Pipeline Big Data - Surveillance Temps Réel IDFM

Monitoring de la santé de la station La Croix de Berny

Blidi, Tlemsani

18 janvier 2026

Contexte et Objectifs

Problématique

Surveiller en temps réel la ponctualité du RER B à la station **La Croix de Berny**

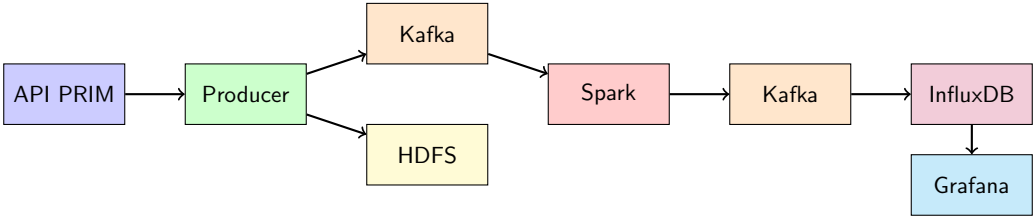
Objectifs :

- Comparer horaires théoriques vs réels
- Calculer les retards en temps réel
- Visualiser la santé du réseau
- Historiser les métriques

Source de données :

- API PRIM (Île-de-France Mobilités)
- Données en temps réel
- Format JSON

Architecture Globale



Pipeline en 4 étapes : Ingestion → Traitement → Stockage → Visualisation

Infrastructure du Cluster Azure

Machine	Rôle	Composants
master	Gestion	NameNode, ResourceManager, Kafka Broker, Zookeeper, Spark Master
worker_1	Calcul	DataNode, NodeManager, Spark Executor
worker_2	Multi-rôle	DataNode, NodeManager, Kafka Broker, Spark Executor, **InfluxDB**, **Grafana**
worker_0	Stockage	DataNode, NodeManager

Configuration

Cluster de 4 VMs Azure avec distribution des services pour la haute disponibilité

Étape 1 : Ingestion des Données

Producer Python

- Interroge API PRIM périodiquement
- Envoie vers Kafka topic `idfm-realtime`
- Sauvegarde brute dans HDFS

Stockage :

- Topic Kafka : streaming
- HDFS : archivage long terme
- Format : JSON

```python
# producer.py
def fetch_and_send():
    data = requests.get(API_URL)

    # Kafka
    producer.send(
        'idfm-realtime',
        data.json()
    )

    # HDFS
    save_to_hdfs(
        '/data/idfm/raw.json'
    )
```

Étape 2 : Analytics avec Spark Streaming

Calcul du retard

`delay_sec = heure_prévue - heure_théorique`

Traitement :

- Lecture depuis Kafka `idfm-realtime`
- Parsing JSON
- Calcul des métriques
- Envoi vers `idfm-processed`

```
# spark_analytics.py
df = spark.readStream \
    .format("kafka") \
    .option("topic", "idfm-realtime") \
    .load()

processed = df.select(
    col("line"),
    (col("predicted") - col("scheduled"))
    .alias("delay_sec")
)

processed.writeStream \
    .format("kafka") \
    .option("topic", "idfm-processed")
```

Étape 3 : Stockage dans InfluxDB

Consumer Python

- Consomme topic `idfm-processed`
- Écrit dans InfluxDB
- Optimisé pour séries temporelles
- Sur **worker_2**

Avantages InfluxDB :

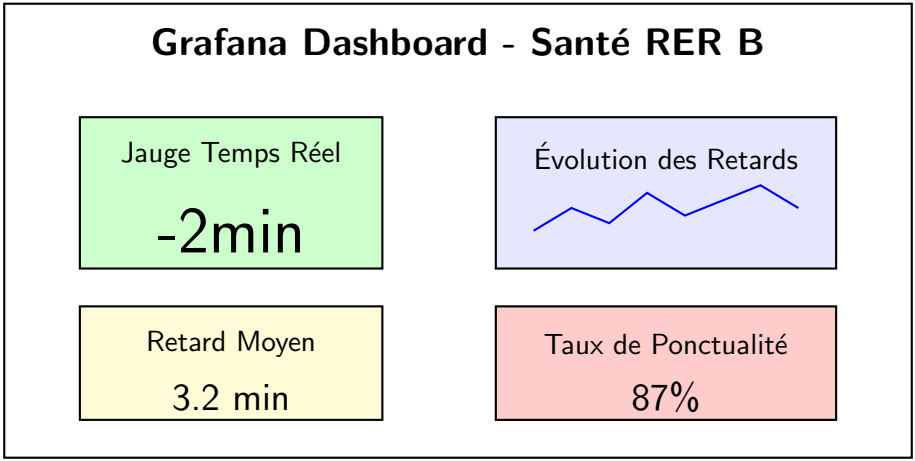
- Compression temporelle
- Requêtes rapides
- Rétention configurable

```python
# influx_consumer.py
consumer = KafkaConsumer(
    'idfm-processed',
    bootstrap_servers=[
        'master:9092',
        'worker_2:9092'
    ]
)

for msg in consumer:
    point = Point("health") \
        .tag("line", msg['line']) \
        .field("delay", msg['delay_sec'])

    write_api.write(
        bucket="idfm_metrics",
        record=point
    )
```

Étape 4 : Dashboard Grafana



Métriques affichées : Retard instantané, historique, moyennes, tendances

Runbook : Ordre de Démarrage

1. **Services de base** (master)

```
start-dfs.sh
start-yarn.sh
bin/kafka-server-start.sh -daemon config/server.properties
```

2. **Topics Kafka**

```
kafka-topics.sh --create --topic idfm-realtime --partitions 3 --replication-factor 2
kafka-topics.sh --create --topic idfm-processed --partitions 3 --replication-factor 2
```

3. **Spark Analytics** (master)

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0 spark_analytics.py
```

4. **InfluxDB Consumer** (worker_2)

```
python3 influx_consumer.py
```

5. **Producer** (master)

```
python3 producer.py
```

Stack Technologique

Big Data :

- **Hadoop HDFS** - Stockage distribué
- **Apache Spark** - Traitement temps réel
- **Apache Kafka** - Streaming

Monitoring :

- **InfluxDB** - Time-Series DB
- **Grafana** - Visualisation

Avantages :

- Scalabilité horizontale
- Traitement temps réel
- Haute disponibilité
- Historisation longue durée

Cas d'usage :

- Surveillance opérationnelle
- Analyse de performance
- Détection d'anomalies
- Reporting automatisé

Conclusion et Perspectives

Réalisations

Pipeline Big Data complète pour le monitoring temps réel du RER B avec archivage et visualisation

Améliorations possibles :

- Prédiction des retards (ML)
- Multi-lignes / Multi-stations
- Alerting automatique
- API REST pour les données

Extensions :

- Analyse des causes de retard
- Corrélation avec météo/événements
- Application mobile
- Intégration autres réseaux

Merci pour votre attention !