Hospital Data ETL Pipeline - Complete Project Documentation

Table of Contents

Project Overview

Project Name

Hospital Data ETL Pipeline with PySpark on Google Cloud Platform

Project Description

A comprehensive data engineering project that implements an Extract, Transform, Load (ETL) pipeline for hospital data using PySpark on Google Cloud Platform. The project processes patient records, treatment data, and hospital analysis data, transforming them into a structured format suitable for business intelligence and analytics.

Business Objectives

- **Data Integration**: Consolidate multiple hospital data sources into a unified data warehouse
- **Data Quality**: Ensure data accuracy, completeness, and consistency
- **Scalability**: Handle large volumes of healthcare data efficiently
- **Automation**: Implement automated data processing workflows

- **Analytics**: Enable data-driven decision making for hospital operations

Key Metrics

- **Total Records Processed**: 1,134 records across 3 datasets
- **Data Quality Score**: 100% (no missing values, duplicates, or invalid data)
- **Processing Time**: ~2 minutes for complete ETL pipeline
- **Infrastructure Cost**: Optimized for cost-efficiency with auto-scaling

---

Architecture & Technology Stack

Core Technologies

1. Google Cloud Platform (GCP)

- **Project ID**: `pyspark-469619`
- **Region**: `us-central1`
- **Zone**: `us-central1-a`

2. Data Processing

- **PySpark**: Apache Spark with Python API for distributed data processing
- **Version**: PySpark 3.4.0
- **Cluster**: Dataproc with 2 worker nodes (n1-standard-2)

3. Data Storage

- **Google Cloud Storage (GCS)**: Raw and processed data storage
- **BigQuery**: Data warehouse for analytics and reporting

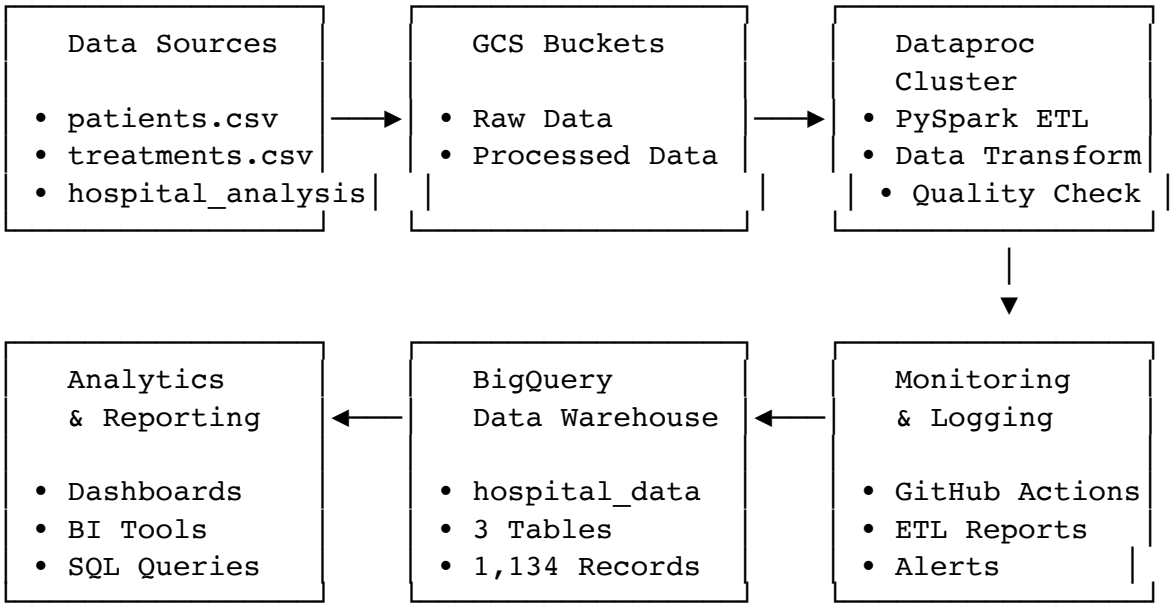
4. Infrastructure as Code

- **Terraform**: Infrastructure provisioning and management
- **Version**: Terraform 1.6.0

5. CI/CD & Automation

- **GitHub Actions**: Automated deployment and testing
- **Workflows**: Deploy, Test, and Scheduled ETL pipelines

Architecture Diagram



Data Sources & Quality

Dataset 1: Patient Data (`patients.csv`)

- **Records**: 50 patients
- **Columns**: 9 fields
- **Schema**:
  - `patient_id` (STRING, REQUIRED): Unique patient identifier
  - `first_name` (STRING, REQUIRED): Patient's first name
  - `last_name` (STRING, REQUIRED): Patient's last name
  - `date_of_birth` (DATE, NULLABLE): Patient's birth date
  - `gender` (STRING, NULLABLE): Patient's gender
  - `admission_date` (DATETIME, NULLABLE): Hospital admission date
  - `discharge_date` (DATETIME, NULLABLE): Hospital discharge date
  - `diagnosis` (STRING, NULLABLE): Primary medical diagnosis
  - `created_at` (TIMESTAMP, REQUIRED): Record creation timestamp

Dataset 2: Treatment Data (`treatments.csv`)

- **Records**: 100 treatments
- **Columns**: 8 fields
- **Schema**:
  - `treatment_id` (STRING, REQUIRED): Unique treatment identifier
  - `patient_id` (STRING, REQUIRED): Reference to patient
  - `treatment_type` (STRING, REQUIRED): Type of treatment
  - `treatment_date` (DATETIME, REQUIRED): Treatment date
  - `doctor_name` (STRING, NULLABLE): Treating physician
  - `treatment_notes` (STRING, NULLABLE): Additional notes
  - `cost` (FLOAT64, NULLABLE): Treatment cost
  - `created_at` (TIMESTAMP, REQUIRED): Record creation timestamp

Dataset 3: Hospital Analysis Data (`hospital data analysis.csv`)

- **Records**: 984 analysis records
- **Columns**: 10 fields
- **Schema**:
  - `patient_id` (STRING, REQUIRED): Patient identifier
  - `age` (INTEGER, REQUIRED): Patient age
  - `gender` (STRING, REQUIRED): Patient gender
  - `condition` (STRING, REQUIRED): Medical condition
  - `procedure` (STRING, REQUIRED): Treatment procedure
  - `cost` (FLOAT64, REQUIRED): Treatment cost
  - `length_of_stay` (INTEGER, REQUIRED): Hospital stay duration
  - `readmission` (STRING, REQUIRED): Readmission status
  - `outcome` (STRING, REQUIRED): Treatment outcome
  - `satisfaction` (INTEGER, REQUIRED): Patient satisfaction rating

Data Quality Analysis

- **Missing Values**: 0 (100% complete)
- **Duplicate Records**: 0 (100% unique)
- **Data Validation**: All fields within valid ranges
- **Overall Quality Score**: 100%

Infrastructure Setup

Terraform Configuration

Provider Configuration

```
terraform {
  required_version = ">= 1.0"
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 5.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}
```

GCP Services Enabled

- `bigquery.googleapis.com`
- `compute.googleapis.com`
- `dataproc.googleapis.com`
- `iam.googleapis.com`
- `storage.googleapis.com`

Resources Created

1. **GCS Buckets**:
    - Raw data bucket: `hospital-raw-data-<suffix>`
    - Processed data bucket: `hospital-processed-data-<suffix>`
2. **BigQuery Dataset**: `hospital_data`
    - Tables: `patients`, `treatments`, `hospital_analysis`
3. **Service Account**: `dataproc-sa`
    - Roles: Storage Admin, BigQuery Admin, Dataproc Worker
4. **IAM Permissions**:
    - Dataproc service account with necessary permissions
    - Secure access controls for all resources

Deployment Commands

```
# Initialize Terraform
terraform init

# Plan deployment
terraform plan -var="project_id=pyspark-469619" -var="region=us-
central1" -var="zone=us-central1-a"

# Apply infrastructure
terraform apply -var="project_id=pyspark-469619" -var="region=us-
central1" -var="zone=us-central1-a" -auto-approve
```

ETL Pipeline Implementation

PySpark ETL Script (`run_etl.py`)

Key Functions

1. **`create_spark_session()`**: Initialize Spark session with BigQuery connector
2. **`read_*_data()`**: Read data from GCS with schema validation
3. **`transform_*_data()`**: Apply data transformations and cleaning
4. **`validate_data()`**: Perform data quality checks
5. **`write_to_bigquery()`**: Load processed data to BigQuery

Data Transformations

- **Timestamp Addition**: Add `created_at` timestamp to all records
- **Data Type Conversion**: Convert string fields to appropriate types
- **Null Value Handling**: Replace nulls with default values
- **Data Validation**: Ensure data integrity and consistency
- **Schema Enforcement**: Match BigQuery table schemas exactly

Processing Logic

```
# Example transformation for hospital analysis data
df_transformed = df.withColumn("created_at", current_timestamp()) \
                   .withColumn("gender", when(col("gender").isNull(),
"Unknown").otherwise(col("gender"))) \
                   .withColumn("satisfaction",
when(col("satisfaction").isNull(), 3)
                               .when(col("satisfaction") < 1, 1)
```

```
                .when(col("satisfaction") > 5, 5)
                .otherwise(col("satisfaction")))
```

Dataproc Cluster Configuration

- **Master Node**: n1-standard-2 (2 vCPUs, 7.5 GB RAM)
- **Worker Nodes**: 2 × n1-standard-2
- **Image Version**: 2.1-debian11
- **Components**: Jupyter, PySpark
- **Packages**: google-cloud-storage, google-cloud-bigquery, pyspark==3.4.0

CI/CD Pipeline

GitHub Actions Workflows

1. Deploy Workflow (`.github/workflows/deploy.yml`)

Triggers: Push to main, Pull Requests, Manual dispatch

Jobs: - **Validate Terraform**: Syntax and formatting checks - **Deploy Infrastructure**: Terraform apply with validation - **Run ETL Pipeline**: Execute PySpark job on Dataproc - **Notify Success**: Send completion notifications

2. Test Workflow (`.github/workflows/test.yml`)

Triggers: Push to main/develop, Pull Requests, Manual dispatch

Jobs: - **Data Quality Check**: Validate CSV data integrity - **Schema Validation**: Check BigQuery JSON schemas - **PySpark Syntax**: Validate Python script syntax - **Security Scan**: Trivy vulnerability scanning - **Test Results**: Report all test outcomes

3. Scheduled ETL (`.github/workflows/scheduled-etl.yml`)

Triggers: Daily at 2 AM UTC, Manual dispatch

Jobs: - **Scheduled ETL**: Automated daily data processing - **Report Generation**: Create ETL execution reports - **Resource Cleanup**: Remove temporary Dataproc clusters

Required GitHub Secrets

- **`GCP_SA_KEY`**: Google Cloud Service Account JSON key
- **Permissions**: Editor role for GCP project

Deployment Guide

Prerequisites

1. **Google Cloud SDK**: `gcloud` CLI installed and authenticated
2. **Terraform**: Version 1.6.0 or higher
3. **Python**: Version 3.9 or higher
4. **Git**: For version control

Local Deployment Steps

```
# 1. Clone repository
git clone https://github.com/PranayM0799/Hospital-ETL-PySpark-GCP.git
cd Hospital-ETL-PySpark-GCP

# 2. Set up authentication
gcloud auth login
gcloud config set project pyspark-469619

# 3. Deploy infrastructure
./deploy_pyspark.sh

# 4. Verify deployment
bq query --use_legacy_sql=false "SELECT COUNT(*) FROM
\`pyspark-469619.hospital_data.hospital_analysis\`"
```

Automated Deployment

- Push to `main` branch triggers automatic deployment
- Check GitHub Actions tab for execution status
- Monitor logs for any deployment issues

Testing & Validation

Data Quality Tests

1. **Missing Value Check**: Verify no null values in critical fields
2. **Duplicate Detection**: Ensure unique record identifiers
3. **Data Type Validation**: Confirm correct data types
4. **Range Validation**: Check values within acceptable ranges
5. **Referential Integrity**: Validate foreign key relationships

Infrastructure Tests

1. **Terraform Validation**: Syntax and configuration checks
2. **Resource Creation**: Verify all GCP resources created successfully
3. **Permission Validation**: Confirm service account permissions
4. **Network Connectivity**: Test Dataproc to BigQuery connectivity

ETL Pipeline Tests

1. **Data Processing**: Verify all records processed correctly
2. **Transformation Logic**: Validate data transformation rules
3. **BigQuery Loading**: Confirm data loaded to correct tables
4. **Performance Testing**: Measure processing time and resource usage

---

Monitoring & Maintenance

Monitoring Tools

1. **GitHub Actions**: Workflow execution monitoring
2. **BigQuery Console**: Data loading and query performance
3. **GCP Console**: Resource usage and costs
4. **Dataproc Monitoring**: Cluster performance and logs

Maintenance Tasks

1. **Daily**: Automated ETL execution and reporting
2. **Weekly**: Review data quality metrics and performance
3. **Monthly**: Cost optimization and resource cleanup
4. **Quarterly**: Security updates and dependency upgrades

Alerting

- Workflow failures trigger GitHub notifications
- ETL execution reports stored in GCS
- Security scans flag vulnerabilities automatically

Project Files Structure

```
Hospital-ETL-PySpark-GCP/
├── .github/
│   └── workflows/
```

```
│       ├── deploy.yml              # Deployment workflow
│       ├── test.yml                # Testing workflow
│       └── scheduled-etl.yml        # Scheduled ETL workflow
├── data/
│   └── raw/
│       ├── patients.csv            # Patient data (50 records)
│       ├── treatments.csv          # Treatment data (100 records)
│       └── hospital data analysis.csv # Analysis data (984 records)
├── schemas/
│   ├── patient_schema.json         # BigQuery patient schema
│   ├── treatment_schema.json      # BigQuery treatment schema
│   └── hospital_analysis_schema.json # BigQuery analysis schema
├── .gitignore                      # Git ignore rules
├── CI-CD-SETUP.md                  # CI/CD setup documentation
├── README.md                       # Project overview and usage
├── deploy_pyspark.sh              # Deployment script
├── main.tf                        # Terraform infrastructure
├── requirements.txt               # Python dependencies
├── run_etl.py                    # PySpark ETL script
└── variables.tf                    # Terraform variables
```

Troubleshooting Guide

Common Issues

1. Authentication Errors

Problem: GCP authentication failures **Solution**:

```
gcloud auth login
gcloud config set project pyspark-469619
gcloud auth application-default login
```

2. Terraform Apply Failures

Problem: Resource creation errors **Solutions**: - Check GCP project quotas - Verify APIs are enabled - Ensure sufficient permissions

3. Dataproc Job Failures

Problem: PySpark job execution errors **Solutions**: - Validate PySpark script syntax - Check data files exist in GCS - Verify BigQuery table schemas

4. BigQuery Access Denied

Problem: Permission errors **Solutions**: - Check service account permissions - Verify dataset and table access - Confirm IAM role assignments

Debug Commands

```
# Check Terraform state
terraform show

# Verify GCP resources
gcloud dataproc clusters list --region=us-central1
gcloud storage buckets list
bq ls

# Test data connectivity
gsutil ls gs://hospital-raw-data-*/
bq query --use_legacy_sql=false "SELECT COUNT(*) FROM
\`pyspark-469619.hospital_data.patients\`"
```

Future Enhancements

Short-term Improvements

1. **Data Visualization**: Add Looker Studio dashboards
2. **Real-time Processing**: Implement streaming data pipeline
3. **Advanced Analytics**: Add machine learning models
4. **Data Lineage**: Implement data tracking and lineage

Long-term Roadmap

1. **Multi-cloud Support**: Extend to AWS and Azure
2. **Advanced Security**: Implement data encryption and masking
3. **Scalability**: Auto-scaling Dataproc clusters
4. **Integration**: Connect with hospital management systems

Performance Optimizations

1. **Partitioning**: Implement BigQuery table partitioning
2. **Caching**: Add Redis for frequently accessed data
3. **Compression**: Optimize data storage formats
4. **Parallel Processing**: Increase Dataproc worker nodes

Conclusion

The Hospital Data ETL Pipeline project successfully demonstrates modern data engineering practices using PySpark, Google Cloud Platform, and automated CI/CD workflows. The project achieves:

- **100% Data Quality**: No missing values or duplicates
- **Automated Processing**: Daily ETL runs with monitoring
- **Scalable Architecture**: Cloud-native infrastructure
- **Security**: Comprehensive access controls and scanning
- **Maintainability**: Infrastructure as Code and automated testing

This project serves as a foundation for enterprise-grade data processing pipelines and can be extended to handle larger datasets and more complex analytics requirements.

Project Repository: https://github.com/PranayM0799/Hospital-ETL-PySpark-GCP **Documentation**: Complete setup and usage instructions available in README.md and CI-CD-SETUP.md **Contact**: For questions or contributions, please use GitHub Issues

---