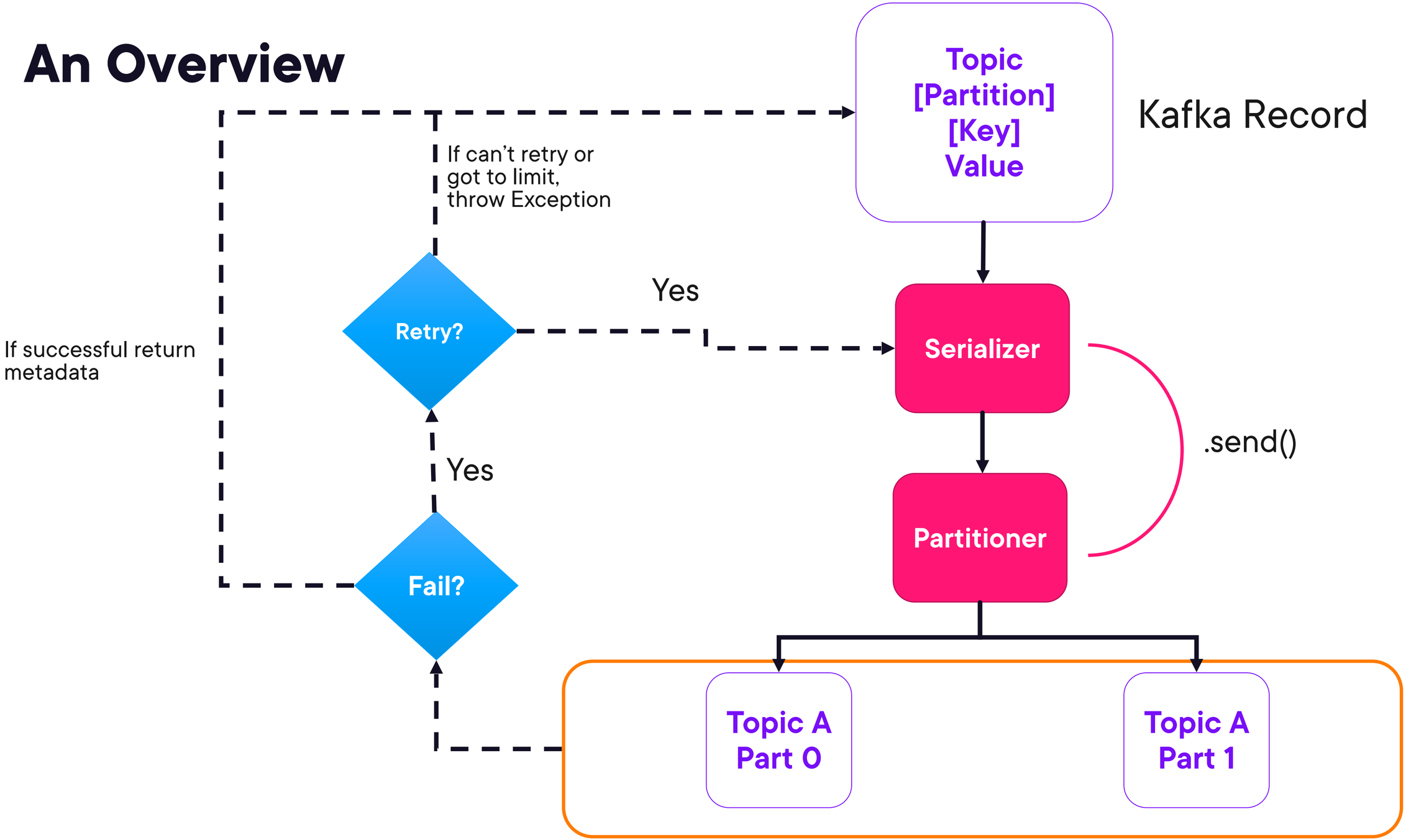
Kafka Producers

Axel Sirota

AI and Cloud Consultant

@AxelSirota

An Overview


Topic
[Partition]
[Key]
Value
Kafka Record
If can't retry or
got to limit,
throw Exception
Retry?
Yes
If successful return
metadata
Serializer
.send()
Yes
Partitioner
Fail?
Topic A
Part 0
Topic A
Part 1

Producing Messages with Kafka CLI

Serializers and Producer Configuration

Construct a java.util Properties object

```
Properties properties = new Properties();

properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
“localhost:9092”);

properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);

properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
IntegerSerializer.class);
```

Provide two or more locations where the Bootstrap servers are located

```
Properties properties = new Properties();

properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
“localhost:9092”);

properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);

properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
IntegerSerializer.class);
```

Provide a Serializer for the key

```
Properties properties = new Properties();

properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
“localhost:9092”);

properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);

properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
IntegerSerializer.class);
```

Provide a Serializer for the value

```
Properties properties = new Properties();

properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
“localhost:9092”);

properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);

properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
IntegerSerializer.class);
```

Producer Object

```
KafkaProducer producer = new KafkaProducer<>(properties);
```

Create a Record

```
ProducerRecord producerRecord = new
ProducerRecord<>(“my_orders”, state, amount);
```

Sending a Message

```
Future send = producer.send(producerRecord);
```

Record Metadata

Contains information about your send including the messages

```
if (metadata.hasOffset()) {
     System.out.format(“offset: %d\n”,
     metadata.offset()),
}
     System.out.format(“partition: %d\n”,
     metadata.partition());

     System.out.format(“timestamp: %d\n”,
     metadata.timestamp());

     System.out.format(“topic: %s\n”,
metadata.topic());

     System.out.format(“toString: %s\n”,
metadata.toString());
```

Capturing a Callback

```
producer.send(producerRecord, new Callback() {

@Override

public void onCompletion(RecordMetadata metadata,
                         Exception e){
…

    }

}
```

Using Lambdas

```
producer.send(producerRecord,(metadata, e) -> {
    if(metadata != null) {

    System.out.printIn(producerRecord.key());
    System.out.printIn(producerRecord.value());

    }
}
```

Be a Good Citizen

```
producer.flush();
producer.close();
```

Processing Messages with Java

Key, Takeaways, and Tips

Takeaways

The producer protocol implies the existence of a Partitioner that redirects messages to the correct partition

The partitions information is caught on the initial instantiation of the Producer Object

The response from the producer.send() method is a Future, but you can capture it in a Callback or Lambda

There are Retryable and non-retryable Exceptions and based on that the Protocol will automatically retry

Keys

Be sure how to configure your own Partitioner as homework

Try to ensure you can write a simple loop of sending messages by yourself

Try playing around what happens with the Producer if a broker is dead

Up Next:

Consumers