## Orchestration of Council Data ETL Process



Council Data ETL Orchestration

### Extraction of the Source data using Data Ingestion Tool

Using a Data Ingestion tool i.e., Extract the data from different sources such as API, Streaming Data, SFTP, SQL DB into Google Cloud Storage.

### Set Up Google Cloud Storage (GCS) for Data Ingestion

Store Our raw datasets (e.g., property\_avg\_price.csv, council datasets) in Google Cloud Storage (GCS) buckets. Each dataset should be organized in directories.

Update the code to read from GCS instead of local file paths.

### Use Google Dataproc for Spark Jobs

**Cluster Management**: Use **Google Cloud Dataproc**, which is a managed Spark and Hadoop service. Dataproc will allow you to run Our PySpark job in a distributed and scalable environment.

**Cluster Setup**: Create a Dataproc cluster through the GCP console / UI.

### Use Biquery for Data Storage and Analytics

**Data Destination**: Instead of writing the ETL results to CSV, write the output directly to

**Google Big Query**, which is a fully managed, serverless, and scalable data warehouse.

**Big Query Connector**: Use the Biquery connector for Spark to write the Data Frames directly into Big Query tables.

Example Code:

spark.conf.set("temporaryGcsBucket", "GCS\_BUCKET")

avg\_price\_df.write \

.format("bigquery") \

.option("table", "iceye.refined.council\_avg\_price") \

.save()

sales\_growth\_df.write \

.format("bigquery") \

.option("table", " iceye.refined.council\_sales\_growth") \

.save()

### Orchestration with Cloud Composer (Apache Airflow)

**Scheduling and Monitoring**: Use **Google Cloud Composer**, a managed Apache Airflow service, to orchestrate Our ETL pipeline.

Define an Airflow DAG to automate the execution of Our Spark job and any subsequent tasks (e.g., loading data to BigQuery, triggering reports).

You can schedule the DAG to run periodically, or on-demand based on new data availability.

**Example Airflow DAG:**

from airflow import DAG

from airflow.providers.google.cloud.operators.dataproc import DataprocSubmitJobOperator

from airflow.utils.dates import days\_ago

from datetime import date, datetime, timedelta

schedule = "30 3 \* \* \*"

default\_args = {

"owner": "sai",

"depends\_on\_past": False,

"start\_date": datetime(2022, 10, 23),

"email": ["svk041994@gmail.com"],

"email\_on\_failure": True,

"email\_on\_retry": False,

"retries": 2,

"retry\_delay": timedelta(minutes=10),

"provide\_context": True,

}

dag = DAG(

dag\_id="etl\_iceye\_pipeline",

default\_args=default\_args,

schedule\_interval=schedule,

render\_template\_as\_native\_obj=True,

catchup=False,

)

spark\_job\_task = DataprocSubmitJobOperator(

task\_id="submit\_spark\_job",

job={

"reference": {"project\_id": "iceye"},

"placement": {"cluster\_name": "iceye\_spark\_cluster"},

"pyspark\_job": {"main\_python\_file\_uri": "gs://iceye\_stg/main.py"},

},

region="eu-west",

dag=dag,

)

### Data Quality Checks and Monitoring

**Validation in Dataflow**: Introduce validation and data quality checks using custom functions that ensure schema integrity and proper data types.

**Logging**: Integrate Google Cloud Logging to capture detailed logs about the pipeline’s performance and errors. Configure logging for critical steps (e.g., before writing data to BigQuery).

**Cloud Monitoring**: Use Cloud Monitoring to track job performance metrics and set up alerts for SLA violations or anomalies.

### Near Real-Time Data Processing

For Near Real time data processing, we need to alter our existing pipeline. If property\_avg\_price.csv is near real-time, consider the following adjustments:

**Streaming Ingestion with Cloud Pub/Sub**: Use **Cloud Pub/Sub** to stream new data into the pipeline. We Can subscribe to data changes and trigger Spark jobs based on these events.

**Streaming with Spark Structured Streaming**: Modify Our PySpark code to handle streaming data using Structured Streaming.

Ex.

streaming\_df = spark.readStream \

.option("header", True) \

.option("schema", schema) \

.csv("gs://iceye\_stg/council\_avg\_price/")

query = streaming\_df.writeStream \

.format("bigquery") \

.option("table", "iceye.refined.council\_avg\_price\_tbl") \

.start()

query.awaitTermination()

### Conclusion

***Q1: What kind of data quality checks would you implement, and how would they be integrated?***

*I would implement schema validation, unique value checks, null checks, and mandatory field checks. These checks can be integrated using Dataflow or PySpark test cases during the ETL process.*

***Q2: How would you design the system to handle larger volumes of data efficiently?***

*The system is designed to handle large data volumes by leveraging serverless solutions like BigQuery for storage and Google Cloud Storage for input/output data, combined with Dataproc for distributed processing.*

***Q3: How would you ensure that the definitions of calculations are available to analysts, for instance, when visualizing data?***

*The data, along with calculation definitions, would be stored in BigQuery, allowing analysts and engineers to easily query and visualize the data using tools like Looker Studio or directly from BigQuery.*