Bash

Guide de création d'un job maître en Bash appelant plusieurs scripts Python pour le Big Data

Introduction

Dans le cadre du Big Data, il est souvent nécessaire d'automatiser et d'orchestrer plusieurs scripts Python via un script Bash maître. Ce guide explique comment créer un script Bash maître qui exécute plusieurs scripts Python différents en leur passant plusieurs arguments, notamment le chemin des fichiers d'entrée et de sortie.

1. Création des scripts Python

Les scripts Python prendront en charge la lecture et l'écriture de fichiers avec Spark.

1.1. Exemple de script Python pour traitement (`job_traitement.py)

```
import argparse
from pyspark.sql import SparkSession

def main(input_path, output_path):
    spark = SparkSession.builder.appName("TraitementJob").getOrCreate()
    df = spark.read.csv(input_path, header=True, inferSchema=True)
    df = df.withColumn("processed_column", df["existing_column"] * 2)
    df.write.csv(output_path, header=True, mode="overwrite")
    spark.stop()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Traitement des données")
    parser.add_argument("--input", required=True, help="Chemin du fichier
d'entrée")
    parser.add_argument("--output", required=True, help="Chemin du fichier de
sortie")
    args = parser.parse_args()
    main(args.input, args.output)
```

1.2. Exemple de script Python pour analyse (`job_analyse.py`)

```python
import argparse
from pyspark.sql import SparkSession

def main(input_path, output_path):
    spark = SparkSession.builder.appName("AnalyseJob").getOrCreate()
    df = spark.read.csv(input_path, header=True, inferSchema=True)
    df.describe().show()
    df.write.csv(output_path, header=True, mode="overwrite")
    spark.stop()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Analyse des données")
    parser.add_argument("--input", required=True, help="Chemin du fichier
d'entrée")
    parser.add_argument("--output", required=True, help="Chemin du fichier de
sortie")
    args = parser.parse_args()
    main(args.input, args.output)
```

2. Création du script Bash maître

Le script Bash maître va exécuter plusieurs jobs Python différents et gérer leur exécution.

2.1. Exemple de script Bash maître (`master_job.sh`)

```bash
#!/bin/bash
if [ "$#" -lt 4 ]; then
    echo "Usage: $0 <input_traitement> <output_traitement> <input_analyse>
<output_analyse>"
    exit 1
fi

TRAITEMENT_INPUT="$1"
TRAITEMENT_OUTPUT="$2"
ANALYSE_INPUT="$3"
ANALYSE_OUTPUT="$4"

echo "Démarrage du job maître..."

# Exécuter le job de traitement

echo "Exécution du job de traitement..."
```

```
spark-submit job_traitement.py --input "$TRAITEMENT_INPUT" --output
"$TRAITEMENT_OUTPUT"

if [ $? -ne 0 ]; then
    echo "Erreur lors de l'exécution du job de traitement."
    exit 1
fi

echo "Exécution du job d'analyse..."

spark-submit job_analyse.py --input "$ANALYSE_INPUT" --output
"$ANALYSE_OUTPUT"

if [ $? -ne 0 ]; then
    echo "Erreur lors de l'exécution du job d'analyse."
    exit 1
fi

echo "Job maître terminé avec succès."
```

3. Exécution du script Bash maître

Avant d'exécuter le script Bash maître, assurez-vous qu'il est exécutable :

```
chmod +x master_job.sh
```

Puis exécutez-le avec les chemins des fichiers d'entrée et de sortie :

```
./master_job.sh /chemin/input_traitement.csv /chemin/output_traitement.csv
/chemin/input_analyse.csv /chemin/output_analyse.csv
```

4. Bonnes pratiques

- **Utiliser des logs** : remplacez `echo` par `logger` pour une meilleure gestion des logs.
- **Gérer les erreurs** : utilisez `trap` pour capturer les erreurs et nettoyer si nécessaire.
- **Surveiller les ressources** : surveillez l'utilisation de la mémoire et du CPU lors de l'exécution.

Conclusion

Avec ce guide, vous pouvez orchestrer plusieurs scripts Python différents à l'aide d'un script Bash maître, assurant ainsi une meilleure gestion des tâches et une intégration facile dans un

environnement de production.