Rapport de Projet sur l'Utilisation de HBase avec Python pour la Gestion de Données

Binôme : KHARBOUCHE Bilal et SAKLAB Youssef

Ce rapport documente le projet réalisé dans le cadre du module de Big Data, portant sur la création d'une application Python interagissant avec un cluster HBase pour le stockage et la récupération de données. Ce projet a été mené par le binôme KHARBOUCHE Bilal et SAKLAB Youssef.

Le rapport détaille les différentes étapes, les défis rencontrés ainsi que les solutions apportées tout au long du processus d'installation, de configuration et de développement de l'application pour interagir avec HBase.

Le but principal de ce projet était de maîtriser l'utilisation de HBase dans un contexte pratique et de développer une application fonctionnelle permettant de stocker, récupérer, et manipuler des données à grande échelle à l'aide de Python.

Le rapport présente une analyse des besoins, les difficultés initiales lors de l'installation de Happybase, les solutions alternatives trouvées pour établir la connexion avec le cluster HBase, ainsi que le développement d'une application Python pour manipuler les données stockées dans HBase.

Les détails techniques, les procédures, les résultats obtenus et les leçons apprises sont exposés dans ce rapport pour fournir une vue exhaustive du processus suivi pour atteindre les objectifs du projet.

BIG DATA

1. Introduction au Projet

- Contexte du Big Data
- Objectif du projet : interaction avec HBase via Python

2. Problématique

- Difficultés lors de l'installation initiale de Happybase avec Python 2.7 sur Cloudera HBase via pip

3. Première tentative d'établir la connexion

- Échec de la connexion Python-HBase à cause d'une version obsolète du package

4. Trouver une Solution Alternative

- Utilisation de commandes spécifiques (ifconfig) pour contourner les problèmes et réussir la connexion

5. Installation de Happybase sur Windows

- Décision d'installer Happybase sur un système Windows pour une meilleure compatibilité

6. Écriture d'une Base de Données HBase pour les Commandes

- Utilisation du HBase Shell pour créer des tables et insérer des données de test

7. Insertion de Données via le HBase Shell

- Vérification de la fonctionnalité et de la structure de la base de données nouvellement créée

8. Développement d'une Application Python pour Manipuler les Commandes

- Création d'une application permettant l'insertion, la suppression et la mise à jour des données dans la base de données HBase

9. Conclusion

- Récapitulation des défis, des solutions trouvées et des leçons apprises
- Importance de la flexibilité et de l'adaptabilité dans le domaine du Big Data

BIG DATA

1. Introduction au projet

Le Big Data a révolutionné la façon dont nous collectons, stockons et analysons les données à grande échelle. Dans cet esprit, ce rapport vise à explorer la création d'une application utilisant Python pour interagir avec un cluster HBase, une base de données NoSQL orientée colonnes. L'objectif principal de ce projet est de stocker efficacement d'énormes volumes de données dans le cluster HBase et de pouvoir récupérer ces données de manière rapide et précise.

Ce projet est motivé par la nécessité croissante de gérer et d'exploiter des ensembles de données massifs, allant des données en temps réel aux données historiques. Le choix de HBase comme solution de stockage repose sur sa capacité à évoluer horizontalement, à gérer des données semi-structurées et à fournir des performances élevées pour les opérations de lecture et d'écriture.

2. Problématique

L'installation initiale de Happybase avec Python 2.7 sur Cloudera HBase via pip a rencontré des obstacles majeurs. Les erreurs de compatibilité avec la bibliothèque Thrift ont rendu l'installation via pip impossible, bloquant ainsi la connexion prévue entre Python et HBase.

3. Première tentative d'établir la connexion

Une première tentative a été faite pour écrire un code Python simple afin d'établir la connexion entre Python et HBase sur Cloudera.

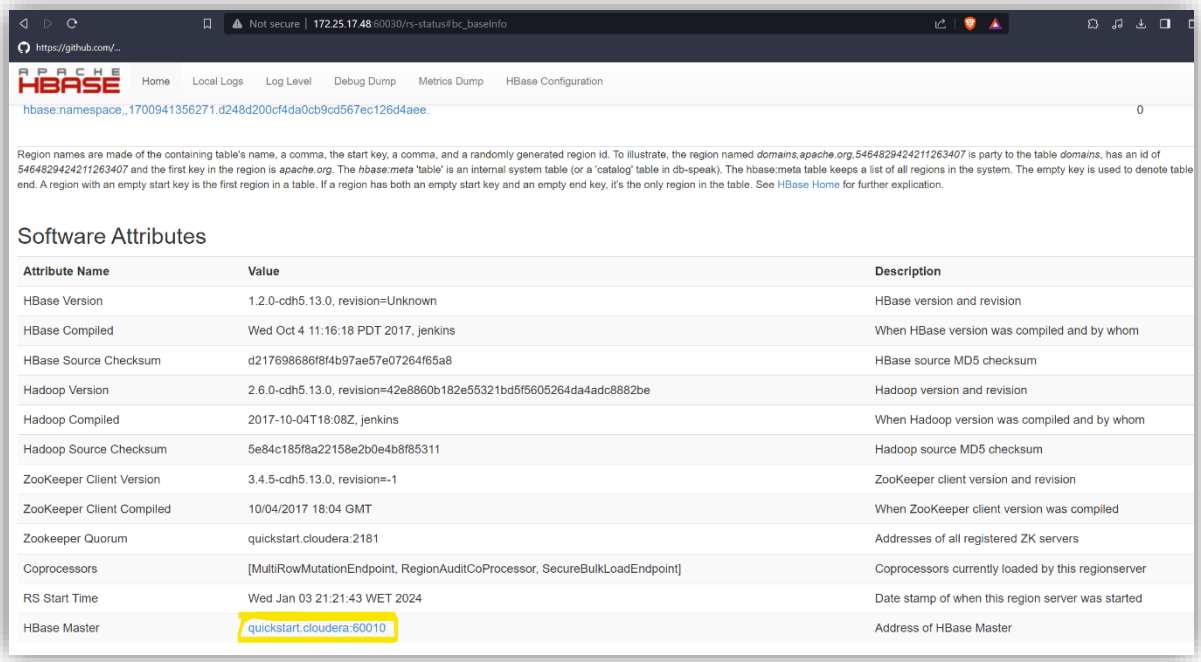
Cependant, cette approche a été entravée par une version obsolète du package, entraînant l'échec de la connexion.

4. Trouver une solution alternative

Face à ces difficultés, une solution alternative a été explorée. En utilisant la commande ifconfig, l'adresse IP a été récupérée,

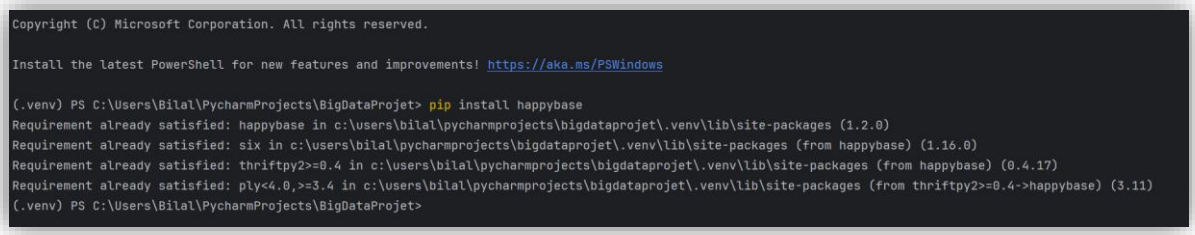
```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ ifconfig
eth1      Link encap:Ethernet  HWaddr 00:0C:29:8B:2F:AF
          inet addr:172.25.17.48  Bcast:172.25.31.255  Mask:255.255.240.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:989 errors:0 dropped:0 overruns:0 frame:0
          TX packets:915 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:266381 (260.1 KiB)  TX bytes:127779 (124.7 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:370306 errors:0 dropped:0 overruns:0 frame:0
          TX packets:370306 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:772267763 (736.4 MiB)  TX bytes:772267763 (736.4 MiB)
```

et le fichier de configuration hbase-site.xml a été modifié pour inclure le port 60030. Cette approche a permis de contourner les problèmes initiaux et d'établir une connexion réussie entre Python et HBase.



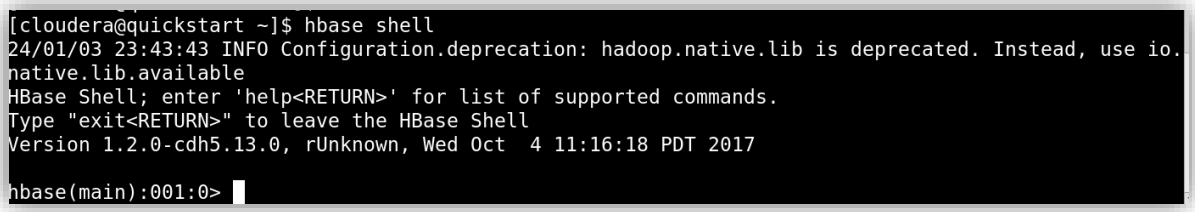
5. Installation de Happybase sur Windows

Pour assurer une meilleure compatibilité et pour éviter les problèmes avec l'environnement de Cloudera sur VMware, Happybase a été installé sur un système Windows. Cette approche a été fructueuse et a permis d'établir une connexion stable entre Python et Cloudera HBase.



6. Écriture d'une base de données HBase pour les commandes

En complément de l'installation et de la configuration de Happybase, une base de données HBase a été mise en place pour stocker des commandes. À l'aide du HBase Shell,



Des commandes ont été définies pour la création de tables et l'insertion de données de test.

```
hbase(main):006:0> list
TABLE
Films
clouderaArbres
commande
3 row(s) in 0.1110 seconds

=> ["Films", "clouderaArbres", "commande"]
hbase(main):007:0> describe "commande"
Table commande is ENABLED
commande
COLUMN FAMILIES DESCRIPTION
{NAME => 'client', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', V
ERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS =>
 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'vente', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VE
RSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS =>
'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
2 row(s) in 0.0660 seconds
```

7. Insertion de données via le HBase Shell

Des données de test ont été insérées dans la base de données HBase en utilisant le HBase Shell pour vérifier la fonctionnalité et la structure de la base de données nouvellement créée.

Test :

```python
import happybase

try:
    connection = happybase.Connection( host: '172.25.17.48', port=60030)
    print("HBase Thrift server is accessible.")
    connection.close()
except Exception as e:
    print("Could not connect to the HBase Thrift server:", e)
```

```
C:\Users\Bilal\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Bilal\PycharmProjects\pythonProject\test.py
HBase Thrift server is accessible.

Process finished with exit code 0
```

8. Développement d'une application Python pour manipuler les commandes

Une application Python a été développée pour interagir avec la base de données HBase. Cette application permet l'insertion, la suppression et la mise à jour des commandes stockées dans la base de données à l'aide de la bibliothèque Happybase.

Il est structuré autour de quatre principales fonctions pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur une table nommée 'commande'

```python
def main():
    connection = happybase.Connection( host: '172.25.17.48', port=9090)

    while True:
        print("\nOptions:")
        print("1. Insert data")
        print("2. Retrieve data")
        print("3. Update data")
        print("4. Delete data")
        print("5. Exit")

        choice = input("Enter your choice (1-5): ")

        if choice == '1':
            insert_data(connection)
        elif choice == '2':
            retrieve_data(connection)
        elif choice == '3':
            update_data(connection)
        elif choice == '4':
            delete_data(connection)
        elif choice == '5':
            print("Exiting the program...")
            break
        else:
            print("Invalid choice. Please select a valid option.")

    connection.close()


if __name__ == "__main__":
    main()
```

8.1 Fonction d'Insertion de Données

8.1.1 Explication de la fonction insert_data()

Cette fonction permet à l'utilisateur d'insérer des données dans la table 'commande'. Elle prend les informations suivantes : row_key, nom, ville, produit, et prix depuis l'entrée utilisateur, puis crée un dictionnaire data avec ces informations. Ensuite, elle utilise la

méthode put() pour insérer ces données dans la table. Après l'insertion, elle affiche un message confirmant l'opération.

8.1.2 Exécution et utilisation de la fonction pour insérer des données

```python
def insert_data(connection):
    table_name = 'commande'
    table = connection.table(table_name)

    row_key = input("Enter the row key for insertion: ")
    nom = input("Enter client's name: ")
    ville = input("Enter client's city: ")
    produit = input("Enter product: ")
    prix = input("Enter price: ")

    data = {
        f'client:nom': nom,
        f'client:ville': ville,
        f'vente:produit': produit,
        f'vente:prix': prix
    }
    table.put(row_key, data)
    print(f"Inserted data with row key {row_key}")
```

8.1.3 Illustration du processus d'insertion de données

```
C:\Users\Bilal\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Bilal\PycharmProjects\pythonProject\main.py

Options:
1. Insert data
2. Retrieve data
3. Update data
4. Delete data
5. Exit
Enter your choice (1-5): 1
Enter the row key for insertion: 1
Enter client's name: bilal
Enter client's city: meknes
Enter product: TV
Enter price: 1500
Inserted data with row key 1
```

8.2 Fonction de Recuperation de Donnees

8.2.1 Explication de la fonction retrieve_data()

Cette fonction permet de récupérer des données à partir de la table 'commande'. Elle prend une row_key en entrée, recherche cette clé dans la table, puis récupère les données associées à cette clé. Si des données sont trouvées, elle les affiche à l'utilisateur avec leurs colonnes et valeurs correspondantes.

8.2.2 Exécution et utilisation de la fonction pour récupérer des données

```
def retrieve_data(connection):
    table_name = 'commande'
    table = connection.table(table_name)

    row_key = input("Enter the row key for retrieval: ")
    row = table.row(row_key)

    if row:
        print(f"Retrieved data for row key {row_key}:")
        for column, value in row.items():
            print(f"{column.decode('utf-8')} : {value.decode('utf-8')}")
    else:
        print(f"No data found for row key {row_key}")
```

8.2.3 Illustration du processus de récupération de données

```
Options:
1. Insert data
2. Retrieve data
3. Update data
4. Delete data
5. Exit
Enter your choice (1-5): 2
Enter the row key for retrieval: 1
Retrieved data for row key 1:
client:nom : bilal
client:ville : meknes
vente:prix : 1500
vente:produit : TV
```

8.3 Fonction de Mise à Jour de Donnees

8.3.1 Explication de la fonction update_data()

Cette fonction permet de mettre à jour les données dans la table 'commande'. Elle demande à l'utilisateur la row_key pour identifier la ligne à mettre à jour et le nouveau prix. Ensuite, elle crée un dictionnaire data avec la nouvelle valeur de prix pour la colonne 'vente:prix' et utilise la méthode put() pour mettre à jour cette colonne pour la row_key spécifiée.

8.3.2 Exécution et utilisation de la fonction pour mettre à jour des données

```python
def update_data(connection):
    table_name = 'commande'
    table = connection.table(table_name)

    row_key = input("Enter the row key for update: ")
    prix = input("Enter the updated price: ")

    data = {
        f'vente:prix': prix
    }
    table.put(row_key, data)
    print(f"Updated data for row key {row_key}")
```

8.3.3 Illustration du processus de mise à jour de données

```
Enter your choice (1-5): 3
Enter the row key for update: 1
Enter the updated price: 500
Updated data for row key 1

Options:
1. Insert data
2. Retrieve data
3. Update data
4. Delete data
5. Exit
Enter your choice (1-5): 2
Enter the row key for retrieval: 1
Retrieved data for row key 1:
client:nom : bilal
client:ville : meknes
vente:prix : 500
vente:produit : TV
```

8.2 Fonction de Suppression de Donnees

8.2.1 Explication de la fonction delete_data()

Cette fonction permet de supprimer des données de la table 'commande'. Elle demande à l'utilisateur la row_key de la ligne à supprimer, puis utilise la méthode delete() pour supprimer complètement cette ligne de la table.

8.2.2 Exécution et utilisation de la fonction pour supprimer des données

```
def delete_data(connection):
    table_name = 'commande'
    table = connection.table(table_name)

    row_key = input("Enter the row key for deletion: ")
    table.delete(row_key)
    print(f"Deleted data with row key {row_key}")
```

8.2.3 Illustration du processus d'insertion de suppression de données

```
Options:
1. Insert data
2. Retrieve data
3. Update data
4. Delete data
5. Exit
Enter your choice (1-5): 4
Enter the row key for deletion: 1
Deleted data with row key 1

Options:
1. Insert data
2. Retrieve data
3. Update data
4. Delete data
5. Exit
Enter your choice (1-5): 2
Enter the row key for retrieval: 1
No data found for row key 1
```

9. Conclusion

Ce projet a été une exploration enrichissante de l'utilisation de Python pour interagir avec un cluster HBase dans le contexte du Big Data. Les différentes étapes ont mis en lumière des défis significatifs tout en présentant des solutions alternatives pour surmonter ces obstacles.

Nous avons constaté que l'installation initiale de Happybase sur Cloudera HBase a rencontré des problèmes de compatibilité avec la bibliothèque Thrift, entraînant des difficultés dans l'établissement de la connexion entre Python et HBase. Cependant, grâce à une solution alternative en modifiant le fichier de configuration, nous avons réussi à établir une connexion stable, démontrant ainsi la flexibilité et la résilience nécessaires pour naviguer dans des environnements techniques complexes.

L'installation de Happybase sur un système Windows s'est avérée être une approche plus harmonieuse, assurant une

meilleure compatibilité et éliminant les problèmes rencontrés dans l'environnement de Cloudera sur VMware.

La création d'une base de données HBase dédiée pour stocker les commandes a été une étape cruciale, utilisant efficacement le HBase Shell pour définir les schémas et insérer des données de test, confirmant ainsi la fonctionnalité et la structure de la base de données.

Enfin, le développement d'une application Python pour manipuler les commandes stockées dans la base de données HBase a été un succès, offrant des fonctionnalités d'insertion, de suppression et de mise à jour des données, démontrant ainsi l'intégration réussie de Python avec HBase via la bibliothèque Happybase.

Ce projet a souligné l'importance de la flexibilité, de la persévérance et de l'adaptabilité lors de l'exploration et de l'implémentation de solutions dans le domaine du Big Data, dévoilant de précieuses leçons à emporter pour des projets futurs.