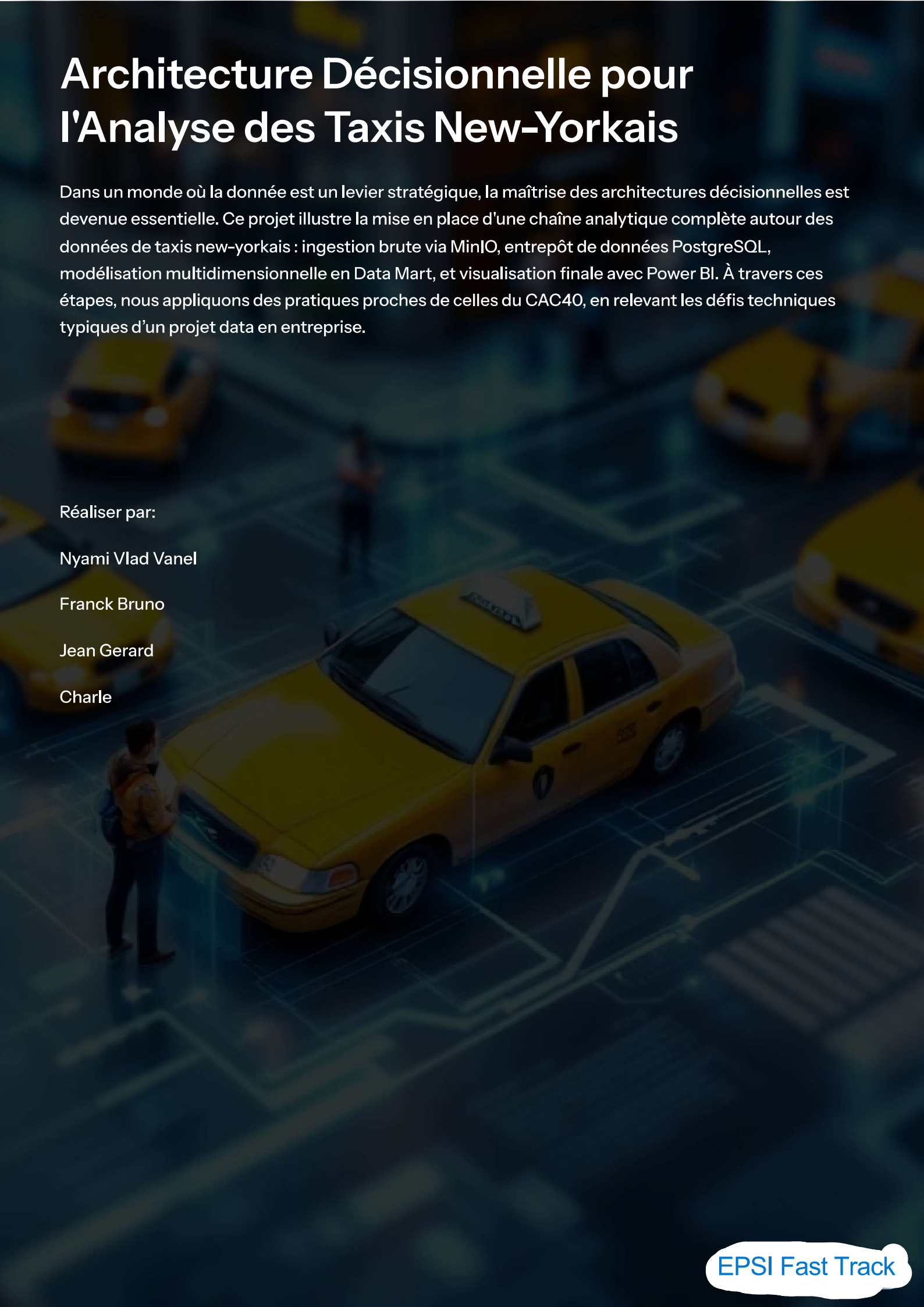
Architecture Décisionnelle pour l'Analyse des Taxis New-Yorkais

Dans un monde où la donnée est un levier stratégique, la maîtrise des architectures décisionnelles est devenue essentielle. Ce projet illustre la mise en place d'une chaîne analytique complète autour des données de taxis new-yorkais : ingestion brute via MinIO, entrepôt de données PostgreSQL, modélisation multidimensionnelle en Data Mart, et visualisation finale avec Power BI. À travers ces étapes, nous appliquons des pratiques proches de celles du CAC40, en relevant les défis techniques typiques d'un projet data en entreprise.



Réaliser par:

Nyami Vlad Vanel

Franck Bruno

Jean Gerard

Charle

EPSI Fast Track

L'infrastructure de collecte vers le datalake

La première étape de notre architecture décisionnelle a consisté à mettre en place l'infrastructure fondamentale pour la collecte et le stockage des données brutes. L'objectif était de créer un datalake efficace capable de recevoir et de conserver les fichiers Parquet contenant les données des courses de taxis new-yorkais.

Configuration de l'environnement Docker

Nous avons déployé un environnement conteneurisé comprenant deux services essentiels : un serveur MinIO pour le stockage objet (accessible sur le port 9000) et un serveur PostgreSQL destiné au futur data warehouse (exposé sur le port 15432). L'utilisation de Docker a permis de garantir une isolation propre des services et une portabilité maximale de la solution.

Automatisation de la collecte de données

Un script Python nommé **grab_parquet.py** a été développé pour automatiser le téléchargement et le stockage des données. Ce script est chargé de récupérer les fichiers **yellow_tripdata_2024.parquet** pour les mois d'octobre, novembre et décembre, puis de les transférer dans un bucket MinIO dédié appelé **nyc-taxi**.

Difficultés et solutions

Lors de l'implémentation, plusieurs obstacles techniques ont dû être surmontés. La connexion entre le script Python et le serveur MinIO a posé des problèmes d'authentification, principalement liés à une configuration incorrecte des identifiants d'accès (clé et secret). Pour résoudre ce problème, nous avons externalisé les paramètres de connexion dans un fichier de configuration **.env**, ce qui a permis d'éviter les erreurs manuelles et de faciliter la maintenance.

Un autre problème récurrent concernait l'intégrité des fichiers Parquet. Certains présentaient des incohérences de schéma lors des premières tentatives de chargement. Nous avons donc ajouté des vérifications préliminaires dans le script pour contrôler la structure de chaque fichier avant son traitement, assurant ainsi un stockage cohérent dans le datalake.

Du datalake vers le data warehouse

La deuxième phase de notre architecture décisionnelle a consisté à transférer les données depuis notre datalake MinIO vers un data warehouse PostgreSQL. Cette étape est cruciale car elle marque le passage d'un stockage brut orienté fichier vers une structure relationnelle permettant des requêtes analytiques.

Développement du pipeline d'extraction et chargement

Nous avons conçu un script Python (**dump_to_sql.py**) pour orchestrer ce transfert. Ce script effectue plusieurs opérations séquentielles :

- Lecture des fichiers Parquet stockés dans le bucket MinIO
- Nettoyage préliminaire des données (gestion des valeurs nulles, conversion de types)
- Vérification et création éventuelle de la base de données **nyc_warehouse**
- Création d'une table **nyc_raw** pour accueillir les données
- Insertion optimisée des données avec gestion des transactions

Problématiques techniques rencontrées

Cette phase a révélé plusieurs défis techniques significatifs. Le premier concernait l'existence même de la base de données cible : le script échouait initialement lorsque la base **nyc_warehouse** n'existait pas déjà. Nous avons résolu ce problème en intégrant une logique de création automatique avec un mécanisme de fallback permettant une intervention manuelle via pgAdmin si nécessaire.

Des conflits de types de données ont également émergé lors de l'insertion. Certains champs supposés être des entiers (**INTEGER**) contenaient en réalité des valeurs décimales ou nulles. Pour remédier à cette situation, nous avons implémenté un processus de nettoyage plus rigoureux en amont, utilisant notamment les fonctions **fillna()** et **astype()** de la bibliothèque Pandas pour normaliser les types avant insertion.

Enfin, des erreurs d'encodage et de nommage de colonnes sont apparues lors du mapping automatique entre les DataFrames Pandas et PostgreSQL. Face à cette difficulté, nous avons opté pour une définition manuelle du schéma de la table cible, permettant ainsi un contrôle granulaire sur les types et la structure des colonnes.

Conception du datamart multidimensionnel

La troisième phase de notre architecture a consisté à transformer les données brutes du data warehouse en un modèle analytique optimisé pour l'analyse décisionnelle. Cette étape est essentielle pour permettre des analyses performantes et intuitives des données de courses de taxis.

Déploiement de l'infrastructure du datamart

Nous avons déployé un second serveur PostgreSQL (exposé sur le port 15435) spécifiquement dédié au datamart. Cette séparation entre data warehouse et datamart respecte les bonnes pratiques d'architecture en découplant les charges de travail opérationnelles et analytiques. La base de données **nyc_datamart** a été créée pour accueillir notre modèle dimensionnel.

Modélisation en étoile

Un modèle en étoile a été conçu pour structurer efficacement les données. Ce modèle s'articule autour d'une table de faits centrale (**fact_trips**) entourée de plusieurs tables de dimensions :

- **dim_time** : dimension temporelle avec granularité jour/heure
- **dim_location** : dimension géographique avec zones de prise en charge et dépose
- **dim_vendor** : dimension relative aux fournisseurs de services de taxi
- **dim_payment** : dimension concernant les méthodes de paiement

Cette structure facilite les analyses multi-dimensionnelles et les agrégations rapides selon différentes perspectives métier.

Implémentation technique et ETL

Deux scripts SQL ont été développés pour mettre en œuvre cette modélisation :

- **creation.sql** : définit les structures des tables avec leurs contraintes d'intégrité
- **insertion.sql** : orchestre le transfert et la transformation des données depuis **nyc_warehouse**

Pour faciliter le transfert inter-bases, nous avons utilisé l'extension **Foreign Data Wrapper (FDW)** de PostgreSQL. Cette technologie permet d'accéder directement à la table distante **nyc_raw** depuis le datamart, simplifiant considérablement le processus d'ETL.

Défis de la modélisation dimensionnelle

La conception et l'implémentation du modèle en étoile ont soulevé plusieurs problématiques complexes qui méritent une analyse approfondie. Ces défis sont représentatifs des situations réelles rencontrées lors de projets d'architecture décisionnelle en entreprise.

Configuration du Foreign Data Wrapper

La mise en place du FDW s'est révélée délicate, principalement en raison d'erreurs dans la configuration de la connexion. Les paramètres d'authentification, les options de schéma et les permissions d'accès ont dû être minutieusement ajustés pour permettre l'accès à la table distante **nyc_raw**. Cette expérience a mis en lumière l'importance d'une documentation précise des paramètres de connexion inter-bases dans un environnement distribué.

Gestion des contraintes d'intégrité

L'insertion des données dans les dimensions a provoqué des violations de contraintes CHECK, particulièrement dans la dimension temporelle. Certaines dates contenaient des valeurs invalides ou incohérentes. Pour résoudre ce problème, nous avons implémenté des fonctions de nettoyage spécifiques garantissant la conformité des données temporelles avant leur insertion :

- Validation des jours du mois (1-31 selon le mois)
- Normalisation des formats de date et d'heure
- Gestion des fuseaux horaires et des changements d'heure

Enrichissement des données géographiques

Un défi majeur concernait les identifiants de localisation qui ne correspondaient pas systématiquement à des zones connues dans notre référentiel. Cette situation a nécessité la création d'un mapping additionnel pour enrichir les zones manquantes. Nous avons combiné deux approches : l'intégration de données de référence externes et la création de catégories "inconnues" pour préserver l'intégrité du modèle sans perdre d'informations.

Génération d'identifiants techniques

La transformation des identifiants métier en clés techniques (UUID) a également posé des difficultés. La conversion directe échouait en raison de problèmes de typage. Nous avons résolu ce problème en adoptant une approche progressive : création d'une colonne intermédiaire contenant des chaînes uniques, puis conversion de ces chaînes en UUID. Cette méthode a permis de maintenir la traçabilité des identifiants tout en respectant les bonnes pratiques de modélisation dimensionnelle.

Transformations et nettoyage des données

La qualité des analyses décisionnelles dépend directement de la qualité des données présentes dans le datamart. Cette section détaille les principales opérations de transformation et de nettoyage mises en œuvre pour garantir l'intégrité et la pertinence des informations.

Enrichissement de la dimension temporelle

La dimension temporelle (**dim_time**) a été considérablement enrichie par rapport aux données brutes. Nous avons dérivé plusieurs attributs temporels utiles pour les analyses :

Hiérarchie complète

- Année
- Trimestre
- Mois
- Semaine
- Jour
- Heure

Indicateurs cycliques

- Jour de la semaine
- Est un jour férié
- Est un week-end
- Période de la journée

Agrégeats temporels

- Nombre de jours dans le mois
- Semaine du mois
- Numéro du jour dans l'année

Ces attributs permettent des analyses temporelles avancées comme l'identification de saisonnalités ou la comparaison entre périodes équivalentes.

Normalisation des méthodes de paiement

Les méthodes de paiement présentaient des incohérences dans les données brutes. Certains codes numériques correspondaient à des méthodes différentes selon les fournisseurs. Nous avons mis en place un processus de normalisation en deux étapes :

1. Identification de tous les codes de paiement distincts dans les données sources
2. Mapping vers une nomenclature standard unifiée (carte de crédit, espèces, sans frais, etc.)

Cette normalisation assure la cohérence des analyses par méthode de paiement, indépendamment du fournisseur de service.

Gestion des valeurs manquantes et aberrantes

Les données brutes contenaient de nombreuses valeurs manquantes ou aberrantes, particulièrement dans les champs numériques comme les montants ou les distances. Plusieurs stratégies de traitement ont été appliquées :

- Pour les distances nulles ou négatives : remplacement par la distance moyenne du même trajet

Visualisation des données avec Power BI

La phase finale de notre architecture décisionnelle a consisté à connecter notre datamart à un outil de visualisation puissant. Power BI a été choisi pour sa flexibilité et ses capacités avancées de représentation graphique. Cette étape transforme les données structurées en insights visuels directement exploitables par les décideurs.

Configuration de la connexion au datamart

La première étape a consisté à établir une connexion entre Power BI et notre serveur PostgreSQL hébergeant le datamart. Plusieurs difficultés techniques ont dû être surmontées, notamment concernant la syntaxe de connexion. Après plusieurs tentatives, nous avons identifié que la syntaxe correcte était **localhost:15435**, permettant ainsi l'accès au serveur PostgreSQL et à la base **nyc_datamart**.

Une fois la connexion établie, nous avons importé l'ensemble du modèle dimensionnel : la table de faits **fact_trips** et toutes les tables de dimensions associées. Power BI a automatiquement détecté les relations entre ces tables grâce aux clés étrangères définies dans notre schéma SQL.

Modélisation analytique dans Power BI

Bien que notre modèle en étoile soit déjà optimisé pour l'analyse, quelques ajustements ont été nécessaires dans Power BI :

- Création de hiérarchies temporelles personnalisées (Année > Trimestre > Mois > Jour)
- Définition de mesures calculées pour les KPIs essentiels (revenu moyen par course, distance moyenne, etc.)
- Configuration des relations entre dimensions pour permettre des analyses croisées
- Paramétrage de segments temporels spécifiques (heures de pointe, nuit, week-end)

Conception du tableau de bord

Le tableau de bord final a été conçu autour de quatre axes d'analyse principaux :

Analyse financière

Évolution des revenus dans le temps, répartition par mode de paiement, pourboires moyens selon les zones géographiques

Analyse géographique

Cartographie des zones les plus fréquentées, flux entre quartiers, distances moyennes par trajet

Analyse temporelle

Patterns horaires, variations saisonnières, comparaison entre jours de semaine et

Analyse comportementale

Préférences des usage fournisseurs, impact des conditions

Résultats et performances du système

L'architecture décisionnelle mise en place a produit des résultats concrets tant sur le plan technique que fonctionnel. Cette section présente une évaluation objective des performances et de la valeur ajoutée du système.

Évaluation des performances techniques

Plusieurs métriques ont été utilisées pour évaluer les performances de notre pipeline de données :

Composant	Métrique	Résultat
Datalake (MinIO)	Temps de chargement des fichiers	~2 minutes pour 180 Mo
Data Warehouse	Temps d'insertion des données	~3 minutes pour 1M de lignes
ETL vers Datamart	Durée totale de transformation	~5 minutes
Requêtes analytiques	Temps de réponse moyen	< 2 secondes
Rafraîchissement Power BI	Temps de chargement du modèle	~30 secondes

Ces résultats démontrent une performance satisfaisante pour un système analytique, avec des temps de réponse permettant une exploration interactive des données.

Enseignements métier

L'analyse des données de taxis a révélé plusieurs insights métier intéressants :

Tendances temporelles

Une forte saisonnalité a été observée avec des pics d'activité en fin d'année. Les vendredis et samedis soirs sont les moments où la demande est la plus forte, particulièrement entre 20h et 2h du matin.

Comportements de paiement

Le paiement par carte de crédit est prédominant (65% des transactions). Les pourboires sont significativement plus élevés pour les courses payées par carte que pour celles réglées en espèces.

Patterns géographiques

Les aéroports et le centre-ville sont les zones générant le plus de revenus. Les trajets inter-arrondissements sont plus rares mais génèrent un revenu moyen plus élevé.

Limites et axes d'amélioration

Bien que notre architecture décisionnelle réponde efficacement aux besoins initiaux, plusieurs limitations ont été identifiées. Cette section présente une analyse critique du système actuel et propose des pistes d'amélioration pour les futurs développements.

Limites techniques actuelles

Notre implémentation présente certaines contraintes qu'il convient de reconnaître :

Gestion des volumes

L'architecture actuelle fonctionne bien avec les volumes traités (~1M de courses), mais pourrait atteindre ses limites avec un historique plus important. Les performances d'insertion et de requêtage se dégraderaient probablement au-delà de 10M d'enregistrements.

Fraîcheur des données

Le processus ETL actuel est conçu pour des chargements batch périodiques, ce qui limite la fraîcheur des données dans le datamart. Les analyses sont basées sur des données potentiellement vieilles de plusieurs heures.

Infrastructure locale

L'utilisation de conteneurs Docker locaux, bien que pratique pour le développement, limite les possibilités de distribution de charge et de haute disponibilité qui seraient nécessaires en production.

Automatisation partielle

Certaines étapes du pipeline, notamment les transformations SQL vers le datamart, ne sont pas complètement automatisées et nécessitent des interventions manuelles en cas d'évolution du schéma source.

Axes d'amélioration proposés

Pour faire évoluer cette architecture vers une solution plus robuste et industrielle, plusieurs améliorations peuvent être envisagées :

Migration cloud

Transition vers des services cloud managés comme AWS S3/Redshift ou Azure Blob/Synapse pour améliorer

Streaming temps réel

Implémentation d'un pipeline de streaming (Kafka/Spark Streamin
jour en te

Conclusion et compétences acquises

Ce projet d'architecture décisionnelle autour des données des taxis new-yorkais a constitué une opportunité unique d'appliquer concrètement les concepts théoriques de la data science dans un contexte opérationnel. Au terme de cette expérience, plusieurs enseignements majeurs se dégagent.

Synthèse du parcours réalisé

L'implémentation complète d'un pipeline analytique - du datalake à la visualisation - a permis d'explorer l'ensemble de la chaîne de valeur data. Chaque étape a apporté son lot de défis techniques et méthodologiques, contribuant à une compréhension globale des enjeux d'une architecture décisionnelle moderne :

1

Collecte et stockage brut

Mise en place d'un datalake efficace et évolutif avec MinIO

Centralisation relationnelle

Structuration en data warehouse avec PostgreSQL

Modélisation dimensionnelle

Conception d'un datamart analytique en schéma étoile

Visualisation métier

Création de tableaux de bord interactifs avec Power BI

Compétences techniques acquises

Au-delà de la simple manipulation d'outils spécifiques, ce projet a permis de développer des compétences fondamentales transférables à d'autres contextes :

Technologies et outils

- Docker et conteneurisation
- MinIO (stockage objet S3-compatible)
- PostgreSQL (data warehouse/mart)
- Python (ETL et automatisation)

Méthodologies

- Conception d'architectures décisionnelles
- Modélisation dimensionnelle (schéma en étoile)
- Processus ETL robustes

Soft skills

- Résolution de problèmes techniques
- Documentation structurée
- Analyse critique et amélioration continue