Plateforme d'Analyse E-commerce en Temps Réel

Projet Big Data

Réalisé par :
AL AZAMI TAREK

Encadré par :
Professeur Hassan BADIR

Année Universitaire 2025-2026

Table des matières

1 Introduction

1.1 Contexte

Dans l'ère du commerce électronique, les entreprises génèrent des millions de transactions quotidiennes. L'analyse en temps réel de ces données est devenue cruciale pour :

- Comprendre le comportement des clients
- Optimiser les stratégies commerciales
- Prendre des décisions data-driven
- Détecter les tendances en temps réel

1.2 Objectifs du Projet

Objectifs Principaux

1. **Pipeline temps réel** : Ingestion et traitement de transactions e-commerce
2. **Analyse streaming** : Traitement avec Apache Spark Streaming
3. **Stockage optimisé** : Format Parquet pour performances maximales
4. **Visualisation** : Dashboard interactif avec mise à jour automatique
5. **Automatisation** : Déploiement Docker simplifié

1.3 Problématique

Comment concevoir une architecture Big Data capable de traiter des milliers de transactions par seconde tout en fournissant des analyses et visualisations en temps réel ?

2 Architecture Technique

2.1 Vue d'Ensemble

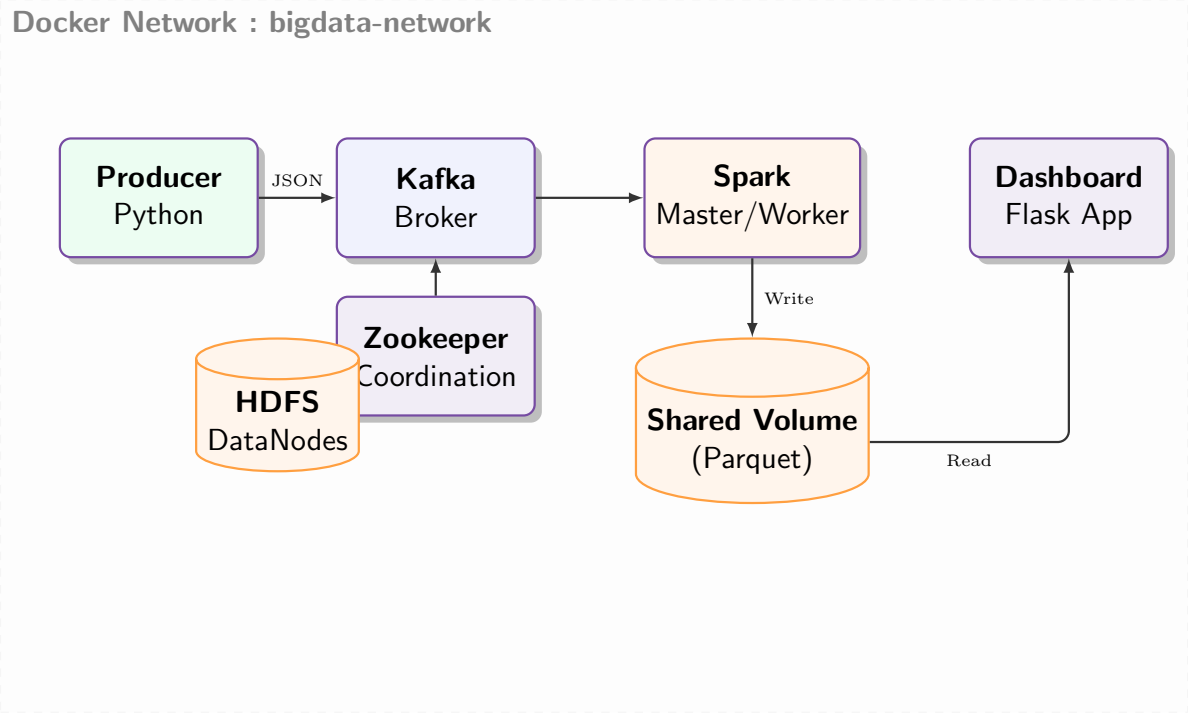

FIGURE 1 – Architecture Docker du Projet

2.2 Composants Principaux

2.2.1 Apache Kafka - Message Broker

— **Version** : Confluent 7.5.0
— **Topic** : `ecommerce-transactions`
— **Partitions** : 3
— **Rôle** : Ingestion fiable des transactions avec latence minimale

2.2.2 Apache Spark - Traitement Distribué

— **Version** : 3.5.0
— **Architecture** : 1 Master + 1 Worker
— **Mémoire** : 2 GB par worker
— **Fonctions** :
  — Consommation messages Kafka
  — Agrégations temps réel (fenêtres 1 minute)
  — Écriture Parquet optimisée

2.2.3 Dashboard Flask - Visualisation

— **Backend** : Flask + Pandas + PyArrow
— **Frontend** : HTML5 + JavaScript + Chart.js
— **Features** :
  — Métriques temps réel
  — Graphiques interactifs
  — Auto-refresh 10 secondes

2.2.4 Hadoop HDFS - Stockage

— **Version** : 3.2.1
— **Composants** : NameNode + DataNode
— **Rôle** : Stockage distribué et fiable

3 Modèle de Données

3.1 Schéma des Transactions

Champ	Type	Description
transaction_id	String	UUID unique
user_id	String	Identifiant utilisateur
city	String	Ville (6 villes marocaines)
category	String	Catégorie produit (6 types)
amount	Double	Montant (10–500 €)
quantity	Integer	Quantité (1–5)
payment_method	String	Méthode de paiement (4 types)
timestamp	Timestamp	Date et heure

3.2 Format de Stockage : Parquet

Avantages du Format Parquet

Compression : Réduction de 75% de l'espace disque

Performance : Lecture columnaire 10x plus rapide

Schéma intégré : Métadonnées incluses

Compatible : Spark, Pandas, PyArrow

4 Flux de Données

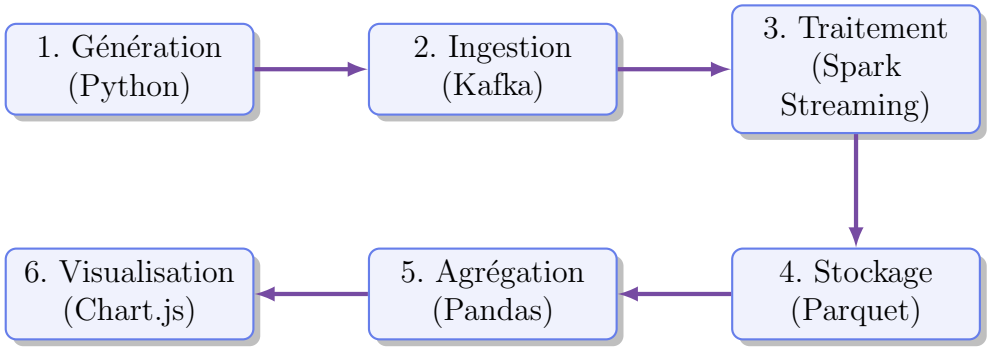

FIGURE 2 – Flux de Données E-commerce

4.1 Génération des Transactions

Listing 1 – Producteur Kafka

```python
from kafka import KafkaProducer
import json, random, uuid

producer = KafkaProducer(
    bootstrap_servers=['kafka:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

while True:
    data = {
        'transaction_id': str(uuid.uuid4()),
        'user_id': f'USER_{random.randint(100, 999)}',
        'city': random.choice(cities),
        'category': random.choice(categories),
        'amount': round(random.uniform(10.0, 500.0), 2),
        'quantity': random.randint(1, 5),
        'payment_method': random.choice(payment_methods),
        'timestamp': datetime.now().isoformat()
    }
    producer.send('ecommerce-transactions', value=data)
    time.sleep(1)  # 1 transaction/seconde
```

4.2 Traitement Spark Streaming

Listing 2 – Consumer Spark

```
# Lecture depuis Kafka
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "kafka:9092") \
    .option("subscribe", "ecommerce-transactions") \
    .load()

# Agregations par fenetre temporelle
aggregated_df = parsed_df \
    .groupBy(window(col("timestamp"), "1 minute"), "category") \
    .agg(
        sum("amount").alias("total_revenue"),
        count("*").alias("num_transactions")
    )

# Sauvegarde Parquet
query = parsed_df.writeStream \
    .format("parquet") \
    .option("path", "/tmp/ecommerce-data/raw") \
    .trigger(processingTime='30 seconds') \
    .start()
```

4.3 Visualisation Dashboard

Le dashboard Flask expose des API REST :

— `/api/stats` - Métriques globales

— `/api/categories` - Analyse par catégorie

— `/api/cities` - Analyse géographique

— `/api/payments` - Méthodes de paiement

Le frontend JavaScript interroge ces API toutes les 10 secondes et met à jour les graphiques Chart.js en temps réel.

5 Déploiement Docker

5.1 Architecture Containerisée

7 Conteneurs Docker

1. **Zookeeper** - Coordination Kafka
2. **Kafka** - Message broker
3. **Spark Master** - Nœud maître
4. **Spark Worker** - Nœud de calcul
5. **HDFS NameNode** - Métadonnées
6. **HDFS DataNode** - Stockage
7. **Dashboard** - Interface web

5.2 Volumes Partagés

— **ecommerce_data** : Fichiers Parquet partagés (Spark Dashboard)
— **hadoop_namenode** : Métadonnées HDFS
— **hadoop_datanode** : Données HDFS

5.3 Scripts d'Automatisation

setup.sh : Installation complète (conteneurs + dépendances + topic Kafka)
start-all.sh : Démarrage automatique (dashboard + consumer + producer)
run-analysis.sh : Génération de rapports analytiques
stop-all.sh : Arrêt de tous les services

6 Résultats

6.1 Exemple de Résultats

Après 3 minutes d'exécution

— **Transactions analysées** : 180
— **Utilisateurs uniques** : 124
— **Chiffre d'affaires** : 36,129.69 €
— **Montant moyen** : 259.93 €
— **Articles vendus** : 450

Top 3 Catégories

1. **Electronics** : 8,500 € (35 transactions)
2. **Fashion** : 7,200 € (28 transactions)
3. **Home & Garden** : 6,800 € (32 transactions)

6.2 Dashboard

Le tableau de bord fournit :

- 4 cartes métriques (revenus, transactions, utilisateurs, moyenne)
- Graphique barres : Revenus par catégorie
- Graphique circulaire : Revenus par ville
- Graphique donut : Méthodes de paiement
- Tableau : Top 10 transactions

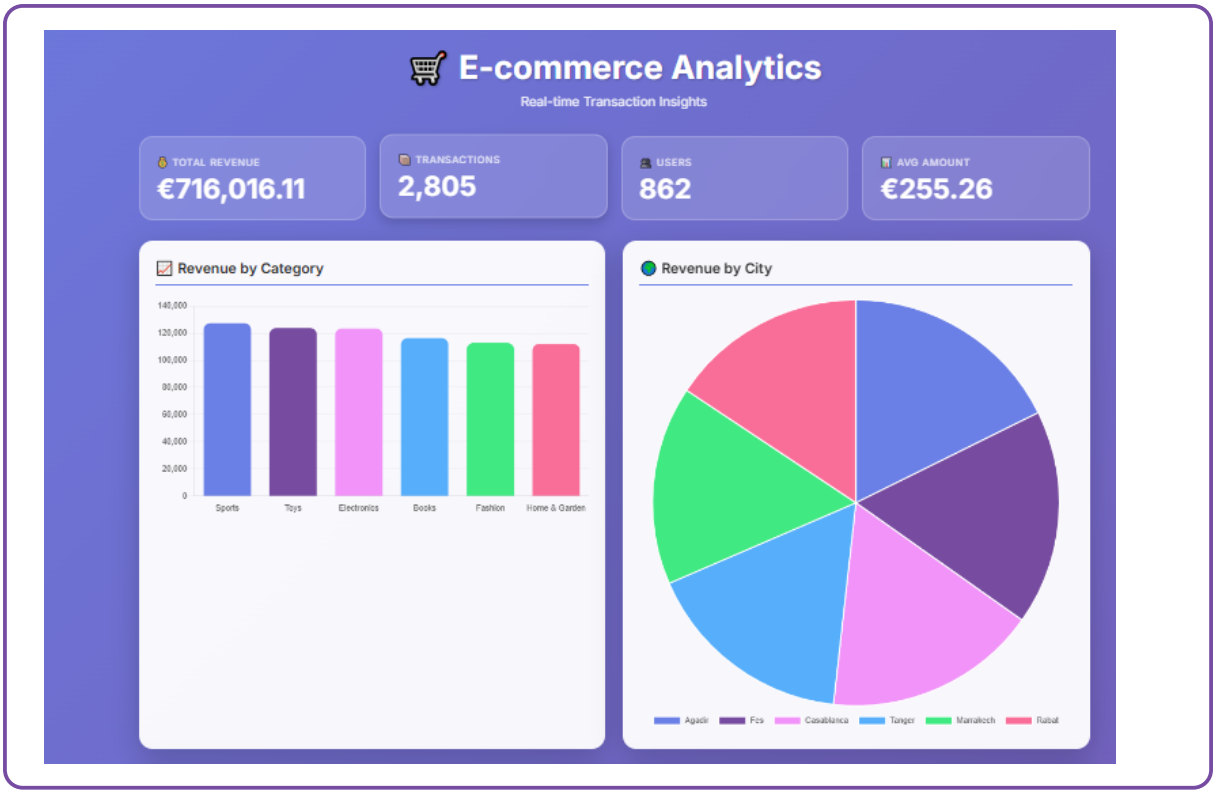

FIGURE 3 – Interface du Dashboard E-commerce

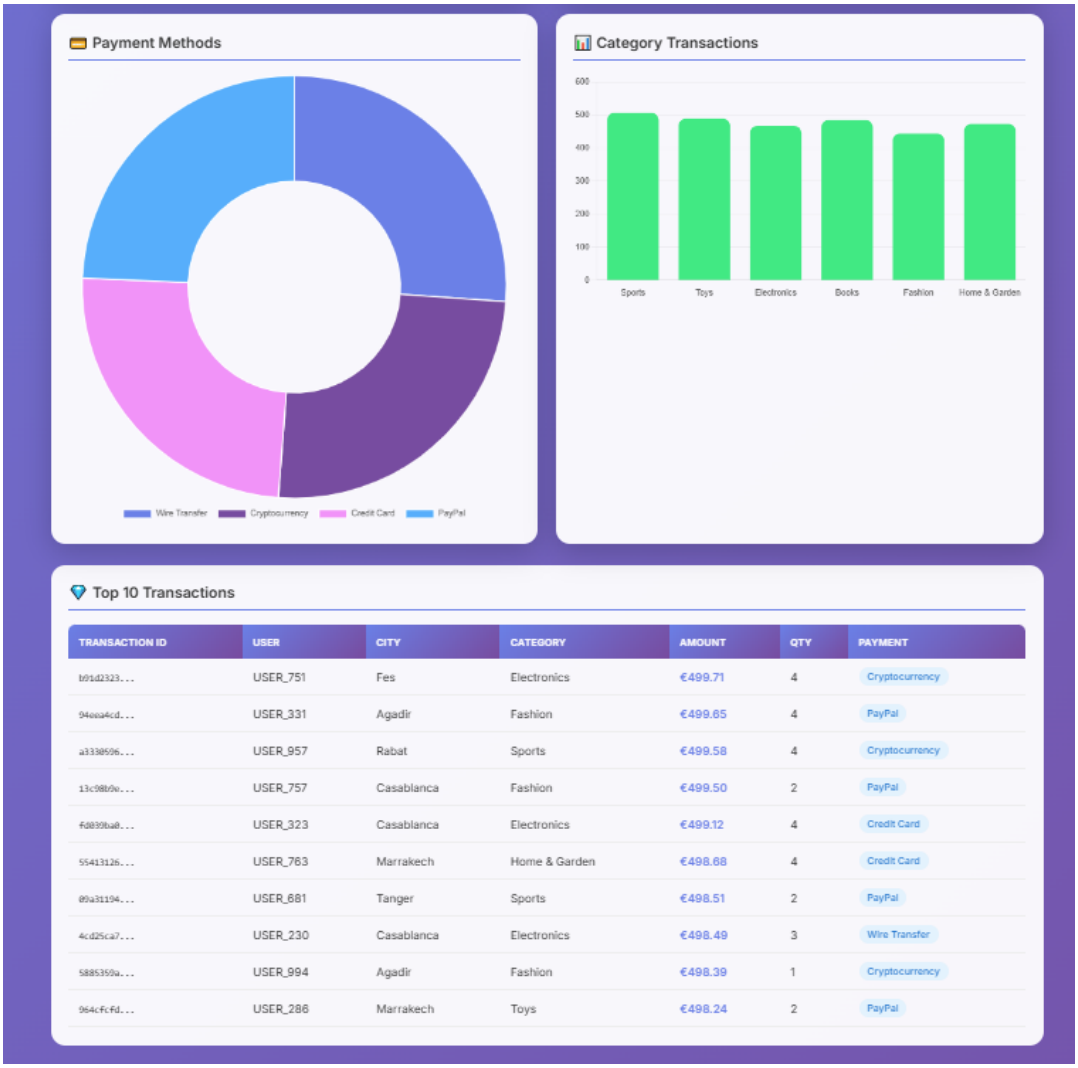

FIGURE 4 – Visualisations des Données

7 Défis Techniques

7.1 Permissions Docker

Problème : Spark ne pouvait pas écrire dans le volume partagé
Solution : Création du répertoire avec permissions appropriées

```
docker exec -u root spark-master mkdir -p /tmp/ecommerce-data/raw
docker exec -u root spark-master chmod -R 777 /tmp/ecommerce-data
```

7.2 Partage de Données

Problème : Dashboard ne pouvait pas accéder aux fichiers Parquet de Spark
Solution : Volume Docker partagé `ecommerce_data` monté sur les deux conteneurs

7.3 Optimisation Dashboard

Problème : PySpark trop lourd pour le conteneur dashboard (nécessite Java)
Solution : Utilisation de Pandas + PyArrow pour lecture efficace des Parquet

8 Conclusion

8.1 Synthèse

Ce projet a permis de concevoir et implémenter une plateforme complète d'analyse Big Data pour le traitement en temps réel de transactions e-commerce. L'architecture démontre la puissance des technologies modernes pour gérer des flux de données massifs.

8.2 Compétences Acquises

Compétences Techniques

- Maîtrise de l'écosystème Apache (Kafka, Spark, Hadoop)
- Développement d'applications streaming temps réel
- Containerisation et orchestration Docker
- Développement web full-stack (Flask + JavaScript)
- Traitement de données (Pandas, PySpark)
- Automatisation avec scripts Bash

Compétences Architecturales

- Conception d'architectures Big Data scalables
- Patterns de traitement streaming et batch
- Optimisation des performances
- Résolution de problèmes de production

8.3 Résultats Atteints

Pipeline fonctionnel

Latence ¡ 15s

Dashboard interactif

Automatisation complète

Documentation exhaustive

8.4 Perspectives

Ce projet constitue une base solide pour des applications Big Data en production. L'expérience acquise est directement applicable dans l'industrie, où les architectures de streaming et l'analyse temps réel sont essentielles pour la compétitivité.

Annexes

Structure du Projet

Arborescence des Fichiers

```
bigdata-project/
          docker-compose.yml          # Configuration des services
          README.md                   # Documentation generale
          requirements.txt            # Dependances Python
          dashboard/
                dashboard.html          # Interface Frontend
          scripts/
                producer.py             # Generateur de transactions
                consumer_spark.py       # Traitement Spark Streaming
                analysis.py             # Analyse Batch & Reporting
                dashboard_server.py     # Backend Flask API
                automation/             # Scripts d'automatisation
                    setup.sh            # Installation
                    start-all.sh        # Demarrage global
                    run-analysis.sh     # Generation de rapports
                    stop-all.sh         # Arret des services
          data/                       # Volume partage (Parquet files)
```

Technologies Utilisées

Technologie	Version
Apache Kafka	7.5.0
Apache Spark	3.5.0
Apache Hadoop	3.2.1
Python	3.9
Flask	3.0.0
Pandas	2.1.4
Chart.js	4.4.0
Docker	20.10+

Commandes Essentielles

Démarrage

```
./scripts/automation/setup.sh
./scripts/automation/start-all.sh
```

Interfaces Web

— Dashboard : `http://localhost:5000`

— Spark Master : `http://localhost:8080`

— HDFS : `http://localhost:9870`

Arrêt

```
./scripts/automation/stop-all.sh
```