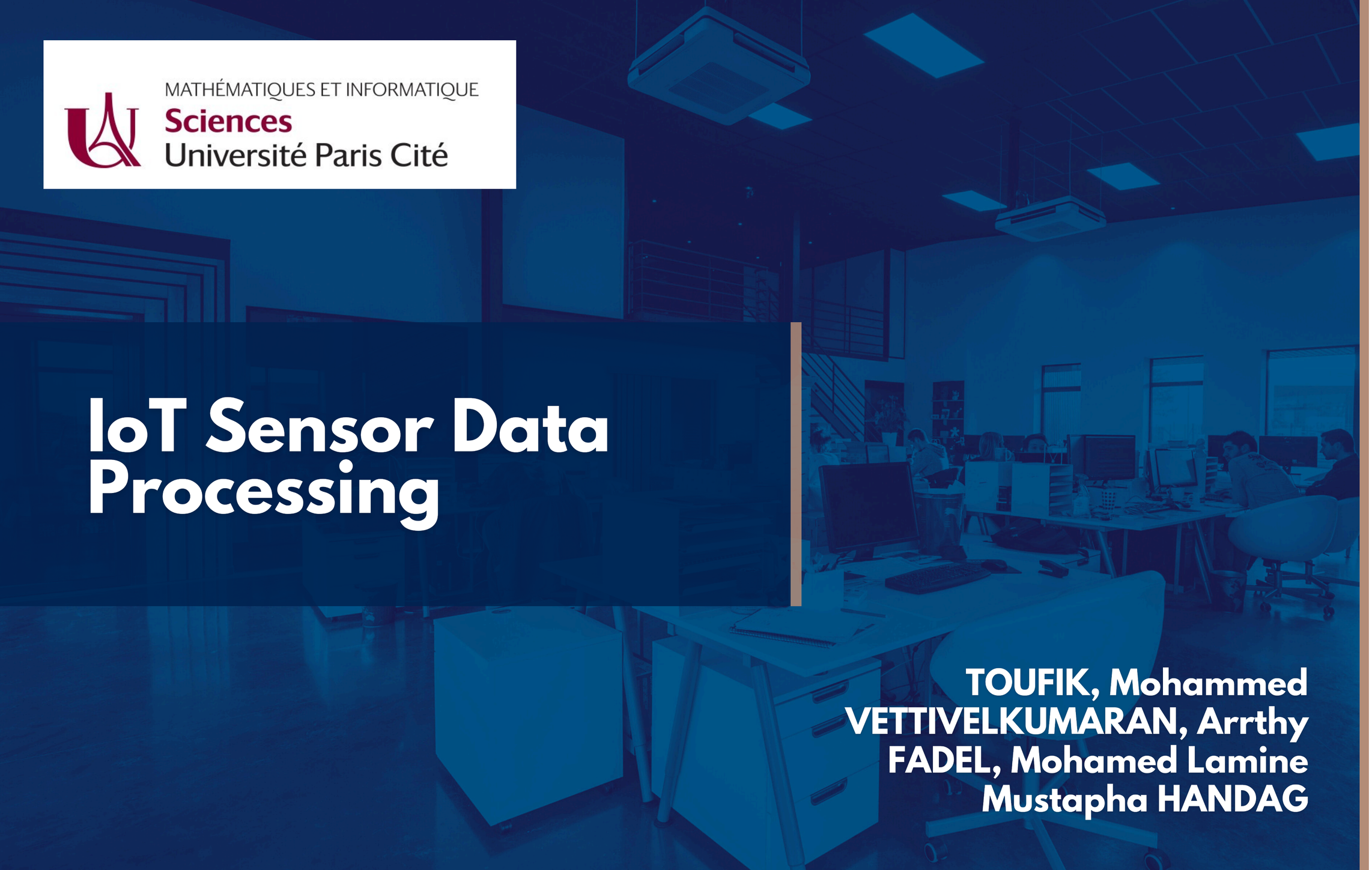


MATHÉMATIQUES ET INFORMATIQUE
Sciences
Université Paris Cité

kafka

IoT Sensor Data Processing

TOUFIK, Mohammed
VETTIVELKUMARAN, Arrthy
FADEL, Mohamed Lamine
Mustapha HANDAG

APACHE Spark™

INDEX

1. Weather IoT

What's Weather IoT

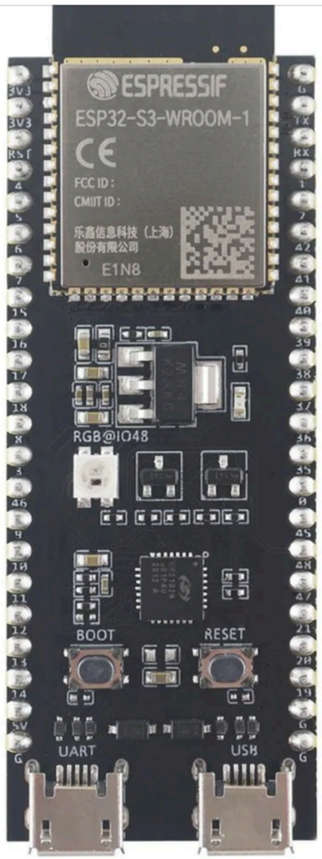
- Weather IoT (Internet of Things) refers to the application of IoT technologies to monitor, analyze, and manage weather-related data using interconnected devices and sensors.

2. Theory of the project (Data collection)

Hardware:

- **ESP32**

	IO Pins WROOM	Used		
22	23	3.3 V for MAC17048		3.3V
21				3.3V
20	4	EN		RST
19				IO4
18				IO5
17				IO6
16				IO7
15	15	GPS Signal UART		IO15
14	14	Temp Signal		IO16
13	13	LORA Reset		IO17
12				IO18
11	9	Relais Trigger		IO8
10				IO3
9				IO46
8				IO9
7	7	Camera MOSI		IO10
6	6	I2C SCL		IO11
5	5	I2C SDA		IO12
4				IO13
3				IO14
2	8	Battery VCC		5V
1	21	GND		GND

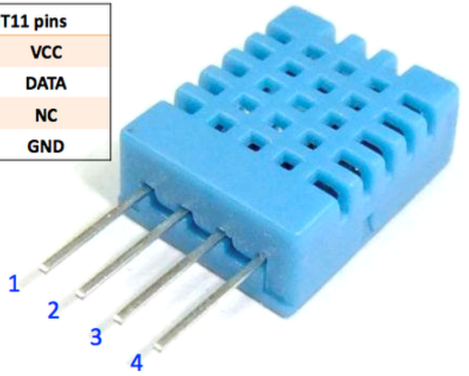


			IO Pins WROOM	
GND				22
UART TxD				21
UART RxD				20
IO1		LORA MOSI	28	19
IO2		LORA MISO	29	18
IO42		Camera SCK	26	17
IO41		Camera MISO	36	16
IO40				15
IO39				14
IO38	Board LED			13
IO37		Could not be used		12
IO36		Could not be used		11
IO35		Could not be used		10
IO0				9
IO45	Strapping Pin be careful during boot			8
IO48		Camera Select	27	7
IO47		LORA SCK	30	6
IO21		LORA CS	31	5
IO20	do not use USB			4
IO19	do not use USB			3
GND				2
GND				1

- **BH1750**

- **DHT11**

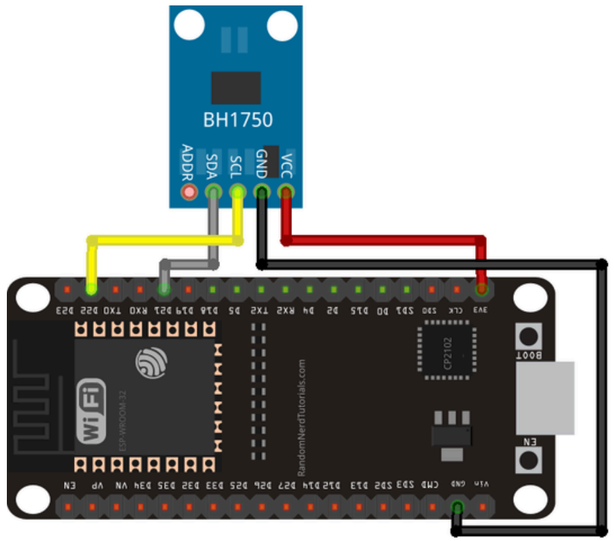
DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND

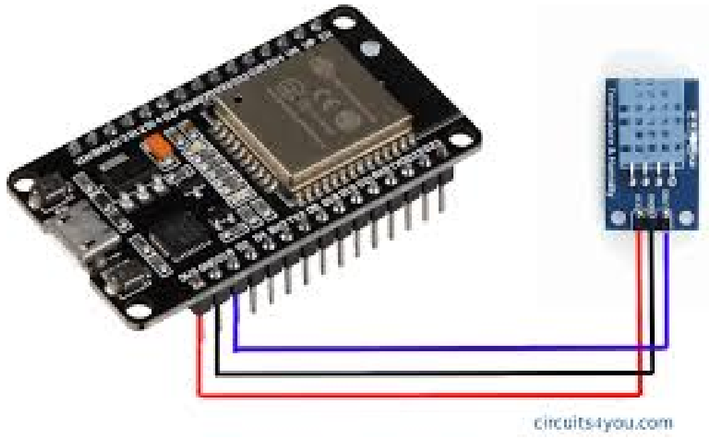


2. Theory of the project (Data collection)

1. **Data collection:**

- The BH1750 sensor, via I2C communication, would provide precise light intensity measurements.
- The DHT11 sensor, connected to GPIO 16, would collect temperature and humidity data.






```c
static char *TAG = "Main";

int delay = 10000; // 10 seconds;
void app_main()
{
    // Initialize Wi-Fi
    ESP_LOGI(TAG, "Initializing the wifi...");
    init_wifi();
    // Initialize SNTP and synchronize time
    ESP_LOGI(TAG, "Initializing the sntp...");
    init_sntp();
    // Initialize MQTT
    ESP_LOGI(TAG, "Initializing the MQTT...");
    init_mqtt();
    ESP_LOGI(TAG, "Initializing the BH1750 sensor...");
    bh1750_init();
    // TODO: Measure temperature and humidity in fixed intervals
    // TODO: Publish temperature and humidity to MQTT broker
    dht_sensor_type_t sensor = DHT_TYPE_DHT11;
    float luxval =0;
    char message[200];
    while (1)
    {
        ESP_LOGI(TAG, "Reading sensor info...(BH1750)");
        luxval = bh1750_read();
        ESP_LOGI(TAG, "Reading sensor info...(Temp/Hum)");
        float temperature, humidity;
        while( dht_read_float_data(sensor, CONFIG_DHT_PIN, &humidity, &temperature) != ESP_OK)

        ESP_LOGI(TAG, "temperature: %.2f", temperature);
        ESP_LOGI(TAG, "humidity: %.2f", humidity);
        mqtt_publish_message(message, temperature, humidity);
        int new_delay = fetch();
        delay = new_delay;
        vTaskDelay(pdMS_TO_TICKS(delay)); // Delay for 10 seconds
    }
}
```

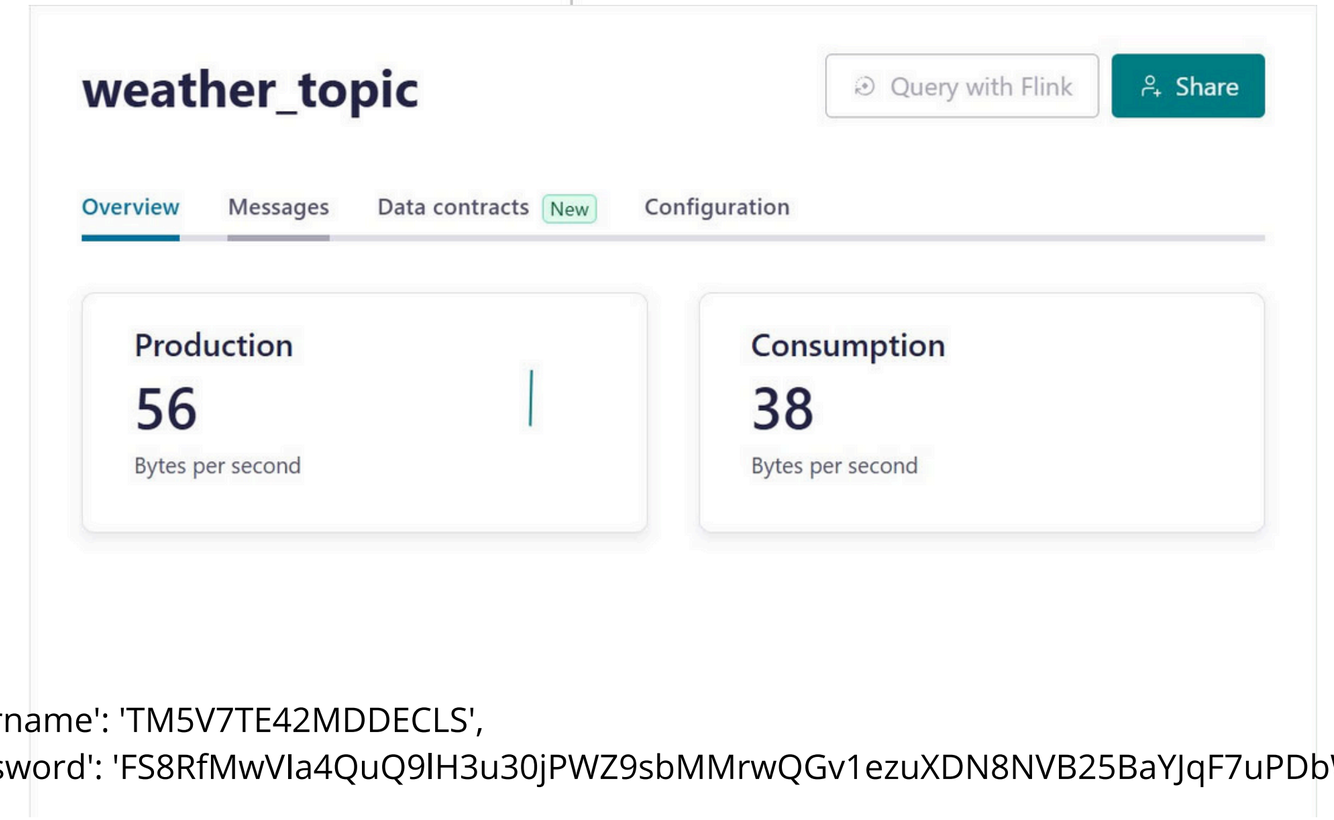
2. Real Situation

As we don’t have the hardware, we simulated the data with the producer code. It generates simulation data for our project.

```python
# Fonction pour simuler des données de capteurs
def simulate_sensor_data(limit=100):  # Limite par défaut à 100 envois
    wind_directions = ['N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW']
    for i in range(limit):
        data = {
            "id":str(random.randint(1000, 9999)),
            "temperature": round(random.uniform(-10, 40), 2),
            "humidity": round(random.uniform(20, 100), 2),
            "air_pressure": round(random.uniform(950, 1050), 2),
            "wind_speed": round(random.uniform(0, 25), 2),
            "wind_direction": random.choice(wind_directions),
            "precipitation": round(random.uniform(0, 50), 2),
            "carbon_dioxide_level": round(random.randint(200,400),2),
            "timestamp": time.time()
        }
        producer.produce("weather_topic",key=data["id"], value=json.dumps(data), callback=delivery_report)
        print(f"Envoyé ({i+1}/{limit}) : {data}")
        time.sleep(5)  # Simulation d'une nouvelle mesure chaque seconde

    print("Simulation terminée après l'envoi de", limit, "messages.")
    producer.flush()
# Simulation des données de capteurs avec une limite de 50 messages
try:
    simulate_sensor_data(limit=50)
except KeyboardInterrupt:
    print("Interruption du programme.")
finally:
    producer.flush()  # Fermer proprement le producer
```

CONFLUENT

kafka



'sasl.username': 'TM5V7TE42MDDECLS',
'sasl.password': 'FS8RfMwVIa4QuQ9lH3u30jPWZ9sbMMrwQGv1ezuXDN8NVB25BaYJqF7uPDbWXe5g',

3. Sending Data to Kafka

```python
# Kafka producer configuration
producer_conf = {
    'bootstrap.servers': 'pkc-e0zxq.eu-west-3.aws.confluent.cloud:9092',
    'security.protocol': 'SASL_SSL',
    'sasl.mechanisms': 'PLAIN',
    'sasl.username': 'TM5V7TE42MDDECLS',
    'sasl.password': 'FS8RfMwVIa4QuQ9lH3u30jPWZ9sbMMrwQGv1ezuXDN8NVB25BaYJqF7uPDbWXe5g',
    'session.timeout.ms': 45000
}
```

```python
def simulate_sensor_data(limit=100):  # Limite par défaut à 100 envois
    wind_directions = ['N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW']
    for i in range(limit):
        data = {
            "id":str(random.randint(1000, 9999)),
            "temperature": round(random.uniform(-10, 40), 2),
            "humidity": round(random.uniform(20, 100), 2),
            "air_pressure": round(random.uniform(950, 1050), 2),
            "wind_speed": round(random.uniform(0, 25), 2),
            "wind_direction": random.choice(wind_directions),
            "precipitation": round(random.uniform(0, 50), 2),
            "timestamp": time.time()
        }
```

```
Envoyé (1/50) : {'id': '3074', 'temperature': -7.79, 'humidity': 98.1, 'air_pressure': 1007.97, 'wind_speed': 18.58, 'wind_direction': 'W',
Envoyé (2/50) : {'id': '7041', 'temperature': 12.69, 'humidity': 45.64, 'air_pressure': 1000.08, 'wind_speed': 13.56, 'wind_direction': 'SW'
Envoyé (3/50) : {'id': '3421', 'temperature': 33.27, 'humidity': 37.41, 'air_pressure': 1026.34, 'wind_speed': 4.36, 'wind_direction': 'W',
Envoyé (4/50) : {'id': '9837', 'temperature': 8.12, 'humidity': 75.36, 'air_pressure': 1044.58, 'wind_speed': 0.29, 'wind_direction': 'SW',
Envoyé (5/50) : {'id': '1995', 'temperature': 9.64, 'humidity': 82.18, 'air_pressure': 983.35, 'wind_speed': 16.27, 'wind_direction': 'SE',
Envoyé (6/50) : {'id': '1659', 'temperature': 22.09, 'humidity': 80.18, 'air_pressure': 995.45, 'wind_speed': 14.59, 'wind_direction': 'SW',
Envoyé (7/50) : {'id': '7707', 'temperature': 23.7, 'humidity': 98.91, 'air_pressure': 1038.04, 'wind_speed': 3.69, 'wind_direction': 'SE',
Envoyé (8/50) : {'id': '6851', 'temperature': 13.32, 'humidity': 47.32, 'air_pressure': 1012.35, 'wind_speed': 6.59, 'wind_direction': 'E',
Envoyé (9/50) : {'id': '8504', 'temperature': 11.14, 'humidity': 61.23, 'air_pressure': 1022.8, 'wind_speed': 21.44, 'wind_direction': 'NE',
Envoyé (10/50) : {'id': '3828', 'temperature': 35.1, 'humidity': 71.13, 'air_pressure': 993.2, 'wind_speed': 3.91, 'wind_direction': 'NE', '
```

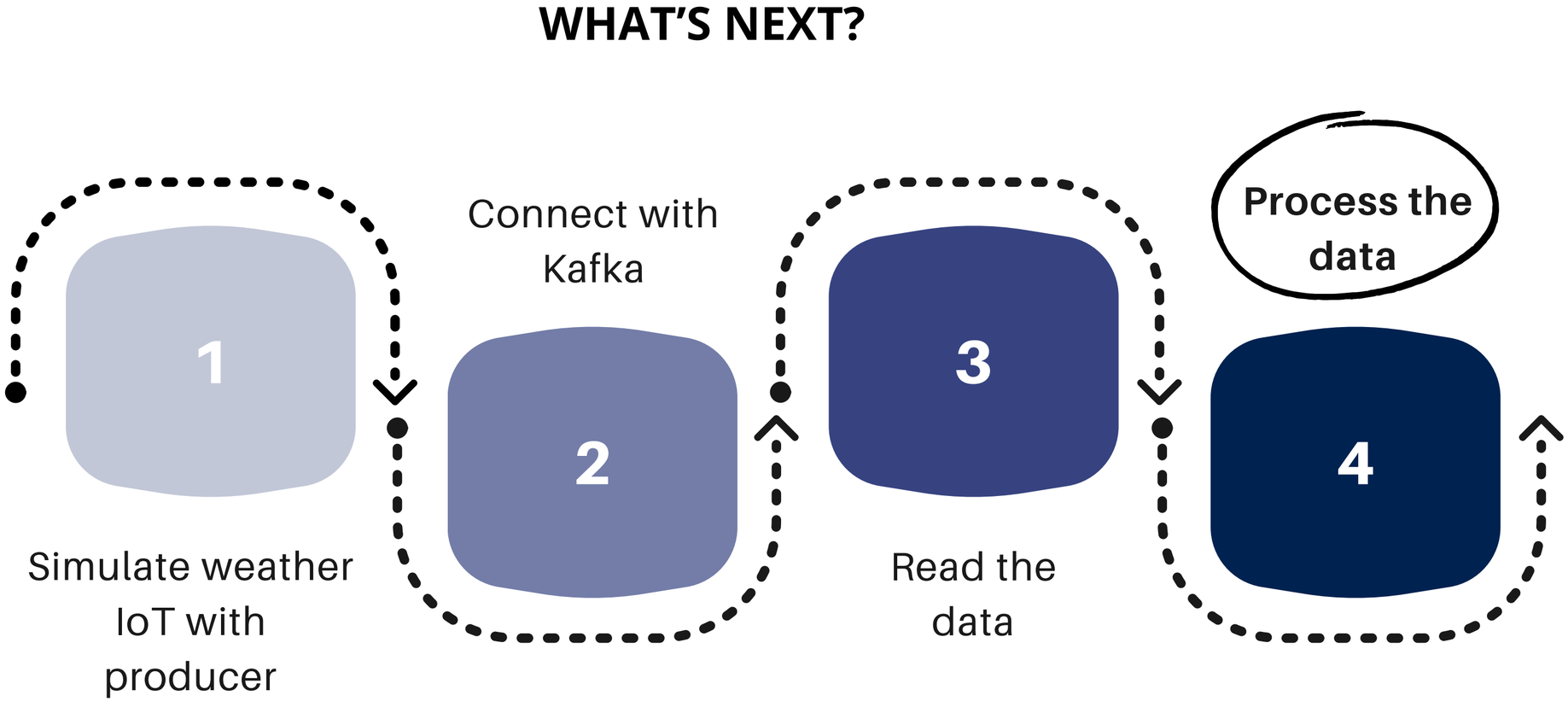
4. Reading Data with Spark

```
raw_stream = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", kafka_bootstrap_servers) \
    .option("subscribe", "topic3") \
    .option("kafka.security.protocol", "SASL_SSL") \
    .option("kafka.sasl.mechanism", "PLAIN") \
    .option("kafka.sasl.username", kafka_username) \
    .option("kafka.sasl.password", kafka_password) \
    .load()
```

readStream :This method is used to read streaming data, which means that Spark will receive the data as it is sent to Kafka.

Data Filtering, Processing, and Structuring

5 Data Filtering, Processing, and Structuring



5. Data Filtering, Processing, and Structuring

What are we looking for?

A method to receive alerts whenever anomalies are detected.

detect_anomalies function :

- checks several weather conditions and detects anomalies based on predefined thresholds
- returns a list of anomalies in the form of strings

```python
# Fonction pour détecter les anomalies dans les données météo
def detect_anomalies(data):
    anomalies = []

    if data.get("temperature") > 35:
        anomalies.append("High Temperature")
    if data.get("humidity") < 30:
        anomalies.append("Low Humidity")
    if data.get("air_pressure") < 980:
        anomalies.append("Low Air Pressure")
    if data.get("wind_speed") > 20:
        anomalies.append("High Wind Speed")
    if data.get("precipitation") > 30:
        anomalies.append("Heavy Precipitation")

    return anomalies
```

5. Data Filtering, Processing, and Structuring

Consuming Kafka messages :

- waits for up to 1 second to receive a message
- no message is received
  - continues the loop and checks again for messages
- message is valid :
  - decodes message (bytes to string)
  - call detect_anomalies() to check anomalies
  - If anomalies are detected -> prints the data and lists the anomalies

```
# Écoute des messages Kafka
try:
    while True:
        msg = consumer.poll(timeout=1.0)  # Timeout de 1 seconde
        if msg is None:
            continue
        if msg.error():
            if msg.error().code() == KafkaError._PARTITION_EOF:
                print(f"End of partition reached {msg.partition} {msg.offset}")
            else:
                raise KafkaException(msg.error())
        else:
            data = json.loads(msg.value().decode('utf-8'))
            anomalies = detect_anomalies(data)
            if anomalies:
                print(f"=== Anomalie détectée ===")
                print(f"  Données : {data}")
                for anomaly in anomalies:
                    print(f"  - {anomaly}")
except KeyboardInterrupt:
    pass
finally:
    consumer.close()
```

5. Data Filtering, Processing, and Structuring

Is it the only way ?

Here is another method using the Spark Structured Streaming pipeline to process streaming data

```
# Détecter les dépassements de seuils
alerts = sensor_data.withColumn(
    "alert",
    expr("""
        CASE
            WHEN temperature > 35 THEN 'High Temperature'
            WHEN temperature < -5 THEN 'Low Temperature'
            WHEN humidity < 30 THEN 'Low Humidity'
            WHEN air_pressure < 980 THEN 'Low Air Pressure'
            WHEN wind_speed > 20 THEN 'High Wind Speed'
            WHEN precipitation > 30 THEN 'Heavy Precipitation'
            ELSE NULL
        END
    """)
).filter("alert IS NOT NULL")
```

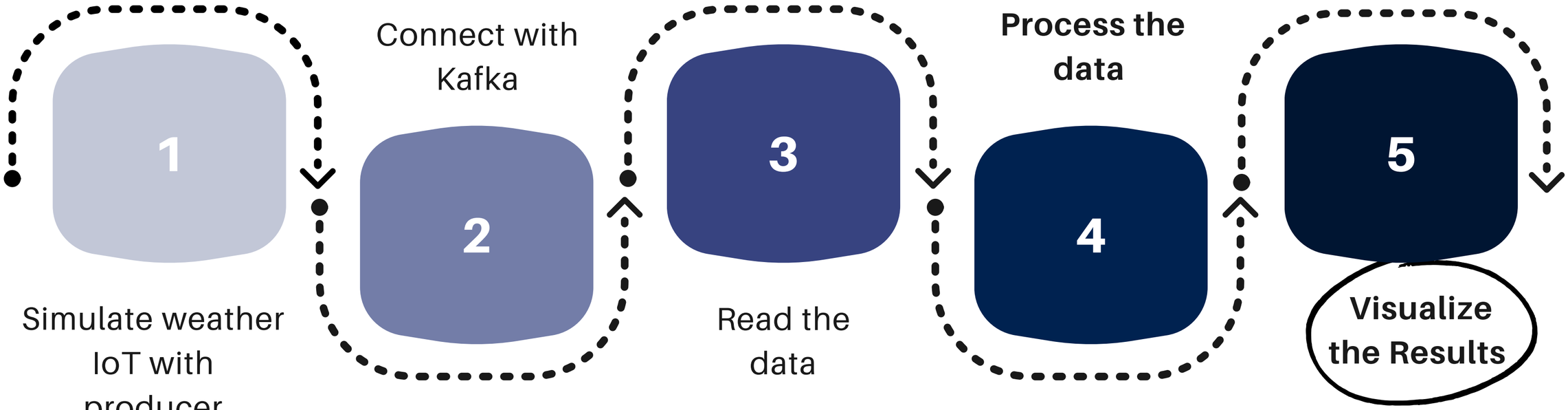
```
# Configurer l'écriture des alertes dans un fichier JSON
query = alert_columns.writeStream \
    .outputMode("append") \
    .format("csv") \
    .option("path", "output/alerts") \
    .option("checkpointLocation", "output/checkpoints") \
    .trigger(processingTime="2 seconds") \
    .start()

print("Démarrage du streaming Spark : écriture des alertes dans un fichier CSV...")

#attendre la fin de la requête # Lancer la requête de streaming avec le mode "complete"
try:
    Timer(50, query.stop).start()
    query.awaitTermination()
except Exception as e:
    print(f"Error occurred: {e}")
```

DEMO

Results and Visualization

6 Results and Visualization

AND THEN ?



6 Results and Visualization

```
Kafka Consumer démarré. Recherche des anomalies...
=== Anomalie détectée ===
  Données : {'id': '8469', 'temperature': 31.35, 'hum
  - Low Humidity
  - Heavy Precipitation
=== Anomalie détectée ===
  Données : {'id': '6683', 'temperature': 33.5, 'humi
  - Low Humidity
  - Heavy Precipitation
=== Anomalie détectée ===
  Données : {'id': '2868', 'temperature': 4.9, 'humid
  - High Wind Speed
  - Heavy Precipitation
=== Anomalie détectée ===
  Données : {'id': '7614', 'temperature': 25.09, 'hum
  - Heavy Precipitation
=== Anomalie détectée ===
  Données : {'id': '3306', 'temperature': 11.07, 'hum
  - Heavy Precipitation
=== Anomalie détectée ===
  Données : {'id': '6481', 'temperature': 38.81, 'hum
  - High Temperature
```

Result obtained with the consumer :

- Kafka Consumer keeps running until manually interrupted or an error occurs
- if an anomaly is detected it will print :

```
=== Anomalie détectée ===
  Données : {data}
  - Anomaly 1
  - Anomaly 2
==========================
```

- Problem: Too much information cannot be easily overviewed.
- But it is good if we only want to know each alert as it happens.

6 Results and Visualization

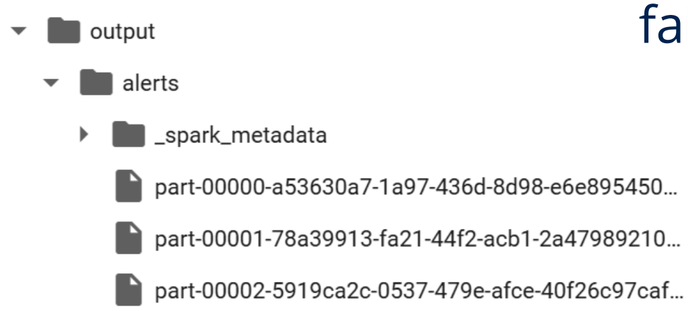
30.32	23.49	1037.94	21.19	W	42.72	Low Humidity
7.28	60.07	1043.99	17.97	NE	33.27	Heavy Precipitation
15.29	41.76	959.48	19.5	W	0.96	Low Air Pressure
29.49	64.52	1035.9	20.12	NW	25.15	High Wind Speed
34.31	72.79	952.09	22.84	NW	18.09	Low Air Pressure
17.91	22.53	981.65	20.79	SE	1.74	Low Humidity
-1.79	80.93	1047.15	21.95	N	47.3	High Wind Speed
2.25	66.08	974.93	24.44	S	0.69	Low Air Pressure
5.99	74.96	951.23	21.63	N	29.27	Low Air Pressure
21.48	82.89	1045.52	23.81	W	48.95	High Wind Speed
27.65	34.42	960.52	7.62	SE	38.71	Low Air Pressure
31.81	54.82	1035.02	7.27	NE	35.62	Heavy Precipitation
1.4	26.68	963.5	6.85	NE	2.12	Low Humidity
-1.2	43.88	1005.33	14.94	SE	47.33	Heavy Precipitation
0.88	91.67	983.18	20.53	N	37.11	High Wind Speed
-5.54	87.06	995.8	23.15	SW	44.69	High Wind Speed
27.12	39.51	1044.53	0.85	NW	48.76	Heavy Precipitation
9.99	66.14	1006.67	20.59	SE	10.64	High Wind Speed
1.34	96.46	1043.48	0.25	N	37.67	Heavy Precipitation
30.16	73.29	994.99	20.86	SE	0.43	High Wind Speed

Result obtained with spark :

- Output consists of multiple CSV files in the output/alerts directory, each containing alert data for 2-second intervals.
- The checkpoint information is saved in output/checkpoints to ensure the stream can resume correctly in case of failure.

Benefit:

- Easy Data Accessibility
- Historical Data Storage
- Flexibility for Further Processing



Challenges and Solutions

The issues encountered

1. ***Installation of the wrong version of Kafka*** (V 3.5.3 instead of 3.3.0): The connection to Kafka data did not work properly. The data producers and consumers were not synchronized with the correct versions, which prevented the establishment of the connection between the producer (which sends the messages) and the consumer (which receives and processes them).

The solution implemented: To resolve this issue, the solution was to uninstall Kafka 3.5.3 and install version 3.3.0, which was specifically compatible with my configurations.

The issues encountered

2.Data sent by the producer in JSON format and not processed by the consumer
The producer was sending the messages in JSON format. However, the consumer could not read these messages directly in JSON format, which caused errors during data processing.

The consumer failed to decode the messages sent in JSON because it was expecting another format.

The solution was to add this line to convert the message:

```
data = json.loads(msg.value().decode('utf-8'))
```

The issues encountered

3.Problem with writing messages as they are analyzed

While analyzing Kafka messages, it was observed that these messages were not being written in real-time; that is, they were processed and recorded as they were received, but they did not display in real-time.

To resolve this issue, we used a consumer that allowed us to display the results in real-time.

```
Version de Spark: 3.3.0
Packages configurés : org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.3
Connection successful!
Démarrage du streaming...
Batch 0 - Nombre de lignes: 675
+-----------+--------+------------+----------+--------------+-------------+---------+-------------------+
|temperature|humidity|air_pressure|wind_speed|wind_direction|precipitation|timestamp|              alert|
+-----------+--------+------------+----------+--------------+-------------+---------+-------------------+
|       6.13|   81.54|       966.8|     11.11|            SW|        36.78|     null|   Low Air Pressure|
|      30.32|   23.49|      1037.94|     21.19|             W|        42.72|     null|       Low Humidity|
|       7.28|   60.07|      1043.99|     17.97|            NE|        33.27|     null|Heavy Precipitation|
|      15.29|   41.76|       959.48|      19.5|             W|         0.96|     null|   Low Air Pressure|
|      29.49|   64.52|       1035.9|     20.12|            NW|        25.15|     null|    High Wind Speed|
|      34.31|   72.79|       952.09|     22.84|            NW|        18.09|     null|   Low Air Pressure|
|      17.91|   22.53|       981.65|     20.79|            SE|         1.74|     null|       Low Humidity|
|      -1.79|   80.93|      1047.15|     21.95|             N|         47.3|     null|    High Wind Speed|
|       2.25|   66.08|       974.93|     24.44|             S|         0.69|     null|   Low Air Pressure|
|       5.99|   74.96|       951.23|     21.63|             N|        29.27|     null|   Low Air Pressure|
|      21.48|   82.89|      1045.52|     23.81|             W|        48.95|     null|    High Wind Speed|
|      27.65|   34.42|       960.52|      7.62|            SE|        38.71|     null|   Low Air Pressure|
|      31.81|   54.82|      1035.02|      7.27|            NE|        35.62|     null|Heavy Precipitation|
|        1.4|   26.68|        963.5|      6.85|            NE|         2.12|     null|       Low Humidity|
|       -1.2|   43.88|      1005.33|     14.94|            SE|        47.33|     null|Heavy Precipitation|
|       0.88|   91.67|       983.18|     20.53|             N|        37.11|     null|    High Wind Speed|
|      -5.54|   87.06|       995.8|     23.15|            SW|        44.69|     null|    High Wind Speed|
|      27.12|   39.51|      1044.53|      0.85|            NW|        48.76|     null|Heavy Precipitation|
|       9.99|   66.14|      1006.67|     20.59|            SE|        10.64|     null|    High Wind Speed|
|       1.34|   96.46|      1043.48|      0.25|             N|        37.67|     null|Heavy Precipitation|
+-----------+--------+------------+----------+--------------+-------------+---------+-------------------+
only showing top 20 rows
```

The issues encountered

4. Problem with adding headers in CSV files : When creating a CSV file, we wanted to include headers to facilitate the reading and interpretation of data. However, despite several attempts, we were unable to add these headers to the CSV file at the time of its creation.

I tried to add the headers before writing the data to the CSV file, but it did not work.

1 to 10 of 25 entries Filter

13.82	29.23	978.1	6.16	W	7.17	Low Humidity
-7.74	40.09	1045.8	12.75	NE	19.73	Low Temperature
21.26	98.72	969.54	6.08	N	26.72	Low Air Pressure
0.99	36.41	962.39	8.85	N	15.11	Low Air Pressure
0.89	70.47	979.74	0.36	N	1.55	Low Air Pressure
18.51	27.97	1023.26	20.37	SW	4.89	Low Humidity
7.83	23.64	987.09	14.59	NW	37.95	Low Humidity

Conclusion

Conclusion

- The simulated sensors generate varied data (temperature, humidity, pressure, etc.), mimicking real-world environmental monitoring scenarios.
- This demonstrates the ability of IoT systems to collect diverse and relevant data for applications in the meteorological or industrial fields.
- The sensor messages are transmitted in real-time via Kafka, illustrating a robust infrastructure for handling large and continuous data streams.
- By using Spark, we can process the collected data and generate alerts for each meteorological factor.

QUESTIONS ?

ANNEX

For more information:

Weather IoT

DHT11 & ESP32

BH1750 & ESP32

DHT11 Technical Data Sheet

BH1750 Technical Data Sheet

ESP32-S3 Technical Data Sheet