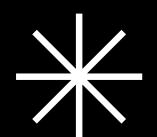


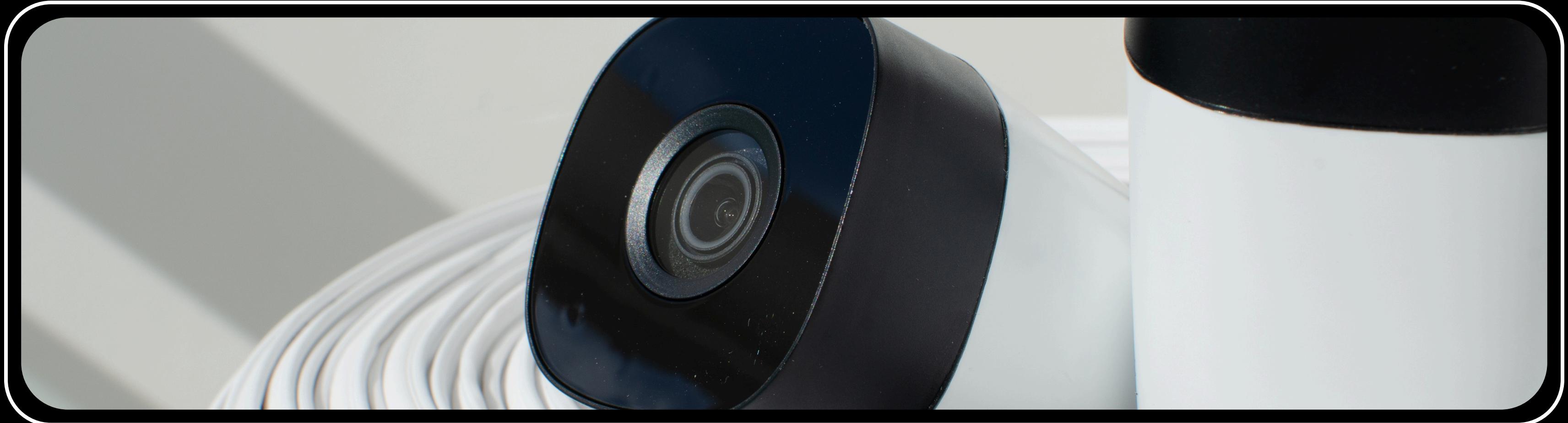


PANACHÉ DE CONCEPTS

BIG DATA & DATA ENGINEERING

MALAK SOULHI



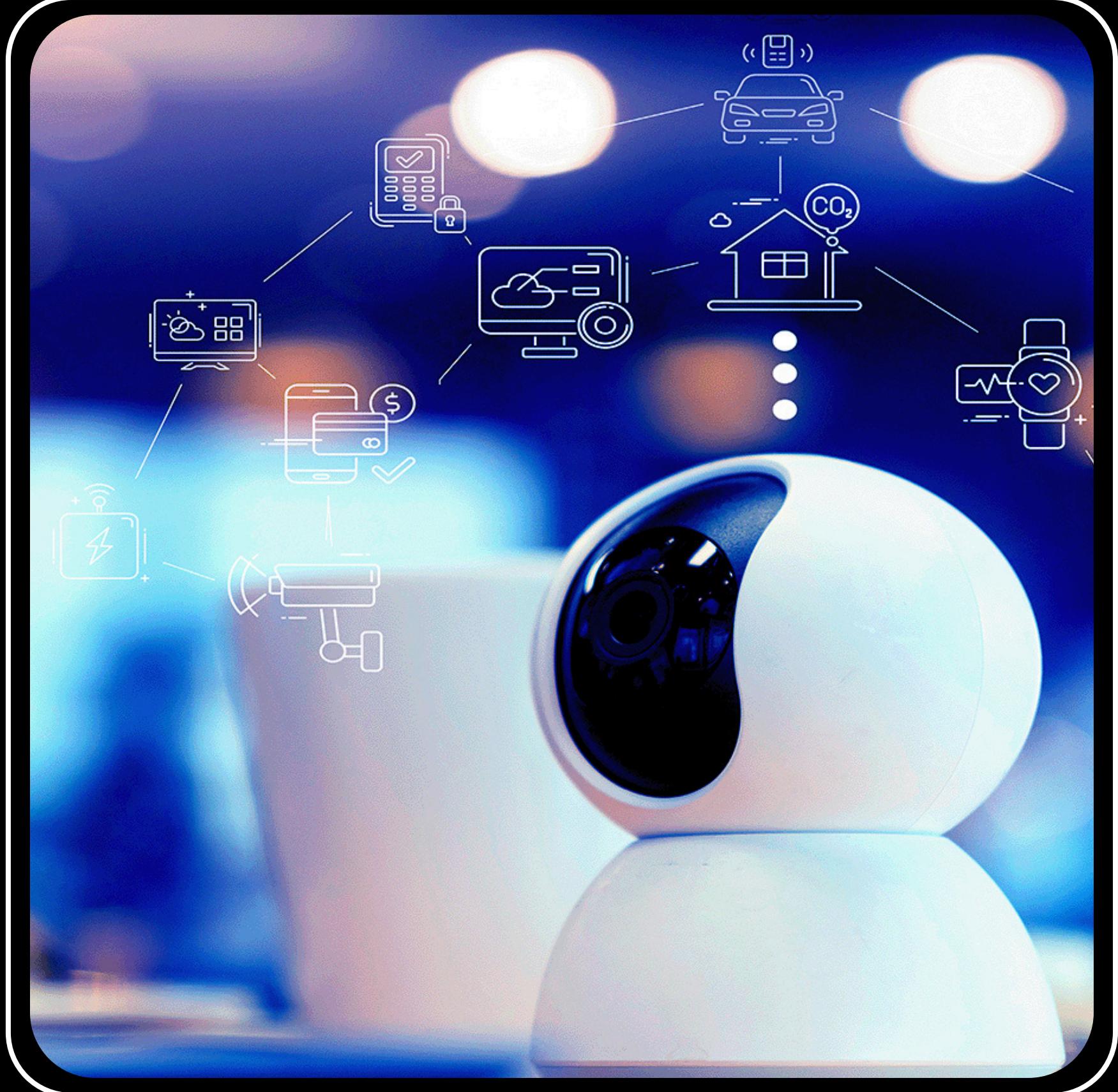


POURQUOI LES IOT ET LE BIG DATA ?

Les capteurs IoT génèrent un volume important de données en continu. Les systèmes traditionnels ne peuvent pas gérer efficacement ces flux massifs et rapides. Le Big Data permet de traiter et analyser ces données pour obtenir des insights en temps réel.

LES PRINCIPAUX CHALLENGES

- Volume massif et hétérogène de données.
- Traitement en temps réel nécessaire pour la réactivité.
- Stockage durable et optimisé pour l'analyse.
- Visualisation claire pour faciliter la prise de décision.



OBJECTIFS GÉNÉRAUX

- Concevoir un pipeline Big Data complet pour IoT.
- Assurer ingestion, traitement et stockage des flux en temps réel.
- Produire des indicateurs clés et un dashboard interactif pour le suivi.

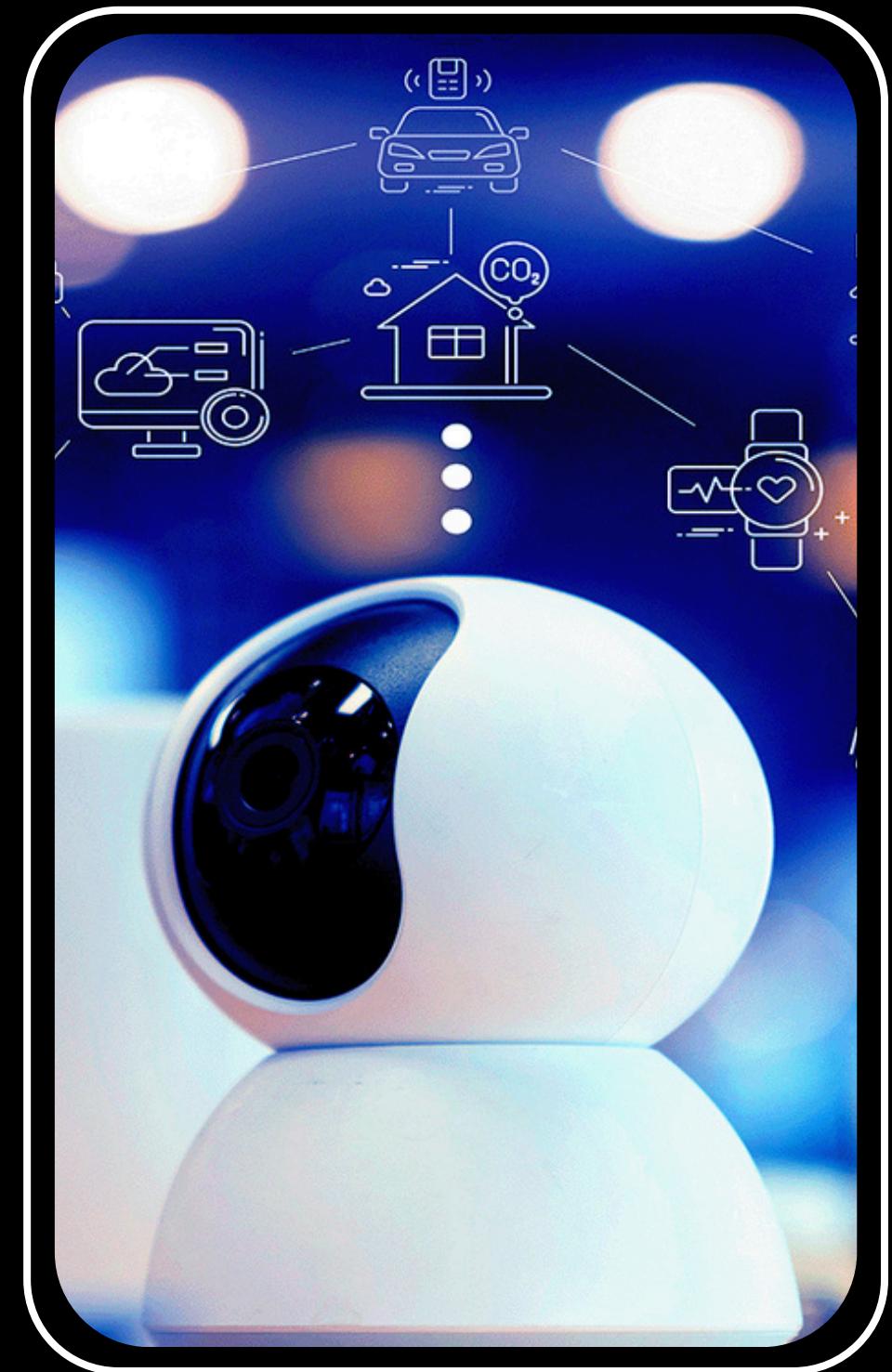
ARCHITECTURE GLOBALE

Architecture distribuée avec Kafka, Spark et HDFS.

Flux de données : **Producteur → Kafka → Spark Streaming → HDFS → Spark Analytics → Dashboard Python.**

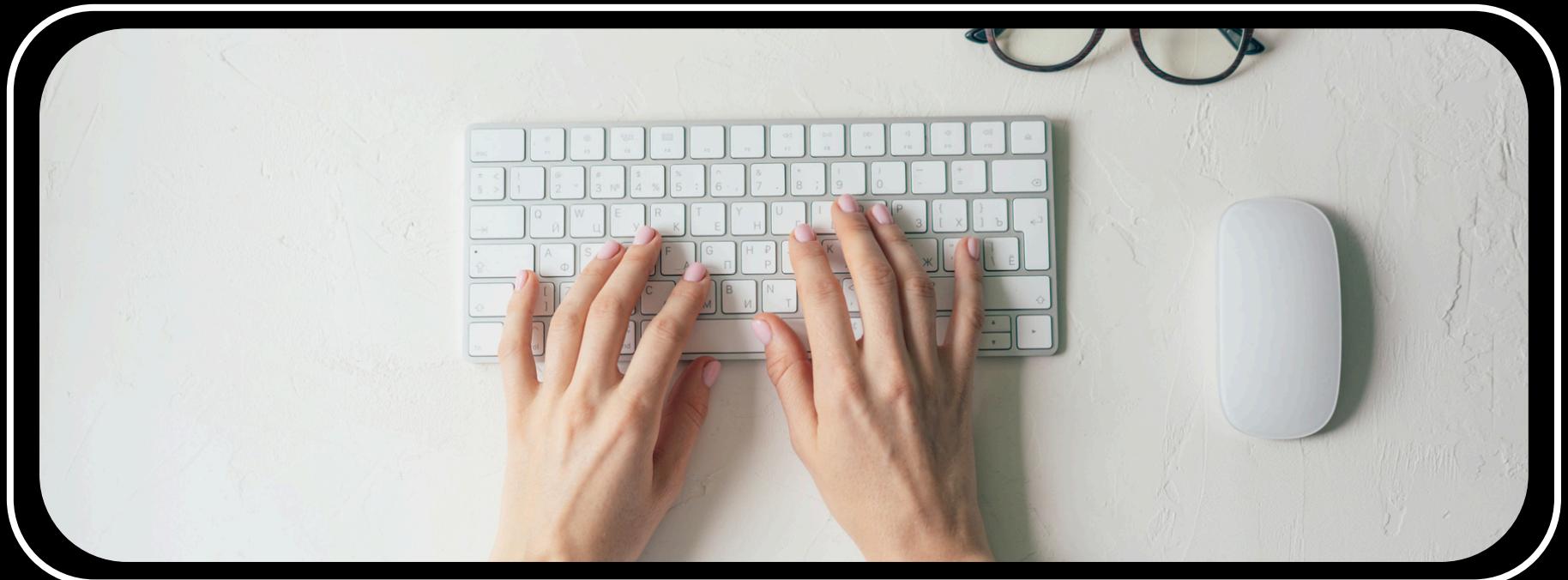
Chaque composant joue un rôle précis pour assurer scalabilité et fiabilité.

* * *



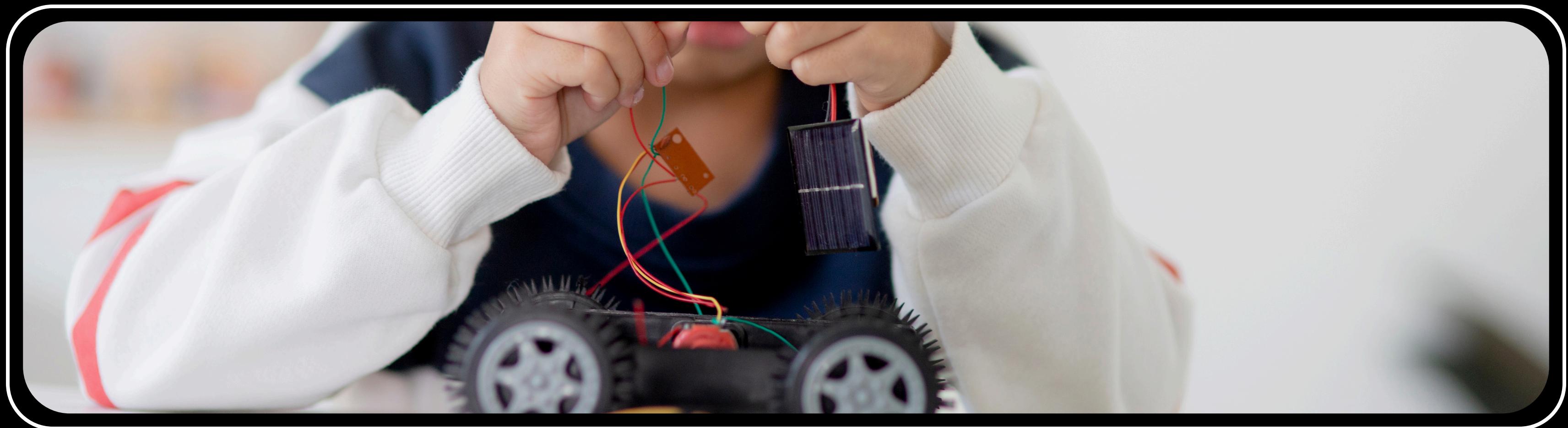
POURQUOI KAFKA ?

Kafka centralise les flux de données des capteurs IoT.
Il assure l'**ingestion parallèle**, la tolérance aux pannes et la distribution des messages aux consommateurs.



POURQUOI SPARK STREAMING ?

Spark Streaming consomme les flux Kafka, applique des transformations et enrichissements.
Permet des calculs distribués et des micro-batchs pour réduire la latence.



POURQUOI HDFS?

HDFS conserve les données de manière fiable et durable.
Format Parquet optimisé pour l'analyse Big Data.
Permet des traitements batch et analyses complexes via
Spark SQL.



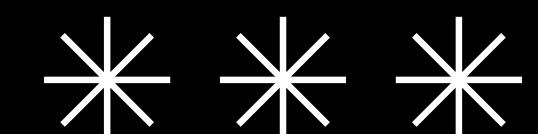
```
kafka-producer > ➜ producer.py > ...
1  from kafka import KafkaProducer
2  import json, time, random
3
4  producer = KafkaProducer(
5      bootstrap_servers='kafka:29092',
6      value_serializer=lambda v: json.dumps(v).encode('utf-8')
7  )
8
9  while True:
10     data = {
11         "deviceId": f"sensor{random.randint(1,10)}",
12         "temperature": round(random.uniform(20.0, 35.0), 2),
13         "humidity": round(random.uniform(30.0, 70.0), 2),
14         "energy_kwh": round(random.uniform(0.5, 5.0), 2),
15         "battery_level": random.randint(10, 100),
16         "status": random.choice(["OK", "WARNING", "CRITICAL"]),
17         "timestamp": int(time.time())
18     }
19     producer.send('iot-sensors', data)
20     time.sleep(5)
21     print(f"Sent: {data}")
```

PRODUCTEUR PYTHON

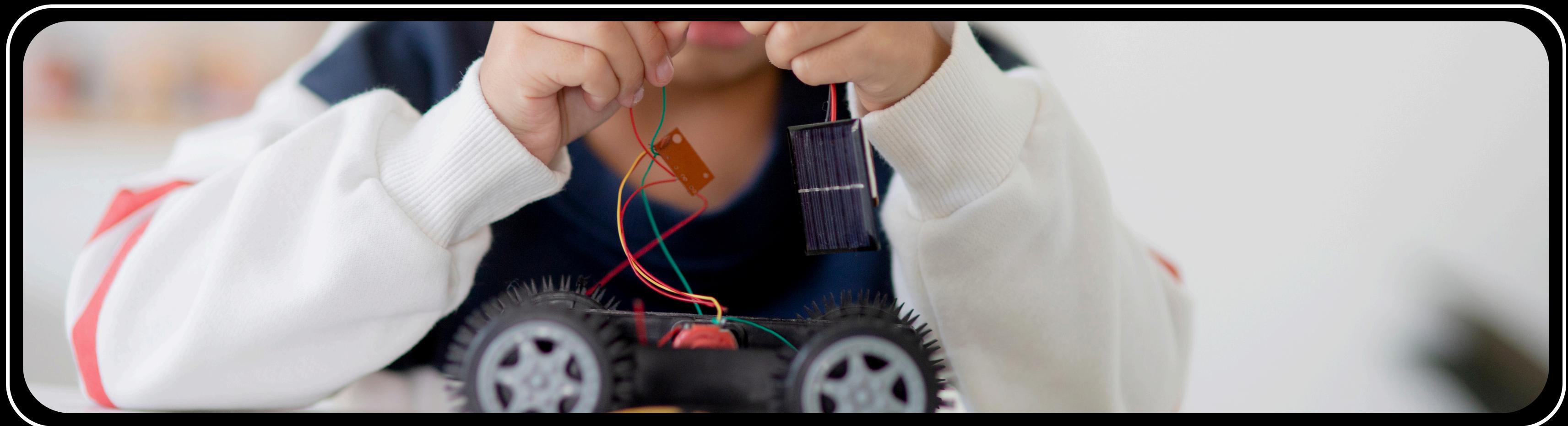
Script Python simulant plusieurs capteurs IoT.

Chaque message JSON contient : température, humidité, énergie, batterie, horodatage.

Rôle : alimenter Kafka avec des données réalistes.



REALISATION DU PIPELINE



```
val schema = new StructType()
    .add("deviceId", StringType)
    .add("temperature", DoubleType)
    .add("humidity", DoubleType)
    .add("energy_kwh", DoubleType)
    .add("battery_level", IntegerType)
    .add("status", StringType)
    .add("timestamp", LongType)

val spark = SparkSession.builder
    .appName("IoTStreaming")
    .getOrCreate()

spark.sparkContext.setLogLevel("WARN")

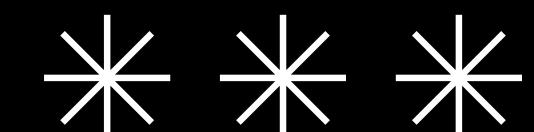
val df = spark
```

SPARK STREAMING

Consomme les messages Kafka et transforme les données en colonnes exploitables.

Enrichit les flux : conversion de température, horodatage lisible, calculs intermédiaires.

Stockage des résultats dans HDFS pour analyse ultérieure.



HDFS DANS LE PIPELINE

HDFS garantit la durabilité et la fiabilité des données.
Permet de relire facilement les données pour des analyses batch avec Spark Analytics.
Le checkpointing assure la reprise automatique en cas de panne.

```
resultDf.writeStream  
.format("parquet")  
.option("path", "hdfs://namenode:9000/data/iot-output")  
.option("checkpointLocation", "hdfs://namenode:9000/checkpoints/iot")  
.trigger(Trigger.ProcessingTime("10 seconds"))  
.outputMode("append")  
.start()  
.awaitTermination()
```

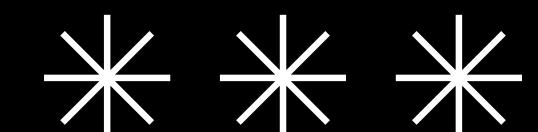
```
println("== ANALYSE SPARK SQL (DATAFRAME) ==\n\nval analyticsDf = spark.sql(\"\"\"\nSELECT\n    deviceId,\n    round(avg(temperature), 2) as avgTemp,\n    round(avg(humidity), 2) as avgHum,\n    round(sum(energy_kwh), 2) as totalEnergy,\n    min(battery_level) as minBattery\nFROM iot_data\nGROUP BY deviceId\n\"\"\")\nanalyticsDf.show()
```

ANALYSE AVEC SPARK SQL

Chargement des données HDFS pour calculer les KPI globaux et par appareil.

Mesures : température moyenne, humidité moyenne, consommation énergétique, niveau de batterie.

Préparation des fichiers CSV pour visualisation.



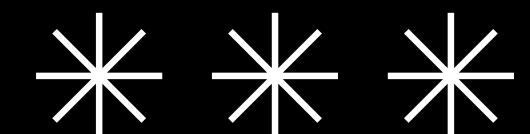
```
println("== ANALYSE BAS NIVEAU (RDD) =")
val rdd = df.rdd
val countByDevice = rdd
    .map(row => (row.getAs[String]("device"), 1))
    .reduceByKey(_ + _)
    .foreach(r => println(s"Appareil: $r._1 - Messages: ${r._2}"))
    .collect()
    .foreach(r => println(s"Appareil: $r._1 - Messages: ${r._2}"))

println("Nombre de messages reçus par appareil :")
countByDevice.collect().foreach { case (device, count) =>
    println(s"Appareil: $device - Messages: $count")
}
```

ANALYSE BAS NIVEAU AVEC RDD

Les RDD (Resilient Distributed Datasets) permettent un traitement distribué à bas niveau.

Exemple : calcul du nombre total de messages reçus par chaque appareil. Cette approche illustre la puissance de Spark pour gérer des transformations et des agrégations distribuées, complétant ainsi les analyses via Spark SQL.

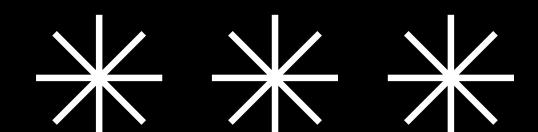


```
2 import plotly.graph_objects as go
3 from plotly.subplots import make_subplots
4
5 try:
6     df_g = pd.read_csv('global.csv')
7     df_d = pd.read_csv('devices.csv')
8
9     colors = {
10         'primary': '#1E3A8A',
11         'secondary': '#3B82F6',
12         'accent': '#60A5FA',
13         'background': '#0F172A',
14         'card': '#1E293B',
15         'text': '#F8FAFC',
16         'success': '#10B981',
17         'warning': '#F59E0B'
18     }
19
20     fig = make_subplots(
21         rows=4, cols=6,
22         specs=[[
23             [{"type": "indicator", "colspan": 2}, None, {"type": "indicate
24             [{"type": "bar", "colspan": 4}, None, None, None, {"type": "pi
25             [{"type": "bar", "colspan": 3}, None, None, {"type": "bar", "c
26             [{"type": "scatter", "colspan": 6}, None, None, None, None, No
27         ],
28         vertical_spacing=0.12,
29         horizontal_spacing=0.07
```

VISUALISATION PYTHON

Lecture des CSV HDFS avec Pandas.
Création de graphiques interactifs
avec Plotly : jauge, barres,
camemberts, lignes.

Permet de suivre la santé des
capteurs et de détecter rapidement
les anomalies.



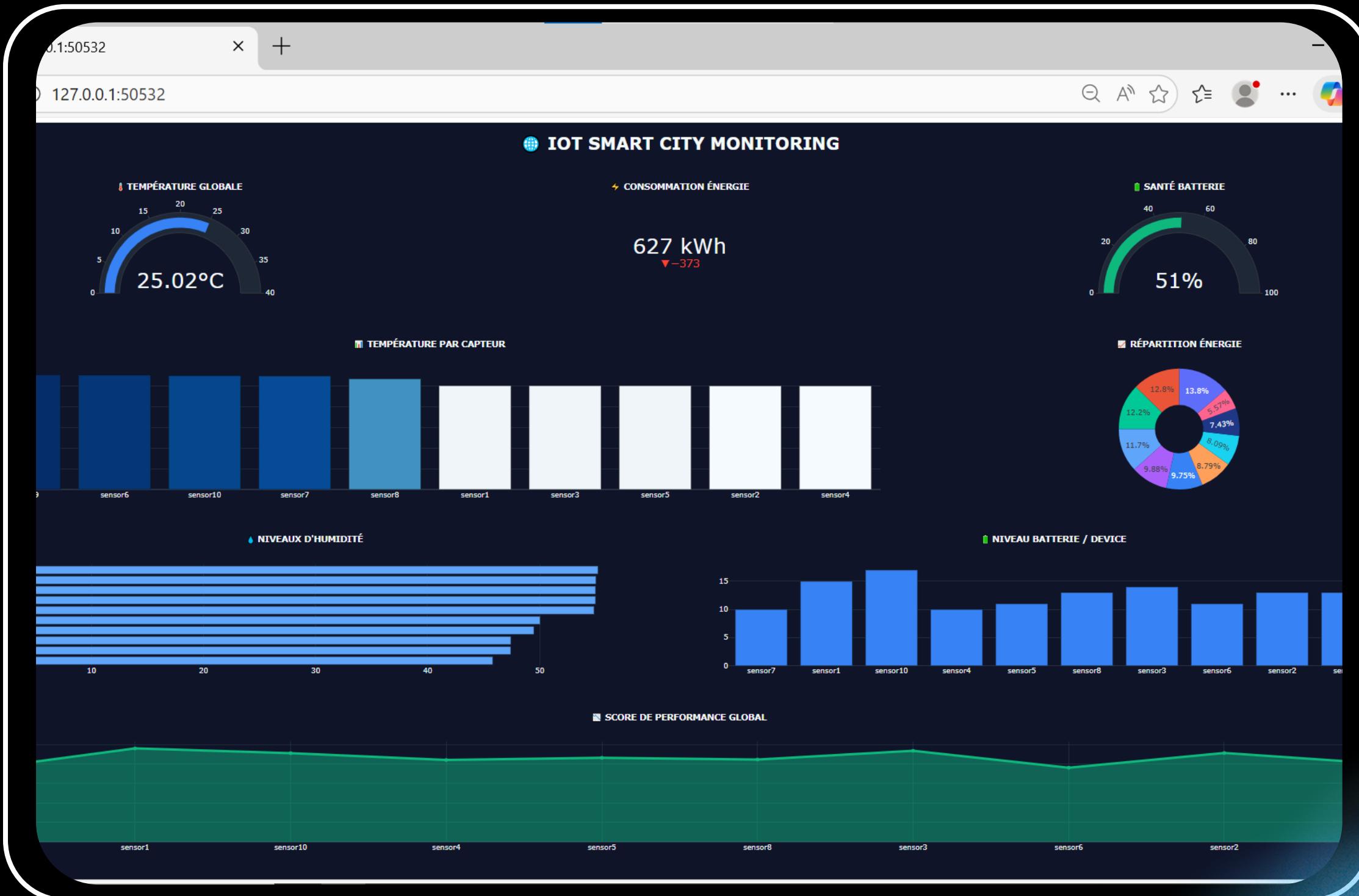
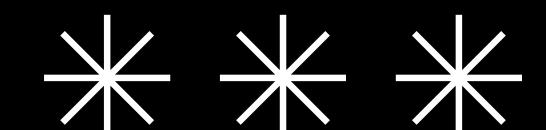
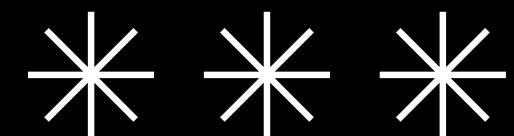


TABLEAU DE BORD INTERACTIF

Affiche 8 graphiques principaux : température, énergie, batterie, humidité, répartition par device et score global.

Permet un suivi en temps réel et une analyse rapide de la performance du système IoT.





CONCLUSION

Le projet a permis de développer un pipeline Big Data complet, depuis la collecte des données via les capteurs IoT jusqu'à la visualisation dans un dashboard interactif. Grâce à Kafka, Spark Streaming et HDFS, nous avons pu traiter les flux en temps réel, générer des indicateurs clés et préparer le terrain pour des analyses avancées. Ce système offre un monitoring efficace, la possibilité de détecter les anomalies et de planifier la maintenance prédictive, tout en restant évolutif, scalable et adapté aux environnements industriels.

