

Rapport de Mise en Place de la

Pipeline de Données

1. Introduction

Dans le cadre de ce projet, une pipeline de données a été conçue pour collecter, transformer et stocker des informations pertinentes. Ces données serviront à l'entraînement de modèles permettant de proposer des recommandations personnalisées en matière de recyclage. L'objectif principal était d'assurer une collecte efficace des données, leur transformation en un format exploitable, ainsi que leur intégration dans une infrastructure distribuée, tout en garantissant la résilience du système.

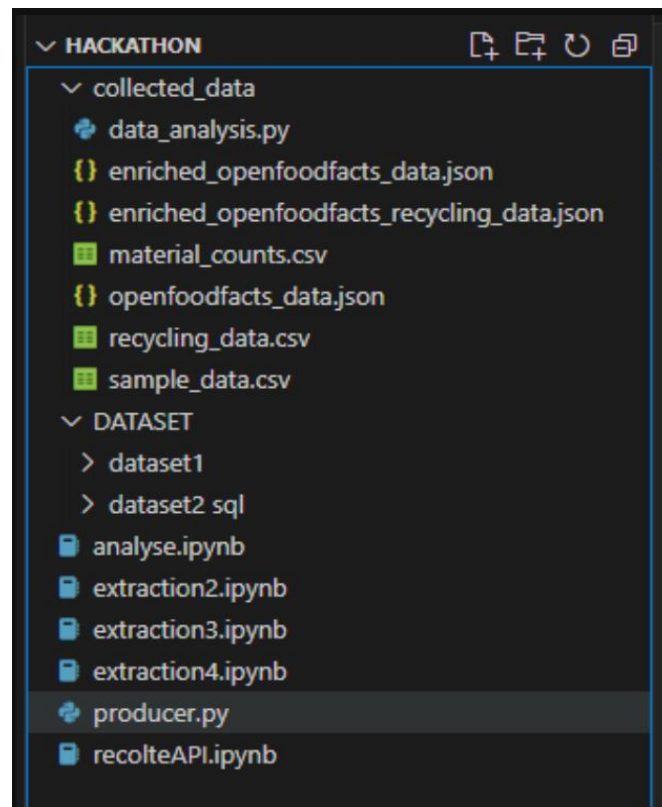
2. Collecte des Données

Les données ont été collectées principalement à partir de l'API d'Open Food Facts. Cette source a permis de récupérer des informations précises et structurées sur les produits alimentaires, leurs emballages, ainsi que leurs caractéristiques environnementales. En complément, des datasets disponibles sur Kaggle ont été téléchargés afin d'enrichir les données initiales. Ces ajouts offrent une vue plus complète et diversifiée pour renforcer la pertinence des analyses futures.

3. Transformation des Données

Une fois collectées, les données ont été nettoyées à l'aide de la librairie Pandas. Cette étape a permis de supprimer les doublons, de traiter les valeurs manquantes et de standardiser les formats tels que les unités de mesure ou les dates. Le filtrage des colonnes essentielles a également été réalisé afin de conserver uniquement les informations nécessaires à l'entraînement des modèles. Les données ainsi traitées ont été consolidées dans une base prête pour les traitements ultérieurs.

Après toutes ces procédures voici à quoi ressemble le dossier, il y a 2 sous dossier dans lesquelles il y a des datasets puis les scripts qu'on a utilisé pour récolter les données



4. Infrastructure de la Pipeline

1. Outils Utilisés

- ZooKeeper & Kafka : Streaming des données en temps réel.
- Git Bash : Gestion des scripts.

2. Collecte et Transformation

- Les données collectées via l'API sont enrichies et publiées sur Kafka en format JSON.

3. Résilience

- Callback delivery_report pour vérifier la livraison des messages.
- Gestion proactive des erreurs avec producer.flush() et des alertes.

4. Intégration

- Envoi des données conformes au modèle Open Food Facts vers Kafka (« enriched-product-data »).

5. Stockage

- Les données transformées sont sauvegardées dans une base de données locale.

L'infrastructure de la pipeline s'appuie sur plusieurs outils clés. ZooKeeper a été utilisé pour

coordonner les services et assurer une pipeline stable. Kafka a joué un rôle central en permettant de diffuser les données transformées en temps quasi réel. Git Bash a facilité la gestion et l'exécution des scripts nécessaires au fonctionnement du système. Les données collectées via l'API ont été enrichies et publiées sur Kafka, chaque message étant sérialisé au format JSON. Une fonction de callback a été implémentée pour vérifier que chaque message était correctement livré, avec un système d'alerte en cas d'échec. Les erreurs ont été gérées de manière proactive, et la commande flush a été utilisée pour garantir que tous les messages étaient bien envoyés avant la fin de l'exécution.

Le stockage des données transformées a été effectué dans une base de données locale prête à l'emploi. Les fichiers enrichis ont été conçus pour être conformes au modèle de données d'Open Food Facts et publiés dans un topic Kafka, intitulé enriched-product-data, permettant ainsi une intégration harmonieuse avec les consommateurs et les modèles exploitant ces informations.

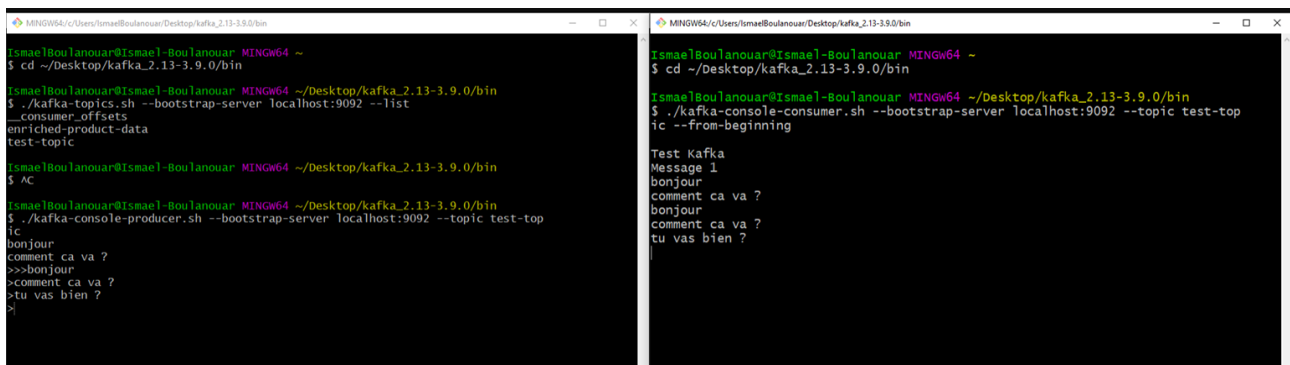
Afin de faire le test du bon fonctionnement de kafka on a paramétré les variables d'environnements pour avoir un poste de travail apte à l'utiliser, on a ensuite laissé tourné kafka et zookeeper

```
ismaelBoulanaouar@ismael-Boulanaouar MINGW64 ~/Desktop/kafka_2.13-3.9.0/bin
$ ./zookeeper-server-start.sh ./config/zookeeper.properties
[2025-01-21 08:43:41,051] INFO Reading configuration from: ./config/zookeeper.p
properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-21 08:43:41,052] WARN ./config/zookeeper.properties is relative. Prepe
nd \ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPe
erConfig)
[2025-01-21 08:43:41,053] WARN \tmp\zookeeper is relative. Prepend .\ to indicat
e that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-21 08:43:41,054] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zoo
keeper.server.quorum.QuorumPeerConfig)
[2025-01-21 08:43:41,054] INFO secureClientPort is not set (org.apache.zookeeper
.server.quorum.QuorumPeerConfig)
[2025-01-21 08:43:41,054] INFO observerMasterPort is not set (org.apache.zookeep
er.server.quorum.QuorumPeerConfig)
[2025-01-21 08:43:41,054] INFO metricsProvider.className is org.apache.zookeeper
.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumP
eerConfig)
[2025-01-21 08:43:41,055] INFO autopurge.snapRetainCount set to 3 (org.apache.zo
ookeeper.server.DataDirCleanManager)
[2025-01-21 08:43:41,055] INFO autopurge.purgeInterval set to 0 (org.apache.zook
eeper.server.DataDirCleanManager)
[2025-01-21 08:43:41,055] INFO Purge task is not scheduled. (org.apache.zookeep
er.server.DataDirCleanManager)
[2025-01-21 08:43:41,055] WARN Either no config or no quorum defined in config,
running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2025-01-21 08:43:41,056] INFO Log4j 1.2 jmx support not found; jmx disabled. (o
rg.apache.zookeeper.jmx.ManagedLogger)
[2025-01-21 08:43:41,057] INFO Reading configuration from: ./config/zookeeper.p
properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-21 08:43:41,057] WARN ./config/zookeeper.properties is relative. Prepe
nd \ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPe
erConfig)
[2025-01-21 08:43:41,057] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zoo
keeper.server.quorum.QuorumPeerConfig)
[2025-01-21 08:43:41,057] INFO secureClientPort is not set (org.apache.zookeep
er.server.quorum.QuorumPeerConfig)
[2025-01-21 08:43:41,057] INFO observerMasterPort is not set (org.apache.zooke
keeper.server.quorum.QuorumPeerConfig)

ismaelBoulanaouar@ismael-Boulanaouar MINGW64 ~/Desktop/kafka_2.13-3.9.0/bin
$ ./kafka-server-start.sh ./config/server.properties
[2025-01-21 08:51:33,787] INFO Registered kafka:type=kafka.Log4jController MBean
(kafka.utils.Log4jControllerRegistration$)
[2025-01-21 08:51:33,952] INFO Setting -o jdk.tls.rejectClientInitiatedRenegotia
tion=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.co
mmon.X509Util)
[2025-01-21 08:51:34,021] INFO starting (kafka.server.KafkaServer)
[2025-01-21 08:51:34,022] INFO Connecting to zookeeper on localhost:2181 (kafka.
server.KafkaServer)
[2025-01-21 08:51:34,040] INFO [ZookeeperClient kafka server] Initializing a new
session to localhost:2181. (kafka.zookeeper.ZookeeperClient)
[2025-01-21 08:51:34,091] INFO client environment:zookeeper.version=3.8.4-9316c2
a7a97e1666d8f4593f34dd6fc36ecc436c, built on 2024-02-12 22:16 UTC (org.apache.zo
ookeeper.Zookeeper)
[2025-01-21 08:51:34,091] INFO client environment:host.name=ismael-Boulanaouar.la
n (org.apache.zookeeper.Zookeeper)
[2025-01-21 08:51:34,091] INFO client environment:java.version=23.0.1 (org.apach
e.zookeeper.Zookeeper)
[2025-01-21 08:51:34,092] INFO client environment:java.vendor=Oracle Corporation
(org.apache.zookeeper.Zookeeper)
[2025-01-21 08:51:34,092] INFO client environment:java.home=C:\Program Files\Jav
a\jdk-23 (org.apache.zookeeper.Zookeeper)
[2025-01-21 08:51:34,092] INFO client environment:java.class.path=C:/Users/Ismae
lBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/activation-1.1.1.jar;C:/Users/Ismae
lBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/aopalliance-repackaged-2.6.1.jar;C:/U
sers/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/argparse4j-0.7.0.jar;C:/U
sers/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/audience-annotations-0.12.0.jar;C:/
Users/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/cafeine-2.9.3.jar;C:/Users
/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/commons-beanutils-1.9.4.jar;C:/U
sers/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/commons-cli-1.4.jar;C:/Users
/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/commons-collections-3.2.2.jar;C:/
Users/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/commons-digester-2.1.jar;C:/
Users/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/commons-io-2.14.0.jar;C:/U
sers/IsmaelBoulanaouar/Desktop/kafka_2.13-3.9.0/libs/commons-lang3-3.12.0.jar;C/
```

Nous avons testé une configuration locale de Kafka en envoyant et consommant des messages via

un topic nommé « test-topic ». Nous avons commencé par lister les topics disponibles à l'aide de la commande « kafka-topics.sh ». Ensuite, nous avons produit des messages textuels (par exemple : "bonjour", "comment ça va ?") grâce à la commande « kafka-console-producer.sh ». Enfin, ces messages ont été consommés en utilisant « kafka-console-consumer.sh », ce qui nous a permis de vérifier que le flux de données entre le producteur et le consommateur fonctionne correctement via Kafka.

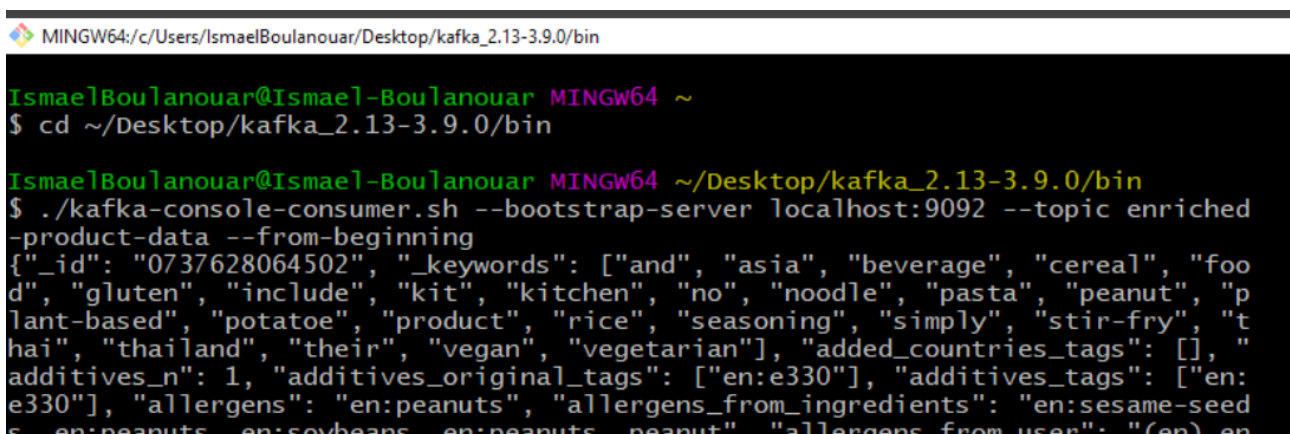


The image shows two terminal windows side-by-side. The left window shows the user navigating to the Kafka bin directory, listing topics with 'kafka-topics.sh', and then producing messages with 'kafka-console-producer.sh'. The right window shows the user consuming those messages with 'kafka-console-consumer.sh', displaying the received text: 'Test Kafka', 'Message 1', 'bonjour', 'comment ca va ?', 'bonjour', 'comment ca va ?', 'tu vas bien ?'.

```
MINGW64/c/Users/IsmaelBoulanouar/Desktop/kafka_2.13-3.9.0/bin
IsmaelBoulanouar@Ismael-Boulanouar MINGW64 ~
$ cd ~/Desktop/kafka_2.13-3.9.0/bin
IsmaelBoulanouar@Ismael-Boulanouar MINGW64 ~/Desktop/kafka_2.13-3.9.0/bin
$ ./kafka-topics.sh --bootstrap-server localhost:9092 --list
__consumer_offsets
enriched-product-data
test-topic
IsmaelBoulanouar@Ismael-Boulanouar MINGW64 ~/Desktop/kafka_2.13-3.9.0/bin
$ AC
IsmaelBoulanouar@Ismael-Boulanouar MINGW64 ~/Desktop/kafka_2.13-3.9.0/bin
$ ./kafka-console-producer.sh --bootstrap-server localhost:9092 --topic test-top
ic
ic
bonjour
comment ca va ?
>>>bonjour
>comment ca va ?
>tu vas bien ?
>

MINGW64/c/Users/IsmaelBoulanouar/Desktop/kafka_2.13-3.9.0/bin
IsmaelBoulanouar@Ismael-Boulanouar MINGW64 ~
$ cd ~/Desktop/kafka_2.13-3.9.0/bin
IsmaelBoulanouar@Ismael-Boulanouar MINGW64 ~/Desktop/kafka_2.13-3.9.0/bin
$ ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-top
ic --from-beginning
Test Kafka
Message 1
bonjour
comment ca va ?
bonjour
comment ca va ?
tu vas bien ?
|
```

Cette commande permet de consommer et afficher en temps réel, ou depuis le début, les messages JSON publiés dans un sujet Kafka (enriched-product-data). Elle connecte un **consumer** Kafka au serveur local pour lire ces données, qui transitent comme un flux, et les afficher dans le terminal. Cela ne rend pas les données persistantes comme dans une base de données, mais simplement visibles et disponibles pour un traitement ou une analyse directe.



The image shows a terminal window where the user runs 'kafka-console-consumer.sh' to consume a message from the 'enriched-product-data' topic. The output is a large JSON object containing product information like keywords, ingredients, and allergens.

```
MINGW64/c/Users/IsmaelBoulanouar/Desktop/kafka_2.13-3.9.0/bin
IsmaelBoulanouar@Ismael-Boulanouar MINGW64 ~
$ cd ~/Desktop/kafka_2.13-3.9.0/bin
IsmaelBoulanouar@Ismael-Boulanouar MINGW64 ~/Desktop/kafka_2.13-3.9.0/bin
$ ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic enriched-product-data --from-beginning
{"_id": "0737628064502", "_keywords": ["and", "asia", "beverage", "cereal", "foo", "d", "gluten", "include", "kit", "kitchen", "no", "noodle", "pasta", "peanut", "p", "lant-based", "potatoe", "product", "rice", "seasoning", "simply", "stir-fry", "t", "hai", "thailand", "their", "vegan", "vegetarian"], "added_countries_tags": [], "additives_n": 1, "additives_original_tags": ["en:e330"], "additives_tags": ["en:e330"], "allergens": "en:peanuts", "allergens_from_ingredients": "en:sesame-seed s, en:peanuts, en:soybeans, en:peanuts, peanut", "allergens_from_user": "(en) en
```

5. Prochaines Étapes

1. Entraînement du Modèle

- Utiliser les données consolidées pour des recommandations personnalisées.

2. Optimisation de la Pipeline

- Automatiser les traitements récurrents et améliorer la gestion des erreurs.

3. Documentation

- Créer des guides pour faciliter la reproduction et la maintenance.

6. Conclusion

La pipeline établit une base solide pour les analyses futures grâce à Kafka et Pandas, garantissant performance et évolutivité. Elle est prête pour intégrer des modèles prédictifs et répondre aux objectifs du projet.