Automation of Apache Spark Deployment with Ansible and Terraform on GCP

Phi Doan Minh Luong - 2440046

1. Architecture Overview

1.1 Infrastructure Design

The project implements a fully automated big data environment on Google Cloud Platform, consisting of:

Cluster Components:

- Spark Master Node: Cluster coordinator managing job scheduling and resource allocation
- Spark Worker Nodes (2 instances): Data processing executors handling distributed computations
- Edge Node: Job submission gateway with client tools and application deployment
- Custom VPC Network: Isolated network environment with subnets in the asia-southeast1 region

Technical Specifications:

- Instance Type: e2-medium (2 vCPUs, 4GB RAM each)
- Operating System: Ubuntu 22.04 LTS
- Spark Version: 2.4.3 with Hadoop 2.7.1
- Java Version: JDK 1.8.0_202

1.2 Network Architecture

Internet → GCP Firewall Rules → Custom VPC → Spark Cluster

- Spark Master (Ports: 7077, 8080)
- Spark Workers (Ports: 8081, various)
- Edge Node (SSH, Client Access)

1.3 Security Implementation

- SSH Key Authentication: Secure command-line access
- IAM-based Access Control: Least privilege principle for service accounts
- Custom Firewall Rules: Restricted port access (22, 7077, 8080, 8081)
- Network Segmentation: Isolated VPC with controlled ingress/egress

2. Methodology

2.1 Infrastructure as Code with Terraform

Key Terraform Components:

- Provider Configuration: GCP authentication and project setup
- Resource Definitions: Compute instances, VPC, firewall rules
- Variable Management: Environment-specific configurations
- State Management: Tracking infrastructure state in the remote backend

After the infrastructure deployment, we record the output, which is the IP address of all machines

```
terraform output > ../ansible/terraform_outputs.txt
```

2.2 Configuration Management with Ansible

Phase 1: Base system configuration

- Install system dependencies (common.yml)

Phase 2: Spark Cluster Deployment

- User and group management for Spark services
- Software installation (Java, Spark)
- Environment configuration (JAVA_HOME, SPARK_HOME, PATH)
- Directory structure setup (/var/log/spark, /var/lib/spark/work)

Phase 3: Service Configuration

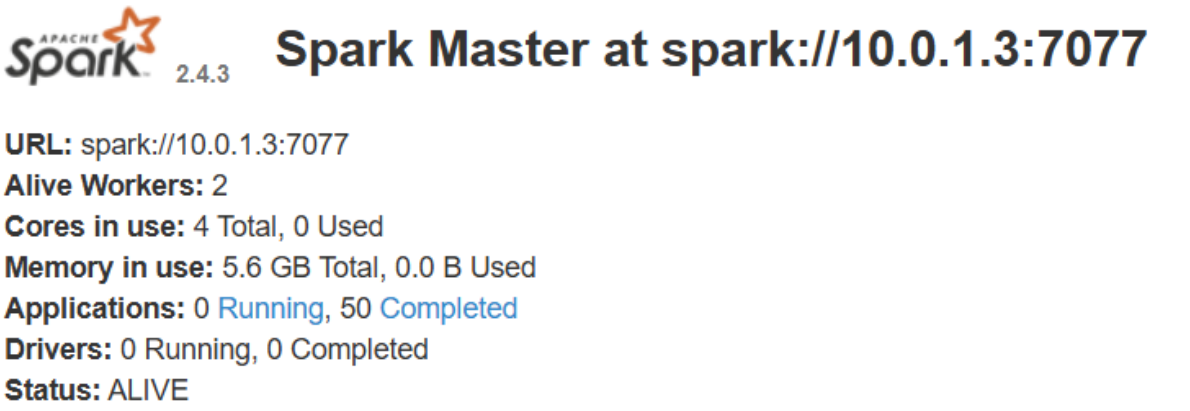
- spark-env.sh configuration with cluster settings
- spark-defaults.conf for application defaults
- Systemd service files for automatic process management
- Slave/worker registration with the master node

3. Testing and Validation

3.1 Cluster health verification

Service status checks:

- Spark Master Web UI (port 8080): Active and responsive
- Worker Registration: All nodes successfully joined the cluster
- Resource Allocation: CPU and memory are properly allocated
- Network Connectivity: Inter-node communication verified



3.2 WordCount Application Testing

Test methodology

The WordCount application was deployed to validate cluster functionality and measure performance across different executor configurations.

Test Data

- Input: Sample text files (1GB)
- Processing: Tokenization, mapping, reduction operations
- Output: Sorted word frequency counts

Performance Results:

Excutors	Data Size	Executor Memory	Execution Time
1	1 GB	2 GB	35s
2	1 GB	2 GB	33s

Performance improved, but not much, probably because the file is not big enough or there are not enough workers to see a noticeable difference

4. Conclusion

4.1 Project achievements

Successfully Implemented:

- Full Automation: Infrastructure to application deployment
- Reproducible Environment: Consistent cluster provisioning
- Performance Validation: Scalability demonstrated through testing
- Security Best Practices: Secure access and network isolation
- Cost Optimization: Automatic teardown prevents resource waste

4.2 Technical

- Terraform + Ansible Integration: Seamless infrastructure-to-application pipeline
- GCP Integration: Native cloud services provided a reliable foundation
- Spark Architecture: Well-designed for distributed processing workloads
- Automation Benefits: Repeatable, consistent deployments with minimal manual intervention