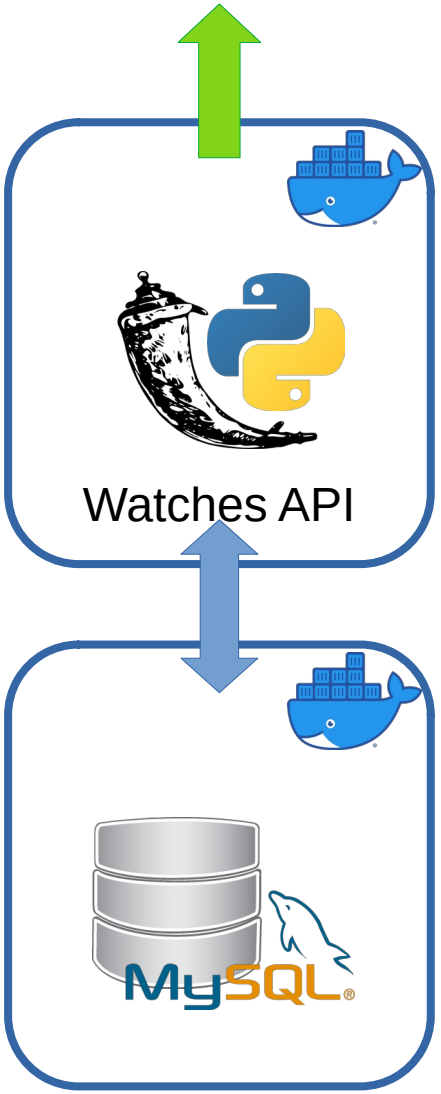
CLOUD COMPUTING PROJECT

WATCHES
WEBSERVICES

PART I

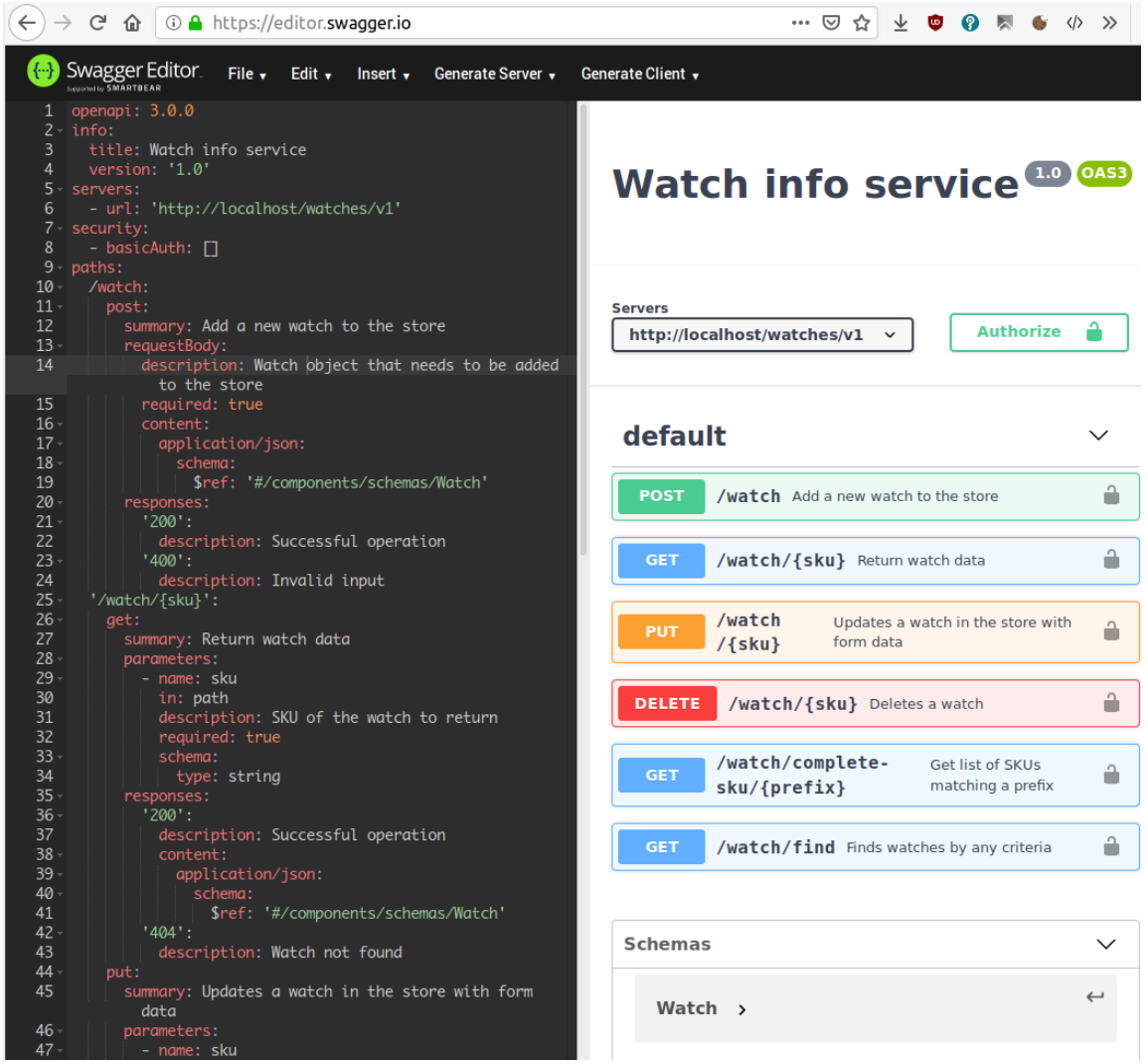
Part I - Objectives

- Develop a watch info service API
  - OpenAPI Spec
  - MySQL Data
  - Flask / Python
- Containerize
  - Docker
    - Rest API
    - MySQL



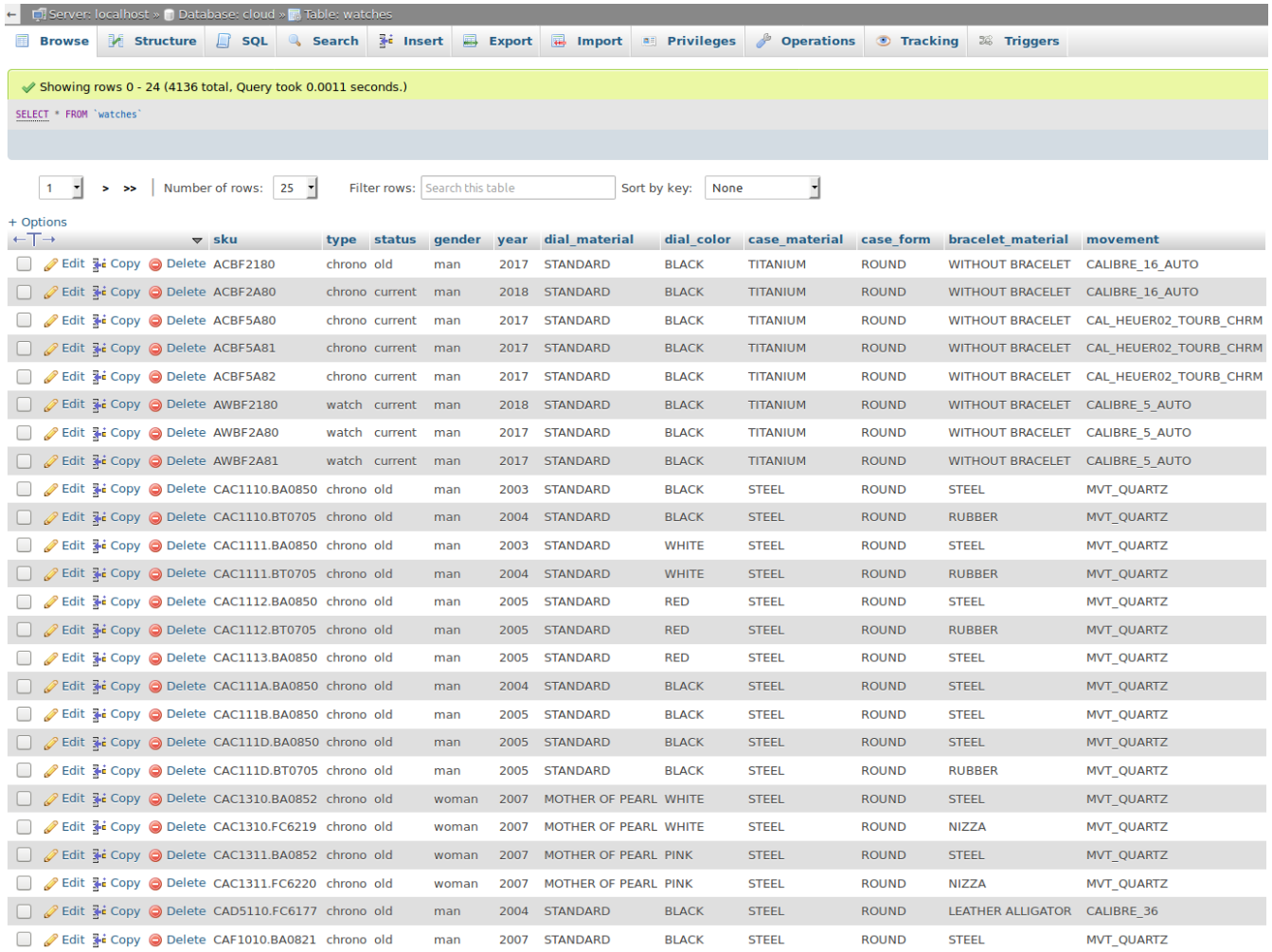
Swagger

- A tool to create and test OpenAPI specifications
- https://editor.swagger.io/



OPENAPI INITIATIVE

MySQL Data



Server: localhost » Database: cloud » Table: watches

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

Showing rows 0 - 24 (4136 total, Query took 0.0011 seconds.)

```
SELECT * FROM `watches`
```

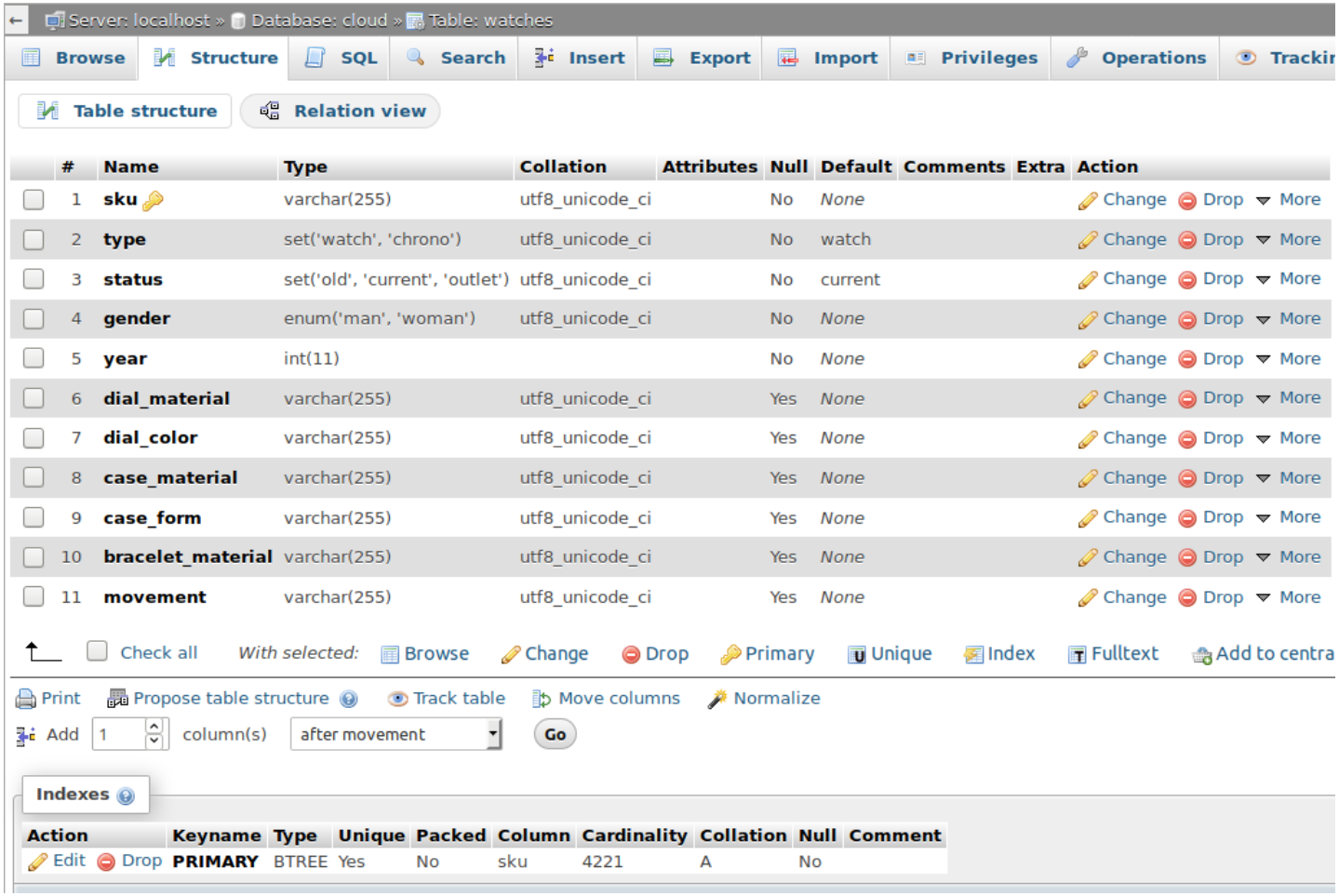
1 > >> | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

+ Options

	sku	type	status	gender	year	dial_material	dial_color	case_material	case_form	bracelet_material	movement
Edit Copy Delete	ACBF2180	chrono	old	man	2017	STANDARD	BLACK	TITANIUM	ROUND	WITHOUT BRACELET	CALIBRE_16_AUTO
Edit Copy Delete	ACBF2A80	chrono	current	man	2018	STANDARD	BLACK	TITANIUM	ROUND	WITHOUT BRACELET	CALIBRE_16_AUTO
Edit Copy Delete	ACBF5A80	chrono	current	man	2017	STANDARD	BLACK	TITANIUM	ROUND	WITHOUT BRACELET	CAL_HEUER02_TOURB_CHRM
Edit Copy Delete	ACBF5A81	chrono	current	man	2017	STANDARD	BLACK	TITANIUM	ROUND	WITHOUT BRACELET	CAL_HEUER02_TOURB_CHRM
Edit Copy Delete	ACBF5A82	chrono	current	man	2017	STANDARD	BLACK	TITANIUM	ROUND	WITHOUT BRACELET	CAL_HEUER02_TOURB_CHRM
Edit Copy Delete	AWBF2180	watch	current	man	2018	STANDARD	BLACK	TITANIUM	ROUND	WITHOUT BRACELET	CALIBRE_5_AUTO
Edit Copy Delete	AWBF2A80	watch	current	man	2017	STANDARD	BLACK	TITANIUM	ROUND	WITHOUT BRACELET	CALIBRE_5_AUTO
Edit Copy Delete	AWBF2A81	watch	current	man	2017	STANDARD	BLACK	TITANIUM	ROUND	WITHOUT BRACELET	CALIBRE_5_AUTO
Edit Copy Delete	CAC1110.BA0850	chrono	old	man	2003	STANDARD	BLACK	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC1110.BT0705	chrono	old	man	2004	STANDARD	BLACK	STEEL	ROUND	RUBBER	MVT_QUARTZ
Edit Copy Delete	CAC1111.BA0850	chrono	old	man	2003	STANDARD	WHITE	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC1111.BT0705	chrono	old	man	2004	STANDARD	WHITE	STEEL	ROUND	RUBBER	MVT_QUARTZ
Edit Copy Delete	CAC1112.BA0850	chrono	old	man	2005	STANDARD	RED	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC1112.BT0705	chrono	old	man	2005	STANDARD	RED	STEEL	ROUND	RUBBER	MVT_QUARTZ
Edit Copy Delete	CAC1113.BA0850	chrono	old	man	2005	STANDARD	RED	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC111A.BA0850	chrono	old	man	2004	STANDARD	BLACK	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC111B.BA0850	chrono	old	man	2005	STANDARD	BLACK	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC111D.BA0850	chrono	old	man	2005	STANDARD	BLACK	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC111D.BT0705	chrono	old	man	2005	STANDARD	BLACK	STEEL	ROUND	RUBBER	MVT_QUARTZ
Edit Copy Delete	CAC1310.BA0852	chrono	old	woman	2007	MOTHER OF PEARL	WHITE	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC1310.FC6219	chrono	old	woman	2007	MOTHER OF PEARL	WHITE	STEEL	ROUND	NIZZA	MVT_QUARTZ
Edit Copy Delete	CAC1311.BA0852	chrono	old	woman	2007	MOTHER OF PEARL	PINK	STEEL	ROUND	STEEL	MVT_QUARTZ
Edit Copy Delete	CAC1311.FC6220	chrono	old	woman	2007	MOTHER OF PEARL	PINK	STEEL	ROUND	NIZZA	MVT_QUARTZ
Edit Copy Delete	CAD5110.FC6177	chrono	old	man	2004	STANDARD	BLACK	STEEL	ROUND	LEATHER ALLIGATOR	CALIBRE_36
Edit Copy Delete	CAF1010.BA0821	chrono	old	man	2007	STANDARD	BLACK	STEEL	ROUND	STEEL	MVT_QUARTZ

MySQL Schema

- Data directly maps to API spec



Step #1 - Local install

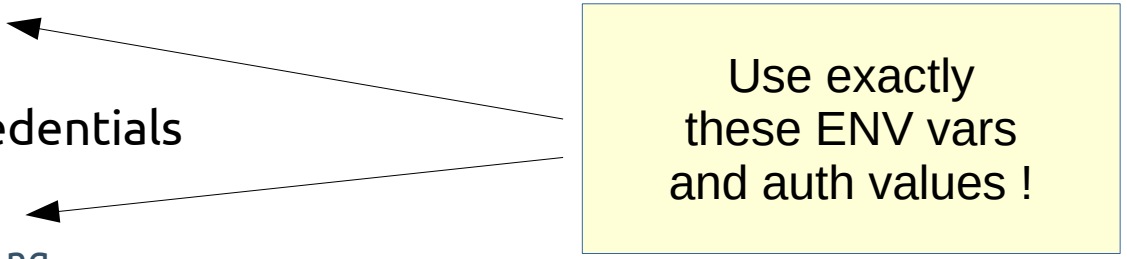
- Install MySQL
  - `$ sudo apt install mysql-server mysql-client-5.7`
  - Install PHPMyAdmin to create a new user and database
    - `$ sudo apt install phpmyadmin`
  - Load the data (CLI or PHPMyAdmin)
    - `$ mysql -u <username> -p <database> < watches.sql`
- Install Python 3 & pip3
  - `$ sudo apt install python3 python3-pip`

Step #2 - Local dev

- Develop your API in Python
  - Use pip3 for dependencies management
    - https://pip.pypa.io/en/stable/
    - Dependencies in: **`requirements.txt`**
      - Flask
      - PyMySQL
  - Server (single file): **`server.py`**
    - Listen on port **`1080`** (not privileged)

Step #3 - ENV vars

- DB parameters
  - `DB_HOST=127.0.0.1`
  - `DB_PORT=3306`
  - `DB_DBNAME=watches`
  - `DB_USER=watches`
  - `DB_PASS=watches`
- HTTP basic auth credentials
  - `HTTP_USER=cloud`
  - `HTTP_PASS=computing`
- Create a shell script **`run.sh`**
  - Set ENV vars
  - Start: `server.py`
    - `Using flask run`



Step #4 - Validate API

- Using Swagger
  - https://editor.swagger.io/
  - Load info_openapi_v1.yaml
  - Set authorization
  - **Test all endpoints**
    - Curl commands are also generated
      - Adapt the port to use 1080 instead of 80
- Do not proceed with next steps until your API works as expected

Step #5 - Optimizations

- 1. Add indexes to the DB Data
  - In order to improve lookup speed
  - Update `watches.sql` with your changes (dump the DB)
- 2. Set HTTP expiration headers
  - All data GET should be valid 1 hour
- These 2 optimizations are optional and will be considered as bonus points

Step #6 - Info-service in Docker

- Create a Dockerfile
  - Embed your Python development in a Docker image
  - Default action: start the server
  - `$ docker build -t info-service-v1 .`
-
```
$ docker run -d -p 1080:1080 --network=host \
  -e "HTTP_USER=cloud" \
  -e "HTTP_PASS=computing" \
  -e "DB_HOST=127.0.0.1" \
  -e "DB_PORT=3306" \
  -e "DB_DBNAME=watches" \
  -e "DB_USER=watches" \
  -e "DB_PASS=watches" \
  info-service-v1
```

Using `--network=host`, your docker instance should be able to connect directly to the MySQL instance running on host machine

Step #7 - MySQL in Docker

- Use https://hub.docker.com/_/mysql/
  - Read the documentation
    - See how to load `watches.sql` at startup or via an external **volume**

Step #8 - Compose

- Write **`docker-compose.yml`**
  - Set ENV vars
  - Run images
    - `info-service-v1`
    - `mysql`
  - Bind `cc-server` with `mysql`
- `$ docker-compose up`
  - Everything should start and work
  - More infos about docker compose in 2 weeks

Deliverables

- Python server
  - `server.py`
  - `requirements.txt`
  - `run.sh`
  - (updated) `watches.sql`
- Docker
  - `Dockerfile` → `info-service-v1`
  - `docker-compose.yml`
- `README`
  - Indicate clearly **what is working or not** and additional information in order to test your project

Committing

- Push your development in your Gitlab assignment repository (`/project`)
  - Commit your work step by step with commit message
  - If working by team (max 2 students)
    - Indicate all the participants and the central repository to Rémi
    - Every member of the team should commit regularly with their own user
  - Grant access and share information with Rémi until next week
    - Gitlab: **dulongr**
    - remi.dulong@unine.ch

Grading

- You start with 1 point !
- Server in Python
  - 3 points
- Dockerize service
  - 1 point
- MySQL in Docker + Docker compose
  - 1 point
- Bonus optimizations
  - 0.5 point

Delay

- Part I
  - Documentation: TODAY
  - Deadline: 2019-10-30T23:59:59+02:00
- Part II
  - Documentation: 2019-10-24
  - Deadline: 2019-11-20T23:59:59+02:00
- Part III
  - Documentation: 2019-11-14
  - Deadline: 2019-12-11T23:59:59+02:00