National College of Ireland

Enhancing Kubernetes Traffic Distribution with Dynamic Load Balancer using serverless

MSc Research Project
Cloud Computing

Ritika Chatterjee
Student ID: 23303808

School of Computing
National College of Ireland

Supervisor: Luis Bernardo Pulido Gaytan

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ritika Chatterjee
Student ID:	23303808
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Luis Bernardo Pulido Gaytan
Submission Due Date:	14th September 2025
Project Title:	Enhancing Kubernetes Traffic Distribution with Dynamic Load Balancer using serverless
Word Count:	917
Page Count:	6

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:	
Date:	14th September 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):	

1 Introduction

This document provides a comprehensive, step-by-step guide detailing the setup and configuration of a Google Kubernetes Engine (GKE) environment. The process begins with the initialization of the Google Cloud SDK and proceeds through the creation of a GKE cluster, deployment of a sample application, and installation of the Kube-Prometheus stack for robust monitoring. Finally, it covers the development and deployment of a Python-based Google Cloud Function designed to act as a load balancer, routing external traffic to the application running in the GKE cluster. Each step is documented with the exact commands used and their corresponding outputs.

2 Google Cloud SDK Initial Setup

The first step is to initialize the Google Cloud SDK, creating a new configuration profile to manage credentials and project settings.

2.1 Initialize SDK Configuration

A new configuration named `ritika-account` was created and linked to the specified Google account and project.

```
gcloud init
```

- **Configuration Name:** `ritika-account`
- **Account Email:** ritika.mscloud@gmail.com
- **Project ID:** avid-compound-463219-p0

2.2 Update SDK Components

```
gcloud components update
```

2.3 Set Default Region and Zone

```
gcloud config set compute/region us-central1
```

```
gcloud config set compute/zone us-central1-c
```

3 Configuration Verification

3.1 Verify Authenticated Account

```
gcloud auth list
```

```
* ritika.mscloud@gmail.com
```

3.2 Verify Active Project

```
gcloud config list project
```

```
[core] project = avid-compound-463219-p0
Your active configuration is: [ritika-account]
```

3.3 Verify Active Configuration Details

```
gcloud config configurations list
```

NAME IS$_ACTIVEACCOUNTPROJECTritika - accountTrueritika.mscloud@gmail.comavid - compound - 463219 - p0$
COMPUTE$_DEFAULT_ZONECOMPUTE_DEFAULT_REGIONus - central1 - cus - central1$

4 Enabling Required APIs

Essential Google Cloud APIs were enabled.

```
gcloud services enable container.googleapis.com cloudfunctions.googleapis.com
monitoring.googleapis.com
```

5 GKE Cluster Creation and Verification

5.1 Create GKE Cluster

```
gcloud container clusters create k8s-demo-cluster –zone us-central1-c –num-
nodes=3
```

Creating cluster k8s-demo-cluster in us-central1-c...done. Cluster is being health-checked...done. Created [.../clusters/k8s-demo-cluster]. kubeconfig entry generated for k8s-demo-cluster. NAME LOCATION $\text{MASTER}_{V}ERSIONMASTER_{I}Pk8s - demo - clusterus - central1 - c1.32.4 - gke.141500034.69.9.145$ $\text{MACHINE}_{T}YPENODE_{V}ERSIONNUM_{N}ODESSTATUSe2 - medium1.32.4 - gke.14150003RUNNING$

5.2 Get Cluster Credentials

```
gcloud container clusters get-credentials k8s-demo-cluster –zone us-central1-c
```

Fetching cluster endpoint and auth data. kubeconfig entry generated for k8s-demo-cluster.

5.3 Verify Cluster Nodes

```
kubectl get nodes
```

NAME STATUS AGE VERSION gke-k8s-demo-cluster-default-pool-6bf87d61-0spc Ready 3m42s v1.32.4-gke.1415000 gke-k8s-demo-cluster-default-pool-6bf87d61-r9sv Ready 3m43s v1.32.4-gke.1415000 gke-k8s-demo-cluster-default-pool-6bf87d61-txf8 Ready 3m43s v1.32.4-gke.1415000

6 Deploying a Sample Application

6.1 Create Deployment Manifest

A file named `deployment.yaml` was created.

apiVersion: apps/v1 kind: Deployment metadata: name: test-app spec: replicas: 3 selector: matchLabels: app: test-app template: metadata: labels: app: test-app spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80 — apiVersion: v1 kind: Service metadata: name: test-app-service spec: selector: app: test-app ports: - protocol: TCP port: 80 targetPort: 80 type: LoadBalancer

6.2 Apply the Manifest and Get External IP

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

deployment.apps/test-app created service/test-app-service created

kubectl get svc test-app-service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) test-app-service Load-Balancer 34.118.231.25 34.61.146.68 80:32623/TCP

7 Installing and Configuring Prometheus

The Kube-Prometheus stack was installed via Helm to collect pod metrics.

7.1 Baseline Load Test

A load test was performed using `wrk` to generate baseline traffic.

Install wrk on WSL sudo apt-get update sudo apt-get install -y wrk
Run load test wrk -t4 -c10 -d30s http://34.61.146.68

Running 30s test @ http://34.61.146.68 4 threads and 10 connections Thread Stats Avg Stdev Max +/- Stdev Latency 276.07ms 8.07ms 295.76ms 65.38Req/Sec 7.60 2.55 10.00 65.59858 requests in 30.03s, 714.72KB read Requests/sec: 28.57 Transfer/sec: 23.80KB

7.2 Install Kube-Prometheus Stack

The `kube-prometheus-stack` was successfully installed.

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts helm repo update helm install kube-prometheus prometheus-community/kube-prometheus-stack

7.3 Verify and Access Prometheus

The Prometheus UI was accessed by forwarding the service port.

kubectl port-forward svc/kube-prometheus-kube-prome-prometheus 9090:9090

The following PromQL query was executed at `http://localhost:9090` to verify metrics collection. The query successfully returned tabular and graphical data.

$$\text{rate}(\text{container}_{c}pu_{u}sage_{s}econds_{t}otalpod = \text{"}test - app.*\text{"}[1m])$$

8 Deploying a Cloud Function Load Balancer

A serverless Cloud Function was created to act as an HTTP proxy.

8.1 Initial Attempt and Timeout Issue

The first version attempted to route requests to internal pod IPs. This failed with a timeout error because Cloud Functions cannot directly access GKE pod IPs without a VPC Connector.

Listing 1: main.py (V1 - Failed)

```
import requests

def route_request(request):
    # This will fail from Cloud Functions
    pod_endpoints = [
        "http://10.60.2.11",
        "http://10.60.0.5",
        "http://10.60.1.6"
    ]
    target_pod = pod_endpoints[0]

    try:
        response = requests.get(target_pod)
        return (response.text, response.status_code)
    except Exception as e:
        return (f"Error contacting pod: {str(e)}", 500)
```

Testing this version resulted in an `upstream request timeout`.

8.2 Corrected Implementation

The function was corrected to route requests to the **external IP of the Kubernetes LoadBalancer Service**.

main.py (V2 - Successful)

Listing 2: main.py (V2 - Success)

```
import requests

def route_request(request):
    """
    Forwards HTTP request to the NGINX
    LoadBalancer service in GKE.
```

```
    """
    target = "http://34.61.146.68"

    try:
        response = requests.get(target)
        return (response.text, response.status_code)

    except Exception as e:
        return (f"Error contacting service: {str(e)}", 500)
```

```
requests
```

8.3 Deploying and Verifying the Function

```
gcloud functions deploy route_request --runtime python310 --trigger-http --allow-unauthenticated
```

The function's public URL was successfully tested, returning the NGINX welcome page and confirming correct operation.

9 Conclusion

This manual has successfully detailed the complete end-to-end process of configuring a production-ready environment on Google Cloud. A GKE cluster was provisioned, a sample application was deployed, and the Kube-Prometheus stack was installed to provide real-time insights into application performance. The architecture was further enhanced with a serverless Cloud Function acting as a resilient entry point for traffic. All components were integrated correctly and performed as expected under load, demonstrating a robust and observable system.