Assignment #1

Overview

In this exercise you will be developing several services that will reside within a bigger application comprised of a set of small, independent services. These services need to communicate between each other regardless of their implementation language or framework. In the description below, we will list all of the services, but you are to implement only those that are explicitly requested.

Problem Description

The airport undertook restructuring and invested significantly in expanding its capacity, building new runways and terminals, following up with a comprehensive marketing campaign. This resulted in increased number of flights as expected. However, due to limited funding, apart from the core business investments, the airport failed to follow suit with their aging IT infrastructure.

Their airport software was engineered years ago as a complex, monolithic application, and is maintained by a small team of airport employees. When the increasing number of flights and passengers caused increasing workloads, the system struggled to keep up. Limited scaling options were shown 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 framework, patching policies or keeping up to date with recent technical developments. Consequently, system downtime and staff overtime are common, so it became clear that the system needs some rethinking to serve the business requirements at the level the company has scaled to.

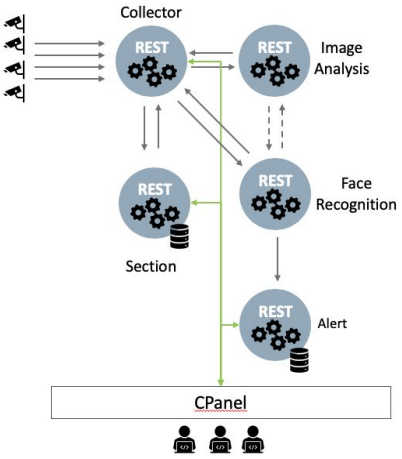
The airport conducted a small-scale analysis to identify upgrades that will address the most critical services and minimize the time needed to achieve meaningful improvements. Having in mind under-capacitated IT staff, the development has been divided into several contracts to expedite development. Each 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 main focus of the first phase of your contract is the airport security. Airport's surveillance cameras are placed on each entrance or exit to track the number of people passing through different section of the airport terminal. You have received a following set of preferences:

- They prefer having a number of 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 Docker as a platform.

Detailed Description

Images from surveillance cameras are used to collect statistical data on how many people passed through a particular gate, entered shop or restaurant, so they can put additional staff to better support people on gates, and to improve overall security in crowded areas. Each captured image and its timestamp is first sent to the Collector service and then passed through the system for further analysis. The goal is to have statistical data on how many people (identified only by estimated age and gender) passed through a particular section of the airport in a given timeframe. Additionally, they want to be able to identify persons of interest, if they entered the airport area by using a Face Recognition service against a list of known persons of interest.



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

- *CameraAgent*: represents a surveillance camera, capturing images, and sending them to the system for analysis
- *Collector Service*: mainly receives images from Camera Agents, and forwards them for processing, and/or forwards data to the Section service
- *ImageAnalysis Service*: takes an image, attempts to detect if there are any human faces on it, and then tries to estimate their age and gender
- *FaceRecognition Service*: compares images of persons on a given image against the list of persons in a predefined database for possible matches
- *Section Service(s)*: manages and provides statistical data for each section
- *Alert Service*: manages and provides information about detected persons of interest
- *CPanel Service*: RestAPI for system management

CameraAgent, ImageAnalysis and FaceRecognition services have been already implemented. You are to implement **Collector**, **Section**, **Alert** and **CPanel** services. Functional requirements are given below.

CameraAgents (already implemented)

These services simulate the surveillance cameras at the airport. Each produced image is signifying a person entering or exiting a section, depending on where the camera is located. When switched on, the service stream images (frames) to a specified Collector service instance. A captured image has two different representations - a plain raw image, with no other info attached, and the Frame object that also captures the time when the frame was captured, an id of the section in which the photo was taken, and whether the person was entering or exiting the area

Remark: Please note that since we do not have real cameras in this exercise, this service uses the dataset of human faces collected over the internet (http://vis-www.cs.umass.edu/fddb/). The service randomly chooses an image from this data set, and sends it to the Collector service. This service was implemented with JAX-RS.

The CameraAgents service API specification is as follows:

/state

GET

Returns the current operating state of the camera agent in JSON format

Response:

```
{
   "streaming": true,
   "cycle": 42,
   "section": 1, (set after first streaming started)
   "event": "entry", (set after first streaming started)
   "destination": "" (set after first streaming started)
}
```

/stream?toggle=on

POST

Turns ON or OFF streaming to the specified destination (section id), sending Frame objects in specified format to the destination. If toggle is not supplied, “on” is the assumed value.

Body

```
{
   "section": 1,
   "destination": "",
   "event": "entry",
   "format": "json", (optional)
   "extra-info": ""  (optional)
}
```

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

/frame (for testing purposes)

GET

Captures the latest image and returns a JSON or XML object, or raw image data. You can use HTTP headers to switch between the types.

Response (JSON example)

```
{
   "timestamp": "2010-10-14T11:19:18.039111Z",
   "section": 1,
   "event": "exit",
   "image": "<base64-encoded-string>",
   "extra-info": ""
}
```

Response (“image/jpeg” example)

```
raw mage, with "image/jpeg" content-type
```

Remarks: The `section` field is meant to signify in which section a frame was taken. The `destination` and `format` signify where the camera should stream images, while the `event` signifies if the camera captures frames when people enter or exit a section. The `extra-info` field is optional, and you can use it to pass around extra meta-data in case you need it. The `format` field is also optional, in case you want the CameraAgent service to produce XML objects. You can test the functionality of this service by invoking /frame, and see how the produced object looks like. The same object will be streamed to the specified `destination` in case of streaming.

ImageAnalysis (already implemented)

This service analyzes a given image for human faces, and tries to estimate age and gender of people detected on the image (note that this service cannot recognize a person). The collected JSON data is sent back as a response, or forwarded to specified destination.

The ImageAnalysis service API specification is as follows:

/frame

POST

Receives an image (i.e. frame in JSON) for analysis, and returns a list with estimated information on detected persons.

Body

```
{
   "timestamp": "",
   "image": "<base64-encoded-string>",
   "section": "1",
   "event": "entry",
   "destination": "", (optional)
   "extra-info": "" (optional)
}
```

Response

```
{
   "persons":[
      {
         "age": "8-12",
         "gender": "female",
         "section": "1",
         "event": "entry",
         "timestamp": "2010-10-14T11:19:18.039111Z"
      }
   ],
   "extra-info": "" (optional)
}
```

Code: On error 4xx is returned.

POST (for raw image)

Receives an image for analysis, and returns a list with information on detected persons.

Body (raw image example)

```
raw image, with "image/jpeg" content-type
```

Response (example - raw image body POSTed)

```
{
   "persons":[
      {
         "age":"[8-12]",
         "gender":"male",
         "timestamp":"2010-10-14T11:19:18.039111Z"
      }
   ]
}
```

Code: On error 4xx is returned.

Remarks: In case the "**destination**" field is provided, the result is forwarded to the destination.

FaceRecognition (already implemented)

This service compares received image against the predefined database of persons of interest for possible candidates.

The FaceRecognition service API specification is as follows:

/frame

POST

Receives an image for analysis in JSON format, and returns a list with information on detected persons in JSON.

Body

```
{
   "timestamp": "2010-10-14T11:19:18.039111Z",
   "image": "<base64-encoded-string>",
   "section": 1,
   "event": "entry",
   "destination": 1, (optional)
   "extra-info": "" (optional)
}
```

Response

```
{
   "timestamp":"2010-10-14T11:19:18.039111Z",
   "section": 1,
   "event": "entry",
   "image": "<base64-encoded-string>",
   "persons": [
      { "name": "Ms.X" }
   ],
   "extra-info": "" (optional)
}
```

Code: On error 4xx is returned, 2xx otherwise.

POST (raw image)

Checks if the provided image contains any known persons of interest. Accepts raw image data, and returns a list of detected persons names

Body

```
raw image, with "image/jpeg" content-type
```

Response

```
{
   "persons":[
      {
         "name":"Mr.X"
      }
   ]
}
```

Code: On error 4xx is returned, 2xx otherwise.

Remarks: It may take a while until you detect a person of interest, there are around 20k images, and only around 10 persons of interest. In case the `destination` field is provided, the result is forwarded to the destination.

Section (needs to be implemented)

This service takes care of statistical data of a single section. The list of all persons that passed through this section of the airport is kept in a data storage (e.g.: file or database).

The Sections service API specification is as follows:

/persons

POST

Creates a new entry in the list of detected persons passing through this section. Additionally, if a person is also captured on while exiting the section (event=exit), its flags a Person by setting the “departed: true”.

Body

```
{
   "persons":[
      {
         "age":"8-12",
         "gender":"male",
         "timestamp":"2010-10-14T11:19:18.039111Z",
         "section":"1",
         "event":"exit"
      }
   ],
   "extra-info": "" (optional)
}
```

Code: On error 4xx is returned, 2xx otherwise.

/persons?from={from}&to={to}&aggregate=count&departed=false

GET

Returns a JSON with a list of persons in this section in a given time frame (use below specified time format). Additionally, it allows `aggregate=count` for returning total number of persons in the section, and filtering by event type (if departed is false than it returns only people where `{"event":"entry"}`).

Response (/persons?from={from}&to={to}&departed=false)

```
{
   "persons":[
      {
         "age":"8-12",
         "gender":"male",
         "timestamp":"2010-10-14T11:19:18.039111Z",
         "section":"1",
         "event":"entry"
      }
   ],
   "extra-info": "" (optional)
}
```

Response (/persons?from={from}&to={to}&aggregate=count)

16

Collector (needs to be implemented)

This service acts as a collection point for multiple camera services. When it receives a frame (through POST /frame), the service forwards the image for analysis to the *ImageAnalysis* service, which returns estimated person's age and gender. This data is then transformed and sent in the corresponding format to the Section service (see the Section service description for /persons).

Additionally, the service sends the image to the FaceRecognition service, which tries to identify persons of interests. If destination is sent to the FaceRecognition service, it will try to send the result to the specified destination, otherwise, the result will be returned in a response to the Collector service.

The Collector service API specification is as follows:

/frame

POST

Adds a new frame to the Collector service for analysis.

Body

```
{
   "timestamp": "2010-10-14T11:19:18.039111Z",
   "image": "<base64-encoded-string>",
   "section": 1,
   "event": "exit",
   "extra-info": "" (optional)
}
```

Code: On error 4xx is returned, 2xx otherwise.

/persons

POST

Allows user or another service to push detected person information

Body

```
{
   "section": 1,
   "destination": "",
   "persons":[
      {
         "age": "8-12",
         "gender": "female",
         "timestamp":"2010-10-14T11:19:18.039111Z",
         "section": 42,
         "event": "exit"
      }
   ],
   "extra-info": "" (optional)
}
```

Code: On error 4xx is returned, 2xx otherwise.

Alerts (needs to be implemented)

This simple service handles alerts in the system. Alerts represent a detected persons of interest.

The Collector service API specification is as follows:

/

POST

Creates a new alert by providing an estimated age and gender of a person, as well as the image of the face along with the timestamp when it was captured.

Body

```
{
   "persons":[
      {
         "id":"", (generated)
         "name":"Ms.X",
         "timestamp":"2010-10-14T11:19:18.039111Z",
         "section": 1,
         "event": "entry",
         "image": "<base64-encoded-string>"
      }
   ],
   "extra-info": "" (optional)
}
```

/?from={from}&to={to}&aggregate=count

GET

Returns a JSON with a list of alerts in a given time frame (use below specified time format). Additionally, it allows `aggregate=count` for returning total number of created alerts.

Response

```
[
   {
      "id": 123,
      "timestamp":"2010-10-14T11:19:18.039111Z",
      "section": 1,
      "event": "exit",
      "persons":[
         {
            "name":"Mr.X"
         }
      ]
   }
]
```

/{id}

GET

Return the alert details, i.e. person's info, corresponding image and if the person is still present in this section.

Response

```
{
   "id": 122,
   "timestamp":"2010-10-14T11:19:18.039111Z",
   "section": 1,
   "event": "exit",
   "persons":[
      {
         "name":"Ms.X"
      }
   ]
}
```

DELETE

Removes the alert.

Code: On error 4xx is returned, 2xx otherwise.

CPanel (needs to be implemented)

This service loads a configuration of the system and allows user to retrieve and modify it. The configuration of the system addresses at least list of all camera agents, list of all sections and their descriptions, and the relations between the two (which sections use which cameras).

The Collector service API specification is as follows:

/sections

GET

Returns a list of all available sections

Response

```
{
   "sections":[
      {
         "id": 1,
         "description": "Gate #1"
       }
   ]
}
```

/cameras

GET

Returns a list of all available cameras. This value is read from an internal service storage (file or database).

Response

```
{
   "cameras":[
      {
         "id": 2,
         "type": "exit",
         "section": 1,
      }
   ]
}
```

/cameras/{id}/*

Map to CameraAgent service API

/sections/{id}/*

Map to Section service API

/alerts/*

Map to Alerts service API

Start with a configuration of the system that runs a double instance of CameraAgent (with one entry and one exit camera), and a single instance for each other service. Once your system functions properly, try to reconfigure system for more instances and observe how your system functions. You will need to configure your system to somehow see new instances after reconfiguration. Try to identify bottlenecks, and weak parts of the system. Write down your comments.

Objectives

Your objective is to implement specified services, using technology of your choice. The services should function independently, and each service should run in a Docker container. Keep in mind:

- Implementation work goes beyond implementing the specified API. There is more than one way to implement the task, and you may want to use additional back-end functionalities to organize communication, configure the system, etc.

- For this prototype we assume that the system collects information only for some limited amount of time. Actually, a Camera Agent will stop streaming automatically after 1 day of operation.
- For all dates in this exercise use ISO8601 UTC format e.g: “2019-10-12T09:49:14.749622Z” (https://en.wikipedia.org/wiki/ISO_8601).
- There are different ways to pass data around the system, document your approach.
- All the source codes are located on our Gitlab server under cc19/task1.git
- If a response HTTP code is not specified, select an appropriate one.
- For the implementation, you may choose any technology you prefer, as long as it can in a Docker container and ships with a Dockerfile.

Deliverables

Your work results should include the following:

1. **Source code of your services**
   All your submissions must be pushed to our Gitlab Server, hosted on https://gitta-lab.cs.univie.ac.at. You will receive your GitLab account info via email, and you will be able to use it once you activate it. Use separate folder for each service and remember to upload (git push) your source codes.
   More on how to use git: https://git-scm.com/docs

2. **Dockerfile for builds**
   A Dockerfile should be included with your service source code. You should have your containers runnable on the Nova machine.

3. **Documentation**
   a. Describe your implementation and discuss problems you have encountered during the development.
   b. How your application could be improved?
   c. Try to identify potential problems or challenges as the demand increases
      For example, what happens when the number of cameras increase, how will your code handle more data being streamed? Could some parts of the system be improved to support better scaling?

For each service, provide a README file with instructions on how to build and run the service, as well for the setup of the whole system.

Setup and Getting Started

First, download already implemented services from the git:

```
git clone https://gitta-lab.cs.univie.ac.at/cc19/task1.git
```

It is probably easier to develop the system on your laptop. Try to grab the available Dockefiles from the git repo, and run them on your host. Develop other services, and as you complete, put them in containers. Finally, put all your containers on Nova, and try to set them up, with their own network.

Setting up containers on Nova can be challenging though, as you will need to setup a network for your containers to use (read more in the link below).

Environment Setup

You can develop this assignment on your own computer, but you should try to run it, and test it on the "Nova" cluster (www.par.univie.ac.at/index.php?goto=hardware). You can login via ssh at nova.cs.univie.ac.at with a username/password that will be emailed to you.

If you would like to use Nova from the outside for testing, you will need to create ssh local port forwarding on your machine (see below).

Accessing your services from outside of the Nova cluster

Your application running on the Nova machine may use one or more ports to perform its function. From within the cluster you can access and test your service as you would on any other machine. However, communication with the Nova machine from the outside world is possible only through port 22. As this is the ssh port, you can only access the machine from the outside by local port forwarding. In case you would like to use a web interface, or Postman, or some graphical tool for testing your API, then use port forwarding. For instance, if you would like to use localhost:8888 to access a service that is running on port 9200 on Nova, you could forward port of the service on nova by doing:

ssh -L 8888:localhost:9200 <your-username>@nova.cs.univie.ac.at

Running in Docker containers

You will need to make a Dockerfile, build it, and run it, and probably connect to a network. For example, you want to have your app in a Docker image called "my-docker-image", assuming you have already created your Dockerfile, you can just build it:

```
docker build -t my-docker-image .
```

And then run ti:

```
docker run -p8080:8080 -d --network <network-name> --name my-container my-docker-image
```

The second command tells docker to run your container in the background, and to connect it to the specified network. In docker you can define on which Docker-internal network your containers will be running, and communicate to each other. On your laptop, you can probably use "host" as a network as it makes it easier.

However, on the Nova system you will not be able to connect your container to the **host** network. That is because of the user namespaces and shared system. The workaround is to create a network for your containers where they can communicate between each other. Learn more here:

https://docs.docker.com/v17.09/engine/userguide/networking/work-with-networks/#connect-containers

Running inspect command will give you network configuration of your containers:

```
docker inspect my-container-name
```

You will probably be interested in the following fields: "IPAddress": "172.17.0.x", as this will be the IP address you container uses on the internal network.

Not everything has to be done manually, you can use `docker-compose` for this purposes. If you specify a list of your services in `docker-compose.yml` file, docker-compose can automate the process of building and configuring network, services, and communication between containers. For instance you can also access your services via `http://service-name`:port, instead of having a configurations with `http://hostname:port`. You can find more info on the following link: https://docs.docker.com/compose/gettingstarted/.

Special remarks for Docker on Nova

Due to user namespaces that are used by Docker on Nova, you may encounter the following error:

"failed to copy files: failed to copy directory: Error processing tar file(exit status 1): Container ID 101001 cannot be mapped to a host ID"

In that case your Dockerfile needs to change ownership of the files you are trying to copy. Just include " && chown root:root" in your RUN commands so that the COPY command later has access to copy the files.

Due to special setup on nova, your application server within the Docker container should not run on localhost:<port>, but on 0.0.0.0:<port>.