MMINISTRY OF HIGHER EDUCATION
SCIENTIFIC RESEARCH AND INNOVATION

SULTAN MOULAY SLIMANE UNIVERSITY

NATIONAL SCHOOL OF APPLIED SCIENCES
KHOURIBGA

End of studies dissertation

for obtaining the

State engineer diploma

Sector : Computer Science and Data Engineering

Presented by :
Ziyad **RAKIB**

Thesis title

Supervised by :
Abdelghani **Ghazdali**
Nidal **Lamghari**

Jury members :

Chairman's surname and first name	Entity	Chairman
Surname and first name of the examiner	Entity	Examiner
Surname and first name of the reporter	Entity	Reporter
Surname and first name of the reporter	Entity	Reporter

Academic Year : **2023**

Sommaire

Dedication

“

À ma mère, qui m'a comblée de son soutien et dévouée moi avec un amour inconditionnel. Tu es pour moi un exemple de courage et sacrifice continuel. Que cet humble travail porte témoigner de mon affection, de mon attachement éternel et qu'il appelle sur moi ta bénédiction continuelle.,

À mon père, aucune dédicace ne peut exprimer l'amour, l'estime, dévouement et le respect que j'ai toujours eu pour vous. Rien dans le monde vaut les efforts faits jour et nuit pour mon éducation et mon bien-être. Ce travail est le fruit de vos sacrifices que vous avez fait pour mon éducation et ma formation,

À mes chers frères, merci pour votre amour, soutien, et encouragements,

À tous mes chers amis, pour le soutien que vous m'avez apporté, je dis Merci encore une fois à tous ceux qui me sont chers, à vous tous

Merci.

”

- Ziyad

Acknowledgements

Tout d'abord, je remercie le grand Dieu puissant de nous donné la puissance pour continuer et dépasser toutes les difficultés.

J'adresse mes remerciements à, **Monsieur Abdelghani Ghazdali**, chef de filière Informatique et Ingénierie des Données qui a su assurer le bon déroulement des séances d'encadrement du début à la fin.

Je tiens également à adresser mes sincères remerciements à **Professeur Nidal Lamghari**, mon encadrante interne, pour tous les conseils qu'elle m'a prodigués ainsi que ses encouragements.

Au terme de ce travail, je tiens également à exprimer mes sincères remerciements et ma gratitude envers tous ceux qui, par leur enseignement, par leur soutien et leurs conseils, ont contribué au déroulement de ce projet.

J'adresse mes remerciements à **Monsieur Reda EL Mejad**, PDG et co-fondateur de Izicap et lauréat de l'ENSA, pour l'opportunité de passer mon stage au sein de cet entreprise.

Je tiens à remercier mon encadrant à Izicap, **Monsieur Bilal SLAYKI** qui m'a supervisée avec patience et n'a épargné aucun effort pour mettre à ma disposition les explications nécessaires et les directives précieuses. Son assistance et ses conseils permanents m'ont été un apport remarquable. Je veux remercier aussi Nasr, Youness et Anas; l'équipe avec laquelle j'ai travaillé et qui ont généreusement contribué à ce travail.

Je remercie également l'agente RH **Madame Sanaa ARROUCHE** pour ses bons conseils car elle est toujours là pour accéder à nos demandes et répondre au mieux à nos questions.

Un remerciement particulier aux membres du jurys qui m'ont honoré en acceptant de juger ce travail et de me faire profiter de leurs remarques et conseils.

Finalement, mes remerciements s'adressent aussi à l'ensemble du corps professoral et administratif de l'ENSA Khouribga pour l'effort qu'ils fournissent afin de nous garantir une bonne formation et à l'équipe administrative et technique pour tous les services offerts.

Résumé

Ce rapport fournit une analyse approfondie du projet complexe entrepris pour faire évoluer la source de données de Izicap. Le projet a nécessité la suppression de MariaDB et la mise en place de Delta Lake, une solution de stockage et de traitement de données hautes performances. De plus, l'architecture monolithique existante a été divisée en microservices utilisant Spring Boot comme backend avec un connecteur approprié pour Delta Lake, et les micro-frontends ReactJS comme frontend.

Le projet présentait de nombreux défis qui nécessitaient l'application de technologies et de stratégies avancées. Celles-ci comprenaient la migration des données, la garantie de la cohérence et de l'exactitude des données, la gestion de la complexité des systèmes distribués et l'intégration de diverses technologies et services. Le rapport décrit les différentes solutions développées pour relever ces défis, notamment l'utilisation d'algorithmes de traitement de données avancés, d'architectures informatiques distribuées et de la conteneurisation.

Malgré les défis rencontrés, le projet reste un travail en cours. Le rapport donne un aperçu des efforts en cours pour améliorer l'évolutivité, les performances et la fonctionnalité globale du projet.

Mots clés : Projet, Source de données, Delta Lake, Monolith, Microservice, Spring Boot, ReactJS, Micro-Frontend, Back-End, Front-End, Données, Évolutivité, Performance.

Abstract

This report provides an in-depth analysis of the complex project undertaken to scale the data source of Izicap. The project required the removal of MariaDB and implementation of Delta Lake, a high-performance data storage and processing solution. In addition, the existing monolithic architecture was divided into microservices utilizing Spring Boot as the backend with a suitable connector to Delta Lake, and ReactJS micro-frontends as the frontend.

The project presented numerous challenges that required the application of advanced technologies and strategies. These included data migration, ensuring data consistency and accuracy, managing the complexities of distributed systems, and integrating various technologies and services. The report outlines the various solutions developed to address these challenges, including the use of advanced data processing algorithms, distributed computing architectures, and containerization.

Despite the challenges encountered, the project remains a work in progress. The report provides insights into the ongoing efforts to improve the project's scalability, performance, and overall functionality.

Keywords: Project, Data Source, Delta Lake, Monolith, Microservice, Springboot, ReactJS, Micro-frontends, Back-End, Front-End, Data, Scalability, Performance.

ملخص

يقدم هذا التقرير تحليلاً متعمقًا للمشروع المعقد الذي تم إجراؤه لتوسيع نطاق مصدر بيانات الشركة. يتطلب المشروع إزالة MariaDB وتنفيذ Delta Lake ، وهو حل تخزين ومعالجة بيانات عالي الأداء. بالإضافة إلى ذلك ، تم تقسيم المونوليث الحالي إلى خدمات مصغرة باستخدام Spring Boot كواجهة خلفية مع موصل مناسب لـ Delta
Lake ، وواجهة ReactJS كواجهة أمامية. قدم المشروع العديد من التحديات التي تطلبت تطبيق التقنيات والاستراتيجيات المتقدمة. وشمل ذلك ترحيل البيانات ، وضمان اتساق البيانات ودقتها ، وإدارة تعقيدات الأنظمة الموزعة ، ودمج التقنيات والخدمات المختلفة. يحدد التقرير الحلول المختلفة التي تم تطويرها لمواجهة هذه التحديات ، بما في ذلك استخدام خوارزميات معالجة البيانات المتقدمة ، وبنى الحوسبة الموزعة ، والحاويات. على الرغم من التحديات التي واجهتها ، لا يزال المشروع قيد التنفيذ. يقدم التقرير رؤى حول الجهود الجارية لتحسين قابلية المشروع للتوسع والأداء والوظائف العامة

كلمات مفتاحية : المشروع، مصدر البيانات، Delta Lake ، المونوليث، الميكروسيرفس، Springboot ، ReactJS ، الميكروواجهات، الخلفية، الواجهة الأمامية، البيانات، القابلية للتوسع، الأداء. Delta Lake ، Springboot ، ReactJS ، الميكروواجهات.

Table des matières

Table des figures

Liste des tableaux

TABLE 1 : List of Abbreviations

API	Application programming interface
VSC	Visual Studio Code
DL	Delta Lake
REST	RepresEntational State Transfer
AWS	Amazon Web Services
GCP	Google Cloud Platform
AZR	Microsoft Azure
S3	Simple Storage Service
IAM	Identity and Access Management
MinIO	Minimal Object Storage
SQL	Structured Query Language
DB	Database
ACID	Atomicity, Consistency, Isolation, Durability
OLTP	Online Transaction Processing
OLAP	Online Analytical Processing
ReactJS	React JavaScript
SPA	Single Page Application
JS	JavaScript
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
UI	User Interface
API	Application Programming Interface
Spark	Apache Spark
Hadoop	Apache Hadoop
HDFS	Hadoop Distributed File System
YARN	Yet Another Resource Negotiator
MapReduce	MapReduce Programming Model
Metadata	Data about Data
ETL	Extract, Transform, Load
ELT	Extract, Load, Transform
CRM	Customer Relationship Management
RDBMS	Relational Database Management System
SPI	Server Provider Interface

General Introduction

Le monde des données continue de se développer à un rythme sans précédent. Cela a conduit au besoin de systèmes de stockage et de gestion de données efficaces capables de suivre cette croissance. Dans notre quête d'une meilleure solution, nous avons rencontré le problème de l'évolutivité avec notre système de gestion de base de données actuel, MariaDB. En conséquence, nous nous sommes lancés dans un projet visant à trouver une source de données évolutive qui répondrait à nos besoins. Après de nombreuses recherches et réflexions, nous avons décidé d'utiliser Delta Lake avec un stockage S3.

Cependant, nous avons rencontré plusieurs défis lors de la mise en œuvre de Delta Lake. L'un des principaux problèmes auxquels nous avons été confrontés était le manque de popularité et de documentation de Delta Standalone. Il s'est également avéré lent et ne pouvait pas être interrogé avec SQL. Par conséquent, nous avons opté pour Trino, un moteur de requête SQL distribué qui pourrait facilement interagir avec Delta Lake.

Un autre défi important était l'architecture monolithique de notre système, qui entravait sa flexibilité et son évolutivité. Par conséquent, nous avons décidé de passer aux microservices pour rendre notre système plus efficace et flexible.

Dans ce rapport, nous donnerons un aperçu de la société Izicap au chapitre 1, suivi d'une explication de Delta Lake et de ses défis de mise en œuvre au chapitre 2. Au chapitre 3, nous discuterons de Trino et de la manière dont il nous a aidés à surmonter les défis auxquels nous étions confrontés. avec le lac Delta.

Chapitre 1

Izicap

Introduction

1.1 About

Izicap est un pionnier dans l'utilisation des données de cartes de paiement et les transforme en une connaissance client et des informations commerciales puissantes pour les commerçants, leur permettant de gérer leurs propres programmes de fidélité et campagnes de marketing numérique. Ces services marketing, fournis par les acquéreurs utilisant les solutions SaaS d'Izicap, redonnent rapidement de la croissance aux entreprises marchandes en renforçant les dépenses et la fidélité de leurs clients (les titulaires de carte). La solution innovante de CRM et de fidélité liée aux cartes d'Izicap donne aux acquéreurs un avantage concurrentiel en monétisant leurs données de transactions de paiement, en générant de nouvelles sources de revenus et en améliorant leurs capacités de rétention. Après s'être solidement implanté en France grâce à des partenariats avec le Groupe BPCE et le Crédit Agricole, Izicap s'est associé à Nexi, le premier acquéreur et Fintech en Italie et a rejoint le programme StartPath de Mastercard dans le but d'étendre considérablement sa portée mondiale. Izicap s'associe aux principaux fournisseurs de solutions de paiement tels qu'Ingenico, Verifone, Poynt et PAX, et rend sa solution CRM et Fidélité liée à la carte disponible sur les terminaux de paiement les plus populaires et les plus innovantes.

1.2 Organigramme

informations sur Izicap

Conclusion

Chapitre 2

Delta Lake

Introduction

Les data warehouses et les lacs de données sont les référentiels de données centraux les plus couramment utilisés par la plupart des organisations axées sur les données aujourd'hui, chacun avec ses propres forces et compromis. D'une part, alors que les data warehouses permettent aux entreprises d'organiser des ensembles de données historiques à utiliser dans la Business Intelligence (BI) et l'analyse, ils deviennent rapidement plus coûteux à mesure que les ensembles de données augmentent en raison de l'utilisation combinée des ressources de calcul et de stockage. De plus, les data warehouses ne peuvent pas gérer la nature variée des données (structurées, non structurées et semi-structurées) observées aujourd'hui.

Dans ce chapitre, nous explorerons les principales caractéristiques de Delta Lake, son fonctionnement et pourquoi il s'agit d'un bon choix pour le traitement du Big Data. Nous fournirons également des exemples d'utilisation de Delta Lake avec d'autres outils de Big Data, tels que Spark et Trino, plus loin dans ce rapport.

2.1 Définition

Delta Lake est une couche de stockage open source construite au-dessus d'un lac de données qui confère fiabilité et transactions ACID (atomicité, cohérence, isolation et durabilité). Il permet une architecture de données continue et simplifiée pour les organisations. Un lac de données stocke les données au format Parquet et permet une architecture de données Lakehouse, qui aide les organisations à créer un système de données unique et continu qui combine les meilleures fonctionnalités de l'entrepôt de données et du lac de données tout en prenant en charge le traitement en continu et par lots.

2.2 Comment fonctionne Delta Lake

Un Delta Lake permet la construction d'une data lakehouse. Les datalakehouses courantes incluent Databricks Lakehouse et Azure Databricks. Cette architecture de données continue permet aux organisations d'exploiter les avantages des data warehouses et des lacs de données avec une complexité et des coûts de gestion réduits. Voici quelques façons dont Delta Lake améliore l'utilisation des data warehouses et des lacs :

- **Permet une architecture Lakehouse :** Delta Lake permet une architecture de données continue et simplifiée qui permet aux organisations de gérer et de traiter d'énormes volumes de données en continu et par lots sans les tracas de gestion et d'exploitation liés à la gestion séparée du streaming, des data warehouses et des lacs de données.
- **Permet une gestion intelligente des données pour les lacs de données :** Delta Lake offre une gestion efficace et évolutive des métadonnées, qui fournit des informations sur les volumes de données massifs dans les lacs de données. Grâce à ces informations, les tâches de gouvernance et de gestion des données se déroulent plus efficacement.
- **Application du schéma pour une meilleure qualité des données :** Étant donné que les lacs de données n'ont pas de schéma défini, il devient facile pour les données mauvaises/incompatibles d'entrer dans les systèmes de données. La qualité des données est améliorée grâce à la validation automatique du schéma, qui valide la compatibilité DataFrame et table avant les écritures.
- **Permet la transaction ACID :** La plupart des architectures de données organisationnelles impliquent de nombreux mouvements ETL et ELT entrant et sortant du stockage de données, ce qui l'ouvre à plus de complexité et d'échecs aux points d'entrée des nœuds. Delta Lake garantit la durabilité et la persistance des données pendant l'ETL et d'autres opérations de données. Delta Lake capture toutes les modifications apportées aux données pendant les opérations de données dans un journal des transactions, garantissant ainsi l'intégrité et la fiabilité des données pendant les opérations de données.

2.3 Diagramme d'architecture du Delta Lake

Delta Lake est une amélioration de l'architecture lambda dans laquelle le traitement par flux et par lots se produit en parallèle et les résultats fusionnent pour fournir une réponse à la requête. Cependant, cette méthode signifie plus de complexité et de difficulté à maintenir et à exploiter à la fois les processus en continu et par lots. Contrairement à l'architecture lambda, Delta Lake est une architecture de données continues qui combine des flux de travail en continu et par lots dans un magasin de fichiers partagé via un pipeline connecté.

Le fichier de données stocké comporte trois couches, les données s'affinant au fur et à mesure qu'elles progressent en aval dans le flux de données :

- **Bronze tables :** Cette table contient les données brutes ingérées à partir de plusieurs sources telles que les systèmes Internet des objets (IoT), les fichiers CRM, RDBMS et JSON.
- **Silver tables :** Cette couche contient une vue plus raffinée de nos données après avoir subi des processus de transformation et d'ingénierie de fonctionnalités.
- **Gold tables :** Cette couche finale est souvent mise à la disposition des utilisateurs finaux dans les rapports et l'analyse BI ou utilisée dans les processus d'apprentissage automatique.

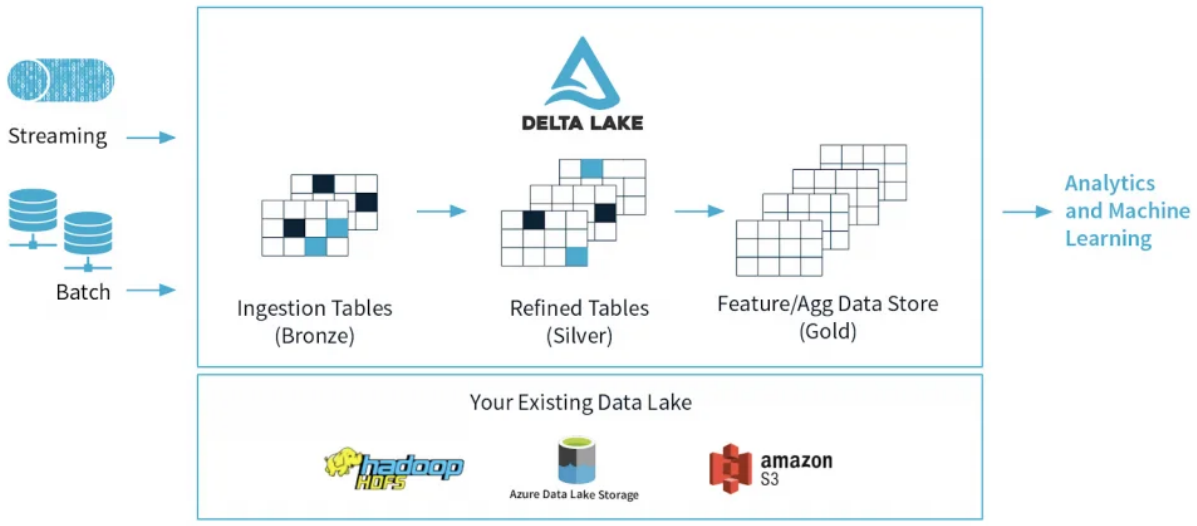

FIGURE 2.1 : Architecture multi-sauts de Delta Lake

2.4 Principaux avantages et caractéristiques de Delta Lake

- **Pistes d'audit et historique :**Dans Delta Lake, chaque écriture existe en tant que transaction et est enregistrée en série dans un journal des transactions. Par conséquent, toutes les modifications ou validations apportées au journal des transactions sont enregistrées, laissant une trace complète à utiliser dans les audits historiques, la gestion des versions ou à des fins de voyage dans le temps. Cette fonctionnalité de Delta Lake permet de garantir l'intégrité et la fiabilité des données pour les opérations de données d'entreprise.
- **Voyager dans le temps et versionner les données :** Étant donné que chaque écriture crée une nouvelle version et stocke l'ancienne version dans le journal des transactions, les utilisateurs peuvent afficher/restaurer les anciennes versions de données en fournissant l'horodatage ou le numéro de version d'une table ou d'un répertoire existant à l'API de lecture Sparks. À l'aide du numéro de version fourni, Delta Lake construit ensuite un instantané complet de la version avec les informations fournies par le journal des transactions. Les retours en arrière et la gestion

des versions jouent un rôle essentiel dans l'expérimentation de l'apprentissage automatique, où les scientifiques des données modifient de manière itérative les hyperparamètres pour former des modèles et peuvent revenir aux modifications si nécessaire.

- **Unifie le traitement par lots et par flux :** Chaque table d'un Delta Lake est un puits de lot et de flux. Avec le streaming structuré Sparks, les organisations peuvent diffuser et traiter efficacement les données de streaming. De plus, grâce à la gestion efficace des métadonnées, à la facilité d'évolutivité et à la qualité ACID de chaque transaction, l'analyse en temps quasi réel devient possible sans utiliser une architecture de données à deux niveaux plus compliquée.

- **Gestion efficace et évolutive des métadonnées :** Delta Lakes stocke les informations de métadonnées dans le journal des transactions et exploite la puissance de traitement distribuée de Spark pour traiter rapidement, lire et gérer efficacement de gros volumes de métadonnées de données, améliorant ainsi la gouvernance des données.

- **transactions ACID :** Delta Lakes garantit que les utilisateurs voient toujours une vue de données cohérente dans une table ou un répertoire. Il garantit cela en capturant chaque modification effectuée dans un journal de transactions et en l'isolant au niveau d'isolation le plus fort, le niveau sérialisable. Au niveau sérialisable, chaque opération existante a et suit une séquence en série qui, lorsqu'elle est exécutée une par une, fournit le même résultat que celui indiqué dans le tableau.

- **Opérations du langage de manipulation de données :** Delta Lakes prend en charge les opérations DML telles que les mises à jour, les suppressions et les fusions, qui jouent un rôle dans les opérations de données complexes telles que la capture de données de modification (CDC), les upserts en continu et la dimension à évolution lente (SCD). Des opérations comme CDC assurent la synchronisation des données dans tous les systèmes de données et minimisent le temps et les ressources consacrés aux opérations ELT. Par exemple, en utilisant le CDC, au lieu d'ETL toutes les données disponibles, seules les données récemment mises à jour depuis la dernière opération subissent une transformation.

- **Schema Enforcement :** Delta Lakes effectue une validation automatique du schéma en vérifiant un ensemble de règles pour déterminer la compatibilité d'une écriture d'un DataFrame vers une table. L'une de ces règles est l'existence de toutes les colonnes DataFrame dans la table cible. Une occurrence d'une colonne supplémentaire ou manquante dans le DataFrame génère une erreur d'exception. Une autre règle est que le DataFrame et la table cible doivent contenir les mêmes types de colonnes, ce qui, sinon, déclenchera une exception. Delta Lake utilise également DDL (Data Definition Language) pour ajouter explicitement de nouvelles colonnes. Cette fonctionnalité de lac de données permet d'éviter l'ingestion de données incorrectes, garantissant ainsi une qualité élevée des données.

- **Compatibilité avec l'API de Spark :** Delta Lake est basé sur Apache Spark et est entièrement compatible avec l'API Spark, qui permet de créer des pipelines de données volumineuses efficaces et fiables.

- **Flexibilité et intégration :** Delta Lake est une couche de stockage open source et utilise le format Parquet pour stocker des fichiers de données, ce qui favorise le partage de données et facilite l'intégration avec d'autres technologies et stimule l'innovation.

2.5 Implémentation

Pour utiliser Delta Lake de manière interactive dans le shell Spark SQL, Scala ou Python, nous avons besoin d'une installation locale d'Apache Spark. Selon que nous voulons utiliser SQL, Python ou Scala, nous pouvons configurer respectivement le shell SQL, PySpark ou Spark.

Spark SQL Shell :

```
bin/spark-sql --packages io.delta:delta-core_2.12:2.3.0
--conf "spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension"
--conf "spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
```

PySpark Shell :

1. Installez la version de PySpark compatible avec la version de Delta Lake en exécutant ce qui suit :

```
pip install pyspark==<compatible-spark-version>
```

2. Exécutez PySpark avec le package Delta Lake et des configurations supplémentaires :

```
pyspark --packages io.delta:delta-core_2.12:2.3.0
--conf "spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension"
--conf "spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
```

Scala Shell :

Téléchargez la version compatible d'Apache Spark en suivant les instructions de Téléchargement de Spark, soit en utilisant pip, soit en téléchargeant et en extrayant l'archive et en exécutant spark-shell dans le répertoire extrait.

```
bin/spark-shell --packages io.delta:delta-core_2.12:2.3.0
--conf "spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension"
--conf "spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
```

Create Table :

Pour créer une table Delta, écrivez un DataFrame au format delta. Nous pouvons utiliser le code Spark SQL existant et changer le format de parquet, csv, json, etc., en delta.

1. **SQL :**

```
CREATE TABLE delta.`/tmp/delta-table` USING DELTA
AS SELECT col1 as id FROM VALUES 0,1,2,3,4;
```

2. **Python :**

```
data = spark.range(0, 5)
data.write.format("delta").save("/tmp/delta-table")
```

3. **Scala :**

```
val data = spark.range(0, 5)
data.write.format("delta").save("/tmp/delta-table")
```

4. **Java :**

```
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;

SparkSession spark = ...   // create SparkSession

Dataset<Row> data = spark.range(0, 5);
data.write().format("delta").save("/tmp/delta-table");
```

Ces opérations créent une nouvelle table Delta en utilisant le schéma qui a été déduit de votre DataFrame

Conclusion

Delta Lake est un outil important pour le traitement du Big Data, fournissant une gestion fiable des données et garantissant l'intégrité des données à grande échelle. Ses transactions ACID, l'application des schémas et les fonctionnalités de gestion des versions des données en font un choix populaire pour les entreprises qui doivent traiter de grandes quantités de données avec une grande précision et fiabilité.

En utilisant Delta Lake, les ingénieurs de données et les scientifiques des données peuvent facilement gérer la qualité des données, suivre la lignée des données et collaborer sur des projets d'analyse de données. Grâce à son intégration transparente avec d'autres outils de Big Data. Delta Lake fournit une solution puissante pour le traitement du Big Data qui peut aider les entreprises à mieux comprendre leurs données plus rapidement et plus efficacement.

Chapitre 3

Trino

Introduction

3.1 Définition

Trino est un moteur de requête SQL distribué open source. Il s'agit d'un hard fork du projet Presto original créé par Facebook. Il permet aux développeurs d'exécuter des analyses interactives sur de gros volumes de données. Avec Trino, les organisations peuvent facilement utiliser leurs compétences SQL existantes pour interroger des données sans avoir à apprendre de nouveaux langages complexes. L'architecture est assez similaire aux systèmes traditionnels de traitement analytique en ligne (OLAP) utilisant des architectures informatiques distribuées, dans lesquelles un nœud de contrôleur coordonne plusieurs nœuds de travail.

3.2 Comment ça fonctionne

Trino est un système distribué qui s'exécute sur Hadoop et utilise une architecture similaire aux bases de données de traitement massivement parallèle (MPP). Il a un nœud coordinateur travaillant avec plusieurs nœuds de travail. Les utilisateurs soumettent SQL au coordinateur qui utilise le moteur de requête et d'exécution pour analyser, planifier et planifier un plan de requête distribué sur les nœuds de travail. Il prend en charge le SQL ANSI standard, y compris les requêtes complexes, les agrégations de jointures et les jointures externes.

Tirant parti de cette architecture, le moteur de requête Trino est capable de traiter des requêtes SQL sur de grandes quantités de données en parallèle sur un cluster d'ordinateurs ou de nœuds. Trino s'exécute en tant que processus à serveur unique sur chaque nœud. Plusieurs nœuds exécutant Trino, qui sont configurés pour collaborer les uns avec les autres, constituent un cluster Trino.

La figure suivante affiche une vue d'ensemble de haut niveau d'un cluster Trino composé d'un coordinateur et de plusieurs nœuds de travail. Un utilisateur Trino se connecte au coordinateur avec

un client, tel qu'un outil utilisant le pilote JDBC ou la CLI Trino. Le coordinateur collabore ensuite avec les workers, qui accèdent aux sources de données.

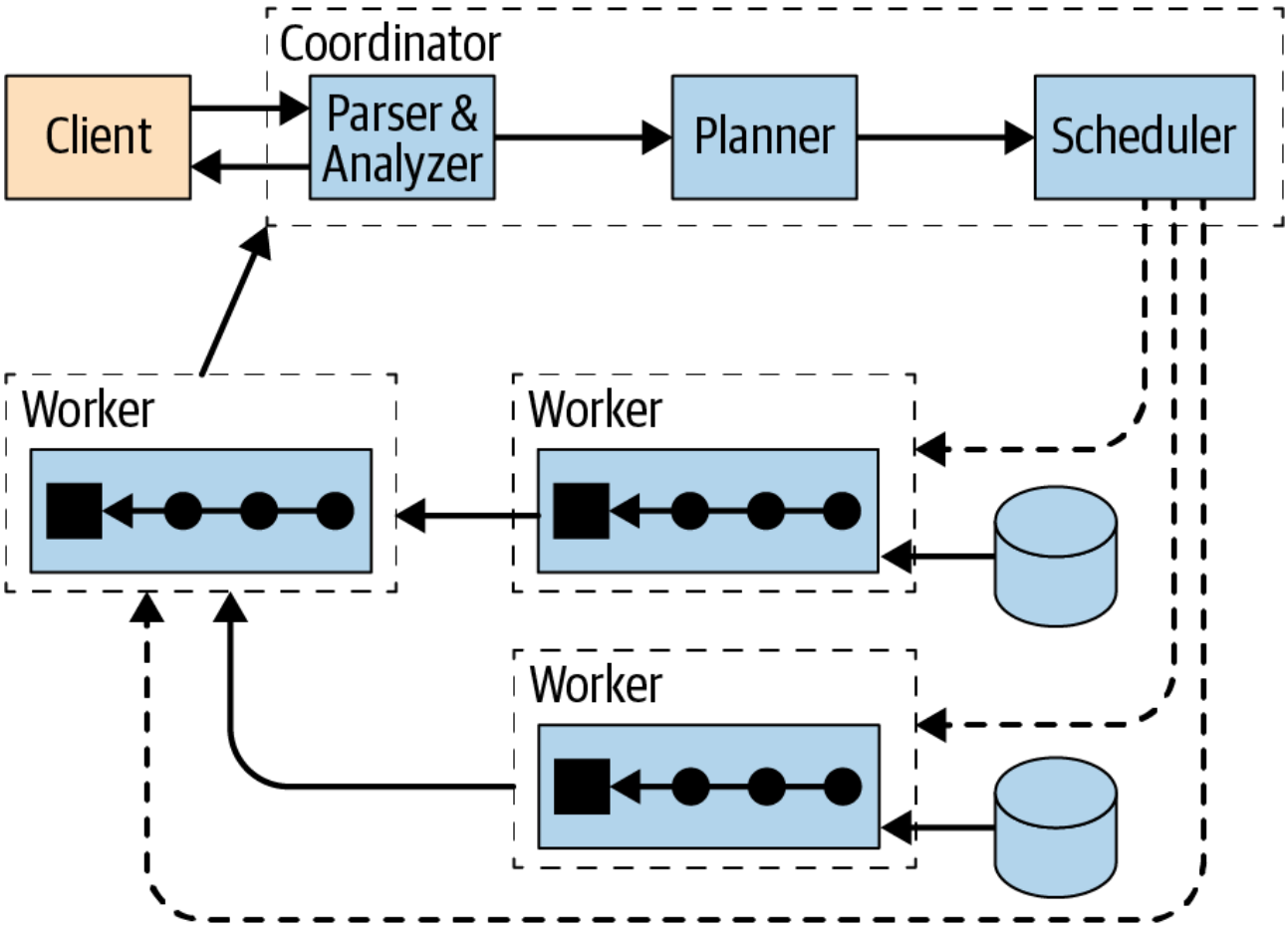

FIGURE 3.1 : Vue d'ensemble de l'architecture Trino avec le coordinateur et les workers

1. Un coordinateur est un serveur Trino qui gère les requêtes entrantes et gère les workers pour exécuter les requêtes.
2. Un worker est un serveur Trino responsable de l'exécution des tâches et du traitement des données.
3. Le service de découverte s'exécute généralement sur le coordinateur et permet aux workers de s'inscrire pour participer au cluster.
4. Toutes les communications et tous les transferts de données entre les clients, le coordinateur et les workers utilisent des interactions basées sur REST sur HTTP/HTTPS.

La figure suivante montre comment la communication au sein du cluster se produit entre le coordinateur et les workers, ainsi que d'un worker à l'autre. Le coordinateur discute avec les workers pour attribuer le travail, mettre à jour le statut et récupérer l'ensemble de résultats de niveau supérieur à renvoyer aux utilisateurs. Les workers se parlent pour récupérer des données à partir de tâches en amont, exécutées sur d'autres workers. Et les workers récupèrent les ensembles de résultats à partir de la source de données.

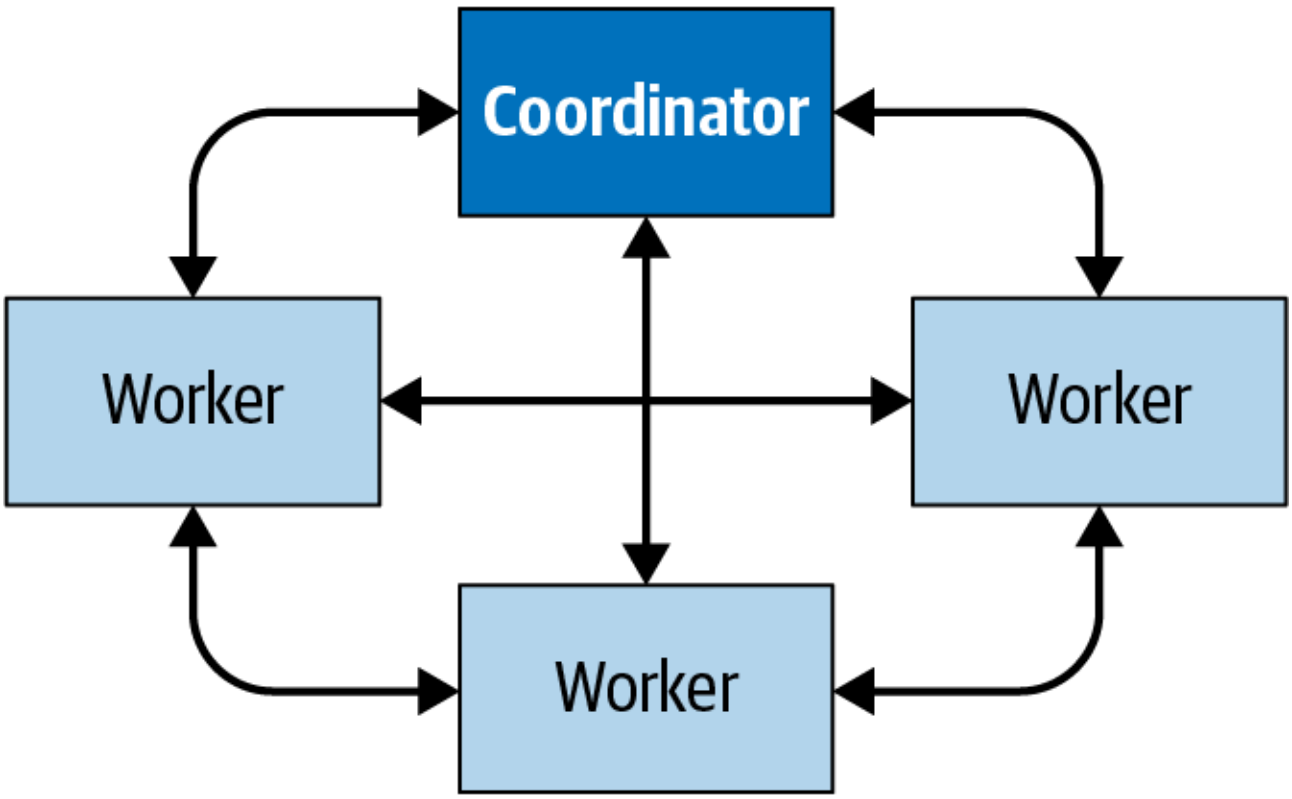

FIGURE 3.2 : Communication entre le coordinateur et les workers dans un cluster Trino

3.3 Coordinateur

Le coordinateur Trino est le serveur responsable de la réception des instructions SQL des utilisateurs, de l'analyse de ces instructions, de la planification des requêtes et de la gestion des nœuds de travail. C'est le cerveau d'une installation Trino et le nœud auquel un client se connecte. Les utilisateurs interagissent avec le coordinateur via la CLI Trino, les applications utilisant les pilotes JDBC ou ODBC, ou toute autre bibliothèque client disponible pour une variété de langues. Le coordinateur accepte les instructions SQL du client telles que les requêtes SELECT pour l'exécution.

Chaque installation Trino doit avoir un coordinateur aux côtés d'un ou plusieurs workers. À des fins de développement ou de test, une seule instance de Trino peut être configurée pour remplir les deux rôles.

Le coordinateur suit l'activité de chaque worker et coordonne l'exécution d'une requête. Le coordinateur crée un modèle logique d'une requête impliquant une série d'étapes.

Une fois qu'il reçoit une instruction SQL, le coordinateur est responsable de l'analyse, de l'analyse, de la planification et de la planification de l'exécution de la requête sur les nœuds de travail Trino. L'instruction est traduite en une série de tâches connectées s'exécutant sur un cluster de workers. Au fur et à mesure que les workers traitent les données, les résultats sont récupérés par le coordinateur et exposés aux clients sur un tampon de sortie. Une fois qu'un tampon de sortie est complètement lu par le client, le coordinateur demande plus de données aux workers au nom du client. Les workers, à leur tour, interagissent avec les sources de données pour en obtenir les données. En conséquence, les données sont continuellement demandées par le client et fournies par les

workers à partir de la source de données jusqu'à ce que l'exécution de la requête soit terminée.

Les coordinateurs communiquent avec les workers et les clients à l'aide d'un protocole basé sur HTTP.

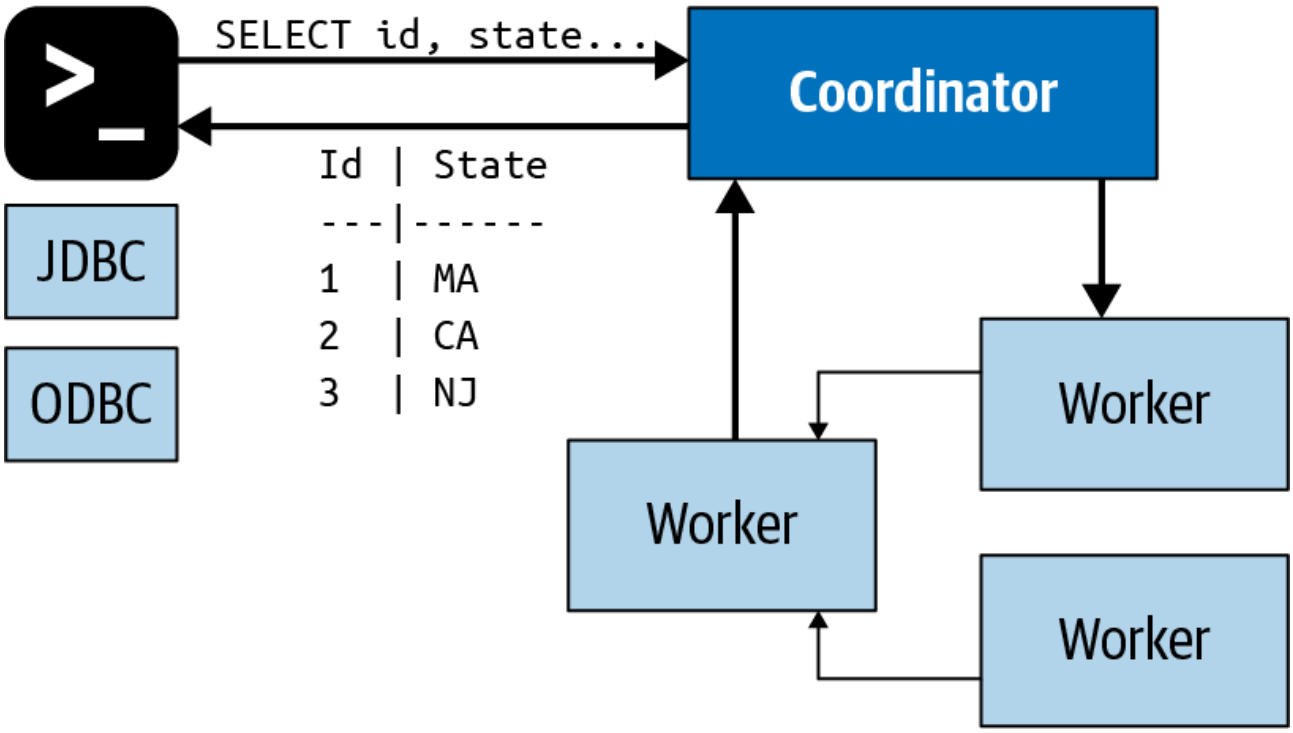

FIGURE 3.3 : Communication client, coordinateur et worker traitant une instruction SQL

3.4 Workers

Un worker Trino est un serveur dans une installation Trino. Il est responsable de l'exécution des tâches assignées par le coordinateur et du traitement des données. Les nœuds de travail récupèrent des données à partir de sources de données à l'aide de connecteurs, puis échangent des données intermédiaires entre eux. Les données finales qui en résultent sont transmises au coordinateur. Le coordonnateur est chargé de recueillir les résultats des workers et de fournir les résultats finaux au client.

Lors de l'installation, les agents sont configurés pour connaître le nom d'hôte ou l'adresse IP du service de découverte du cluster. Lorsqu'un agent démarre, il s'annonce au service de découverte, qui le met à la disposition du coordinateur pour l'exécution de la tâche.

Les workers communiquent avec d'autres workers et le coordinateur à l'aide d'un protocole basé sur HTTP.

La figure suivante montre comment plusieurs workers récupèrent des données à partir des sources de données et collaborent pour traiter les données, jusqu'à ce qu'un worker puisse fournir les données au coordinateur.

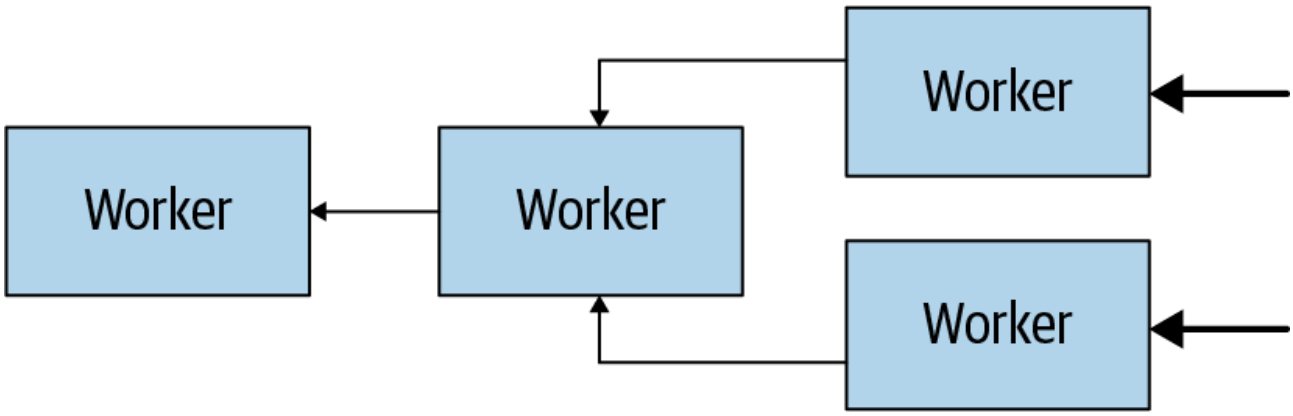

FIGURE 3.4 : Les workers d'un cluster collaborent pour traiter les instructions et les données SQL

3.5 Architecture basée sur les connecteurs

Au cœur de la séparation du stockage et du calcul dans Trino se trouve l'architecture basée sur les connecteurs. Un connecteur fournit à Trino une interface pour accéder à une source de données arbitraire.

Chaque connecteur fournit une abstraction basée sur une table sur la source de données sous-jacente. Tant que les données peuvent être exprimées en termes de tables, de colonnes et de lignes à l'aide des types de données disponibles pour Trino, un connecteur peut être créé et le moteur de requête peut utiliser les données pour le traitement des requêtes.

Trino fournit une interface de fournisseur de services (SPI), qui est un type d'API utilisé pour implémenter un connecteur. En implémentant le SPI dans un connecteur, Trino peut utiliser des opérations standard en interne pour se connecter à n'importe quelle source de données et effectuer des opérations sur n'importe quelle source de données. Le connecteur prend en charge les détails relatifs à la source de données spécifique.

- Opérations pour récupérer les métadonnées de table/vue/schéma
- Opérations pour produire des unités logiques de partitionnement de données, afin que Trino puisse paralléliser les lectures et les écritures
- Sources et récepteurs de données qui convertissent les données source vers/depuis le format en mémoire attendu par le moteur de requête

Trino fournit de nombreux connecteurs aux systèmes, vous trouverez la liste des connecteurs au moment de la rédaction de ce rapport ci-dessous.

Le SPI de Trino vous donne également la possibilité de créer vos propres connecteurs personnalisés. Cela peut être nécessaire si vous avez besoin d'accéder à une source de données sans connecteur compatible. Si vous finissez par créer un connecteur, nous vous encourageons fortement à en savoir plus sur la communauté open source Trino, à utiliser notre aide et à contribuer votre connecteur. Consultez « Ressources Trino » pour plus d'informations. Un connecteur personnalisé peut également être nécessaire si vous disposez d'une source de données unique ou propriétaire au sein de

votre organisation. C'est ce qui permet aux utilisateurs de Trino d'interroger n'importe quelle source de données en utilisant SQL, vraiment SQL-on-Anything.

La figure suivante montre comment le Trino SPI inclut des interfaces distinctes pour les métadonnées, les statistiques de données et l'emplacement des données utilisées par le coordinateur, et pour le flux de données utilisé par les workers.

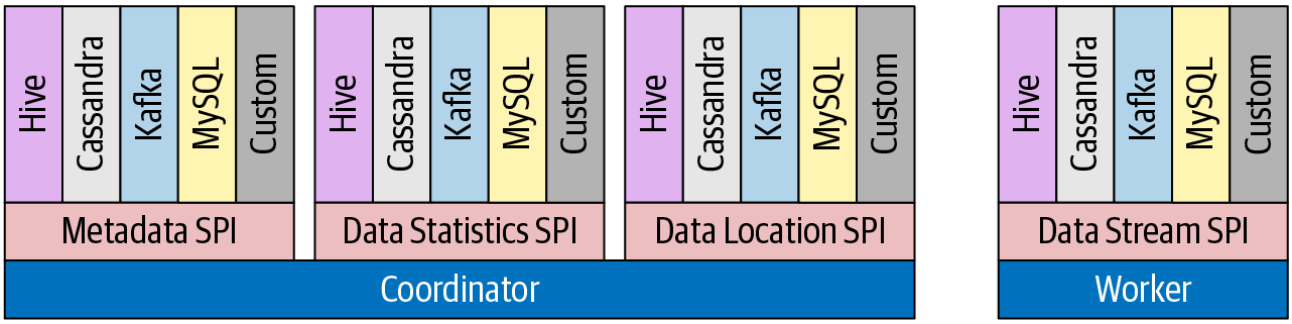

FIGURE 3.5 : Vue d'ensemble de l'interface du fournisseur de services Trino (SPI)

Les connecteurs Trino sont des plug-ins chargés par chaque serveur au démarrage. Ils sont configurés par des paramètres spécifiques dans les fichiers de propriétés du catalogue et chargés à partir du répertoire des plug-ins.

3.6 Catalogues, Schémas et Tables

Le cluster Trino traite toutes les requêtes en utilisant l'architecture basée sur les connecteurs décrite précédemment. Chaque configuration de catalogue utilise un connecteur pour accéder à une source de données spécifique. La source de données expose un ou plusieurs schémas dans le catalogue. Chaque schéma contient des tables qui fournissent les données dans des lignes de table avec des colonnes utilisant différents types de données. Vous pouvez en savoir plus sur les catalogues, les schémas, les tables. Plus précisément dans 'Catalogues', 'Schémas' et 'Tables'.

Conclusion

L'architecture Trino a un coordinateur recevant les demandes des utilisateurs, puis utilisant des travailleurs pour assembler toutes les données à partir des sources de données. Chaque requête est traduite en un plan de requête distribué de tâches en plusieurs étapes. Les données sont renvoyées par les connecteurs en fractions et traitées en plusieurs étapes jusqu'à ce que le résultat final soit disponible et fourni à l'utilisateur par le coordinateur.

Nom de connecteur	Lien de documentation
Accumulo	https://trino.io/docs/current/connector/accumulo.html
Atop	https://trino.io/docs/current/connector/atop.html
BigQuery	https://trino.io/docs/current/connector/bigquery.html
Black Hole	https://trino.io/docs/current/connector/blackhole.html
Cassandra	https://trino.io/docs/current/connector/cassandra.html
ClickHouse	https://trino.io/docs/current/connector/clickhouse.html
Delta Lake	https://trino.io/docs/current/connector/delta-lake.html
Druid	https://trino.io/docs/current/connector/druid.html
Elasticsearch	https://trino.io/docs/current/connector/elasticsearch.html
Google Sheets	https://trino.io/docs/current/connector/googlesheets.html
Hive	https://trino.io/docs/current/connector/hive.html
Hudi	https://trino.io/docs/current/connector/hudi.html
Iceberg	https://trino.io/docs/current/connector/iceberg.html
Ignite	https://trino.io/docs/current/connector/ignite.html
JMX	https://trino.io/docs/current/connector/jmx.html
Kafka	https://trino.io/docs/current/connector/kafka.html
Kinesis	https://trino.io/docs/current/connector/kinesis.html
Kudu	https://trino.io/docs/current/connector/kudu.html
Local File	https://trino.io/docs/current/connector/localfile.html
MariaDB	https://trino.io/docs/current/connector/mariadb.html
Memory	https://trino.io/docs/current/connector/memory.html
MongoDB	https://trino.io/docs/current/connector/mongodb.html
MySQL	https://trino.io/docs/current/connector/mysql.html
Oracle	https://trino.io/docs/current/connector/oracle.html
Phoenix	https://trino.io/docs/current/connector/phoenix.html
Pinot	https://trino.io/docs/current/connector/pinot.html
PostgreSQL	https://trino.io/docs/current/connector/postgresql.html
Prometheus	https://trino.io/docs/current/connector/prometheus.html
Redis	https://trino.io/docs/current/connector/redis.html
Redshift	https://trino.io/docs/current/connector/redshift.html
SingleStore	https://trino.io/docs/current/connector/singlestore.html
SQL Server	https://trino.io/docs/current/connector/sqlserver.html
System	https://trino.io/docs/current/connector/system.html
Thrift	https://trino.io/docs/current/connector/thrift.html
TPCDS	https://trino.io/docs/current/connector/tpcds.html
TPCH	https://trino.io/docs/current/connector/tpch.html

TABLE 3.1 : Liste des connecteurs Trino et leur documentation

General Conclusion