Ciências ULisboa | Informática

Cloud Computing

Project definition

1 Introduction

The goal of the project is to develop a cloud native application that offers a set of services that provide relevant information extracted from a dataset, or multiple datasets, as a set of business capabilities and use cases. The services that are the frontend for the use cases will be provided through a REST API. Each group will choose its dataset(s) and set of services. Interconnected use cases should be emphasized, because the idea is to create a single application and not just a cluster of small independent applications. A suitable architecture should be designed. A typical architecture is organized in layers. A 3-Tier architecture provides a first layer of microservices that support the external REST API, the second layer implements the business logic, and a third layer handles the database(s) access. Communication between the services should preferably use gRPC. Any other contribution in the context of cloud computing will be valued. The main areas of contribution are microservice patterns, serverless, devops, security and data science. The resulting application is to be deployed in a cloud environment and make use of cloud services. Automation should be emphasized as much as possible for building, deploying and running the application.

2 Group organization

Each group should work as a new startup delivering business value in the form of a cloud native application. The ideal group size is between 3 and 4 students. Each participant is expected to make clear contributions that are relevant in the context of the course.
A suitable approach is for each student to participate in all phases of the project and be responsible for his/her own (small) set of microservices from its conception up to its deployment in the cloud.

3 Project development and evaluation

This project has two main deliveries, where the project is evaluated, and the individual contributions are scored. To support those deliveries, there are ten submission phases that correspond to project milestones and, again, the individual contributions are scored.
Completing these milestones in a timely manner awards each student with a score for the continuous evaluation that reflects its contribution. Completing the first delivery in a timely manner with a project release that can be deployed and run in the cloud awards each student with a score for the periodic evaluation that reflects its contribution.
The purpose of the milestones (continuous evaluation) is to set a pace for project development. A student that keeps the weekly, or bi-weekly, pace is awarded with a score. If, on the other hand, a student can't keep up with the pace (doesn't get a score for continuous evaluation) but is able to complete the two main deliveries in a timely manner it is still eligible to be awarded with full marks for the periodic evaluation. Lastly, if a student misses all deadlines will not have a chance to be awarded with a score.
The deadlines for the phases are already published at Moodle and should be considered as final. The deadline for phase 5 should be seen as the deadline for the first delivery and the deadline for phase 8 should be seen as the deadline for the second delivery.
Scores for the periodic evaluation depend on submissions to phases 5, 8, 9 and 10. These account respectively for first and second deliveries, CloudyDay and documentation/report.
Complementary information regarding the evaluation can be found at the course synopsis and slides about the course organization.

3.1 Repository

The group should use a single code repository, such as GitHub, GitLab or BitBucket, and possibly use a branch for each member to develop and test his/her contributions.
There will be two mandatory releases, one per delivery, that must be created from the main branch. The releases should be named, respectively, "Project1" and "Project2".
The repository should add the lecturer of the course (mcalha@ciencias.ulisboa.pt) as a viewer member.
The repository should have all the source code, scripts and documentation (in a text format or Markdown format. Never store credentials, secrets, specific configurations such as IPs, URLs, binaries, images and other files that can be produced from sources. In short, minimize the attack surface through revealing the least possible and minimize the repository size, while at the same time providing all that's necessary for the team to make progress in the development and release of the project. There are many guidelines for best practices such as https://docs.github.com/en/enterprise-cloud@latest/admin/managing-accounts-and-repositories/managing-organizations-in-your-enterprise/best-practices-for-structuring-organizations-in-your-enterprise and https://docs.github.com/en/enterprise-cloud@latest/code-security/getting-started/quickstart-for-securing-your-repository.

3.2 First delivery (Project 1)

There are 5 milestones, or phases, to support the first delivery.

3.2.1 Phase 1 – Dataset, business capabilities and use cases

In the first phase, each group must select a dataset, or multiple datasets, that will become the data source of the system to be developed.

A good dataset should have, at least, the following characteristics:

- ascii/utf8 file, stored on plain text files (like CSV or other text file format);
- size, between 1GB (or close to) and a couple of GBs;
- recent data, for example, having been produced or updated in the last couple of years;
- have interesting information suitable to be analyzed, or data mined;
- not too simple, the data set can be comprised of several files, from different sources that can be correlated.

The dataset can have information that is relevant to some scientific project or to the members of the group. The topic of the dataset is each group's choice. A group can combine interests (and datasets).
For your reference, trendy topics are LLMs and AI agents such as https://github.com/karpathy/llama2.c and https://github.com/exo-explore/llama98.c. If you like this field make sure that the computational requirements are low, otherwise the project will not be able to be deployed without significant costs.
Examples of sources, aggregators, and lists of public datasets:
https://www.kaggle.com/datasets
https://elitedatascience.com/datasets
https://cloud.google.com/public-datasets
https://ec.europa.eu/eurostat/data/database
https://knoema.com/atlas/Portugal/datasets
https://github.com/awesomedata/awesome-public-datasets
https://www.altexsoft.com/blog/datascience/best-public-machine-learning-datasets/

Business capabilities are related to the main sources of revenue for a business. For instance, a car company that sells and rents cars has these two business capabilities. The use cases define the general scenarios that should be offered, like configuring a car for buying and returning a rented car.

In this phase, each student should fill in the self-assessment form and submit a Markdown file with the identification of the datasets (namely topic, size and date of release), business capabilities to be implemented and use cases that are your contributions. The document is basically descriptive, so diagrams are not necessary for this submission. Nevertheless, diagrams can also be done by following instructions at https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/creating-diagrams. This markdown should also be added to the project repository so that it begins the README.md that documents the whole project.

3.2.2 Phase 2 – API specification

The goal for this phase is to specify a REST API using the OpenAPI format to support the use cases. The specification should cover all endpoints, parameters, responses and descriptions. The API specification should be delivered as a YAML (or JSON).
The specification can be done by using https://editor.swagger.io/ . You can find the documentation at https://swagger.io/docs/specification/about/

An advantage of specifying the API in this way is that the code can be automatically generated. In this phase, each group must submit the actual specification of the REST API for the basic services. As a reminder, the application is not expected to have a client-side front-end.

In this phase, each student should fill in the self-assessment form and submit the OpenAPI specification file.

3.2.3 Phase 3 – Functional requirements and application architecture

In this phase the functional requirements and application architecture should be defined.

The functional requirements describe what the product does and focus on the user requirements. They provide a more detailed description of each use case. Functional requirements focus on the system's functionality and behavior, while use cases focus on the user's goals and needs. Functional requirements cover the entire system and all its features, while use cases cover only a subset of the system and its functions.

The specification of the architecture has two aspects: application architecture and technical architecture.
The definition of an **application architecture** involves defining:

- The components like services and databases developed in the context of the project
- The interactions between these components (including the protocols such as HTTP, REST and gRPC)

This architecture is driven by the functional requirements and should be presented with a simple diagram that shows both the components, their connections and properties. The type of connections should be defined (e.g. REST, gRPC). This diagram should be accompanied by a description.

On the other hand, the **technical architecture** is driven by the non-functional requirements. This architecture mainly shows the cloud provider services/tools and any tool/script developed by the group that supports the fulfillment of the non-functional requirements. This phase requires the submission of **only the application architecture**.

In this phase, each student should fill in the self-assessment form and submit a Markdown file with the functional requirements, description of the architecture and the diagram of the application architecture.