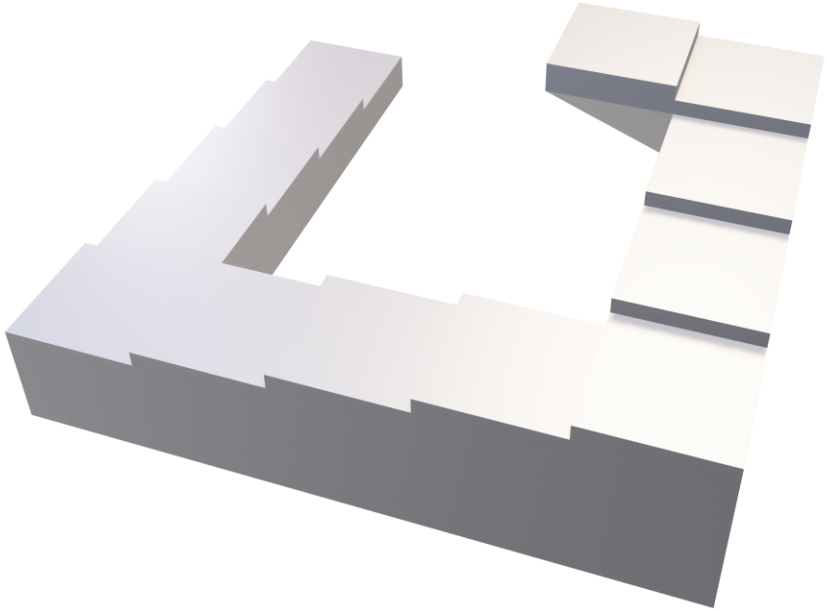
Smells and Refactoring

Alessandro Bocci
name.surname@phd.unipi.it

Advanced Software Engineering (Lab)
09/11/2023

What will you do?

- Experiment with μFreshener to identify smells and refactorings
- Understand the problems that may raise from architectural smells
- Fix smells in a microservice-based application by writing code

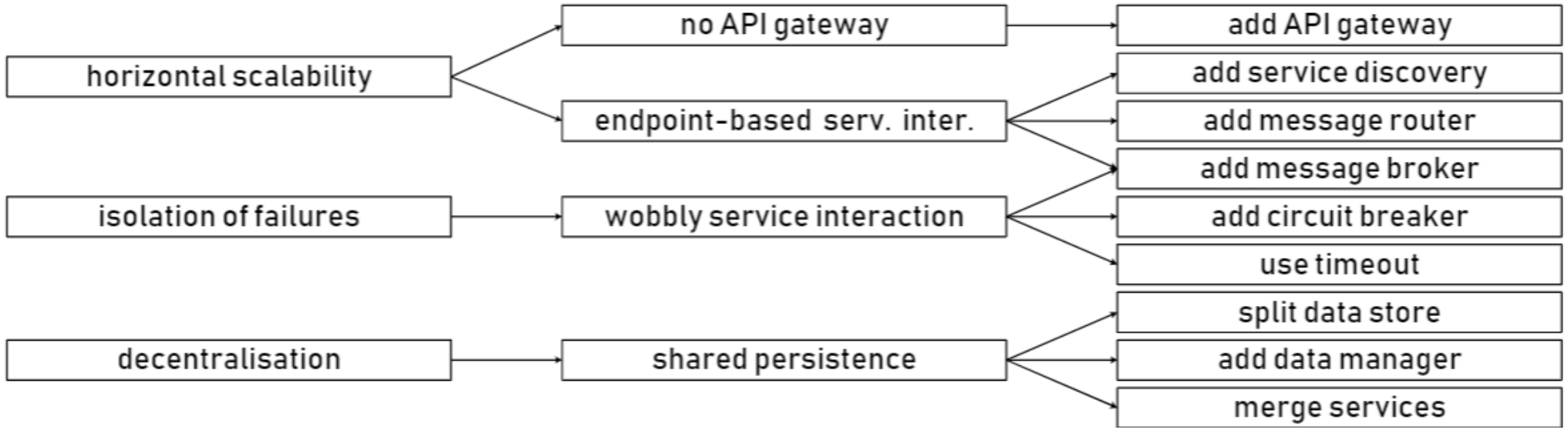


Software Prerequisites

- μFreshener:
  1. Download it from the Moodle
  2. `docker compose up --build`
  3. try it @127.0.0.1:8080
- Docker images (`docker pull <image>`):
  - `python:3.9.18-slim`
  - `redis:6.2-alpine`

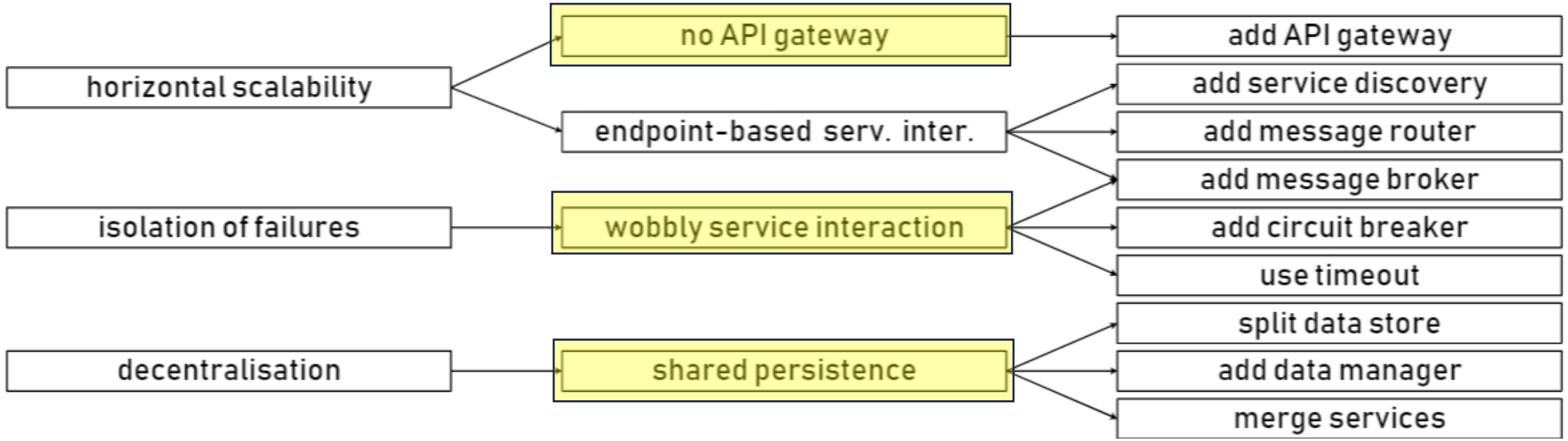
Lab Goal

Learn how to identify **architectural smells** and implement the corresponding **refactorings** in the `microase2324` application.



Lab Goal

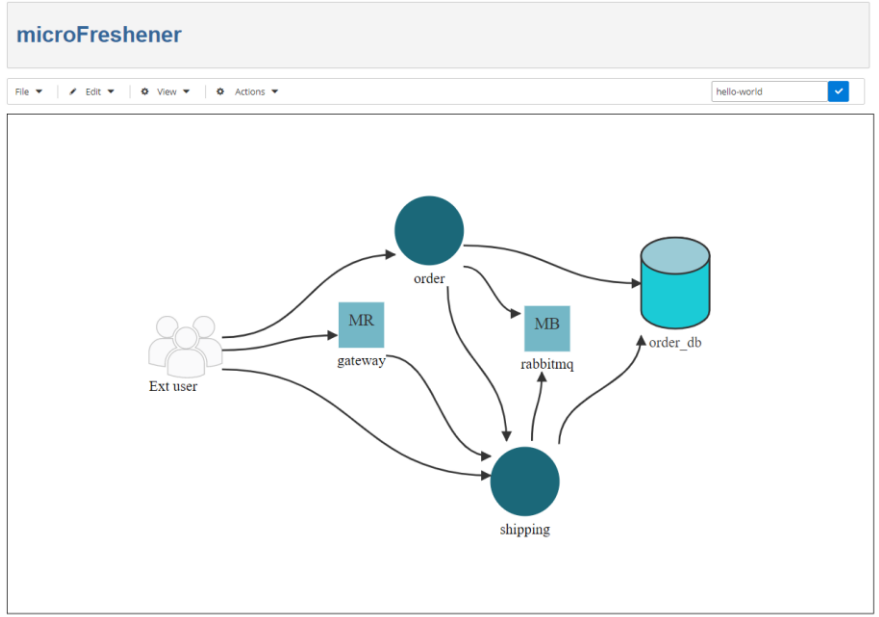
Learn how to identify **architectural smells** and implement the corresponding **refactorings** in the `microase2324` application.



μFreshener

Tool to draw microservice architectures, able to

- Analyse the architecture graph to identify smells
- Suggest refactoring to resolve smells
- Apply the selected refactoring to the architecture



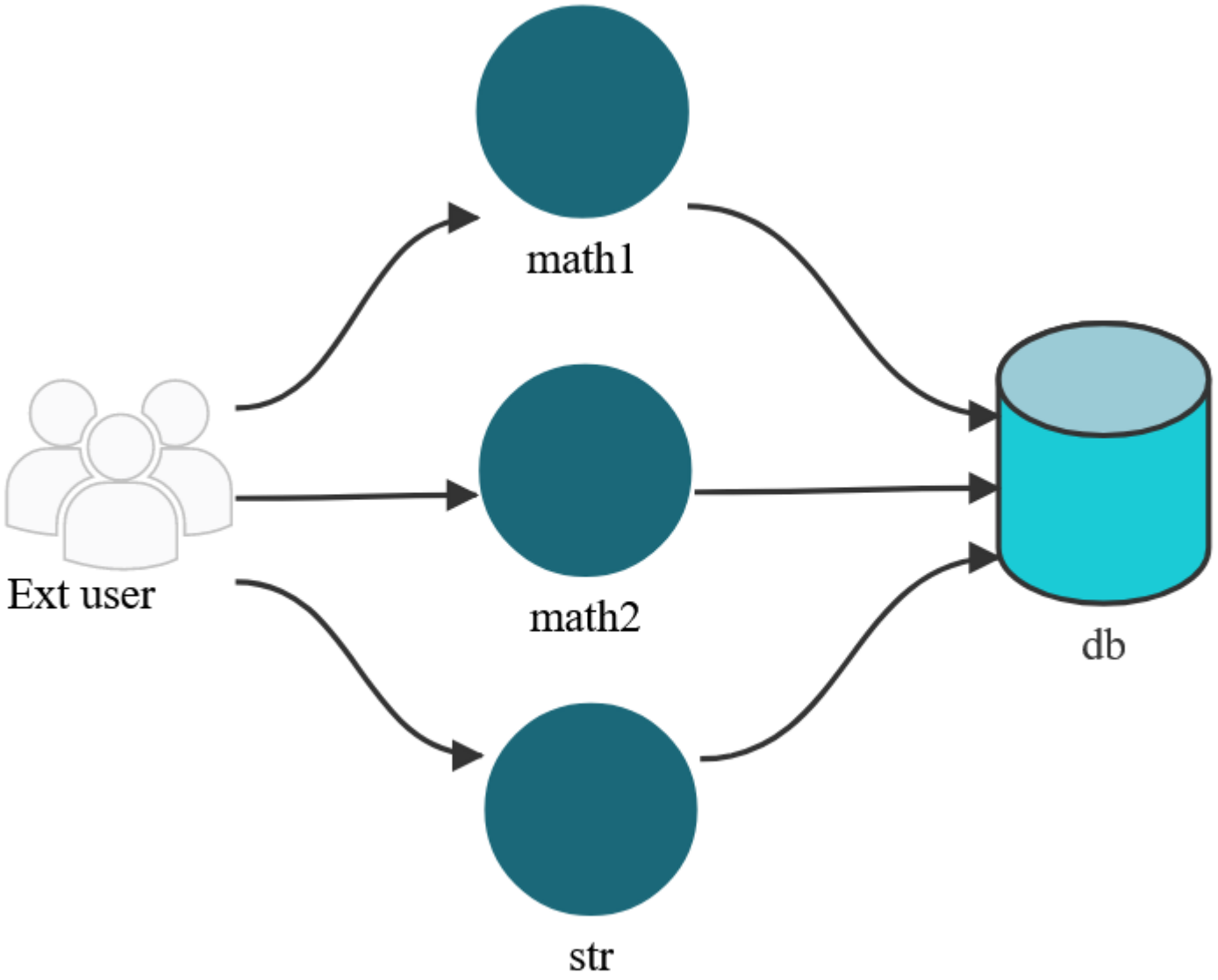

μFreshener
no API gateway
shared persistence
wobbly service interaction
[adult swim]

First things first

1. Download the latest version of **`microase2324`** from the Moodle.

2. Use μFreshener to draw the basic architecture of **`microase2324`**
3. Use μFreshener to identify smells and possible refactorings.

microase2324 basic



Drawing cheat sheet

- Circles (basic ver services): Edit -> add -> node -> Service
- DB (basic ver service): Edit -> add -> node -> Datastore
- API Gateway: Edit -> add -> node -> Communication Pattern -> Message Router
- Links: click on 1st node -> it appears an orange box -> click on 2nd node

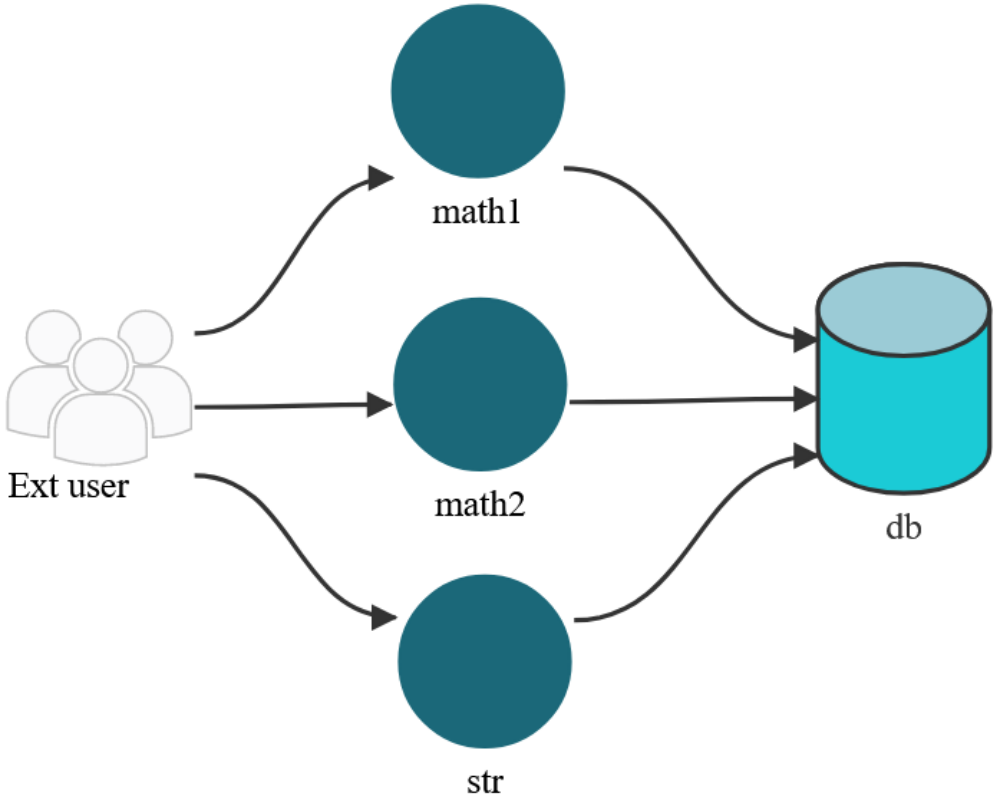
  (Ext user is the node representing

Sometime the selection of a node got stuck ☹

You have to refresh the page and restart from scratch.

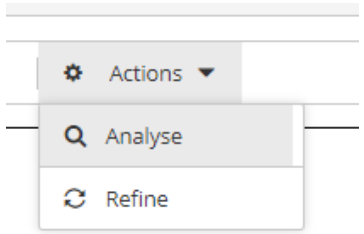
microase2324 basic

- The services **`math1`** and **`math2`** represent the horizontal scaling for the **`math`** service
- The **`db`** stores the logs of all the successful operations
- **`docker compose up --build`** starts the application (use a new terminal without stopping μFreshener)

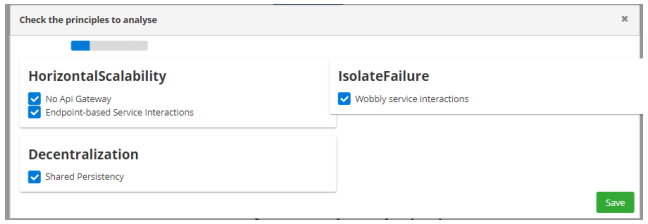


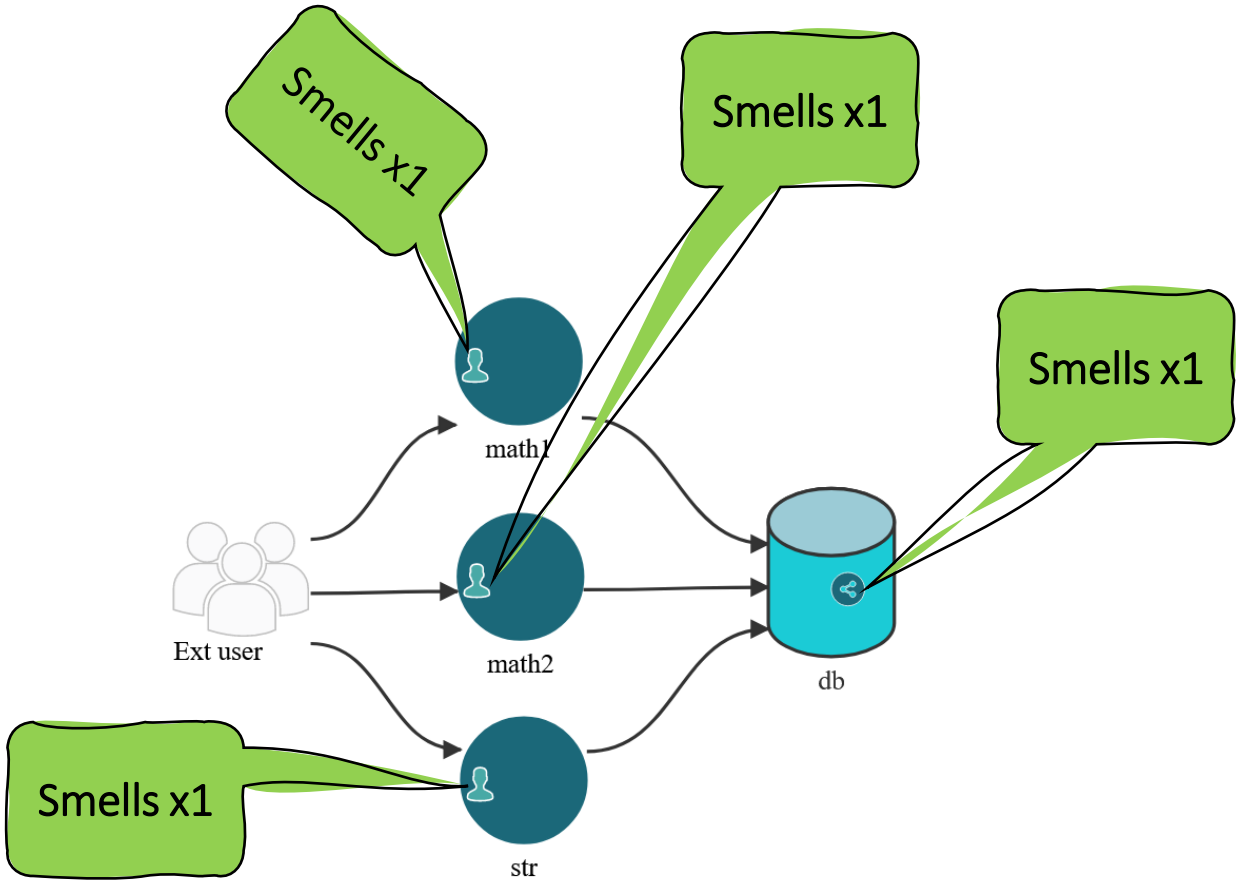
Smell the architecture

- Actions → Analyse



- Save





Every time you modify the architecture you have to smell it again

- Actions → Analyse → Save

Today’s Lab

You have to click on smells, understand the problems and resolve them:

PART ONE: No API Gateway

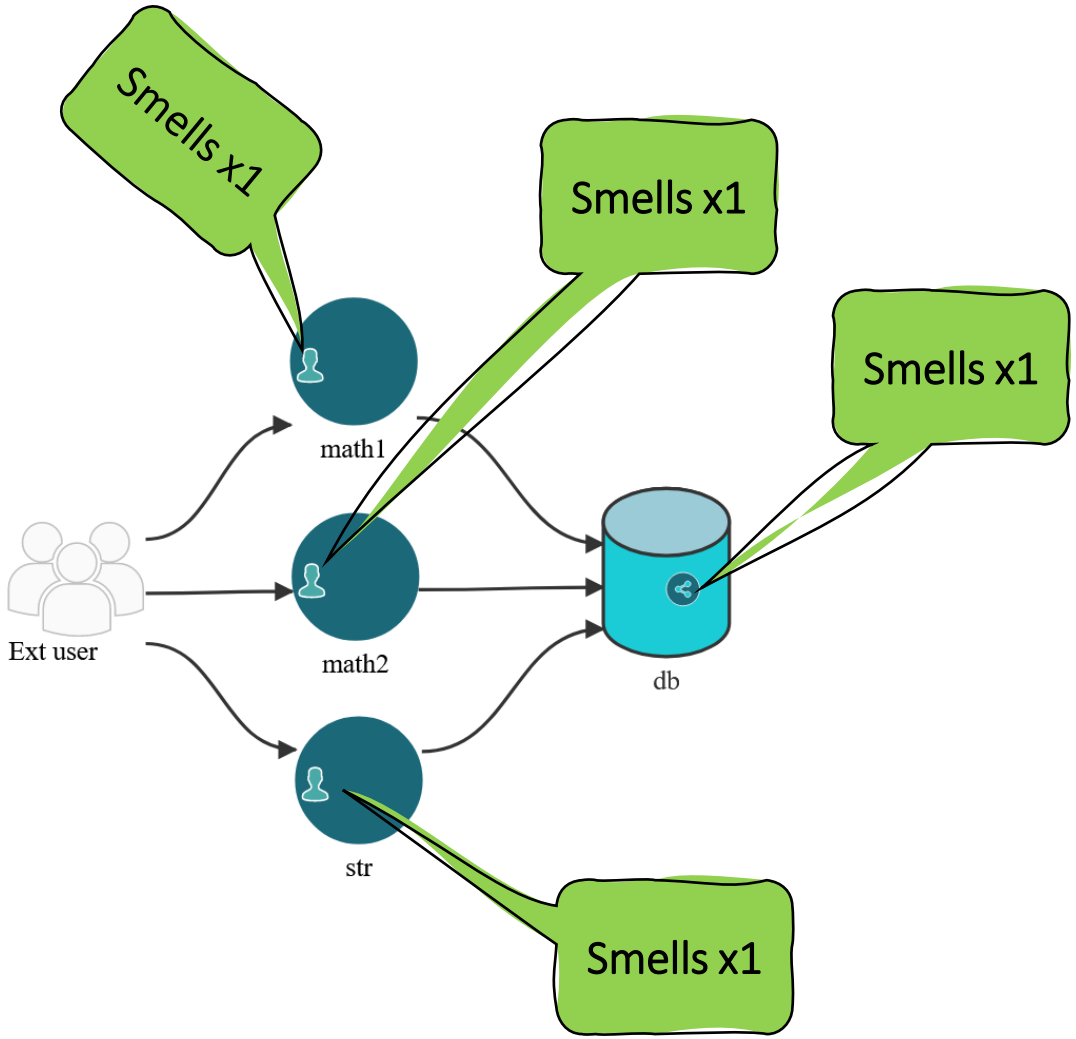
- Details from Slide 14

PART TWO: Shared persistance

- Details from Slide 17

PART THREE: Wobbly service interaction

- Details From Slide 19

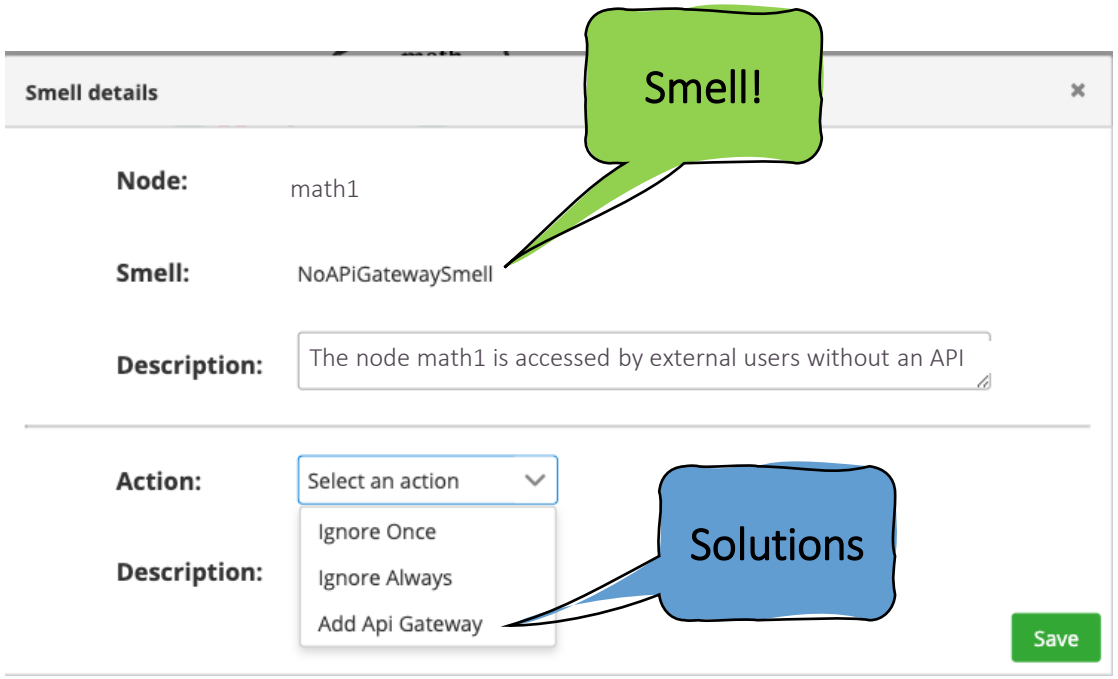


No API gateway - Problem

- Try a `math` service (e.g. `http:127.0.0.1:5001/add?a=6&b=3`)
- Crash it! (e.g. `http:127.0.0.1:5001/crash`)
- Try it again and…

No API Gateway

The **no API gateway** smell occurs whenever the external clients of an application directly interact with some internal services. If one of such services is scaled out, the horizontal scalability of microservices may get violated because external clients may keep invoking the same instance, without reaching any replica.



No API gateway - Resolution

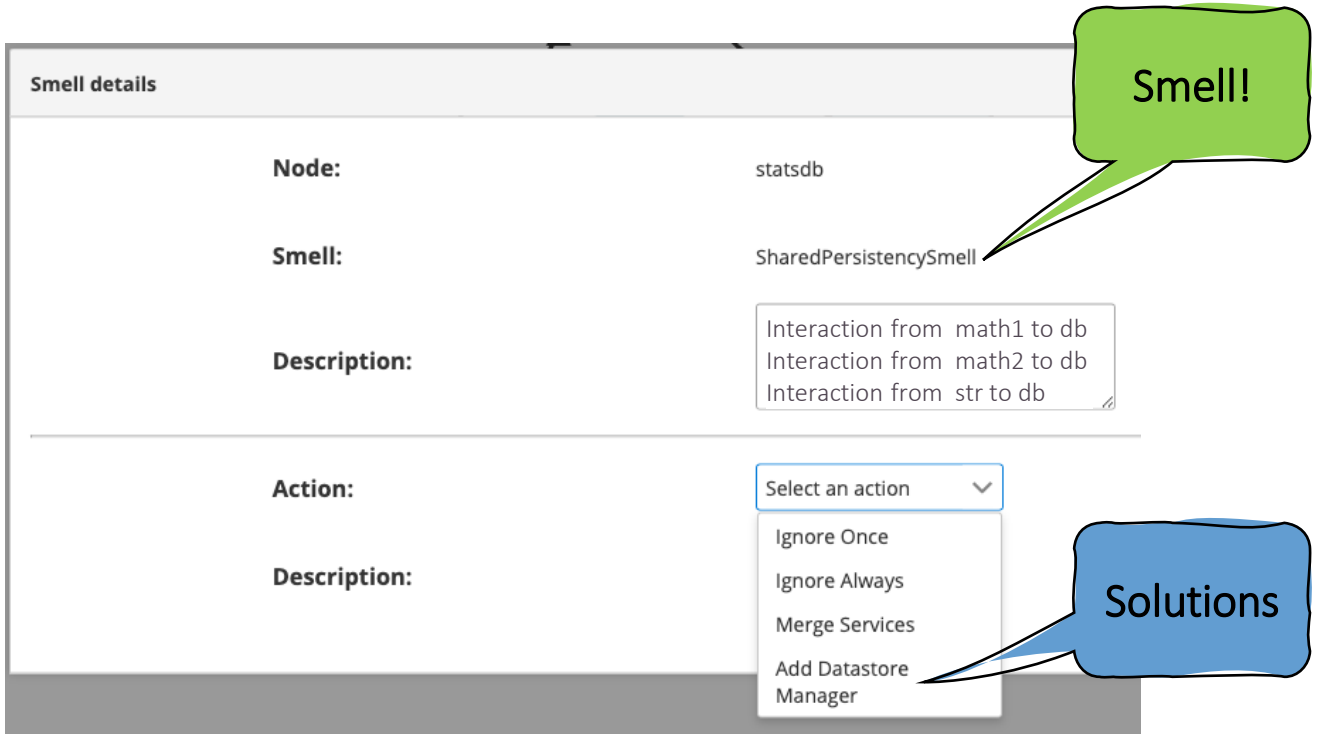
- Add the API gateway to the architecture on μFreshener

  This particular refactoring most of the time should be done by hand: remove the links from the Ext user, add the API gateway (as message router) and link it with the services.

- Stop docker composer (`docker compose down`)
- Change the `docker-compose.yml` adding the `gateway` and removing the direct access from the other services (`gateway` port should be `5000:5000`)
- `docker compose up --build`
- Try the `math` service (e.g. `http:127.0.0.1:5000/math/add?a=6&b=3`)
- Crash it (do only a single call to the crash endpoint)
- Try it again, and again, and again! (sometimes it could be slow but you should receive an answer)
- Smell the refactored architecture

Shared Persistency

The **shared persistency** smell occurs whenever multiple services access or manage the same DB, possibly violating the decentralisation design principle (i.e. business logic of an application should be fully decentralised and distributed among its microservices, each of which should own its own domain logic).



Shared Persistency - Resolution

- Add a Datastore manager to the architecture on μFreshener using the refactory solution (and link it also with the API gateway)
- Stop docker composer (`docker compose down`)
- Change the `docker-compose.yml` adding the `log-service` (this name exactly)
- Change the code of `gateway`, `math` and `string` services to contact the `log` service and not directly the `db`

  (You have to change <u>only</u> the `return` statement of the last function of each `app.py`)
- `docker compose up --build`
- Try some math or string operation, crash the log and then ask for the log (`http:127.0.0.1:5000/log/getLogs`).

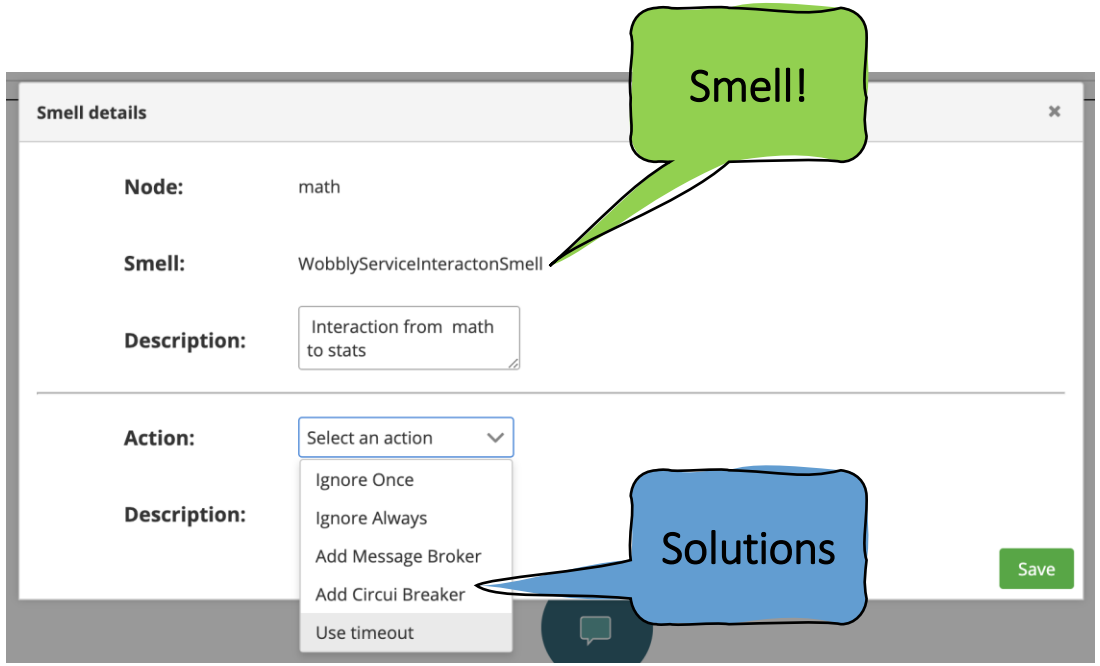
  You should receive the message «Log service is down»
- Smell the architecture again

Wobbly service interaction - Problem

- Crash the `log` service (e.g. `http:127.0.0.1:5000/log/crash`)
- Try a math or string operation and…

Wobbly service interaction

The interaction of a microservice m_i with another microservice m_f is **wobbly** when a failure in m_f can result in triggering a failure also in m_i. This typically happens when m_i is directly consuming one or more functionalities offered by m_f, and m_i is not provided with any solution for handling the possibility of m_f to fail and be unresponsive (which can lead to failure cascades).



You could implement easily two solutions:

- Use a timeout: Flask sometimes ignore it
- Add Circuit breaker: avoid to crash math or string services when the log service is down

Go for the second one

Wobbly service interaction - Resolution

- Add a Circuit Breakers to math and string services with μFreshener using the refactory solution (after every refactoring analyse again)
- Near each link should appear a ⌄ symbol
- Stop docker composer (`docker compose down`)
- Change the code of `math` and `string` services to avoid the crash when the `log` service is down

  (You should modify the code of `sendLogService(a,b,op,res,URL)` of each `app.py`)
- `docker compose up --build`
- Crash again the `log` service and try some math or string operation
- Now you should be able to perform the operations (slowly but you should get the answer). Obviously the operations will not be logged in the db
- Smell the architecture again and…

BONUS STAGE!

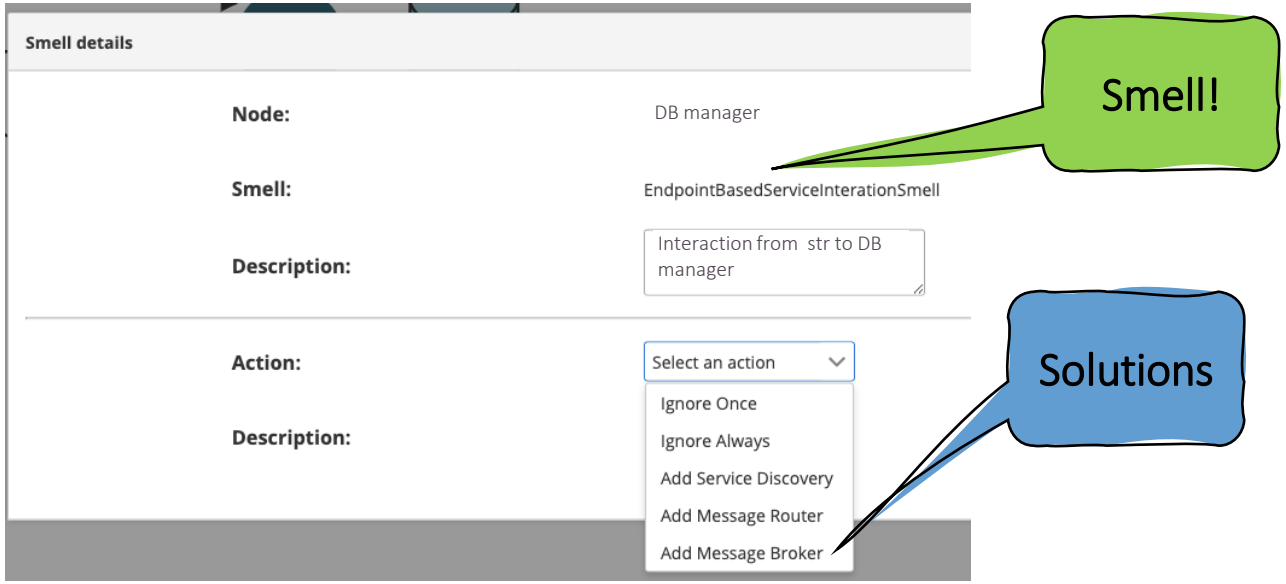

9
29
HiScore 090810580
Score 043130800
Player
Bomb
Power MAX
Graze 184
点 143
東方紅魔郷
THE EMBODIMENT OF SCARLET DEVIL
58.82fps

Bonus stage

It remains only one smell!

Endpoint-based interaction

The **endpoint-based interaction** smell occurs in an application when one or more of its microservices invoke a specific instance of another microservice (e.g., because its location is hardcoded in the source code of the microservices invoking it, or because no load balancer is used). If this is the case, when scaling out the latter microservice by adding new replicas, these cannot be reached by the invokers, hence only resulting in a waste of resources.



Bonus stage

Add a Message Broker between the DB manager (`log`) and `math` and `string` services

- First with μFreshener (and analyse to see no smells)
- Then, by adding a new service
  - Code it

    (we suggest using dockerised **RabbitMQ** and modify the services using python **pika**)
  - Add it to the docker compose file
  - Try it!

Lab take away

- ❑ Draw a microservice architecture and analyse it with μFreshener
- ❑ Learn which problems emerge from smells
- ❑ Learn how the architectural refactoring impact on the software

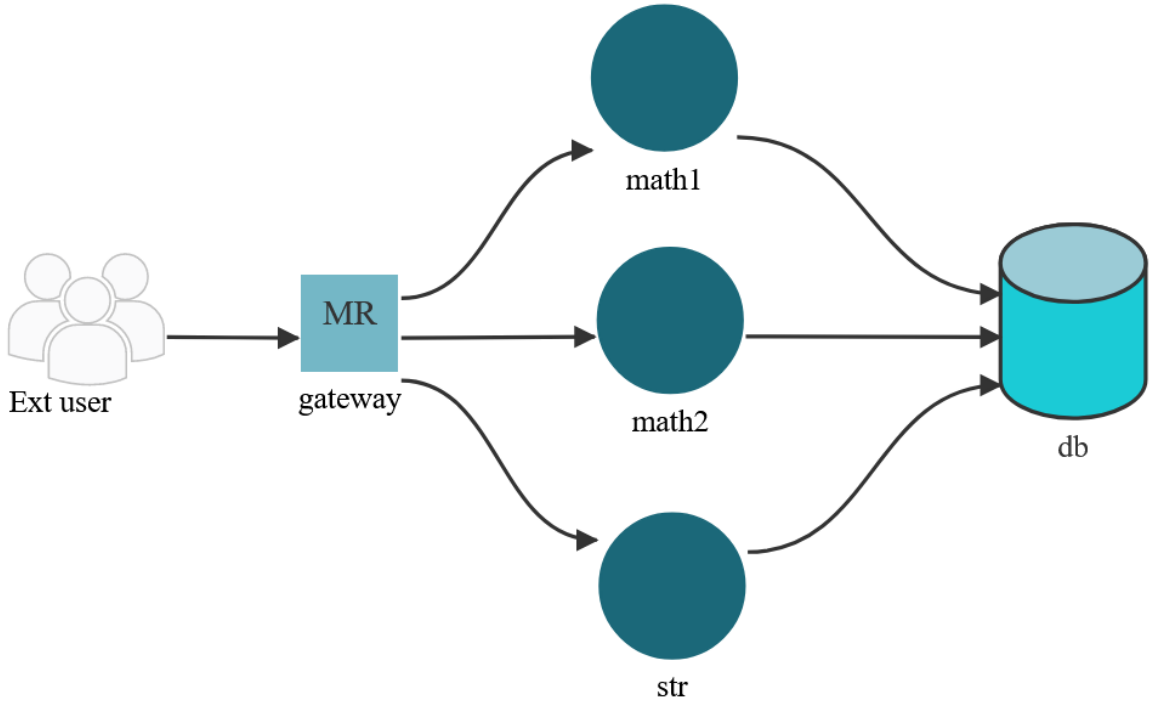
No API Gateway - Solution

In the `docker-compose.yml` file:

- Add as a service

```
gateway:
    build: ./gateway
    ports:
      - 5000:5000
```

- Remove `ports` of any other service



Shared persistency - Solution

- In the `docker-compose.yml` file:

Add as a service

```
log-service:
    build: ./log_py
```

- In math and service `app.py` put:

```
return sendLogService(a,b,op,res,URL)
```

instead of

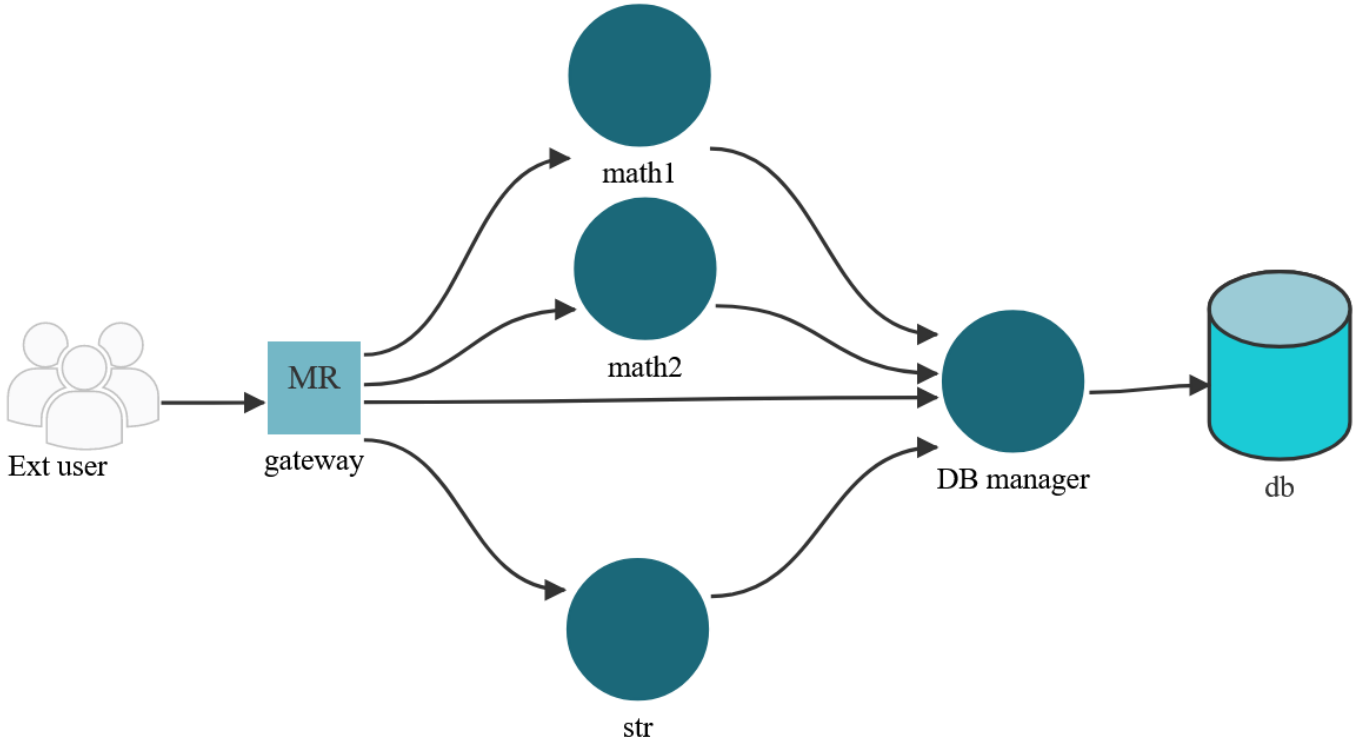
```
return sendLogDB(a,b,op,res,URL)
```

- In gateway's `app.py` put:

```
return getLogService(a,b,op,res,URL)
```

instead of

```
return getLogDB(a,b,op,res,URL)
```



Wobbly service interaction - Solution

In math and service `app.py` change the `sendLogService(a,b,op,res,URL)` function by putting all the code in a `try` statement and adding as exception:

```
except (ConnectionError, HTTPError):
        return
```