National College of Ireland

Configuration Manual

MSC Research Project
Cloud Computing

Rushikesh Sawant
Student ID: 23303891

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Rushikesh Sawant

Student ID: 23303891

Programme: Cloud Computing **Year:** 2024-2025

Module: MSC Research Project

Supervisor: Sean Heeney

Submission Due Date: 11/08/2025

Project Title: Optimizing Workload Scheduling and Cost Efficiency Using Cloud Orchestration Frameworks in Multi-Cloud Environments

Word Count: **3782 Page Count: 47**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rushikesh Sawant

Date: 11/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

1.Install WSL on your windows machine.

2. Install AWS CLI:

1. sudo apt update && sudo apt upgrade -y
2. sudo apt install -y curl unzip tar git build-essential
3. curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o awscliv2.zip
4. unzip awscliv2.zip
5. sudo ./aws/install
6. aws --version

3. Install KubeCtl:

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

2. chmod +x kubectl
3. sudo mv kubectl /usr/local/bin

4. Install eksctl:

curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

4. AWS Credentials & IAM Setup:

aws configure

aws sts get-caller-identity

5. Create IAM Roles (Only once)

```
aws iam create-role --role-name eksClusterRole \
 --assume-role-policy-document file://<(cat <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": { "Service": "eks.amazonaws.com" },
   "Action": "sts:AssumeRole"
```

```
    }
  ]
}
EOF
)
```

7.Attach Policies:

```
aws iam attach-role-policy \
--role-name eksClusterRole \
--policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

8. Create Node Role:

```
aws iam create-role --role-name AmazonEKSNodeRole \
 --assume-role-policy-document file://<(cat <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
   {
    "Effect": "Allow",
    "Principal": { "Service": "ec2.amazonaws.com" },
    "Action": "sts:AssumeRole"
   }
 ]
}
EOF
)
```

9. Attach Node Role Policies

```
aws iam attach-role-policy --role-name AmazonEKSNodeRole \
```

```
--policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy

aws iam attach-role-policy --role-name AmazonEKSNodeRole \
--policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly

aws iam attach-role-policy --role-name AmazonEKSNodeRole \
--policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

10. Create the EKS Cluster:

```
eksctl create cluster \
--name wsl-eks-cluster \
--version 1.28 \
--region us-east-1 \
--nodegroup-name linux-nodes \
--node-type t3.medium \
--nodes 2 \
--nodes-min 1 \
--nodes-max 3 \
--managed \
--with-oidc \
--ssh-access \
```

11. Confirm Kubernetes Nodes

```
kubectl get nodes
kubectl get pods --all-namespaces
```

12. Verify the context:

```
kubectl config get-contexts
```

12. Helm Installation Steps for WSL

==========

1) Update WSL Packages

---

sudo apt update && sudo apt upgrade -y

2) Install Helm via Official Script (Recommended)

---

curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

3) Verify Installation

---

helm version

13. Datadog Monitoring Setup on EKS from WSL

==========

0) Prerequisites

- AWS CLI installed and configured (`aws configure`)
- kubectl installed and pointing to your EKS cluster
- Helm installed (see helm_install_wsl.txt)

1) Add Datadog Helm Repository

---

helm repo add datadog https://helm.datadoghq.com

helm repo update

kubectl create namespace datadog

2) Create Datadog Secrets

Replace with your actual API and APP keys

kubectl -n datadog create secret generic datadog-secret --from-literal=api-key="<YOUR_DD_API_KEY>"

kubectl -n datadog create secret generic datadog-appkey --from-literal=app-key="<YOUR_DD_APP_KEY>"

3) Create datadog-values.yaml

Example minimal configuration:

datadog:

apiKeyExistingSecret: datadog-secret

appKeyExistingSecret: datadog-appkey

site: datadoghq.com # Change to your Datadog site

clusterName: wsl-eks-cluster # Your EKS cluster name

criSocketPath: /var/run/containerd/containerd.sock

logs:

enabled: true

containerCollectAll: true

apm:

socketEnabled: true

processAgent:

enabled: true

processCollection: true

kubeStateMetricsCore:

enabled: true

kubeStateMetricsEnabled: false

```
  tags:
    - env:dev
    - cluster:eks

clusterAgent:
  enabled: true
  rbac:
    create: true
```

4) Install Datadog with Helm

```
helm upgrade --install datadog datadog/datadog   --namespace datadog   -f datadog-values.yaml
```

5) Verify the Deployment

```
kubectl -n datadog get pods
kubectl -n datadog get svc
kubectl -n datadog exec -it deploy/datadog-cluster-agent -- agent status
kubectl -n datadog exec -it <datadog-agent-pod> -- agent status
```

6) Optional: Enable HPA/External Metrics

```
# Requires APP key in secret
Add to datadog-values.yaml:
clusterAgent:
  metricsProvider:
    enabled: true
```

14. Grafana Setup on EKS from WSL:

1) Add Grafana Helm Repository

---

helm repo add grafana https://grafana.github.io/helm-charts

helm repo update

kubectl create namespace grafana

2) Create grafana-values.yaml

Example configuration for basic Grafana installation

adminUser: admin

adminPassword: admin123 # Change this in production

service:

 type: LoadBalancer

persistence:

 enabled: true

 storageClassName: gp2 # Adjust based on your EKS storage class

 size: 10Gi

3) Install Grafana with Helm

---

helm upgrade --install grafana grafana/grafana --namespace grafana -f grafana-values.yaml

4) Verify the Deployment

---

kubectl -n grafana get pods

kubectl -n grafana get svc

Note the EXTERNAL-IP for the grafana service

5) Access Grafana

- Use the LoadBalancer EXTERNAL-IP from `kubectl get svc`.

- Login with the admin username and password from grafana-values.yaml.

15. Deploy App + PostgreSQL:

Flask API (simple endpoint that hits DB)

PostgreSQL (in-cluster)

PostgreSQL Deployment (in-cluster):

helm repo add bitnami https://charts.bitnami.com/bitnami

helm install my-postgres bitnami/postgresql \

--set auth.username=myuser \

--set auth.password=mypassword \

--set auth.database=mydb

16. Store the DB connections:

export DB_HOST=my-postgres.default.svc.cluster.local

export DB_USER=myuser

export DB_PASSWORD=mypassword

export DB_NAME=mydb

17. Create app.py

app.py

from flask import Flask

import psycopg2, time

app = Flask(__name__)

```
@app.route("/")
def index():
    start = time.time()
    conn = psycopg2.connect(
        host=os.getenv("DB_HOST"),
        dbname=os.getenv("DB_NAME"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"))
    cur = conn.cursor()
    cur.execute("SELECT 1;")
    cur.fetchall()
    cur.close()
    conn.close()
    return f"Query OK - Time: {round((time.time() - start)*1000, 2)} ms\n"
```

18. Create Docker file:

```
FROM python:3.9
WORKDIR /app
COPY . .
RUN pip install flask psycopg2-binary
CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]
```

19. Create deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: flask-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-api
  template:
    metadata:
      labels:
        app: flask-api
    spec:
      containers:
        - name: flask-api
          image: your-dockerhub-username/flask-db-api
          ports:
            - containerPort: 5000
          env:
            - name: DB_HOST
              value: "my-postgres.default.svc.cluster.local"
            - name: DB_USER
              value: "myuser"
            - name: DB_PASSWORD
              value: "mypassword"
            - name: DB_NAME
              value: "mydb"
```

20. Create Service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: flask-api-service
spec:
  type: LoadBalancer
  selector:
    app: flask-api
  ports:
    - port: 80
      targetPort: 5000
```

Once deployed, Wait for 1-2mins to get external Ip

kubectl get svc flask-api-service

21. Build and Push:

docker build -t yourdockerhub/test-api .

docker push yourdockerhub/test-api

Deploy:

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

22. Get External IP & Test:

kubectl get svc flask-api-service

Curl http://<IP>

23. Automate testing with K6:

sudo apt update

sudo apt install -y gnupg software-properties-common

curl -s https://dl.k6.io/key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/k6-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list

sudo apt update

sudo apt install k6

vi loadtest.js

```
import http from 'k6/http';

import { sleep, check } from 'k6';


export let options = {

  stages: [

    { duration: '10s', target: 10 },

    { duration: '20s', target: 20 },

    { duration: '10s', target: 0 },

  ],

};



export default function () {
```

```
  const res = http.get(external ip');

  check(res, {

    'is status 200': (r) => r.status === 200,

    'no 500 error': (r) => !r.body.includes('500 Error'),

  });

  sleep(1);

}
```

k6 run loadtest.js

after running the script you'll get the result of latency and get the graph run the med and max load script as well.

1. GCP GKE Cluster Setup from WSL:

1) Install Google Cloud SDK in WSL

---

sudo apt update && sudo apt install -y apt-transport-https ca-certificates gnupg

echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] https://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list

curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/cloud.google.gpg

sudo apt update && sudo apt install -y google-cloud-sdk

Verify installation

gcloud version

2) Authenticate and Set Project

---

gcloud auth login

If you don't have a project yet:

gcloud projects create my-gke-project

gcloud config set project my-gke-project

Set default project and compute zone

gcloud config set project <PROJECT_ID>

gcloud config set compute/zone us-central1-a

3) Enable Required APIs

---

gcloud services enable container.googleapis.com compute.googleapis.com

4) Create a GKE Cluster

Standard mode

gcloud container clusters create my-gke-cluster --zone us-central1-a --num-nodes 3 --machine-type e2-medium

OR Autopilot mode

gcloud container clusters create-auto my-gke-cluster --region us-central1

5) Configure kubectl

---

gcloud container clusters get-credentials my-gke-cluster --zone us-central1-a --project <PROJECT_ID>

Verify nodes

kubectl get nodes

6) Node Pool Management

Create a new node pool

gcloud container node-pools create extra-pool --cluster=my-gke-cluster --zone=us-central1-a --num-nodes=2 --machine-type=e2-standard-2

Resize a node pool

gcloud container clusters resize my-gke-cluster --node-pool=default-pool --num-nodes=4 --zone=us-central1-a

7) Deploy a Test Application

---

kubectl create deployment hello-world --image=gcr.io/google-samples/hello-app:1.0

kubectl expose deployment hello-world --type=LoadBalancer --port 80

kubectl get svc hello-world

8) Add Datadog Helm Repository

---

helm repo add datadog https://helm.datadoghq.com

helm repo update

kubectl create namespace datadog

9) Create Datadog Secrets

Replace with your actual keys from https://app.datadoghq.com/organization-settings/api-keys

kubectl -n datadog create secret generic datadog-secret --from-literal=api-key="<YOUR_DD_API_KEY>"

APP key is optional but needed for certain features (e.g., HPA, external metrics)

kubectl -n datadog create secret generic datadog-appkey --from-literal=app-key="<YOUR_DD_APP_KEY>"

10) Create datadog-values.yaml

Minimal but production-ready example for EKS/GKE (containerd runtime)

```
datadog:
  apiKeyExistingSecret: datadog-secret
  appKeyExistingSecret: datadog-appkey   # remove if APP key not created
  site: datadoghq.com                    # Change to your Datadog site (datadoghq.com, datadoghq.eu,
us3.datadoghq.com, etc.)
  clusterName: my-cluster-name           # Replace with your cluster name
  criSocketPath: /var/run/containerd/containerd.sock

  logs:
    enabled: true
    containerCollectAll: true

  apm:
    socketEnabled: true

  processAgent:
    enabled: true
    processCollection: true

  kubeStateMetricsCore:
    enabled: true
  kubeStateMetricsEnabled: false

  tags:
    - env:dev
    - cluster:k8s

clusterAgent:
  enabled: true
  rbac:
    create: true
```

admissionController:

enabled: false

11) Install Datadog with Helm

---

helm upgrade --install datadog datadog/datadog --namespace datadog -f datadog-values.yaml

12) Verify the Deployment

---

kubectl -n datadog get pods

kubectl -n datadog get svc

Check cluster agent status

kubectl -n datadog exec -it deploy/datadog-cluster-agent -- agent status

Check node agent status

kubectl -n datadog exec -it <datadog-agent-pod> -- agent status

13) View Data in Datadog

- Log into your Datadog account

- Go to Infrastructure -> Kubernetes

- View your cluster, nodes, pods, metrics, and logs

14) Grafana Setup on EKS from WSL:

Add Grafana Helm Repository

---

helm repo add grafana https://grafana.github.io/helm-charts

helm repo update

kubectl create namespace grafana

15. Create grafana-values.yaml

Example configuration for basic Grafana installation

adminUser: admin

adminPassword: admin123 # Change this in production

service:

  type: LoadBalancer

persistence:

  enabled: true

  storageClassName: gp2 # Adjust based on your EKS storage class

  size: 10Gi

16. Install Grafana with Helm

---

helm upgrade --install grafana grafana/grafana --namespace grafana -f grafana-values.yaml

17. Verify the Deployment

---

kubectl -n grafana get pods

kubectl -n grafana get svc

Note the EXTERNAL-IP for the grafana service

18. Access Grafana

- Use the LoadBalancer EXTERNAL-IP from `kubectl get svc`.

- Login with the admin username and password from grafana-values.yaml.

Deploy App + PostgreSQL:

Flask API (simple endpoint that hits DB)

PostgreSQL (in-cluster)

19. PostgreSQL Deployment (in-cluster):

helm repo add bitnami https://charts.bitnami.com/bitnami

helm install my-postgres bitnami/postgresql \

--set auth.username=myuser \

--set auth.password=mypassword \

--set auth.database=mydb

20. Store the DB connections:

export DB_HOST=my-postgres.default.svc.cluster.local

export DB_USER=myuser

export DB_PASSWORD=mypassword

export DB_NAME=mydb

21. Create app.py

```
# app.py
from flask import Flask
import psycopg2, time

app = Flask(__name__)

@app.route("/")
def index():
```

```
    start = time.time()
    conn = psycopg2.connect(
        host=os.getenv("DB_HOST"),
        dbname=os.getenv("DB_NAME"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"))
    cur = conn.cursor()
    cur.execute("SELECT 1;")
    cur.fetchall()
    cur.close()
    conn.close()
    return f"Query OK - Time: {round((time.time() - start)*1000, 2)} ms\n"
```

22. Create Docker file:

```
FROM python:3.9
WORKDIR /app
COPY . .
RUN pip install flask psycopg2-binary
CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]
```

-23. Create deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-api
spec:
  replicas: 2
```

```
  selector:
    matchLabels:
      app: flask-api
  template:
    metadata:
      labels:
        app: flask-api
    spec:
      containers:
        - name: flask-api
          image: your-dockerhub-username/flask-db-api
          ports:
            - containerPort: 5000
          env:
            - name: DB_HOST
              value: "my-postgres.default.svc.cluster.local"
            - name: DB_USER
              value: "myuser"
            - name: DB_PASSWORD
              value: "mypassword"
            - name: DB_NAME
              value: "mydb"
```

24. Create Service.yaml:

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: flask-api-service
spec:
  type: LoadBalancer
  selector:
    app: flask-api
  ports:
    - port: 80
      targetPort: 5000
```

Once deployed, Wait for 1-2mins to get external Ip

kubectl get svc flask-api-service

25. Build and Push:

docker build -t yourdockerhub/test-api .

docker push yourdockerhub/test-api

26. Deploy:

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

27. Get External IP & Test:

kubectl get svc flask-api-service

Curl http://<IP>

28. Automate testing with K6:

```
sudo apt update

sudo apt install -y gnupg software-properties-common

curl -s https://dl.k6.io/key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/k6-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list

sudo apt update

sudo apt install k6


vi loadtest.js

import http from 'k6/http';

import { sleep, check } from 'k6';


export let options = {

  stages: [

    { duration: '10s', target: 10 },

    { duration: '20s', target: 20 },

    { duration: '10s', target: 0 },

  ],

};




export default function () {

  const res = http.get(external ip');

  check(res, {

    'is status 200': (r) => r.status === 200,
```

```
    'no 500 error': (r) => !r.body.includes('500 Error'),
  });
  sleep(1);
}
```

k6 run loadtest.js

after running the script you'll get the result of latency and get the graph run the med and max load script as well.

3. AZURE SETUP:

1) Install Azure CLI in WSL

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

# Verify installation
az version
```

2) Authenticate with Azure

```
az login

# Optional: set default subscription
az account list --output table
az account set --subscription "<SUBSCRIPTION_NAME_OR_ID>"
```

3) Create Resource Group

```
az group create --name myResourceGroup --location eastus
```

4) Create AKS Cluster

Standard cluster with system-assigned managed identity

az aks create --resource-group myResourceGroup --name myAKSCluster --node-count 3 --enable-addons monitoring --generate-ssh-keys

Optional: Specify node size and Kubernetes version

--node-vm-size Standard_DS2_v2

--kubernetes-version 1.28.3

5) Get AKS Credentials for kubectl

---

az aks get-credentials --resource-group myResourceGroup --name myAKSCluster

Verify cluster connection

kubectl get nodes

6) Node Pool Management

Create an additional node pool

az aks nodepool add --resource-group myResourceGroup --cluster-name myAKSCluster --name extraPool --node-count 2 --node-vm-size Standard_DS2_v2

Scale a node pool

az aks nodepool scale --resource-group myResourceGroup --cluster-name myAKSCluster --name nodepool1 --node-count 4

7) Deploy a Test Application

---

kubectl create deployment hello-world --image=mcr.microsoft.com/azuredocs/aks-helloworld:v1

kubectl expose deployment hello-world --type=LoadBalancer --port=80

kubectl get svc hello-world

Access via EXTERNAL-IP from 'kubectl get svc'

8. **DATADOG SETUP:**

Add Datadog Helm Repository

---

helm repo add datadog https://helm.datadoghq.com

helm repo update

kubectl create namespace datadog

9. Create Datadog Secrets

Replace with your actual keys from https://app.datadoghq.com/organization-settings/api-keys

kubectl -n datadog create secret generic datadog-secret --from-literal=api-key="<YOUR_DD_API_KEY>"

APP key is optional but needed for certain features (e.g., HPA, external metrics)

kubectl -n datadog create secret generic datadog-appkey --from-literal=app-key="<YOUR_DD_APP_KEY>"

10. Create datadog-values.yaml

Minimal but production-ready example for EKS/GKE (containerd runtime)

datadog:

apiKeyExistingSecret: datadog-secret

appKeyExistingSecret: datadog-appkey # remove if APP key not created

site: datadoghq.com # Change to your Datadog site (datadoghq.com, datadoghq.eu, us3.datadoghq.com, etc.)

clusterName: my-cluster-name # Replace with your cluster name

criSocketPath: /var/run/containerd/containerd.sock

```
  logs:
    enabled: true
    containerCollectAll: true

  apm:
    socketEnabled: true

  processAgent:
    enabled: true
    processCollection: true

  kubeStateMetricsCore:
    enabled: true
  kubeStateMetricsEnabled: false

  tags:
    - env:dev
    - cluster:k8s

clusterAgent:
  enabled: true
  rbac:
    create: true

admissionController:
  enabled: false
```

11. Install Datadog with Helm

```
helm upgrade --install datadog datadog/datadog   --namespace datadog   -f datadog-values.yaml
```

12. Verify the Deployment

---

kubectl -n datadog get pods

kubectl -n datadog get svc

Check cluster agent status

kubectl -n datadog exec -it deploy/datadog-cluster-agent -- agent status

Check node agent status

kubectl -n datadog exec -it <datadog-agent-pod> -- agent status

13) View Data in Datadog

- Log into your Datadog account

- Go to Infrastructure -> Kubernetes

- View your cluster, nodes, pods, metrics, and logs

14.Grafana Setup on EKS from WSL:

Add Grafana Helm Repository

---

helm repo add grafana https://grafana.github.io/helm-charts

helm repo update

kubectl create namespace grafana

Create grafana-values.yaml

Example configuration for basic Grafana installation

adminUser: admin

adminPassword: admin123 # Change this in production

service:

type: LoadBalancer

persistence:

enabled: true

storageClassName: gp2 # Adjust based on your EKS storage class

size: 10Gi

15. Install Grafana with Helm

---

helm upgrade --install grafana grafana/grafana --namespace grafana -f grafana-values.yaml

16. Verify the Deployment

---

kubectl -n grafana get pods

kubectl -n grafana get svc

Note the EXTERNAL-IP for the grafana service

17. Access Grafana

- Use the LoadBalancer EXTERNAL-IP from `kubectl get svc`.

- Login with the admin username and password from grafana-values.yaml.

18. PostgreSQL Deployment (in-cluster):

helm repo add bitnami https://charts.bitnami.com/bitnami

helm install my-postgres bitnami/postgresql \

--set auth.username=myuser \

--set auth.password=mypassword \

```
  --set auth.database=mydb
```

19. Store the DB connections:

```
export DB_HOST=my-postgres.default.svc.cluster.local
export DB_USER=myuser
export DB_PASSWORD=mypassword
export DB_NAME=mydb
```

20. Create app.py

```
# app.py
from flask import Flask
import psycopg2, time

app = Flask(__name__)

@app.route("/")
def index():
    start = time.time()
    conn = psycopg2.connect(
        host=os.getenv("DB_HOST"),
        dbname=os.getenv("DB_NAME"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"))
    cur = conn.cursor()
    cur.execute("SELECT 1;")
    cur.fetchall()
```

```
    cur.close()
    conn.close()
    return f"Query OK - Time: {round((time.time() - start)*1000, 2)} ms\n"
```

21. Create Docker file:

```
FROM python:3.9
WORKDIR /app
COPY . .
RUN pip install flask psycopg2-binary
CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]
```

22. Create deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-api
  template:
    metadata:
      labels:
        app: flask-api
    spec:
      containers:
```

```
      - name: flask-api
        image: your-dockerhub-username/flask-db-api
        ports:
          - containerPort: 5000
        env:
          - name: DB_HOST
            value: "my-postgres.default.svc.cluster.local"
          - name: DB_USER
            value: "myuser"
          - name: DB_PASSWORD
            value: "mypassword"
          - name: DB_NAME
            value: "mydb"
```

23. Create Service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: flask-api-service
spec:
  type: LoadBalancer
  selector:
    app: flask-api
  ports:
    - port: 80
```

targetPort: 5000

Once deployed, Wait for 1-2mins to get external Ip

kubectl get svc flask-api-service

24. Build and Push:

docker build -t yourdockerhub/test-api .

docker push yourdockerhub/test-api

25. Deploy:

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

26. Get External IP & Test:

kubectl get svc flask-api-service

Curl http://<IP>

27. Automate testing with K6:

sudo apt update

sudo apt install -y gnupg software-properties-common

curl -s https://dl.k6.io/key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/k6-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list

sudo apt update

sudo apt install k6

vi loadtest.js

```
import http from 'k6/http';
import { sleep, check } from 'k6';


export let options = {
  stages: [
    { duration: '10s', target: 10 },
    { duration: '20s', target: 20 },
    { duration: '10s', target: 0 },
  ],
};



export default function () {
  const res = http.get(external ip');
  check(res, {
    'is status 200': (r) => r.status === 200,
    'no 500 error': (r) => !r.body.includes('500 Error'),
  });
  sleep(1);
}
```

28. k6 run loadtest.js

after running the script you'll get the result of latency and get the graph run the med and max load script as well.

IBM Cloud Kubernetes Service (IKS) Setup from WSL

==========

1) Install IBM Cloud CLI in WSL

---

curl -fsSL https://clis.cloud.ibm.com/install/linux | sh

Verify installation

ibmcloud version

2) Install Required IBM Cloud CLI Plugins

Log in to IBM Cloud

ibmcloud login --sso

Install Kubernetes Service plugin

ibmcloud plugin install container-service

Install Container Registry plugin (optional, for pushing images)

ibmcloud plugin install container-registry

Verify plugins

ibmcloud plugin list

3) Choose Target Region

List available regions

ibmcloud regions

Target a specific region (example: us-south)

ibmcloud target -r us-south

4) Create a Resource Group (optional)

---

ibmcloud resource group-create myResourceGroup

Set the target resource group

ibmcloud target -g myResourceGroup

5) Create an IKS Cluster

Standard cluster

ibmcloud ks cluster create classic --name myIKSCluster --zone dal10 --flavor b3c.4x16 --workers 3

Or VPC cluster

ibmcloud ks cluster create vpc-gen2 # --name myIKSCluster # --zone us-south-1 # --flavor bx2.4x16 # --workers 3 # --vpc-id <your-vpc-id>

Check cluster provisioning status

ibmcloud ks cluster ls

6) Configure kubectl Access

Download the kubeconfig

ibmcloud ks cluster config --cluster myIKSCluster

Verify

kubectl get nodes

7) Node Management

List worker nodes

ibmcloud ks workers --cluster myIKSCluster

Add worker node

ibmcloud ks worker-add --cluster myIKSCluster --flavor b3c.4x16

8) Deploy a Test Application

---

kubectl create deployment hello-world --image=icr.io/hello/hello-world

kubectl expose deployment hello-world --type=LoadBalancer --port=80

kubectl get svc hello-world

Access the app using the EXTERNAL-IP

Helm Installation Steps for WSL

==========

9. Update WSL Packages

---

sudo apt update && sudo apt upgrade -y

10) Install Helm via Official Script (Recommended)

---

curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

11) Verify Installation

---

helm version

Datadog Monitoring Setup on EKS from WSL

==========

1) Add Datadog Helm Repository

---

helm repo add datadog https://helm.datadoghq.com

helm repo update

kubectl create namespace datadog

2) Create Datadog Secrets

Replace with your actual API and APP keys

kubectl -n datadog create secret generic datadog-secret --from-literal=api-key="<YOUR_DD_API_KEY>"

kubectl -n datadog create secret generic datadog-appkey --from-literal=app-key="<YOUR_DD_APP_KEY>"

3) Create datadog-values.yaml

Example minimal configuration:

```
datadog:
  apiKeyExistingSecret: datadog-secret
  appKeyExistingSecret: datadog-appkey
  site: datadoghq.com              # Change to your Datadog site
  clusterName: wsl-eks-cluster     # Your EKS cluster name
  criSocketPath: /var/run/containerd/containerd.sock

  logs:
    enabled: true
    containerCollectAll: true

  apm:
    socketEnabled: true
```

```
  processAgent:
    enabled: true
    processCollection: true

  kubeStateMetricsCore:
    enabled: true
  kubeStateMetricsEnabled: false

  tags:
    - env:dev
    - cluster:eks

clusterAgent:
  enabled: true
  rbac:
    create: true
```

4) Install Datadog with Helm

```
helm upgrade --install datadog datadog/datadog   --namespace datadog   -f datadog-values.yaml
```

5) Verify the Deployment

```
kubectl -n datadog get pods
kubectl -n datadog get svc
kubectl -n datadog exec -it deploy/datadog-cluster-agent -- agent status
kubectl -n datadog exec -it <datadog-agent-pod> -- agent status
```

6) Enable HPA/External Metrics

Requires APP key in secret

Add to datadog-values.yaml:

```
clusterAgent:
  metricsProvider:
    enabled: true
```

Grafana Setup on EKS from WSL:

1) Add Grafana Helm Repository

```
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update
kubectl create namespace grafana
```

2) Create grafana-values.yaml

```
# Example configuration for basic Grafana installation

adminUser: admin
adminPassword: admin123   # Change this in production

service:
  type: LoadBalancer

persistence:
  enabled: true
  storageClassName: gp2   # Adjust based on your EKS storage class
  size: 10Gi
```

3) Install Grafana with Helm

```
helm upgrade --install grafana grafana/grafana   --namespace grafana   -f grafana-values.yaml
```

4) Verify the Deployment

---

kubectl -n grafana get pods

kubectl -n grafana get svc

Note the EXTERNAL-IP for the grafana service

5) Access Grafana

- Use the LoadBalancer EXTERNAL-IP from `kubectl get svc`.
- Login with the admin username and password from grafana-values.yaml.

1. **PostgreSQL Deployment (in-cluster):**

helm repo add bitnami https://charts.bitnami.com/bitnami

helm install my-postgres bitnami/postgresql \

--set auth.username=myuser \

--set auth.password=mypassword \

--set auth.database=mydb

2. **Store the DB connections:**

export DB_HOST=my-postgres.default.svc.cluster.local

export DB_USER=myuser

export DB_PASSWORD=mypassword

export DB_NAME=mydb

3. **Create app.py**

app.py

```
from flask import Flask
import psycopg2, time

app = Flask(__name__)

@app.route("/")
def index():
    start = time.time()
    conn = psycopg2.connect(
        host=os.getenv("DB_HOST"),
        dbname=os.getenv("DB_NAME"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"))
    cur = conn.cursor()
    cur.execute("SELECT 1;")
    cur.fetchall()
    cur.close()
    conn.close()
    return f"Query OK - Time: {round((time.time() - start)*1000, 2)} ms\n"
```

4. **Create Docker file:**

```
FROM python:3.9
WORKDIR /app
COPY . .
RUN pip install flask psycopg2-binary
CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]
```

5. **Create deployment.yaml:**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-api
  template:
    metadata:
      labels:
        app: flask-api
    spec:
      containers:
        - name: flask-api
          image: your-dockerhub-username/flask-db-api
          ports:
            - containerPort: 5000
          env:
            - name: DB_HOST
              value: "my-postgres.default.svc.cluster.local"
            - name: DB_USER
              value: "myuser"
```

```
        - name: DB_PASSWORD
          value: "mypassword"
        - name: DB_NAME
          value: "mydb"
```

6. **Create Service.yaml:**

```
apiVersion: v1
kind: Service
metadata:
  name: flask-api-service
spec:
  type: LoadBalancer
  selector:
    app: flask-api
  ports:
    - port: 80
      targetPort: 5000
```

Once deployed, Wait for 1-2mins to get external Ip

```
kubectl get svc flask-api-service
```

7. **Build and Push:**

```
docker build -t yourdockerhub/test-api .
docker push yourdockerhub/test-api
```

8. **Deploy:**

```
kubectl apply -f deployment.yaml
```

kubectl apply -f service.yaml

9. **Get External IP & Test:**

kubectl get svc flask-api-service

Curl http://<IP>

10. **Automate testing with K6:**

sudo apt update

sudo apt install -y gnupg software-properties-common

curl -s https://dl.k6.io/key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/k6-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list

sudo apt update

sudo apt install k6

vi loadtest.js

```
import http from 'k6/http';

import { sleep, check } from 'k6';


export let options = {

  stages: [

    { duration: '10s', target: 10 },

    { duration: '20s', target: 20 },

    { duration: '10s', target: 0 },

  ],
```

```
};



export default function () {
  const res = http.get(external ip');
  check(res, {
    'is status 200': (r) => r.status === 200,
    'no 500 error': (r) => !r.body.includes('500 Error'),
  });
  sleep(1);
}
```

11. **k6 run loadtest.js**

after running the script you'll get the result of latency and get the graph run the med and max load script as well.