Cloud Computing - Assignment 1

Contents

1 Implementation details

1.1 General

For this Assignment I used Python/Flask.

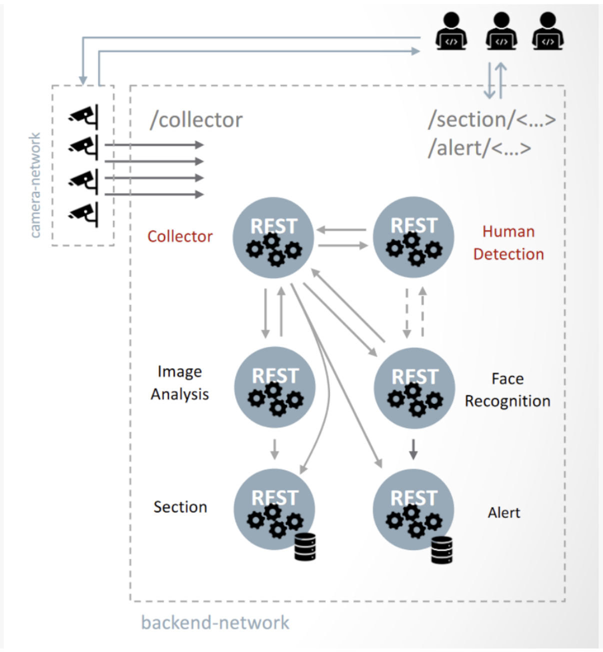


Figure 1: Overview of the whole system

My implementation is very similar to the image above. I have created two networks, one for the camera and one for the backend (all other services), where the Collector is the connection between those two networks.
Camera, Image Analysis, Face Recognition, Section and Alert were already implemented.

1.2 Human Detection

When I send a POST request to /frame it first checks if all requirements for further processing are met (e.g if the request contains a json file), then the request gets forwarded to the Google Vision API, which returns a response. In this response I check if a person was detected. In addition a person count is added to the original request.
If there is at least one person detected I am sending the request to other services (Image Analysis and Face Recognition) too for further analysis. The responses from those services will be received from the Collector not Human Detection!

1.3 Collector

The camera streams images to the collector (via POST requests), to /frame. Here the process is the same like in Human Detection where I check the request before further processing.
Then I send the request to Human Detection (from there see above).
But the Collector (as mentioned before) also receives POST request from other services, which we sent request to from Human Detection.
The Image Analysis sends the age and gender of the person on the image to the Collector at /persons, this information gets then send to Section by the Collector.

The Face Recognition sends to the Collector at /known-persons, from there I send the information to Alert (this was not specified in the Assignment but I wanted to include all services).

1.4 Both

For both services I added a ”start page” at /, which displays a message (this was mainly for testing purposes).
Both services are on port 8080, because I had some problems when using port 80.

1.5 Docker

The Dockerfile and requirements.txt are pretty much the same to the example project we got on Gitlab. [1]
I only added curl, which made it easier to test from inside the container.

1.6 Testing

I implemented the services locally, and tested it inside of the container network. I entered the container using:
docker-compose exec collector sh
From there I used curl to start streaming to the camera with:
curl -X POST http://camera1/stream?toggle=on -H ’Content-Type: application/json’ -d ’”destination”:”http://collector:8080/frame”,”max-frames”:10’

2 Problems during development

I had no prior experience with any of the used technologies in this assignment so it took me a lot of time to get familiar with them.

3 Possible Improvements

I played around a bit with the CPU usage. At first I had the limit at 0.5 for all services. But streaming 10 frames from the camera took more than a minute so I raised the limit to 1 for the most used services (camera, collector and human detection), which resulted in the streaming only taking 15-18 seconds. And I am pretty sure that changing the resource limits in other ways (which I have no idea how) could make it even faster.

4 Scalability

4.1 Is the application scalable?

Yes, it would be possible to add multiple instances of a service in the docker-compose file.

4.2 Which ones are the most beneficial services to scale?

The ones that are the most computationally intensive services: Camera, Collector and Human Detection.

4.3 What happens when the number of cameras increases?

Using multiple cameras made the streaming a bit faster, but at a certain amount of cameras there are no more improvements visible.

5 Service discovery

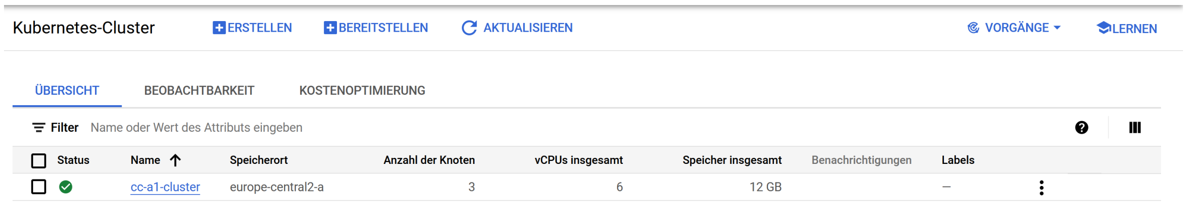
The services are reaching each other through predefined adresses inside the docker network, e.g: http://human-detection:8080/frame for Human Detection.

6 Kubernetes objects

I only used deployments and services, for each service. And I used LoadBalancer to be able to expose the services externally.

7 GKE Cluster configuration

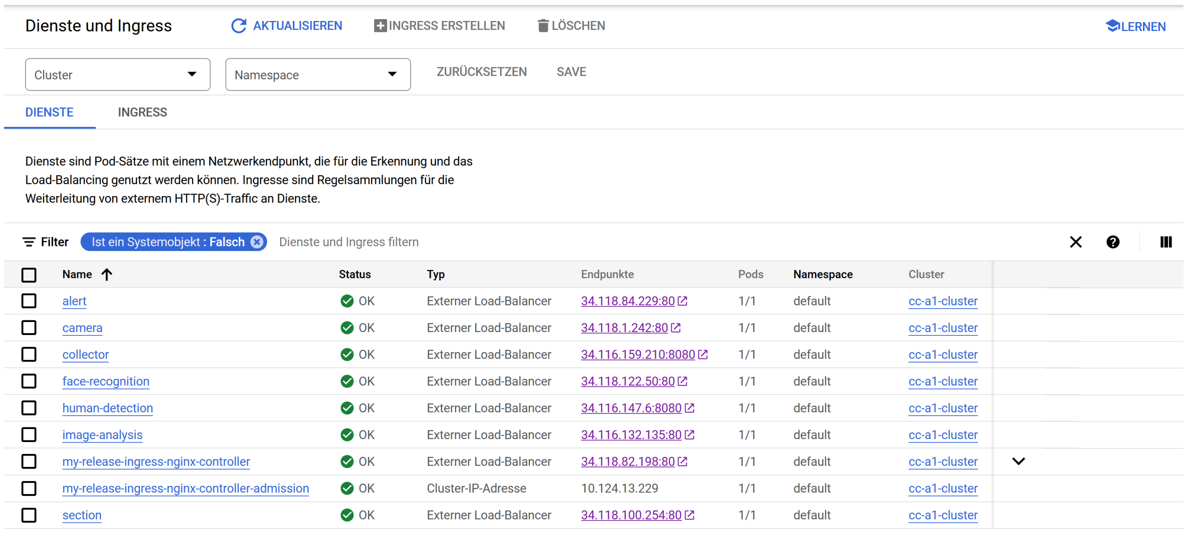
I have used the Standard GKE with the default configuration with 3 nodes.



Status	Name	Speicherort	Anzahl der Knoten	vCPUs insgesamt	Speicher insgesamt	Benachrichtigungen	Labels
✓	cc-a1-cluster	europe-central2-a	3	6	12 GB		–

Figure 2: Configuration

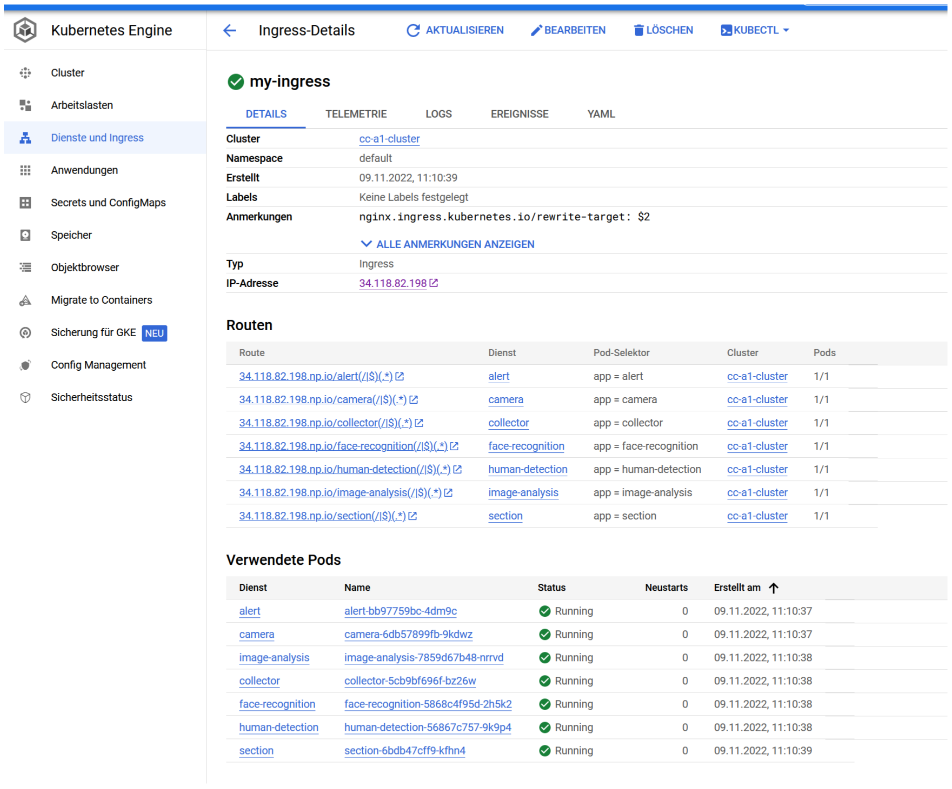
8 Ingress



Dienste sind Pod-Sätze mit einem Netzwerkendpunkt, die für die Erkennung und das Load-Balancing genutzt werden können. Ingresse sind Regelsammlungen für die Weiterleitung von externem HTTP(S)-Traffic an Dienste.

Name	Status	Typ	Endpunkte	Pods	Namespace	Cluster
alert	OK	Externer Load-Balancer	34.118.84.229:80	1/1	default	cc-a1-cluster
camera	OK	Externer Load-Balancer	34.118.1.242:80	1/1	default	cc-a1-cluster
collector	OK	Externer Load-Balancer	34.116.159.210:8080	1/1	default	cc-a1-cluster
face-recognition	OK	Externer Load-Balancer	34.118.122.50:80	1/1	default	cc-a1-cluster
human-detection	OK	Externer Load-Balancer	34.116.147.6:8080	1/1	default	cc-a1-cluster
image-analysis	OK	Externer Load-Balancer	34.116.132.135:80	1/1	default	cc-a1-cluster
my-release-ingress-nginx-controller	OK	Externer Load-Balancer	34.118.82.198:80	1/1	default	cc-a1-cluster
my-release-ingress-nginx-controller-admission	OK	Cluster-IP-Adresse	10.124.13.229	1/1	default	cc-a1-cluster
section	OK	Externer Load-Balancer	34.118.100.254:80	1/1	default	cc-a1-cluster

Figure 3: Ingress and Services



Cluster	cc-a1-cluster
Namespace	default
Erstellt	09.11.2022, 11:10:39
Labels	Keine Labels festgelegt
Anmerkungen	`nginx.ingress.kubernetes.io/rewrite-target: $2`
Typ	Ingress
IP-Adresse	34.118.82.198

Routen

Route	Dienst	Pod-Selektor	Cluster	Pods
34.118.82.198.np.io/alert(/|$)(.*)	alert	app = alert	cc-a1-cluster	1/1
34.118.82.198.np.io/camera(/|$)(.*)	camera	app = camera	cc-a1-cluster	1/1
34.118.82.198.np.io/collector(/|$)(.*)	collector	app = collector	cc-a1-cluster	1/1
34.118.82.198.np.io/face-recognition(/|$)(.*)	face-recognition	app = face-recognition	cc-a1-cluster	1/1
34.118.82.198.np.io/human-detection(/|$)(.*)	human-detection	app = human-detection	cc-a1-cluster	1/1
34.118.82.198.np.io/image-analysis(/|$)(.*)	image-analysis	app = image-analysis	cc-a1-cluster	1/1
34.118.82.198.np.io/section(/|$)(.*)	section	app = section	cc-a1-cluster	1/1

Verwendete Pods

Dienst	Name	Status	Neustarts	Erstellt am
alert	alert-bb97759bc-4dm9c	Running	0	09.11.2022, 11:10:37
camera	camera-6db57899fb-9kdwz	Running	0	09.11.2022, 11:10:37
image-analysis	image-analysis-7859d67b48-nrrvd	Running	0	09.11.2022, 11:10:38
collector	collector-5cb9bf696f-bz26w	Running	0	09.11.2022, 11:10:38
face-recognition	face-recognition-5868c4f95d-2h5k2	Running	0	09.11.2022, 11:10:38
human-detection	human-detection-56867c757-9k9p4	Running	0	09.11.2022, 11:10:38
section	section-6bdb47cff9-kfhn4	Running	0	09.11.2022, 11:10:39

Figure 4: Ingress

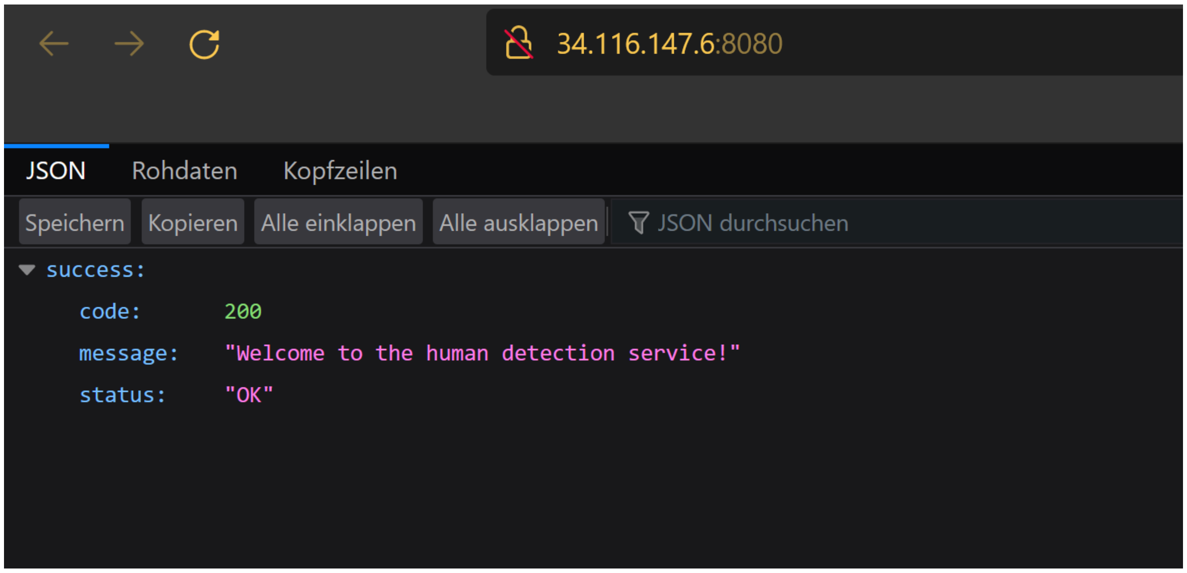
With
gcloud container clusters get-credentials cc-a1-cluster –zone europe-central2-a –project $PROJECT_ID
I connected to my GKE Cluster.

To be able to access the services I had to create the images in Google Cloud and push them to the Artifact registry, the already implemented services could be accessed through Dockerhub.

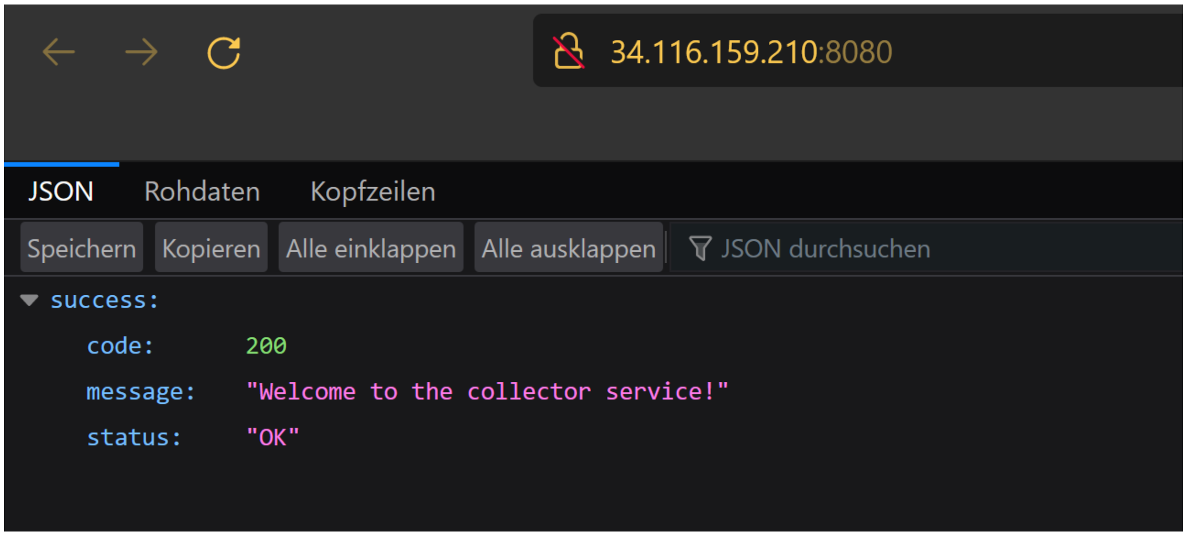
I used
kubectl apply -f .
to apply all .yml file in the manifest folders in GKE.

For the ingress file I followed the tutorial from the forum. [2]

The services can then be accesses through the ingress. The follwing images show the access to collector and human detection service trough the ingress.



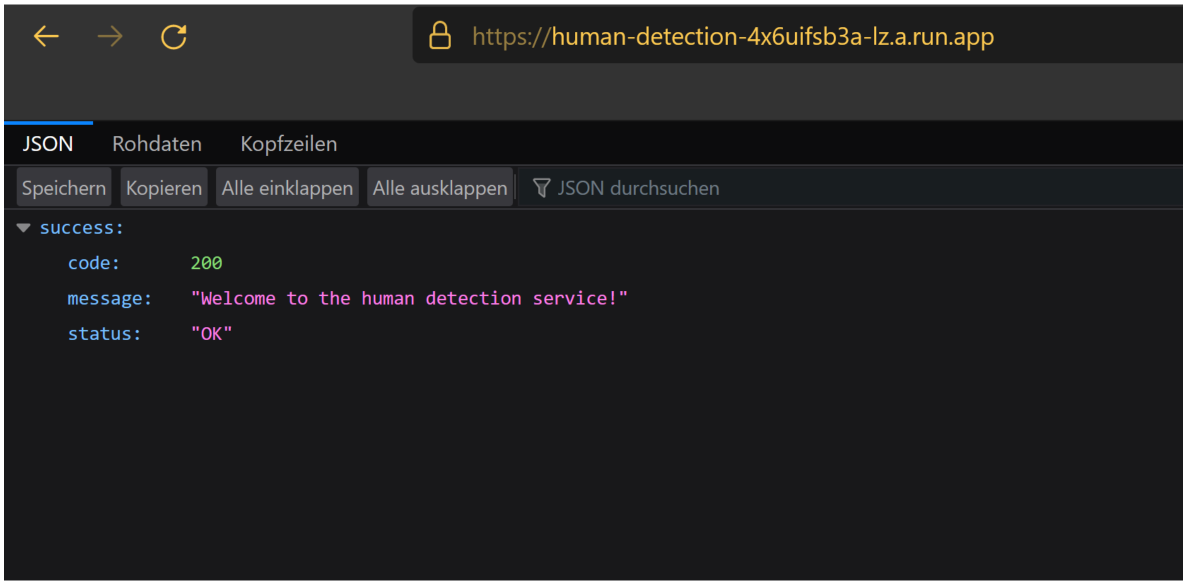
```
success:
    code:       200
    message:    "Welcome to the human detection service!"
    status:     "OK"
```



```
success:
    code:       200
    message:    "Welcome to the collector service!"
    status:     "OK"
```

9 Google Cloud Run

I used Google Cloud Run for the human detection service, therefore I used the human detection image which I pushed to the Artifact registry before.
Link to the service: `https://human-detection-4x6uifsb3a-lz.a.run.app`



```
▼ success:
    code:       200
    message:    "Welcome to the human detection service!"
    status:     "OK"
```

10 GCP Pricing Calculator

10.1 Standard GKE cluster

Total number of nodes in Node Pool: 3
Operating System/Software: Free: Container-optimized
Provisioning model: Regular
Machine Familly: General purpose
Series: N1
Machine type: n1-standard-2 (vCPUs: 2, RAM: 7.5GB)
Boot disk type: Balanced persistent disk
Boot disk size: 50 GiB
No GPUs
Local SSD: 0
Datacenter location: Warsaw (europe-central2)
Committed usage: 1 Year
Number of Zonal clusters: 1
Total Estimated Cost: USD 188.21 per 1 month[3]

10.2 Autopilot GKE cluster

Location: Warsaw (europe-central2)
Provisioning model: Regular
Replicas: 2
CPU: 2
Memory: 7.5 GiB

Ephemeral Storage: 50
Committed usage: 1 Year
GKE Autopilot Clusters: 1
Total Estimated Cost: USD 266.53 per 1 month [4]

10.3 Standard vs Autopilot

As expected, Autopilot is more expensive than Standard, but therefore it comes with many advantages: "With Autopilot, you no longer have to monitor the health of your nodes or calculate the amount of compute capacity that your workloads require. Autopilot supports most Kubernetes APIs, tools, and its rich ecosystem. You stay within GKE without having to interact with the Compute Engine APIs, CLIs, or UI, as the nodes are not accessible through Compute Engine, like they are in Standard mode. You pay only for the CPU, memory, and storage that your Pods request while they are running." [5]

10.4 Google Cloud Run

Location: Warsaw (europe-central2)
CPU: 1
Memory: 512MB
CPU allocation and pricing: CPU is only allocated during request processing
Concurrent requests per container instance: 20
Execution Time per Request: 500ms
Requests per Month: 100000000
Minimum number of instances: 1
Committed usage: 1 Year
Total Estimated Cost: USD 111.27 per 1 month [6]
100 Mio. requests per month to reach the price range of the other two services.

11 References

[1] https://gitta-lab.cs.univie.ac.at/public-examples/cloud-computing-extra-materials/-/tree/master/docker.

[2] https://moodle.univie.ac.at/mod/forum/discuss.php?d=3262765.

[3] https://cloud.google.com/products/calculator/id=e77ad11d-05de-41c1-b8da-8b96c2f1dcea.

[4] https://cloud.google.com/products/calculator/id=3a3a1353-0a82-4559-852a-f2a4d82cdf1b.

[5] https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview.

[6] https://cloud.google.com/products/calculator/id=3e4dc61c-38c1-4c0d-b291-92ecdc2ca54b.