TESTING

Alessandro Bocci

name.surname@unipi.it

Advanced Software Engineering (Lab)

01/12/2023

What will you do?

- Write unit tests to find bugs.
- Write performance tests to find bottlenecks.
- Remove bugs and bottlenecks.

Software Prerequisites

- Pytest (install from pip).
- Locust (install from pip).
- Docker image for `microase2324` `(python:3.9.18-slim)`
- `microase2324` folder (from Moodle).

pytest

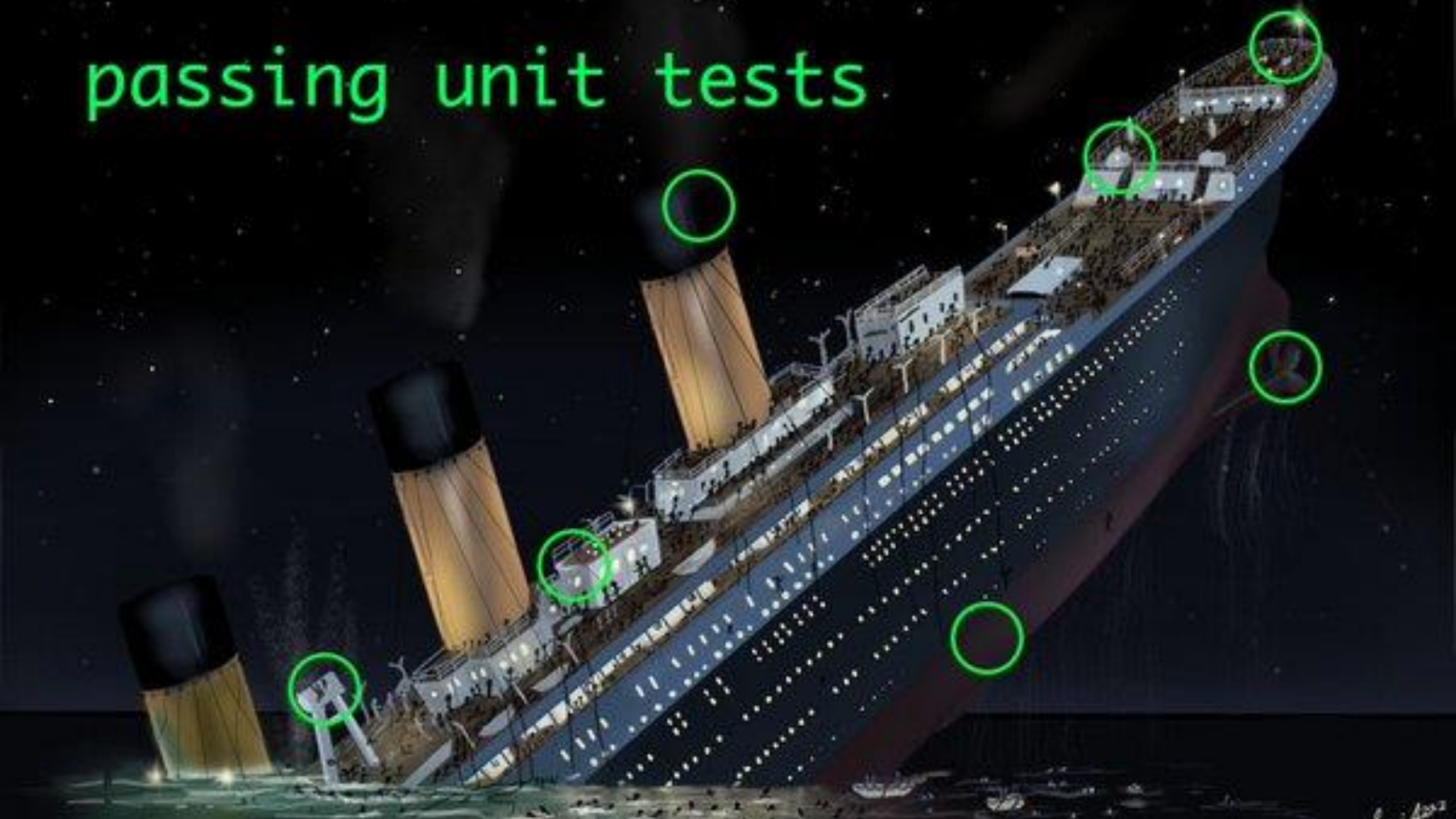
Testing micro-services

- **Functional testing**: Test the functionality of the whole system (unit -> feature -> system -> release testing)
- **User testing**: Test usability by end-users.
- **Performance testing**: Measure the microservice performances against varying workload
- **Security testing**: Still remember bandit and OWASP Zap?

Unit tests 101

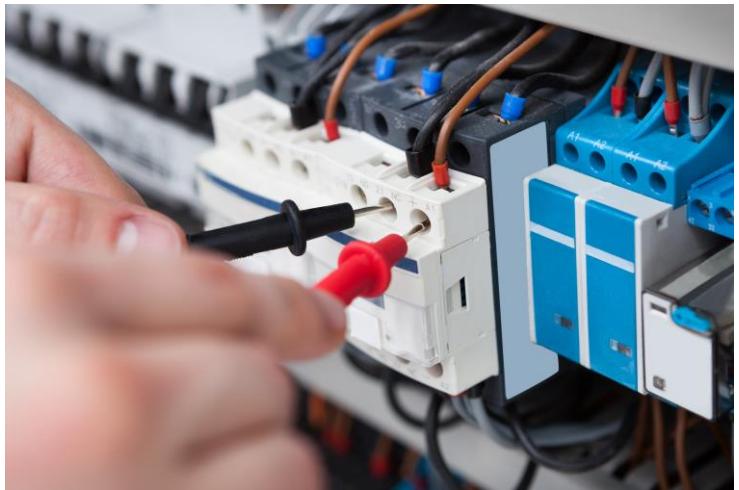
- In Flask projects, there usually are some functions and classes, which can be **unit-tested in isolation**.
- In Python, calls to a class are *mocked* to achieve isolation.

Pattern: Instantiate a class or call a function and verify that you get the expected results.


passing unit tests

Unit tests (cont'd)

- Functional tests for a microservice project are all the tests that interact with the **published API** by sending HTTP requests and asserting the HTTP responses.
- Important to test:
  - **that the application does what it is built for,**
  - **that a defect that was fixed is not happening anymore**.



Pattern: Create an instance of the component in a test class and interact with it by mock (or actual) network calls.

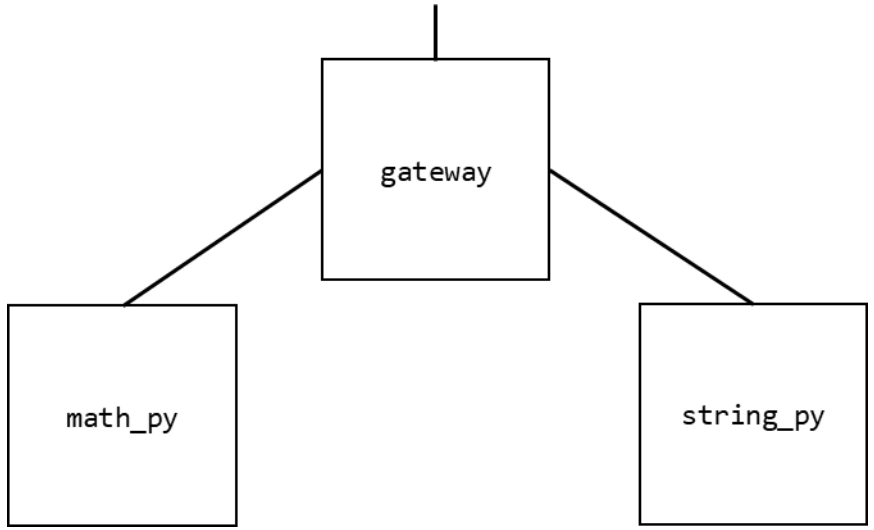
Today's Lab

PART ONE

1. Write unit tests for all the operations of **`math`** and **`string`** services.
2. Run the tests with **`pytest`**.
3. Spot the bugs and resolve them (one for service).

PART TWO

1. Write performance tests for 3 endpoints for both **`math`** and **`string`** services.
2. Run **`microase2324`** (with docker compose).
3. Run the tests with **`locust`**.
4. Spot the bottlenecks and resolve them.



Unit testing with `pytest`

```
pip install pytest
```

1. Download **`microase2324.zip`**
2. Import in Swagger Editor (https://editor.swagger.io/) the API of **`math`**.
3. Add tests in the **`u_test.py`** (subfolder **`tests`**) for all the operations of the API (check more than one value and limits when needed, e.g. division by 0).
4. From the **`microase2324`** folder, run the test with

```
python3 -m pytest math_py/tests/ -v
```

5. Find and resolve the bug and repeat for the **`string`** service.

Note:

- **`conftest.py`** configures the mock microservice, do not touch it.

pytest 101

- `pytest` launches all `test*` files inside the `tests` folder.
  It performs test discovery by file name.
- A useful extension to evaluate test coverage is:

```
pip install pytest-cov
```

Add argument `--cov` when running tests

```
---------- coverage: platform win32, python 3.7.0-final-0 ----------
Name                                Stmts   Miss  Cover
-------------------------------------------------------
myservice\__init__.py                   1      0   100%
myservice\app.py                       12      1    92%
myservice\classes\__init__.py           0      0   100%
myservice\classes\poll.py              47      2    96%
myservice\tests\__init__.py             0      0   100%
myservice\tests\_test_int.py           32     32     0%
myservice\tests\test_home.py          112      0   100%
myservice\tests\test_poll.py           12      1    92%
myservice\views\__init__.py             2      0   100%
myservice\views\doodles.py             59      0   100%
-------------------------------------------------------
TOTAL                                 277     36    87%
```

Load Test

- The goal of a load test is to understand your service's bottlenecks under stress.
- Understanding your system limits will help you determining how you want to deploy it and if its design is future-proof in case the load increases.
- Shoot at it!

Pattern: Create an instance of the component and stress test it by mocking different amount of workload.

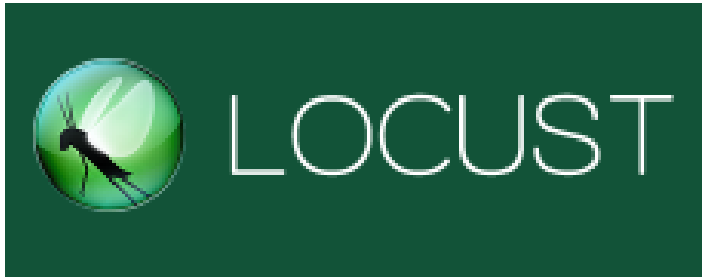
Locust

```
pip install locust
```

An open-source load testing tool used by Big Companies.

- It is needed a **`locustfile.py`** in your root project folder to define users' behaviours (you have one with a first test example).
- Write tests for the **`gateway`** (you should write 3 tests for **`math`** and 3 for **`string`**)
- Start **`microase2324`**

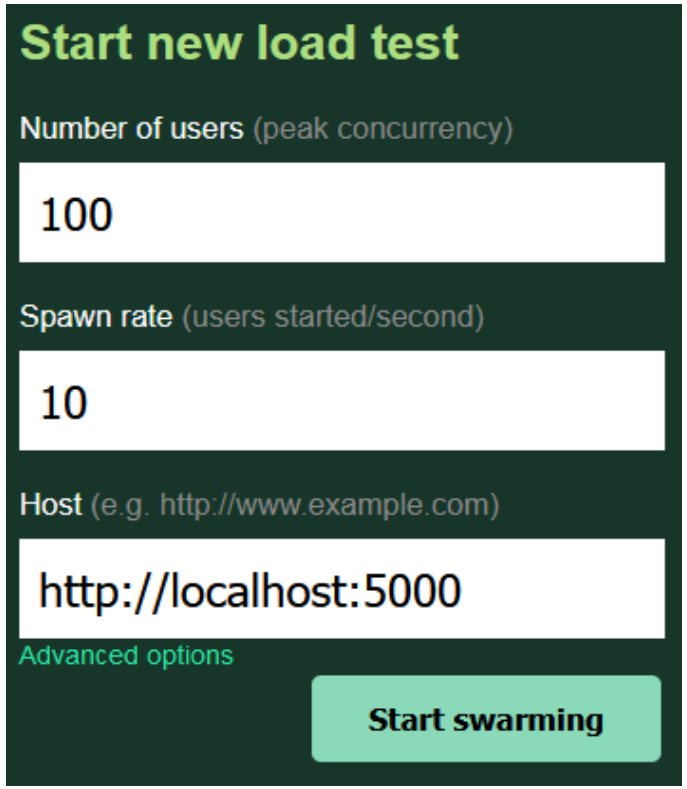
```
docker compose build
docker compose up
```

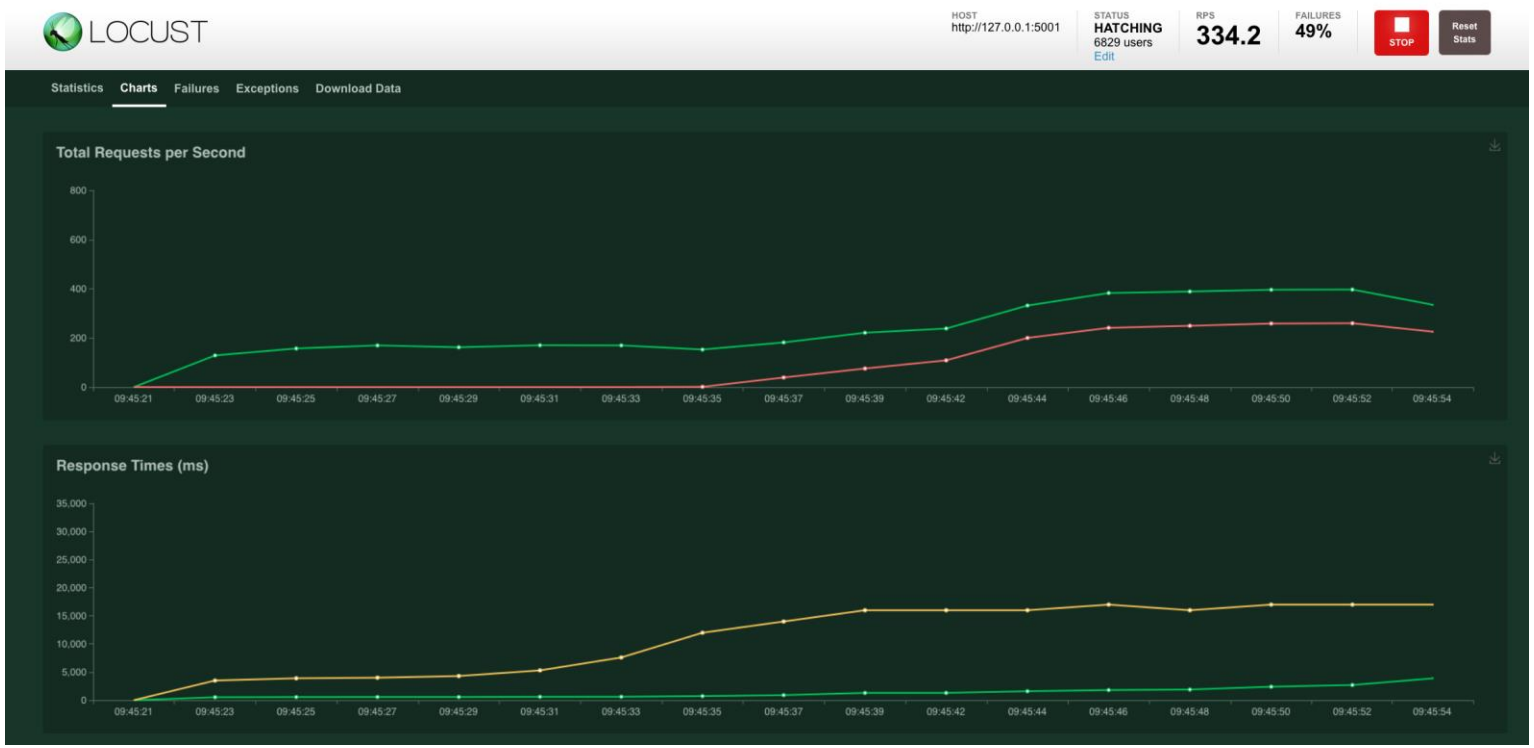


- Run locust: `locust` in the root folder (new terminal).
- Browse to http://localhost:8089, set up and run your tests!

Stress your microservice!

http://localhost:8089





Look for the bottleneck

- Analyse locust's stats and graphs.
- Spot the bottlenecks endpoint, if any.
- Check the code of the endpoints in the **`gateway`** to resolve the possible problem.

In general, if you have a service (not an endpoint) that performs poorly, you can scale it by means of:

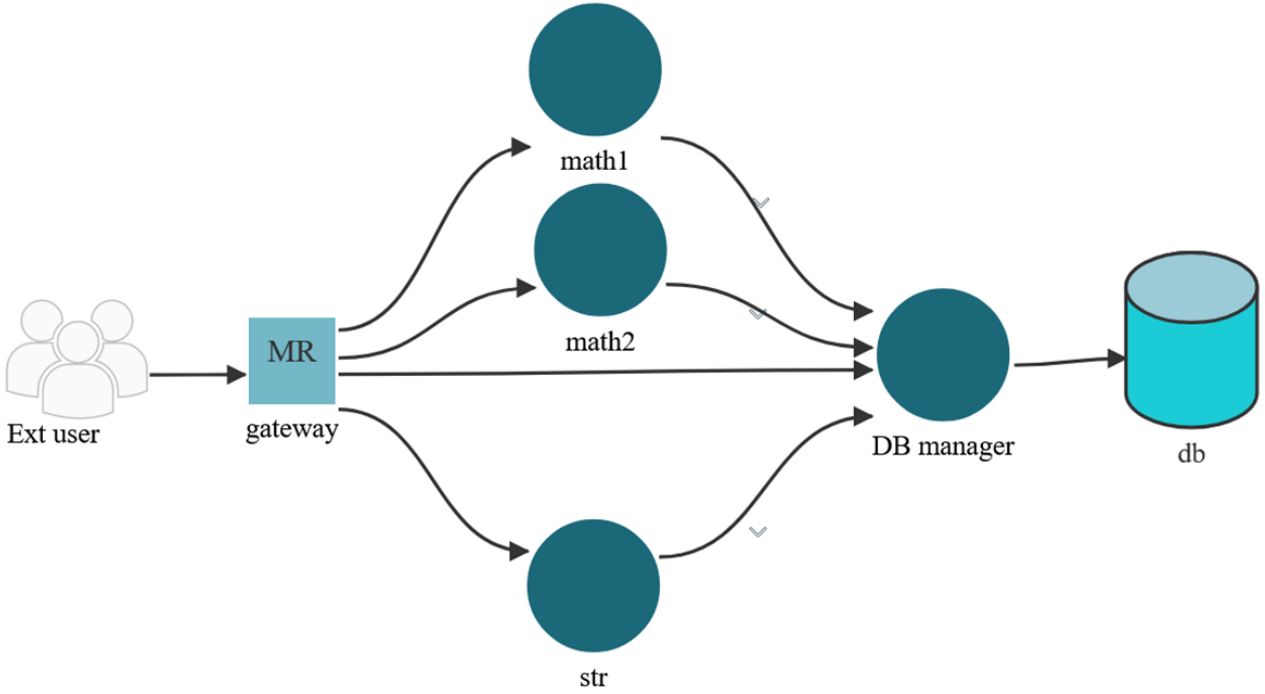
```
docker-compose up -d --scale service=6 --no-recreate
```

BONUS STAGE!

Bonus stage

Use the complete version of microase2324 and do performance tests with locust.

The full API of the gateway was given in the kubehound lab.



You should find at least another non-artificial bottleneck.

Try to resolve it!

Lab take away

- ❑ Write and run unit tests to discover bugs in microservices.
- ❑ Write and run performance tests to discover bottlenecks microservice applications.