



MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR  
DE LA RECHERCHE SCIENTIFIQUE ET DE L'INNOVATION  
**UNIVERSITÉ SULTAN MOULAY SLIMANE**  
**ECOLE NATIONALE DES SCIENCES APPLIQUÉS  
DE KHOURIBGA**



# Mémoire de fin d'études

pour l'obtention du

**Diplôme d'Ingénieur d'État**

**Filière : Informatique et Ingénierie des Données**



**Présenté par :**

**Ziyad RAKIB**

## **Participation à la migration d'une architecture monolithique à une architecture microservices**

**Sous la supervision :**

**Abdelghani Ghazdali**

**Nidal Lamghari**

**Membres du jury :**

Nom et prénoms du président	Entité	Président
Nom et prénoms de l'examineur	Entité	Examineur
Nom et prénoms du rapporteur	Entité	Rapporteur
Nom et prénoms du rapporteur	Entité	Rapporteur

**Année Académique : 2023**



# Sommaire

Dédicace	ii
Remerciements	iii
Résumé	iv
Abstract	v
Table des matières	vii
Table des figures	ix
Liste des tableaux	x
Introduction	1
1 Izicap	2
2 Delta Lake	4
3 Trino	8
Conclusion	15

# Dédicace

“

*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 continuuel. Que cet humble travail porte témoignage 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**

# Remerciements

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 متعمقاً 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 ، وبناء الحوسبة الموزعة ، والحاويات. على الرغم من التحديات التي واجهها ، 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

Dédicace	ii
Remerciements	iii
Résumé	iv
Abstract	v
Table des matières	vii
Table des figures	ix
Liste des tableaux	x
Introduction	1
1 Izicap	2
Introduction . . . . .	2
1.1 About . . . . .	2
1.2 Organigramme . . . . .	3
Conclusion . . . . .	3
2 Delta Lake	4
Introduction . . . . .	4
2.1 Smart Data . . . . .	4
2.2 L'architecture du système . . . . .	6
Conclusion . . . . .	7
3 Trino	8
Introduction . . . . .	8
3.1 Définition . . . . .	8
3.2 Comment ça fonctionne . . . . .	8
3.3 Coordinateur . . . . .	10
3.4 Workers . . . . .	11
3.5 Architecture basée sur les connecteurs . . . . .	12
3.6 Catalogues, Schémas et Tables . . . . .	13
Conclusion . . . . .	13

<b>Conclusion</b>	<b>15</b>
-------------------	-----------

# Table des figures

1.1	Organigramme de Izicap . . . . .	3
2.1	Smart data Izicap . . . . .	5
2.2	Architecture du système actuelle . . . . .	6
3.1	Vue d'ensemble de l'architecture Trino avec le coordinateur et les workers . . . . .	9
3.2	Communication entre le coordinateur et les workers dans un cluster Trino . . . . .	10
3.3	Communication client, coordinateur et worker traitant une instruction SQL . . . . .	11
3.4	Les workers d'un cluster collaborent pour traiter les instructions et les données SQL . . . . .	12
3.5	Vue d'ensemble de l'interface du fournisseur de services Trino (SPI) . . . . .	13

# Liste des tableaux

1	List of Abbreviations . . . . .	xi
3.1	Liste des connecteurs Trino et leur documentation . . . . .	14

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

# Introduction Générale

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.

# 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

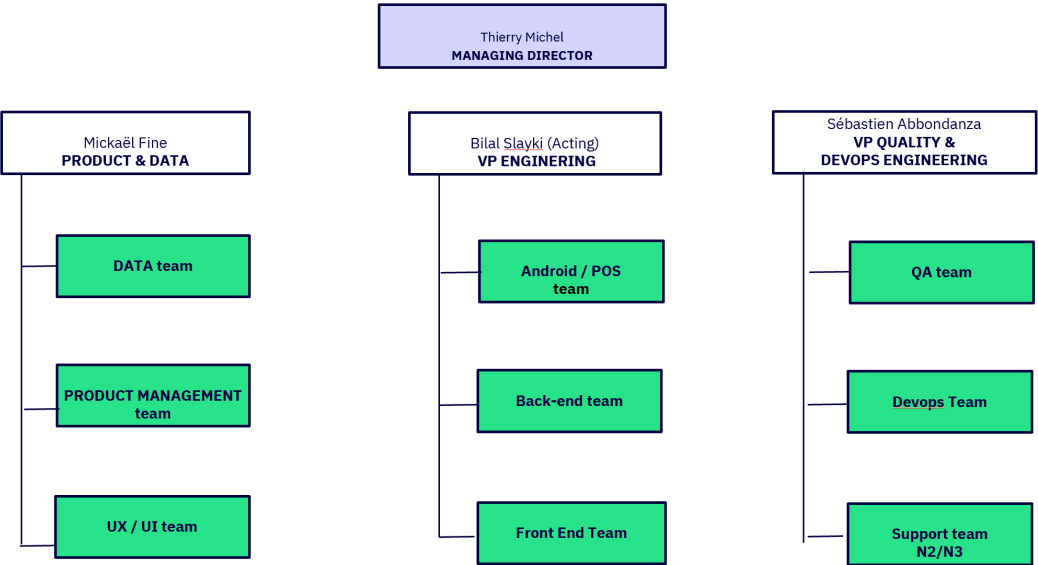


FIGURE 1.1 : Organigramme de Izicap

Conclusion



## Delta Lake

### Introduction

Dans ce chapitre, nous explorerons la schématisation détaillée du projet visant à remplacer notre architecture monolithique par une approche basée sur des microservices et des micro-frontends, tout en migrer notre système de gestion de base de données de MariaDB vers Delta Lake. Cette transition marque une étape cruciale dans notre parcours d'évolution technologique, offrant des avantages significatifs en termes de flexibilité, d'évolutivité et de maintenabilité.

### 2.1 Smart Data

La smart data fait référence au processus d'extraction d'informations précieuses à partir de grandes quantités de données afin de prendre des décisions commerciales éclairées.

La smart data nous permet de :

1. **Collecte des données** : Izicap aide les entreprises à collecter des données clients à partir de différents points de contact tels que les systèmes de point de vente, les programmes de fidélité, les interactions en ligne, etc. Assurez-vous d'intégrer leurs outils de collecte de données dans vos systèmes ou processus existants.
2. **Analyse des données** : La solution de smart data d'Izicap vous permet d'analyser les données collectées pour découvrir des tendances, des modèles et des comportements clients. Utilisez leurs outils d'analyse et leurs algorithmes pour obtenir des informations exploitables à partir des données.
3. **Segmentation des clients** : Segmentez votre base de clients en fonction de leurs préférences, de leur historique d'achat, de leurs caractéristiques démographiques ou d'autres critères pertinents. La solution de smart data d'Izicap peut vous aider à identifier différents segments de clients et à créer des stratégies marketing personnalisées pour chaque segment.

4. **Marketing personnalisé** : Exploitez les informations tirées de la solution de smart data d'Izicap pour créer des campagnes marketing ciblées. Envoyez des offres personnalisées, des promotions ou des recommandations à des segments spécifiques de clients, augmentant ainsi les chances de conversion et de satisfaction client.
5. **Fidélisation de la clientèle** : Utilisez la solution de smart data d'Izicap pour identifier les clients qui risquent de résilier leur abonnement ou de ne plus acheter chez vous. Développez des stratégies de fidélisation en leur offrant des incitations, des programmes de fidélité ou des communications personnalisées afin de les maintenir engagés et fidèles à votre marque.
6. **Suivi des performances** : Surveillez régulièrement les performances de vos campagnes marketing et de vos efforts d'engagement client. La solution d'Izicap peut vous fournir des métriques et des rapports pour évaluer l'efficacité de vos stratégies et apporter des ajustements basés sur les données lorsque nécessaire.
7. **Amélioration continue** : Les solutions de smart data sont les plus efficaces lorsqu'elles sont utilisées de manière itérative. Analysez régulièrement les données, adaptez vos stratégies et peaufinez votre approche en fonction de nouvelles informations et de l'évolution du comportement client.

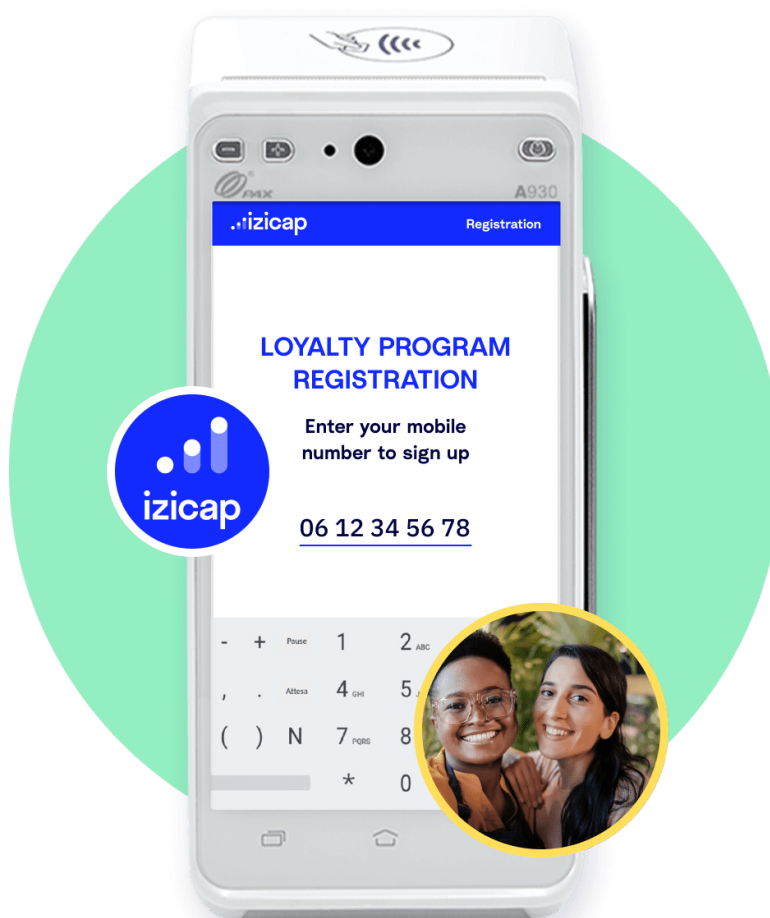


FIGURE 2.1 : Smart data Izicap

## 2.2 L'architecture du système

La figure ci-dessous visualise la structure et les composants clés de notre système, ainsi que les interactions entre eux, elle constitue ainsi le fondement de notre système et joue un rôle essentiel dans la fourniture de fonctionnalités et de services à nos utilisateurs.

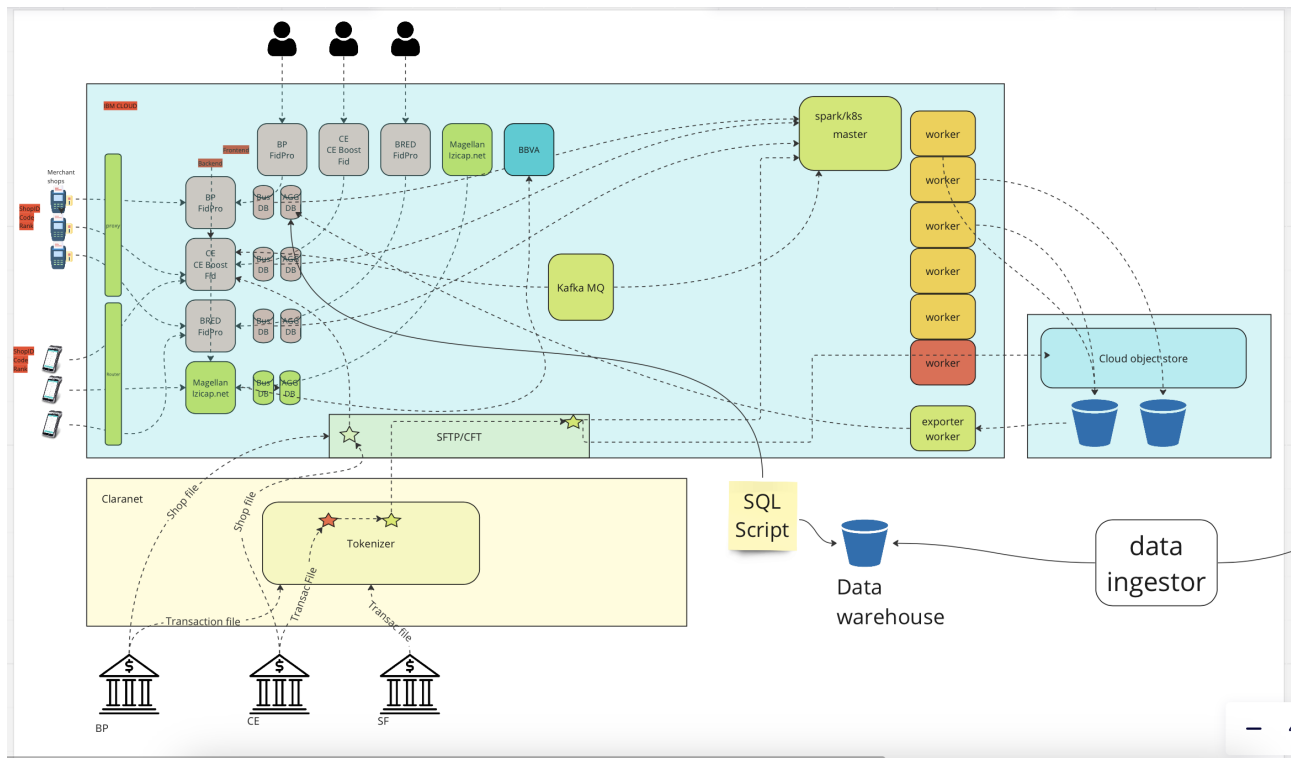


FIGURE 2.2 : Architecture du système actuelle

Ce schéma représente le fonctionnement de l'architecture monolithique actuelle qui présente plusieurs inconvénients qui peuvent entraver l'efficacité, la flexibilité et la maintenabilité du système. Voici une description détaillée des principaux inconvénients :

1. **Complexité et dépendances** : L'architecture monolithique implique que tous les modules et fonctionnalités du système sont regroupés en un seul bloc. Cela crée une forte interdépendance entre les différents composants, rendant la compréhension, la gestion et les mises à jour complexes. Les modifications apportées à une partie du système peuvent avoir des répercussions sur d'autres parties, ce qui rend les tests, les déploiements et les corrections d'erreurs plus difficiles.
2. **Scalabilité limitée** : L'architecture monolithique a souvent des difficultés à s'adapter à une augmentation de la charge ou à une demande croissante. Étant donné que tous les composants sont regroupés dans un seul monolithe, il est difficile de faire évoluer sélectivement une partie spécifique du système. Cela peut entraîner des goulots d'étranglement, des performances réduites et une mauvaise répartition des ressources lorsqu'il s'agit de traiter des volumes de données importants ou de gérer une augmentation du nombre d'utilisateurs.

3. **Déploiements complexes et risques élevés** : Avec une architecture monolithique, les déploiements nécessitent la mise à jour de l'ensemble du système, même pour des modifications mineures. Cela augmente les risques d'erreurs et de régressions, car une seule erreur peut entraîner l'indisponibilité du système tout entier. De plus, les déploiements doivent être soigneusement planifiés et coordonnés, ce qui peut entraîner des temps d'arrêt plus longs et des interruptions de service pour les utilisateurs.
4. **Difficulté de choix technologiques** : Dans une architecture monolithique, les technologies utilisées sont souvent liées et intégrées de manière étroite. Cela peut rendre difficile l'adoption de nouvelles technologies ou l'intégration de composants spécialisés. Les mises à niveau de versions ou les ajouts de nouvelles fonctionnalités peuvent être limités par les choix technologiques initiaux, ce qui peut entraver l'innovation et l'adaptation aux évolutions du marché.
5. **Cohésion et responsabilités** : L'architecture monolithique ne facilite pas une séparation claire des responsabilités et des fonctionnalités. Les différents modules du système sont souvent intimement liés et peuvent partager des fonctionnalités communes. Cela rend difficile l'isolation des problèmes, la maintenance spécifique des modules et la réutilisation de code spécifique à un domaine.

## 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.

# 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.

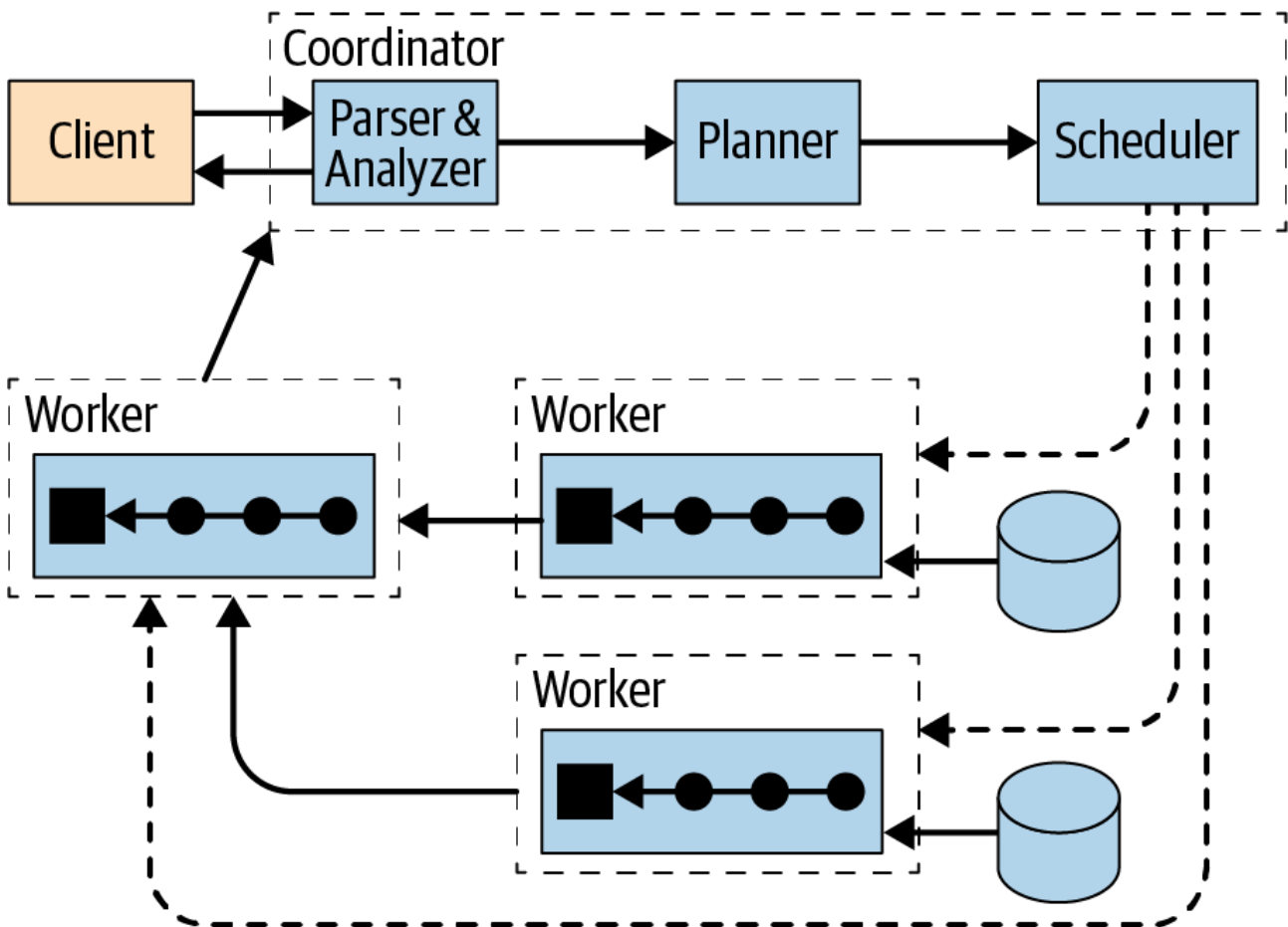


FIGURE 3.1 : Vue d'ensemble de l'architecture Trino avec le coordinateur et les workers

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.

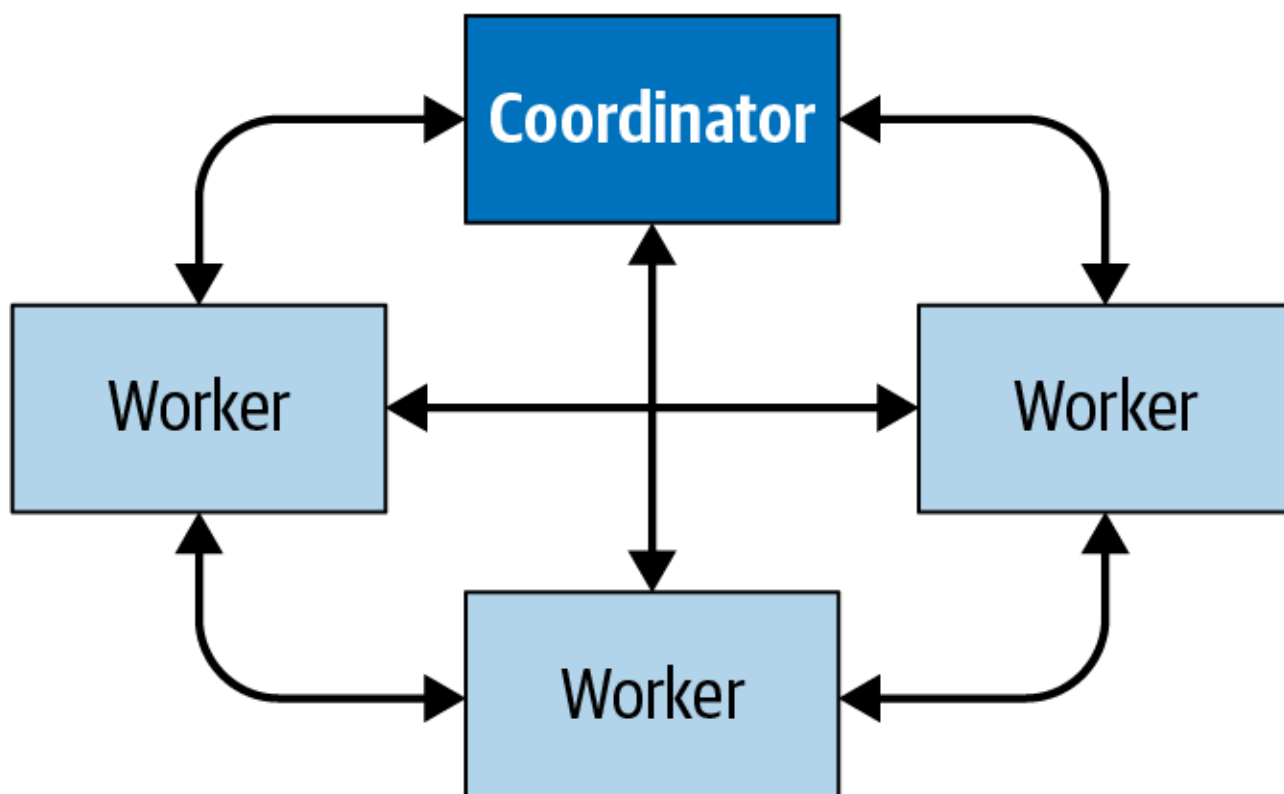


FIGURE 3.2 : Communication entre le coordinateur et les workers dans un cluster Trino

### 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 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.

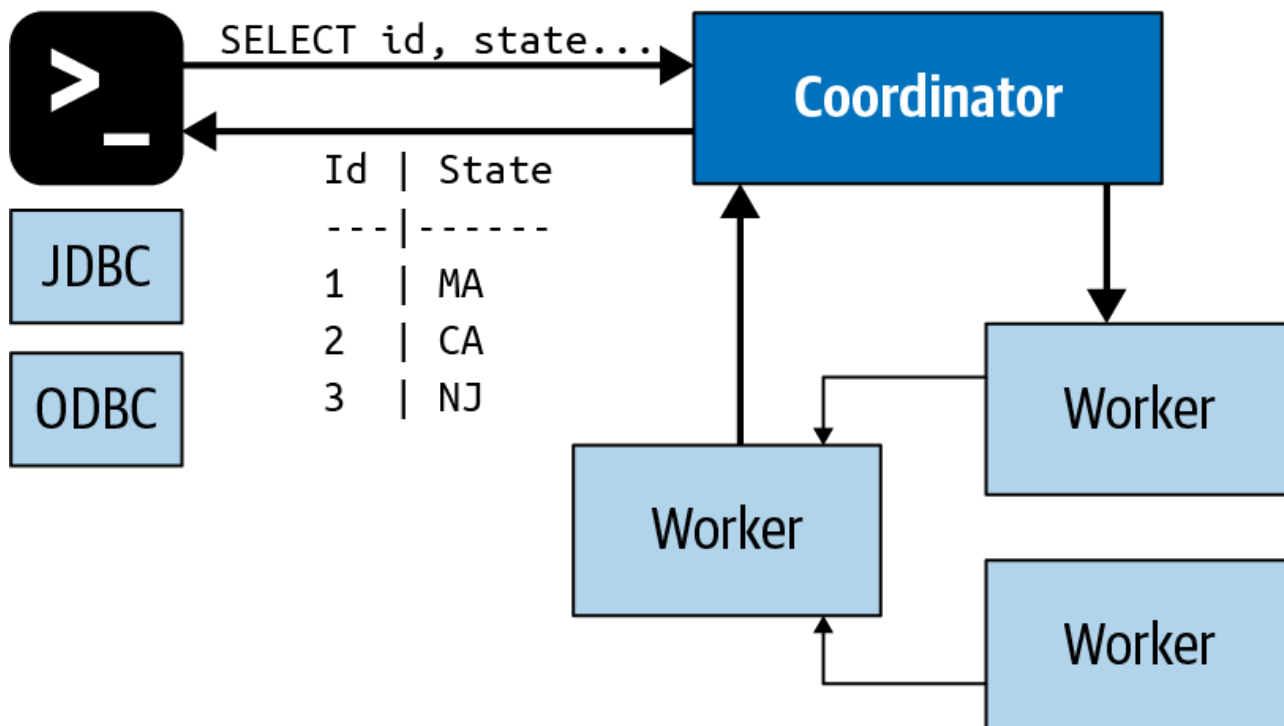


FIGURE 3.3 : Communication client, coordinateur et worker traitant une instruction SQL

### 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 coordinateur 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.



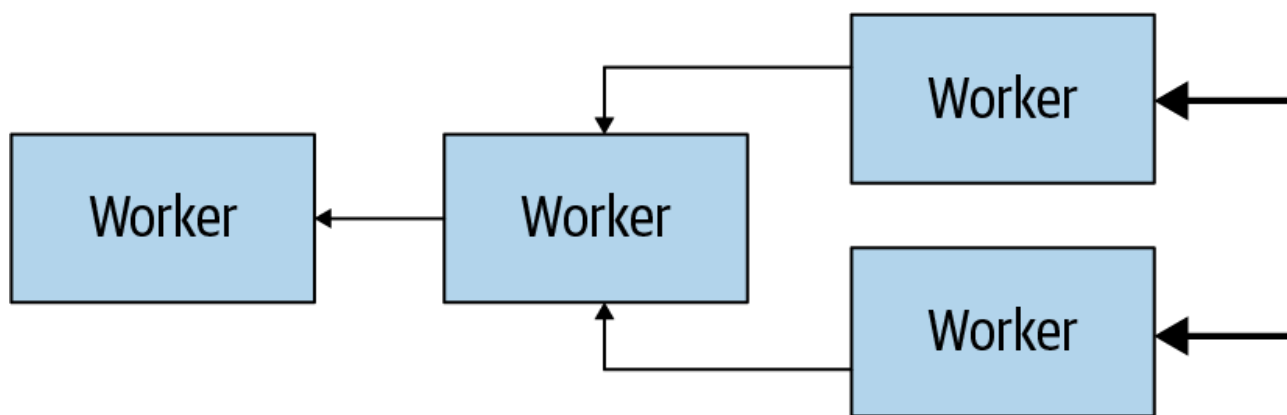


FIGURE 3.4 : Les workers d'un cluster collaborent pour traiter les instructions et les données SQL

### 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

vosre 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.

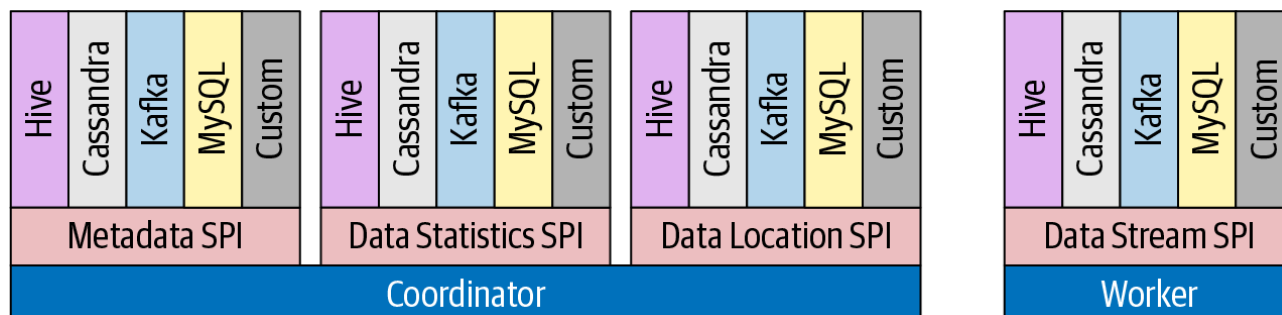


FIGURE 3.5 : Vue d'ensemble de l'interface du fournisseur de services Trino (SPI)

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

TABLE 3.1 : Liste des connecteurs Trino et leur documentation

## Conclusion Générale

---