PARIS PANTHÉON-ASSAS UNIVERSITÉ	**Data Lake**	M2- Data Engineering & IA 2024 - 2025

Devoir :

Conception, Développement, et Exploitation d'un Data Lake pour une Entreprise Digitale

Groupe	Date de rendu	Professeur	Page
Mathias ENGAMBE Zijun KONG Thi Cam Linh HO	22/11/2024	Lionel SOUOP PEKAM	2

Table des matières

Partie 1 : Conception du Data Lake

Analyse des besoins et proposition d'architecture

Compréhension des besoins :

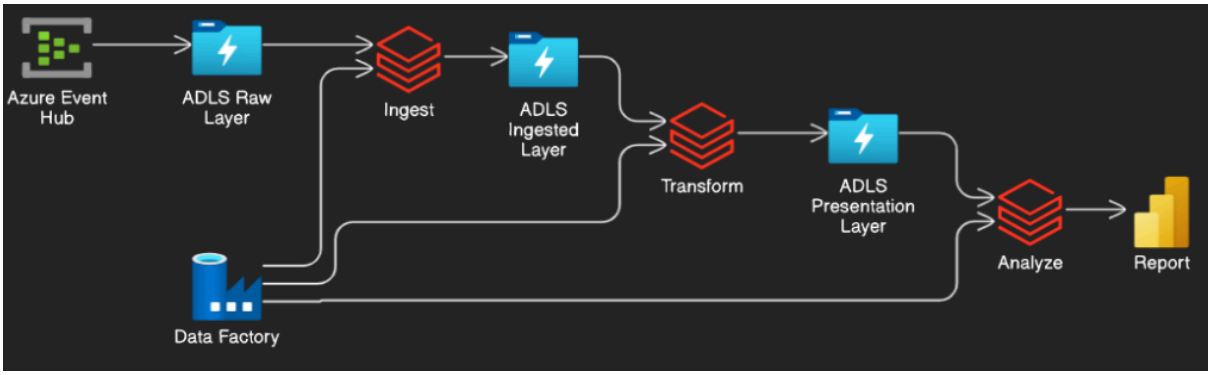
- **Collecte de données hétérogènes:** Les données proviennent de sources variées (bases de données relationnelles, logs, médias sociaux, flux en temps réel).
- **Volume :** Le système doit être capable de gérer des volumes de données croissants.
- **Stockage centralisé:** On doit rassembler toutes les données de l'entreprise en un seul endroit pour une vue unifiée. Un Data Lake peut permettre de stocker ces données dans un format brut et structuré.
- **Traitement des données:** Les données doivent être ingérées, transformées et enrichies pour être exploitables.
- **Analyse:** On doit permettre à l'entreprise d'effectuer des analyses prédictives et générer des rapports.
- **Gouvernance des données:** On doit mettre en place un système de gouvernance pour assurer la qualité, la sécurité et la conformité des données.

Proposition d'architecture :

- **Microsoft Entra ID (anciennement Azure Active Directory) :**
  - Pour gérer les identités et les accès aux ressources Azure.
- **Azure Event Hubs:**
  - Ingérer les flux de données en temps réel des campagnes publicitaires.
  - Intégrer avec Azure Databricks pour un traitement en temps réel.
- **Azure Data Factory (**Axe d'amélioration**):**
  - Pour orchestrer l'ingestion des données batch (transactions, logs, médias sociaux) à partir de différentes sources vers le Data Lake.
- **Azure Data Lake Storage Gen2:**
  - Stocker toutes les données (transactions, logs, données sociales, flux en temps réel) dans un seul espace de stockage sécurisé et étant capable d'évoluer.
  - Bénéficier des fonctionnalités avancées de stockage de données multi-sources et distribuée Hadoop Distributed File System (HDFS).
- **Azure Databricks:**
  - Utiliser des clusters Apache Spark pour le traitement distribué des données.
  - Bénéficier d'une interface utilisateur intuitive pour développer et exécuter des notebooks.
  - Intégrer facilement avec Azure Data Lake Storage Gen2 pour accéder aux données.
  - et la compatibilité avec Hadoop Distributed File System (HDFS).

- **Spark** (utilisé dans Databricks) est un outil puissant pour le traitement de données sur des clusters Hadoop ou des systèmes de stockage compatibles HDFS comme Azure Data Lake Gen2.
- **Unity Catalog :**
    - Pour cataloguer et gérer les données, en assurant la traçabilité et la conformité.
- **Power BI (**Axe d'amélioration**):**
    - Pour créer des tableaux de bord interactifs et des rapports personnalisés.

Ci-dessous notre diagramme de flux de données :



Architecture de notre data pipeline Azure

Choix des technologies pour le stockage

Où intervient HDFS ?

Bien qu'on ne parle pas directement de HDFS dans cette architecture, le concept est le même : Azure Data Lake Gen2 joue le rôle de HDFS, offrant un stockage distribué et scalable pour les données. Un outil comme Databricks peut accéder à ces données en utilisant des interfaces compatibles HDFS. Le terme Azure est **ABFSS** (**A**zure **B**lob **F**ileSystem **S**torage)

Format de stockage

En ce qui concerne le format de stockage, nous choisissons de stocker les données au format Delta. Basé sur Parquet, ce format respecte bien les transactions ACID, offrant ainsi une garantie de qualité des données. De plus, le format Delta est évolutif et optimisé pour les lacs de données (Data Lakes), en utilisant des techniques comme le partitionnement dans notre script python on améliore les performances des requêtes.
En optant pour un stockage de données en mode lakehouse, nous bénéficions des

avantages combinés des data lakes et des data warehouses, avec une flexibilité de stockage élevée et des performances de requêtes optimisées.

Notre algorithme de calcul distribué : Spark?

Comme évoqué plus haut nous avons choisi d'utiliser Spark comme algorithme de calcul distribué qui va diviser les données dans les **dataNodes** et les exécuter simultanément en parallèle dans les **dataNodes** de notre cluster. Spark est également parfaitement adapté pour des cas d'utilisations typiques tels que : l'**analyse de données massives** et le **streaming de données temps réel** selon la documentation officielle ce qui est parfaitement adapté à notre besoin. En outre, son stockage en mémoire vive (**In-memory**) réduit considérablement la latence en accédant directement aux données dans la RAM, ce qui évite de fréquents accès disques, qui sont beaucoup plus lents.
Contrairement à MapReduce qui écrits souvent ses données sur disque après chaque phase intermédiaire, Spark permet de réutiliser les données en mémoire dans plusieurs étapes d'un pipeline de traitement, ce qui est extrêmement efficace pour des opérations répétitives comme les itérations sur des données et correspond bien à notre problématique.
Enfin Spark permet les deux modes de traitement : par lots et en streaming temps réel (via Spark Streaming). Ainsi il peut être utilisé pour traiter de grandes quantités de données à intervalles réguliers (transactions clients) mais aussi les flux de données en temps réel des campagnes de publicité en ligne ou des logs de serveurs web…

En conclusion, Spark constitue un framework unifié combinant à la fois les avantages de Kafka et MapReduce.

Partie 2 : Ingestion et Transformation des Données

Création de l'infrastructure

Comme évoqué dans la phase de conception, nous avons effectué différents choix de solutions clairement définis pour répondre à la problématique de l'entreprise de commerce en ligne souhaitant exploiter au maximum ses données en centralisant toutes ses sources de données dans un Data Lake. Nous avons choisi de travailler spécialement avec des solutions cloud du fournisseur Azure.
Succinctement nous allons structurer notre infrastructure (cf. Proposition d'architecture) de la façon suivante :

- Création d'un data lake (storage account Azure) qui va assurer le stockage distribué à grande échelle
- Script Python qui va simuler l'envoi de données en continu d'un objet connecté
- Azure Event Hub ingère le flux de données en temps réel et fait de la rétention.

- ❖ Script qui gérer tout le processus de stockage de la donnée, son raffinage, et sa présentation au travers de 3 couches (cf. Architecture Databricks)
- ❖ Script API nodeJs pour l'exploitation de la donnée et son analyse.
- ❖ Power BI pour la visualisation

Pourquoi le Cloud

Nous avons directement choisi de faire notre projet dans le Cloud car nous pensons que le Cloud offre une solution flexible, scalable et économique pour mettre en place un Data Lake et exploiter les données d'une entreprise de commerce en ligne. Grâce à lui, on peut répondre aux besoins croissants en matière de stockage, de traitement et d'analyse de données, tout en réduisant les coûts et les complexités de gestion. Dans le **contexte spécifique de ce scénario, le cloud nous permet de :**

- **Stocker des volumes importants de données hétérogènes:** Les Data Lakes doivent souvent gérer d'énormes quantités de données très diverses. C'est là que le cloud offre une solution flexible et évolutive.
- **Traiter les données en temps réel :** Pour les flux de données des campagnes publicitaires en ligne, un traitement en temps réel est indispensable. Le cloud permet de créer des clusters de calcul à la demande, parfaits pour ce genre de tâche.
- **Analyser les données de manière approfondie :** Les outils d'analyse du cloud, comme Azure Synapse Analytics, AWS EMR ou Google BigQuery, permettent des analyses complexes et profondes dans les données du Data Lake.
- **Collaborer efficacement :** Le cloud simplifie le travail collaboratif de notre équipe, grâce à des outils de partage comme Azure Databricks qui rendent notre travail en commun plus facile.

En outre, le Cloud permet d'augmenter ou diminuer les ressources en fonction des besoins, évitant les sur-provisionnements ou les sous-provisionnements. Le déploiement d'un environnement Cloud est plus rapide qu'une infrastructure non cloud et le paiement à l'usage permet de ne payer seulement pour les ressources consommées. Enfin, en tant que client, on est pas chargé de la maintenance et des mises à jour de l'infrastructure, mais les fournisseurs de cloud.

Pourquoi Azure ?

Nous avons privilégié le cloud Azure plutôt que AWS ou Google Cloud pour les raisons suivantes :

- **Intégration avec d'autres services Microsoft:** En partant du principe que l'entreprise e-commerce utilise déjà d'autres services Microsoft (*Office 365, Copilot*), l'intégration avec Azure sera plus fluide.
- **Fonctionnalités spécifiques:** Azure propose des fonctionnalités spécifiques qui peuvent être particulièrement adaptées aux besoins d'exploitations de données

(*Azure Synapse Analytics* pour l'analyse de données, *Power BI* pour la visualisation, *Data Factory* pour l'orchestration…)

Azure Event Hub

Pourquoi ce service ?

Ingestion en temps réel: *Event Hubs* est idéal pour ingérer des flux de données en continu, comme les données de transactions bancaires.
Scalabilité automatique: Contrairement à **Kafka,** *Azure Event Hubs* offre une scalabilité automatique. Le service ajuste automatiquement les ressources en fonction de la charge, ce qui signifie que l'on a pas à se soucier de la gestion manuelle de la capacité. Par conséquent, la scalabilité horizontale de **Kafka** nécessite une configuration plus complexe.
Intégration avec d'autres services Azure: Il s'intègre facilement avec Databricks pour le traitement des données.

En conclusion, pour trancher de façon définitive entre *Azure Event Hubs* et *Kafka* nous nous sommes posez trois questions fondamentales ensemble :
Nous recherchons une solution entièrement gérée et facile à mettre en œuvre ?
Azure Event Hubs est un excellent choix. La scalabilité automatique et la gestion simplifiée sont des avantages importants.
Nous avons besoin d'une solution hautement personnalisable et avec une scalabilité presque illimitée?
Kafka est plus approprié. Cependant, cela implique une gestion plus complexe du cluster ce qui reste une contrainte majeure dans la gestion d'une infrastructure d'entreprise.
Une solution aisément implémentable avec les autres services Azure que nous utilisons déjà ?
Azure Event Hubs s'intégrera plus facilement dans notre écosystème.

Solution :

On crée une ressource ***Azure Event Hub Namespace*** qui va constituer un conteneur pouvant contenir toute une collection de ***Event Hub***. On met en place la configuration auto-scaling pour l'unité de débit afin d'automatiser l'optimisation. Sa création génère automatiquement une première stratégie de ***signature d'accès partagé (SAS)*** avec des clés primaires et secondaires, ainsi que des chaînes de connexion qui donnent chacune un contrôle total sur tous les aspects de la solution. Dans notre cas, on crée un ***Event Hub*** unique sur le *portail Azure* pour mettre en place le service d'ingestion en temps réel que l'on paramètre avec 4 partitions pour débuter ainsi que 7 jours de rétention des événements de données reçues.

Nous pouvons parfaitement monitorer le flux de données de messagerie envoyé par notre script python simulant un flux continu de transactions bancaire, poste d'avis, et de journalisation (logs de serveurs) grâce au print de chaque évènement en-dessous de notre cellule dans le notebook Azure Databricks. De cette manière on a le résultat ci-dessous :

```
            await producer.send_batch(event_data_batch)
            print(combined_event, end='\n\n')
            time.sleep(5) # Pause pour faciliter le traitement des événements

await run()
```

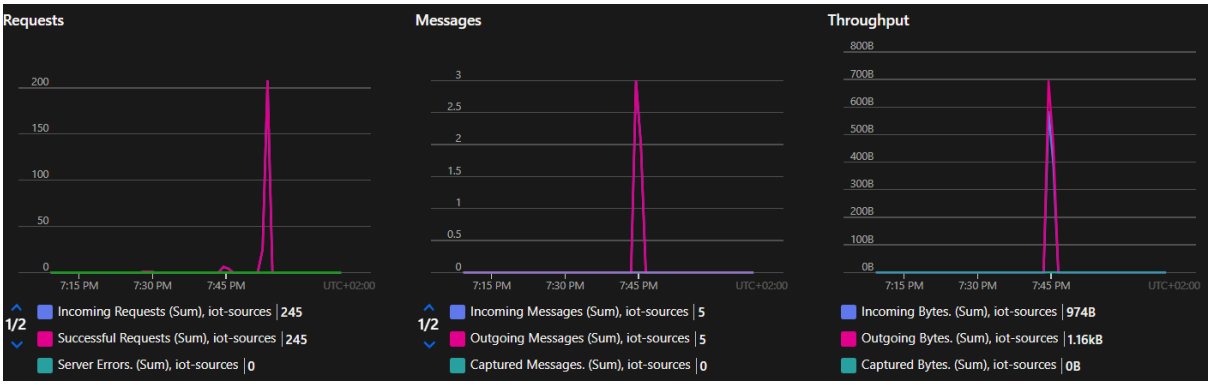
```
{'transactionID': 3228, 'customerID': 22, 'amount': 204.46, 'timestamp': '2024-11-14T21:41:09.041017', 'lastTransactionDate': '2024-11-15', 'type': 'Achat vêtements', 'customerEmail': 'customer21@gmail.com', 'customerFirstName': 'Amelia', 'customerLastName': 'King', 'logID': 69, 'status': 'INFO', 'message': 'Processed transaction 3228', 'logDate': '2024-11-15', 'postID': 4796, 'content': 'Great customer service!', 'postDate': '2024-11-15', 'pseudonym': 'LoloAmelia_KingUmami85'}

{'transactionID': 2371, 'customerID': 63, 'amount': 64.3, 'timestamp': '2024-11-14T21:41:14.057255', 'lastTransactionDate': '2021-06-21', 'type': 'Achat vêtements', 'customerEmail': 'customer8@gmail.com', 'customerFirstName': 'Mason', 'customerLastName': 'Johnson', 'logID': 699, 'status': 'INFO', 'message': 'Processed transaction 2371', 'logDate': '2021-06-21', 'postID': 3303, 'content': 'Will definitely buy again.', 'postDate': '2021-06-21', 'pseudonym': 'LoloMason_JohnsonUmami85'}

{'transactionID': 4174, 'customerID': 69, 'amount': 107.21, 'timestamp': '2024-11-14T21:41:19.083500', 'lastTransactionDate': '2019-11-03', 'type': 'Achat vêtements', 'customerEmail': 'customer54@gmail.com', 'customerFirstName': 'Nora', 'customerLastName': 'Mitchell', 'logID': 506, 'status': 'INFO', 'message': 'Processed transaction 4174', 'logDate': '2019-11-03', 'postID': 1755, 'content': 'Great customer service!', 'postDate': '2019-11-03', 'pseudonym': 'LoloNora_MitchellUmami85'}

{'transactionID': 8419, 'customerID': 15, 'amount': 255.87, 'timestamp': '2024-11-14T21:41:24.108953', 'lastTransactionDate': '2022-12-21', 'type': 'Achat vêtements', 'customerEmail': 'customer14@gmail.com', 'customerFirstName': 'Aria', 'customerLastName': 'Robinson', 'logID': 535, 'status': 'WARNING', 'message': 'Processed transaction 8419', 'logDate': '2022-12-21', 'postID': 4883, 'content': 'Will definitely buy again.', 'postDate': '2022-12-21', 'pseudonym': 'LoloAria_RobinsonUmami85'}
```

Nous pouvons bien évidemment aussi monitorer sous forme graphique l'envoie des messages en observant directement sur le portail Azure comme l'indique l'image ci-dessous :



Architecture Databricks

Pourquoi Databricks ?

Nous avons choisi la solution Azure Databricks car il s'agit d'une plateforme cloud basée sur le framework *Spark* pour créer des notebooks nous permettant d'implémenter des scripts de traitement distribué de façon optimale pour notre problématique Spark. De plus Databricks

nous fournit la possibilité de monter et configurer notre cluster pour notre charge de travail. Pour tout dire nous avons opté pour les raisons suivantes :
Traitement distribué: Databricks est basé sur Apache Spark, ce qui permet de traiter de grands volumes de données de manière parallèle.
Intégration avec Azure: Cette plateforme s'intègre nativement avec d'autres services Azure, comme Event Hubs, Data Lake Storage Gen2, Data Factory et Power BI. Cela facilite la construction de pipelines de données.
Facilité d'utilisation: L'interface utilisateur de Databricks est intuitive, facilitant le développement de script dans les notebooks.
Collaboration: Databricks offre une interface collaborative pour nous permettre de travailler à plusieurs membres sur le même projet.

Solution :

En préambule, afin de préparer une infrastructure conforme aux normes conventionnelles entreprise nous nous appuyons sur la documentation technique da Azure Databricks : https://www.databricks.com/notebooks/iiot/iiot-end-to-end-part-1.html. On va établir une **architecture en médaillon** qui est une architecture recommandée par Microsoft et Databricks avec **3 tables** nommées **bronze, silver et gold**. Cette approche s'inspire du paradigme Lambda . Chacune des tables représente des couches de différents états de la donnée avec l'Ingestion et le stockage de la donnée brute, son traitement et sa transformation, puis la présentation de la donnée exploitable enrichie.

- **Couche Bronze:** Correspondant au *batch layer* de l'architecture Lambda, cette couche accueillera les données brutes ingérées telles qu'elles sont produites par nos sources. Sans transformation ni enrichissement, elle sert de zone de stockage initiale.
- **Couche Silver:** Similaire au *speed layer* de Lambda, cette couche contiendra les données transformées et structurées pour l'analyse. Les données brutes sont nettoyées, enrichies et préparées pour les traitements ultérieurs.
- **Couche Gold:** Comparable au *serving layer* de Lambda, cette couche regroupe les données finales, optimisées pour la consommation par les utilisateurs finaux. Elles sont agrégées, agrémentées de métadonnées et prêtes pour la visualisation et le reporting.

Metastore:

Ensuite nous avons aussi créer des tables externes qui constituent une copie des métadonnées stockées dans le data lake azure. Les métadonnées sont ainsi stockées dans Databricks File System (DBFS). De cette manière nous pouvons les interroger et les référencer sans les déplacer ou encore les déplacer entre plusieurs clusters en ne prenant aucun risque (intégrité comprise, pertes…).

Montage cluster

Dans Databricks, il est important d'avoir une bonne configuration de cluster afin d'avoir d'un côté des **machines virtuels** ayant suffisamment de capacité pour supporter les **charges de travail** de notre infrastructure mais d'un autre côté, veiller à ne pas configurer inutilement le maximum de ressources car cela impacterait fortement les coûts lors des exécutions des charges de travail "**workloads**". En outre, lors de la configuration du cluster, Databricks nous propose néanmoins la possibilité de paramétrer un scalabilité automatique "auto-scaling" afin d'optimiser le tout. En somme nous avons donc configuré notre cluster avec les critères suivants :

Flexibilité: La configuration permet d'adapter les ressources en fonction des charges de travail.

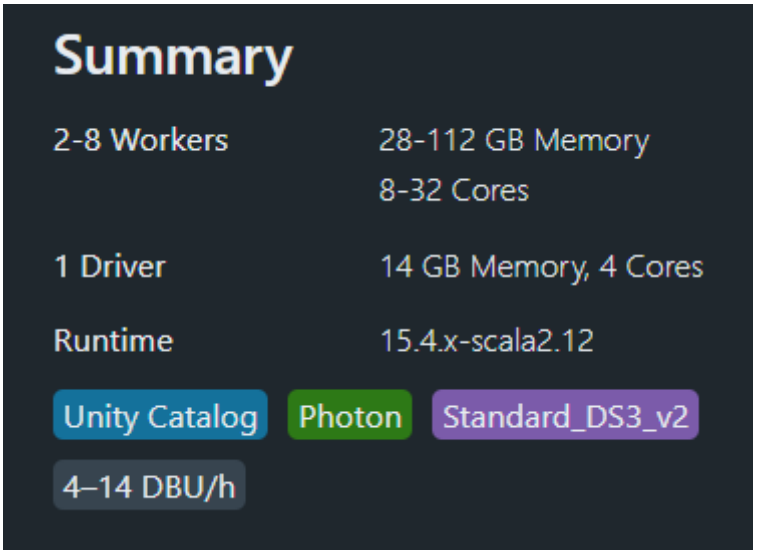
Performance: Les nombreux cœurs et la grande mémoire permettent de traiter rapidement les données.

Runtime: Le runtime 15.4.x-scala2.12 est compatible avec la librairie Event Hubs et offre les dernières fonctionnalités de Spark.

Ensuite nous avons aussi ajouter la librairie maven "*com.microsoft.azure:azure-eventhubs-spark_2.12:2.3.22*" pour les raisons suivantes :

Connectivité: Cette librairie permet à Spark de se connecter à Azure Event Hubs et de lire les données en streaming.

Version: La version 2.12 indique que la librairie est compatible avec Scala 2.12, la version utilisée dans le cluster Databricks.



Configuration cluster de calcul dans Azure Databricks

Data Factory

Pour le moment, bien que tout le pipeline ait été mis en place, ainsi que les api, nous n'avons pas encore eu le temps de mettre en place ou plutôt de connecter notre databricks workspace et notre data lake à la solution d'orchestration cloud **Azure Data Factory**.

Nous avons choisi d'utiliser la solution **Azure Data Factory** pour l'automatisation et la planification de l'orchestration de notre pipeline de données. Nous préférons utiliser **ADF** pour les raisons suivantes :

Orchestration: Data Factory permet de créer des pipelines de données complexes pour orchestrer l'ensemble du parcours de nos flux de données de bout en bout allant de l'ingestion à la présentation.

Connectivité: Il offre une large gamme de connecteurs pour intégrer diverses sources et données cibles .

Visibilité: Data Factory fournit une interface graphique intuitive pour concevoir, planifier et surveiller les pipelines.

Scalabilité: Il peut gérer des volumes de données importants.

Intégration: Il s'intègre facilement avec d'autres services Azure.

Partie 3 : Analyse et Exploitation des Données

Power BI

Pour le moment, les API sont en place ainsi que les 'access token' générés, toutefois nous n'avons pas encore eu le temps d'établir les différents graphiques de visualisation dans l'outil de notre choix **PowerBI**. En effet, notre choix s'est porté sur cet outil pour ses capacités de création rapports interactifs, de modélisation et de visualisation. En résumé, il permet de fournir des insights exploitables facilitant ainsi la prise de décision.

API & Web Services

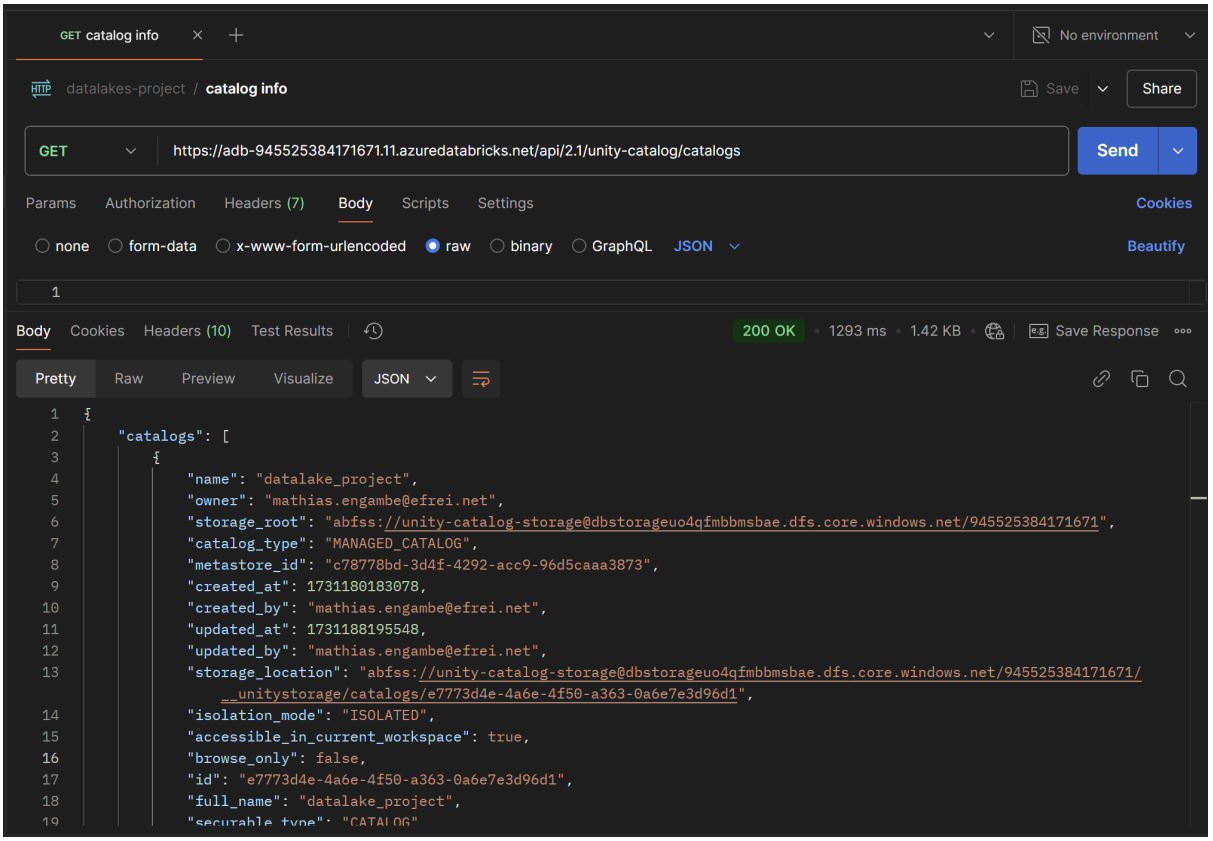
Databricks fournit un ensemble d'API pour manipuler le **Unity Catalog**.

Il faut noter que pour toutes les opérations API, nous devons générer un token dans databricks et le fournir dans la section d'autorisation de postman afin de communiquer correctement avec databricks.

Par exemple, si nous avons besoin d'obtenir des informations sur le catalog, nous pouvons utiliser la commande suivante dans postman :

GET https://adb-945525384171671.11.azuredatabricks.net/api/2.1/unity-catalog/catalogs

Cette commande nous donnera le résultat suivant :

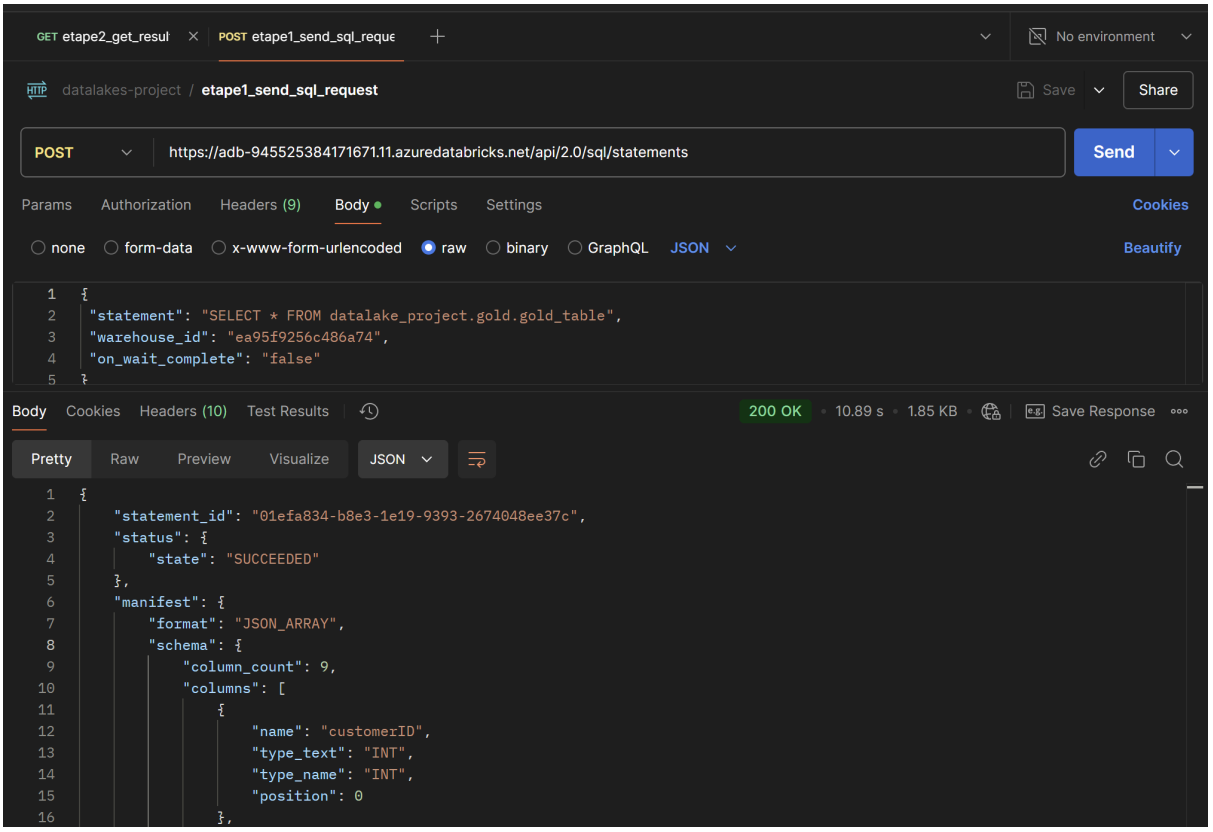


De plus, si nous voulons utiliser les commandes SQL, nous devons suivre les étapes suivantes :

1. utiliser la commande POST pour faire une demande de requête SQL à databricks, pour cela, nous devons utiliser la commande suivante :
POST https://adb-945525384171671.11.azuredatabricks.net/api/2.0/sql/statements
En outre, nous devons remplir le body de cet élément en format json pour contenir l'information de notre demande comme ceci :

```
{
 "statement": "SELECT * FROM datalake_project.gold.gold_table",
 "warehouse_id": "ea95f9256c486a74",
 "on_wait_complete": "false"
}
```

2. Après avoir cliqué sur envoyer, databricks renverra un json qui contient statement id comme suivant :



3. Après avoir obtenu statement id, nous pouvons envoyer la commande suivante à databricks pour obtenir le résultat final :

GET https://adb-9455253841716 71.11.azuredatabricks.net/api/2.0/sql/statements/<statement_id>

Partie 4 : Sécurité, Gouvernance et Qualité des Données

Unity catalog

Nous avons choisi d'utiliser ***Unity Catalog***, il s'agit d'un outil de catalogage qui permet de copier les métadonnées des données du Data Lake Azure, créées à partir de tables externes, sans risquer de compromettre l'intégrité des vraies données. Ainsi on peut interroger les métadonnées sans altérer les données originales.

Pour la gouvernance, ***Unity Catalog*** centralise la gestion des données, permettant une gouvernance simplifiée directement dans Databricks. Cela offre une vue unifiée sur les données, facilitant la classification des données structurées et non structurées. Concernant la sécurité, ***Unity Catalog*** offre un contrôle d'accès granulaire, où l'administrateur gère les permissions d'accès des utilisateurs au niveau d'une ou plusieurs tables ou bien un ou plusieurs schéma au sein d'un même catalogue (datalake). On utilise des politiques basées sur les attributs pour sécuriser l'accès aux données.

L'image ci-dessous est un exemple concret de la manière dont l'administrateur a pu gérer les permissions d'accessibilité aux groupes d'utilisateurs choisi :