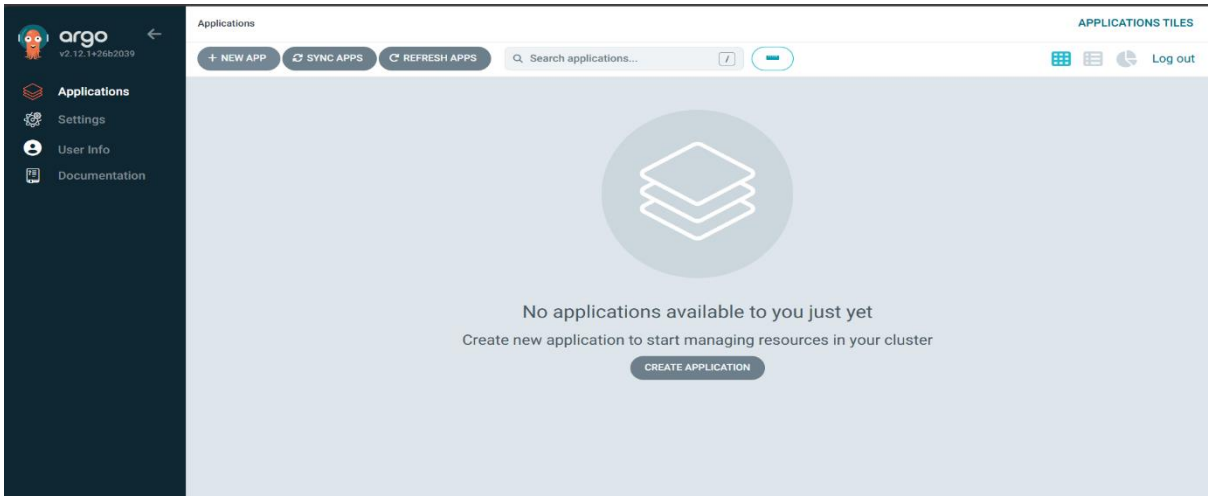
List the K8 secrets and get the admin password for ArgoCD

```
punith@punith:~$ kubectl get secret
NAME                                  TYPE                  DATA   AGE
argocd-secret                         Opaque                5      4m18s
example-argocd-ca                     kubernetes.io/tls     3      4m18s
example-argocd-cluster                Opaque                1      4m18s
example-argocd-default-cluster-config Opaque                4      4m18s
example-argocd-redis-initial-password Opaque                2      4m18s
example-argocd-tls                    kubernetes.io/tls     2      4m18s
punith@punith:~$ kubectl edit secret example-argocd-cluster
```

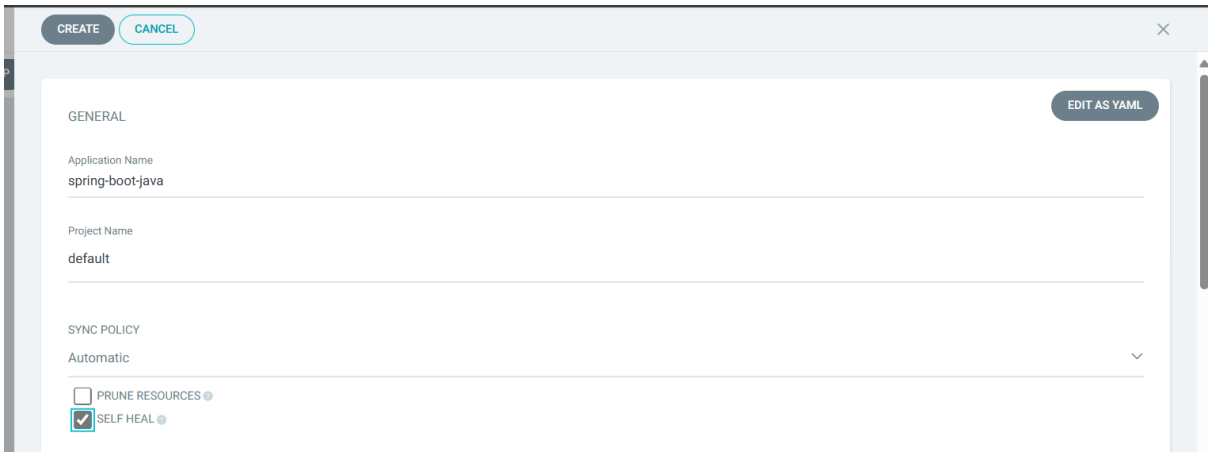
Encode the password to normal text password

```
punith@punith:~$ echo czNOU1EySmhIa0xxb0NheVBpd0ZZVjE1VWNPOERibEs= | base64 -d
s3NSQ2JhHkLqoCayPiwFYV15UcO8DblKpunith@punith:~$
```

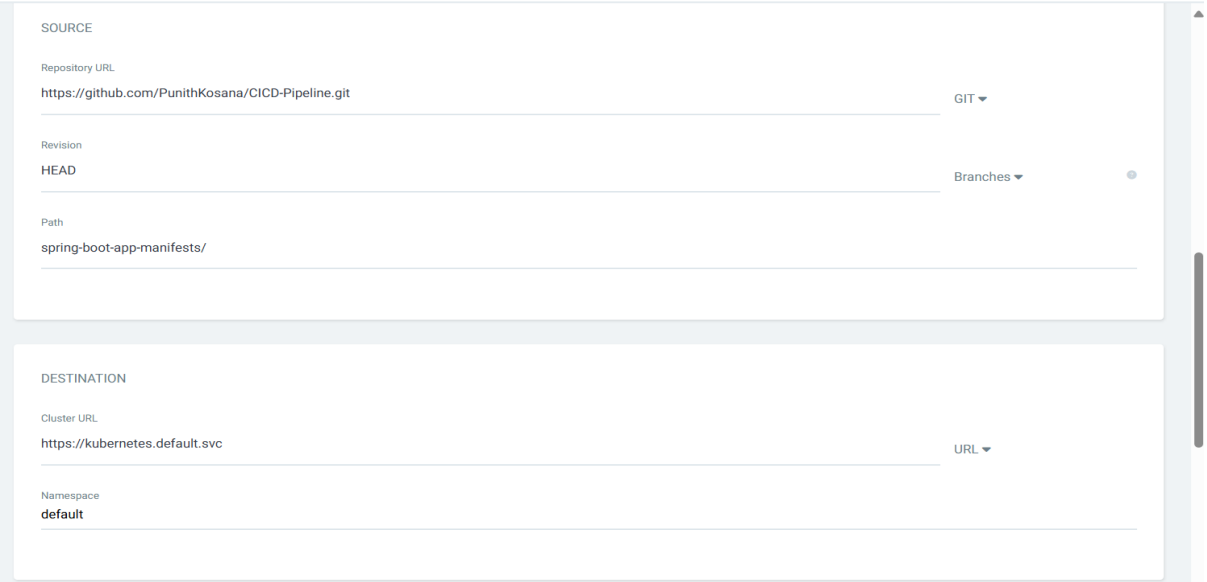
Use 'admin' as username and the generated password and login to the ArgoCD dashboard. Create Application to deploy the pods



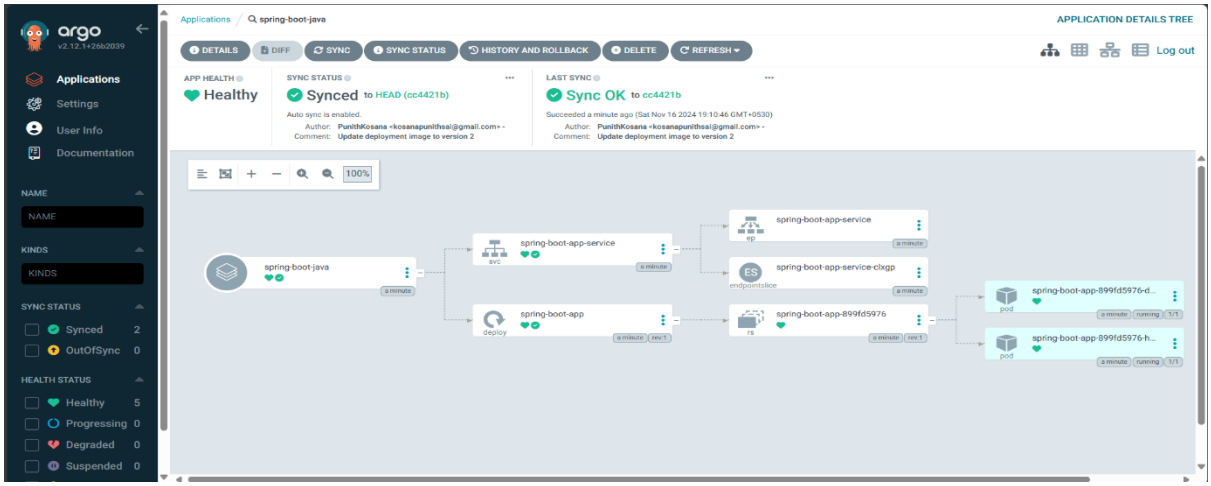
Name the application and select as default. Choose the SYNC as Automatic and SELF HEAL



Enter the GitHub URL of the project and specify the path of the deployment.yml file which is used for deploying the pods. Select the Cluster URL and Namespace as default



The application is created successfully and it is in Sync and Healthy.



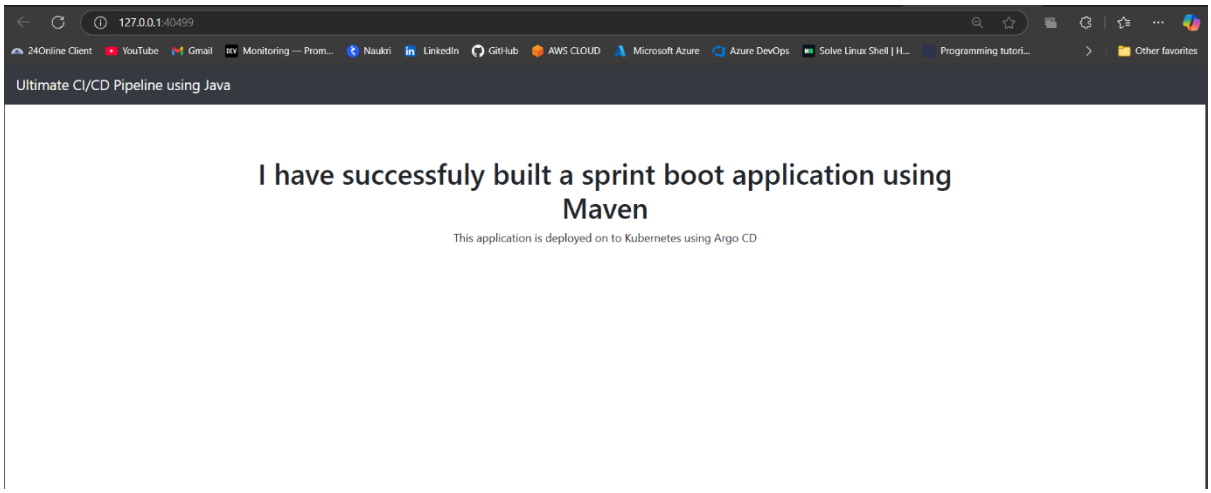
List the deployment and pods and the resources for the application are created as expected

```
punith@punith:~$ kubectl get deploy
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
example-argocd-redis      1/1     1            1           12m
example-argocd-repo-server   1/1     1            1           12m
example-argocd-server     1/1     1            1           12m
spring-boot-app           2/2     2            2           66s
punith@punith:~$ kubectl get pods
NAME                                      READY   STATUS    RESTARTS   AGE
example-argocd-application-controller-0   1/1     Running   0          12m
example-argocd-redis-6545fd6d6c-kb4qk     1/1     Running   0          12m
example-argocd-repo-server-869d5757c7-7glgm   1/1     Running   0          12m
example-argocd-server-76bb84cddc-kr7dm    1/1     Running   0          12m
spring-boot-app-899fd5976-dv4b5           1/1     Running   0          71s
spring-boot-app-899fd5976-hp2hq           1/1     Running   0          71s
punith@punith:~$
```

Start the minikube service of the pod created for the application

```
punith@punith:~$ kubectl get svc
NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                    AGE
example-argocd-metrics        ClusterIP   10.104.172.40   <none>        8082/TCP                   13m
example-argocd-redis          ClusterIP   10.104.202.21   <none>        6379/TCP                   13m
example-argocd-repo-server    ClusterIP   10.110.34.119   <none>        8081/TCP,8084/TCP          13m
example-argocd-server         NodePort    10.103.208.52   <none>        80:32338/TCP,443:30796/TCP 13m
example-argocd-server-metrics ClusterIP   10.111.50.207   <none>        8083/TCP                   13m
kubernetes                    ClusterIP   10.96.0.1       <none>        443/TCP                    63m
spring-boot-app-service       NodePort    10.98.193.227   <none>        80:32707/TCP               115s
punith@punith:~$ minikube service spring-boot-app-service
|-----------|-------------------------|-------------|---------------------------|
| NAMESPACE |          NAME           | TARGET PORT |            URL            |
|-----------|-------------------------|-------------|---------------------------|
| default   | spring-boot-app-service | http/80     | http://192.168.58.2:32707 |
|-----------|-------------------------|-------------|---------------------------|
🏃  Starting tunnel for service spring-boot-app-service.
|-----------|-------------------------|-------------|------------------------|
| NAMESPACE |          NAME           | TARGET PORT |          URL           |
|-----------|-------------------------|-------------|------------------------|
| default   | spring-boot-app-service |             | http://127.0.0.1:40499 |
|-----------|-------------------------|-------------|------------------------|
🎉  Opening service default/spring-boot-app-service in default browser...
👉  http://127.0.0.1:40499
❗  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

Access the URL generated by minikube service. The application is running successfully



Monitoring Setup:

Installing Prometheus and Grafana using Helm. Refer to the README file from GitHub Repository for more information on installation steps

GitHub Link: https://github.com/PunithKosana/CICD-Pipeline.git

After installating helm, update the repo

```
punith@punith:~$ helm repo update
```

Add the Prometheus helm repo

```
punith@punith:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

Add Grafana helm repo

```
punith@punith:~$ helm repo add grafana https://grafana.github.io/helm-charts
```

Install the Prometheus from the helm repo

```
punith@punith:~$ helm install prometheus prometheus-community/prometheus
```

Install Grafana from the helm repo

```
punith@punith:~$ helm install grafana grafana/grafana
```

List the pods and wait for the pods to get in running state

```
punith@punith:~$ kubectl get pods
NAME                                                  READY   STATUS    RESTARTS   AGE
example-argocd-application-controller-0               1/1     Running   0          133m
example-argocd-redis-6545fd6d6c-w4qr7                 1/1     Running   0          133m
example-argocd-repo-server-869d5757c7-4jxxj           1/1     Running   0          133m
example-argocd-server-76bb84cddc-78p8b                1/1     Running   0          133m
grafana-59dd7db7b6-t5jqs                              1/1     Running   0          106m
prometheus-alertmanager-0                             1/1     Running   0          117m
prometheus-kube-state-metrics-88947546-27z8x          1/1     Running   0          117m
prometheus-prometheus-node-exporter-6ld8v             1/1     Running   0          117m
prometheus-prometheus-pushgateway-9f8c968d6-wtvkb     1/1     Running   0          117m
prometheus-server-5f75d65fbc-m942h                    2/2     Running   0          117m
spring-boot-app-899fd5976-qdxgg                       1/1     Running   0          119m
spring-boot-app-899fd5976-svq5q                       1/1     Running   0          119m
punith@punith:~$
```

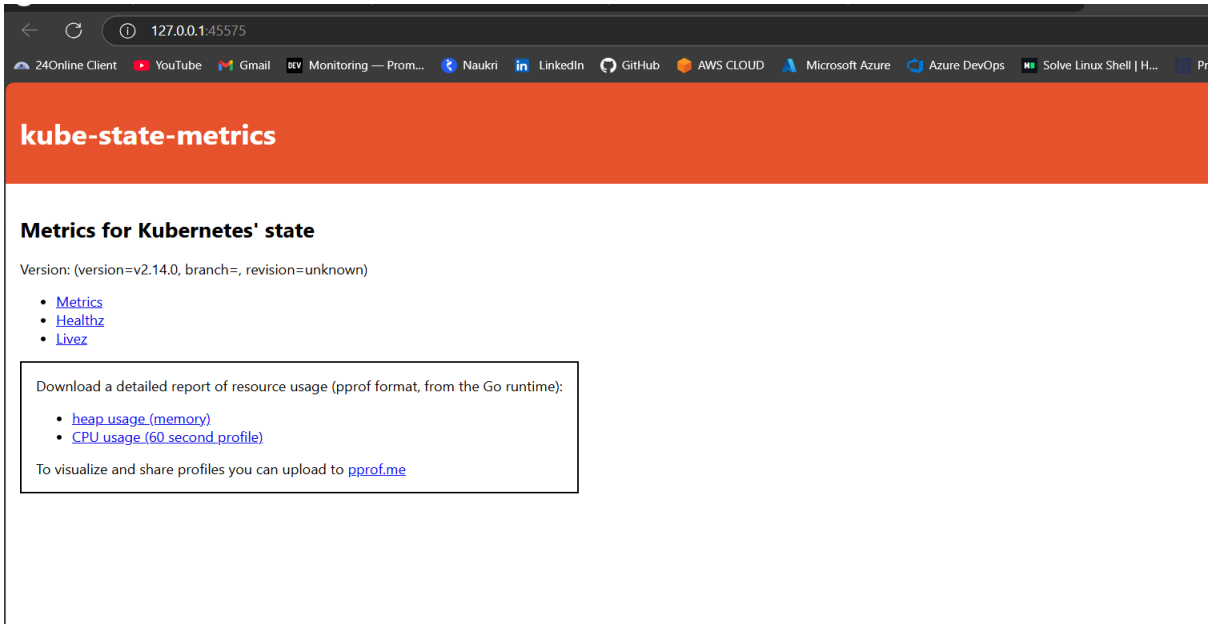
List the services. Edit prometheus-server, grafana and prometheus-kube-state-metrics from ClusterIP to Nodeport to access them

```
punith@punith:~$ kubectl get svc
NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                    AGE
example-argocd-metrics                ClusterIP   10.109.147.185   <none>        8082/TCP                   134m
example-argocd-redis                  ClusterIP   10.96.136.203    <none>        6379/TCP                   134m
example-argocd-repo-server            ClusterIP   10.109.120.226   <none>        8081/TCP,8084/TCP          134m
example-argocd-server                 NodePort    10.106.5.47      <none>        80:31875/TCP,443:31488/TCP 134m
example-argocd-server-metrics         ClusterIP   10.106.10.203    <none>        8083/TCP                   134m
grafana                               NodePort    10.99.135.49     <none>        80:32152/TCP               107m
kubernetes                            ClusterIP   10.96.0.1        <none>        443/TCP                    145m
prometheus-alertmanager               ClusterIP   10.98.138.142    <none>        9093/TCP                   117m
prometheus-alertmanager-headless      ClusterIP   None             <none>        9093/TCP                   117m
prometheus-kube-state-metrics         NodePort    10.96.227.195    <none>        8080:31135/TCP             117m
prometheus-prometheus-node-exporter   ClusterIP   10.102.137.66    <none>        9100/TCP                   117m
prometheus-prometheus-pushgateway     ClusterIP   10.104.174.226   <none>        9091/TCP                   117m
prometheus-server                     NodePort    10.108.70.237    <none>        80:31880/TCP               117m
spring-boot-app-service               NodePort    10.108.152.0     <none>        80:32025/TCP               119m
punith@punith:~$
```

Use minikube service to start the kube-state-metrics

```
punith@punith:~$ minikube service prometheus-kube-state-metrics
|-----------|-------------------------------|-------------|---------------------------|
| NAMESPACE |             NAME              | TARGET PORT |            URL            |
|-----------|-------------------------------|-------------|---------------------------|
| default   | prometheus-kube-state-metrics | http/8080   | http://192.168.49.2:31135 |
|-----------|-------------------------------|-------------|---------------------------|
🏃  Starting tunnel for service prometheus-kube-state-metrics.
|-----------|-------------------------------|-------------|------------------------|
| NAMESPACE |             NAME              | TARGET PORT |          URL           |
|-----------|-------------------------------|-------------|------------------------|
| default   | prometheus-kube-state-metrics |             | http://127.0.0.1:45575 |
|-----------|-------------------------------|-------------|------------------------|
🎉  Opening service default/prometheus-kube-state-metrics in default browser...
👉  http://127.0.0.1:45575
❗  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

Access the kube-state-metrics on IPAddress generated by minikube



List the ConfigMaps

```
punith@punith:~$ kubectl get cm
NAME                          DATA    AGE
argocd-cm                     17      137m
argocd-gpg-keys-cm            0       137m
argocd-rbac-cm                3       137m
argocd-ssh-known-hosts-cm     1       137m
argocd-tls-certs-cm           0       137m
example-argocd-ca             1       137m
grafana                       1       110m
kube-root-ca.crt              1       148m
prometheus-alertmanager       1       121m
prometheus-server             6       121m
punith@punith:~$
```

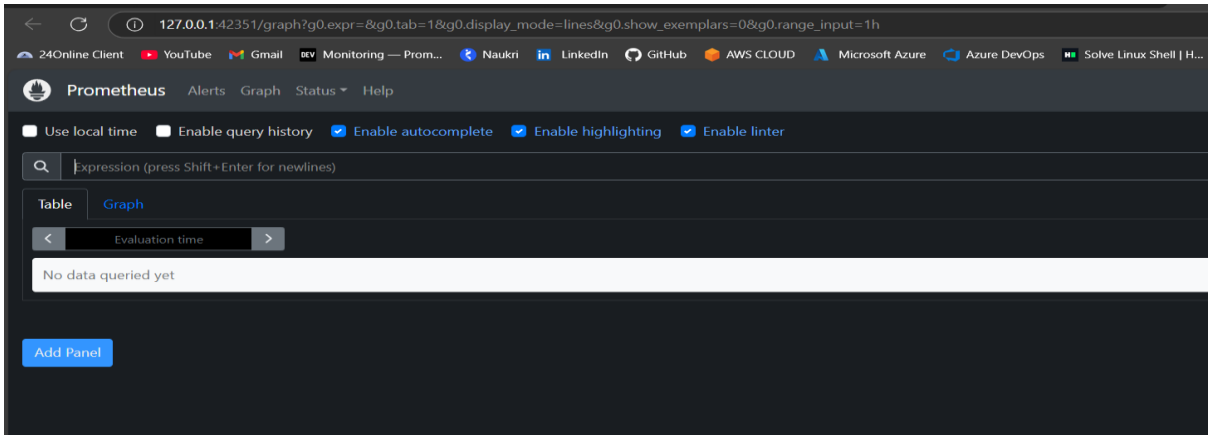
Add the kube_metrics URL in the scrape_configs so that prometheus scrapes the metrics from the cluster

```
    scrape_configs:
    - job_name: prometheus
      static_configs:
      - targets:
        - localhost:9090
    - job_name: prometheus
      static_configs:
      - targets:
        - 192.168.49.2:31135
```

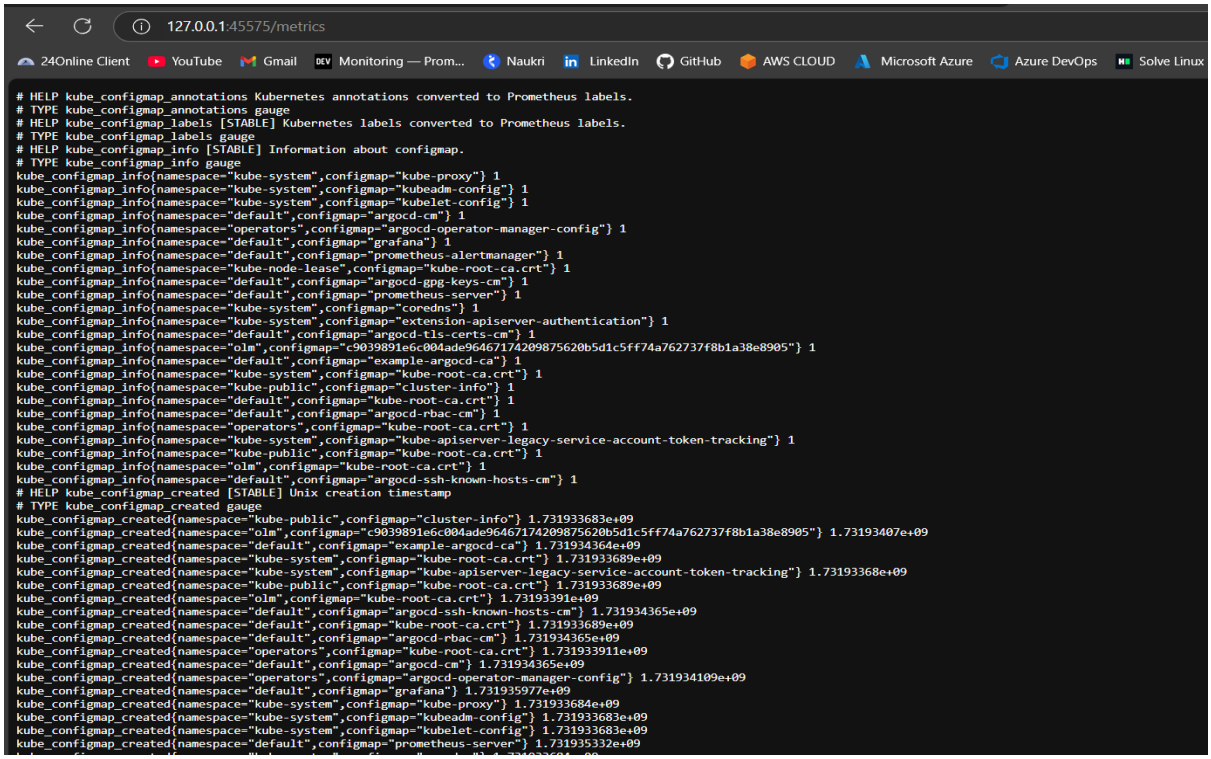
Start the Prometheus service using minikube

```
punith@punith:~$ minikube service prometheus-server
|-----------|-------------------|-------------|--------------------------|
| NAMESPACE |       NAME        | TARGET PORT |           URL            |
|-----------|-------------------|-------------|--------------------------|
| default   | prometheus-server | http/80     | http://192.168.49.2:31880 |
|-----------|-------------------|-------------|--------------------------|
🏃  Starting tunnel for service prometheus-server.
|-----------|-------------------|-------------|-----------------------|
| NAMESPACE |       NAME        | TARGET PORT |          URL          |
|-----------|-------------------|-------------|-----------------------|
| default   | prometheus-server |             | http://127.0.0.1:42351 |
|-----------|-------------------|-------------|-----------------------|
🎉  Opening service default/prometheus-server in default browser...
👉  http://127.0.0.1:42351
❗  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

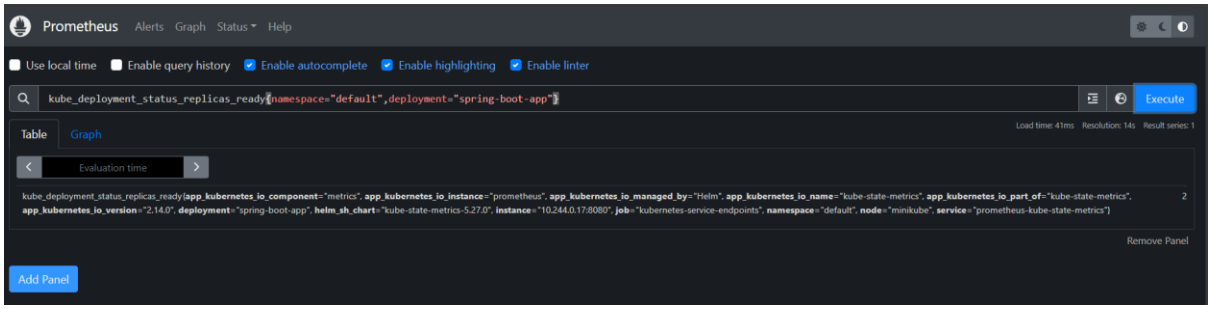
Access Prometheus on IPAddress generated by minikube



These are the metrics of the cluster and can be accessed on kube-metrics



Run query from the kube-metrics on Prometheus to check the replica status



List the secrets

```
punith@punith:~$ kubectl get secret
NAME                                  TYPE                  DATA   AGE
argocd-secret                         Opaque                5      140m
example-argocd-ca                     kubernetes.io/tls     3      140m
example-argocd-cluster                Opaque                1      140m
example-argocd-default-cluster-config Opaque                4      140m
example-argocd-redis-initial-password Opaque                2      140m
example-argocd-tls                    kubernetes.io/tls     2      140m
grafana                               Opaque                3      113m
sh.helm.release.v1.grafana.v1         helm.sh/release.v1    1      113m
sh.helm.release.v1.prometheus.v1      helm.sh/release.v1    1      124m
punith@punith:~$
```

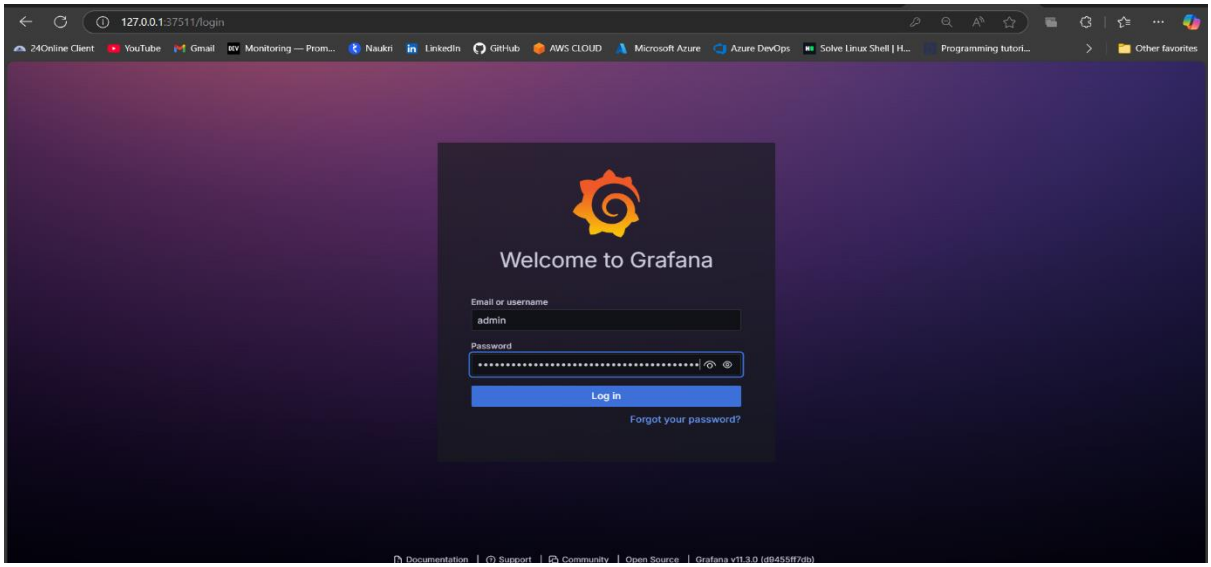
Get the admin_password and encode the password

```
punith@punith:~$ kubectl get secret grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
hD9vf5Tb1CRVTlFz3S8x3Z2f5YVWUE5TIZKF2UYc
punith@punith:~$
```

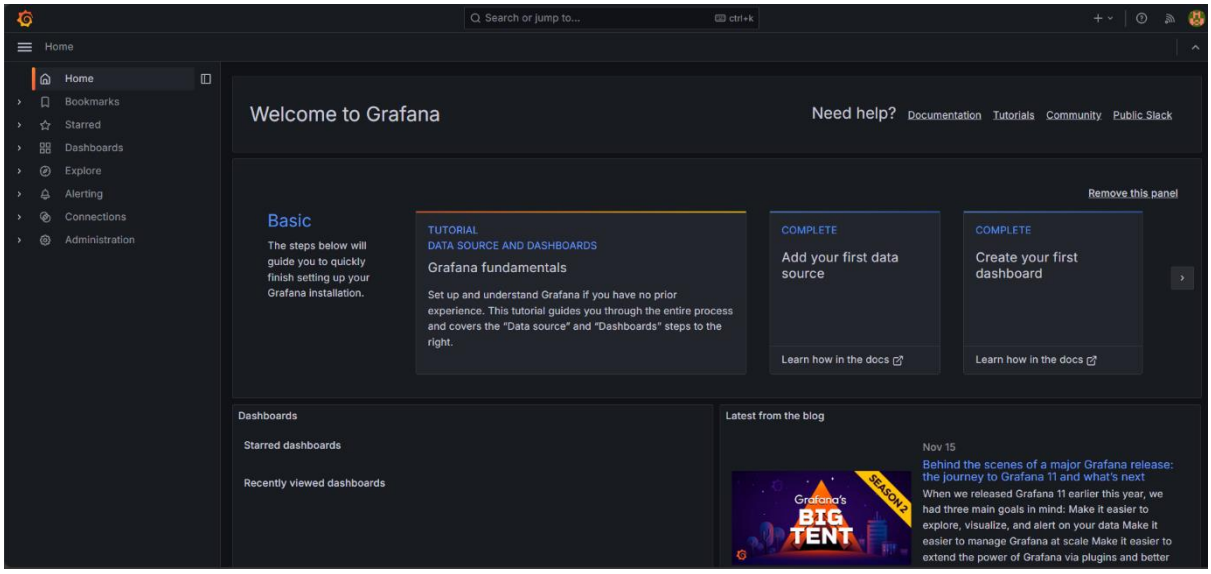
Start the Grafana service using minikube

```
punith@punith:~$ minikube service grafana
|-----------|---------|-------------|---------------------------|
| NAMESPACE |  NAME   | TARGET PORT |            URL            |
|-----------|---------|-------------|---------------------------|
| default   | grafana | service/80  | http://192.168.49.2:32152 |
|-----------|---------|-------------|---------------------------|
🏃  Starting tunnel for service grafana.
|-----------|---------|-------------|------------------------|
| NAMESPACE |  NAME   | TARGET PORT |          URL           |
|-----------|---------|-------------|------------------------|
| default   | grafana |             | http://127.0.0.1:37511 |
|-----------|---------|-------------|------------------------|
🎉  Opening service default/grafana in default browser...
👉  http://127.0.0.1:37511
❗  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

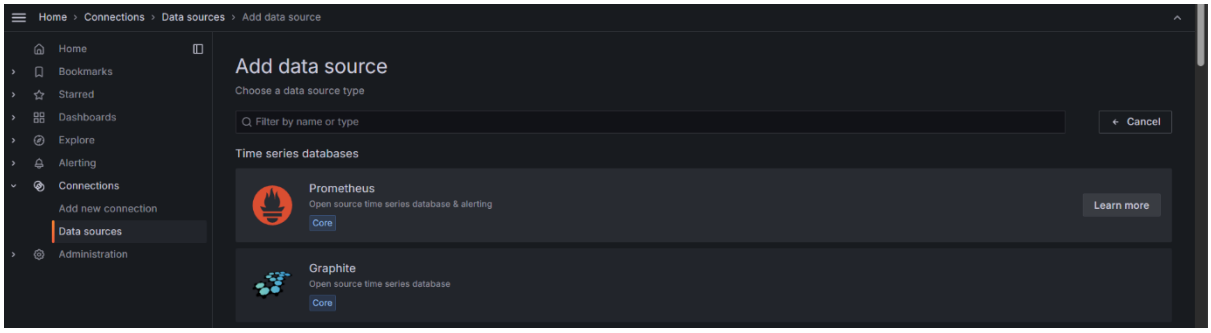
Enter ‘admin’ as username and generated password. Login to the dashboard



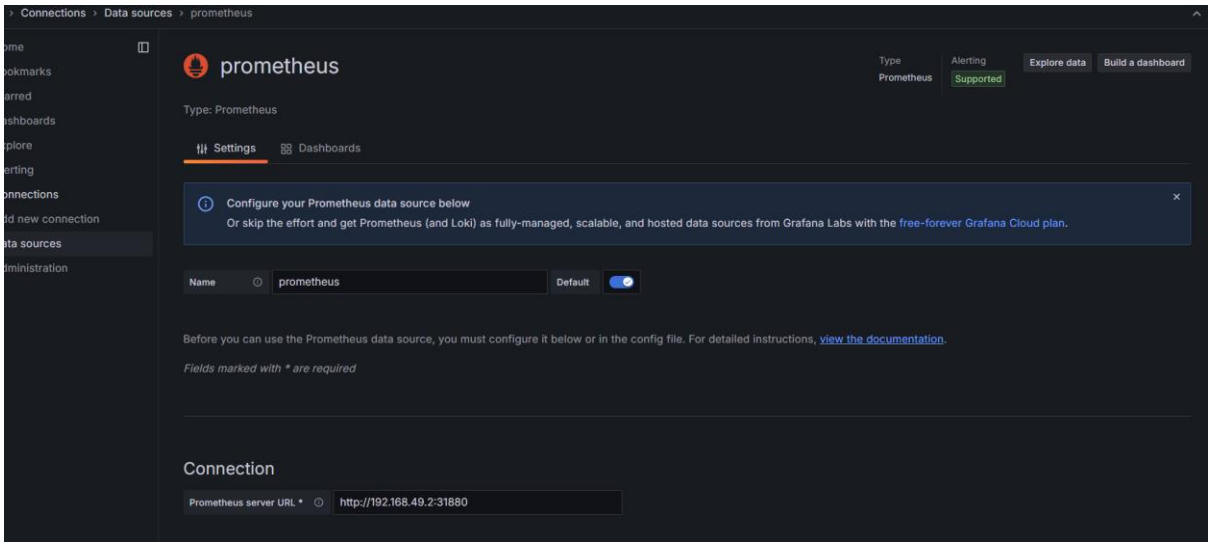
This is how Grafana dashboard looks like



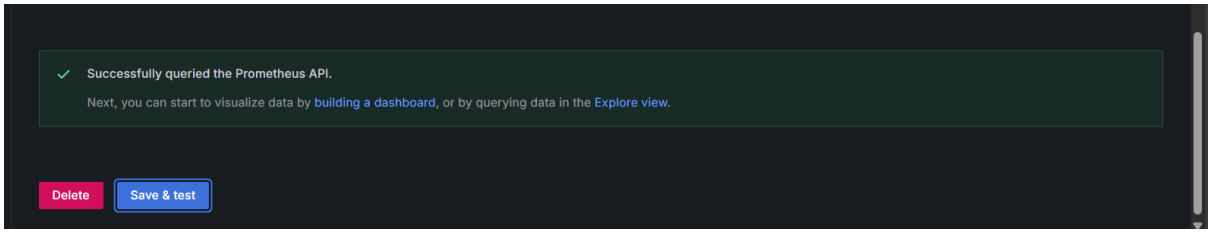
Choose Add data source and select prometheus



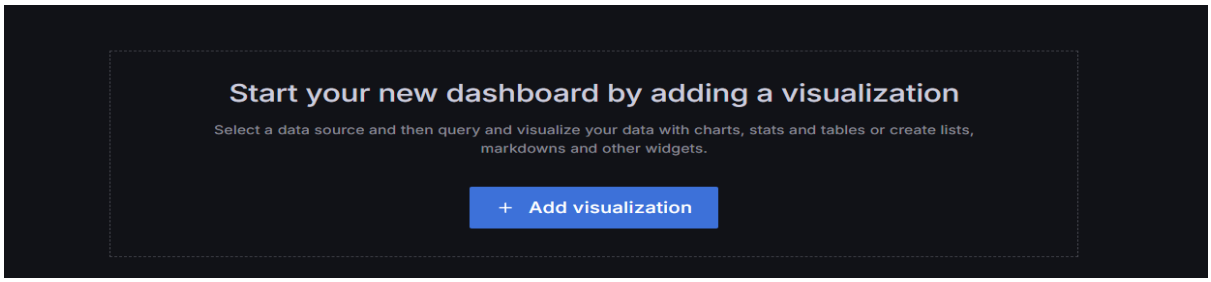
Enter the Prometheus URL to connect the prometheus



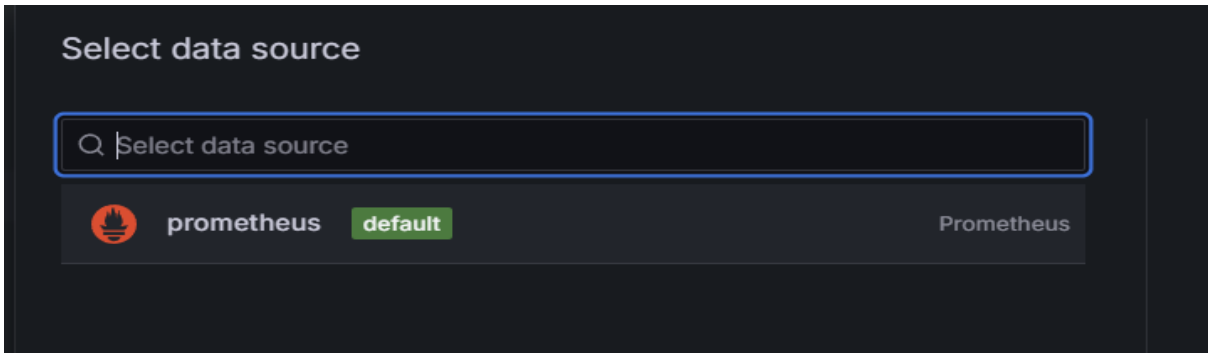
Test the connection and start building a dashboard



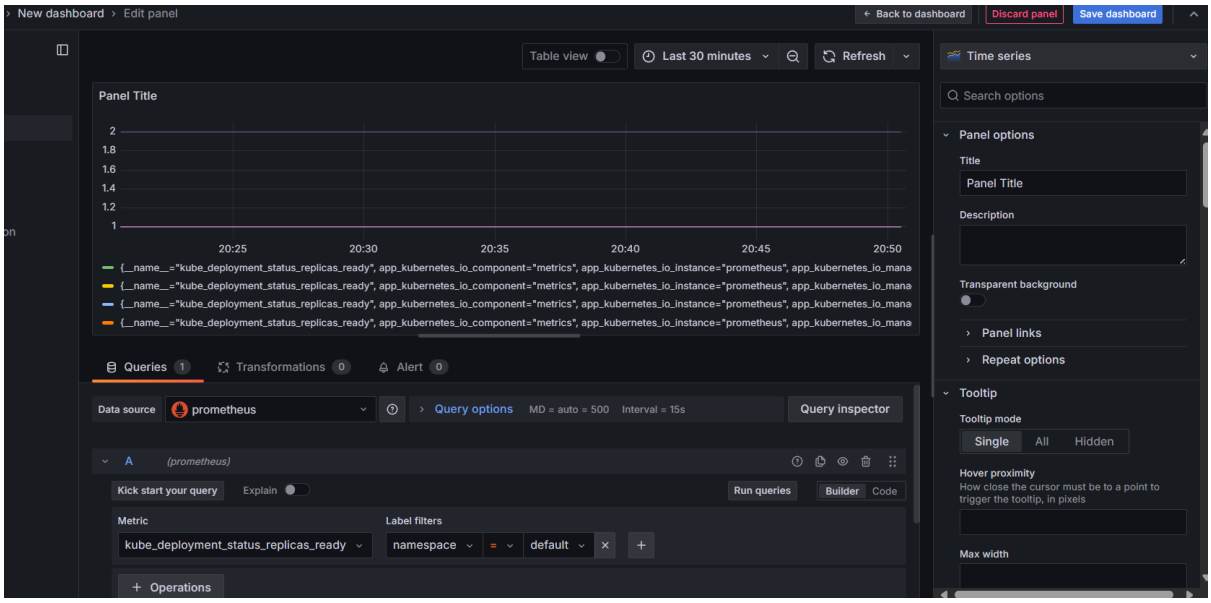
Add visualization



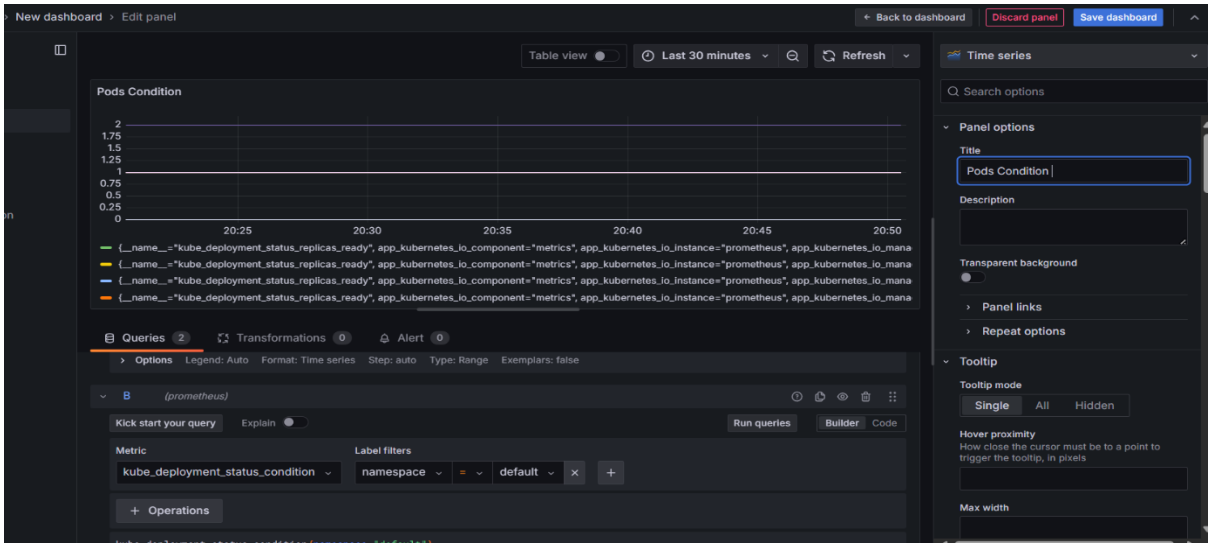
Select prometheus



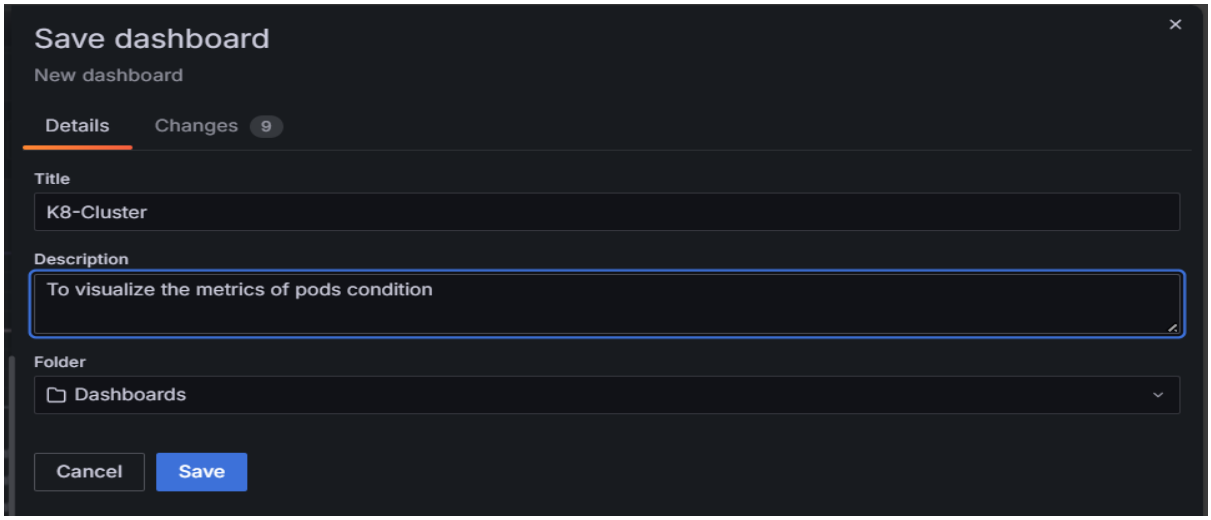
Enter the required query of the metrics to visualize on dashboard



Run query to check the pods condition. Name the Panel

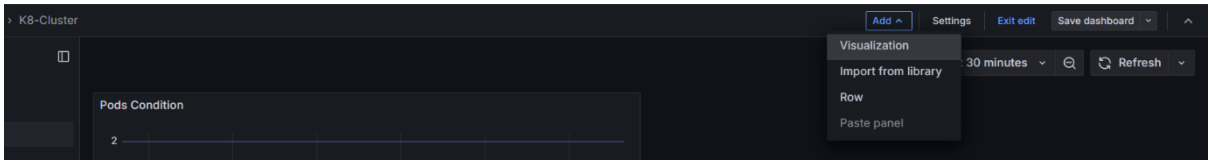


Name the dashboard and Give Description. Save the dashboard

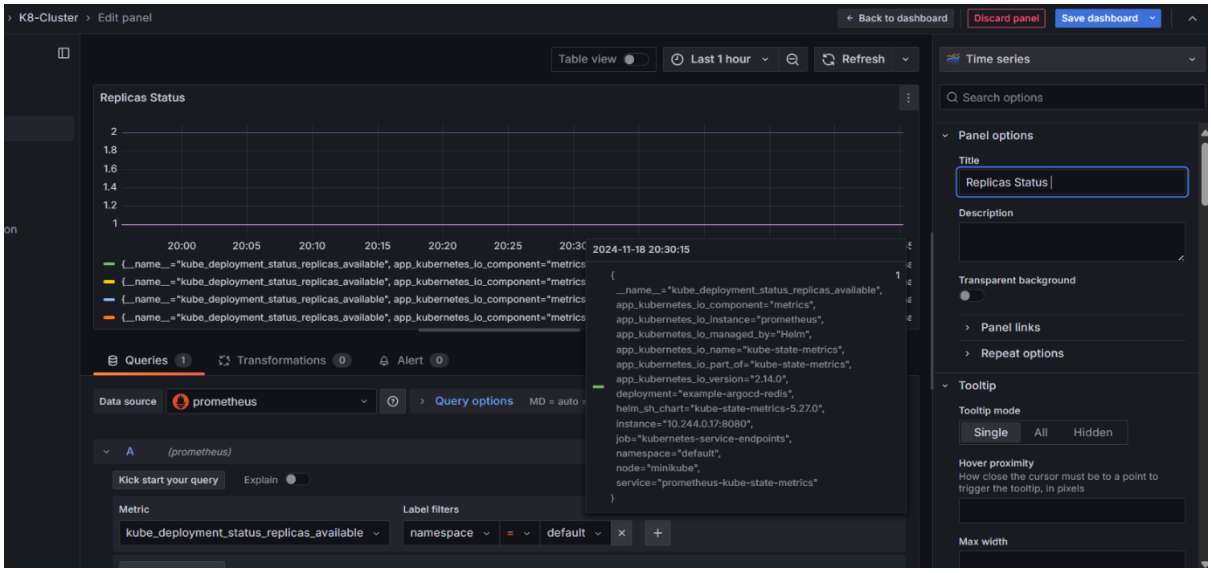


The dashboard got created successfully

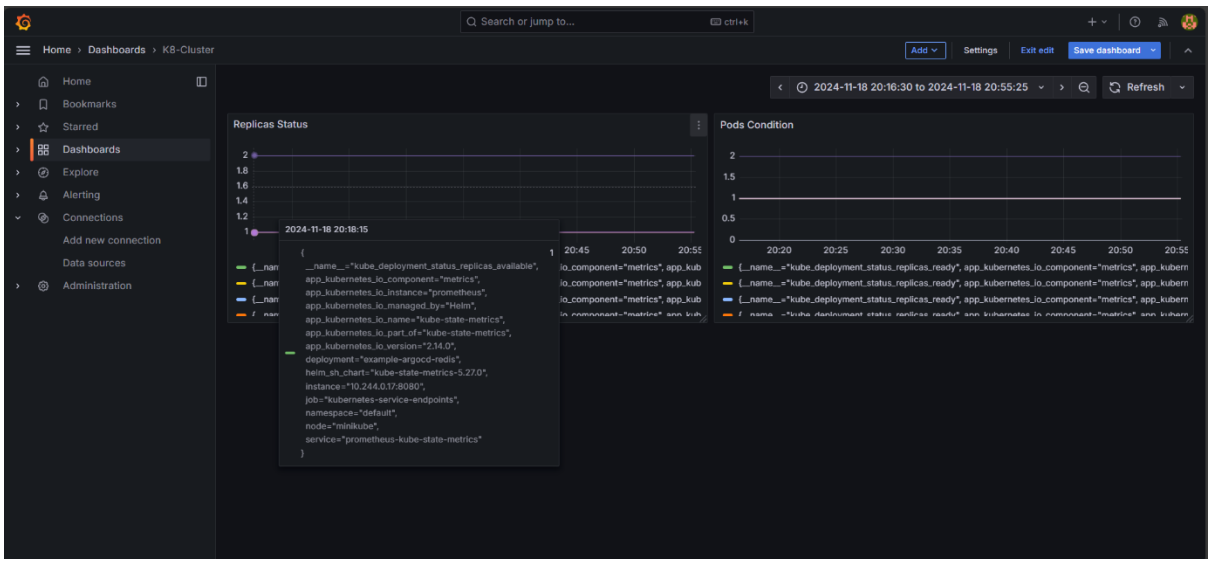
To add more panels, Click on Add and Choose Visualization



Run the query to check replica status of the pods. Save the dashboard



The panels are created and added to the dashboard



Deleting the Infrastructure:

After successful execution, Delete the infrastructure on AWS on EC2 and terminate the instance and Delete the minikube cluster.

```
punith@punith:~$ minikube delete
🔥  Deleting "minikube" in docker ...
🔥  Deleting container "minikube" ...
🔥  Removing /home/punith/.minikube/machines/minikube ...
💀  Removed all traces of the "minikube" cluster.
punith@punith:~$
```

HAPPY LEARNING!