Back Market Data Engineering Serve team Interview

Data Pipeline Assessment

> You can develop & refactor your code (using your versioning tool) following this pipeline:
>
> 1. Import the following file from S3 to GCP: https://backmarket-data-jobs.s3-eu-west-1.amazonaws.com/data/product_catalog.csv (Should we mention the file format?)
>
> 2. Make the data available in BigQuery for the feature team A which wants only rows for product with an image
>
> 3. Make the data available in BigQuery for the feature team B which wants to understand which Samsung products released after 2000 have no images

The **bm.py** script retrieves the data from the s3 link (as it's a public HTTP URL we don't need to use the s3 API) and makes the data available in BigQuery through 3 tables :

- raw table (exactly what's in the source data file)
- two specific tables for feature teams

The transformation part has been handled through SQL queries as it's a quite simple & efficient way of doing it with BigQuery.

> Now Back Market is growing so fast, there are lot more data to import into GCP. How would you adapt your code to scale it up to handle the increasing amount?

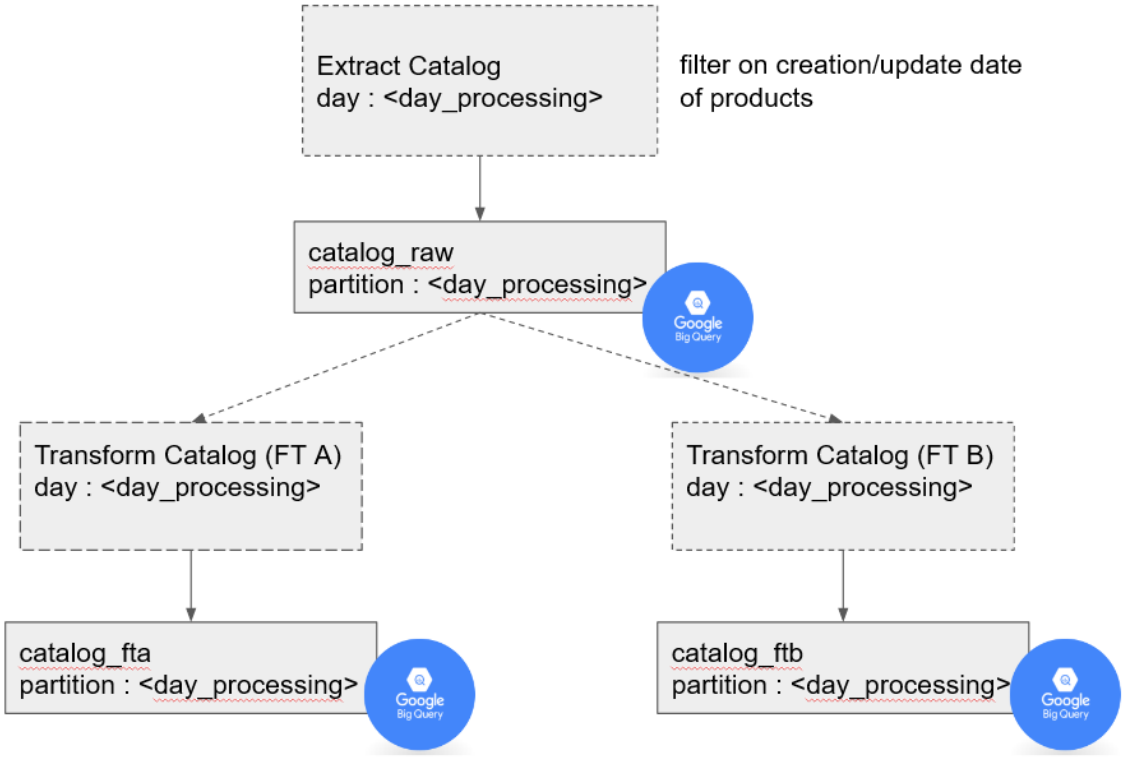
The bm.py imports data with this unique file as a full dataset, erasing the previous one. It works with a small volumetry but with higher amounts of data, it should be imported incrementally.

I've added two files in the repository to be used as an example. It should be seen either as an extract of a database made by an cron.

For example by getting data from an operational database / raw table with a filter on a created_at/updated_at field in the table to retrieve latest products added/updated, if it's possible) or a file made available by a data producer.

=> The **bm_incremental.py** script is designed to retrieve a data file with a specific date (the extraction part from the source is omitted, let's say it's already there) to insert it in a BigQuery partition.

The behaviour used (separate tables per partition and erasing them at loading time) is very Bigquery-centric but it's done with more traditional DWH or even RDBMS by deleting & inserting/updating when processing data from a specific date to have idempotent pipelines.



Eventually, more and more feature teams want to expose data and we want to provide a standard way of doing so. How would you build a tooling library and make it available to them (deployment, versioning, security, legal, etc.)?

A tooling library could be developed by the team responsible of the DWH to make external **tech** teams able to push data to the DWH without having to develop specific code and handle access control by themselves.

The tooling library could be designed like this :

- Available in a git repository as python code.
- Be pushed by the CI to a pypi index as a versioned package : every time the DWH team pushes changes on the code & update the version, a new tag would be generated and a new package available in pypi. This can be done easily by plugging the git repository to a CI/CD tool as Jenkins, Travis, or Gitlab CI.
- Handle access control through simple setup on their environment & with appropriate IAM roles given to their service account (created in Google Cloud IAM). A good way to do this is to define a separate service account by data producers group (eg FT) to only make them able to have access to data they are allowed to read, and write in datasets they can legitimately write.

The drawback of providing a simple “tooling library” for external teams is the team in charge of the DWH can’t easily have an eye on everything which is loaded, and the other team can’t take advantage of all the aspects around it which can be setup in the data platform (alerting, monitoring).

It can be mitigated by pushing some metrics directly in the library, for example by registering data in a data catalog or pushing metrics in a monitoring service directly through the library.

Other possible alternatives to make external teams able to loaded data could be to :

- Make it possible to create tables through external resources upload.
  For example, an ETL (let's say an Airflow DAG) which polls a GCS bucket to retrieve new data files uploaded with either a good nomenclature to create the appropriate tables :
  gcs://backmarket-external-teams/ftA/<table_name>_<date>.csv
  Or configuration files in a repository/in the GCS bucket to load the data in the right tables
- Use external tables with data hosted in GCS
- Make the FTs able to deploy ETL jobs on the data platform, either with a ready-to-use DAG (I'm still thinking with *Airflow*) factory, or top-level configuration files to define DAGs
- Provide a MS as intermediary to handle the loading (either by posting directly some data, or file deposit on a cloud file system + notification on the MS)