OptiFlow API

Noah Depp, Oscar Depp

Project Introduction

Inventory shortages and excess stock are critical challenges faced in regions with limited technological infrastructure, leading to increased costs, waste, and missed opportunities. We developed a predictive inventory management system leveraging AWS services for scalability, efficiency, and automation to address this, with the overall workflow represented in Figure 1.

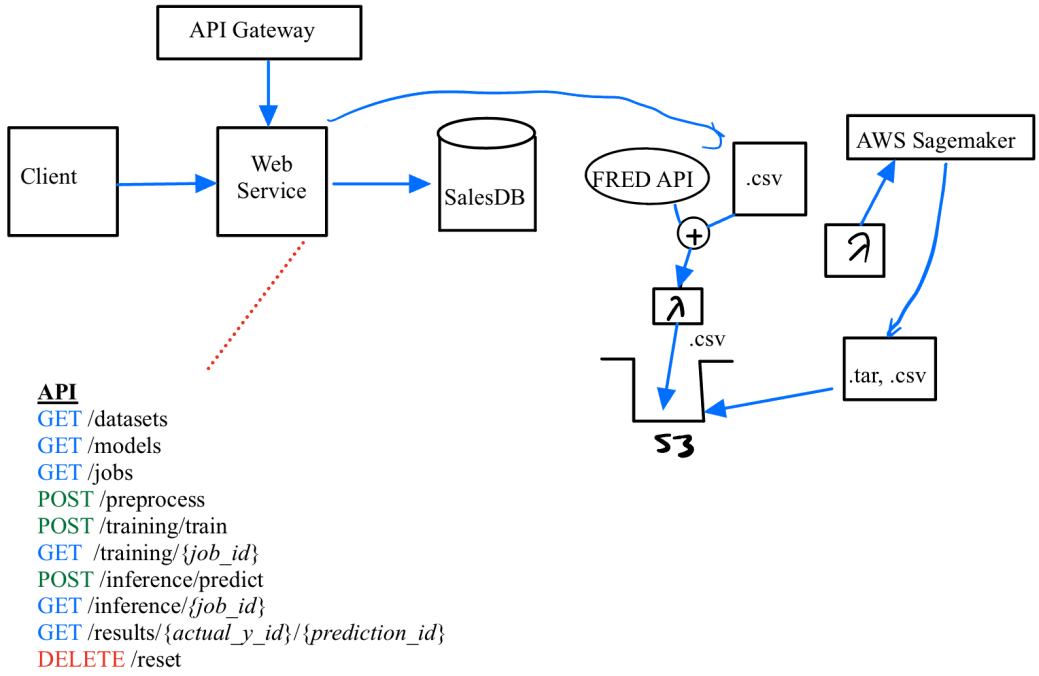

Figure 1. High-level depiction of the OptiFlow pipeline

Pipeline overview

Our system employs an asynchronous RESTful API architecture implemented through AWS API Gateway, integrating various AWS services. Key data components like historical inventory levels, sales data, and supply chain variables make up the input dataset, storing the files in S3 and information in RDS, which is organized into datasets, models, and jobs tables, where jobs are Sagemaker jobs defining whether the training or inference is done yet or not. User interactions flow through a streamlined process described in the next section using lambda functions to achieve these functionalities.

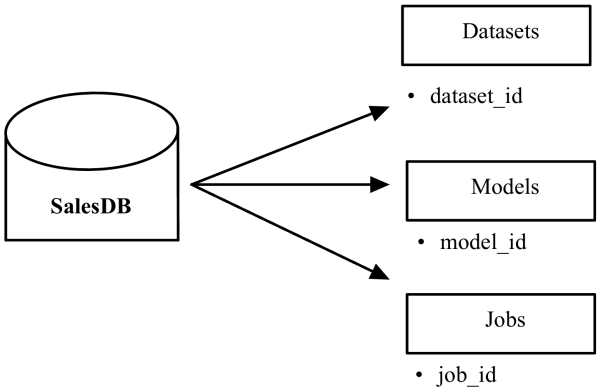

Figure 2. Tables used in SalesDB Database in RDS.

Explanation of Server-side Functions

The system begins with data ingestion and preprocessing, where input files are uploaded and enhanced with additional features such as 10-year yield and volatility, which are user-inputted(*/preprocess*). We don't memorize the series ID/ticker of each series and there are a lot of tickers, so we ask clients for keywords of the series they want and order by popularity so we can select the series that is most used. We do this by interacting with the FRED API. For example, if we want to look at the 10-year yields, an indicator of economic health, we type this into the client app, using the query /search?search_text=10+year+yield. We encode the parameter by using the function urlencode for the FRED API call. There are a lot of results so we pageify the results into 10 results/page ordering by popularity so the client doesn't have to scroll a lot to get to the most used ticker. These additional features are taken from FRED's API and a column is appended to the user's inputted data matching the OrderDate with the feature's date. The null values are filtered out in this function. The preprocessed data is stored in Amazon S3, generating a dataset_id for tracking.

Users can then initiate model training via an API call(*/training/train*), using the dataset_id and hyperparameters as input for a model (a tree-based XGBoost model) through AMSagemaker, which produces a job_id. Polling functions that poll whether training is done is important because training often takes longer than the 15 minutes than lambda function allow. The status of this job can be checked by users by polling for the training status that Sagemaker is running(*/training/{job_id}*).

Once the model is trained, SageMaker is utilized to test and deploy it. Predictions are generated through API calls using the model_id and dataset_id, resulting in a job_id that links to the prediction results (*/inference/{job_id}*). The error of the model prediction can be retrieved as well once the predictions are uploaded, inputting a truth value dataset and predictions dataset (/results/{*actual_y_id*}{*prediction_id*}) There is also an embedded functionality that reset the database to default settings(*/reset*). Every

dataset, model, and job can also be retrieved via GET methods (*/dataset, /models, /jobs*).

Other features

We moved away from access keys by directly attaching policies to the lambda functions, enabling us to test faster, with one user authorized to test this function. We added a Sagemaker role with full permissions to allow Sagemaker to help with this. Another extension of this project would be to restrict a user to only be able to view/add the processed and predicted results of the model, reinforcing the least privileges of each user. We attempted to use REST API features by posting.