CSC4160 Assignment2

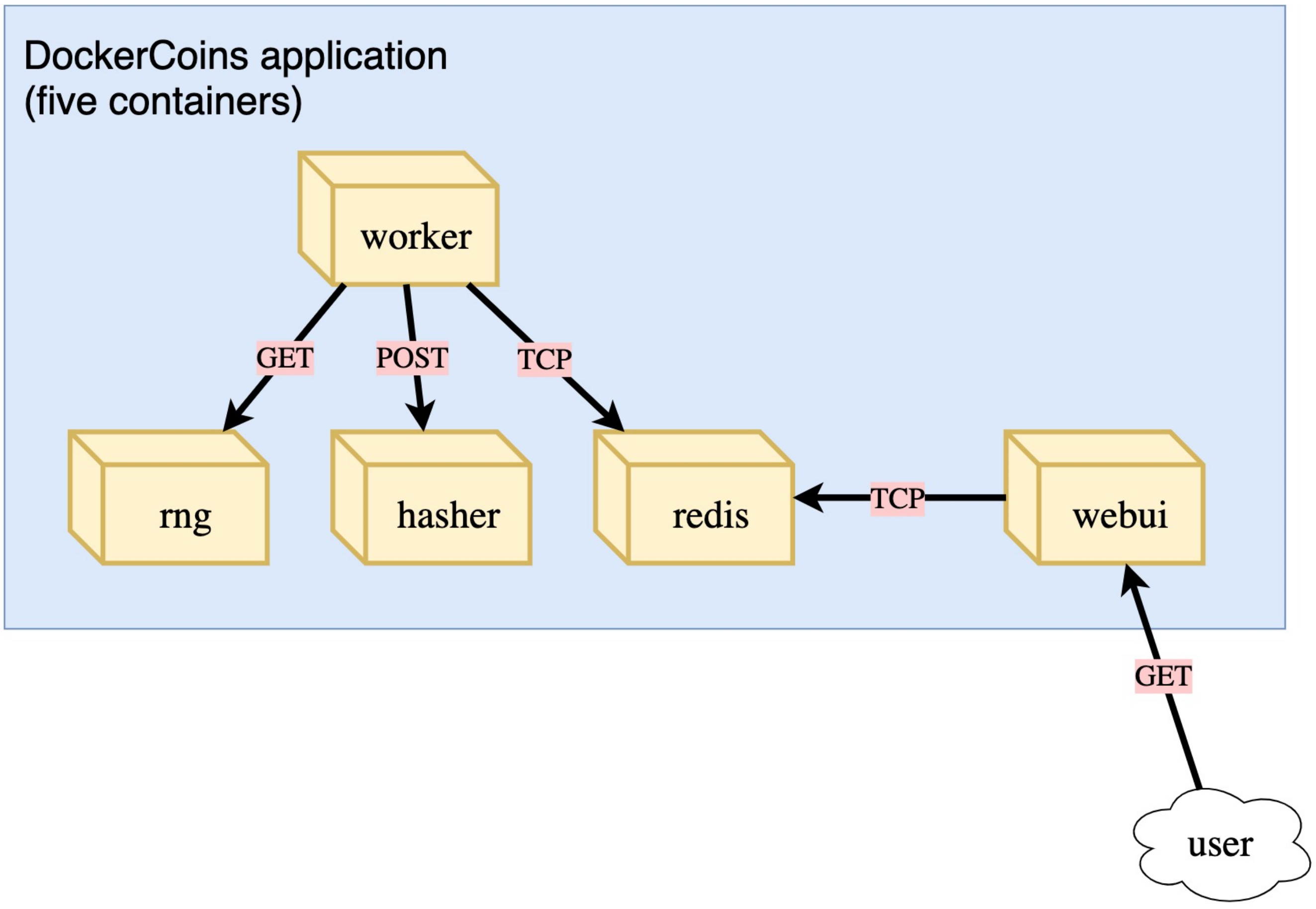
Report

Name: Xu Boshi

Student ID: 122040075

Introduction:

This assignment involves deploying the DockerCoins application, identifying performance bottlenecks, and using Kubernetes' Horizontal Pod Autoscaler (HPA) to scale the application based on specific metrics.



1. Environment Setup

AWS Settings:

> OS: Ubuntu Server 24.04 LTS (HVM), SSD Volume Type as AMI
> Type: m4.large
> Storage: 20 GB

Install Minikube

```
sudo curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Install kubectl

```
sudo curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Install Docker

```
sudo apt-get update && sudo apt-get install docker.io -y
```

Install conntrack

```
sudo apt-get install -y conntrack
```

Install httping

```
sudo apt-get install httping
```

Install Helm

```
sudo curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
sudo chmod 700 get_helm.sh
sudo ./get_helm.sh
```

2. Running Minikube on EC2 Ubuntu

Add user to “docker” group and switch primary group

```
sudo usermod -aG docker ubuntu && newgrp docker
```

Start Minikube

```
minikube start
```

Git Clone from my depository

```
git clone https://github.com/xh2002/CSC4160.git
```

Start the application from the dockercoins yaml

```
kubectl apply -f dockercoins.yaml
kubectl get po
minikube service webui
```

Check the result The name of webui is printed in the commend line

```
kubectl port-forward --address 0.0.0.0 webui-78cf59b54c-5ch26 8080:80
```

Open in default browser The URL is listed in the CMD

```
http://100.26.146.184:8080
```

3. Bottleneck detection

Scale the number of workers by editing the command below:

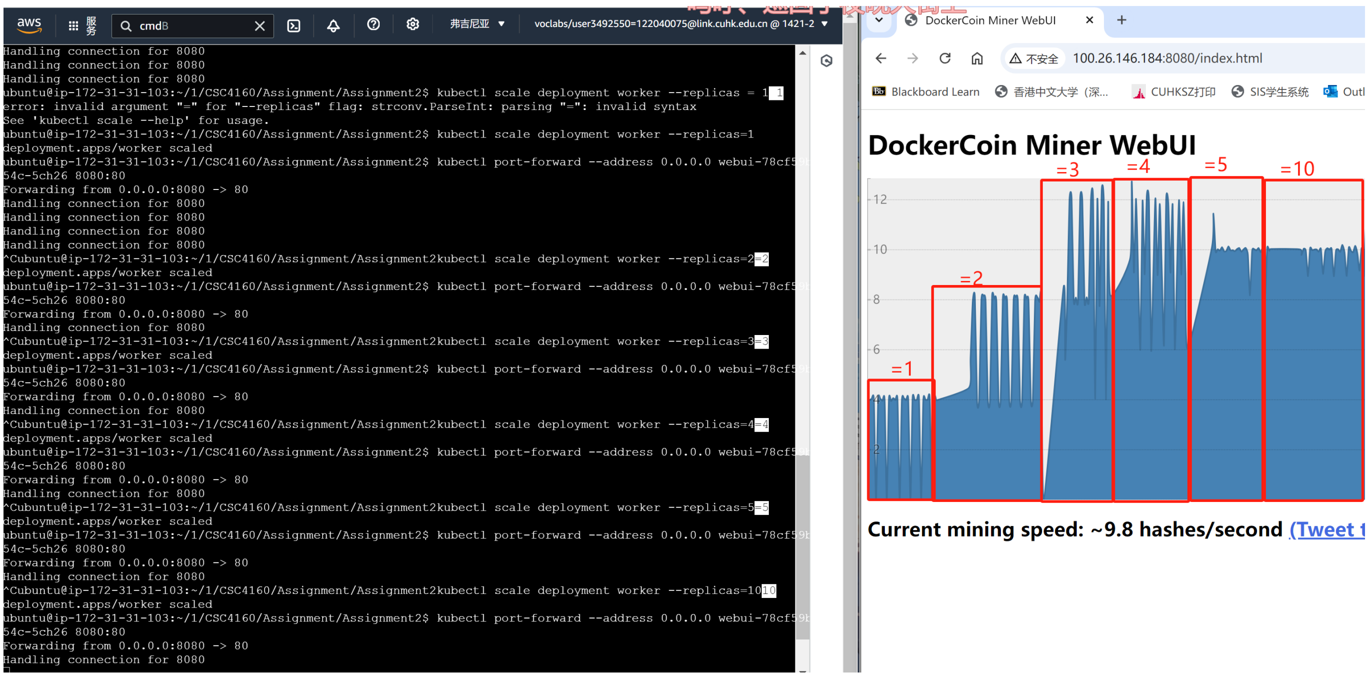
```
kubectl scale deployment worker --replicas=<required_workers_number>
```

Measuring the hashing speed in the DockerCoins application:

# of workers	1	2	3	4	5	10
hashes/second	4	8	12	12	10	10

Analysis:

> From these data, we can see that: when the number of workers expands from 1 to 10, the speed only increases from 4 hashes/sec to 10 hashes/sec. The speed increase is not 10 times, but only 2.5 times (from 4 to 10), which is far less than the ideal 10 times speed increase.This shows that in the process of increasing the number of workers, the application encountered a bottleneck, which made it impossible to further increase the processing speed.



```
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
ubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2$ kubectl scale deployment worker --replicas = 1 1
error: invalid argument "=" for "--replicas" flag: strconv.ParseInt: parsing "=": invalid syntax
See 'kubectl scale --help' for usage.
ubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2$ kubectl scale deployment worker --replicas=1
deployment.apps/worker scaled
ubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2$ kubectl port-forward --address 0.0.0.0 webui-78cf5
54c-5ch26 8080:80
Forwarding from 0.0.0.0:8080 -> 80
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
^Cubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2kubectl scale deployment worker --replicas=2=2
deployment.apps/worker scaled
ubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2$ kubectl port-forward --address 0.0.0.0 webui-78cf5
54c-5ch26 8080:80
Forwarding from 0.0.0.0:8080 -> 80
Handling connection for 8080
^Cubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2kubectl scale deployment worker --replicas=3=3
deployment.apps/worker scaled
ubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2$ kubectl port-forward --address 0.0.0.0 webui-78cf5
54c-5ch26 8080:80
Forwarding from 0.0.0.0:8080 -> 80
Handling connection for 8080
^Cubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2kubectl scale deployment worker --replicas=4=4
deployment.apps/worker scaled
ubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2$ kubectl port-forward --address 0.0.0.0 webui-78cf5
54c-5ch26 8080:80
Forwarding from 0.0.0.0:8080 -> 80
Handling connection for 8080
^Cubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2kubectl scale deployment worker --replicas=5=5
deployment.apps/worker scaled
ubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2$ kubectl port-forward --address 0.0.0.0 webui-78cf59b
54c-5ch26 8080:80
Forwarding from 0.0.0.0:8080 -> 80
Handling connection for 8080
^Cubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2kubectl scale deployment worker --replicas=1010
deployment.apps/worker scaled
ubuntu@ip-172-31-31-103:~/1/CSC4160/Assignment/Assignment2$ kubectl port-forward --address 0.0.0.0 webui-78cf59b
54c-5ch26 8080:80
Forwarding from 0.0.0.0:8080 -> 80
Handling connection for 8080
```

DockerCoin Miner WebUI

Current mining speed: ~9.8 hashes/second (Tweet t

4. Application monitoring

Deploy Prometheus

```
kubectl expose service rng --type=NodePort --target-port=80 --name=rng-np
kubectl expose service hasher --type=NodePort --target-port=80 --name=hasher-np
```

Get the URL of Server

```
kubectl get svc rng-np hasher-np
# Result
NAME        TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
rng-np      NodePort   10.107.118.250   <none>        80:31055/TCP   21s
hasher-np   NodePort   10.110.183.148   <none>        80:31886/TCP   13s
```

Find the minikube address

```
kubectl cluster-info
# Result
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy
```

Minikube address is: 192.168.49.2

Detect the latency of hasher

```
httping 192.168.49.2:31886

# Result (part of)
--- http://192.168.49.2:31886/ ping statistics ---
10 connects, 10 ok, 0.00% failed, time 9444ms
round-trip min/avg/max = 0.9/1.2/1.8 ms
```

Detect the latency of rng

```


httping 192.168.49.2:31055

# Result (part of)
--- http://192.168.49.2:31055/ ping statistics ---
10 connects, 10 ok, 0.00% failed, time 17174ms
round-trip min/avg/max = 726.6/732.5/770.5 ms
```

Analysis:

> In the DockerCoins application, workers are background processes that handle calculations. The whole process is: The worker requests random bytes from rng. The worker then sends these random bytes to hasher for hashing. Finally, the worker updates the Redis counter and records the number of completed hashes. Hashes/second therefore represents the rate at which complete hashing tasks are successfully performed by workers in the system.
>
> Judging from the latency data, the hasher service performs well under the current load, has a short response time, and can efficiently process requests. It can be inferred that hasher is not the bottleneck of the system. The latency of the rng service is significantly higher, much higher than the latency of the hasher service . The average latency is 732.5 milliseconds, indicating that the rng service is slow to respond, limiting the overall hashing speed of the system.

5. Application Monitor

Collecting metrics with Prometheus

```
# Install prometheus
helm repo add prometheus-community https://prometheus-community.github.io/helm-
charts
helm install prometheus prometheus-community/prometheus
# Result
NAME: prometheus
LAST DEPLOYED: Mon Oct  7 01:37:53 2024
NAMESPACE: default
STATUS: deployed

# Expose service port
kubectl expose service prometheus-server --type=NodePort --target-port=9090 --
```

```
name=prometheus-server-np

# Enable Metric-server
minikube addons list
minikube addons enable metrics-server
```

Edit yaml

```
# open with nano
nano httplat.yaml

# Change to:
# args: ["http://192.168.49.2:31055"] #rng

# run the service
kubectl apply -f httplat.yaml
kubectl expose deployment httplat --port=9080
```

Check if the deployment is ready

```
kubectl get deploy httplat
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
httplat   1/1     1            1           15s
```

Configure Prometheus to gather the detected latency, "\" means this line is not the end of the command

```
kubectl annotate service httplat \
          prometheus.io/scrape=true \
          prometheus.io/port=9080 \
          prometheus.io/path=/metrics
```

Connect to Prometheus

```
# Please notice that, starting from this step, I rebuild the EC2 because of error,
so the port number and ip address may be different from the previous one.

kubectl get svc prometheus-server-np

# Result
NAME                   TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
prometheus-server-np   NodePort   10.96.225.252   <none>        80:31330/TCP   
2m7s
```

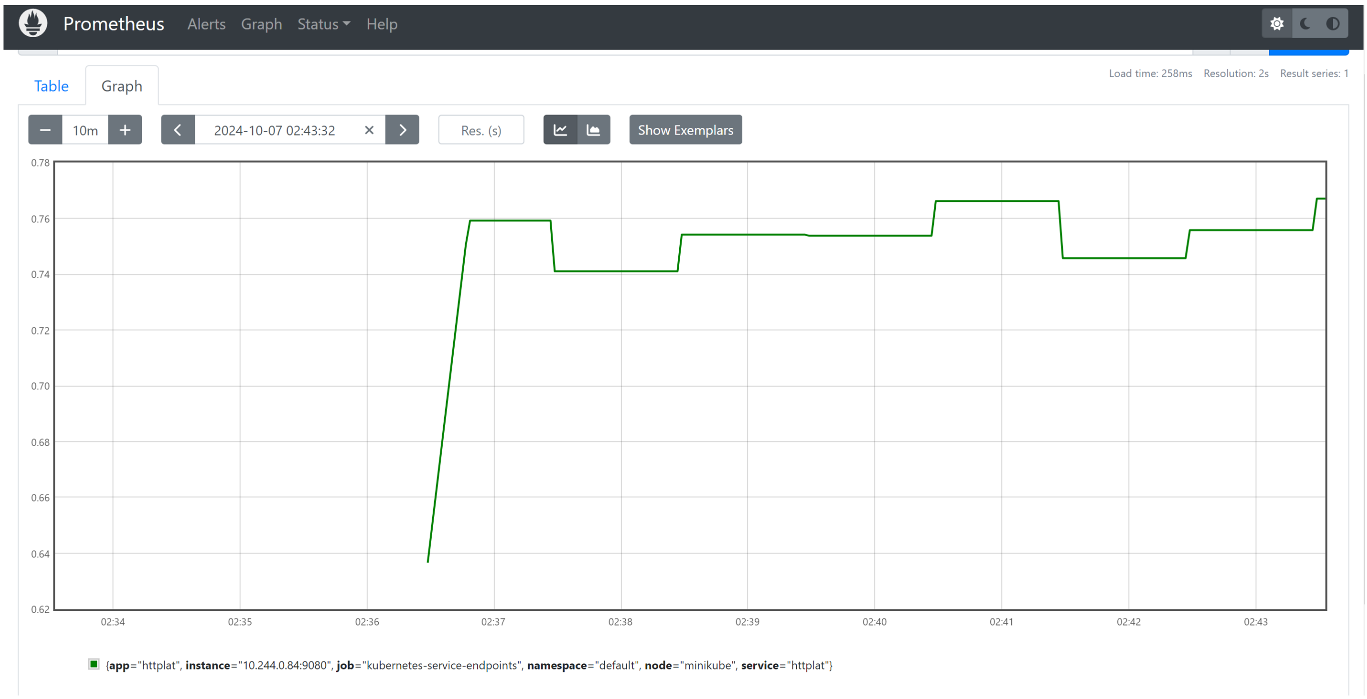
Port forwarding for Prometheus UI

```
kubectl port-forward --address 0.0.0.0 prometheus-server-644d686bc6-r5fp9
9090:9090

# Visit Via blowser
http://18.212.242.215:9090/

# Control via website
rate(httplat_latency_seconds_sum[2m])/rate(httplat_latency_seconds_count[2m])
```

Result:



Load time: 258ms Resolution: 2s Result series: 1

🟩 {**app**="httplat", **instance**="10.244.0.84:9080", **job**="kubernetes-service-endpoints", **namespace**="default", **node**="minikube", **service**="httplat"}

Difference between two ways:

> Port forwarding through a Pod: This method directly maps the service port inside the Pod to a specific port on your local machine. It is suitable when you want to temporarily debug or access a service within a particular Pod, or when you need to connect directly to a specific Pod. Port forwarding through a Service: Instead of connecting directly to the Pod, you connect to the Service of Prometheus. This means you can access multiple Prometheus Pods via the Service. It is suitable for scenarios where you need to access services through a Kubernetes Service, such as load-balancing across multiple replicas. For testing convenience, method 1 is chosen.

6. HPA

Generating and launch a load test

New build and edit the file

```
nano loadtest-job.yaml
```

Edit by:

```
apiVersion: batch/v1
kind: Job
metadata:
  generateName: loadtest
spec:
  template:
    spec:
      containers:
      - name: siege
        image: schoolofdevops/loadtest:v1
        command: ["siege",  "--concurrent=15", "--benchmark", "--time=4m",
"http://192.168.49.2:31055"] # FILL IN: rng or hasher
      restartPolicy: Never
  backoffLimit: 4
```

Lanuch

```
kubectl create -f loadtest-job.yaml
```

Get info of jobs

```
kubectl get jobs
```

Result

```
NAME            STATUS    COMPLETIONS   DURATION   AGE
loadtestmppdj   Running   0/1           9s         9s
```

Check for the PodID:

```
kubectl describe job loadtestmppdj
# Result
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  50s   job-controller  Created pod: loadtestmppdj-zm46f
```

Check the load test output

```
kubectl get pods #find podid
kubectl logs -f loadtestmppdj-zm46f
```

Result

```
** SIEGE 3.0.8
** Preparing 15 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.

Transactions:                 119365 hits
Availability:                 100.00 %
Elapsed time:                 239.48 secs
Data transferred:               4.10 MB
Response time:                  0.03 secs
Transaction rate:             498.43 trans/sec
Throughput:                     0.02 MB/sec
Concurrency:                   14.96
Successful transactions:      119365
Failed transactions:               0
Longest transaction:            0.23
Shortest transaction:           0.00

FILE: /var/log/siege.log
You can disable this annoying message by editing
the .siegerc file in your home directory; change
the directive 'show-logfile' to false.
```

Create the autoscaling policy

```
nano hpa.yaml
# Edit in the file:
metadata:
  name: rng # optional: rng, hasher
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: rng # optional: rng, hasher
```

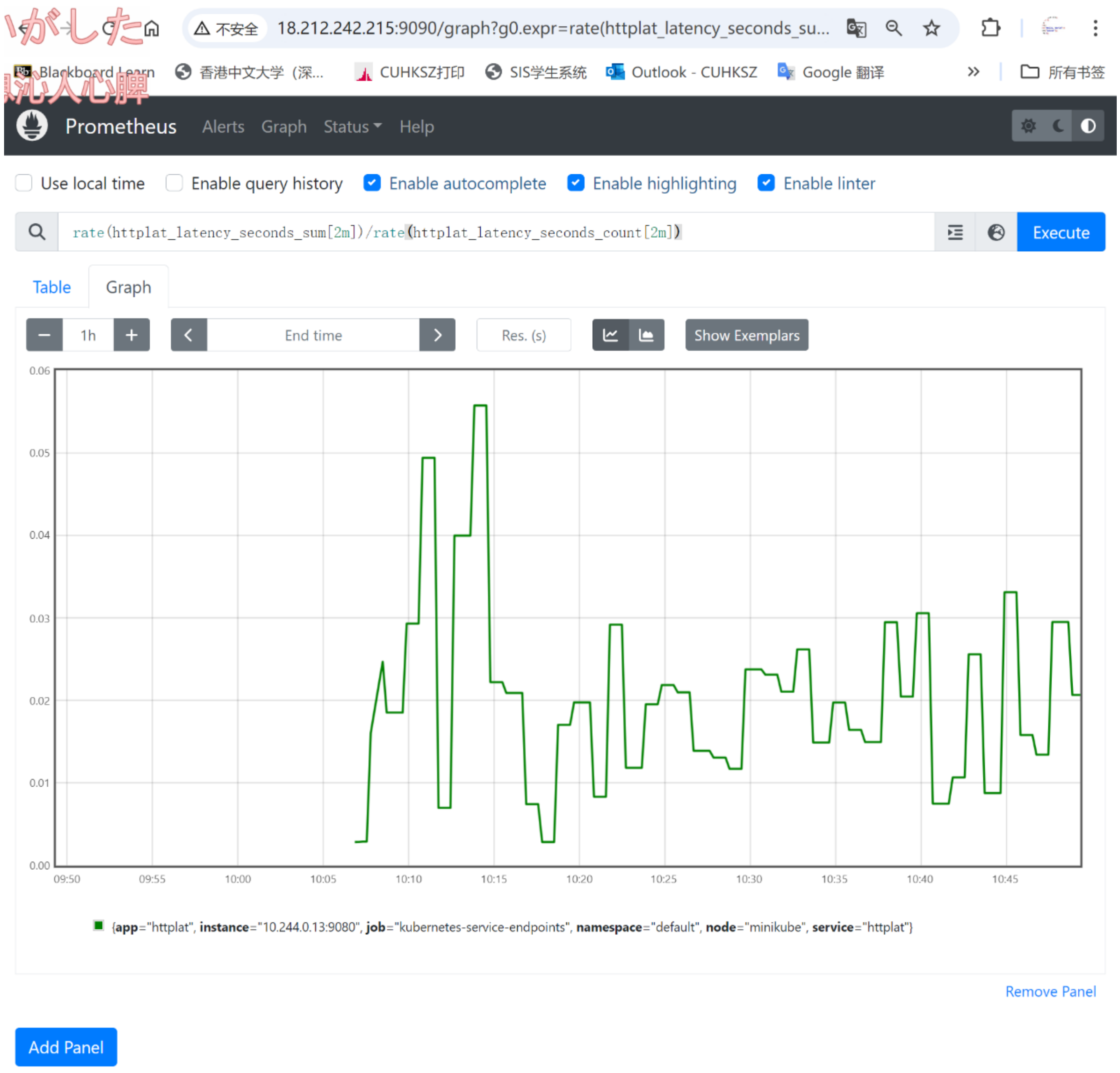
Configer the port

```
kubectl port-forward --address 0.0.0.0 prometheus-server-644d686bc6-r5fp9
9090:9090
```

Visit Via blowser and Configure:

```
http://18.212.242.215:9090/
```

```
rate(httplat_latency_seconds_sum[2m])/rate(httplat_latency_seconds_count[2m])
```



[ ] Use local time [ ] Enable query history [✓ Enable autocomplete] [✓ Enable highlighting] [✓ Enable linter]

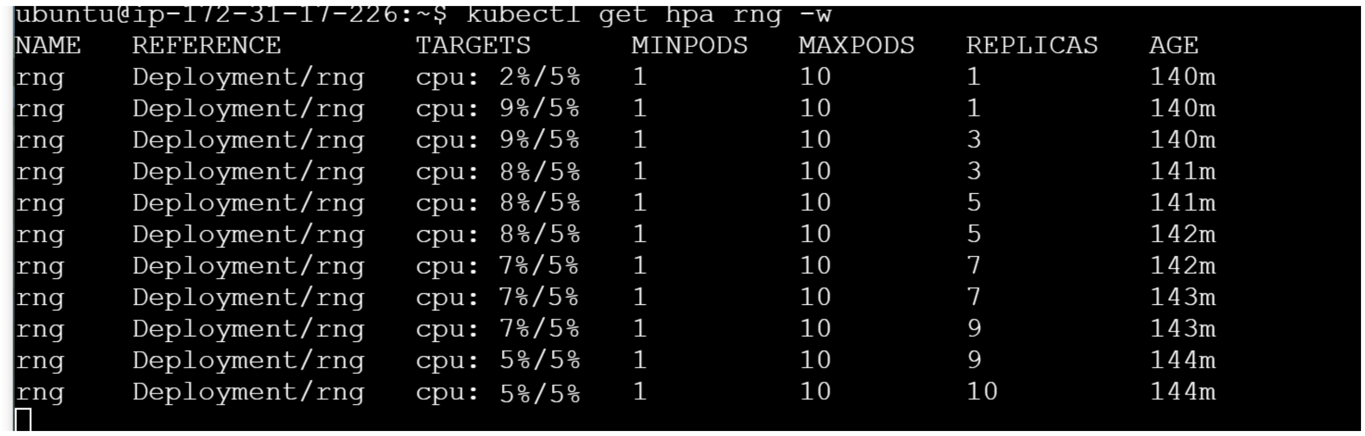
```
rate(httplat_latency_seconds_sum[2m])/rate(httplat_latency_seconds_count[2m])
```

🟩 {app="httplat", instance="10.244.0.13:9080", job="kubernetes-service-endpoints", namespace="default", node="minikube", service="httplat"}

The load curve is shown like this.

Real-time Update for HPA in CMD:

```
kubectl apply -f hpa.yaml
kubectl describe hpa rng
kubectl get hpa rng -w
```



```
ubuntu@ip-172-31-17-226:~$ kubectl get hpa rng -w
NAME   REFERENCE          TARGETS     MINPODS   MAXPODS   REPLICAS   AGE
rng    Deployment/rng     cpu: 2%/5%   1         10        1          140m
rng    Deployment/rng     cpu: 9%/5%   1         10        1          140m
rng    Deployment/rng     cpu: 9%/5%   1         10        3          140m
rng    Deployment/rng     cpu: 8%/5%   1         10        3          141m
rng    Deployment/rng     cpu: 8%/5%   1         10        5          141m
rng    Deployment/rng     cpu: 8%/5%   1         10        5          142m
rng    Deployment/rng     cpu: 7%/5%   1         10        7          142m
rng    Deployment/rng     cpu: 7%/5%   1         10        7          143m
rng    Deployment/rng     cpu: 7%/5%   1         10        9          143m
rng    Deployment/rng     cpu: 5%/5%   1         10        9          144m
rng    Deployment/rng     cpu: 5%/5%   1         10        10         144m
```

The result is shown as the image above.

Question: Does evaluation based on CPU make sence?

> Evaluating whether to scale `rng` and `hasher` services based on CPU utilization depends on their characteristics and bottlenecks.
>
> 1. Rationality of Scaling Based on CPU Utilization In Kubernetes, horizontal autoscaling (HPA) based on CPU utilization is common as many applications rely on CPU. Hasher service is compute-intensive, calculating hash values, which makes scaling based on CPU reasonable. As workload increases, CPU load will rise, enhancing performance. However, for `rng`, which generates random bytes, its bottleneck might not be CPU but other factors like I/O. Scaling based on CPU utilization might not reflect its actual load.
>
> 2. Alternatives Consider scaling based on Request latency, as it reflects performance better than CPU. If latency exceeds a threshold (e.g., 200ms), scaling is triggered. Another option is Request rate (QPS) which tracks the number of requests per second handled by the service. Other methods include Memory usage for services with different bottlenecks or Error rate, which monitors the percentage of failed requests.

============ This is the End of Assignment 2 Report ============