Kafka

Agenda

- Introduction
- Event Streaming
- Notification
- Publish and Subscribe Model
- Messaging System
- Kafka Streams Topology
- Brokers and Topics
- Topic Replication
- Kafka CLI
- Consumers

Streaming

1. The technology of transmitting data in a continuous flow over a wired or wireless internet connection.
2. Streaming refers to any data delivered to computers and mobile devices via the internet
   - TV shows and music videos are common forms of streaming content.

Event Streaming

- Challenges with microservices
  - Coupling
- Don’t tell architecture, data is gathered on demand.
  - Service A asks Services B, C, and D, “What’s your current state?”
  - This assumes B, C, and D are always available to respond.
  - However, if they moved or they’re offline, they can’t respond to Service A’s query.

Event Streaming

- Event streaming attempts to solve this problem
- An event-driven architecture utilizes a *tell* - don’t ask approach
  - Services B, C, and D *publish* continuous streams of data as events.
  - Service A subscribes to these event streams
    - Processing the results, and caching/storing them locally
- However, Service A only needs to act, or perform its function, when it is delivered a specific type of event.

Notification

- Benefit of implementing notification feature
  - User Engagement
    - Increases user engagement
      - E-commerce — 278%, Music — 177%, Food & Drink -100%
    - Read within the first 5 minutes compared with emails.
  - Usability & Effectiveness
    - Immediate responses and reception of important information.
    - Help to maximize the efficiency and productivity of workers.
    - Quick and easy to manage and track.

Types of Notification

- On Page/ Real Time Notifications
  - Triggered by the server when application is open.
  - User is connected and using the application.
  - Usually opens a browser pop-up or displays a message within the application window.
  - Message delivery managed by the application itself.
- Off Page/Push Notifications
  - Pushed by the server even when application is not running on browser.
  - User is disconnected and not using the application.
  - Opens a bubble outside the application window.
  - Message delivery managed by individual browser’s push service.

Data Flow

- Data is every where
- Data flows from
  - Method – Method
  - Class – Class
  - Module – Module
  - System – System

What is a Messaging System?

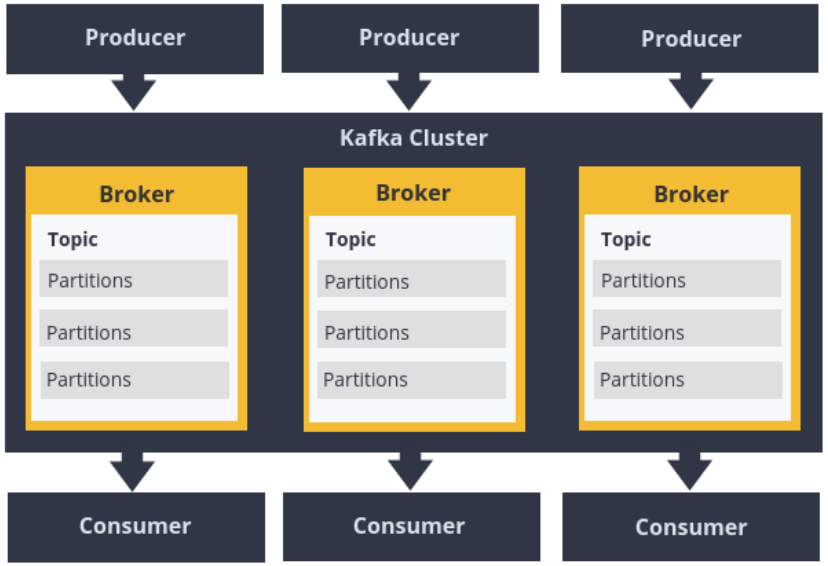
- A Messaging System is responsible for transferring data from one application to another
- Applications can focus on data, but not worry about how to share the data.
- Distributed messaging is based on the concept of reliable message queuing.
- Messages are queued asynchronously between client applications and messaging system.
- Two types of messaging patterns are available
  - point to point messaging system.
  - publish-subscribe (pub-sub) messaging system.
    - Most of the messaging patterns follow **pub-sub**.

Messaging systems

- Loose coupling between modules
- Queuing data for later delivery
- Asynchronous processing
- Reliable load balancing

Kafka Architecture

- High volume publish-subscribe messages and streams platform
  - Durable, fast and scalable.
- Durable message store
  - like a log, run in a server cluster, which keeps streams of records in topics (categories).
- Messages
  - made up of a value, a key and a timestamp.
- Dumb broker / smart consumer model
  - does not try to track which messages are read by consumers and only keeps unread messages. Kafka keeps all messages for a set period of time.
- Requires external services to run
  - in some cases Apache Zookeeper.



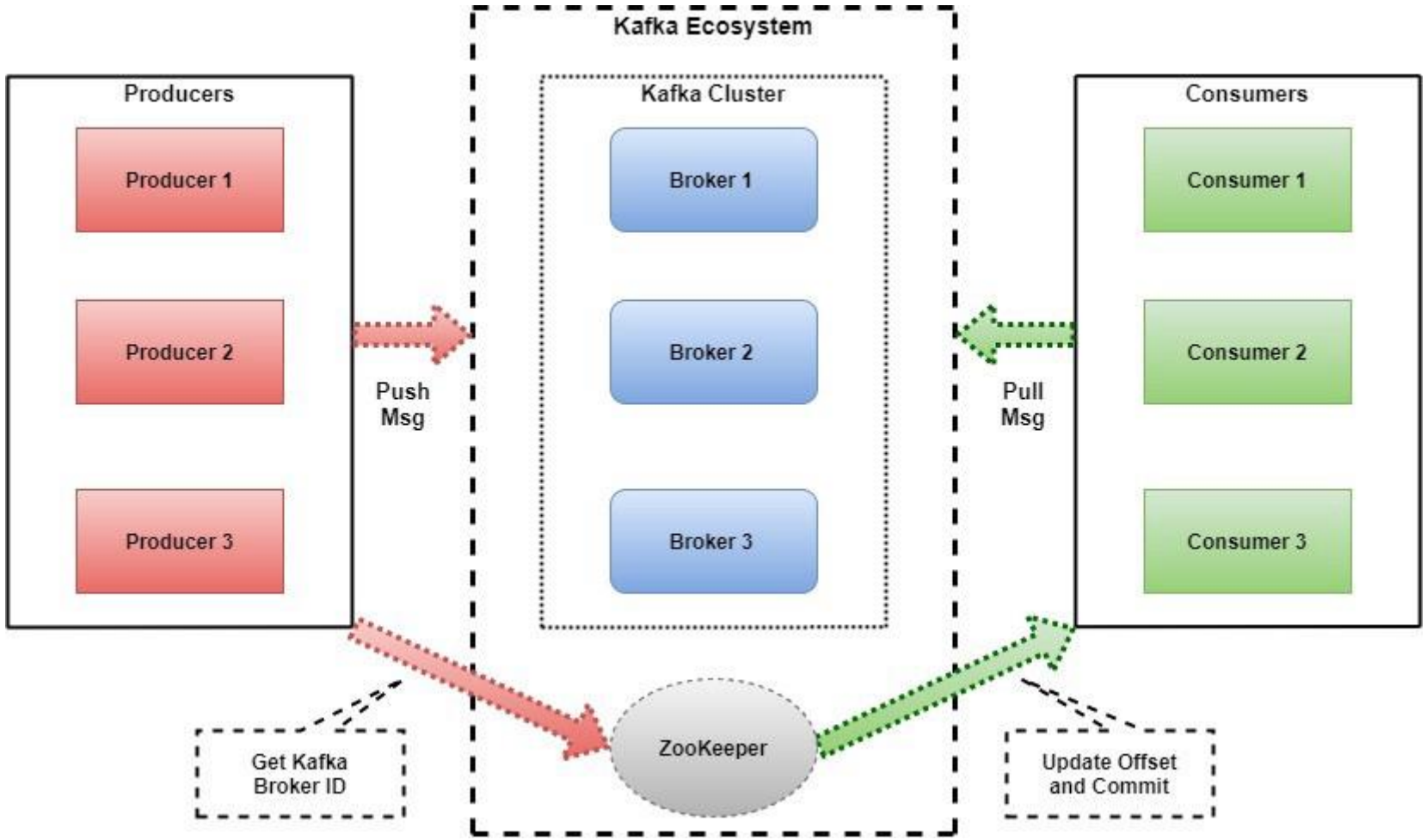
Kafka Terminology

- Topics
  - A stream of messages belonging to a particular category is called a topic. Data is stored in topics.
  - Topics are split into partitions. For each topic, Kafka keeps a mini-mum of one partition. Each such partition contains messages in an immutable ordered sequence. A partition is implemented as a set of segment files of equal sizes.
- Partition
  - Topics may have many partitions, so it can handle an arbitrary amount of data.
- Partition offset
  - Each partitioned message has a unique sequence id called as offset.
- Replicas of partition
  - Replicas are nothing but backups of a partition. Replicas are never read or write data. They are used to prevent data loss.
- Brokers
  - Brokers are simple system responsible for maintaining the pub-lished data. Each broker may have zero or more partitions per topic. Assume, if there are N partitions in a topic and N number of brokers, each broker will have one partition.
  - Assume if there are N partitions in a topic and more than N brokers (n + m), the first N broker will have one partition and the next M broker will not have any partition for that particular topic.
  - Assume if there are N partitions in a topic and less than N brokers (n-m), each broker will have one or more partition sharing among them. This scenario is not recommended due to unequal load distri-bution among the broker.

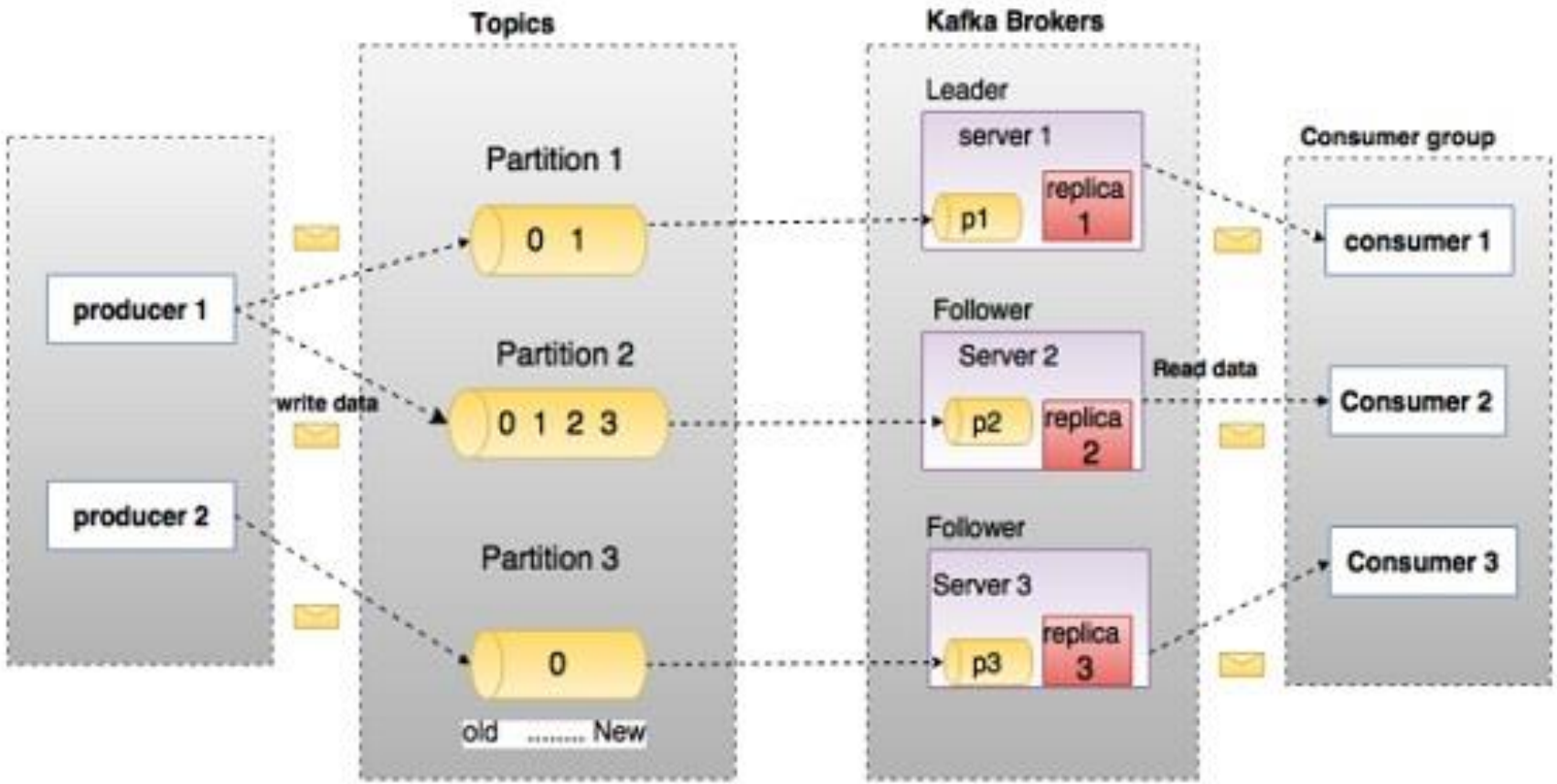
Kafka Terminology

- Kafka Cluster
  - Kafka’s having more than one broker are called as Kafka cluster. A Kafka cluster can be expanded without downtime. These clusters are used to manage the persistence and replication of message data.
- Producers
  - Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers. Every time a producer publishes a message to a broker, the broker simply appends the message to the last segment file. Actually, the message will be appended to a partition. Producer can also send messages to a partition of their choice.
- Consumers
  - Consumers read data from brokers. Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers.
- Leader
  - Leader is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader.
- Follower
  - Node which follows leader instructions are called as follower. If the leader fails, one of the follower will automatically become the new leader. A follower acts as normal consumer, pulls messages and up-dates its own data store.

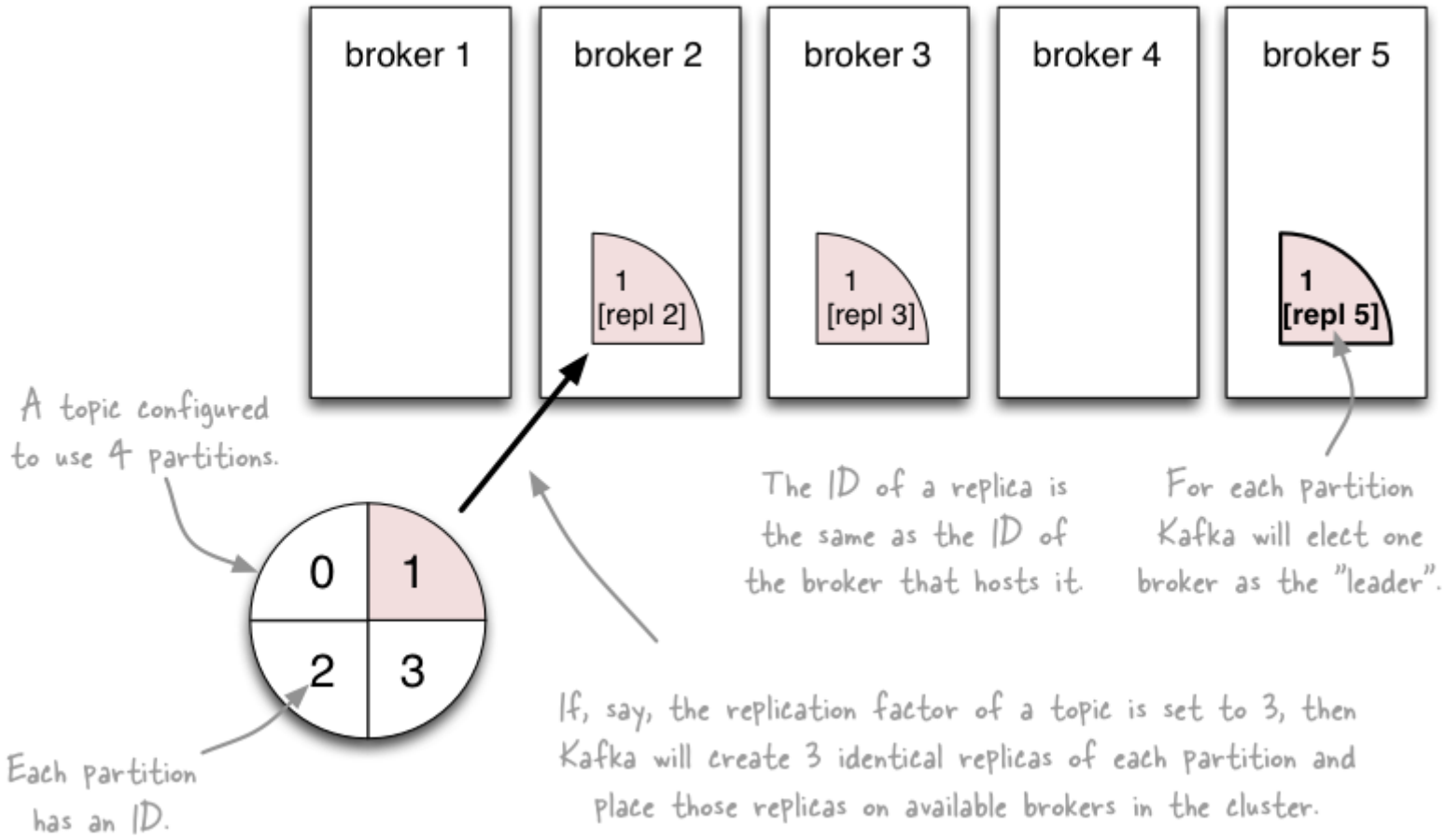
Kafka Cluster



Basic Flow

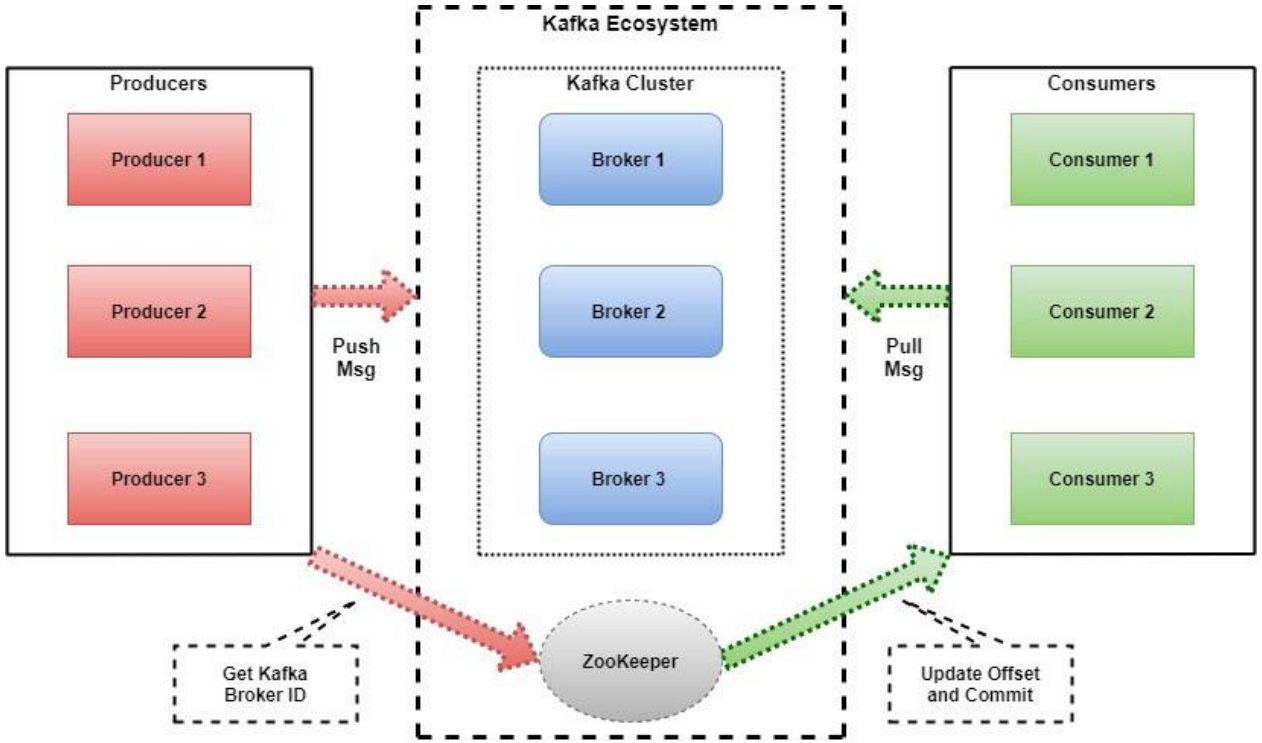

Topics
Kafka Brokers
Leader
server 1
p1
replica 1
Partition 1
0 1
Consumer group
consumer 1
producer 1
Follower
Server 2
p2
replica 2
Partition 2
write data
Read data
0 1 2 3
Consumer 2
producer 2
Follower
Server 3
p3
replica 3
Partition 3
Consumer 3
0
old ..... New

Kafka – Topics, Partitions and Replicas



Kafka - Zookeeper

- Zookeeper is used for managing and coordinating Kafka broker
  - Zookeeper service is mainly used for co-ordinating between brokers in the Kafka cluster.
  - Kafka cluster is connected to ZooKeeper to get information about any failure nodes.



Kafka CLI

- docker run --name zookeeper -p 2181:2181 zookeeper
- docker run -p 9092:9092 --name kafka -e KAFKA_ZOOKEEPER_CONNECT=192.168.99.100:2181 -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://192.168.99.100:9092 -e KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1 -e KAFKA_BROKER_ID=2 -e KAFKA_AUTO_CREATE_TOPICS_ENABLE=true confluentinc/cp-kafka
- **Verify topic**
  - docker exec kafka kafka-topics --describe --topic Kafka_Example --zookeeper 192.168.99.100:2181
- **Create a topic named: Kafka_Example**
  - docker exec kafka bash -c "kafka-topics --create --zookeeper 192.168.99.100:2181 --replication-factor 1 --partitions 1 --topic Kafka_Example"
- **Produce string message**
  - docker exec kafka bash -c 'seq 100000 | kafka-console-producer --request-required-acks 1 --broker-list 192.168.99.100:9092 --topic Kafka_Example --producer-property acks=1 && echo "Produce messages.“’
- **Produce Json message**
  - docker exec -it kafka bash
  - kafka-console-producer --broker-list 192.168.99.100:9092 --topic Kafka_Example_json
    - {"name":"demo","dept":"security"}