

# Kubernetes-Based Application Hosting and Monitoring System

## Introduction

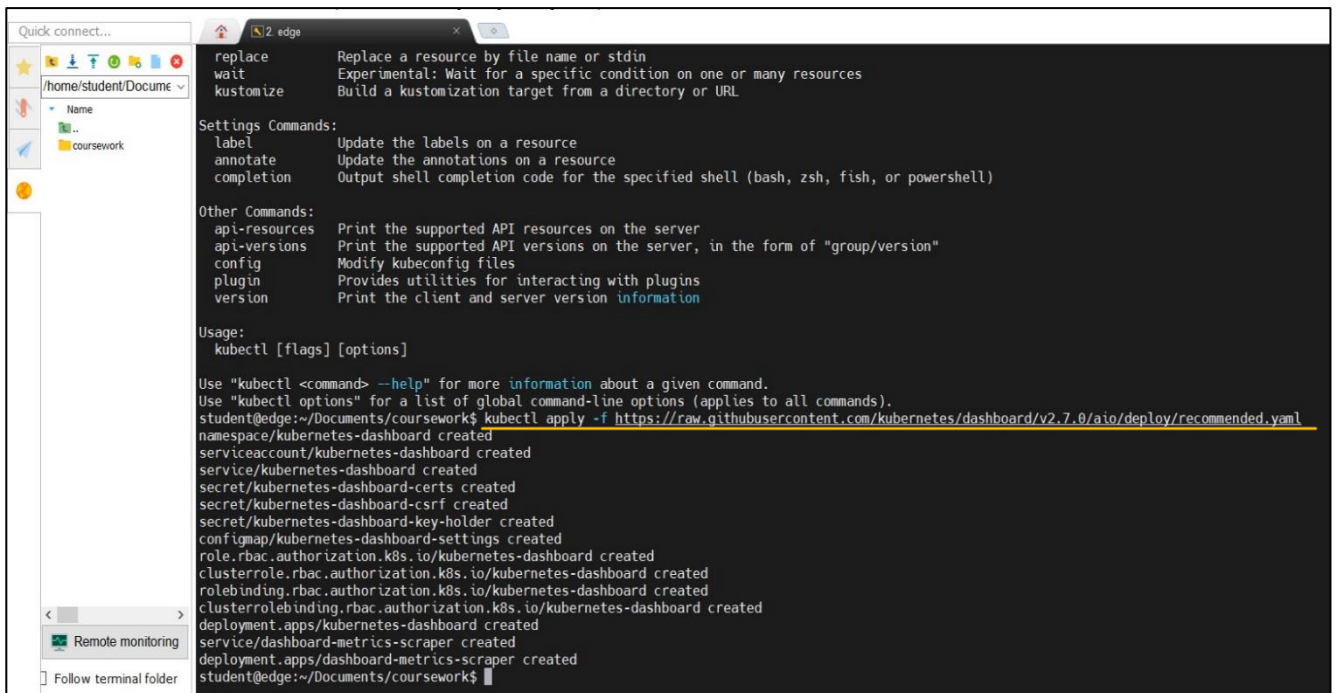
This project is based on understanding basic concepts of Kubernetes along with deploying a Kubernetes based application hosting environment. Understanding docker and learn how to build, push and pull image from docker hub. To understand about monitoring stack of Kubernetes such as Grafana, Prometheus, Metrics server and to deploy and use them.

## Task 1: Deploy and access the Kubernetes Dashboard and a Web Application Component

### 1.1 Deploying Kubernetes Dashboard on edge virtual machine

Kubernetes is basically an open-source platform which can be used to deploy containerized applications. In order to deploy Kubernetes dashboard, we run the below command into CLI of edge virtual machine.

*"kubectl apply -f <https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml>"*



```
Quick connect...
/home/student/Documents/coursework
Name
coursework

replace      Replace a resource by file name or stdin
wait         Experimental: Wait for a specific condition on one or many resources
customize    Build a customization target from a directory or URL

Settings Commands:
label        Update the labels on a resource
annotate    Update the annotations on a resource
completion   Output shell completion code for the specified shell (bash, zsh, fish, or powershell)

Other Commands:
api-resources Print the supported API resources on the server
api-versions  Print the supported API versions on the server, in the form of "group/version"
config        Modify kubeconfig files
plugin        Provides utilities for interacting with plugins
version       Print the client and server version information

Usage:
kubectl [flags] [options]

Use "kubectl <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all commands).
student@edge:~/Documents/coursework$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/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/coursework$
```

Now in order to login/access the dashboard we need to create a user using the service account mechanism of Kubernetes. Service account allows processes running in the pods to authenticate and interact with Kubernetes API.

To create a service account in Kubernetes we can specify yaml manifest file. In this file we first specify the version of Kubernetes API followed by defining the kind of Kubernetes resource being created, which is a service account. Next, we specify metadata such as name of service account which we give as admin-user and namespace in which service account will be created i.e kubernetes-dashboard.

```
1 apiVersion: v1
2 kind: ServiceAccount
3 metadata:
4   name: admin-user
5   namespace: kubernetes-dashboard
6
7
```

To execute this, run the below command in CLI

*“kubectl apply -f serviceaccount.yaml”*

```
• MobaXterm Personal Edition v22.1 •
(SSH client, X server and network tools)

> SSH session to student@edge.local
• Direct SSH : ✓
• SSH compression : ✓
• SSH-browser : ✓
• X11-forwarding : ✓ (remote display is forwarded through SSH)

> For more info, ctrl+click on help or visit our website.

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-88-generic x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

Expanded Security Maintenance for Applications is not enabled.

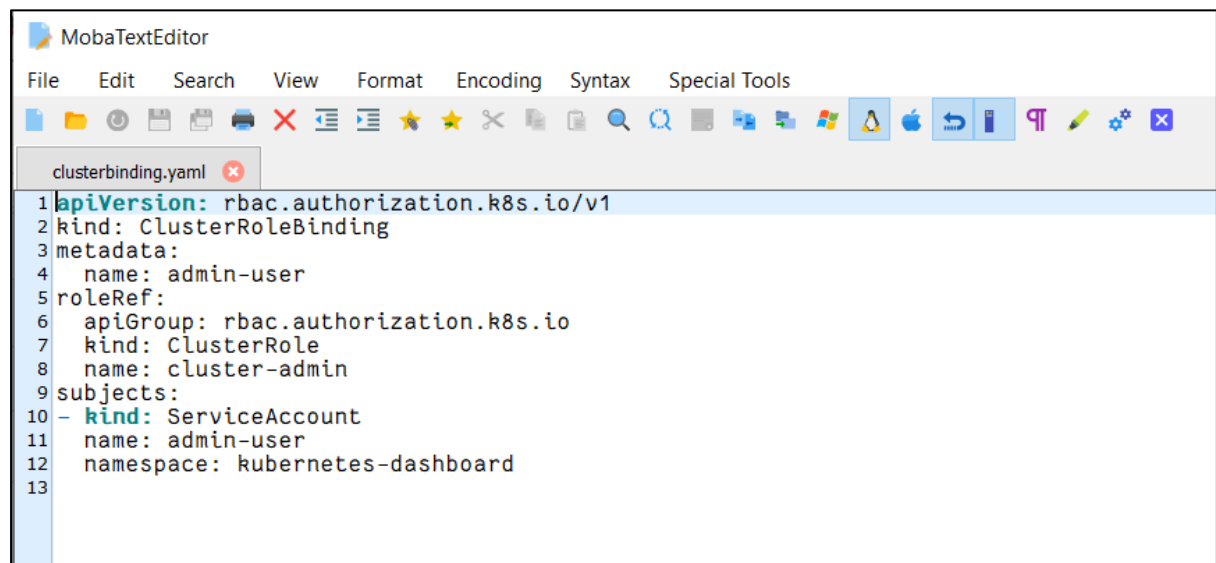
0 updates can be applied immediately.

9 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

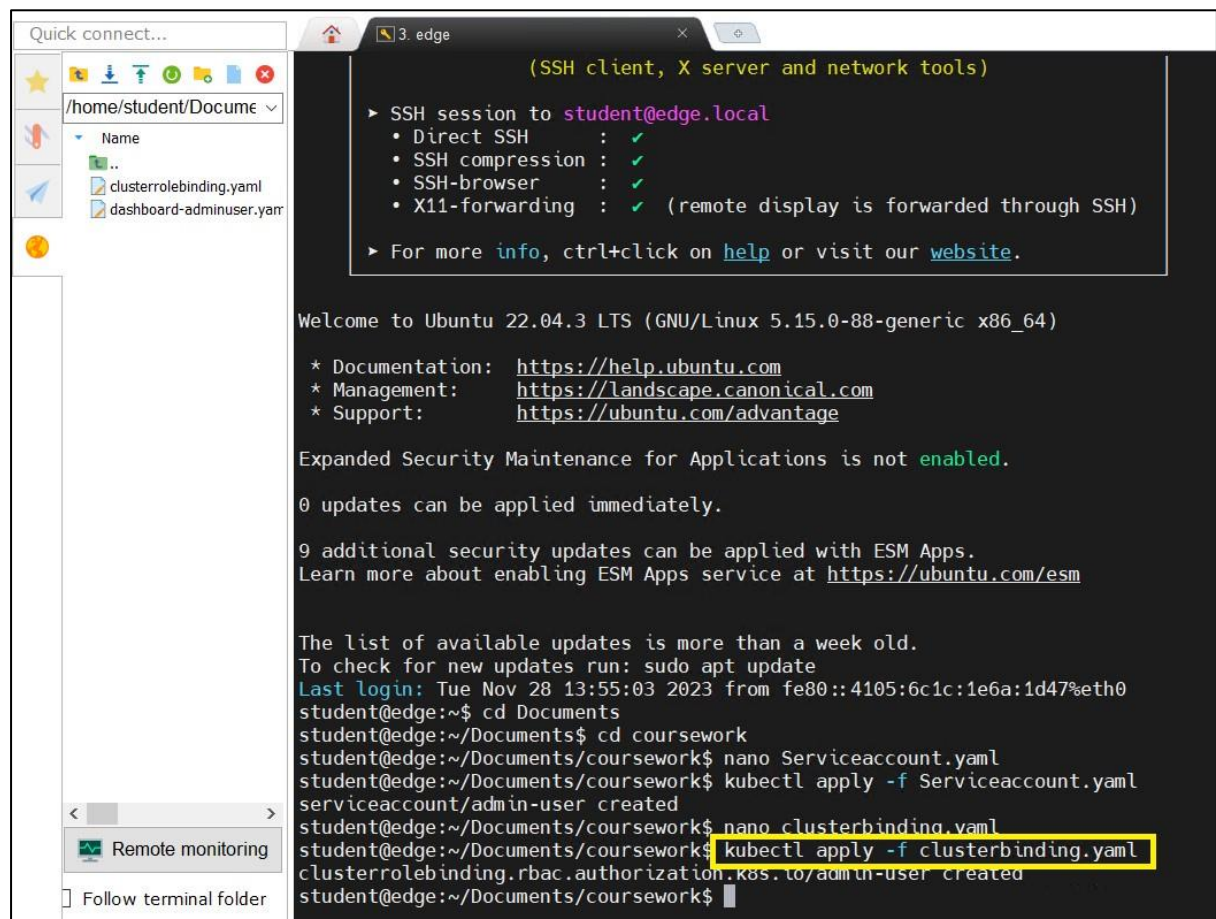
The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Last login: Tue Nov 28 13:55:03 2023 from fe80::4105:6c1c:1e6a:1d47%eth0
student@edge:~$ cd Documents
student@edge:~/Documents$ cd coursework
student@edge:~/Documents/coursework$ nano Serviceaccount.yaml
student@edge:~/Documents/coursework$ kubectl apply -f Serviceaccount.yaml
serviceaccount/admin-user created
student@edge:~/Documents/coursework$
```

Next, we create a cluster role binding where we bind the cluster-admin role to the admin user service account. For this we create the yaml manifest file with the name “clusterbinding.yaml” and apply this manifest file by running the following command,

*“kubectl apply -f clusterbinding.yaml”*



```
1 apiVersion: rbac.authorization.k8s.io/v1
2 kind: ClusterRoleBinding
3 metadata:
4   name: admin-user
5 roleRef:
6   apiGroup: rbac.authorization.k8s.io
7   kind: ClusterRole
8   name: cluster-admin
9 subjects:
10 - kind: ServiceAccount
11   name: admin-user
12   namespace: kubernetes-dashboard
13
```



```
Quick connect...
/home/student/Documents
Name
..
clusterrolebinding.yaml
dashboard-adminuser.yaml

(SSH client, X server and network tools)
> SSH session to student@edge.local
  • Direct SSH : ✓
  • SSH compression : ✓
  • SSH-browser : ✓
  • X11-forwarding : ✓ (remote display is forwarded through SSH)
> For more info, ctrl+click on help or visit our website.

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-88-generic x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

Expanded Security Maintenance for Applications is not enabled.

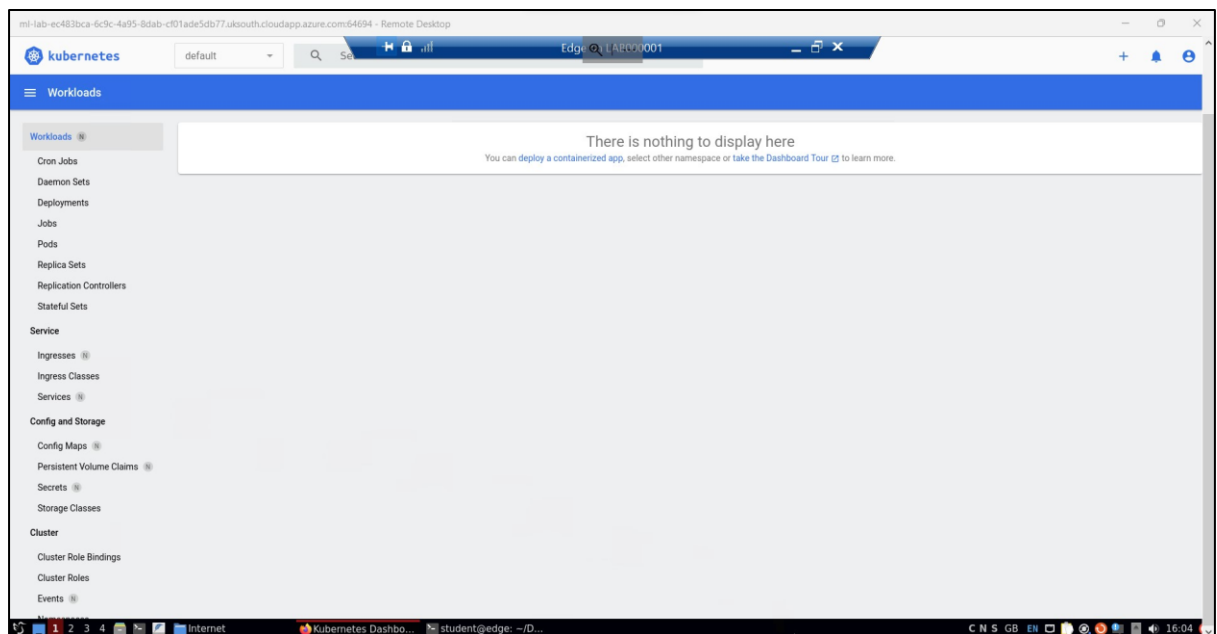
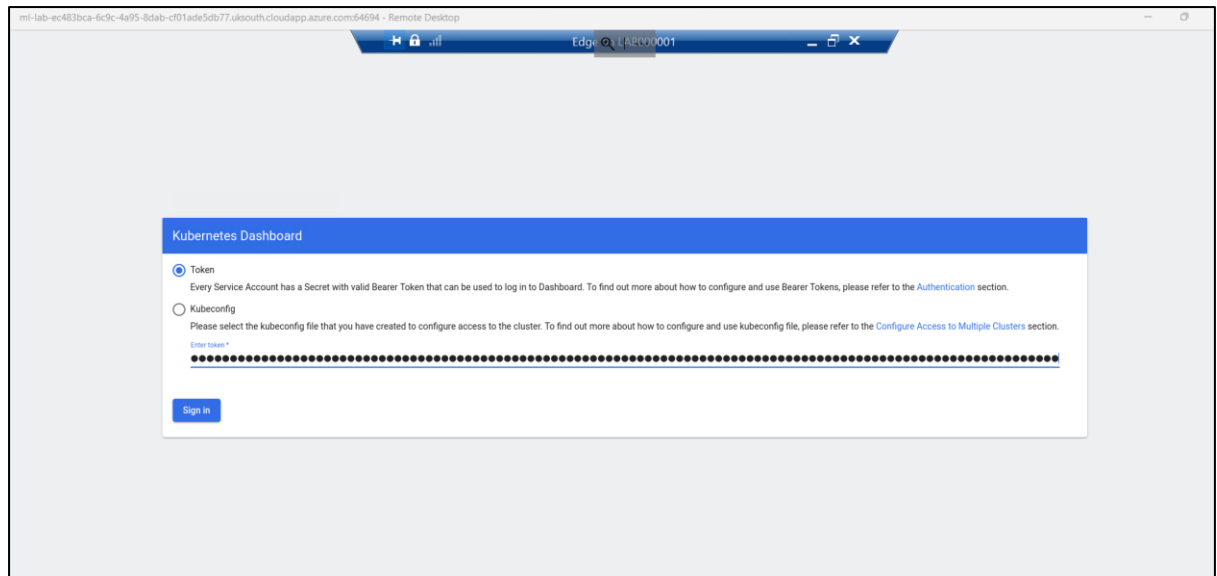
0 updates can be applied immediately.

9 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Last login: Tue Nov 28 13:55:03 2023 from fe80::4105:6c1c:1e6a:1d47%eth0
student@edge:~$ cd Documents
student@edge:~/Documents$ cd coursework
student@edge:~/Documents/coursework$ nano Serviceaccount.yaml
student@edge:~/Documents/coursework$ kubectl apply -f Serviceaccount.yaml
serviceaccount/admin-user created
student@edge:~/Documents/coursework$ nano clusterbinding.yaml
student@edge:~/Documents/coursework$ kubectl apply -f clusterbinding.yaml
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
student@edge:~/Documents/coursework$
```

Now in order to login to the dashboard we need to generate a token. I have generated the long live bearer token which will be saved in the secret and can be used to login into the dashboard. We can also generate a temporary token to login into the dashboard however that token will keep changing every time we generate a new one. We execute the below yaml file in order to generate a token and store it in a secret. Run the following command to apply the manifest file,  
"kubectl apply -f generatetoken.yaml"





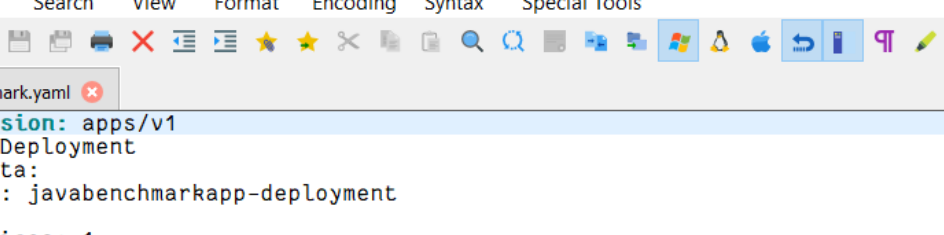
## 1.2 Deploying the instance of provided docker image for javabenchmarkapp

In order to deploy an image using CLI we need to use Kubernetes manifest file in yaml format which will contain the configurations of image deployment.

Firstly, we specify the version of Kubernetes API being used followed by the type of Kubernetes resource being created, which is "Deployment". Next, we provide name for our deployment which is given as javabenchmarkapp-deployment.

Further in spec, we define 1 replica to run. In selector we define which pods this deployment manages. Template defines the pod template that Kubernetes will use to create new pods. First part is metadata, that specifies the label for pods created using this template followed by pods specification such as containers within the pod. It contains name of the container, we give as "javabenchmarkapp-container" and docker image which is "nclcloudcomputing/javabenchmarkapp" and in port we specify the port number which application inside the container is using.



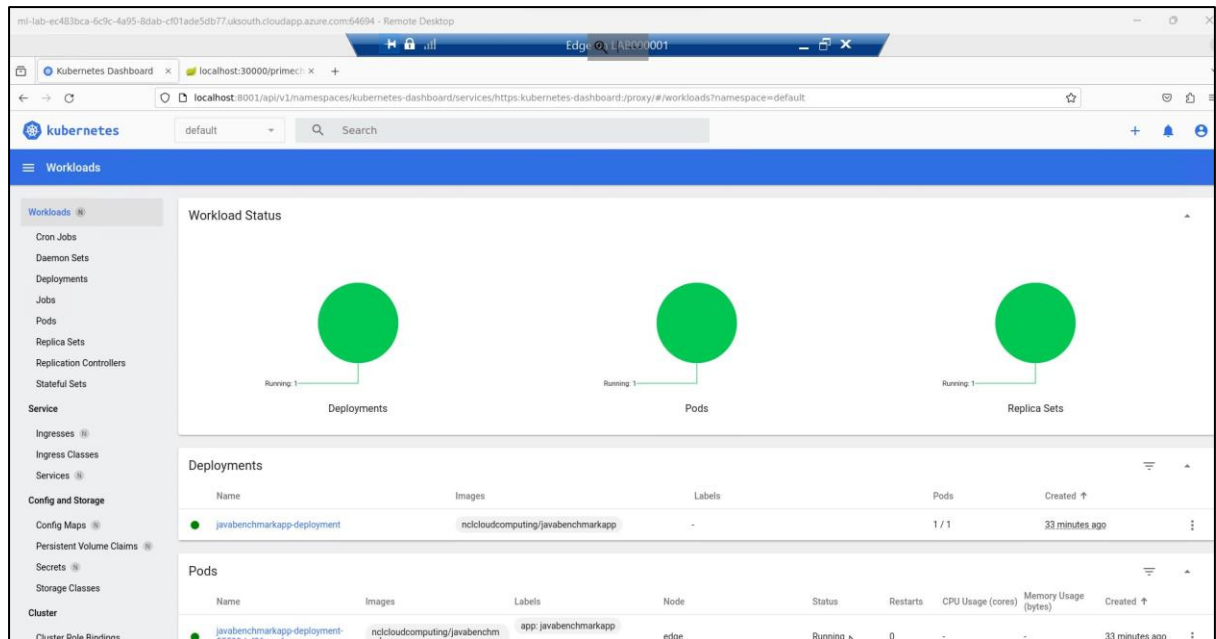


The screenshot shows the MobaTextEditor application with a menu bar (File, Edit, Search, View, Format, Encoding, Syntax, Special Tools) and a toolbar. The active file is 'javabenchmark.yaml'. The content of the file is a YAML manifest for a Kubernetes Deployment, with line numbers 1 through 19 on the left margin.

```

1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: javabenchmarkapp-deployment
5 spec:
6   replicas: 1
7   selector:
8     matchLabels:
9       app: javabenchmarkapp
10  template:
11    metadata:
12      labels:
13        app: javabenchmarkapp
14    spec:
15      containers:
16        - name: javabenchmarkapp-container
17          image: nclcloudcomputing/javabenchmarkapp
18          ports:
19            - containerPort: 8080

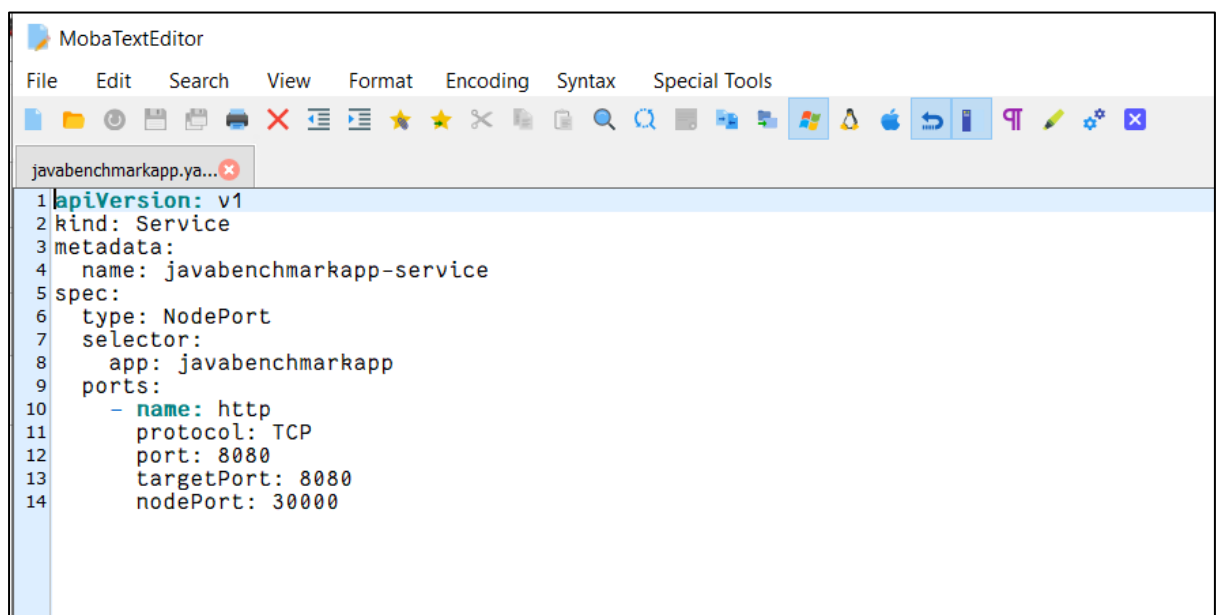
```



### 1.3 Deploying a NodePort service

We can deploy a NodePort service by specifying the configurations in a yaml file. The kind of Kubernetes resource we are deploying will be "Service". Then in metadata we specify the name of service as "javabenchmarkapp-service". In spec, we set the type of service as NodePort and in ports we define name as http, protocol used by the port as TCP, port 8080 exposed on the service and target port to which service will forward the traffic. Then in nodePort we provide the specific node port 30000 to use.

Basically, what a NodePort does is, it publicly exposes a service on a fixed port number and lets us access the service from outside of our cluster.







## Task 2: Deploy the monitoring stack of Kubernetes

### 2.1 Enable observability service from microk8s addons

In order to install observability stack, we run the below command into our CLI.

*“microk8s enable observability”*

It prints out the status of different add-ons that it is enabling and installs the missing add-ons.

Running this command basically enables observability-related add-ons and functionalities. It provides us with the set of monitoring, logging, and observability tools which we can use to gain insights into the performance of Kubernetes cluster and its running applications.

```
mi-lab-ec43bca-6c9c-4d95-8dab-cf01ade5db77@ubuntu20.04:~$ microk8s enable observability
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
jvabenmarkapp-deployment 1/1      10.152.183.1      <none>           443/TCP          32m
jvabenmarkapp-service     NodePort  10.152.183.225    <none>           8080:30080/TCP   32m
student@edge:~/Documents/coursework$ microk8s enable observability
Infer repository core for add-on observability
Add-on core/dns is already enabled
Add-on core/helm3 is already enabled
Add-on core/hostpath-storage is already enabled
Enabling observability
Release "kube-prom-stack" does not exist. Installing it now.
NAME: kube-prom-stack
LAST DEPLOYED: Wed Nov 29 17:33:16 2023
NAMESPACE: observability
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace observability get pods -l "release=kube-prom-stack"
Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
Release "loki" does not exist. Installing it now.
NAME: loki
LAST DEPLOYED: Wed Nov 29 17:33:50 2023
NAMESPACE: observability
STATUS: deployed
REVISION: 1
NOTES:
The Loki stack has been deployed to your cluster. Loki can now be added as a datasource in Grafana.
See http://docs.grafana.org/features/datasources/loki/ for more detail.
Release "tempo" does not exist. Installing it now.
NAME: tempo
LAST DEPLOYED: Wed Nov 29 17:33:54 2023
NAMESPACE: observability
STATUS: deployed
REVISION: 1
TEST SUITE: None
[sudo] password for student:
Sorry, try again.
[sudo] password for student:
Note: the observability stack is setup to monitor only the current nodes of the MicroK8s cluster.
For any nodes joining the cluster at a later stage this add-on will need to be set up again.
Observability has been enabled (user/pass: admin/prom-operator)
student@edge:~/Documents/coursework$
```

### 2.2 Editing the Grafana service

We will first check if the Grafana service is present by listing all the services within our namespace which is “observability”. For this we run the below command into CLI,

*“kubectl get svc -n observability”*

```
student@edge:~/Documents/coursework$ kubectl get svc -n observability
NAME                                TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)
kube-prom-stack-kube-prometheus     ClusterIP  10.152.183.226   <none>           9090/TCP
kube-prom-stack-kube-state-metrics  ClusterIP  10.152.183.18    <none>           8080/TCP
kube-prom-stack-prometheus-node-exporter ClusterIP  10.152.183.118   <none>           9100/TCP
kube-prom-stack-kube-prometheus-operator ClusterIP  10.152.183.168   <none>           443/TCP
kube-prom-stack-kube-prometheus-alertmanager ClusterIP  10.152.183.33    <none>           9093/TCP
alertmanager-operated               ClusterIP  None             <none>           9093/TCP,9094/TCP,9094/UDP
prometheus-operated                 ClusterIP  None             <none>           9090/TCP
loki-headless                       ClusterIP  None             <none>           3100/TCP
loki-memberlist                     ClusterIP  None             <none>           7946/TCP
loki                                 ClusterIP  10.152.183.53    <none>           3100/TCP
tempo                                ClusterIP  10.152.183.45    <none>           3100/TCP,16687/TCP,16686/TCP,6831/UDP,6832/UDP,14268/TCP,14250/TCP,9411/TCP,55
kube-prom-stack-grafana              NodePort  10.152.183.27    <none>           80:31000/TCP
student@edge:~/Documents/coursework$
```

Now edit the Grafana service by executing below command,  
"kubect! edit service kube-prom-stack-grafana -n observability"

```
student@edge:~$ cd Documents/coursework
student@edge:~/Documents/coursework$ kubectl get svc -n observability
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)
kube-prom-stack-kube-prometheus     ClusterIP    10.152.183.226   <none>           9090/TCP
kube-prom-stack-kube-state-metrics   ClusterIP    10.152.183.18    <none>           8080/TCP
kube-prom-stack-prometheus-node-exporter ClusterIP     10.152.183.118   <none>           9100/TCP
kube-prom-stack-kube-prometheus-operator ClusterIP     10.152.183.168   <none>           443/TCP
kube-prom-stack-kube-prometheus-alertmanager ClusterIP     10.152.183.33    <none>           9093/TCP
alertmanager-operated               ClusterIP    None             <none>           9093/TCP,9094/TCP,9094/UDP
prometheus-operated                 ClusterIP    None             <none>           9090/TCP
loki-headless                       ClusterIP    None             <none>           3100/TCP
loki-memberlist                     ClusterIP    None             <none>           7946/TCP
loki                                 ClusterIP    10.152.183.53    <none>           3100/TCP
tempo                               ClusterIP    10.152.183.45    <none>           3100/TCP,16687/TCP,16686/TCP,6831/UDP,6832/UDP,14268/TCP,14250/TCP,9411/TCP,55681/TCP,55681/TCP,4317/TCP,4318/TCP,55678/TCP
kube-prom-stack-grafana              NodePort     10.152.183.27    <none>           80:31000/TCP

student@edge:~/Documents/coursework$ kubectl edit service kube-prom-stack-grafana -n observability
```

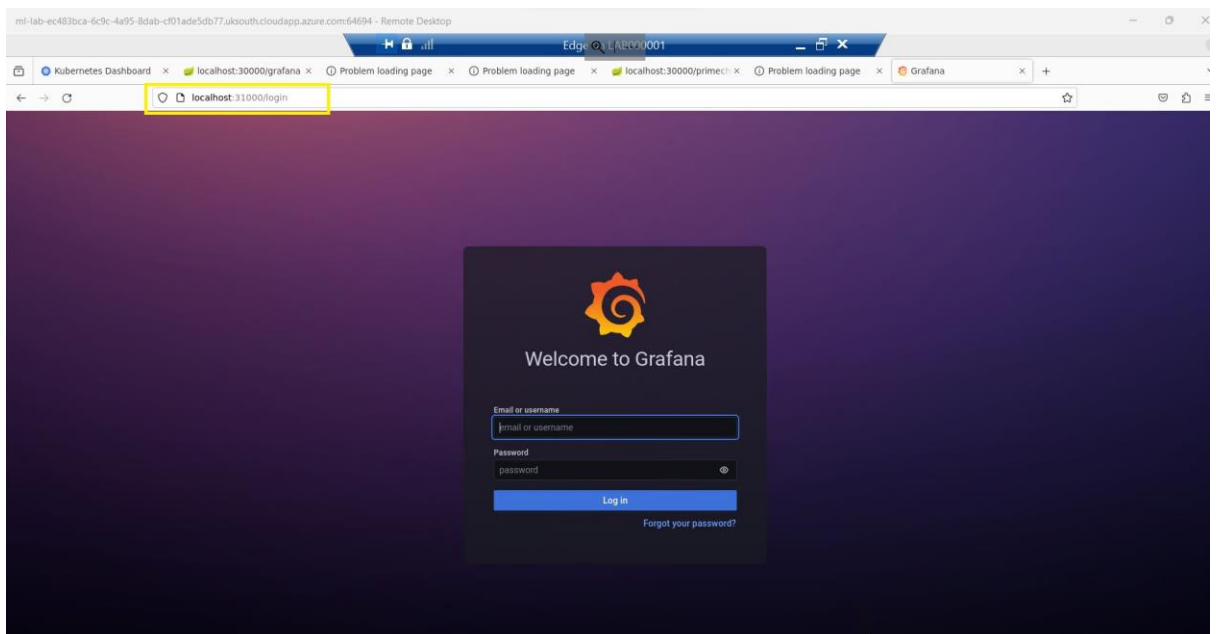
```
ml-lab-ec483bca-6c9c-4a95-8dab-cf01ade5db77.uksouth.cloudapp.azure.com:64694 - Remote Desktop
Edge 118.0.2003.0
student@edge: ~/Documents/coursework x student@edge: ~/Documents/coursework x
# Please edit the object below. Lines beginning with a '#' will be ignored, and an empty file will abort the 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: "2023-11-29T17:35:38Z"
  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: "4471"
  uid: 81186e3e-4a6e-4f92-bfaa-e71a9e55fb54
spec:
  clusterIP: 10.152.183.27
  clusterIPs:
    - 10.152.183.27
  internalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - name: http-web
      port: 80
      protocol: TCP
      targetPort: 3000
  selector:
    app.kubernetes.io/instance: kube-prom-stack
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: ClusterIP
status: {}
loadBalancer: {}
Storage Classes
Cluster
Cluster Role Bindings
```

Here we change the type from ClusterIP to NodePort and then we define the port 31000 for nodePort.

```
mi-lab-ec483bca-6c9c-4a95-8dab-cf01ade5db77.uksouth.cloudapp.azure.com:64694 - Remote Desktop
Edge LAR0000001
File Actions Edit View Help
student@edge: ~/Documents/coursework x student@edge: ~/Documents/coursework x
# 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: "2023-11-29T17:33:30Z"
  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: "38778"
  uid: 81186e3e-4a6e-4f92-bfaa-e71a9e55fb54
spec:
  selector:
    app.kubernetes.io/instance: kube-prom-stack
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
  Storage Class
  Cluster
  Cluster Role Bindings
```

## 2.3 Log in to Grafana dashboard

After editing the Grafana service, now we can access it from localhost and port number 31000 that we have specified while editing the service.



Login into the Grafana dashboard by using the credentials provided in the text document present on edge virtual machine.



```

41
42
43 if __name__ == "__main__":
44     target = os.getenv("TARGET_ADDRESS", "http://10.152.183.225:8080/primecheck")
45     frequency = float(os.getenv("REQUESTS_PER_SECOND", 10))
46
47     load_generator = LoadGenerator(target, frequency)
48     load_generator.generate_load()

```

The above code for load generation is written in python programming language. Initially we import all the required libraries such as requests, time and os. The code consists of a main method and LoadGenerator class. In the main method we specify two variables which are namely target and frequency. In “target” we will basically store the URL on which we will be generating the load and set the frequency value to 10 requests per second. Next, we are creating the object of LoadGenerator class by passing target and frequency and then calling generate\_load() function of LoadGenerator class.

In generate\_load function we have used a never ending while loop which will keep sending requests to our benchmark app. If response for the sent request is not successful i.e status code is not 200 then it will be counted as a failure. We have also specified a timeout of 10 seconds which means if a particular request is taking more than 10 seconds then it will timeout and be treated as a failure. We collect total number of failures and average response time. Next, we call the print\_metrics function to output the average response time and total number of accumulated failures.

### 3.2 Building the image and pushing it to local registry

We have to first write a docker file to build the image of our load generator python code.

```

Dockerfile
1 FROM python:3.9
2 WORKDIR /app
3 COPY loadgenerator.py .
4 RUN pip install requests
5 CMD ["python", "loadgenerator.py"]

```

First, we specify the base image of python 3.9 and then set the working directory to /app. Now we copy our loadgenerator python code to the working directory and specify the command that needs to execute after image starts running.

Now we will build the image and name it as “load-generator” by using the below command,

*“docker build -t load-generator.”*

```

student@edge:~/Documents/coursework$ docker build -t load-generator .
[+] Building 13.2s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 153B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.9
=> [1/4] FROM docker.io/library/python:3.9@sha256:b2e47a7eca3178e4ce6c095d3a2d1cd05bfa616efe7f2047c95fffe159e00166
=> [internal] load build context
=> => transferring context: 1.72kB
=> CACHED [2/4] WORKDIR /app
=> [3/4] COPY loadgenerator.py .
=> [4/4] RUN pip install requests
=> exporting to image
=> => exporting layers
=> writing image sha256:de6b8740f3f6127bff8efd8251e815ab5ba59a9c4819bd00feb57adc3ca79eaa
=> naming to docker.io/library/load-generator

```

Now we need to push this image to the local registry at port 32000. We will pull the registry image by executing the below command.

*“docker run -d -p 32000:5000 --restart=always --name registry registry:2”*



```

student@edge:~/Documents/coursework$ docker run -d -p 32000:5000 --restart=always --name registry registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
c926b61bad3b: Pull complete
5501dced60f8: Pull complete
e875fe5e6b9c: Pull complete
21f4bf2f86f9: Pull complete
98513cca25bb: Pull complete
Digest: sha256:0a182cb82c93939407967d6d71d6caf11dcef0e5689c6afe2d60518e3b34ab86
Status: Downloaded newer image for registry:2
4d85846a7fb3b16fb48eeff5b0fc8e41d99a244f95df4b91a3fa7ca37dbe3aad
student@edge:~/Documents/coursework$ docker tag load-generator localhost:32000/load-generator

```

Now we will tag the image and push it to the local registry at port 32000

```

student@edge:~/Documents/coursework$ docker build -t load-generator .
[+] Building 13.2s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 153B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.9
=> [1/4] FROM docker.io/library/python:3.9@sha256:b2e47a7eca3178e4ce6c095d3a2d1cd05bfa616efe7f2047c95ffe159e00166
=> [internal] load build context
=> => transferring context: 1.72kB
=> CACHED [2/4] WORKDIR /app
=> [3/4] COPY loadgenerator.py
=> [4/4] RUN pip install requests
=> exporting to image
=> exporting layers
=> => writing image sha256:de6b8740f3f6127bff8efd8251e815ab5ba59a9c4819bd00feb57adc3ca79eaa
=> => naming to docker.io/library/load-generator
student@edge:~/Documents/coursework$ docker tag load-generator localhost:32000/load-generator
student@edge:~/Documents/coursework$ docker push localhost:32000/load-generator
Using default tag: latest
The push refers to repository [localhost:32000/load-generator]
90f00dde2ecc: Pushed
08ccd560c8d8: Pushed
81eaa271543c: Layer already exists
d5ad3ac69862: Layer already exists
4929be171cab: Layer already exists
f021e1878a27: Layer already exists
a04a14a911a5: Layer already exists
80bd043d4663: Layer already exists
30f5cd833236: Layer already exists
7c32e0608151: Layer already exists
7ceal7427f83: Layer already exists
latest: digest: sha256:c5df4bbdd30839a5b90deba8a50887e9264894d2170a7e343d263600b15dae51 size: 2631

```

## Task 4: Monitor benchmarking results

### 4.1 To deploy load-generator service that we created in task 3

We can deploy load-generator service using Kubernetes manifest file in yaml format. We will specify all the required configurations in this file. First, we specify Kubernetes API version and specify "Deployment" as the kind of Kubernetes resource being created. In metadata field we name our deployment as "generateload" and specify 1 replica set to be created. Next is template, which is basically used by Kubernetes while creating new pods. In this we specify name as "generateload" followed by container specification such as name, image that we built and pushed to local registry in task3, container port 8080 and environment variables which are target and frequency. We name this file as "loadgeneratorservice.yaml".

```
loadgeneratorservice.... x
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: generateload
5 spec:
6   replicas: 1
7   selector:
8     matchLabels:
9     app: generateload
10  template:
11    metadata:
12      labels:
13        app: generateload
14    spec:
15      containers:
16      - name: generateload
17        image: localhost:32000/load-generator
18        ports:
19        - containerPort: 8080
20        env:
21        - name: target
22          value: "http://10.152.183.225:8080/primecheck"
23        - name: frequency
24          value: "10.0"
25        command: ["python", "loadgenerator.py"]
```

Now we can do the deployment by using following command,

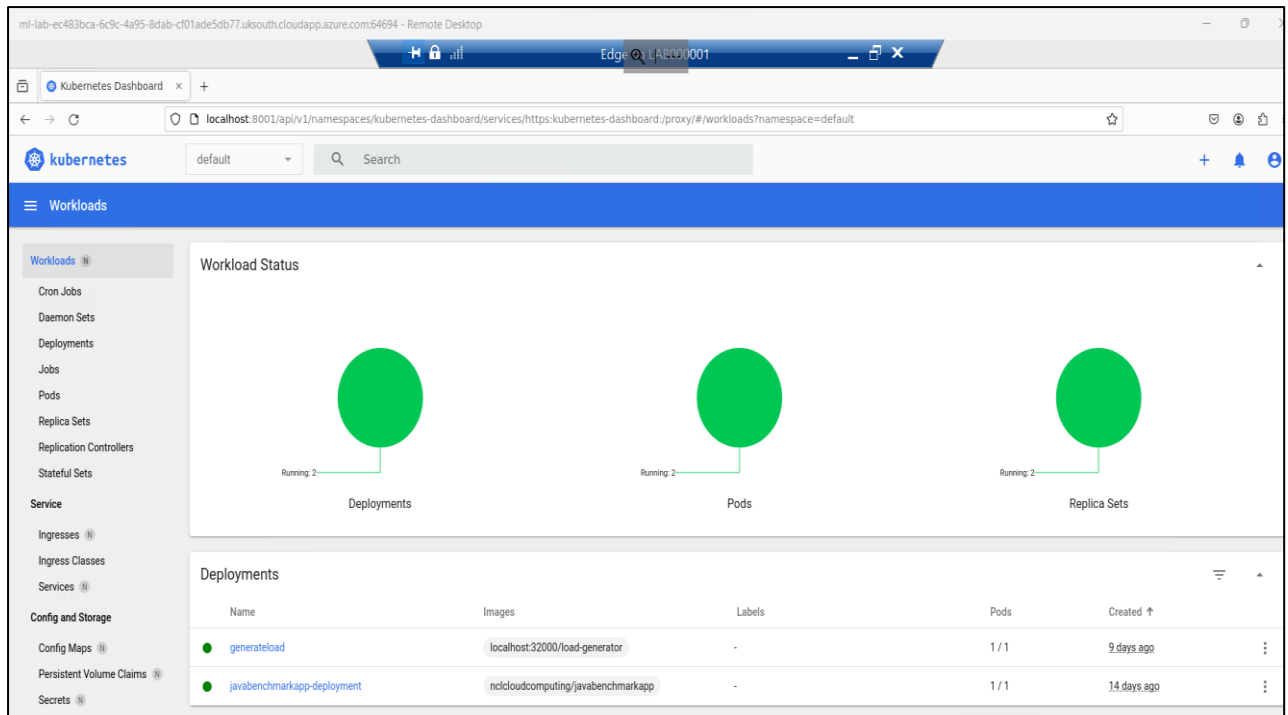
*"kubectl apply -f loadgeneratorservice.yaml"*

```
student@edge:~/Documents/coursework$ kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/javabenchmarkapp-deployment-55599dcf96-xncfn  1/1      Running   2 (18m ago)  4d22h

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
service/kubernetes                  ClusterIP     10.152.183.1  <none>         443/TCP          4d22h
service/javabenchmarkapp-service    NodePort      10.152.183.225 <none>         8080:30000/TCP   4d22h

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/javabenchmarkapp-deployment  1/1      1              1            4d22h

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/javabenchmarkapp-deployment-55599dcf96  1          1          1        4d22h
student@edge:~/Documents/coursework$ kubectl apply -f loadgeneratorservice.yaml
deployment.apps/generateload created
```



```

student@edge:~/Documents/coursework$
student@edge:~/Documents/coursework$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/javabenchmarkapp-deployment-55599dcf96-xncfn  1/1     Running   3 (10h ago)  14d
pod/generateload-75b4544769-n9bqs             1/1     Running   2 (10h ago)  9d

NAME                                TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/kubernetes                  ClusterIP      10.152.183.1    <none>        443/TCP          14d
service/javabenchmarkapp-service    NodePort      10.152.183.225  <none>        8080:30000/TCP   14d

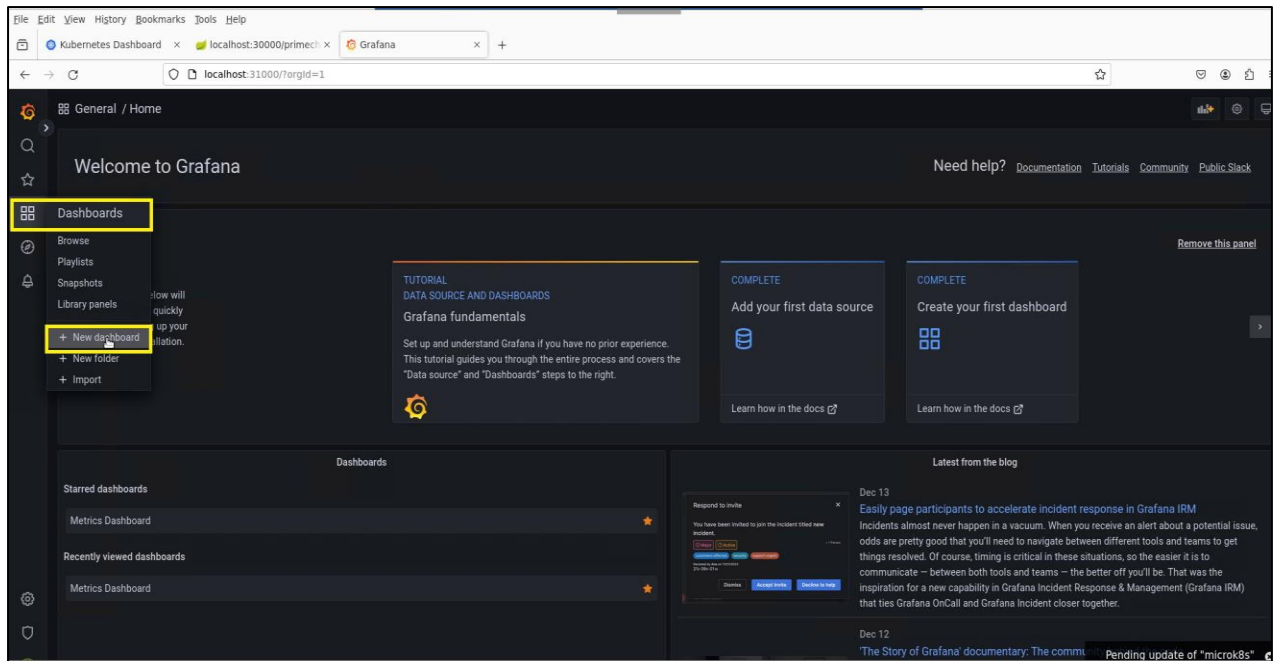
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/javabenchmarkapp-deployment  1/1     1             1           14d
deployment.apps/generateload                1/1     1             1           9d

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/javabenchmarkapp-deployment-55599dcf96  1         1         1       14d
replicaset.apps/generateload-75b4544769                1         1         1       9d
student@edge:~/Documents/coursework$

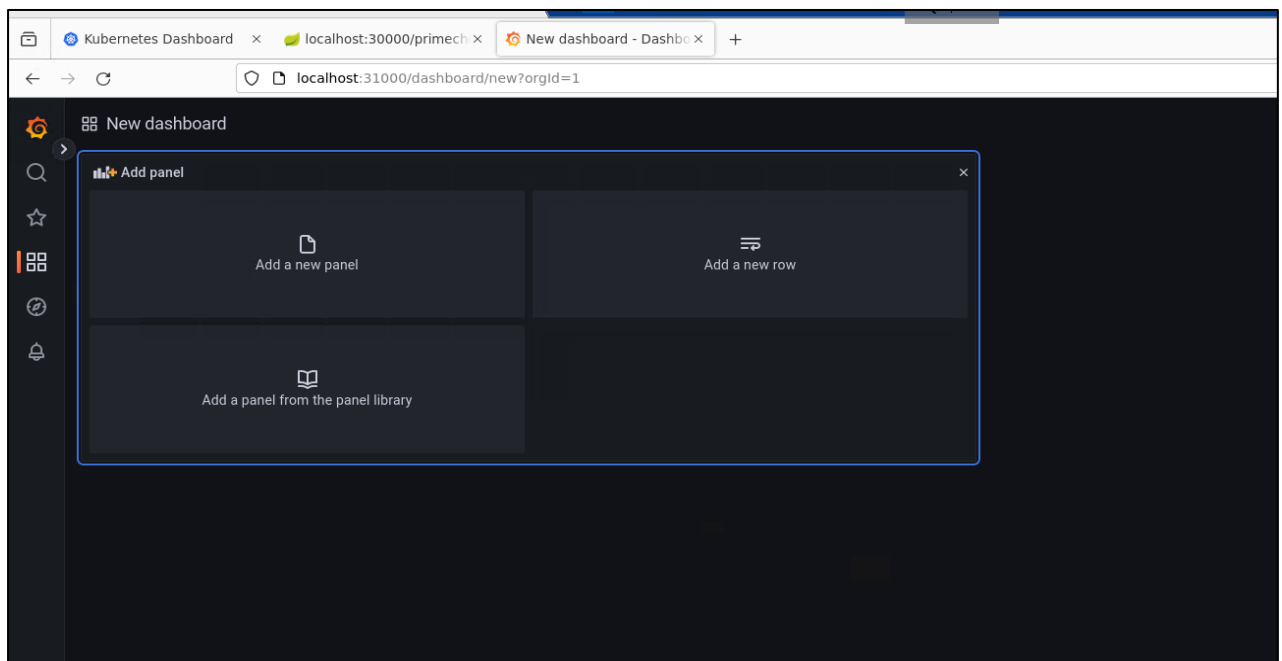
```

## 4.2 Creating dashboard and adding panels in Grafana

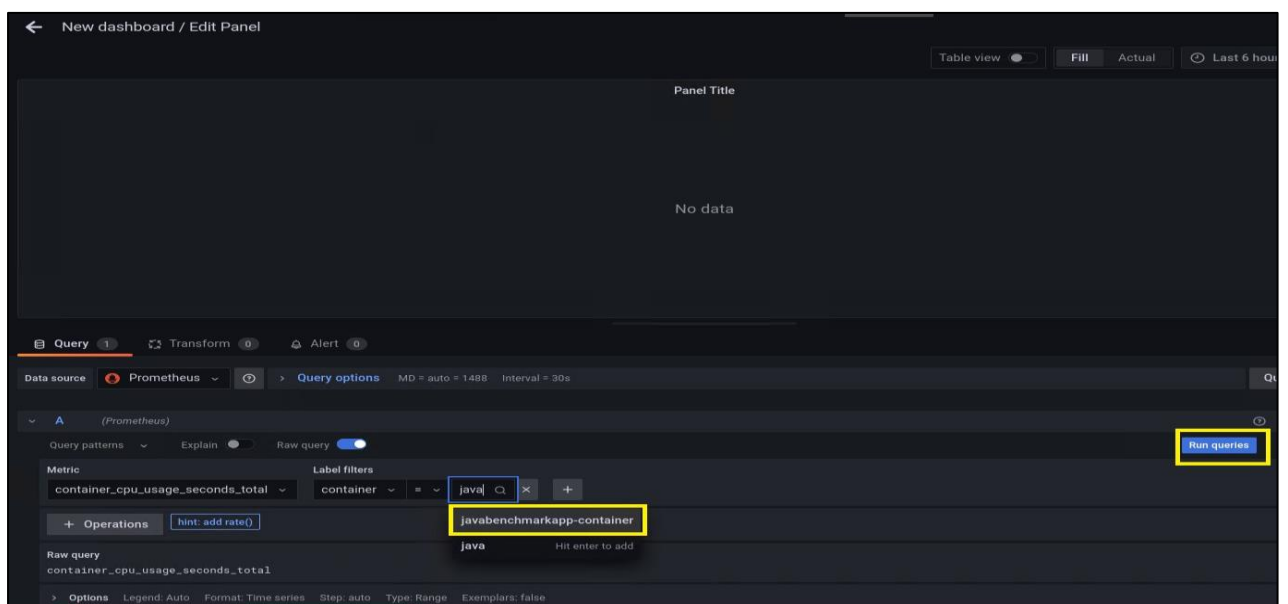
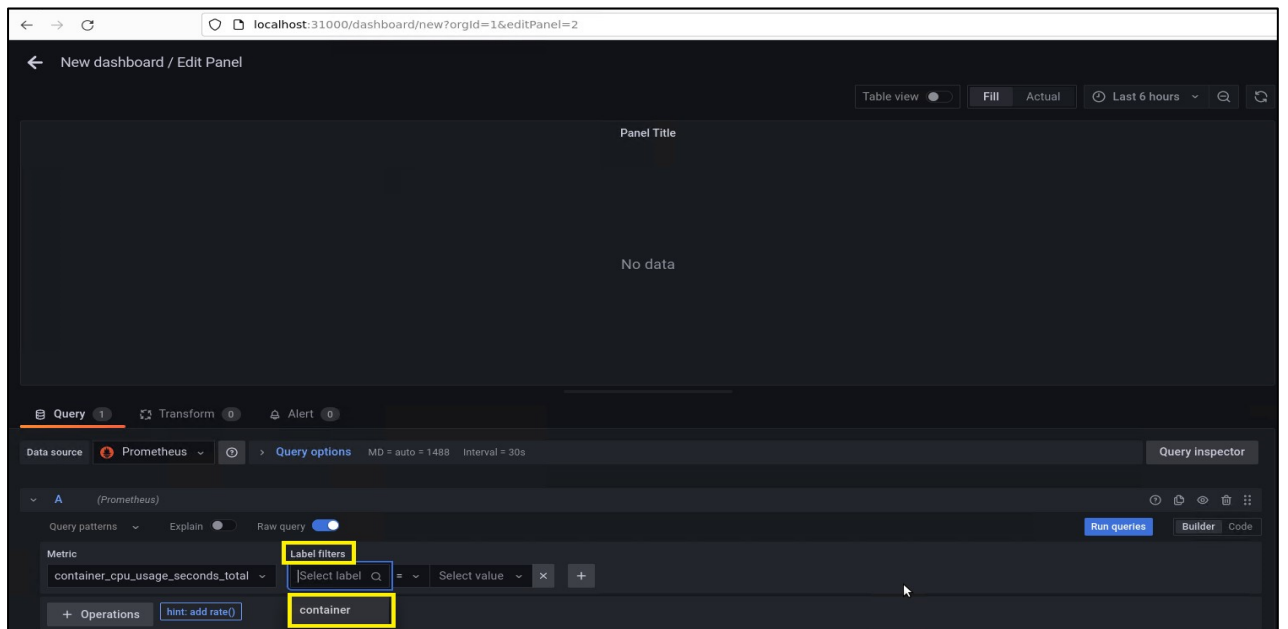
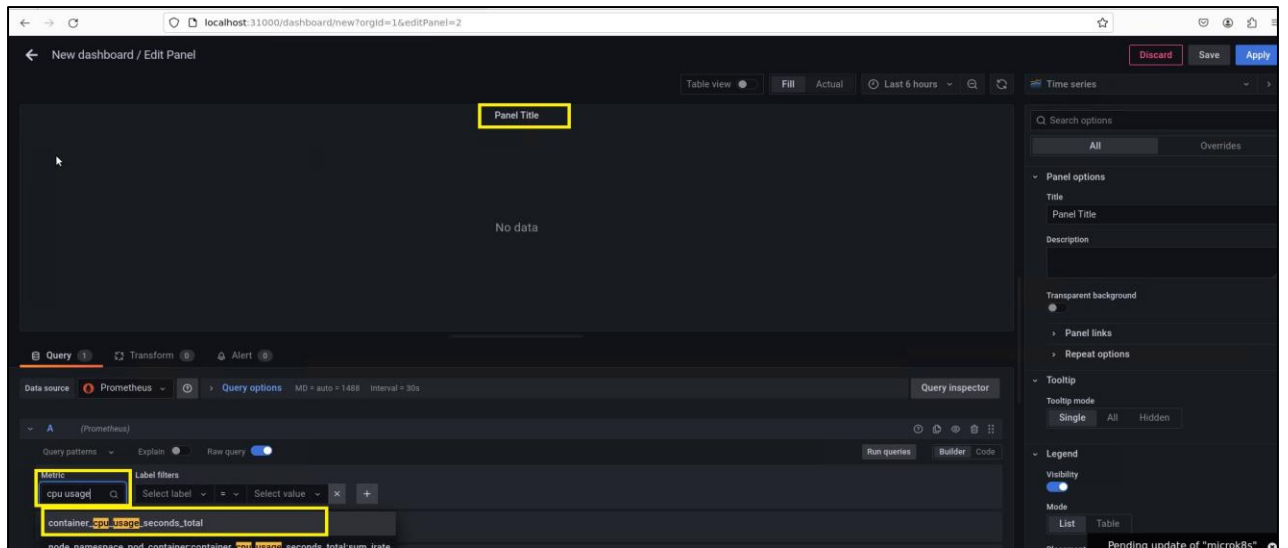
Login into Grafana and create a new dashboard.



Click on Add a new panel



Give the title to the panel as "cpu\_usage" and select the metric "container\_cpu\_usage\_seconds\_total". Then in Label filters select "container" and select the value as "javabenchmarkapp-container" and click on run query.



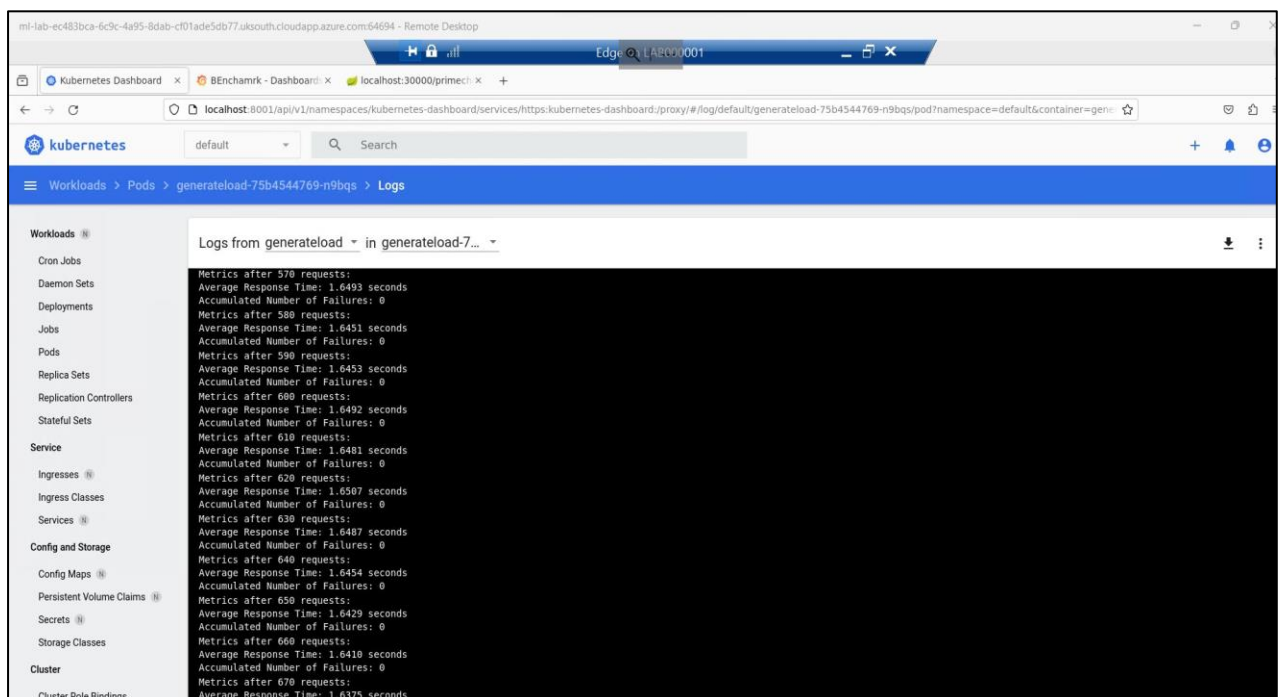


Now we need to add one more panel for memory usage. Follow the same steps as above by giving title to the panel as “memory” then in metrics select “container\_memory\_usage\_bytes”, in label filters select “container” and select value as “javabenchmarkapp-container” and click on run query.

Our dashboard will now contain two panels with metrics for total CPU usage in seconds and memory usage in bytes.



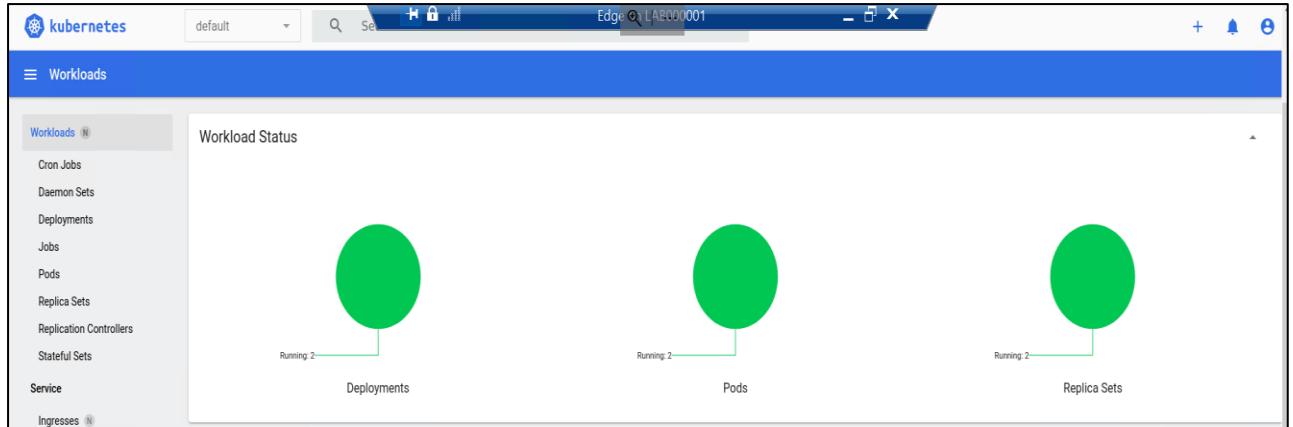
We can also see the logs of our “generateload” pod as we print the metrics such as “Average response time” and “Accumulated number of failures” in our load generator python code.



## Task 5

### Screenshot of running services in Kubernetes

#### Workload Status



#### Deployments

The screenshot shows the 'Deployments' page in the Kubernetes UI. The left sidebar lists various workload types: Workloads, Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets, Service, Ingresses, and Ingress Classes. The main area displays a table of active deployments.

Name	Images	Labels	Pods	Created ↑
generateload	localhost:32000/load-generator	-	1 / 1	9 days ago
javabenchmarkapp-deployment	ncicloudcomputing/javabenchmarkapp	-	1 / 1	14 days ago

#### Pods

The screenshot shows the 'Pods' page in the Kubernetes UI. The left sidebar lists various workload types: Workloads, Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets, and Service. The main area displays a table of active pods.

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
generateload-75b4544769-n9bqs	localhost:32000/load-generator	app: generateload pod-template-hash: 75b4544769	edge	Running	2	-	-	9 days ago
javabenchmarkapp-deployment-55599dcf96-xncfn	ncicloudcomputing/javabenchmarkapp	app: javabenchmarkapp pod-template-hash: 55599dcf96	edge	Running	3	-	-	14 days ago

## Services

Services							
Name	Labels	Type	Cluster IP	Internal Endpoints	External Endpoints	Created	
javabenchmarkapp-service		NodePort	10.152.183.225	javabenchmarkapp-service:8080 TCP javabenchmarkapp-service:30000 TCP	-	14 days ago	
kubernetes	component: apiserver provider: kubernetes	ClusterIP	10.152.183.1	kubernetes:443 TCP kubernetes:0 TCP	-	14 days ago	

```
student@edge:~/Documents/coursework$ kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/javabenchmarkapp-deployment-55599dcf96-xncfn  1/1     Running   3 (10h ago)  14d
pod/generateload-75b4544769-n9bqs             1/1     Running   2 (10h ago)   9d

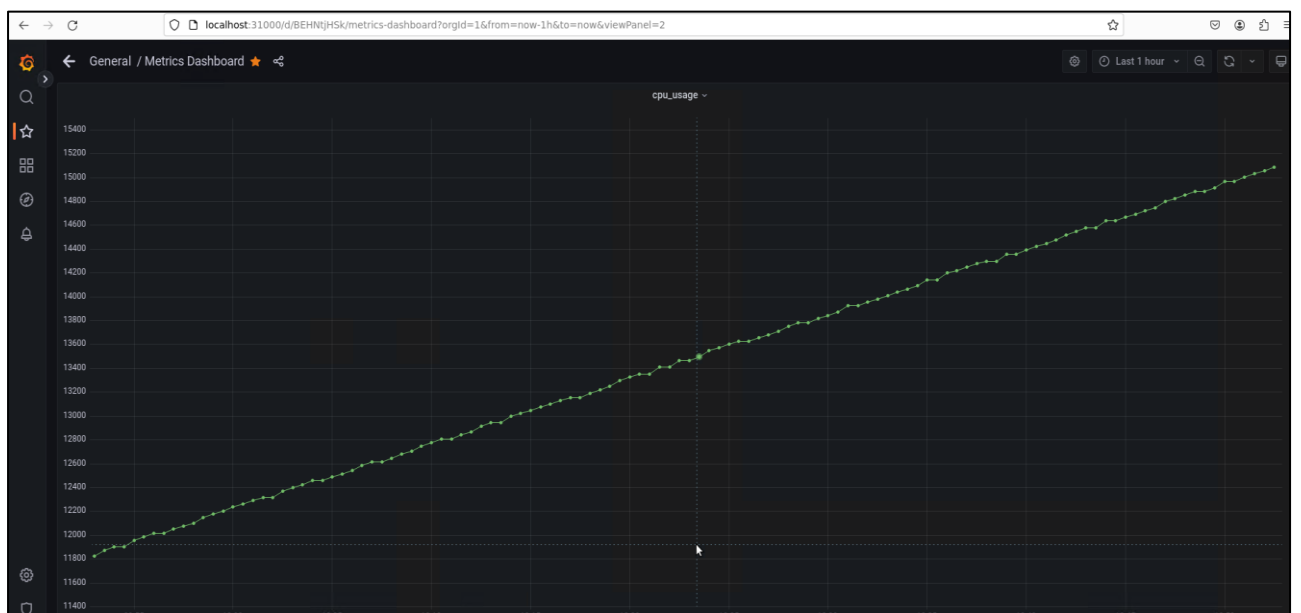
NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
service/kubernetes                  ClusterIP     10.152.183.1  <none>        443/TCP          14d
service/javabenchmarkapp-service    NodePort      10.152.183.225 <none>        8080:30000/TCP   14d

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/javabenchmarkapp-deployment  1/1     1             1           14d
deployment.apps/generateload                1/1     1             1           9d

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/javabenchmarkapp-deployment-55599dcf96  1         1         1       14d
replicaset.apps/generateload-75b4544769                1         1         1       9d
student@edge:~/Documents/coursework$
```

## Plots of benchmarking results

### 1. container\_cpu\_usage\_seconds\_total



## 2. container\_memory\_usage\_bytes



### Results and Conclusion

Talking about our first plot which is “container\_cpu\_usage\_seconds\_total”. Our load generator python code is continuously sending requests to the javabenchmarkapp. This javabenchmarkapp basically checks whether a number is prime or not on a very large number which in turn leads to generation of high load on the CPU. As a result of which we can see in the graph that CPU usage seconds keeps on increasing.

Our next plot is about “container\_memory\_usage\_bytes”. The memory consumption by the container keeps fluctuating and is in the range of 400 to 420 megabytes. To conclude we have deployed the Kubernetes dashboard and javabenchmarkapp docker image on Kubernetes. Then with the help of our load generator python code’s deployment we are simulating the load and benchmarking the systems performance using the monitoring stack of Kubernetes and Grafana.