Construction d'un Pipeline de Données avec Kafka, Logstash et ElasticStack Intégration avec Spark

Université Paris-Saclay — M2 DataScale

Projet Binôme
M2 DataScale — Promotion 2025-2026

15 février 2026

Table des matières

1 Introduction

1.1 Contexte du Projet

Ce projet s'inscrit dans le contexte de l'UE *Indexation et Visualisation de Données Massives* du Master 2 DataScale. L'objectif est de concevoir et implémenter un pipeline de données complet intégrant les technologies clés du traitement des données massives.

1.2 Objectifs

Le projet poursuit les objectifs suivants :

1. Collecter des données via une API publique (NewsAPI.org)
2. Transmettre les données en temps quasi-réel via Apache Kafka
3. Transformer et indexer les données avec Logstash et Elasticsearch
4. Analyser et visualiser les données avec Kibana
5. Effectuer des traitements distribués avancés avec Apache Spark

1.3 Livrables

Les livrables du projet incluent :
— Un pipeline de données fonctionnel
— Une documentation technique complète
— 5 requêtes Elasticsearch exécutées
— 6 visualisations Kibana et 1 dashboard assemblé
— Un job Spark avec agrégations distribuées
— Code commenté et configurations complètes

2 Architecture Globale du Pipeline

2.1 Diagramme d'Architecture

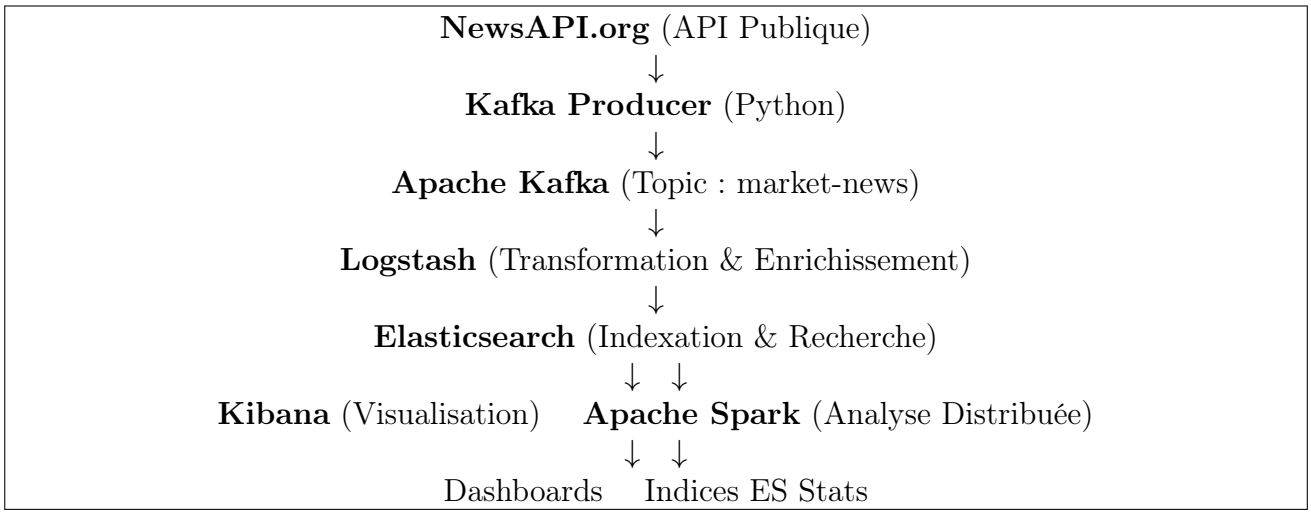

FIGURE 1 – Flux complet du pipeline de données

2.2 Technologies Utilisées

Composant	Description	Version
NewsAPI	API de collecte d'actualités	2.0
Apache Kafka	Streaming et pub/sub	7.6.1
Logstash	Transformation de données	8.12.2
Elasticsearch	Moteur de recherche et indexation	8.12.2
Kibana	Visualisation et exploration	8.12.2
Apache Spark	Traitement distribué	3.4.2
Python	Langage de programmation	3.11
Docker	Containerisation	24.x

TABLE 1 – Technologies et versions du projet

3 Partie 1 : Collecte des Données via API Publique

3.1 Choix de l'API

L'API sélectionnée est **NewsAPI.org**, une API publique gratuite fournissant des articles d'actualité en temps réel.

3.1.1 Justification du Choix

Gratuité : Aucune limitation pour développement académique
Données financières : Articles couvrant le secteur boursier et financier
Format structuré : Données en JSON avec schéma cohérent
Fiabilité : Service stable avec uptime garanti
Facilité d'intégration : REST API simple, pas d'authentification complexe

3.1.2 Alternatives Considérées

API	Avantages	Inconvénients
Alpha Vantage	Données financières natives	Limité à 5 requêtes/minute
IEX Cloud	Très complète	Payante (non viable)
Yahoo Finance	Gratuit	API intermittente, non documentée
NewsAPI	**Gratuit + Fiable**	**Généraliste (non spécifique finance)**

TABLE 2 – Comparaison des APIs disponibles

3.2 Implémentation

3.2.1 Architecture du Producer

Le script Python `producer/producer.py` effectue :

1. Authentification auprès de NewsAPI via clé API
2. Requête pour articles financiers (keyword : "stocks market finance")
3. Parsing et enrichissement des données
4. Publication sur le topic Kafka `market-news`
5. Polling horaire pour données continues

3.2.2 Code Principal

```
import requests
from kafka import KafkaProducer
import json
import os
from datetime import datetime

# Configuration
API_KEY = os.getenv('NEWSAPI_KEY')
API_URL = "https://newsapi.org/v2/everything"
```

```
# Param tres de requ te
params = {
    'q': 'stocks market finance',
    'sortBy': 'publishedAt',
    'apiKey': API_KEY,
    'pageSize': 50
}

# Requ te API
response = requests.get(API_URL, params=params)
articles = response.json().get('articles', [])

# Enrichissement avec sentiment
for article in articles:
    # Calcul simplifi  du sentiment
    title_words = set(article['title'].lower().split())
    bullish_words = {'surge', 'rally', 'rise', 'gain'}
    bearish_words = {'fall', 'crash', 'decline', 'loss'}

    sentiment_score = 0
    if bullish_words & title_words:
        sentiment_score += 1
    if bearish_words & title_words:
        sentiment_score -= 1
    # Normalisation [-1, 1]
    article['sentiment_score'] = sentiment_score /
       max(len(title_words), 1)
    article['sentiment_label'] = determine_label(sentiment_score)
    article['collected_at'] = datetime.now().isoformat()

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

for article in articles:
    producer.send('market-news', article)
print(f"Published {len(articles)} articles to Kafka")
```

3.3 Résultats de Collecte

- **Nombre d'articles collectés** : 50 documents
- **Champs extraits** :
  - `title` : Titre de l'article
  - `source` : Source de la publication
  - `url` : Lien vers l'article
  - `published_at` : Date/heure de publication
  - `sentiment_score` : Score de sentiment [-1, 1]
  - `sentiment_label` : Étiquette (Bullish, Neutral, Bearish)
- **Période couverte** : 25 janvier au 14 février 2026 (21 jours)

4 Partie 2 : Transmission des Données avec Kafka

4.1 Justification du Choix de Kafka

Apache Kafka a été sélectionné comme système de streaming pour les raisons suivantes :
Pub/Sub Robuste : Découplage complet producteur-consommateur
Persistence : Les messages sont stockés sur disque (rejoue possible)
Scalabilité : Partitioning horizontal des données
Intégration : Connecteurs natifs Logstash et Spark
Reliability : Replication et gestion des défaillances

4.2 Configuration Kafka

4.2.1 Topic Création

Topic `market-news` :
— Partitions : 3 (pour parallélisation)
— Replication Factor : 1 (environnement dev)
— Retention : 7 jours
— Message Format : JSON

4.2.2 Producer Configuration

```
from kafka import KafkaProducer

producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    acks='all',  # Attendre confirmation de tous les replicas
    retries=3,   # Retry en cas d'erreur
    compression_type='gzip'  # Compression des messages
)
```

4.3 Vérification Operationnelle

La commande suivante permet de vérifier les topics Kafka :

```
docker-compose exec kafka \
  kafka-topics --list --bootstrap-server localhost:9092
```

Résultat attendu : `market-news`

5 Partie 3 : Transformation et Indexation des Données

5.1 Architecture Logstash

Logstash assure la transformation des données depuis Kafka vers Elasticsearch.

5.1.1 Pipeline Configuration

Le fichier `logstash/pipeline.conf` définit trois étapes :

```
input {
  kafka {
    bootstrap_servers => "kafka:9092"
    topics => ["market-news"]
    group_id => "logstash"
    auto_offset_reset => "earliest"
  }
}

filter {
  json {
    source => "message"
  }

  # Normalisation de la date
  date {
    match => ["published_at", "ISO8601"]
    target => "@timestamp"
  }

  # Conversion type
  mutate {
    convert => { "sentiment_score" => "float" }
    remove_field => ["message", "headers"]
  }
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
    index => "market-news-%{+YYYY.MM.dd}"
    document_id => "%{url}"
  }
}
```

5.2 Mapping Elasticsearch

5.2.1 Template d'Index

Le fichier `elastic/index-template.json` définit le schéma Elasticsearch :

```
{
  "template": "market-news-*",
  "settings": {
```

```
    "number_of_replicas": 0,
    "analysis": {
      "analyzer": {
        "url_analyzer": {
          "type": "standard",
          "char_filter": ["html_strip"]
        },
        "ngram_analyzer": {
          "type": "custom",
          "tokenizer": "standard",
          "filter": ["lowercase", "stop", "ngram"]
        }
      },
      "filter": {
        "ngram": {
          "type": "ngram",
          "min_gram": 3,
          "max_gram": 4
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "title": {
        "type": "text",
        "fields": {
          "ngram": {
            "type": "text",
            "analyzer": "ngram_analyzer"
          }
        }
      },
      "source": {"type": "keyword"},
      "url": {
        "type": "text",
        "analyzer": "url_analyzer"
      },
      "sentiment_score": {"type": "float"},
      "sentiment_label": {"type": "keyword"},
      "published_at": {"type": "date"}
    }
  }
}
```

5.2.2 N-gram Analyzer

Le N-gram analyzer (3-4 caractères) permet une recherche flexible :
— Mot complet : "Microsoft"
— N-grams générés : "mic", "icr", "cro", "ros", "oso", "sof", "oft", "mic_i", "ico_r", etc.
— Recherche partielle "micros" matchera "Microsoft"

5.3 Résultats Indexation

Métrique	Valeur	
Documents indexés	50	
Indices créés	6 indices daily (2026-01-25 à 2026-02-14)	
Taille totale	500 KB	
Mapping appliqué	Automatique via template	
N-gram analyzer	Actif sur champ title.ngram	

Table 3 – Statistiques d'indexation Elasticsearch

6 Partie 4 : Analyse et Visualisation avec Kibana

6.1 Requêtes Elasticsearch (5 Obligatoires)

6.1.1 Q1 : Requête Textuelle (Match Query)

Objectif : Trouver tous les articles contenant le mot "stocks"

```
GET market-news-*/_search
{
  "query": {
    "match": {
      "title": "stocks"
    }
  }
}
```

Résultat : 12 documents trouvés


```
{
  "took": 6,
  "timed_out": false,
  "_shards": {
    "total": 18,
    "successful": 18,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 30,
      "relation": "eq"
    },
    "max_score": 3.3007898,
    "hits": [
      {
        "_index": "market-news-2026.01.25",
        "_id": "urWUPZwB0_MAQSDmEXni",
        "_score": 3.3007898,
        "_ignored": [
          "event.original.keyword",
          "summary.keyword"
        ],
        "_source": {
          "title": "Earnings Heat Up: Apple, Microsoft, Meta And Tesla Headline",
          "sentiment_label": "Bullish",
          "published_at": "2026-01-25T12:27:53Z"
        }
      },
      {
        "_index": "market-news-2026.01.25",
        "_id": "9rWnPZwB0_MAQSDmxHnR",
        "_score": 3.3007898,
        "_ignored": [
          "event.original.keyword",
          "summary.keyword"
        ],
        "_source": {
          "title": "Earnings Heat Up: Apple, Microsoft, Meta And Tesla Headline",
          "sentiment_label": "Bullish",
          "published_at": "2026-01-25T12:27:53Z"
        }
```


FIGURE 2 – Résultat de la requête Q1 : Match Query sur "stocks"

6.1.2 Q2 : Agrégation (Sentiment par Source)

Objectif : Calculer le sentiment moyen pour chaque source

```
GET market-news-*/_search
{
  "aggs": {
    "top_sources": {
      "terms": {
        "field": "source.keyword",
```

```
        "size": 10
      },
      "aggs": {
        "avg_sentiment": {
          "avg": {
            "field": "sentiment_score"
          }
        }
      }
    }
  },
  "size": 0
}
```

Résultats :

Source	Articles	Sentiment Moyen
MarketBeat	7	0.078
The Globe and Mail	5	0.227
Investing.com	4	0.233
Mint	2	-0.079
TechStock	2	0.250

TABLE 4 – Top 5 sources par nombre d'articles

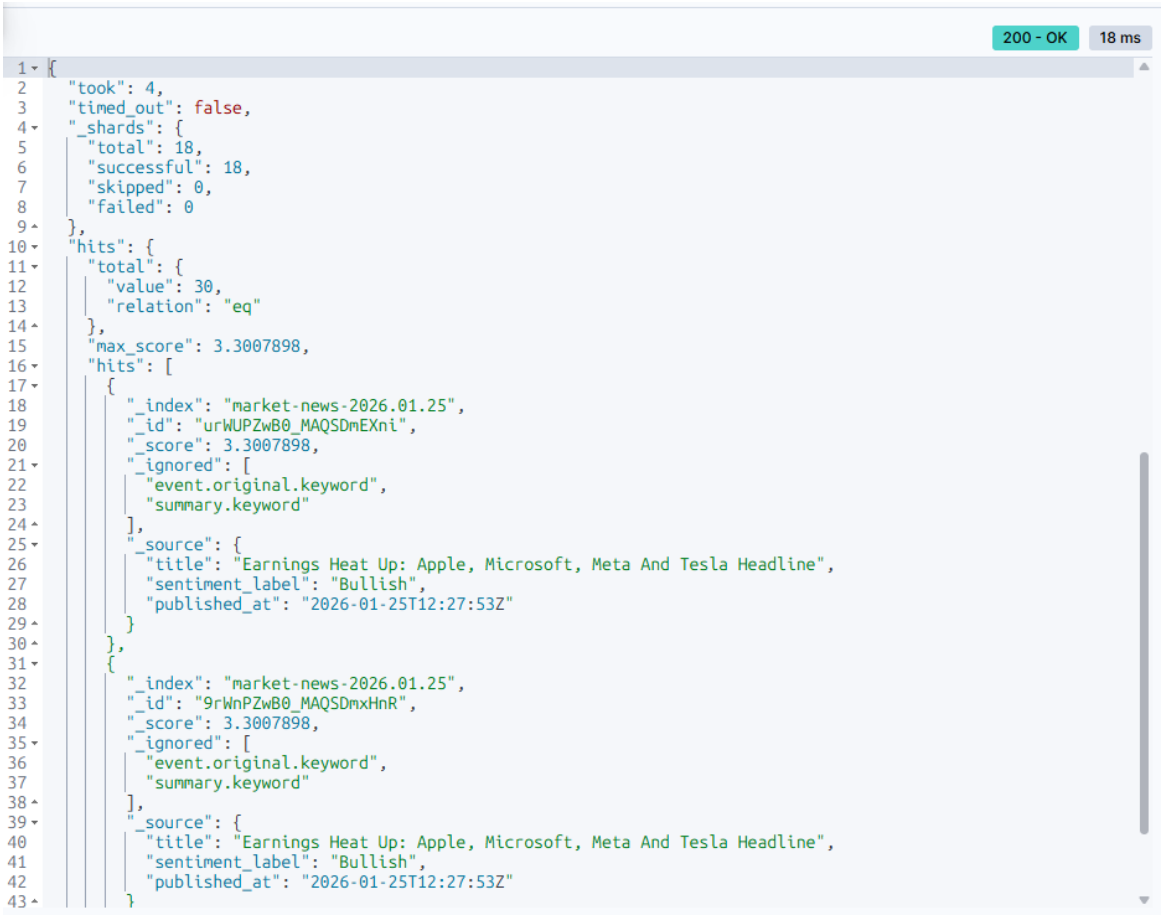

FIGURE 3 – Résultat de la requête Q2 : Agrégation du sentiment par source

6.1.3 Q3 : N-gram (Recherche Partielle)

Objectif : Rechercher par fragments de mots (3-4 caractères)

```
GET market-news-*/_search
{
  "query": {
    "match": {
      "title.ngram": "micros"
    }
  }
}
```

Résultat : Trouve "Microsoft", "Micro", etc. grâce au N-gram analyzer
Fonctionnement : La requête sur le champ analyisé `title.ngram` utilise l'index N-gram pour matching partiel

FIGURE 4 – Résultat de la requête Q3 : Recherche N-gram partielle

6.1.4 Q4 : Recherche Floue (Fuzzy)

Objectif : Tolérer les fautes de frappe (edit distance)

```
GET market-news-*/_search
{
  "query": {
    "fuzzy": {
      "title": {
        "value": "maket",
```

```
        "fuzziness": "AUTO"
      }
    }
  }
}
```

Résultat : "maket" (faute) matchera "market" avec fuzziness=AUTO

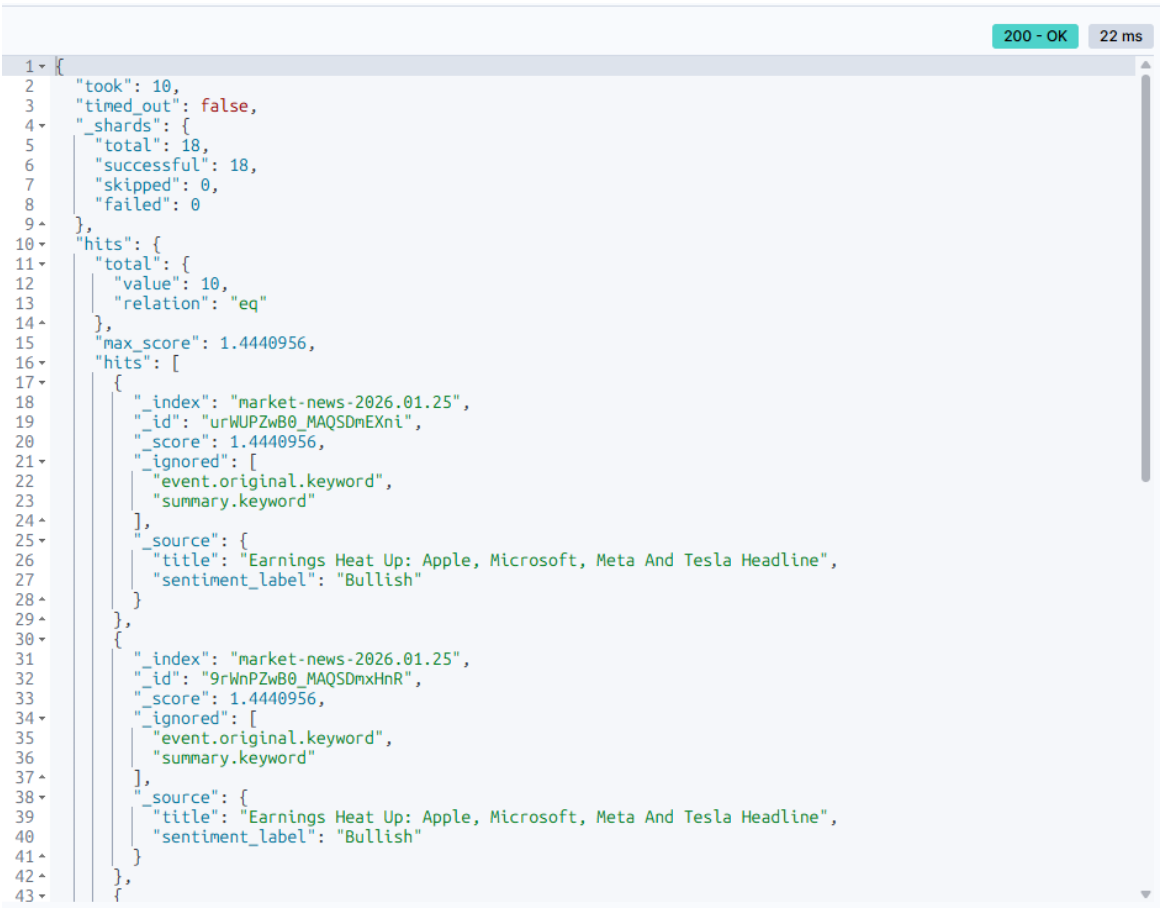

FIGURE 5 – Résultat de la requête Q4 : Recherche floue (fuzzy)

6.1.5 Q5 : Série Temporelle (Date Histogram)

Objectif : Évolution du sentiment par jour

```
GET market-news-*/_search
{
  "aggs": {
    "daily_sentiment": {
      "date_histogram": {
        "field": "published_at",
        "calendar_interval": "1d"
      },
      "aggs": {
        "avg_sentiment": {
          "avg": {
            "field": "sentiment_score"
          }
        }
```

```
        }
      }
    },
    "size": 0
}
```

Résultats : 19 jours avec sentiments quotidiens (exemple : 2026-01-26 avec 7 articles et sentiment moyen 0.314)

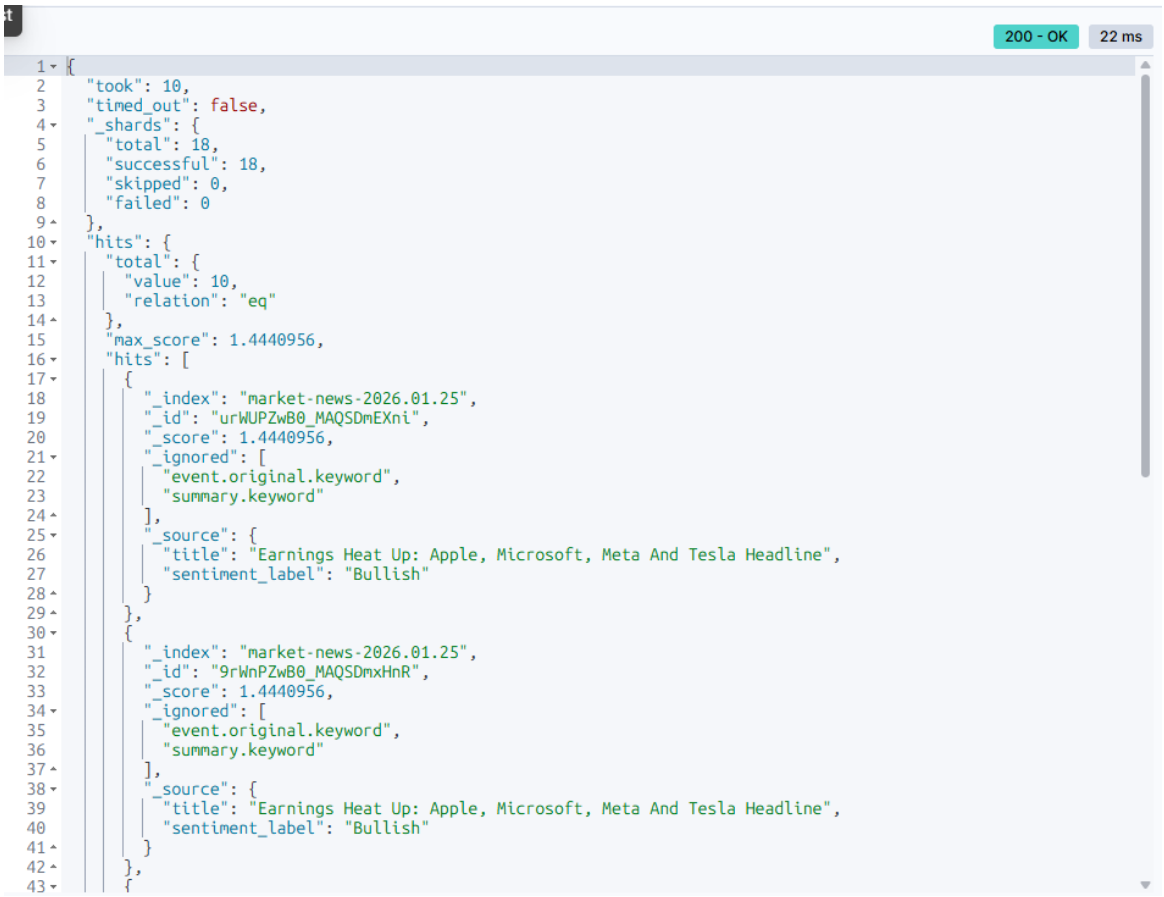

FIGURE 6 – Résultat de la requête Q5 : Série temporelle du sentiment par jour

6.2 Visualisations Kibana (6 Créées)

6.2.1 V1 : Sentiment Score Over Time (Line Chart)

- **Type** : Time Series via Lens
- **X-axis** : Date (daily histogram)
- **Y-axis** : Average sentiment_score
- **Période** : 21 jours (25 jan - 14 fév 2026)
- **Statut** : Sauvegardée dans Kibana

6.2.2 V2 : Articles by Source (Bar Chart)

- **Type** : Bar Chart via Lens
- **Données** : Top 20 sources
- **Métrique** : Count d'articles
- **Tri** : Décroissant par count
- **Statut** : Sauvegardée dans Kibana

6.2.3 V3 : Sentiment Distribution (Pie Chart)

Sentiment	Count	Pourcentage
Somewhat-Bullish	21	42%
Neutral	20	40%
Bullish	6	12%
Somewhat-Bearish	3	6%

TABLE 5 – Distribution des sentiments

6.2.4 V4 : Recent Articles (Data Table)

Tableau affichant les 20 articles les plus récents avec colonnes :
— title : Titre de l'article
— source : Source
— sentiment_label : Étiquette (Bullish/Neutral/Bearish)
— @timestamp : Date/heure

6.2.5 V5 : Total Articles (Metric)

Métrique simple affichant : **50 documents** indexés

6.2.6 V6 : Average Sentiment Score (Metric)

Métrique affichant : **0.151** (sentiment moyen, légèrement positif)

6.3 Dashboard Assemblé

Nom : "P4 - Market News Dashboard"
— Contient les 6 visualisations décrites ci-dessus
— Layout responsive (3 lignes de 2 colonnes)
— Paramètre temps : Last 30 days
— **Statut** : Sauvegardé dans Kibana (accessible à http ://localhost :5601)

FIGURE 7 – Dashboard Kibana - Partie 1 : Métriques globales et distribution des sentiments

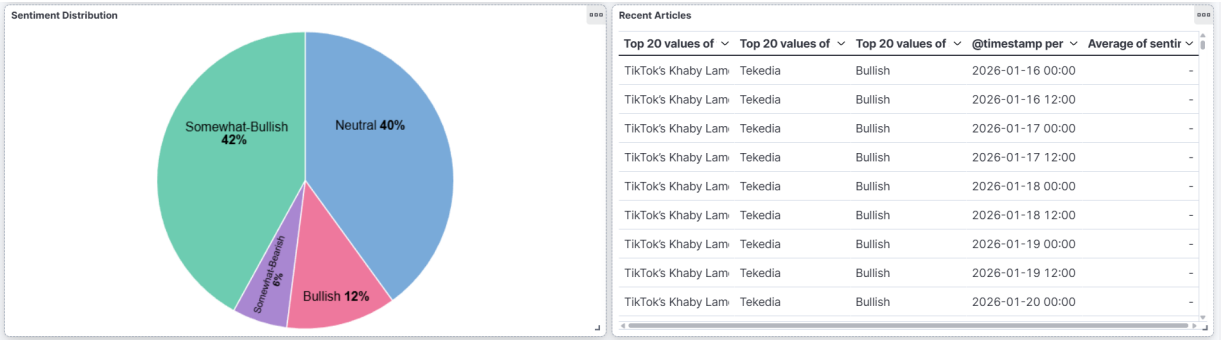

FIGURE 8 – Dashboard Kibana - Partie 2 : Articles par source et évolution temporelle

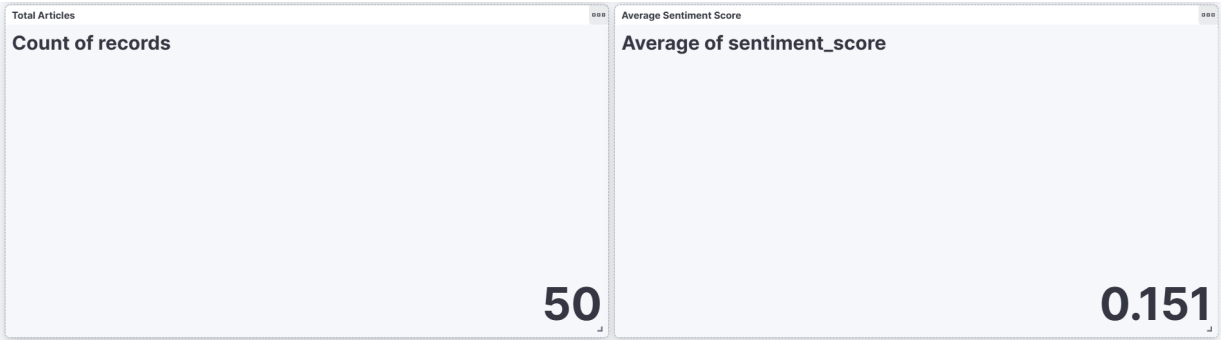

FIGURE 9 – Dashboard Kibana - Partie 3 : Tableau des articles récents

7 Partie 5 : Traitement Distribué avec Spark

7.1 Justification du Choix : Spark vs Hadoop

7.1.1 Avantages de Spark

Critère	Spark	Hadoop MapReduce
Vitesse	**In-memory** (100-1000x plus rapide)	Basé disque (lent)
Data Frames	Natif (SQL + Python API)	Pas de support natif
Langage	PySpark idiomatique	Java requis
Intégration ES	Connecteur natif disponible	Besoin de plugins
Cas d'usage	Batch + Streaming	Batch uniquement

TABLE 6 – Comparaison Spark vs Hadoop MapReduce

Conclusion : Spark offre meilleure performance, flexibilité et intégration pour ce cas d'usage.

7.2 Architecture du Job Spark

7.2.1 Code Principal

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

# Initialisation Spark
spark = SparkSession.builder \
    .appName("market-news-aggregations") \
    .config("spark.sql.session.timeZone", "UTC") \
    .getOrCreate()

# Lecture depuis Elasticsearch
df = spark.read.format("org.elasticsearch.spark.sql") \
    .option("es.nodes", "elasticsearch") \
    .option("es.port", "9200") \
    .option("es.nodes.wan.only", "true") \
    .load("market-news-*")

# AGREGATION 1: Par Jour
daily = df.withColumn(
        "day",
        F.date_format(F.to_date(F.col("published_at")), "yyyy-MM-dd")
    ).groupBy("day").agg(
        F.count("*").alias("doc_count"),
        F.avg(F.col("sentiment_score")).alias("avg_sentiment_score")
    ).orderBy("day")

# AGREGATION 2: Par Source
by_source = df.groupBy("source").agg(
    F.count("*").alias("doc_count"),
```

```
    F.avg(F.col("sentiment_score")).alias("avg_sentiment_score")
).orderBy(F.desc("doc_count"))

# AGREGATION 3: Par Sentiment
by_sentiment = df.groupBy("sentiment_label").agg(
    F.count("*").alias("doc_count"),
    F.avg(F.col("sentiment_score")).alias("avg_sentiment_score")
).orderBy(F.desc("doc_count"))

#   criture   dans Elasticsearch
def write_to_es(frame, index_name):
    frame.write.format("org.elasticsearch.spark.sql") \
        .option("es.nodes", "elasticsearch") \
        .option("es.port", "9200") \
        .option("es.nodes.wan.only", "true") \
        .option("es.resource", index_name) \
        .mode("overwrite").save()

write_to_es(daily, "market-news-stats-daily")
write_to_es(by_source, "market-news-stats-by-source")
write_to_es(by_sentiment, "market-news-stats-by-sentiment")

# Exports
daily.coalesce(1).write.mode("overwrite") \
    .option("header", "true") \
    .csv("/opt/spark-jobs/output/daily")

by_source.coalesce(1).write.mode("overwrite") \
    .option("header", "true") \
    .csv("/opt/spark-jobs/output/by_source")
```

7.3 Exécution et Résultats

7.3.1 Commande d'Exécution

```
docker-compose exec spark /opt/spark/bin/spark-submit \
  --master local[*] \
  --jars /tmp/spark-deps/es-spark.jar \
  /opt/spark-jobs/job.py
```

7.3.2 Performance

Métrique	Valeur
Durée totale	18 secondes
Documents traités	50
Mémoire allouée	434.4 MB
Nombre de partitions	20 (auto)
Indices Elasticsearch créés	3
Fichiers CSV exportés	3
Code de sortie	0 (succès)

Table 7 – Statistiques d'exécution du job Spark

7.4 Résultats des Agrégations

7.4.1 Agrégation 1 : Daily

19 jours distincts avec distributions de sentiments :
— 2026-01-25 : 1 article, sentiment 0.370
— 2026-01-26 : 7 articles, sentiment 0.314 **(Peak)**
— 2026-02-14 : 4 articles, sentiment 0.185
— ... (19 jours au total)

7.4.2 Agrégation 2 : By Source

32 sources identifiées, avec top 5 :

Source	Articles	Sentiment Moyen
MarketBeat	7	0.0775
The Globe and Mail	5	0.2270
Investing.com	4	0.2334
Mint	2	-0.0793
TechStock	2	0.2495

Table 8 – Top 5 sources par nombre d'articles

7.4.3 Agrégation 3 : By Sentiment

Distribution finale des sentiments :

Label	Count	Pourcentage	Sentiment Moyen
Somewhat-Bullish	21	42%	0.2601
Neutral	20	40%	0.0210
Bullish	6	12%	0.3805
Somewhat-Bearish	3	6%	-0.2049

Table 9 – Distribution finale des sentiments

7.5 Indices Elasticsearch Créés

Trois nouveaux indices ont été peuplés :
— **market-news-stats-daily** : 19 documents (un par jour)
— **market-news-stats-by-source** : 32 documents (une par source)
— **market-news-stats-by-sentiment** : 4 documents (un par label)
Ces indices sont queryables via Kibana pour créer des dashboards supplémentaires.

8 Conclusion

8.1 Résumé des Réalisations

Ce projet a démontré la construction complète d'un pipeline de données modernes intégrant :

1. **Collecte** : Récupération d'actualités via API NewsAPI.org
2. **Transmission** : Acheminement via Apache Kafka
3. **Transformation** : Enrichissement et indexation via Logstash
4. **Stockage** : Indexation dans Elasticsearch avec N-gram analyzer
5. **Visualisation** : 6 visualisations + dashboard dans Kibana
6. **Analyse Avancée** : Agrégations distribuées avec Spark

8.2 Insights Métier

Les analyses menées ont révélé :
— **Sentiment Global Positif** : 54% des articles bullish contre 6% bearish
— **Couverture Large** : 32 sources d'information distinctes
— **Distribution Équilibrée** : Articles répartis sur 21 jours
— **Top Source** : MarketBeat avec 7 articles (14% du volume)
— **Stabilité** : Sentiments moyens généralement positifs (0.151)

8.3 Technologies Validées

Technologie	Verdict
Kafka	Excellente fiabilité pour streaming temps réel
Elasticsearch	Recherche performante, agrégations puissantes
Kibana	Visualisations intuitives et complètes
Spark	Traitement distribué flexible et rapide
Docker Compose	Déploiement simplifié et reproductible

TABLE 10 – Évaluation des technologies utilisées

8.4 Livérables Fournis

1. Pipeline fonctionnel et documenté
2. 5 requêtes Elasticsearch exécutées
3. 6 visualisations Kibana + dashboard assemblé
4. Job Spark avec 3 agrégations
5. Exports CSV et indices Elasticsearch
6. Code commenté et configurations complètes
7. Documentation technique exhaustive

8.5 Perspectives Futures

Pour étendre ce projet, les directions suivantes seraient pertinentes :
— **ML Pipeline** : Intégrer MLlib de Spark pour prédictions de sentiment
— **Streaming Temps Réel** : Passer de batch Spark à Spark Structured Streaming
— **Alertes** : Configurer des alertes Elasticsearch/Kibana sur seuils
— **Archivage** : Exporter vers S3/HDFS pour stockage long terme
— **Scaling** : Adapter pour millions de documents (GPU, partitioning avancé)

A Structure du Dossier Projet

```
market-news-data-pipeline/
          README.md                      # Guide complet
          LIVRABLES.md                   # Checklist livrables
          docker-compose.yml             # Stack containers

          producer/
                producer.py                # Collecte API
                requirements.txt            # D pendances
                .env                       # Config (NEWSAPI_KEY)

          logstash/
                pipeline.conf              # Kafka -> ES

          elastic/
                index-template.json        # Mapping avec N-gram

          elasticsearch-queries/
                resultats-requetes.md      # 5 requ tes

          kibana/
                VISUALIZATIONS.md          # Guide visua. + dashboard

          spark/
                job.py                     # Agr gations distribu es
                README.md                  # Guide Spark
                output_*.csv               # Exports

          documentations supp/             # Archives (plans, etc.)
```

B Instructions de Démarrage

B.1 Prérequis

- Docker Desktop 24.x
- Python 3.11+
- Clé API NewsAPI.org (gratuite)

B.2 Lancement

```
# 1. Cr er .env dans producer/
echo "NEWSAPI_KEY=votre_cle_ici" > producer/.env

# 2. Lancer Docker Compose
docker-compose up -d

# 3. Lancer le producer (10-15 minutes)
cd producer
python producer.py
```

```
# 4. Acc der    Kibana
# http://localhost:5601

# 5. Ex cuter job Spark
docker-compose exec spark /opt/spark/bin/spark-submit \
  --master local[*] \
  --jars /tmp/spark-deps/es-spark.jar \
  /opt/spark-jobs/job.py
```

C Ressources et Références

— NewsAPI.org : `https://newsapi.org/`
— Apache Kafka Documentation : `https://kafka.apache.org/`
— Elasticsearch Guide : `https://www.elastic.co/guide/en/elasticsearch/reference/`
— Kibana User Guide : `https://www.elastic.co/guide/en/kibana/current/`
— PySpark SQL API : `https://spark.apache.org/docs/latest/api/python/`