National College of Ireland

ConfigurationManual

MScResearchProject
MScCloud Computing

Bramha Theja Gadikota
StudentID:23197994

SchoolofComputing
NationalCollegeofIreland

Supervisor: Prof.Shaguna gupta

National College of Ireland
Project Submission Sheet School of Computing

Student Name:	Bramha Theja Gadikota
Student ID:	23197994
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof. Shaguna Gupta
Submission Due Date:	12/12/2023
Project Title:	Configuration Manual
Word Count:	710
Page Count:	10

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:	Bramha Theja Gadikota
Date:	10th December 2024

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

Bramha Theja Gadikota
23197994

INTRODUCTION

This configuration manual provides in-depth, step-by-step instructions for installing, configuring, and deploying all of the software, tools, and files that are necessary for the implementation of the proposed system.

PREREQUISITES

- **Programming**: Python, Flask
- **Containerization**: Docker
- **Cloud Services**: Azure ACR, AKS, AWS, ECR, EKS
- **Tools**: VS Code, Locust, Kubernetes (kubectl)

WORKFLOW

1. **Application Development**:
    - Implement a Flask web application with endpoints for:
        - **Home page** (/): Serves the main web interface.
        - **Start CPU Load** (/start-load): Initiates CPU stress testing with optional duration and core configuration.
        - **Stop CPU Load** (/stop-load): Stops ongoing stress testing.
        - **Status** (/status): Checks if CPU stress testing is running.
    - Use threading and multiprocessing for parallel stress simulation across all CPU cores.
2. **Containerization**:

- Create a Dockerfile to package the Flask application into a container image.
  - Include necessary dependencies, Python environment, and application files.
  - Expose the application on port 5000.
- Build and test the Docker image locally.

3. **Azure Deployment**:
   - **Azure Container Registry (ACR)**:
     - Push the Docker image to ACR for centralized image management.
   - **Azure Kubernetes Service (AKS)**:
     - Deploy the application to an AKS cluster.

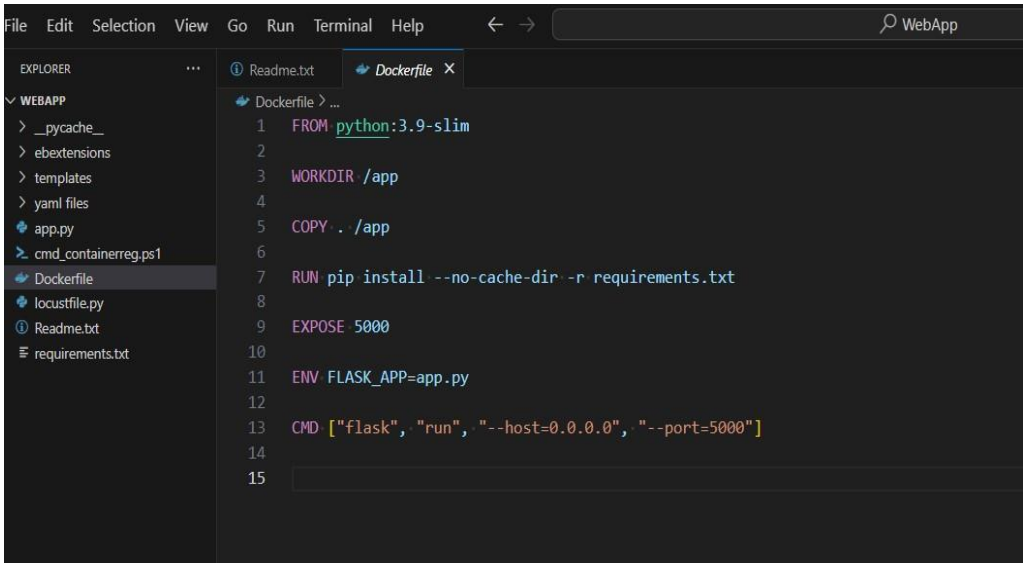
4. **AWS Deployment**:
   - **Elastic Container Registry (ECR)**:
     - Push the Docker image to ECR for centralized image management.
   - **Elastic Kubernetes Service (EKS)**:
     - Deploy the application to an EKS cluster.

5. **Locust**:
   - Load Test

IMPLEMENTATION

CONTAINERIZATION

Dockerfile:



Docker Build Commands:

Build Docker Image:

docker build -t cpuloadgenflask .

Run Docker Locally: docker run -p

5000:5000 -d cpuloadgenflask flask run

AZURE DEPLOYMENT

Azure Container Registry (ACR) Login

to ACR:

az acr login --name cpuloadgenacr

Push Docker Image: docker tag cpuloadgenflask

cpuloadgenacr.azurecr.io/cpuloadgenflask docker push

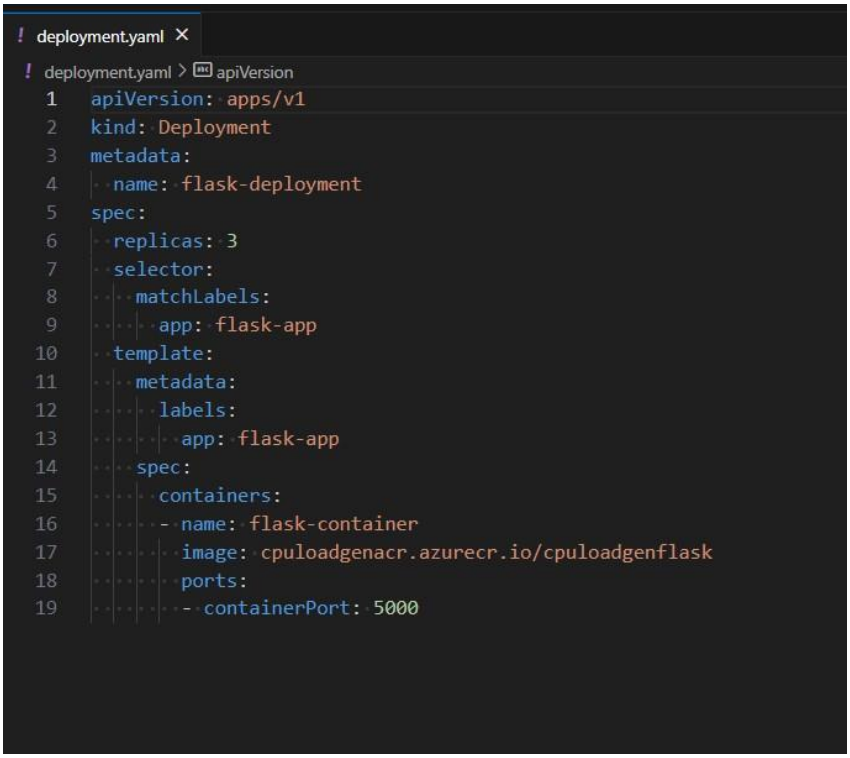
cpuloadgenacr.azurecr.io/cpuloadgenflask

Azure Kubernetes Service (AKS) Connect

to Cluster:

az aks get-credentials --resource-group cpuloadgenrg --name cpuloadgenaks --overwriteexisting

Deploy to AKS:



kubectl apply -f deployment.yaml

service.yaml

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

kubectl apply -f service.yaml

kubectl get service flask-service

```
e
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
flask-service   NodePort   10.100.237.2   <none>        80:30007/TCP   14d
```

AWS DEPLOYMENT

Elastic Container Registry (ECR) Login

to ECR:

aws ecr get-login-password --region eu-west-2 | docker login --username AWS --password-stdin 820242902892.dkr.ecr.eu-west-2.amazonaws.com

Push Docker Image:

docker tag cpuloadgenflask:latest 820242902892.dkr.ecr.eu-west-2.amazonaws.com/cpuloadgenecr:latest

docker push 820242902892.dkr.ecr.eu-west-2.amazonaws.com/cpuloadgenecr:latest

Elastic Kubernetes Service (EKS) Create

EKS Cluster:

eksctl create cluster --name cpuloadgeneks-cluster --version 1.31 --region eu-west-2 -nodegroup-name linux-nodes1 --node-type t3.medium --nodes 3 --nodes-min 1 --nodesmax 10 –managed

Connect to Cluster:

aws eks update-kubeconfig --name cpuloadgeneks-cluster

Deploy to EKS:

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-deployment1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      name: myapp-pod
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-container
        image: 820242902892.dkr.ecr.eu-west-2.amazonaws.com/cpuloadgenecr:latest
        ports:
        - containerPort: 5000
```

kubectl apply -f deployment.yaml --validate=false

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-service1
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 5000
  selector:
    app: flask-app
```

kubectl apply -f service.yaml --validate=false

kubectl get service flask-service1

```
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)        AGE
flask-service1 LoadBalancer   10.100.211.170  a14e82085a0ac4ba5af09fb49438ab86-1898328803.eu-west-2.elb.amazonaws.com   80:32015/TCP   6d6h
```

INSTALLATION OF MONITORING AND VISUALISATION TOOLS

Created a kubernetes service of Grafana and Prometheus using YAML file. GRAFANA

grafana_service1.yaml | grafana_deployment1.yaml

yaml files > grafana_deployment1.yaml > {} spec > {} template > {} spec > [ ] volumes > {} 0

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana1
  namespace: default
  labels:
    app: grafana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
        - name: grafana
          image: grafana/grafana:latest
          ports:
            - containerPort: 3000
          volumeMounts:
            - name: grafana-storage
              mountPath: /var/lib/grafana
      volumes:
        - name: grafana-storage
          emptyDir: {}
```

kubectl apply -f grafana_deployment1.yaml

grafana_service1.yaml grafana_deployment1.yaml

yaml files > grafana_service1.yaml > apiVersion

```yaml
apiVersion: v1
kind: Service
metadata:
  name: grafana1
  namespace: default
  labels:
    app: grafana
spec:
  selector:
    app: grafana
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer

```

kubectl apply -f grafana_service1.yaml

kubectl get service grafana1

```
NAME       TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
grafana1   LoadBalancer   10.100.26.196   a1cd4b035250a42d1a98cdd9ef46ebfe-1636237739.eu-west-2.elb.amazonaws.com     80:31949/TCP   6d5h
```

PROMETHEUS

prometheus_deployment1.yaml | prometheus_service1.yaml

yaml files > prometheus_deployment1.yaml > {} spec > {} template > {} spec > [ ] volumes > {} 1

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: default
  labels:
    app: prometheus
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus:latest
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
            - "--storage.tsdb.path=/prometheus"
            - "--web.enable-lifecycle"
          ports:
            - containerPort: 9090
          volumeMounts:
            - name: prometheus-config-volume
              mountPath: /etc/prometheus
            - name: prometheus-data
              mountPath: /prometheus
      volumes:
        - name: prometheus-config-volume
          configMap:
            name: prometheus-config
        - name: prometheus-data
          emptyDir: {}
```

kubectl apply -f prometheus_deployment1.yaml

prometheus_deployment1.yaml | prometheus_service1.yaml

yaml files > prometheus_service1.yaml > apiVersion

```yaml
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: default
  labels:
    app: prometheus
spec:
  selector:
    app: prometheus
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9090
  type: LoadBalancer
```

kubectl apply -f prometheus_service1.yaml kubectl get service prometheus

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)        AGE
prometheus   LoadBalancer   10.100.180.139  a828207e0a9474c66872e35ba98d30e2-1355634064.eu-west-2.elb.amazonaws.com   80:31082/TCP   6d5h
```

GET ALL RUNNING SERVICE

kubectl get svc -n default

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
flask-service	NodePort	10.100.237.2	<none>	80:30007/TCP	15d
flask-service1	LoadBalancer	10.100.211.170	a14e82085a0ac4ba5af09fb49438ab86-1898328803.eu-west-2.elb.amazonaws.com	80:32015/TCP	6d6h
grafana	NodePort	10.100.62.97	<none>	80:32000/TCP	6d6h
grafana1	LoadBalancer	10.100.26.196	a1cd4b035250a42d1a98cdd9ef46ebfe-1636237739.eu-west-2.elb.amazonaws.com	80:31949/TCP	6d5h
kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	15d
prometheus	LoadBalancer	10.100.180.139	a828207e0a9474c66872e35ba98d30e2-1355634064.eu-west-2.elb.amazonaws.com	80:31082/TCP	6d5h

LINEAR MODEL FOR CPU RESOURCE SCHEDULING

```
def linear_cpu_prediction():
    #linear model for cpu prediction
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_squared_error

    # Train the model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Evaluate the model
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean Squared Error: {mse}")
    return y_pred



def schedule_pod():
    def schedule_pod_resource():
        return linear_cpu_prediction()
    predicted_cpu, predicted_memory = schedule_pod_resource()

    print(f"Predicted CPU Usage: {predicted_cpu}m")
    print(f"Predicted Memory Usage: {predicted_memory}Mi")

    pod = client.V1Pod(
        metadata=client.V1ObjectMeta(name="novelty-pod"),
        spec=client.V1PodSpec(
            containers=[
                client.V1Container(
                    name="novelty-container",
                    image="nginx",
                    resources=client.V1ResourceRequirements(
                        requests={
                            "cpu": f"{predicted_cpu}m",
                            "memory": f"{predicted_memory}Mi"
```