TP Cassandra: Politique de Reprise Personnalisée

Ynov Mohammed El Malki

February 7, 2025

1 Introduction

Ce TP explore la création et l'utilisation d'une politique de reprise personnalisée dans Apache Cassandra avec le driver Python.

2 Pré-requis

- Un cluster Cassandra fonctionnel (au moins 2 nœuds).
- Python 3 et le package `cassandra-driver` :

```
pip install cassandra-driver
```

- Création d'une base de test et d'une table :

```
CREATE KEYSPACE IF NOT EXISTS test_retry
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 2};

USE test_retry;

CREATE TABLE IF NOT EXISTS users (
    id UUID PRIMARY KEY,
    name TEXT,
    age INT
);
```

3 Script Python : Politique Personnalisée

```python
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider
from cassandra.policies import RetryPolicy
from cassandra.query import SimpleStatement
import uuid

class CustomRetryPolicy(RetryPolicy):
    def on_read_timeout(self, *args, **kwargs):
        print("␣␣Timeout␣en␣lecture,␣r essai␣forc ")
        return self.RETRY

    def on_write_timeout(self, *args, **kwargs):
        print("Timeout␣en␣ criture ,␣r essai␣forc ")
        return self.RETRY

    def on_unavailable(self, *args, **kwargs):
        print("Noeud␣indisponible,␣r essai␣avec␣coh rence␣
            r duite")
        return self.RETRY

cluster = Cluster(['127.0.0.1'])
session = cluster.connect('test_retry')
session.default_retry_policy = CustomRetryPolicy()

def test_retry_policy():
    print("\n␣Test␣de␣la␣politique␣personnalis e")
    query = SimpleStatement("INSERT␣INTO␣users␣(id,␣name,␣age
        )␣VALUES␣(%s,␣%s,␣%s)")
    try:
        session.execute(query, (uuid.uuid4(), "Alice", 30))
        print("Insertion␣r ussie␣avec␣la␣politique␣
            personnalis e")
    except Exception as e:
        print(f"Erreur␣:␣{e}")

test_retry_policy()

cluster.shutdown()
```

4 Extension du TP : Politique personnalisée

L'objectif est de créer une politique de reprise qui effectue un réessai en cas d'erreur et ajuste automatiquement le niveau de cohérence si nécessaire.

5 Résultats attendus

- La politique doit réessayer automatiquement en cas d'erreur réseau ou de timeout.
- Lorsqu'un nœud est indisponible, elle doit tenter une cohérence plus faible avant d'échouer.
- Les logs doivent afficher les réessais et ajustements effectués.

6 Simulation d'erreurs

- Exécuter le script normalement.
- Arrêter un nœud Cassandra :

```
sudo systemctl stop cassandra
```

- Observer comment la politique personnalisée réagit.

7 Explication des politiques testées

- **RetryPolicy** : Politique par défaut, réessaie plusieurs fois avant d'échouer.
- **DowngradingConsistencyRetryPolicy** : Réduit le niveau de cohérence pour réussir la requête.
- **FallthroughRetryPolicy** : N'essaie pas de ré exécuter et laisse l'erreur se propager.
- **CustomRetryPolicy** : Personnalisée pour améliorer la gestion des erreurs en adaptant dynamiquement les tentatives.

8 Conclusion

Ce TP montre comment une politique de reprise personnalisée peut améliorer la gestion des erreurs en Cassandra. Il permet d'expérimenter les différents comportements et d'adapter la robustesse des requêtes en fonction des besoins applicatifs.