Documentation Technique

Pipeline d'Ingestion Crypto vers Azure Data Lake

Projet Data Engineering - Niveau 1

Yoann LEHONG CHEFFSON

19 décembre 2025

Table des matières

1 Introduction

Ce document technique décrit l'architecture et le fonctionnement du pipeline d'ingestion de données (ETL) conçu pour le suivi des marchés de cryptomonnaies.

Dans un contexte où la fiabilité des données est critique, ce projet simule une récupération de données de marché en temps réel et assure leur stockage sécurisé et pérenne dans un environnement Cloud Enterprise (**Azure Data Lake Gen2**).

1.1 Objectifs du projet

- **Extraction :** Simuler des données de marché réalistes (Bitcoin, Ethereum, Solana, BNB) pour pallier les limitations des API publiques.
- **Transformation :** Structurer et nettoyer les données via Pandas.
- **Chargement (Load) :** Automatiser l'envoi sécurisé vers le Cloud Microsoft Azure.
- **Sécurité :** Implémenter les bonnes pratiques DevSecOps (gestion des secrets via `.env`).

2 Architecture Technique

2.1 Flux de Données (Data Flow)

Le pipeline suit une architecture linéaire de type ETL :

1. **Source :** Générateur de données Python (Simulation API).
2. **Processing Local :** Script Python + Pandas (Création du CSV).
3. **Transport :** SDK Azure Storage via protocole HTTPS sécurisé.
4. **Destination :** Azure Storage Account (Conteneur `raw-data`).

2.2 Stack Technologique

Langage Python 3.x

Cloud Provider Microsoft Azure

Librairies Clés

- `pandas` : Manipulation et structuration des données.
- `azure-storage-blob` : Interface client pour Azure Data Lake.
- `python-dotenv` : Gestion de la sécurité et des variables d'environnement.

Infrastructure Azure Data Lake Storage Gen2 (Hierarchical Namespace activé).

3 Installation et Configuration

3.1 Prérequis

- Python 3.9 ou supérieur installé.
- Un compte Microsoft Azure avec une souscription active.
- Un compte de stockage Azure créé avec un conteneur nommé `raw-data`.

3.2 Mise en place de l'environnement

Il est recommandé d'utiliser un environnement virtuel pour isoler les dépendances du projet.

```
# 1. Cr ation de l'environnement virtuel
python -m venv venv

# 2. Activation (Windows Powershell)
.\venv\Scripts\Activate

# 3. Installation des librairies requises
pip install pandas azure-storage-blob python-dotenv
```

Listing 1 – Installation des dépendances

3.3 Configuration de la Sécurité (.env)

Avertissement de sécurité : Les clés d'accès Azure (Connection Strings) sont des informations sensibles. Elles ne doivent jamais être écrites en dur dans le code source ni committées sur GitHub.

Nous utilisons un fichier `.env` (exclu par `.gitignore`) pour stocker ces secrets localement.

```
AZURE_CONNECTION_STRING="DefaultEndpointsProtocol=https;AccountName=votrecompte;
    AccountKey=votre_cle_secrete_tres_longue==;EndpointSuffix=core.windows.net"
```

Listing 2 – Format du fichier .env

4 Structure du Code (Extract & Load)

Le script `main.py` orchestre l'ensemble du pipeline. Voici les composants critiques.

4.1 Chargement Sécurisé de la Configuration

Le code initialise l'environnement en chargeant les variables cachées avant toute opération.

```
from dotenv import load_dotenv
import os

# Chargement des variables d'environnement depuis .env
load_dotenv()

# R cup ration s curis e de la cha ne de connexion
AZURE_CONNECTION_STRING = os.getenv("AZURE_CONNECTION_STRING")
CONTAINER_NAME = "raw-data"
```

4.2 Simulation des Données (Mocking)

Cette fonction remplace l'appel API pour garantir la stabilité du pipeline lors du développement, indépendamment des quotas d'API ou des pannes réseau.

```
def get_mock_data():
    """G n ration de donn es al atoires r alistes"""
    fake_data = []
    base_prices = {'bitcoin': 95000, 'ethereum': 3500}

    # Logique de variation al atoire (+/- 5%)
    # ...
    return fake_data
```

4.3 Ingestion vers Azure (Upload)

La fonction d'envoi utilise le SDK Azure Blob Storage pour transférer le fichier CSV généré.

```
def upload_to_azure(local_file_name):
    try:
        # Cr ation du client de service
        blob_service_client = BlobServiceClient.from_connection_string(
    AZURE_CONNECTION_STRING)

        # Ciblage du conteneur et du fichier
        blob_client = blob_service_client.get_blob_client(
            container=CONTAINER_NAME,
            blob=local_file_name
        )

        # Envoi des donn es binaires
        with open(local_file_name, "rb") as data:
            blob_client.upload_blob(data, overwrite=True)

        print("Succ s ! Fichier transf r  sur Azure.")

    except Exception as e:
        print(f"Erreur Critique Azure : {e}")
```

5 Exécution et Déploiement

5.1 Lancement du Pipeline

Pour exécuter le script manuellement depuis le terminal :

```
python main.py
```

5.2 Gestion de version (Git)

Le projet respecte les standards DevOps. Le fichier `.gitignore` est configuré pour exclure les fichiers sensibles et temporaires :

```
venv/           # Environnement virtuel
.env            # Secrets (CRUCIAL)
*.csv           # Donn es locales
__pycache__/    # Cache Python
*.db            # Bases de donn es locales
```

6 Conclusion

Ce projet valide les compétences fondamentales d'un Data Engineer :

1. Capacité à scripter un pipeline ETL en Python.
2. Maîtrise de l'interaction avec un Cloud Provider (Azure).
3. Application stricte des normes de sécurité des données.

Le pipeline est désormais prêt pour être étendu avec des outils de visualisation (Power BI) ou de transformation avancée (Databricks).