Reflection Document – Group 1

The objective of this group project was to predict house prices from a Kaggle dataset using machine learning models, by developing an MLOps infrastructure in order to automatize training, model deployment and testing, while keeping track of data and model versions.

The first challenge has been finding which role suited the most each member of the group. We divided our roles by considering each team member's closest interests, background, and also the job title we have been assigned inside our company (Bip).

In fact, two team members, who are now assigned into the Data & AI department, focused on data preprocessing, feature engineering and model training and deployment. The other two members, who are more into DevOps, MLOps and cloud technologies, took on the tasks of setting up automated workflows and managing containerization.

This division allowed us to work on what we are most familiar with, which has been important considering how little we were familiar with the adopted technologies. After sharing our heterogeneous knowledge, and after extensive internet research, we could start tackling the challenge.

The main objective and challenge of this project was to obtain a functioning infrastructure involving all the steps for the lifecycle of the data. Setting up a whole environment seemed a bit overwhelming at first, especially considering the vast amount of features offered by AWS, together with a relatively steep learning curve that comes along with them.

The Architecture we adopted is depicted in figure 1.

The CI/CD pipeline consists of a local development environment on the dev branch and a main branch that includes two GitHub Actions:

1. **Deploy Lambda to ECR** : This action is triggered by a push to the main branch within the lambda folder. After authenticating with AWS and ECR, it pushes all Lambda images to the ECR repository named "gruppo1".
2. **Deploy Sagemaker Job to ECR** : This action works similarly to the first one but is triggered when changes are pushed to the main branch within the “sagemaker_job” folder. It pushes SageMaker job images to ECR.

The first step concerns data ingestion. A weekly schedule triggers the ingestion lambda in order to simulate the retrieval of data changing over time. This lambda function calls Kaggle API to retrieve the dataset. This dataset is split in train/validation/test datasets which are stored in Amazon S3. At the end of this step, another lambda is called with the objective to trigger the step function. The step function activates another Lambda function, which generates a unique string to ensure that each job which calls the SageMaker API is distinct. After this, four SageMaker jobs are called sequentially. These jobs take care of data preprocessing, model training, model deployment and testing. Each job produces an output which is stored into

Amazon S3.

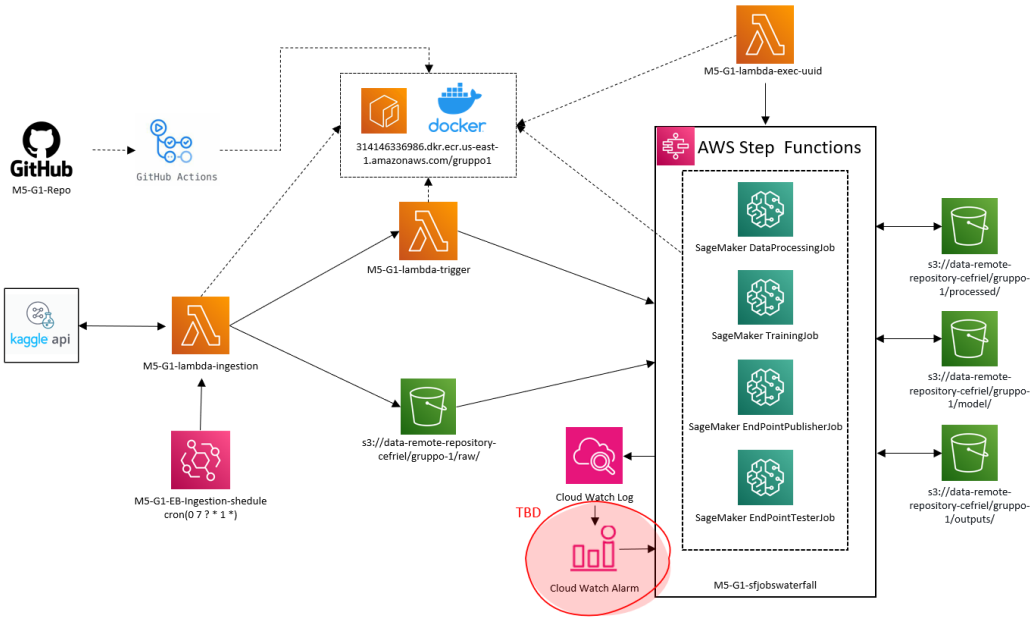

Figure 1: Architecture

The preprocessing job filters the most relevant features, deals with missing values, filling with the mean value. The following step is model training. The model is a Random Forest Regressor, which takes as input the train dataset and checks the output effectiveness on the validation dataset, calculating MSE score. The last two jobs deal with endpoint creation and with endpoint testing, which simulates an endpoint call sending the test dataset.

With this architecture, we obtain a functioning endpoint ready to accept requests and to provide predictions.

During the development of the architecture, we faced several problems. One of the most relevant was the lack of permissions to perform certain tasks on AWS. This brought to delays in the development. Another problem, which is still present, is the missing compatibility between Kaggle API and Lambda functions. This is due to the fact that Kaggle API uses multiprocessing, which is in fact not supported in Lambda. This ingestion therefore works locally but not on AWS.

This would allow for automated monitoring, detection, and retraining mechanisms to maintain model performance over time.

Possible future developments include implementing an MLOps cycle to address issues caused by violations of certain acceptability thresholds due to data drift (highlighted in red), as well as the potential to incorporate techniques like A/B testing to improve model performance and decision-making.

To conclude, this was certainly a very challenging project: MLOps is a complicated workflow and setting it up from scratch required a lot of dedication, especially at first. For sure there is plenty of room for improvement, but, given the limited time we had, setting up the infrastructure has brought a fair amount of satisfaction.