Introduction-> The Kubernetes Dashboard is a vital tool that simplifies the deployment and management of containerized application. Grafana, a flexible platform for observability, further improves this experience at the same time by providing strong analytics and visualizations. Docker is a platform that facilitates the creation, deployment, and operation of programs through containerization. Code, libraries, and dependencies are all packaged as containers so that they may operate on any machine that supports Docker.

Task 1: Deploy and access the Kubernetes Dashboard and a Web Application Component**Task 1 (A)**->Deploy 'Kubernetes Dashboard' on the provided VM with CLI and access/login the Dashboard.

- Dashboard will be available on the following link we need to extract it.
- To extract the link we need to run the command

  Command-> kubectl apply -f (and the link)- https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml

```
ernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
student@edge:~/Documents$ kubectl apply -f Task1Dashboard.yaml
serviceaccount/user created
student@edge:~/Documents$ nano ServiceBinding.yaml
student@edge:~/Documents$ kubectl apply -f ServiceBinding.yaml
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
student@edge:~/Documents$ nano SToken.yaml
student@edge:~/Documents$ kubectl apply -f SToken.yaml
secret/admin-user created
student@edge:~/Documents$ kubectl get secret admin-user -n kubernetes-dashboard
-o jsonpath={".data.token"} | base64 -d
```

Fig 1.1

- **Task 1(B)->** Creating a Service Account named admin-user in the Kubernetes-dashboard namespace involves using Kubernetes configuration file.
- Running command with kubectl tool and can be done using a .yaml file to define the service account.

  Command to create a .yaml file->nano dashboard.yaml(filename).yaml figure 1.2 shows the same

  Code Snippet->

```
student@edge:~$ cat dashborad.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```

Fig 1.2

- **Task 1(C)->**Managing access in Kubernetes dashboard, with default role like cluster -admin are often preconfigured in the cluster.
- This role has some set of permission that provide to access.
- ClusterRoleBinding need to be created to grant some sort of permission to service account manually if the service account does not have the permissions.

Command to create ClusterRolebinding file manually nano dashboard1(filename).yaml

Code Snippet->

```
student@edge:~$ cat dashboard1.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Fig 1.3

- Figure 1.3 shows the code to create cluster role binding

- Managing access controls in Kubernetes, certain default roles like **cluster-admin** are often preconfigured in the cluster. These roles come with a set of permissions that provide extensive access across the entire cluster.
- **Task 1(D)->**To access the Kubernetes dashboard we need to access the create a token so for this there are two ways.
- (i) Temporary token using command-> kubectl -n kubernetes-dashboard create token admin-user one time access the token next time again we need to generate new token to access the dashboard.
- (ii) Create a token with the secret which bound the service account and the token will be saved in the Secret
- After token is created we need to enable the port 8081 on which the Kubernetes dashboard will be accessible using this command->kubectl proxy .

```
student@edge:~$ kubectl get secret admin-user -n kubernetes-dashboard -o jsonpath={".data.token"} | base64 -d
eyJhbGciOiJSUzI1NiIsImtpZCI6IlBPNXZwUHFVMGZ5TTJaeklid2Z4Tkc3M21TVGY5b2l2Vm9iaHo3LXNrV0UifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiIzN2E3MjQwZS0zNzkyLTQyMWMtYmQzYy1jMGEwZjMyNDFkNjIiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZXJuZXRlcy1kYXNoYm9hcmQ6YWRtaW4tdXNlciJ9.VxgKIty_nwSibtYL9A5Fnj5SC00dhjydt6R83FALss9m7m01dxRMUIr9pDhyVNMUh1w4EolLWWs1vA5K_rmFTDD5YoMEH6QYp_1FQl5Rt63Z2tGEmowCNfgBAbcPhuUQ4C8dxkuIwL1ufB10I0_Ard1SFtjSQ7AgC014fwdJ9NBM2hR8AIT3LU3CwEhepJVatRG91GyYAOjvgPC0zAjOltLjpxTQAWILFBZObFfSaT_P2-h7DymIgmxn-Fyax_lAIgyqNQ5xKSrs3aNPlqzaOO2_ps1gT7RHpORKdlYUcJmyXyLT4v72dRI0B1v3s6IDdVBMERu7CvA--6ifIbJXtQstudent@edge
:~$
student@edge:~$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Fig 1.4

- Figure 1.4 shows how token is generated and 8081 port in enables to access the dashboard .
- Figure 1.5 shows we need to provide the token

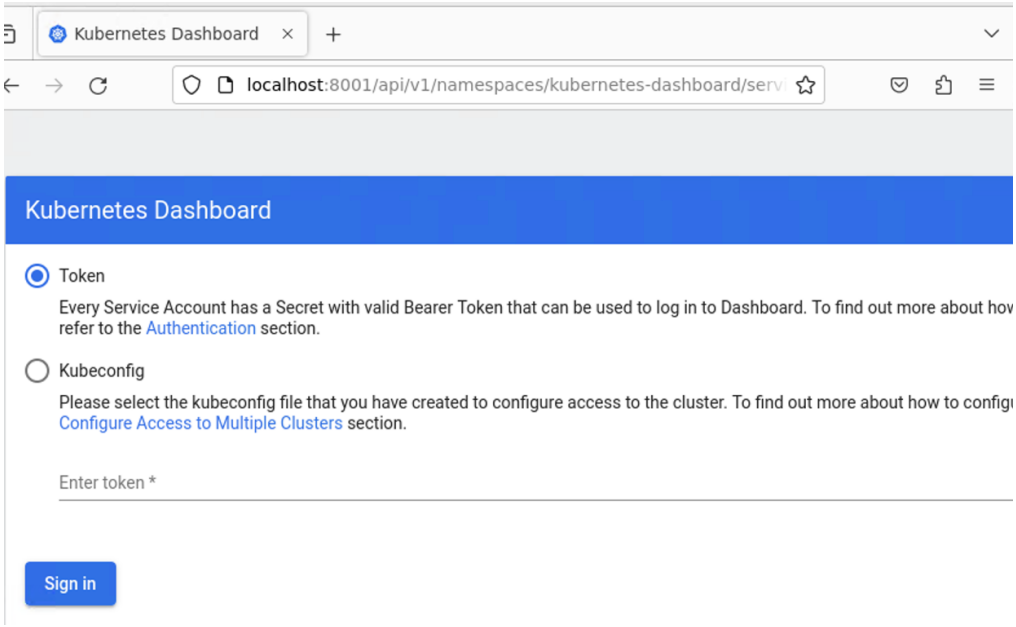


Fig 1.4

- **Task 1(E)->**Deploy an instance of the Docker image "nclcloudcomputing/javabenchmark" vai CLI.
- We need to deploy it manually through command line using .yaml file
- Figure 1.5 shows the code to pull the image.

Command ->nano javabenchmark1.yaml to pull the docker image.

Command->kubectl apply -f javabenchmark1.yaml.

Code Snippet->

```
student@edge:~$ cat javabenchmark1.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: javabenchmarkapp-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: javabenchmarkapp
  template:
    metadata:
      labels:
        app: javabenchmarkapp
    spec:
      containers:
        - name: javabenchmarkapp-container
          image: nclcloudcomputing/javabenchmarkapp
          ports:
            - containerPort: 8080
```

Fig 1.5

- **Task 1(F)->**Deploy a NodePort service so that the web app is accessible vai http://localhost:30000/primecheck.

- NordPort is used so that we can access it locally localhost:30000/primecheck

  Command->kubectl apply -f javabenchmarkapp.yaml

```
student@edge:~$ cat javabenchmarkapp.yaml
apiVersion: v1
kind: Service
metadata:
  name: javabenchmarkapp-service
spec:
  type: NodePort
  selector:
    app: javabenchmarkapp
  ports:
    - name: http
      protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 30000
```

Fig 1.6

- Figure 1.7 primecheck is accessible on 30000 port

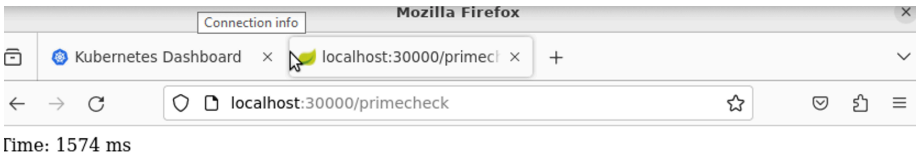


Fig 1.7

Task 2 Deploy the monitoring stack of Kubernetes.

- **Task 2(A)->**Enable observability services from micro8s addon.
- To enable services command is microk8s enable observability.
- Figure 2.2 and 2.3 shows cases to enable observability.
- **Task 2(B)->**Grafana Services has been edited to type NordPort so that we can access it locally and on port 30001 figure 2.4.
- And Grafana can also be access using the cluster IP.
- Figure 2.5 shows all the resources status and running on which port/Cluster IP.
- **Task 2(C)->** Grafana login through **localhost:30001**

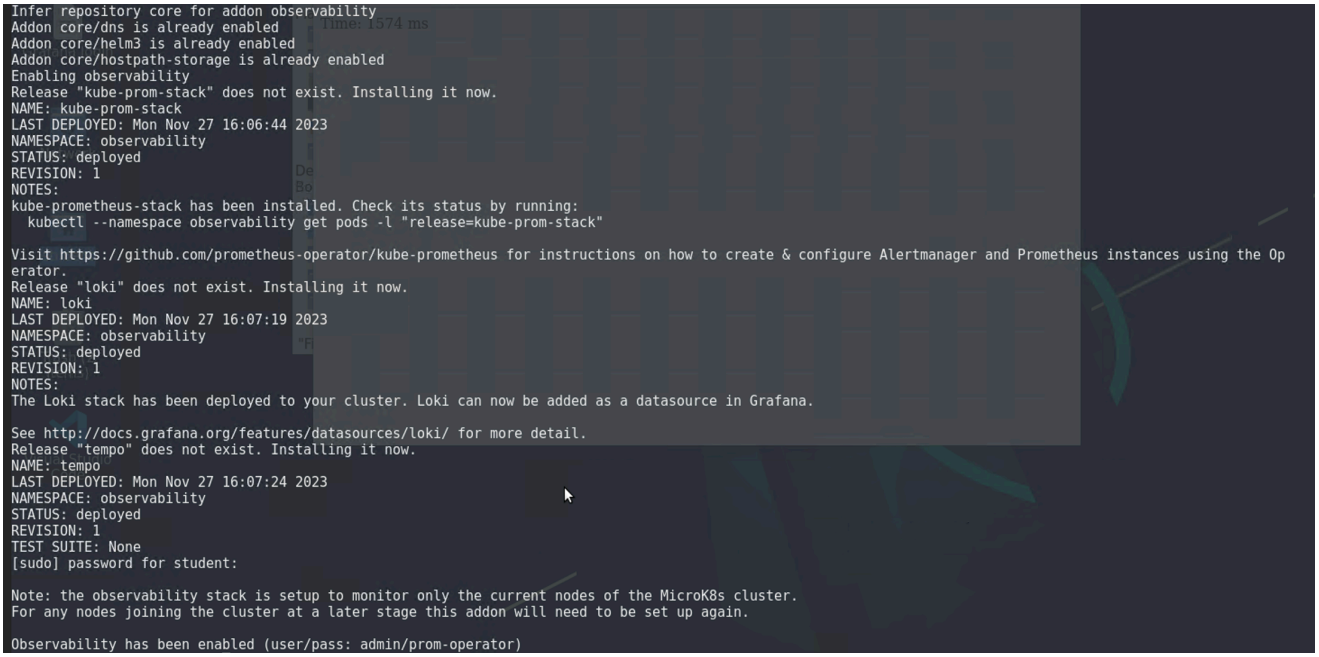


Fig 2.1

Fig 2.2

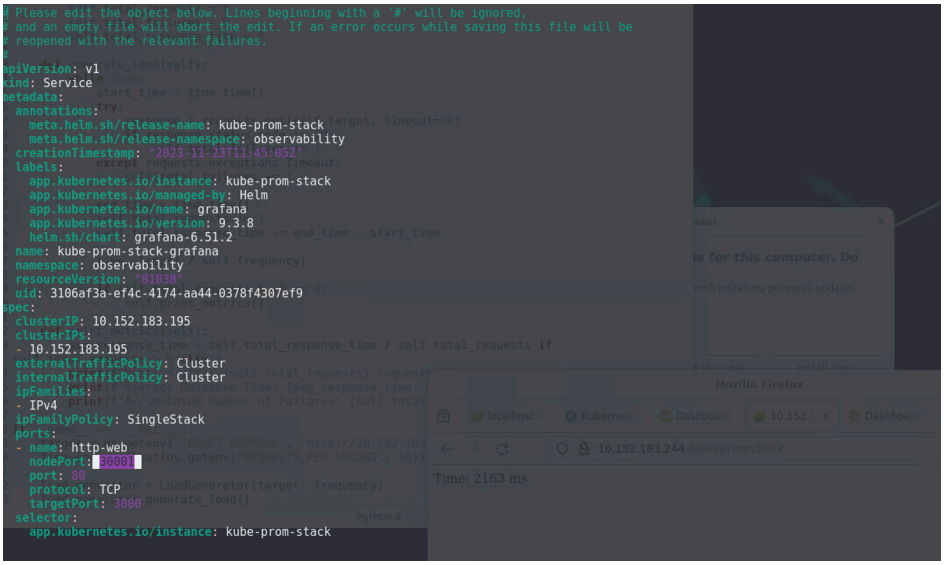

Please edit the object below. Lines beginning with a '#' will be ignored,
and an empty file will abort the edit. If an error occurs while saving this file will be
reopened with the relevant failures.
apiVersion: v1
kind: Service
metadata:
annotations:
meta.helm.sh/release-name: kube-prom-stack
meta.helm.sh/release-namespace: observability
creationTimestamp: "2023-11-23T11:45:05Z"
labels:
app.kubernetes.io/instance: kube-prom-stack
app.kubernetes.io/managed-by: Helm
app.kubernetes.io/name: grafana
app.kubernetes.io/version: 9.3.8
helm.sh/chart: grafana-6.51.2
name: kube-prom-stack-grafana
namespace: observability
resourceVersion: "31035"
uid: 3106af3a-ef4c-4174-aa44-0378f4307ef9
spec:
clusterIP: 10.152.183.195
clusterIPs:
- 10.152.183.195
externalTrafficPolicy: Cluster
internalTrafficPolicy: Cluster
ipFamilies:
- IPv4
ipFamilyPolicy: SingleStack
ports:
- name: http-web
nodePort: 30001
port: 80
protocol: TCP
targetPort: 3000
selector:
app.kubernetes.io/instance: kube-prom-stack
Mozilla Firefox
10.152.183.244:8080/primecheck
Time: 2163 ms

Fig 2.3

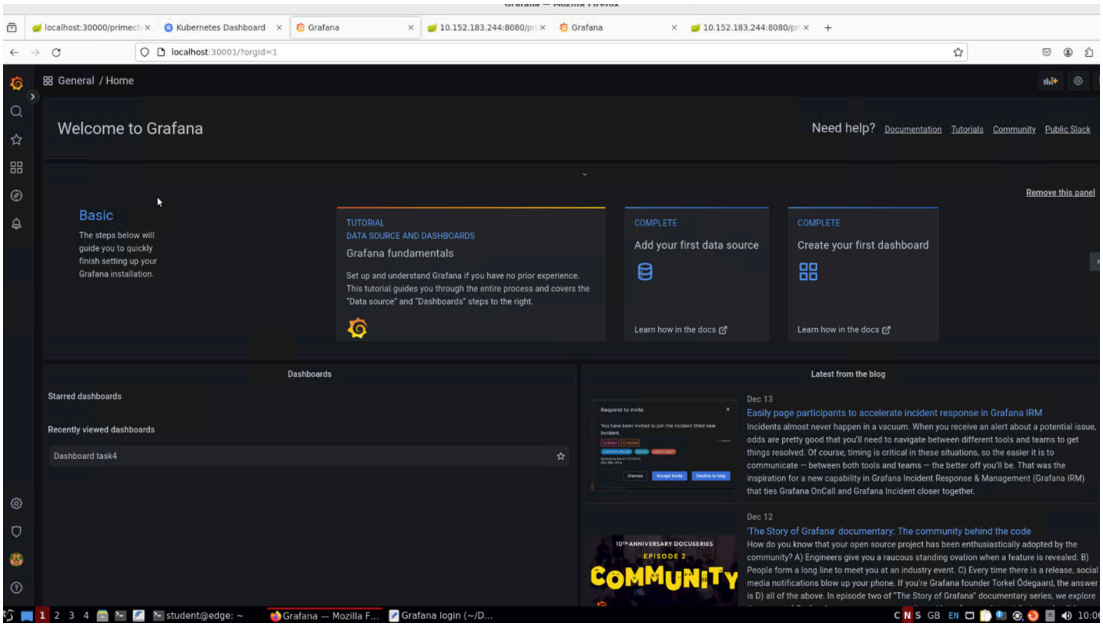
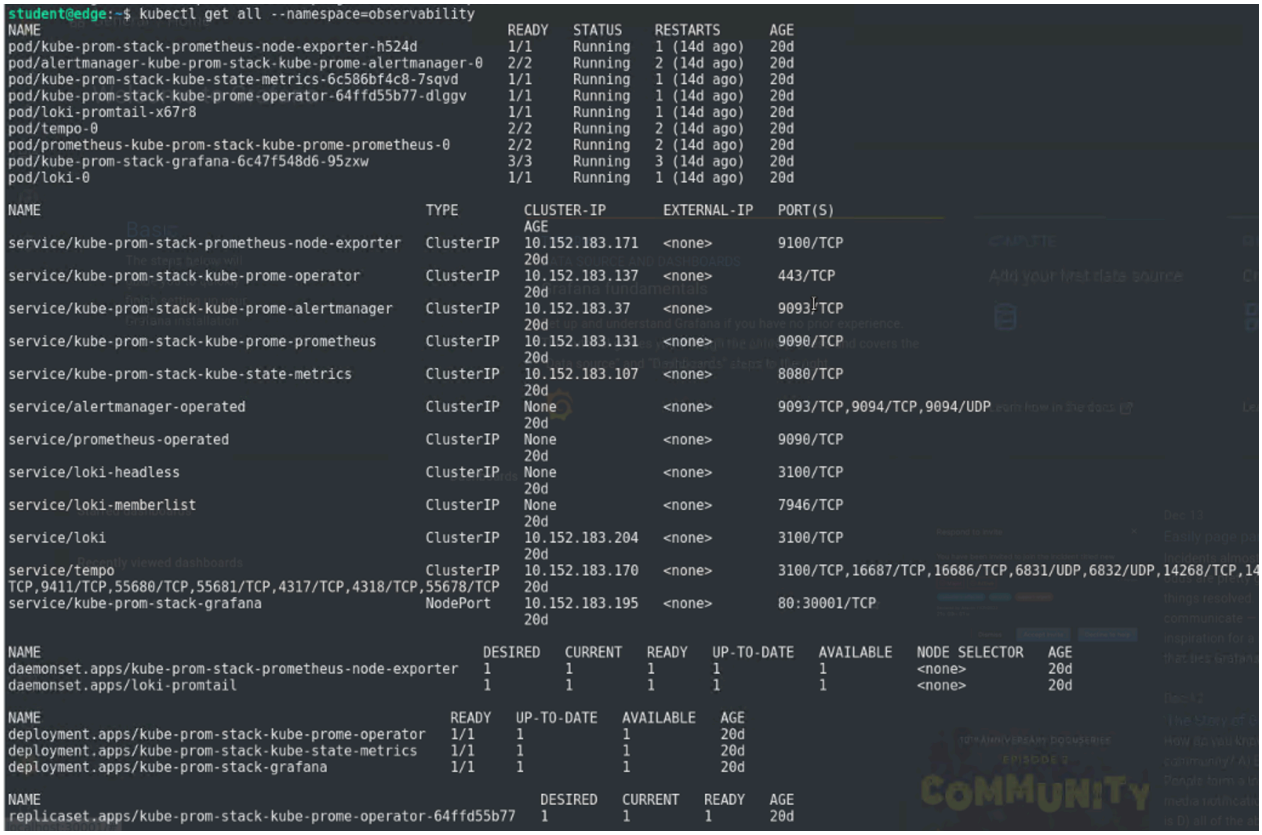

localhost:30000/primech
Kubernetes Dashboard
Grafana
10.152.183.244:8080/pri
localhost:30001/?orgId=1
General / Home
Welcome to Grafana
Need help?
Documentation
Tutorials
Community
Public Slack
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
COMPLETE
Add your first data source
Learn how in the docs
COMPLETE
Create your first dashboard
Learn how in the docs
Dashboards
Starred dashboards
Recently viewed dashboards
Dashboard task4
Latest from the blog
Dec 13
Easily page participants to accelerate incident response in Grafana IRM
Incidents almost never happen in a vacuum. When you receive an alert about a potential issue, odds are pretty good that you'll need to navigate between different tools and teams to get things resolved. Of course, timing is critical in these situations, so the easier it is to communicate -- between both tools and teams -- the better off you'll be. That was the inspiration for a new capability in Grafana Incident Response & Management (Grafana IRM) that ties Grafana OnCall and Grafana Incident closer together.
Dec 12
'The Story of Grafana' documentary: The community behind the code
How do you know that your open source project has been enthusiastically adopted by the community? A) Engineers give you a raucous standing ovation when a feature is revealed. B) People form a long line to meet you at an industry event. C) Every time there is a release, social media notifications blow up your phone. If you're Grafana founder Torkel Ödegaard, the answer is D) all of the above. In episode two of "The Story of Grafana" documentary series, we explore
10TH ANNIVERSARY DOCUSERIES
EPISODE 2
COMMUNITY
student@edge: ~
Grafana — Mozilla F...
Grafana login (~/D...

Fig 2.4



```
student@edge:~$ kubectl get all --namespace=observability
NAME                                                     READY   STATUS    RESTARTS      AGE
pod/kube-prom-stack-prometheus-node-exporter-h524d       1/1     Running   1 (14d ago)   20d
pod/alertmanager-kube-prom-stack-kube-prome-alertmanager-0   2/2     Running   2 (14d ago)   20d
pod/kube-prom-stack-kube-state-metrics-6c586bf4c8-7sqvd  1/1     Running   1 (14d ago)   20d
pod/kube-prom-stack-kube-prome-operator-64ffd55b77-dlggv 1/1     Running   1 (14d ago)   20d
pod/loki-promtail-x67r8                                  1/1     Running   1 (14d ago)   20d
pod/tempo-0                                              2/2     Running   2 (14d ago)   20d
pod/prometheus-kube-prom-stack-kube-prome-prometheus-0   2/2     Running   2 (14d ago)   20d
pod/kube-prom-stack-grafana-6c47f548d6-95zxw             3/3     Running   3 (14d ago)   20d
pod/loki-0                                               1/1     Running   1 (14d ago)   20d

NAME                                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
                                                  AGE
service/kube-prom-stack-prometheus-node-exporter  ClusterIP   10.152.183.171   <none>        9100/TCP
                                                  20d
service/kube-prom-stack-kube-prome-operator       ClusterIP   10.152.183.137   <none>        443/TCP
                                                  20d
service/kube-prom-stack-kube-prome-alertmanager   ClusterIP   10.152.183.37    <none>        9093/TCP
                                                  20d
service/kube-prom-stack-kube-prome-prometheus     ClusterIP   10.152.183.131   <none>        9090/TCP
                                                  20d
service/kube-prom-stack-kube-state-metrics        ClusterIP   10.152.183.107   <none>        8080/TCP
                                                  20d
service/alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP
                                                  20d
service/prometheus-operated                       ClusterIP   None             <none>        9090/TCP
                                                  20d
service/loki-headless                             ClusterIP   None             <none>        3100/TCP
                                                  20d
service/loki-memberlist                           ClusterIP   None             <none>        7946/TCP
                                                  20d
service/loki                                      ClusterIP   10.152.183.204   <none>        3100/TCP
                                                  20d
service/tempo                                     ClusterIP   10.152.183.170   <none>        3100/TCP,16687/TCP,16686/TCP,6831/UDP,6832/UDP,14268/TCP,14
TCP,9411/TCP,55680/TCP,55681/TCP,4317/TCP,4318/TCP,55678/TCP  20d
service/kube-prom-stack-grafana                   NodePort    10.152.183.195   <none>        80:30001/TCP
                                                  20d

NAME                                                 DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/kube-prom-stack-prometheus-node-exporter   1         1         1       1            1           <none>          20d
daemonset.apps/loki-promtail                         1         1         1       1            1           <none>          20d

NAME                                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/kube-prom-stack-kube-prome-operator   1/1     1            1           20d
deployment.apps/kube-prom-stack-kube-state-metrics    1/1     1            1           20d
deployment.apps/kube-prom-stack-grafana               1/1     1            1           20d

NAME                                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/kube-prom-stack-kube-prome-operator-64ffd55b77   1         1         1       20d
```

Fig 2.5

Task 3 Load Generator

- **Task 3(A)->**Write a load generator with following specification.

  (a) Accepts two configurable values either via a config file or environment variables. target (The address for the load generation) and frequency (Request per second)

  (b) Generate web request to the target at the specified frequency

  (c) Collect 2 types of metrics. Average response time and accumulated number of failures

  (d) Request should timeout if it takes more than 10 seconds. Counted as failures

  (e) Test results need to be printed to the console

  (f) There are no requirements in programming language

  Create a python file using Command-> nano loadgenerator(filename).py

  Code Snippet->

```
student@edge:~$ cat loadgenerator.py
import requests
import time
import os

class LoadGenerator:
    def __init__(self, target, frequency):
        self.target = target
        self.frequency = frequency
        self.total_requests = 0
        self.total_failures = 0
        self.total_response_time = 0

    def generate_load(self):
        while True:
            start_time = time.time()
            try:
                response = requests.get(self.target, timeout=10)
                if response.status_code != 200:
                    self.total_failures += 1
            except requests.exceptions.Timeout:
                self.total_failures += 1

            end_time = time.time()
            self.total_requests += 1
            self.total_response_time += end_time - start_time

            time.sleep(1 / self.frequency)

            if self.total_requests % 10 == 0:  # Print metrics every 10 requests
                self.print_metrics()

    def print_metrics(self):
        avg_response_time = self.total_response_time / self.total_requests if self.total_requests > 0 else 0
        print(f"Metrics after {self.total_requests} requests:")
        print(f"Average Response Time: {avg_response_time:.4f} seconds")
        print(f"Accumulated Number of Failures: {self.total_failures}\n")

if __name__ == "__main__":
    target = os.getenv("TARGET_ADDRESS", "http://10.152.183.244:8080/primecheck")  # Replace with default target
    frequency = float(os.getenv("REQUESTS_PER_SECOND", 10))  # Replace with default frequency

    load_generator = LoadGenerator(target, frequency)
    load_generator.generate_load()
```

Fig 3.1

- Figure 3.2 shows the response after running the **loadgenerator** python file manually through command line using command -> kubectl apply -f loadgenerator.py(filename) .

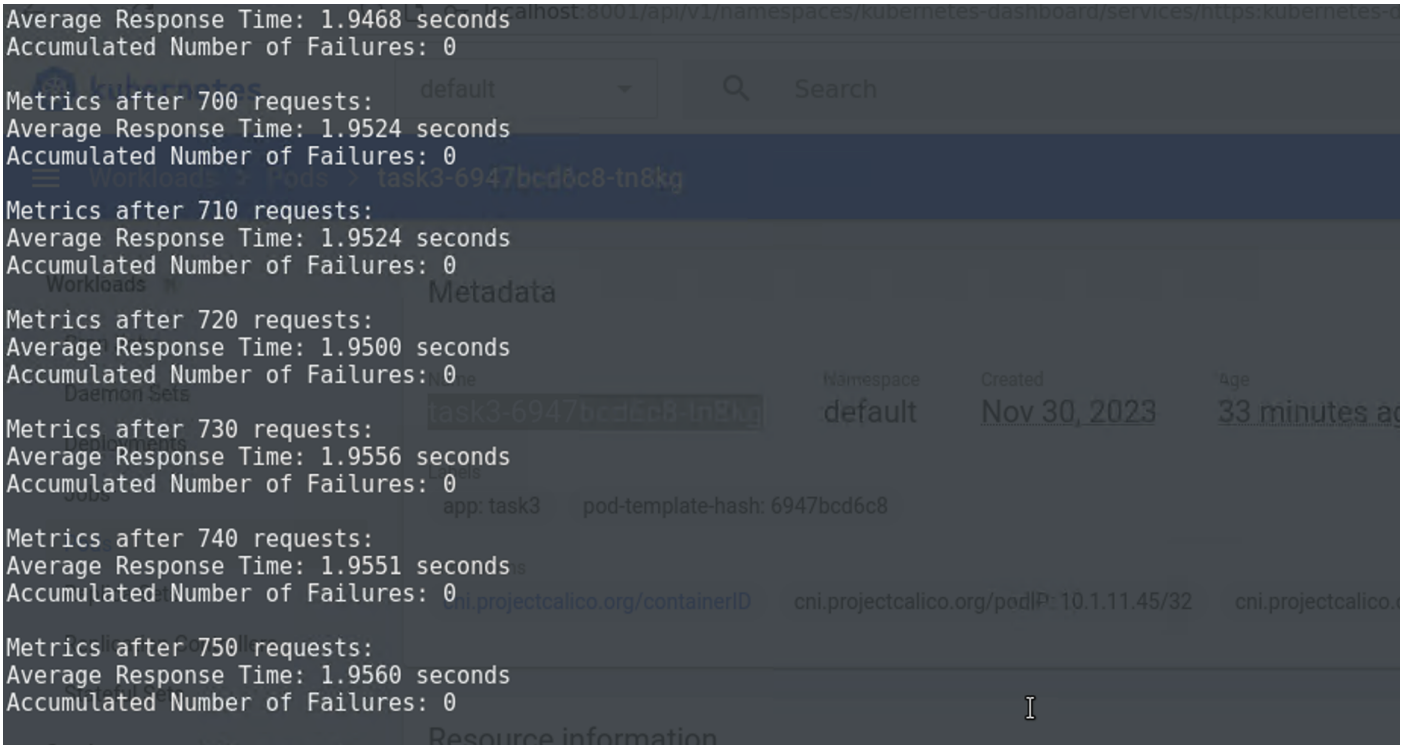


Fig 3.2

- **Task 3(B)**->Once the **loadgenerator** python file is done, next step is to create a standalone docker image of the same with name load-generator.

  Command->docker build -t load-generator .

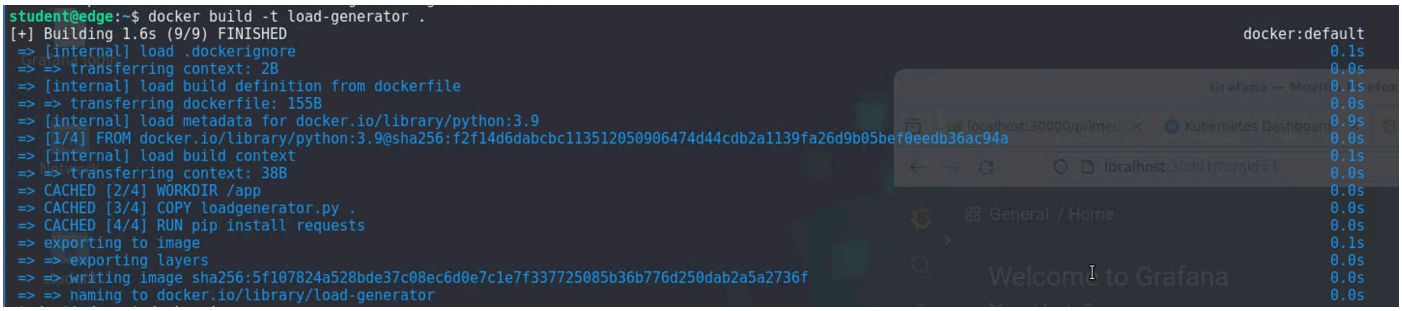


Fig 3.3

- Figure 3.3 shows docker image is created with name load-generator.
- Docker image **load-generator** need to pushed on **local-registry on 32000 port**.

  Command ->docker run -d -p 32000:5000 –restart=always -name registry registry:latest

- After that we need to tag the image.

  Command->docker tag load-generator local:host:32000/load-generator

- Push image to local repository

  Command->docker push localhost:32000/load-generator

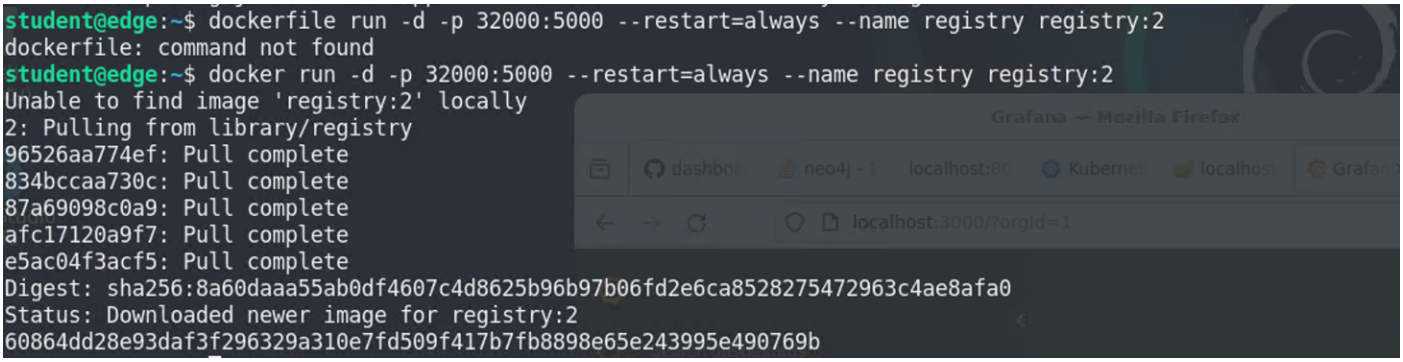


Fig 3.4

- Figure 3.4 docker image **load-generator** is pushed to local registry on 32000

Task 4 Monitor Benchmarking Results

- **Task 4(A)->**Deploy load-generator service created in task 3 through command line.
- First .yaml file need to be created using command ->nano task3.yaml(filename) .

Code Snippet->

```
student@edge:~$ cat task3.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: task3
spec:
  replicas: 1
  selector:
    matchLabels:
      app: task3
  template:
    metadata:
      labels:
        app: task3
    spec:
      containers:
      - name: task3
        image: localhost:32000/load-generator
        ports:
         - containerPort: 8080
        env:
        - name: target
          value: "http://10.152.183.244:8080/primecheck"
        - name: frequency
          value: "10.0"
        command: ["python", "loadgenerator.py"]

student@edge:~$ kubectl apply -f task3.yaml
deployment.apps/task3 configured
```

Fig 4.1

- Figure 4.1 shows that service has been deployed with image localhost:32000/load-generator
- **Task 4(B)**-> Create a new dashboard on Grafana and add 2 new panels which should contain queries of CPU/memory usage of the web application
- First panel with ->CPU usage.
- Figure 4.3 shows the memory panel graph

Metric ->container_memory_usuage_bytes

Label Filter ->container-> javabenchmarkapp-conatiner .

- Memory used in bytes on Y-axis and time on X-axis.
- Figure 4.4 shows the CPU panel graph with

Metric ->container_cpu_usuage_seconds_tools

Label Filter ->container -> javabenchmarkapp-conatiner .

- Figure 4.2 shows the both the panel and from that we can understand the memory and CPU usage.
- Figure 4.6 shows the logs

Fig 4.2

Fig 4.3

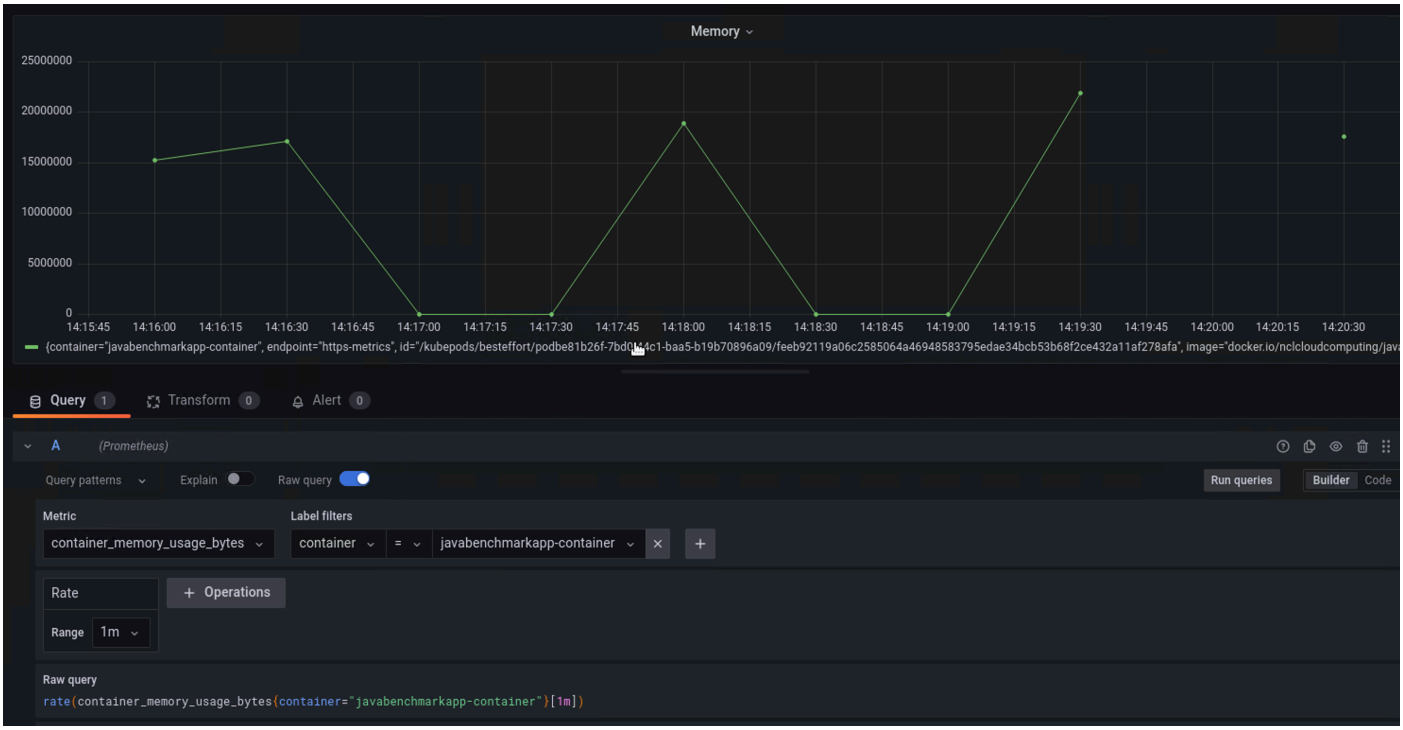

Memory
25000000
20000000
15000000
10000000
5000000
0
14:15:45 14:16:00 14:16:15 14:16:30 14:16:45 14:17:00 14:17:15 14:17:30 14:17:45 14:18:00 14:18:15 14:18:30 14:18:45 14:19:00 14:19:15 14:19:30 14:19:45 14:20:00 14:20:15 14:20:30
{container="javabenchmarkapp-container", endpoint="https-metrics", id="/kubepods/besteffort/podbe81b26f-7bd0-44c1-baa5-b19b70896a09/feeb92119a06c2585064a46948583795edae34bcb53b68f2ce432a11af278afa", image="docker.io/nclcloudcomputing/
Query 1
Transform 0
Alert 0
A
(Prometheus)
Query patterns
Explain
Raw query
Run queries
Builder
Code
Metric
container_memory_usage_bytes
Label filters
container
=
javabenchmarkapp-container
Rate
+ Operations
Range
1m
Raw query
rate(container_memory_usage_bytes{container="javabenchmarkapp-container"}[1m])

Fig 4.4

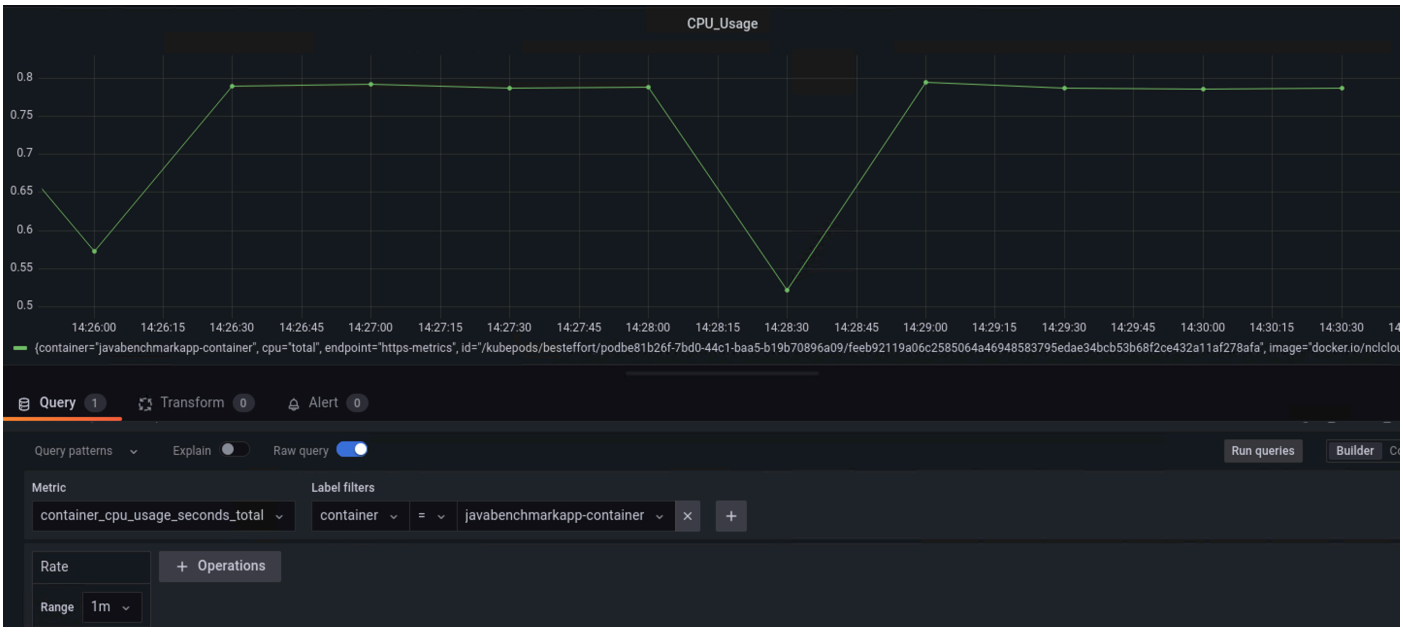

CPU_Usage
0.8
0.75
0.7
0.65
0.6
0.55
0.5
14:26:00 14:26:15 14:26:30 14:26:45 14:27:00 14:27:15 14:27:30 14:27:45 14:28:00 14:28:15 14:28:30 14:28:45 14:29:00 14:29:15 14:29:30 14:29:45 14:30:00 14:30:15 14:30:30
{container="javabenchmarkapp-container", cpu="total", endpoint="https-metrics", id="/kubepods/besteffort/podbe81b26f-7bd0-44c1-baa5-b19b70896a09/feeb92119a06c2585064a46948583795edae34bcb53b68f2ce432a11af278afa", image="docker.io/nclclou
Query 1
Transform 0
Alert 0
Query patterns
Explain
Raw query
Run queries
Builder
Metric
container_cpu_usage_seconds_total
Label filters
container
=
javabenchmarkapp-container
Rate
+ Operations
Range
1m

Fig 4.5

Logs from task3 in task3-6947bcd...

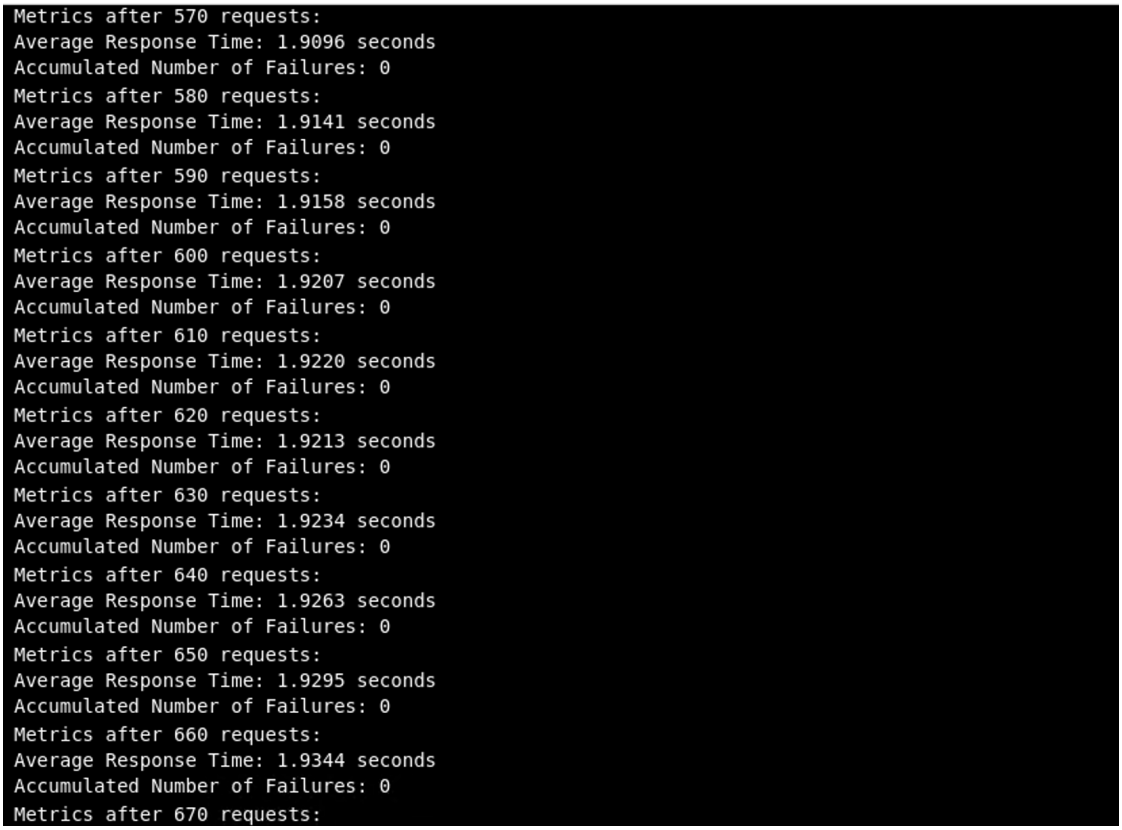


Fig 4.6

Conclusion-> For controlling Kubernetes clusters, Kubernetes Dashboard offers an intuitive user interface. Docker uses containerization to make application deployment easier while maintaining consistency between environments. Grafana provides robust system metrics monitoring and visualization features that improve understanding of application performance and infrastructure health. When combined, they improve efficiency and dependability in contemporary software development and operations by streamlining management, deployment, and monitoring.