University of Science and Technology of Hanoi

Cloud and Big Data

Project Report

Student ID: 2440057

Student name: Nguyen Nhat Anh

I. Architecture Overview

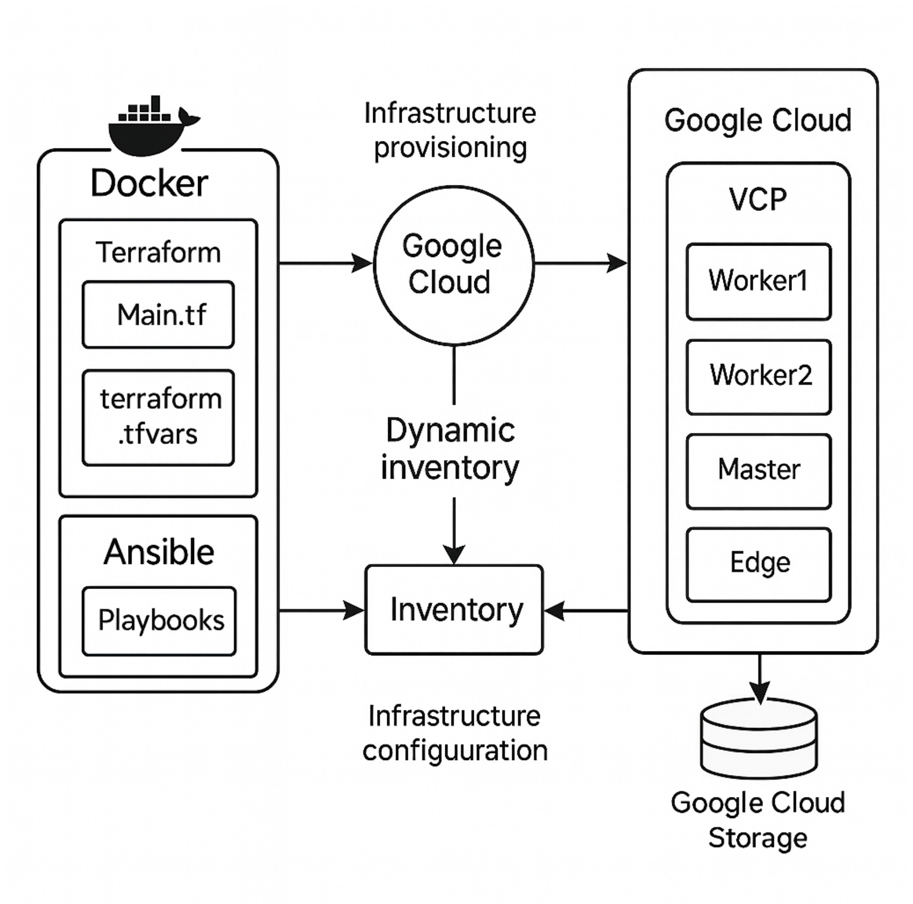

Figure 1: Overall architecture

1. Infrastructure Components

This project automates the deployment of an Apache Spark cluster on Google Cloud Platform (GCP) using Docker, Terraform, and Ansible. Terraform creates the main cloud resources such as the VPC network, firewall rules, and a virtual machine that runs Docker. Ansible installs Docker on this machine and starts all Spark containers.

The Spark cluster runs completely inside Docker:

- **Spark Master**: manages scheduling and coordinates the workers.
- **Spark Workers**: run tasks in parallel.
- **Edge Container**: sends jobs to the cluster and handles data.
- **Google Cloud Storage (GCS)**:
  - Stores datasets and the WordCount JAR file.
  - Provides safe and scalable storage for the cluster.
  - Acts as the main place where the Edge container reads and writes data.

2. Automation Workflow

The workflow works as follows:

1. **Terraform** creates the VPC, firewall rules, and the VM that will host Docker.
2. Terraform outputs are used to build a **dynamic inventory** for Ansible.
3. **Ansible** installs Docker and starts the Spark Master, Worker, and Edge containers.
4. The input file and the WordCount JAR are uploaded to **Google Cloud Storage**.
5. The **Edge container** downloads the data from GCS and sends the WordCount job to the Spark Master.

This automated setup makes the Spark deployment simple, repeatable, and easy to scale. The system is tested by running the WordCount program with different numbers of executors to measure performance.

II. Methodology

1. Terraform Configuration

The Terraform design prioritizes modularity and minimal infrastructure:

- One compute instance used as a **Docker host** for all Spark containers
- GCS bucket automatically created for:
  - storing datasets,
  - storing compiled JAR applications,
  - logs or result output
- Firewall rules restricted using `source_ranges = [var.admin_ip]`
- Service accounts created to provide GCS access for containers through key-less metadata authentication

2. Ansible Roles

Three roles automate the deployment:

- **common**: Installs Docker, configures networking, prepares directories
- **master**: Starts the Spark Master container with published ports
- **worker**: Starts Spark Worker containers and links them to the Master

The dynamic inventory reads Terraform outputs to determine the IP of the Docker host.

3. Security

Security is enhanced across the pipeline through:

- Firewall allowing only necessary ingress traffic
- IAM roles restricting access to GCS buckets
- SSH key authentication to the Docker host
- No exposed Docker APIs; container control is local only

III. Benchmark

At this stage, the benchmark for the WordCount application cannot be completed because the Spark cluster fails to start the job. During execution, Spark returns a `NullPointerException` when initializing the `BlockManager`, as shown below:

```
ERROR SparkContext: Error initializing SparkContext.
java.lang.NullPointerException
    at org.apache.spark.storage.BlockManagerMaster.registerBlockManager
    at org.apache.spark.storage.BlockManager.initialize
    at org.apache.spark.SparkContext.¡init¿(SparkContext.scala:510)
```

This error indicates that the Spark workers and the Spark master were unable to register their block managers with each other. In a Docker-based cluster, this usually occurs when:

- Spark Master and Workers cannot reach each other through the internal Docker network.
- The `SPARK_MASTER_HOST` or `SPARK_MASTER_URL` is misconfigured.
- The master hostname does not match the hostname advertised inside the container.
- Ports required for block manager communication are not open or not mapped correctly.

Because the cluster cannot initialize properly, the WordCount job cannot run and benchmark data cannot be collected at this time.

A network reconfiguration is currently in progress. The Spark containers are being updated so that the master and workers can communicate reliably using the correct Docker bridge network and hostnames. Once the Spark context starts correctly and the workers register successfully, the WordCount benchmark will be executed again.

A placeholder for the benchmark table is shown below and will be updated once the cluster becomes operational:

Total Executor Cores	**Execution Time (s)**
2	—
4	—
6	—

Table 1: Benchmark results will be added once Spark initializes correctly.