Université de Paris Cité
UFR Mathématiques et Informatique

IoT Sensor Data Processing

Master 2 Réseaux et Systèmes Autonomes
*T*raitement de flux de données

Yan ZUO – Zineb TAIEB – Sarah KASDI – Chen WANG
Encadré par Hoang Nhat Minh NGUYEN

Année universitaire 2024-2025

Table des matières

1 Introduction

L'essor des dispositifs IoT signifie que nous devons collecter, traiter et analyser des quantités de données exponentiellement plus importantes qu'auparavant. À mesure que les capteurs et les dispositifs deviennent de plus en plus omniprésents, cette tendance à l'augmentation des données ne fera que s'accentuer.

Les dispositifs IoT équipés de capteurs génèrent en continu de grandes quantités de données, telles que la température, l'humidité, la pression, etc. Ces flux de données peuvent présenter des formats très variés qui évoluent indépendamment les uns des autres.

C'est dans ce contexte que nous avons choisi d'aborder le sujet 2, qui porte sur l'utilisation des données des capteurs IoT, comme la température et l'humidité, et leur acheminement efficace vers Kafka dans un flux continu d'informations. Grâce à Spark Streaming, nous pouvons traiter ces données en temps réel. Notre objectif est de mettre en œuvre des mécanismes permettant de déclencher des notifications lorsque des seuils spécifiques sont dépassés. Cela signifie que nous ne nous contenterons pas de collecter les données, mais que nous chercherons également à obtenir des informations instantanées et pertinentes pouvant indiquer des conditions exceptionnelles ou des événements significatifs dans l'environnement surveillé.

Avec ce projet, nous espérons acquérir une meilleure compréhension de l'utilisation de Kafka et de Spark Structured Streaming, ainsi que de leur fonctionnement conjoint pour analyser les données en flux en temps réel.

2 Outils Utilisé

Conda

Conda est largement utilisé pour la gestion d'environnements virtuels et l'installation de bibliothèques dédiées à la science des données, au calcul scientifique et à l'apprentissage automatique.

Spark Structured Streaming

Il est utilisé pour des applications nécessitant l'analyse en temps réel, telles que le traitement de données provenant de capteurs IoT ou l'intégration avec des systèmes de messagerie comme Kafka.

Confluent Kafka

Confluent Kafka est employé pour la collecte, le traitement et la distribution en temps réel de données provenant de diverses sources, offrant une infrastructure robuste pour les pipelines de données.

Jupyter-Dash

Il est utilisé pour la visualisation interactive des données et la création de tableaux de bord intégrés dans des notebooks Jupyter, offrant une interface utilisateur conviviale pour l'exploration des données.

3 Partie Pratique

Cette section décrit les étapes pratiques entreprises pour la réalisation du projet, depuis la création d'un environnement Kafka jusqu'à l'analyse en temps réel des données simulées.

1. Création d'un Topic Kafka et d'un Environnement Virtuel
   La première étape consistait à configurer l'environnement de travail :
   - Environnement virtuel Python : Un environnement isolé a été configuré pour installer les dépendances nécessaires au projet, telles que Kafka, Spark, et les bibliothèques pour la manipulation des données.
   - Configuration de Kafka : Après avoir installé et démarré le serveur Kafka, un topic Kafka nommé sensor_data a été créé pour permettre la communication entre le producteur et le consommateur de données.

2. Génération de Données Simulées
   Pour simuler des données provenant de capteurs environnementaux, un générateur de données aléatoires a été mis en place. Les données générées incluaient des informations comme :
   — Identifiant du capteur.
   — Température ambiante.
   — indice UV.
   — Timestamp correspondant.
   — Rainfall.
   — Vitesse du vent.

   Ces données ont été envoyées en temps réel au topic Kafka configuré qui s'appelle sensor
3. Consommation des Données depuis Kafka
   Un consommateur Kafka a été configuré pour recevoir et traiter les données envoyées par les capteurs. Ces données étaient ensuite transmises à l'étape suivante pour une analyse plus approfondie.
4. Analyse en Temps Réel avec Spark Structured Streaming
   La phase d'analyse a été réalisée à l'aide de Spark Structured Streaming. Les données ont été extraites du topic Kafka, puis transformées et analysées en temps réel pour identifier des conditions spécifiques :

   Les données brutes ont été converties en une structure tabulaire pour faciliter leur manipulation. Des règles ont été appliquées pour détecter des situations particulières (par exemple, une température supérieure à 30°C ou un indice UV supérieur à 8).

5. Détection et Gestion des Alertes.
   Enfin, des alertes ont été déclenchées lorsque des conditions spécifiques ont été détectées. Ces alertes ont été affichées en temps réel dans la console, comme :

6. Détection de températures élevées, Indices UV dangereux.
   Ces alertes pourraient être intégrées à un système externe pour des notifications ou une gestion automatisée.

3.1 Topic Kafka sensor et environnement virtuel

mise en place des environnement et création du Topic Kafka sur confluent

- Topic Kafka : notre topic kafka sur confluent s'appelle sensor

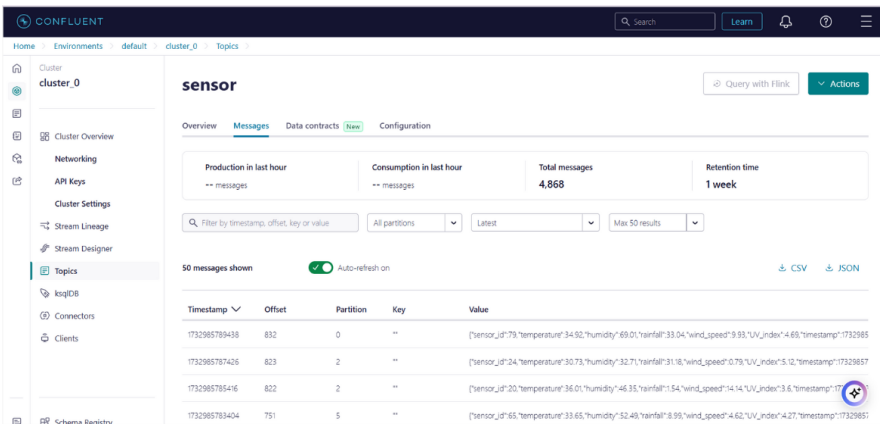


FIGURE 1 – Topic

-Activation de l'environnement virtuelle :

```
C:\Users\zy115\upc_streaming_data_processing>conda activate td5
```

FIGURE 2 – activation de l'environnement

3.2 Simulation des données en streaming

— On a utilisé le script sensor.py qui génère des données simulées de capteurs (température, humidité, précipitations, vitesse du vent) et les envoie à Kafka.
— les partie importante de ce script sont :

3.2.1 generate_sensor_data()

la fonction def generate_sensor_data() : générer des données de manière aléatoire pour simuler les changements météorologiques en temps réel.
Id de capteur, température, humidité, pluvieux, vitesse du vent, indice UV, temps réel

```python
# genere les data
def generate_sensor_data():
    return {
        "sensor_id": random.randint(1, 100),
        "temperature": round(random.uniform(20.0, 40.0), 2),
        "humidity": round(random.uniform(30.0, 70.0), 2),
        "rainfall": round(random.uniform(0.0, 50.0), 2),
        "wind_speed": round(random.uniform(0.0, 20.0), 2),
        "UV_index": round(random.uniform(0.0, 11.0), 2),
        "timestamp": time.time()
    }
```

FIGURE 3 – géneration des données

Ensuite nous créons une boucle while pour générer des données toute les deux secondes.

```python
try:
    while True:
        data = generate_sensor_data()
        producer.send(TOPIC, data)
        print(f"Sent: {data}")
        time.sleep(2)

except KeyboardInterrupt:
    print("Stopped by user.")
```

FIGURE 4 – fonction qui génère data sensor

3.2.2 Kafka producer

```python
# Kafka Producer
producer = KafkaProducer(
    bootstrap_servers='pkc-e0zxq.eu-west-3.aws.confluent.cloud:9092',
    security_protocol="SASL_SSL",
    sasl_mechanism="PLAIN",
    sasl_plain_username="[redacted]",
    sasl_plain_password="[redacted]",
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)
```

FIGURE 5 – Kafka producer

Nous avons mis en place un Kafka Producer en Python en utilisant la bibliothèque kafka-python pour se connecter à un cluster Kafka hébergé sur Confluent Cloud.
Ci-dessous les messages écrit sur notre cluster Kafka.

{"sensor_id":87,"temperature":37.27,"humidity":67.43,"rainfall":16.18,"wind_speed":4.77,"UV_index":6.45,"timestamp":17328797...

{"sensor_id":40,"temperature":30.87,"humidity":54.67,"rainfall":38.43,"wind_speed":4.17,"UV_index":10.5,"timestamp":17328797...

{"sensor_id":45,"temperature":35.67,"humidity":62.26,"rainfall":16.18,"wind_speed":0.82,"UV_index":10.96,"timestamp":1732879...

{"sensor_id":4,"temperature":34.79,"humidity":51.8,"rainfall":19.5,"wind_speed":19.55,"UV_index":2.67,"timestamp":1732879699....

{"sensor_id":53,"temperature":24.62,"humidity":33.76,"rainfall":46.26,"wind_speed":18.74,"UV_index":4.98,"timestamp":173287...

FIGURE 6 – écriture des messages dans kafka

```
Sent: {'sensor_id': 75, 'temperature': 39.96, 'humidity': 65.23, 'rainfall': 4.45, 'wind_speed': 2.08, 'UV_index': 0.26,
 'timestamp': 1732879729.9970934}
Sent: {'sensor_id': 95, 'temperature': 32.04, 'humidity': 56.61, 'rainfall': 31.36, 'wind_speed': 1.72, 'UV_index': 10.0
1, 'timestamp': 1732879732.006293}
Sent: {'sensor_id': 36, 'temperature': 38.02, 'humidity': 64.71, 'rainfall': 6.83, 'wind_speed': 3.47, 'UV_index': 5.92,
 'timestamp': 1732879734.014719}
Sent: {'sensor_id': 30, 'temperature': 30.01, 'humidity': 68.25, 'rainfall': 1.33, 'wind_speed': 16.09, 'UV_index': 5.0,
 'timestamp': 1732879736.0214825}
Sent: {'sensor_id': 18, 'temperature': 36.95, 'humidity': 53.2, 'rainfall': 49.15, 'wind_speed': 3.72, 'UV_index': 8.52,
 'timestamp': 1732879738.0285892}
Sent: {'sensor_id': 45, 'temperature': 20.49, 'humidity': 33.06, 'rainfall': 21.67, 'wind_speed': 12.01, 'UV_index': 3.8
2, 'timestamp': 1732879740.0393903}
Sent: {'sensor_id': 47, 'temperature': 36.52, 'humidity': 55.66, 'rainfall': 47.77, 'wind_speed': 2.54, 'UV_index': 1.84
, 'timestamp': 1732879742.048054}
Sent: {'sensor_id': 60, 'temperature': 32.68, 'humidity': 67.62, 'rainfall': 30.48, 'wind_speed': 10.48, 'UV_index': 1.6
7, 'timestamp': 1732879744.0513003}
Sent: {'sensor_id': 28, 'temperature': 23.17, 'humidity': 48.64, 'rainfall': 7.84, 'wind_speed': 10.99, 'UV_index': 8.01
, 'timestamp': 1732879746.055299}
Sent: {'sensor_id': 60, 'temperature': 39.13, 'humidity': 32.72, 'rainfall': 33.65, 'wind_speed': 13.74, 'UV_index': 9.8
5, 'timestamp': 1732879748.0639122}
Sent: {'sensor_id': 18, 'temperature': 26.43, 'humidity': 65.45, 'rainfall': 41.51, 'wind_speed': 16.24, 'UV_index': 2.2
2, 'timestamp': 1732879750.0754278}
```

FIGURE 7 – écriture des message dans kafka

3.3 Traitement des données

Pour cette partie on a utilisé un autre script Process_iot_streaming.py qui gére le traitement en temps réel des flux de données IoT provenant de Kafka.

1. Lire les donnée à partir de kafka.

```
raw_stream = spark.readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", bootstrapServers) \
        .option("kafka.sasl.mechanism", "PLAIN") \
        .option("kafka.security.protocol", "SASL_SSL") \
        .option("kafka.sasl.jaas.config", "org.apache.kafka.common.security.scram.ScramLoginModule
        required username=\"LIQN5IFO4VF2U5KV\" password=\"Qp6Zl3PuSayJSCRD8/
        pXJBJNNPIE6NCKWIMUsUk701c2HX6Pf/zVFg2Z1kVJzJei\";") \
        .option("subscribe", topics) \
        .option("startingOffsets", "earliest") \
        .load()
```

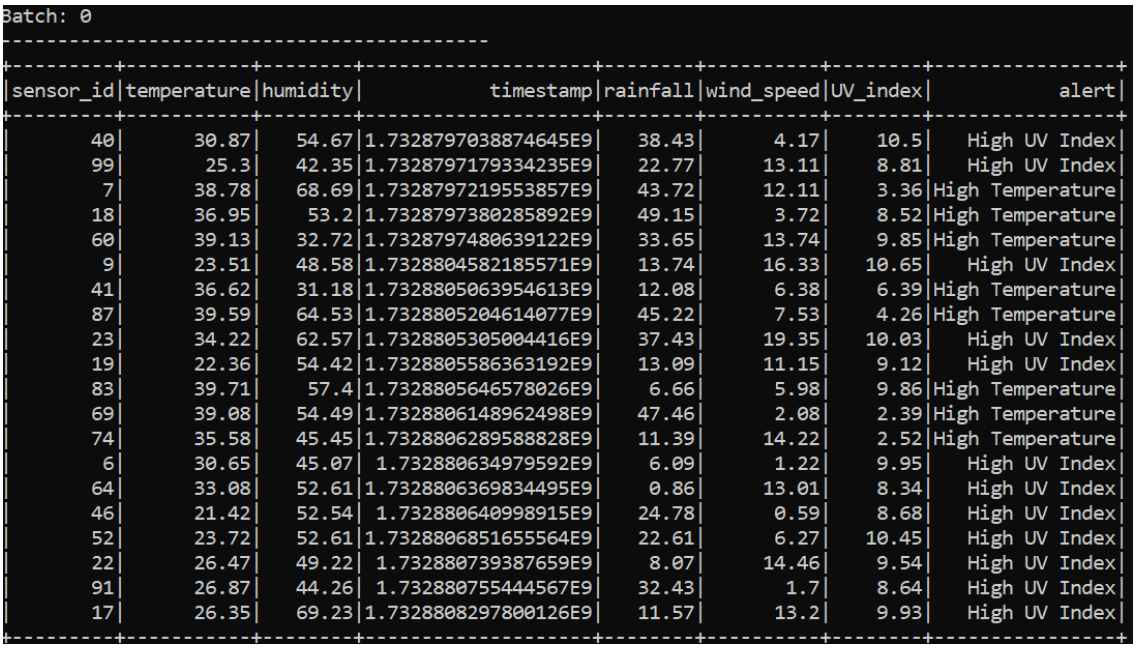
2. Stocker les données récupérées dans des tables mémoires consultables via des requêtes SQL Spark.

```
parsed_stream = raw_stream.selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
    .select("data.*")
parsed_stream.writeStream \
        .format("memory") \
        .queryName("raw_stream") \
        .outputMode("append") \
        .trigger(processingTime="2 seconds") \
        .start()
```

3. Définir les alertes
   On envoie des alertes sous des conditions spécifiques,par exemple si la température est > 35, il va écrire "High Temperature".
   Les alertes sont ajoutées en tant que colonne supplémentaire (alert) aux données.

```
# Création des alertes avec conditions multiples
alerts = parsed_stream.filter(
    (col("temperature") > 35.0) |
    (col("humidity") < 20.0) |
    (col("UV_index") > 8.0) |
    (col("rainfall") > 50.0) |
    (col("wind_speed") > 20.0)
).withColumn(
    "alert",
    when(col("temperature") > 35.0, "High Temperature")
    .when(col("humidity") < 20.0, "Low Humidity")
    .when(col("UV_index") > 8.0, "High UV Index")
    .when(col("rainfall") > 50.0, "Heavy Rainfall")
    .when(col("wind_speed") > 20.0, "High Wind Speed")
    .otherwise("No Alert")
).select(
    "sensor_id",
    "temperature",
    "humidity",
    "timestamp",
    "rainfall",
    "wind_speed",
    "UV_index",
    "alert"
)
#  output
query = alerts.writeStream \
    .outputMode("append") \
    .format("console") \
    .trigger(processingTime="30 second")\
    .start()
```

4. Résultats



```
Batch: 0
-------------------------------------------
+---------+-----------+--------+--------------------+--------+----------+--------+----------------+
|sensor_id|temperature|humidity|           timestamp|rainfall|wind_speed|UV_index|           alert|
+---------+-----------+--------+--------------------+--------+----------+--------+----------------+
|       40|      30.87|   54.67|1.7328797038874645E9|   38.43|      4.17|    10.5|   High UV Index|
|       99|       25.3|   42.35|1.732879717933423 5E9|   22.77|     13.11|    8.81|   High UV Index|
|        7|      38.78|   68.69|1.732879721955385 7E9|   43.72|     12.11|    3.36|High Temperature|
|       18|      36.95|    53.2|1.7328797380285892E9|   49.15|      3.72|    8.52|High Temperature|
|       60|      39.13|   32.72|1.7328797480639122E9|   33.65|     13.74|    9.85|High Temperature|
|        9|      23.51|   48.58|1.7328804582185571E9|   13.74|     16.33|   10.65|   High UV Index|
|       41|      36.62|   31.18|1.7328805063954613E9|   12.08|      6.38|    6.39|High Temperature|
|       87|      39.59|   64.53|1.7328805204614077E9|   45.22|      7.53|    4.26|High Temperature|
|       23|      34.22|   62.57|1.7328805305004416E9|   37.43|     19.35|   10.03|   High UV Index|
|       19|      22.36|   54.42|1.7328805586363192E9|   13.09|     11.15|    9.12|   High UV Index|
|       83|      39.71|    57.4|1.732880564657802 6E9|    6.66|      5.98|    9.86|High Temperature|
|       69|      39.08|   54.49|1.7328806148962498E9|   47.46|      2.08|    2.39|High Temperature|
|       74|      35.58|   45.45|1.7328806289588828E9|   11.39|     14.22|    2.52|High Temperature|
|        6|      30.65|   45.07| 1.732880634979592E9|    6.09|      1.22|    9.95|   High UV Index|
|       64|      33.08|   52.61|1.7328806369834495E9|    0.86|     13.01|    8.34|   High UV Index|
|       46|      21.42|   52.54| 1.732880640998915E9|   24.78|      0.59|    8.68|   High UV Index|
|       52|      23.72|   52.61|1.7328806851655564E9|   22.61|      6.27|   10.45|   High UV Index|
|       22|      26.47|   49.22| 1.732880739387659E9|    8.07|     14.46|    9.54|   High UV Index|
|       91|      26.87|   44.26| 1.732880755444567E9|   32.43|       1.7|    8.64|   High UV Index|
|       17|      26.35|   69.23|1.732880829780012 6E9|   11.57|      13.2|    9.93|   High UV Index|
+---------+-----------+--------+--------------------+--------+----------+--------+----------------+
```

FIGURE 8 – Détection des alertes

3.4 Visualisation

— Nous utilison le script Process_iot_streaming_v2.py en Python, Spark et Jupyter Dash.
— C'est pour voir et analyser l'évolution des variables généré par les capteurs en temps réel

3.4.1 Fonctions principales

Au lieu de sauvegarder des données dans la table de mémoire interne, nous avons décidé de les sauvegarder dans des parquet externe pour les utiliser dans la visualisation.
Dans Spark Structured Streaming, les parquet est un type de file sink qui est utilisé pour écrire les résultats d'un flux de données dans un répertoire sous forme de fichiers.

```python
# Définir le chemin de sortie
ALERTS_PATH = "./alerts_parquet"
CHECKPOINT_PATH = "./alerts_checkpoint"

# Écrire les données d'alerte dans Parquet
alerts_query = alerts.writeStream \
    .format("parquet") \
    .option("path", ALERTS_PATH) \
    .option("checkpointLocation", CHECKPOINT_PATH) \
    .outputMode("append") \
    .start()
```

FIGURE 9 – Définition de la chemin de sortie

Ensuite, nous Chargeons périodiquement les données via des fichiers Parquet et les convertissons en Dataframe.
On définit une fonction fetch_alert_data qui charge les données depuis un fichier Parquet en utilisant Spark. Il utilise une variable globale alerts_data pour stocker ces données, converties en DataFrame Pandas grâce

à .toPandas().
Si la lecture réussit, les données sont affectées à la variable, tandis que si une erreur survient (gérée par le try/except), un message d'erreur est affiché avec print, et un DataFrame vide est créé. Une ligne (commentée) montre également une conversion de la colonne timestamp en datetime.

```python
# Charger périodiquement les données des fichiers Parquet
def fetch_alert_data():
    global alerts_data
    try:
        alerts_data = spark.read.parquet(ALERTS_PATH).toPandas()
        #alerts_data['timestamp'] = pd.to_datetime(alerts_data['timestamp'].astype(float), unit='s')
    except Exception as e:
        print(f"Erreur de lecture du fichier Parquet: {e}")
        alerts_data = pd.DataFrame()
```

FIGURE 10 – traitement des données pour visualisation

Nous définissons l'interface d'une application Dash pour un tableau de bord IoT (Internet des Objets).
La disposition inclut un titre centralisé, "IoT Alerts Dashboard", et un intervalle de mise à jour de 5 secondes (5000 ms) grâce au composant dcc.Interval. Il affiche plusieurs graphiques indépendants via dcc.Graph pour des métriques comme température, humidité, UV-index, précipitations et vitesse du vent.
Enfin, un autre graphique représente la répartition des alertes avec l'ID alert-distribution.

```python
# Définir la disposition de l'application Dash
app.layout = html.Div([
    html.H1("IoT Alerts Dashboard", style={'textAlign': 'center'}),

    dcc.Interval(id='update-interval', interval=5000),

    # Créer un graphique indépendant pour chaque variable
    html.Div([
        html.H2("Sensor Metrics Over Time"),
        dcc.Graph(id='temperature-trend'),
        dcc.Graph(id='humidity-trend'),
        dcc.Graph(id='UV-index-trend'),
        dcc.Graph(id='rainfall-trend'),
        dcc.Graph(id='wind-speed-trend')
    ]),

    # Carte de répartition des alertes
    html.H2("Number of Alerts"),
    dcc.Graph(id='alert-distribution')
])
```

FIGURE 11 – définition la page web

Enfin, Nous mettons à jour dynamiquement le graphique des tendances de température dans une application Dash.
Par exemple, la fonction update_temperature_trend est déclenchée par un callback qui utilise l'entrée update-interval (actualisation périodique) et met à jour la sortie temperature-trend. La fonction récupère les données via fetch_alert_data() et vérifie si elles sont vides.
Si c'est le cas, un graphique vide est renvoyé. Sinon, un graphique en ligne "px.line" est affiché avec l'axe des X pour les timestamps et l'axe des Y pour les valeurs de température, utilisant une couleur orange pour la ligne.

```python
# Mettre à jour le graphique de tendance de la température
@app.callback(
    Output('temperature-trend', 'figure'),
    Input('update-interval', 'n_intervals')
)
def update_temperature_trend(n_intervals):
    global alerts_data
    fetch_alert_data()
    if alerts_data.empty:
        return px.scatter(title="No Data Available")
    return px.line(alerts_data, x="timestamp", y="temperature",
                   title="Temperature Over Time",
                   labels={"temperature": "Temperature (°C)", "timestamp": "Timestamp"},
                   color_discrete_sequence=["orange"])
```

FIGURE 12 – Mettre à jour le graphique

3.4.2 Résultats

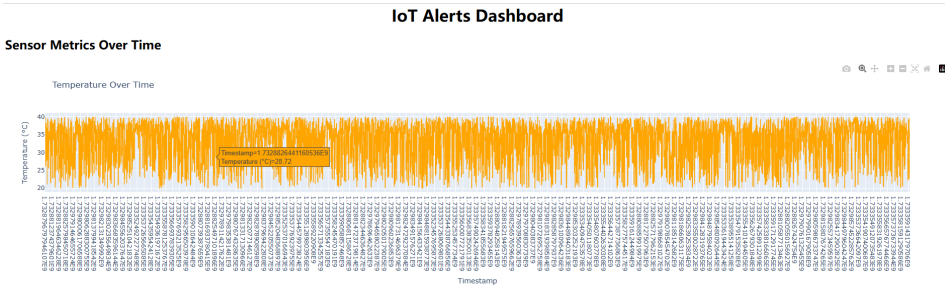


FIGURE 13 – Température

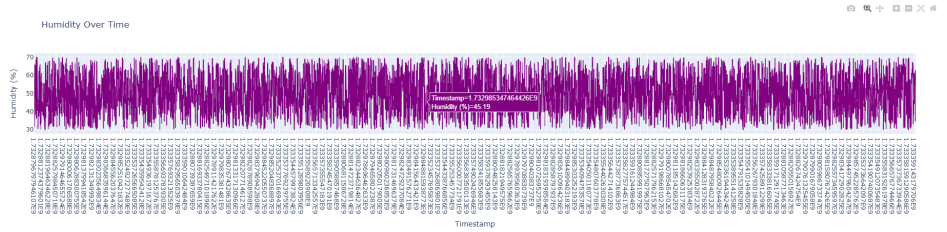


FIGURE 14 – Humidité

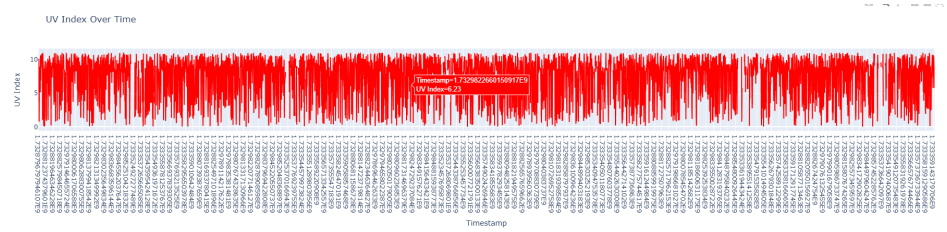


FIGURE 15 – UV index

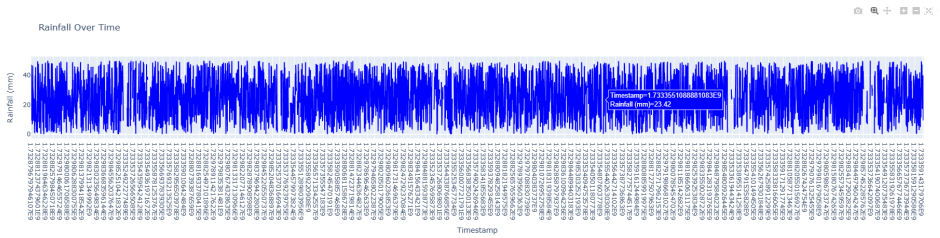


FIGURE 16 – Rain full

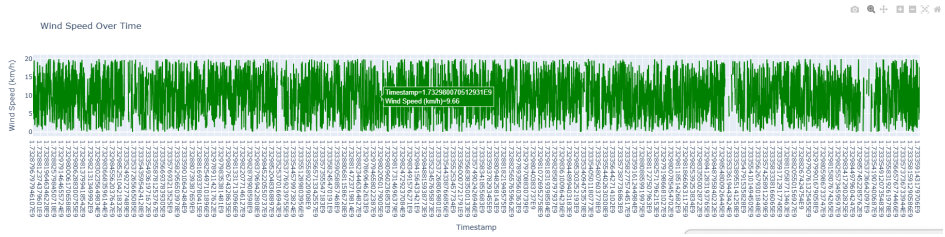


FIGURE 17 – Vitesse du vent

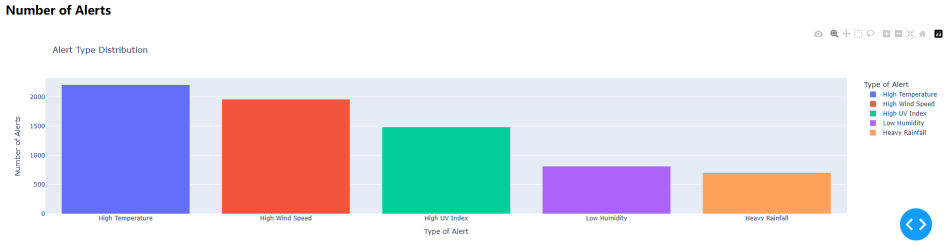


FIGURE 18 – Taux d'Alert

3.4.3 Interprétation des résultats

D'après le graphe de Température, on observe des variations importantes de la température, oscillant principalement entre 20°C et 40°C. Les données semblent denses et continues, ce qui indique une surveillance régulière des capteurs sur une longue période. Le comportement fluctuant pourrait refléter des changements diurnes et des cycles environnementaux.

Pour le graphe Taux d'Alert, Les types d'alertes "Low Humidity" et "Heavy Rainfall" sont moins fréquents, avec moins de 1000 occurrences chacune. Cela indique que les conditions de température et de vent sont les principales sources d'alertes.

3.4.4 Amélioration de l'affichage

Nous avons tenté de modifier la date des graphes tendances, il nous a fourni les graphes comme le graphique ci-dessous, car il y a beaucoup de sensors générés de manière aléatoire par sensor.py.

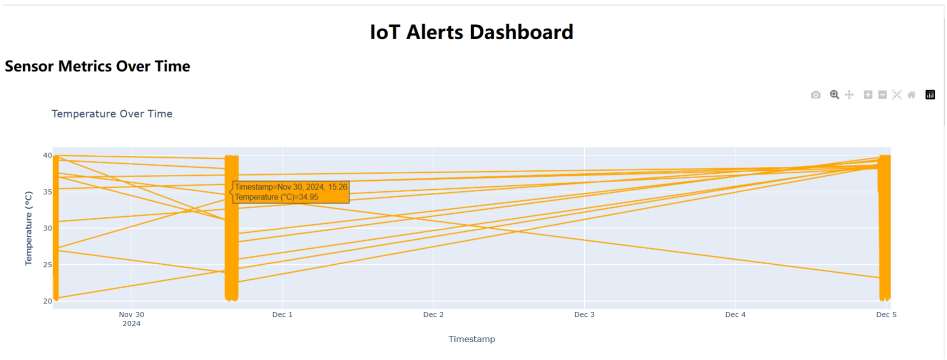


FIGURE 19 – Température avec le format de date

4 Conclusion

Ce projet a démontré l'efficacité de l'intégration de technologies modernes telles que Kafka et Spark Structured Streaming dans la gestion et le traitement des flux de données IoT en temps réel. En simulant des capteurs environnementaux, nous avons conçu un pipeline capable de collecter, analyser et déclencher des alertes basées sur des seuils prédéfinis, répondant ainsi à des besoins critiques de surveillance et de prise de décision.

Cette réalisation illustre le potentiel des solutions IoT pour transformer des données brutes en informations exploitables, renforçant ainsi la réactivité et l'efficacité dans divers secteurs. Elle constitue une base solide pour le développement de systèmes encore plus performants et prédictifs.