Data Pipeline: Docker, Kafka, and Jupyter Notebook

1. Project Overview

In this project, we address the creation of a Big Data environment using Docker to orchestrate several essential services for processing and analyzing large volumes of data. We use Kafka as a tool for ingesting and managing data flows, Jupyter Notebook for interactivity and data analysis, and several other frameworks that are widely used in the Big Data ecosystem.
The configuration of the environment is done in an automated way using Docker and Docker Compose, which allows the creation and management of multiple containers to run the services in an efficient and isolated way. The integration between these tools provides a robust foundation for exploring data at scale, processing it in real-time, and performing detailed analysis with ease.

Tools and Skills Used:
Docker: Container platform for creating and managing isolated environments, enabling efficient execution of applications and services.
Docker Compose: Tool to define and run multi-container applications, simplifying the configuration and orchestration of services.
Apache Kafka: Distributed messaging system used for ingesting, storing, and processing large volumes of data in real time.
Jupyter Notebook: Interactive interface for data analysis, which facilitates the execution of Python code and visualization of results in a dynamic way.
Pandas: Python library for data manipulation and analysis, essential for information processing and visualization.
Matplotlib: Library for creating graphs and visualizations, allowing you to represent the distribution of logs and analyzed data.

2. Creating the Directory and Configuring the Environment

Access the Big Data Docker repository on GitHub, where all the necessary files are available for download, including the docker-compose.yml file. Download the repository to set up your big data environment. Follow the detailed instructions in the repository to complete the configuration.

GitHub Repository:

https://github.com/Gustavo-Saffiotti/bigdata_docker

3. Starting Services with Docker Compose

3.1 Initializing Kafka

Navigate to the project directory:

cd "C:\docker\bigdata_docker"

Upload Kafka's service:

docker-compose up -d kafka

3.2 Initializing Jupyter Notebook

Upload the Jupyter Notebook service:

docker-compose up -d jupyter-notebook-custom

3.3 Check Active Containers

Run the command to check the containers:

docker os

Output Example:

CONTAINER ID IMAGE COMMAND STATUS PORTS
dd860f2ccd5d fjardim/kafka "start-kafka.sh" Up 16 minutes 0.0.0.0:9092->9092/tcp
7c5671d41277 fjardim/jupyter-spark "/opt/docker/bin/ent..." Up 24 seconds 0.0.0.0:8889->8889/tcp
a3ec15345d8a fjardim/zookeeper "/bin/sh -c '/usr/sb..." Up 16 minutes 0.0.0.0:2181->2181/tcp

---

4. Kafka in Docker Environment

4.1 Accessing the Kafka Container

Enter the container:

Cmd
Docker exec -it Kafka bash

Navigate to the Kafka directory:

cd /opt/kafka/bin

4.2 Creating Topics

Create topics for testing:

./kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic lesson
./kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic test
./kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic msg
./kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic logs

List the topics created:

./kafka-topics.sh --list --zookeeper zookeeper:2181

4.3 Sending Messages

Use producer to send messages:

./kafka-console-producer.sh --broker-list kafka:9092 --topic class

Send messages like:

test
Data ingestion
new msg from console

4.4 Consuming Messages

In the Kafka container, use the consumer to view the messages:

./kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic lesson

5. Case Study - Kafka with Jupyter Notebook

5.1 Sending Messages

In Jupyter Notebook, run the following code:

```
from kafka import KafkaProducer

import time

import random

# Configure the producer

producer = KafkaProducer(bootstrap_servers=['kafka:9092'])
```

```
# Log simulation
logs=[
    "INFO - System Started",
    "WARN - High latency detected",
    "ERROR - Failed to connect to database",
    "INFO - Requisition processed",
    "ERROR - Service Unavailable"
]

# Set the number of logs to be generated
num_logs = 20# For example, 20 messages

# Send logs to Kafka
for _ in range(num_logs):
    log=random.choice(logs)
    producer.send(topic='logs', value=log.encode('utf-8'))
    print(f"Log sent: {log}")
    time.sleep(2) # Send a log every 2 seconds

print("Log shipping completed.")
```

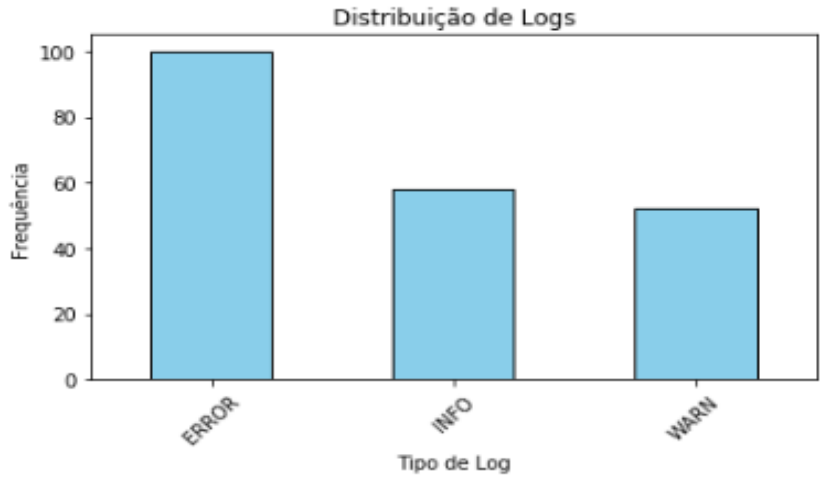
5.2 Consuming Messages

Use the code below to consume messages:

```
import pandas the pd
from kafka import KafkaConsumer
from IPython.display import display, clear_output
import matplotlib.pyplot as plt

# Configure the consumer
consumer = KafkaConsumer(
    'logs', # Topic Name
    group_id='log_group',
    bootstrap_servers=['kafka:9092'],
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    value_deserializer=lambda x: x.decode('utf-8')
)

# Initialize data list to store logs
logs_data = []

print("Consuming Kafka messages...")

try:
    for mensagem in consumer:
        log=message.value# Decode the log
        logs_data.append(log) # Add log to list

        # Create DataFrame
        df = pd. DataFrame(logs_data, columns=['Log'])

        # Update real-time display
        clear_output(wait=True) # Clear previous exit
```

```
        display(df.tail(10)) # Show last 10 logs

except KeyboardInterrupt:
    print("\nConsumption stopped by user.")

# Create distribution chart after consuming logs
if logs_data:
    # Create the 'Type' column by extracting the category from the log
    df['Type'] = df['Log'].str.split(' - ').str[0]

    # Count the frequency of each log type
    type_counts = df['Type'].value_counts()

    # Plot bar chart
    type_counts.plot(kind='bar', color='skyblue', edgecolor='black')
    plt.title("Log Distribution")
    plt.xlabel("Log Type")
    plt.ylabel("Frequency")
    plt.xticks(rotation=45) # Rotate labels for better viewing
    plt.tight_layout() # Adjust layout to avoid cropping
    plt.show()
else:
    print("No logs were consumed.")
```

```
Consumo interrompido pelo usuário.
```



6. Conclusion

With the completion of this project, it was possible to configure a Big Data environment in a practical and effective way using Docker. The integration of Kafka and Jupyter Notebook made it possible to ingest and analyze logs in real time, while the graphical visualization in matplotlib allowed for easy and immediate tracking of the processed data.

The use of Docker and Docker Compose made the configuration of services simple and reusable, making it possible to replicate the Big Data environment on any system. In addition, the implementation of a producer and consumer for Kafka demonstrated a practical application of data ingestion and analysis, with logging and error detection.

This environment provides a solid foundation for future studies, where it is possible to expand and integrate other large-scale data processing tools, such as Apache Spark or HDFS, to perform more complex analyses and handle large volumes of information efficiently.

This project demonstrates the importance of tools such as Docker and Kafka in the Big Data ecosystem, offering a scalable and flexible architecture to explore, process, and visualize data in real time.