Presentation Content

Brief Intro about Messaging Technologies

Messaging is one of the most fundamental feature in every Enterprise application. There are two types of messaging:

- Synchronous messaging
- Asynchronous messaging

In async messaging sender doesn't block or wait for response. There can be a response or not, but the sender will carry out his remaining tasks.

There are following types of asynchronous messaging:

- JMS (Java Messaging Service)
- Advanced Message Queueing Protocol (AMQP) e.g Apache ActiveMQ, RabbitMQ
- Message Queueing Telemetry Transport (MQTT)

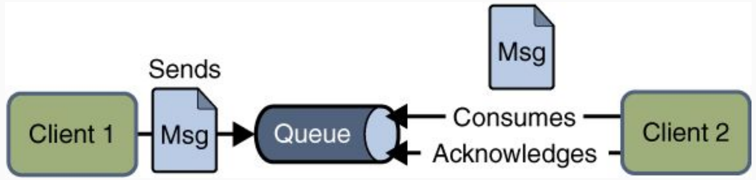
Java Messaging Service (JMS)

JMS is a standard API for sending and receiving messages. It allows components to create, send, receive, and read messages.

JMS pros: 1) Fault tolerant 2) Asynchronous messaging 3) Loose coupling

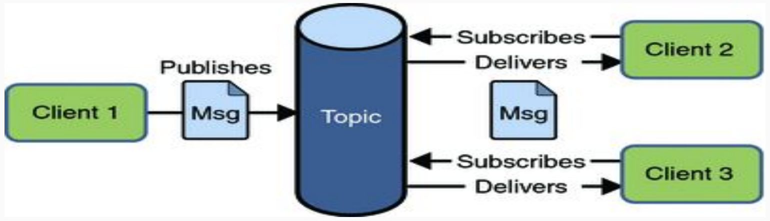
JMS provides two types of message domain

- Point to Point Messaging
- Pub/Sub messaging



JMS Provider:

A JMS provider is a messaging system that implements the JMS interfaces and provides administrative and control features. Examples are ActiveMQ, RabbitMQ,



Apache ActiveMQ VS RabbitMQ

ActiveMQ is one of the JMS provider that provides both PTP and pub/sub messaging.

- Pros: 1) Schedule delayed deliveries. 2) Database row-level locking techniques, file system and other modes are used for high-availability. 3) It supports Heterogeneous applications.
- Cons: 1) prefer to use when small amounts of data is involved. 2) uses the imperative programming paradigm, which can be substantially slower. 3) can handle routing but throughput in not too high.

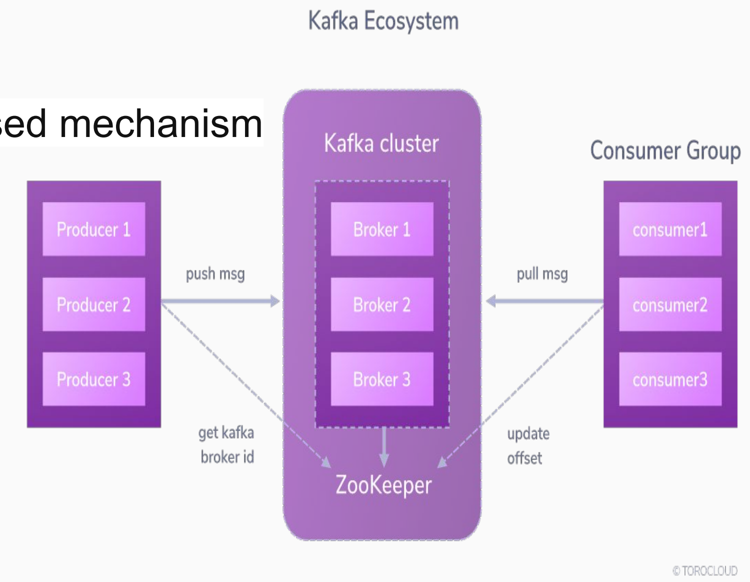
RabbitMQ is another widely used open source message broker, Written in Erlang. Several messaging techniques are supported, including pub-sub, point-to-point, and request-reply messaging.

- Pros: 1) You can write your own and feed it as a plugin. 2) High availability is ensured by making multiple server into a single cluster and replication queues across multiple nodes. 3) it supports complex routing by using exchange.
- Cons: 1) overhead of mirroring and the latency from the central node significantly decreases the throughput of RabbitMQ. 2) if message is consumed it will immediately removed from the queue.

Apache Kafka

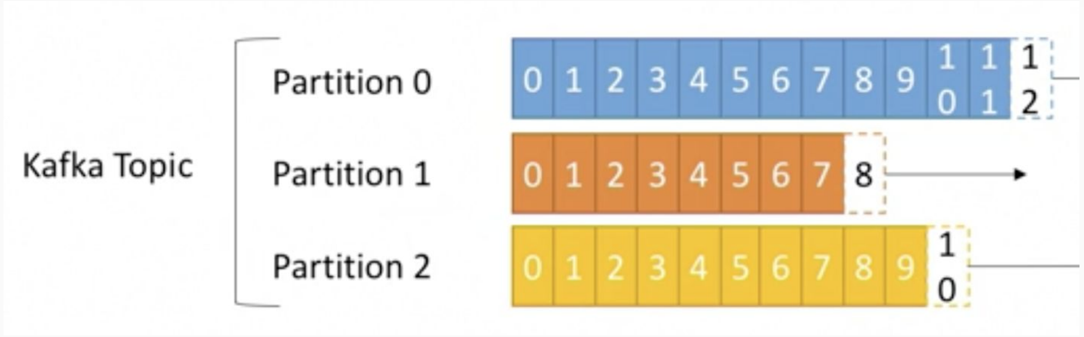
Kafka is a distributed publish-subscribe messaging system that was created as a fast, scalable, and durable alternative to existing solutions.

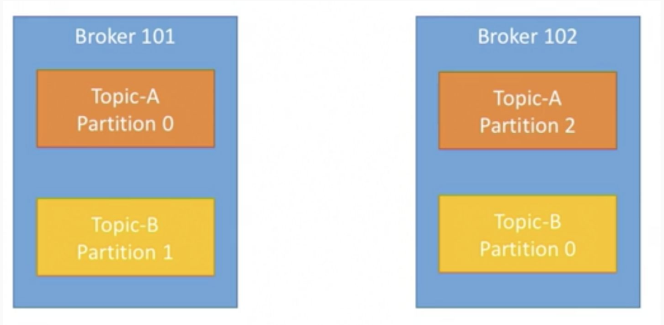
- Works on pub/sub method or we can say pull based mechanism
- Highly scalable, durable and reliable
- Highly distributed and can be scaled horizontally
- Provides high throughput



Kafka - Key Components

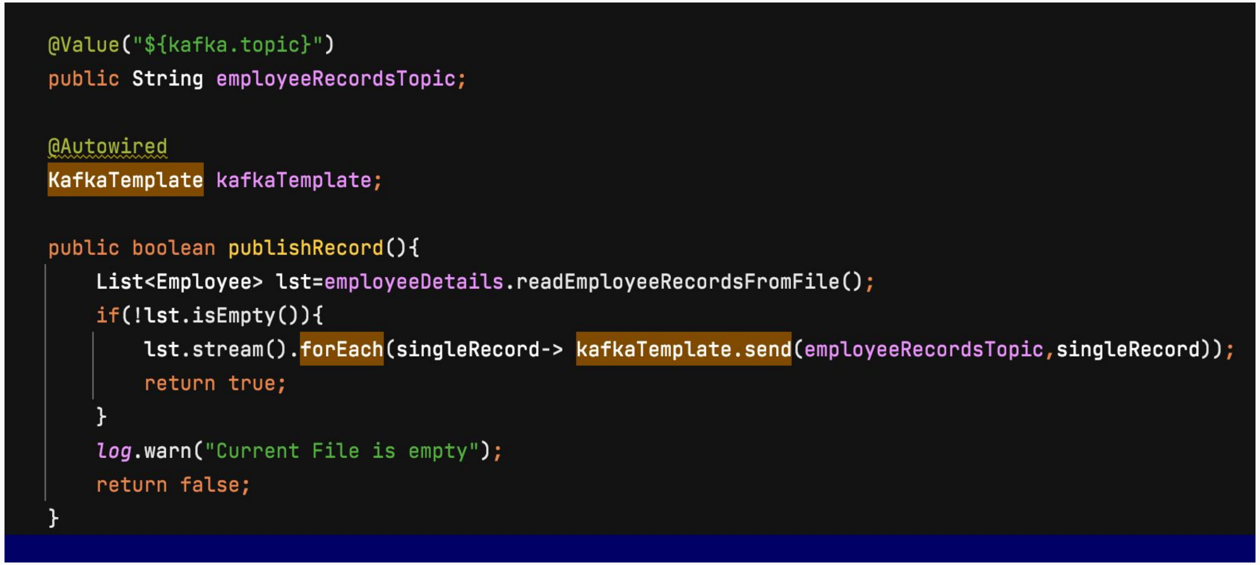
- Topics - by name
- Topic vs Queue
- Partitions
- Partition Offset
- Immutable Data
- Brokers (Servers)
- Bootstrap Server





Kafka - Producer

- Auto Connect
- Round-Robin
- Message Keys
- Acknowledgments



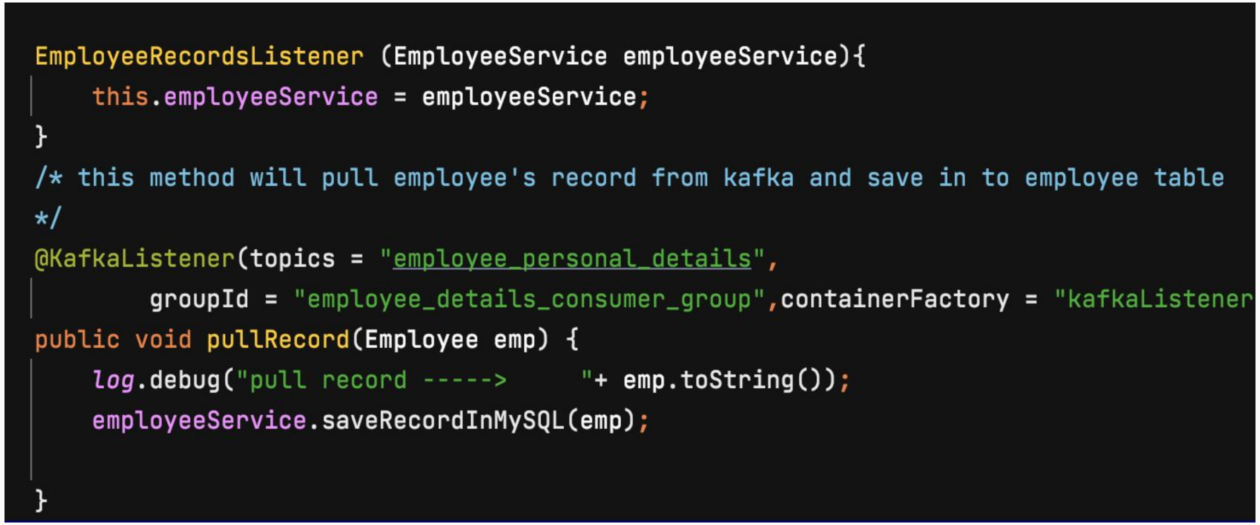
```
@Value("${kafka.topic}")
public String employeeRecordsTopic;

@Autowired
KafkaTemplate kafkaTemplate;

public boolean publishRecord(){
    List<Employee> lst=employeeDetails.readEmployeeRecordsFromFile();
    if(!lst.isEmpty()){
        lst.stream().forEach(singleRecord-> kafkaTemplate.send(employeeRecordsTopic,singleRecord));
        return true;
    }
    log.warn("Current File is empty");
    return false;
}
```

Kafka - Consumer

- Auto Connect
- Reading In-Order
- Consumer Offset
- At Most, At Least

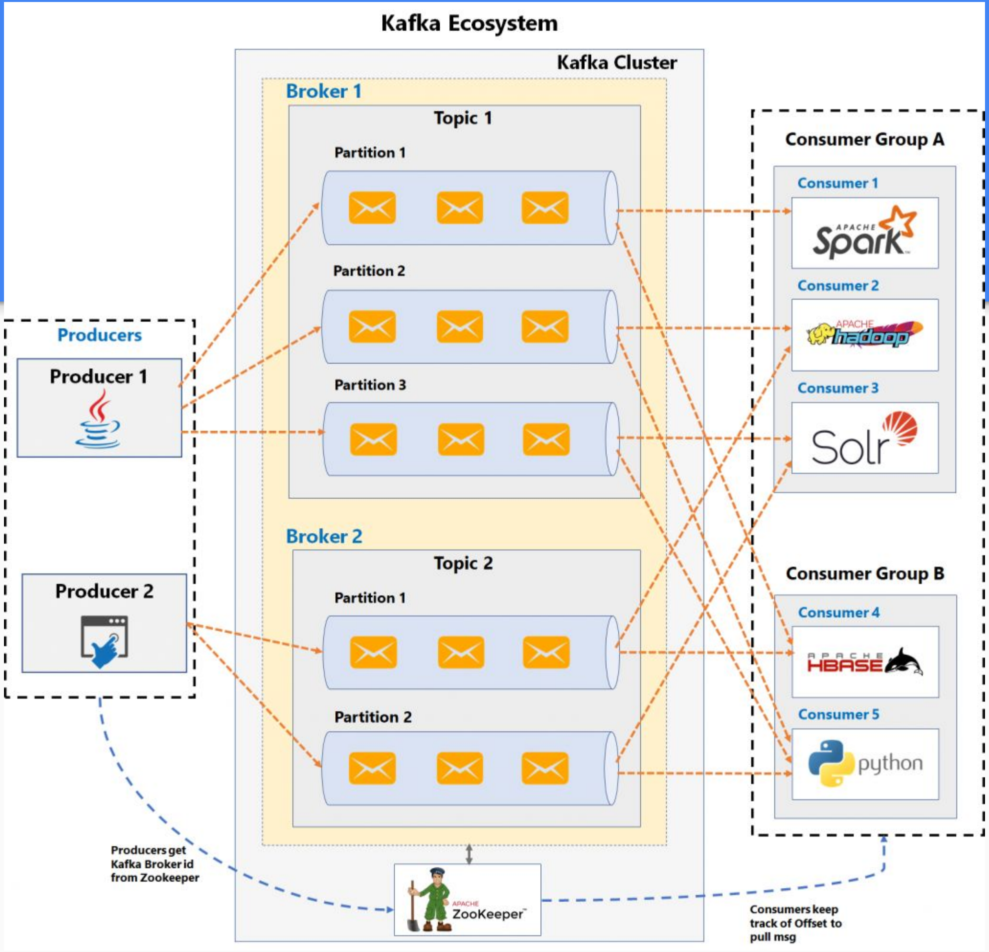


```
EmployeeRecordsListener (EmployeeService employeeService){
    this.employeeService = employeeService;
}
/* this method will pull employee's record from kafka and save in to employee table
*/
@KafkaListener(topics = "employee_personal_details",
        groupId = "employee_details_consumer_group",containerFactory = "kafkaListener
public void pullRecord(Employee emp) {
    log.debug("pull record ----->      "+ emp.toString());
    employeeService.saveRecordInMySQL(emp);

}
```

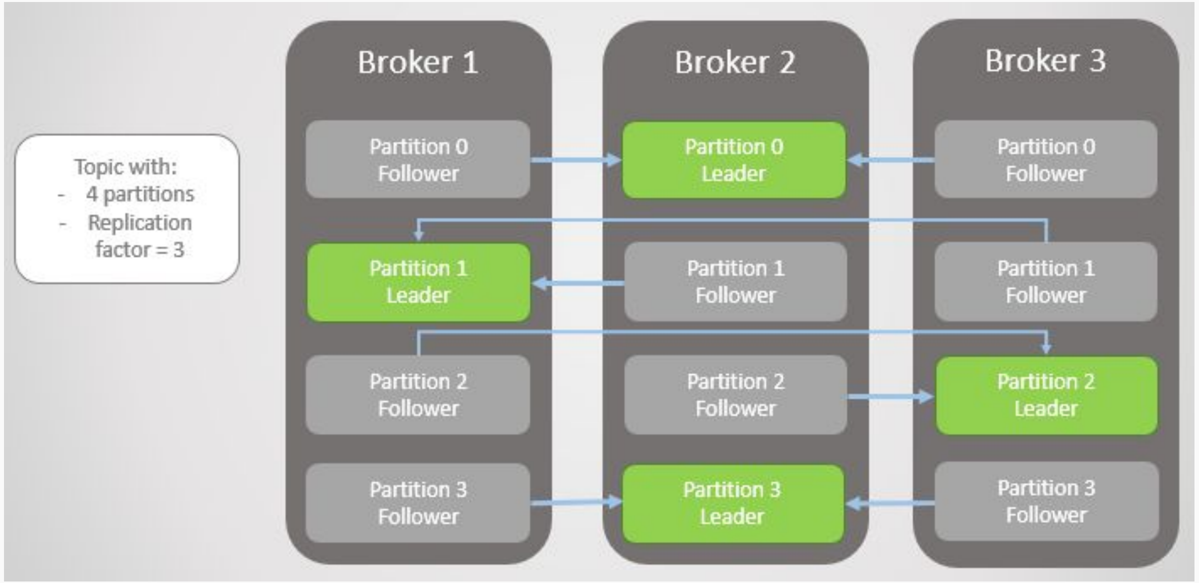
Kafka - Workflow

- Zookeeper
- Broker List
- Notify Kafka
- Leader (Writes)
- Follower (Reads)



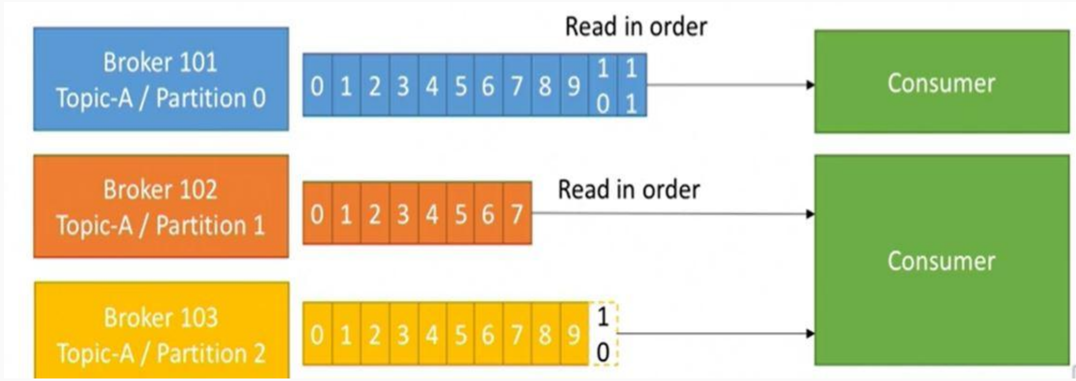
Replication Factor

- Replication factor > 1
- Leader of partition
- ISR (in-sync replica)
- acks = 0 | 1 | all



Consumers

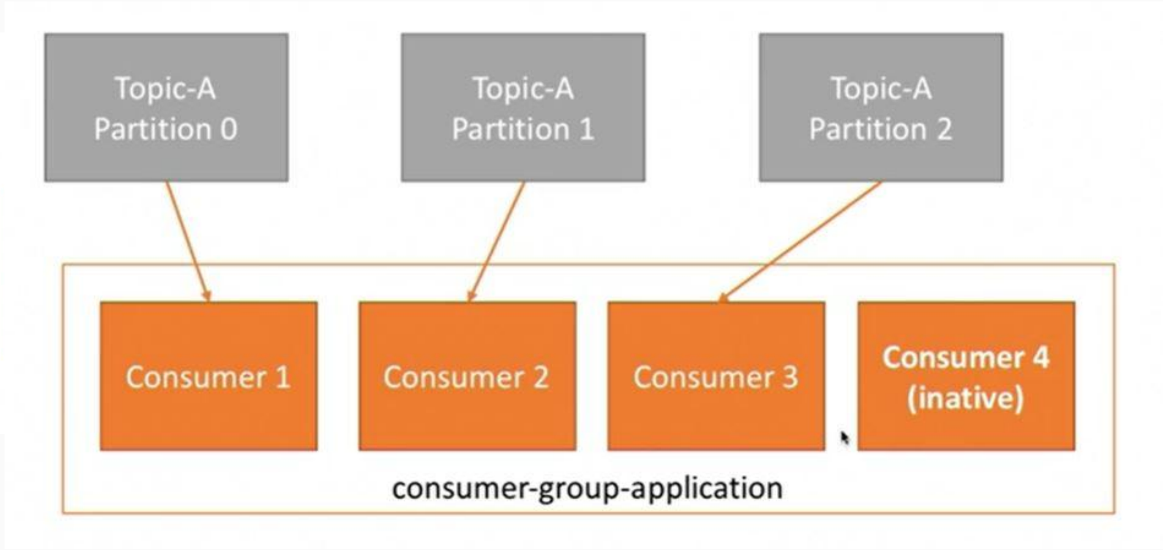
- Read data from a topic
- Incase of broker failure,
  Consumer knows how to recover
- Read in order within partition



Consumer Groups

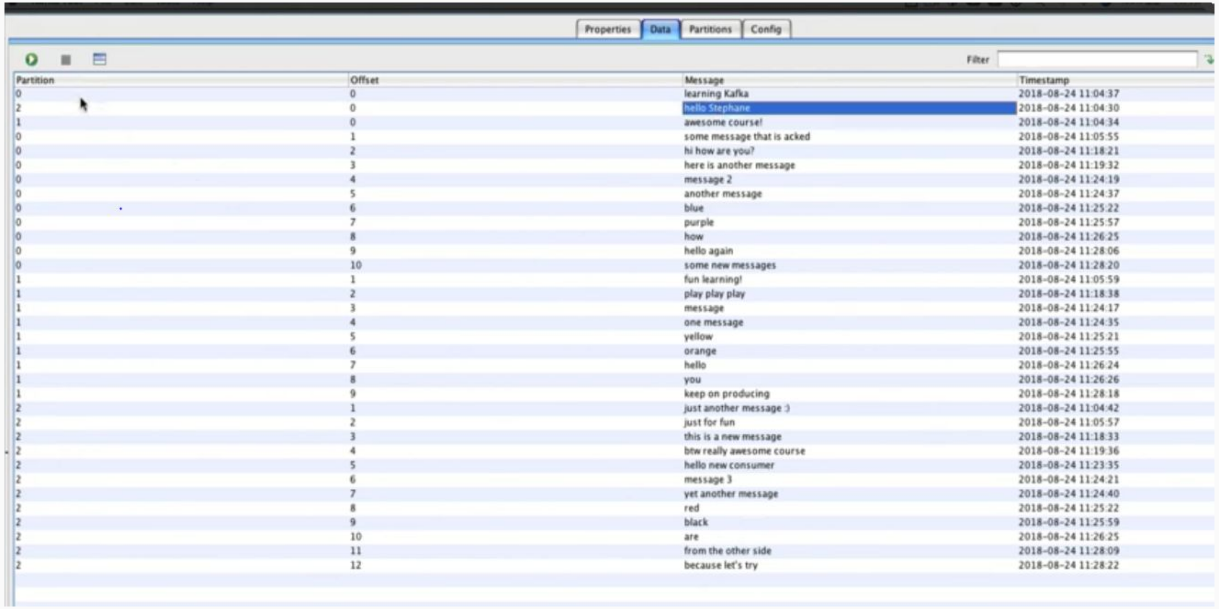
- Read data in groups
- Each consumer reads from exclusive partitions
- Partitions are auto-assigned

- If Consumers > Partitions?



Kafka-GUI

- Kafka Tool
- Details of all the topics, consumers etc.
- Helps in debugging



Partition	Offset	Message	Timestamp
0	0	learning Kafka	2018-08-24 11:04:37
2	0	hello Stephane	2018-08-24 11:04:30
1	0	awesome course!	2018-08-24 11:04:34
0	1	some message that is acked	2018-08-24 11:05:55
0	2	hi how are you?	2018-08-24 11:18:21
0	3	here is another message	2018-08-24 11:19:32
0	4	message 2	2018-08-24 11:24:19
0	5	another message	2018-08-24 11:24:37
0	6	blue	2018-08-24 11:25:22
0	7	purple	2018-08-24 11:25:57
0	8	how	2018-08-24 11:26:25
0	9	hello again	2018-08-24 11:28:06
0	10	some new messages	2018-08-24 11:28:20
1	1	fun learning!	2018-08-24 11:05:59
1	2	play play play	2018-08-24 11:18:38
1	3	message	2018-08-24 11:24:17
1	4	one message	2018-08-24 11:24:35
1	5	yellow	2018-08-24 11:25:21
1	6	orange	2018-08-24 11:25:55
1	7	hello	2018-08-24 11:26:24
1	8	you	2018-08-24 11:26:26
1	9	keep on producing	2018-08-24 11:28:18
2	1	just another message :)	2018-08-24 11:04:42
2	2	just for fun	2018-08-24 11:05:57
2	3	this is a new message	2018-08-24 11:18:33
2	4	btw really awesome course	2018-08-24 11:19:36
2	5	hello new consumer	2018-08-24 11:23:35
2	6	message 3	2018-08-24 11:24:21
2	7	yet another message	2018-08-24 11:24:40
2	8	red	2018-08-24 11:25:22
2	9	black	2018-08-24 11:25:59
2	10	are	2018-08-24 11:26:25
2	11	from the other side	2018-08-24 11:28:09
2	12	because let's try	2018-08-24 11:28:22

Problems with Zookeeper

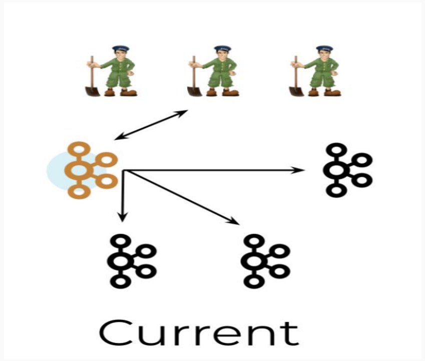
Managing metadata externally is not a good idea that is a main problem in having Zookeeper

Managing two systems doubles the system complexity in terms of network communication, security, monitoring and configuration.This limits the kafka scalability.

Often system have same number of zookeeper nodes as kafka nodes.

For more details visit the following site:

https://www.confluent.io/blog/removing-zookeeper-dependency-in-kafka/



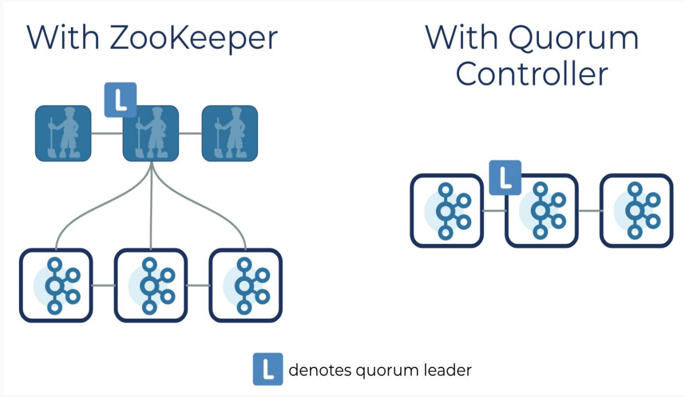
GoodBye Zookeeper...!

From kafka v2.8.0 the zookeeper has been replaced with Self-Managed Metadata Quorum. For further details you can see KIPS 500.

Kraft is the consensus protocol that was introduced to remove Apache Kafka's dependency on ZooKeeper.

Benefits of Kafka's new quorum controller:

- Enables Kafka clusters to scale to millions of partitions.
- Allows Kafka to have a single security model for the whole system
- Makes controller failover near-instantaneous



Why and When Kafka?

So kafka is not the ultimate messaging choice for every distributed systems.

When to use:

- You need fast real time or big data processing.
- High availability and throughput is required.
- Best suited for event driven applications.

When not use:

- Complex routing required for messaging.
- Need request/reply messaging capabilities.
- Delay and Schedule Message Delivery
- Need different transport protocols except TCP like STOMP, Openwire
- Your system don’t have real time or big data to process.

Reference:

- https://cwiki.apache.org/confluence/display/KAFKA/KIP-500%3A+Replace+ZooKeeper+with+a+Self-Managed+Metadata+Quorum
- https://kafka.apache.org/downloads
- https://issues.apache.org/jira/browse/KAFKA-9119
- https://www.confluent.io/blog/removing-zookeeper-dependency-in-kafka
- https://www.linkedin.com/learning/learn-apache-kafka-for-beginners/intro-to-apache-kafka?autoAdvance=true&autoSkip=false&autoplay=true&resume=true&u=2111049