MATHÉMATIQUES ET INFORMATIQUE
Sciences
Université Paris Cité

IoT Sensor Data Processing

Traitement des flux de données

Etudiantes: Zineb TAIEB, Yan ZUO,Sarah KASDI, Chen WANG

Encadrants: Minh H.N NGUYEN

M2 Informatique - RSA
Université Paris Cité
UFR Mathématiques et Informatique

Contenue

- Introduction
- Outils Utilisées
- Partie Pratique
- Résultat
- Visualization
- Conclusion

Introduction

- **Objectif**
  - Transformation de la surveillance et de la compréhension de l'environnement
  - Importance des données IoT en temps réel
  - Traitement en temps réel, une nécessité
  - Impact sur la vie quotidienne et les décisions stratégiques

Outils Utilisées

- **Conda**
  - Un gestionnaire de paquets et d'environnements open-source qui simplifie l'installation et la gestion des bibliothèques et de leurs dépendances pour des langages tels que Python ou R.
  - Il permet de créer des environnements isolés, facilitant ainsi la gestion de projets aux besoins spécifiques

Outils Utilisées

- **Confluent Kafka**
  - Une plateforme de streaming de données basée sur Apache Kafka.
  - Elle fournit des outils supplémentaires pour faciliter la gestion et l'intégration des flux de données en temps réel au sein des entreprises.

Outils Utilisées

- **Spark**

  - Spark Structured Streaming est un module d'Apache Spark conçu pour construire des applications de traitement de flux en temps réel. Il est capable de traiter des données en continu avec une haute tolérance aux pannes et un débit élevé.

  - Analyse en temps réel, traitement des données de capteurs IoT, et intégration avec des systèmes de messagerie comme Kafka.

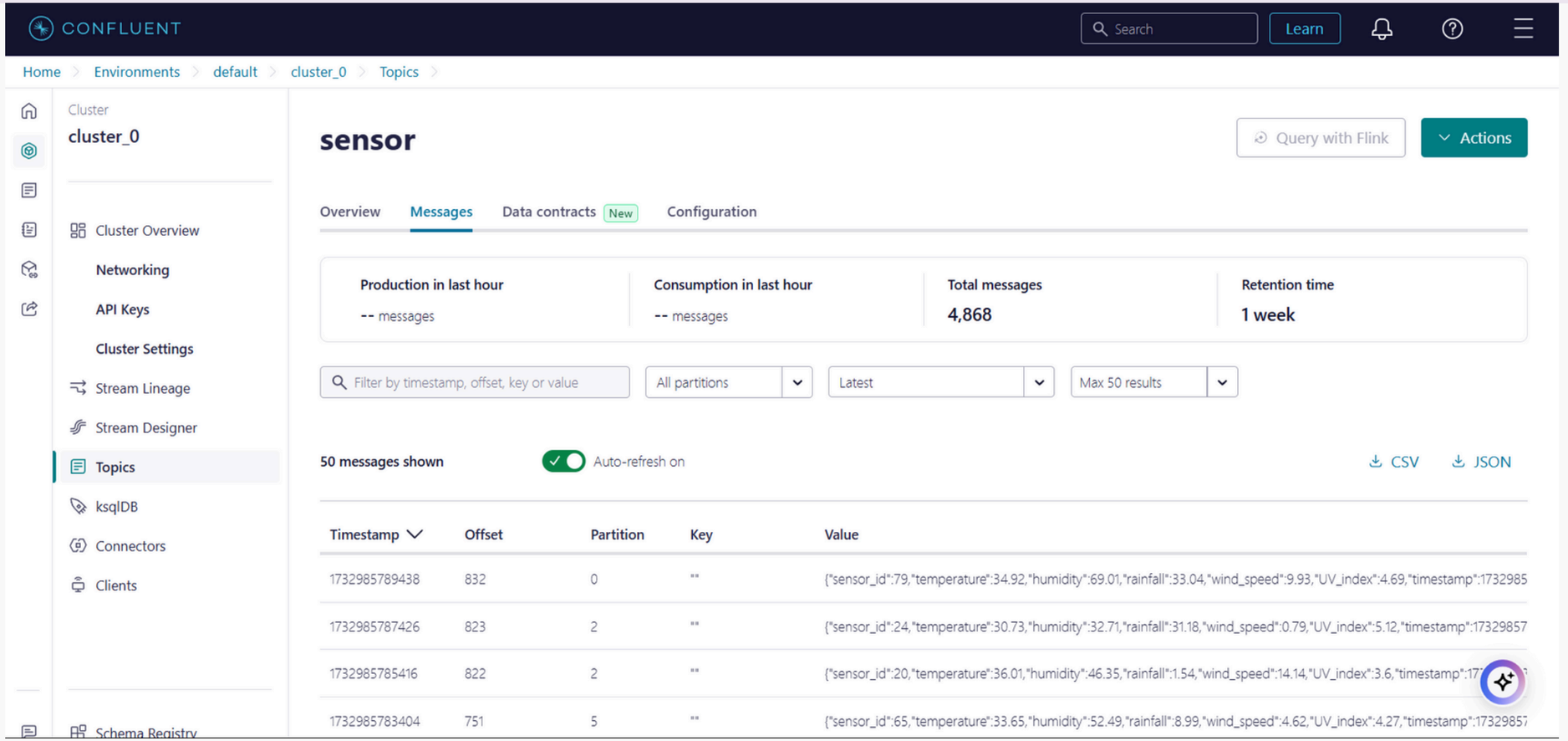
Outils Utilisées

- **Jupyter-Dash**
  - Une extension de Jupyter Notebook qui permet de créer et de déployer des applications interactives Dash directement depuis un notebook.
  - Cela facilite le développement rapide de tableaux de bord analytiques interactifs.

Partie Pratique

- Créer un topic Kafka et un environnement virtuel.
- Générer des données aléatoires pour simuler des données de capteurs environnementaux (Température, Indice UV, ect)
- Consommer les données des capteurs via le topic Kafka.
- Analyser et traiter les données en temps réel avec Spark Structured Streaming.
- Détecte des conditions spécifiques pour déclencher des alertes.

Topic Kafka et l’environnement virtuel




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

sensor.py

- **Description :**
  - Ce script génère des données simulées de capteurs (température, humidité, précipitations, vitesse du vent) et les envoie à Kafka.

Fonction

```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()
    }
```

- 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

Fonction

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

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

- Nous créons une boucle while pour générer des données à l'infini et pouvons toujours les transmettre à Kafka

Fonction

```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')
)
```

- 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

Process_iot_streaming.py

- **Description :**
  - Ce script utilise Python et Spark pour gérer le traitement en temps réel des flux de données IoT provenant de Kafka.

Fonction

- Lis des données fictives pour les capteurs IoT.
- Stocks les données récupérés dans des tables mémoires consultables via des requêtes SQL Spark.

```
parsed_stream.writeStream \
        .format("memory") \
        .queryName("raw_stream") \
        .outputMode("append") \
        .trigger(processingTime="2 seconds") \
        .start()
```

Fonction

- Envoie des alertes sous des conditions spécifiques, Les alertes sont ajoutées en tant que colonne supplémentaire (alert) aux données.

```
#  output
query = alerts.writeStream \
    .outputMode("append") \
    .format("console") \
    .trigger(processingTime="30 second")\
    .start()

query.awaitTermination()
```

```
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"
)
```

Process_iot_streaming_v2.py

- **Description :**
  - Ce script utilise Python, Spark et Jupyter Dash.
  - C'est pour voir et analyser l'évolution des variables généré par les capteurs en temps réel

Fonction

```
# 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()
```

- 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.

Fonction

```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()
```

- Charger périodiquement les données via des fichiers Parquet
- Convertir les données en Dataframe

Fonction

```
# 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')
])
```

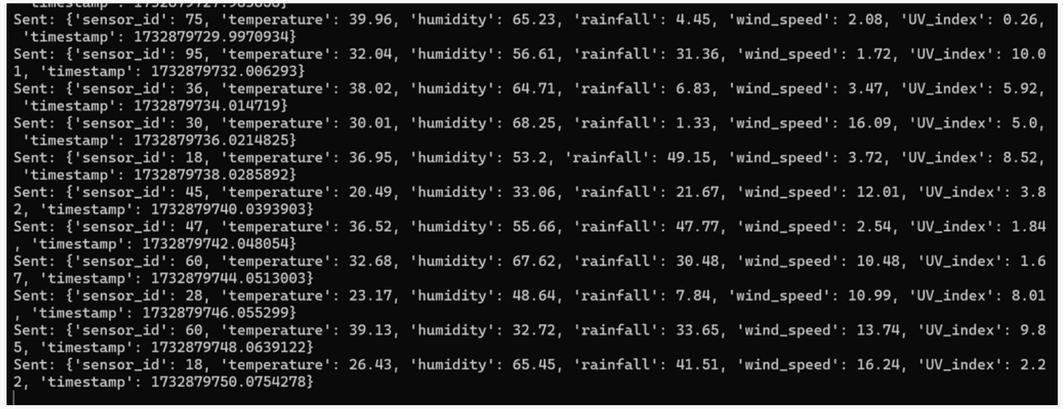
- Définir la page web de jupyter dash et les graphiques

Fonction

```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"])
```

- Mettre à jour le graphique de tendance de la température

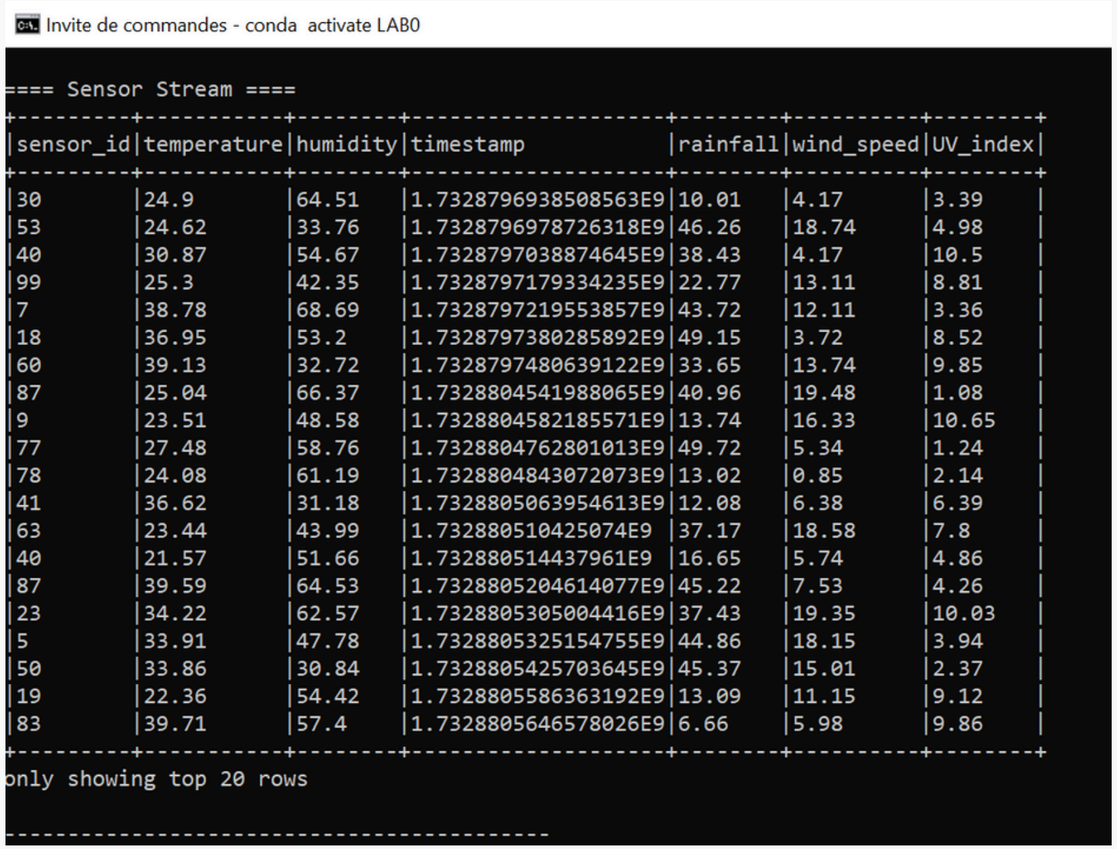
Résultat



Un flux continu envoyé par sensor.py vers Kafka, utilisé pour tester le traitement en temps réel par Process_iot_streaming.py. Les données montrent des relevés environnementaux émis par des capteurs simulés.

Les données affiché sur le dashboard topic kafka “sensor”

Résultat



Les données des capteurs IoT, collectées en temps réel et stockées temporairement dans une **table mémoire**.

Cette approche est typique dans Spark Structured Streaming pour visualiser le flux de données et effectuer des opérations analytiques.

Résultat

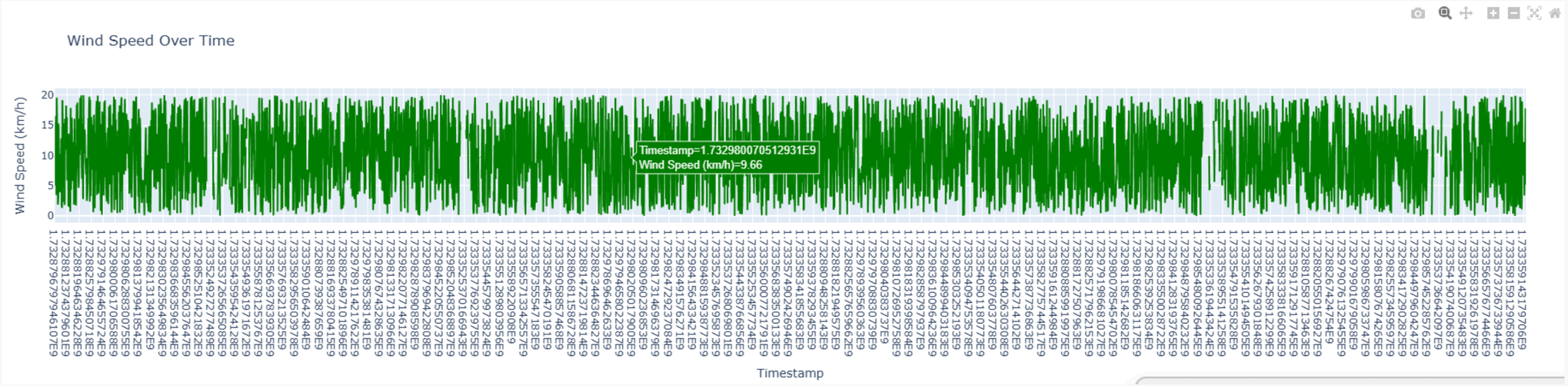
Batch: 0

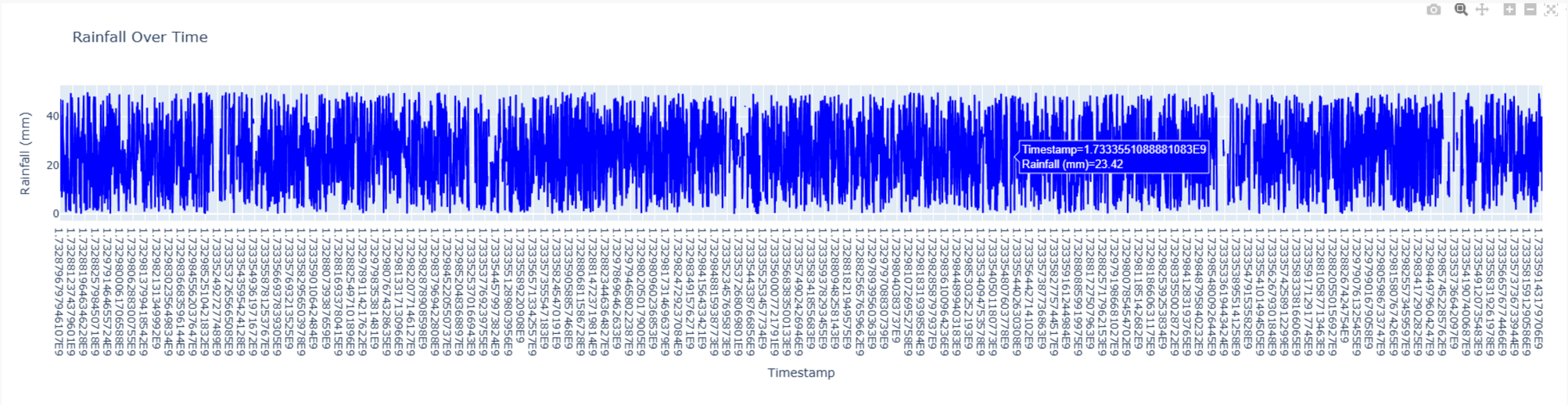
sensor_id	temperature	humidity	timestamp	rainfall	wind_speed	UV_index	alert
40	30.87	54.67	1.7328797038874645E9	38.43	4.17	10.5	High UV Index
99	25.3	42.35	1.732879717933423 5E9	22.77	13.11	8.81	High UV Index
7	38.78	68.69	1.7328797219553857E9	43.72	12.11	3.36	High Temperature
18	36.95	53.2	1.7328797380285892E9	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	1.7328808297800126E9	11.57	13.2	9.93	High UV Index

Les données de capteurs IoT traitées en temps réel à l'aide de **Spark Structured Streaming**, avec une colonne supplémentaire **alert** pour indiquer les conditions déclenchant des alertes.

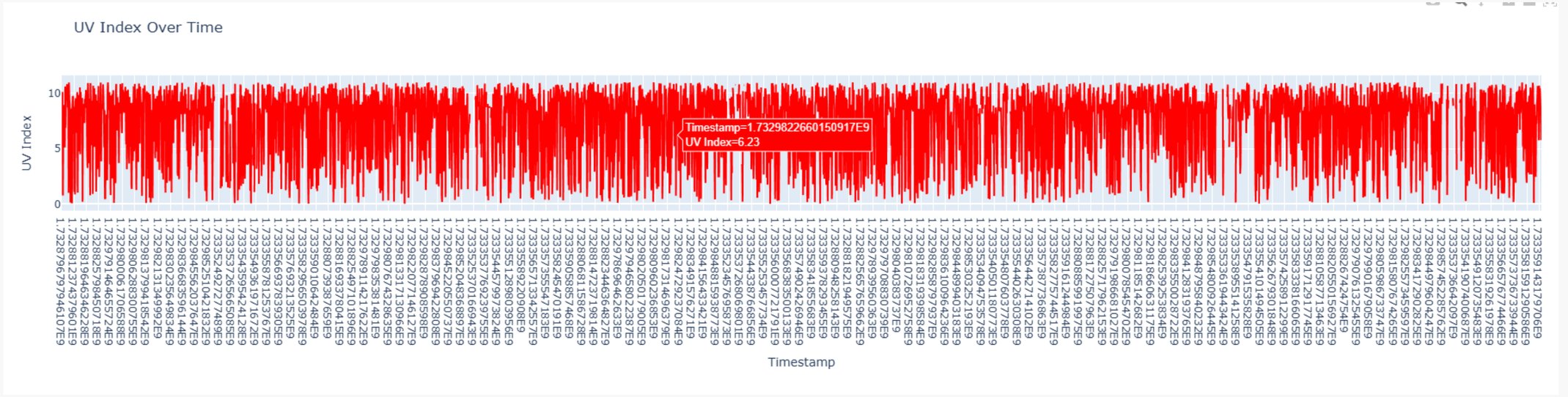
Les données montrent que des alertes ont été déclenchées pour certains capteurs en raison de valeurs dépassant un seuil prédéfini.

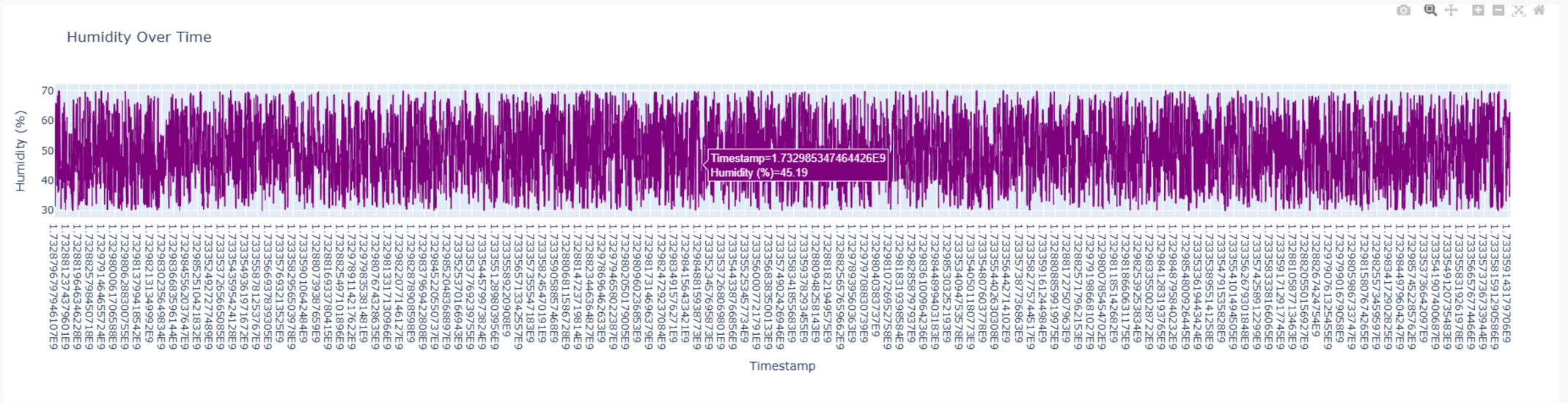
Visualisation



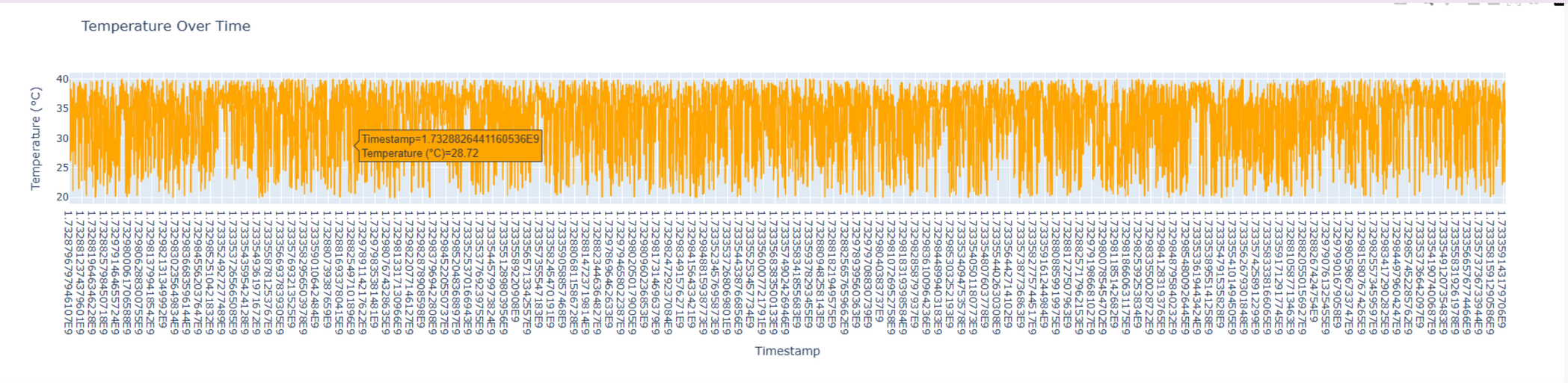


Visualisation

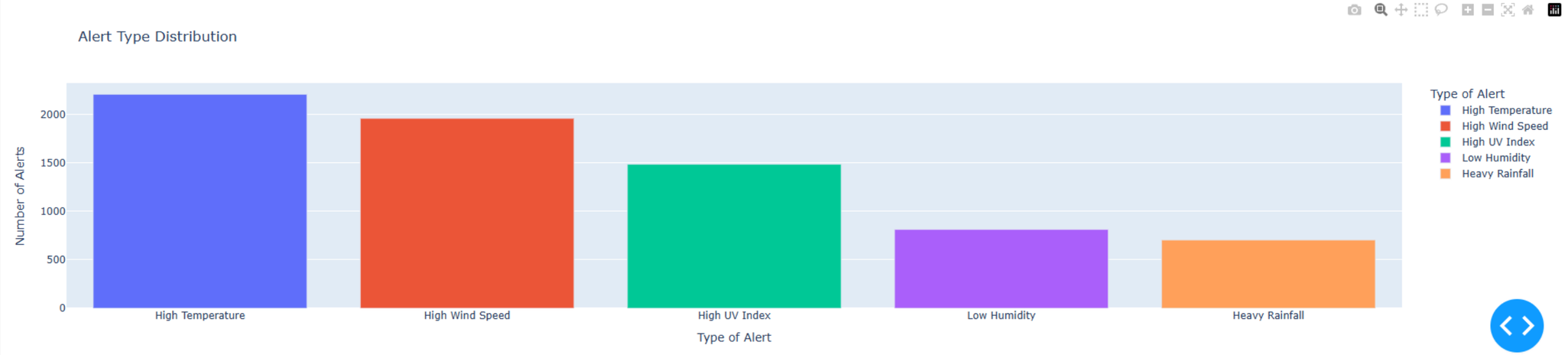




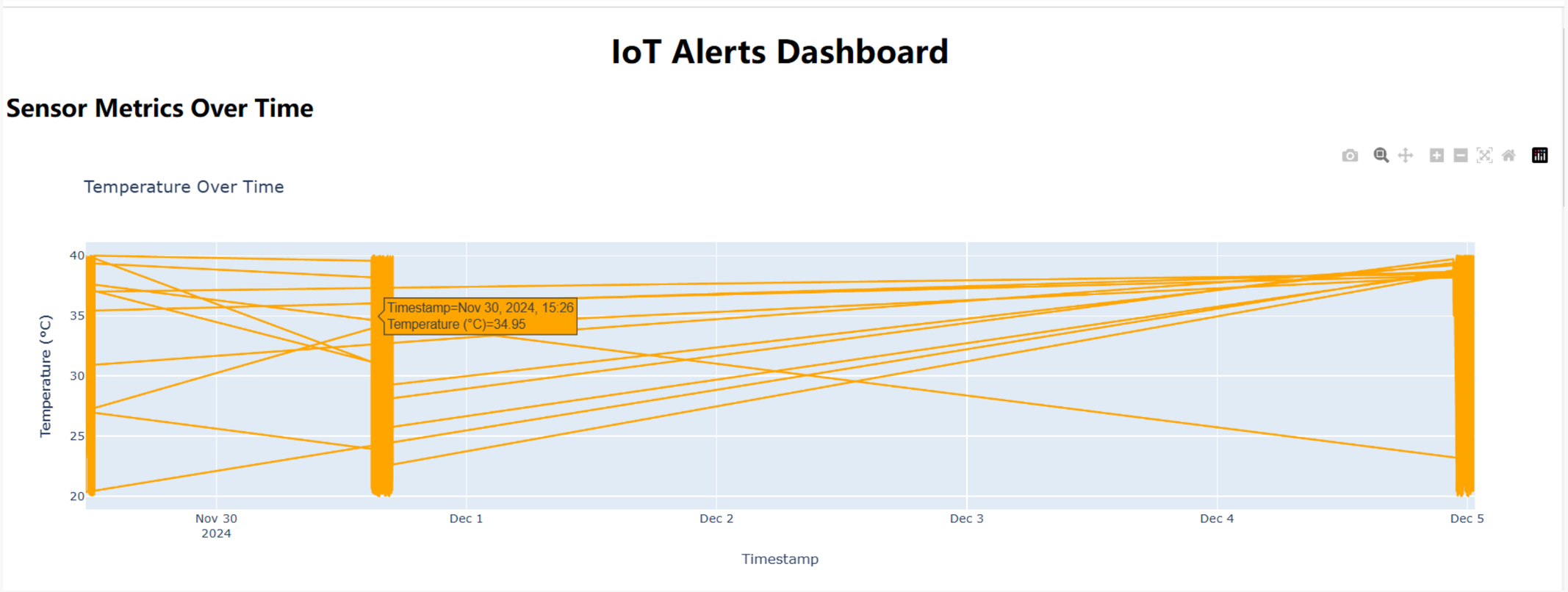
Visualisation



Number of Alerts



Visualisation



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.

Merci de votre attention

Avez-vous des questions?