**Project: On-Premises Decommissioning and Migration to Google Cloud Platform (GCP)**

**By: Sakina Banu**  
**Date: April 11, 2025**  
**Practical Hands-on Homework Assignment**

**Objective**

The objective of this project is to demonstrate an in-depth understanding of cloud migration strategies by documenting the complete process of decommissioning a legacy on-premises data processing environment and transitioning to a cloud-native infrastructure using Google Cloud Platform (GCP). This assignment simulates a real-world use case in which an organization aims to modernize its data stack to improve scalability, performance, and cost-efficiency.

**Scenario Overview**

In this assignment, I assumed the role of a data engineer working in a mid-sized enterprise that relies heavily on on-prem infrastructure to manage and process data. The existing data environment consists of:

* **Hadoop Distributed File System (HDFS)** for storing large volumes of structured and semi-structured data.
* **Hive** as the primary data warehouse, used for querying and managing tabular data.
* **Hadoop Pig** and **PySpark** for writing data transformation and ETL (Extract, Transform, Load) jobs.

Over time, the limitations of the on-prem infrastructure became evident, including scalability challenges, increased maintenance costs, limited agility, and slower performance. As a result, the organization decided to migrate its infrastructure to GCP, taking advantage of scalable, managed services such as **Google Cloud Storage (GCS)**, **Dataproc**, and **BigQuery**.

**1. Current On-Prem Architecture Overview**

The existing infrastructure is designed around the Hadoop ecosystem. Data is ingested from multiple sources and stored in HDFS. For transformations, the engineering team uses Pig scripts for batch processing and PySpark jobs for distributed real-time processing. Hive is layered on top of HDFS to allow SQL-like queries on the stored data.

* **HDFS** stores data in a distributed manner across physical machines.
* **Pig** handles ETL logic using scripts written in Pig Latin.
* **PySpark** is used for more flexible and complex transformations at scale.
* **Hive** provides a metadata layer and allows analysts to write queries for reporting and analysis.

The components are tightly coupled, making independent scaling difficult. Also, cluster management and hardware maintenance consume valuable IT resources.

**2. Decommissioning Strategy**

The decommissioning process was carefully planned to prevent data loss and minimize operational disruptions.

1. **Data Backup**:
   * All datasets from HDFS were exported and backed up on secure external storage.
   * Hive metastore information was exported using hive-schema-tool.
   * ETL scripts for Pig and PySpark were saved to GitHub and external drives.
2. **Downtime Mitigation**:
   * A phased migration plan was introduced where non-critical jobs were halted first.
   * Business-critical data and jobs were scheduled for migration during low-usage hours.
3. **Job Termination & Rerouting**:
   * Pig and PySpark jobs were paused from their existing job schedulers.
   * Notifications were sent to downstream users who depended on Hive query results.
4. **Communication**:
   * A change management notice was sent to system users.
   * Key stakeholders were informed about timelines, risks, and fallback plans.

**3. Target GCP Architecture**

The target architecture was designed using managed services in GCP to reduce maintenance and improve performance:

* **Google Cloud Storage (GCS)** replaced HDFS, providing scalable and durable object storage.
* **Dataproc** replaced the Hadoop cluster used for Pig and PySpark processing. It allowed running batch and streaming jobs without infrastructure provisioning.
* **BigQuery** replaced Hive as the cloud-native, serverless data warehouse. It supported standard SQL and delivered faster analytical processing.

This modular architecture ensured that each component could scale independently. IAM roles and service accounts were configured to handle security and access control.

1. **Migration Plan**

The migration plan was broken into three main phases, covering both data and job migration.

**Phase 1: Data Transfer to GCS**  
Data stored in HDFS was moved to GCS using gsutil. To maintain folder structure and metadata integrity, checksums and folder hierarchies were validated post-transfer. Large files were uploaded using parallel composite uploads.

**Phase 2: Job Migration to Dataproc**  
Pig scripts and PySpark jobs were moved to Dataproc. Scripts were adapted to use GCS paths instead of HDFS. Dataproc clusters were created with initialization actions to install required libraries. Sample jobs were executed to test output consistency.

**Phase 3: Query Transformation and Warehousing in BigQuery**  
Hive table definitions were converted to BigQuery tables. HiveQL queries were manually rewritten to Standard SQL. BigQuery’s native features like partitioning and clustering were applied for performance tuning. Data was loaded using bq load.

**Tools and Strategies Used:**

* **Schema Conversion**: Hive DDL exported and modified to fit BigQuery schema constraints.
* **Script Adaptation**: File paths, formats, and libraries updated for GCP compatibility.
* **Security Configuration**: IAM roles mapped to GCP resources with principle of least privilege.

**5. Validation and Testing**

To ensure a smooth and reliable migration, thorough testing was performed:

* **Data Validation**:  
  Row counts and column summaries were compared between HDFS and GCS datasets using PySpark.
* **Job Validation**:  
  ETL outputs from Dataproc were compared to on-prem results. Logs were checked for job success/failure states.
* **Query Validation**:  
  Sample Hive queries and equivalent BigQuery queries were executed and compared to ensure consistency.
* **Performance Benchmarking**:  
  BigQuery performed significantly faster than Hive, reducing query times from minutes to seconds in some cases.

**6. Operationalization and Optimization**

After the migration, the GCP environment was prepared for regular production workloads:

* **Automation**:  
  Daily and weekly ETL workflows were automated using **Cloud Composer**, which orchestrated tasks across GCS, Dataproc, and BigQuery.
* **Monitoring**:  
  **Stackdriver (Cloud Monitoring)** was configured to alert on job failures, cluster usage, and cost spikes.
* **Cost Optimization**:  
  Dataproc clusters were configured with **preemptible VMs** to reduce compute costs. Data was stored in **Parquet format** to reduce storage space and query costs. Bucket lifecycle policies were applied to archive or delete temporary data automatically.

**Conclusion**

This project provided practical exposure to enterprise-grade cloud migration. From planning to execution and optimization, I worked hands-on with CLI tools (gsutil, bq), configured IAM roles, adapted transformation jobs, and tested BigQuery queries. The experience strengthened my understanding of scalable data infrastructure and positioned me for real-world data engineering roles.