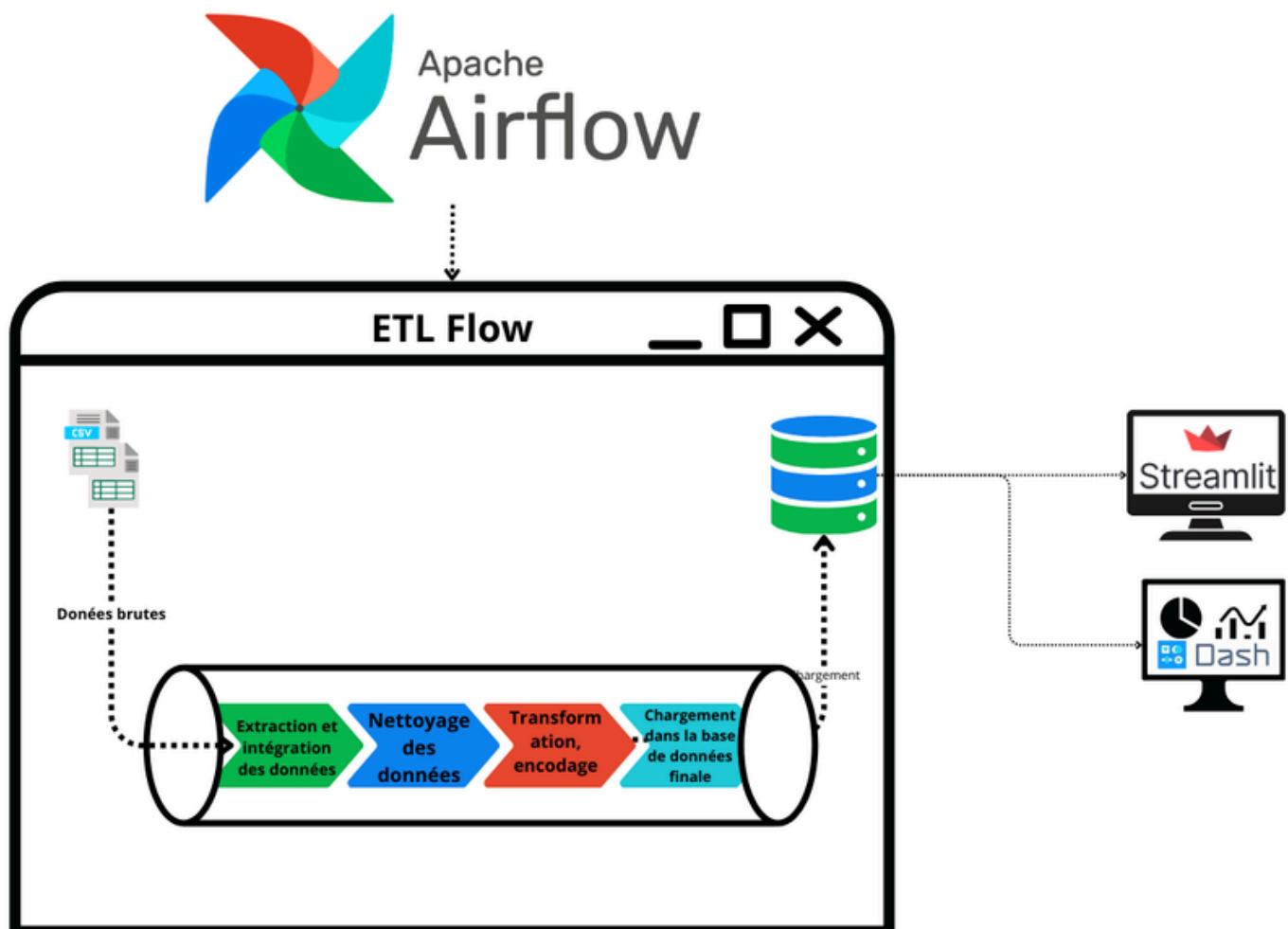


AUTOMATISATION ET ORCHESTRATION DES DONNÉES AVEC APACHE AIRFLOW : DE L'ETL AU TABLEAU DE BORD

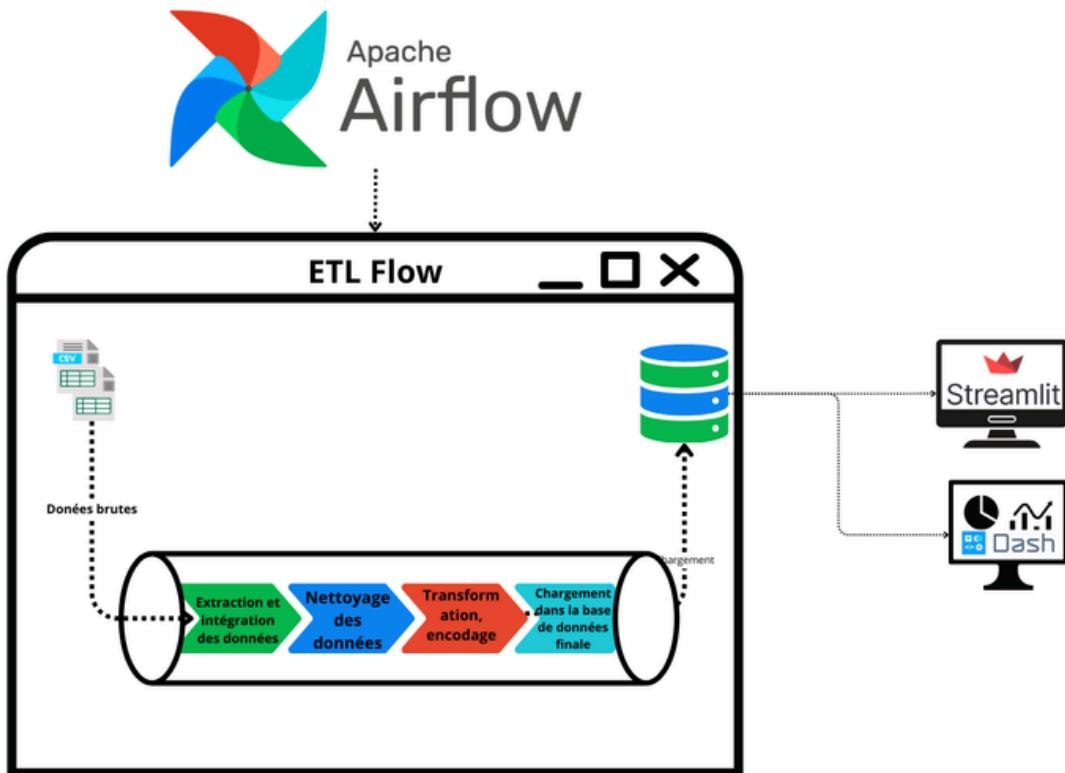
04/01/2025



Présenter par :
Laadaili Aya

Encadrer Par :
Mr.Marzak Abdelaziz
Mr.Assahla Abderrahim

OBJECTIFS DE L'ATELIER



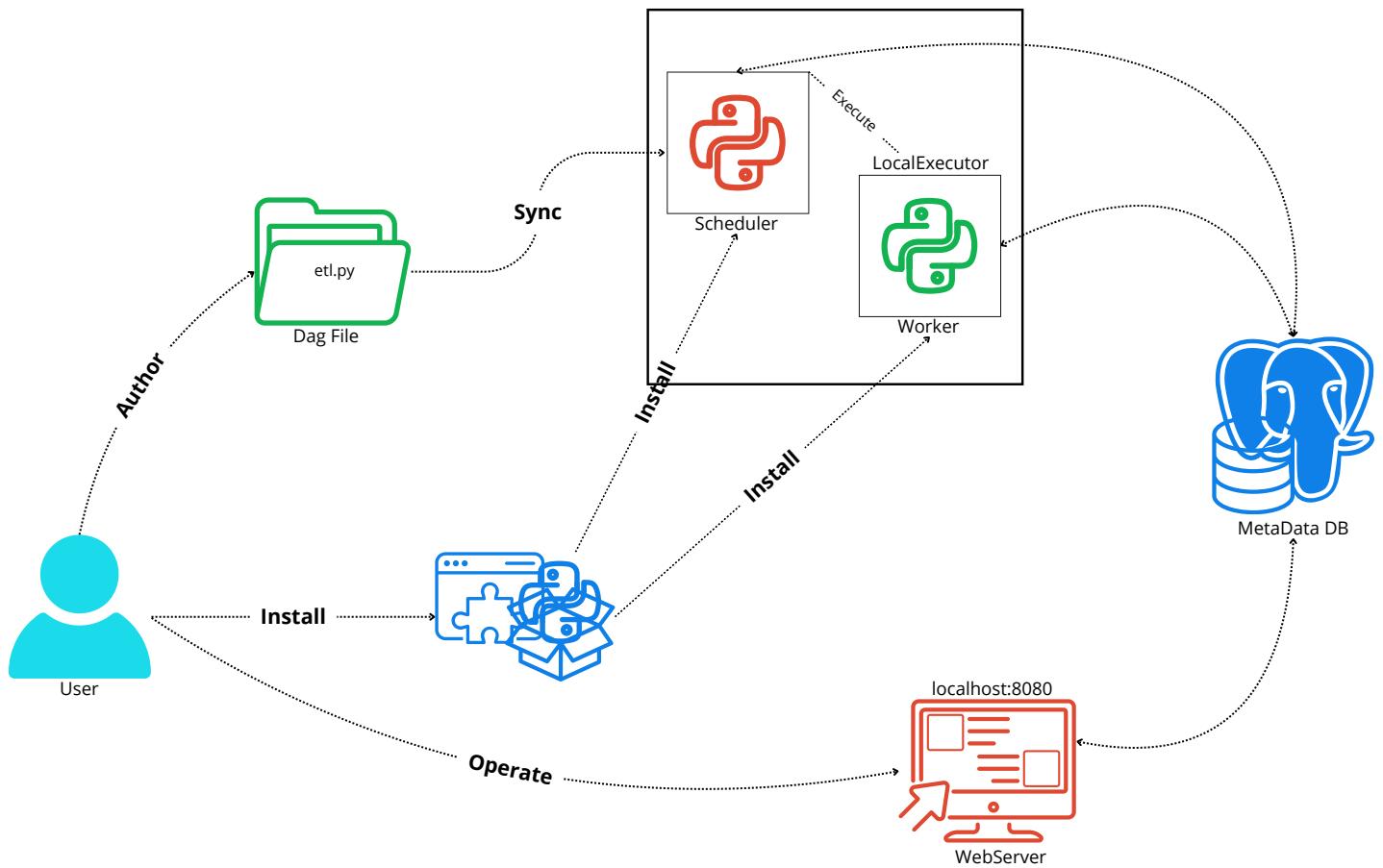
Découvrir comment automatiser et orchestrer un pipeline ETL : Configurer et exécuter un pipeline complet avec Apache Airflow, de l'extraction des données brutes à leur transformation et leur chargement dans une base de données.

Intégrer et nettoyer des données accidentologiques pour en extraire des insights exploitables : Explorer des techniques de manipulation de données, incluant la détection des valeurs aberrantes, le traitement des valeurs manquantes, et la création de nouvelles caractéristiques pertinentes.

Utiliser les résultats dans des tableaux de bord interactifs : Mettre en valeur les données transformées à l'aide d'outils visuels comme Dash et Streamlit pour générer des graphiques et des rapports accessibles et compréhensibles.

“Apache Airflow agit comme un chef d'orchestre pour coordonner chaque étape du pipeline ETL. Il assure l'exécution des tâches dans l'ordre défini, en gérant les dépendances entre les étapes telles que l'extraction, le nettoyage, la transformation et le chargement des données. De plus, Airflow offre une supervision complète via son interface utilisateur.”

ARCHITECTURE D'APACHE AIRFLOW



L'utilisateur est responsable de la création et de l'écriture des fichiers DAG (Directed Acyclic Graphs) pour définir le pipeline (etl.py).

-->L'utilisateur "auteur" les DAGs et installe les **dépendances nécessaires**.

Les fichiers DAG contiennent les définitions des tâches et des dépendances sous forme de code Python.

-->Ces fichiers sont synchronisés avec le Scheduler pour planifier l'exécution des tâches.

Le Scheduler est le composant d'Airflow qui orchestre les DAGs en programmant les tâches pour leur exécution.

-->Il communique avec **la base de données Metadata** pour stocker et récupérer l'état des tâches et DAGs.

Le Worker exécute les tâches planifiées par le Scheduler. Dans notre architecture, le LocalExecutor est utilisé, ce qui signifie que toutes les tâches sont exécutées localement sur une seule machine.

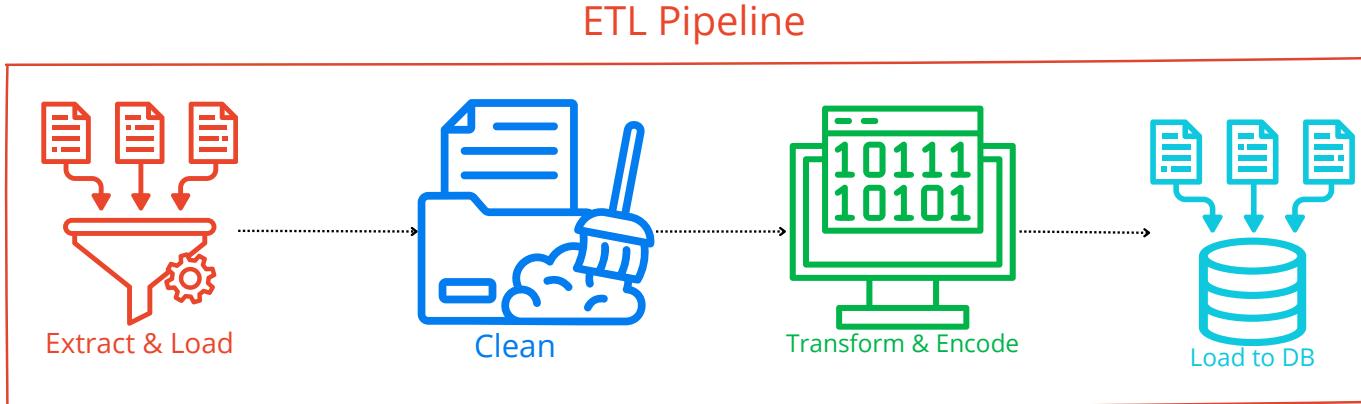
Metadata Database(PostgreSQL) est la base de données qui stocke toutes les métadonnées d'Airflow, telles que l'état des DAGs, les logs, les configurations, et les connexions.

--> Elle est mise à jour par le **Scheduler**, le **Worker** et le **WebServer**.

Le WebServer fournit une interface utilisateur graphique accessible à localhost:8080 .

-->Permet à **l'utilisateur** de surveiller, modifier et déclencher les DAGs

PIPELINE ETL AVEC DAG



Le pipeline ETL est orchestré par Apache Airflow et défini dans le fichier `etl.py`. Ce DAG, nommé `accidents_data_etl_pipeline`, comprend les étapes suivantes :

Extract and Integrate :

- Extraction des fichiers CSV situés dans le répertoire `/opt/airflow/data`.
- Intégration des fichiers pour créer un seul fichier consolidé nommé `integrated_data.csv`.

Clean :

- Nettoyage des données intégrées :
 - Suppression des colonnes inutiles ou fortement nulles.
 - Traitement des valeurs manquantes en utilisant le mode de chaque colonne.
 - Gestion des outliers (e.g., âge des conducteurs).
- Résultat : `cleaned_data.csv`.

Transform and Encode :

- Transformation et encodage des données :
 - Création de nouvelles colonnes (e.g., `Week_end`, `Time Bin`, `winter_condition`).
 - Encodage des variables catégoriques et normalisation des données numériques.
- Résultat : `preprocessed_data.csv`.

Load to Database :

- Chargement des données prétraitées dans une base de données SQLite située à `/opt/airflow/data/accidents.db`.
- Table cible : `final_data`.

APPLICATIONS POUR LA VISUALISATION



Les données transformées et stockées dans SQLite sont consommées par deux applications interactives, **Dash** et **Streamlit**, pour fournir des visualisations claires et accessibles.

Ces visualisations sont des exemples standards destinés à démontrer les possibilités d'exploitation des données. Elles peuvent être adaptées et enrichies en fonction des besoins spécifiques des utilisateurs.

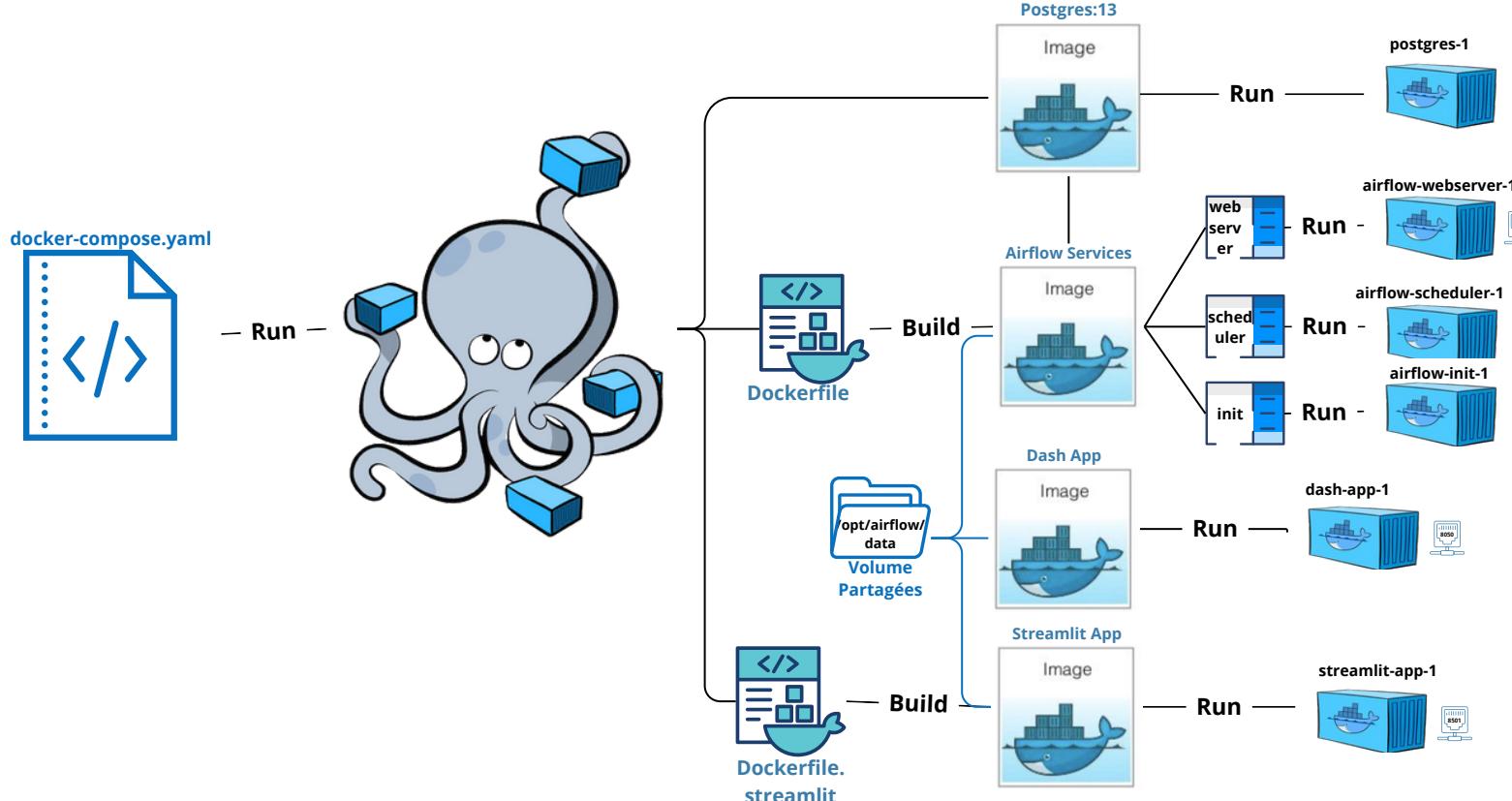
- **Dash (dashboard.py) :**

- Fournit un tableau de bord pour visualiser les relations entre les variables.
- Exemple : Affichage d'un graphique de type scatter montrant la relation entre age_of_driver et number_of_casualties.
- Accessible via le port 8050.

- **Streamlit (app.py) :**

- Offre une interface intuitive pour explorer les tables SQLite et afficher des graphiques.
- Fonctionnalités principales :
 - Exploration paginée des tables, telles que integrated_data et final_data.
 - Visualisations interactives, notamment :
 - Distribution des âges des conducteurs.
 - Nombre d'accidents en fonction des conditions hivernales.
 - Corrélations entre number_of_vehicles et number_of_casualties.
 - Répartition des accidents par jour de la semaine.
- Accessible via le port 8501.

CONTENEURISATION AVEC DOCKER



Tous les composants de l'architecture sont conteneurisés à l'aide de Docker pour faciliter le déploiement, l'isolation et la portabilité du projet.

Le fichier docker-compose.yaml définit les services nécessaires à l'exécution du projet.

- **airflow-webserver :**
 - Fournit l'interface utilisateur d'Airflow, accessible via localhost:8080.
 - Permet de surveiller, gérer et déclencher les DAGs.
- **airflow-scheduler :**
 - Programme et orchestre l'exécution des tâches définies dans les DAGs.
- **postgres :**
 - Base de données PostgreSQL utilisée par Airflow pour stocker les métadonnées :
 - États des DAGs et des tâches.
 - Logs et connexions configurées.
- **dash-app :**
 - Application Dash pour les visualisations des relations entre les variables du jeu de données.
- **streamlit-app :**
 - Application Streamlit pour explorer les tables SQLite et visualiser les données transformées sous forme de graphiques interactifs.

Volumes Partagés :

- Représenté par /opt/airflow/data, utilisé pour partager les fichiers entre les services.

DÉMONSTRATION PRATIQUE :

