Final Project Requirements

2019 / 2020 SDE class

The goal of the final project is to ideate, design, develop and demonstrate a service-oriented application. Following the spirit that we apply to the lab sessions, we leave this exercise in a quite open form. We ask you to find a problem that you deem interesting and which makes sense to be solved with a service-oriented approach.

You are free to choose your own programming language/framework to implement the project. Ideally, the content of the lab sessions can serve as a good starting point for what technologies are available and you might want to use. As for the ideation and design, we will offer you dedicated sessions (for the entirety of December) to share and discuss the idea and design of your final project.

We strongly encourage you to work as a pair or triplet for this task. As we have seen in the class, one of the strengths of service-oriented approach is that services can be developed separately and then composed at a quite high level. So the service-oriented approach is good for team work.

Your project will be evaluated along three main axes:

- Execution (i.e., quality of code and API documentation)
- Architecture (i.e., correctly applying the service-oriented logic to your application)
- Complexity level (i.e., pinpointing a topic which is feasible, but also sufficiently complex (more details below))

During your chosen exam session, we will ask you to present and demo your project to the teachers. Grading for the project will be individual.

Example Architecture

The final service oriented application **will need to be structured hierarchically similarly to the generic and schematic architectures below**. Of course, **you should adjust** these generic examples as you find more appropriate for your own design.

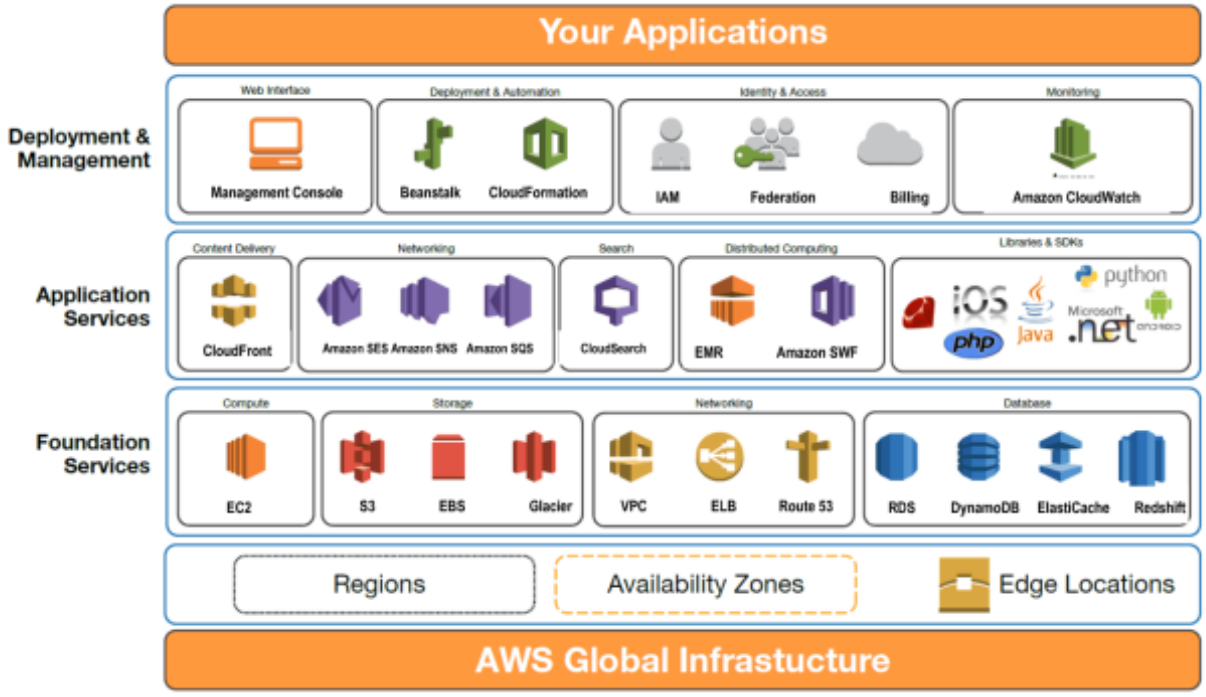

Image 1: Amazon Services Architecture.

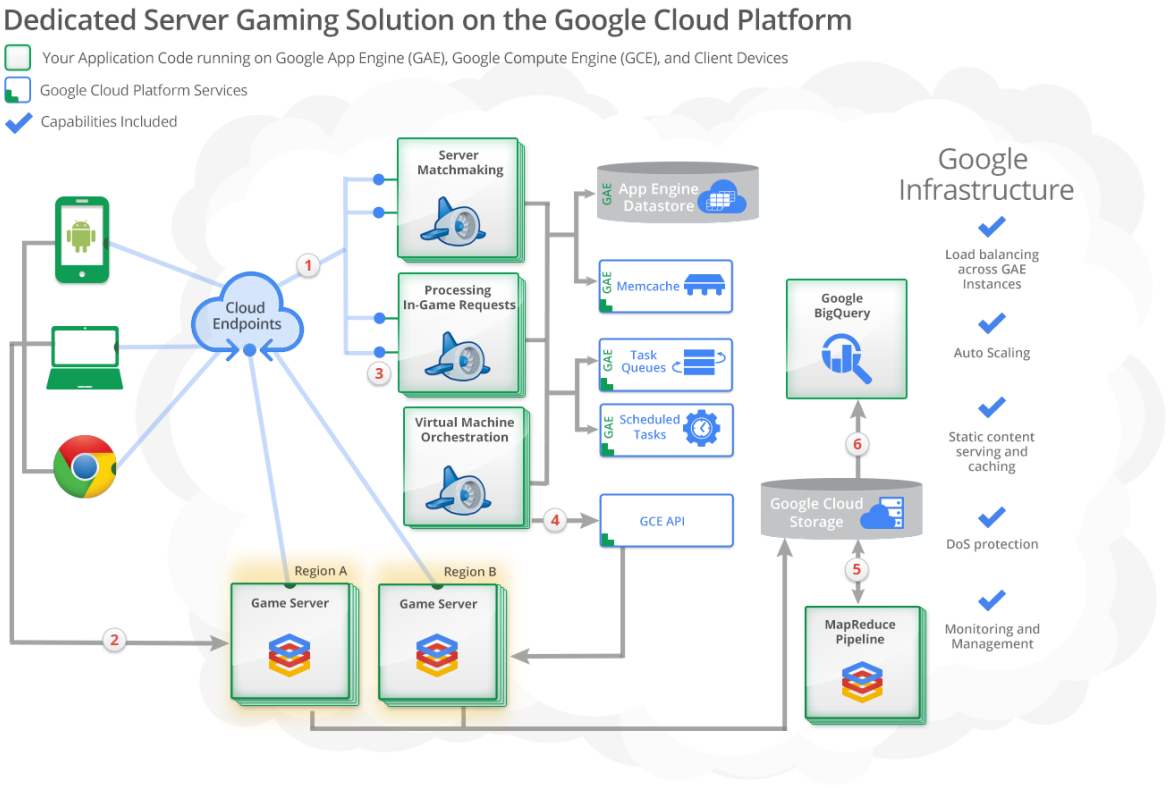

Image 2: Google Cloud Generic Services Architecture.

We expect in the final project at least the following logical layers:

- **Data Services Layer:** responsible for handling all data related requests. Mainly, services in this layer include, *persistence* and *retrieval* related tasks. The data services layer sits on top of internal databases and external data sources in order to provide data to all other modules in the application. This layer can talk to different adapters (adapter layer), like database adapter (MySQL DB code), external API adapter (e.g., Google Maps API) or HTML adapter (e.g., an adapter that crawl data from HTML pages).
- **Adapter Services Layer:** these services link the application to the outside world in terms of data or external services using different technologies. This layer extracts, fetches data from various external data sources and or services. These services include, html crawling, external database access, external APIs and software as a service frameworks.
- **Business Logic Services Layer:** this layer is responsible for all kinds of data manipulations, decision making, calculations, reminders sending, items recommendation, decisions, etc. This layer receives requests from layers above (either from the process layer or in some cases directly from the App User Interface Layer), it gets the needed data from the data layer and processes them to send results back to the user.
- **Process centric services Layer:** these layer serves all requests coming directly from users (from application interface). The process centric services are the gateway to all other modules/services in an application context. This layer is ***orchestrating and coordinating the composition of a number of existing services*** in the layers below to fulfil complex user requests or goals.

Project requirements

General requirements

- Implement at least one service for each different layer mentioned in the above architecture (min 4 in total, but normally more than 4):
    - **Data Layer**
    - **Adapter Layer**
    - **Business logic Layer**
    - **Process Centric Layer**
- Service implementations should be **general, reusable** in different scenarios and **scalable**
- The project should incorporate both **internal and external** sources of data.
- Services must be based on **RESTful** (or **SOAP**) web service technology.
- Each service can interact with other services **only through API**.
- We expect you to **define data structures** (JSON or XML) you use as an input/output for your services. In your documentation provide these structures as XML schema documents or JSON examples.
- Any **database management system** (SQLite, MySQL, MongoDB, Oracle, etc.) can be used as an internal database (e.g. consider using ClearDB as Heroku Plugin).
- **Deploy your project to a VM, Docker container or Heroku.** Should you use a VM, we ask you to deliver that to us. Should you use Docker or Heroku, provide us with all necessary links.
- **Host all the code on Github**. Start using GIT from the very beginning. Tip: Commit often, so you won't lose your work!
- Should you find it useful, you can create a **Github organization** to group several repositories (in order to have a separate repository for each service you have running). Should you do this, remember to make all repositories public.
- Create a short **report of your project** with a figure describing your designed architecture. Make sure your description together with the figure **does not break 3-pages limit**.
- Create the **documentation of your APIs**. To do that, feel free to use the tools presented in the lab session. Alternatively, you can create a wiki page on github or use other services, such as Apiary or ReadTheDocs.

1 Student (choose this if you are limited in time)

- **At least a command line user interface.**
- Implement **one process centric service, i.e. one integration logic** (check the service orchestration lecture), which invokes **at least 4 services** (i.e. instantiate at least one process: sequential and/or parallel).
- Should your application require registration, this should be implemented as part of the project. Some API endpoints should require a basic form of authentication (such as HTTP header Token authorization). It is fine to lack a comprehensive access control.

2 students (we prefer you choose this option)

- **At least utilize a third party service for the user interface**. The user interface can be based on Postman App. Alternatively, you could develop the user interface based on Slack incoming and outgoing webhooks, so you can interact with your service via Slack (more convenient than pure command line). Or you can implement a Telegram bot (docs, example 1, example 2) to talk to write and read data from process centric services. You can consider any other way to implement your UI which is more usable than command line interface.
- Implement **2 process centric services, i.e. 2 integration logics** (check the service orchestration lecture), which call **at least 4 services each** (sequentially and/or in parallel).
- Should your application require registration, this should be implemented as part of the project. Some API endpoints should require a basic form of authentication (such as HTTP header Token authorization). It is fine to lack a comprehensive access control.

3 students (choose this if you plan to implement a production prototype system other people can use)

- **Implement visual interface** (even very simple). Web, Mobile or Desktop interfaces are all acceptable (check implementations of front-ends in different frameworks).
- Implement **3 process centric service, i.e. 3 integration logics** (check the service orchestration lecture), which call **at least 4 services** each (sequentially and/or in parallel).
- **Add OAuth access control**. Make sure you have permissions to certain services limited via access control. Users login to their accounts using login and password obtained from an OAuth provider (such as Google or Facebook). Make sure one person can not edit data (user profile) of another person unless it is specifically designed to work in this way (e.g. an user add “likes” to another user or an administrator edits records of a user). Decide - and design and implement accordingly - who owns the Items and can perform CRUD operations on them

Submission

Each student should submit a **FORM** 48 hours before the exam date time. The link will be shared later in due time.

Appendix