Architecture Design

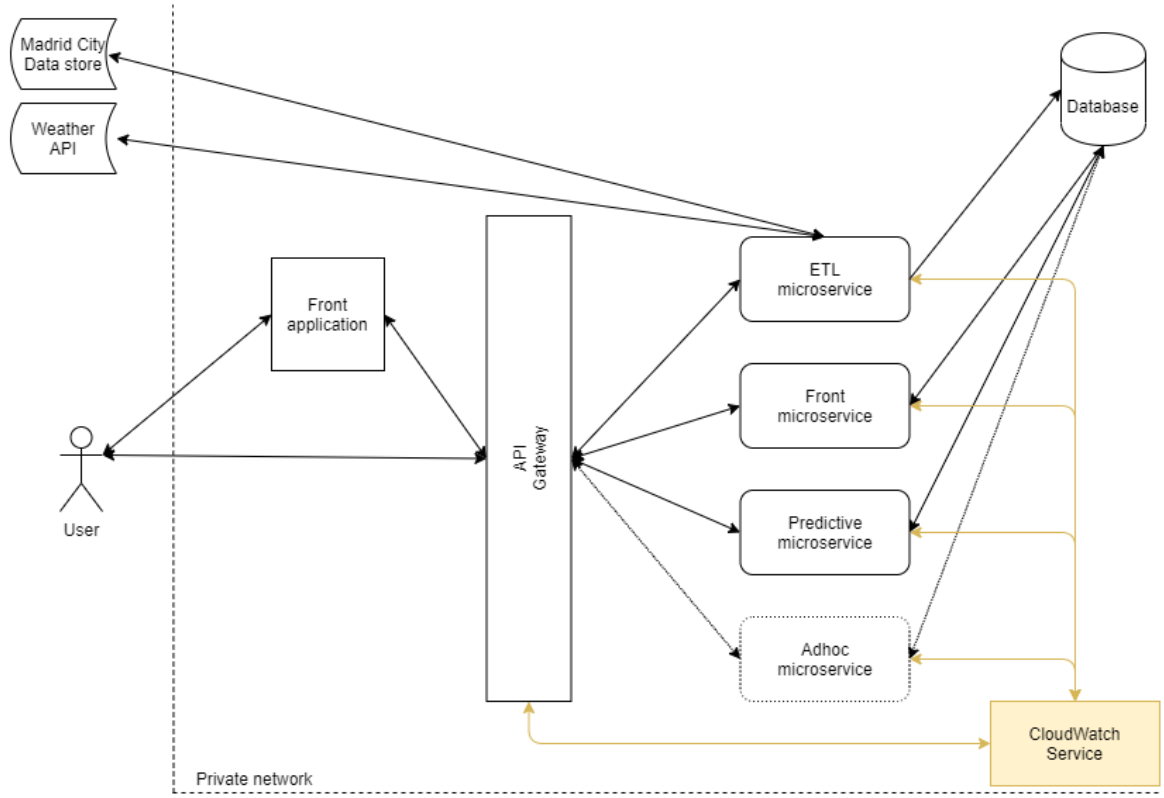
For the architecture design I will go through some general design decision then draw a basic diagram and, at last, explain all the different parts.

General design

Since this is a small team and there is no systems engineer I would recommend to use a cloud service provider. They usually have simple web interfaces and serverless applications that require 0 maintenance. I would use AWS because I have experience using it and find it extremely complete and easy to use.

All decisions should be discussed with all the team but since I cannot do that I am assuming that AWS will be used and all my recommendations followed. I have not tested all of this and don't have practical experience with all the modules but based on the docs I have read I think it will work and would love to do it. :)

Diagram



Module Explanation

CloudWatch Service

Cloudwatch is the AWS logging and monitoring tool. It is integrated with almost all other AWS tools making this process much easier than in a company developed environment. It will take care of:

- Gathering all our system logs and monitor all the pieces.

- Optimize resource utilization.
- Automatically register relevant metrics and detect anomalous behavior.

The yellow arrows in the diagram indicate that the data exchange is not of business information but logging and monitoring data.

Database

For the database I would use Amazon Aurora because it is:

- Compatible with PostgreSQL → We don't need a noSQL database, it is open source, used extensively.
- Serverless → Does not require instance managing and The database will automatically start up, shut down based on your application needs.
- Automatic scalability → The database resources will be automatically managed to match application needs.
- Pay for use only → The company will only be charged for what is used. Since we are not having a lot of clients this will be very cost effective.

API Gateway

The api gateway is the only entrance to our backend. It makes sure that some security standards are met: takes care of authentication and authorization, logging and endpoint monitoring among other things.

I think this is a key piece of the architecture because, from a simple UI you can control the access to all your endpoints using, for example API keys for third party developers. It also has integration with Cloudwatch.

Microservices

Microservices are simple applications that take care of one "relatively simple" job. Instead of working on an application that solves all our problems we will break our problems in simpler tasks and create a microservice for each one of them. This is done to avoid complex applications and isolate functionalities. If there is a problem it will not affect all the system, only one of our microservices.

To make the deployment of our microservices easy I suggest to use Docker because:

- It makes sure that the environment where our application is tested and used never changes (avoiding the classic "It was working a minute ago! what happened?!").
- Docker allows for horizontal scalability, this means that if a microservice is too busy you can quickly start a new one and move some of the load to it.
- A lot of microservice frameworks have docker integrations such as Quarkus for Java microservices. Python's Flask does not have it but deployment on a container is not hard at all.
- You can automate the container build process and sync it with code repositories so when a code is pushed it is automatically uploaded and a container started with the new code in the development/testing environment. (This is done using docker hub)

AWS, as always, knows about this stuff and has serverless integration with Docker's containers through the Amazon Elastic Container Service and AWS Fargate. This means that container management will not require server maintenance and will be done easily. (The docs say it is easy but I've checked and it does not look that easy without a systems engineer, there are other alternative and maybe easier ways of doing it.)

ETL

All rest requests related with data loading (incremental and massive) will be sent to this service. It will send requests to external services, process the response's data and load it in our databases. It also might have some kind of scheduled behavior for incremental loads.

Front

The front application will need data to build it's views, it will ask for it using rest requests that the Api Gateway will redirect to this microservice. The service will then gather all the necessary information from the database, transform it to be used in the front application and send them back in a http response.

Predictive

This service might be complex, it may require some extra resources like more CPU I/O or Memory. This means that the container needs to be run in a bigger machine (EC2) not using the Amazon Elastic Container Service and AWS Fargate but this does not affect our architecture.

From an architectural point of view, this service is not that complex. It will receive some requests, process the data received and send a response back. The process might take a long time, a lot of resources, access to a data warehouse, model manipulation, etc. If complexity increases a lot then the microservice can be split in two, for example model management microservice and prediction microservice.

Examples of requests might be:

1. Asking for contamination values for a given future date and it's confidence level.
2. Telling the service to train or test a model (might be in the database or given as a file attachment in the request).
3. Changing training parameters such as algorithm, distance, weights, etc.

Adhoc

You can add as many microservices as needed for any task imaginable and, with the architecture explained here, you will always know:

1. Adding a new microservice will require minimum extra code in other modules (zero if the new functionality is isolated), only configuration in the API gateway.
2. The authentication and authorization will be managed by our API gateway. We will only need to create new users and groups.
3. Deployment will be easy and almost automatic.
4. Logging and monitoring will be exactly as in the other microservices as long as the new microservice uses the libraries and tools provided by amazon.