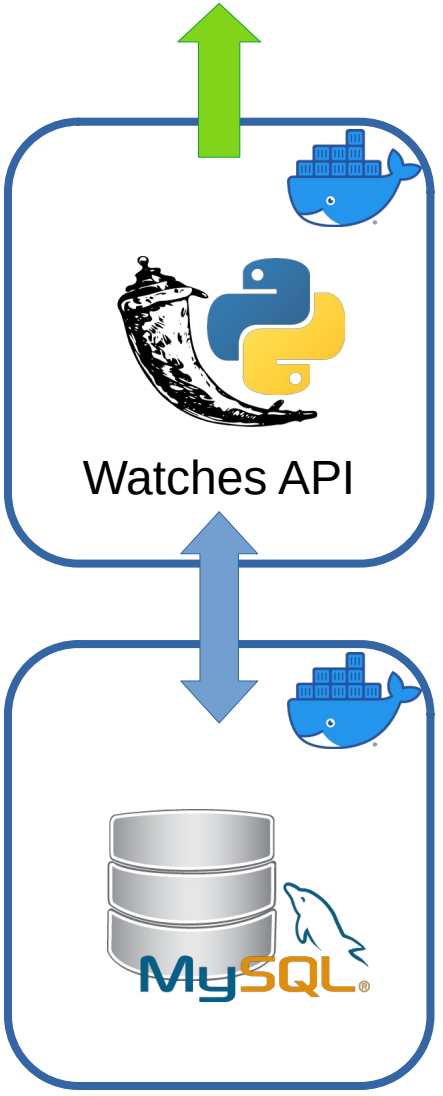
CLOUD COMPUTING PROJECT

WATCHES
WEBSERVICES

PART I

Part I - Objectives

- Develop a watch info service API
  - Follow OpenAPI Spec
  - Practice Docker containers
    - MySQL/MariaDB Database
    - Webservice API



Data

sku	type	gender	year	dial_material	case_material	bracelet_material	movement
ACBF2180	chrono	man	2017	STANDARD	TITANIUM	WITHOUT BRACELET	CALIBRE_16_AUTO
ACBF2A80	chrono	man	2018	STANDARD	TITANIUM	WITHOUT BRACELET	CALIBRE_16_AUTO
ACBF5A80	chrono	man	2017	STANDARD	TITANIUM	WITHOUT BRACELET	CAL_HEUER02_TOURB_CHRM
ACBF5A81	chrono	man	2017	STANDARD	TITANIUM	WITHOUT BRACELET	CAL_HEUER02_TOURB_CHRM
ACBF5A82	chrono	man	2017	STANDARD	TITANIUM	WITHOUT BRACELET	CAL_HEUER02_TOURB_CHRM
AWBF2180	watch	man	2018	STANDARD	TITANIUM	WITHOUT BRACELET	CALIBRE_5_AUTO
AWBF2A80	watch	man	2017	STANDARD	TITANIUM	WITHOUT BRACELET	CALIBRE_5_AUTO
AWBF2A81	watch	man	2017	STANDARD	TITANIUM	WITHOUT BRACELET	CALIBRE_5_AUTO
CAC1110.BA0850	chrono	man	2003	STANDARD	STEEL	STEEL	MVT_QUARTZ
CAC1110.BT0705	chrono	man	2004	STANDARD	STEEL	RUBBER	MVT_QUARTZ
CAC1111.BA0850	chrono	man	2003	STANDARD	STEEL	STEEL	MVT_QUARTZ
CAC1111.BT0705	chrono	man	2004	STANDARD	STEEL	RUBBER	MVT_QUARTZ
CAC1112.BA0850	chrono	man	2005	STANDARD	STEEL	STEEL	MVT_QUARTZ
CAC1112.BT0705	chrono	man	2005	STANDARD	STEEL	RUBBER	MVT_QUARTZ
CAC1113.BA0850	chrono	man	2005	STANDARD	STEEL	STEEL	MVT_QUARTZ
CAC111A.BA0850	chrono	man	2004	STANDARD	STEEL	STEEL	MVT_QUARTZ
CAC111B.BA0850	chrono	man	2005	STANDARD	STEEL	STEEL	MVT_QUARTZ
CAC111D.BA0850	chrono	man	2005	STANDARD	STEEL	STEEL	MVT_QUARTZ
CAC111D.BT0705	chrono	man	2005	STANDARD	STEEL	RUBBER	MVT_QUARTZ
CAC1310.BA0852	chrono	woman	2007	MOTHER OF PEARL	STEEL	STEEL	MVT_QUARTZ
CAC1310.FC6219	chrono	woman	2007	MOTHER OF PEARL	STEEL	NIZZA	MVT_QUARTZ
CAC1311.BA0852	chrono	woman	2007	MOTHER OF PEARL	STEEL	STEEL	MVT_QUARTZ
CAC1311.FC6220	chrono	woman	2007	MOTHER OF PEARL	STEEL	NIZZA	MVT_QUARTZ
CAD5110.FC6177	chrono	man	2004	STANDARD	STEEL	LEATHER ALLIGATOR	CALIBRE_36
CAF1010.BA0821	chrono	man	2007	STANDARD	STEEL	STEEL	MVT_QUARTZ
CAF1010.FT8011	chrono	man	2007	STANDARD	STEEL	RUBBER	MVT_QUARTZ

SQL Schema

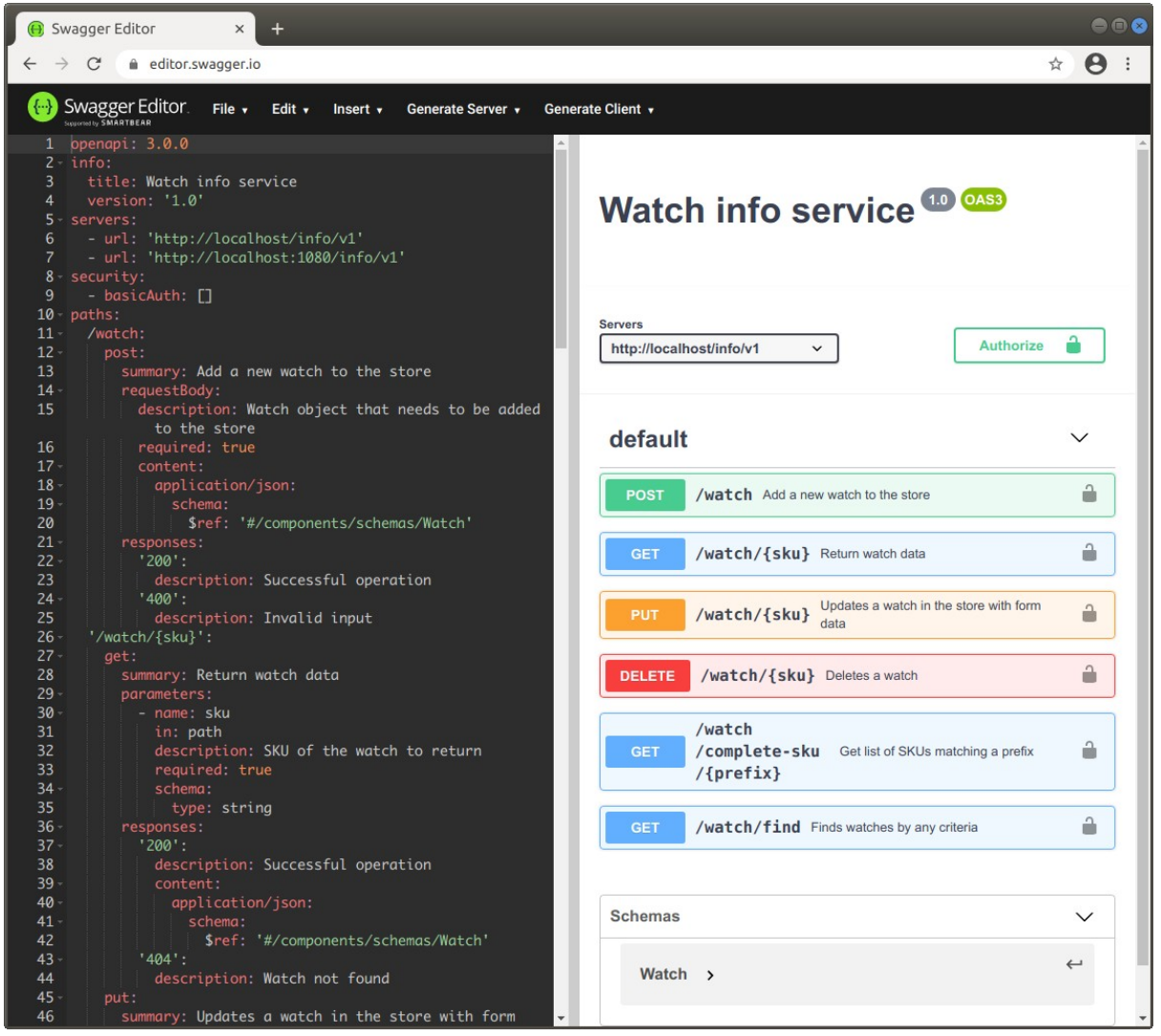
- Data directly maps to API spec
  - SKU is a unique identifier (primary key)

Colonne	Type
sku	varchar(255)
type	enum('watch','chrono')
gender	enum('man','woman')
year	int
dial_material	varchar(255)
case_material	varchar(255)
bracelet_material	varchar(255)
movement	varchar(255)

Swagger

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

OPENAPI INITIATIVE



REST API

- Main components
  - **Web server**: route the **HTTP** requests/replies to the backend app
  - **Backend app**: do the processing
    - Convert from/to **JSON**
  - **ORM**: library that map objects into SQL queries
- The webserver and the backend app can be completely independant...
  - Typical example: Apache + PHP
- ... or integrated
  - Typical example: Flask + Python

REST API #2

- For the project, we recommend using Python with Flask and SQLAlchemy (ORM)
  - Explanations on following slides consider this solution is used
- Other languages/servers/ORMs are accepted
  - A container can contain any language but when deploying «serverless», only a limited set of languages are supported:
    - https://www.techtarget.com/searchcloudcomputing/tip/Compare-AWS-Lambda-vs-Azure-Functions-vs-Google-Cloud-Functions

Tutorials

- Flask
  - https://flask.palletsprojects.com/en/2.2.x/quickstart/
- SQLAlchemy
  - https://docs.sqlalchemy.org/en/14/tutorial/index.html
- Docker
  - https://docs.docker.com/get-started/
  - https://docs.docker.com/language/python/build-images/
- https://www.youtube.com/watch?v=WFzRy8KVcrM

Be familiar with these apps before attempting doing the project!

Step by Step

- The next slides explain a step by step strategy to develop the API on Linux Debian/Ubuntu
  - First develop the app completely outside containers
  - Once it works, put (only) the webservice inside a container and connect it with the DB
  - Finally, also deploy the DB in a container and deploy them together

Step #1 - Local install

- Install MySQL
  - `$ sudo apt install mysql-server mysql-client`
  - 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 python3-dev libmysqlclient-dev`
    - Depending of the distribution replace libmysqlclient-dev by libmariadb-dev

Step #2 - Local dev

- Develop your API in Python
  - Use pip3 for dependencies management
    - https://pip.pypa.io/en/stable/user_guide/
    - List dependencies in: **`requirements.txt`**
      - Flask, MySQL, SQLAlchemy, ...
  - **`app.py`**
    - Single file webservice
  - `flask_run.sh`
    - `flask --debug run`
    - Listen on port 5000 by default
    - Debug: hot reloading and infos during dev

Step #3 - ENV vars

- Using ENV vars, the webservice container image can remain generic and be used both in a local and cloud deployment without any changes
  - `DB_HOST=127.0.0.1`
  - `DB_PORT=3306`
  - `DB_DBNAME=watches`
  - `DB_USER=watches`
  - `DB_PASS=watches`

These ENV vars are already set in the scripts provided

Step #4 - Validate API

- Using Swagger
  - https://editor.swagger.io/
  - Load info_openapi_v1.yaml
  - **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 - Create Info-service Image

- Create a Dockerfile
  - Embed your Python app inside a Docker image
    - https://docs.docker.com/language/python/build-images/
- `build.sh`
  - `docker build -t` **`info-service-v1`** `.`
- `docker_run.sh`
  - Run with --network=host

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

Step #6 - MySQL in Docker

- Use https://hub.docker.com/_/mysql/
  - Read the documentation !
  - Seed the DB
    - See section «Initializing a fresh instance»
  - To fix authentication problems:
    - `command: --default-authentication-plugin=mysql_native_password`

Step #7 - Compose

- `stack.yml`
  - Use ENV vars
  - Run images
    - `info-service-v1`
    - `mysql`
    - `PHPMyAdmin/Adminer` (optionally to view database)
  - Connect them together
  - Persist DB data
- `up.sh`
  - The API should be up and running !

Deliverables

- Python server
  - `app.py`
  - `requirements.txt`
- Docker / Docker compose
  - `Dockerfile`
  - `stack.yml`
- Additional files as you need (DB init, ...)
- The scripts provided should execute properly
  - `build.sh` ⇒ `info-service-v1` image
  - `up.sh` ⇒ `http://localhost:1080/`

Tag 'part1' in the git repo once it is complete

Part II - Sneak Preview

- Cloud (GCloud or AWS) deployment of the API
  - Deploy container(s) in the Cloud
    - Or Functions as a Service
  - Managed DB
  - Add scalability (multiples instances, load-balancers, ...)
  - Automate the deployment
- Learn new technologies
  - Read documentation, follow tutorials, pratice using free-tier
- Collaborative work
  - Split tasks, share ideas and experience with other team members

Planning

- Part I (3 weeks)
  - Documentation: TODAY (week #3)
  - **Deadline**: 2022-11-03T23:59:59+02:00 (4 weeks)
- Part II (5 weeks)
  - Documentation: 2022-10-27 (week #6)
  - **Deadline**: 2022-12-01T23:59:59+02:00
- Report about Part II (1 week)
  - Architecture diagram
  - Discussion about your technical choices, failures, performances, maintainability, limitations, future improvement, costs, ...
- Evaluation demo
  - Show/explain me how you deploy your service on the cloud

Next Steps (until next week)

- Create a private team repository
  - Github or Gitlab
- Get `project-part1-scaffolding.tar.gz` on Moodle
  - Put the extracted files at the root of your repo
  - Set name of participants in README
  - Do the initial commit
- Grant write access to all team members
- Send me the repo URL
  - TODO Google doc link in the teams doc
- Grant me a read access
  - Username 'lleonini'