Assignment 1 Overview

Due: 3.11.2024

In this exercise, you need to develop one service that operates within a system composed of multiple other services. All services need to communicate with each other, and the application needs to run in a Kubernetes cluster.

This assignment consists of two main tasks*:

1. [87%] Developing a Functional System Comprised of Services
    - Develop a service that works with the provided services
        - Services must be able to communicate between each other.
            - The system should tolerate failures to some extent; for example, a single failed request should not crash to whole system.
        - Write Kubernetes manifest files.
        - All services must have resource **requests** and **limits** specified.
    - Test and run the system on Kubernetes (Google Kubernetes Engine (GKE))
        - Development can be done locally.
        - On GKE the only entry point should be through an **Ingress**, which should forward requests to your services.
        - It should be possible to scale services using Kubernetes; some services need to scale automatically.
        - Deliver Kubernetes manifest files, source code of the service and write a report.
        - Use GCP Monitoring tools to check on cluster utilization, deploy Prometheus and Grafana and take a snapshot of your running application.

2. [13%] Estimating Costs Using the GCP Pricing Calculator
    - Estimate the infrastructure costs for one year for three different scenarios (assuming a standard configuration of a 4-node cluster):

        a. **Standard GKE cluster**
        Discuss the number of E2 machines you need, and what type with respect to the resource usage described in your deployment files (e.g., the number of E2 machines with 4vCPUs cores and 4GB RAM).

        b. **Autopilot GKE cluster**
        Compare to (a) GKE prices for a year, use the same number of vCPU as for the standard cluster (you don't need to deal with both clusters just calculate the pricings). Are there price differences and what are the pros and cons?

        c. **Google Cloud Run**
        Compare the pricing to what you would need to pay for Google Cloud Run. You can ignore the eventual storage requirements you may need, and try calculate how many requests per month you would need to have to reach to have roughly the same costs as with the previous two. Select 1 CPU, 512MB of memory per container, 500ms execution time per request, minimum 1 number of instances to be kept "warm".

*Details are provided on subsequent pages.

Deliverables

Your work results should include the following:

1. Source code of your services (upload to our Gitlab)

   - Submissions should be pushed to your repository on our Gitlab Server in `a1` folder.
   - A **Dockerfile** should be included with the source code of your service and **Kubernetes manifest files** (in YAML format). If you use **Docker Compose** for development, include it in your submission, along with resource constraints for each service.
   - Place the source code of the implemented service in one folder and **all manifest files** in a folder named `a1/manifests` (include Kubernetes Services, Deployments and HorizontalPodAutoscaler objects).

2. Report (A PDF in Moodle or PDF/README.md on our Gitlab):

   Your report should include **Implementation Details and Discussion of Problems Encountered** during development, you can use the following bullet points as a guideline:

   - **Communication / Data Flow:**
     - What information does the Collector service receive from other services, and what does it send to them?
     - Are there alternative communication methods, and why did you choose this one? Is the communication between service synchronous or asynchronous? Does this influence the performance, and can this be improved?
     - Did messages sent from camera services reach the Alert and Section services?
     - Explain how and why the services can see each other in your setup. Why does this work?
   - **Scalability and Bottlenecks (after testing on a Kubernetes cluster)**
     - Is your application scalable? Do you scale the whole system or only some services? Which ones are most beneficial services to scale? Is there anything that hinders scaling and can it be improved?
     - What happens when the number of Camera services increases? How would/does your code handle multiple streams (e.g., up to 4 cameras sending 100 frames)
     - For example, if you use multiple Camera services and scale the system by scaling individual services, could you (or have you been able to) process more images?
     - Describe how you **scaled** your application on Kubernetes with respect to Pod **replicas and autoscaling.** Could some services use Cloud Run, and could it be beneficial?
   - **Configuration of your cluster on GKE** (if different from the suggested one).
   - **Kubernetes Objects Used:**
     - List the types of Kubernetes objects you used (Deployments, Services, Pods, etc.) and for what purposes?
     - What kinds of services (NodePort, ClusterIP, or LoadBalancer)?
   - **Resource Constraints:** Identify the most computationally intensive services and explain how you specified resource constraints in your manifest files (Compose/Kubernetes)?
   - **Ingress:** Discuss the setup, type, and to which services are requests forwarded.
   - **Cost Analysis:** Discuss yearly costs of provisioning the cluster. **How does this compare to buying and maintaining your own hardware** (use rough estimates)? Share your opinion on the costs of running serverless, containerless vs clusters in GKE.
   - **Resource usage**: How many vCPUs and memory your system required, and how does this compare to your calculation?
   - **Screenshots:** Include screenshots of your GKE Cluster, Workloads, Services, and Ingress on GCP, as well as snapshots from Grafana.

   Document all that you found interesting, challenging, or problematic.

Task Description

The airport has undertaken a restructuring program, investing significantly in expanding its capacity by building new runways and terminals, followed up with a comprehensive marketing campaign. This resulted in an increased number of flights as expected. However, beyond the core business investments, the airport did not update its aging IT infrastructure due to limited funding.

The airport's software was developed years ago as a complex, monolithic application, maintained by a small team of employees. As the increasing number of flights and passengers grew, the system struggled to handle the increased workload. Limited scaling options were viable for a monolithic application, so server hardware upgrades made little impact. Furthermore, it was hard to keep up with new requests for changes (such as implementing evolving regulatory frameworks, patching policies, or adopting recent technological developments) proved challenging. Consequently, system downtime and staff overtime became frequent, making it clear that the system needs to be rethought to meet the business requirements at the level the company has scaled to.

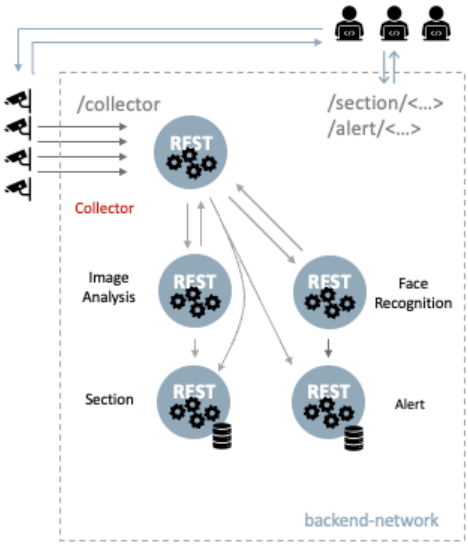
The airport conducted a small-scale analysis to identify upgrades that address the most critical services and minimize the time needed to achieve meaningful improvements. Considering the IT staff limited capacity, the development has been divided into several contracts to expedite progress. Each contract covers a group of related functionalities, and all will be executed simultaneously by different companies. One of those contracts has been awarded to you.

The focus of the first phase of your contract is airport security. The airport's surveillance cameras are positioned at each entrance or exit to monitor the number of people passing through various sections of the airport terminal. You have received the following set of requirements:

- They prefer having several smaller, more manageable services rather than a bigger monolithic application.
- The existing REST interfaces should be preserved.
- They decided to containerize their services, and they would like to use Kubernetes.

Detailed Description

Images from surveillance cameras are used to collect statistical data on how many people passed through specific gates or enter shops or restaurants. This data helps allocate additional staff to better support passengers at gates and improve overall security in crowded areas. Each captured image, including metadata (such as originating section, timestamp, etc.), must be sent to internal services for further analysis. The objective is to have statistical data on how many people (identified only by estimated age and gender by the ImageAnalysis service) pass through a particular section of the airport within a specified timeframe. Additionally, they want to be able to identify persons of interest and to be able to determine if they entered the airport area by using FaceRecognition service against a list of known individuals.



The current prototype includes the following services (details in subsequent sections):

- Camera: Represents a surveillance camera that captures images and sending them to the system for analysis.
- ImageAnalysis: Processes an image, identifies human faces, and estimates their age and gender.
- FaceRecognition: Compares a given image against a predefined database for possible matches.
- Section: Manages and provides statistical data for a specific section.
- Alert: Manages and provides information about detected persons of interest.
- Collector: Handles the communication flow, such as receiving images from Camera services and forwards them for processing to other services.

The Camera, FaceRecognition, ImageAnalysis, Section, and Alert services have already been implemented. You need to implement the Collector service. The requirements are outlined below (full API description is in another document *Assignment1-API.docx*).

Collector (to be implemented)

This service primarily handles the communication flow within the system (at least in part). For example, when it receives an image frame (via POST **/frame**), the frame should eventually reach the Section service(s), and some frames need to reach the Alert service(s). However, each image also needs to be processed by the **ImageAnalysis** and **FaceRecognition** services beforehand. Depending on your design, the **Collector** service may first forward the received image (*frame*) for analysis to the **ImageAnalysis** service, which returns an estimated age and gender of the person (if a person is found on the image), then transform this data if necessary and sends it in the appropriate format to the Section service (see the **Section** service description for **/persons**) for storage of statistical information. Additionally, the service should also send the frame to the **FaceRecognition** service, which will try to identify any persons of interest. Note that if the "destination: <URL>" field is sent to the **FaceRecognition** or **ImageAnalysis** service, these services will try to send the result to the specified destination; otherwise, the result will be returned in response to the original requestor, which is the Collector service

in this case. This service may interact with all or some of the other services, depending on your design decisions.

The Collector service API specification is as follows:

[Base URLs: http://collector (container network)]

POST /frame

Adds a new frame to the Collector service.

Body ("application/json"):

```
{
    "timestamp": "2010-10-14T11:19:18.03",
    "image": "<base64-encoded-string>",
    "section": 1,
    "event": "entry",
    "frame_uuid": "<uuid>",
    "destination": "<destination-url>", (optional)
    "extra-info": "" (optional)
}
```

Response ("if destination is provided"):

```
remote service response or just a code
```

Codes: 2xx if the operation was successful, otherwise 4xx.

POST /persons (optional)

Allows the user or another service to push detected person information.

Body ("application/json"):

```
{
  "timestamp": "2010-10-14T11:19:18",
  "section": 1,
  "event": "exit",
  "persons":[ { "age": "8-12", "gender": "male" } ],
  "destination": "<destination-url>", (optional)
  "image": "<base64-encoded-string>", (optional)
  "frame_uuid": "<uuid>", (optional)
  "extra-info": "" (optional)
}
```

POST /known-persons (optional)

Allows the user or another service to push detected information on persons of interest.

Body ("application/json"):

```
{
  "timestamp": "2010-10-14T11:19:18",
  "section": 1,
  "event": "exit",
  "known-persons":[  { "name": "Ms. X" }  ],
  "destination": "<destination-url>", (optional)
  "image": "<base64-encoded-string>", (optional)
  "frame_uuid": "<uuid>", (optional)
  "extra-info": "" (optional)
}
```

Response ("if destination is provided"):

```
remote service response or just a code
```

Codes: 2xx if the operation was successful, otherwise 4xx.

Objective Details

Your objective is to implement specified service using **Docker** for containerization, and a technology of your choice inside the container, and **Kubernetes** as a container orchestrator. In addition to ensuring basic communication and a functional system, you should decide how to **scale the system** or the individual services to better process streaming data (consider both vertical and horizontal scaling). Written documentation on your design decisions and implementation details is an essential part of this task. You need to elaborate on potential bottlenecks of your system, identify areas for improvement, and explain the reasons behind them.

There are different ways to pass data around the system. Make sure that you **include your choices** and rationale in your report. Additionally, describe how you use your Section service(s) and how data is saved and accessed. This service can operate in two ways: either using a database per service or share a common database by specifying environment variables. Briefly explain which option you have used and why you believe it is the better approach.

The implementation work **goes beyond simply implementing** the specified API. There are **multiple ways to approach the task**, and you may want to use additional back-end or containerization functionalities to establish communication and configure the system. Your decisions should be documented.

Additional remarks:

For this prototype, we assume that the system collects information for a limited duration. Unless specified otherwise, Camera services will stop streaming automatically after a predefined number of images in the database have been streamed.

All dates in this exercise should follow the format: "2019-10-12T09:49:14" or "2010-10-14T11:19:18.039111" (without time zone information).

For the implementation of the Collector service, you may choose any technology of your choice, as long as it can run in a Docker container and is accompanied by a Dockerfile. Note that already implemented services may occasionally fail, return errors, or become unresponsive; this is the intended behavior and should be discussed in the documentation, i.e., how to handle errors and how much of the whole system is affected.

Each service (deployment) should have defined resource requests and limits, including at least the CPU and memory usage for each service.

Tips:

- Begin with a configuration of the system that runs a single instance of each service. Once your system functions properly, attempt to reconfigure the system to run more instances of some or all services and observe how your system functions.
- Already implemented services are single-threaded and will not perform any asynchronous calls. This is implementation is intentional.

Infrastructure, Docker, Kubernetes

Note that camera services need to be able to stream to the **Collector** service. To reduce traffic, we will simulate the cameras by replicating the Camera service **inside** the GKE cluster. It is recommended to always send a predefined number of frames by setting the (`"max-frames": 1`) in the JSON payload when you send your **POST** request to `camera/stream?toggle=on`. If you want to start another instance of the Camera service with some initial configuration, you can start the Camera service with environment variables (see the full API document for details).

You will need to use the already implemented services. In Kubernetes you can specify them by using the URLs below in your Kubernetes manifest files. You can test the individual services by pulling these images from DockerHub:

docker pull ccuni/camera-service-2024w

docker pull ccuni/image-analysis-service-2024w

docker pull ccuni/face-recognition-service-2024w

docker pull ccuni/section-service-2024w

docker pull ccuni/alert-service-2024w

Once the image is on your local machine, you can test the service using Postman, Curl or any other tool of your choice. For example, running the Alert service would look something like:

> **docker run -p <yourport>:8080 /alert-service-2024w**

the you can test if the service works with

> **curl http://<yourport>:80/probe**

These are very simple, containerized Python Flask applications that implement the specified APIs to some extent. Remember that you need to work with the containerized services, not with the Python applications running on the host.

Note that a request to Camera service triggers a chain of requests throughout the system, so if the message is not reaching the final destinations, you will need to figure out where exactly the problem lies, and this may not be the service you are sending a request to. Note again that all already implemented services operate sequentially.

Google Cloud Platform

You can use GCP through the Web UI (Cloud Shell: https://console.cloud.google.com), which also has included a browser IDE. However, it may also be convenient to use the cloud via your own terminal. For these purposes you can use Google Cloud CLI.

Installation & Getting Started with Google Cloud CLI

To get started, you need to follow this tutorial: https://cloud.google.com/sdk/docs/install-sdk

Additionally, to use Kubernetes (GKE), you need an additional plugin, which you can install with the following command:

```
gcloud components install gke-gcloud-auth-plugin
```

You can read more about this plugin here: https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke

Creating a Kubernetes Cluster (example)

1. Go to https://console.cloud.google.com and create a new project with credits that you have received.
2. Open the side menu, select “Kubernetes Engine” and then choose “Clusters”.
3. Click “Create” to start creating a new cluster. Create a standard cluster with custom E2 nodes with 4 vCPUs and 4 GB of memory.
4. You can choose between a pool of 3 nodes or enable “Cluster Autoscaler” (after clicking on “default-pool”), choose “Standard Persistent Disk” as the boot disk type. Go to features and enable Cloud Monitoring (you may need enable Cloud Monitoring API for this).

Connecting to a cluster (example)

Before doing anything on your cluster, you need to connect to it. You can use your **Google Cloud CLI** and enter the following command:

```
gcloud container clusters get-credentials CLUSTER_NAME --zone ZONE --project PROJECT_ID
```

Alternatively, you can just copy the command from the **WebUI**. On the right side, where your clusters are listed, there is a menu that you can click for each cluster, which includes a “**Connect**” button.

Install NGINX Ingress

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm install my-release ingress-nginx/ingress-nginx
```

When the installation is done, you can see newly installed Kubernetes objects:

```
$ kubectl get services
NAME                                       TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)                      AGE
kubernetes                                 ClusterIP      10.8.0.1      <none>         443/TCP                      6m35s
my-release-ingress-nginx-controller        LoadBalancer   10.8.5.206    34.28.55.96    80:31575/TCP,443:31685/TCP   104s
my-release-ingress-nginx-controller-admission ClusterIP   10.8.14.56    <none>         443/TCP                      105s
```

Note that there is an IP address assigned to the newly created objects (in this case 34.28.55.96). Now you need to add your Kubernetes Ingress object.

Note that "34.28.55.96.**nip.io**" simply maps this hostname to the IP value before the "nip.io" part, allowing you to use it as a hostname for development purposes. You can then use Postman from your local machine to issue a **GET** request to http://34.28.55.96.nip.io/probe, or whichever path you want to test. For rewrite rules, you can use the following link: https://kubernetes.github.io/ingress-nginx/examples/rewrite/

For more details, refer to these links:

- https://kubernetes.github.io/ingress-nginx/deploy/
- https://kubernetes.github.io/ingress-nginx/examples/rewrite/
- https://cloud.google.com/community/tutorials/nginx-ingress-gke

Install Prometheus

Run the following command in Google Cloud Shell, when connected to a cluster:

```
helm install prom-release oci://registry-1.docker.io/bitnamicharts/kube-prometheus
```

Deploy Grafana

Follow the instruction in this link to deploy Grafana:

https://grafana.com/docs/grafana/latest/setup-grafana/installation/kubernetes/

Find additional tutorials and resources in Moodle.