

Cloud Computing Assignment Report

Student: Mustafa Fahimy

Course: Cloud Computing (WS2024/25)

University of Vienna

1. Introduction

This assignment involved designing, implementing, deploying, scaling, and cost-estimating a microservice-based camera processing system using **Google Kubernetes Engine (GKE)**.

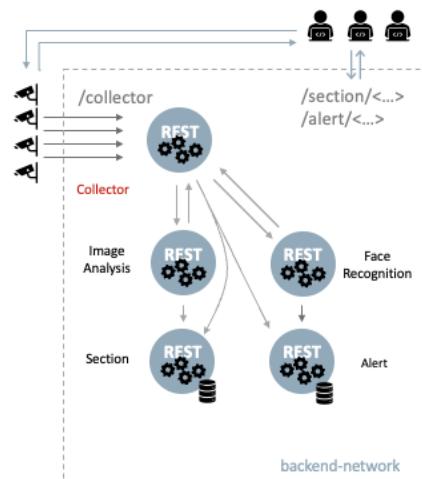
The architecture consists of six services:

1. **Camera** – Sends simulated frames.
2. **Collector** – *Implemented by me* (primary coding task).
3. **Image Analysis** – Detects persons from images.
4. **Face Recognition** – Detects known persons.
5. **Section** – Stores person-related events.
6. **Alert** – Stores alerts for known persons.

Goal of the system:

Camera → Collector → Image Analysis & Face Recognition → Section & Alert

I implemented the full Collector service using Flask, including logic to forward data to downstream services. All services were deployed into a Kubernetes namespace (assignment), with autoscaling configured via Horizontal Pod Autoscalers (HPAs), resource limits and requests defined, Ingress routing set up, and cost estimates calculated using the Google Cloud Pricing Calculator.



2. Implementation Details

2.1 Collector Service Implementation (my contribution)

Technologies used

- **Flask** (HTTP server)
- **requests** library (HTTP client for forwarding)
- **Kubernetes-ready Dockerfile**
- **Liveness/Readiness probes**
- **JSON forwarding to other services**

Collector responsibilities

- Receive frames from Camera
- Forward frame to Image Analysis
- Forward frame to Face Recognition
- Forward results to Section (persons detected)
- Forward known persons to Alert

Why synchronous communication?

My implementation uses **synchronous HTTP calls** (Flask + requests).

Pros:

- Easy to implement
 - Ensures ordered processing
- Cons:
- Blocking requests → lower throughput
 - Slower if Image Analysis takes time

Possible improvement:

Use **FastAPI + aiohttp** or a **message queue (Kafka/PubSub)** to allow asynchronous processing.

2.2 Communication Flow in Kubernetes

Inside the cluster, services communicate via **ClusterIP**:

- http://image-analysis:80/frame
- http://face-recognition:80/frame
- http://section:80/persons
- http://alert:80/alerts

Kubernetes DNS resolves service names automatically within the assignment namespace.

Flow:

1. Camera → Collector (/frame)
2. Collector → Image Analysis
3. Collector → Face Recognition
4. Collector → Section
5. Collector → Alert

2.3 Scalability & Bottlenecks

I deployed HPAs:

- Camera
- Collector
- Image Analysis
- Face Recognition

Findings

- **Image Analysis** = highest CPU & memory usage (up to 1 CPU, 1Gi RAM).
- **Collector** uses low CPU but increases when many cameras stream at once.
- **Camera** autoscaled correctly (1 → 4 replicas at ~50% CPU).
- **Face Recognition** scaled moderately but stayed stable.

Main bottleneck

Image Analysis due to heavy computation per frame, its resource usage grows nonlinearly with more cameras.

```

PS D:\Vienna University\WS_Third_sem\Cloud Computing\assignments\Ass1\Ass1> kubectl get hpa -n assignment --watch
  NAME          REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
  camera-hpa    Deployment/camera-deployment   cpu: 1%50%   1        4        1        87s
  collector-hpa Deployment/collector-deployment  cpu: 1%50%   1        5        1        5h21m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 2%60%   1        6        1        59m
  image-analysis-hpa Deployment/image-analysis-deployment  cpu: 0%60%   1        6        1        59m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 404%60%  1        6        1        60m
  collector-hpa  Deployment/collector-deployment   cpu: 9%50%   1        5        1        5h22m
  camera-hpa    Deployment/camera-deployment   cpu: 3%50%   1        4        1        2m37s
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 404%60%  1        6        4        60m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 33%60%   1        6        6        60m
  image-analysis-hpa Deployment/image-analysis-deployment  cpu: 33%60%   1        6        1        60m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 419%60%  1        6        6        60m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 43%60%   1        6        6        60m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 282%60%  1        6        6        61m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 137%60%  1        6        6        61m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 223%60%  1        6        6        61m
  camera-hpa    Deployment/camera-deployment   cpu: 3%50%   1        4        2        3m45s
  collector-hpa  Deployment/collector-deployment   cpu: 18%50%  1        5        1        5h23m
  camera-hpa    Deployment/camera-deployment   cpu: 2%50%   1        4        1        3m51s
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 190%60%  1        6        6        61m
  image-analysis-hpa Deployment/image-analysis-deployment  cpu: 25%60%   1        6        1        61m
  face-recognition-hpa Deployment/face-recognition-deployment  cpu: 4%60%    1        6        6        62m

```

2.4 Kubernetes Objects

Deployments

Each service has its own Deployment with replica settings.

Services (ClusterIP)

Used for internal communication between services.

Ingress

Ingress routes configured as follows:

- /collector/* → collector-service
- /image-analysis/*
- /face-recognition/*
- /section/*
- /alert/*
- /camera/*

The nip.io domain was used to expose services externally, as required by the assignment.

Probes

Collector includes:

- /livenessProbe
- /readinessProbe

Resource Requests & Limits

All services include CPU & memory constraints (required by HPA).
Image Analysis uses highest resources:

- 500m CPU request → 1 CPU limit
- 512Mi → 1Gi memory.

2.5 Ingress Setup

Ingress Controller: **NGINX**

Namespace: **assignment**

Domain: **34.52.178.235.nip.io**

Purpose:

- Provide external access
- Route traffic to appropriate downstream service
- Enable testing with Postman

The screenshot shows the Postman application interface. A POST request is being made to `http://34.52.178.235.nip.io/frame`. The request body is a JSON object:

```
1 {  
2   "frameId": "test-frame-001",  
3   "timestamp": "2025-11-14T17:15:00Z",  
4   "cameraId": "camera-01",  
5   "imageData": "base64encodedstringhere",  
6   "personsDetected": [  
7     {  
8       "personId": "p001",  
9       "name": "John Doe",  
10      "confidence": 0.92  
11    }  
12  ]  
13 }
```

The response status is 200 OK, and the response body is:

```
1 {  
2   "status": "received"  
3 }
```

3. Cost Analysis

3.1 GKE Standard (Estimated)

Node: **e2-standard-2 (2 vCPU, 8GB RAM)**

1 Node → **\$136.50/month**

3 Nodes → **\$263.51/month**

Pros:

- Full control

- Best for complex microservices

Cons:

- You pay for nodes 24/7 even if idle

Cost Estimate Summary

As of Nov 14, 2025 • 11:38 PM
Prices in USD

Total estimated cost
\$263.51 / mo

COMPUTE

	\$263.51
GKE (Kubernetes Engine)	\$263.51
Service type	GKE
Number of Zonal clusters	1
Boot disk type	Standard persistent disk
Boot disk size (GiB)	10 GiB
Node-time	2190 Hours
Machine type	e2-standard-2, vCPUs: 2, RAM: 8 GB
Number of Nodes	3

Next steps

Request a custom quote
Connect with our sales team and learn

Total estimated cost
\$136.50 / mo

GKE (Kubernetes Engine)
\$136.50

	\$136.50
Service type	GKE
Number of Zonal clusters	1
Boot disk type	Standard persistent disk
Boot disk size (GiB)	10 GiB
Node-time	730 Hours
Machine type	e2-standard-2, vCPUs: 2, RAM: 8 GB
Number of Nodes	1
Operating System / Software	Free: Container-optimized
Kubernetes Edition	GKE
Provisioning model	Regular
Enable Confidential GKE Nodes	false

Next steps

Request a custom quote
Connect with our sales team and learn about any eligible discounts.

3.2 GKE Autopilot (Based on the standard estimated)

The workload baseline: ~1.7 vCPU, 2Gi RAM

Autopilot cost: $\approx \$150/\text{month}$

Pros:

- No node management
- Bills per Pod, not machine

Cons:

- Higher cost if large clusters

3.3 Cloud Run (Based on standard)

For REST services like Collector/Camera/Section/Alert:

100,000 requests / month $\rightarrow \approx \$20/\text{month}$

Pros:

- Cheapest for intermittent traffic
- Auto-scales instantly
- No nodes

Cons:

- Not suitable for stateful heavy ML tasks

3.4 Vision API (Based on standard)

Face detection + label detection

10,000 images / month $\rightarrow \approx \$13.50/\text{month}$

Pros:

- Cheapest AI option
- Best performance
- No servers needed

Cons:

- Requires redesign of system

3.5 Cost Comparison Table

Option	Monthly Cost	Annual Cost	Notes
GKE Standard	\$136–263	\$1,638–3,162	Manual node mgmt
GKE Autopilot	\$150	\$1,800	Easiest to manage
Cloud Run	\$20	\$240	Serverless
Vision API	\$13.50	\$162	Serverless ML

4. Resource Usage

Baseline usage

- ~1.7 vCPU total
- ~2.0 GiB total memory

Peak with scaling

- 3–4 vCPU
- 3 GiB memory

5. Problems Encountered & Solutions

1. Camera HPA showed <unknown>

Cause: missing CPU/memory requests.

Fix: Added requests.

2. Duplicate HPAs

Cause: duplicate application of manifests

Fix: Deleted extra resources → reapplied manifests.

3. Pending pods on Image Analysis

Cause: CPU/memory requests too high.

Fix: Reduced limits OR increased node count.

4. Ingress debugging required

Cause: Regex rewrite rules.

Fix: Added use-regex + rewrite target.

5. Cost estimation confusion

Fix: Clarified difference between node-based billing (Standard) and pod-based billing (Autopilot)

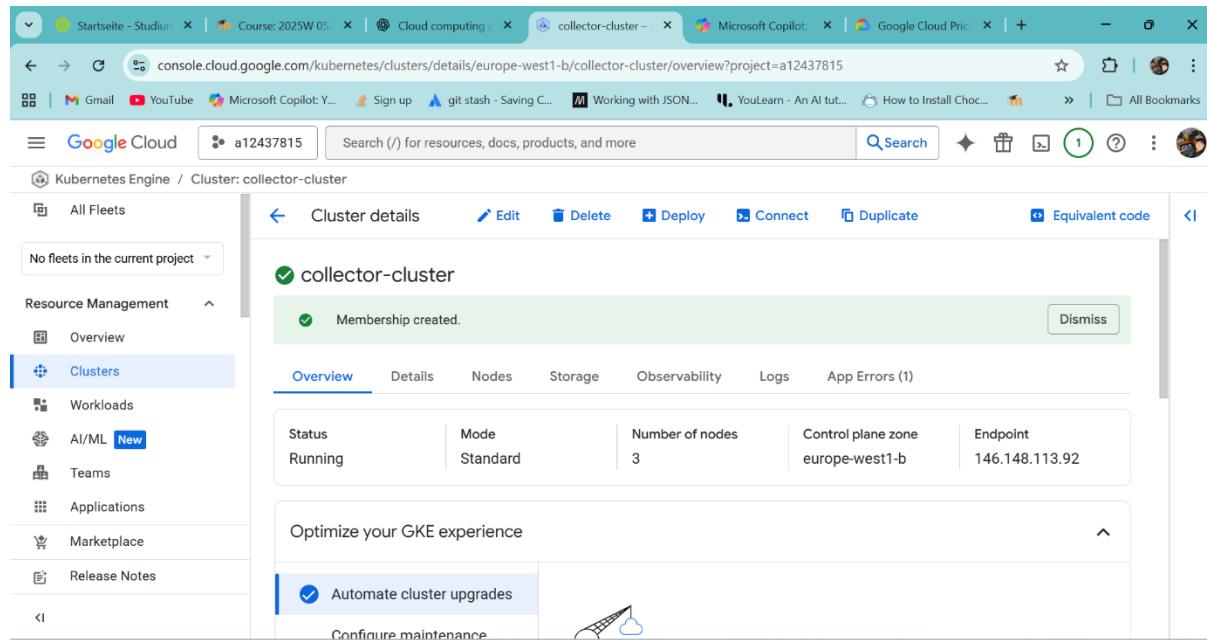
Note: I implemented and tested GKE Standard; the other estimates were based on its resource statistics.

6. Conclusion

This report documents the design, deployment, scaling, and cost analysis of a microservice-based camera processing system on GKE. The Collector service was implemented in Flask and integrated with five other services. Kubernetes features such as HPAs, resource constraints, and Ingress were configured. Cost estimates were calculated for GKE Standard, Autopilot, Cloud Run, and Vision API. The system was successfully deployed and scaled, with Image Analysis identified as the main bottleneck. Recommendations include exploring asynchronous communication and serverless alternatives for future optimization.

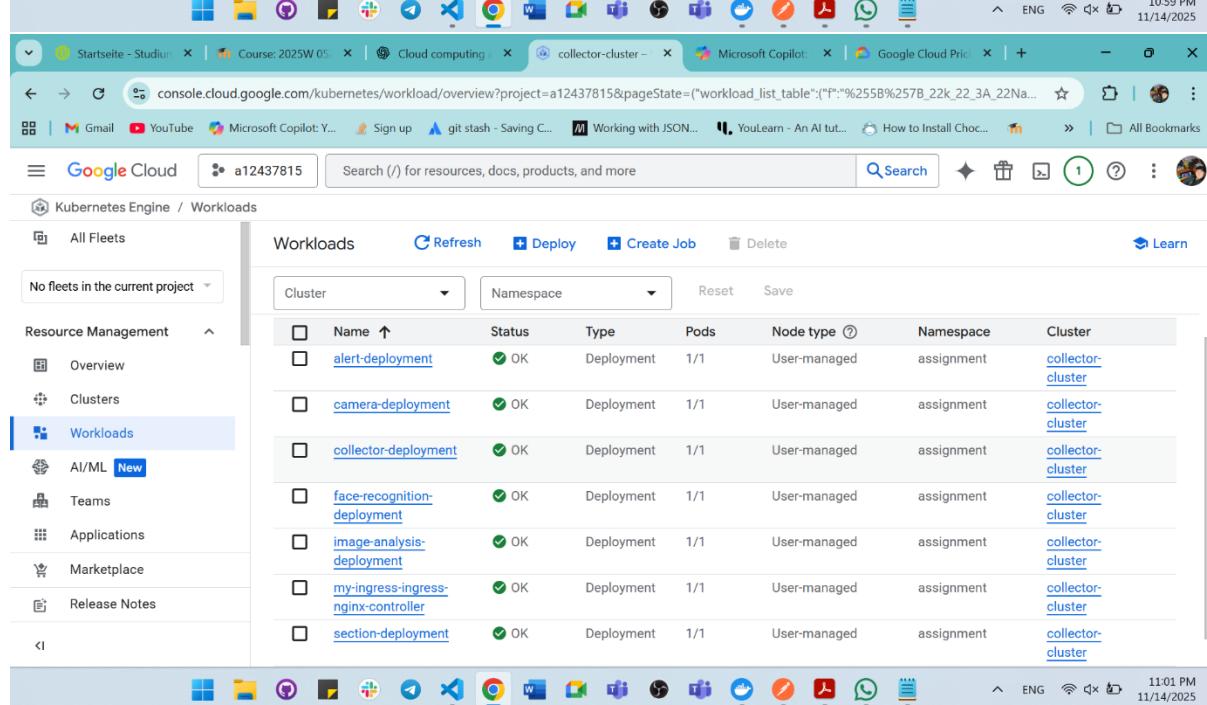
7. Screenshots

This are the screenshots assignment required.



The screenshot shows the 'Cluster details' page for the 'collector-cluster'. The cluster is currently running in Standard mode with 3 nodes. The endpoint is 146.148.113.92. A notification indicates 'Membership created.' A section titled 'Optimize your GKE experience' includes a checkbox for 'Automate cluster upgrades' which is checked. The status bar at the bottom right shows the date as 11/14/2025 and the time as 10:59 PM.

Status	Mode	Number of nodes	Control plane zone	Endpoint
Running	Standard	3	europe-west1-b	146.148.113.92



The screenshot shows the 'Workloads' page for the 'collector-cluster'. It lists seven deployments: alert-deployment, camera-deployment, collector-deployment, face-recognition-deployment, image-analysis-deployment, my-ingress-ingress-nginx-controller, and section-deployment. All are in an 'OK' status. The status bar at the bottom right shows the date as 11/14/2025 and the time as 11:01 PM.

Name	Status	Type	Pods	Node type	Namespace	Cluster
alert-deployment	OK	Deployment	1/1	User-managed	assignment	collector-cluster
camera-deployment	OK	Deployment	1/1	User-managed	assignment	collector-cluster
collector-deployment	OK	Deployment	1/1	User-managed	assignment	collector-cluster
face-recognition-deployment	OK	Deployment	1/1	User-managed	assignment	collector-cluster
image-analysis-deployment	OK	Deployment	1/1	User-managed	assignment	collector-cluster
my-ingress-ingress-nginx-controller	OK	Deployment	1/1	User-managed	assignment	collector-cluster
section-deployment	OK	Deployment	1/1	User-managed	assignment	collector-cluster

Google Cloud Platform - Kubernetes Engine / Services

Gateways, Services & Ingress

Cluster: a12437815 | Namespace: default

Gateway	Route	Status	Type	IP	Ports	Policies	Ingress
alert		OK	Cluster IP	34.118.237.121	1/1	assignment	collector-ingress
camera-service		OK	Cluster IP	34.118.228.124	1/1	assignment	collector-ingress
collector-service		OK	Cluster IP	34.118.231.254	1/1	assignment	collector-ingress
face-recognition		OK	Cluster IP	34.118.231.195	1/1	assignment	collector-ingress
image-analysis		OK	Cluster IP	34.118.237.21	1/1	assignment	collector-ingress
my-ingress-ingress-nginx-controller		OK	External load balancer	34.52.178.235:80	1/1	assignment	collector-ingress
my-ingress-ingress-nginx-controller-admission		OK	Cluster IP	34.118.226.205	1/1	assignment	collector-ingress
section		OK	Cluster IP	34.118.228.151	1/1	assignment	collector-ingress

Google Cloud Platform - Kubernetes Engine / Ingresses

Gateways, Services & Ingress

Cluster: a12437815 | Namespace: default

Name	Status	Type	Frontends	Services	Namespace	Clusters	
collector-ingress	OK	Custom	34.52.178.235.nip.io/collector(/ \$)(*)	...	collector-ingress	assignment	collector-ingress