

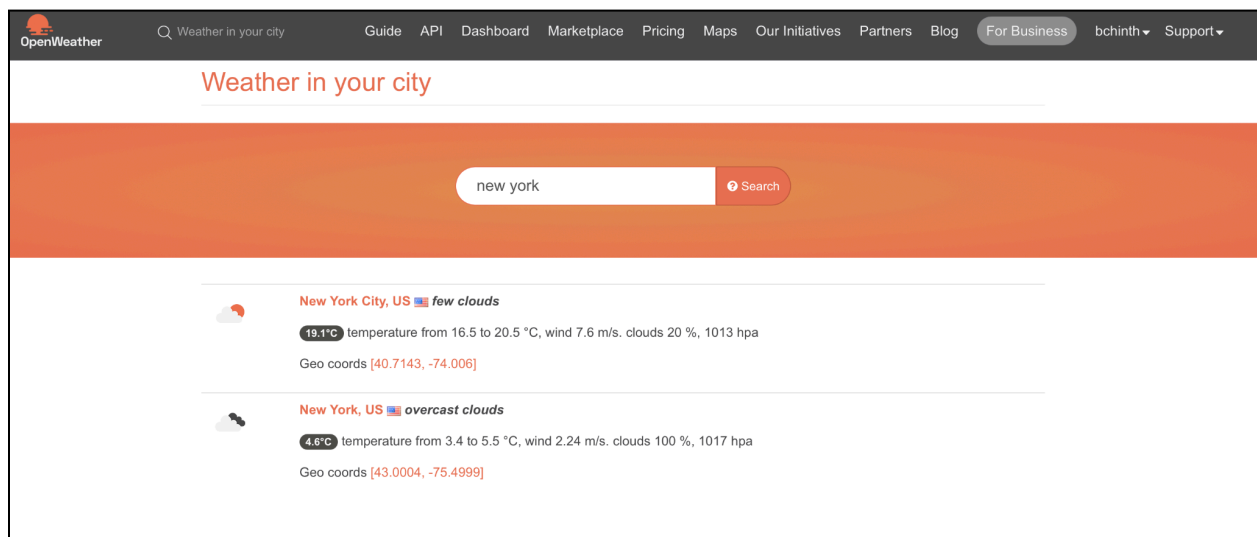
Kafka Weather Data Streaming Report

Team: Big Data Bandits

Blessy Chinthapalli, Meghana Kanthadai, Rachita Harit, Shriya Yegalapati

Introduction

In this project, we established a foundation for working with Apache Kafka, a powerful distributed streaming platform. By setting up Kafka on a Mac environment and developing Python scripts for a producer and consumer, we created a data pipeline that streams weather data for New York City. The data is fetched from the OpenWeatherMap API at regular intervals.



Installation and Kafka Setup

Kafka was installed using Homebrew, a package manager for Mac, following these steps:

1. Install Homebrew: Using the command line, Homebrew was installed.
2. Install Kafka: Kafka, along with its dependency ZooKeeper, was installed via Homebrew.
3. Configure PATH: The installation path of Kafka was added to the system's PATH environment variable to allow easy execution of Kafka commands.

```
brew install kafka
% cd /opt/homebrew/opt/kafka/bin/
ls
```

```

Last login: Wed Apr 24 14:03:11 on ttys005
(base) blessy@Blessys-MacBook-Air ~ % brew install kafka
zsh: command not found: brew
(base) blessy@Blessys-MacBook-Air ~ % brew install kafka
==> Downloading https://formulae.brew.sh/api/formula.jws.json
##### 100.0%
==> Downloading https://formulae.brew.sh/api/cask.jws.json
##### 100.0%
Warning: kafka 3.7.0 is already installed and up-to-date.
To reinstall 3.7.0, run:
  brew reinstall kafka
(base) blessy@Blessys-MacBook-Air ~ % brew --prefix kafka
/opt/homebrew/opt/kafka
(base) blessy@Blessys-MacBook-Air ~ % export PATH="$(brew --prefix kafka)/bin:$PATH"
(base) blessy@Blessys-MacBook-Air ~ % PATH="/opt/homebrew/opt/kafka/3.7.0/bin:$PATH"
(base) blessy@Blessys-MacBook-Air ~ % source ~/.bash_profile
(base) blessy@Blessys-MacBook-Air ~ % cd /opt/homebrew/opt/kafka/bin/
(base) blessy@Blessys-MacBook-Air bin % ls
connect-distributed      kafka-log-dirs
connect-mirror-maker    kafka-metadata-quorum
connect-plugin-path      kafka-metadata-shell
connect-standalone      kafka-mirror-maker
kafka-acls               kafka-producer-perf-test
kafka-broker-api-versions kafka-reassign-partitions
kafka-client-metrics     kafka-replica-verification
kafka-cluster            kafka-run-class
kafka-configs             kafka-server-start
kafka-console-consumer   kafka-server-stop
kafka-console-producer    kafka-storage
kafka-consumer-groups    kafka-streams-application-reset
kafka-consumer-perf-test  kafka-topics
kafka-delegation-tokens   kafka-transactions
kafka-delete-records       kafka-verifiable-consumer
kafka-dump-log             kafka-verifiable-producer
kafka-e2e-latency          trogdor
kafka-features            zookeeper-security-migration
kafka-get-offsets         zookeeper-server-start
kafka-jmx                 zookeeper-server-stop
kafka-leader-election     zookeeper-shell
(base) blessy@Blessys-MacBook-Air bin %

```

ZooKeeper and Kafka services were initiated in separate terminal windows, establishing the backbone for our Kafka cluster.

```

zookeeper-server-start /opt/homebrew/etc/kafka/zookeeper.properties
kafka-server-start /opt/homebrew/etc/kafka/server.properties

```

Kafka Topic Creation:

A Kafka topic named "weather" was created to hold the streaming weather data. The topic was set up with one partition and one replication factor, suitable for development and testing.

```
kafka-topics --list --bootstrap-server localhost:9092
```

```

Requirement already satisfied: kafka-python in /anaconda3/lib/python3.11/site-packages (2.0.
(base) blessy@Blessys-MacBook-Air ~ % kafka-topics --list --bootstrap-server localhost:9092
__consumer_offsets
test
weather

```

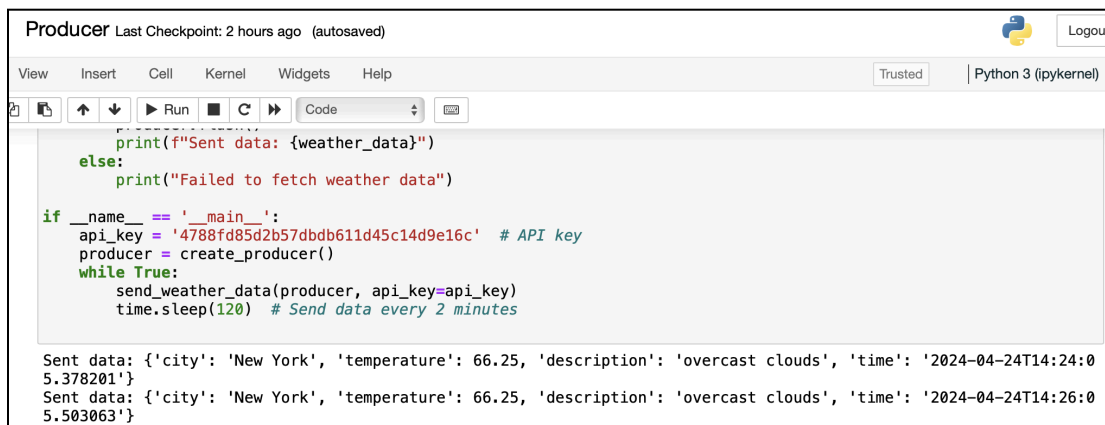
Producer Script Development:

The Kafka producer script was written in Python. It uses the `kafka-python` library to connect to the Kafka instance and send data to the "weather" topic. The weather data, including temperature and weather conditions, is fetched from the OpenWeatherMap API for New York.

Key Components of the Producer Script:

- **KafkaProducer Configuration:** The producer is configured to serialize JSON data and connect to Kafka on the default port 9092.
- **Data Fetching:** The script periodically makes API calls to OpenWeatherMap to retrieve real-time weather data.
- **Message Sending:** The script sends the fetched data as messages to the Kafka "weather" topic.

USING API KEY (Real - Time):



```
Producer Last Checkpoint: 2 hours ago (autosaved)

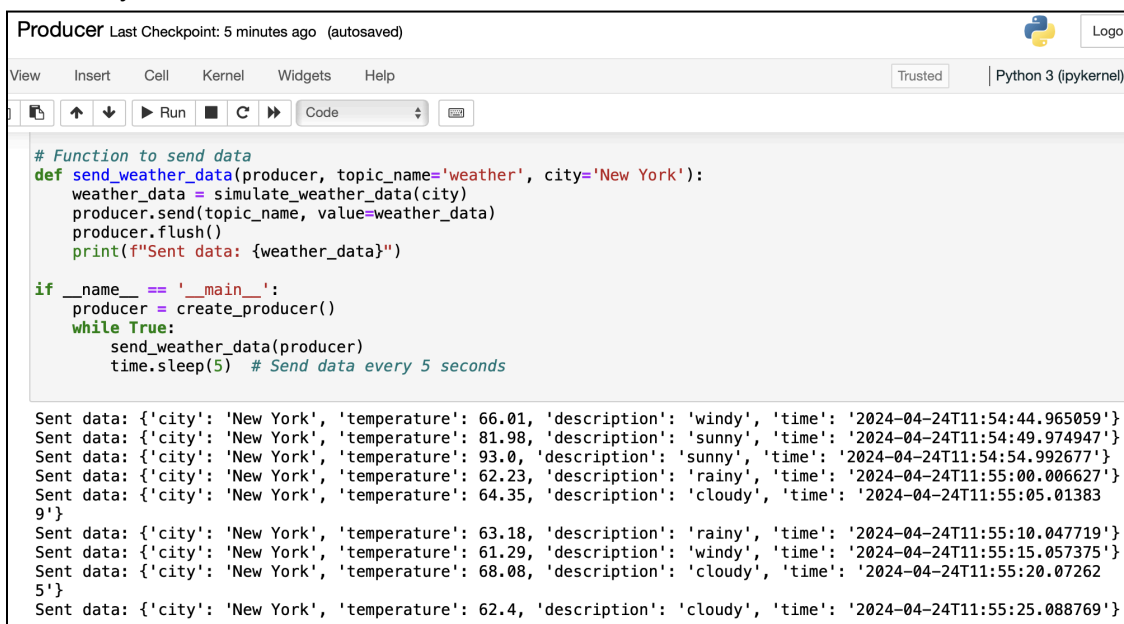
View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

print(f"Sent data: {weather_data}")
else:
    print("Failed to fetch weather data")

if __name__ == '__main__':
    api_key = '4788fd85d2b57dbdb611d45c14d9e16c' # API key
    producer = create_producer()
    while True:
        send_weather_data(producer, api_key=api_key)
        time.sleep(120) # Send data every 2 minutes

Sent data: {'city': 'New York', 'temperature': 66.25, 'description': 'overcast clouds', 'time': '2024-04-24T14:24:05.378201'}
Sent data: {'city': 'New York', 'temperature': 66.25, 'description': 'overcast clouds', 'time': '2024-04-24T14:26:05.503063'}
```

Randomly Generated Weather Data:



```
Producer Last Checkpoint: 5 minutes ago (autosaved)

View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

# Function to send data
def send_weather_data(producer, topic_name='weather', city='New York'):
    weather_data = simulate_weather_data(city)
    producer.send(topic_name, value=weather_data)
    producer.flush()
    print(f"Sent data: {weather_data}")

if __name__ == '__main__':
    producer = create_producer()
    while True:
        send_weather_data(producer)
        time.sleep(5) # Send data every 5 seconds

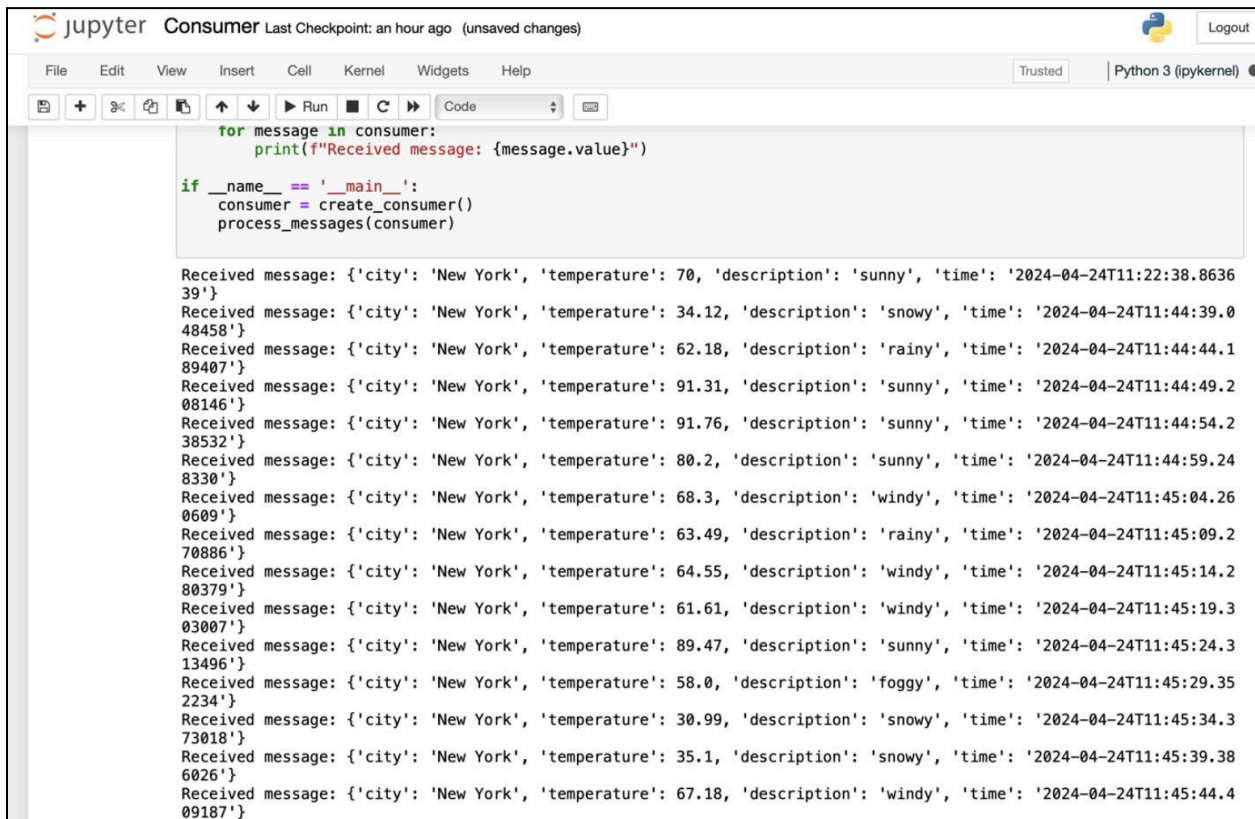
Sent data: {'city': 'New York', 'temperature': 66.01, 'description': 'windy', 'time': '2024-04-24T11:54:44.965059'}
Sent data: {'city': 'New York', 'temperature': 81.98, 'description': 'sunny', 'time': '2024-04-24T11:54:49.974947'}
Sent data: {'city': 'New York', 'temperature': 93.0, 'description': 'sunny', 'time': '2024-04-24T11:54:54.992677'}
Sent data: {'city': 'New York', 'temperature': 62.23, 'description': 'rainy', 'time': '2024-04-24T11:55:00.006627'}
Sent data: {'city': 'New York', 'temperature': 64.35, 'description': 'cloudy', 'time': '2024-04-24T11:55:05.013839'}
Sent data: {'city': 'New York', 'temperature': 63.18, 'description': 'rainy', 'time': '2024-04-24T11:55:10.047719'}
Sent data: {'city': 'New York', 'temperature': 61.29, 'description': 'windy', 'time': '2024-04-24T11:55:15.057375'}
Sent data: {'city': 'New York', 'temperature': 68.08, 'description': 'cloudy', 'time': '2024-04-24T11:55:20.072625'}
Sent data: {'city': 'New York', 'temperature': 62.4, 'description': 'cloudy', 'time': '2024-04-24T11:55:25.088769'}
```

Consumer Script Development:

A Python script acts as a Kafka consumer and is responsible for reading the messages/data sent by the producer. For scalability and advanced processing, PySpark was used to set up the streaming context.

Key Components of the Consumer Script:

- **KafkaConsumer Configuration:** Utilizes the `kafka-python` library to read from the "weather" topic.
- **PySpark Streaming Context:** PySpark is used to read data from Kafka, enabling complex processing such as running machine learning algorithms on the streaming data.
- **Data Processing:** The data is processed and could be stored, aggregated, or analyzed in real-time.



The screenshot shows a Jupyter Notebook interface with a title bar that says "Consumer" and "Last Checkpoint: an hour ago (unsaved changes)". The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code cell contains the following Python script:

```
for message in consumer:
    print(f"Received message: {message.value}")

if __name__ == '__main__':
    consumer = create_consumer()
    process_messages(consumer)
```

The output of the script is displayed below the code cell, showing 15 lines of received messages. Each message is a JSON object with the following fields: city, temperature, description, and time. The messages are as follows:

```
Received message: {'city': 'New York', 'temperature': 70, 'description': 'sunny', 'time': '2024-04-24T11:22:38.863639'}
Received message: {'city': 'New York', 'temperature': 34.12, 'description': 'snowy', 'time': '2024-04-24T11:44:39.048458'}
Received message: {'city': 'New York', 'temperature': 62.18, 'description': 'rainy', 'time': '2024-04-24T11:44:44.189407'}
Received message: {'city': 'New York', 'temperature': 91.31, 'description': 'sunny', 'time': '2024-04-24T11:44:49.208146'}
Received message: {'city': 'New York', 'temperature': 91.76, 'description': 'sunny', 'time': '2024-04-24T11:44:54.238532'}
Received message: {'city': 'New York', 'temperature': 80.2, 'description': 'sunny', 'time': '2024-04-24T11:44:59.248330'}
Received message: {'city': 'New York', 'temperature': 68.3, 'description': 'windy', 'time': '2024-04-24T11:45:04.260609'}
Received message: {'city': 'New York', 'temperature': 63.49, 'description': 'rainy', 'time': '2024-04-24T11:45:09.270886'}
Received message: {'city': 'New York', 'temperature': 64.55, 'description': 'windy', 'time': '2024-04-24T11:45:14.280379'}
Received message: {'city': 'New York', 'temperature': 61.61, 'description': 'windy', 'time': '2024-04-24T11:45:19.303007'}
Received message: {'city': 'New York', 'temperature': 89.47, 'description': 'sunny', 'time': '2024-04-24T11:45:24.313496'}
Received message: {'city': 'New York', 'temperature': 58.0, 'description': 'foggy', 'time': '2024-04-24T11:45:29.352234'}
Received message: {'city': 'New York', 'temperature': 30.99, 'description': 'snowy', 'time': '2024-04-24T11:45:34.373018'}
Received message: {'city': 'New York', 'temperature': 35.1, 'description': 'snowy', 'time': '2024-04-24T11:45:39.386026'}
Received message: {'city': 'New York', 'temperature': 67.18, 'description': 'windy', 'time': '2024-04-24T11:45:44.409187'}
```

Real-world Application Scenario

The Kafka weather data streaming setup could be applied in a Smart City Initiative to manage and optimize urban services based on weather conditions. For instance, the streamed weather data could be used to adjust public transportation schedules, energy grid loads, or even inform residents and tourists about weather conditions through a mobile app.

Conclusion

This project demonstrated the capability of Kafka to facilitate real-time data streaming, and the potential of PySpark to process that data effectively. The producer and consumer scripts together create a robust system capable of handling real-time weather data streaming and processing, showcasing Kafka's utility in real-world applications.

The document highlights the technical details and the broader context of how such a data streaming setup could be employed to solve real-world problems.