National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cloud Computing

Yashvanth S B
Student ID: 23336641

School of Computing
National College of Ireland

Supervisor: Luis Bernardo Pulido Gaytan

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Yashvanth SB
Student ID:	23336641
Programme:	Research in computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Luis Bernardo Pulido Gaytan
Submission Due Date:	29/08/2025
Project Title:	Reducing Latency Through Adaptive Load Balancing in Multi-Cloud Kubernetes
Word Count:	1773
Page Count:	14

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:	Yashvanth S B
Date:	30th August 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

Yashvanth SB
23336641

1 Introduction

This configuration manual provides the step-by-step, UI-first procedure to reproduce the multi-cloud setup comprising Azure AKS, AWS EKS, NGINX Ingress, Horizontal Pod Autoscaler (HPA), and dual CDNs (Azure CDN and Amazon CloudFront) placed in front of the clusters. It includes common Kubernetes manifests, platform-specific configuration, global DNS routing, monitoring, and JMeter validation.

2 Prerequisites

Required software, accounts, and base configuration:

- macOS with Docker Desktop; tools: kubectl, Helm, JMeter, Minikube, GitHub, and VS Code.
- Cloud accounts: Microsoft Azure and Amazon Web Services (AWS).
- Container image hosted in Docker Hub.
- Kubernetes versions aligned across AKS and EKS.
- NGINX Ingress Controller installed on both clusters.
- Metrics Server installed on both clusters for HPA.

3 Implementation

3.1 Kubernetes Cluster on AWS EKS

Steps:

- Open AWS Console → **EKS** → **Add cluster** → **Create**.
- Add a **managed node group** with instance type t3.medium; set **min** nodes to **2** and **max** nodes to **4**.
- Enable **CloudWatch** logging for node CPU utilization.

```
yashvanthsb@Yashvanths-MacBook-Air multi-cloud-thesis-app % eksctl create cluster \
  --name eks-eu-2 \
  --region eu-west-1 \
  --version 1.31 \
  --with-oidc \
  --nodegroup-name ng-1 \
  --node-type t3.medium \
  --nodes 2 --nodes-min 2 --nodes-max 4 --managed
```

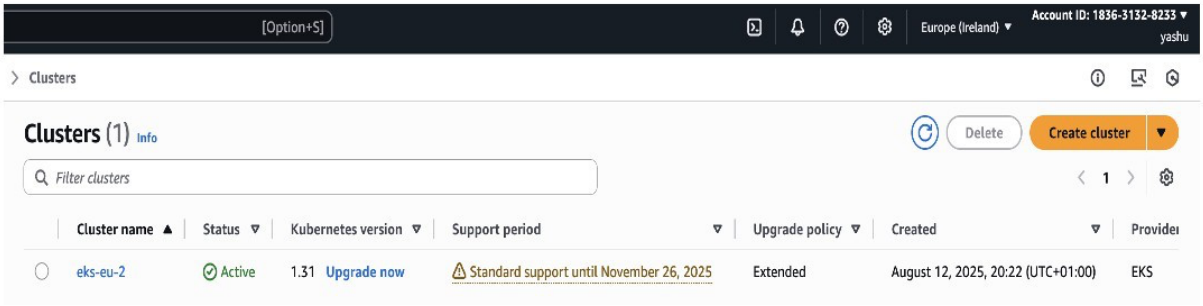

Figure 1: AWS EKS cluster creation

3.2 EKS Managed Node Group Configuration

The screenshot shows the managed node group created for the EKS cluster. Key settings:

These choices align with the evaluation goals: maintain a steady baseline of two workers, add capacity quickly under load without preemption risk, and keep the node configuration immutable via a launch template so successive experiments are comparable.

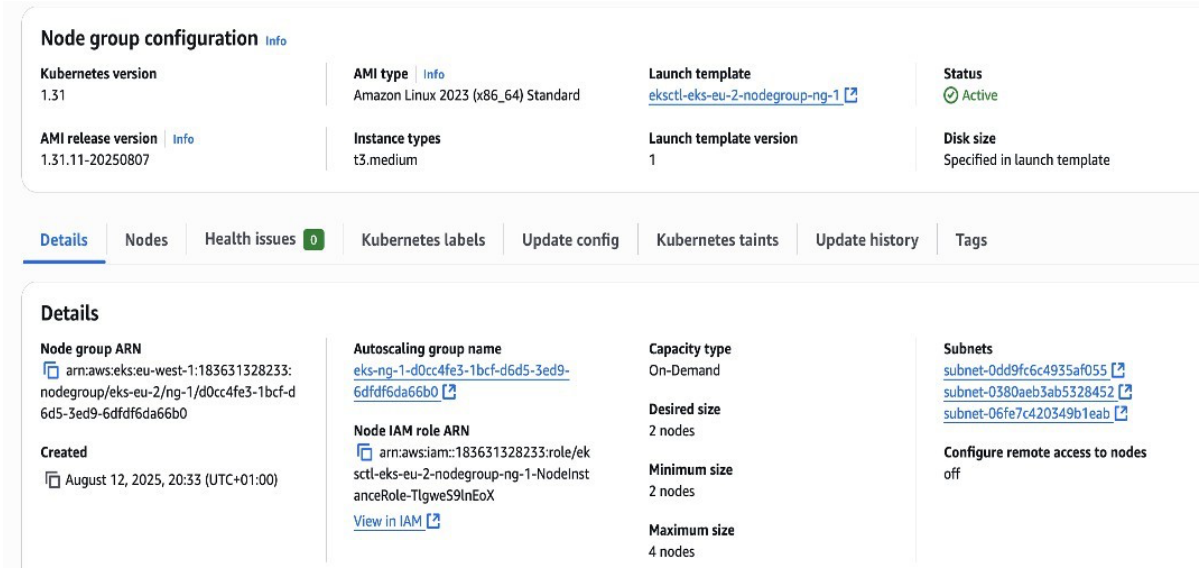

Figure 2: AWS EKS managed node group configuration

3.3 Kubernetes Cluster on Azure AKS

Portal steps:

- **Azure Portal → Kubernetes services → Create → Create a Kubernetes cluster**.

- **Basics:** choose Subscription/Resource Group, set **cluster name**, pick a **region** (near Ireland preferred), and select the **Kubernetes version**.
- **Node pool:** pick a VM size and node count comparable to EKS (e.g., two workers with headroom to scale).
- **Monitoring:** enable **Azure Monitor for containers (Insights)**.

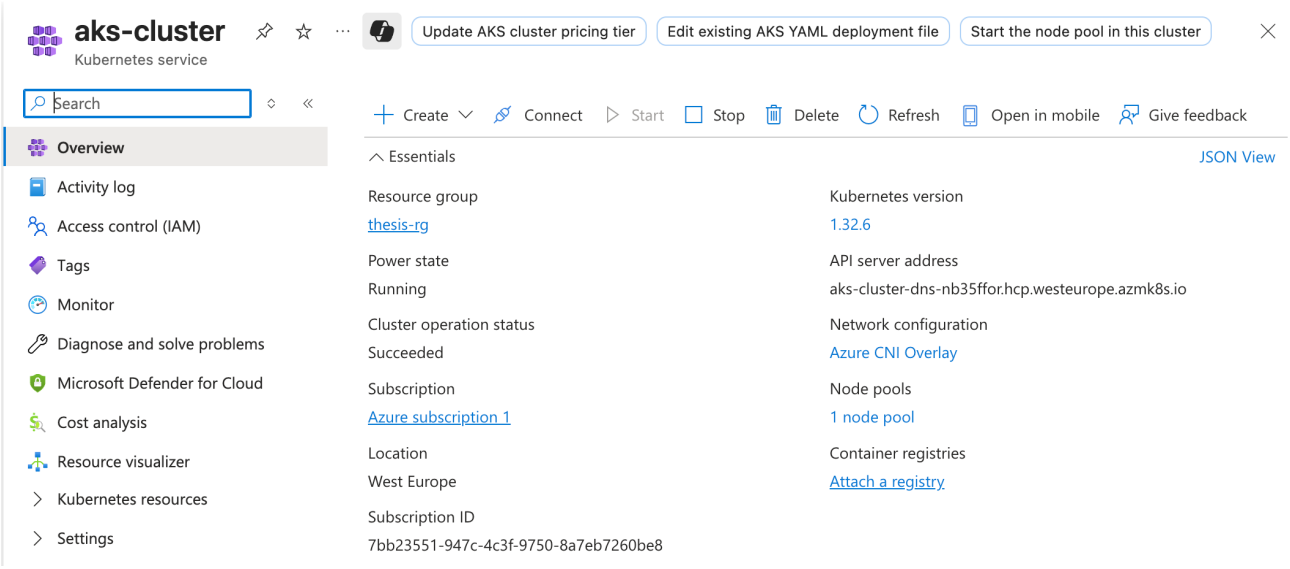

Figure 3: Azure *AKS* cluster creation

Configure kubectl **access.** Use Azure CLI to merge AKS cluster credentials into your local kubeconfig, then select the context and verify nodes:

```
az aks get-credentials -g thesis-rg -n aks-cluster

kubectl get nodes
```

```
yashvanthsb@MAC-849698 multi-cloud-thesis-app % kubectl get nodes
NAME                                STATUS   ROLES    AGE   VERSION
aks-agentpool-26959541-vmss000000   Ready    <none>   24d   v1.32.6
aks-agentpool-26959541-vmss000001   Ready    <none>   24d   v1.32.6
```

Figure 4: kubectl get nodes showing AKS worker nodes Ready

4 Configuration Files

The following manifests are applied to both clusters. Replace placeholders as needed.

4.1 Deployment.yaml

The deployment.yaml defines a Kubernetes **Deployment** (apps/v1) named web-app. It declares **2 replicas** and uses a selector/label pair (app=web-app) so the ReplicaSet manages only these pods. Each pod runs a container from yashusb/multi-cloud-thesis-app:v1 and exposes containerPort: 3000. The Deployment controller maintains the desired state and performs safe *RollingUpdate*s; the commented namespace implies the object targets the current/default namespace.

```
yashvanthsb@MAC-849698 multi-cloud-thesis-app % cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
  #namespace: web-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
    spec:
      containers:
      - name: web-app
        image: yashusb/multi-cloud-thesis-app:v1
        ports:
        - containerPort: 3000
```

Figure 5: Deployment.yaml configuration file code snippet

4.2 Service manifest

The service.yaml defines a Kubernetes **Service** (v1) named web-app-service. It selects pods with the label app=web-app and exposes **port 80**, forwarding traffic to **targetPort 3000** on the pods. With no type specified, the Service defaults to **ClusterIP**, giving a stable virtual IP/DNS inside the cluster and decoupling the client-facing port from the container's port. An Ingress or external LoadBalancer can then route traffic to this Service; the commented namespace implies it targets the current/default namespace.

```
yashvanthsb@MAC-849698 multi-cloud-thesis-app % cat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: web-app-service
 # namespace: web-app
spec:
  selector:
    app: web-app
  ports:
    - port: 80
      targetPort: 3000
```

Figure 6: Service.yaml configuration file code snippet

4.3 Ingress manifest

The ingress.yaml declares a **networking.k8s.io/v1 Ingress** named thesis-ingress for the NGINX Ingress Controller (ingressClassName: nginx). It exposes the app at the root path "/" with pathType: Prefix, routing all HTTP requests to the Service web-app-service on port **80**. The annotations instruct NGINX to speak **HTTP** to the backend and to enable **HTTP/2** for clients. No host or tls section is set, so the ingress answers on its external address over plain HTTP; you can add a hostname and TLS secret later for HTTPS.

```
yashvanthsb@MAC-849698 multi-cloud-thesis-app % cat ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: thesis-ingress
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTP"
    nginx.ingress.kubernetes.io/enable-http2: "true"
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web-app-service   # Match this to your actual service name
                port:
                  number: 80
```

Figure 7: Ingress.yaml configuration file code snippet

4.4 Horizontal Pod Autoscaler (HPA) manifest

The hpa.yaml defines an **HPA** (autoscaling/v2) named thesis-hpa that targets the **Deployment** web-app. It keeps at least **2** replicas and can scale up to **6** based on **CPU** usage, aiming for averageUtilization: 70%. With metrics from the cluster

metrics-server, the HPA increases pods during load and reduces them when demand falls; ensure pod CPU *requests* are set so utilization is meaningful.

```
yashvanthsb@MAC-849698 multi-cloud-thesis-app % cat hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: thesis-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: web-app
  minReplicas: 2
  maxReplicas: 6
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Figure 8: Ingress.yaml configuration file code snippet

5 Cluster Information

Useful commands to verify cluster state and endpoints:

```
kubectl cluster-info
kubectl get nodes -o wide
kubectl get svc -n ingress-nginx
```

```
yashvanthsb@MAC-849698 multi-cloud-thesis-app % kubectl cluster-info
kubectl get nodes -o wide
kubectl get svc -n ingress-nginx

Kubernetes control plane is running at https://aks-cluster-dns-nb35ffor.hcp.westeurope.azmk8s.io:443
CoreDNS is running at https://aks-cluster-dns-nb35ffor.hcp.westeurope.azmk8s.io:443/api/v1/namespaces/kube-system/services/kube-dns:
dns/proxy
Metrics-server is running at https://aks-cluster-dns-nb35ffor.hcp.westeurope.azmk8s.io:443/api/v1/namespaces/kube-system/services/ht
tps:metrics-server:/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
NAME                                STATUS   ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSIO
N      CONTAINER-RUNTIME
aks-agentpool-26959541-vmss000000   Ready    <none>   24d   v1.32.6   10.224.0.5    <none>        Ubuntu 22.04.5 LTS   5.15.0-1091-a
zure   containerd://1.7.27-1
aks-agentpool-26959541-vmss000001   Ready    <none>   24d   v1.32.6   10.224.0.4    <none>        Ubuntu 22.04.5 LTS   5.15.0-1091-a
zure   containerd://1.7.27-1
NAME                                 TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.0.235.128   74.178.148.100   80:30265/TCP,443:31286/TCP   24d
ingress-nginx-controller-admission   ClusterIP      10.0.241.165   <none>           443/TCP                      24d
```

Figure 9: Cluster information and service endpoints (example output).

6 CDN and Global DNS

6.1 Azure CDN (Standard Microsoft)

- Create a CDN profile and endpoint.
- Origin: AKS Ingress public IP / hostname.
- Rules: cache /assets/* with long TTL.
- Force **HTTPS** and enable **compression**.

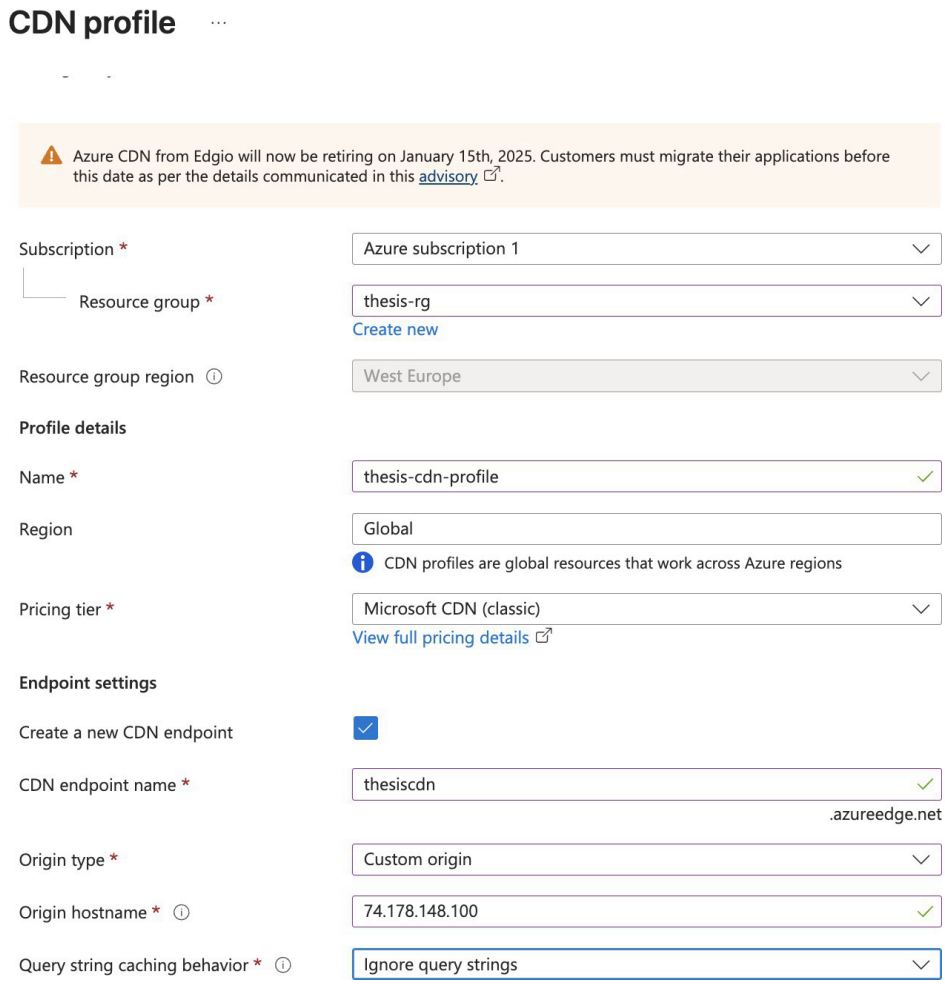

Figure 10: Azure CDN origin and endpoint configuration.

6.2 Amazon CloudFront

- Create a **distribution**; set **Origin** to the EKS Ingress ALB/NLB DNS name.
- Enable **HTTP/3** for viewer; choose cache policies for dynamic vs. static content.
- (Optional) Configure an **origin group** with AKS as failover.

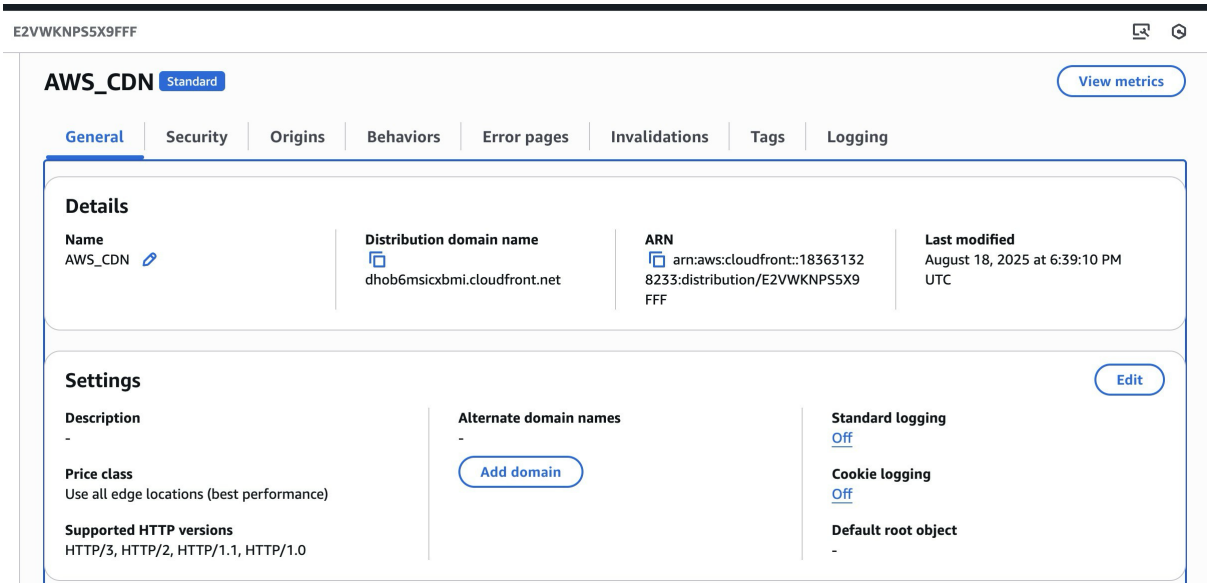

Figure 11: CloudFront distribution with HTTP/3 enabled.

6.3 Global Routing (Azure Traffic Manager)

- Configure **Azure Traffic Manager** (Performance): endpoint is Azure CDN as an *external* endpoint.
- Enable **health probes** for automatic failover.

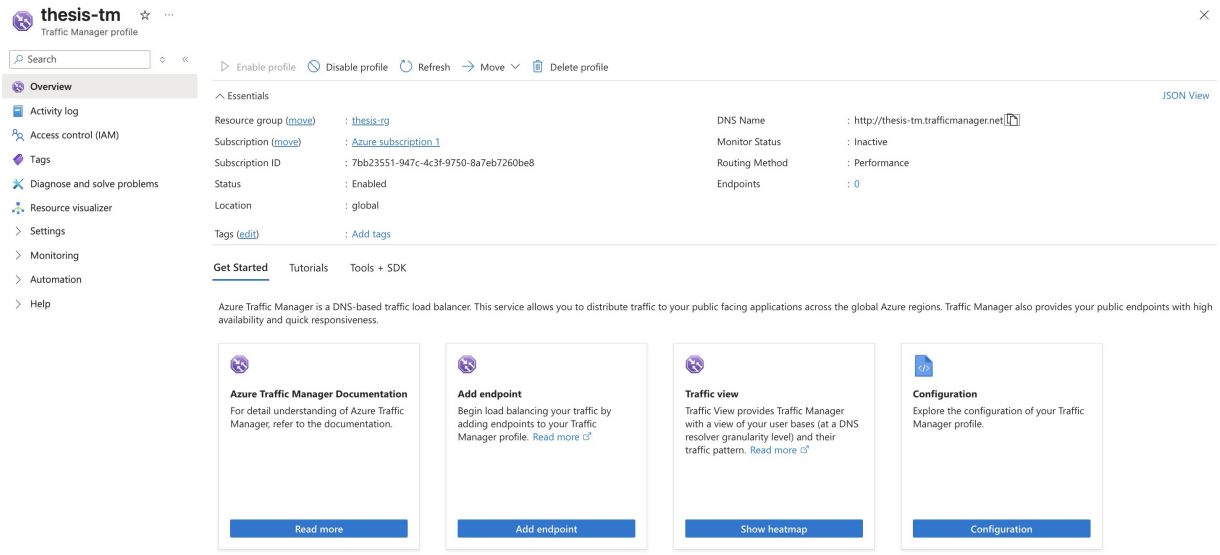

Figure 12: Azure Traffic Manager (Performance) endpoints and health probe configuration.

7 Web Application

7.1 Developing Web Application

- Application URL: http://localhost:3000/
- Developed a simple web application using **HTML**, **CSS**, and **Node.js**.

- The application listens on **port** 3000

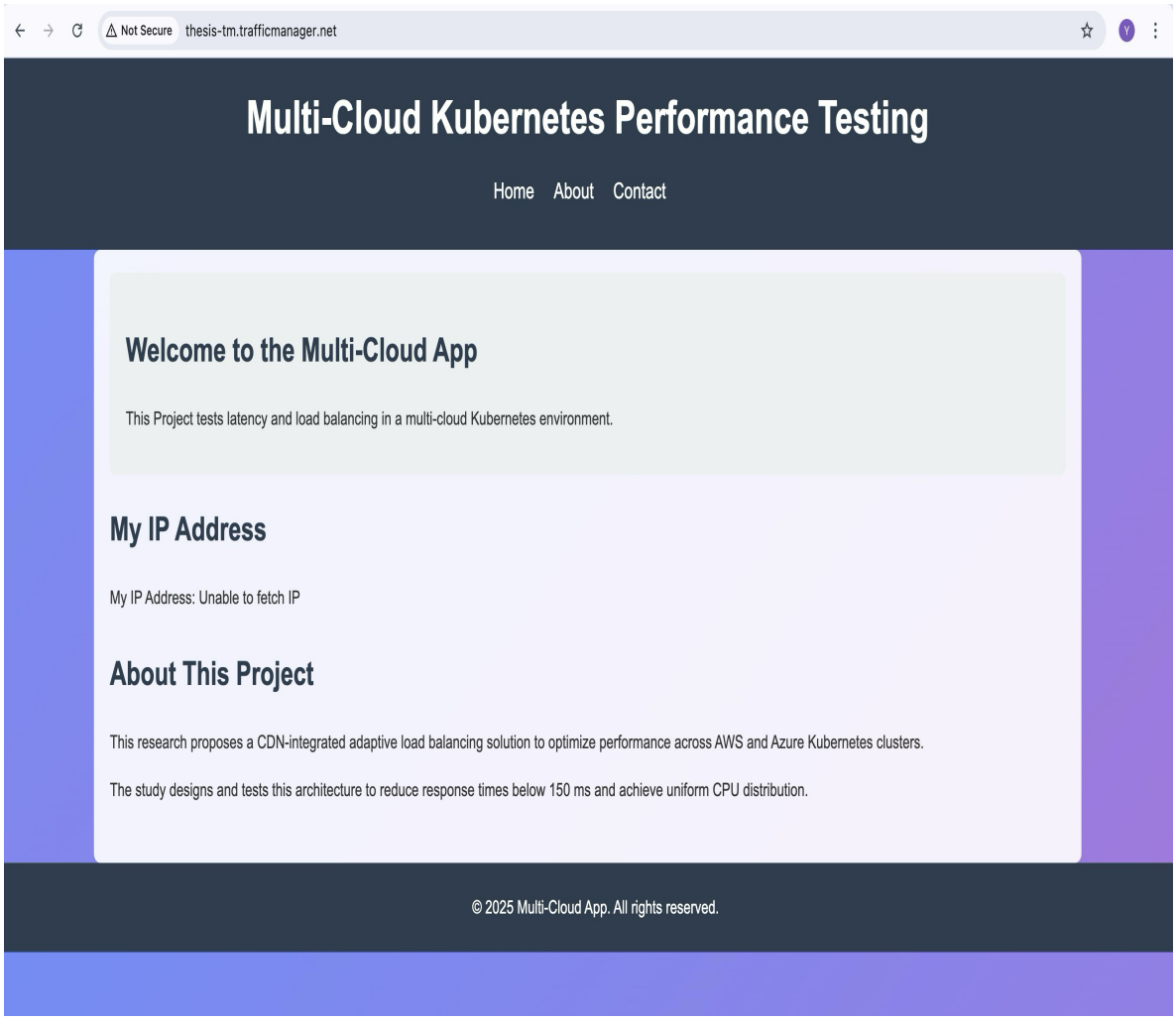

Figure 13: Web Application code Snippet

7.2 Deploying The Application to the Docker Container

- Created a Docker image of my web application and pushed that image to the Docker container.
- access the web application through the Docker container
- Pushed the Docker image to Docker Hub

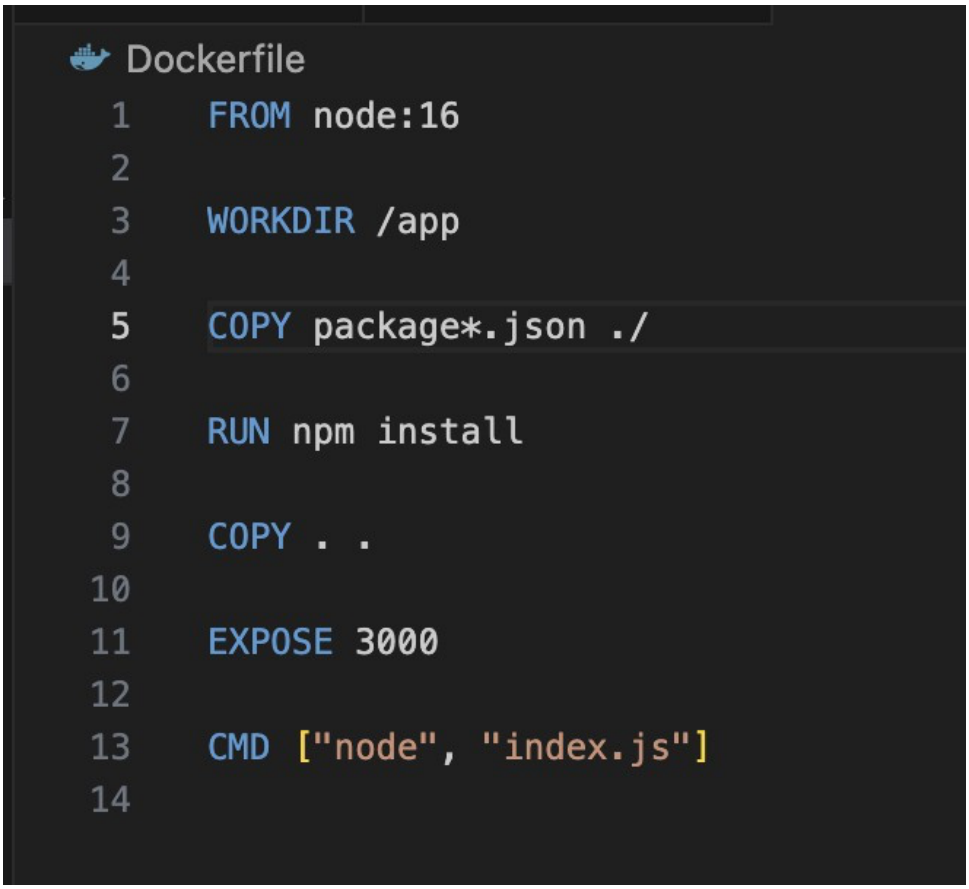

Figure 14: Docker Image creation code snippet

8 Monitoring

8.1 AWS CloudWatch

- EKS Container Insights: Pod/Node CPU (%) trends.
- CloudFront: Requests, Bytes
- From the below (Figure16) Two worker nodes from the same node group are plotted (green and red), plus a blue “Expression” series that represents an aggregate.
- At the highlighted time **2025-08-14 12:25 UTC**, the nodes read about **2.61%** and **2.36%**, with the aggregate around **2.55%**.
- he y-axis range sits between roughly **2.3% and 2.6%**, indicating **consistently low CPU usage** throughout the period.

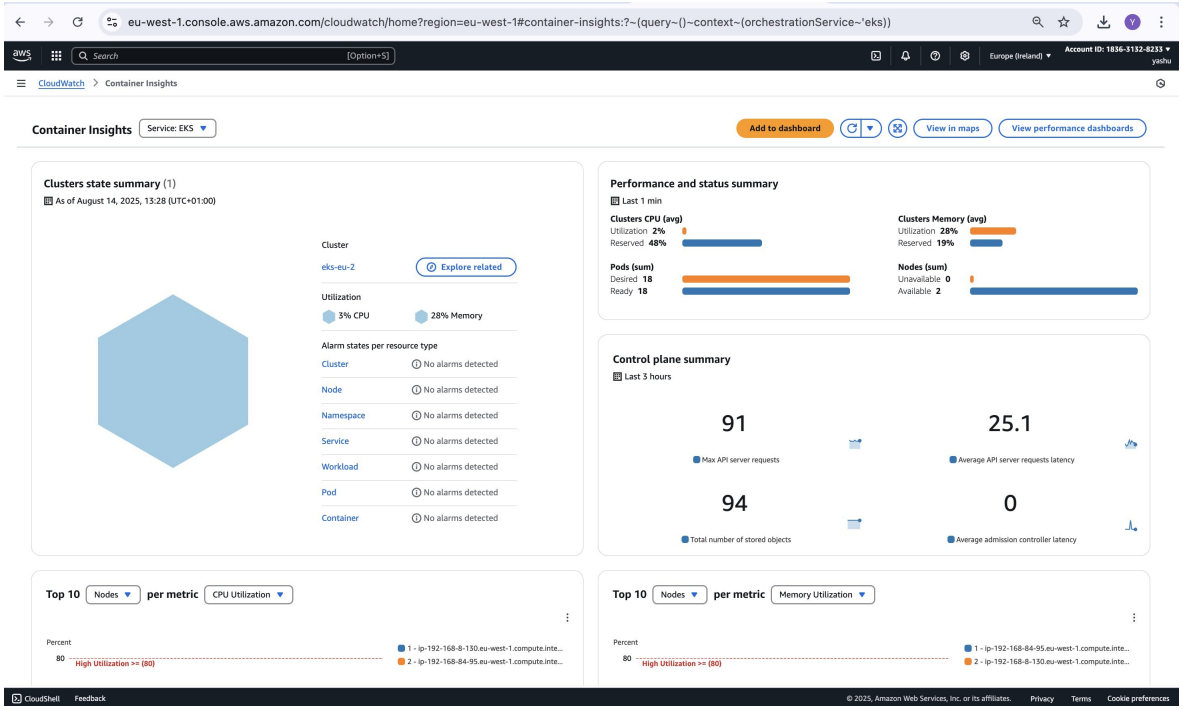

Figure 15: AWS CloudWatch dashboard snippet

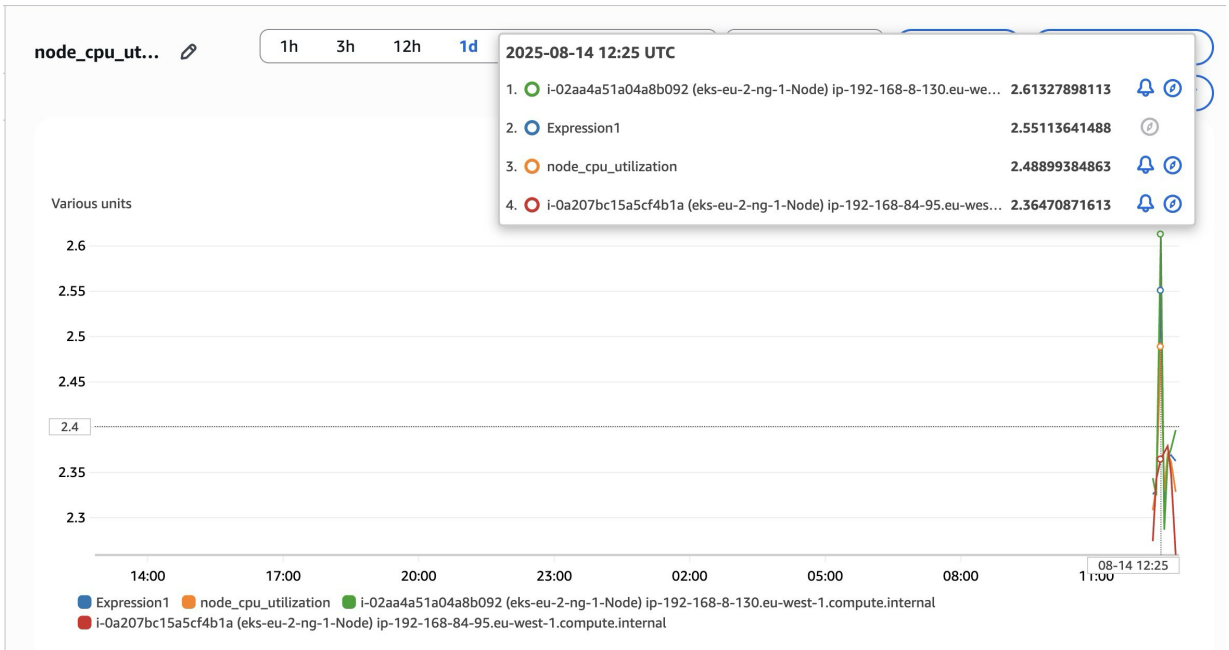

Figure 16: node cpu utilization snippet

8.2 Azure Monitor

- AKS Insights: Node and Pod CPU %, memory, and replica counts.
- Azure CDN analytics: Requests, Cache hit ratio, Latency.

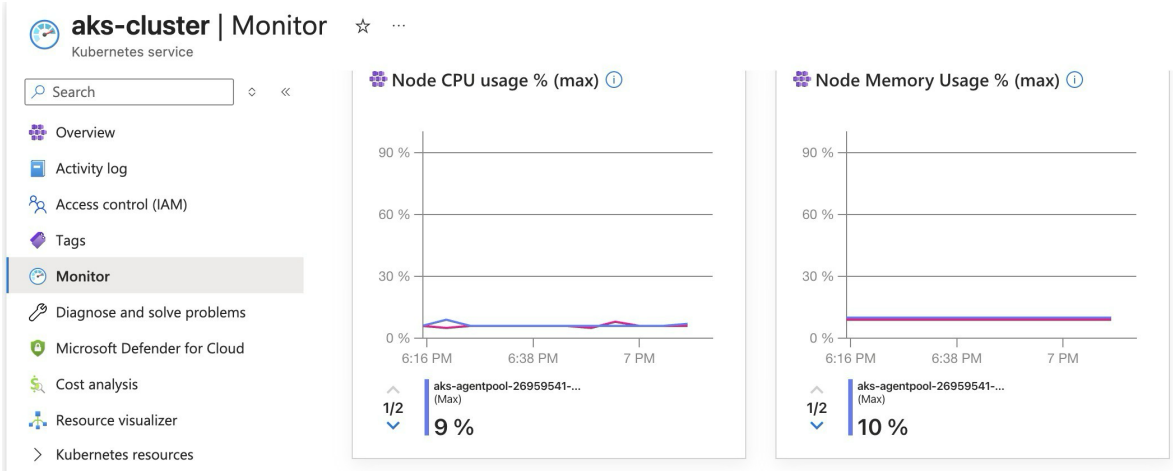

Figure 17: Azure node cpu and memory usage snippet.

9 Observations

9.1 JMeter HTTP Request — AKS via Azure Traffic Manager

This sampler exercises the Azure path by calling the Traffic Manager endpoint (thesis-tm.trafficmanager.net) on **port 80**. We issue an HTTP **GET** to the root path / with redirects enabled, keeping the request lightweight so it represents end-to-end latency through CDN/Ingress to the AKS Service. This configuration is used for the multi-cloud "Azure" leg in A/B tests.

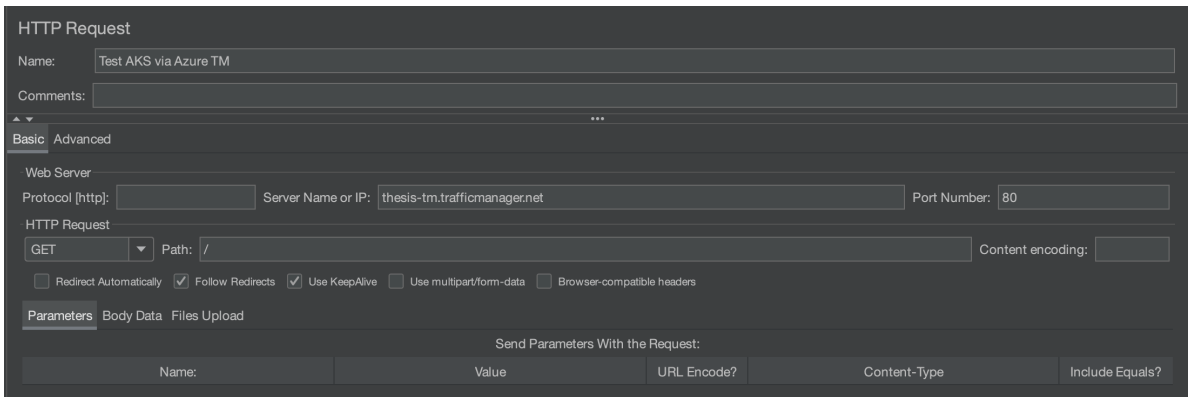

Figure 18: JMeter HTTP Request targeting Azure Traffic Manager

9.2 JMeter HTTP Request — EKS via AWS Load Balancer

This sampler targets the AWS path by calling the EKS Ingress **ALB/NLB** DNS name on **port 80**. We again send a **GET /** to capture the end-to-end response time through CloudFront/Ingress (or direct ALB) to the EKS Service. Results from this request are compared against the Azure leg to evaluate latency and variability under the same load profile.

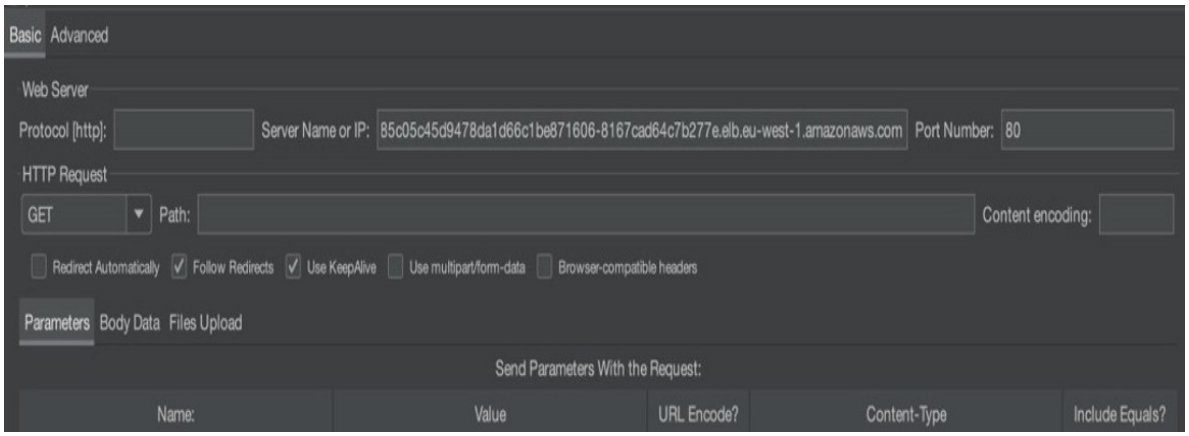

Figure 19: JMeter HTTP Request targeting the EKS Ingress Load Balancer

9.3 Final Result

A representative sample (Fig. 20) returned HTTP 200 with **122 ms** latency and **0 ms** connect time, confirming **KeepAlive** reuse. The payload is small (~2 KB), so the value reflects network + ingress processing rather than transfer time. This meets the sub-150 ms first-access objective and provides traceable evidence for later aggregation.

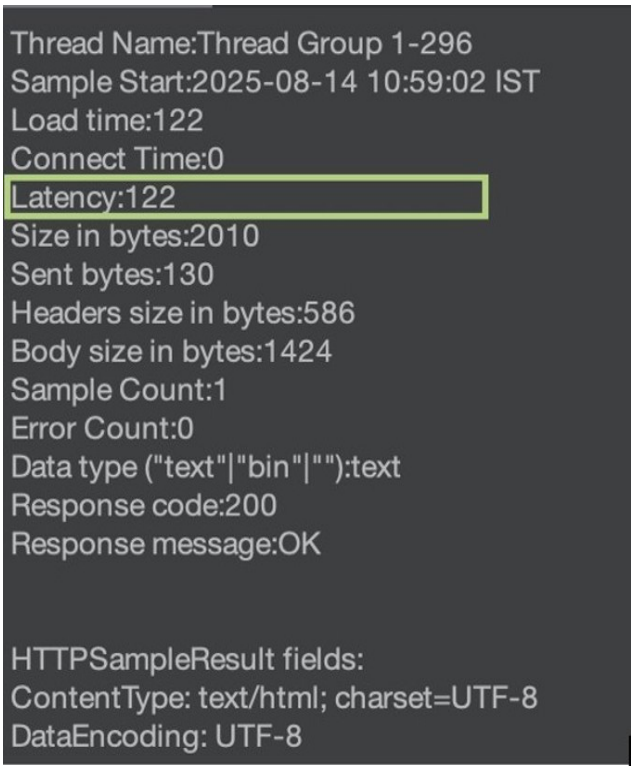

Figure 20: Response time when the web application is accessed for the first time.

9.4 Latency Evaluation

Cloud	Users	Min (ms)	Max (ms)	Average (ms)	Std dev (ms)	Mean (ms)
AWS	250	11	180	57	37.56	**50.33**
	350	10	1115	48	60.57	
	500	8	1089	46	64.00	
Azure	250	22	753	46	54.41	**59.33**
	350	23	194	39	17.33	
	500	57	1541	93	152.12	

Table 1: End-to-end latency summary by cloud and user load.

References

Afzal, S., Kavitha, G. Load balancing in cloud computing – A hierarchical taxonomical classification. J Cloud Comp 8, 22 (2019). https://doi.org/10.1186/s13677-019-0146-7

Dong, Y., Xu, G., Zhang, M., & Meng, X. (2021). A high-efficient joint cloud–edge aware strategy for task deployment and load balancing. IEEE Access, 9, 12791–12802. https://doi.org/10.1109/ACCESS.2021.3051672

Kumari, N. (2023). *Analysis of dynamic application load balancing in Kubernetes using CDN*. Master's thesis, National College of Ireland. https://norma.ncirl.ie/7086/

Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., & Kim, S. (2020). Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration. Sensors, 20(16), 4621. https://doi.org/10.3390/s20164621

Radhika, D., & Duraipandian, M. (2021). Load balancing in cloud computing using support vector machine and optimized dynamic task scheduling. *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)*, 1–6. https://doi.org/10.1109/ICRITO51393.2021.9596289

Shafiq, D. A. et al. (2021). Load balancing techniques in cloud computing environment: A review. Review paper. https://doi.org/10.1016/j.jksuci.2021.02.007

Simić, M., Stojkov, M., Sladić, G., & Milosavljević, B. (2020). CRDTs as replication strategy in large-scale edge distributed systems: An overview. In *Proceedings of ICIST 2020*, 46–50. https://www.eventiotic.com/eventiotic/library/paper/582

Varma, A. et al. (2023). Dynamic user routing for paid and free users in web applications using cdn, *International Journal for Research in Applied Science and Engineering Technology* . https://doi.org/10.22214/ijraset.2023.54654

Yang, H., Pan, H., & Ma, L. (2023). A review on software-defined content delivery network: A novel combination of CDN and SDN. *IEEE Access, 11*, 43822–43846. https://doi.org/10.1109/ACCESS.2023.3267737