SOYHUCE

Challenge Technique

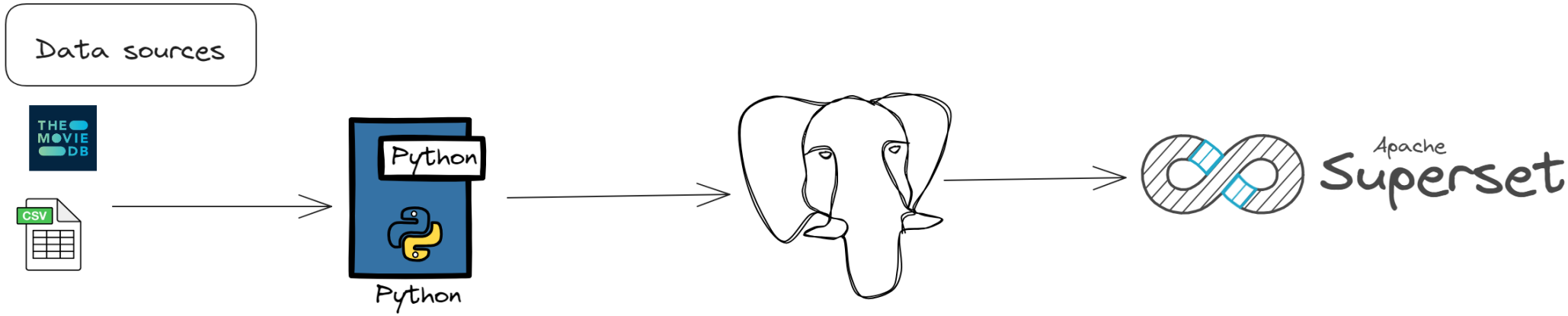
Data Engineer

Abdoul-Karim CAMARA

Contexte du projet

- Le projet avait pour but de concevoir une architecture de données permettant d'ingérer, de nettoyer et de croiser des données issues de différentes sources, principalement une API (TMDB) et un fichier CSV. L'objectif final était de fournir un jeu de données utilisable pour des analyses et des modélisations par des data scientists ou des analystes métiers.
- Le fichier plat (csv) contient des informations sur diverses maisons de production de films, telles que Marvel Comics, Legendary Pictures et Lucasfilm. Les colonnes incluent :Marque : Le nom de la marque.Total : Les revenus totaux générés par la marque.Sorties : Le nombre de sorties (films ou autres productions) de la marque.#1 Sortie : La production la plus rentable de la marque.Revenu à Vie : Les revenus totaux générés par la sortie la plus rentable de la marque.

Architecture



- **Ingestion des données**: Utilisation de scripts Python pour appeler l'API TMDB et lire le fichier CSV.
- **Nettoyage et Transformation**: Vérification des données pour les incohérences et transformation des données pour assurer une intégration homogène dans la base de données.
- **Stockage**: Les données transformées sont stockées dans une base de données PostgreSQL.
- **Croisement des données**: Scripts SQL pour joindre les données de l'API TMDB avec celles du CSV et en extraire des insights.
- **Visualisation**: Visualiser les données dans Superset ou création de dashbords

Choix d’Implémentation Technique

- **Sources de Données**:
  - **API TMDB**: Fournit des informations sur les films, y compris les titres, genres, résumés, popularité, dates de sortie, notes moyennes et nombre de votes.
  - **Fichier CSV**: Contient des données sur les performances de diverses marques de films (par exemple, Marvel, Pixar) avec des détails tels que les revenus totaux, le nombre de sorties, le film numéro un et ses revenus à vie.
- **Technologies Utilisées**:
  - **Python**: Utilisé pour les scripts de chargement et de transformation des données.
  - **PostgreSQL**: Base de données SQL utilisée pour stocker les données ingérées.
  - **Docker**: Conteneurisation des services pour assurer la portabilité et la facilité de déploiement.
  - **SQL**: Pour croiser et analyser les données ingérées.
  - **Superset**: Pour la visualisation des données

Démarche

- **Ingestion des Données**:
  - **API TMDB**: Extraction des informations suivantes pour chaque film :
    - Titre original
    - Genres
    - Résumé
    - Popularité
    - Date de sortie
    - Note moyenne
    - Nombre de votes
    - Titre normalisé
  - **CSV**: Lecture des données concernant les marques et leurs films de succès.

- **Nettoyage et Transformation**:
  - Normalisation des titres pour assurer la correspondance entre les films de TMDB et ceux du CSV.
  - Gestion des valeurs manquantes et des incohérences.
- **Stockage des Données**:
  - Insertion des données nettoyées et transformées dans PostgreSQL.
  - Création de tables pour stocker séparément les données de TMDB et celles du CSV.
- **roisement des Données**:
  - Utilisation de requêtes SQL pour joindre les tables sur les titres normalisés.
  - Calcul des statistiques agrégées telles que la popularité moyenne par marque, la comparaison des notes moyennes et du nombre de votes entre les films de différentes marques ainsi que d’autres statistiques

Possibilités d’Amélioration

Déploiement d'Image Privée: Au lieu de pousser mon image sur le registre public Docker, je peux la mettre dans un registre privé, tel que Azure Container Registry, pour une sécurité accrue et un contrôle d'accès amélioré.

Surveillance et Monitoring: Utilisation de Prometheus pour surveiller la santé du pipeline, notamment l'instance PostgreSQL, et remonter des métriques affichées sur Grafana pour un monitoring en temps réel.

Orchestration du Pipeline: Utilisation d'Airflow pour orchestrer le pipeline de manière efficace, permettant des exécutions planifiées (par jour, semaine, mois) et la gestion des dépendances entre tâches.

Haute Disponibilité et Tolérance aux Pannes: Assurer que le pipeline soit hautement disponible, tolérant aux pannes et scalable. Par exemple, mettre en place des mécanismes de redondance pour que les utilisateurs puissent continuer à utiliser le système même en cas de défaillance d'un composant du pipeline.

Utilisation des Services Cloud: Si les ressources financières le permettent, utiliser directement des services du cloud (Azure, AWS, GCP) pour déployer le pipeline, bénéficiant ainsi de plateformes entièrement managées et se concentrant sur les aspects métier.

Outils d'Intégration de Données: Pour l'intégration des données, on pourrait utiliser des outils comme Fivetran ou Airbyte, qui automatisent et simplifient les processus d'ingestion.

Transformation Haute Performance: Utilisation d'Apache Spark pour des transformations de données à haute performance, permettant de traiter des volumes de données massifs de manière efficace.

Qualité des Données: Utilisation d'outils de qualité des données, tels que Great Expectations, pour valider, documenter et assurer la qualité des données tout au long du pipeline.

Gouvernance de Données: Mise en place de politiques de gouvernance de données pour assurer la conformité, la sécurité et la gestion des accès aux données.

Difficultés Rencontrées

- Une des principales difficultés rencontrées a été la connexion à PostgreSQL en utilisant Kubernetes et Helm. Mon script Python d'extraction et de transformation des données a bien été poussé sur Docker et récupéré correctement par la création d’une chart helm (a été également exécuté mais le pipeline bloc au niveau de la connexion à postgres pour charger les données). Cependant, je n'ai pas réussi à établir une connexion stable avec PostgreSQL pour charger les données transformées. Cette difficulté était principalement due à des configurations complexes de Helm et à des problèmes de connectivité au sein du cluster Kubernetes.
  Par manque de temps, j’ai utilisé l’option docker-compose.