Description about the cloud architecture

The layout of the architecture defined with GCP components is shown in the following picture. The expected behavior, as well as a justification for the components used will be provided next. However, first I would like to point out some notes on the interpretation of the problem definition:

From the problem definition it has been inferred that here it should only be defined the part of the system handling the predictions. The part managing the monitoring of the performance in the predictions plus the re-training of the model when needed has been ommited. The real sales_per_day obtained from the stores can be used for both monitoring the performance of the model and re-training this model when a performance lower threshold has been surpassed.

On the other hand, as it was previously mentioned the current trained model is a KNN, so the target variable is not used in the predictions. A (possibly) better option for developing this model is using Recurrent Neural Networks (RNN), specifically an LSTM. The reason for this is that this NNs can take past real values of the target variable into consideration for making more accurate predictions. They are good choices in time series predictions. This would be especially useful in timeframes where there is an extraordinary tendency on sales per day. However, this is not the case of the model implemented where the target variable is not used in predicting process. For this reason, the data provided by the source system A is not used for the predictions. However, it is still included in the architecture schema (in orange) just to make clear where it should be added if required. In fact, it would probably be much more interesting to have the customers_per_day available in real-time and the __sales_per_day__ (from the past days) uploaded to an ftp once per day. The latter would be the one used for model monitoring and re-training. In fact, the model trained uses the variable customers_per_day for the predictions and if the system needs to provide predictions in real-time, ideally it would need its input variables (including customers_per_day) in real-time.

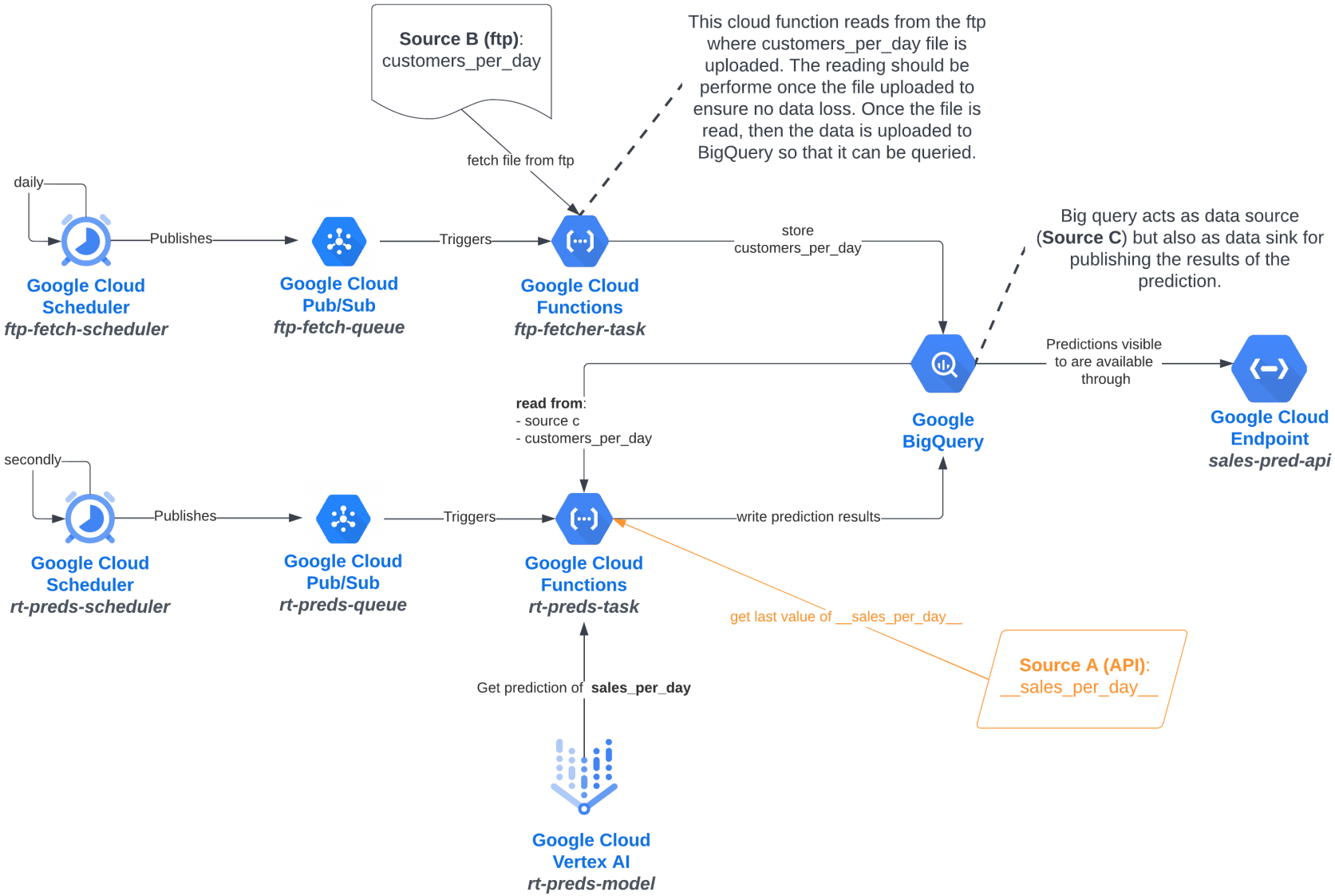
The system is formed by multiple components of different natures. The predictions model itself is implemented using the Vertex AI component. This component is indicated for training and performing predictions and for this reason, this will be where our KNN is encapsulated. Then we have two parallel pipelines triggered by schedulers. Google Cloud Schedulers are like cron jobs which are executed following a schedule plan. In our architecture we have two defined. The first one has the task of fetching the customers_per_day file on the ftp server and accordingly is scheduled once a day. The second scheduler has the task of performing the predictions itself and that is why it is scheduled secondly, so that they can be available in real-time.

The only task of the Schedulers itself is posting a message on a queueing system which is the Pub/Sub component on Google Cloud Platform. Once this message is posted, all the subscriber services to the queue will be triggered. Google Cloud Functions are the services subscribing to the Pub/Sub queues. The Cloud Functions are small chunks of code that can be executed on demand serverless, the idea is that when they are configured to be subscribers to a Pub/Sub they will be triggered for each message enqueued. Namely for the *ftp-fetcher-task* this will be once a day and for the *rt-preds-task* every second.

The ftp-fetcher-task will be accessing the SFTP downloading the appropriate file and storing the value of customers_per_day on a BigQuery table. The ftp-preds-task will be obtaining from BigQuery the data from system C as well as the one provided by system B, which will also be posted in BigQuery. However, at this point the granularity of the customer_per_day data may not be suitable for the aforementioned KNN model. The reason

is that this variable is needed for the predictions and if we need to provide predictions on a real-time basis, we would need this variable real-time as well.

This Cloud Function would then trigger the model trained on Vertex AI and then fetch the results and post them on BigQuery so that they are available to be queried. These results could be directly queried there on real-time. Moreover, it has also been added an endpoint which reads from BigQuery the value of the predictions and enables to be pulled as an API.


Source B (ftp):
customers_per_day
fetch file from ftp
This cloud function reads from the ftp where customers_per_day file is uploaded. The reading should be performe once the file uploaded to ensure no data loss. Once the file is read, then the data is uploaded to BigQuery so that it can be queried.
daily
Publishes
Triggers
store
customers_per_day
Big query acts as data source (Source C) but also as data sink for publishing the results of the prediction.
Google Cloud Scheduler
ftp-fetch-scheduler
Google Cloud Pub/Sub
ftp-fetch-queue
Google Cloud Functions
ftp-fetcher-task
Predictions visible to are available through
Google BigQuery
Google Cloud Endpoint
sales-pred-api
read from:
- source c
- customers_per_day
secondly
Publishes
Triggers
write prediction results
Google Cloud Scheduler
rt-preds-scheduler
Google Cloud Pub/Sub
rt-preds-queue
Google Cloud Functions
rt-preds-task
get last value of __sales_per_day__
Source A (API):
__sales_per_day__
Get prediction of sales_per_day
Google Cloud Vertex AI
rt-preds-model