Projet Data Engineering on Cloud:

Medical Data Warehouse

Mastère Data Engineering & IA

Réalisé par :

- Darryll Genève Junior JOSEPH
- Mariem BEN SALAH

Année scolaire : 2024 – 2025

Résumé du projet :

Dans le cadre de ce projet de data engineering sur le cloud, notre mission a été de concevoir une infrastructure robuste, scalable et sécurisée, capable de collecter, stocker, transformer, modéliser et visualiser des données médicales hétérogènes issues d'un environnement hospitalier simulé.

Nous avons mis en place une architecture cloud-native complète sur **Microsoft Azure**, s'appuyant sur le modèle Lakehouse et le framework ELT, en exploitant des technologies modernes pour automatiser l'ingestion, le traitement et l'analyse des données à des fins décisionnelles.

Objectifs du projet :

- Créer un pipeline de traitement de données structuré autour des couches **Bronze, Silver et Gold**
- Nettoyer, enrichir et standardiser des données hospitalières simulées
- Réaliser des agrégations et extractions de KPI médico-financiers
- Mettre en place un **data model en étoile** dans **Azure Synapse**
- Automatiser l'exécution des traitements avec **Azure Data Factory**
- Construire des dashboards cliniques interactifs avec **Power BI**

Jeu de données utilisé :

- **Nom :** *MedSynora – Medical Data Warehouse*
- **Provenance :** Kaggle
- **Contenu :** données simulées sur l'année 2024 incluant les patients, consultations, diagnostics, traitements, coûts, signes vitaux, etc.

Technologies utilisées :

- Cloud & Stockage : Azure Data Lake Storage Gen2 (ADLS)
- Traitement & Ingestion : Azure Databricks (PySpark), GitLab API
- Orchestration : Azure Data Factory
- Modélisation : Azure Synapse Analytics (modèle en étoile)
- Visualisation : Power BI connecté à Synapse Serverless
- Sécurité : IAM, Managed Identity, Chiffrement AES-256

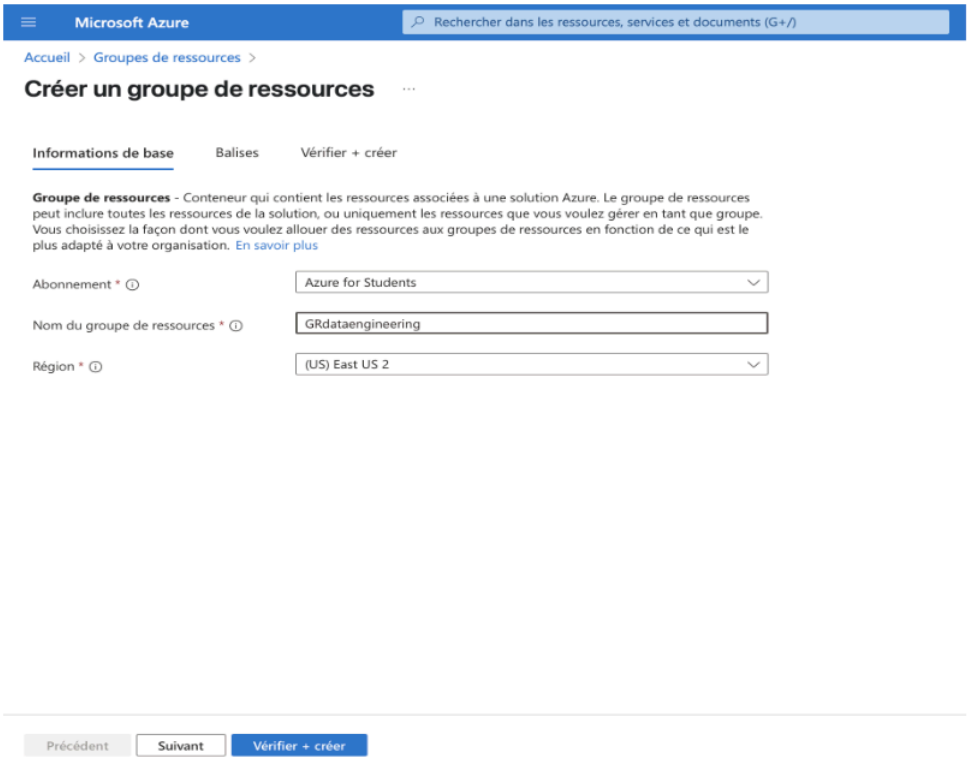
Dans le cadre de ce projet, notre objectif était de concevoir une infrastructure cloud robuste et scalable, permettant la collecte, le stockage, le traitement, la modélisation et la visualisation de données hétérogènes issues d'un environnement hospitalier simulé. Pour cela, nous avons structuré notre projet sur Microsoft Azure.

Les données utilisées dans ce projet proviennent du jeu de données synthétique MedSynora DW (Data Warehouse), disponible sur Kaggle. Ce dataset complet simule l'activité hospitalière sur une année 2024, en incluant les consultations, traitements, diagnostics, signes vitaux et autres paramètres médico-économiques.

1. Mise en place de l'environnement Cloud sur Azure :

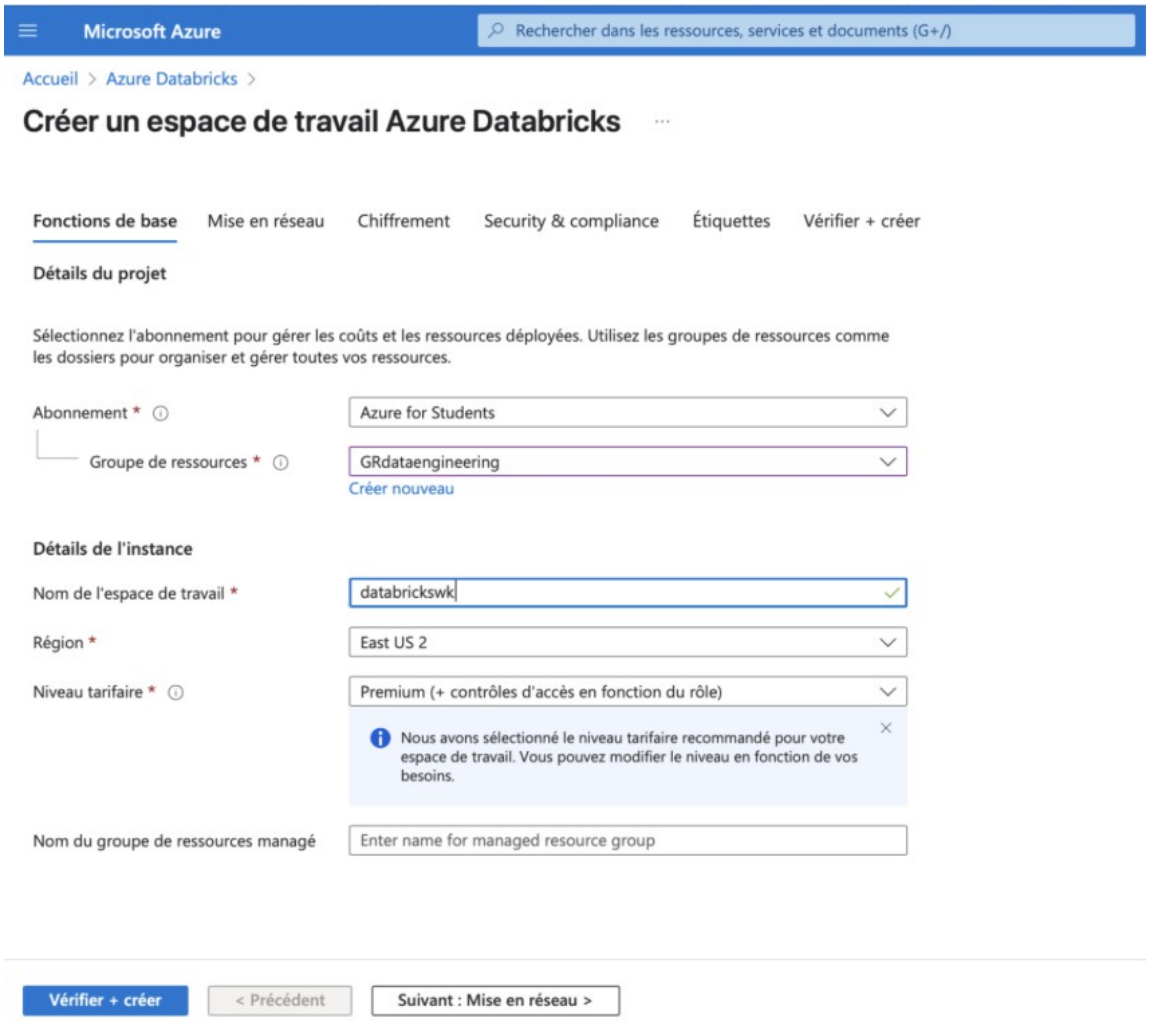
A. Création du groupe de ressources

La première étape de notre projet a été la structuration de notre environnement de travail sur Microsoft Azure. Nous avons commencé par la création d'un groupe de ressources intitulé **GRdataengineering**. Ce groupe joue un rôle essentiel en tant qu'enveloppe logique permettant de centraliser la gestion des autorisations, des coûts et de la supervision des services cloud associés à notre projet.



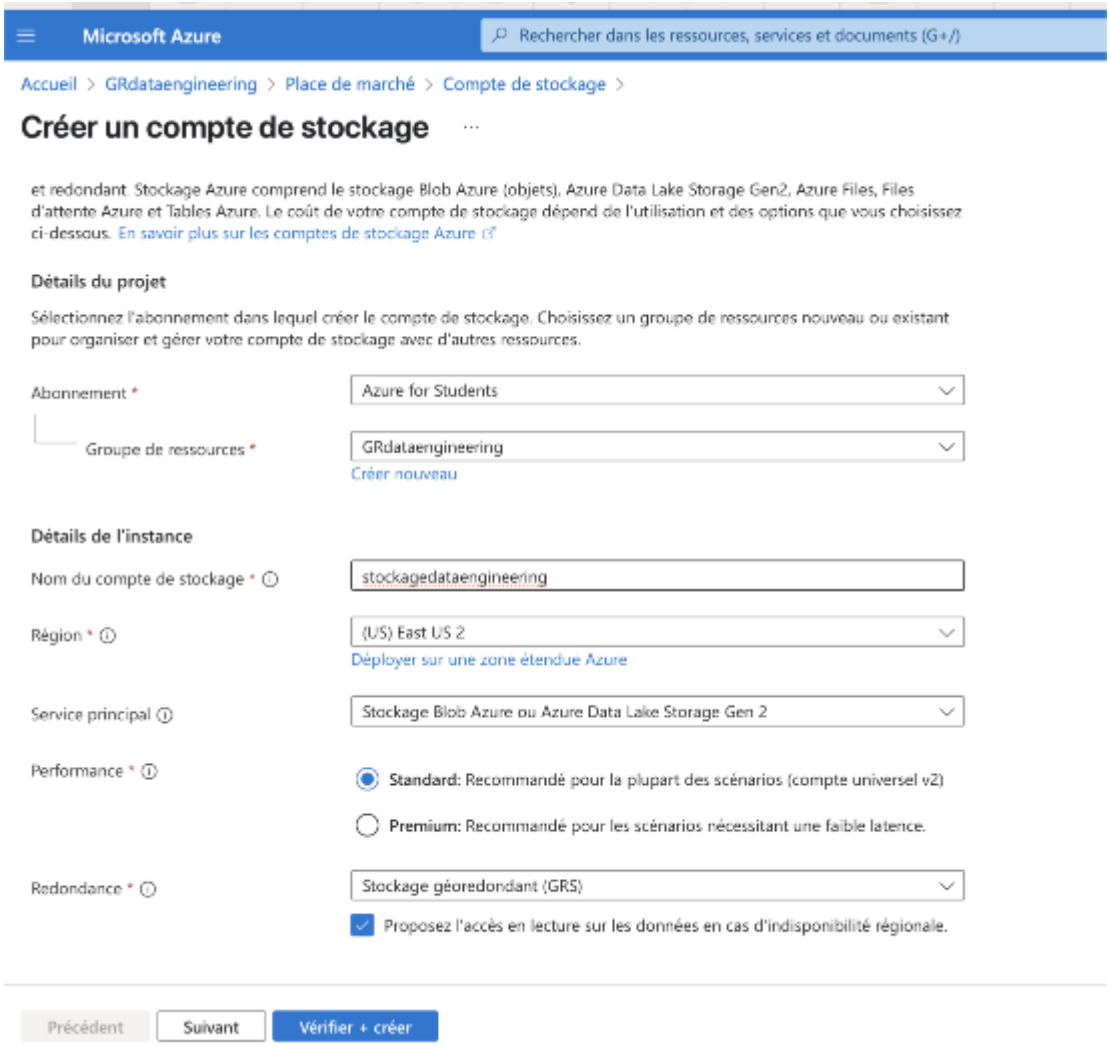
B. Création de l’espace de travail Databricks :

La première étape technique a été la mise en place d’un espace de travail Databricks (databrickswk)dans la région EAST US 2 pour exécuter les notebooks de traitement PySpark, ainsi qu’un compte de stockage Azure Data Lake Storage Gen2. Ce compte a été configuré avec l’option de namespace hiérarchique activée, afin d’assurer une gestion efficace et granulaire des fichiers stockés et avec un **niveau tarifaire Premium**, adapté à des besoins de contrôle d’accès et de scalabilité.



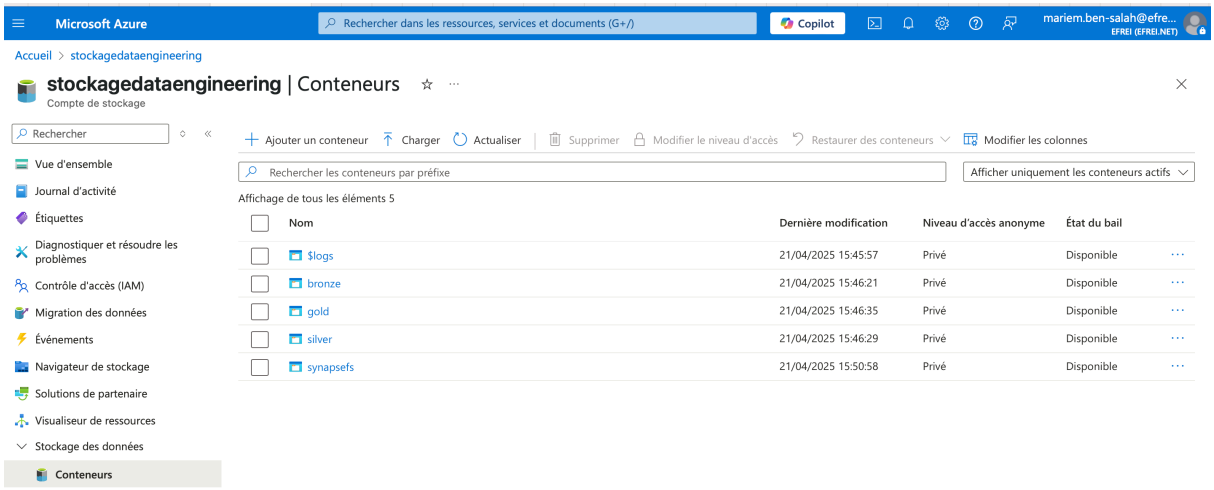
C. Mise en place du stockage ADLS Gen2 :

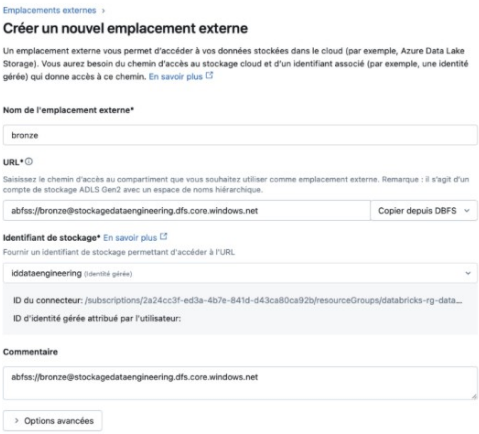
Nous avons créé un compte de stockage ADLS Gen2 (stockagedataengineering) avec l'option de noms hiérarchiques activée pour une meilleure gestion des fichiers.

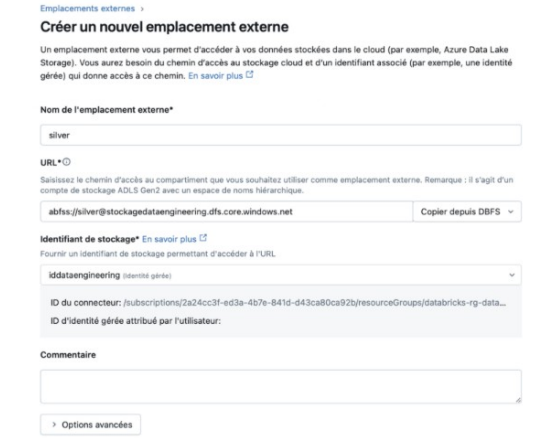


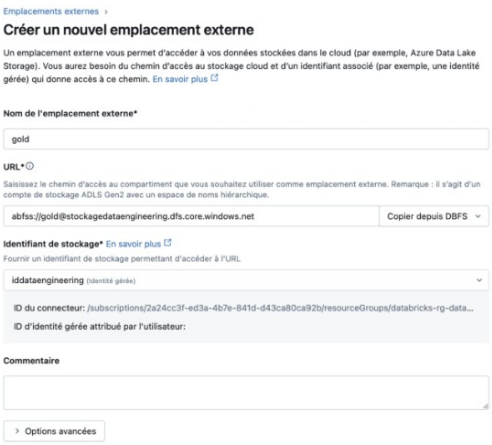
2. Structuration de la chaîne de traitement

Trois conteneurs de stockage ont été définis : bronze, silver, et gold, chacun représentant un niveau de traitement des données.









Nous avons utilisé l'identité gérée IDdataengineering pour sécuriser et authentifier l'accès aux conteneurs et les chemins abfss://... sont bien utilisés, ce qui garantit la connexion cloud-native entre Databricks et ADLS.

A. Mise en œuvre de la couche Bronze :

Après avoir créé les ressources Azure (groupe de ressources, Databricks, Synapse, stockage), nous avons implémenté notre premier notebook sur Databricks, dédié à la couche bronze.

L'objectif principal de cette couche est de centraliser et archiver les données brutes en provenance des différentes sources, ici simulées à travers un dépôt GitLab contenant des fichiers CSV issus du jeu de données MedSynora Data Warehouse. Le script Databricks interagit directement avec l'API GitLab pour automatiser la récupération et la sauvegarde des fichiers dans le conteneur Azure via le protocole ABFSS (Azure Blob File System Secure).

Comme illustré ci-dessous, le script commence par définir les chemins des conteneurs (bronze, silver, gold) :

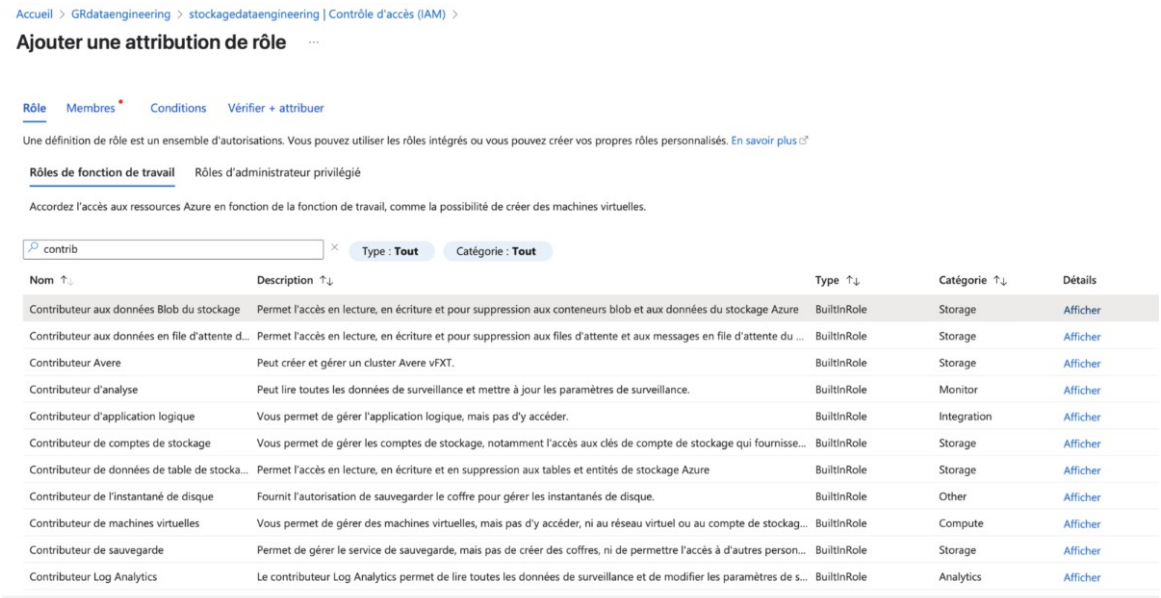
```
tiers = ["bronze", "silver", "gold"]
adls_paths={tier:f"abfss://{tier}@stockagedataengineering.dfs.core.windows.net/"fortierin tiers}

#Accessing paths
bronze_adls=adls_paths["bronze"]
silver_adls = adls_paths["silver"]
gold_adls = adls_paths["gold"]


dbutils.fs.ls(bronze_adls)
dbutils.fs.ls(silver_adls)
dbutils.fs.ls(gold_adls)
```

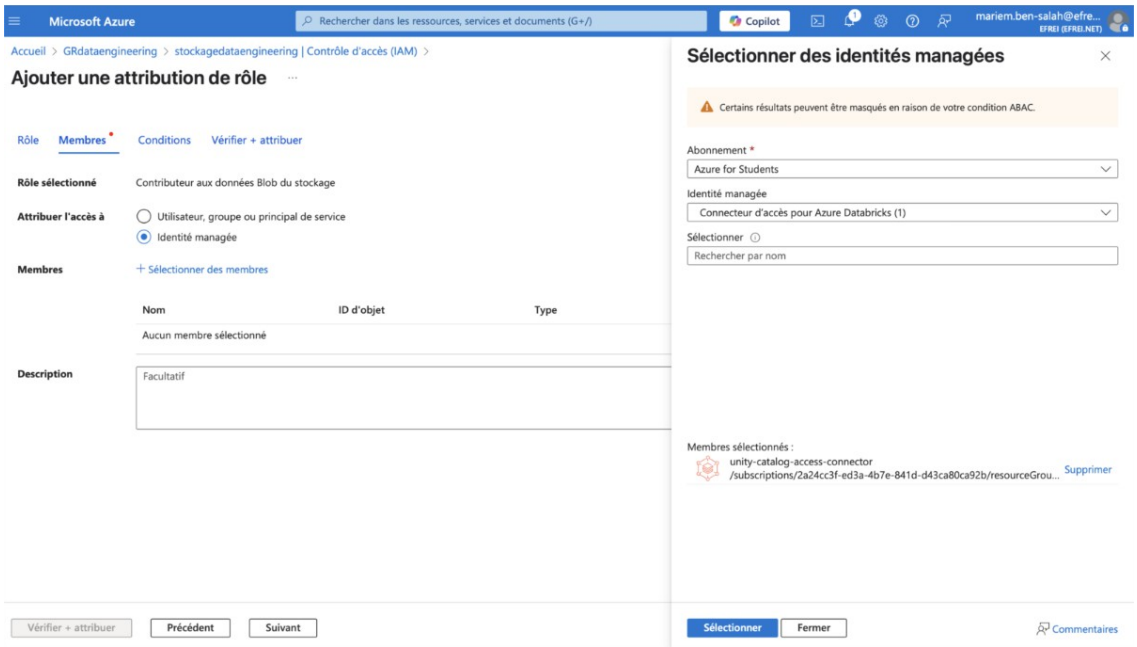
Puis, il utilise les APIs GitLab avec une authentification via *token privé* pour lister les fichiers, filtrer ceux au format CSV, les télécharger et les stocker directement dans le conteneur bronze grâce à dbutils.fs.put() :

```
import requests
import os

# Configuration GitLab
GITLAB_API_URL = "https://gitlab.com/api/v4"
PROJECT_ID = "69289862" # l'ID du projet GitLab
ACCESS_TOKEN = "glpat-Xr-VaueKhxp9Z3dPhmHr" # token d'accès personnel GitLab

# Configuration Azure
AZURE_CONTAINER_PATH = "abfss://bronze@stockagedataengineering.dfs.core.windows.net/" # Chemin vers le conteneur Azure via DBFS

# Fonction pour récupérer la liste des fichiers dans un dépôt GitLab
def get_gitlab_files(project_id, access_token):
url = f"{GITLAB_API_URL}/projects/{project_id}/repository/tree"
headers = {"Private-Token": access_token}
params =
{ "ref": "main", `
"recursive": "true" # Pour récupérer tous les fichiers du projet
}
response = requests.get(url, headers=headers, params=params)
if response.status_code == 200:
return response.json()
else:
print(f"Erreur lors de la récupération des fichiers depuis GitLab: {response.status_code}")
```

```
return []

# télécharger un fichier spécifique depuis GitLab
def download_file_from_gitlab(file_path, project_id, access_token):
url = f"{GITLAB_API_URL}/projects/{project_id}/repository/files/{file_path}/raw"
headers = {"Private-Token": access_token}
params = {"ref": "main"} # Branche spécifique
response = requests.get(url, headers=headers, params=params)
if response.status_code == 200:
return response.content.decode('utf-8') # Convertir les bytes en str
else:
print(f"Erreur lors du téléchargement du fichier {file_path}: {response.status_code}")
return None

# sauvegarder le fichier dans Azure via DBFS
def save_file_to_azure(file_name, file_content):
# Stocker le fichier dans le conteneur Azure Blob via DBFS
file_path = os.path.join(AZURE_CONTAINER_PATH, file_name)
dbutils.fs.put(file_path, file_content, overwrite=True) # Sauvegarder avec dbutils
print(f"Fichier {file_name} sauvegardé dans Azure Blob Storage à {file_path}")

# Main: Récupérer les fichiers CSV depuis GitLab et les stocker dans Azure
def main():
files = get_gitlab_files(PROJECT_ID, ACCESS_TOKEN)
for file in files:
if file['name'].endswith('.csv'): # Filtrer pour ne récupérer que les fichiers CSV
print(f"Récupération du fichier CSV: {file['path']}")
file_content = download_file_from_gitlab(file['path'], PROJECT_ID, ACCESS_TOKEN)
if file_content:
save_file_to_azure(file['name'], file_content)

if _name_ == "_main_":
main()
```

L’utilisation de Databricks dans ce contexte apporte une grande flexibilité pour automatiser les étapes d'ingestion. Elle permet également un accès natif au Data Lake, et ce, de façon sécurisée via Azure Managed Identity et l’authentification par rôle IAM, comme montré dans les captures de configuration :





Ce traitement représente la première étape du pipeline ELT (Extract – Load – Transform), où les données restent encore non nettoyées, non transformées, mais stockées de manière organisée et exploitable pour les étapes ultérieures.

B. Traitement des données dans la couche Silver : nettoyage, enrichissement et standardisation :

Après avoir stocké les données brutes dans la couche bronze, la deuxième phase consiste à effectuer un premier niveau de traitement, de nettoyage et de structuration au sein de la couche silver. Cette étape est cruciale pour garantir l'homogénéité, la fiabilité et la lisibilité des données qui seront exploitées en aval.

La logique du traitement repose sur trois grandes opérations :

- **Détection et traitement des valeurs manquantes** : pour les colonnes numériques, les valeurs manquantes sont remplacées par la **médiane** afin d'éviter l'influence des extrêmes, tandis que pour les colonnes textuelles, le **mode (valeur la plus fréquente)** est utilisé. Ce choix assure la cohérence statistique sans introduire de biais fort.

```
#Remplirlesvaleursmanquan
tes for column, dtype in
df.dtypes:
ifdtypein["double","int","float","bigint"]:
median=df.approxQuantile(column,[0.5], 0.001)[0]
df=df.withColumn(column,when(col(column).isNull(),median).otherwise(col(column)
)) else:
mode_df =
df.groupBy(column).count().orderBy(F.desc("count")).first() if
```

- **Dé-duplication** des enregistrements via df.dropDuplicates(), une étape essentielle pour éviter les doublons dans les futurs rapports ou agrégations.

- **Standardisation linguistique** : les noms des fichiers et des entités métiers sont traduits de l'anglais vers le français, à l'aide d'un dictionnaire personnalisé Python. Cela facilite la compréhension pour les utilisateurs finaux non anglophones, tout en préparant les noms pour une structuration en **modèle en étoile** dans la couche gold.

```
#Dictionnairedesnomstraduits
translations = {
"Allergy": "Allergie",
"Date": "Date",
"Disease":"Maladie",
"Doctor": "Medecin",
"Patient": "Patient",
"Room": "Chambre",
"Treatment": "Traitement",
"TreatmentCost": "CoutTraitement",
"Cost": "Cout",
"Encounter": "Consultation","Vitals":
"SignesVitaux",
"Patient_Allergy":"Patient_Allergie",
"BridgeEncounterDoctor":"Docteur_consultation"
```

Les fichiers nettoyés et enrichis sont enregistrés de façon temporaire, puis renommés et déplacés dans le conteneur Silver avec un format unique .csv, comme illustré dans le bloc suivant :

```
# Écriture dans un dossier temporaire
tmp_output_path = silver_path + "_tmp_" + translated_name +
"/" (df.coalesce(1)
.write
.mode("overwrite")
.option("header", "true")
.csv(tmp_output_path))

# Trouver le fichier part-xxxxx.csv
tmp_files = dbutils.fs.ls(tmp_output_path)
part_file = [f.path for f in tmp_files if f.name.startswith("part-")][0]

# Définir le chemin final dans Silver
final_output_path = silver_path + final_filename

# Déplacer et renommer
dbutils.fs.mv(part_file, final_output_path)
```

Enfin, les répertoires temporaires sont systématiquement supprimés, assurant une bonne hygiène de l'environnement cloud et un contrôle des coûts :

```
# Supprimer les fichiers temporaires
dbutils.fs.rm(tmp_output_path, recurse=True)

print(" Tous les fichiers transformés et enregistrés avec noms traduits en français.")
```

Ce notebook silver illustre l'importance de la couche Silver comme pivot de transformation, où les données passent d'un état brut à un état propre, standardisé et prêt à être modélisé dans une base analytique. Ce processus est au cœur de toute architecture Lakehouse, en garantissant à la fois la scalabilité, la traçabilité et la qualité de la donnée.

C. Mise en place de la couche Gold : agrégation, enrichissement et préparation analytique :

La couche Gold de notre architecture Data Lake représente l'étape finale de transformation, dans laquelle les données nettoyées et standardisées issues de la couche Silver sont consolidées, agrégées et enrichies afin d'alimenter des modèles analytiques ou des outils de visualisation décisionnelle.

Dans ce notebook Databricks, plusieurs processus critiques sont mis en œuvre. Tout d'abord, des fichiers essentiels tels que Date.csv et Consultation.csv sont reformatés. Par exemple, les dates sont converties dans un format uniforme dd/MM/yyyy à l'aide des fonctions Spark to_timestamp et date_format, garantissant une homogénéité dans les futures analyses

```
df_date = df_date.withColumn("Date", date_format(to_timestamp("Date"), "dd/MM/yyyy"))
df_consult = (df_consult
.withColumn("CheckinDate", date_format(to_timestamp("CheckinDate"), "dd/MM/yyyy"))
.withColumn("CheckoutDate", date_format(to_timestamp("CheckoutDate"), "dd/MM/yyyy")))
```

Ensuite, nous avons appliqué plusieurs agrégations sur des dimensions clés du fichier Patient.csv, pour produire des statistiques essentielles : répartition par sexe, par tranche d'âge, et par nationalité. Cela permet de dresser un profil sociodémographique des patients. Les données ont été regroupées, puis stockées dans des fichiers .csv dans le répertoire agregations/ du conteneur Gold, comme illustré ici :

```
from pyspark.sql.functions import year, when, col
from pyspark.sql import functions as F

# Charger le fichier Patient.csv
df_patient = spark.read.option("header", "true").csv(silver_adls + "Patient.csv")

#  Répartition par sexe
agg_sexe = df_patient.groupBy("Gender").count()

#  Répartition par tranche d'âge
df_patient = df_patient.withColumn("AnneeNaissance", year("Birth Date").cast("int"))
df_patient = df_patient.withColumn("TrancheAge",
when(col("AnneeNaissance") >= 2006, "0-18")
.when(col("AnneeNaissance") >= 1989, "19-35")
.when(col("AnneeNaissance") >= 1964, "36-60")
.otherwise("60+"))
agg_age = df_patient.groupBy("TrancheAge").count()

#  Répartition géographique (nationalité)

agg_nationalite = df_patient.groupBy("Nationality").count().orderBy("count",
ascending=False)

# Chemin vers le dossier d'agrégations dans le container Gold
agg_path = gold_adls + "agregations/"

# === Sauvegarder les résultats un par un ===
def save_aggregation(df, name):
```

```
tmp_path = agg_path + "_tmp_" + name + "/"
final_path = agg_path + name +
".csv" (df.coalesce(1)
.write
.mode("overwrite")
.option("header", "true")
.csv(tmp_path))
part_file = [f.path for f in dbutils.fs.ls(tmp_path) if f.name.startswith("part-")][0]
dbutils.fs.mv(part_file, final_path)
dbutils.fs.rm(tmp_path, recurse=True)

# Sauvegardes
save_aggregation(agg_sexe, "repartition_par_sexe")
save_aggregation(agg_age, "repartition_par_tranche_age")
save_aggregation(agg_nationalite, "repartition_par_nationalite")
```

Une étape analytique plus poussée a été mise en place par jointures successives entre les tables Patient, Consultation, Cout et Traitement. Cela nous a permis de construire un DataFrame riche contenant l'historique de chaque patient, les coûts associés aux soins reçus et les détails des traitements effectués. Grâce à cela, plusieurs KPIs financiers ont été extraits :

- Coût total par patient
- Coût moyen par consultation
- Coût total par traitement

Ces indicateurs ont été calculés avec groupBy().agg() et sauvegardés dans la même logique de fichiers .csv optimisés pour la restitution ou l'analyse en BI :

```
from pyspark.sql.functions import year, when, col
from pyspark.sql import functions as F

#  Coût total par patient

agg_couttotal =
df_jointure_3.groupBy("Patient_ID").agg(F.sum("CostAmount").alias("Montant"))

#  Coût moyen par consultation
agg_coutmoyerparconsultation =
df_jointure_3.groupBy("Encounter_ID").agg(F.avg("CostAmount").alias("CoutMoyen"))

# Coût total des traitements
```

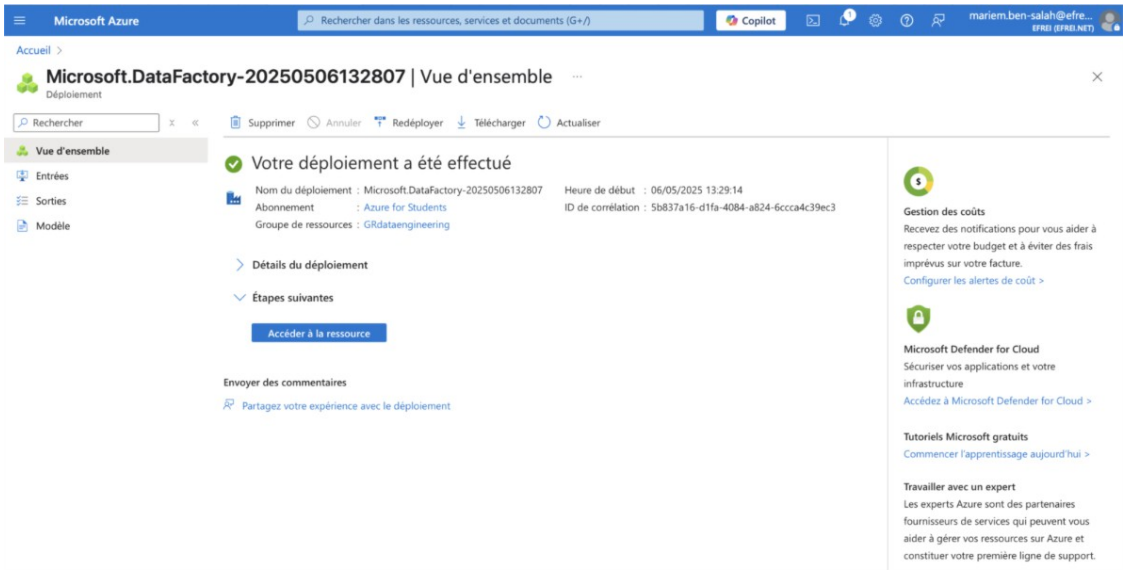
```
from pyspark.sql import functions as F

df_jointure_3 =
df_jointure_3.withColumn( "Cout_Total_Par_Enc
ontre", col("Drug_Cost").cast("double") +
col("Surgery_Cost").cast("double") +
col("Post_Surgery_Care_Cost").cast("double") +
col("Education_Rehab_Cost").cast("double")
)

agg_couttotaltraitement = df_jointure_3.groupBy("Treatment_ID") \
.agg(F.sum("Cout_Total_Par_Encontre").alias("CoutTotal")) \
# Chemin vers le dossier d'agrégations dans le container Gold
agg_path = gold_adls + "agregations/"

# === Sauvegarder les résultats un par un ===
def save_aggregation(df, name):
tmp_path = agg_path + "_tmp_" + name + "/"
final_path = agg_path + name +
".csv" (df.coalesce(1)
.write
.mode("overwrite")
.option("header", "true")
.csv(tmp_path))
part_file = [f.path for f in dbutils.fs.ls(tmp_path) if f.name.startswith("part-")][0]
dbutils.fs.mv(part_file, final_path)
dbutils.fs.rm(tmp_path, recurse=True)

# Sauvegardes
save_aggregation(agg_couttotal, " Coût total par patient")
save_aggregation(agg_coutmoyerparconsultation, "Coût moyen par consultation")
save_aggregation(agg_couttotaltraitement, "Coût total des traitements")
```

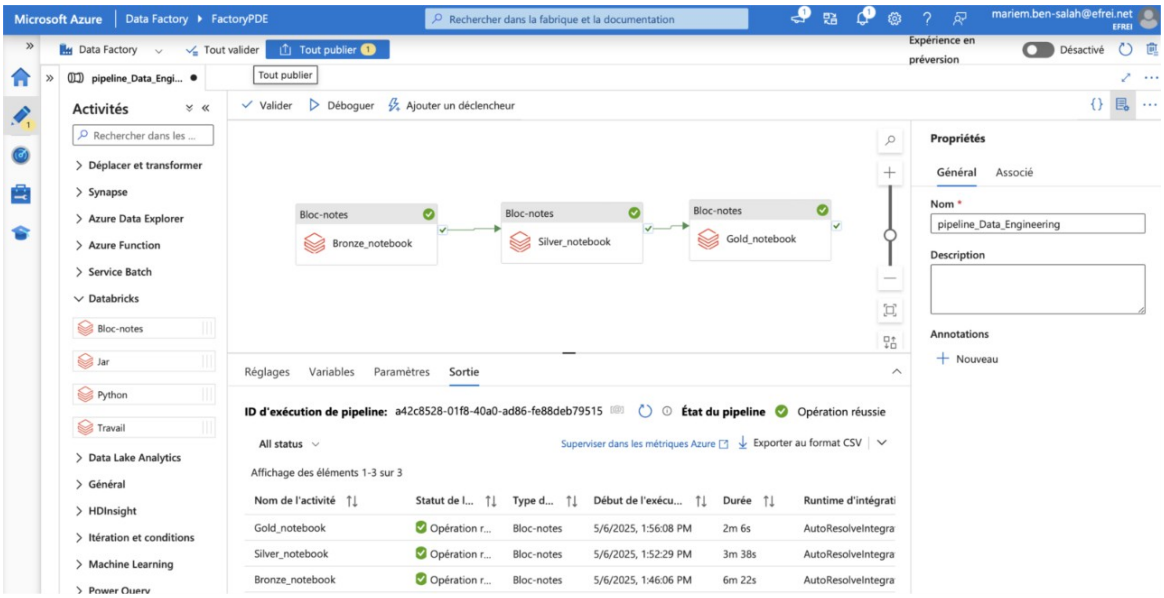
Enfin, tous les autres fichiers présents dans Silver ont été recopiés dans Gold (à l'exception de ceux déjà traités comme Consultation.csv et Date.csv), assurant la complétude des données pour la visualisation finale ou l'analyse exploratoire avancée.

Cette étape marque donc l'aboutissement d'un pipeline cloud-native entièrement industrialisé, depuis le sourcing jusqu'à la livraison de jeux de données exploitables, répondant aux meilleures pratiques du modèle Lakehouse. Elle met aussi en lumière les capacités analytiques offertes par Azure Databricks pour un traitement évolutif et distribué des données massives.

3. Orchestration avec Azure Data Factory :

Pour automatiser l'ensemble de ce pipeline, nous avons utilisé Azure Data Factory. Un pipeline unique nommé pipeline_Data_Engineering a été conçu pour orchestrer l'exécution séquentielle des trois notebooks (bronze, silver, gold), en respectant une logique conditionnelle basée sur le succès de chaque étape. Ce pipeline est associé à un déclencheur mensuel, afin de relancer automatiquement le traitement à une fréquence définie, garantissant ainsi la fraîcheur des données dans les conteneurs.L'authentification est assurée via un service lié (Linked Service) avec Databricks utilisant un token sécurisé. Cette approche garantit l'autonomie du processus ETL et son exécution régulière sans intervention manuelle.





Modifier le déclencheur

Nom *

Declencheur_mensuel

Description

Type *

ScheduleTrigger

Date de début *

5/6/2025, 12:03:00 PM

Fuseau horaire *

Brussels, Copenhagen, Madrid, Paris (UTC+2)

Ce fuseau horaire observe l'heure d'été. Le déclencheur se règle automatiquement avec une heure de différence.

Périodicité *

Chaque 15 Mois

Option de récurrence avancée

(•) Jours du mois ( ) Jours de la semaine

Sélectionner un ou plusieurs jours du mois pour l'exécution

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21

OK Annuler

4. Modélisation en étoile et exploitation des données dans Azure Synapse :

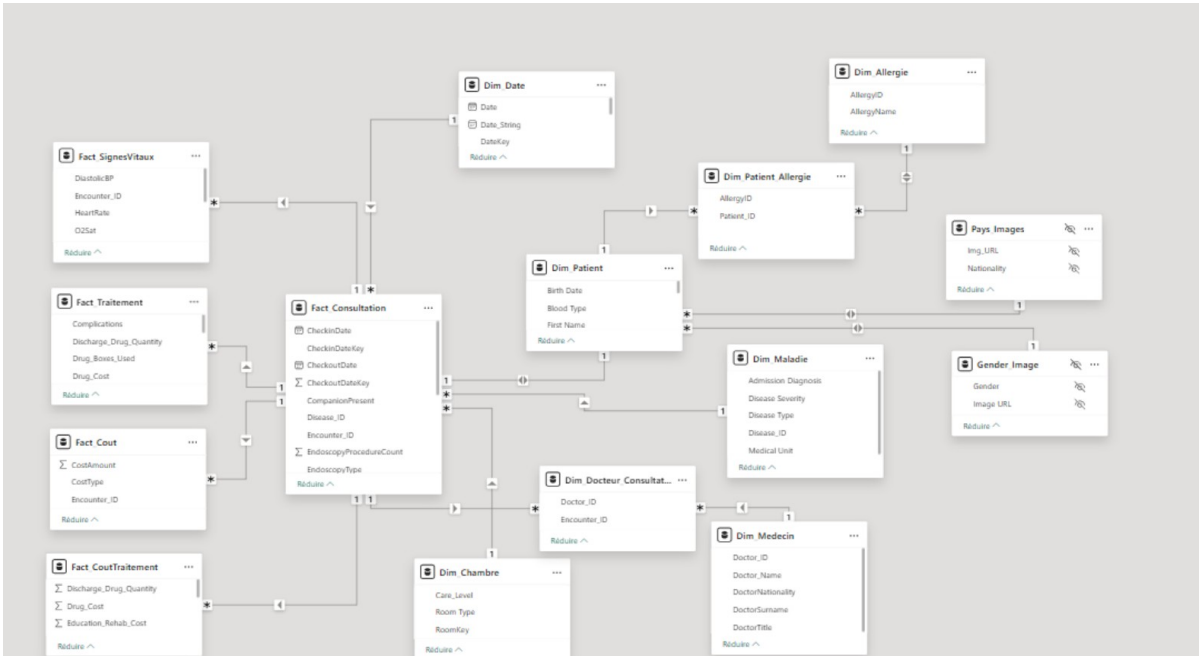
Une fois les données transformées et enrichies, elles sont modélisées dans Azure Synapse Analytics. Nous avons créé un espace de travail Synapse lié à notre Data Lake, puis utilisé la fonctionnalité "Créer une table externe à partir d'un lac de données" pour définir un modèle en étoile composé de :

Tables de dimensions

- Dim_Date
- Dim_Patient
- Dim_Allergie
- Dim_Medecin
- Dim_Chambre
- Dim_Maladie

Tables de faits

- Fact_Consultation
- Fact_Traitement
- Fact_SignesVitaux
- Fact_Cout
- Fact_CoutTraitement

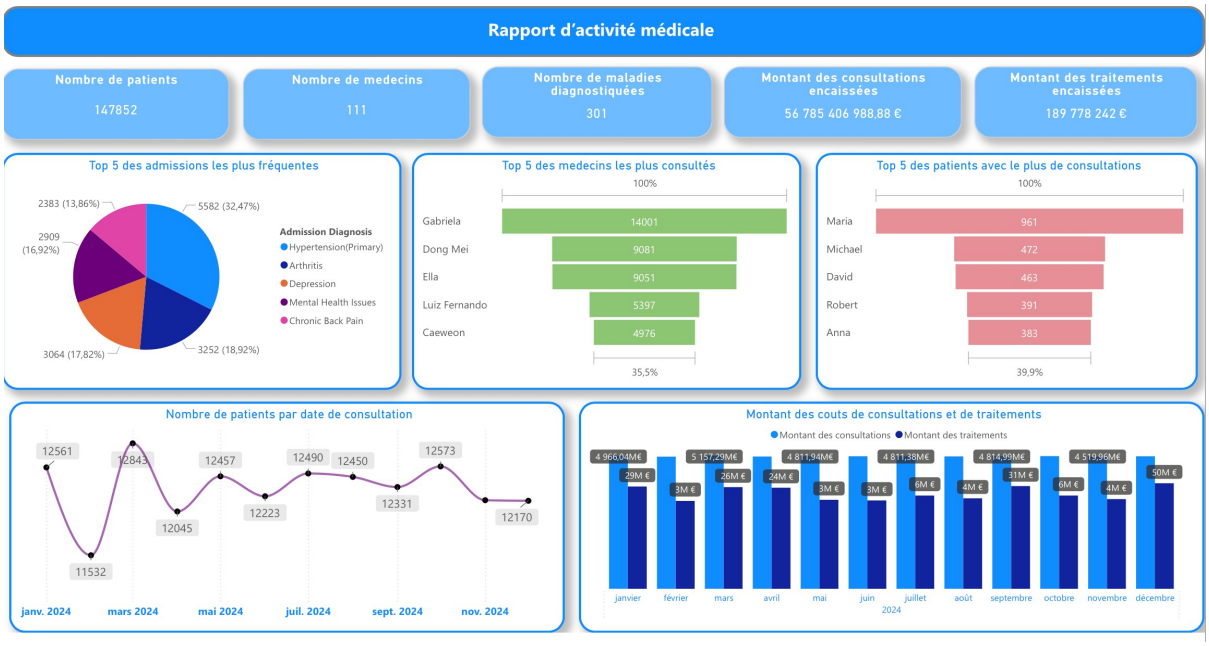


Cette modélisation relationnelle permet de faciliter les requêtes analytiques sur les grands volumes de données.

5. Visualisation avec Power BI :

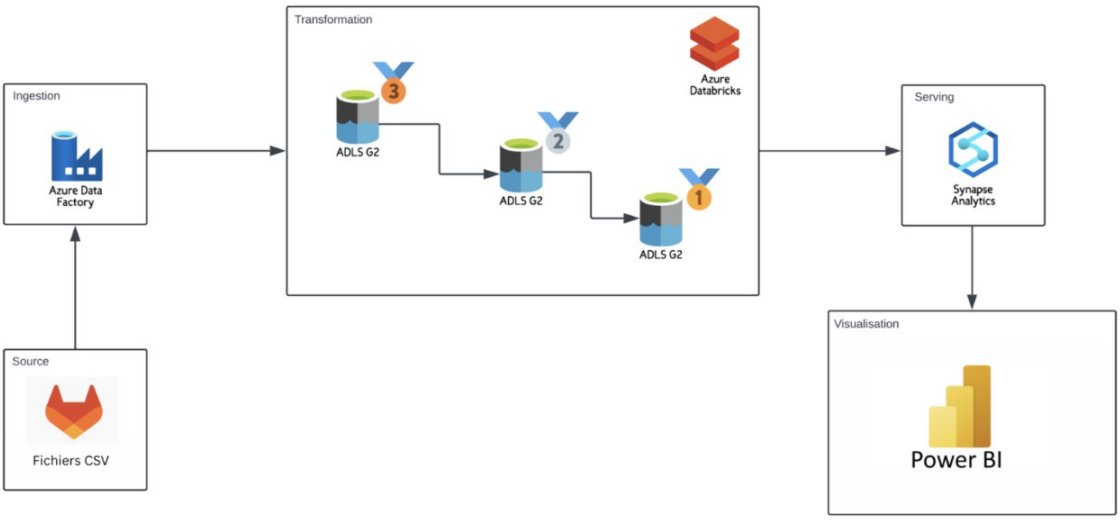
Pour la visualisation finale, Power BI a été connecté à notre point de terminaison SQL Serverless Synapse. Cette méthode permet d'exploiter directement les tables externes stockées dans le Data Lake, sans duplication ni importation locale. Les dashboards créés permettent de suivre les indicateurs extraits dans le conteneur Gold (par exemple : répartition des patients par sexe, nationalité, tranche d'âge, coûts, etc.)

Nous avons travaillé sur deux rapports présentant une vue globale et détaillée de l'activité médicale à l'échelle institutionnelle ainsi qu'individuelle. Le premier tableau de bord fournit une analyse statistique complète des performances médicales en 2024, incluant le nombre de patients (147 852), médecins (111), maladies diagnostiquées (301), et les montants encaissés pour consultations et traitements. Il utilise des graphiques variés : un camembert pour illustrer les cinq diagnostics d'admission les plus fréquents, des barres horizontales pour les médecins et patients les plus actifs, une courbe linéaire pour le nombre mensuel de patients, et un histogramme double pour comparer les coûts mensuels des consultations et traitements. Le deuxième tableau de bord se concentre sur le profil individuel de la patiente Barbara Smith, présentant son identité, ses allergies (via un camembert), ses dépenses médicales (via une courbe d'évolution), et son historique de diagnostics (représenté sur une frise chronologique). Ensemble, ces visualisations permettent une lecture claire et efficace de l'activité médicale à différents niveaux.



6. Architecture globale du projet :

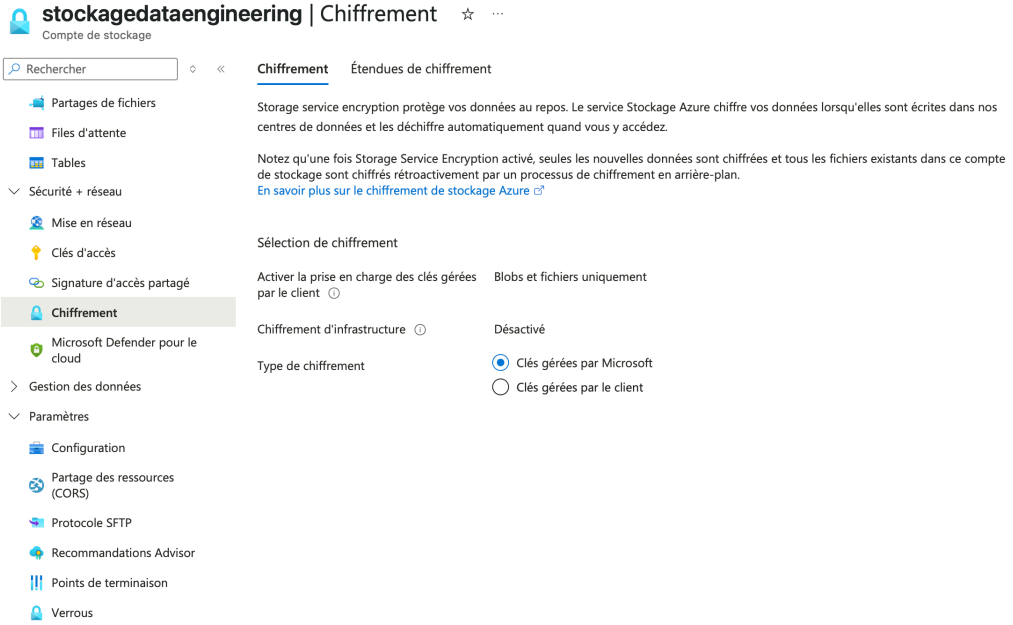
L'ensemble de l'architecture mise en place est illustré dans le schéma ci-dessous :



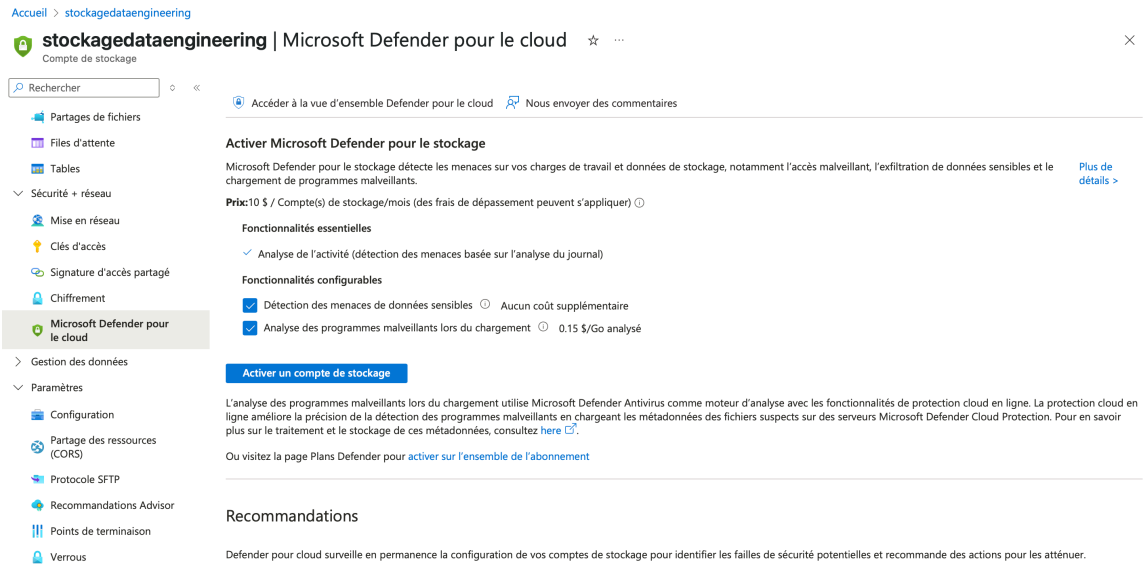
On y observe le flux complet, depuis la source GitLab, l'ingestion via Azure Data Factory, la transformation progressive dans les conteneurs bronze, silver et gold grâce à Azure Databricks, la modélisation via Synapse Analytics, jusqu'à la visualisation interactive dans Power BI. Cette infrastructure cloud native respecte les bonnes pratiques de découplage, de scalabilité et de gouvernance des données, tout en assurant la sécurité des accès grâce à l'implémentation des identités managées et des rôles IAM sur le stockage.

7. **Sécurité, performance et gouvernance des données dans le cloud :**

- Chiffrement et sécurité : les données sont chiffrées au repos grâce au chiffrement automatique AES-256 géré par Microsoft. Les flux entre les services (Databricks, ADLS, ADF, Synapse) sont chiffrés en transit via HTTPS et OAuth2. Pour l'accès aux données, nous avons défini une identité managée et attribué le rôle Storage Blob Data Contributor uniquement aux services autorisés



- Sécurité renforcée via Defender for Storage : fonction activée pour : une analyse de l'activité basée sur les journaux, détection des menaces de données sensibles, analyse des programmes malveillants au chargement et protection cloud assurée par Microsoft Defender Antivirus.



- Journalisation et supervision : les logs d'exécution de Databricks et ADF sont visibles dans Azure Monitor. Les erreurs, durées et temps de réponse sont mesurés en continu. Cela nous permet de monitorer les performances, d'optimiser le coût et de garantir la résilience du pipeline.

- Réplication et disponibilité : nous avons activé la redondance GRS (Geo-Redundant Storage) sur notre compte ADLS. Cela garantit la haute disponibilité et la récupération en cas de panne régionale.

8. Conclusion :

Grâce à ce projet cloud mené sur Azure, nous avons mis en œuvre une architecture robuste et industrialisée de traitement de données en mode cloud-native. Du data lake aux visualisations BI, en passant par la gouvernance, l'orchestration et la modélisation, notre solution répond à l'ensemble des exigences du Bloc 3 avec une approche moderne, scalable et sécurisée.