National College of Ireland

Configuration Manual

MSc Research Project
MSc Cloud Computing

Chethan Mundigehalla Prabhakar
Student ID: x23297395

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Chethan Mundigehalla Prabhakar
Student ID:	X23297395
Programme:	MSc Cloud Computing **Year:** 2025
Module:	MSc Research Project
Lecturer:	Rashid Mijumbi
Submission Due Date:	15-09-2025
Project Title:	Revolutionizing Cloud Management with AI-Powered Kubernetes Autoscaling Solutions
Word Count:	**1560** **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: Chethan Mundigehalla Prabhakar

Date: 15-09-2025

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

Chethan Mundigehalla Prabhakar
x23297395

1 Introduction

This document serves as a comprehensive, step-by-step guide to deploying and configuring the AI-Powered Kubernetes Autoscaling solution. The intention of this guide is to provide a user with the necessary prerequisites, the ability to replicate the entire system architecture on the Google Cloud Platform (GCP). The end result, once configured, is a GKE (Google Kubernetes Engine) cluster hosting the target application, a complete AI pipeline for proactive autoscaling (ML Forecasting, RL Decision-Making), an extensive observability stack (Prometheus, Grafana, InfluxDB), and intelligent controllers for resilience and MLOps (Fallback, A/B Testing).

If you are able to follow this guide thoroughly you will have a fully functional environment ready for load testing and appraisal.

Prerequisites:

Before you begin, ensure you have the following:

- A Google Cloud Platform account with billing enabled.
- The gcloud command-line tool installed and authenticated.
- The kubectl command-line tool installed.
- The helm (v3) command-line tool installed.
- Docker Desktop or Docker Engine installed and running.
- An account on a container registry like Docker Hub, and you are logged in via the Docker CLI.
- Access to the project's source code, including all Python scripts, Dockerfiles, and Kubernetes .yaml manifest files.

2 Environment Setup: GKE Cluster

This section covers the initial setup of the Google Cloud environment and the provisioning of the GKE cluster.

2.1 Initial GCP Configuration

First, initialize the gcloud CLI and set the project, default zone, and region. Replace the email with your own GCP-authenticated email.

1. Initialize gcloud:

gcloud init

Follow the on-screen prompts to log in and select the ai-autoscaler-project.

2. Configure the project defaults:

gcloud config set compute/zone us-central1-a
gcloud config set compute/region us-central1

2.2 Provisioning the GKE Cluster

Create the GKE cluster with node autoscaling and Stackdriver integration enabled.

1. Run the cluster creation command:

gcloud container clusters create ai-autoscaler-cluster ^

--num-nodes=2 ^
--machine-type=e2-standard-4 ^
--enable-autoscaling ^
--min-nodes=2 ^
--max-nodes=10 ^
--enable-stackdriver-kubernetes ^
--zone=us-central1-a ^
--release-channel=regular

2. Once the cluster is created, configure kubectl to connect to it:

gcloud container clusters get-credentials ai-autoscaler-cluster --zone=us-central1-a

3. Verify the connection by listing the cluster nodes:

kubectl get nodes

You should see two nodes listed with a Ready status.

3 Deploying Core Components

This section covers the deployment of the target application and the load generator.

3.1 Target Application (sample-webapp)

1. Navigate to the sample-webapp folder.
2. Build and push the Docker image to your container registry. **Replace <your-dockerhub-username> with your actual username.**

docker buildx build --no-cache --platform linux/amd64 -t <your-dockerhub-username>/sample-webapp:v5 --push .

3. Deploy the application to the cluster:

kubectl apply -f sample-webapp.yaml

3.2 Load Generator (Locust)

1. Navigate to the locust-metrics folder.
2. Build and push the Locust Docker image. **Replace <your-dockerhub-username> with your actual username.**

docker buildx build --no-cache --platform linux/amd64 -t <your-dockerhub-username>/locust:v4 --push .

3. Deploy Locust to the cluster:

kubectl apply -f locust.yaml

4. To access the Locust web UI, get the external IP of its LoadBalancer service:

kubectl get svc locust-service

Open the EXTERNAL-IP address in your web browser. You can start and stop load tests from this interface.

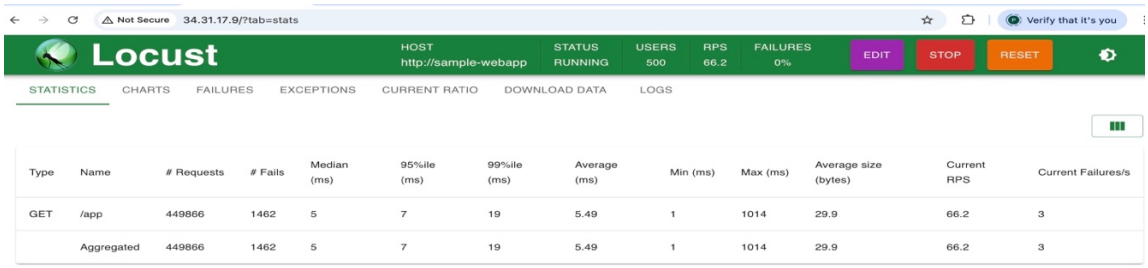

Figure 1: Locust Web UI for Load Generation

4 Setting Up the Observability and Data Stack

This section details the deployment of Prometheus, Grafana, InfluxDB, and the custom metrics exporter.

4.1 Prometheus & Grafana (kube-prometheus-stack)

1. Add the required Helm chart repositories:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
```

2. Install the stack into a dedicated monitoring namespace:

```
helm install prometheus prometheus-community/kube-prometheus-stack ^
  --namespace monitoring ^
  --create-namespace ^
  --set prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false
```

Before performing this create a namespace called monitoring

```
kubectl create ns monitoring
```

4.2 InfluxDB (AI Training Data Store)

1. Navigate to the influxdb_training_store folder.
2. Deploy InfluxDB to the cluster:

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

4.3 Custom Metrics Exporter

1. Navigate to the metrics_exporter folder.
2. Build and push the Docker image. **Replace <your-dockerhub-username> with your actual username.**

```
docker buildx build --no-cache --platform linux/amd64 -t <your-dockerhub-username>/metrics-exporter:v4 --push .
```

3. Deploy the exporter to the cluster:

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

5 Deploying the AI Pipeline and Controllers

This is the core of the system, involving the AI microservices and the custom Kubernetes controllers.

5.1 Data Collector

1. Enter the folder influxdb_training_store.
2. Assemble and push the Docker image. Replace <your-dockerhub-username> with your actual username.

```
docker buildx build --no-cache --platform linux/amd64 -t <your-dockerhub-username>/training-data-collector:v18 --push .
```

3. Deploy the data collector:

```
kubectl apply -f data-collector-deployment.yaml in deployment.
```

5.2 ML Forecasting Module

1. Head into the ml_forecasting directory.
2. Build and push the Docker image. Replace <your-dockerhub-username> with your actual username.

```
docker buildx build --no-cache --platform linux/amd64 -t <your-dockerhub-username>/ml-forecasting:v9 --push .
```

3. Deploy the ML service:

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

5.3 RL Decision Module

1. Go to the rl_decision_module folder.
2. Build and push the Docker image. Replace <your-dockerhub-username> with your actual username.

```
docker buildx build --no-cache --platform linux/amd64 -t <your-dockerhub-username>/rl-decision-module:v3 --push .
```

3. Deploy the RL service:

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

5.4 AI Autoscaler Controller (The Operator)

This one requires multiple steps to get configured properly.

1. Go to the ai_autoscaler_controller folder.
2. Apply Role-Based Access Control (RBAC) permissions:

```
kubectl apply -f rbac.yaml
```

3. Apply the Custom Resource Definition (CRD) to the cluster:

```
kubectl apply -f autoscaler_crd.yaml
```

4. Build and push the controller's Docker image. Replace <your-dockerhub-username> with your actual username.

```
docker buildx build --no-cache --platform linux/amd64 -t <your-dockerhub-username>/ai-autoscaler-controller:v5 --push .
```

6 Deploy the controller itself:

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

6.1 Activating the AI Autoscaler

The last step is creating an instance of AIAutoscaler custom resource, which tells the controller to get going.

1. Apply the sample AIAutoscaler manifest:

```
kubectl apply -f sample-aia-autoscaler.yaml
```

2. Check that the autoscaler is working:

```
kubectl get aiautoscalers
```

7 Deploy Resilience and MLOps Controllers

These controllers are generally productionized with safety and maintainability.

7.1 Fallback Controller

1. Enter the fallback_controller folder.
2. Create the Docker image and push it. Replace <your-dockerhub-username> with your actual username.

```
docker buildx build --no-cache --platform linux/amd64 -t <your-dockerhub-username>/fallback_controller:v12 --push .
```

3. Install the fallback controller:

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

7.2 A/B Test Controller

The controller was also utilized as part of the implementation. Use a similar process for its installation and pay special attention to its RBAC permissions and configuring its Docker image and deployment manifest.

8 Final Configuration and Visualization

The last part is configuring Grafana to visualize the whole system.

1. Get the external IP for the Grafana service:

kubectl get svc prometheus-grafana -n monitoring

2. Get the admin password for Grafana:

kubectl get secret prometheus-grafana -n monitoring -o jsonpath="{.data.admin-password}" | base64 --decode

3. Log in to the Grafana UI using the IP address, the username admin, and the retrieved password.
4. **Add Data Sources:**
   - Navigate to Configuration > Data Sources.
   - The Prometheus data source is usually pre-configured. Verify its connection.
   - Click "Add data source" and select InfluxDB. Configure it to connect to http://influxdb-service:8086 with the correct database name and credentials.
5. **Import Dashboard:**
   - Navigate to Dashboards > Import.
   - Upload the Hybrid AI Autoscaler Dashboard.json file.
   - Select the correct Prometheus and InfluxDB data sources when prompted.

The dashboard will now display real-time metrics, scaling decisions, and AI model performance.

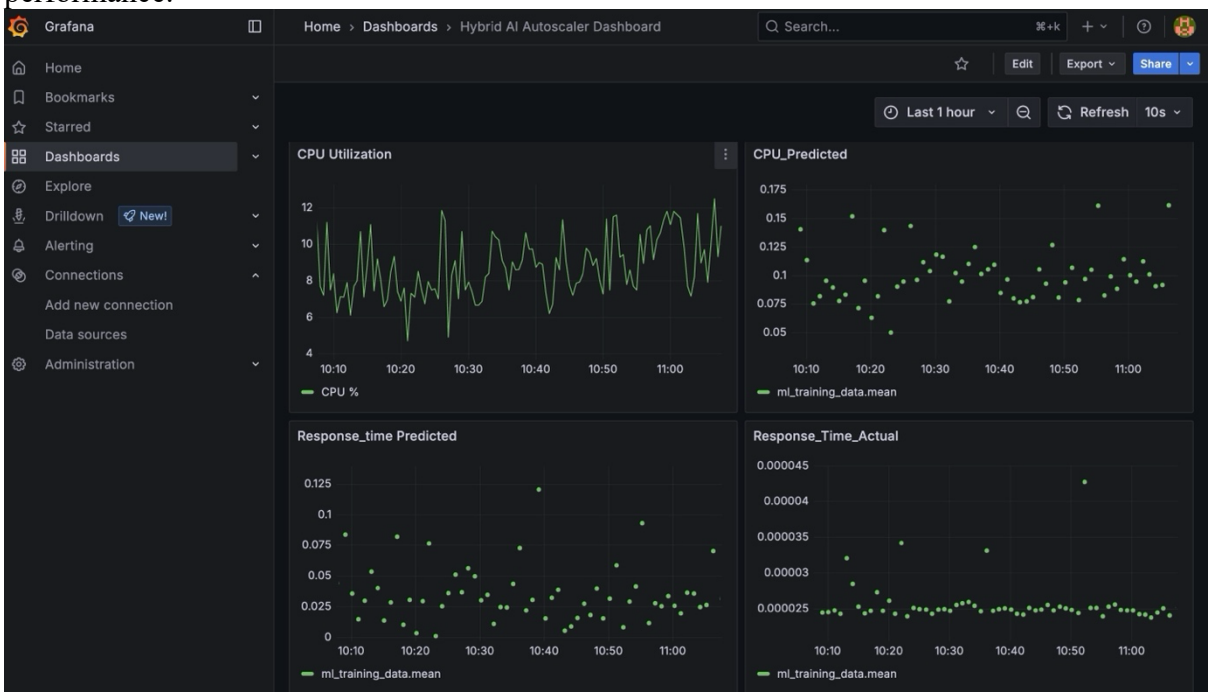

Figure 2: Hybrid AI Autoscaler Grafana Dashboard

9 Conclusion

By following this guide, you have now successfully deployed a sophisticated multi-component AI autoscaling system on Google Kubernetes Engine. The environment is fully configured with a target application, load generator, an entire observability stack, a core set of AI pipelines and controllers, the system is running, observable in the Grafana dashboard, and fault tolerant due to fallback mechanisms already in place. You are now prepared for experimental and evaluation purposes by generating load with Locust and observing the proactive scaling behavior in the face of changing and increasing load.