# UpCloud – Data Design Document

The purpose of this document is to elucidate the creation of the data ingestion process, ensuring seamless integration of data into our BigQuery data warehouse.

## Source System:

Billing data files are delivered daily to an S3 bucket. Since these files are accessible via direct download links, they can be programmatically retrieved using a data integration tool or a Python script. Once downloaded, the files can be ingested into the data warehouse for further processing and analysis.  
  
File path to download from S3: <https://f7r4a.upcloudobjects.com/dwh-fina/billing/year=2025/month=04/day=30/billing.csv>

## Data Ingestion Process:

For this specific task, we developed a Python script that dynamically constructs the download URL for the CSV file stored in the S3 bucket. The URL is generated based on the maximum date currently present in our final (fact) table within Google BigQuery. The script downloads the corresponding CSV file and, after adding necessary additional columns, loads the data into a staging table in BigQuery. This loading process uses a truncate-insert approach to ensure the staging table is cleared before each new data load.

In the subsequent step, the data is transformed with appropriate data types and loaded into the final fact table, which serves as the primary source for downstream data analysis.  
  
Currently, the Python function provided in the assignment is in a preliminary/raw state. The plan is to develop two separate Cloud Functions: one dedicated to daily data loads, and another designed for historical data loads, which will accept start\_date and end\_date as input parameters to handle flexible backfills when required. An alternative can be to use integration tools like Apache NIFI or google native ETL tools like cloud dataflow for this purpose.

A screenshot of a computer

AI-generated content may be incorrect.

*Sample snapshot of data load in bigquery*

## ETL Layout:

A screen shot of a computer

AI-generated content may be incorrect.

## DDL:

## I have included the DDL statements for both the staging and fact tables. The data\_loaded field in the staging table serves as a flag to indicate whether the data was successfully loaded. The refresh\_timestamp column records the exact time the data was ingested. Additionally, the billing\_date field is derived by combining the year, month, and day columns from the CSV file to establish a standardized date format.

## Data Aggregations and Visualizations:

The data in the Excel file is captured at an hour-and-minute level of granularity, which is valuable for intraday analysis. However, storing data at this level can be expensive. To optimize for cost and performance, it's important to retain data only at the necessary level of detail. By creating daily, monthly, and yearly aggregates on top of this granular data, we can support effective visualizations while significantly reducing the volume of data scanned, making the overall solution more cost-efficient.

I have developed and executed several analytical queries on the provided dataset. To ensure meaningful analysis, I used the script to load 30 days of data from April. The insights derived from this data are supported by visualizations, which are compiled in the accompanying Excel sheet for reference.

Future Enhancements:  
  
I am listing down some of the enhancements that I would like to do in this process:

1. Initially, we can schedule the data load process using a **CRON** cloud function at a specific time, once we're confident that the file has been delivered to the S3 bucket. Later, to make the process near real-time, we can enhance it by checking the S3 bucket every 5 minutes for the availability of the latest file. Instead of downloading the file each time, we can use functions like list\_objects\_v2 (or equivalent) to check for its presence. If the file is found, we will proceed with the rest of the data processing workflow.
2. As with any robust data pipeline, I prefer to maintain an enhanced logging table that tracks the status of all processes whether they are currently running, have completed successfully, or have failed. This logging mechanism will be crucial in ensuring that once the file for a given day has been successfully loaded from S3, the pipeline does not continue to search for it unnecessarily.
3. To make the pipeline resilient to source system issues, we can implement a separate historical load function. This function will allow loading data for a specified date range, enabling us to address cases where data might be partially loaded or contain inconsistencies.
4. We can consider migrating the data flow from Python scripts to dedicated data integration tools that offer better debugging and monitoring capabilities. For example, I have experimented with creating a simple workflow in Apache NiFi.
5. Modern BI tools perform exceptionally well with a star schema design. For this example, we can implement a star schema consisting of dimension tables such as User, Resource, Region, Service\_Tier, Operation\_Type, Date, and Time, all linked to a central Fact\_Billing table. This structure enables efficient querying and straightforward integration with BI platforms.  
     
   A screenshot of a computer

   AI-generated content may be incorrect.
6. Introduce DBT as a transformation layer to modularize transformations and aggregations following software engineering best practices. DBT also helps enforce data quality rules, such as ensuring that the credits column does not contain positive values, thereby maintaining data integrity.