Migrating Amazon Reviews Sentiment Analysis Pipeline to Google Cloud Platform with Infrastructure as Code

Project Overview

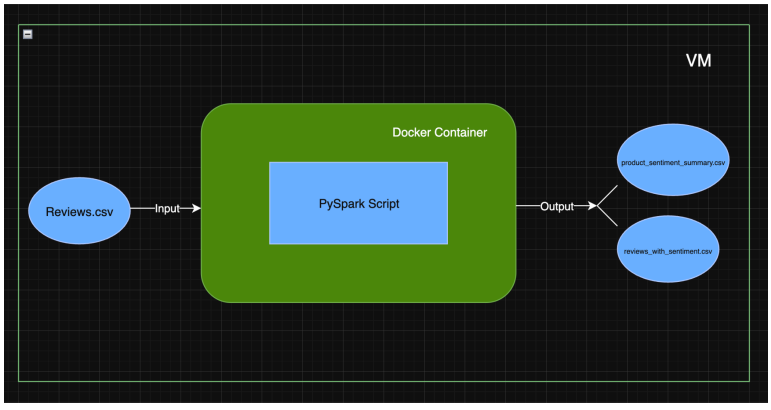
This document provides complete technical documentation for migrating an existing Amazon reviews sentiment analysis pipeline from a simulated on-premises environment to Google Cloud Platform. The project demonstrates modern cloud engineering practices by codifying all infrastructure components using Terraform, automating deployment through Python orchestration script, and implementing production-grade practices including IAM service accounts, autoscaling and resource lifecycle management.

The migration transforms a local Docker-based Spark cluster into a fully managed, scalable, and cost-effective cloud-native solution running on Google Cloud Dataproc with BigQuery as the analytics data warehouse.

Architecture Evolution

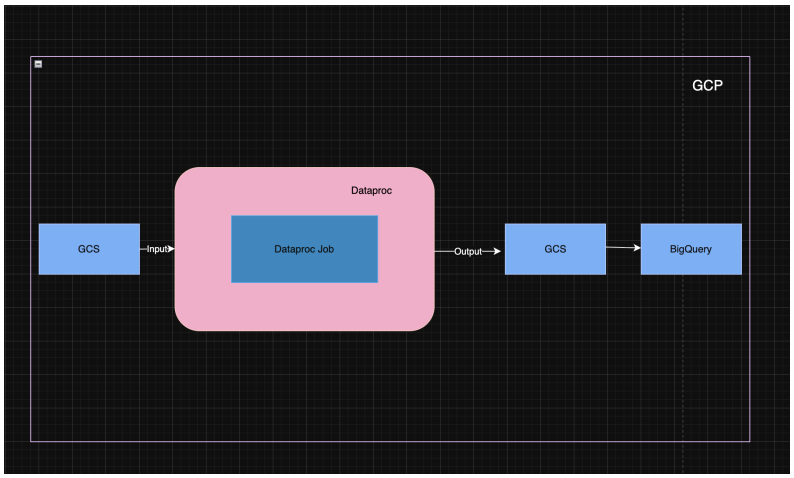
Pre-Migration Architecture

The initial architecture simulates a traditional on-premises data processing environment running on a virtual machine. Docker containers provide the compute infrastructure through a Compose configuration defining one master node and one worker node. The raw dataset containing Amazon food reviews resides in a local directory mounted into the containers. A PySpark script performs sentiment analysis using the NLTK library and generates two output CSV files stored locally.



Post-Migration Architecture

The cloud architecture includes managed Google Cloud services. Raw data and processing scripts are stored in Google Cloud Storage. Google Cloud Dataproc executes sentiment analysis jobs on ephemeral clusters created on demand and deleted after completion. Processed results are written in Parquet format to Cloud Storage, then loaded into BigQuery for analytics. All infrastructure is defined as code using Terraform.



Technology Stack

Pre-Migration Environment

- Docker Desktop creates isolated containers simulating on-premises infrastructure.
- Docker Compose orchestrates the multi-container Spark application.
- Apache Spark 3.5.0 provides the distributed computing framework.
- PySpark serves as the Python API.
- The Natural Language Toolkit handles sentiment analysis through its VADER lexicon.
- Python 3 with pandas and textblob support the pipeline.

Post-Migration Environment

- Terraform manages all Google Cloud resources.
- Google Cloud Storage provides object storage.
- Google Cloud Dataproc offers managed Spark clusters with autoscaling.
- BigQuery serves as the serverless data warehouse.
- The Google Cloud SDK enables command-line interaction.
- Python orchestration scripts coordinate the complete workflow.

Dataset Description

The project processes the Amazon Fine Food Reviews dataset from Kaggle containing 568,454 food reviews. Each record includes identifiers, user information, helpfulness ratings, a one to five star score, timestamp, summary, and full review text. The CSV file is approximately 300 megabytes. Find the data here.

Data Processing Logic

Sentiment Analysis Methodology

The sentiment analysis uses NLTK's VADER Sentiment Intensity Analyzer to compute compound sentiment scores ranging from -1 to +1. Scores are classified into five satisfaction levels: very dissatisfied (at or below -0.7), somewhat dissatisfied (-0.7 to -0.1), neutral (-0.1 to 0.1), somewhat satisfied (0.1 to 0.7), and very satisfied (above 0.7).

Product-Level Aggregation

The pipeline aggregates results at the product level calculating total review count, average sentiment and review scores, percentage without text, distribution across satisfaction levels, and distribution of star ratings.

Phase 1: Pre-Migration Environment Setup

Prerequisites Installation

Update the system and install Docker:

```shell
sudo apt update && sudo apt upgrade -y
sudo apt install -y apt-transport-https ca-certificates curl
software-properties-common
```

Setup Docker repository:

```shell
sudo mkdir -p /etc/apt/keyrings
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

Install Docker and add your user to the docker group:

Shell
```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
sudo usermod -aG docker $USER
```

After running this command, log out and log back in. Start Docker and verify:

Shell
```
sudo systemctl start docker
sudo systemctl enable docker
docker --version
```

Install Terraform:

Shell
```
sudo snap install terraform --classic
terraform --version
```

Project Directory Structure

Create directories for organizing components:

```shell
cd ~
mkdir -p data scripts output docker
ls -la
```

Verify the dataset file:

```shell
du -h ~/data/Reviews.csv
wc -l ~/data/Reviews.csv
head -3 ~/data/Reviews.csv
```

The file should show approximately 300 megabytes and 568,455 lines including the header.

Docker Compose Configuration

Navigate to the docker directory and create the configuration file:

```shell
cd ~/docker
nano docker-compose.yml
```

Copy the complete contents from the provided **docker-compose.yml** file. This configuration defines two services: spark-master and spark-worker. The master runs on ports 8080 and 7077. The worker is configured with 2 CPU cores and 4GB memory. Both containers mount three volumes for data, scripts, and output directories. They connect through a bridge network named spark-network.

PySpark Script for Local Processing

Navigate to the scripts directory and create the sentiment analysis script:

```shell
cd ~/scripts
nano sentiment_analysis.py
```

Copy the complete contents from the provided **sentiment_analysis.py** file. The script performs these key operations:

- Imports PySpark, NLTK, and logging modules
- Downloads NLTK vader_lexicon data if needed
- Initializes Spark session with legacy time parser policy
- Reads Reviews.csv from /opt/spark-data/ with header and schema inference
- Defines user-defined functions to calculate sentiment scores using VADER
- Classifies sentiment scores into five satisfaction levels
- Applies transformations to add sentiment_score and satisfaction_level columns
- Writes detailed review results to /opt/spark-output/reviews_with_sentiment.csv
- Performs product-level aggregation calculating counts, averages, and percentages
- Writes product summary to /opt/spark-output/product_sentiment_summary.csv
- Logs execution statistics and stops Spark session

Container Dependency Installation

Start the Docker containers:

```shell
cd ~
docker compose -f ~/docker/docker-compose.yml up -d
```

Verify both containers are running:

```shell
docker ps
docker logs spark-master
docker logs spark-worker
```

Install Python dependencies in the spark-master container:

```shell
docker exec -it spark-master bash
apt-get update && apt-get install -y python3-pip
pip3 install nltk textblob
exit
```

Repeat for the spark-worker container:

```shell
docker exec -it spark-worker bash
apt-get update && apt-get install -y python3-pip
pip3 install nltk textblob
exit
```

Job Execution and Validation

Submit the Spark job:

```shell
docker exec -it spark-master /opt/spark/bin/spark-submit \
  --master spark://spark-master:7077 \
  --deploy-mode client \
  /opt/spark-scripts/sentiment_analysis.py
```

The job takes approximately 30 minutes. After completion, verify the outputs:

```shell
ls ~/output/reviews_with_sentiment.csv/
ls ~/output/product_sentiment_summary.csv/
head -10 ~/output/reviews_with_sentiment.csv/part-*.csv
wc -l ~/output/reviews_with_sentiment.csv/part-*.csv
wc -l ~/output/product_sentiment_summary.csv/part-*.csv
```

Phase 2: Cloud Migration with Infrastructure as Code

Terraform Configuration Files

Create a directory for Terraform files:

```shell
cd ~
mkdir terraform
cd terraform
```

Provider Configuration

Create provider.tf:

```shell
nano provider.tf
```

Copy the contents from the provided **provider.tf** file which configures the Google Cloud provider with required version 1.0 or higher and Google provider version approximately 5.0. It references var.project_id and var.region for configuration.

Backend Configuration

Before creating Terraform configuration files, you need to set up a Google Cloud Storage bucket to store Terraform's state file remotely. This enables state locking, version control, and team collaboration.

Create the GCS bucket for Terraform state:

```shell
gsutil mb -l us-central1 gs://YOUR-PROJECT-ID-terraform-state

gsutil versioning set on gs://YOUR-PROJECT-ID-terraform-state
```

The first command creates the bucket in the us-central1 region. The second command enables versioning to keep historical versions of your state file.

Now create the backend configuration file:

```
Shell
nano backend.tf
```

Copy the contents from the provided backend.tf file which configures Terraform to use the GCS bucket for remote state storage. The backend configuration specifies the bucket name and a prefix path where the state file will be stored. This must be configured before running terraform init to ensure state is stored remotely rather than locally.

Variable Definitions

Create variables.tf:

```
Shell
nano variables.tf
```

Copy the contents from the provided **variables.tf** file which defines three variables: project_id (required string), region (string with default us-central1), and zone (string with default us-central1-a).

Variable Values

Create terraform.tfvars:

```
Shell
nano terraform.tfvars
```

Copy the contents from the provided **terraform.tfvars** file setting your actual project_id, region, and zone values.

Storage Resources

Create storage.tf:

```
Shell
nano storage.tf
```

Copy the contents from the provided **storage.tf** file which defines two google_storage_bucket resources: staging_bucket and output_bucket. Both use US location and enable uniform_bucket_level_access.

BigQuery Resources

Create bigquery.tf:

```
Shell
nano bigquery.tf
```

Copy the contents from the provided **bigquery.tf** file which defines one google_bigquery_dataset resource and two google_bigquery_table resources with complete schemas. The reviews_with_sentiment table has 12 columns including the original review fields plus sentiment_score and satisfaction_level. The product_sentiment_summary table has 16 columns with aggregated metrics.

Dataproc Resources

Create dataproc.tf:

```
Shell
nano dataproc.tf
```

Copy the contents from the provided **dataproc.tf** file which defines a google_dataproc_autoscaling_policy resource. The policy configures worker instances between 2 and 10 with YARN-based autoscaling parameters.

IAM Resources

Create iam.tf:

```
Shell
nano iam.tf
```

Copy the contents from the provided **iam.tf** file which defines one google_service_account resource and four IAM binding resources granting dataproc.worker role, storage.objectViewer on staging bucket, storage.objectAdmin on output bucket, and bigquery.dataEditor on the dataset.

Output Definitions

Create outputs.tf:

```shell
nano outputs.tf
```

Copy the contents from the provided **outputs.tf** file which defines outputs for bucket names and URLs, BigQuery dataset and table IDs, autoscaling policy information, and service account email.

Authentication Setup

Authenticate with Google Cloud:

```shell
gcloud auth login
gcloud auth list
gcloud config set project YOUR_PROJECT
gcloud auth application-default login
```

Replace `YOUR_PROJECT` with your actual project ID. Follow the browser authentication flows when prompted.

Infrastructure Deployment

Initialize Terraform:

```shell
terraform init
```

Review planned changes:

```shell
terraform plan
```

Apply the configuration to create all resources:

```shell
terraform apply
```

Type yes when prompted. This creates two storage buckets, one BigQuery dataset, two BigQuery tables, one autoscaling policy, one service account, and four IAM bindings.

View the created outputs:

```shell
terraform output
terraform output -raw service_account_email
terraform output -raw staging_bucket_url
```

Verify resources:

```shell
gcloud storage buckets list | grep sentiment
bq ls sentiment_analysis
gcloud iam service-accounts list | grep dataproc
```

Cluster Initialization Script

Create the initialization script:

```shell
cd ~
nano init_cluster.sh
```

Copy the contents from the provided **init_cluster.sh** file. This script updates apt, installs python3-pip, upgrades pip, installs nltk, textblob, and pandas with specific versions, then downloads NLTK vader_lexicon, punkt, and averaged_perceptron_tagger datasets.

Modified PySpark Script for Cloud

Create the cloud-optimized script:

```shell
cd ~/scripts
nano sentiment_analysis_parquet.py
```

Copy the contents from the provided **sentiment_analysis_parquet.py** file. The key difference from the local version is writing in Parquet format instead of CSV.

Configuration File

Create the configuration file:

```shell
cd ~
nano config.json
```

Define region, zone, bucket names, dataset ID, and cluster configuration including machine types, disk sizes, number of workers, image version, and max idle time in **config.json**.

Python Orchestration Script

Create the orchestration script:

```shell
cd ~
nano run_pipeline.py
```

Copy the contents from the provided run_pipeline.py file. This script orchestrates the complete workflow:

- Loads configuration from config.json
- Generates unique cluster name with timestamp
- Applies Terraform to ensure infrastructure is current
- Retrieves service account email from Terraform outputs
- Uploads scripts and data to Cloud Storage
- Creates Dataproc cluster with initialization action and autoscaling policy
- Submits PySpark job to the cluster
- Deletes the cluster after job completion
- Loads Parquet data into BigQuery tables
-

Make the script executable:

```shell
chmod +x run_pipeline.py
```

Pipeline Execution

Execute the complete pipeline:

```shell
python3 run_pipeline.py
```

The script executes each step with detailed logging. The pipeline takes approximately 15 to 20 minutes. Monitor the console output showing Terraform apply, file uploads, cluster creation, job submission, cluster deletion, and BigQuery loading.

Validation and Verification

- Check uploaded files in Cloud Storage
- Query BigQuery tables
- Find products with highest positive sentiment
- Verify cluster was deleted

File Organization

Local Project Structure

```
~/
├── config.json
├── init_cluster.sh
├── run_pipeline.py
├── phase1.sh
├── data/
│   └── Reviews.csv
├── scripts/
│   ├── sentiment_analysis.py
│   └── sentiment_analysis_parquet.py
```

```
├── output/
│   ├── reviews_with_sentiment.csv/
│   └── product_sentiment_summary.csv/
├── docker/
│   └── docker-compose.yml
└── terraform/
    ├── provider.tf
    ├── backend.tf
    ├── variables.tf
    ├── terraform.tfvars
    ├── storage.tf
    ├── bigquery.tf
    ├── dataproc.tf
    ├── iam.tf
    └── outputs.tf
```

Cloud Storage Organization

None

```
gs://YOUR-PROJECT-ID-sentiment-staging/
├── scripts/
│   ├── init_cluster.sh
│   └── sentiment_analysis_parquet.py
└── data/
    └── Reviews.csv

gs://YOUR-PROJECT-ID-sentiment-output/
├── reviews_with_sentiment_parquet/
│   └── *.parquet files
└── product_sentiment_summary_parquet/
    └── *.parquet files
```

BigQuery Schema

reviews_with_sentiment table: 12 columns including Id, ProductId, UserId, ProfileName, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, Text, sentiment_score, and satisfaction_level. Approximately 568,454 rows.

product_sentiment_summary table: 16 columns including product_id, total_number_of_reviews, averages, and percentage distributions for satisfaction levels and scores. Approximately 74,000 rows.

Troubleshooting Guide

Common Local Environment Issues

If Docker commands require sudo after adding user to docker group, log out and log back in, or restart Docker:

```Shell
sudo systemctl restart docker
```

If containers fail to start, check logs:

```Shell
docker logs spark-master
docker logs spark-worker
```

If Spark job fails with module not found errors, install dependencies manually:

```Shell
docker exec -it spark-master bash
apt-get update && apt-get install -y python3-pip
pip3 install nltk textblob
python3 -c "import nltk; nltk.download('vader_lexicon')"
exit
```

If job produces no output, verify volume mounts and file paths:

```shell
ls -la ~/data/Reviews.csv
ls -la ~/scripts/sentiment_analysis.py
```

Project Outcomes

The project successfully demonstrates a complete migration from simulated on-premises to cloud-native data processing. The local environment processes 568,454 reviews in 10 to 15 minutes while the cloud environment completes in 5 to 8 minutes with better reliability. All infrastructure is codified enabling reproducible deployments. Service account implementation follows least privilege security principles. Automatic cluster lifecycle management minimizes costs. Parquet format reduces storage costs by approximately 70 percent and improves query performance. BigQuery enables fast SQL analytics. The orchestration script fully automates the workflow. The autoscaling policy ensures cost-efficient resource utilization.

Future Enhancements

Use Composer for Orchestration: Migrate orchestration to Cloud Composer based on Apache Airflow for robust workflow scheduling, dependency management, and retry logic. This would eliminate the need for the VM and provide a fully managed orchestration solution.

BigQuery Optimization: Partition the reviews_with_sentiment table by date or time ranges to improve query performance and reduce costs. Cluster by product_id to optimize joins and filtering. Create materialized views for common aggregations. Implement table expiration policies for automatic data deletion.

Further code and cost optimizations can be applied as well.

Conclusion

This project demonstrates a complete end-to-end migration of a data processing pipeline from on-premises infrastructure to Google Cloud Platform using modern engineering practices. The transformation showcases Infrastructure as Code for managing cloud resources in a reproducible manner. The solution leverages managed services to eliminate operational overhead while improving scalability and cost-effectiveness.

The migration path from local Docker-based processing to cloud-native Dataproc and BigQuery represents a common pattern for organizations modernizing their data infrastructure. The Infrastructure as Code approach ensures the solution remains maintainable, auditable, and evolvable as requirements change over time.