Maylis Martin
ICT Master 2

Rapport: Automation of Apache Spark Deployment with Ansible and Terraform on GCP

1. Introduction

L'objectif de ce projet est de déployer automatiquement un cluster Apache Spark sur Google Cloud Platform (GCP) en utilisant deux outils d'infrastructure-as-code : Terraform pour la création des machines virtuelles et Ansible pour leur configuration.

Le cluster est composé d'un nœud Spark Master, de trois nœuds Spark Workers, et d'un Edge Node utilisé pour la soumission des jobs Spark.

Pour valider le bon fonctionnement du cluster, l'application WordCount a été exécutée, permettant de vérifier la distribution des données et la coordination entre le Master et les Workers.

2. Architecture du système

2.1 Composants du cluster

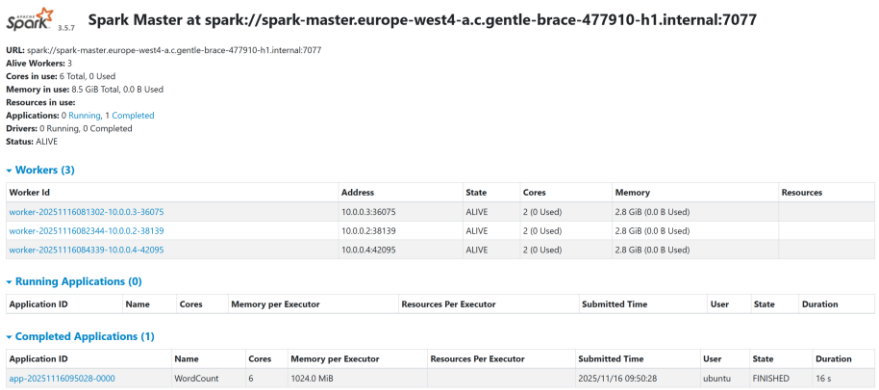
L'infrastructure déployée inclut un VPC personnalisé, un Master Node avec interface Web sur le port 8080, trois Workers s'enregistrant automatiquement auprès du Master, et un Edge Node pour soumettre des jobs.

L'ensemble est configuré pour démarrer automatiquement les services Spark, formant ainsi un cluster fonctionnel et distribué.

2.2 Sécurité

Le déploiement applique les bonnes pratiques suivantes :

- Authentification SSH par clé privée uniquement.
- Contrôle d'accès via IAM pour sécuriser GCP.
- Firewall restreint aux ports nécessaires : SSH (22), Spark Master UI (8080) et port Spark Master (7077).



3. Méthodologie

3.1 Provisioning via Terraform

L'infrastructure du cluster Spark a été déployée automatiquement à l'aide de Terraform. La configuration a permis la création d'un VPC personnalisé avec un sous-réseau dédié, assurant un réseau isolé pour le cluster. Les règles de firewall ont été appliquées afin de sécuriser les communications, en n'autorisant que le trafic nécessaire pour SSH, l'interface Spark Master et le port Spark principal.

Les instances Compute Engine ont été provisionnées pour le Master et deux Workers. Terraform a comparé la configuration déclarée avec l'état réel de l'infrastructure et a confirmé que tout correspondait, ce qui a permis de valider que les ressources étaient correctement déployées et prêtes à l'usage.

L'ensemble des ressources créées est visible via :
terraform state list

```
root@7505ef04724c:/# cd /root/spark-automation/terraform
root@7505ef04724c:~/spark-automation/terraform# terraform state list
google_compute_firewall.spark_firewall
google_compute_instance.spark_master
google_compute_instance.spark_workers[0]
google_compute_instance.spark_workers[1]
google_compute_network.spark_vpc
google_compute_subnetwork.spark_subnet
```

Cela confirme que :

- La création du réseau, du firewall, du subnet, du master et des workers a bien été prise en compte dans l'état Terraform.

3.1.1 Application de l'infrastructure

Une fois les fichiers Terraform définis, la commande suivante a été exécutée :
terraform apply

```
root@7505ef04724c:~/spark-automation/terraform# terraform apply
google_compute_network.spark_vpc: Refreshing state... [id=projects/gentl
e-brace-477910-h1/global/networks/spark-vpc]
google_compute_firewall.spark_firewall: Refreshing state... [id=projects
/gentle-brace-477910-h1/global/firewalls/spark-firewall]
google_compute_subnetwork.spark_subnet: Refreshing state... [id=projects
/gentle-brace-477910-h1/regions/europe-west4/subnetworks/spark-subnet]
google_compute_instance.spark_workers[0]: Refreshing state... [id=projec
ts/gentle-brace-477910-h1/zones/europe-west4-a/instances/spark-worker-1]
google_compute_instance.spark_workers[1]: Refreshing state... [id=projec
ts/gentle-brace-477910-h1/zones/europe-west4-a/instances/spark-worker-2]
google_compute_instance.spark_master: Refreshing state... [id=projects/g
entle-brace-477910-h1/zones/europe-west4-a/instances/spark-master]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your
configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

master_ip = "34.91.73.6"
spark_master_public_ip = "34.91.73.6"
spark_worker_public_ips = [
  "34.158.81.79",
  "34.12.70.81",
]
worker_ips = [
  "34.158.81.79",
  "34.12.70.81",
]
```

Résultat :

-Le cluster existe déjà et correspond exactement à la configuration.
-Aucune modification n'est requise → l'infrastructure est stable.
-Les outputs donnent les IP publiques du master et des workers automatiquement générées par Terraform.

3.2 Configuration via Ansible

Ansible a permis d'automatiser l'installation et la configuration d'Apache Spark sur l'ensemble du cluster, incluant le master, les workers et l'edge node. Le playbook met en place l'environnement nécessaire en installant Java, en créant l'arborescence Spark et en déployant automatiquement Spark 3.x sur chaque machine. Il configure le master, assure l'enregistrement automatique des workers auprès de celui-ci et active les services spark-master et spark-worker. Toutes les commandes Ansible ont été exécutées depuis l'edge node, où est stocké le projet Ansible, garantissant un déploiement reproductible et homogène sur tout le cluster.

Inventaire Ansible

L'inventaire (inventory.ini) définit les rôles et les IP :
cat inventory.ini

```
root@7505ef04724c:~/spark-automation/terraform# cd ~/spark-automation/ansible
root@7505ef04724c:~/spark-automation/ansible# cat inventory.ini
[master]
34.91.73.6

[workers]
34.158.81.79
34.12.70.81

[all:vars]
ansible_user=ubuntu
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

Pour automatiser la configuration des instances, un playbook Ansible nommé spark_setup.yml a été utilisé. Il installe Java et les utilitaires nécessaires, crée le répertoire Spark sur chaque machine, télécharge et décompresse Spark, et met à jour le PATH pour que les commandes Spark soient accessibles globalement.

Avant de lancer le playbook, la connectivité avec tous les nœuds a été vérifiée via la commande :
ansible all -i inventory.ini -m ping

```
root@7505ef04724c:~/spark-automation/ansible# ansible all -i inventory.ini -m ping
[WARNING]: Platform linux on host 34.158.81.79 is using the discovered Python interpreter at /usr/bin/python3.10, but
future installation of another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
34.158.81.79 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.10"
    },
    "changed": false,
    "ping": "pong"
}
[WARNING]: Platform linux on host 34.12.70.81 is using the discovered Python interpreter at /usr/bin/python3.10, but
future installation of another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
34.12.70.81 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.10"
    },
    "changed": false,
    "ping": "pong"
}
[WARNING]: Platform linux on host 34.91.73.6 is using the discovered Python interpreter at /usr/bin/python3.10, but
future installation of another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
34.91.73.6 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.10"
    },
    "changed": false,
    "ping": "pong"
```

Tous les hôtes ont répondu correctement (pong), confirmant que la communication SSH fonctionnait et que les clés privées étaient correctement configurées.

Le playbook a ensuite été exécuté sur tous les nœuds pour préparer le cluster.

nano spark_setup.yml

```
 GNU nano 6.2
---
- hosts: all
  become: yes
  tasks:
    - name: Mettre à jour les paquets
      apt:
        update_cache: yes

    - name: Installer Java et utilitaires
      apt:
        name:
          - openjdk-11-jdk
          - wget
          - tar
        state: present

    - name: Créer le dossier Spark
      file:
        path: /opt/spark
        state: directory

    - name: Télécharger Spark
      get_url:
        url: https://downloads.apache.org/spark/spark-3.5.7/spark-3.5.7-bin-hadoop3.tgz
        dest: /tmp/spark.tgz

    - name: Décompresser Spark
      unarchive:
        src: /tmp/spark.tgz
        dest: /opt/spark
        remote_src: yes
        extra_opts: [--strip-components=1]

    - name: Ajouter Spark au PATH
      lineinfile:
        path: /etc/profile.d/spark.sh
        line: 'export PATH=$PATH:/opt/spark/bin:/opt/spark/sbin'
        create: yes
```

4. Test de validation: WordCount

Script utilisé

Pour vérifier le bon fonctionnement du cluster Spark, nous avons exécuté l'application WordCount depuis le Master Node (spark-master). Le script wordcount.py lit un fichier texte (filesample.txt) et calcule le nombre d'occurrences de chaque mot.

Le code du script est le suivant (capture d'écran du fichier wordcount.py prise avec nano) :

```
 GNU nano 6.2
from pyspark import SparkContext, SparkConf
import sys

if len(sys.argv) != 2:
    print("Usage: wordcount.py <file>")
    sys.exit(-1)

file_path = sys.argv[1]

conf = SparkConf().setAppName("WordCount")
sc = SparkContext(conf=conf)

# Lire le fichier
lines = sc.textFile(file_path)

# Compter les mots
counts = lines.flatMap(lambda line: line.split()) \
              .map(lambda word: (word, 1)) \
              .reduceByKey(lambda a, b: a + b)

# Afficher le résultat
for word, count in counts.collect():
    print(f"{word}: {count}")

sc.stop()
```

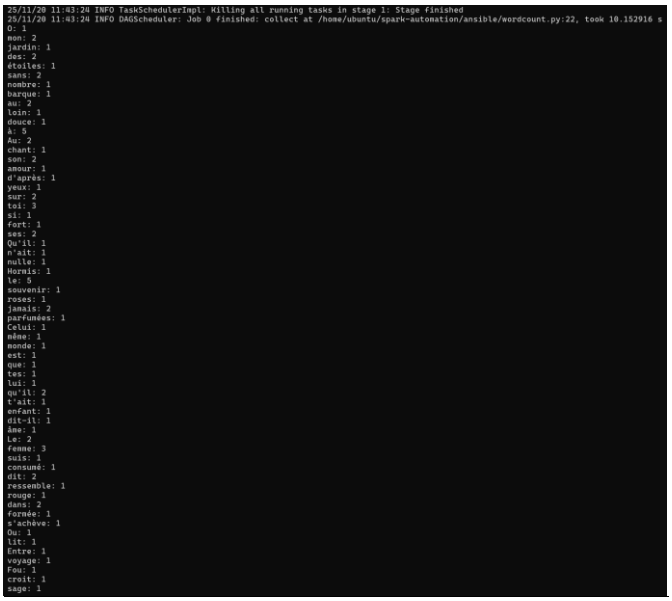
Explication du script **:**

- On initialise un contexte Spark (SparkContext) avec le nom d'application WordCount.
- Le fichier texte est lu ligne par ligne.
- Chaque ligne est découpée en mots, puis on crée des paires (mot, 1) et on additionne les occurrences par mot.

- Le résultat est ensuite affiché dans la console.

Le job a été lancé avec la commande suivante depuis le Master :
/opt/spark/bin/spark-submit --master spark://34.91.73.6:7077 wordcount.py filesample.txt

Sortie obtenue :
Le script a produit le nombre d'occurrences pour chaque mot. Par souci de lisibilité, nous incluons un extrait des résultats(ci-dessous). On remarque le WordCount a pris 10,15sec pour être exécuté.



5. Conclusion

Ce projet a permis de mettre en place un cluster Apache Spark complet sur Google Cloud Platform en automatisant l'intégralité du déploiement grâce à Terraform et Ansible. Terraform a assuré la création cohérente et reproductible de l'infrastructure (réseau, règles firewall, machines virtuelles), tandis qu'Ansible a permis de configurer automatiquement chaque nœud avec les dépendances nécessaires et l'installation de Spark.

L'ensemble des machines a été détecté comme opérationnel par l'interface Spark du nœud Master, validant le bon enregistrement des Workers et le fonctionnement du cluster. L'exécution de l'application WordCount a enfin permis de confirmer que les données étaient distribuées correctement et que le cluster pouvait traiter un job Spark réel.

Le projet remplit ainsi pleinement son objectif : disposer d'un environnement Big Data automatisé, fonctionnel et facilement réutilisable, démontrant la complémentarité des outils IaC et la capacité de Spark à opérer sur une architecture distribuée.