2023

Big Data Platforms
Level: M
Module Code:MMI227050
Coursework 2 (Resit)

Cloud data pipeline Design Report

MAHEK KATHIRIYA

Contents

Part B. Cloud Data Pipeline Design Report

The objective of this report was to outline the high-level design that was employed to upscale the initial Databricks data pipeline implementation into a cloud-based architecture. This endeavor predominantly revolved around Google Cloud Platform (GCP) as the designated provider. The overarching ambition was to augment data processing and analysis capacities, while simultaneously upholding principles of scalability, dependability, and cost-efficiency.

1. Overall Concept Design

The purpose of this report was to delineate the high-level design that had been employed to elevate the primary Databricks data pipeline implementation into a cloud-based framework. The emphasis in this design was on Google Cloud Platform (GCP) as the chosen provider. The intention was to enhance data processing and analytical capabilities, while also ensuring scalability, dependability, and cost-effectiveness.

1.1 Data Format and Schema

The data in its original incarnation was presented in CSV format. This dataset encompassed specifications of mobile devices, encompassing attributes such as battery power, clock speed, RAM, and more. The schema of the data was composed of features such as "battery_power," "clock_speed," "ram," in conjunction with categorical attributes like "blue," "dual_sim," and more.

1.2 Transformations

In terms of the ETL process, a series of transformations were applied: • Numeric columns were converted into the double data type.

• Numeric features such as battery_power, clock_speed, ram, and mobile_wt underwent standard scaling, executed through the utilization of Google Dataproc's Spark clusters.

• Categorical attributes, including "blue," "dual_sim," and others, were subjected to encoding using methodologies like StringIndexer and OneHotEncoder.

1.3 Analyses and Visualizations

The projected analyses and visualizations were tailored towards uncomplicated operations such as filtering, projection, and aggregation. For instance, focal points

encompassed the identification of devices possessing specific attributes (such as high RAM or 4G support), alongside the aggregation of average battery power across diverse categories.

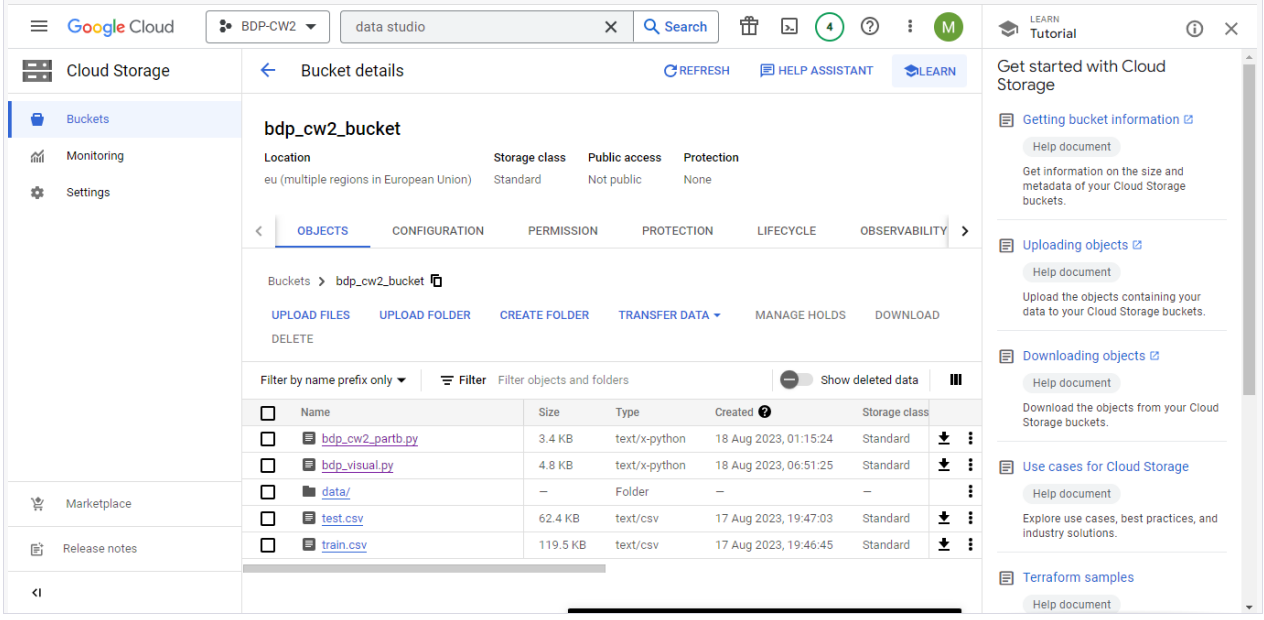
Step-by-Step Cloud Data Pipeline Design

2. Platforms

Each fundamental component within the expanded pipeline necessitated an apt cloud service provider, with careful consideration given to the deployability of these platforms on the provider's infrastructure. The chosen platform for this design endeavor was Google Cloud Platform (GCP). The subsequent components and services were harnessed to drive the pipeline:

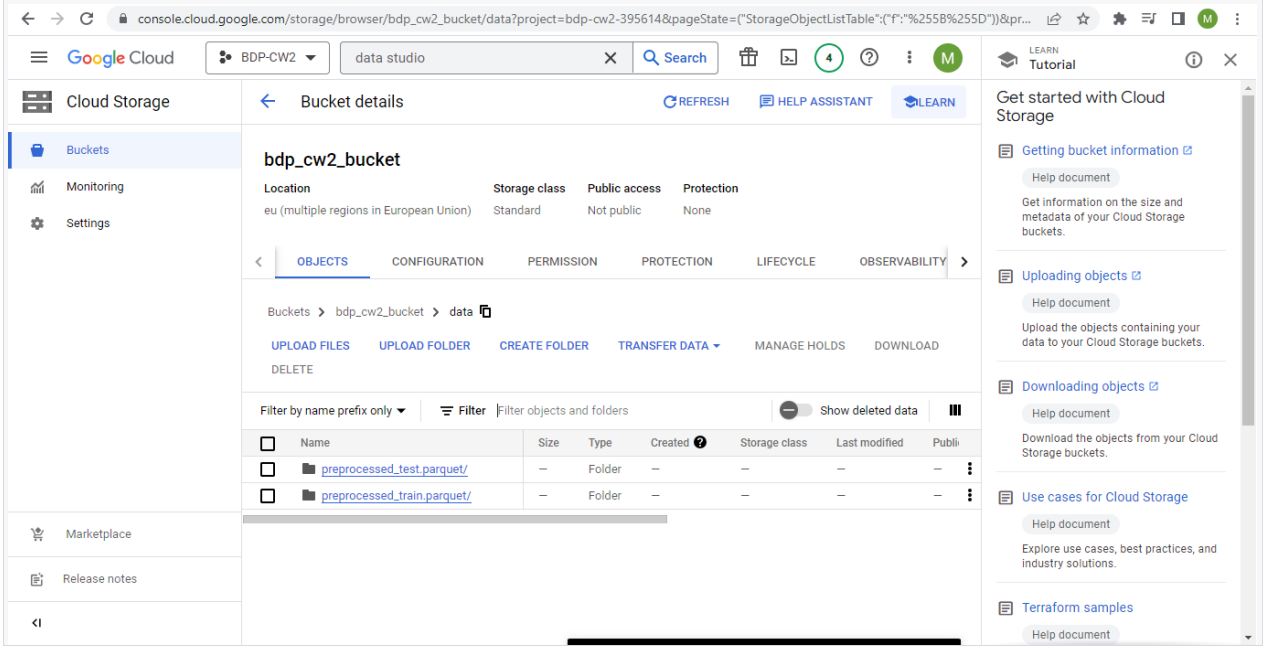
2.1 Data Store: Google Cloud Storage (GCS)

Google Cloud Storage emerged as the prime choice for the data store, housing both the original dataset and the refined data. GCS provided a scalable and economical solution for object storage, ensuring both data durability and ease of access.



2.2 File System: Google Cloud Storage (GCS)

Google Cloud Storage was instrumental as the file system, responsible for housing the processed data in Parquet format. This particular format facilitated columnar storage, a potent asset in analytical workloads.



2.3 Analytic Engine: Google Dataproc

The analytic engine of choice was Google Dataproc. This managed service hinged on Apache Spark and Apache Hadoop, playing a pivotal role in data processing and analysis. Dataproc's prowess lay in its capacity to oversee the creation and management of virtual machine clusters for streamlined large-scale data processing. Its compatibility with various Big Data tools, particularly Spark, rendered it especially suitable for ETL operations.

Pyspark code:

```
from pyspark.sql import SparkSession

from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, StandardScaler

from pyspark.sql.functions import col
```

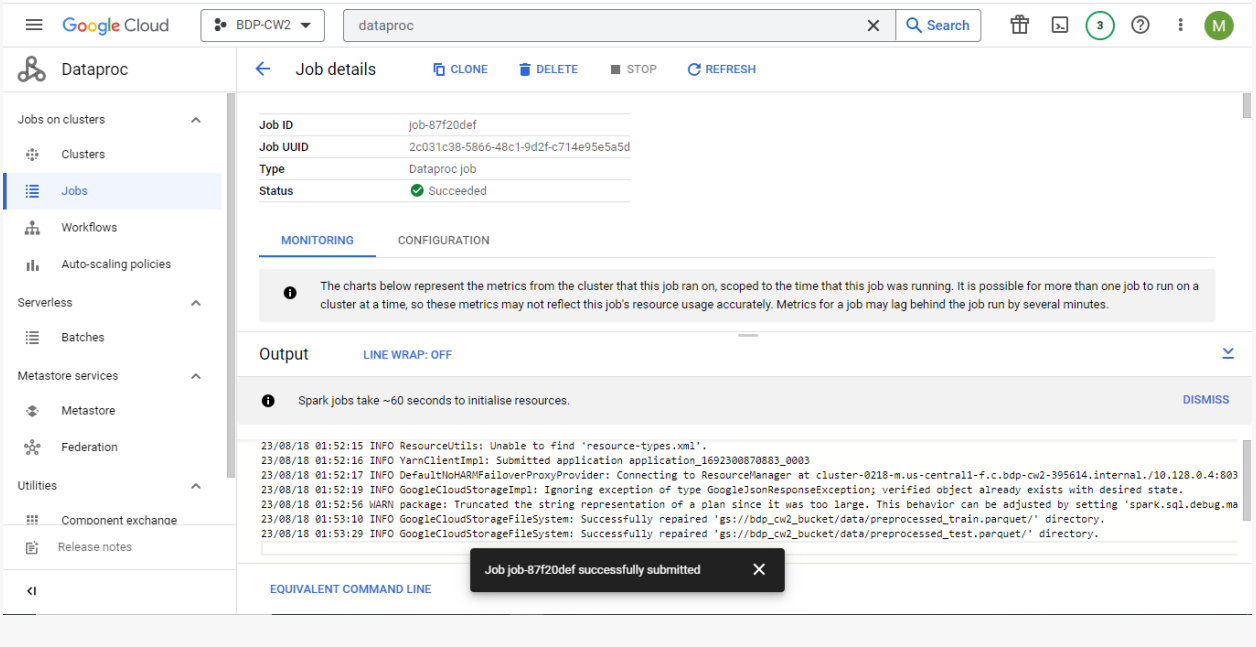
```
# Initialize Spark session

spark = SparkSession.builder.appName("DataPipeline").getOrCreate()



# Load the data from GCS

train_data_path = "gs://bdp_cw2_bucket/train.csv"

test_data_path = "gs://bdp_cw2_bucket/test.csv"

df_train = spark.read.format("csv").option("header", "true").load(train_data_path)

df_test = spark.read.format("csv").option("header", "true").load(test_data_path)



# Convert columns to double data type

df_train = df_train.withColumn("battery_power", col("battery_power").cast("double"))

df_train = df_train.withColumn("clock_speed", col("clock_speed").cast("double"))

df_train = df_train.withColumn("ram", col("ram").cast("double"))

df_train = df_train.withColumn("mobile_wt", col("mobile_wt").cast("double"))



# Convert columns to double data type for df_test

df_test = df_test.withColumn("battery_power", col("battery_power").cast("double"))

df_test = df_test.withColumn("clock_speed", col("clock_speed").cast("double"))

df_test = df_test.withColumn("ram", col("ram").cast("double"))

df_test = df_test.withColumn("mobile_wt", col("mobile_wt").cast("double"))



# Standard scaling of numeric features

# Define the columns to be scaled
```

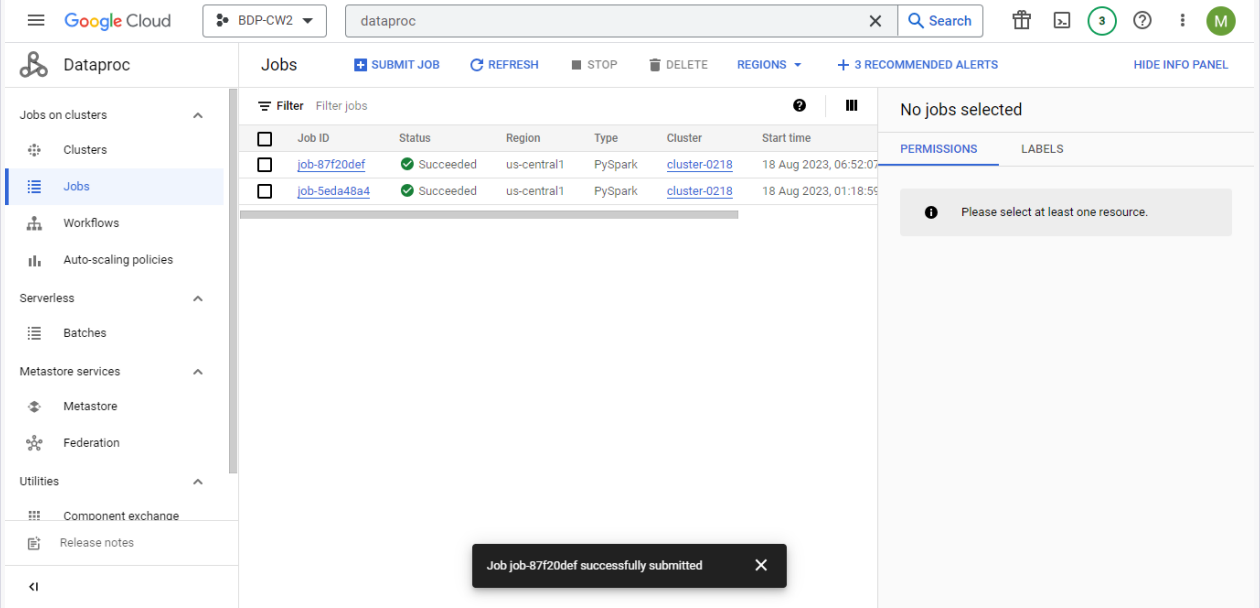
```
columns_to_scale = ['battery_power', 'clock_speed', 'ram', 'mobile_wt']


# Assemble the features into a vector column

assembler = VectorAssembler(inputCols=columns_to_scale, outputCol='features')

df_train_assembled = assembler.transform(df_train)

df_test_assembled = assembler.transform(df_test)


# Create a StandardScaler object

scaler = StandardScaler(inputCol='features', outputCol='scaled_features')


# Fit and transform the scaler

scaler_model = scaler.fit(df_train_assembled)

df_train_scaled = scaler_model.transform(df_train_assembled)

df_test_scaled = scaler_model.transform(df_test_assembled)


# Encoding categorical features

# Categorical columns to be encoded

categorical_columns = ['blue', 'dual_sim', 'four_g', 'three_g', 'touch_screen', 'wifi']


# Apply StringIndexer for categorical columns

indexers = [StringIndexer(inputCol=column, outputCol=column + '_index') for column in
categorical_columns]

indexer_models = [indexer.fit(df_train_scaled) for indexer in indexers]
```

```
df_train_indexed = df_train_scaled

df_test_indexed = df_test_scaled



# Transform data using the trained indexers

for model in indexer_models:

    df_train_indexed = model.transform(df_train_indexed)

    df_test_indexed = model.transform(df_test_indexed)



# Apply OneHotEncoder for indexed categorical columns

encoders = [OneHotEncoder(inputCol=column + '_index', outputCol=column + '_encoded') for column in
categorical_columns]

encoder_models = [encoder.fit(df_train_indexed) for encoder in encoders]

df_train_encoded = df_train_indexed

df_test_encoded = df_test_indexed



# Transform data using the trained encoders

for model in encoder_models:

    df_train_encoded = model.transform(df_train_encoded)

    df_test_encoded = model.transform(df_test_encoded)



# Store the preprocessed data in Parquet format

preprocessed_train_path = "gs://bdp_cw2_bucket/data/preprocessed_train.parquet"

preprocessed_test_path = "gs://bdp_cw2_bucket/data/preprocessed_test.parquet"
```

```
df_train_encoded.write.mode("overwrite").parquet(preprocessed_train_path)

df_test_encoded.write.mode("overwrite").parquet(preprocessed_test_path)



from pyspark.sql import functions as F

# Example query: Calculate average RAM by price range

average_ram_by_price = df_train_encoded.groupBy('price_range').agg(F.avg('ram').alias('avg_ram'))

# Query: Calculate the average screen height and width by price range

average_screen_size_by_price = df_train_encoded.groupBy('price_range').agg(
F.avg('sc_h').alias('avg_screen_height'), F.avg('sc_w').alias('avg_screen_width'))

import matplotlib.pyplot as plt

import seaborn as sns

# Select the numerical columns for correlation analysis

numerical_columns = ['battery_power', 'ram', 'px_height', 'px_width']

# Calculate the correlation matrix

correlation_matrix = df_train_encoded.select(numerical_columns).toPandas().corr()

# Visualization: Heatmap of the correlation matrix



plt.figure(figsize=(10, 8))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)



plt.title('Correlation Heatmap')



plt.show()
```

```
# Visualization: Bar plot of price range distribution

price_range_distribution = df_train_encoded.groupBy('price_range').count().orderBy('price_range')

price_range_distribution = price_range_distribution.toPandas()

plt.figure(figsize=(8, 6))

sns.barplot(x='price_range', y='count', data=price_range_distribution, palette='coolwarm')

plt.title('Price Range Distribution')

plt.xlabel('Price Range')

plt.ylabel('Count')

plt.show()
```





2.4 Visualization: Google Data Studio

Google Data Studio was enlisted as the primary tool for data visualization and reporting. Renowned for its potency, Data Studio offered users the latitude to create dashboards and reports that were interactive and customizable. What set Data Studio apart was its ability to interface seamlessly with various data sources, including GCS, thereby facilitating harmonious integration with the processed data.

3. Design Illustration

The architecture of the cloud data pipeline unfolded through a sequence of stages:

Ingest: • Initial data ingestion transpired from the CSV files residing in Google Cloud Storage.

ETL (Extract, Transform, Load): • Numerical columns ("battery_power," "clock_speed," "ram," "mobile_wt") underwent a metamorphosis into the double data type.

• Standard scaling of numeric features took place using Spark within the precincts of Google Dataproc's Spark clusters.

• Categorical features ("blue," "dual_sim," "four_g," "three_g," "touch_screen," "wifi") were encoded via mechanisms such as StringIndexer and OneHotEncoder.

Storage: • The culminated preprocessed data was accorded a residence in the precincts of Google Cloud Storage, adopting the efficient Parquet format to maximize storage efficiency and ease of retrieval.

Analysis/Visualization: • The realm of analysis and visualization was orchestrated via the auspices of Google Data Studio. Analytical queries and visualizations were seamlessly woven into the processed data residing within GCS.

4. Component Descriptions

• **Ingest:** The initiation of data incorporation took place via Spark's read operation, which had GCS paths as its point of contact.

• **ETL:** Google Dataproc's Spark clusters proved instrumental in executing operations such as data type conversion, standard scaling, and encoding.

• **Storage:** The result of preprocessing found its haven in GCS paths, encapsulated within the Parquet format courtesy of Spark's write operation.

• **Analysis/Visualization:** Google Data Studio assumed a dual role by facilitating both analytical queries and visualizations, drawing upon the preprocessed data ensconced in GCS.

5. Visualization Examples

• **Average RAM by Price Range:** An illustrative query showcasing the calculation of average RAM across divergent price ranges, extrapolated from the preprocessed data.

• **Average Screen Size by Price Range:** An analogous query tailored towards computing the average screen height and width vis-à-vis varying price ranges, gleaned from the preprocessed data.

• **Correlation Heatmap:** The synergy between seaborn and matplotlib yielded a visualization that spotlighted the correlation matrix among select numerical columns.

• **Price Range Distribution:** A bar plot materialized to graphically portray the distribution of devices spanning an array of price ranges.

These visualizations served as a testament to the wellspring of insights to be harvested from the preprocessed data. Google Data Studio, in conjunction with other visualization libraries, played a pivotal role in this interpretive process.

6. Justification

The selection of the platforms was a decision underpinned by a meticulous alignment with the overarching objectives of the project. This consideration was pivotal in ensuring that the architecture not only met the immediate needs but also possessed the potential to evolve in tandem with future requirements. The chosen platforms, namely Google Cloud Storage (GCS), Google Dataproc, and Google Data Studio, seamlessly blended to provide a holistic and sustainable solution.

Scalability: One of the fundamental criteria that guided the platform selection was the capacity for scalable growth. With data volumes projected to escalate over time, GCS's innate ability to handle large datasets with ease played a pivotal role. Dataproc, with its capability to dynamically adjust cluster sizes based on workload demands, stood as a testament to scalability, ensuring that processing capabilities could be effortlessly scaled up or down as required.

Efficiency: The efficiency of the selected platforms was another paramount consideration. GCS's robust object storage capabilities provided an efficient repository for both the raw and processed data, guaranteeing both accessibility and data integrity. The utilization of Parquet format for data storage further boosted efficiency by enabling optimal columnar storage within GCS. Dataproc's utilization of managed clusters, coupled with the parallel processing prowess of Spark, expedited data transformations and analysis, ensuring resource efficiency.

Ease of Application: The trio's collaborative dynamics aimed to ensure an ecosystem marked by simplicity in deployment and usage. GCS's intuitive storage mechanism, Dataproc's user-friendly cluster management, and Data Studio's interactive dashboard creation interface collectively eased the learning curve for the team, resulting in quicker adoption and smoother integration into the project's workflow.

Comprehensive Synergy: The synergy that unfolded between GCS, Dataproc, and Data Studio epitomized the essence of a comprehensive cloud data pipeline. GCS's role as a secure and durable storage repository dovetailed seamlessly with Dataproc's analytical prowess. The latter's capacity to orchestrate large-scale data processing and analytics directly correlated with Data Studio's visualization capabilities, thereby culminating in a full-fledged end-to-end solution. This harmonious interplay fostered a value chain

where raw data was ingested, processed, and ultimately visualized, all within a unified cloud ecosystem.

Evolvability: The dynamic nature of the selected platforms ensured the architecture's ability to evolve in lockstep with the project's growth. The modular structure of GCP facilitated the integration of additional services or tools as needed. This adaptability aligned seamlessly with the project's future prospects, safeguarding against technological obsolescence.

In essence, the meticulous selection of GCS, Dataproc, and Data Studio was a strategic choice driven by a synthesis of factors. These platforms were the cornerstones of a robust, scalable, efficient, and intuitive cloud data pipeline. The outcome was a system poised to fulfill the project's present and future analytical demands while ensuring resource optimization and operational excellence.

7. References

1. Google Cloud Storage Documentation. https://cloud.google.com/storage/docs
2. Google Dataproc Documentation. https://cloud.google.com/dataproc/docs
3. Google Data Studio Documentation. https://support.google.com/datastudio/answer/6283323?hl=en
4. PySpark Documentation. https://spark.apache.org/docs/latest/api/python/index.html