Airport Security System - Assignment 1 Report

Student ID: 01642049
Course: Cloud Computing 2025W, University of Vienna

1. System Overview

The Airport Security System is a microservices-based application deployed on Google Kubernetes Engine (GKE) that processes surveillance camera feeds in real-time.

Architecture

The system consists of 6 microservices:

- **Camera Service**: Simulates surveillance cameras
- **Collector Service**: Orchestrates data flow (my implementation)
- **Image Analysis Service**: Detects persons, estimates age/gender
- **Face Recognition Service**: Identifies persons of interest
- **Section Service**: Stores person statistics
- **Alert Service**: Manages security alerts

Deployment Evidence: See `screenshots/GKE CLUSTER OVERVIEW.png` and `screenshots/WORKLOADS.png`

2. Communication & Data Flow

Data Flow

```
Camera → Collector → Image Analysis → Section Service
                  ↓
                  └→ Face Recognition → Alert Service
```

Collector Implementation

The Collector service orchestrates the entire data pipeline:

1. **Receives** frames from Camera (base64 image + metadata)
2. **Forwards** to Image Analysis for person detection
3. **Extracts** person data and sends to Section Service
4. **Sends** frame to Face Recognition for identification
5. **Forwards** alerts to Alert Service if persons of interest detected

Communication Method: Synchronous HTTP REST with 3-second timeouts

Service Discovery: Kubernetes DNS (`http://image-analysis`, `http://section`, etc.)

Verification: See `screenshots/TEST2.png` and `screenshots/TEST3.png` showing 68 persons detected and 4 alerts triggered.

---

3. Scalability & Bottlenecks

Horizontal Pod Autoscaler (HPA)

The Collector service uses HPA to automatically scale based on CPU usage:

- **Min replicas:** 1
- **Max replicas:** 5
- **Target CPU:** 60%

Current Status: See `screenshots/TEST4.png` showing HPA at 3% CPU utilization (1 replica sufficient for current load).

Bottlenecks

1. **Synchronous HTTP calls** - Sequential processing limits throughput
2. **Single-threaded services** - Each pod handles one request at a time
3. **Database contention** - Multiple Section/Alert pods share MongoDB

Cloud Run Suitability

- **Collector, Image Analysis, Face Recognition** - Stateless, event-driven
- **Section, Alert** - Need persistent database connections
- **Camera** - Continuous streaming would be expensive

---

4. Kubernetes Configuration

GKE Cluster

- **Name:** a1-cluster
- **Location:** europe-central2-a (Warsaw)
- **Machine Type:** e2-standard-4 (4 vCPU, 16GB RAM)
- **Nodes:** 1 (autoscaling 1-3)

Evidence: See `screenshots/GKE CLUSTER OVERVIEW.png`

Resource Allocation

All services have defined resource requests and limits to ensure proper scheduling and prevent resource contention. See manifest files in `a1/manifests/`.

---

5. Kubernetes Objects Used

Deployments

All 6 services deployed with resource limits, readiness/liveness probes. See `screenshots/WORKLOADS.png`.

Services

All use **ClusterIP** type for internal communication. External access via Ingress only. See `screenshots/SERVICES.png`.

Namespace

All resources deployed in namespace `a1` for isolation.

HorizontalPodAutoscaler

Configured for Collector service (1-5 replicas, 60% CPU target).

Ingress

NGINX Ingress Controller with path-based routing for all services. See `screenshots/INGRESS.png`.

---

6. Ingress Configuration

Setup

- **Type:** NGINX Ingress Controller
- **External IP:** 34.116.166.55
- **Hostname:** 34.116.166.55.nip.io

Routing

Path-based routing to all 6 services with URL rewriting:

- `/collector/*` → `collector:80`
- `/camera/*` → `camera:80`
- `/section/*` → `section:80`
- `/alert/*` → `alert:80`
- `/image-analysis/*` → `image-analysis:80`
- `/face-recognition/*` → `face-recognition:80`

Evidence: See `screenshots/INGRESS.png`

Benefits: Single LoadBalancer ($18/month) vs 6 separate LoadBalancers ($108/month)

---

7. Cost Analysis

Current Setup (GKE Standard)

Component	Monthly Cost
GKE Management	$73.00

Component	Monthly Cost
e2-standard-4 (1 node)	$97.82
LoadBalancer	$18.25
Storage & Egress	$3.60
TOTAL	**$192.67/month**

Yearly: $2,312/year

Alternative: GKE Autopilot

- Pay only for pod resources (850m CPU, 1152Mi RAM)
- **Cost:** ~$47/month ($568/year)
- **Savings:** 76% cheaper

Alternative: Cloud Run

- For this workload (~360k requests/month)
- **Cost:** ~$71/month ($852/year)
- Better for variable loads

Cloud Vision API

- Face Detection: $1.50 per 1k images
- For 360k images/month: **$540/month** (3× more expensive than self-hosted)
- Only cost-effective for <10k images/month

---

8. Resource Usage & Monitoring

Actual Resource Usage

See `screenshots/TEST5.png` for pod resource consumption:

- Face Recognition: 333m CPU, 439Mi memory (most intensive)
- Image Analysis: 128m CPU, 516Mi memory
- Other services: <20m CPU, <60Mi memory each

Monitoring

Prometheus + Grafana stack deployed for monitoring:

- **Cluster Overview:** See `screenshots/grafana-cluster-overview.png`
- **Namespace Metrics:** See `screenshots/grafana-namespace-a1.png`

Monitoring provides insights into:

- CPU/Memory usage trends
- Pod health and availability
- Network traffic patterns

- Resource optimization opportunities

---

9. Testing & Validation

System Tests

Comprehensive testing performed across all components:

1. **Cluster Status** - See `screenshots/TEST1.png`
    - All nodes healthy and ready
    - Kubernetes version: 1.33.5-gke.1201000
2. **Pod Health** - All 6 pods running and ready
    - Readiness/liveness probes passing
    - No crashes or restarts
3. **Service Connectivity** - All internal services accessible
    - DNS resolution working correctly
    - ClusterIP routing functional
4. **HPA Functionality** - See `screenshots/TEST4.png`
    - Current CPU: 3% (low load)
    - Ready to scale from 1 to 5 replicas
5. **End-to-End Data Flow** - See `screenshots/TEST2.png` and `screenshots/TEST3.png`
    - 68 persons detected and stored in Section service
    - 4 alerts triggered and stored in Alert service
    - Complete pipeline validated

---

10. Conclusion

Summary

Successfully deployed a scalable microservices-based airport security system on GKE with:

- 6 microservices with proper resource management
- Automatic scaling via HPA
- NGINX Ingress for external access
- Prometheus/Grafana monitoring
- Comprehensive testing and validation

Key Findings

- **Current cost:** ~$193/month (GKE Standard)
- **Optimization:** Could save 76% by switching to GKE Autopilot

- **Bottleneck:** Synchronous HTTP calls limit throughput
- **Scalability:** System handles multiple camera streams effectively

Recommendations

1. Switch to GKE Autopilot for cost savings
2. Implement async processing for better performance
3. Use managed databases (Cloud SQL) for Section/Alert services
4. Add message queue for high-volume scenarios

Screenshots Reference

- `GKE CLUSTER OVERVIEW.png` - Cluster configuration and status
- `WORKLOADS.png` - All deployments and pods
- `SERVICES.png` - Service endpoints
- `INGRESS.png` - Ingress configuration
- `TEST1.png` - Cluster status test
- `TEST2.png` - Person detection results (68 persons)
- `TEST3.png` - Alert system results (4 alerts)
- `TEST4.png` - HPA status
- `TEST5.png` - Resource usage metrics
- `grafana-cluster-overview.png` - Grafana cluster monitoring
- `grafana-namespace-a1.png` - Namespace-specific metrics

End of Report

Student ID: 01642049

University of Vienna - Cloud Computing 2025W