

PRÉSENTATION DE SPARK

QU'EST-CE QUE SPARK ?

Apache Spark est un moteur de traitement de données à grande échelle. Il permet le traitement rapide de grands ensembles de données en distribuant les tâches sur plusieurs ordinateurs.

LES COMPOSANTS DE SPARK

- **Spark Core** : Le cœur du moteur, gère les tâches de base de traitement de données.
- **Spark SQL** : Module pour travailler avec des données structurées.
- **Spark Streaming** : Permet le traitement de données en temps réel.
- **MLlib** : Bibliothèque pour l'apprentissage automatique.
- **GraphX** : Pour le traitement de graphes.

AVANTAGES DE SPARK

- **Vitesse** : Traite les données jusqu'à 100 fois plus vite que Hadoop MapReduce.
- **Facilité d'utilisation** : Offre des API en Scala, Java, Python et R.
- **Polyvalence** : Combine SQL, streaming, et complex analytics.
- **Optimisé** : Supporte le calcul en mémoire.

CAS D'UTILISATION DE SPARK

- **Analyse de données en temps réel** : Surveillance et réaction immédiate.
- **Machine Learning** : Modélisation prédictive et classification.
- **Traitement de graphes** : Analyse des réseaux sociaux.
- **Traitement de logs** : Analyse de logs pour la surveillance ou le diagnostic.

PRÉSENTATION DE HIVE

QU'EST-CE QUE HIVE ?

Apache Hive est un système de gestion de données construit sur Hadoop, permettant la simplification des requêtes et la gestion de grands ensembles de données. Il est conçu pour faciliter la lecture, l'écriture et la gestion de grands volumes de données en utilisant SQL.

HISTORIQUE DE HIVE

Hive a été initialement développé par Facebook en 2007 pour gérer leurs énormes volumes de données. Il a été ensuite rendu open-source et est maintenant géré par la Apache Software Foundation depuis 2010.

ARCHITECTURE DE HIVE

- **Hive Metastore** : Stocke les métadonnées.
- **Driver** : Gère les sessions et les requêtes.
- **Compiler** : Traduit les requêtes SQL en plans d'exécution.
- **Executor** : Exécute les plans d'exécution sur Hadoop.
- **HDFS** : Stockage de données.

FONCTIONNALITÉS CLÉS DE HIVE

- **HiveQL** : Langage de requête similaire à SQL.
- **Support du stockage** : Compatible avec HDFS et d'autres systèmes de fichiers.
- **Optimisation** : Optimisation automatique des requêtes.
- **Extensibilité** : Supporte l'ajout de fonctions personnalisées.

CAS D'UTILISATION DE HIVE

- **Analyse de données** : Utilisé pour l'analyse de grands ensembles de données.
- **Reporting** : Génération de rapports basés sur de grandes quantités de données.
- **Data Warehousing** : Utilisé comme entrepôt de données pour gérer, stocker et récupérer de grandes quantités de données.

INSTALLATION DE SPARK ET HIVE

PRÉREQUIS POUR L'INSTALLATION DE SPARK ET HIVE

- Système d'exploitation supporté (Linux, Mac OS, Windows)
- Java JDK 8 ou supérieur
- Scala 2.12 ou supérieur
- Python 2.7 ou Python 3.4+
- Hadoop 2.7 ou version ultérieure pour Hive

TÉLÉCHARGEMENT DE SPARK

1. Accédez au site officiel de Spark : [Apache Spark](#)
2. Choisissez la version de Spark
3. Sélectionnez le package pré-construit pour Hadoop
4. Téléchargez le fichier tar.gz

TÉLÉCHARGEMENT DE HIVE

1. Visitez le site officiel de Hive : [Apache Hive](#)
2. Sélectionnez la version de Hive adaptée à votre version de Hadoop
3. Téléchargez le fichier tar.gz

CONFIGURATION DE SPARK

1. Décompressez le fichier téléchargé de Spark
2. Configurez les variables d'environnement :
 - **SPARK_HOME**
 - Ajoutez `$SPARK_HOME/bin` au PATH
3. Modifiez le fichier `conf/spark-env.sh` (copiez depuis `spark-env.sh.template`)

CONFIGURATION DE HIVE

1. Décompressez le fichier téléchargé de Hive
2. Configurez les variables d'environnement :
 - `HIVE_HOME`
 - Ajoutez `$HIVE_HOME/bin` au `PATH`
3. Modifiez le fichier `conf/hive-site.xml` pour configurer le stockage et la connexion à Hadoop

INTÉGRATION DE SPARK AVEC HIVE

1. Assurez-vous que Hive et Hadoop sont correctement configurés
2. Dans Spark, activez le support Hive :
 - Modifiez `spark-defaults.conf` et ajoutez `spark.sql.catalogImplementation=hive`
3. Redémarrez Spark pour appliquer les modifications

VÉRIFICATION DE L'INSTALLATION

1. Lancez Spark Shell :

```
$SPARK_HOME/bin/spark-shell
```

2. Exéutez des commandes SQL pour tester l'intégration avec Hive :

```
spark.sql("SHOW TABLES").show()
```

3. Vérifiez la sortie pour confirmer l'accès aux tables Hive

ARCHITECTURE DE SPARK

COMPOSANTS PRINCIPAUX DE SPARK

Spark est composé de plusieurs éléments clés :

- **Driver Program** : Coordonne les processus.
- **Spark Context** : Connecte les services de Spark.
- **Executors** : Exécutent le code.
- **Cluster Manager** : Gère les ressources.

FONCTIONNEMENT DE SPARK EN CLUSTER

1. Le **Driver** divise l'application en tâches.
2. Le **Cluster Manager** alloue des ressources.
3. Les **Executors** exécutent les tâches.
4. Les résultats sont renvoyés au **Driver**.

RÔLE DU DRIVER ET DES EXECUTORS

- **Driver :**

- Planifie les tâches.
- Gère les méta-données de l'application.

- **Executors :**

- Exécutent les tâches.
- Gèrent le stockage et le calcul des données.

GESTION DE LA MÉMOIRE ET DES RESSOURCES

- **Mémoire** : Partitionnée entre les tâches.
- **CPU** : Allocation dynamique selon les besoins.
- **Stockage** : Utilisation de mémoire cache pour l'accès rapide.

INTÉGRATION AVEC HADOOP ET LE SYSTÈME DE FICHIERS HDFS

- **Hadoop** : Spark peut exécuter des tâches sur des clusters Hadoop.
- **HDFS** : Utilisé pour stocker de grandes quantités de données.
- **Compatibilité** : Spark fonctionne bien avec les outils de l'écosystème Hadoop.

ARCHITECTURE DE HIVE

COMPOSANTS DE HIVE

- **Hive Metastore** : Stocke les métadonnées des tables et des partitions.
- **Driver** : Gère la réception des requêtes et leur exécution.
- **Compiler** : Traduit les requêtes HiveQL en plans d'exécution.
- **Executor** : Exécute les tâches et les plans d'exécution.
- **Hive Server** : Interface permettant aux clients d'interagir avec Hive.

INTERACTION ENTRE HIVE ET HADOOP

- **Stockage des données** : Hive stocke les données dans HDFS (Hadoop Distributed File System).
- **Exécution des tâches** : Utilise MapReduce ou Tez pour traiter les requêtes.
- **Utilisation de YARN** : Gestion des ressources et planification des tâches sur le cluster Hadoop.

ARCHITECTURE DE STOCKAGE DE HIVE

- **Tables gérées par Hive** : Stockage et gestion directe par Hive.
- **Tables externes** : Référence à des données stockées en dehors de Hive.
- **Partitions** : Amélioration des performances en subdivisant les tables selon certaines colonnes.
- **Buckets** : Division des données d'une partition en groupes plus gérables.

PROCESSUS DE REQUÊTE DANS HIVE

1. **Soumission de la requête** : L'utilisateur soumet une requête HiveQL.
2. **Compilation** : La requête est compilée en un plan d'exécution.
3. **Optimisation** : Le plan est optimisé pour améliorer l'efficacité.
4. **Exécution** : Le plan est exécuté en utilisant MapReduce ou Tez.
5. **Résultats** : Les résultats sont renvoyés à l'utilisateur.

INTRODUCTION AUX DATA LAKES

DÉFINITION D'UN DATA LAKE

Un Data Lake est un système de stockage centralisé qui permet de stocker de grandes quantités de données brutes dans leur format natif jusqu'à ce qu'elles soient nécessaires. Contrairement aux entrepôts de données traditionnels, un Data Lake peut contenir des données structurées, semi-structurées et non structurées.

AVANTAGES DES DATA LAKES

- **Flexibilité** : Capacité à stocker divers types de données.
- **Évolutivité** : Gestion efficace de grandes quantités de données.
- **Coût-efficacité** : Utilisation de solutions de stockage à faible coût.
- **Analyse avancée** : Supporte l'analyse de données complexes pour de meilleurs insights.

DIFFÉRENCES ENTRE DATA LAKE ET DATA WAREHOUSE

Critère	Data Lake	Data Warehouse
Type de données	Structurées, semi-structurées, non-structurées	Principalement structurées
Flexibilité	Haute, format de données natif	Limitée, nécessite des schémas fixes
Utilisateurs	Data scientists, analystes de données	Utilisateurs business, rapporteurs
Processus	ELT (Extract, Load, Transform)	ETL (Extract, Transform, Load)

COMPOSANTS D'UN DATA LAKE

- **Zone d'atterrissement** : Stockage initial des données brutes.
- **Zone de travail** : Espace pour l'analyse et le traitement des données.
- **Zone de données curées** : Données transformées et validées pour des usages spécifiques.
- **Catalogue de données** : Métadonnées pour la gestion et la découverte des données.

EXEMPLES D'UTILISATION DES DATA LAKES

- **Analyse prédictive** : Utilisation des données pour prévoir les tendances et comportements futurs.
- **Machine Learning** : Entraînement de modèles sur de vastes ensembles de données diverses.
- **Sécurité des données** : Analyse des logs pour détecter les menaces et les anomalies.
- **Recherche personnalisée** : Amélioration des recommandations basées sur l'analyse des données des utilisateurs.

BASES DE DONNÉES HADOOP

ARCHITECTURE DE HADOOP

Hadoop est une architecture de traitement distribué qui permet de stocker et de traiter de grandes quantités de données de manière efficace. Elle se compose de plusieurs composants clés :

- **HDFS (Hadoop Distributed File System)** : pour le stockage de données.
- **MapReduce** : un modèle de traitement pour analyser les données.
- **YARN (Yet Another Resource Negotiator)** : pour la gestion des ressources du cluster.

HDFS (HADOOP DISTRIBUTED FILE SYSTEM)

HDFS est le système de fichiers distribué utilisé par Hadoop. Il est conçu pour stocker de très grands fichiers en les découplant en blocs de données, répartis sur plusieurs machines :

- **Haute disponibilité** : RéPLICATION des données sur plusieurs nœuds.
- **Évolutivité** : Facile à étendre en ajoutant plus de nœuds.
- **Faute tolérance** : Capable de continuer à fonctionner en cas de défaillance de certains nœuds.

YARN (YET ANOTHER RESOURCE NEGOTIATOR)

YARN est le gestionnaire de ressources de Hadoop qui permet :

- **Allocation dynamique** : Ressources attribuées aux applications selon les besoins.
- **Gestion multi-utilisateur** : Supporte plusieurs utilisateurs et applications simultanément.
- **Optimisation des ressources** : Maximise l'utilisation des ressources du cluster.

MAPREDUCE

MapReduce est un modèle de programmation pour le traitement de données sur Hadoop :

- **Map (Mapper)** : Traite les données en entrée et produit des paires clé/valeur.
- **Reduce (Reducer)** : Aggrège les résultats intermédiaires pour produire le résultat final.
- **Parallélisme** : Exécution en parallèle sur plusieurs nœuds pour une meilleure performance.

RDD (RESILIENT DISTRIBUTED DATASET)

DÉFINITION DE RDD

Les RDD (Resilient Distributed Dataset) sont des collections distribuées d'objets, immuables et tolérantes aux pannes, qui permettent de réaliser des calculs en parallèle. Ils sont la pierre angulaire de Apache Spark.

CRÉATION D'UN RDD

```
# Création d'un RDD à partir d'une liste
rdd = sc.parallelize([1, 2, 3, 4, 5])

# Création d'un RDD à partir d'un fichier
rdd = sc.textFile("path/to/file.txt")
```

OPÉRATIONS SUR LES RDD

Type d'opération	Exemple de méthode
Transformation	<code>rdd.map(lambda x: x * 2)</code>
Action	<code>rdd.collect()</code>
Filtre	<code>rdd.filter(lambda x: x > 3)</code>
Agrégation	<code>rdd.reduce(lambda x, y: x + y)</code>

PERSISTENCE DES RDD

Niveau de persistence	Code
Mémoire	<code>rdd.cache()</code>
Mémoire et disque	<code>rdd.persist(StorageLevel.MEMORY_AND_DISK)</code>
Disque seulement	<code>rdd.persist(StorageLevel.DISK_ONLY)</code>

La persistance permet de stocker les RDD intermédiaires utilisés fréquemment pour améliorer la performance des calculs.

DATAFRAMES ET DATASETS EN SPARK

CRÉATION DE DATAFRAMES

Pour créer un DataFrame en Spark, vous pouvez utiliser plusieurs méthodes, telles que :

- Lecture de données à partir de fichiers (CSV, JSON, etc.)
- Conversion d'un RDD existant
- À partir d'une base de données via JDBC

Exemple de création à partir d'un fichier CSV :

```
val spark = SparkSession.builder.appName("ExempleDataFrame").getOrCreate()
val df = spark.read.csv("chemin/vers/fichier.csv")
```

OPÉRATIONS SUR LES DATAFRAMES

Les DataFrames en Spark supportent de nombreuses opérations, telles que :

- `select` : pour sélectionner des colonnes spécifiques
- `filter` : pour filtrer les données
- `groupBy` : pour regrouper les données
- `sort` : pour trier les données
- `join` : pour joindre deux DataFrames

Exemple d'utilisation de `filter` :

```
df.filter($"age" > 18)
```

CONVERSION ENTRE RDD ET DATAFRAME

Conversion d'un RDD en DataFrame :

```
val rdd = sc.parallelize(Seq(("Alice", 1), ("Bob", 2)))
val df = rdd.toDF("name", "id")
```

Conversion d'un DataFrame en RDD :

```
val rddFromDf = df.rdd
```

INTRODUCTION AUX DATASETS

Les Datasets en Spark sont une extension typée des DataFrames qui fournissent des vérifications de type en temps de compilation et un meilleur support pour les données complexes.

Exemple de création d'un Dataset :

```
val dataset = spark.createDataset(Seq(1, 2, 3))
```

MANIPULATION DES DATASETS

Les Datasets permettent des manipulations similaires aux DataFrames mais avec des fonctions typées, par exemple :

- map
- flatMap
- filter
- groupBy

Exemple de `map` sur un Dataset :

```
val ds = dataset.map(_ + 1)
```

COMPARAISON ENTRE DATAFRAMES ET DATASETS

Critère	DataFrames	Datasets
Type-safety	Non typé à la compilation	Typé à la compilation
Performance	Optimisé pour les grandes données	Moins optimisé que DataFrames
Facilité d'utilisation	Moins de contraintes de type	Nécessite une connaissance des types
Utilisation	Idéal pour le traitement de grandes quantités de données	Adapté pour les données complexes avec des types nécessaires

TABLES ET BASES DE DONNÉES EN HIVE

CRÉATION DE BASES DE DONNÉES HIVE

Pour créer une base de données dans Hive, utilisez la commande suivante :

```
CREATE DATABASE nom_de_la_base;
```

Cette commande crée une nouvelle base de données. Si la base existe déjà, Hive renvoie une erreur sauf si vous ajoutez **IF NOT EXISTS**.

CRÉATION DE TABLES HIVE

Pour créer une table dans Hive, utilisez la syntaxe suivante :

```
CREATE TABLE nom_de_la_table (colonne1 type1, colonne2 type2, ...)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ',';
```

Cette commande crée une table avec les colonnes spécifiées.

TYPES DE TABLES EN HIVE (INTERNES ET EXTERNES)

- **Tables internes** : Les données sont stockées dans le warehouse de Hive. Si la table est supprimée, les données le sont aussi.
- **Tables externes** : Les données restent à l'emplacement spécifié même après la suppression de la table.

```
CREATE EXTERNAL TABLE nom_de_la_table (...);
```

GESTION DES SCHÉMAS DE TABLES

Modifier le schéma d'une table existante :

```
ALTER TABLE nom_de_la_table CHANGE colonne ancien_type nouveau_type;
```

Ajouter une nouvelle colonne :

```
ALTER TABLE nom_de_la_table ADD COLUMNS (nouvelle_colonne type);
```

CHARGEMENT DE DONNÉES DANS LES TABLES HIVE

Pour charger des données depuis un fichier dans Hive :

```
LOAD DATA LOCAL INPATH 'chemin_du_fichier' INTO TABLE nom_de_la_table;
```

Cette commande ajoute des données à la table depuis un fichier local.

VISUALISATION ET DESCRIPTION DES TABLES ET BASES DE DONNÉES

Pour voir les tables dans une base de données :

```
SHOW TABLES IN nom_de_la_base;
```

Pour décrire la structure d'une table :

```
DESCRIBE nom_de_la_table;
```

REQUÊTES HIVEQL

SYNTAXE DE BASE DE HIVEQL

HiveQL est le langage de requête utilisé par Hive pour manipuler des données dans un DataLake. Voici sa syntaxe de base :

```
SELECT colonne1, colonne2 FROM table WHERE condition;
```

SÉLECTION ET FILTRAGE DE DONNÉES

Pour sélectionner et filtrer des données dans HiveQL, utilisez :

```
SELECT colonne1, colonne2 FROM table WHERE condition;
```

Exemple :

```
SELECT nom, age FROM employes WHERE age > 30;
```

JOINTURES DE TABLES

Les jointures permettent de combiner des données de plusieurs tables. Syntaxe :

```
SELECT t1.colonne1, t2.colonne2
FROM table1 t1
JOIN table2 t2 ON t1.id = t2.id;
```

AGRÉGATION ET GROUPEMENT

Pour agréger des données et les grouper par catégorie :

```
SELECT COUNT(*), categorie
FROM table
GROUP BY categorie;
```

TRI ET LIMITATION DE RÉSULTATS

Pour trier et limiter les résultats dans HiveQL :

```
SELECT colonnel FROM table
ORDER BY colonnel DESC
LIMIT 10;
```

GESTION DES DONNÉES AVEC HIVE

CRÉATION DE TABLES HIVE

Pour créer une table dans Hive, utilisez la syntaxe suivante :

```
CREATE TABLE nom_table (
    colonne1 type_donnee,
    colonne2 type_donnee,
    ...
) STORED AS format_fichier;
```

Exemples de formats de fichier : TEXTFILE, PARQUET, ORC.

CHARGEMENT DE DONNÉES DANS HIVE

Pour charger des données dans une table Hive :

```
LOAD DATA INPATH 'chemin_du_fichier' INTO TABLE nom_table;
```

Vous pouvez charger des données depuis un fichier local ou HDFS.

GESTION DES MÉTADONNÉES

Hive stocke les métadonnées (structure de table, emplacement des données) dans une base de données relationnelle appelée Metastore. Utilisez les commandes suivantes pour gérer les métadonnées :

- `SHOW TABLES;` : affiche toutes les tables.
- `DESCRIBE nom_table;` : affiche la structure d'une table.

OPTIMISATION DES REQUÊTES HIVE

Techniques d'optimisation :

1. **Partitionnement** : Divise les données en partitions séparées basées sur les valeurs d'une colonne spécifique.
2. **Buckétisation** : Divise les données en un nombre fixe de buckets, ce qui peut améliorer les jointures.
3. **Indexation** : Crée des index sur les colonnes pour accélérer les requêtes.

SÉCURITÉ DES DONNÉES DANS HIVE

Hive offre plusieurs mécanismes pour sécuriser les données :

- **Contrôle d'accès basé sur les rôles (RBAC)** : Gère les permissions d'accès aux tables.
- **Intégration avec Kerberos** : Authentification pour les utilisateurs accédant au cluster.
- **Chiffrement** : Chiffrement des données stockées et en transit.

INTÉGRATION DE SPARK AVEC HIVE

CONFIGURATION DE SPARK POUR UTILISER HIVE

Pour utiliser Hive avec Spark, assurez-vous que Hive est correctement configuré et que le fichier `hive-site.xml` est accessible par Spark.

```
# Configuration de SparkSession pour intégrer Hive
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Hive Integration") \
    .config("spark.sql.warehouse.dir", "/user/hive/warehouse") \
    .enableHiveSupport() \
    .getOrCreate()
```

CRÉATION DE DATAFRAMES À PARTIR DE TABLES HIVE

Vous pouvez accéder aux tables Hive directement en utilisant Spark SQL pour créer des DataFrames.

```
# Création d'un DataFrame à partir d'une table Hive
df = spark.sql("SELECT * FROM ma_table_hive")
```

UTILISATION DE SPARK SQL POUR INTERROGER HIVE

Spark SQL permet d'exécuter des requêtes SQL directement sur des tables stockées dans Hive.

```
# Exécution d'une requête SQL sur une table Hive
resultat = spark.sql("SELECT nom, age FROM utilisateurs WHERE age > 30")
resultat.show()
```

LECTURE ET ÉCRITURE DE DONNÉES DANS HIVE AVEC SPARK

Spark facilite la lecture et l'écriture des données dans les tables Hive.

```
# Écriture dans une table Hive
nouveau_df.write.mode("overwrite").saveAsTable("ma_nouvelle_table_hive")

# Lecture d'une table Hive
df_lecture = spark.table("ma_nouvelle_table_hive")
df_lecture.show()
```

EXEMPLES DE TRAITEMENT DE DONNÉES

FILTRAGE DE DONNÉES

Le filtrage permet de sélectionner des données spécifiques à partir d'un ensemble plus large.

```
SELECT * FROM table WHERE condition;
```

Exemple : Sélectionner tous les enregistrements où l'âge est supérieur à 30.

```
SELECT * FROM utilisateurs WHERE age > 30;
```

AGRÉGATION DE DONNÉES

L'agrégation consiste à résumer des données en utilisant des fonctions comme **SUM**, **AVG**, **MAX**, **MIN**, **COUNT**.

```
SELECT AVG(colonne) FROM table GROUP BY colonne_group;
```

Exemple : Calculer la moyenne des salaires par département.

```
SELECT departement, AVG(salaire) FROM employes GROUP BY departement;
```

JOINTURE DE TABLES

Les jointures permettent de combiner des lignes de deux ou plusieurs tables basées sur une colonne relationnelle commune.

```
SELECT * FROM table1 JOIN table2 ON table1.common_field = table2.common_field;
```

Exemple : Associer les données des employés avec leur département.

```
SELECT employes.nom, departements.nom FROM employes JOIN departements ON employes.dep_id = departements.id;
```

TRI DES DONNÉES

Le tri des données permet d'organiser les enregistrements selon une ou plusieurs colonnes.

```
SELECT * FROM table ORDER BY colonne ASC|DESC;
```

Exemple : Trier les employés par salaire décroissant.

```
SELECT * FROM employes ORDER BY salaire DESC;
```

CALCUL DE STATISTIQUES

Calculer des statistiques descriptives pour comprendre mieux les données.

```
SELECT MAX(colonne), MIN(colonne), AVG(colonne) FROM table;
```

Exemple : Statistiques de salaire dans une entreprise.

```
SELECT MAX(salaire), MIN(salaire), AVG(salaire) FROM employes;
```

EXTRACTION DE DONNÉES SPÉCIFIQUES

Extraction de sous-ensembles de données basés sur des critères spécifiques.

```
SELECT colonne1, colonne2 FROM table WHERE condition;
```

Exemple : Extraire le nom et l'âge des employés ayant un salaire supérieur à 5000.

```
SELECT nom, age FROM employes WHERE salaire > 5000;
```