easy Former
La formation à 360°

Ansible

Automatisation du déploiement d'un cluster MariaDB avec Galera via Ansible

Référence : TP-ANSIBLE-3123

Auteur(s) :
Yann BENHAMRON

Destinataire(s) :
Easyformer

Date de modification : 14/02/23

Version : 1

Sommaire

page

1 Introduction

L'automatisation du déploiement d'un cluster MariaDB avec Galera via Ansible est un processus permettant de déployer automatiquement un cluster MariaDB à plusieurs nœuds avec la technologie de réplication synchrone Galera, en utilisant l'outil de gestion de configuration Ansible.

Galera permet de synchroniser les données entre les nœuds du cluster en temps réel, ce qui permet d'assurer la haute disponibilité et la tolérance aux pannes du système.

Ansible est un outil de gestion de configuration et d'orchestration qui permet d'automatiser le déploiement et la gestion des configurations des serveurs. L'utilisation de ces deux technologies ensemble permet de déployer et de gérer efficacement un cluster MariaDB avec Galera.

Le script Ansible pour le déploiement du cluster MariaDB avec Galera est écrit en langage YAML et contient des instructions pour installer MariaDB, Galera, configurer les fichiers de configuration et lancer le cluster. Le script peut être personnalisé en fonction des besoins spécifiques du déploiement et peut être exécuté sur plusieurs nœuds en même temps.

L'automatisation du déploiement d'un cluster MariaDB avec Galera via Ansible permet de gagner du temps et d'assurer la cohérence de la configuration sur l'ensemble du cluster. Elle offre également une solution fiable et reproductible pour le déploiement de clusters MariaDB avec Galera.

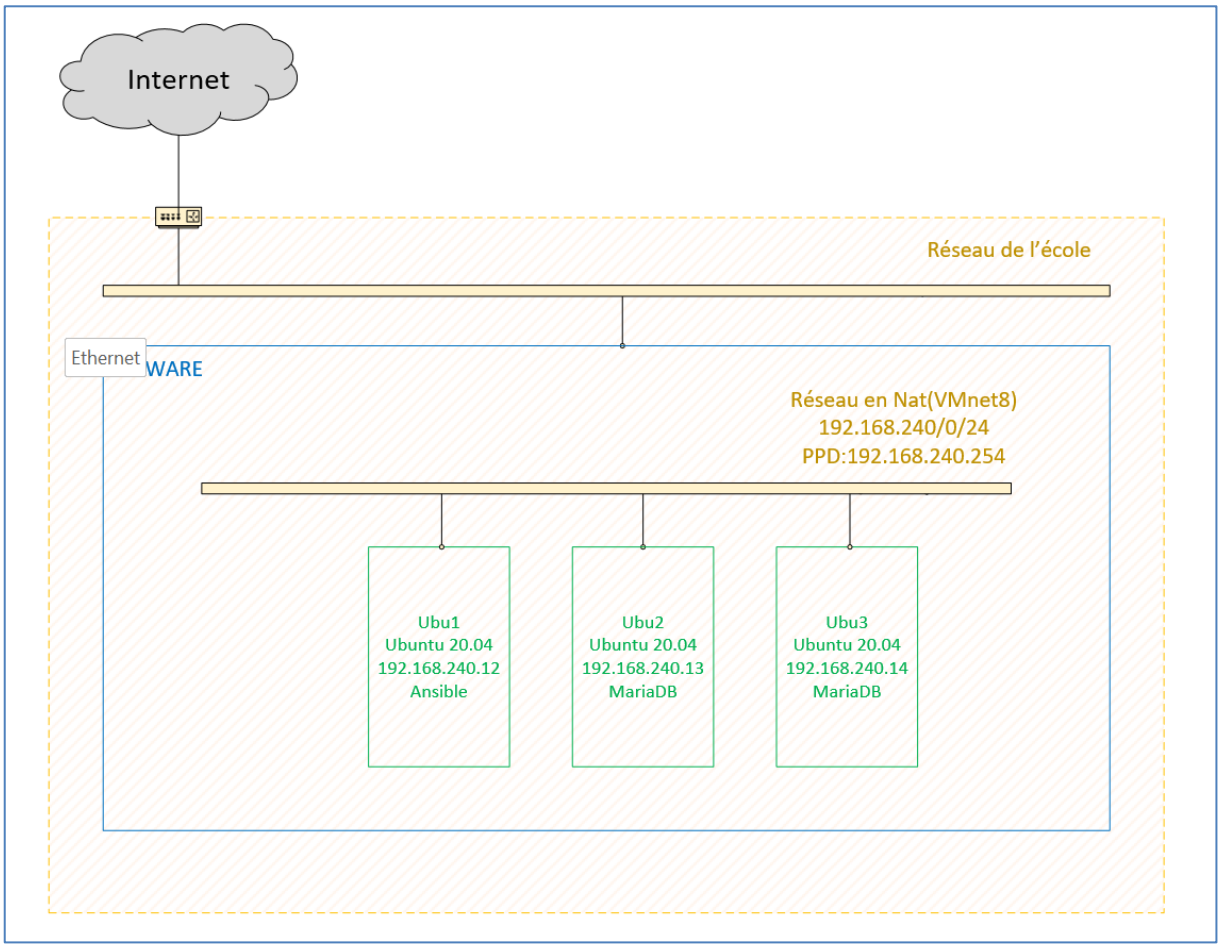
2 Prérequis

Voici les prérequis pour automatiser le déploiement d'un cluster MariaDB avec Galera via Ansible:

- **Système d'exploitation** : Les nœuds du cluster doivent exécuter un système d'exploitation compatible avec MariaDB et Galera. Les distributions Linux couramment utilisées pour les serveurs MariaDB incluent Ubuntu, Debian, CentOS, Red Hat Enterprise Linux et SUSE Linux Enterprise. Pour notre cas, nous utiliserons 2 systèmes d'exploitations Ubuntu version 20.04 avec MariaDB installé.
- **Accès SSH** : Les nœuds du cluster doivent être accessibles via SSH à partir de la machine exécutant Ansible.
- **Ansible** : Ansible doit être installé sur la machine depuis laquelle vous souhaitez déployer le cluster. Vous pouvez installer Ansible en utilisant le gestionnaire de packages de votre distribution Linux. Pour notre cas, nous utiliserons 1 systèmes d'exploitations Ubuntu version 20.04 avec Ansible installé.
- **Inventaire Ansible** : Vous devez créer un fichier d'inventaire Ansible qui liste les nœuds du cluster. Cela peut être un fichier texte simple ou un fichier au format INI.
- **Connexion MariaDB** : Les nœuds du cluster doivent pouvoir se connecter à la base de données MariaDB. Vous devez créer un utilisateur avec les privilèges nécessaires pour installer et configurer le cluster MariaDB.
- **Fichiers de configuration** : Vous devez créer les fichiers de configuration nécessaires pour MariaDB et Galera, y compris les fichiers de configuration my.cnf et galera.cnf.

Une fois que vous avez vérifié que tous les prérequis sont satisfaits, nous pourrons commencer à écrire notre script Ansible pour déployer le cluster MariaDB avec Galera.

Voici le Schéma :

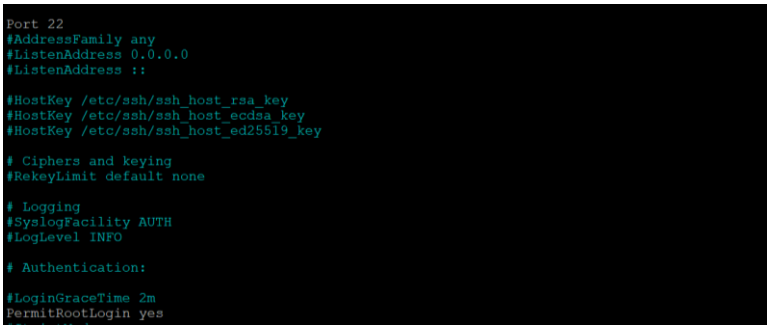


3 Procédure

3.1 Configuration de SSH

Pour commencer, nous allons activé SSH sur nos 3 machines Ubuntu (ubu1, ubu2, ubu3) en activant le port 22 et le PermitRootLogin :

```
nano /etc/ssh/sshd_config
```



3.2 Installation d'Ansible

Afin d'utiliser Ansible comme moyen de gestion de votre infrastructure de serveur, nous devons installer le logiciel Ansible sur la machine qui servira de nœud de contrôle Ansible.

Depuis notre nœud de contrôle (ubu1), on exécute la commande suivante pour inclure le PPA (archive personnelle de packages) du projet officiel dans la liste des sources de notre système :

```
sudo apt-add-repository ppa:ansible/ansible
```

Appuyez sur ENTER lorsque vous êtes invité à accepter l'ajout de PPA. Ensuite, on actualise l'index des packages de notre système afin qu'il connaisse les packages disponibles dans le PPA nouvellement inclus :

```
sudo apt update
```


Suite à cette mise à jour, nous devons installer le logiciel Ansible avec :

```
sudo apt install ansible
```

Notre nœud de contrôle Ansible dispose désormais de tous les logiciels nécessaires pour administrer nos hôtes. Ensuite, nous verrons comment ajouter nos hôtes au fichier d'inventaire du nœud de contrôle afin qu'il puisse les contrôler.

3.3 Configuration du fichier d'inventaire

Le fichier d'inventaire contient des informations sur les hôtes que vous gérerez avec Ansible. Nous pouvons inclure entre un et plusieurs centaines de serveurs dans votre fichier d'inventaire, et les hôtes peuvent être organisés en groupes et sous-groupes.

Le fichier d'inventaire est également souvent utilisé pour définir des variables qui ne seront valides que pour des hôtes ou des groupes spécifiques, afin d'être utilisées dans des playbooks et des modèles.

Certaines variables peuvent également affecter la façon dont un playbook est exécuté, comme la **ansible_python_interpretervariable** que nous verrons dans un instant.

Pour modifier le contenu de votre inventaire Ansible par défaut, on ouvre le répertoire **/etc/ansible/hosts**, sur votre nœud de contrôle Ansible :

```
nano /etc/ansible/hosts
```

Le fichier d'inventaire par défaut fourni par l'installation d'Ansible contient un certain nombre d'exemples que vous pouvez utiliser comme références pour configurer votre inventaire.

L'exemple suivant définit 3 groupes :

- Un groupe nommé **[servers]** avec deux serveurs différents, chacun identifié par un alias personnalisé (ubu2, ubu3) et associé à leurs adresse IP
- Un groupe nommé [server2] avec un serveur Ubuntu (ubu2)
- Un groupe nommé [server3] avec un serveur Ubuntu (ubu3)

Le **[all: vars]** sous-groupe définit le **ansible_python_interpreter** paramètre d'hôte qui sera valide pour tous les hôtes inclus dans cet inventaire.

Ce paramètre garantit que le serveur distant utilise **/usr/bin/python3** exécutable Python 3 au lieu de **/usr/bin/python** (Python 2.7), qui n'est pas présent sur les versions récentes d'Ubuntu.

```
[servers]
ubu2 ansible_host=192.168.240.13
ubu3 ansible_host=192.168.240.14

[server2]
ubu2 ansible_host=192.168.240.13

[server3]
ubu3 ansible_host=192.168.240.14

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

Lorsque vous avez terminé, enregistrez et fermez le fichier en appuyant sur CTRL+X puis Y et ENTER pour confirmer vos modifications.

Chaque fois que vous souhaitez vérifier votre inventaire, vous pouvez exécuter :

```
ansible-inventory --list -y
```

```
root@ubu1:~# ansible-inventory --list -y
{
    "_meta": {
        "hostvars": {
            "ubu2": {
                "ansible_host": "192.168.240.13",
                "ansible_python_interpreter": "/usr/bin/python3"
            },
            "ubu3": {
                "ansible_host": "192.168.240.14",
                "ansible_python_interpreter": "/usr/bin/python3"
            }
        }
    },
    "all": {
        "children": [
            "server2",
            "server3",
            "servers",
            "ungrouped"
        ]
    },
    "server2": {
        "hosts": [
            "ubu2"
        ]
    },
    "server3": {
        "hosts": [
            "ubu3"
        ]
    },
    "servers": {
        "hosts": [
            "ubu2",
            "ubu3"
        ]
    }
}
```


3.4 Rattachement aux serveurs gérés et premières commandes

A présent, notre serveur Ansible est prêt, nous devons établir notre premier contact entre les machines gérées (nos deux serveurs Ubuntu) et notre "tour de contrôle" qui est notre serveur Ansible.

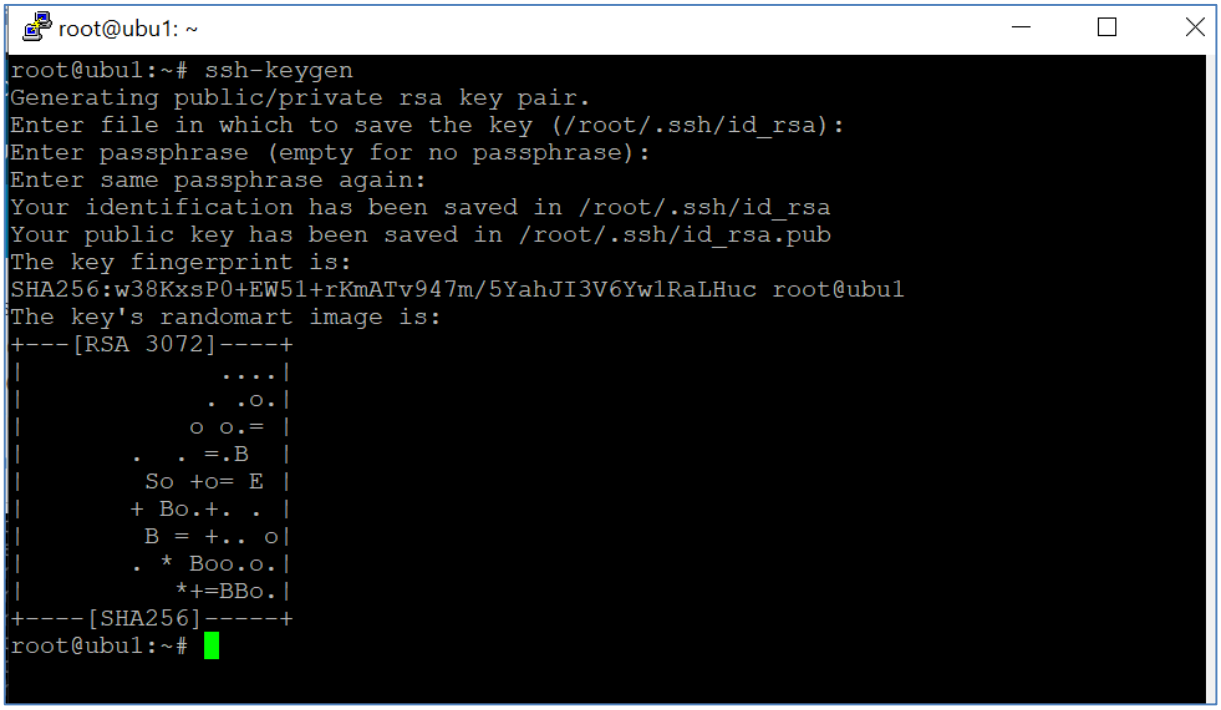
Comme indiqué plus haut, les échanges seront effectués en utilisant le protocole SSH. A ce titre, deux méthodes d'authentifications sont utilisables :

- **La méthode login/password :** Méthode standard et la plus utilisée lors d'une connexion SSH manuelle, on saisit le login avec lequel on souhaite se connecter, puis le mot de passe
- **La méthode clé :** un échange de clé est à établir. Le serveur souhaitant se connecter doit avoir sa clé publique comme autorisée du côté du serveur sur lequel il souhaite se connecter.

Avec Ansible, nous préférons travailler avec l'échange de clé, il serait sinon nécessaire de saisir un mot de passe à chaque fois qu'une exécution de commande est effectuée sur un serveur distant.

Cela est difficilement pensable pour plusieurs dizaines de serveurs. Pour cela, il est nécessaire de créer une paire de clé sur notre serveur Ansible, nous utiliserons pour cela la commande :

```
ssh-keygen
```



A présent, je vais envoyer ma clé publique sur tous les serveurs que j'aurai à gérer avec Ansible :

```
ssh-copy-id root@192.168.240.13
ssh-copy-id root@192.168.249.14
```

Après chacune de ces deux commandes, le mot de passe de l'utilisateur root sur sera à saisir. Après cela, l'authentification SSH est automatiquement prise en compte :

```
root@ubu1:~# ssh-copy-id root@192.168.240.13
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa
.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
root@192.168.240.13's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@192.168.240.13'"
and check to make sure that only the key(s) you wanted were added.
```

```
root@ubu1:~# ssh-copy-id root@192.168.240.14
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '192.168.240.14 (192.168.240.14)' can't be established.
ECDSA key fingerprint is SHA256:qjZdSjjk3xDn2foN+hoBPXFO0EyzhqwPpjdCAtRIZ7w.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys
root@192.168.240.14's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@192.168.240.14'"
and check to make sure that only the key(s) you wanted were added.
```

3.5 Test de connexion

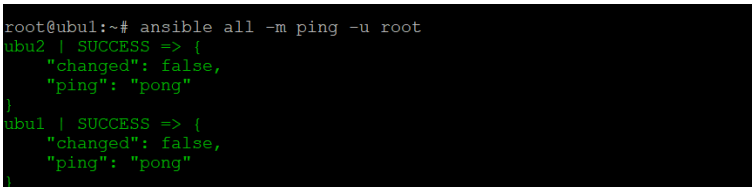
Après avoir configuré le fichier d'inventaire pour inclure vos serveurs, il est temps de vérifier si Ansible est capable de se connecter à ces serveurs et d'exécuter des commandes via SSH.

Pour ce guide, nous utiliserons le compte racine Ubuntu car c'est généralement le seul compte disponible par défaut sur les serveurs nouvellement créés. Si vos hôtes Ansible ont déjà créé un utilisateur sudo régulier, nous vous encourageons à utiliser ce compte à la place.

Vous pouvez utiliser **–u** argument pour spécifier l'utilisateur du système distant. S'il n'est pas fourni, Ansible essaiera de se connecter en tant qu'utilisateur système actuel sur le nœud de contrôle.

À partir de votre ordinateur local ou du nœud de contrôle Ansible, exécutez :

```
ansible all -m ping -u root
```



Cette commande utilisera le **ping** module intégré d'Ansible pour exécuter un test de connectivité sur tous les nœuds de votre inventaire par défaut, en vous connectant en tant que root. Le ping module testera :

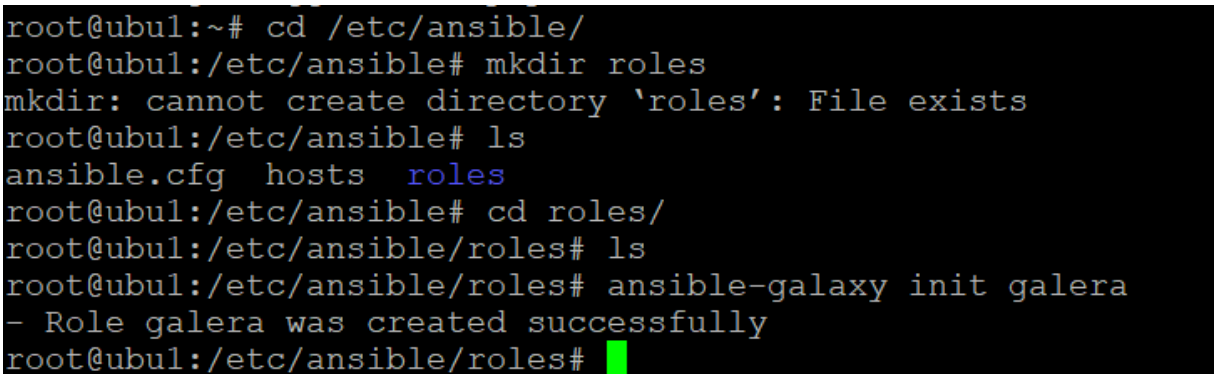
- Si les hôtes sont accessibles ;
- Si vous disposez d'informations d'identification SSH valides ;
- Si les hôtes sont capables d'exécuter des modules Ansible à l'aide de Python.

3.6 Création du cluster avec Galera

3.6.1 Création de nos fichiers de config Galera

Nous allons créer notre rôle avec ansible-galaxy pour se faciliter le travail :

```
cd /etc/ansible/
mkdir roles
cd roles
ansible-galaxy init galera
```



Ensuite, vous devrez créer un fichier de configuration Galera pour chaque nœud (ubu2, ubu3) afin que chaque nœud puisse communiquer entre eux.

Pour cela, nous allons créez un fichier my2.cnf.j2 pour notre premier nœud (ubu2) à l'aide de la commande suivante :

```
nano /etc/ansible/roles/galera/templates/my2.cnf.j2
```

Voici ce qui faut coller dans le fichier :

```
[mysqld]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0

# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

# Galera Cluster Configuration
wsrep_cluster_name="test_cluster"
wsrep_cluster_address="gcomm://192.168.240.13,192.168.240.14"

# Galera Synchronization Configuration
wsrep_sst_method=rsync

# Galera Node Configuration
wsrep_node_address="192.168.240.13"
wsrep_node_name="ubu2"
```

Ensuite, nous allons créez un fichier my3.cnf.j2 pour notre second nœud (ubu3) à l'aide de la commande suivante :

```
nano /etc/ansible/roles/galera/templates/my3.cnf.j2
```

Voici ce qui faut coller dans le fichier :

```
[mysqld]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0

# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

# Galera Cluster Configuration
wsrep_cluster_name="test_cluster"
wsrep_cluster_address="gcomm://192.168.240.13,192.168.240.14"

# Galera Synchronization Configuration
wsrep_sst_method=rsync

# Galera Node Configuration
wsrep_node_address="192.168.240.14"
wsrep_node_name="ubu3"
```

3.6.2 Création du Playbook

Un playbook Ansible est un fichier de configuration écrit au format YAML qui définit un ensemble de tâches à exécuter par la plateforme d'automatisation Ansible. Les playbooks sont utilisés pour automatiser diverses tâches informatiques, telles que la configuration des systèmes, le déploiement d'applications et la gestion des mises à jour logicielles.

Un playbook contient une série de tâches, chacune étant définie à l'aide de modules Ansible. Les tâches peuvent être exécutées sur un ou plusieurs hôtes distants, individuellement ou en groupe. Les hôtes distants peuvent être spécifiés à l'aide de diverses méthodes, notamment des noms d'hôte, des adresses IP ou des groupes d'hôtes définis dans un fichier d'inventaire Ansible.

Les playbooks peuvent également inclure des variables et des conditions, qui vous permettent de personnaliser l'exécution du playbook en fonction de l'état actuel de l'environnement. Cela permet d'écrire des playbooks réutilisables, flexibles et évolutifs.

En plus d'effectuer des tâches automatisées, les playbooks peuvent également être utilisés pour appliquer des politiques et garantir que les systèmes sont configurés de manière cohérente et sécurisée.

Cela fait des playbooks un outil puissant pour les administrateurs informatiques et les ingénieurs DevOps qui doivent gérer un grand nombre de systèmes et s'assurer qu'ils sont configurés correctement et en toute sécurité.

Pour cela, éditer le fichier /etc/ansible/galera/tasks/main79.yml :

```
nano /etc/ansible/galera/tasks/main79.yml
```

Voici ci qui faut coller dans le fichier :

```
---
- name: Deploy MariaDB Cluster
  hosts: servers
  become: yes
  gather_facts: false
  tasks:
    - name: Install MariaDB
      apt:
        name: mariadb-server
        state: present

    - name: Start and enable MariaDB service
      service:
        name: mysql
        state: started
        enabled: yes

    - name: Stop MariaDB service
      service:
        name: mysql
        state: stopped

- name: Configure MariaDB for Galera on node 1
  hosts: server2
  tasks:
    - name: Copy Galera on node 1
      template:
        src: "/etc/ansible/roles/galera/templates/my2.cnf.j2"
        dest: "/etc/mysql/conf.d/my.cnf"

- name: Configure MariaDB for Galera on node 2
  hosts: server3
  tasks:
    - name: Copy Galera on node 2
      template:
        src: "/etc/ansible/roles/galera/templates/my3.cnf.j2"
        dest: "/etc/mysql/conf.d/my.cnf"

- name: Execute command on node 1
  hosts: server2
  become: yes
  tasks:
    - name: Run shell command
      shell: |
        galera_new_cluster

- name: Start MariaDB service on node 1
  hosts: server2
  become: yes
```


```
  tasks:
    - name: Start MariaDB service
      service:
        name: mysql
        state: started

- name: Start MariaDB service on node 2
  hosts: server3
  become: yes
  tasks:
    - name: Start MariaDB service
      service:
        name: mysql
        state: started
```

Ce script Ansible permettra de déployer le cluster MariaDB avec Galera. Il comporte plusieurs sections (ou "plays"), chacune étant destinée à être exécutée sur une liste définie de serveurs :

1. La première section s'appelle "**Deploy MariaDB Cluster**". Elle définit les serveurs cibles en utilisant l'alias "servers". Elle active également les privilèges administrateur ("become: yes") et désactive la collecte de faits ("gather_facts: false"). Les tâches définies dans cette section sont :
    - Installer MariaDB en utilisant le module apt
    - Démarrer et activer le service MariaDB
    - Arrêter le service MariaDB
2. La seconde section s'appelle "**Configure MariaDB for Galera on node 1**". Elle définit les serveurs cibles en utilisant l'alias "server2". La tâche définie dans cette section est de copier le fichier de configuration Galera à partir d'un modèle en utilisant le module Template.
3. La troisième section s'appelle "**Configure MariaDB for Galera on node 2**". Elle définit les serveurs cibles en utilisant l'alias "server3". La tâche définie dans cette section est similaire à la section précédente, mais elle utilise un modèle de configuration différent.
4. La quatrième section s'appelle "**Execute command on node 1**". Elle définit les serveurs cibles en utilisant l'alias "server2". Elle active les privilèges administrateur ("become: yes"). La tâche définie dans cette section est d'exécuter la commande shell "**galera_new_cluster**" qui permettra d'initialisez le cluster MariaDB Galera.
5. La cinquième section s'appelle "**Start MariaDB service on node 1**". Elle définit les serveurs cibles en utilisant l'alias "server2". Elle active les privilèges administrateur ("become: yes"). La tâche définie dans cette section est de démarrer le service MariaDB.
6. La sixième section s'appelle "**Start MariaDB service on node 2**". Elle définit les serveurs cibles en utilisant l'alias "server3". Elle active les privilèges administrateur ("become: yes"). La tâche définie dans cette section est similaire à la section précédente, mais cible un autre serveur.

3.6.3 Démarrer le script

A partir de là, il n'y a plus lancer le script :

```
ansible-playbook main79.yml
```

Si aucune erreur s'affiche lors du « PLAY RECAP », alors tout est ok :

```
PLAY RECAP *********************************************************************
***
ubu2                       : ok=7    changed=4    unreachable=0    failed=0
  skipped=0    rescued=0    ignored=0
ubu3                       : ok=5    changed=3    unreachable=0    failed=0
  skipped=0    rescued=0    ignored=0
```

3.6.4 Vérification

Après avoir lancé notre script, nous allons vérifier l'état de notre cluster. Pour cela, nous allons nous sur chacun de nœud et exécuter la commande suivante :

```
mysql -u root -proot -e "SHOW STATUS LIKE 'wsrep_cluster_size'"
```

```
root@ubu2:~# mysql -u root -proot -e "SHOW STATUS LIKE 'wsrep_cluster_siz
e'"
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 2     |
+--------------------+-------+
```

```
root@ubu3:~# mysql -u root -proot -e "SHOW STATUS LIKE 'wsrep_cluster_size'"
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 2     |
+--------------------+-------+
```

Wsrep_cluster_size est une variable d'état dans Galera Cluster pour MariaDB qui indique le nombre de nœuds dans le cluster. Cette valeur est mise à jour dynamiquement lorsque les nœuds rejoignent ou quittent le cluster. La valeur de wsrep_cluster_size doit être égale au nombre de nœuds du cluster à un moment donné.

Dans notre cas, nous pouvons voir qu'il y'a bien nos deux nos dans le cluster.

3.7 Vérifier la réplication du cluster

Ensuite, vous devrez vérifier si la réplication fonctionne ou non. Sur le premier nœud(ubu2), connectez-vous à MariaDB avec la commande suivante :

```
mysql -u root –proot
```

Une fois connecté, créez une base de données avec la commande suivante :

```
CREATE DATABASE db1 ;
CREATE DATABASE db2 ;
```

Ensuite, quittez MariaDB avec la commande suivante :

```
exit
```

Ensuite, accédez au deuxième nœud(ubu3) et connectez-vous à MariaDB avec la commande suivante :

```
mysql -u root –p
```

Ensuite, exécutez la commande suivante pour afficher toutes les bases de données :

```
show databases;
```

Vous devriez voir que les deux bases de données que nous avons créées sur le premier nœud sont répliquées sur le deuxième nœud :