* 1. **Project overview**

The data set on which I am going to work on is from Olist, the largest department store in Brazilian marketplaces. Olist connects small businesses from all over Brazil to channels without hassle and with a single contract. Those merchants are able to sell their products through the Olist Store and ship them directly to the customers using Olist logistics partners. There are 9 source csv files:

▪ olist\_customers\_dataset.csv   
▪ olist\_geolocation\_dataset.csv   
▪ olist\_orders\_dataset.csv   
▪ olist\_order\_items\_dataset.csv   
▪ olist\_order\_payments\_dataset.csv   
▪ olist\_order\_reviews\_dataset.csv   
▪ olist\_products\_dataset.csv   
▪ olist\_sellers\_dataset.csv   
▪ product\_category\_name\_translation.csv   
This data set has been retrieved from kaggle.com, and it is protected by a Creative Commons licence - CC BY-NC-SA 4.0.

* 1. **Objectives**

This project involves building an ETL (Extract, Transform, Load) pipeline to transfer data from CSV files into a SQL Server database. The goal is to create a small Data Warehouse (Data Mart) according to a specific data model that supports analytical reporting on sales, customer, product, and order details for Olist, the largest department store in Brazilian marketplaces.

The project will integrate data from nine source CSV files, covering various business entities such as customers, orders, products, sellers, and payment information. By structuring this data into a Data Mart, we aim to enable analysis of key performance metrics related to orders, customers, and sales patterns.

* 1. **Tasks**
* Building the source DB from the csv files based on the below ERD schema.
* Building a data warehouse. The DWH will contain one Fact table and three Dimension tables.

a. Delivered Orders Fact table

b. Dim Customer table

c. Dim Product table

d. Dim Seller table

- Building a stored procedure for a simple report. The report will show a year-on-year comparison (2017 vs. 2018) of product sales revenue, shipping revenue and average review score for customer and seller state. The procedure should be scheduled to execute daily at 10 am.

**2.Architecture Overview**

The architecture for this project follows a cloud-based, modern data pipeline approach using Azure services to efficiently extract, transform, and load data. The design of the pipeline is inspired by the **Medallion Architecture**, which organizes data into distinct layers to optimize processing, cleaning, and analytics. This approach divides the data flow into different stages, allowing for greater flexibility, scalability, and maintainability.

I opted to deploy the entire solution on **Microsoft Azure**, leveraging its suite of data engineering tools to ensure a robust and scalable pipeline. The raw data, consisting of nine CSV files sourced from the Olist dataset, was uploaded to **Azure Blob Storage**, a secure and scalable cloud storage service. Blob Storage served as the **landing zone** for the raw data, which is the first layer in the Medallion Architecture.

To automate the data movement from Blob Storage to the Data Warehouse, I used **Azure Data Factory (ADF)**, a managed service for orchestrating data workflows. ADF's pipeline functionality was used to extract the raw data files from Blob Storage and load them into **Azure SQL Database**, the analytical storage layer of the architecture. The transformation and cleansing tasks were handled within the database itself to maintain data integrity and consistency across the pipeline.

In **Azure SQL Database**, I designed and implemented a **Data Mart** according to the medallion model, which consists of three key layers:

1. **Bronze Layer** (Raw Data): Ingested raw data directly from Blob Storage, preserving its original structure for future reference.
2. **Silver Layer** (Cleansed Data): Transformed and cleaned data, applying necessary data quality checks, business logic, and joins between tables.
3. **Gold Layer** (Aggregated/Optimized Data): Created facts and dimension tables (star schema) for analysis. Key tables include a **Fact Orders** table linked with dimension tables for customers, products, and sellers.

This layered architecture aligns with the Medallion framework, ensuring that raw, clean, and processed data are separated, allowing data to move through incremental stages of transformation. Each stage of the architecture is optimized for performance and supports the extraction of meaningful insights.

After the data was loaded into the Azure SQL Database, I developed a **stored procedure** to further process the aggregated data. This stored procedure performs post-load data transformations and calculations, including summary metrics such as total sales and customer order counts, making the data ready for analysis and reporting.

The Azure-based architecture allows for easy scaling to handle growing data volumes or additional data sources. By leveraging Azure's elasticity, the system can dynamically allocate resources based on processing needs, ensuring efficiency and cost-effectiveness. The use of Azure Data Factory also simplifies the integration of additional data pipelines, making this architecture flexible for future business requirements.

2**.1 Data cleansing**

Before loading the data into the Azure SQL Database and Data Warehouse, an initial **data cleansing** process was carried out using **Pandas** in Python. The raw data from the Olist e-commerce datasets contained various issues, such as missing values, inconsistent formats, and redundant records. These issues were resolved through a series of cleansing steps to ensure the data was accurate, consistent, and ready for transformation and analysis in the later stages of the ETL pipeline.

**3. Pipelines**

**3. Pipelines Documentation**

**3.1. Pipeline Create Tables: Data Ingestion with Dynamic Table Creation**

**3.1.1. Purpose:**

This pipeline automates the process of loading data from CSV files stored in Azure Blob Storage into corresponding tables in Azure SQL Database. It dynamically selects the appropriate CSV files and database tables based on configurations stored in a `conf.tablelist` table. The pipeline also uses Azure Data Factory's "auto-create table" feature to automatically generate the necessary table structures.

**3.1.2. Steps:**

A. Retrieve Configuration from `conf.tablelist`:

- A script task is added at the start of the pipeline to query the `conf.tablelist` table in Azure SQL Database.

- This table contains metadata including:

- filename: The exact name of each CSV file to be processed.

- schema name: The schema in the SQL database where the table resides.

- table name: The destination table where the CSV data will be stored.

B. Iterate Over Each File and Table Pair:

- The pipeline implements a For-Each Loop that iterates over the results from the configuration table. For each CSV file listed, the pipeline will execute the data copy process.

C. Data Copy Using Azure Data Factory’s Copy Activity:

- Inside the loop, a Copy Activity is used to transfer data from the CSV files in Blob Storage to the Azure SQL Database.

- Parameters are added to the Copy Activity for dynamic assignment of the table names, ensuring that each file is copied into the appropriate table based on the information from the `conf.tablelist`.

**4. Auto Table Creation:**

- The Auto Create Table option is enabled in the Copy Activity, which allows Azure Data Factory to automatically create the destination tables in the database if they do not already exist. The schema of each table is inferred from the structure of the corresponding CSV file.

**3.1.3. Technology Used:**

- Azure Data Factory (ADF): The main tool for orchestrating and executing the pipeline.

- Azure Blob Storage: Stores the raw CSV files.

- Azure SQL Database: Serves as the destination for data, with the `conf.tablelist` providing the mapping between the source files and target tables.

**3.1.4. Input**

- conf.tablelist table in Azure SQL Database:

- Columns:

- filename (e.g., `olist\_customers\_dataset.csv`)

- schema\_name (e.g., `dbo`)

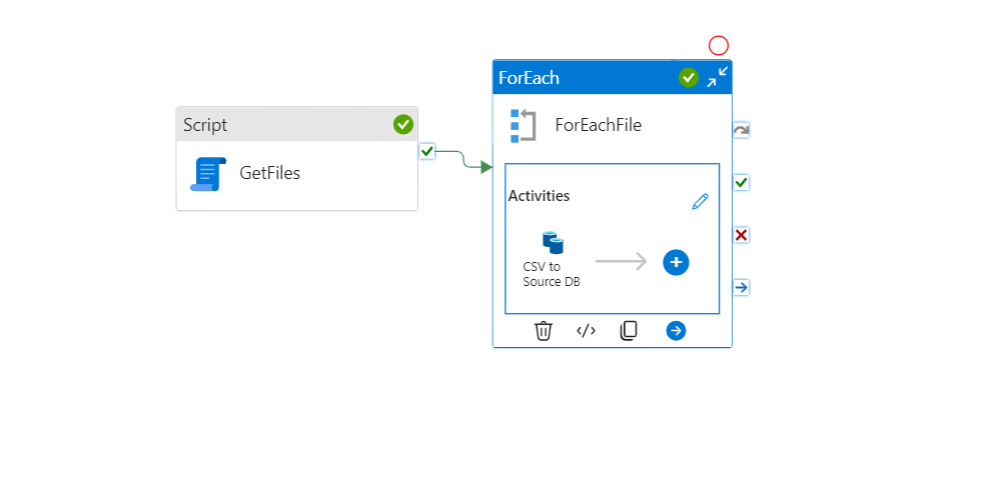
- table\_name (e.g., `customers`)

- CSV files in Azure Blob Storage, containing raw e-commerce data.

**3.1.5. Output**

- Data loaded into Azure SQL Database. Each file is copied into a dynamically created table as specified in the `conf.tablelist`.

After the tables are created from the csv files, the columns of each table are created in varchar data type. I have altered each of the columns using the appropriate data types based on their values.



**3.2. Pipeline 2: Import Data to Database with Table Truncation**

**3.2.1. Purpose:**

* The purpose of this pipeline is to import data from CSV files stored in Azure Blob Storage into pre-existing tables in Azure SQL Database. Unlike the first pipeline, which creates tables dynamically, this pipeline assumes that the tables already exist in the database. Each time the pipeline runs, it truncates the existing tables and then refills them with data from the corresponding CSV files. This ensures that the database always contains the most up-to-date data, while preserving the table structures.

**3.2.2. Steps:**

1. **Retrieve Configuration from conf.tablelist:**
   * Similar to the first pipeline, a **script task** retrieves data from the conf.tablelist table, which contains the mapping between the CSV files and the corresponding tables in Azure SQL Database.
   * The table includes columns for:
     + **filename**: The name of the CSV file.
     + **schema name**: The schema of the database table.
     + **table name**: The name of the destination table.
2. **Iterate Over Each File and Table Pair:**
   * A **For-Each Loop** is used to iterate over each entry in the conf.tablelist. For each file, the pipeline identifies the corresponding table in the database where the data will be imported.
3. **Truncate Table:**
   * Before importing new data, the pipeline performs a **Truncate Table** operation to remove all existing records from the database table. This ensures that no duplicate or outdated data remains from previous pipeline runs.
4. **Data Copy Using Azure Data Factory’s Copy Activity:**
   * After truncating the table, a **Copy Activity** is executed to import data from the CSV files into the truncated tables.
   * Parameters from the conf.tablelist are used to ensure that the correct CSV file is loaded into its corresponding table. This allows the pipeline to dynamically load data without hardcoding table names or file paths.

**3.2.3. Technology Used:**

* **Azure Data Factory (ADF)**: To orchestrate the pipeline, including truncating tables and copying data.
* **Azure Blob Storage**: Stores the source CSV files.
* **Azure SQL Database**: The destination for data, with existing tables where the data is loaded after truncation.

**3.2.4. Input:**

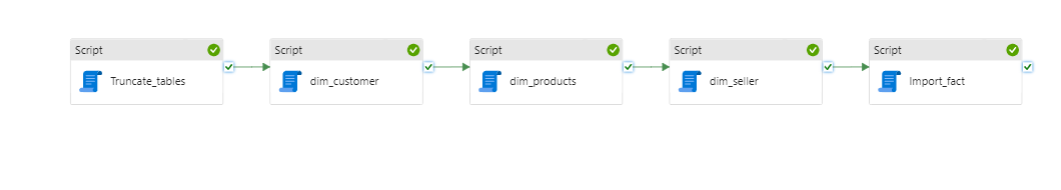
* **conf.tablelist table** in Azure SQL Database:
  + Contains columns:
    - filename (e.g., olist\_customers\_dataset.csv)
    - schema\_name (e.g., dbo)
    - table\_name (e.g., customers)
* **CSV files** in Azure Blob Storage.

**3.2.5. Output:**

* Existing tables in Azure SQL Database are truncated and then refilled with the data from the corresponding CSV files.

**3.2.6. Table Update Strategy:**

* The truncation step ensures that the tables are always refreshed with new data upon every pipeline execution. This is useful in scenarios where data is updated frequently and the system needs to reflect the latest state without duplicating records.



**3.3. Pipeline 3: Import Data to Data Warehouse**

**3.3.1. Purpose:**

* This pipeline is responsible for loading data into the Data Warehouse (DWH) by creating and populating the dimension tables and the fact table. It supports the star schema model, including **dim\_customer**, **dim\_products**, **dim\_sellers**, and a **fact\_orders** table. The pipeline ensures that each table is refreshed by truncating its contents before repopulating it with fresh data.

**3.3.2. Steps:**

1. **Truncate Existing Data:**
   * At the start of the pipeline, a **script task** is executed to truncate all dimension and fact tables (i.e., **dim\_customer**, **dim\_products**, **dim\_sellers**, and **fact\_orders**) if they are already populated. This ensures the tables are emptied and ready to be refilled with new, accurate data without duplications.
2. **Create and Populate Dimension Tables:**
   * Following the truncation, a **script task** is executed to create and populate the **dimension tables**:
     + **dim\_customer**: Stores customer-related data such as customer ID, name, and location.
     + **dim\_products**: Contains product-related data including product ID, product name, and category.
     + **dim\_sellers**: Includes seller details like seller ID, seller name, and region.
   * These dimension tables are populated by transforming the raw data from the relevant tables in the Azure SQL Database, ensuring clean and structured data for analytical queries.
3. **Create and Populate Fact Table:**
   * After the dimensions are created, another **script task** is executed to create and populate the **fact\_orders** table. This table serves as the central point of the star schema and contains metrics such as order quantities, total amounts, and links to the foreign keys from the dimension tables (e.g., customer ID, product ID, seller ID).
   * The fact table aggregates transactional data, enabling reporting on sales, orders, and revenue performance.
4. **Schema and Referential Integrity:**
   * The pipeline ensures that each table follows the correct schema and maintains referential integrity between the fact and dimension tables. The foreign key relationships link the dimension tables to the fact table, allowing efficient querying and analysis.

**3.3.3. Technology Used:**

* **Azure Data Factory (ADF)**: Used to orchestrate the pipeline and run SQL script tasks.
* **Azure SQL Database**: Hosts the staging tables, and data transformations required for the DWH.
* **SQL Scripts**: Used for truncating tables and creating dimension and fact tables, ensuring schema consistency and handling large datasets efficiently.

**3.3.4. Input:**

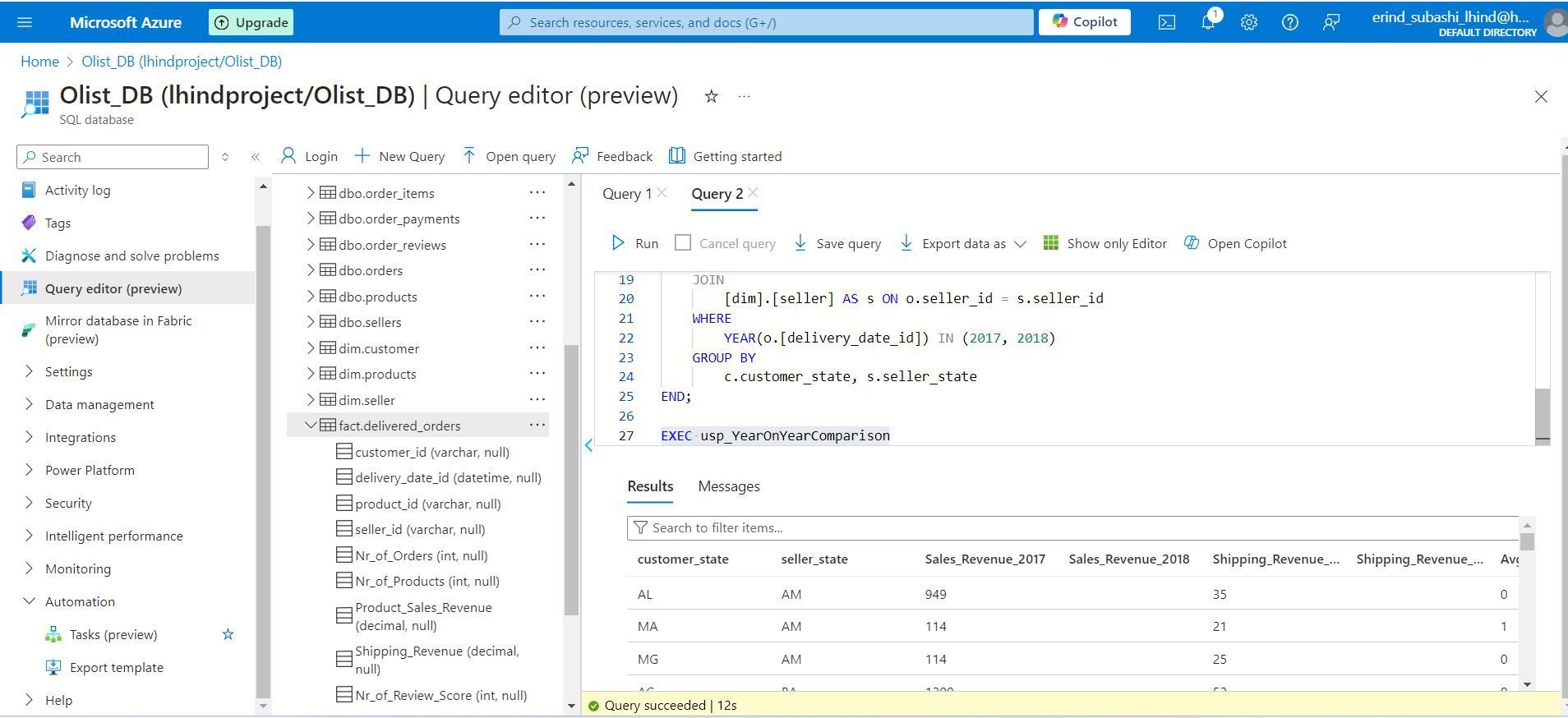
* Data from staging tables in Azure SQL Database, which are populated by prior pipelines (e.g., customer, product, order data).

**3.3.5. Output:**

* **dim\_customer**: Contains cleaned and structured customer data.
* **dim\_products**: Stores product-related information.
* **dim\_sellers**: Holds seller data.
* **fact\_orders**: Central fact table for sales and order metrics.

**3.3.6. Table Update Strategy:**

* The pipeline uses a truncation strategy to ensure the tables are refreshed upon each execution. This guarantees that the Data Warehouse always reflects the most up-to-date and accurate data for reporting and analytics.



**4. Stored Procedure Documentation: usp\_YearOnYearComparison**

**4.1. Purpose:**

The stored procedure usp\_YearOnYearComparison is designed to provide a year-on-year comparison of key metrics related to product sales, shipping revenue, and review scores. The procedure aggregates this data by customer state and seller state, focusing on the years 2017 and 2018. This enables the organization to analyze and compare performance across different regions for both sales and customer satisfaction metrics over time.

**4.2. Key Metrics Calculated:**

* **Sales Revenue Comparison:**
  + Compares total **product sales revenue** in 2017 vs. 2018.
* **Shipping Revenue Comparison:**
  + Compares total **shipping revenue** in 2017 vs. 2018.
* **Average Review Score Comparison:**
  + Compares **average customer review scores** in 2017 vs. 2018.

**4.3. Tables Involved:**

* **[fact].[delivered\_orders]:**
  + Contains data about delivered orders, including product sales revenue, shipping revenue, delivery dates, and review scores.
* **[dim].[customer]:**
  + Contains customer-related data, including the customer’s state.
* **[dim].[sellers]:**
  + Contains seller-related data, including the seller’s state.

**4.4. Logic and Grouping:**

* The procedure calculates the **sum of sales and shipping revenue** and the **average review score** for each year by grouping the results by both **customer state** and **seller state**. This allows for a state-level comparison of the revenue and customer satisfaction metrics.
* **Conditional aggregation** using the CASE statements ensures that revenue and review scores are segregated by year.

**4.5. Execution and Scheduling:**

This stored procedure is scheduled to run **every day at 10:00 AM**. The daily execution ensures that business users and analysts always have up-to-date insights into year-on-year trends for sales revenue, shipping revenue, and customer satisfaction metrics. The daily refresh also accounts for any new data entries that may have been added to the system overnight.