Phase 9 - Testing and evaluation

**Group 2:**

Diogo Lança – fc53495

Meuri Canhanga – fc42559

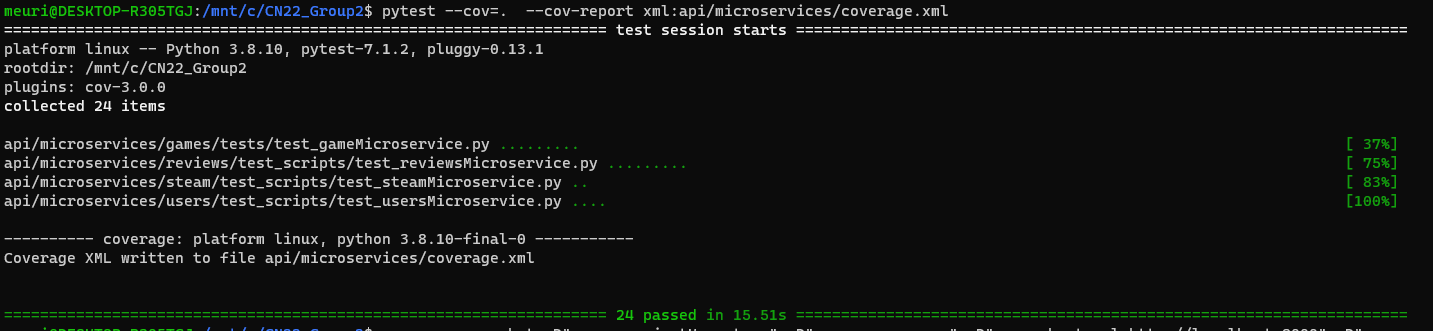
Miguel Silva – fc58974

Pedro Santos – fc46417

Renato Ribeiro – fc57433

**Operational support**

To confirm that the microservices were acting in accordance with the system requirements defined, we use the pytest module to develop unit tests for each. component of the "games", "users", "reviews" and "steam" microservices. We try to cover not only the positive cases, but also the cases in which invalid inputs were passed to the components. In total were defined 24 unit tests, and the result was:



Based on test results, we note that the unit tests defined for the games Microservice aren't enough, since the code coverage obtained was less than 70% (which we considered an acceptable level).

We also created a GitHub action “Python application” described in the file .github/workflows/python-app.yml :



This action creates a workflow for running a job that builds the application, starts the microservices and then runs the unit tests, which the purpose is to run that workflow when a pull request is created for the main branch.

**Authentication**

For the authentication on our API, we used the Auth0 external service. With Auth0 we configured an authentication API and created an App so that the integration with our Steam Reviews API was possible.

On this Auth0 authentication API we defined two different roles/scopes, the normal users, and the admins. With this we have three types of users that can use the Steam Reviews API, the non-authenticated users, the authenticated users with normal permissions, and the admins. The difference between normal users and admins is essentially that admins can add or remove games and accounts.

Uma imagem com texto

Descrição gerada automaticamente

The flow used for this authentication App was the implicit flow.

So, we blocked some operations so that they could only be accessible by authenticated users, some for general users, some restricted to admins. This can be tested on swagger openAPI, for that, the user can click in “Authorize” and choose the permission he wants that session to have, either normal user or admin. After this he will be redirected to an auth0 login/register screen, that can be personalized to our liking. The user authenticate himself and then is redirected back to swagger, but the token of the authentication is stored by swagger and included on the request’s headers, and this is the way that when a request is made the token is verified and if it is valid the request goes through, if is not valid the unauthorized response is given.

Uma imagem com texto

Descrição gerada automaticamente

**Reliability**

To measure the reliability of our implementation we used **SonarQube**. The scanning was locally, where we called the sonar-scanner to our project folder (cloned-repo where all pulled files go to from our repository) and the scanner checked all folders and files, measuring all the metrics specified within SonarQube.

On the SonarQube the primary indication of reliability is the number of bug issues. The difficulty of individual issues, their number, statuses, types, and severities are used to determine reliability rating and reliability remediation effort.

The **reliability remediation effort** is the effort to fix all bug issues. The measure is stored in minutes in the DB. An 8-hour day is assumed when values are shown in days.

|  |  |
| --- | --- |
| A | 0 Bugs |
| B | at least 1 Minor Bug |
| C | at least 1 Major Bug |
| D | at least 1 Critical Bug |
| E | at least 1 Blocker Bug |

SonarQube score rating

Uma imagem com texto

Descrição gerada automaticamente

Project final score

**Cost Evaluation**

|  |  |
| --- | --- |
| **Services** | **Cost** |
| GKE Standard Node Pool (Kubernetes Engine) | 25.30€ |
| GKE Cluster Management Fee | 0.00€ |
| Anthos | 15.16€ |
| Cloud Build | 0.00€ |
| **Total** | **40.47€** |

Table 1: Expected costs after 1 month

These are the expected costs after one month of running our application in the cloud. To obtain the cost of each service, we used Google Cloud Pricing Calculator, obtaining a total value of 25.30€, which is possible to cover in its entirety by the coupons given by the professor.

The Kubernetes engine turned out to be the most expensive where it was defined that our node was running 24 hours a day having a total of 730 hours per month.

GKE Cluster is a zonal cluster that has a total of 730 hours per month where one zonal cluster is free per billing account.

We used Anthos for Prometheus and Grafana, where we need 2 vCPUs running 24 hours a day.

Although Prometheus and Grafana are not yet implemented for this phase, it is something we intend to have implemented in the final delivery and so we decided to include its value in the costs.

Finally, the Cloud Build has a cost of 0.00€, due to the fact that the first 120 minutes of building per day are free.

**Observability**

In order to be able to observe and evaluate the performance of our cluster, we had to implement a Prometheus module that is supposed to evaluate and send data to a Grafana interface which would in turn allow us to see the data represented with more accuracy. We found several difficulties doing this, the first one being that the initial cluster we implemented didn’t support the use of Prometheus. We had to roll back and reimplement our project in a cluster running a previous version of the GKE. The GCP already deploys a metrics server by default, so we didn’t need to deploy our own. We used the GCP Marketplace to deploy a Prometheus and Grafana framework, and used a bash script to get it’s pods alongside our own as we can see here:

**Uma imagem com texto, eletrónica, computador, captura de ecrã

Descrição gerada automaticamente**

After we got Prometheus up and running as well as sending data to Grafana, we were able to access the Grafana UI by connecting to the IP specified in this service, using the specified port and logging in with the credentials we generated for Grafana:

****

In the end we were able to see data about our clusters in the Grafana interface:

