Kafka:

Apache Kafka is an Open source distributed event streaming platform

Creating Real Time Stream: Sending of real time data from paytm to kafka server is called creating real time stram

Processing Real Time Stream: Contineously listen to the kafka server and process them is called processing real time stream.

Distributed: Kafka is distributed system like microservices suppose that one kafka server goes down then another kafka server handle the load without any application down time.

Where does kafka come from?
Kafka developed by LinkedIn and open sourced in 2011

Why do we need kafka?
Suppose that App1 wants to send data to App2 but App2 is not available or down So App2 will lost the data. So overcome these problem kafka came into picture.

Kafka will be installed between two applications to handle the request.

if there are too many connections is required to send data from one app to n app just go for Kafka, So kafka will reduce the connections count

Kafka Architecture :

Producer: producer is the source of data who publish

the data/event

Consumer: consumer is act as receiver and it is responsible for receiving the data

Broker: Kafka Server/Broker it is just intermidiate entity that helps in message exchange
between producer and consumer.

Cluster: Group of server/broker

Topic: topic is an entity which stored the message which is published by the publisher.

Partition: Kafka topic is broken into multiple parts . This process is partioning and each
each part is called Partition

Offset: A sequence number is assigned to each message in each Partition of a kafka topic

Consumer Groups: If we have multiple consumers for acheiving better throughput so each
Consumer will read the data from each partition or each partition is assigned to single consumers

Zookeeper: it coordinates between each components of the kafka and track the status of cluster (topic,partition,offset etc.)

Install Kafka
1. Start Zookeeper
2. Start Kafka Server
3. Create a Topic

To start apache kafka Zookeeper:
bin/zookeeper-server-start.sh config/

zookeeper.properties

Confluent kafka:
bin/zookeeper-server-start etc/kafka/
zookeeper.properties

To start Broker:
bin/kafka-server-start.sh config/server.properties

Confluent kafka:
bin/kafka-server-start etc/kafka/server.properties

Zookeeper: 2181
Kafka Server/Broker: 9092

To create Topic:
bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic java-tp --partitions 3 --replication-factor 1

Confluent Kafka:
bin/kafka-topics --bootstrap-server localhost:9092 --create --topic confluent-tp --partitions 3 --replication-factor 1

To list down all topics:

bin/kafka-topics.sh --bootstrap-server localhost:9092 --list

confluent kafka: bin/kafka-topics --bootstrap-server localhost:9092 --list

To describe topic:
bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe java-tp

Confluent: bin/kafka-topics --bootstrap-server localhost:9092 --describe confluent-tp

To start producer:
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic java-tp

Confluent kafka:

bin/kafka-console-producer --broker-list localhost:9092 --topic confluent-tp

To push file to topic:
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic java-tp </Users/albelsinghbhodeliya/Downloads/Data/customers-100.csv

Load bulk data into topic confluent kafka:
bin/kafka-console-producer --broker-list localhost:9092 --topic confluent-tp </Users/albelsinghbhodeliya/Downloads/Data/customers-100.csv

To start consumer:
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic java-tp --from-beginning

confluent kafka: bin
bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic confluent-tp --from-beginning

****
****

Start Kafka without Zookeeper with Kraft mode:

KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"

bin/kafka-storage.sh format --standalone -t $KAFKA_CLUSTER_ID -c config/kraft/reconfig-server.properties

bin/kafka-server-start.sh config/kraft/reconfig-server.properties

bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic new-tp --partitions 3 --replication-factor 1

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic new-tp

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic new-tp --from-beginning

it will create all log file here: log.dirs=/tmp/kraft-combined-logs

Also we can decode log file using kafka-dump-log.sh file.

kafka server:
bin/kafka-metadata-quorum.sh --bootstrap-server localhost:9092 describe --status

Replication:
bin/kafka-metadata-quorum.sh --bootstrap-server localhost:9092 describe --replication

Kraft mode it created own metadata and it logs

everything rather than depeneding on zookeeper.
****

Benefits:

1.Eliminating system complexities
2.Data redundancy while running kafka without Zookeeper.
3.Simplified kafka Architecture without any third party service dependency

---
create docker file:

```
version: '3'

services:
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: localhost
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

run below command to install kafka using docker:

```
docker compose -f docker-compose.yml up -d
```

```
docker images
```

to check container:
docker ps

To go inside it:
docker exec -it kafka /bin/sh

go to /opt/kafka_2.13-2.8.1/bin and run below command:
Topic Creation:
kafka-topics.sh --zookeeper zookeeper:2181 --create --topic quickstart --partitions 3 --replication-factor 1

producer:
kafka-console-producer.sh --topic quickstart --bootstrap-server localhost:9092

Consumer:
kafka-console-consumer.sh --topic quickstart --bootstrap-server localhost:9092 --from-beginning

****

Producer -------->kafka -------->Consumer

Happy path:

publishing below message from publisher

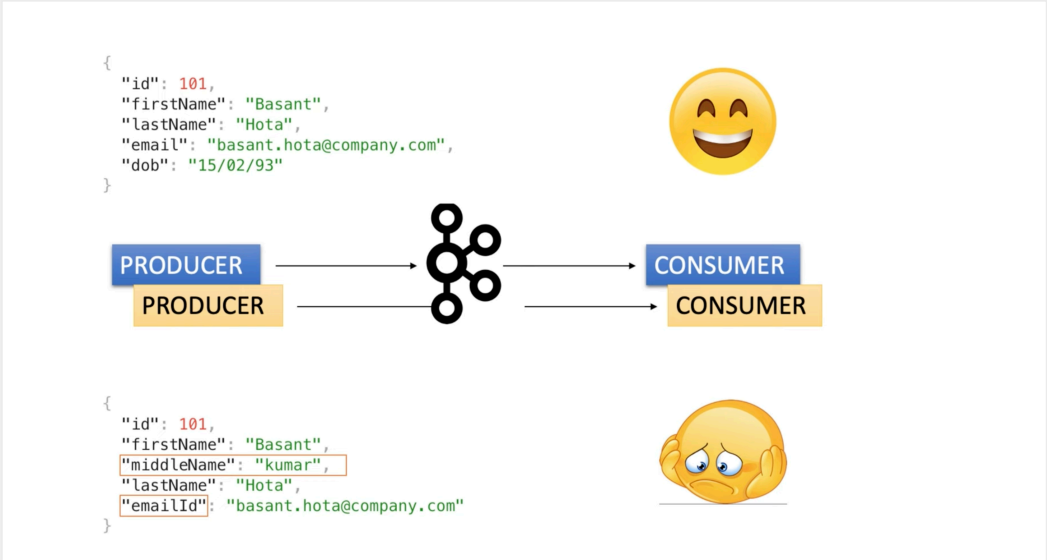

```
{
   "id": 55676,
   "fistName": "Albel",
   "email": "albelsing21@gmail.com",
   "contactNo": "8959460021"
}
```


and consuming same message from topic at the

consumer end.


```
{
  "id": 55676,
  "fistName": "Albel",
  "email": "albelsing21@gmail.com",
  "contactNo": "8959460021"
}
```


what if data got changed at the publisher end.

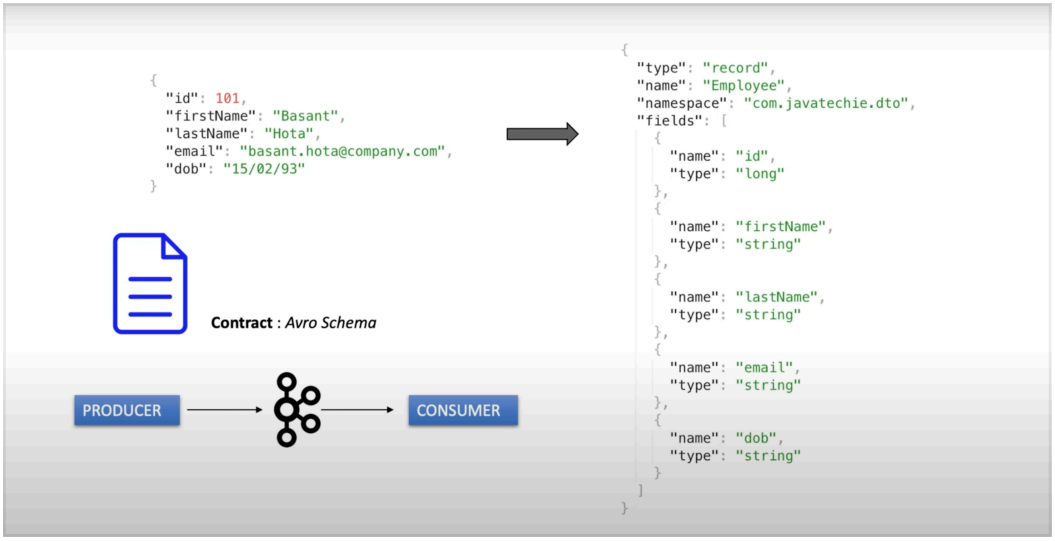



```
{
  "id": 55676,
  "fistName": "Albel",
  "lastName": "Bhodeliya",
  "email": "albelsing21@gmail.com",
  "mobile": "8959460021"
}
```

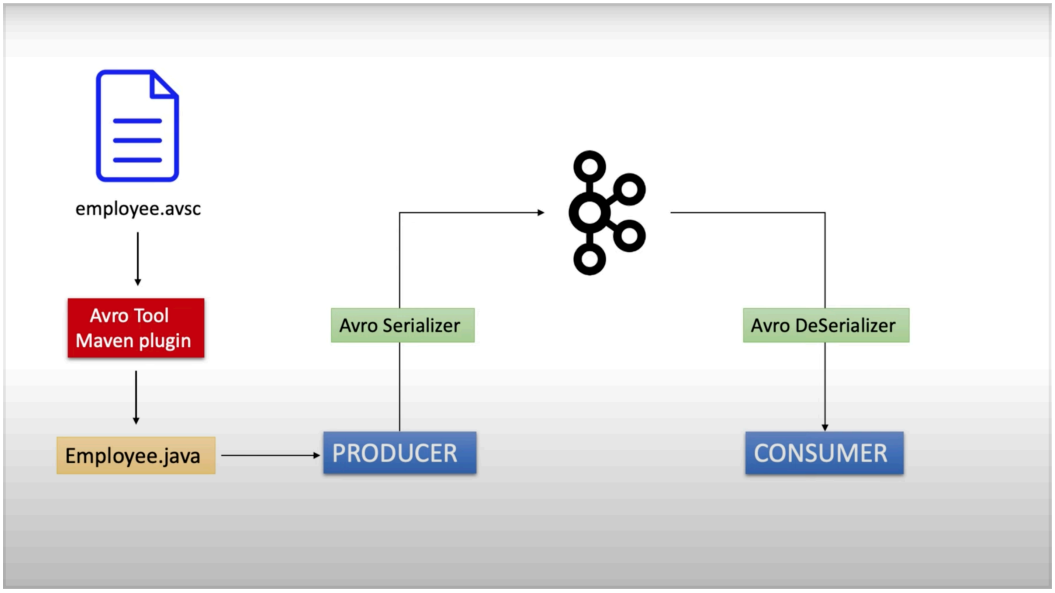

in this case consumer won't consume this message and this is big problem.
to overcome this problem write new producer and consumer with same dto without
impacting the existing flow. this is required too much work then how to solve this problem?

Confluent kafka introduce avro schema to handle data change and to store those schema it has schema

registry.



AVRO Tool or Maven plugin: To convert avrò schema to avrò object which will produce employee object.
Avro Serializer: To serialize the avro encoded message and send them back to Kafka topic
Avro Deserialiser: To deserialise the avrò encoded message and convert them back to object



Avro Schema: it is contract between producer and consumer
Schema Registry: it store the schema and when Avro serializer serialize the record so first validate and update it and store the schema in schema registry
And when Avro deserializer deserialize the record then first take the schema from registry and validate it with message and deserialize it back to object.

If we are making any changes in the schema, it will create a version and store it in the registry. Since schema registry have the flexibility backward and forward
Compatibility if any upgrade happens it will support old schema as well as new schema. That is how it handle schema evolution.