

Projet de Data Engineering End-to-End sur Azure : Analyse des Données AdventureWorks

Gorelov Bogdan

github.com/Kepard

Master Informatique pour la Science des Données (ISD)

Université Paris-Saclay

4 juin 2025

Table des matières

1	Introduction	1
1.1	Importance du Data Engineering	1
1.2	Contexte et Motivation Personnelle	1
1.3	Objectifs du Projet	1
1.4	Technologies Utilisées	1
1.5	Structure du Document	2
2	Architecture du Pipeline de Données	2
2.1	Vue d'Ensemble	2
2.2	Description des Couches de Données (Medallion Architecture)	3
2.2.1	Couche Bronze (Raw Data Store)	3
2.2.2	Couche Silver (Processed Layer)	3
2.2.3	Couche Gold (High-Quality Layer / Serving Layer)	3
2.3	Justification des Choix Technologiques	3
3	Configuration de l'Environnement Azure	4
3.1	Création d'un Compte Azure	4
3.2	Ressources Azure Créées	4
4	Ingestion des Données (Couche Bronze)	5
4.1	Source de Données	5
4.2	Azure Data Factory (ADF)	6
4.3	Stockage dans ADLS Gen2 (Raw Data Store - Couche Bronze)	7
5	Transformation des Données avec Databricks (Couche Silver)	7
5.1	Configuration de l'Espace de Travail Databricks	7
5.2	Scripts PySpark pour la Transformation	9
5.3	Stockage dans ADLS Gen2 (Processed Layer - Couche Silver)	10
6	Analyse et Service des Données avec Azure Synapse Analytics (Couche Gold)	10
6.1	Configuration de l'Espace de Travail Synapse	10
6.2	Concept de Lakehouse et Économies de Coûts	11
6.3	Configuration de l'Accès aux Données depuis Synapse	11
6.4	Création de Vues SQL (Couche Gold) avec OPENROWSET	12
6.5	Création de Tables Externes (Alternative)	12
7	Visualisation avec Power BI	13
7.1	Connexion à Synapse Analytics	13
7.2	Exemples de Visualisations Simples	14
8	Conclusion et Perspectives	14
8.1	Résumé des Réalisations	14
8.2	Apprentissages Clés et Défis Rencontrés	15
8.3	Perspectives d'Amélioration	15

1 Introduction

1.1 Importance du Data Engineering

Le Data Engineering est la discipline qui consiste à concevoir, construire et maintenir les systèmes et les infrastructures qui permettent de collecter, stocker, traiter et analyser de grandes quantités de données. Dans un monde où la donnée est devenue un actif stratégique majeur, le Data Engineer joue un rôle crucial en s'assurant que les données sont fiables, accessibles et prêtes à être exploitées par les Data Scientists, les Data Analysts et les décideurs métier pour générer de la valeur.

1.2 Contexte et Motivation Personnelle

Travaillant actuellement chez Fnac en tant que représentant de la marque Microsoft, j'ai toujours été curieux de comprendre les mécanismes internes de gestion des données dans un environnement retail et e-commerce, notamment en ce qui concerne le suivi des ventes, la gestion des stocks et des commandes clients. Cette curiosité m'a motivé à entreprendre ce projet afin d'acquérir une expérience pratique des technologies et des architectures utilisées pour de tels systèmes. Le dataset 'AdventureWorks', simulant un cycle de vente complet avec des produits, des clients et des transactions, m'a semblé particulièrement pertinent pour explorer ces concepts.

1.3 Objectifs du Projet

- Mettre en œuvre un pipeline de données complet ("End-to-End") sur la plateforme Microsoft Azure.
- Ingérer des données brutes (fichiers CSV) depuis une source externe (simulée via GitHub).
- Stocker et transformer ces données en suivant une architecture en couches (Bronze, Silver, Gold / Medallion Architecture).
- Utiliser les services Azure clés : Azure Data Factory pour l'ingestion, Azure Data Lake Storage Gen2 pour le stockage, Azure Databricks (avec Apache Spark/PySpark) pour la transformation, et Azure Synapse Analytics pour la couche de service et d'analyse.
- Préparer les données pour une visualisation simple dans Power BI.

1.4 Technologies Utilisées

Les principales technologies et services utilisés dans ce projet sont :

- **Microsoft Azure Cloud Platform :**
 - Azure Data Factory (ADF V2)
 - Azure Data Lake Storage Gen2 (ADLS Gen2)
 - Azure Databricks
 - Azure Synapse Analytics (Pool SQL Serverless)
- **Langages & Frameworks :**
 - Apache Spark (via Databricks)
 - PySpark (Python pour Spark)
 - SQL (via Synapse Analytics)
- **Source de Données :** Fichiers CSV du dataset AdventureWorks, hébergés sur GitHub et accessibles via des URL `raw.githubusercontent.com`.
- **Visualisation :**
 - Microsoft Power BI
 - Capacités de visualisation intégrées à Azure Databricks (pour l'exploration)

1.5 Structure du Document

Ce document est structuré comme suit : la section 2 présente l'architecture globale du pipeline de données. La section 3 détaille la configuration de l'environnement Azure. Les sections 4, 5 et 6 décrivent respectivement les étapes d'ingestion, de transformation et de service des données. La section 7 montre l'intégration avec Power BI pour la visualisation. Enfin, la section 8 conclut ce rapport et évoque les perspectives d'amélioration.

2 Architecture du Pipeline de Données

2.1 Vue d'Ensemble

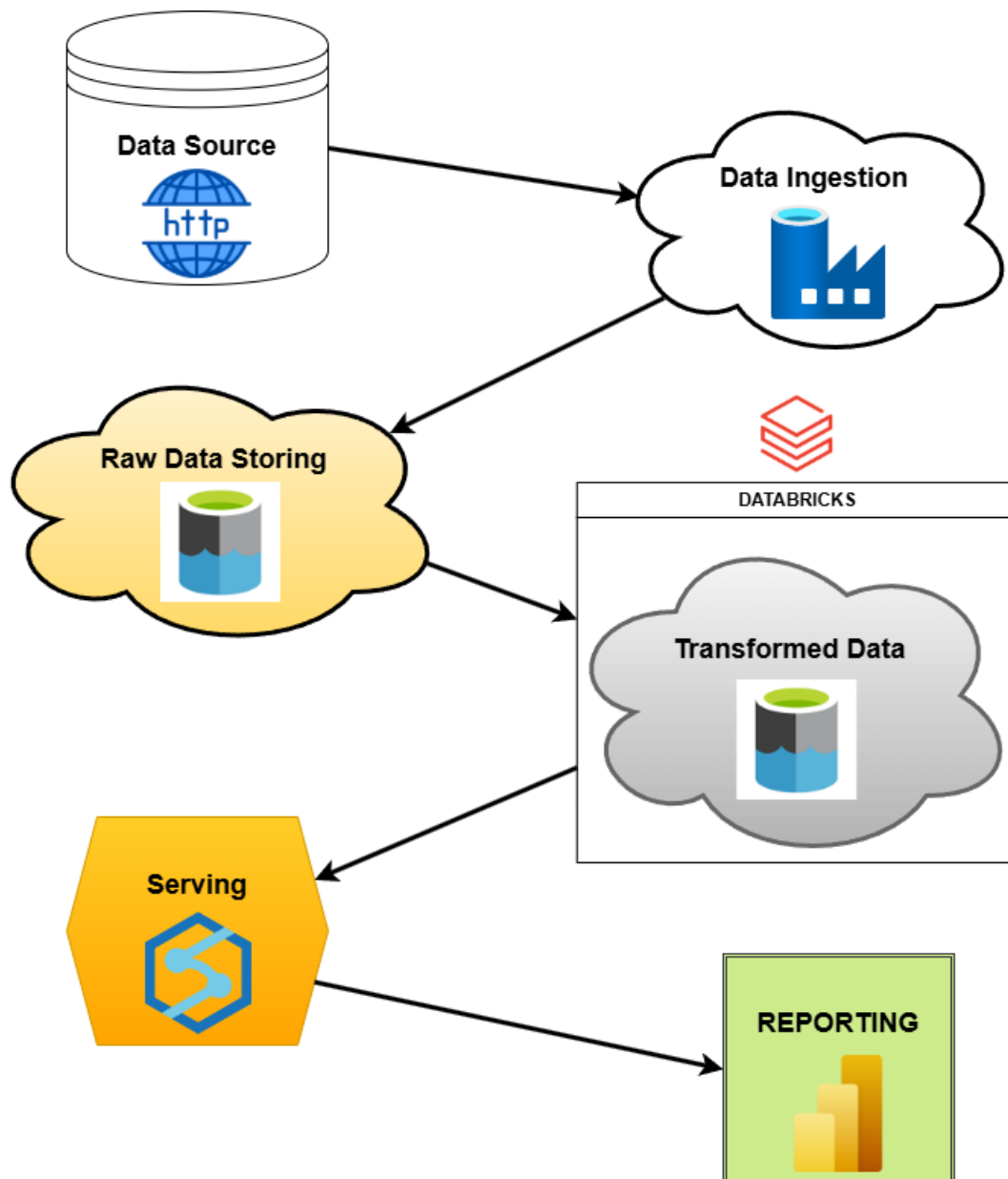


FIGURE 1 – *Architecture du Pipeline de Données End-to-End sur Azure.*

L'architecture mise en place suit un flux de données classique en Data Engineering (voir Figure 1). Les données brutes sont d'abord ingérées depuis une source externe via Azure Data

Factory et stockées dans une première couche ‘Bronze’ sur Azure Data Lake Storage Gen2. Ensuite, Azure Databricks est utilisé pour transformer ces données brutes en données nettoyées et structurées, stockées dans une couche ‘Silver’. Finalement, Azure Synapse Analytics permet d’exposer ces données transformées dans une couche ‘Gold’, optimisée pour l’analyse et le reporting, qui alimente ensuite des visualisations dans Power BI.

2.2 Description des Couches de Données (Medallion Architecture)

2.2.1 Couche Bronze (Raw Data Store)

- **Objectif** : Conserver une copie exacte des données sources, sans aucune transformation, pour assurer la traçabilité, la rejouabilité des traitements et l’audit.
- **Format** : Fichiers CSV originaux.
- **Stockage** : Conteneur `raw` dans ADLS Gen2.

2.2.2 Couche Silver (Processed Layer)

- **Objectif** : Fournir des données nettoyées, dédoublonnées, conformées, filtrées et enrichies. Ces données sont prêtes pour des analyses ad-hoc ou pour alimenter la couche Gold.
- **Format** : Fichiers Parquet (format colonnaire optimisé pour les performances avec Spark).
- **Stockage** : Conteneur `processed` dans ADLS Gen2.

2.2.3 Couche Gold (High-Quality Layer / Serving Layer)

- **Objectif** : Présenter les données sous une forme optimisée pour les utilisateurs finaux (analystes, outils de BI).
- **Format** : Exposées via des vues SQL et des tables externes dans Azure Synapse Analytics, pointant sur les fichiers Parquet ou sur un conteneur `high-quality` dédié dans ADLS Gen2.
- **Stockage** : Principalement via des abstractions Synapse sur ADLS Gen2 (conteneur `high-quality`).

2.3 Justification des Choix Technologiques

Les technologies Azure ont été choisies pour leur intégration native, leur scalabilité et leur popularité sur le marché du Data Engineering.

- **Azure Data Factory (ADF)** : Sélectionné pour l’ingestion et l’orchestration des données en raison de sa capacité à se connecter à une multitude de sources, sa facilité d’utilisation pour créer des pipelines graphiquement, et ses fonctionnalités de planification et de monitoring. Il est particulièrement adapté pour des tâches ETL/ELT à grande échelle.
- **Azure Data Lake Storage Gen2 (ADLS Gen2)** : Utilisé pour sa capacité à stocker de très grands volumes de données de formats variés à faible coût, tout en offrant des fonctionnalités de système de fichiers hiérarchique et une intégration fine avec les autres services Azure comme Databricks et Synapse.
- **Azure Databricks (Apache Spark/PySpark)** : Choisi pour les transformations de données complexes grâce à la puissance du moteur Apache Spark, qui permet des traitements distribués et performants. L’environnement de notebook collaboratif de Databricks et le langage PySpark offrent une grande flexibilité pour le nettoyage, l’enrichissement et l’analyse exploratoire des données.
- **Azure Synapse Analytics (Pool SQL Serverless)** : Privilégié pour la couche de service et d’analyse en raison de sa capacité à requêter directement les données stockées dans le Data Lake via une interface SQL familière (concept de Lakehouse). Le pool SQL

serverless est particulièrement intéressant pour sa flexibilité et son modèle de coût à l'usage.

3 Configuration de l'Environnement Azure

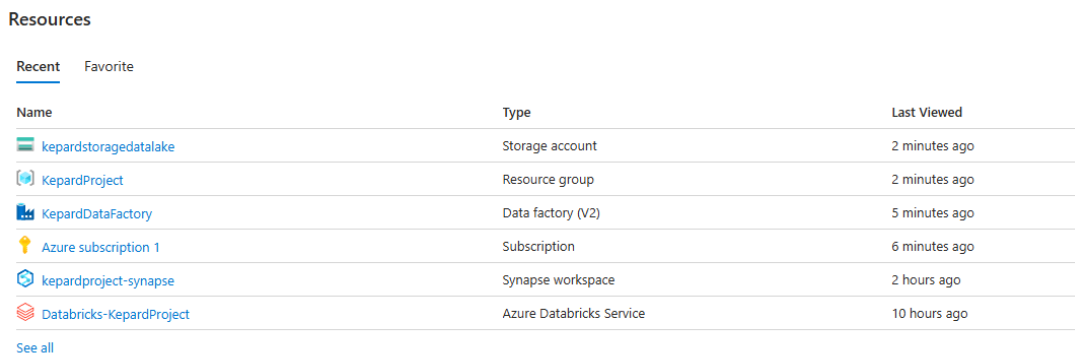
3.1 Création d'un Compte Azure

Ce projet a été réalisé en utilisant un compte Azure gratuit, ce qui permet d'explorer la majorité des services nécessaires pour un pipeline de Data Engineering.







3.2 Ressources Azure Créées

Les principales ressources Azure déployées pour ce projet sont regroupées au sein d'un groupe de ressources unique nommé **KepardProject** pour faciliter la gestion et le suivi (voir Figure 2).

- Compte de stockage (Storage account) : **kepardstoragedatalake** (Azure Data Lake Storage Gen2).
- Data Factory (V2) : **KepardDataFactory**.
- Azure Databricks Service : **Databricks-KepardProject**.
- Synapse workspace : **kepardproject-synapse**.



The screenshot shows the 'Resources' page in the Azure portal. It features a table with three columns: 'Name', 'Type', and 'Last Viewed'. The table lists several resources, including 'kepardstoragedatalake' (Storage account), 'KepardProject' (Resource group), 'KepardDataFactory' (Data factory (V2)), 'Azure subscription 1' (Subscription), 'kepardproject-synapse' (Synapse workspace), and 'Databricks-KepardProject' (Azure Databricks Service). The 'Recent' tab is selected, and a 'See all' link is at the bottom.

Resources		
Recent Favorite		
Name	Type	Last Viewed
 kepardstoragedatalake	Storage account	2 minutes ago
 KepardProject	Resource group	2 minutes ago
 KepardDataFactory	Data factory (V2)	5 minutes ago
 Azure subscription 1	Subscription	6 minutes ago
 kepardproject-synapse	Synapse workspace	2 hours ago
 Databricks-KepardProject	Azure Databricks Service	10 hours ago

[See all](#)

FIGURE 2 – *Ressources Azure déployées pour le projet.*

Les conteneurs suivants ont été créés dans le compte de stockage **kepardstoragedatalake** (voir Figure 3) : **raw**, **processed**, **high-quality** et **parameters**.

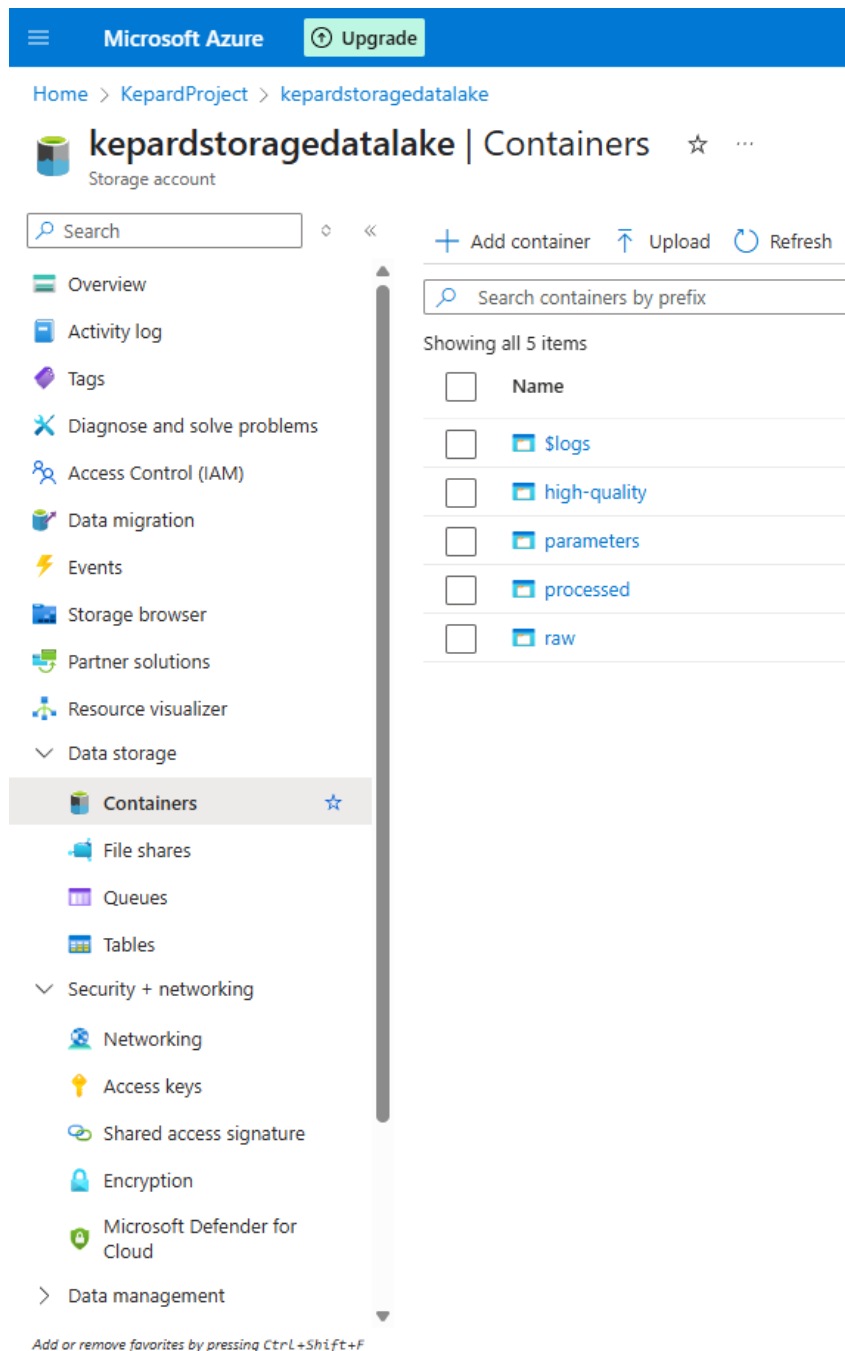


FIGURE 3 – Conteneurs dans Azure Data Lake Storage Gen2.

4 Ingestion des Données (Couche Bronze)

4.1 Source de Données

Les données sources proviennent du dataset ‘AdventureWorks’ (disponible sur Kaggle), qui simule les opérations d’une entreprise de fabrication et de vente de vélos. Pour ce projet, les fichiers CSV de ce dataset ont été hébergés sur un dépôt GitHub personnel (<https://github.com/Kepard/kepard-data-engineering/tree/main/Data>). L’accès aux fichiers bruts se fait via les URL `raw.githubusercontent.com` qui servent de point d’API pour l’ingestion. La structure des données est tabulaire, chaque fichier CSV représentant une entité métier.

4.2 Azure Data Factory (ADF)

Un pipeline dynamique nommé `DynamicGitToRaw` a été créé dans Azure Data Factory pour ingérer les différents fichiers CSV depuis GitHub vers la couche Bronze de notre Data Lake (voir Figure 4).

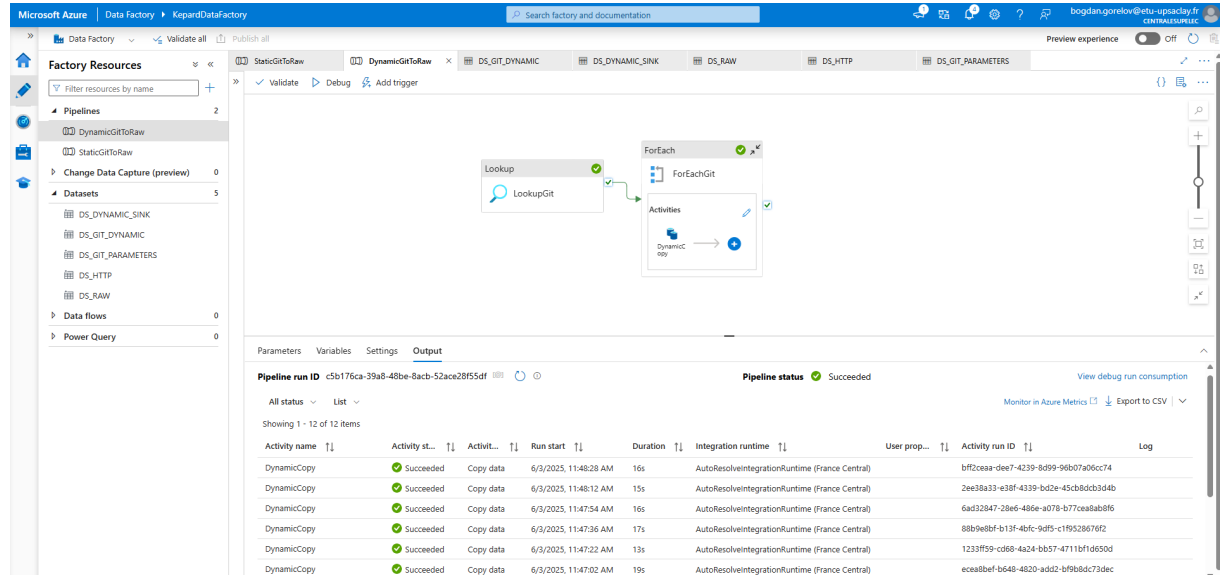


FIGURE 4 – Pipeline dynamique d’ingestion `DynamicGitToRaw` dans Azure Data Factory.

Ce pipeline utilise les activités suivantes :

- **Lookup Activity (LookupGit)** : Cette activité lit un fichier de configuration `git.json` (voir Listing 1) stocké dans le conteneur `parameters` d’ADLS Gen2. Ce fichier contient une liste d’objets JSON, chacun décrivant l’URL relative d’un fichier CSV sur GitHub, ainsi que le dossier et le nom de fichier de destination dans la couche `raw` du Data Lake.
- **ForEach Activity (ForEachGit)** : Cette activité itère sur chaque élément (chaque fichier à ingérer) retourné par l’activité Lookup.
- **Copy Data Activity (nommée DynamicCopy à l’intérieur du ForEach)** : Pour chaque fichier CSV, cette activité est configurée dynamiquement :
 - **Source** : Se connecte à la source HTTP (GitHub) en utilisant l’URL relative (ex : `@item().p_relative_url`) et un service lié HTTP paramétré.
 - **Sink (Destination)** : Écrit les données dans le dossier et le fichier spécifiés (ex : `@item().p_sink_folder`, `@item().p_sink_file`) dans le conteneur `raw` d’ADLS Gen2, via un service lié Azure Data Lake Storage Gen2. Les fichiers sont conservés au format CSV dans cette couche.

```

1 [
2   {
3     "p_relative_url": "Kepard/kepard-data-engineering/refs/heads/main/Data/AdventureWorks_Products.csv",
4     "p_sink_folder": "AdventureWorks_Products",
5     "p_sink_file": "AdventureWorks_Products.csv"
6   },
7   {
8     "p_relative_url": "Kepard/kepard-data-engineering/refs/heads/main/Data/AdventureWorks_Calendar.csv",
9     "p_sink_folder": "AdventureWorks_Calendar",
10    "p_sink_file": "AdventureWorks_Calendar.csv"
11  },
12  % ... (autres fichiers listes de la meme maniere) ...
13  {
14    "p_relative_url": "Kepard/kepard-data-engineering/refs/heads/main/Data/AdventureWorks_Territories.csv",
15    "p_sink_folder": "AdventureWorks_Territories",
16    "p_sink_file": "AdventureWorks_Territories.csv"
17  }
18 ]

```

Listing 1 – Contenu du fichier de configuration `git.json` pour l’ingestion dynamique.

4.3 Stockage dans ADLS Gen2 (Raw Data Store - Couche Bronze)

Les fichiers CSV ingérés sont stockés dans des sous-dossiers dédiés au sein du conteneur **raw** d'ADLS Gen2, comme illustré en Figure 5.

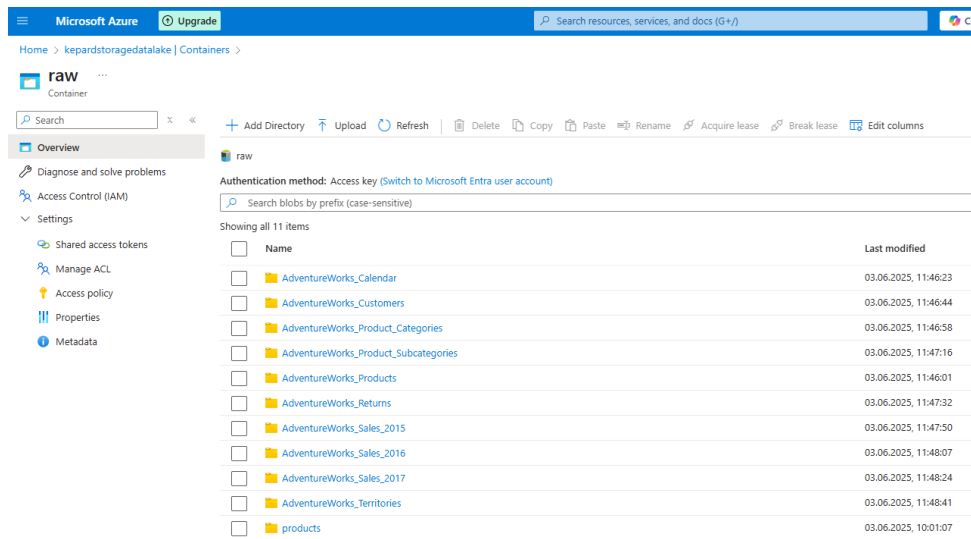


FIGURE 5 – Structure des données brutes dans ADLS Gen2 (Couche Bronze).

5 Transformation des Données avec Databricks (Couche Silver)

5.1 Configuration de l'Espace de Travail Databricks

Un cluster Azure Databricks a été configuré pour exécuter les transformations PySpark (voir Figure 6).

The screenshot shows the Databricks configuration interface for a cluster named 'KopardProject'. The 'Compute' tab is active, and the 'Simple form: OFF' dropdown is visible. The 'Configuration' tab is selected, showing the following settings:

- Policy:** Unrestricted
- Access mode:** No isolation shared
- Performance:**
 - Databricks Runtime Version:** 15.4 LTS (includes Apache Spark 3.5.0, Scala 2.12)
 - Node type:** Standard_D4s_v3 (16 GB Memory, 4 Cores)
 - Use Photon Acceleration:** Unchecked
 - Terminate after:** 20 minutes of inactivity (checked)
- Tags:** No custom tags
- Advanced options:** Collapsed

FIGURE 6 – Configuration du cluster Azure Databricks.

Le type de nœud `Standard_D4s_v3` (16 Go de mémoire, 4 cœurs) a été choisi, représentant un équilibre coût/performance pour ce dataset. La version `15.4 LTS` du Databricks Runtime (incluant Apache Spark 3.5.0) a été utilisée. L'accès à ADLS Gen2 depuis Databricks a été configuré en utilisant la clé d'accès du compte de stockage directement dans la configuration Spark du notebook (voir Listing 2). Bien que simple pour le développement, des méthodes comme les Service Principals ou la délégation d'informations d'identification sont recommandées en production.

```
1 spark.conf.set(
2   "fs.azure.account.key.kepardstoragedatalake.dfs.core.windows.net",
```

```

3 "YOUR_ADLS_ACCESS_KEY" # Remplace par une mention generique pour la
  documentation
4 )

```

Listing 2 – Configuration de l'accès à ADLS Gen2 dans Databricks.

5.2 Scripts PySpark pour la Transformation

Un notebook PySpark (processed_layer.ipynb) a été développé pour les transformations. Les étapes clés incluent :

- **Lecture des données CSV** : Chargement des fichiers depuis le conteneur **raw** en inférant le schéma.

```

1 # Import des fonctions Pyspark necessaires
2 from pyspark.sql.functions import *
3
4 df_cal = spark.read.format("csv").option("header", "true").option("
  inferSchema", True).load("abfss://raw@kepardstoragedatalake.dfs.core.
  windows.net/AdventureWorks_Calendar")
5
6 df_sal = spark.read.format("csv").option("header", "true").option("
  inferSchema", True).load("abfss://raw@kepardstoragedatalake.dfs.core.
  windows.net/AdventureWorks_Sales*")
7
8 # ... chargement similaire pour df_cus, df_procat, df_prosub, df_pro,
  df_ret, df_ter ...
9

```

Listing 3 – Lecture des données CSV (exemple pour Calendar et Sales).

- **Nettoyage et Transformation** : Application de diverses transformations (voir Listings 4 et 5).

```

1 # Transformation sur df_cus (Customers)
2 df_cus = df_cus.withColumn("FullName", concat_ws(" ", col("Prefix") , col(
  "FirstName"), col("LastName")))
3 df_cus = df_cus.drop("Prefix", "FirstName", "LastName")
4
5 # Transformation sur df_pro (Products)
6 df_pro = df_pro.withColumn("ProductSKU", split(col("ProductSKU"), "-").
  getItem(0)) \
7   .withColumn("ProductName", split(col("ProductName"), " ").
  getItem(0))
8

```

Listing 4 – Exemples de transformations sur Customers et Products.

```

1 # Transformation sur df_sal (Sales) - Dates et OrderNumber
2 df_sal = df_sal.withColumn("OrderDate", to_date(col("OrderDate"), "M/d/
  yyyy"))
3 df_sal = df_sal.withColumn("StockDate", to_date(col("StockDate"), "M/d/
  yyyy")) # Garder en type Date
4 df_sal = df_sal.withColumn("OrderNumber", regexp_replace(col("OrderNumber"
  ), "SO", ""))
5
6 # Transformation sur df_cal (Calendar) - Extraction Mois/Ann e
7 df_cal = df_cal.withColumn("Date", to_date(col("Date"), "M/d/yyyy"))
8 df_cal = df_cal.withColumn("Month", month(col("Date"))) \
  .withColumn("Year", year(col("Date")))
9
10

```

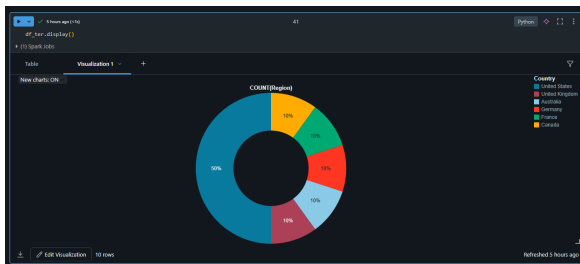
Listing 5 – Exemples de transformations sur Sales et Calendar.

- **Écriture en format Parquet** : Les DataFrames transformés sont écrits au format Parquet dans le conteneur `processed`. Le mode `append` a été utilisé ici ; `overwrite` serait plus typique pour un batch complet.

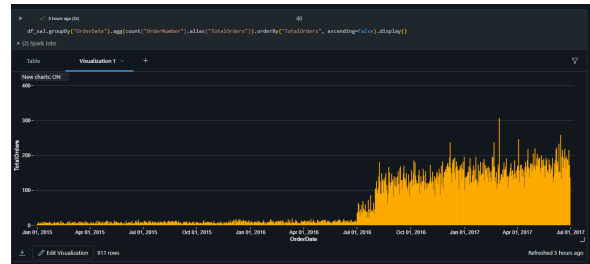
```
1 df_cus.write.format('parquet').mode('append').option("path", "abfss://
  processed@kepardstoragedatalake.dfs.core.windows.net/
  AdventureWorks_Customers").save()
2 # ... ecritures similaires pour les autres dataframes ...
3
```

Listing 6 – Écriture des données transformées en Parquet (exemple pour Customers).

Des visualisations exploratoires ont été générées dans le notebook Databricks (voir Figures 7a et 7b).



(a) Répartition des régions (Territories).



(b) Nombre de commandes par date (Sales).

FIGURE 7 – Visualisations exploratoires dans Azure Databricks.

5.3 Stockage dans ADLS Gen2 (Processed Layer - Couche Silver)

Les données transformées sont stockées au format Parquet dans des sous-dossiers dédiés au sein du conteneur `processed` (voir Figure 8). Le format Parquet est privilégié pour sa compression efficace et ses performances avec les moteurs analytiques.

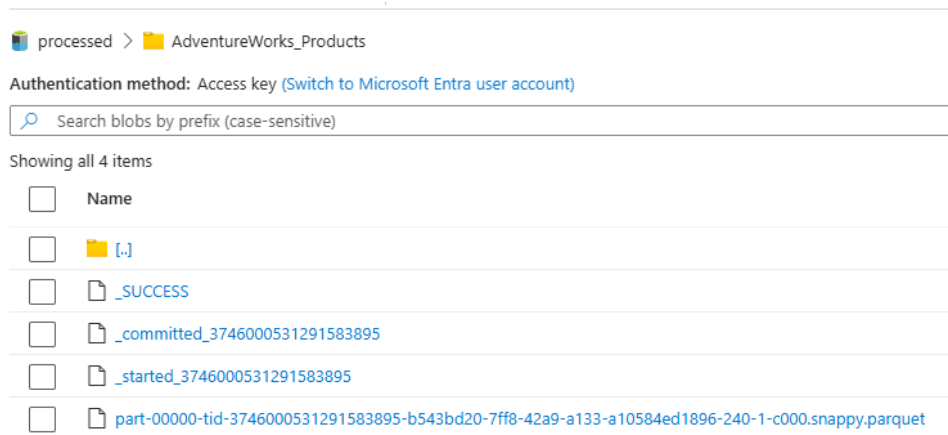


FIGURE 8 – Données transformées (Parquet) dans ADLS Gen2 (Couche Silver).

6 Analyse et Service des Données avec Azure Synapse Analytics (Couche Gold)

6.1 Configuration de l'Espace de Travail Synapse

L'espace de travail Azure Synapse Analytics a été utilisé, en s'appuyant principalement sur son pool SQL serverless.

6.2 Concept de Lakehouse et Économies de Coûts

Ce projet met en œuvre une architecture de type **Lakehouse**. Le Lakehouse combine la flexibilité et l'économie du Data Lake avec les fonctionnalités de gestion, de performance et de gouvernance des Data Warehouses.

Fonctionnement concret : Les données (brutes et transformées, ici en format Parquet) résident physiquement dans Azure Data Lake Storage Gen2. Le pool SQL serverless d'Azure Synapse Analytics agit comme un moteur de requêtage puissant qui lit ces données *directement* depuis le Data Lake. Il n'est pas nécessaire de dupliquer ou de déplacer les données dans un système de stockage propriétaire de Data Warehouse. Une couche de métadonnées (définie par les tables externes et les vues dans Synapse) est appliquée sur les fichiers du Data Lake, permettant de les structurer, de leur appliquer un schéma et de les interroger avec le langage SQL standard.

Économies de coûts :

- **Stockage optimisé :** ADLS Gen2 offre un coût de stockage significativement plus bas que les systèmes de stockage des Data Warehouses traditionnels provisionnés.
- **Découplage calcul/stockage :** Avec le pool SQL serverless, les coûts de calcul sont basés sur la quantité de données traitées par chaque requête (modèle "pay-as-you-go"). Il n'y a pas de coût pour un cluster de calcul qui tournerait en permanence s'il n'y a pas de requêtes.
- **Flexibilité des données :** Moins de transformations initiales coûteuses sont nécessaires pour commencer à explorer les données, car le Lakehouse peut gérer divers formats.

La Figure 9 illustre ce concept dans notre projet.

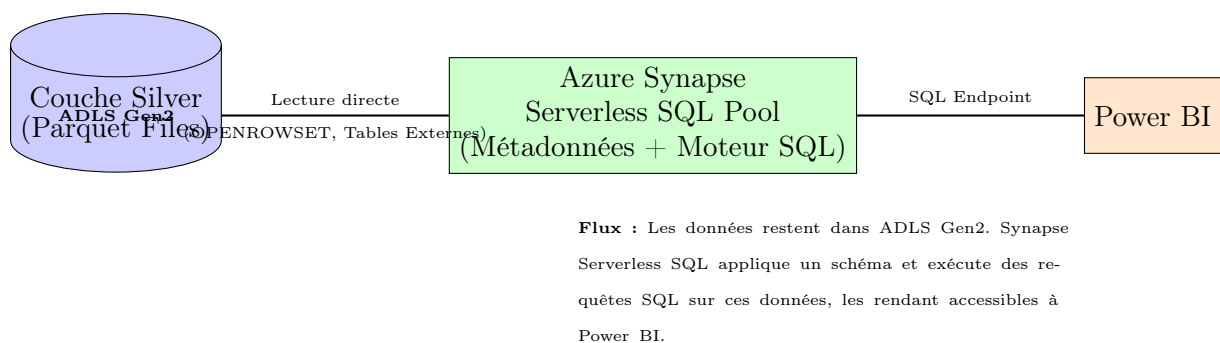


FIGURE 9 – Concept du Lakehouse avec Synapse Serverless SQL Pool dans le projet.

6.3 Configuration de l'Accès aux Données depuis Synapse

Pour permettre à Synapse d'accéder aux données dans ADLS Gen2 (voir Figure 10 pour la structure résultante), les objets suivants ont été créés via des scripts SQL :

- **DATABASE SCOPED CREDENTIAL (cred_bogdan) :** Créée en utilisant l'identité managée pour l'authentification de Synapse auprès d'ADLS Gen2.
- **EXTERNAL DATA SOURCE (source_processed, source_highquality) :** Pointe vers les conteneurs processed et high-quality dans ADLS Gen2.
- **EXTERNAL FILE FORMAT (format_parquet) :** Spécifie le format Parquet et la compression Snappy.

```
1 -- Creation de la credential (exemple avec Managed Identity)
2 CREATE DATABASE SCOPED CREDENTIAL cred_bogdan
3     WITH IDENTITY = 'Managed Identity';
4
5 -- Creation de la source de données pour le conteneur 'processed'
6 CREATE EXTERNAL DATA SOURCE source_processed
7 WITH (
8     LOCATION = 'https://kepardstoragedatalake.dfs.core.windows.net/processed',
9     CREDENTIAL = cred_bogdan -- Utilisation de la credential d'identité
```

```

10 );
11
12 -- Creation du format de fichier externe pour Parquet
13 CREATE EXTERNAL FILE FORMAT format_parquet
14 WITH ( FORMAT_TYPE = PARQUET, DATA_COMPRESSION = 'org.apache.hadoop.io.compress.
    SnappyCodec');

```

Listing 7 – Extraits du script de configuration Synapse.

6.4 Création de Vues SQL (Couche Gold) avec OPENROWSET

Des vues SQL ont été créées dans un schéma `highquality` pour exposer les données transformées de la couche Silver. Ces vues utilisent `OPENROWSET` pour lire directement les fichiers Parquet (voir Listing 8). Une vue stocke une requête et ne duplique pas les données.

```

1 CREATE SCHEMA highquality;
2 GO
3
4 CREATE VIEW highquality.sales
5 AS SELECT * FROM OPENROWSET(
6     BULK 'https://kepardstoragedatalake.dfs.core.windows.net/processed/
    AdventureWorks_Sales/',
7     FORMAT = 'PARQUET'
8 ) as query1;
9 GO
10 -- ... (creation similaire pour les autres vues : calendar, customers, products,
    etc.)
11 CREATE VIEW highquality.territories
12 AS SELECT * FROM OPENROWSET(
13     BULK 'https://kepardstoragedatalake.dfs.core.windows.net/processed/
    AdventureWorks_Territories/',
14     FORMAT = 'PARQUET'
15 ) as query1;
16 GO

```

Listing 8 – Exemple de création de vues dans Synapse.

6.5 Création de Tables Externes (Alternative)

À titre d'exemple, une table externe `highquality.extsales` a aussi été créée. Elle référence les données dans le Data Lake mais possède un schéma défini, offrant une expérience plus proche d'une table SQL traditionnelle.

```

1 CREATE EXTERNAL TABLE AS SELECT highquality.extsales (
2     OrderDate DATE,
3     StockDate DATE,
4     OrderNumber VARCHAR(50),
5     ProductKey INT,
6     CustomerKey INT,
7     TerritoryKey INT,
8     OrderLineItem INT,
9     OrderQuantity INT
10 ) WITH (
11     LOCATION = 'extsales/', -- Dossier dans le conteneur point par
    source_highquality
12     DATA_SOURCE = source_highquality,
13     FILE_FORMAT = format_parquet
14 );

```

Listing 9 – Exemple de création d'une table externe dans Synapse.

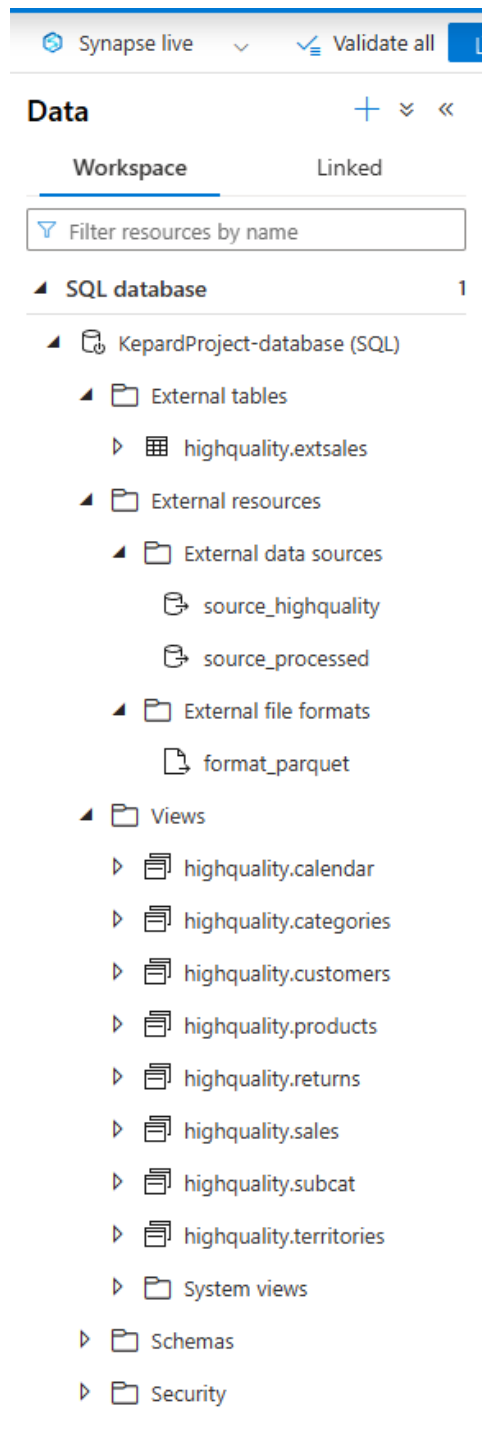


FIGURE 10 – *Structure des données (vues et tables externes) dans Azure Synapse Analytics.*

7 Visualisation avec Power BI

7.1 Connexion à Synapse Analytics

Power BI Desktop a été utilisé pour se connecter à la couche Gold exposée par Azure Synapse Analytics via son pool SQL serverless endpoint (voir Figure 11). L'authentification s'est faite en utilisant les informations d'identification du compte Microsoft.

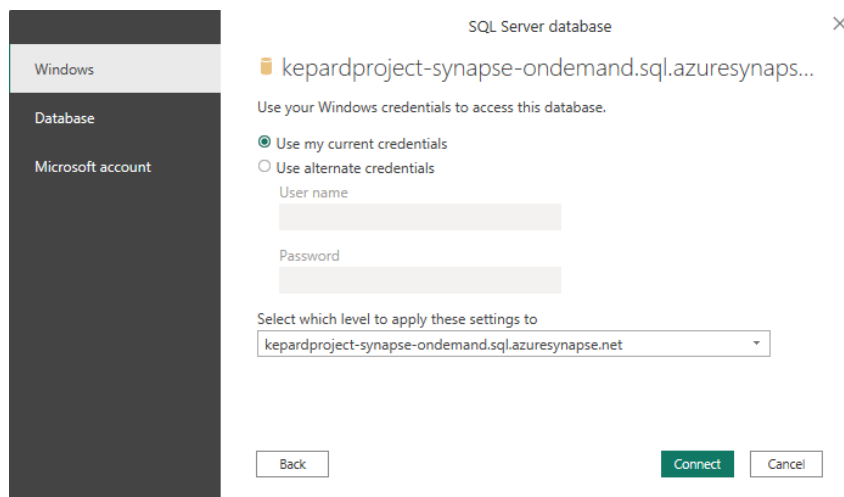


FIGURE 11 – Configuration de la connexion Power BI à Azure Synapse Analytics.

7.2 Exemples de Visualisations Simples

Quelques visualisations basiques ont été créées pour illustrer l’exploitation des données (voir Figure 12) : un diagramme circulaire montrant la répartition des clients par genre, et un histogramme empilé affichant le revenu annuel par profession.

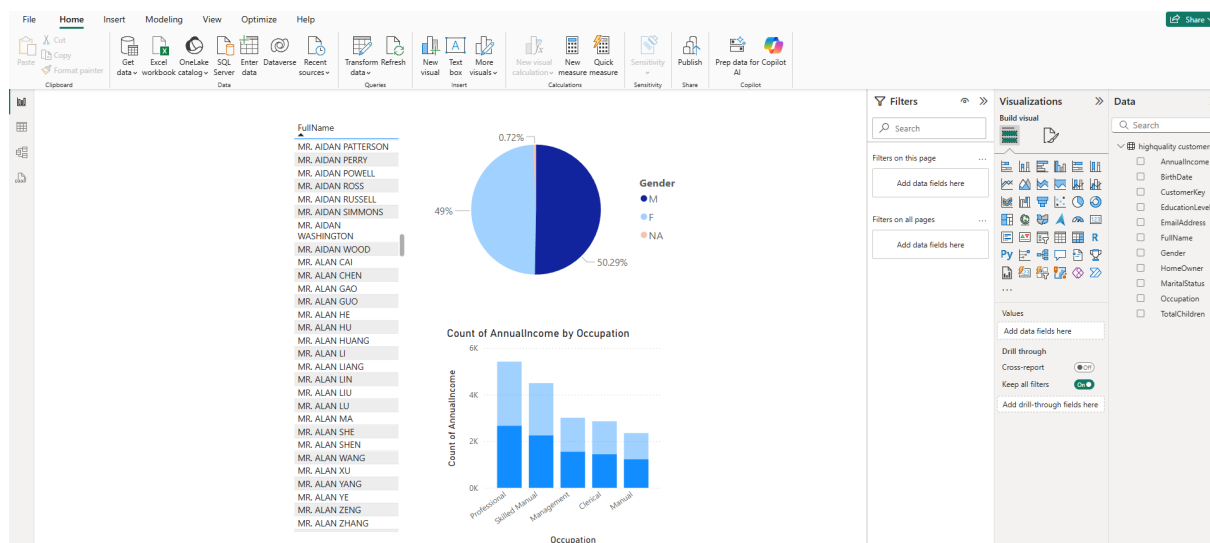


FIGURE 12 – Tableau de bord Power BI avec visualisations des données AdventureWorks.

8 Conclusion et Perspectives

8.1 Résumé des Réalisations

Ce projet a permis de mettre en œuvre avec succès un pipeline de données end-to-end sur la plateforme Azure, depuis l’ingestion de données brutes jusqu’à leur visualisation, en passant par des étapes de stockage, de transformation et de service. Il a démontré l’intégration des services Azure Data Factory, ADLS Gen2, Databricks et Synapse Analytics pour construire une architecture Medallion (Bronze, Silver, Gold) et explorer le concept de Lakehouse.

8.2 Apprentissages Clés et Défis Rencontrés

L'un des principaux apprentissages a été la compréhension de l'interaction et de l'intégration entre les différents services Azure. Configurer correctement les droits d'accès et les connexions (par exemple, entre Databricks et ADLS Gen2 via clé d'accès, ou entre Synapse et ADLS Gen2 en utilisant l'IAM du compte de stockage pour gérer l'accès) a représenté un défi initial, soulignant l'importance de la gestion des identités et des accès dans le cloud. La prise en main d'Azure Synapse Analytics, notamment le pool SQL serverless et la distinction entre vues `OPENROWSET` et tables externes, a également été un point d'apprentissage important. Le développement du pipeline dynamique dans ADF pour ingérer plusieurs fichiers à partir d'une configuration JSON a été une expérience formatrice.

8.3 Perspectives d'Amélioration

Ce projet a posé des bases solides. Pour aller plus loin, les prochaines étapes pourraient inclure la mise en place de **déclencheurs (triggers)** dans Azure Data Factory pour automatiser l'exécution du pipeline, par exemple lors de la mise à jour des datasets sources. De plus, des **transformations de données plus poussées et spécifiques aux besoins métier** pourraient être développées dans Databricks, en explorant des techniques d'agrégation plus complexes ou d'enrichissement de données avancé. Enfin, l'intégration de **tests de qualité de données automatisés** à différentes étapes du pipeline (par exemple avec des assertions en PySpark ou des outils dédiés) serait essentielle pour garantir la fiabilité et la robustesse de la solution dans un contexte plus opérationnel.