

Projet NoSQL : Kafka + Elasticsearch + Kibana

Un pipeline de données véhicules connectés

Amine, Samy, Mehdi
Étudiants BDML, EFREI

3 avril 2025

Table des matières

1	Introduction	2
2	Installation	4
2.1	Prérequis	4
2.2	Étapes d'installation et de configuration	4
2.2.1	Captures d'écran (Installation)	5
3	Création et manipulation des données (CRUD)	6
3.1	Mise en place du producteur de données	6
3.2	Consommation et insertion dans Elasticsearch	7
3.3	CRUD : Créer, Lire, Mettre à jour, Supprimer	8
3.3.1	Créer (Create)	8
3.3.2	Lire (Read)	8
3.3.3	Mettre à jour (Update)	8
3.3.4	Supprimer (Delete)	8
4	Fonctionnalités avancées : agrégations et visualisations	9
4.1	Exemple de dashboard	9
5	Fonctionnalités avancées	10
5.1	Fonctionnalités avancées de Kafka + Elasticsearch + Kibana	10
5.1.1	Kafka – Ingestion temps réel hautement scalable	10
5.1.2	Elasticsearch – Indexation et recherche rapide de documents	10
5.1.3	Kibana – Visualisation des données indexées	10
5.1.4	Couplage Kafka + Elasticsearch – Pipeline temps réel	11
5.1.5	Autres atouts techniques	11
5.2	Fonctionnalités implémentées	11
5.2.1	1. Simulation multi-véhicules avec logique de parcours réaliste	11
5.2.2	2. Partitionnement Kafka pour la scalabilité	12
5.2.3	3. Alertes en temps réel sur l'état de la batterie	12
5.2.4	4. Indexation enrichie avec champ <code>geo_point</code>	12
5.2.5	5. Requêtes analytiques avancées dans Kibana	12
5.2.6	Conclusion intermédiaire	16
6	Conclusion et recommandations	17
6.1	Avantages de notre pipeline	17
6.2	Limites	17
6.3	Comparaison succincte	17
6.4	Analyse du pipeline selon le théorème CAP	17
6.5	Perspectives	18
6.6	Mot de la fin	18

Chapitre 1

Introduction

Dans le cadre de notre projet NoSQL, nous avons choisi de travailler sur un écosystème combinant plusieurs technologies :

- **Apache Kafka** : une plateforme de streaming distribuée, open source, initialement développée par LinkedIn puis gérée par la fondation Apache. Elle est principalement écrite en Java et Scala.
- **Elasticsearch** : un moteur de recherche et d'analytics temps réel très performant, développé en Java par la société Elastic. Il est historiquement open source (sous licence Apache), puis a évolué vers la *Elastic License 2.0* (certaines fonctionnalités payantes).
- **Kibana** : un outil de visualisation et d'exploration des données pour Elasticsearch, également proposé par Elastic.

Notre projet consiste à mettre en place un **pipeline de données** où nous simulons des véhicules (positions GPS, vitesse, niveau de batterie, etc.) afin de tester la scalabilité, la résilience et les fonctionnalités de ces technologies. Nous avons choisi ces outils car ils sont très répandus dans le monde de la data (ingestion temps réel, indexation, analytics, monitoring...) et ils couvrent un large panel de cas d'usage : ingestion massive, stockage distribué, recherche full-text, création de dashboards, etc.

Comparaison avec d'autres NoSQL (MongoDB, Cassandra, Neo4j, Redis) :

Technologie	Modèle	Points forts	Points faibles
MongoDB	Document (JSON)	<ul style="list-style-type: none">— Très flexible— Facile à installer	<ul style="list-style-type: none">— Moins efficace pour la recherche full-text (comparé à Elastic)— Peut nécessiter un schéma plus rigoureux pour les grosses apps

Cassandra	Colonnes distribuées	<ul style="list-style-type: none"> — Très scalable en écriture — Faible latence 	<ul style="list-style-type: none"> — Langage de requête (CQL) plus limité que SQL — Pas idéal pour la recherche textuelle
Neo4j	Graphes	<ul style="list-style-type: none"> — Excellente gestion des relations — Cypher = requêtes graphiques puissantes 	<ul style="list-style-type: none"> — Moins adapté à l'ingestion massive temps réel — Moins efficace pour l'analytics basique
Redis	Clé-Valeur en mémoire	<ul style="list-style-type: none"> — Extrêmement rapide — Bonne pour le cache 	<ul style="list-style-type: none"> — Stockage en RAM = coûts élevés — Pas d'analytics avancé
Kafka + Elasticsearch	<i>(Middleware + Moteur de recherche)</i>	<ul style="list-style-type: none"> — Ingestion temps réel (Kafka) — Recherche/analytics (Elastic) — Visualisation (Kibana) 	<ul style="list-style-type: none"> — Plus complexe à mettre en place (plusieurs services) — Infrastructure distribuée (maintenance, monitoring)

Ainsi, Kafka + Elasticsearch + Kibana présentent un excellent compromis entre ingestion temps réel et analyses rapides, notamment sur des volumes importants de données.

Chapitre 2

Installation

2.1 Prérequis

- Docker Desktop (ou Docker Engine) installé sur la machine.
- Python 3 (pour simuler les données).

2.2 Étapes d'installation et de configuration

Nous avons créé un fichier `docker-compose.yml` permettant de lancer :

- **Zookeeper** (nécessaire pour Kafka)
- **Kafka**
- **Elasticsearch**
- **Kibana**

Voici le contenu principal (résumé) :

Listing 2.1 – Extrait du `docker-compose.yml`

```
version: '3.8'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    ...

  kafka:
    image: confluentinc/cp-kafka:7.5.0
    ...

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.11.1
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
    ports:
      - "9200:9200"
    networks:
      - elk

  kibana:
    image: docker.elastic.co/kibana/kibana:8.11.1
    environment:
```

```

    - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
ports:
    - "5601:5601"
networks:
    - elk

```

networks:
elk:

Une fois ce fichier en place, on lance la commande : `docker-compose up -d`

2.2.1 Captures d'écran (Installation)

Nous avons réalisé des captures d'écran pendant l'installation. Ci-dessous, **nous incluons les images sans réécrire leur contenu**, conformément aux consignes :

```

• (base) aminemzali@MacBook-Pro-de-Sylvain ProjetNoSQL % mkdir kafka-vehicle-project
cd kafka-vehicle-project
touch docker-compose.yml
• (base) aminemzali@MacBook-Pro-de-Sylvain kafka-vehicle-project % docker-compose up -d
WARN[0000] /Users/aminemzali/Documents/Efrei/S8/ProjetNoSQL/kafka-vehicle-project/docker-compose.yml: the attribute `version` is obsolete, it will
be ignored, please remove it to avoid potential confusion
[+] Running 39/39
  ✓ elasticsearch Pulled          91.2s
  ✓ zookeeper Pulled             101.9s
  ✓ kibana Pulled                 69.3s
  ✓ kafka Pulled                  101.9s
[+] Running 6/6
  ✓ Network kafka-vehicle-project_elk Created      0.1s
  ✓ Network kafka-vehicle-project_default Created  0.1s
  ✓ Container zookeeper Started                   0.4s
  ✓ Container elasticsearch Started                0.4s
  ✓ Container kibana Started                      0.4s
  ✓ Container kafka Started                       0.4s
○ (base) aminemzali@MacBook-Pro-de-Sylvain kafka-vehicle-project %

```

FIGURE 2.1 – Création du dossier `kafka-vehicle-project` et exécution de `docker-compose up -d`

```

• (base) aminemzali@MacBook-Pro-de-Sylvain kafka-vehicle-project % mkdir producer
cd producer
touch simulate_vehicles.py
• (base) aminemzali@MacBook-Pro-de-Sylvain producer % python3 -m venv venv
source venv/bin/activate
pip install kafka-python faker
Collecting kafka-python
  Downloading kafka_python-2.1.4-py2.py3-none-any.whl (276 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 276.6/276.6 kB 4.1 MB/s eta 0:00:00
Collecting faker
  Downloading faker-37.1.0-py3-none-any.whl (1.9 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 12.0 MB/s eta 0:00:00
Collecting tzdata
  Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: kafka-python, tzdata, faker
Successfully installed faker-37.1.0 kafka-python-2.1.4 tzdata-2025.2

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
○ (venv) (base) aminemzali@MacBook-Pro-de-Sylvain producer %

```

FIGURE 2.2 – Installation de Python `kafka-python` et `faker`

Chapitre 3

Création et manipulation des données (CRUD)

3.1 Mise en place du producteur de données

Pour simuler des données de véhicules, nous avons un script `simulate_vehicles.py` qui envoie des messages vers Kafka.

— **Création du topic :**

```
docker exec -it kafka \
  kafka-topics --create --topic vehicle-data \
  --bootstrap-server localhost:9092 \
  --partitions 1 \
  --replication-factor 1
```

— **Producteur Python :**

Listing 3.1 – Extrait de `simulate_vehicles.py`

```
from kafka import KafkaProducer
from faker import Faker
import json, time, random

fake = Faker()
producer = KafkaProducer(
    bootstrap_servers='localhost:9092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

def generate_vehicle_data(vehicle_id):
    return {
        "vehicle_id": vehicle_id,
        "@timestamp": time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime()),
        "latitude": float(fake.latitude()),
        "longitude": float(fake.longitude()),
        "speed_kmh": round(random.uniform(0, 120), 2),
        "battery_level": random.randint(10, 100)
    }

if __name__ == "__main__":
    vehicle_ids = [f"veh-{i}" for i in range(1, 6)]
```

```

while True:
    vehicle = random.choice(vehicle_ids)
    data = generate_vehicle_data(vehicle)
    print(f"Sending: {data}")
    producer.send("vehicle-data", data)
    time.sleep(1)

```

Quelques extraits de la console :

```

el': 57}
Sending: {'vehicle_id': 'veh-4', 'timestamp': 1743664644.611537, 'latitude': 34.305018, 'longitude': 143.505552, 'speed_kmh': 59.36, 'battery_level': 99}
Sending: {'vehicle_id': 'veh-2', 'timestamp': 1743664645.61519, 'latitude': -25.614558, 'longitude': 89.926027, 'speed_kmh': 39.91, 'battery_level': 93}
Sending: {'vehicle_id': 'veh-4', 'timestamp': 1743664646.619885, 'latitude': 6.4787735, 'longitude': 177.222666, 'speed_kmh': 3.92, 'battery_level': 79}
Sending: {'vehicle_id': 'veh-4', 'timestamp': 1743664647.6234848, 'latitude': -53.427157, 'longitude': 73.846512, 'speed_kmh': 104.39, 'battery_level': 43}
Sending: {'vehicle_id': 'veh-4', 'timestamp': 1743664648.6288981, 'latitude': 39.9791005, 'longitude': -22.054631, 'speed_kmh': 15.1, 'battery_level': 75}
Sending: {'vehicle_id': 'veh-5', 'timestamp': 1743664649.631018, 'latitude': 49.169727, 'longitude': 28.261668, 'speed_kmh': 85.02, 'battery_level': 32}
Sending: {'vehicle_id': 'veh-2', 'timestamp': 1743664650.6359448, 'latitude': -72.305269, 'longitude': 159.576207, 'speed_kmh': 20.96, 'battery_level': 87}
Sending: {'vehicle_id': 'veh-5', 'timestamp': 1743664651.6398091, 'latitude': -41.6065435, 'longitude': -2.282485, 'speed_kmh': 47.29, 'battery_level': 57}

```

FIGURE 3.1 – Console du producteur : envoi des données simulées

3.2 Consommation et insertion dans Elasticsearch

Pour lire les données depuis Kafka et les envoyer à Elasticsearch, nous utilisons un `consumer_to_es.py` :

Listing 3.2 – Extrait de `consumer_to_es.py`

```

from kafka import KafkaConsumer
from elasticsearch import Elasticsearch
import json

# Connexion Elasticsearch
es = Elasticsearch("http://localhost:9200")

# Connexion au topic Kafka
consumer = KafkaConsumer(
    "vehicle-data",
    bootstrap_servers="localhost:9092",
    value_deserializer=lambda v: json.loads(v.decode("utf-8")),
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id="vehicle-consumer-group"
)

for message in consumer:
    data = message.value
    print(f"Received: {data}")
    # Indexation
    es.index(index="vehicle-data", document=data)

```



```
(venv) (base) aminemzali@MacBook-Pro-de-Sylvain producer % python consumer_to_es.py
Received: {'vehicle_id': 'veh-1', '@timestamp': '2025-04-03T07:31:50Z', 'latitude': -44.7262915, 'longitude': -1
58.483988, 'speed_kmh': 3.59, 'battery_level': 13}
Received: {'vehicle_id': 'veh-2', '@timestamp': '2025-04-03T07:31:51Z', 'latitude': -47.789725, 'longitude': 116
.624331, 'speed_kmh': 20.68, 'battery_level': 50}
Received: {'vehicle_id': 'veh-4', '@timestamp': '2025-04-03T07:31:52Z', 'latitude': 58.216725, 'longitude': -142
.907469, 'speed_kmh': 35.53, 'battery_level': 10}
Received: {'vehicle_id': 'veh-4', '@timestamp': '2025-04-03T07:31:53Z', 'latitude': 59.4942935, 'longitude': 83.
211519, 'speed_kmh': 17.69, 'battery_level': 69}
Received: {'vehicle_id': 'veh-4', '@timestamp': '2025-04-03T07:31:54Z', 'latitude': 75.139365, 'longitude': 64.2
12467, 'speed_kmh': 39.71, 'battery_level': 75}
Received: {'vehicle_id': 'veh-5', '@timestamp': '2025-04-03T07:31:55Z', 'latitude': -15.007959, 'longitude': 78.
296096, 'speed_kmh': 94.04, 'battery_level': 86}
Received: {'vehicle_id': 'veh-1', '@timestamp': '2025-04-03T07:31:56Z', 'latitude': 31.4283075, 'longitude': 35.
395944, 'speed_kmh': 73.6, 'battery_level': 10}
Received: {'vehicle_id': 'veh-4', '@timestamp': '2025-04-03T07:31:57Z', 'latitude': -81.028558, 'longitude': 16.
888941, 'speed_kmh': 86.07, 'battery_level': 96}
Received: {'vehicle_id': 'veh-5', '@timestamp': '2025-04-03T07:31:58Z', 'latitude': -74.770449, 'longitude': -40
.474780, 'speed_kmh': 96.00, 'battery_level': 60}
```

FIGURE 3.2 – Console du consommateur : réception et indexation dans Elasticsearch

3.3 CRUD : Créer, Lire, Mettre à jour, Supprimer

3.3.1 Créer (Create)

L'instruction `es.index(index="vehicle-data", document=data)` permet de *créer* automatiquement un document dans l'index `vehicle-data`. Si l'index n'existe pas, il est créé dynamiquement.

3.3.2 Lire (Read)

Nous pouvons lire les données via Kibana (section “Discover”) ou via une requête REST :

```
curl -X GET http://localhost:9200/vehicle-data/_search?pretty
```

3.3.3 Mettre à jour (Update)

Pour modifier un document précis (connaissant son `_id`) :

```
curl -X POST "localhost:9200/vehicle-data/_update/<document_id>" \
-H 'Content-Type: application/json' -d '{
  "doc": {
    "speed_kmh": 42.0
  }
}'
```

3.3.4 Supprimer (Delete)

```
curl -X DELETE "localhost:9200/vehicle-data/_doc/<document_id>"
```

Ou pour supprimer l'index complet :

```
curl -X DELETE http://localhost:9200/vehicle-data
```

Chapitre 4

Fonctionnalités avancées : agrégations et visualisations

Nous utilisons Kibana pour créer des **Index Patterns** (Data Views) et des **dashboards**. Une fois que **vehicle-data** est créé et que le champ **@timestamp** est reconnu comme date, nous pouvons :

- Explorer les données dans Kibana (Discover)
- Créer des visualisations (histogramme, pie chart, bar chart, etc.)

4.1 Exemple de dashboard

Nous avons créé trois visualisations :

1. Un **histogramme** du nombre de messages par tranche de temps
2. Un **pie chart** pour la répartition par **vehicle_id**
3. Un **bar chart** de la vitesse moyenne par véhicule

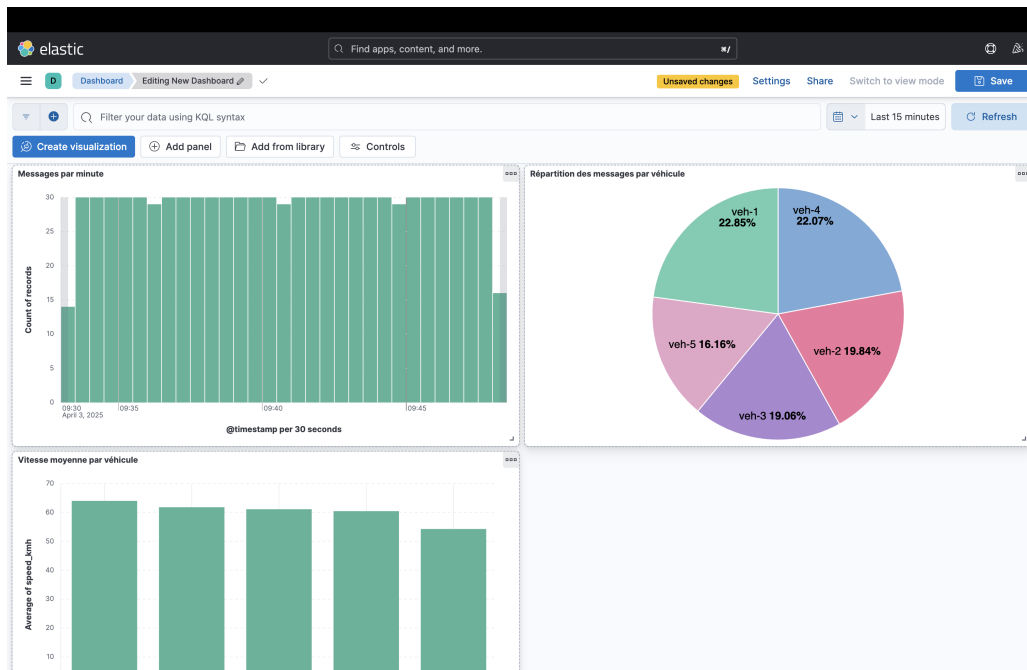


FIGURE 4.1 – Dashboard dans Kibana : histogramme, pie chart et bar chart

Chapitre 5

Fonctionnalités avancées

5.1 Fonctionnalités avancées de Kafka + Elasticsearch + Kibana

Notre pipeline présente plusieurs **fonctionnalités avancées** et **atouts différenciateurs** qui répondent aux enjeux de scalabilité, résilience, ingestion temps réel et visualisation.

5.1.1 Kafka – Ingestion temps réel hautement scalable

- **Partitionnement automatique** : Kafka répartit les messages sur plusieurs partitions pour permettre un traitement parallèle.
- **Scalabilité horizontale** : il est possible d'ajouter des brokers pour augmenter la capacité.
- **Durabilité et persistance** : les messages sont stockés sur disque et peuvent être relus plusieurs fois.
- **Tolérance aux pannes** : via la réplication des partitions entre plusieurs brokers.
- **Gestion fine des offsets** : permet à chaque consommateur de reprendre exactement là où il s'était arrêté.
- **Kafka Streams** (optionnel) : pour traiter les flux de manière distribuée avec des transformations en temps réel.

Avantage : Kafka est particulièrement adapté aux systèmes distribués modernes où la *latence faible* et la *résilience* sont critiques. Il est plus performant que Redis ou MongoDB pour des cas d'usage liés à l'ingestion continue de données.

5.1.2 Elasticsearch – Indexation et recherche rapide de documents

- **Indexation full-text** avec analyse linguistique (stopwords, racinisation, etc.)
- **Recherches booléennes complexes** (avec AND, OR, NOT sur plusieurs champs)
- **Scores de pertinence (TF-IDF)** pour classer les résultats
- **Mapping de types** : définition fine de chaque champ (date, float, keyword, etc.)
- **Geo-queries** : support natif des données géographiques (type `geo_point`)
- **Alias d'index** : pour gérer des vues logiques sur plusieurs index
- **Scripts dans les requêtes** : pour des agrégations ou filtres dynamiques
- **Monitoring des performances par shard**

Avantage : Contrairement à MongoDB ou Cassandra, Elasticsearch est spécialisé dans la recherche rapide et le tri intelligent sur de gros volumes de données. Il permet aussi des requêtes analytiques avancées très rapidement, même en temps réel.

5.1.3 Kibana – Visualisation des données indexées

- **Dashboards dynamiques** et personnalisables
- **Visualisations riches** : histogrammes, cartes, courbes, bar charts, gauges, pie charts, etc.

- **Filtres interactifs** et KQL (Kibana Query Language)
- **Time series explorer** pour l'analyse de données temporelles
- **Drill-down et zoom dans les données**
- **Alertes et visualisation de logs (Stack ELK)**

Avantage : Kibana offre une interface clé en main pour que même les non-techniques puissent explorer et visualiser les données. C'est un plus considérable pour l'aide à la décision.

5.1.4 Couplage Kafka + Elasticsearch – Pipeline temps réel

- Traitement de données en flux (*stream*) → insertion immédiate dans un moteur de recherche.
- Possibilité de créer des **pipelines temps réel** sur plusieurs services.
- Intégration naturelle avec des dashboards pour faire du **real-time analytics**.
- Ajout possible de pré-traitement ou enrichissement avant l'indexation (dans le consumer Python).

5.1.5 Autres atouts techniques

- **Schéma dynamique** : Elasticsearch accepte des documents JSON même sans schéma préalable.
- **Index multishards** : Elastic répartit automatiquement l'index sur plusieurs shards pour scalabilité.
- **Recherche multi-index** : Une seule requête peut balayer plusieurs index.
- **Sécurité (activable)** : Authentification, SSL, rôles utilisateurs (si xpack activé).
- **Requêtes distribuées** : Elastic répartit la charge des requêtes sur plusieurs nœuds.
- **Compression + cache** : Gestion efficace de la mémoire et du cache de requêtes.

Conclusion de la section

Le combo **Kafka + Elasticsearch + Kibana** nous permet de couvrir des cas très puissants qu'aucune autre technologie NoSQL vue en TP ne couvre aussi bien à elle seule :

- ingestion en masse + traitement distribué (Kafka),
- indexation rapide et recherche avancée (Elastic),
- dashboards dynamiques et explorables (Kibana).

C'est une solution moderne, scalable, visuelle, bien adaptée aux cas industriels (IoT, flotte de véhicules, logs serveurs, données capteurs...).

5.2 Fonctionnalités implémentées

Dans cette section, nous présentons les améliorations techniques et fonctionnelles apportées à notre projet basé sur Kafka, Elasticsearch et Kibana. Ces fonctionnalités démontrent la scalabilité, la souplesse et la puissance d'analyse de notre stack NoSQL distribuée.

5.2.1 1. Simulation multi-véhicules avec logique de parcours réaliste

Afin de dépasser les simples données aléatoires, nous avons mis en place un simulateur capable d'émuler plusieurs types de véhicules (**car**, **bus**, **truck**), chacun suivant un parcours distinct :

- **Paris** pour **veh-42** (voiture)
- **Lyon** pour **bus-7**
- **Marseille** pour **truck-21**

Chaque véhicule suit un tracé cohérent avec une vitesse réaliste, des arrêts simulés, et une batterie qui diminue progressivement. Les données sont envoyées dans Kafka avec une **key** définie (**vehicle_id**) pour permettre une bonne distribution sur les partitions.

5.2.2 2. Partitionnement Kafka pour la scalabilité

Nous avons créé un topic Kafka `vehicle-data-v2` avec **3 partitions**, permettant une répartition efficace de la charge.

- Chaque message est dirigé vers une partition via sa clé `vehicle_id`.
- Cela nous permettrait à terme d'implémenter un traitement parallèle avec plusieurs consommateurs.

5.2.3 3. Alertes en temps réel sur l'état de la batterie

Nous avons développé un second consumer Kafka dédié aux alertes de batterie. Celui-ci identifie les véhicules dont le niveau est inférieur à 20%, et affiche une alerte en console.

5.2.4 4. Indexation enrichie avec champ `geo_point`

Pour permettre la visualisation cartographique dans Kibana, nous avons enrichi les documents indexés avec un champ `location` de type `geo_point`. Celui-ci est généré à partir des champs `latitude` et `longitude` lors du traitement dans le consumer.

5.2.5 5. Requêtes analytiques avancées dans Kibana

Nous avons conçu des requêtes complexes combinant plusieurs critères à l'aide du moteur de recherche booléen d'Elasticsearch. Par exemple :

- Identifier tous les camions roulant à plus de 60 km/h avec une batterie < 30%
- Représenter graphiquement la vitesse moyenne par type de véhicule
- Afficher la carte en temps réel avec les localisations

Exemple de requête booléenne

Listing 5.1 – Requête Elasticsearch dans Kibana Dev Tools

```
GET vehicle-data-v2/_search
{
  "query": {
    "bool": {
      "must": [
        { "term": { "type": "truck" } },
        { "range": { "battery_level": { "lt": 30 } } },
        { "range": { "speed_kmh": { "gt": 60 } } }
      ]
    }
  }
}
```

Visualisation des vitesses moyennes

Carte en temps réel

Grâce au champ `geo_point`, Kibana permet une visualisation dynamique des positions GPS sur une carte.

```

● (base) aminemzali@MacBook-Pro-de-Sylvain ProjetNoSQL % cd kafka-vehicle-project
○ (base) aminemzali@MacBook-Pro-de-Sylvain kafka-vehicle-project % cd producer
source venv/bin/activate
python simulate_multi_producers.py
Sending: {'vehicle_id': 'truck-2', 'type': 'truck', '@timestamp': '2025-04-03T08:20:39Z', 'latitude': -88.973596, 'longitude': -91.879162, 'speed_kmh': 63.47, 'battery_level': 32}
Sending: {'vehicle_id': 'scooter-2', 'type': 'scooter', '@timestamp': '2025-04-03T08:20:40Z', 'latitude': -84.1866525, 'longitude': 12.214716, 'speed_kmh': 37.63, 'battery_level': 69}
Sending: {'vehicle_id': 'scooter-2', 'type': 'scooter', '@timestamp': '2025-04-03T08:20:41Z', 'latitude': -43.0737895, 'longitude': 46.689551, 'speed_kmh': 14.74, 'battery_level': 98}
Sending: {'vehicle_id': 'scooter-2', 'type': 'scooter', '@timestamp': '2025-04-03T08:20:42Z', 'latitude': 14.264091, 'longitude': -60.553476, 'speed_kmh': 73.78, 'battery_level': 37}
Sending: {'vehicle_id': 'truck-1', 'type': 'truck', '@timestamp': '2025-04-03T08:20:43Z', 'latitude': 53.5813995, 'longitude': -38.101901, 'speed_kmh': 34.49, 'battery_level': 24}
Sending: {'vehicle_id': 'truck-2', 'type': 'truck', '@timestamp': '2025-04-03T08:20:44Z', 'latitude': 51.9319875, 'longitude': -23.040538, 'speed_kmh': 74.39, 'battery_level': 20}
Sending: {'vehicle_id': 'scooter-1', 'type': 'scooter', '@timestamp': '2025-04-03T08:20:45Z', 'latitude': -3.404106, 'longitude': -49.186769, 'speed_kmh': 115.28, 'battery_level': 57}
Sending: {'vehicle_id': 'veh-3', 'type': 'car', '@timestamp': '2025-04-03T08:20:46Z', 'latitude': -56.5562785, 'longitude': 57.268847, 'speed_kmh': 67.94, 'battery_level': 20}
Sending: {'vehicle_id': 'scooter-1', 'type': 'scooter', '@timestamp': '2025-04-03T08:20:47Z', 'latitude': -25.428905, 'longitude': -11.272847, 'speed_kmh': 93.5, 'battery_level': 91}
Sending: {'vehicle_id': 'truck-2', 'type': 'truck', '@timestamp': '2025-04-03T08:20:48Z', 'latitude': -20.15203, 'longitude': -164.599557, 'speed_kmh': 11.53, 'battery_level': 30}
Sending: {'vehicle_id': 'truck-2', 'type': 'truck', '@timestamp': '2025-04-03T08:20:49Z', 'latitude': 7.6785565, 'longitude': -145.287443, 'speed_kmh': 52.93, 'battery_level': 37}
Sending: {'vehicle_id': 'veh-2', 'type': 'car', '@timestamp': '2025-04-03T08:20:50Z', 'latitude': 81.901126, 'longitude': 143.767085, 'speed_kmh': 110.68, 'battery_level': 53}
"AdministratorAccess-872515283809" (click to change) {'@timestamp': '2025-04-03T08:20:51Z', 'latitude': 4.85391, 'longitude': 143.767085, 'speed_kmh': 110.68, 'battery_level': 57}

```

FIGURE 5.1 – Envoi des données depuis le simulateur multi-véhicules

```

● (venv) (base) aminemzali@MacBook-Pro-de-Sylvain producer % docker exec -it kafka kafka-topics \
--bootstrap-server localhost:9092 \
--create \
--topic vehicle-data-v2 \
--partitions 3 \
--replication-factor 1
Created topic vehicle-data-v2.

What's next:
Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug kafka
Learn more at https://docs.docker.com/go/debug-cli/
● (venv) (base) aminemzali@MacBook-Pro-de-Sylvain producer % docker exec -it kafka kafka-topics \
--bootstrap-server localhost:9092 \
--describe --topic vehicle-data-v2
Topic: vehicle-data-v2 TopicId: FyyN9R9GSv2ibGpFC9Ngcw PartitionCount: 3 ReplicationFactor: 1 Configs:
Topic: vehicle-data-v2 Partition: 0 Leader: 1 Replicas: 1 Isr: 1
Topic: vehicle-data-v2 Partition: 1 Leader: 1 Replicas: 1 Isr: 1
Topic: vehicle-data-v2 Partition: 2 Leader: 1 Replicas: 1 Isr: 1

What's next:
Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug kafka
Learn more at https://docs.docker.com/go/debug-cli/
○ (venv) (base) aminemzali@MacBook-Pro-de-Sylvain producer %

```

FIGURE 5.2 – Création et visualisation du topic `vehicle-data-v2` avec 3 partitions

```

● (base) aminemzali@MacBook-Pro-de-Sylvain kafka-vehicle-project % cd producer
○ (base) aminemzali@MacBook-Pro-de-Sylvain producer % source venv/bin/activate
python battery_alert_consumer.py
⚠ Battery Alert Consumer lancé...
⚠ ALERTE : truck-1 a une batterie faible (19%)
⚠ ALERTE : truck-1 a une batterie faible (16%)
⚠ ALERTE : scooter-1 a une batterie faible (19%)
⚠ ALERTE : veh-2 a une batterie faible (10%)
⚠ ALERTE : scooter-1 a une batterie faible (18%)
⚠ ALERTE : scooter-2 a une batterie faible (15%)
⚠ ALERTE : scooter-1 a une batterie faible (14%)
⚠ ALERTE : scooter-2 a une batterie faible (19%)
⚠ ALERTE : scooter-2 a une batterie faible (15%)
⚠ ALERTE : scooter-2 a une batterie faible (13%)
⚠ ALERTE : veh-2 a une batterie faible (19%)
⚠ ALERTE : scooter-1 a une batterie faible (11%)
⚠ ALERTE : scooter-2 a une batterie faible (15%)
⚠ ALERTE : scooter-1 a une batterie faible (17%)
⚠ ALERTE : veh-2 a une batterie faible (19%)
⚠ ALERTE : veh-2 a une batterie faible (14%)
⚠ ALERTE : veh-1 a une batterie faible (13%)
⚠ ALERTE : veh-2 a une batterie faible (19%)
⚠ ALERTE : veh-2 a une batterie faible (17%)

```

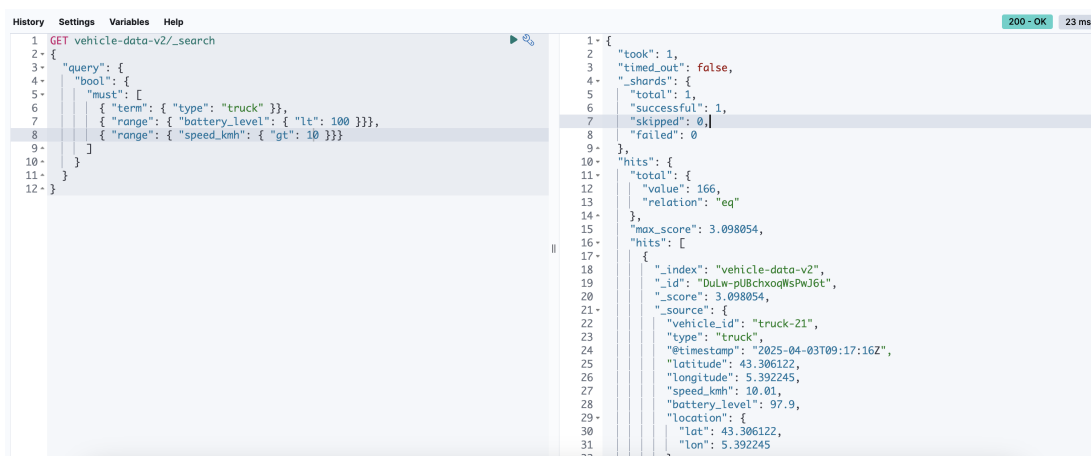
FIGURE 5.3 – Alertes batterie faible reçues par le *battery_alert_consumer.py*

```

(venv) (base) aminemzali@MacBook-Pro-de-Sylvain producer % python create_index_with_mapping.py
Index 'vehicle-data-v2' créé avec mapping personnalisé.

```

FIGURE 5.4 – Index enrichi avec mapping personnalisé (dont *geo_point*)



```

1 GET vehicle-data-v2/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         { "term": { "type": "truck" } },
7         { "range": { "battery_level": { "lt": 100 } } },
8         { "range": { "speed_kmh": { "gt": 10 } } }
9       ]
10     }
11   }
12 }

```

```

1 {
2   "took": 1,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 166,
13      "relation": "eq"
14    },
15    "max_score": 3.098054,
16    "hits": [
17      {
18        "_index": "vehicle-data-v2",
19        "_id": "DuLw-plJ8chxaqWsPwJ6t",
20        "_score": 3.098054,
21        "_source": {
22          "vehicle_id": "truck-21",
23          "type": "truck",
24          "@timestamp": "2025-04-03T09:17:16Z",
25          "latitude": 43.306122,
26          "longitude": 5.392245,
27          "speed_kmh": 10.01,
28          "battery_level": 97.9,
29          "location": {
30            "lat": 43.306122,
31            "lon": 5.392245
32          }
33        }
34      }
35    ]
36  }
37 }

```

FIGURE 5.5 – Résultats d'une requête complexe dans Dev Tools



FIGURE 5.6 – Vitesse moyenne par type de véhicule



FIGURE 5.7 – Carte des véhicules mise à jour en temps réel

5.2.6 Conclusion intermédiaire

Ces fonctionnalités démontrent la richesse et l'évolutivité de notre architecture. Nous avons su exploiter les forces de Kafka (scalabilité, partitionnement), Elasticsearch (recherche rapide, index géo), et Kibana (tableaux de bord en temps réel) pour offrir un projet complet, réutilisable et visuellement convaincant.

Chapitre 6

Conclusion et recommandations

6.1 Avantages de notre pipeline

- **Ingestion temps réel** : Kafka gère un flux continu de données et permet de scaler horizontalement en augmentant le nombre de partitions.
- **Indexation et recherche rapide** : Elasticsearch permet de chercher dans les documents quasi instantanément, et de faire des agrégations complexes.
- **Visualisation ergonomique** : Kibana fournit une interface simple pour créer des dashboards, analyser les données et effectuer des recherches.

6.2 Limites

- L'**infrastructure** est plus lourde que des solutions “simples” (ex. MongoDB). Il faut maintenir Kafka, Elasticsearch, Kibana, Zookeeper.
- Des **coûts d'hébergement** plus élevés dès lors qu'on veut de la haute disponibilité (clusters distribués).

6.3 Comparaison succincte

En comparaison à MongoDB, Cassandra, Neo4j ou Redis, ce pipeline met l'accent sur :

- **Consistance et scalabilité** (Kafka + ES sont distribués)
- **Recherche avancée et analytics temps réel** (Elasticsearch excelle en full-text et agrégations)
- **Résistance aux pannes** via la réplication Kafka et Elasticsearch.

D'autres solutions NoSQL (MongoDB, Redis, etc.) sont parfois plus simples à installer et maintenir, mais moins performantes pour un cas “ingestion massive + requêtes complexes + visualisation temps réel”.

6.4 Analyse du pipeline selon le théorème CAP

Le **théorème CAP**, fondamental dans les systèmes distribués, stipule qu'un système ne peut garantir simultanément les trois propriétés suivantes :

- **C : Consistance** — tous les nœuds voient les mêmes données au même moment.
- **A : Disponibilité** — chaque requête reçoit une réponse (même si ce n'est pas la plus à jour).
- **P : Tolérance au partitionnement** — le système continue à fonctionner même en cas de perte de communication entre les nœuds.

Comportement de notre solution (Kafka + Elasticsearch)

Propriété	Présente ?	Explication
Consistance (C)	Non garantie	Kafka et Elasticsearch préfèrent la disponibilité en cas de partition. Par exemple, un consommateur Kafka peut lire des messages légèrement en retard, et Elasticsearch peut retourner des données partiellement indexées.
Disponibilité (A)	Oui	Kafka garantit la disponibilité via la réplication des partitions, et Elasticsearch accepte les requêtes même en cas de perte partielle de nœuds.
Tolérance aux partitions (P)	Oui	Les deux systèmes continuent à fonctionner même si des nœuds sont temporairement déconnectés, grâce à leurs mécanismes de réplication.

Conclusion CAP : Notre pipeline Kafka → Elasticsearch est un système de type **AP** : il **favorise la Disponibilité et la Tolérance aux partitions** au détriment d'une forte consistance immédiate. Cela le rend idéal pour les systèmes distribués à haut débit comme l'IoT, les logs, ou les flottes de véhicules, où il vaut mieux accepter un léger délai plutôt que de refuser une requête.

6.5 Perspectives

Pour aller plus loin, nous pourrions :

- Ajouter un **Kafka Connect** pour automatiser l'ingestion vers Elasticsearch
- Jouer sur le **nombre de partitions** Kafka et de **shards/réplicas** ES pour mesurer les performances
- Gérer la **géolocalisation** avec un type `geo_point` pour afficher les véhicules sur une carte.

6.6 Mot de la fin

Ce pipeline répond aux besoins d'une entreprise souhaitant *collecter et analyser en temps réel* une grande quantité de données de capteurs ou de logs. Grâce à l'élasticité de Kafka et la puissance d'Elasticsearch, associées à la visualisation de Kibana, nous obtenons une solution performante, évolutive et relativement simple à déployer via Docker.

— Fin du Rapport —