Instituto Superior Técnico

Management and Administration of IT Infrastructures and Services (AGISIT)

2022/2023

Capstone Project Specification

Introduction

The Capstone Project will provide a hands-on lab environment that will allow to validate the skills and knowledge you have learned. It corresponds to 50% of the course's final grade.

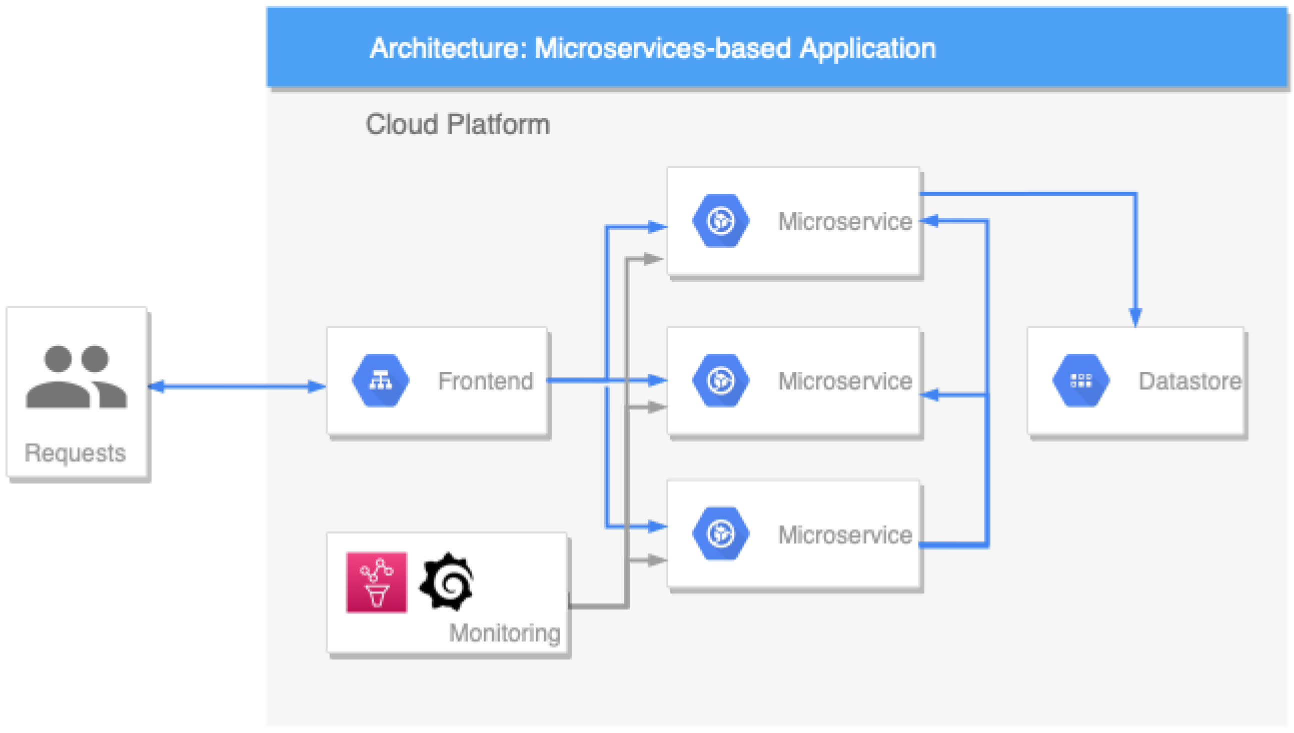
The Project is sized to be developed by a team of (preferably) 3 students in a Group.

The objective, although simple, corresponds to a **realistic** example on how to deploy and provision a tiered (frontend, backend) **Microservices-based** containerized Web Application on a Public Cloud provider, such as **Google Cloud Platform** (GCP) or **Amazon web Services** (AWS), using automation tools such as **Terraform**, **Ansible**, **Pulumi**, as well as implementing instrumentation on the applications, services and infrastructure components of the solution, to allow monitoring and logging features by using tools such as **Prometheus**, **Fluentd**, **Grafana**, etc.

The Architecture of the base solution should consist of the following main services:

- **Frontend:** the entry point (ingress) that exposes an HTTP server (serving the Web site);
- **Backend services:** provide the desired functionalities;
- **DataStore/Cache/Database:** persistent data storage.
- **Monitoring Server:** provides insight over the system

The following figure depicts a typical application architecture.



Implementation

The solution should correspond to a simple but functional system. It is up to each Group to decide how and where (e.g., in which Cloud provider) to implement and deploy the solution.

The solution should be reproducible, i.e., each Group must include deployment instructions in the respective Project Report, so that anyone following those instructions is able to deploy the system (avoiding, as much as possible, *vendor lock-in* situations).

Each Group should also decide the scope of their implementation:

- A **Base Solution**: a functional system without enhancements;
- An **Enhanced Solution**: addition of Optional Components or Deployments Methods to provide more functionalities (see below).

Base Solution

A functional basic system can be achieved following the steps described here.

This Base solution has a **weight of 80%** on the evaluation of the implemented Project.

The recommended implementation strategy involves the following steps:

- **Step 1:** Building a simple Web Application using a Microservices-based architecture. You can use some open source application (see some examples below) or develop one of your own.
- **Step 2:** Deploying the required foundational resources for the Public Cloud of choice. You will need to have the Account and Project credentials of the Cloud Provider, eventually **Virtual Private Cloud** (VPC),

networking resources and the HTTP proxy service that secures outbound communication for the application.

- **Step 3:** Building the Infrastructure using a generic toolchain for deploying the containerized application and/or necessary cloud services, using a fully automated, **Infrastructure as Code** (IaC) approach.
- **Step 4:** Deploying the Operations tooling that will allow basic traceability and performance monitoring of the solution. For that, you will need to have configured the application (and other resources) to expose some metrics in order to show their health and status, and then deploy a monitoring server using for example **Prometheus**, and create a Dashboard (e.g., using **Grafana**) to display the metrics in real time.

The Source Code of the Project should be located in the folder `project` of the Group Repository in RNL Git, and used for team collaboration and versioning.

The update of the repository must be done regularly, corresponding to the distribution of work among the team members and the various stages of development. Each Group member must update their group's repository as they complete the various tasks assigned to them.

Optional Components or Deployments Methods

The Optional components for the solution either provide additional features or correspond to different deployment methods.

The implementation of these optional features has an additional **weight of 20%** on the evaluation of the implemented Project.

The top evaluation of the Project can be reached by the following options to the solution (chose one):

1. Create and run a **Continuous Integration** (CI)/**Continuous Deployment** (CD) pipeline for the Web Microservices-based Application, using for example **Jenkins**, in order to test, package and deploy the application.
2. Deploy the Web Microservices-based Application, with *high availability*), which requires Balancing and a database backend service with persistent storage.
3. Use orchestration tools such as **Docker Compose**, **Nomad**, **Kubernetes**, to *automagically* **deploy**, **scale**, have adequate **networking** (i.e., a *Service Mesh* ensuring a streamlined communication process in between the microservices), and restart services in case they are stopped.

Deliverables

After finalizing your project, you need to prepare documentation on it (deliverables) and submit it for evaluation.

There is only one submission for each group, produced collaboratively, and needs to be submitted by just one member of the group (any one).

The deadline for this submission is Saturday, November 5, 2022, by 11h59 pm.

The deliverables to submit (to the specified platform) are the following:

1. **Fenix**: a Report describing the Project, including the architecture of the solution, the components, the tools used for the deployment.
2. **RNL Git**: the Source Code of the project, in the `project` folder of the repository of the Group (`team-XX`).
3. **YouTube**: a small Video (tutorial or demonstration) showing the deployments steps of the solution until being running in production and demonstrating its behavior.

Project Report

The Report must be a PDF file submitted to Fenix, in project **AGISIT-PROJECT**.

The name of the file **must** be `project-team-XX.pdf`, in which `XX` is the number of the Group in Fenix.

The report's format is flexible and to your liking, although the following guidelines should be taken into account:

- The number and name of each active member of the group should be included (this is to double check the composition of the groups, since sometimes there are changes that are not reflected in Fenix). The recommendation is to use the same table that has already been used in the lab reports;
- The report is be written preferably in English, but can also be written in Portuguese, as long as it can be understood by all the members of the group;
- The link to the video must be provided in the beginning (e.g., in the introduction);
- The report should include figures or diagrams, for example the Architecture of the solution, which can be produced using for example the Diagrams.net tool;
- The content of the Report should include an Introduction to the project (its context), the methodology (design, architecture), and a description of the implementation (components, prerequisites, configuration, implementation options, tools used, etc.);
- Relevant aspects for the evaluation of the report are the document organization, the amount and quality (technical/scientific) of relevant information, as well as its quality as a written document (grammar, orthography, terminology, etc.).

Source Code

The Source Code of the Project should be located in the folder `project` of the Group Repository in RNL Git, and contain all files and folders necessary for reproducing the experiment, including instructional READMEs, screenshots and diagrams illustrating the architecture, the steps, the procedures, and application usage (including screenshots of the Web UI), etc.

The `README.md` in the `project` folder should contain:

- a brief description of the application and of the solution and its components;
- instructions for installing and configuring the entire system so that it can be setup in fully operational conditions.

Video

The video can be structured in any way. It can be an interactive tour, a demo, a slideshow with voice overlay, etc. It constitutes an opportunity to better explain how the application was developed, how it is implemented, the main hurdles encountered, and above all to demonstrate that it works!

The requirements for the video are:

- Should be done in collaboration by all the members, not just one.;
- Should be released on **YouTube**;
- Should be public (or unlisted, if students are not comfortable with showing it to the world);
- The link to the video MUST be provided in the beginning of the Project Report; (e.g., in the introduction);
- The duration should be **around 10 minutes** (not critical, just a guideline);
- The resolution should be at least HD-720p (1280x720);
- Its quality can be assessed by the degree to which the application, its architecture, implementation, configuration, etc., the viewer is able to understand the work done and by how successfully it demonstrates that the application actually works.

Examples of open-source applications

The following examples are given as inspiration material for the Project. Some of these examples are used as demonstrators or sandboxes for the technologies and/or tools used for the build and deployment of microservice-based applications, either locally and on-premises (in a development computer) or on different Cloud providers.

You are **free to "re-use"** or adapt for your Project the open-source applications in those examples.

However, **you are not allowed to just "reproduce"** *ipsis-verbis* in your project the procedures there exemplified, as it would constitute an act of **plagiarism**, and also because in most cases those procedures are not appropriate, may not be applicable in this Project, or may even be deprecated (due to the normal and very fast evolution of the Tools and Services being used in those examples).

Build a Continuous Deployment Pipeline with Jenkins and Kubernetes

This example from Google, demonstrates the steps necessary to continuously deliver a software application in containers using Jenkins to orchestrate the software delivery pipeline.

The example is available at:

- Continuous Deployment on Kubernetes [https://tinyurl.com/j2n65t9]

A Cloud-native Online Boutique

This example, from Google, demonstrates the deployments of a cloud-native web-based e-commerce **Online Boutique**, consisting of a 10-tier microservices application where users can browse items, add them to a cart, and purchase them.

The application works on any Kubernetes cluster, even locally in a personal computer.

The example is available at:

- Online Boutique [https://tinyurl.com/y5vs98d3]
- Online Boutique in AWS [https://tinyurl.com/yfwxaj6w]

A Browser-based Calculator

This example is a simple browser-based Calculator with a fancy Web frontend in which the operations (addition, multiplication etc.) are served by microservices written in various programming languages.

The example is available at:

- Calculator [https://tinyurl.com/yeazcdon]

The Stan's Robot Shop

This example is a microservice application that can be used as a sandbox to test and learn containerized application orchestration and monitoring techniques.

The application works on any Kubernetes cluster, even locally in a personal computer.

- The example is available at: Stan's Robot Shop [https://tinyurl.com/ygszh9bf]

The BookInfo

This example is a simple application composed of four separate microservices, orchestrated with a Service Mesh.

The application works on any Kubernetes cluster, even locally in a personal computer.

The example is described using different Service Mesh Tools and deployments at:

- Bookinfo with Openshift Service Mesh [https://tinyurl.com/yjhor7tn]
- Bookinfo with Anthos Service Mesh in Google Cloud [https://tinyurl.com/yjmno4nv]
- Bookinfo with Istio Service Mesh [https://tinyurl.com/yzzzja9m]