Working with kafka

School of Science and Engineering

Nouamane Zanboui

Meriem Lmoubariki

Adam Sterheltou

Dr. Tajjeeddine Rachidi

Al Akhawayn University in Ifrane

November 11/12/2024

Table of Contents

Introduction

In this report, we explore how Kafka, a powerful tool for handling real-time data, can be set up and used to manage streams of log entries. Kafka is widely used because it allows large amounts of data to be processed quickly and efficiently. This makes it invaluable for businesses that need to analyze data in real time, such as monitoring website traffic through log files.

The purpose of this assignment is to understand the fundamental roles of Kafka as a message broker, compare it with other similar technologies, and practically implement it by streaming data from log files. This hands-on experience will help us see how Kafka fits into the bigger picture of data management and processing.

Question 1 : Explain the Function of Kafka as a Message Broker

Apache Kafka is a distributed system that serves as a message broker, enabling the fast, reliable exchange of data across various applications. As a message broker, Kafka allows different applications to communicate by sending, storing, and retrieving data, all through a central hub. Rather than connecting directly, systems send their data to Kafka, which then organizes stores, and delivers it to other systems that need this information (Kafka Documentation, 2023).

Kafka uses a **publish-subscribe model** for managing data. In this model, **producers** (systems that generate data) send messages to specific topics in Kafka. Topics act like channels that organize data by category, making it easy for **consumers** (systems or applications that need data) to subscribe to the topics they need. Kafka's durable storage system saves messages to disk, ensuring consumers can access data even if they were offline when the messages were initially sent. This setup is useful for a wide range of applications, including tracking website activity, managing logs, and processing real-time financial transactions.

Kafka is highly efficient and scalable, capable of handling large data volumes with low delay. Its ability to decouple producers from consumers also makes it flexible for complex, data-driven systems, where applications need to operate independently without direct dependency on each other (Kreps et al., 2011; Confluent, 2023).

Question 2 : Comparison of Apache Kafka with Other Message Brokers and Pub/Sub Systems

(Pivotal Software, 2023; Red Hat, 2023)

Criteria	**Apache Kafka**	**RabbitMQ**	**ActiveMQ**	**AWS SNS (Simple Notification Service)**
Architecture	Distributed, log-based	Centralized, queue-based	Centralized, queue-based	Managed pub/sub service
Message Delivery	At least once, supports exactly-once	At most once, supports at-least-once	At most once, supports at-least-once	At least once
Persistence	Highly durable, persists messages on disk	Optional, mainly in-memory	Optional, persistent and in-memory	No built-in persistence
Scalability	Highly scalable, supports partitioning	Limited horizontal scaling	Limited horizontal scaling	Scalable, managed by AWS
Throughput	Very high throughput, optimized for logs	Moderate, better for transactional data	Moderate	High, but depends on AWS infrastructure
Latency	Low latency, ideal for real-time processing	Low latency, but may vary	Higher latency than Kafka	Low latency, managed by AWS
Ordering	Guarantees ordering per partition	Ordering optional, but not guaranteed	Supports ordering with configuration	No strict ordering
Protocol	Custom binary protocol, efficient (TCP)	AMQP, HTTP	OpenWire, STOMP, AMQP	HTTP-based
Use Cases	High-throughput logging, event streaming	Real-time updates, task queues	Enterprise messaging, legacy systems	Notifications, broadcast messaging
Durability	Very durable storage, keeps logs	Less durable, focuses on quick delivery	Moderate durability, disk or memory	Managed durability by AWS
Management	Requires setup, managed solutions exist	Requires setup, easier to manage	Requires setup, suited for smaller setups	Fully managed by AWS

Apache Kafka: Kafka is well-suited for applications that handle a lot of data at high speeds, like log processing, real-time analytics, and data pipelines. Its architecture is distributed, meaning it can be spread across multiple servers, making it very scalable. Kafka stores messages on disk, so even if the system goes down, data is preserved. It's best for large-scale applications where data needs to be processed quickly and reliably.

RabbitMQ: RabbitMQ is a popular choice for tasks that need reliable message delivery but don't require the massive scale that Kafka supports. It's easier to set up and integrates well with many messaging protocols like AMQP and HTTP. RabbitMQ is commonly used for smaller tasks like job queues or website updates, where it's important that messages are delivered reliably but not necessarily at the extremely high speeds Kafka can handle.

ActiveMQ: ActiveMQ is often used in businesses that need reliable messaging but don't require the high throughput that Kafka offers. It works well with older (or "legacy") systems and supports a range of messaging protocols. ActiveMQ can persist messages if needed, but it is typically used in smaller applications that need straightforward message passing rather than massive data streaming.

AWS SNS: Amazon Simple Notification Service (SNS) is a fully managed service provided by AWS, meaning Amazon handles setup and scaling. It's ideal for sending notifications and alerts, especially for applications already running in the AWS environment. Unlike Kafka, AWS SNS does not keep messages after they're sent, making it less suited for applications that need to retain message history. It's perfect for cases where instant notifications are needed, such as mobile alerts or system alerts.

Scalability and Throughput: Kafka is designed to handle extremely high data loads, making it ideal for big data applications where large amounts of data need to be processed in real-time. RabbitMQ and ActiveMQ don't scale to the same degree, but they are great for handling smaller tasks that need reliable message delivery. AWS SNS is highly scalable within the AWS ecosystem, making it convenient for applications hosted on AWS.

Persistence and Durability: Kafka is highly durable, storing messages on disk so they can be retrieved even after downtime or disruptions. This makes it useful for applications that need data stored reliably. In contrast, RabbitMQ and ActiveMQ offer optional message storage, while AWS SNS doesn't keep messages once they're sent.

Ideal Use Cases: Kafka is the go-to choice for high-volume, real-time applications like log processing and streaming analytics. RabbitMQ and ActiveMQ are preferred in scenarios needing reliable, smaller-scale messaging, and AWS SNS works well for quick notifications and mobile push alerts.

Question 3

In this part, we explain how we set up and got Apache Kafka running using Docker. Our main aim was to create a working system that could handle and manage streams of log data efficiently. We chose Docker because it makes the setup process easier and keeps everything neat by running Kafka and Zookeeper in separate, contained environments.

We chose the Confluent Kafka Docker image because it's developed by the creators of Kafka, ensuring it's highly reliable and integrates smoothly with necessary components like Zookeeper. Confluent provides a user-friendly setup with robust support and documentation, making it an ideal choice for both ease of use and access to advanced features for effective learning and deployment in our project (Confluent, 2023).

Step 1:

Setting Up the Environment

- **Docker Environment**: We began by ensuring Docker was installed on our system. (It was obvious that it was installed from the beginning of the semester)

Step 2:

Acquiring Necessary Docker Images

- **Downloading Images**: We executed the following commands to download the latest Docker images for Zookeeper and Kafka from Confluent. This step ensures that we have reliable and up-to-date versions of both services, which are essential for Kafka's operation.

Commands used:

```
docker pull confluentinc/cp-zookeeper
docker pull confluentinc/cp-kafka
```

Step 3:

Launching Zookeeper

- **Running Zookeeper Container**: We used the following command to start a Zookeeper container:

```
docker run --name zookeeper -p 2181:2181 -e ZOOKEEPER_CLIENT_PORT=2181 confluentinc/cp-zookeeper
```

This command sets up a Zookeeper instance necessary for Kafka's management, specifying the client port to enable Kafka's connection to it. The -p option maps the port from the container to our host.

Step 4:

Deploying Kafka Broker

- **Starting Kafka Container**: To deploy Kafka, we issued:

```
docker run --name kafka-broker -p 9092:9092 --link zookeeper:zookeeper -e KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181 -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://localhost:9092 -e KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1 confluentinc/cp-kafka
```

This command initializes a Kafka broker, linking it to the Zookeeper container. It configures Kafka to use Zookeeper for coordination and sets Kafka to advertise itself on localhost for connections. The environment variables specify configuration settings critical for Kafka's interaction with Zookeeper and clients.

Step 5:

Testing the Installation

- **Creating and using a Kafka Topic**:
    - We created a topic named "test-topic" using:

```
docker exec kafka-broker kafka-topics --create --topic test-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
```

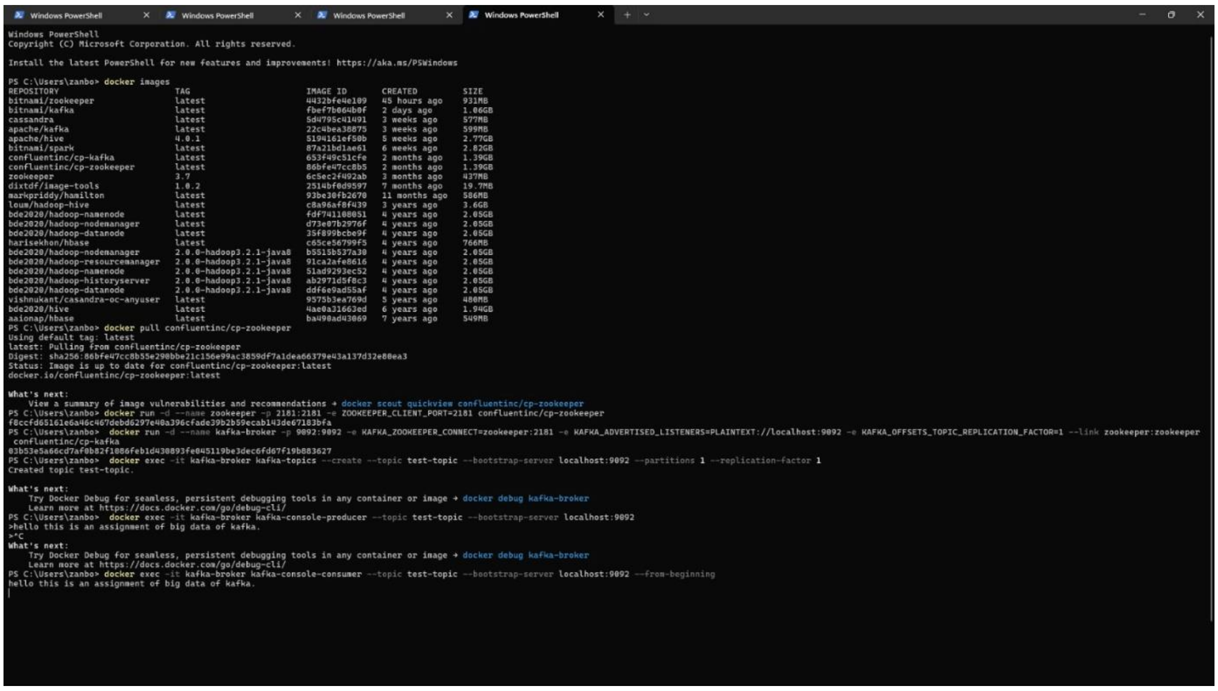
This command establishes a topic where messages will be stored. Setting partitions and replication factors defines how data is distributed and duplicated across the cluster.

To test the setup, we used the Kafka console producer and consumer to send and receive messages:

```
docker exec -it kafka-broker kafka-console-producer --topic test-topic --bootstrap-server localhost:9092
```

```
docker exec -it kafka-broker kafka-console-consumer --topic test-topic --from-beginning --bootstrap-server localhost:9092
```

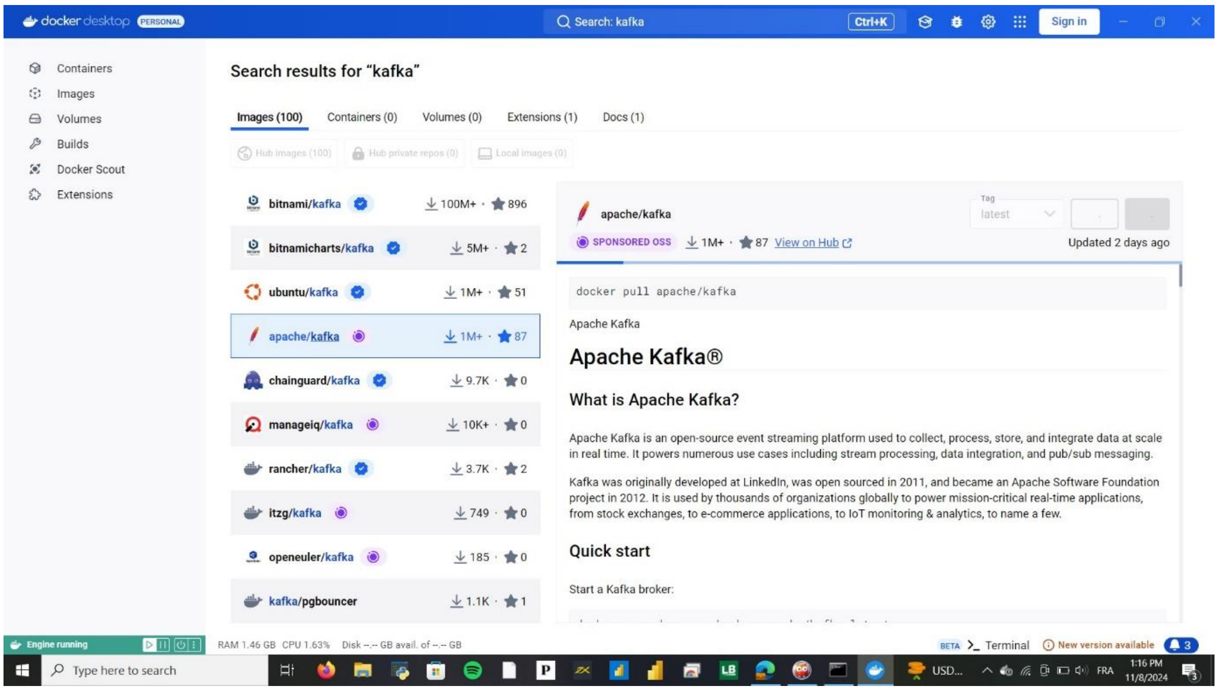
Results and Verification



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\zanbo> docker images
REPOSITORY                       TAG                          IMAGE ID       CREATED          SIZE
bitnami/zookeeper                latest                       4432bfe4e109   45 hours ago     931MB
bitnami/kafka                    latest                       fbef7b064b0f   2 days ago       1.06GB
cassandra                        latest                       5d4795c41491   3 weeks ago      577MB
apache/kafka                     latest                       22c4bea38875   3 weeks ago      599MB
apache/hive                      4.0.1                        5194161ef50b   5 weeks ago      2.77GB
bitnami/spark                    latest                       87a21bd1ae61   6 weeks ago      2.82GB
confluentinc/cp-kafka            latest                       653f49c51cfe   2 months ago     1.39GB
confluentinc/cp-zookeeper        latest                       86bfe47cc8b5   2 months ago     1.39GB
zookeeper                        3.7                          6c5ec2f492ab   3 months ago     437MB
dixtdf/image-tools               1.0.2                        2514bf0d9597   7 months ago     19.7MB
markpriddy/hamilton              latest                       93be30fb2670   11 months ago    586MB
loum/hadoop-hive                 latest                       c8a96af8f439   3 years ago      3.6GB
bde2020/hadoop-namenode          latest                       fdf741108051   4 years ago      2.05GB
bde2020/hadoop-nodemanager       latest                       d73e07b2976f   4 years ago      2.05GB
bde2020/hadoop-datanode          latest                       35f899bcbe9f   4 years ago      2.05GB
harisekhon/hbase                 latest                       c65ce56799f5   4 years ago      766MB
bde2020/hadoop-nodemanager       2.0.0-hadoop3.2.1-java8      b5515b537a30   4 years ago      2.05GB
bde2020/hadoop-resourcemanager   2.0.0-hadoop3.2.1-java8      91ca2afe8616   4 years ago      2.05GB
bde2020/hadoop-namenode          2.0.0-hadoop3.2.1-java8      51ad9293ec52   4 years ago      2.05GB
bde2020/hadoop-historyserver     2.0.0-hadoop3.2.1-java8      ab2971d5f8c3   4 years ago      2.05GB
bde2020/hadoop-datanode          2.0.0-hadoop3.2.1-java8      ddf6e9ad55af   4 years ago      2.05GB
vishnukant/casandra-oc-anyuser   latest                       9575b3ea769d   5 years ago      480MB
bde2020/hive                     latest                       4ae0a31663ed   6 years ago      1.94GB
aaionap/hbase                    latest                       ba490ad43069   7 years ago      549MB
PS C:\Users\zanbo> docker pull confluentinc/cp-zookeeper
Using default tag: latest
latest: Pulling from confluentinc/cp-zookeeper
Digest: sha256:86bfe47cc8b55e290bbe21c156e99ac3859df7a1dea66379e43a137d32e80ea3
Status: Image is up to date for confluentinc/cp-zookeeper:latest
docker.io/confluentinc/cp-zookeeper:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview confluentinc/cp-zookeeper
PS C:\Users\zanbo> docker run -d --name zookeeper -p 2181:2181 -e ZOOKEEPER_CLIENT_PORT=2181 confluentinc/cp-zookeeper
f8ccfd65161e6a46c467debd6297e40a396cfade39b2b59ecab143de67183bfa
PS C:\Users\zanbo>  docker run -d --name kafka-broker -p 9092:9092 -e KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181 -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://localhost:9092 -e KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1 --link zookeeper:zookeeper
 confluentinc/cp-kafka
03b53e5a66cd7af0b82f1086feb1d430893fe045119be3dec6fd67f19b883627
PS C:\Users\zanbo>  docker exec -it kafka-broker kafka-topics --create --topic test-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
Created topic test-topic.

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug kafka-broker
    Learn more at https://docs.docker.com/go/debug-cli/
PS C:\Users\zanbo>  docker exec -it kafka-broker kafka-console-producer --topic test-topic --bootstrap-server localhost:9092
>hello this is an assignment of big data of kafka.
>^C
What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug kafka-broker
    Learn more at https://docs.docker.com/go/debug-cli/
PS C:\Users\zanbo> docker exec -it kafka-broker kafka-console-consumer --topic test-topic --bootstrap-server localhost:9092 --from-beginning
hello this is an assignment of big data of kafka.
```

The attached previous screenshot (Figure 1) shows the successful execution of these commands and the message flow verification in Kafka. This visual confirmation is crucial for demonstrating the Kafka setup's functionality.

In the following figure, you will find another image with is bitnami KAFKA, it also worked.



Search results for "kafka"

bitnami/kafka	100M+	896
bitnamicharts/kafka	5M+	2
ubuntu/kafka	1M+	51
apache/kafka	1M+	87
chainguard/kafka	9.7K	0
manageiq/kafka	10K+	0
rancher/kafka	3.7K	2
itzg/kafka	749	0
openeuler/kafka	185	0
kafka/pgbouncer	1.1K	1

apache/kafka

1M+ · 87

Updated 2 days ago

```
docker pull apache/kafka
```

Apache Kafka

Apache Kafka®

What is Apache Kafka?

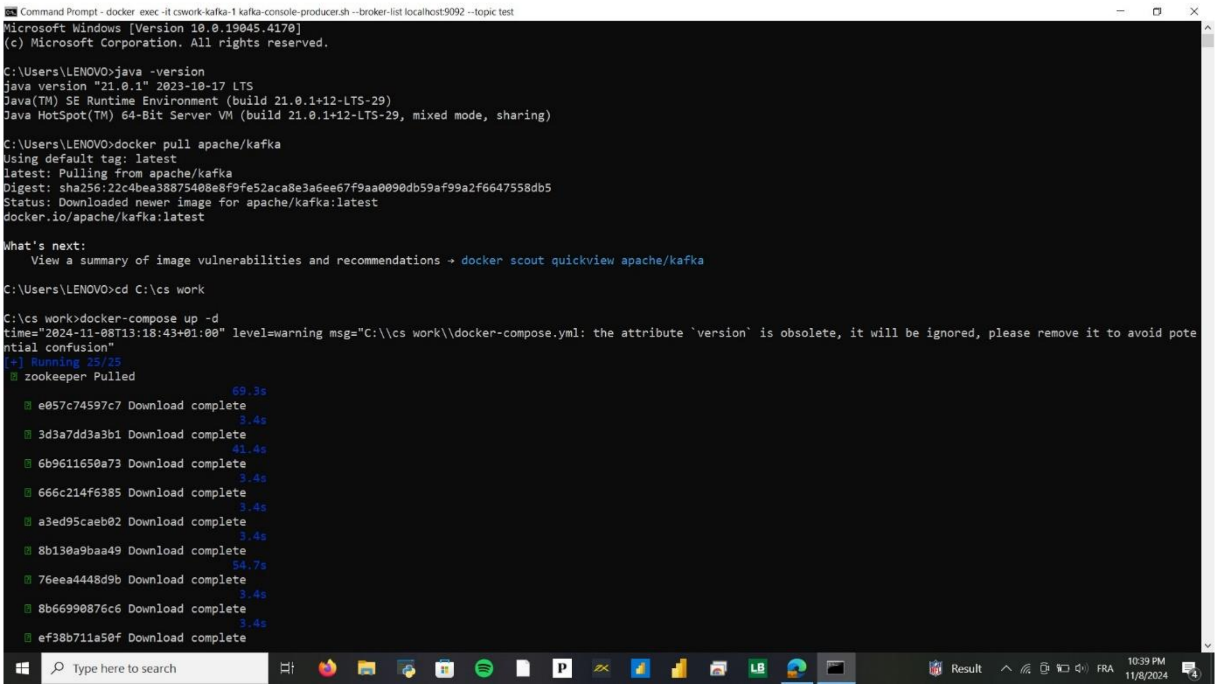
Apache Kafka is an open-source event streaming platform used to collect, process, store, and integrate data at scale in real time. It powers numerous use cases including stream processing, data integration, and pub/sub messaging.

Kafka was originally developed at LinkedIn, was open sourced in 2011, and became an Apache Software Foundation project in 2012. It is used by thousands of organizations globally to power mission-critical real-time applications, from stock exchanges, to e-commerce applications, to IoT monitoring & analytics, to name a few.

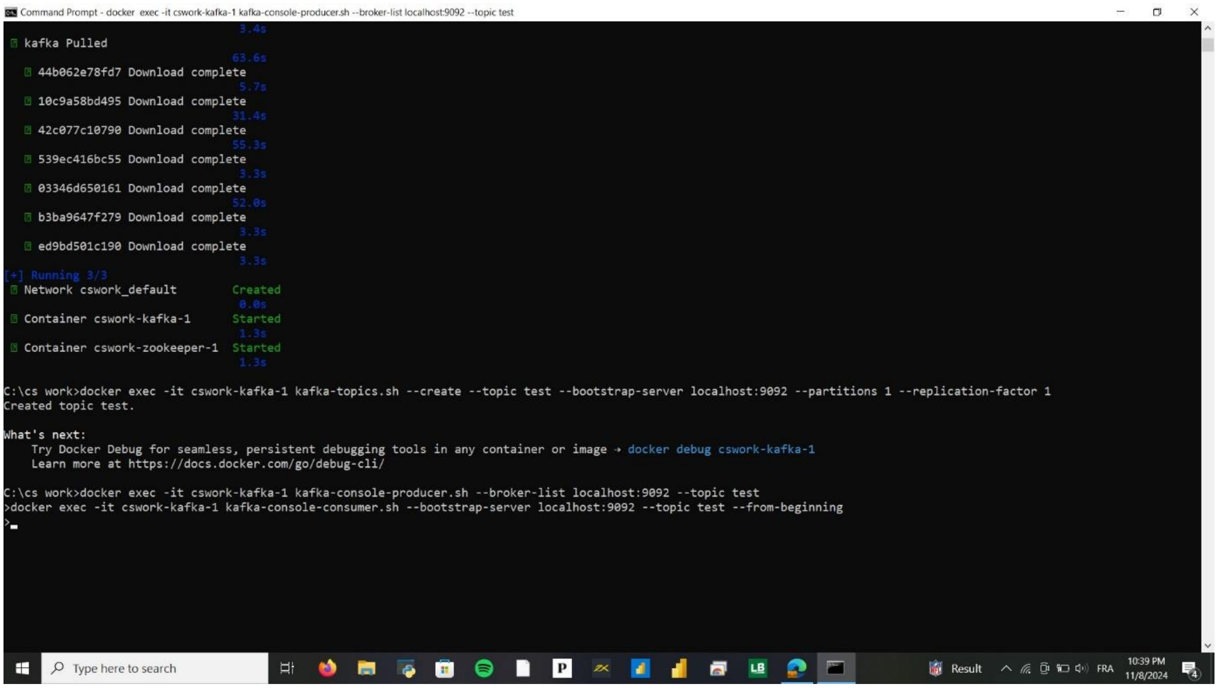
Quick start

Start a Kafka broker:

RAM 1.46 GB CPU 1.63% Disk --.-- GB avail. of --.-- GB

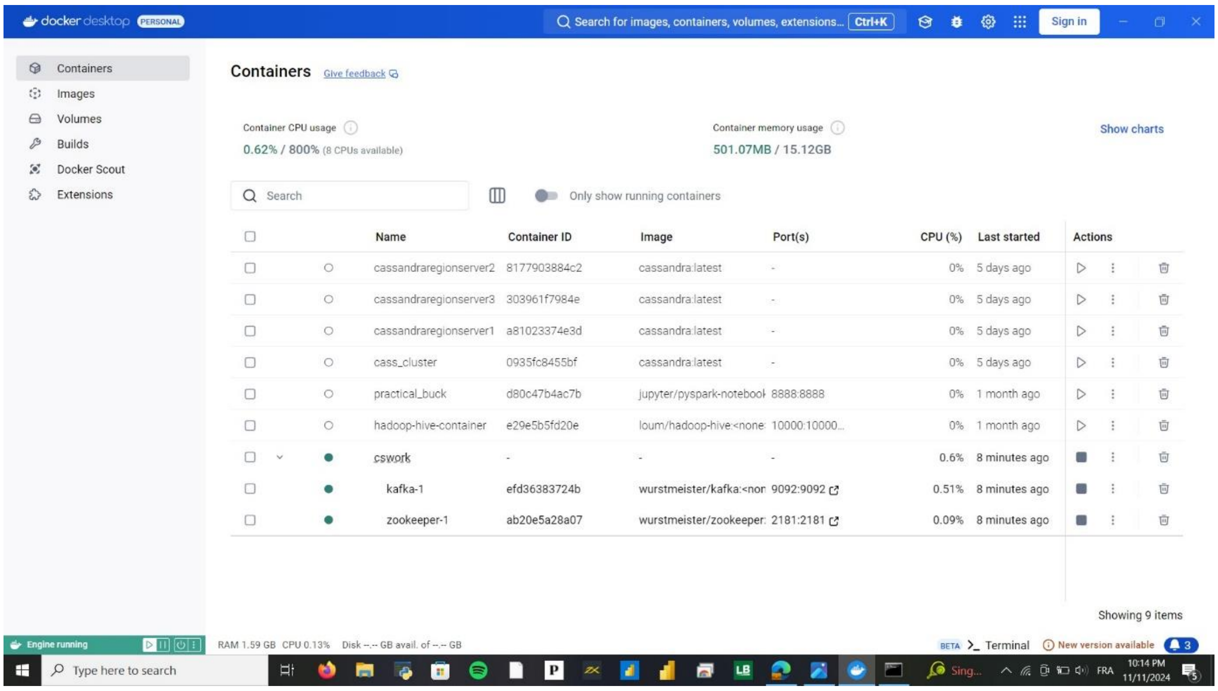

Command Prompt - docker exec -it cswork-kafka-1 kafka-console-producer.sh --broker-list localhost:9092 --topic test
Type here to search
Result
FRA
10:39 PM
11/8/2024

```
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.

C:\Users\LENOVO>java -version
java version "21.0.1" 2023-10-17 LTS
Java(TM) SE Runtime Environment (build 21.0.1+12-LTS-29)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.1+12-LTS-29, mixed mode, sharing)

C:\Users\LENOVO>docker pull apache/kafka
Using default tag: latest
latest: Pulling from apache/kafka
Digest: sha256:22c4bea38875408e8f9fe52aca8e3a6ee67f9aa0090db59af99a2f6647558db5
Status: Downloaded newer image for apache/kafka:latest
docker.io/apache/kafka:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview apache/kafka

C:\Users\LENOVO>cd C:\cs work

C:\cs work>docker-compose up -d
time="2024-11-08T13:18:43+01:00" level=warning msg="C:\\cs work\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid pote
ntial confusion"
[+] Running 25/25
 ▯ zookeeper Pulled
                                69.3s
   ▯ e057c74597c7 Download complete
                                 3.4s
   ▯ 3d3a7dd3a3b1 Download complete
                                41.4s
   ▯ 6b9611650a73 Download complete
                                 3.4s
   ▯ 666c214f6385 Download complete
                                 3.4s
   ▯ a3ed95caeb02 Download complete
                                 3.4s
   ▯ 8b130a9baa49 Download complete
                                54.7s
   ▯ 76eea4448d9b Download complete
                                 3.4s
   ▯ 8b66990876c6 Download complete
                                 3.4s
   ▯ ef38b711a50f Download complete
```


Command Prompt - docker exec -it cswork-kafka-1 kafka-console-producer.sh --broker-list localhost:9092 --topic test
Type here to search
Result
FRA
10:39 PM
11/8/2024

```
                                 3.4s
 ▯ kafka Pulled
                                63.6s
   ▯ 44b062e78fd7 Download complete
                                 5.7s
   ▯ 10c9a58bd495 Download complete
                                31.4s
   ▯ 42c077c10790 Download complete
                                55.3s
   ▯ 539ec416bc55 Download complete
                                 3.3s
   ▯ 03346d650161 Download complete
                                52.0s
   ▯ b3ba9647f279 Download complete
                                 3.3s
   ▯ ed9bd501c190 Download complete
                                 3.3s
[+] Running 3/3
 ▯ Network cswork_default       Created
                                 0.0s
 ▯ Container cswork-kafka-1     Started
                                 1.3s
 ▯ Container cswork-zookeeper-1  Started
                                 1.3s

C:\cs work>docker exec -it cswork-kafka-1 kafka-topics.sh --create --topic test --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
Created topic test.

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug cswork-kafka-1
    Learn more at https://docs.docker.com/go/debug-cli/

C:\cs work>docker exec -it cswork-kafka-1 kafka-console-producer.sh --broker-list localhost:9092 --topic test
>docker exec -it cswork-kafka-1 kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
>_
```

We did identically the same process but only with a different image.



Containers Give feedback

Container CPU usage
0.62% / 800% (8 CPUs available)

Container memory usage
501.07MB / 15.12GB

Show charts

Search

Only show running containers

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
○	cassandraregionserver2	8177903884c2	cassandra:latest	-	0%	5 days ago	
○	cassandraregionserver3	303961f7984e	cassandra:latest	-	0%	5 days ago	
○	cassandraregionserver1	a81023374e3d	cassandra:latest	-	0%	5 days ago	
○	cass_cluster	0935fc8455bf	cassandra:latest	-	0%	5 days ago	
○	practical_buck	d80c47b4ac7b	jupyter/pyspark-notebool	8888:8888	0%	1 month ago	
○	hadoop-hive-container	e29e5b5fd20e	loum/hadoop-hive:<none:	10000:10000...	0%	1 month ago	
●	cswork	-	-	-	0.6%	8 minutes ago	
●	kafka-1	efd36383724b	wurstmeister/kafka:<nor	9092:9092	0.51%	8 minutes ago	
●	zookeeper-1	ab20e5a28a07	wurstmeister/zookeeper:	2181:2181	0.09%	8 minutes ago	

Showing 9 items

Engine running RAM 1.59 GB CPU 0.13% Disk --.-- GB avail. of --.-- GB

BETA Terminal New version available 3

Type here to search

Sing... FRA 10:14 PM 11/11/2024

Question 4

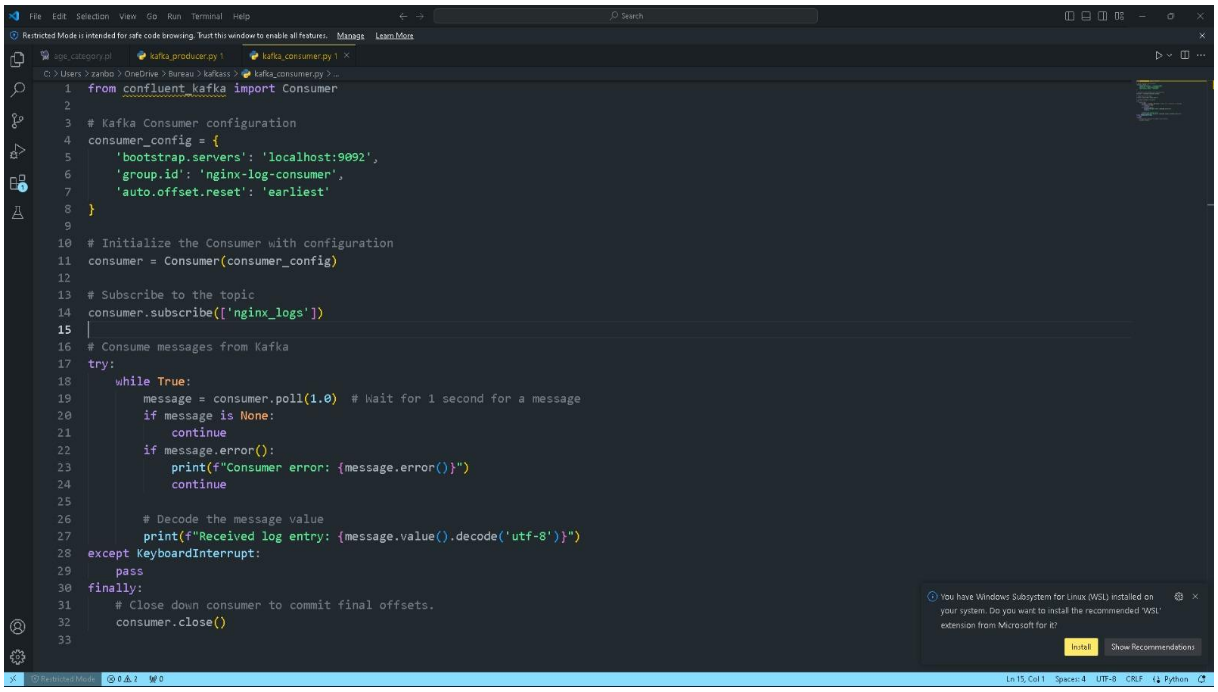
In this part of the assignment, we set up a Python application to send (produce) and receive (consume) log file entries using Kafka. The log files we used are from an Nginx web server. We first created a Python script to read log entries from a file (a text file)and send them to Kafka. Then, we wrote another script to read these entries from Kafka, mimicking how real-world applications might process log data in real-time.

Development Environment and Tools:

Operating System: The scripts are run on a Windows 10 machine

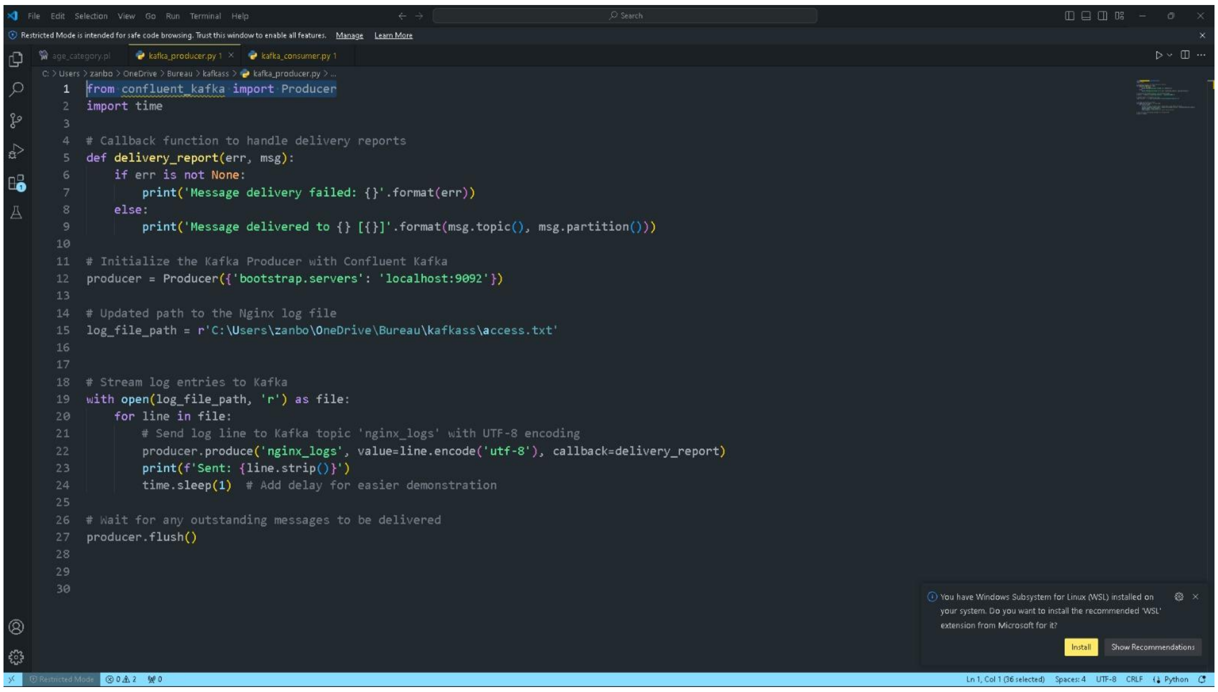
Python Environment: Python 3 is used

Producer and consumer set ups:


File Edit Selection View Go Run Terminal Help
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
age_category.pl
kafka_producer.py 1
kafka_consumer.py 1
C: > Users > zanbo > OneDrive > Bureau > kafkass > kafka_consumer.py > ...
Restricted Mode
Ln 15, Col 1 Spaces: 4 UTF-8 CRLF Python

```python
from confluent_kafka import Consumer

# Kafka Consumer configuration
consumer_config = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'nginx-log-consumer',
    'auto.offset.reset': 'earliest'
}

# Initialize the Consumer with configuration
consumer = Consumer(consumer_config)

# Subscribe to the topic
consumer.subscribe(['nginx_logs'])

# Consume messages from Kafka
try:
    while True:
        message = consumer.poll(1.0)  # Wait for 1 second for a message
        if message is None:
            continue
        if message.error():
            print(f"Consumer error: {message.error()}")
            continue

        # Decode the message value
        print(f"Received log entry: {message.value().decode('utf-8')}")
except KeyboardInterrupt:
    pass
finally:
    # Close down consumer to commit final offsets.
    consumer.close()
```

You have Windows Subsystem for Linux (WSL) installed on your system. Do you want to install the recommended 'WSL' extension from Microsoft for it?


File Edit Selection View Go Run Terminal Help
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
age_category.pl
kafka_producer.py 1
kafka_consumer.py 1
C: > Users > zanbo > OneDrive > Bureau > kafkass > kafka_producer.py > ...

```python
from confluent_kafka import Producer
import time

# Callback function to handle delivery reports
def delivery_report(err, msg):
    if err is not None:
        print('Message delivery failed: {}'.format(err))
    else:
        print('Message delivered to {} [{}]'.format(msg.topic(), msg.partition()))

# Initialize the Kafka Producer with Confluent Kafka
producer = Producer({'bootstrap.servers': 'localhost:9092'})

# Updated path to the Nginx log file
log_file_path = r'C:\Users\zanbo\OneDrive\Bureau\kafkass\access.txt'


# Stream log entries to Kafka
with open(log_file_path, 'r') as file:
    for line in file:
        # Send log line to Kafka topic 'nginx_logs' with UTF-8 encoding
        producer.produce('nginx_logs', value=line.encode('utf-8'), callback=delivery_report)
        print(f'Sent: {line.strip()}')
        time.sleep(1)  # Add delay for easier demonstration

# Wait for any outstanding messages to be delivered
producer.flush()
```

You have Windows Subsystem for Linux (WSL) installed on your system. Do you want to install the recommended 'WSL' extension from Microsoft for it?

Python Libraries Used

- **Confluent Kafka Python Library**: This is the main library used for creating Kafka producers and consumers
  - **Installation**: Before running the scripts, we installed the Confluent Kafka Python library using pip, Python's package installer, with the command:

pip install confluent-kafka

Kafka and Zookeeper Setup

- **Apache Kafka**: A distributed streaming platform that handles publishing and subscribing to streams of records. It is used here to manage the flow of data from the Nginx logs.
- **Zookeeper**: Used by Kafka for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these facilities help in managing the Kafka cluster.
  - **Docker Containers**: Both Kafka and Zookeeper are run as Docker containers, which isolates their environment and simplifies deployment

Python Producer Script

- **Purpose**: The producer script reads from a Nginx access log file and sends each line as a message to a Kafka topic called nginx_logs.
- **Key Functions**:
  - **File Handling**: Python's built-in open() function is used to read from the Nginx log file.

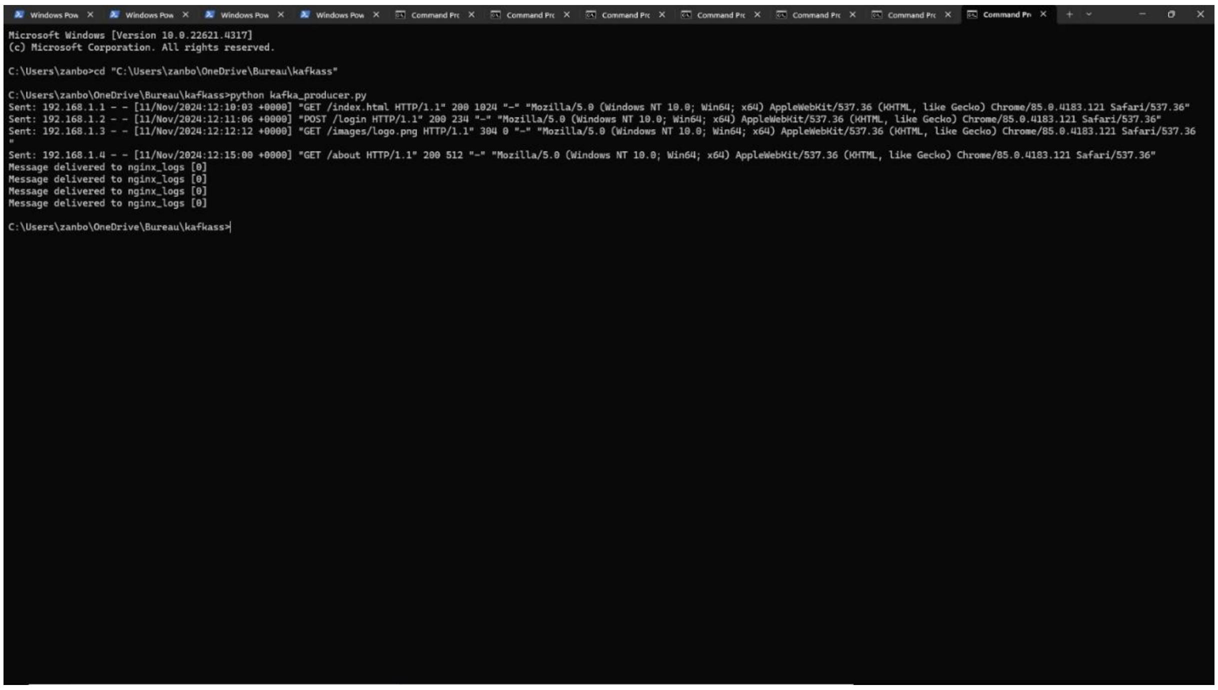
  - **Message Sending**: The produce() method of the Kafka Producer instance is used to send messages to the Kafka server. Messages are encoded in UTF-8 to handle any text data properly.
  - **Callback for Delivery Reports**: This provides asynchronous confirmation of message delivery, helping to debug and ensure reliability in message delivery.

Python Consumer Script

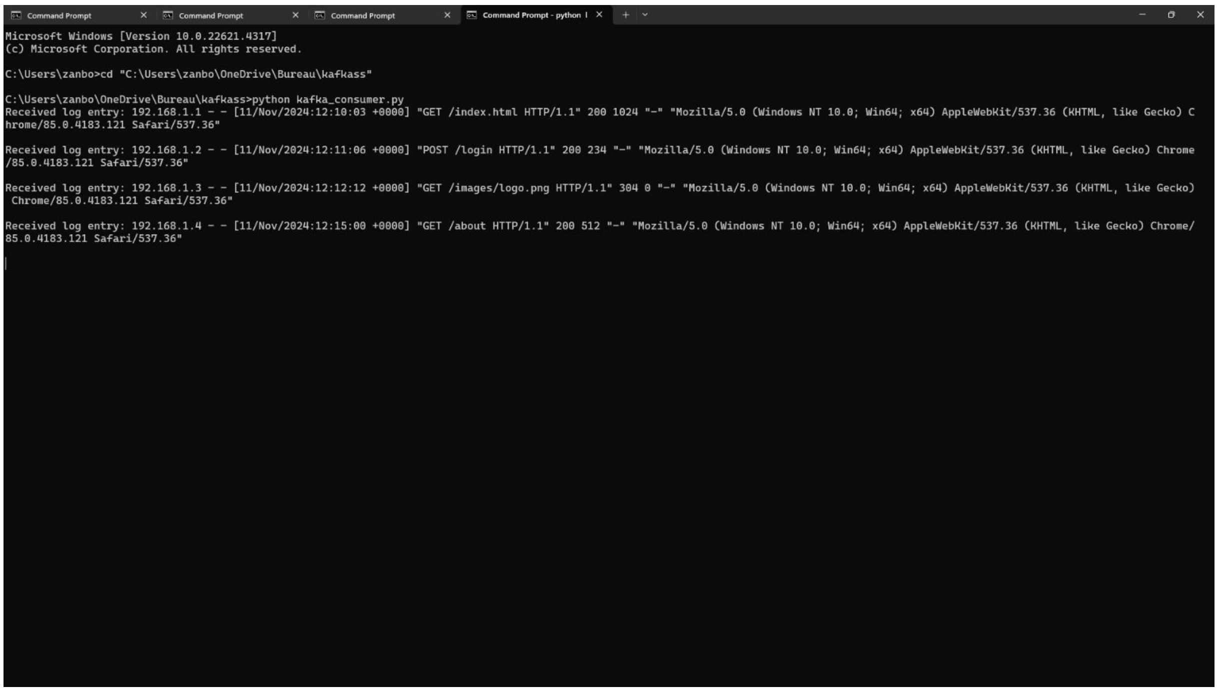
- **Purpose**: The consumer script listens for messages on the nginx_logs topic and processes each message as it comes in.
- **Key Functions**:
  - **Message Polling**: The poll() method is used to wait for new messages from the Kafka server, demonstrating a typical event loop used in real-time data processing applications.
  - **Error Handling**: The script checks for errors in each message, which helps in robustly handling real-world scenarios where network issues or Kafka errors might occur.

We run the producer first then the consumer:

Result:



```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zanbo>cd "C:\Users\zanbo\OneDrive\Bureau\kafkass"

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_producer.py
Sent: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Sent: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Sent: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
"
Sent: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]

C:\Users\zanbo\OneDrive\Bureau\kafkass>
```



```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zanbo>cd "C:\Users\zanbo\OneDrive\Bureau\kafkass"

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_consumer.py
Received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
85.0.4183.121 Safari/537.36"
```

Each line in the Command Prompt window begins with "Received log entry:", indicating that the consumer script is successfully retrieving messages. These messages are log entries originally written to an Nginx access log file and then published to Kafka by the producer script. The details shown in each log entry include the client's IP address, the timestamp of the request, the HTTP method and resource requested, the HTTP status code and response size, and the client's browser information. For instance, entries like "GET /index.html HTTP/1.1" followed by "200 1024" denote a successful fetch of the 'index.html' page with a 200 OK status and a response size of 1024 bytes.

This output not only validates the correct functioning of the consumer script but also showcases real-time data processing capabilities. It demonstrates the integration between the producer and consumer components, affirming that they can communicate effectively through Kafka. Moreover, this visualization of log data is crucial for monitoring the data flow, assisting in debugging and operational oversight, ensuring all parts of the Kafka setup operate cohesively.

Question 5

In this bonus question, we explored advanced capabilities of Kafka consumer groups by setting them up in two distinct configurations: fanout and load balancing. These configurations allow Kafka to distribute and manage data streams more effectively across multiple consumers, enhancing both the efficiency and reliability of data processing.

Consumer Groups Concept

- **Consumer Groups**: A consumer group in Kafka consists of multiple consumers that jointly process the data contained in a Kafka topic. Each consumer within a group reads from exclusive partitions of the topic, ensuring that each message is processed just once by the group as a whole.
- **Fanout**: This setup involves multiple consumer groups consuming the same data independently of each other. It's useful for scenarios where different systems or applications need to process the same data in parallel without interfering with each other.
- **Load Balancing**: In this configuration, all consumers belong to the same group but share the work of consuming messages. Kafka automatically distributes the messages among available consumers in the group, balancing the load and increasing throughput.

Implementation of the work

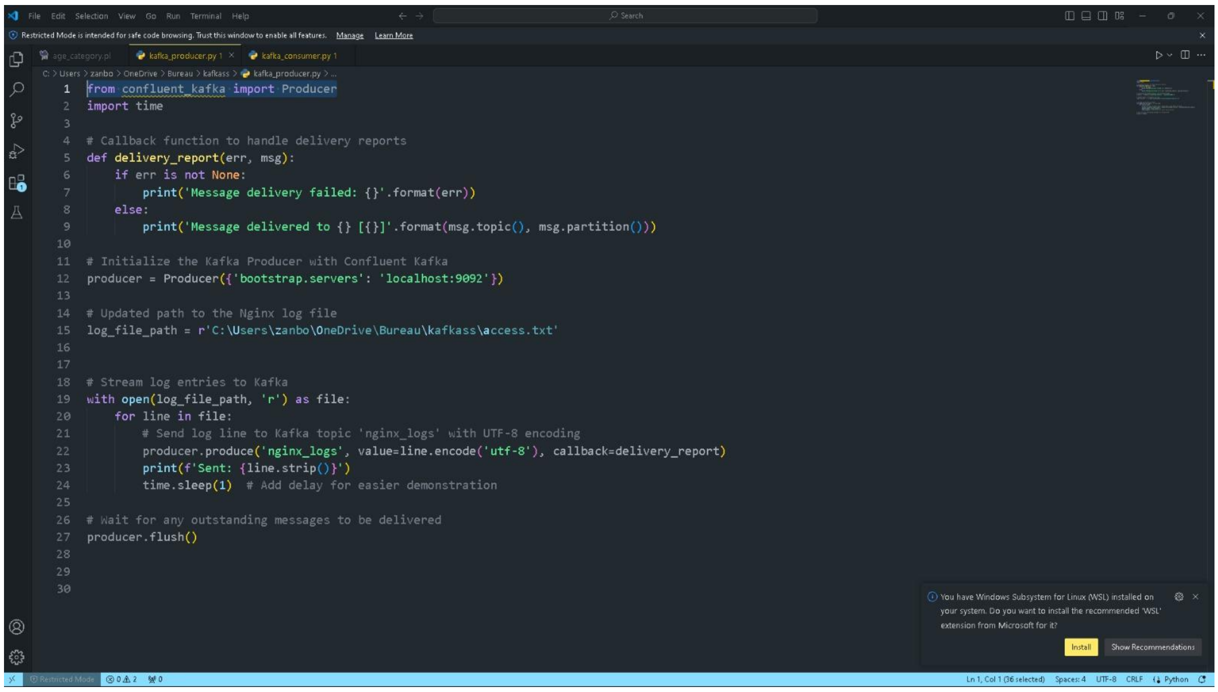
- **Kafka Producer Script**: As with earlier tasks, the producer script sends log entries from an Nginx access log file to a Kafka topic named 'nginx_logs'. The producer is configured to connect to Kafka running on localhost:9092 and sends messages with UTF-8 encoding to ensure accurate text representation.
- **Fanout Configuration**: We set up multiple consumer groups, each configured to consume messages from the 'nginx_logs' topic independently. This demonstrates the fanout configuration, where each group processes the same set of data for different purposes or in different ways.
- **Load Balancing Configuration**: We configured multiple consumers under a single group ID to demonstrate load balancing. Kafka distributes the incoming messages among all consumers in this group, effectively balancing the workload and enhancing processing speed.

Kafka Producer Script

- **Content**: Shows the Kafka producer script in an IDE. This script is responsible for reading Nginx log entries from a file and sending them to a Kafka topic (nginx_logs).
- **Highlights**:
  - Initialization of the Kafka producer with the server address localhost:9092.
  - Sending log entries with a callback to confirm delivery.
  - This script is crucial for supplying data to Kafka, which is then consumed by different consumer groups.

Here is it’s implementation : fig 1

Producer:

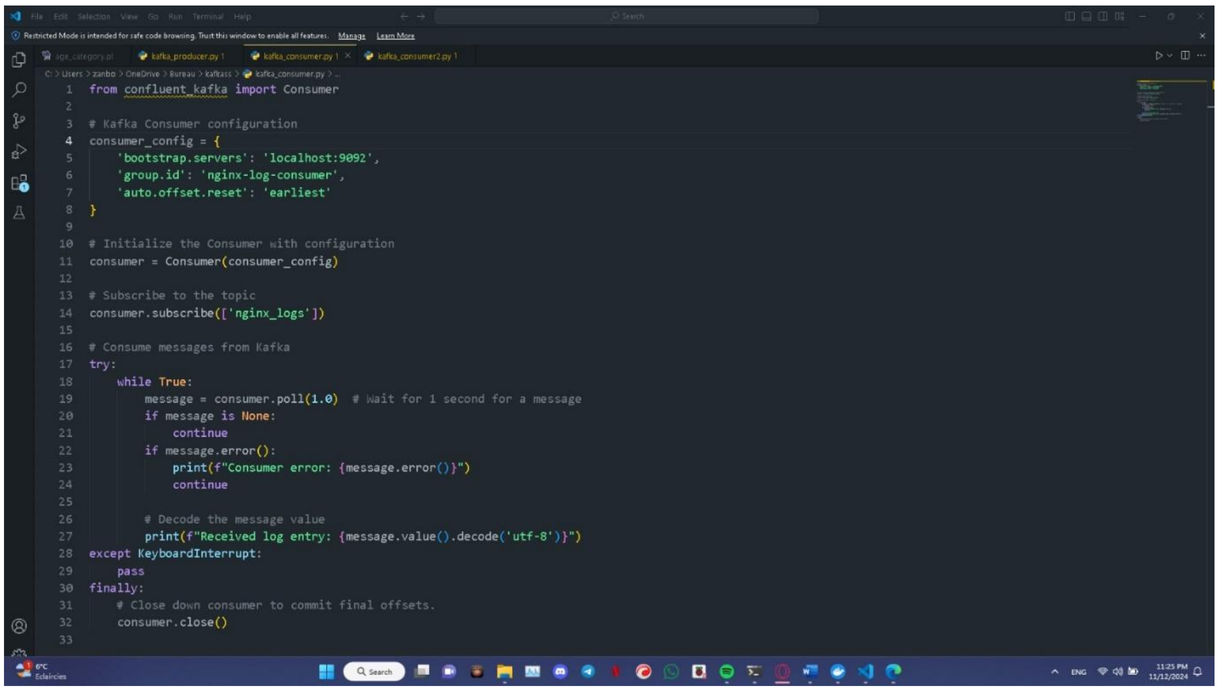


Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

```python
from confluent_kafka import Producer
import time

# Callback function to handle delivery reports
def delivery_report(err, msg):
    if err is not None:
        print('Message delivery failed: {}'.format(err))
    else:
        print('Message delivered to {} [{}]'.format(msg.topic(), msg.partition()))

# Initialize the Kafka Producer with Confluent Kafka
producer = Producer({'bootstrap.servers': 'localhost:9092'})

# Updated path to the Nginx log file
log_file_path = r'C:\Users\zanbo\OneDrive\Bureau\kafkass\access.txt'


# Stream log entries to Kafka
with open(log_file_path, 'r') as file:
    for line in file:
        # Send log line to Kafka topic 'nginx_logs' with UTF-8 encoding
        producer.produce('nginx_logs', value=line.encode('utf-8'), callback=delivery_report)
        print(f'Sent: {line.strip()}')
        time.sleep(1)  # Add delay for easier demonstration

# Wait for any outstanding messages to be delivered
producer.flush()
```

You have Windows Subsystem for Linux (WSL) installed on your system. Do you want to install the recommended 'WSL' extension from Microsoft for it?

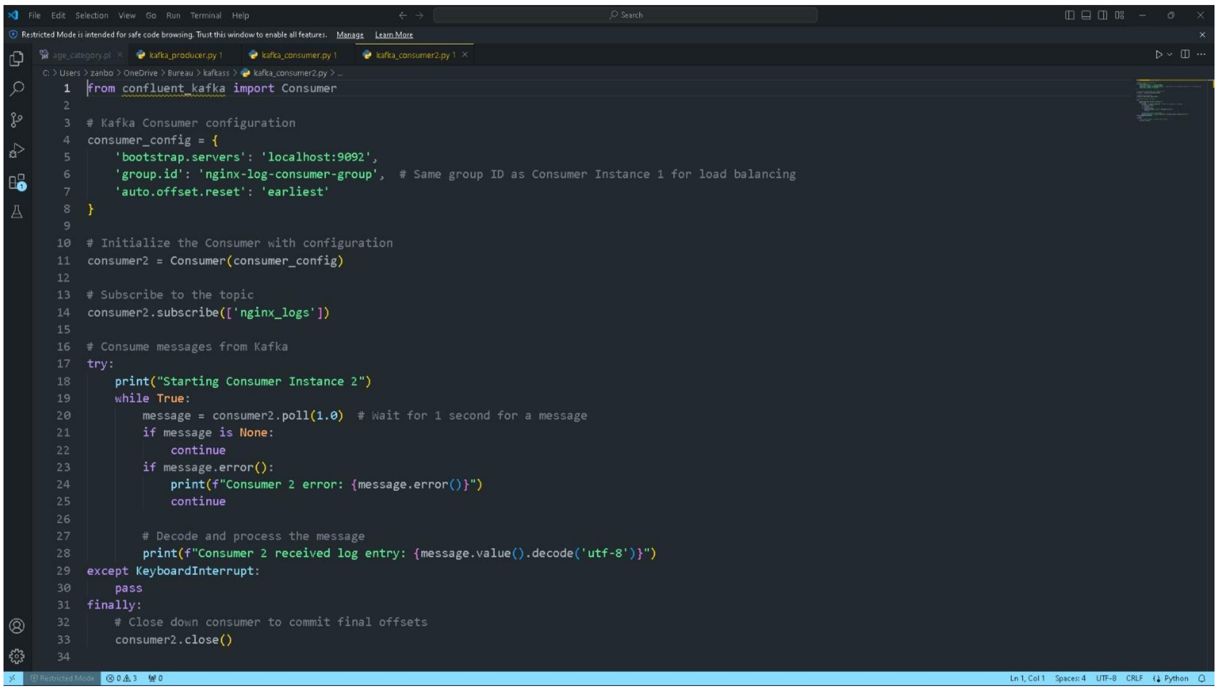
consumer1:



Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

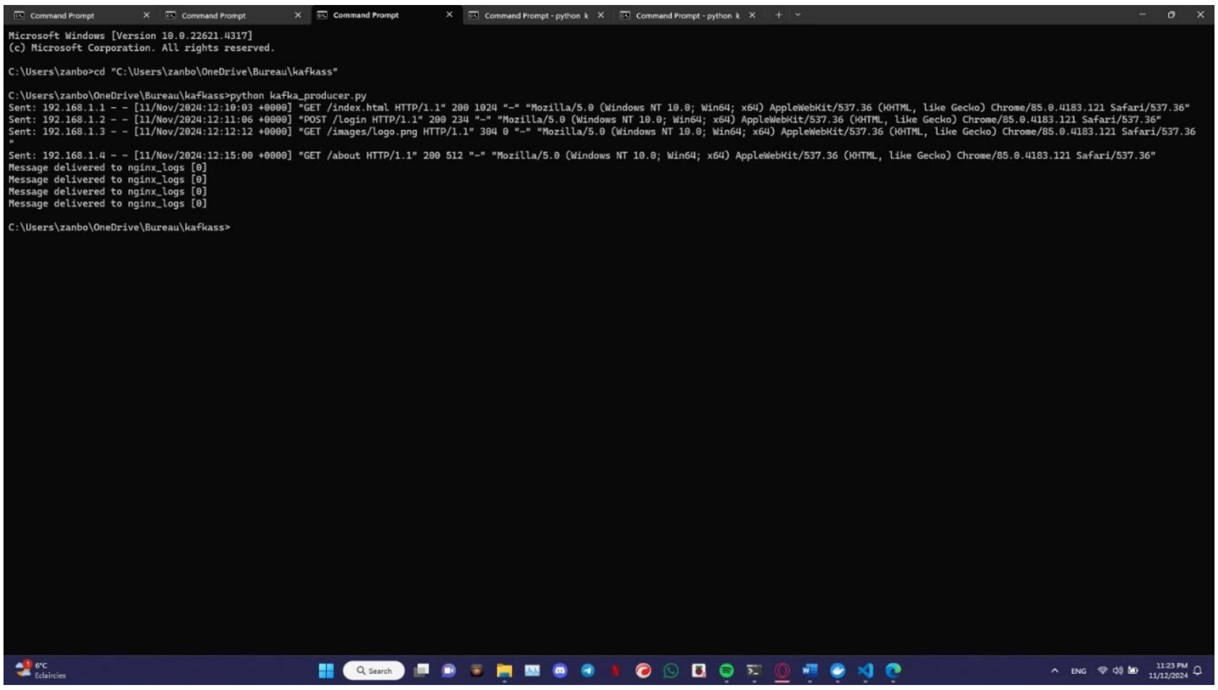
```python
from confluent_kafka import Consumer

# Kafka Consumer configuration
consumer_config = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'nginx-log-consumer',
    'auto.offset.reset': 'earliest'
}

# Initialize the Consumer with configuration
consumer = Consumer(consumer_config)

# Subscribe to the topic
consumer.subscribe(['nginx_logs'])

# Consume messages from Kafka
try:
    while True:
        message = consumer.poll(1.0)  # Wait for 1 second for a message
        if message is None:
            continue
        if message.error():
            print(f"Consumer error: {message.error()}")
            continue

        # Decode the message value
        print(f"Received log entry: {message.value().decode('utf-8')}")
except KeyboardInterrupt:
    pass
finally:
    # Close down consumer to commit final offsets.
    consumer.close()
```

consumer2:

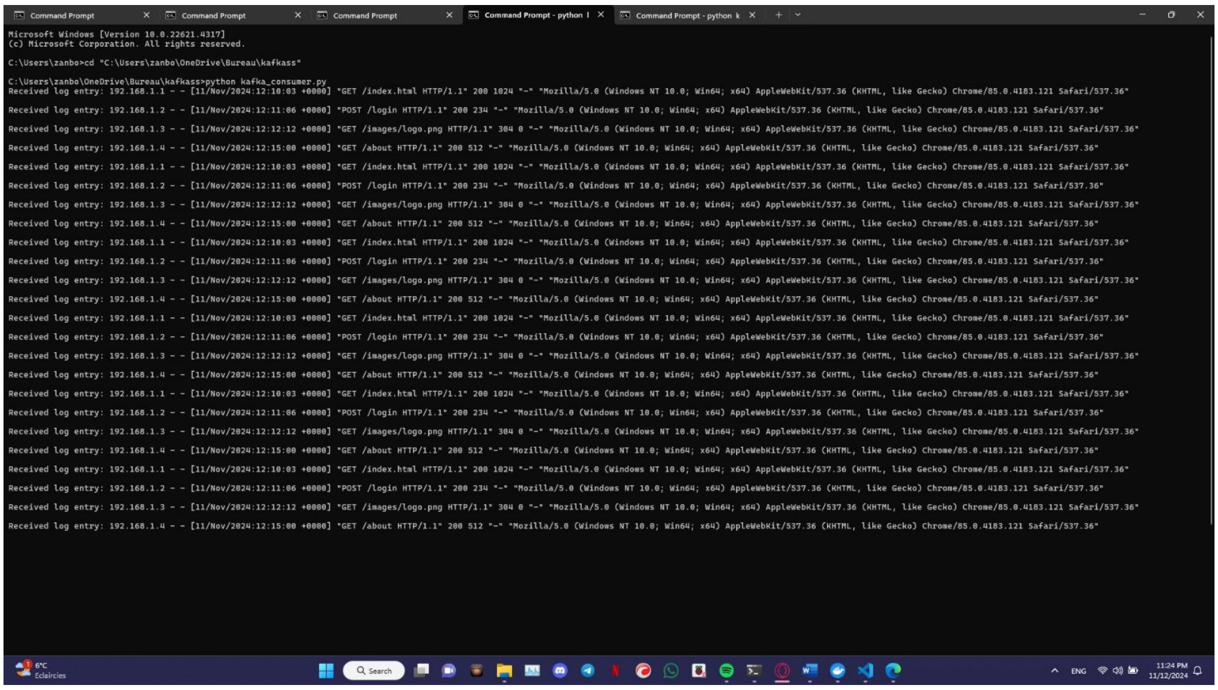

```python
from confluent_kafka import Consumer

# Kafka Consumer configuration
consumer_config = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'nginx-log-consumer-group',  # Same group ID as Consumer Instance 1 for load balancing
    'auto.offset.reset': 'earliest'
}

# Initialize the Consumer with configuration
consumer2 = Consumer(consumer_config)

# Subscribe to the topic
consumer2.subscribe(['nginx_logs'])

# Consume messages from Kafka
try:
    print("Starting Consumer Instance 2")
    while True:
        message = consumer2.poll(1.0)  # Wait for 1 second for a message
        if message is None:
            continue
        if message.error():
            print(f"Consumer 2 error: {message.error()}")
            continue

        # Decode and process the message
        print(f"Consumer 2 received log entry: {message.value().decode('utf-8')}")
except KeyboardInterrupt:
    pass
finally:
    # Close down consumer to commit final offsets
    consumer2.close()

```

Then the results :

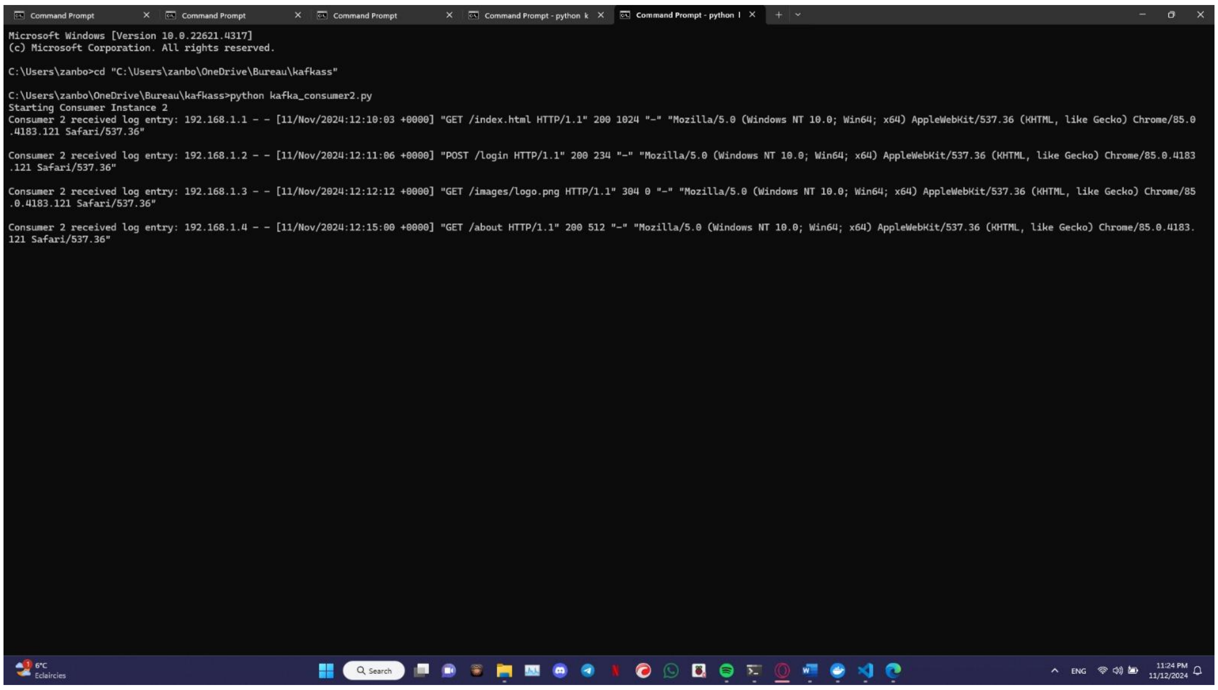
Running the producer first:


```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zanbo>cd "C:\Users\zanbo\OneDrive\Bureau\kafkass"

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_producer.py
Sent: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Sent: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Sent: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
"
Sent: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]

C:\Users\zanbo\OneDrive\Bureau\kafkass>
```

Then running the consumers:



```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zanbo>cd "C:\Users\zanbo\OneDrive\Bureau\kafkass"

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_consumer.py
Received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"

Received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
```



```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zanbo>cd "C:\Users\zanbo\OneDrive\Bureau\kafkass"

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_consumer2.py
Starting Consumer Instance 2
Consumer 2 received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0
.4183.121 Safari/537.36"

Consumer 2 received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183
.121 Safari/537.36"

Consumer 2 received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85
.0.4183.121 Safari/537.36"

Consumer 2 received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.
121 Safari/537.36"
```

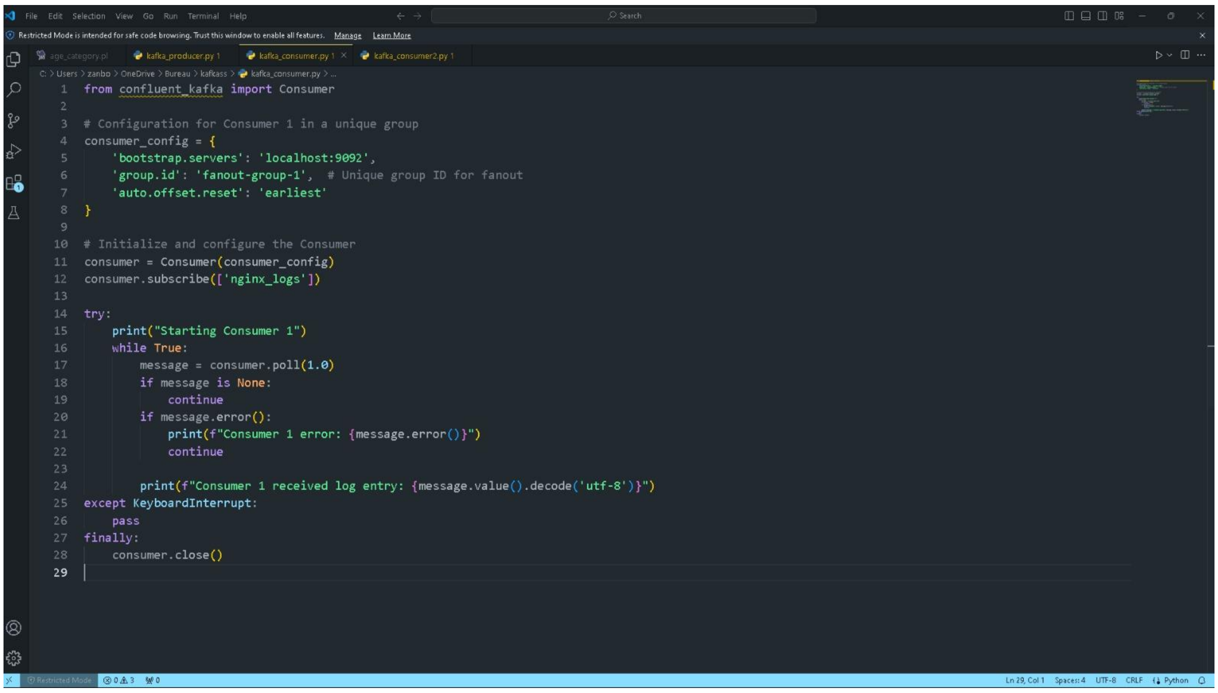
So we see that the consumers received different log entries from the same topic. By setting up multiple consumers within the same group, Kafka's load balancing capability was effectively utilized. This configuration enables scalable data processing, making it well-suited for high-throughput applications that demand efficient message management across distributed systems. Kafka's consumer group load balancing provides a strong solution for applications requiring scalable and reliable data streaming.

FIG 2 : Kafka Consumer Script for Fanout Configuration

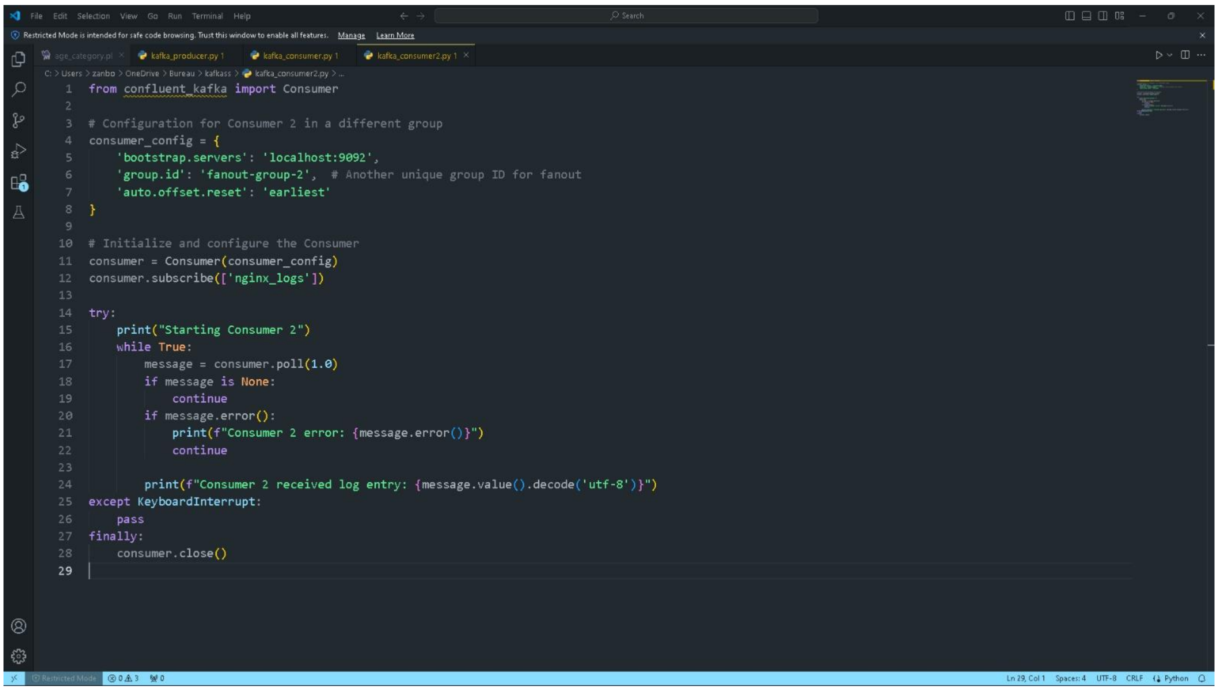
- Displays a Kafka consumer script configured for a fanout scenario. This script is set to consume messages from the nginx_logs topic independently from other consumers.

- Consumer is set up with a unique group ID (fanout-group-2), allowing it to consume messages in parallel with others.
- Script includes real-time message consumption logs, showing each entry as it is processed.

Here is the code for the consumers for the fanout configuration:



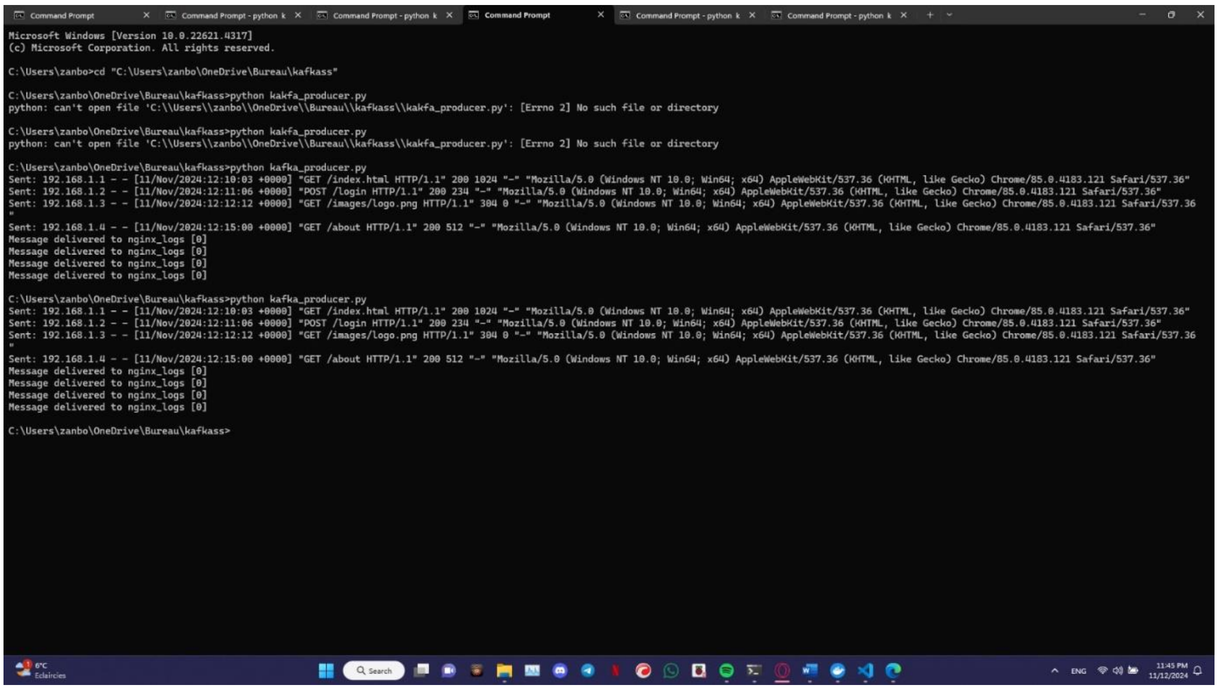
```python
from confluent_kafka import Consumer

# Configuration for Consumer 1 in a unique group
consumer_config = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'fanout-group-1',  # Unique group ID for fanout
    'auto.offset.reset': 'earliest'
}

# Initialize and configure the Consumer
consumer = Consumer(consumer_config)
consumer.subscribe(['nginx_logs'])

try:
    print("Starting Consumer 1")
    while True:
        message = consumer.poll(1.0)
        if message is None:
            continue
        if message.error():
            print(f"Consumer 1 error: {message.error()}")
            continue

        print(f"Consumer 1 received log entry: {message.value().decode('utf-8')}")
except KeyboardInterrupt:
    pass
finally:
    consumer.close()
```



```python
from confluent_kafka import Consumer

# Configuration for Consumer 2 in a different group
consumer_config = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'fanout-group-2',  # Another unique group ID for fanout
    'auto.offset.reset': 'earliest'
}

# Initialize and configure the Consumer
consumer = Consumer(consumer_config)
consumer.subscribe(['nginx_logs'])

try:
    print("Starting Consumer 2")
    while True:
        message = consumer.poll(1.0)
        if message is None:
            continue
        if message.error():
            print(f"Consumer 2 error: {message.error()}")
            continue

        print(f"Consumer 2 received log entry: {message.value().decode('utf-8')}")
except KeyboardInterrupt:
    pass
finally:
    consumer.close()
```

Fig 3 : Messages are successfully sent to the Kafka topic, as indicated by "Message delivered to nginx_logs [0]".

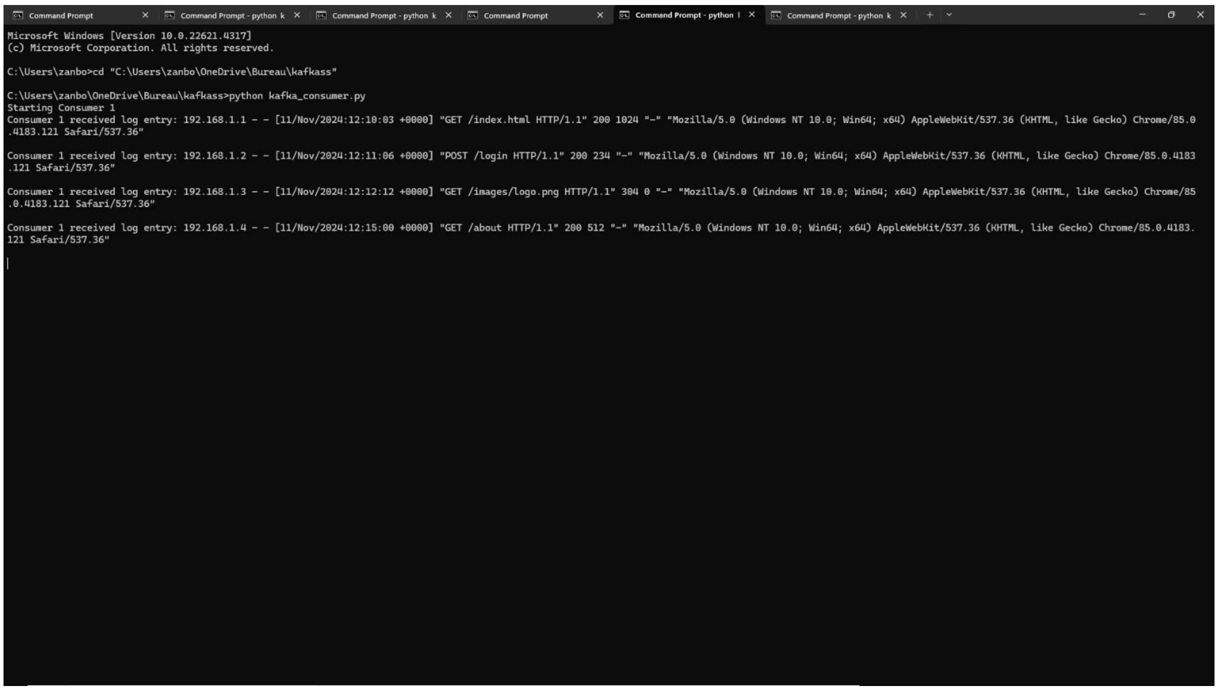
- **Shows the real-time logging of HTTP requests being processed and confirmed as sent.**
- **Validates that the producer script is actively interacting with Kafka and the messages are queued correctly in the topic.**



```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zanbo>cd "C:\Users\zanbo\OneDrive\Bureau\kafkass"

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kakfa_producer.py
python: can't open file 'C:\\Users\\zanbo\\OneDrive\\Bureau\\kafkass\\kakfa_producer.py': [Errno 2] No such file or directory

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kakfa_producer.py
python: can't open file 'C:\\Users\\zanbo\\OneDrive\\Bureau\\kafkass\\kakfa_producer.py': [Errno 2] No such file or directory

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_producer.py
Sent: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Sent: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Sent: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
"
Sent: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_producer.py
Sent: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Sent: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Sent: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
"
Sent: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36"
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]
Message delivered to nginx_logs [0]

C:\Users\zanbo\OneDrive\Bureau\kafkass>
```

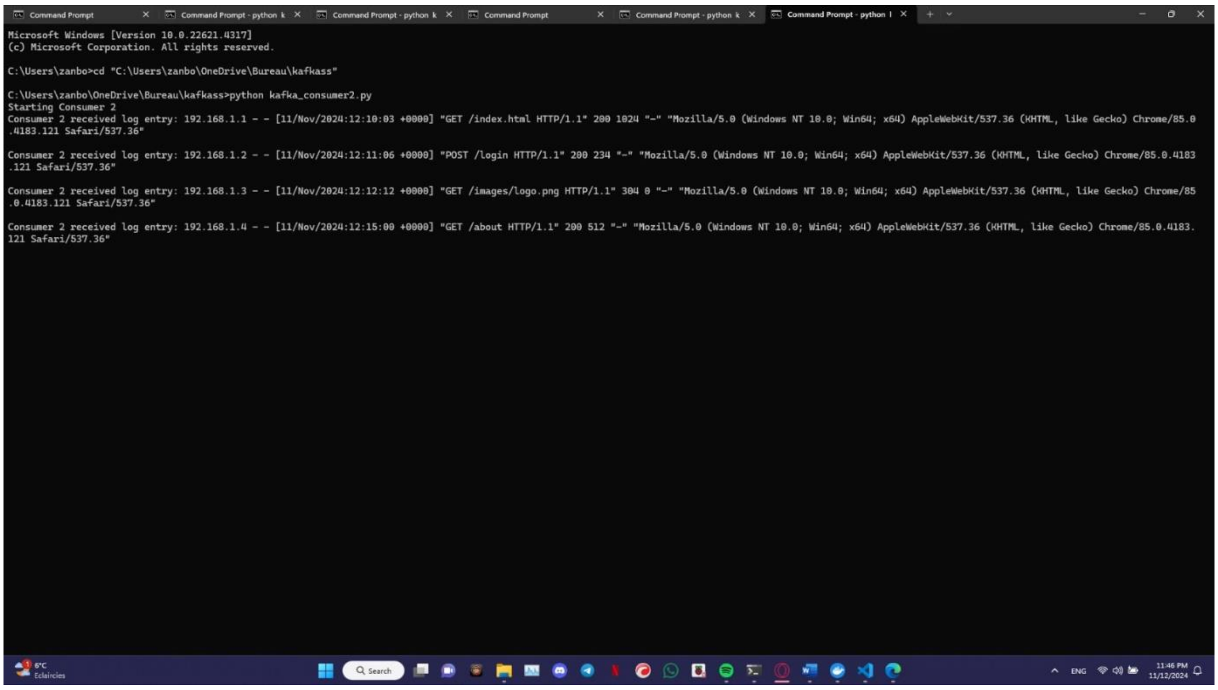
Command Prompt
Command Prompt - python k
Command Prompt - python k
Command Prompt
Command Prompt - python k
Command Prompt - python k

Fig 4 : Displays multiple log entries being received and processed by the consumer.


Command Prompt
Command Prompt - python k
Command Prompt - python k
Command Prompt
Command Prompt - python l
Command Prompt - python k

```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zanbo>cd "C:\Users\zanbo\OneDrive\Bureau\kafkass"

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_consumer.py
Starting Consumer 1
Consumer 1 received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0
.4183.121 Safari/537.36"

Consumer 1 received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183
.121 Safari/537.36"

Consumer 1 received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85
.0.4183.121 Safari/537.36"

Consumer 1 received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.
121 Safari/537.36"
```

FIG 5 :   Each consumer receives and logs the same data, demonstrating the fanout model where each consumer group receives all messages independently.   Useful for scenarios where multiple applications or services need to process the same data simultaneously without interference.


Command Prompt
Command Prompt - python k
Command Prompt - python k
Command Prompt
Command Prompt - python k
Command Prompt - python l

```
Microsoft Windows [Version 10.0.22621.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zanbo>cd "C:\Users\zanbo\OneDrive\Bureau\kafkass"

C:\Users\zanbo\OneDrive\Bureau\kafkass>python kafka_consumer2.py
Starting Consumer 2
Consumer 2 received log entry: 192.168.1.1 - - [11/Nov/2024:12:10:03 +0000] "GET /index.html HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0
.4183.121 Safari/537.36"

Consumer 2 received log entry: 192.168.1.2 - - [11/Nov/2024:12:11:06 +0000] "POST /login HTTP/1.1" 200 234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183
.121 Safari/537.36"

Consumer 2 received log entry: 192.168.1.3 - - [11/Nov/2024:12:12:12 +0000] "GET /images/logo.png HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85
.0.4183.121 Safari/537.36"

Consumer 2 received log entry: 192.168.1.4 - - [11/Nov/2024:12:15:00 +0000] "GET /about HTTP/1.1" 200 512 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.
121 Safari/537.36"
```

So here for the fanout we see that the consumers receive the same log entries. The configuration show that the   each one act independently, receiving a full copy of the log.

References

1. Kreps, J., Narkhede, N., & Rao, J. (2011). *Kafka: A Distributed Messaging System for Log Processing.* LinkedIn. Available at: https://kafka.apache.org

2. Confluent. (2023). *Apache Kafka Overview.* Confluent Documentation. Available at: https://docs.confluent.io

3. Amazon Web Services. (2023). *What is Amazon Simple Notification Service (SNS)?* Available at: https://aws.amazon.com/sns/

4. Pivotal Software. (2023). *RabbitMQ Overview.* RabbitMQ Documentation. Available at: https://www.rabbitmq.com

5. Red Hat. (2023). *Apache ActiveMQ Overview.* Available at: https://activemq.apache.org

6. Kafka Documentation. *Producer and Consumer API.* Available at: https://kafka.apache.org/documentation