Lab 5

Distributed Software Systems
UNIBO - 2024/25

Anouk Wuyts - 1900124167

Davide De Rosa - 1186536

Indirect Communication

Method where **components** or **processes** do not communicate directly with each other. Instead, they exchange information through **intermediaries**, such as **message queues**, **shared data spaces**, or **publish-subscribe** mechanisms.

This approach decouples the *sender* from the *receiver*, meaning the sender doesn’t need to know the identity or location of the receiver, and they don’t need to be active at the same time.

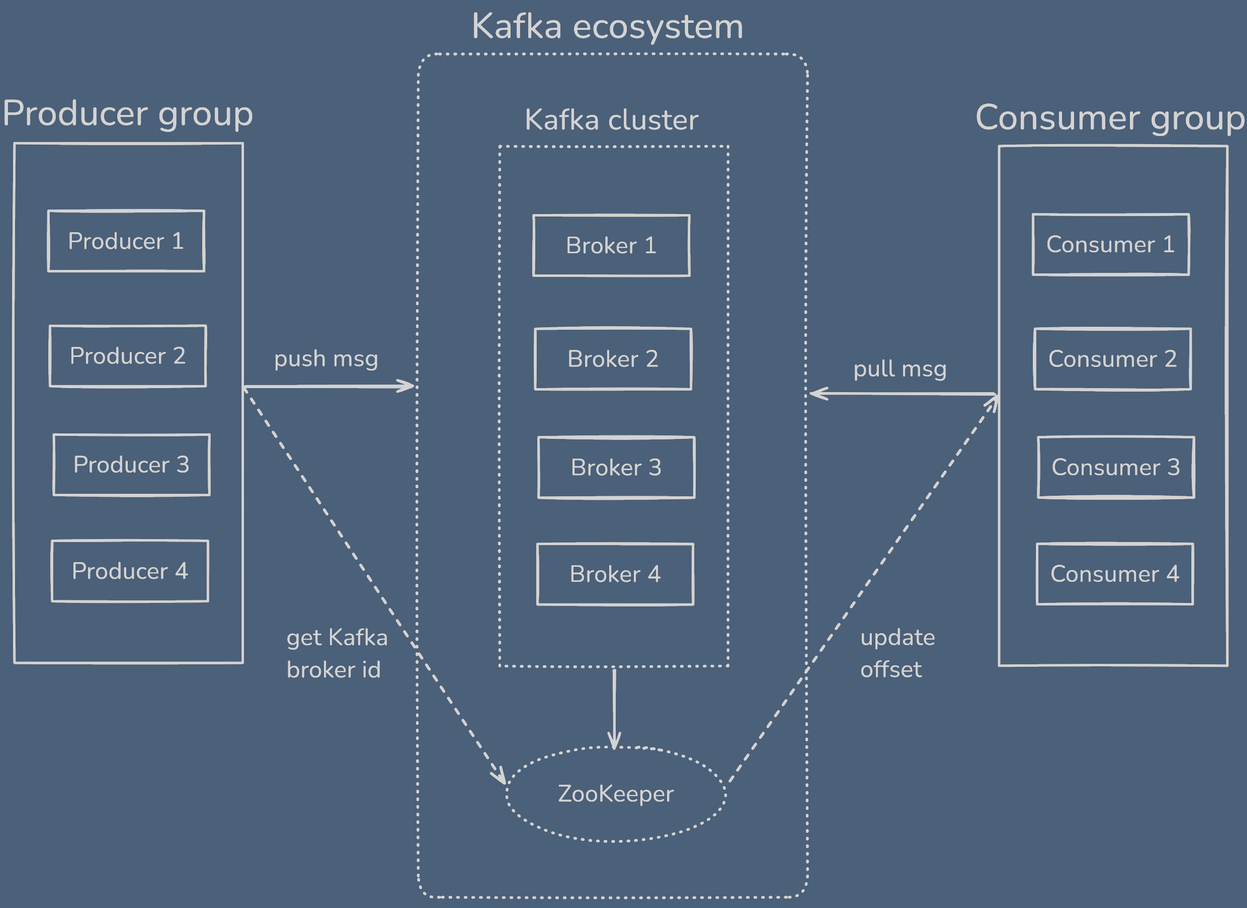
It allows for **flexibility**, **scalability**, and **easier management** of **distributed systems**. Examples include *message brokers* or *event-driven* architectures.

Apache Kafka

An open-source, distributed event streaming platform designed for high-throughput, real-time data pipelines and messaging.

The Kafka ecosystem is composed of a **Kafka Cluster**, which consists of a different number of **Brokers**.

It all works through **ZooKeeper**, which is a centralized service used by Kafka to manage cluster metadata, leader election, and coordinate brokers.



Different definitions

Kafka Cluster: a group of *Kafka brokers* working together to manage and store data streams in a distributed, fault-tolerant system.

Kafka Broker: a server in a *Kafka cluster* that stores and serves data streams, handling message storage and retrieval for clients.

Kafka Topic: a logical channel in *Apache Kafka* where messages are published by producers and consumed by consumers. Each topic can have multiple producers and consumers, supporting real-time data streaming.

Producers: a client application that publishes messages to *Kafka topics*, sending data to brokers.

Consumers: a client application that reads and processes messages from *Kafka topics*, consuming data from brokers.

Implementation - Setup

First of all, we had to download and setup everything we needed.

We started by downloading **Apache ZooKeeper** and **Apache Kafka**.

We installed it locally, but it would have been optimal to use *Docker containers*.

Once everything was installed, we started both services on the machine.

When they were up and running, we created the *test-topic* through the console via the following command:

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

Implementation - Producer

Using **Python**, we created a script to run a *Producer*.

We used the library **confluent_kafka** to manage the connection between the script and the *broker*.

The main function checks if the user wants to stop the execution of the script. If true, the producer is interrupted.

```
bootstrap_servers = 'localhost:9092'
topic_name = 'test-topic'
producer = Producer({'bootstrap.servers': bootstrap_servers})
```

```
if __name__ == '__main__':
    try:
        produce_messages()
    except KeyboardInterrupt:
        print("Kafka Producer interrupted")
    finally:
        producer.flush()
```

Implementation - Producer

The main function of the *producer* constantly produces a random *message* and pushes it to the *Kafka Broker,* once every 5 seconds.

producer.poll(0) checks for and processes the events in the background **without blocking the main thread**.

If the delivery of the message fails, an error is printed.

```python
def produce_messages():
    print("Starting Kafka producer...\n")

    while True:
        message = {
            'timestamp': datetime.now().strftime('%H:%M:%S'),
            'content': random.randint(0, 100)
        }

        producer.produce(topic_name, value=json.dumps(message), callback=delivery_report)
        print(f'[{message["timestamp"]}] Message sent: {message["content"]}')

        producer.poll(0)
        sleep(5)
```

```python
def delivery_report(err, msg):
    if err is not None:
        print(f"Message delivery failed: {err}")
    else:
        print(f"Message delivered to {msg.topic()} [{msg.partition()}]")
```

Implementation - Consumer

Using **Python**, we created a script to run a *Consumer*.

We used the library **confluent_kafka** to manage the connection between the script and the *broker*.

The consumer has to subscribe to the topic he wants to listen to.

The main function checks if the user wants to stop the execution of the script. If true, the consumer is interrupted.

```
bootstrap_servers = 'localhost:9092'
topic_name = 'test-topic'
consumer = Consumer({
    'bootstrap.servers': bootstrap_servers,
    'group.id': 'test-group',
    'auto.offset.reset': 'earliest'
})
consumer.subscribe([topic_name])
```

```
if __name__ == '__main__':
    try:
        consume_messages()
    except KeyboardInterrupt:
        print("Kafka Consumer interrupted")
    finally:
        consumer.close()
```

Implementation - Consumer

The main function of the *consumer* constantly consumes a random *message* received by the *Kafka Broker.*

consumer.poll(1.0) fetches messages from the topics. It allows the consumer to consume messages from a *Kafka broker* asynchronously.

If the received message is an error, an exception is raised.

Otherwise, the message is consumed and printed.

```python
def consume_messages():
    print("Starting Kafka consumer...\n")

    while True:
        msg = consumer.poll(1.0)

        if msg is None:
            continue

        if msg.error():
            raise KafkaException(msg.error())

        value = json.loads(msg.value().decode('utf-8')) if msg.value() is not None else None

        print(f"Consumed: {value}")
```

Usage

After running both *ZooKeeper* and *Kafka*, we run the *Producer* and the *Consumer* scripts.

The *producer* prints the random message it produced and also acknowledges the delivery to the *test-topic*.

The *consumer* prints the random message it consumed from the *test-topic*.

For both of them, a **KeyboardInterrupt** stops the execution.

```
Starting Kafka producer...

[16:35:11] Message sent: 54
[16:35:16] Message sent: 15
Message delivered to test-topic [0]
[16:35:21] Message sent: 89
Message delivered to test-topic [0]
[16:35:26] Message sent: 90
Message delivered to test-topic [0]
[16:35:31] Message sent: 71
Message delivered to test-topic [0]
[16:35:36] Message sent: 83
Message delivered to test-topic [0]
^CKafka Producer interrupted
```

```
Starting Kafka consumer...

Consumed: {'timestamp': '16:35:11', 'content': 54}
Consumed: {'timestamp': '16:35:16', 'content': 15}
Consumed: {'timestamp': '16:35:21', 'content': 89}
Consumed: {'timestamp': '16:35:26', 'content': 90}
Consumed: {'timestamp': '16:35:31', 'content': 71}
Consumed: {'timestamp': '16:35:36', 'content': 83}
^CKafka Consumer interrupted
```

Thank you!