**Rapport Général du Projet: Plateforme E-commerce**

Introduction

La mise en place d'une plateforme e-commerce pour la vente de produits électroniques est un projet stratégique visant à offrir une expérience utilisateur optimale tout en exploitant les données pour améliorer les ventes et réagir en temps réel aux actions des utilisateurs. Cette introduction expose le contexte du projet, ses objectifs et les grandes lignes des phases à venir.

Phase 1: Collecte des Données

Données Collectées

Les données collectées comprennent plusieurs champs qui capturent des informations essentielles sur les produits disponibles sur la plateforme e-commerce. Voici un aperçu des champs disponibles dans les données :

1. itemId: Identifiant unique du produit.

2. title: Titre du produit.

3. available: Indique si le produit est disponible (True/False).

4. catId: Identifiant de la catégorie du produit.

5. sales: Nombre de ventes du produit.

6. wishCount: Nombre de souhaits pour le produit.

7. itemUrl: URL du produit sur la plateforme.

8. images: Liste d'URLs d'images représentant le produit.

9. properties\_list: Liste de propriétés du produit (caractéristiques).

10. description\_images: Liste d'URLs d'images pour la description du produit.

11. sku\_quantity: Quantité en stock.

12. sku\_price: Prix de l'unité du produit.

13. sku\_promotionPrice: Prix promotionnel de l'unité du produit.

14. sku\_unit: Unité de mesure du produit (e.g., "piece").

15. sku\_isBulk: Indique si le produit est vendu en gros (True/False).

16. sku\_bulk\_quantity: Quantité en gros disponible.

17. sku\_bulk\_discount: Réduction pour l'achat en gros.

18. reviews\_count: Nombre total d'avis sur le produit.

19. reviews\_averageStarRate: Note moyenne du produit basée sur les avis.

20. shippingFrom: Pays d'expédition.

21. shippingTo: Pays de destination.

22. shippingOutDays: Jours d'expédition.

23. packageDetail: Détails sur l'emballage (poids, dimensions).

24. service\_title: Titre du service lié au produit.

25. service\_desc: Description du service lié au produit.

26. seller\_sellerId: Identifiant du vendeur.

27. seller\_storeTitle: Titre du magasin du vendeur.

28. seller\_storeId: Identifiant du magasin du vendeur.

29. seller\_companyId: Identifiant de la société du vendeur.

30. seller\_storeRating: Note du magasin du vendeur.

31. shipping\_id: Identifiant d'expédition.

32. service\_id: Identifiant de service.

Actions à Entreprendre

1. Nettoyage des Données:

- Vérifier la cohérence des données (valeurs manquantes, incohérences).

- Éliminer les doublons si nécessaire.

2. Exploration des Données:

- Analyser la distribution des ventes, des souhaits, et des avis.

- Explorer les catégories les plus populaires.

3. Étude des Services et Politiques:

- Examiner les services associés aux produits.

- Analyser les politiques de retour et les garanties.

4. Analyse des Vendeurs:

- Explorer la réputation des vendeurs et leur impact sur les ventes.

Phase 2: Mise en place de l'Infrastructure de Stockage

Objectif

L'objectif de cette phase était de mettre en place une infrastructure de stockage centralisée pour les données collectées afin de faciliter leur gestion et leur utilisation ultérieure.

Étapes de Mise en Place

1. Configuration du Client HDFS

Nous avons configuré un client HDFS pour établir une connexion avec le cluster Hadoop, garantissant une communication sécurisée avec la Data Lake.

2. Transfert des Fichiers Locaux vers la Data Lake HDFS

Les fichiers locaux, contenant les données collectées, ont été transférés vers la Data Lake HDFS. Cela assure le stockage sécurisé et efficace des données dans un environnement distribué.

3. Intégration des Données dans la Base de Données

Les données stockées dans la Data Lake ont été intégrées à une base de données relationnelle. Cette étape offre une structuration des données pour des analyses plus avancées et des requêtes SQL.

Phase 3: Transformation des Données

Objectif

L'objectif de cette phase était de transformer les données stockées dans la Data Lake en une structure adaptée pour l'analyse, en appliquant diverses modifications et conversions.

Étapes de Transformation

1. Chargement des Données depuis la Data Lake HDFS

Les données stockées dans les répertoires HDFS spécifiés ont été chargées dans des DataFrames à l'aide de la bibliothèque Pandas.

2. Nettoyage des Données

Plusieurs opérations de nettoyage ont été effectuées sur les données, notamment :

- Remplacement des valeurs nulles dans certaines colonnes par zéro.

- Conversion de certaines colonnes au format numérique.

3. Transformation des Colonnes de Prix

Les colonnes de prix ont été séparées en valeurs minimales et maximales pour permettre une meilleure analyse. Les valeurs ont également été converties en format numérique.

4. Traitement de la Colonne 'sales'

La colonne 'sales' a été convertie en format numérique en supprimant les caractères non numériques.

5. Extraction des Détails du Package

Les détails du package, stockés sous forme de dictionnaire dans une colonne, ont été extraits en colonnes distinctes telles que poids, longueur, hauteur et largeur.

6. Nettoyage des Colonnes de Description

Les colonnes 'service\_title' et 'service\_desc' ont été nettoyées en supprimant les caractères indésirables.

7. Suppression de Colonnes Inutiles

Certaines colonnes jugées inutiles pour l'analyse ont été supprimées.

8. Création des Dimensions et de la Table de Faits

Les dimensions (ProductDim, ShippingDim, ServiceDim, SellerDim) ont été créées à partir des données transformées, et une table de faits (SalesFact) a été générée.

9. Chargement des Données Transformées dans la Base de Données

Les données transformées ont été chargées dans la base de données SQL Server pour permettre une analyse plus avancée.

Modélisation du Data Warehouse

Tables de Dimensions :

1. ProductDim:

- itemId (Clé primaire) : Identifiant unique du produit.

- title : Titre du produit.

- available : Indique si le produit est disponible.

- sku\_isBulk : Indique si le produit est en vrac.

2. ShippingDim:

- shipping\_id (Clé primaire) : Identifiant unique de l'expédition.

- shippingFrom : Lieu d'expédition.

- shippingTo : Lieu de destination.

3. ServiceDim:

- service\_id (Clé primaire) : Identifiant unique du service.

- service\_title : Titre du service.

- service\_desc : Description du service.

4. SellerDim:

- seller\_sellerId (Clé primaire) : Identifiant unique du vendeur.

- seller\_storeTitle : Titre du magasin du vendeur.

Table de Faits :

5. SalesFact:

- Id (Clé primaire) : Identifiant unique de la vente.

- itemId (Clé étrangère) : Référence à ProductDim.

- shipping\_id (Clé étrangère) : Référence à ShippingDim.

- service\_id (Clé étrangère) : Référence à ServiceDim.

- seller\_sellerId (Clé étrangère) : Référence à SellerDim.

- sales : Montant des ventes.

- sku\_quantity : Quantité de produits en stock.

- wishCount : Nombre de souhaits pour le produit.

- reviews\_count : Nombre total d'avis.

- reviews\_averageStarRate : Note moyenne des avis.

- ... (d'autres mesures spécifiques aux ventes)

Cette modélisation utilise des tables de dimensions pour stocker des informations détaillées sur les produits, les expéditions, les services et les vendeurs. La table de faits (SalesFact) contient des mesures liées aux ventes et des clés étrangères pour établir des relations avec les tables de dimensions.

Étapes de visualisation

Automatisation avec Apache Airflow

Nous avons mis en place une automatisation des tâches à l'aide d'Apache Airflow, un orchestrateur de flux de travail largement utilisé. Cette automatisation est encapsulée dans un Directed Acyclic Graph (DAG) nommé "BI". Voici une description des étapes automatisées dans ce DAG :

Paramètres par Défaut

- Propriétaire : airflow

- Dépendances sur le passé : Non activé

- Date de début : 30 janvier 2024

- Nombre de tentatives : 1

- Délai de réessai : 5 minutes

DAG "BI"

- Description : DAG pour les processus Business Intelligence.

Tâches

1. Importation CSV vers HDFS

- ID de tâche : import\_csv\_to\_hdfs

- Fonction Python : Exécute le script Python d'importation de CSV vers HDFS.

- Script : /opt/airflow/docker-hadoop/workspace/import\_csv\_to\_hdfs.py

2. Importation DB vers HDFS

- ID de tâche : import\_db\_to\_hdfs

- Fonction Python : Exécute le script Python d'importation de la base de données vers HDFS.

- Script : /opt/airflow/docker-hadoop/workspace/import\_db\_to\_hdfs.py

3. Exportation DB vers HDFS

- ID de tâche : export\_db\_hdfs

- Fonction Python : Exécute le script Python d'exportation de la base de données vers HDFS.

- Script : /opt/airflow/docker-hadoop/workspace/export\_db\_hdfs.py

Planification

- Interval de Planification : Tous les jours

Cette automatisation quotidienne garantit l'exécution régulière des tâches nécessaires pour alimenter notre Data Warehouse, en créant ainsi un flux de travail robuste et cohérent pour nos processus BI.

Intégration des Données en Temps Réel

Outre l'automatisation quotidienne, il peut être essentiel d'intégrer des données en temps réel pour garantir des informations à jour. Pour cela, nous avons mis en place une solution de streaming en utilisant des technologies adaptées à cette exigence. Voici comment nous réalisons l'intégration des données en temps réel :

Apache Kafka pour le Streaming

Nous utilisons Apache Kafka comme plateforme de streaming pour ingérer et traiter les données en temps réel. Kafka permet une ingestion efficace des données, garantissant la faible latence et la scalabilité.

Producteur Kafka

Un producteur Kafka est configuré pour surveiller les modifications de données dans la source, telles que la base de données transactionnelle. Lorsqu'un événement survient, il est publié sur le topic Kafka approprié.

Consommateur Kafka

Un consommateur Kafka est configuré pour écouter les événements du topic correspondant. Il traite ces événements en temps réel et effectue des opérations appropriées, comme la mise à jour des données dans notre entrepôt de données.

Avantages de l'Intégration en Temps Réel

1. Actualisation Instantanée : Les données sont traitées dès leur arrivée, garantissant des informations à jour en temps réel.

2. Réactivité Améliorée : Les changements dans la source sont reflétés rapidement dans notre entrepôt de données, permettant des réponses rapides aux événements métier.

3. Analyse en Continu : Possibilité de réaliser des analyses continues et de détecter les tendances en temps réel.

API pour l'Accès aux Données

Vous avez mis en place une API Flask pour permettre l'accès aux données de manière simple et structurée. Cela peut être utile pour intégrer vos données dans d'autres applications, services ou interfaces utilisateur. Voici quelques points que vous pouvez inclure dans votre rapport pour expliquer l'API :

API Flask pour l'Accès aux Données

Objectif

Fournir un moyen simple d'accéder aux données stockées dans notre entrepôt de données en utilisant une interface API.

Points Principaux

1. Endpoint - `/get\_next\_rows` :

- Cet endpoint permet d'obtenir les cinq prochaines lignes de données à partir du DataFrame.

- Il utilise une approche de pagination, en maintenant un indice global pour suivre la position actuelle dans le DataFrame.

2. Pagination :

- La pagination garantit un accès contrôlé aux données, évitant le chargement massif de l'ensemble du DataFrame.

- Chaque appel à l'endpoint renvoie les cinq prochaines lignes de données.

3. Gestion de Fin de Données :

- L'API informe lorsque la fin des données est atteinte, permettant aux consommateurs de gérer cet état.

4. Utilisation de Pandas et Flask :

- La bibliothèque Pandas est utilisée pour manipuler les données du DataFrame.

- Flask, un framework Web léger, est utilisé pour créer l'API.

Producteur Kafka pour l'Intégration des Données en Temps Réel

Vous avez créé un producteur Kafka pour intégrer les données de votre API Flask dans un flux en temps réel. Cela permet de propager les données vers un système de traitement en continu, ce qui peut être utile pour des analyses en temps réel ou pour alimenter d'autres applications. Voici comment vous pourriez inclure cette partie dans votre rapport :

Producteur Kafka pour l'Intégration des Données en Temps Réel

Objectif

Intégrer les données de l'API Flask dans un système de messagerie Kafka pour permettre un traitement en temps réel et une consommation par d'autres services.

Points Principaux

1. Librairie Confluent Kafka :

- Utilisation de la librairie `confluent\_kafka` pour créer un producteur Kafka.

- Configuration du serveur bootstrap de Kafka (`bootstrap.servers`).

2. Consommation de l'API Flask :

- Utilisation de l'API Flask précédemment mise en place pour obtenir les cinq prochaines lignes de données.

3. Sérialisation des Données :

- Chaque ligne de données est sérialisée au format JSON avant d'être envoyée à Kafka.

4. Production vers le Sujet Kafka :

- Chaque ligne est produite (envoyée) sur le sujet Kafka spécifié (`product`).

- La clé du message est définie comme l'`itemId` pour assurer un traitement ordonné.

- L'enregistrement Kafka peut être consommé par d'autres applications ou services.

5. Simulation d'Intégration en Temps Réel :

- Une boucle `while` simule la consommation en temps réel des données toutes les 3 secondes.

Consommateur Spark pour l'Intégration des Données en Temps Réel

Vous avez mis en place un consommateur Spark pour traiter les données en temps réel provenant du producteur Kafka. Voici comment vous pourriez inclure cette partie dans votre rapport :

Consommateur Spark pour l'Intégration des Données en Temps Réel

Objectif

Consommer les données du sujet Kafka, les traiter à l'aide de Spark Structured Streaming, et les écrire dans des bases de données Cassandra et Elasticsearch pour permettre des analyses et des requêtes rapides.

Points Principaux

1. Configuration Spark :

- Utilisation de Spark Structured Streaming pour traiter les flux de données en temps réel.

- Configuration des dépendances pour Kafka, Cassandra, et Elasticsearch.

2. Consommation de Données Kafka :

- Définition du serveur bootstrap de Kafka et du sujet à consommer.

- Configuration du consommateur Kafka dans Spark.

3. Désérialisation des Données :

- Désérialisation des données JSON provenant de Kafka en utilisant un schéma défini.

4. Traitement des Données :

- Nettoyage des données (remplacement des valeurs nulles, conversion de types, etc.).

- Définition de transformations pour préparer les données pour l'écriture dans Cassandra et Elasticsearch.

5. Écriture dans Cassandra :

- Création des cléspace et table Cassandra.

- Définition du schéma de la table Cassandra.

- Écriture des données traitées dans Cassandra.

6. Écriture dans Elasticsearch :

- Définition du schéma pour Elasticsearch.

- Écriture des données traitées dans Elasticsearch.

7. Autres Transformations :

- Catégorisation des articles en fonction de leur gamme de prix.

- Calcul de la durée d'expédition.

8. Démarrage des Flux :

- Démarrage des flux de traitement pour Cassandra et Elasticsearch.

- Attente de la terminaison des flux (optionnel).

Conclusion

La mise en œuvre d'une architecture complète pour la gestion et l'analyse des données, couvrant les aspects du Data Warehousing, du traitement en temps réel et de l'intégration avec des bases de données NoSQL, a été réalisée avec succès. Cette architecture offre une solution robuste et évolutive pour répondre aux exigences croissantes de traitement et d'analyse des données dans un environnement moderne.

Points Forts

1. Data Warehousing avec SQLAlchemy :

- Modélisation efficace des dimensions et des faits pour représenter les données.

- Utilisation de SQLAlchemy pour interagir avec une base de données SQL Server.

- Automatisation du processus de création de tables avec des scripts Python.

2. Traitement en Temps Réel avec Apache Kafka :

- Mise en place d'une architecture de streaming pour le traitement des données en temps réel.

- Utilisation de producteurs et de consommateurs Kafka pour garantir un flux continu de données.

3. Traitement en Temps Réel avec Apache Spark Structured Streaming :

- Utilisation de Spark pour le traitement de flux en temps réel.

- Intégration de données provenant de Kafka, avec des transformations et un nettoyage des données.

4. Stockage des Données dans Cassandra :

- Configuration et création d'une base de données Cassandra pour le stockage des données en temps réel.

- Définition d'un schéma Cassandra pour répondre aux besoins spécifiques.

5. Indexation et Recherche avec Elasticsearch :

- Indexation des données dans Elasticsearch pour permettre des requêtes rapides.

- Configuration des index Elasticsearch pour un accès efficace aux informations.

6. API Flask pour l'Accès aux Données en Batch :

- Création d'une API Flask pour accéder aux données du Data Warehouse de manière asynchrone.

- Utilisation de pandas pour charger et manipuler les données en lot.

7. Mise en Place d'un DAG Apache Airflow pour l'Automatisation des Tâches :

- Définition d'un DAG Airflow pour automatiser le processus de traitement de données par lots.

Perspectives Futures

1. Optimisation des Performances :

- Évaluation continue des performances de l'architecture pour identifier et résoudre les éventuels points de congestion.

- Mise en œuvre de techniques d'optimisation pour garantir une évolutivité sans heurts.

2. Sécurité et Gouvernance des Données :

- Renforcement des mesures de sécurité pour protéger l'accès aux données sensibles.

- Mise en place de politiques de gouvernance des données pour garantir la qualité et la cohérence des données.

3. Intégration de Nouvelles Sources de Données :

- Extension de l'architecture pour intégrer de nouvelles sources de données et répondre aux besoins en constante évolution.

4. Amélioration de l'Expérience Utilisateur :

- Enrichissement de l'API Flask pour fournir des fonctionnalités avancées et une expérience utilisateur améliorée.

En conclusion, cette architecture offre une solution complète et flexible pour la gestion des données, de l'ingestion à l'analyse, en passant par le traitement en temps réel. Son évolutivité et sa modularité en font une base solide pour répondre aux défis futurs de gestion et d'analyse des données.