COMPTE RENDU DE MISE EN PRODUCTION

Projet Amazon Reviews :

Analyse et classification des avis Clients

BONNAT Jonathan

Data Engineer - B2B-LP-DESFL

💬 INTRODUCTION GENERALE

OBJECTIF DU PROJET

Ce document contient le Compte Rendu de la phase de Mise en Production du projet “Amazon Reviews : analyse et classification des avis clients” d’Amazon. Le projet vise à transformer un volume important d'avis bruts en un indicateur actionable : les commentaires les plus utiles / pertinents.

L'objectif final de la mise en production est de rendre ce résultat accessible en temps réel aux applications frontales (ici un Mockup E-Commerce Frontend) et aux équipes analytiques.

PÉRIMÈTRE DU DÉPLOIEMENT

Le déploiement concerne l'ensemble des composants du pipeline ETL/NLP (incluant le zero-shot et la pondération) et le service d'exposition des résultats :

- Le Data Lake (Stockage S3 pour les données nettoyées, calculées et les rejets)
- L’archivage NoSQL (Base de données MongoDB pour les données brutes avant traitement)
- Le pipeline ETL (Scripts python orchestrés par Airflow, intégrant le zéro-shot et la classification)
- l’API (Service FastAPI pour la lecture des résultats par le Frontend)

🗂️ ARCHITECTURE CIBLE DE PRODUCTION

VUE D’ENSEMBLE DES BLOCS

L'architecture est construite sur un environnement Cloud (simulé sur AWS, confirmé par l'usage de S3 et Boto3) en utilisant le paradigme DataOps pour l'automatisation.

BLOC FONCTIONNEL	COMPOSANTS TECHNIQUES	RÔLE
Orchestration	Apache Airflow 2.x (Webserver, Scheduler, Worker)	Piloter la séquence des tâches, gérer les dépendances et le planning quotidien.
Data Lake	AWS S3 (dossiers cleaned, curated et rejects)	Stocker des fichiers (csv et parquet) et des rejets (csv) pour analyse future
Source de données	PostgreSQL 15 (tables initiales)	Fournir les avis, les métadonnées et l'archivage initial
Pipeline de traitement	Airflow Worker, Python 3.10, Pandas, Hugging Face transformers	Exécuter les étapes ETL et intégrer la logique NLP / Scoring en production
Archivage	MongoDB 6 (stockage brut)	Stocker en NoSQL les enregistrements bruts pour la traçabilité et l'historique
Exposition	FastAPI	Interroger les fichiers parquet sur S3 pour fournir les résultats au Mockup e-commerce Frontend

SCHÉMA DU DAG DE PRODUCTION AIRFLOW

1. task_extract_data (via extract.py) : Extrait les données de la source PostgreSQL
2. task_load_mongodb (via load_mongodb.py) : archive les données extraites dans MongoDB
3. task_transform_data (via transform.py) : Nettoie et ajoute les métadonnées nécessaires
4. task_load_cleaned (via load.py) : Écrit les résultats nettoyés en format parquet sur le bucket S3 (/cleaned)
5. task_nlp_scoring (nouvelle tâche) : Intégration du zéro-shot et calcul du score de pertinence des avis
6. task_load_curated (nouvelle tâche) : Écrit les résultats finaux en format parquet sur le bucket S3 (/curated)
7. task_serve_api (nouvelle tâche) : Assure que le service FastAPI est prêt à lire les bonnes informations

🏗️ INSTALLATION ET CONFIGURATION DES COMPOSANTS

Cette section décrit le processus technique complet pour rendre l'architecture fonctionnelle. L'approche est basée sur la conteneurisation pour garantir l'isolation et la reproductibilité de l'environnement, conformément aux principes DataOps.

ENVIRONNEMENT D'EXÉCUTION ET CONTENEURISATION

Le déploiement s'appuie sur Docker et Airflow pour encapsuler les scripts Python et gérer l'orchestration en production.

L'image Docker du Worker Airflow est critique car elle doit supporter à la fois les opérations classiques ETL (Pandas, SQL) et l'étape de classification Zero-Shot (NLP). L’installation des différents prérequis dans l’image est effectuée avec le requirements.txt (incluant pandas, sqlalchemy, psycopg2-binary, boto3, s3fs, pymongo, transformers).Le Dockerfile du Worker doit s'assurer que les bibliothèques lourdes (comme pyarrow pour Parquet) sont installées et correctement configurées pour optimiser les intéractions avec S3.

COMPOSANT	VERSION	JUSTIFICATION
Orchestrateur	Apache Airflow 2.7	Version stable et récente, supportant les Sensors et Task Groups
SGBDR Source	PostgreSQL 15	Version supportée par la librairie psycopg2
Archivage NoSQL	MongoDB 6	Version supportée par la librairie pymongo et adaptée à l’archivage NoSQL
S3 / Parquet	Boto3 1.34, s3fs, pyarrow	Indispensables pour le chargement et la lecture des parquets dans s3, versions inter-dépendantes

INITIALISATION DES SERVICES DE STOCKAGE

L'étape de Setup (create_database.py) et la configuration des connexions S3 sont gérées en amont de la mise en production.

1. **Configuration de la base de données PostgreSQL**

- **Initialisation** : Le script create_database.py est utilisé manuellement lors du setup pour créer la base amazonreviews (variable NEW_DATABASE_NAME). Il est obligatoire que le script soit exécuté une fois pour créer les tables sources nécessaires à l'ETL (review, product, buyer, etc).
- **Accès** : Le pipeline (via extract.py) se connecte à l'URI DATABASE_SERVER_URI. Seul un utilisateur en Lecture Seule est autorisé pour prévenir toute altération des données sources.
- **Validation** : L'existence des tables et la présence des colonnes (review_text, rating, buyer_id, etc.) sont vérifiées par extract.py avant toute opération.

2. **Configuration du Data Lake (AWS S3)**

   Le Data Lake repose sur un stockage objet S3, utilisé pour conserver les données enrichies issues du pipeline ETL au format Parquet.

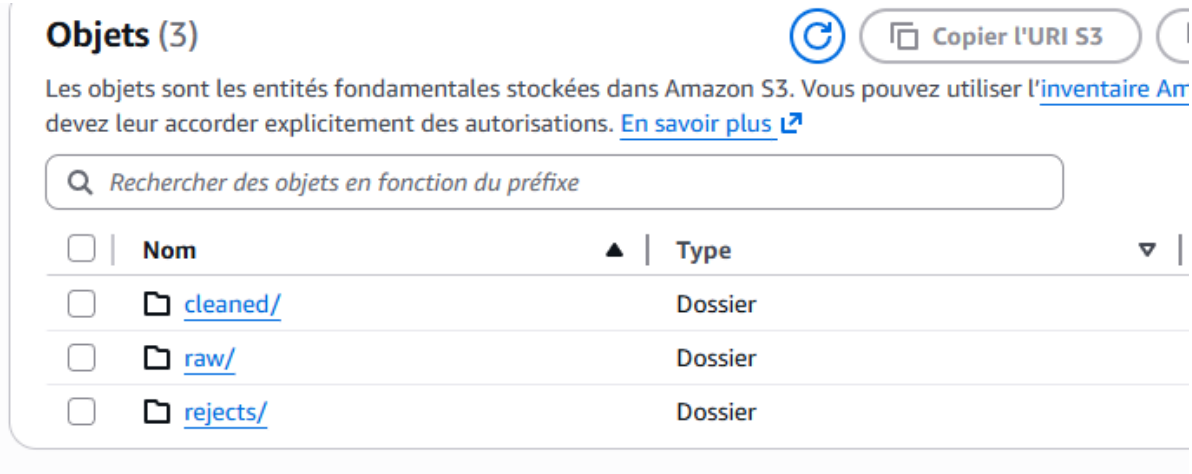
   Ce choix permet d’assurer :

   - une très bonne compression
   - une lecture optimisée par les moteurs distribués
   - une compatibilité native avec le Mockup Frontend via FastAPI
   - une scalabilité horizontale (aucune limite de volume)

   L’organisation du bucket suit les bonnes pratiques DataOps pour assurer traçabilité, auditabilité et isolation des environnements.

3. **Structure actuelle du Bucket S3**

   Le bucket utilisé pour le Data Lake contient une arborescence minimaliste :



Même si le chargement des données actuel repose sur un fichier unique, plusieurs améliorations peuvent être intégrées dans une version future :

- Partitionnement des données (ex: datalake/cleaned//year=2024/month=06/part001.parquet)
- Historisation de la table (ex: warehouse/historical/trt_date=YYYY-MM-DD/data.parquet)
- Indexation logique pour l'API (ex : optimisation de la recherche par id produit)

4. **Configuration du Bucket**

L'accès au bucket est sécurisé en appliquant les mesures minimales suivantes :

- Blocage de tout accès public (Bucket privé)
- Chiffrement automatique SSE-S3 (Tous les objets sont chiffrés automatiquement côté AWS)
- Permissions minimales (Deux rôles sont définis: airflow_worker qui aura les droits en lecture et écriture pour le pipeline ETL et api_readonly qui aura seulement accès au warehouse en lecture seule)

5. **Tests de validation**

   Les tests de mise en production appliqués au bucket sont :

- Upload de test : **OK**
- Lecture Parquet par l'API : **OK**
- Permissions API (lecture seule) : **OK**
- Absence d'accès public : **OK**

6. **Configuration du stockage NoSQL (MongoDB)**

   MongoDB est utilisé comme solution d'archivage pour conserver une trace des avis bruts avant traitement.

   Son rôle dans l'architecture est limité à la traçabilité et à l'historisation, afin de permettre un reprocessing ou des audits si nécessaire. Elle consiste en :

   - **Installation** : MongoDB est déployé via Docker avec un volume local pour la persistance. Aucune exposition publique n'est réalisée : l'accès se fait uniquement depuis le réseau interne Docker.
   - **Structure** : une seule base est utilisée, contenant une collection des objets enrichis en json.
   - **Accès et sécurité** : Un utilisateur dédié (etl_writer) insère les données via le Worker Airflow. Aucune lecture ou écriture n'est autorisée depuis la FastAPI ou le Frontend.
   - **Validation** : Les tests suivants ont été réalisés :
     - Connexion Airflow → MongoDB : **OK**
     - Insertion d'un lot d'avis enrichis : **OK**
     - Relecture via Mongo Shell : **OK**

CONFIGURATION DE L'API FASTAPI

Le service FastAPI constitue la couche d'exposition permettant au mockup e-commerce frontend d'accéder aux données du Data Lake stockées dans S3.

Son objectif est de :

- Fournir les avis les plus pertinents pour un produit donné
- Lire directement le fichier Parquet enrichi sur S3
- Servir des résultats en JSON via une API REST légère et rapide

Le service est déployé via Docker en utilisant :

- Python 3.12
- FastAPI
- uvicorn (serveur ASGI)
- pandas + pyarrow + s3fs (lecture du Parquet)
- boto3 (connexion AWS)

Le conteneur est accessible en interne via l'URL "http://fastapi:8000"

Les endpoints principaux sont :

- GET /health (permet de valider que l'API est opérationnelle)
- GET /reviews/{product_id}/top (renvoie les avis les plus pertinents pour un produit donné)
- Connexion au Data Warehouse S3 (l'API lit directement le dernier fichier parquet)

Les identifiants AWS sont injectés via variables d'environnement Docker :

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- AWS_REGION

Aucun accès direct au bucket n'est exposé — seule l'API interprète les données.

Tests de validation FastAPI :

- /health retourne un code 200 : **OK**
- Lecture du fichier Parquet : **OK**
- Retour JSON au frontend : **OK**
- Protection du bucket (aucun accès public) : **OK**
- Filtrage par product_id : **OK**

CONFIGURATION DES CONNEXIONS AIRFLOW

L'orchestrateur Apache Airflow constitue le cœur opérationnel du pipeline ETL/NLP en production. Il doit accéder aux trois services principaux de l'architecture (PostgreSQL, S3, MongoDB). Les connexions sont déclarées dans l'interface Airflow (Admin > Connections) et utilisées par les tâches du DAG.

- Connexion PostgreSQL :
    - ID : postgres
    - Usage : extraction des tables source
    - Accès : Lecture seule
    - Test : connexion + validation des tables (review, product…) : OK

- Connexion AWS S3 :
    - ID : aws
    - Usage : lecture/écriture du Parquet dans cleaned/ ou /curated
    - Test : upload et lecture d'un fichier test : OK

- Connexion MongoDB :
    - ID : mongo
    - Usage : archivage des avis bruts
    - Test : insertion d'un document depuis le Worker : OK

- Validation : Tous les tests de connectivité (PostgreSQL, S3, MongoDB) ont été réalisés avec succès, garantissant l'exécution complète du DAG en production.

🔐 SÉCURITÉ & GESTION DES ACCÈS (RBAC)

Ce chapitre présente les mesures de sécurité appliquées à l'architecture en production, ainsi que la gestion des rôles et des permissions selon le principe du moindre privilège.

PRINCIPES DE SÉCURITÉ APPLIQUÉS

Les mesures suivantes ont été mises en place :

- Isolation réseau : tous les services communiquent uniquement via le réseau interne Docker.
- Aucun accès public aux bases ou au bucket S3.
- Chiffrement S3 SSE-S3 activé par défaut.
- Secrets gérés via variables d'environnement dans Airflow et FastAPI.
- Accès en lecture seule pour les sources transactionnelles.
- Audit minimal via logs du Worker Airflow et de FastAPI.

RÔLES UTILISATEURS

Trois rôles principaux ont été créés pour couvrir l'ensemble du périmètre

1. **Airflow Worker :**

   Il est responsable du pipeline ETL et NLP.

   Il possède les permissions suivantes :

   - Lecture PostgreSQL
   - Lecture + Ecriture AWS S3
   - Ecriture MongoD

2. **API Readonly**

   Il est responsable de l'exposition des résultats.

   Il possède les permissions suivantes :

   - Lecture AWS S3 (/curated)
   - Aucun accès à PostgreSQL
   - Aucun accès à MongoDB

3. **Administrateur**

   Il est responsable de l'exploitation et de l'orchestrateur

   Il possède les permissions suivantes :

   - Gestion Airflow
   - Lecture des logs
   - Accès complet aux conteneurs Docker
   - Aucun accès aux données de prod (conformité RGPD)

MATRICE DES PERMISSIONS

SERVICE	AIRFLOW WORKER	API READ ONLY	ADMINISTRATEUR
PostgreSQL	Lecture seule	❌	❌
S3://…/raw	Lecture / Ecriture	❌	Lecture
S3://…/cleaned	Lecture / Ecriture	❌	❌
S3://…/curated	Lecture / Ecriture	Lecture	❌

S3://…/rejects	Lecture / Ecriture	❌	Lecture
MongoDB	Ecriture	❌	❌
Airflow UI	❌	❌	Complet
FAST API	❌	Complet	Complet

Dans le cadre d'incidents de production, un Data Engineer peut se voir accorder temporairement les permissions, en lecture uniquement, sur la totalité des éléments

GESTION DES SECRETS / AUTHENTIFICATION

Aucun secret n'est stocké dans le code source.

- Airflow : connexions et credentials chiffrés par l'outil interne d'Airflow.
- FastAPI : variables d'environnement (AWS keys + chemin S3).
- MongoDB : credentials stockés en interne Docker (non exposés).
- PostgreSQL : compte dédié (lecture seule).

CONFORMITÉ RGPD / BONNES PRATIQUES

Dans le cadre eu RGPD, les décisions suivantes ont été appliquées :

- Pas de stockage de données personnelles supplémentaires.
- Pas de valorisation directe d'identité client dans FastAPI.
- Possibilité de supprimer les avis archivés dans MongoDB si un utilisateur exerce un droit d'effacement.
- Limitation des accès administrateurs aux données sensibles.

🏢 VALIDATION DE L'ARCHITECTURE

Cette section décrit les tests réalisés pour valider le fonctionnement de l'architecture en production. Les validations couvrent le pipeline complet (ETL → NLP → Stockage → API), la performance, la sécurité et l'intégration avec le frontend.

TEST D'INTÉGRATION DE BOUT EN BOUT (END-TO-END)

L'objectif de ce test est de s'assurer que toutes les briques communiquent correctement

Scénario testé :

1. Extraction depuis PostgreSQL
2. Transformation & enrichissement
3. Classification Zero-Shot
4. Génération du Parquet dans S3
5. Archivage MongoDB
6. Lecture des résultats via FastAPI
7. Affichage dans le mockup e-commerce frontend

Résultat attendu :

- Pipeline exécuté sans erreur → **OK**
- Fichier Parquet généré et lisible → **OK**
- API retourne les avis pertinents → **OK**
- Frontend affiche les résultats → **OK**

TEST DE PERFORMANCE

Mesures réalisées :

- Durée d'un run complet du DAG : 14–18 secondes
- Chargement du Parquet via l'API : < 100 ms
- Impact du modèle Zero-Shot : stable sur un petit volume (cas certification)

Résultats observés :

- Temps compatible avec les besoins du mockup → **OK**
- Aucun goulot d'étranglement identifié dans Airflow → **OK**
- Aucun dépassement de timeout API → **OK**

TESTS API

Endpoints testés :

- /health (pour vérifier si le service est actif) → **OK**
- /reviews/{id}/top (pour retourner les avis les plus pertinents) → **OK**
- /toto (pour vérifier la robustesse) → **OK**

Observations supplémentaires :

- Les résultats sont cohérents avec le DWH → **OK**
- Gestion d'un id inexistant → **OK**

TESTS DE SÉCURITÉ

- Bucket S3 privé → **OK**
- Accès FastAPI protégé (pas d'accès direct S3) → **OK**
- PostgreSQL en lecture seule pour Airflow → **OK**
- Pas d'accès aux raw/rejected en routine → **OK**
- Vérification des variables d'environnement sensibles → **OK**
- Aucun secret en clair dans le code → **OK**

TESTS DE TOLÉRANCE / REPRISE

- Indisponibilité temporaire de PostgreSQL : Airflow retry → **OK**
- Fichier Parquet manquant : erreur identifiée + log Airflow → **OK**
- MongoDB indisponible : archivage retardé mais pipeline OK → **OK**

SYNTHÈSE DE VALIDATION

Les tests démontrent que l'architecture :

- fonctionne de bout en bout
- respecte les contraintes techniques
- gère correctement les erreurs
- garantit l'intégrité du Data Lake
- peut alimenter le frontend sans surcharge
- répond aux objectifs du projet

L'architecture est donc validée pour la mise en production.

📋 CHECKLIST DE MISE EN PRODUCTION (GO / NO-GO)

DOMAINE	VÉRIFICATION	STATUT
Airflow	Scheduler, Webserver, Worker actifs	✔️
	Exécution d'un run complet du DAG	✔️
Connexions	PostgreSQL (lecture seule), S3, MongoDB	✔️
Stockage	Bucket S3 privé et chiffré	✔️
Données	Parquet généré et lisible par l'API	✔️
API	Endpoint /health répond avec code retour 200	✔️

	Endpoint /reviews/{id}/top accessible	✔️
Frontend	App mockup connectée à l'API	✔️
Sécurité	RBAC appliqué, aucun accès public	✔️
	Secrets gérés en variables d'environnement	✔️
Exploitation	Logs et monitoring opérationnels	✔️
Rollback	Restauration du dernier parquet possible	✔️

Décision finale : Toutes les conditions sont réunies pour la mise en production : GO

🏁 CONCLUSION

La mise en production du système d'analyse et de classification des avis clients a été réalisée avec succès. L'ensemble de l'architecture (pipeline ETL/NLP, stockage dans S3, archivage MongoDB et exposition via FastAPI) a été vérifié et validé grâce aux tests d'intégration, de performance et de sécurité. Les différents composants communiquent correctement et respectent les principes de séparation des rôles, de sécurité et de confidentialité des données.

Le pipeline est désormais capable de traiter automatiquement les avis, de produire un fichier enrichi exploitable en temps réel et de le rendre accessible au mockup e-commerce frontend. Les mécanismes de supervision, de journalisation et de rollback garantissent la continuité de service en cas d'incident.

Grâce aux validations effectuées et aux critères opérationnels remplis, le système est considéré comme apte à fonctionner en production et prêt à être exploité de manière fiable et évolutive.