**Report**

**Introduction**

**Project title**: CovidAWS

**Groups members**: Ciammaruconi Daniele; Caparvi Giorgio

Description

Our project’s purpose is to create a cloud application that keeps track about people affected by covid virus.

We provide a website where a user can see data about people infected and healed by covid virus. An user can also label himself sick or cured with two buttons on the site.

The application relies on the use of Amazon AWS and the following services:

* AuroraDB
* S3 bucket
* Elastic Beanstalk

Deviations

Ci sono??? Altrimenti Rimuovo il paragrafo

**Implementation**

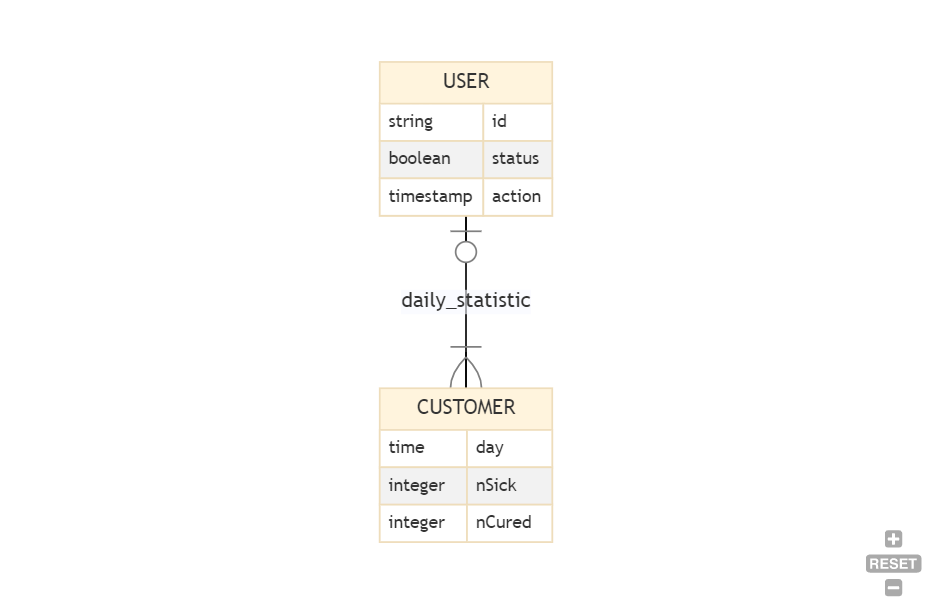
Analysing the service to be implemented, we decided to implement the following features:

* DB service
* Counter microservice
* Eureka microserver

[schema del servizio]

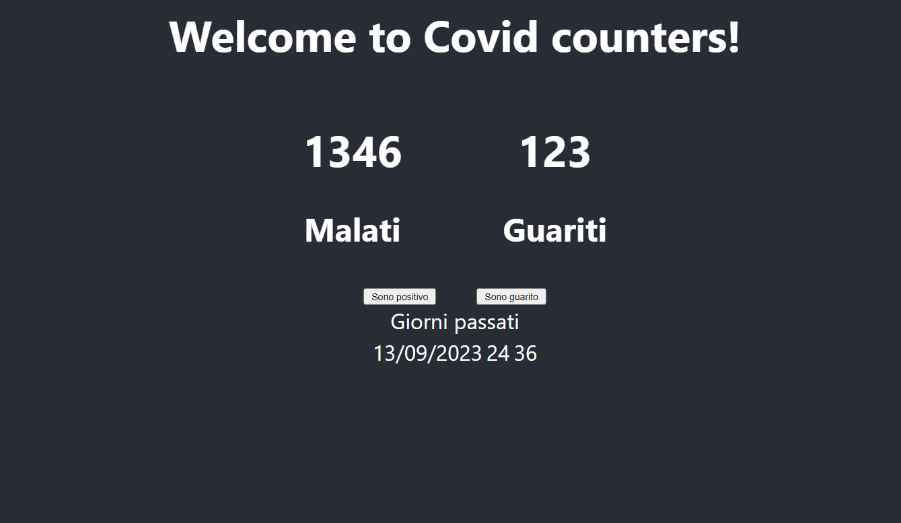
DB

As for the db, the following entities were analyzed: users and counters.

In the users query, all users who were sick or healed are recorded, along with the times when the operations were performed. To optimize the database, the id of each user is unique, so that a user is not allowed to add himself multiple times to the sick counter. In the counters query, on the other hand, the total daily number of sick and healed users is collected at 6 p.m. each day, and to optimize the service, once the counter is created, we remove all healed users from the query, as the system does not benefit from keeping records of healed users after defining that they are no longer sick. Below we can see the ER schema.

### Front

For the front-end part we decided to use react to make a simple interface: it allows you to see the counters and through the buttons below you can notify the service if you are positive or healed; to simplify input operations, the call that starts from the buttons contains only a random id. Each time the user refreshes the page, the services will be called to update the counts. Finally at the bottom you can see the counters of the past days.



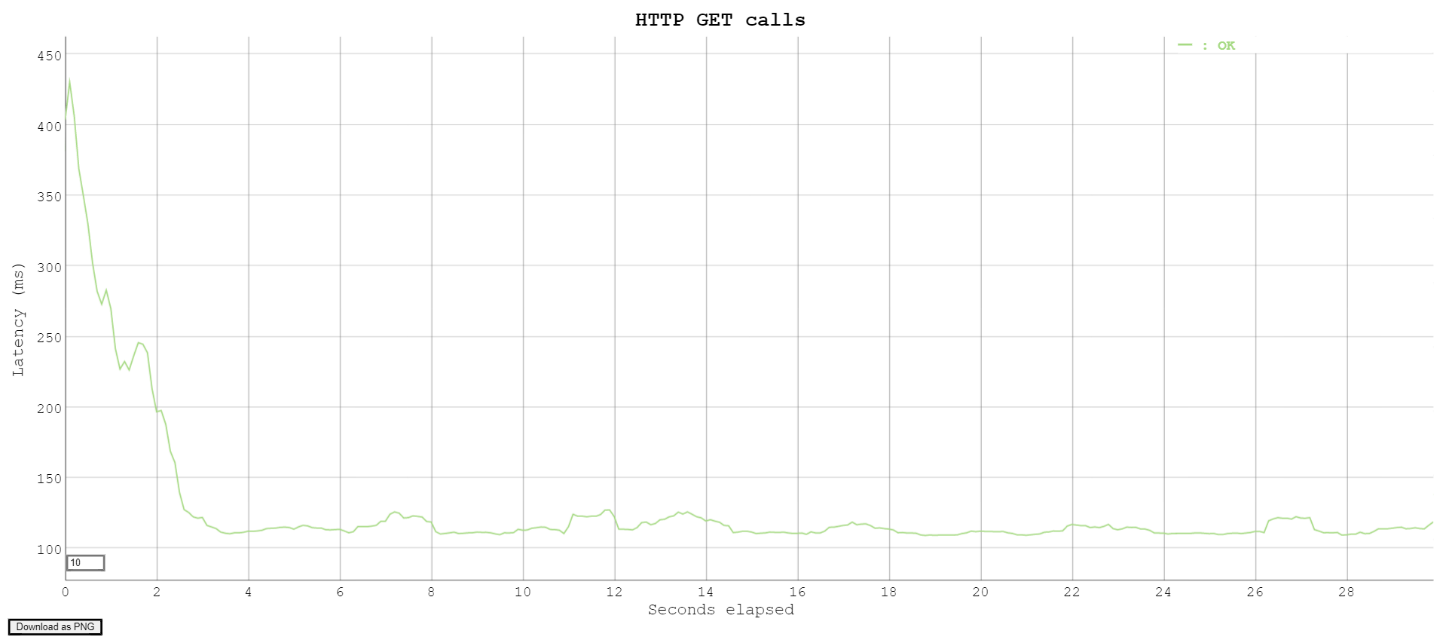
### Microservices

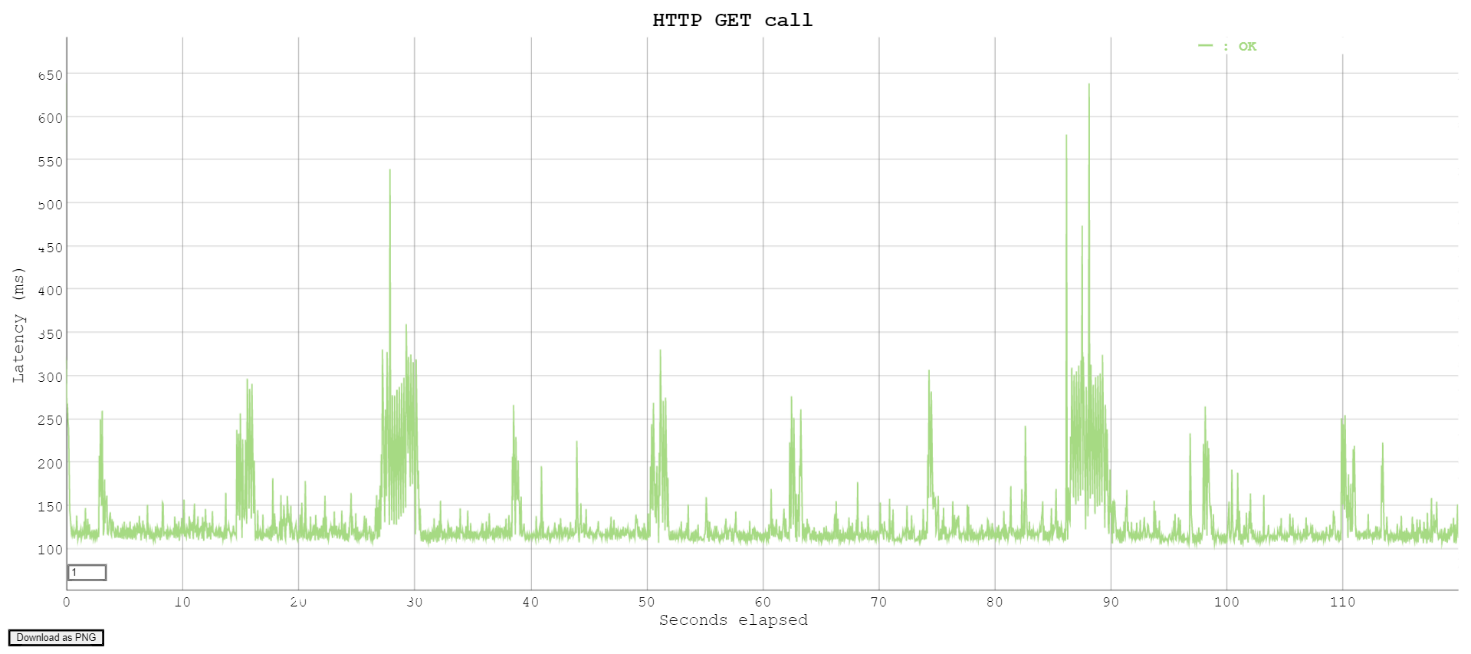
As for the microservice, we implemented it with Java Spring Boot. In the microservice, we have two get calls (to fetch the number of sick and healed users), two post calls (to send statistics) and a scheduled function to generate the daily counter. Thanks to the timestamp in the user query, we were able to correctly implement the function, thus fetching (and subsequently manipulating) all records that were correctly registered by 18:00

[scrivere qui anche la parte inerente allo scaling con Kubernetes]

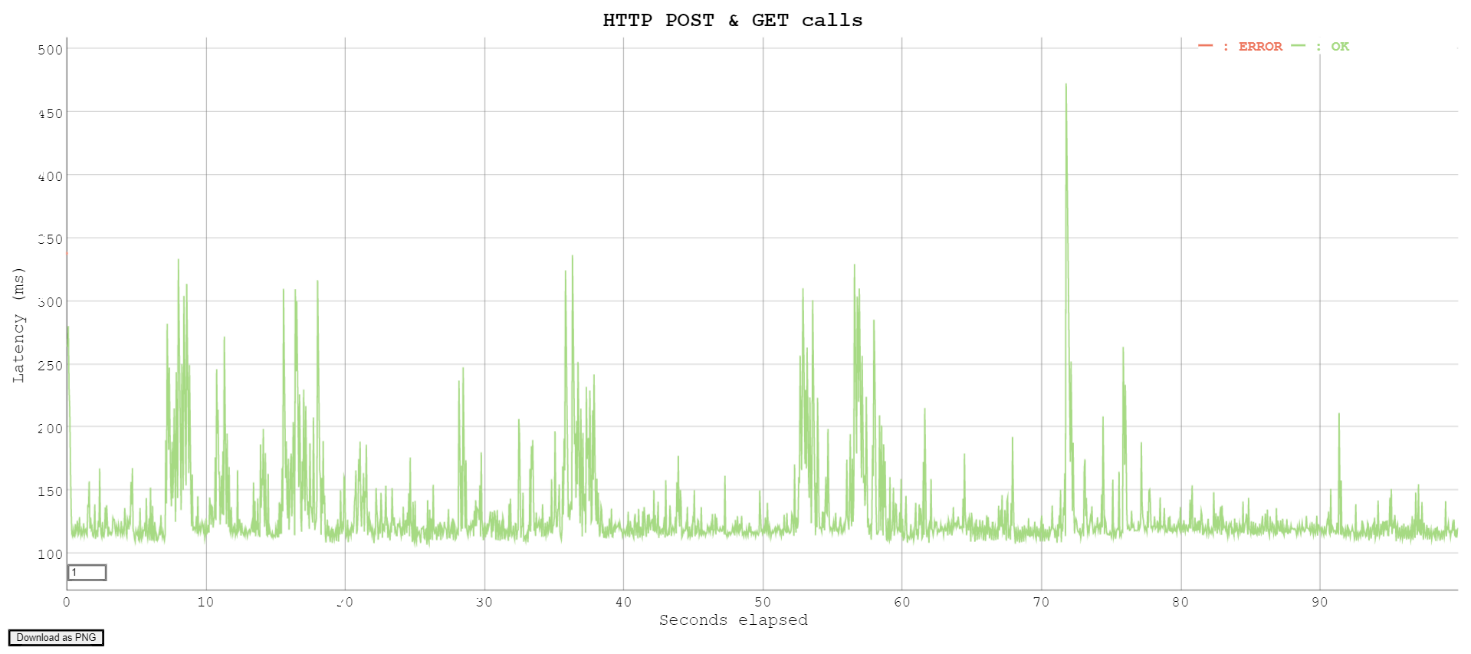
## Test & Validation

We wanted to test and verify the efficiency of our application with 3 types of tests (multiple reading of data, multiple writing of data, hybrid read/write) with different amounts of requests. These were the results:

In this first test we executed 400 GET requests in 10 seconds, so the machine received about 40 requests per second. As we can see in the graph, they were all executed successfully, with a very promising latency.



In this second test we wanted to test an extremely higher load of requests, trying to execute about 120 GET requests per second. This test was also successful, although the machine had a spike in requests and the workload was significantly higher than in the previous test, all requests were fulfilled but the average latency, compared to the first test, increased.



### Finally, in the third test, we tested a different load of requests, in fact we randomly alternated GET and POST requests, always keeping on a workload of about 20 requests per second. This test also succeeded correctly, the machine had a peak of requests, the workload was similar compared to the previous test, and all requests were fulfilled. The average latency this time remained similar to that of the second test

### Conclusion and future implementations

The advantages of using cloud technologies such as Docker and Kubernetes are obvious. Especially after having developed and tested all our services locally on our computers; first as simple programmes, then as Docker images and finally in a cluster. We have seen how it is easier for a developer to work without worrying in detail about the network or scalability in the software logic.

In addition, thanks to Elastic Beanstalk and scaling policies, we gained benefits such as self-healing and scalability. because any crashed pod is automatically replaced or replicated. We achieve high data availability thanks to the AWS platform and the AuroraDB service that hosts our database in the cloud. In the end, our cloud application works well enough, scales well enough not to drop requests and demonstrates how a simple application can serve thousands of requests simply by using cloud technologies.

In the future, we hypothesized that we could improve it by making it a true monitoring service that can handle requests from the population of a nation or country, improving performance based on the requests that will be made and implementing a better interface, thus allowing us to extrapolate anonymous statistical data that will allow us in the future to combat the diseases we encounter in a more targeted way.