Apache Kafka Part 3

Rohanda Hamed
SW ML Engineer, Giza systems

Today's Agenda

- Recap on Part 1,2
  - what is Kafka
  - Purpose
  - Use case
  - Kafka Components
- Streaming Platform versus Message Queuing platform
- How to install Kafka
- Kafka Tuning
- Kafka Monitoring and Debugging

About Kafka

Apache Kafka is a distributed event **streaming** platform.

- A **stream** is a flow of data with no ending event.
- It's the opposite of **Batch** which is a group of data with finite size.

About Kafka

Apache Kafka is a distributed **event** streaming platform.

Event is a record of an **action, change, or occurrence** that has significance to a system or application

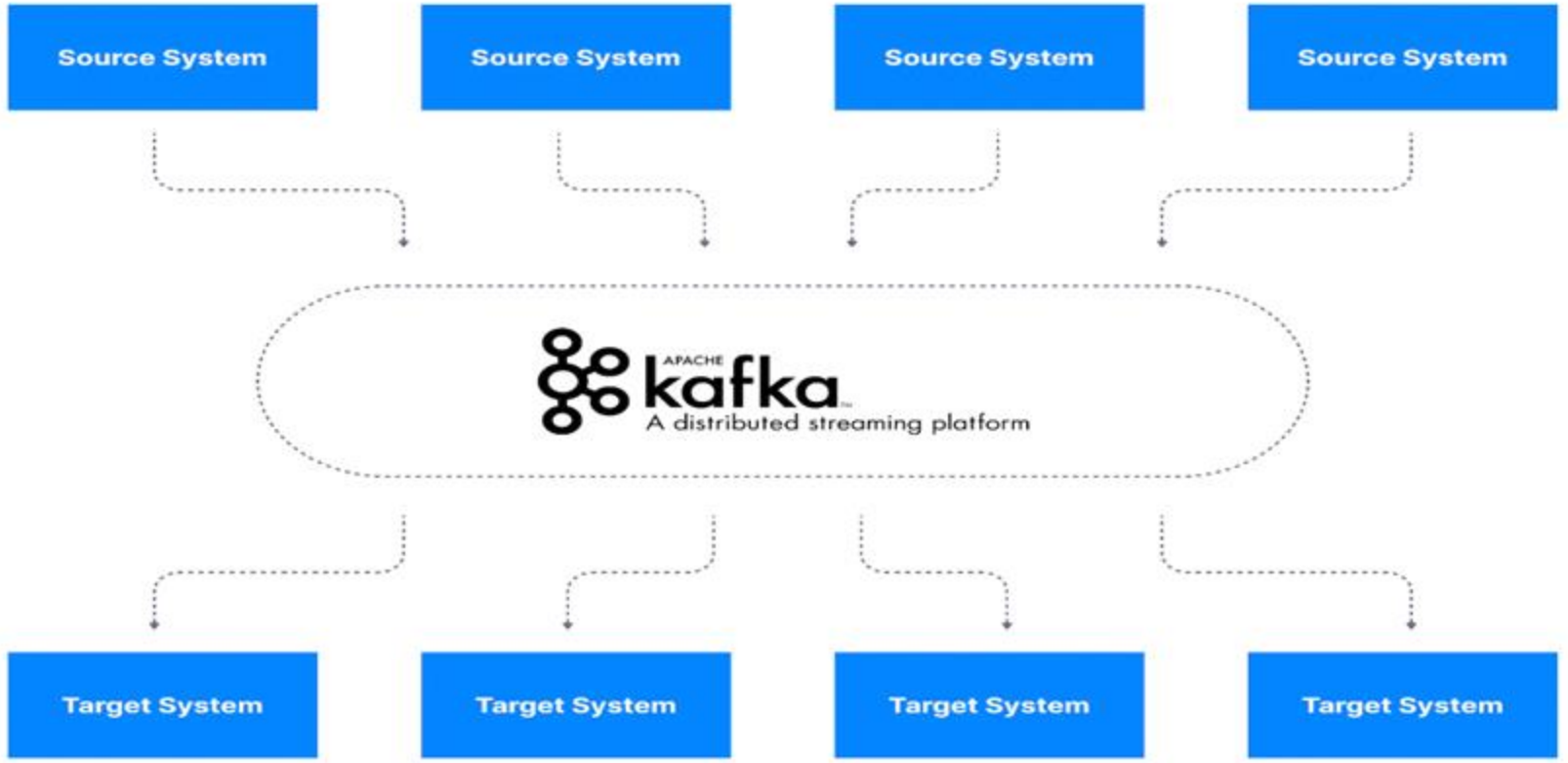
Event has some characteristics:

- Time sensitive
- Immutable
- Data payload
- Asynchronous Communication

Purpose

Build a single distributed pipeline that can achieve:

- low latency
- high throughput
- Resilience
- Scalability
- Simple API for consumers and producers



Use case

Real-Time Order Processing in E-Commerce

You’re building an e-commerce platform with the following requirements:

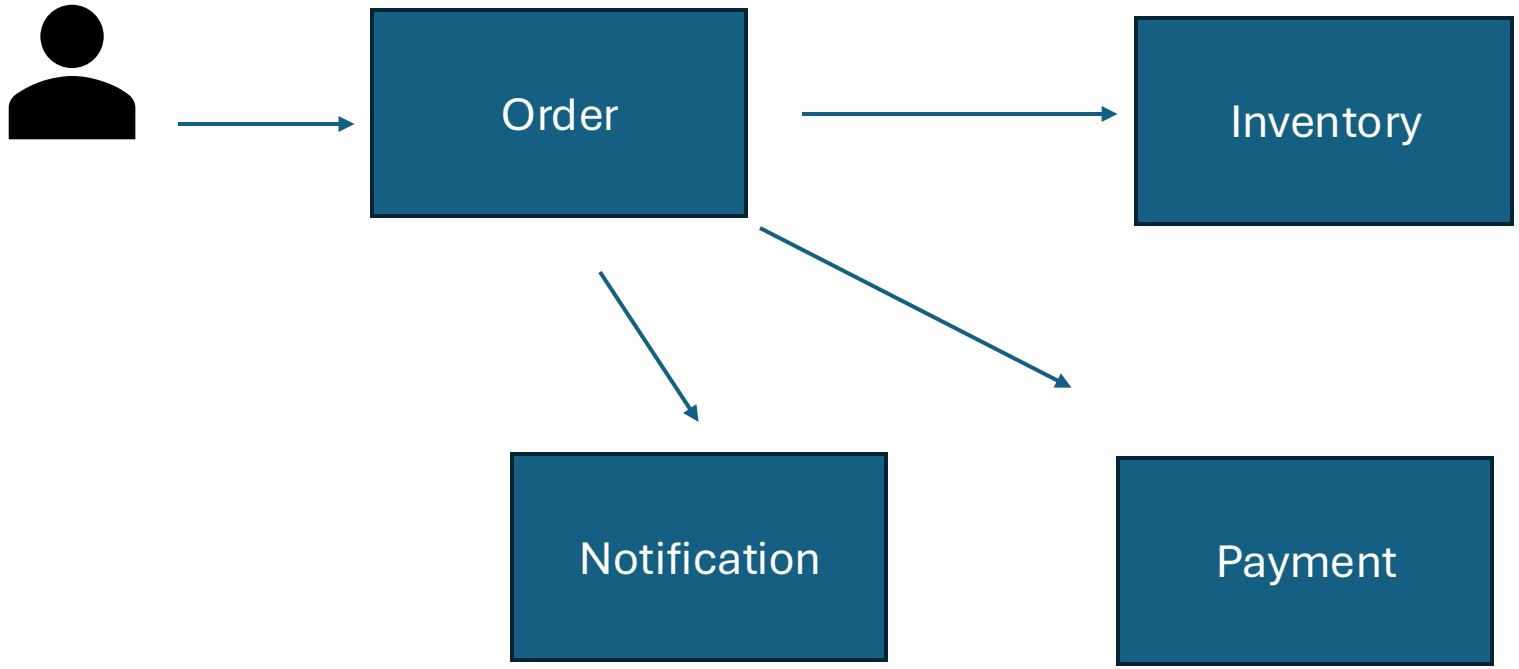
1. **High Throughput**: Handle thousands of orders per second during flash sales.
2. **Real-Time Notifications**: Update inventory, process payments, and notify users as soon as orders are placed.
3. **System Decoupling**: Allow independent teams to manage inventory, payments, and notifications without affecting each other.
4. **Durability**: Ensure no order is lost, even if some systems are temporarily unavailable.
5. **Scalability**: Seamlessly handle peak loads during sales or promotions

Using Http requests

- **Order Service**: Receives HTTP POST requests when users place orders.
- **Inventory Service**: API call to update inventory levels.
- **Payment Service**: API call to process payments.
- **Notification Service**: API call to send user notifications.

Challenges:

1. Tight Coupling
2. Scalability Issues
3. Error Handling
4. Data Loss
5. Real-Time Notifications



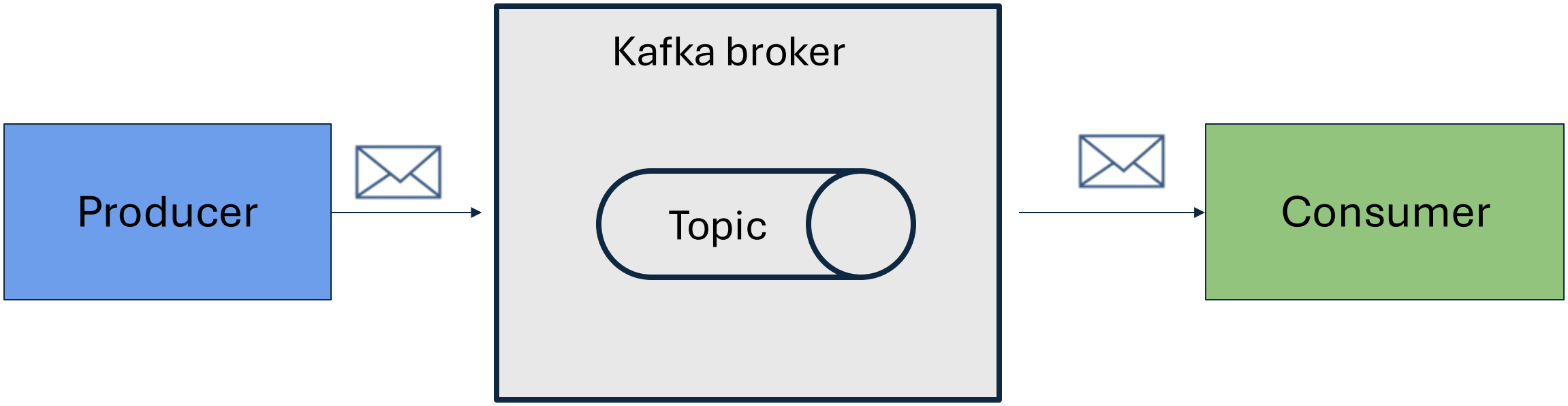
Using Kafka

- **Order Service:** Publishes an event to a Kafka topic (orders).
- **Inventory Service:** Subscribes to the orders topic, consumes events, and updates inventory in real-time.
- **Payment Service:** Subscribes to the orders topic and processes payments asynchronously.
- **Notification Service:** Subscribes to the orders topic to send order confirmation messages to users.

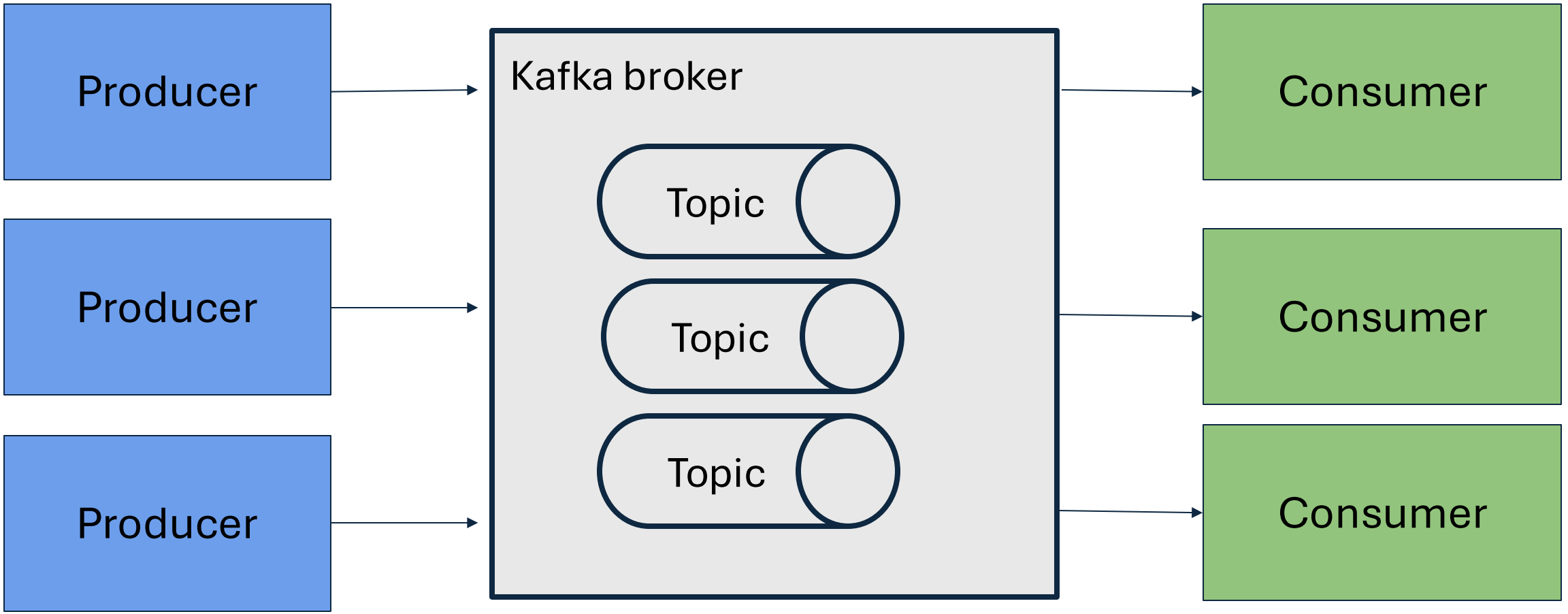
Advantages:

1. Decoupling
2. High Throughput
3. Durability and Fault Tolerance
4. Scalability

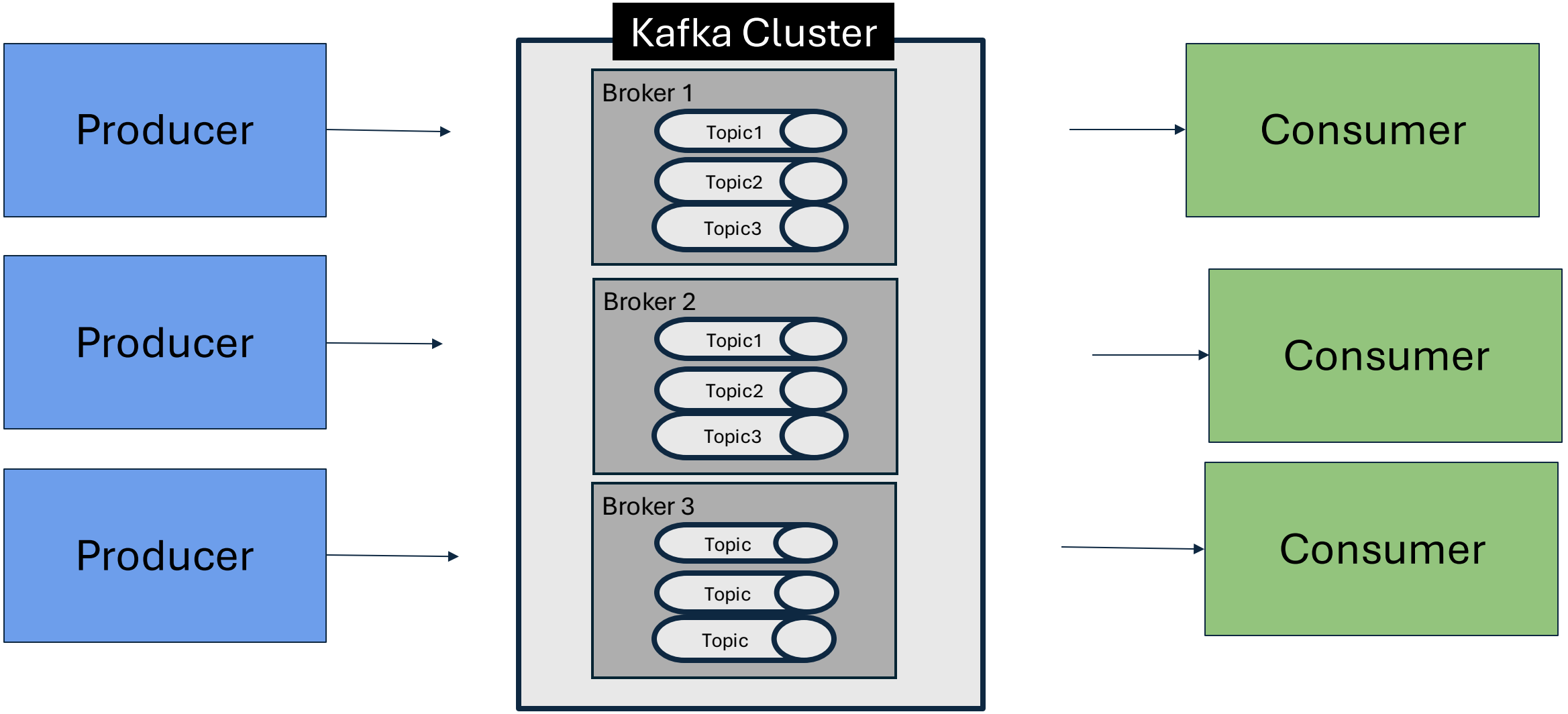
Kafka components



Kafka components

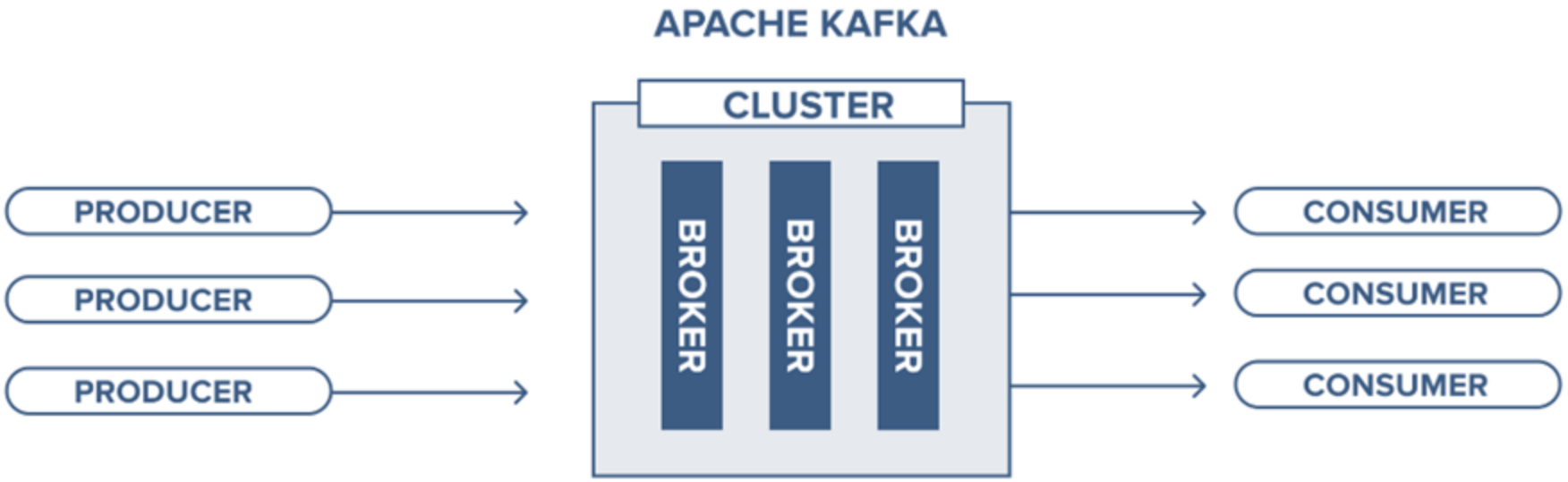


Kafka components



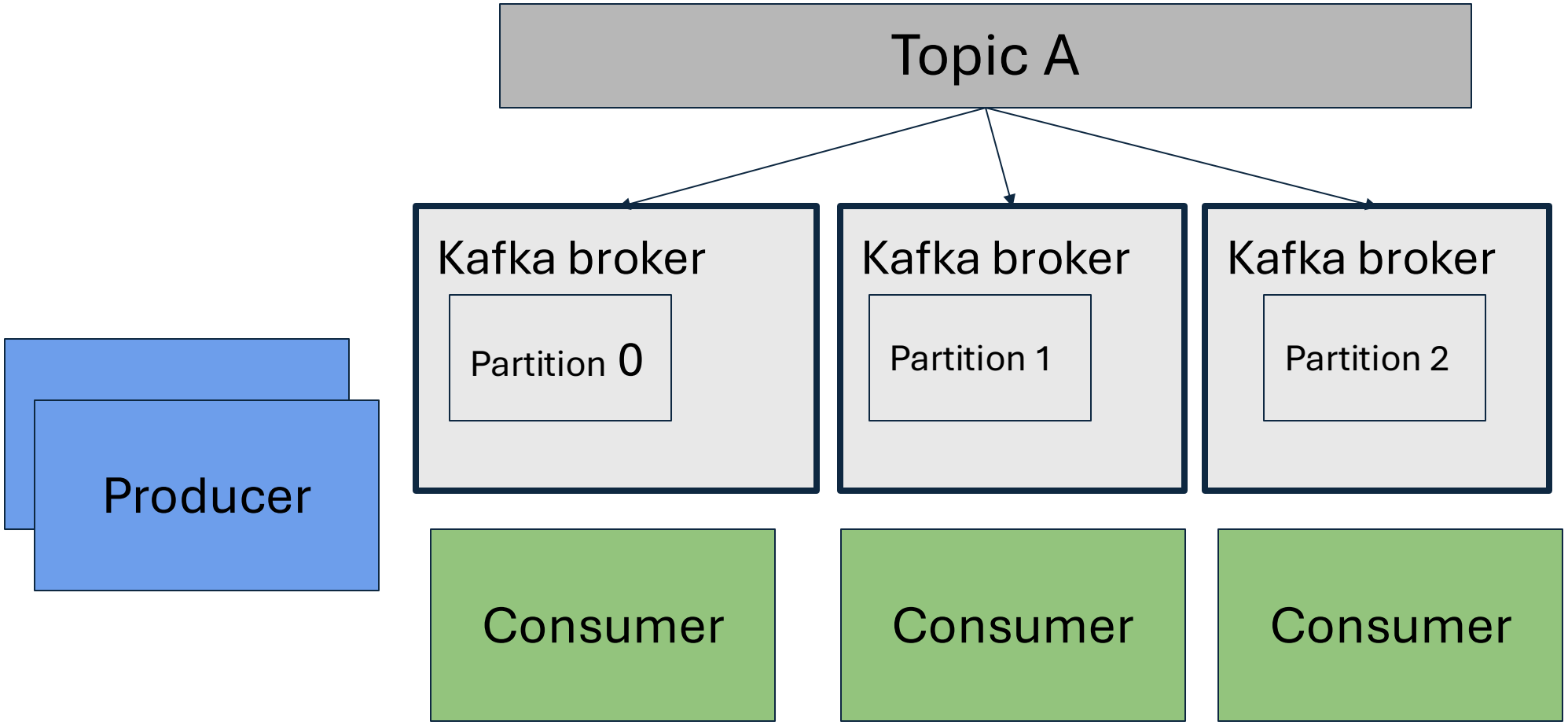
Kafka broker /clusters

- A broker is a server that hold kafka topics with all its partitions .
- A Kafka cluster comprises one or more kafka brokers.
- A broker is identified by a unique ID within kafka cluster.
- It handles the requests from clients (producers and consumers)

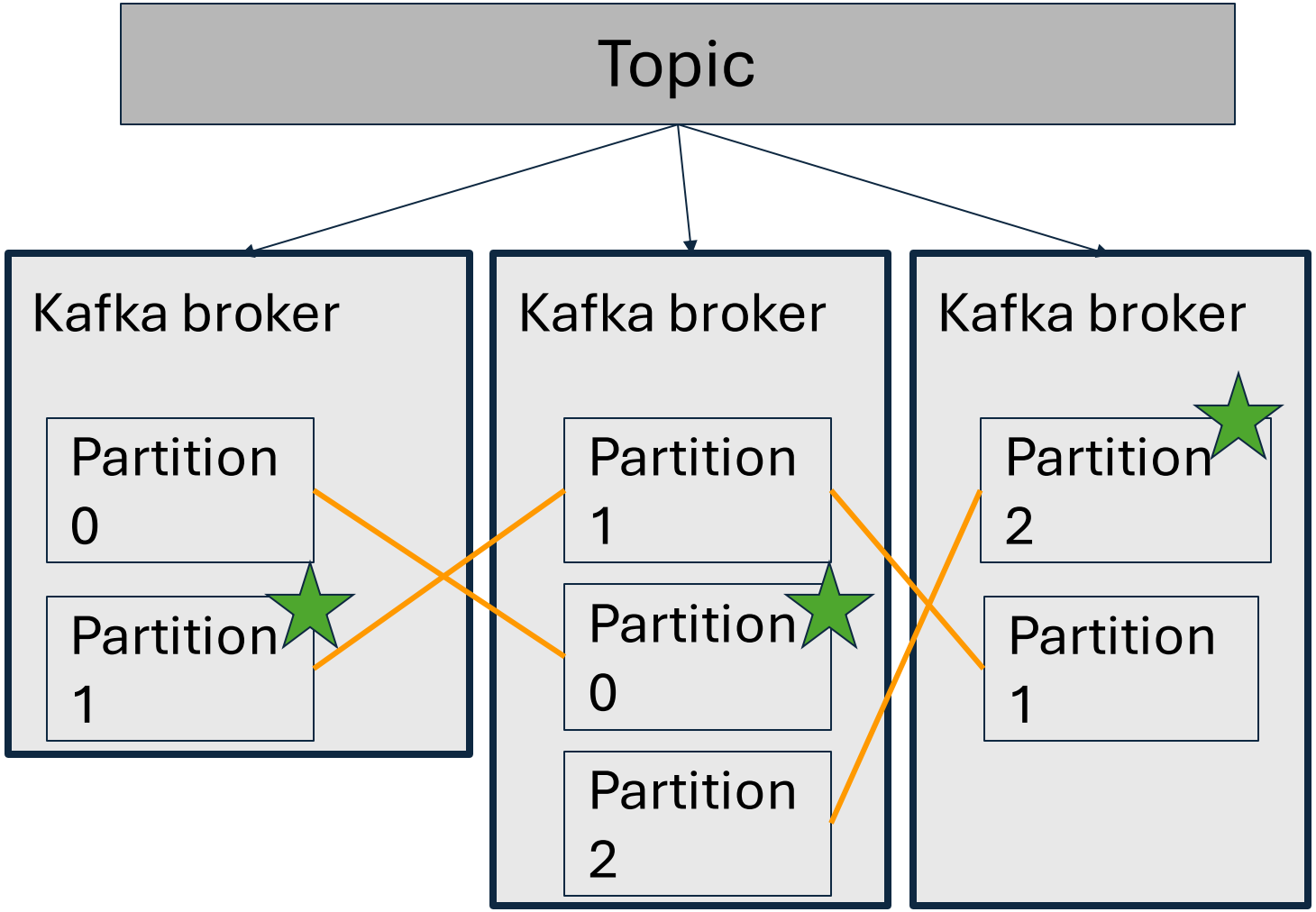


Kafka Topic

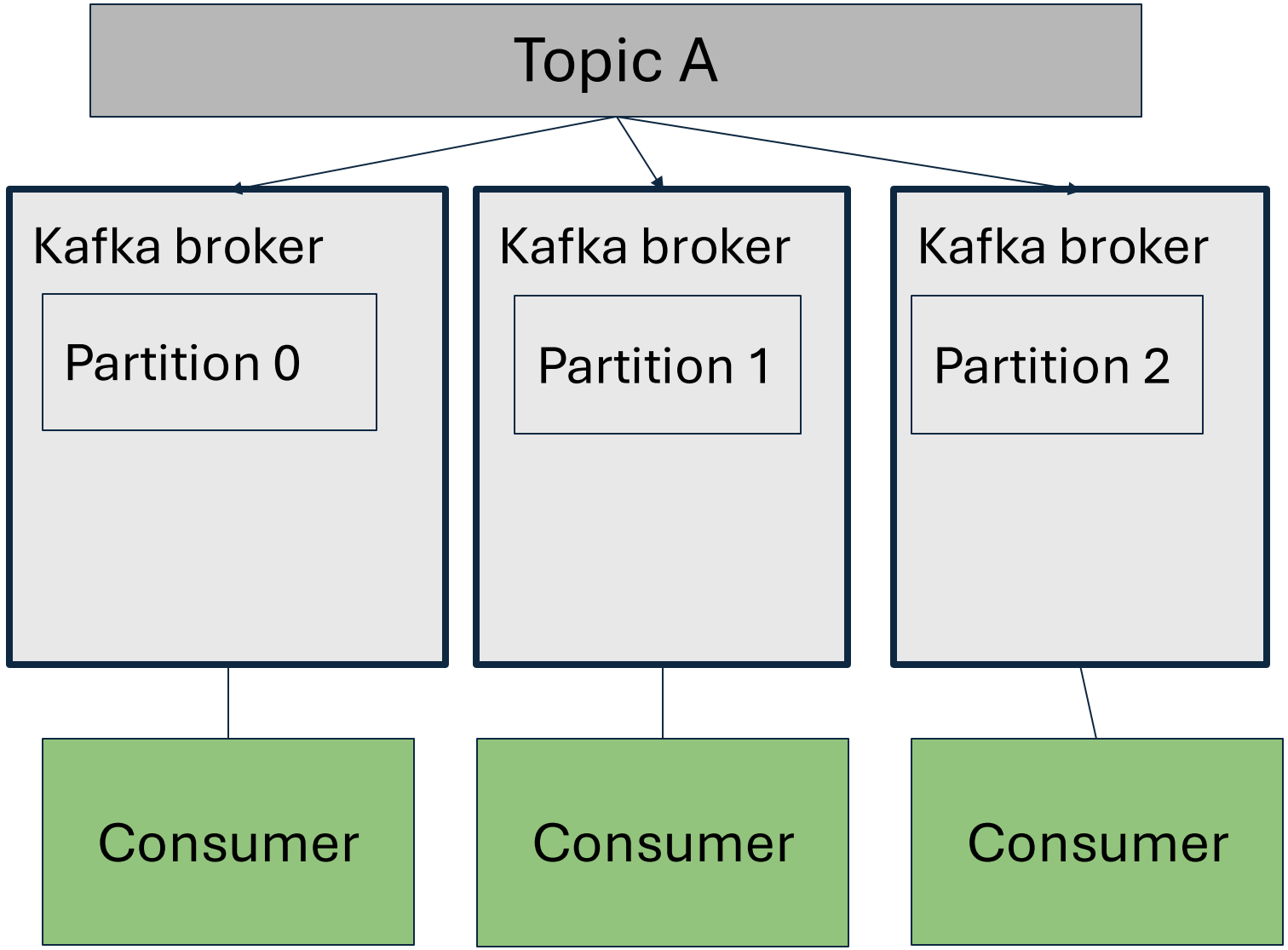
A topic is where messages of same category get stored.
A topic is divided into partitions , which is a unit of parallelism in kafka.



Kafka Topic replication

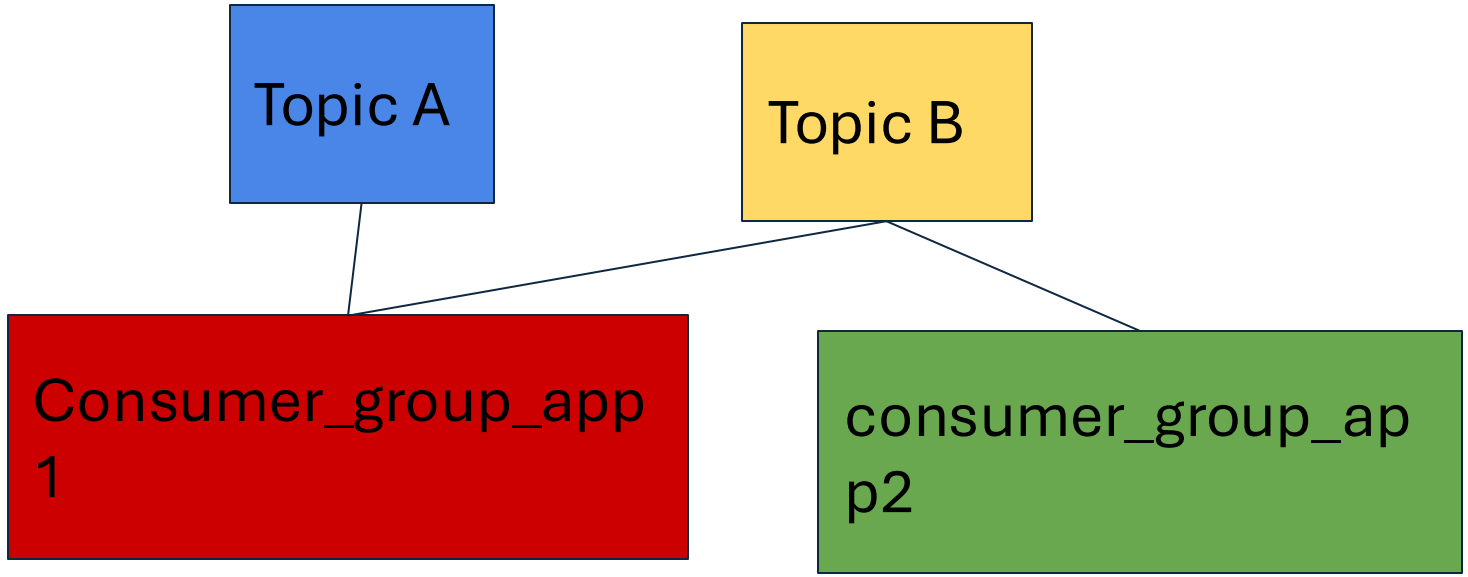


Kafka consumer - partition



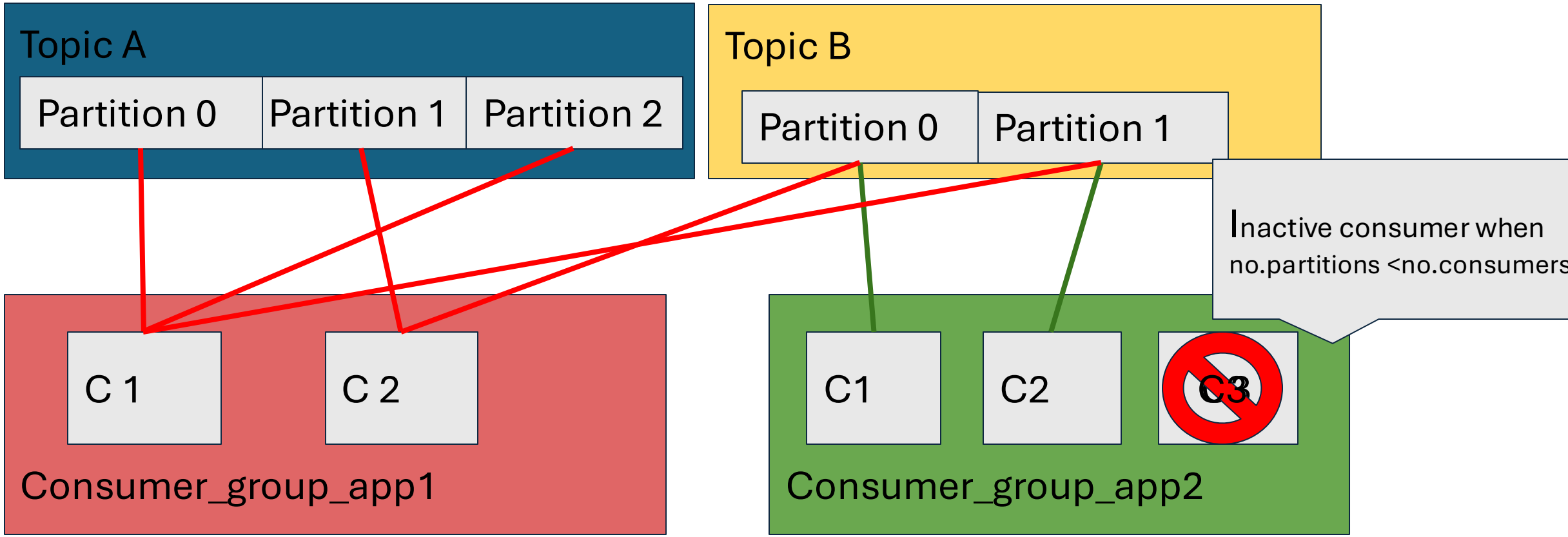
Kafka consumer group

- A consumer is a member in consumer group.
- A group may be assigned to an application or a service .
- A set of topics are assigned to a group .



Kafka consumer group

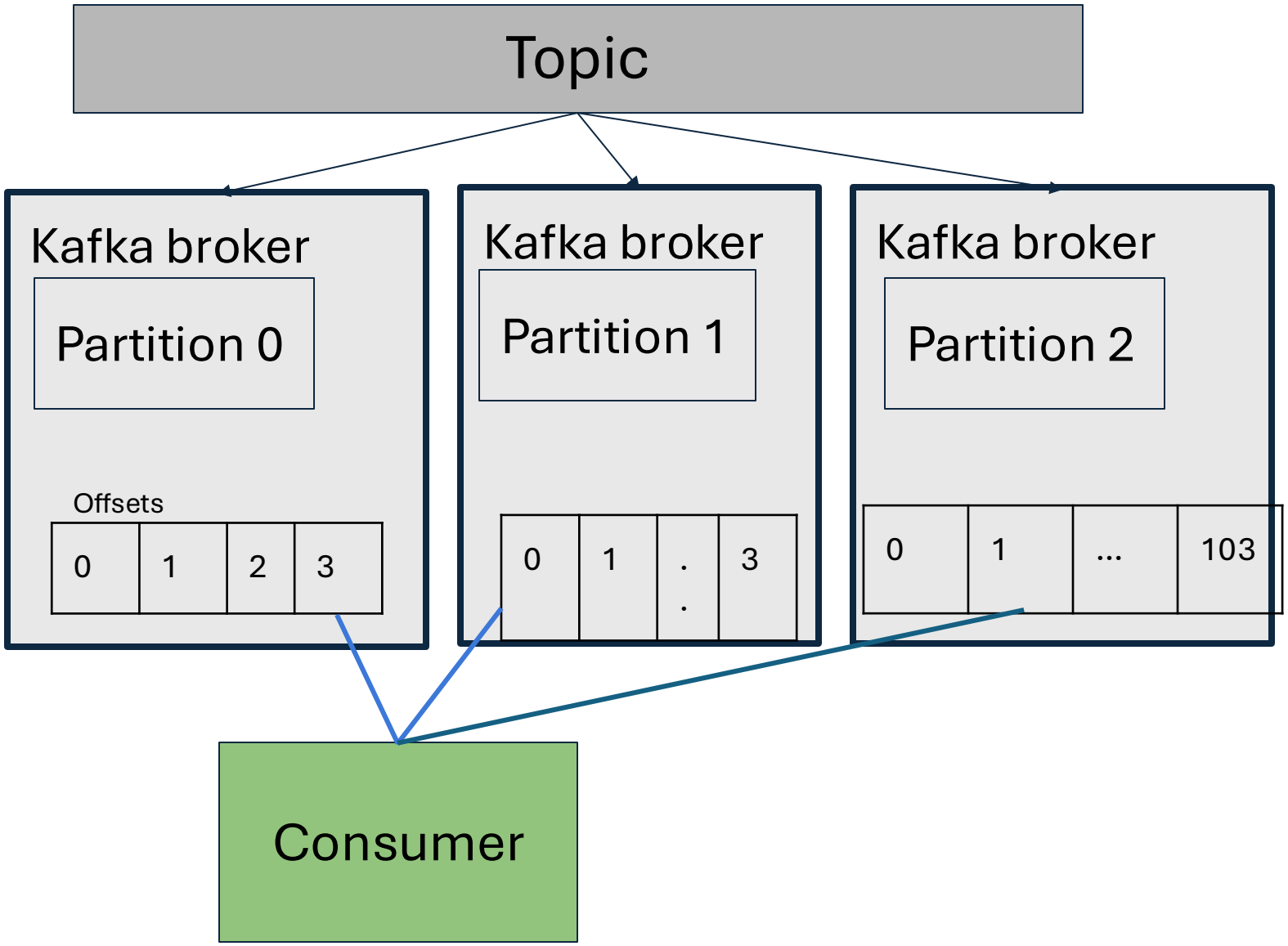
- Each consumer group should read from all partitions of the topic assigned to it.
- A single consumer in a group can read from multiple partitions
- A partition is assigned to one consumer in a consumer group



Kafka partition offsets

It is a number that tracks the consumption of messages per partition

There are two offsets for each consumer:
Current offset: last message sent to a given consumer
Committed offset: last message acknowledged by consumer



QUIZ

Streaming versus Message queueing

Streaming platform	Messaging Platform
Built for handling large streams of continuous data, often in real-time, with support for event-driven architectures and real-time analytics.	Primarily designed for sending discrete messages between systems or services
Stream-based, continuous, and ordered (time series, logs, sensor data, etc.).	Event-driven, message-oriented.
Designed for low-latency, high-throughput streaming of data(ex:partition and consumer group)	Generally low latency
Support replay and long term persistence	Once message consumed no longer needed unless persisted
Event sourcing	Mainly for dispatching events to workers

When to use RabbitMQ

RabbitMQ is ideal for traditional **message queuing** scenarios, especially when reliability and flexibility in routing are critical.

Key Scenarios for RabbitMQ

1. **Asynchronous Task Processing:**
   - **Example:** A web app offloads tasks like sending emails, generating reports, or processing images.

2. **Request/Response Patterns:**
   - **Example:** A service calls another service and expects a response (e.g., processing payment requests and returning a confirmation).

3. **Low Volume, Latency-Sensitive Use Cases:**
   - **Example:** Chat applications or microservices where low latency is crucial for user interactions.

4. **Complex Routing Needs:**
   - **Example:** You want to route messages dynamically based on headers, topics, or fan-out patterns.

When to use Kafka

Kafka is best for **event-driven architectures** and **high-throughput, real-time data streaming** scenarios.

Key Scenarios for Kafka

1. Event Streaming and Real-Time Data Pipelines:

Example: Streaming user activity logs or IoT sensor data for real-time processing.

2. High Volume, Scalable Systems:

Example: A social media platform that needs to process millions of user-generated events per second.

3. Event Replay and Long-Term Storage:

Example: A fraud detection system that reprocesses historical transactions.

4. Distributed Systems with Many Consumers:

Example: A system where multiple services consume the same event stream for different purposes (e.g., analytics, notifications, ML pipelines).

5. Data Integration Across Systems:

Example: Centralizing data from multiple systems (e.g., databases, applications) into a data lake for analytics.

How to install kafka on windows

Prerequisites:

- Install Java
- Add JAVA_HOME to environment variables

Kafka Download:

- Download kafka from here
- Add the location of bin/windows folder to PATH environment variable

Running Kafka Server:

cd to kafka installation directory

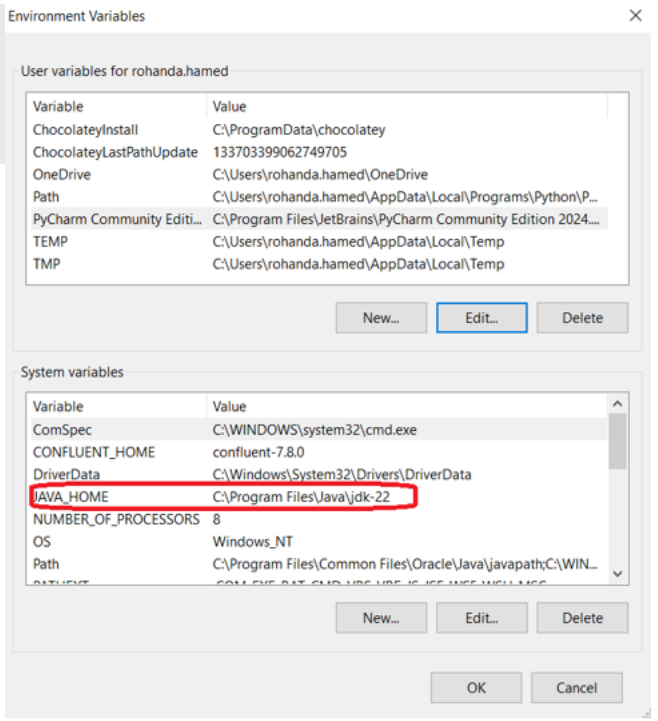
```
C:\kafka_2.12-3.9.0>zookeeper-server-start.bat .config/zookeeper.properties
```

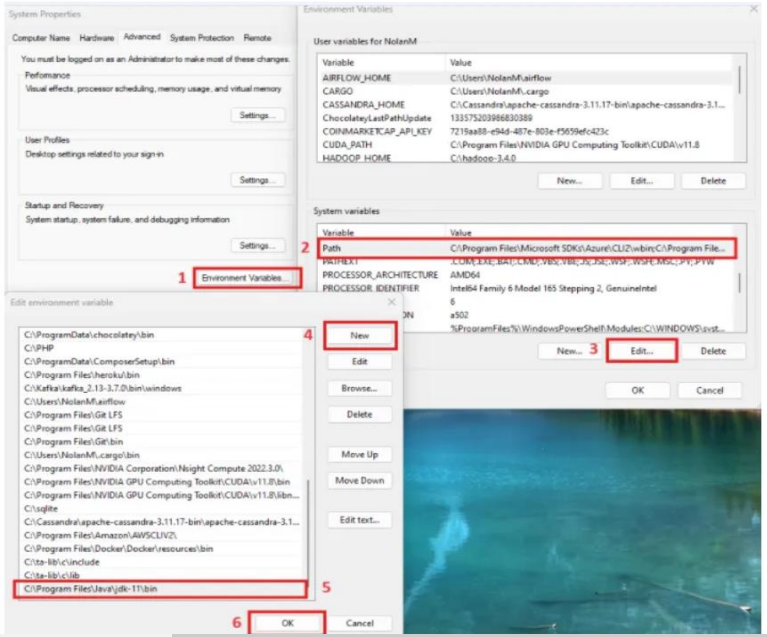
Second run kafka broker :

```
C:\kafka_2.12-3.9.0>kafka-server-start.bat config/server.properties
```

Now, we can create topics, consumer and producers

- You can find the CLI commands here





How to install Kafka on Docker

- cd to the docker-compose file location
- Run the command

```
docker-compose up –d broker
```

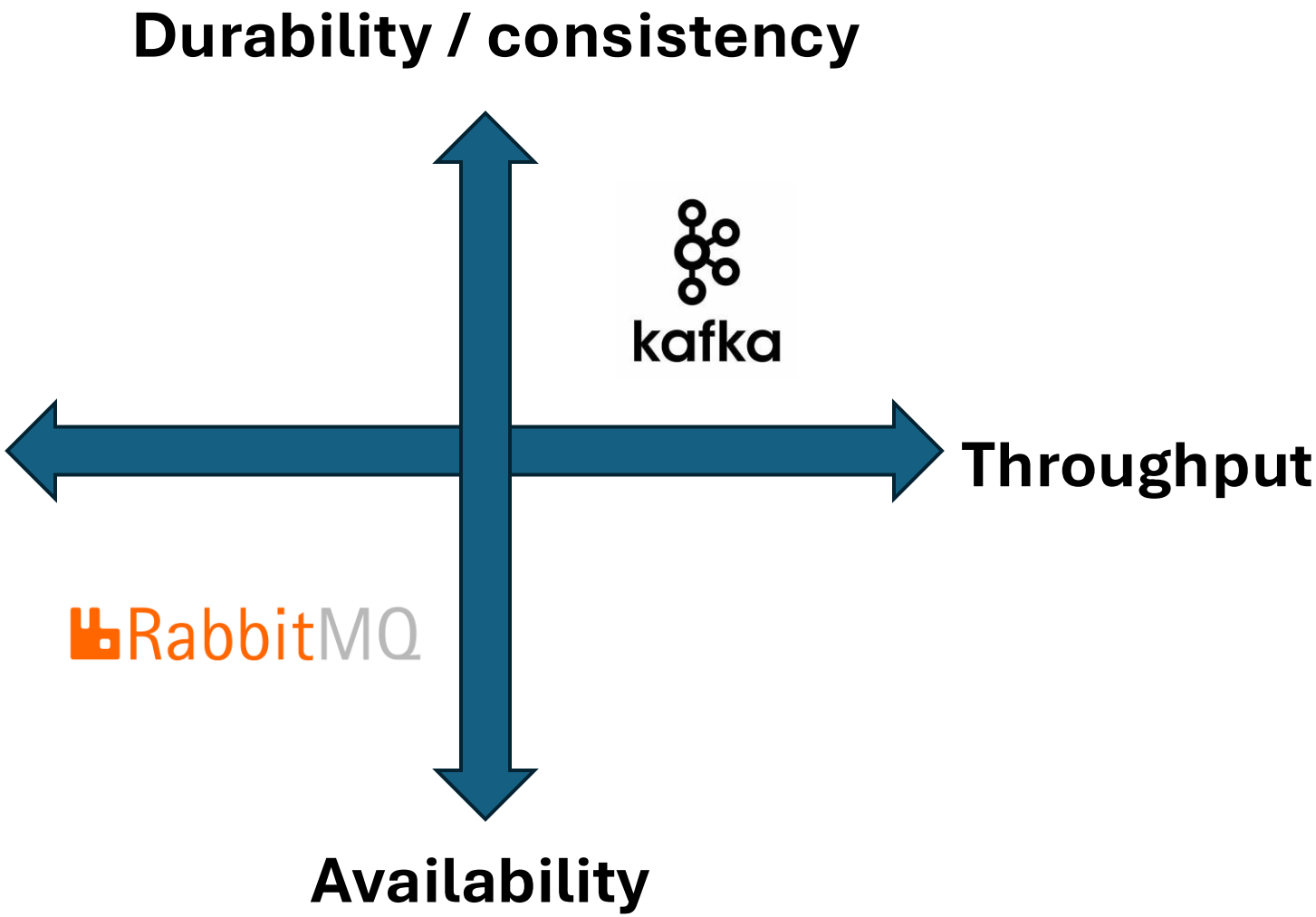
```yaml
services:
  broker:
    volumes:
      - ./demo:/demo
    environment:
      KAFKA_NODE_ID: 1
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: 'CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT'
      KAFKA_ADVERTISED_LISTENERS: 'PLAINTEXT://broker:29092,PLAINTEXT_HOST://localhost:9092'
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_JMX_PORT: 9101
      KAFKA_JMX_HOSTNAME: localhost
      KAFKA_PROCESS_ROLES: 'broker,controller'
      KAFKA_CONTROLLER_QUORUM_VOTERS: '1@broker:29093'
      KAFKA_LISTENERS: 'PLAINTEXT://broker:29092,CONTROLLER://broker:29093,PLAINTEXT_HOST://0.0.0.0:9092'
      KAFKA_INTER_BROKER_LISTENER_NAME: 'PLAINTEXT'
      KAFKA_CONTROLLER_LISTENER_NAMES: 'CONTROLLER'
      KAFKA_LOG_DIRS: '/tmp/kraft-combined-logs'
      # Replace CLUSTER_ID with a unique base64 UUID using "bin/kafka-storage.sh random-uuid"
      # See https://docs.confluent.io/kafka/operations-tools/kafka-tools.html#kafka-storage-sh
      CLUSTER_ID: 'MkU3OEVBNTcwNTJENDM2Qk'
```

Kafka Optimization

The most important parameters to optimize for are:

- Latency : Amount of time taken to process single message
- Throughput: Number of messages that can be processed per second
- Durability: Retention of received data in face of possible failure
- Availability: Ability of the system to remain operational and accessible, even in the face of failures

Kafka Optimization



Kafka Optimization

- To optimize for Throughput

Producer	Consumer
Increase batch.size	Increase fetch.size
Increase linger time	Increase fetch.time
Increase number of partitions	Increase number of consumers to be equal to number of partitions
Compress messages	Increase poll number of records

Increasing wait time optimize for throughput

- To optimize for Durability

Producer	Consumer
Increase Replicas	Decrease auto commit interval
Increase Acks	Decrease session timeout
Increase min.insync.replicas	Increase heartbeat interval

Increasing metadata and sanity checks optimize for Durability

QUIZ

Kafka Monitoring & debugging

- **Confluent Kafka control center** can be used to monitor Kafka topics, consumers , producers and brokers

- Low producer throughput → check that all partitions are receiving messages
- Low consumer throughput → check that partitions are balanced across consumers

Assignment 1/3

Part 1: Data Preparation

1. **Download the Dataset:**
   - Download the provided folder containing a set of images.
2. **Extract the Data:**
   - Unzip or extract the contents of the downloaded folder into a local directory.
3. **Label the Data:**
   - Rename each image file to represent the date of capture in Arabic format (e.g., YYYY-MM-DD as ٢٠٢٤-سبتمبر-١٠).
   - Rotate any inverted image so it's upright

Part 2: Kafka Producer Implementation

4. **Create a Kafka Producer:**
   - Develop a Kafka producer script to read the directory of labeled images.
   - For each image, send the following data as a Kafka message:
     - **Image File**: Serialize the image (e.g., as Base64 or binary).
     - **Annotation**: Include the image filename as metadata (representing the Arabic date).

Assignment 2/3

Part 3: Kafka Consumer Implementation

5. **Create a Kafka Consumer:**
    - Develop a Kafka consumer script to subscribe to the image-data topic.
    - Read messages containing the serialized image and annotation.
    - Deserialize the image and save it to a local directory.
    - Write in a CSV file of two columns one for the data path and the other for image annotation (Arabic date annotation) .

Part 4: Tuning Kafka performance

6. **Test Multiple configurations for improving performance (try to achieve high throughput (process all data in less time))**

7. **Monitor Kafka Performance:**
    - Use Kafka control center to monitor:
        1. Producer throughput.
        2. Consumer lag.
        3. Partition distribution.

Assignment 3/3

Deliverables

1. **A zip folder with labeled dataset with images renamed in Arabic date format.**
2. **Kafka producer code that reads and publishes images and annotations.**
3. **Kafka consumer code that retrieves, deserializes, and saves images and annotations.**
4. **A csv file for the tried configurations and the performance of Kafka achieved for each set of configurations (latency and throughput)**

Deadline for submission: Thursday 19/12/2024

References

- Troubleshooting and debugging Kafka Linkedin course
- Tuning Kafka Lindkedin course
- Kafka | Confluent Documentation