Architecture of Enterprise Applications 2
Messaging - kafka

Haopeng Chen

RE*liable,* ***IN****telligent and* ***S****calable Systems Group (****REINS****)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

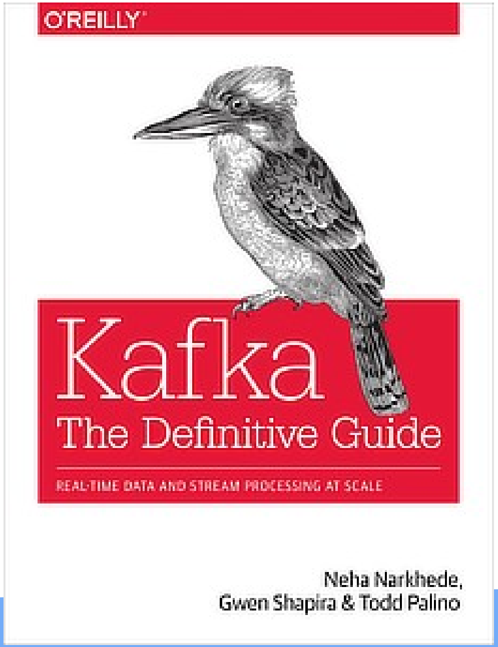
Contents and Objective

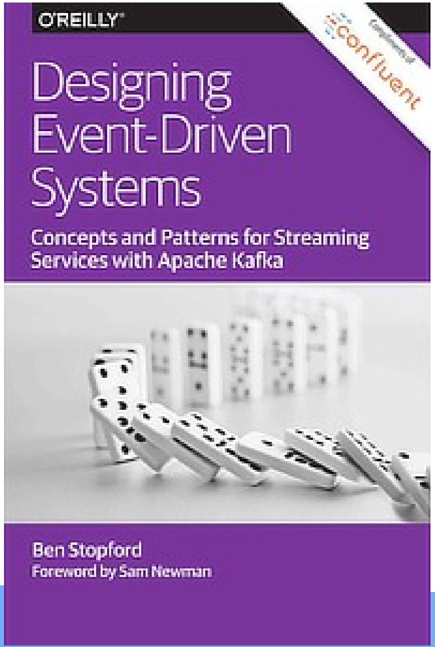
- Contents
  - Kafka
    - Introduction
    - Quick start
    - Communication with Kafka

- Objective
  - 能够根据系统需求，使用 Kafka 消息中间件来实现异步通信，包括跨编程语言的通信

Apache Kafka Resource

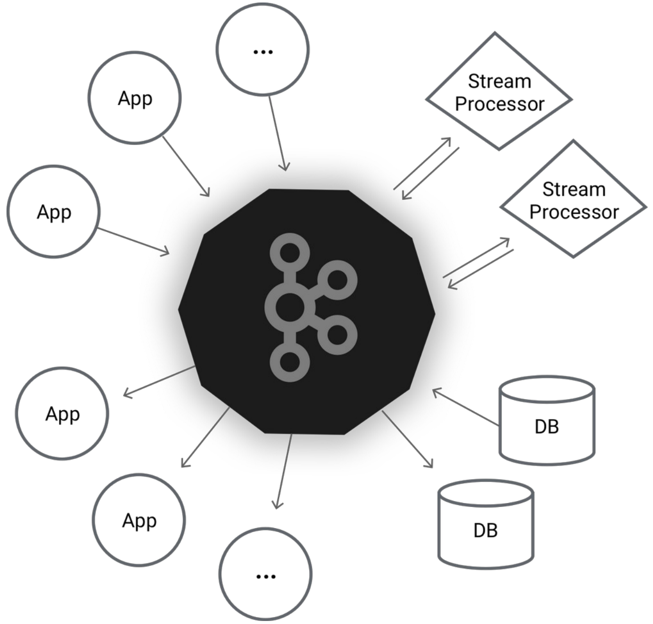
- *from*： https://kafka.apache.org/books-and-papers
- Apache Kafka Quick Start
  - https://kafka.apache.org/quickstart


O'REILLY
Kafka
The Definitive Guide
REAL-TIME DATA AND STREAM PROCESSING AT SCALE
Neha Narkhede, Gwen Shapira & Todd Palino

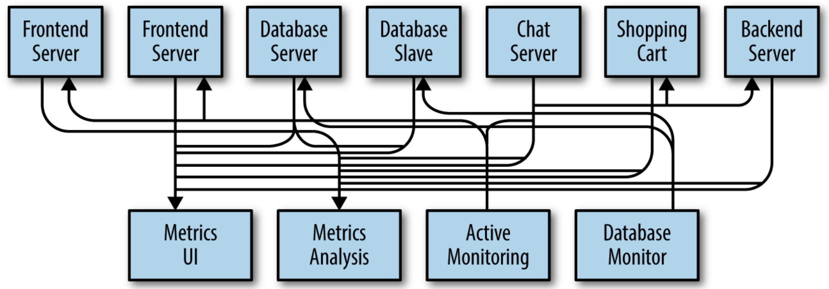

O'REILLY
confluent
Designing Event-Driven Systems
Concepts and Patterns for Streaming Services with Apache Kafka
Ben Stopford
Foreword by Sam Newman

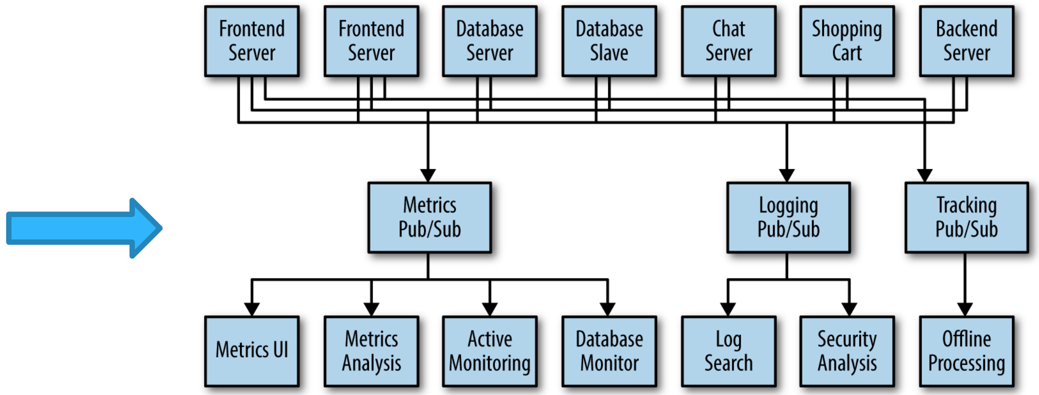
Apache Kafka

- http://kafka.apache.org
- Apache Kafka
  - is an open-source distributed event streaming.
  - To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.
  - To **store** streams of events durably and reliably for as long as you want.
  - To **process** streams of events as they occur or retrospectively.



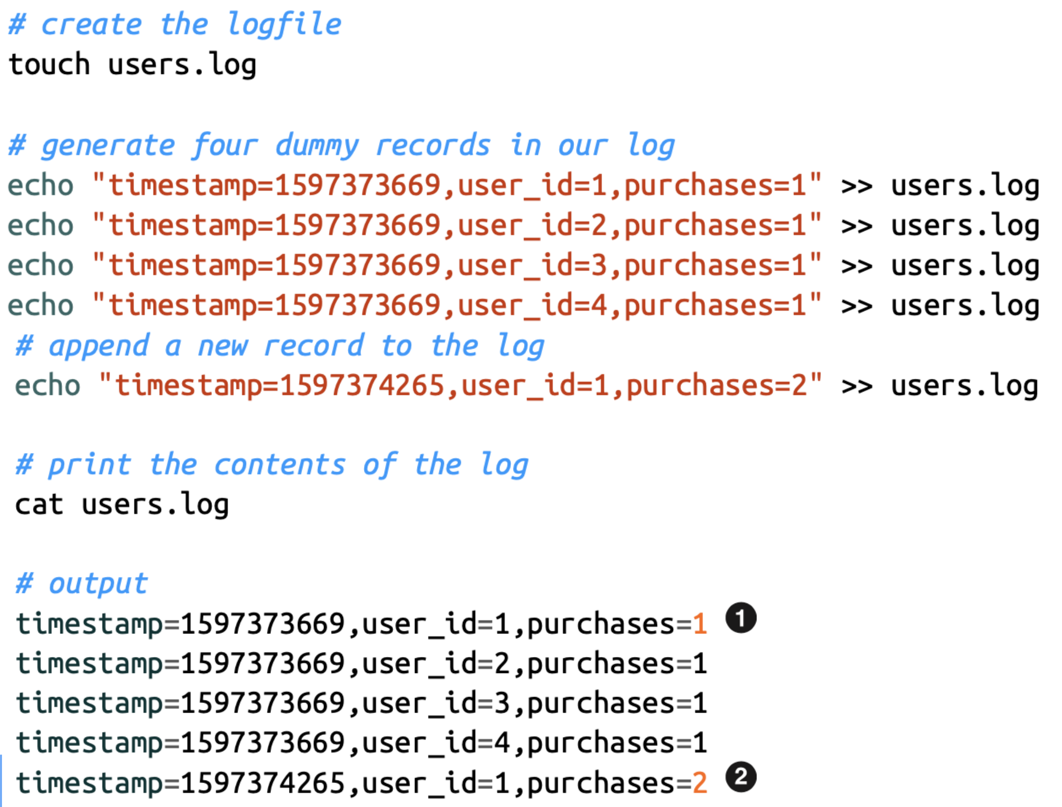
Publish/Subscribe Messaging





How Are Streams Stored?

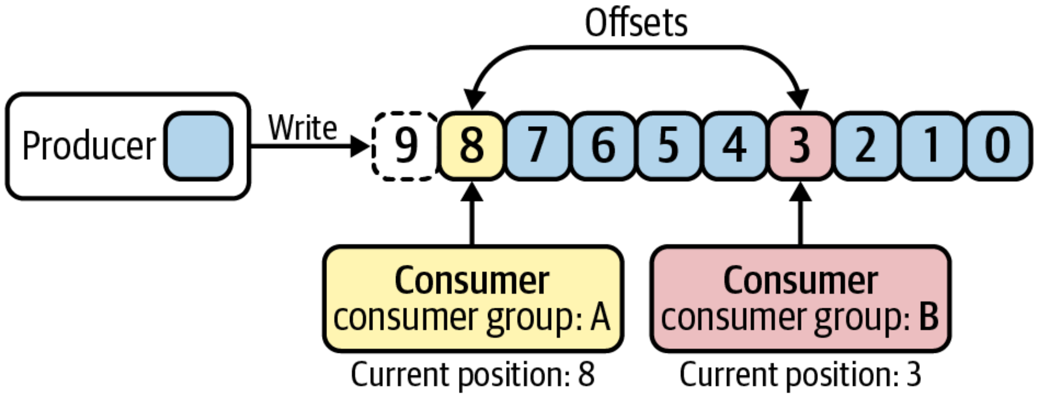
- Logs are *append-only* data structures that capture an *ordered sequence* of events.



```
# create the logfile
touch users.log

# generate four dummy records in our log
echo "timestamp=1597373669,user_id=1,purchases=1" >> users.log
echo "timestamp=1597373669,user_id=2,purchases=1" >> users.log
echo "timestamp=1597373669,user_id=3,purchases=1" >> users.log
echo "timestamp=1597373669,user_id=4,purchases=1" >> users.log
 # append a new record to the log
 echo "timestamp=1597374265,user_id=1,purchases=2" >> users.log

 # print the contents of the log
 cat users.log

 # output
 timestamp=1597373669,user_id=1,purchases=1 ❶
 timestamp=1597373669,user_id=2,purchases=1
 timestamp=1597373669,user_id=3,purchases=1
 timestamp=1597373669,user_id=4,purchases=1
 timestamp=1597374265,user_id=1,purchases=2 ❷
```

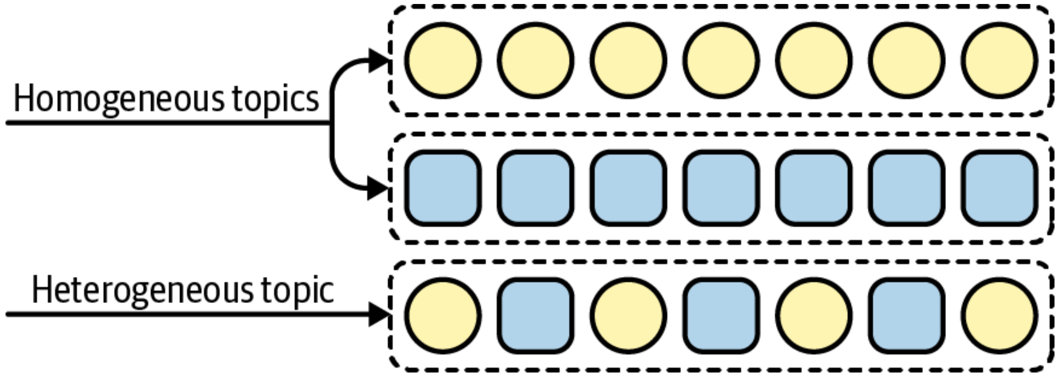
Offset

- Kafka refers to the position of each entry in its distributed log as an *offset*.
  - multiple consumer groups can each read from the same log, and maintain their own positions in the log/stream they are reading from.



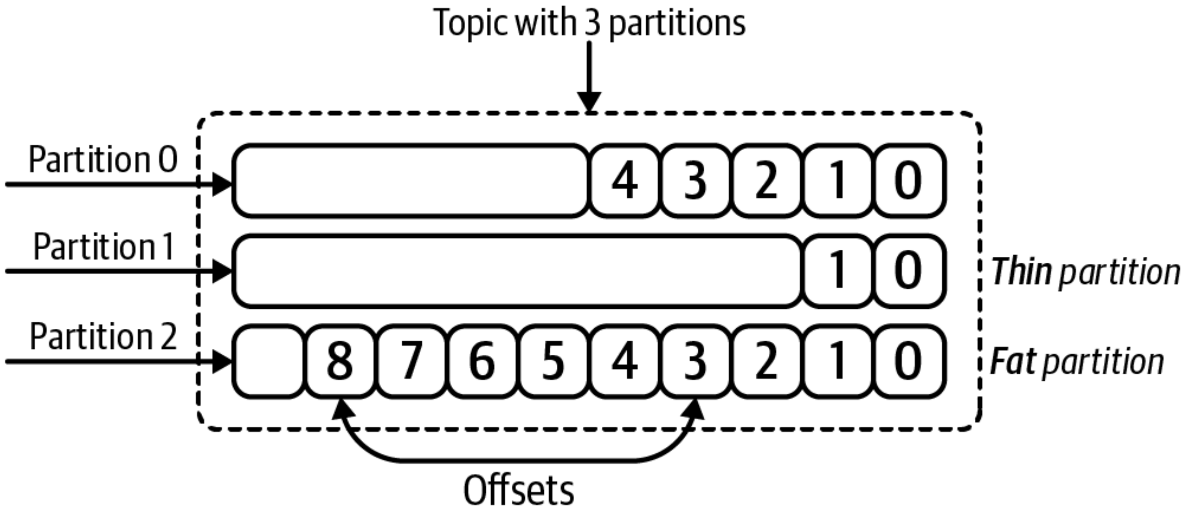
Topics and Partitions

- Topics
  - *homogeneous topics* that contain only one type of data, or *heterogeneous topics* that contain multiple types of data



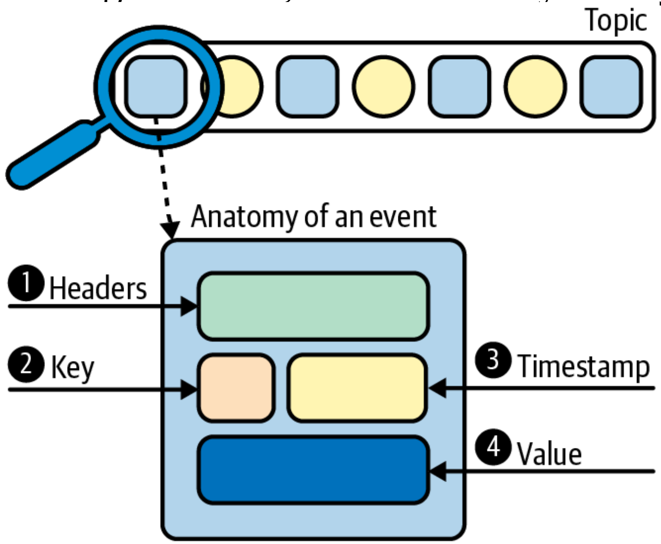
Topics and Partitions

- Kafka topics are broken into smaller units called *partitions*.
  - Partitions are individual logs where data is produced and consumed from
  - The number of partitions for a given topic is configurable, and having more partitions in a topic generally translates to more parallelism and throughput



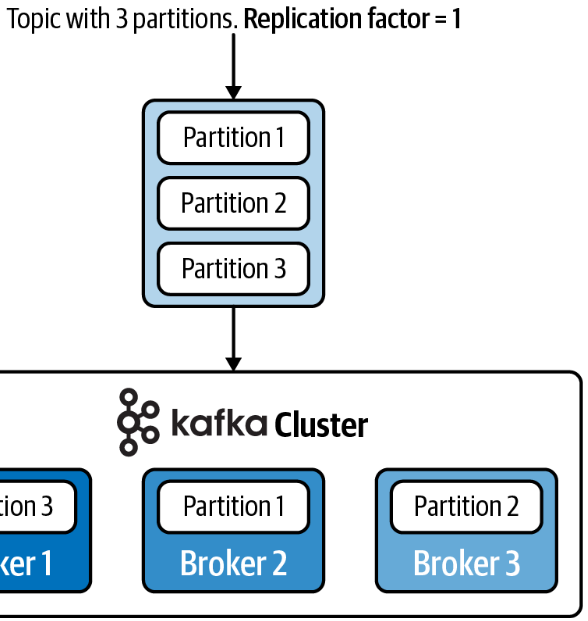
Events

- An event is a timestamped key-value pair that records *something that happened*.
  - Application-level headers contain optional metadata about an event.
  - Keys are also optional, but play an important role in how data is distributed across partitions.
  - Each event is associated with a timestamp.
  - The value contains the actual message contents, encoded as a byte array.



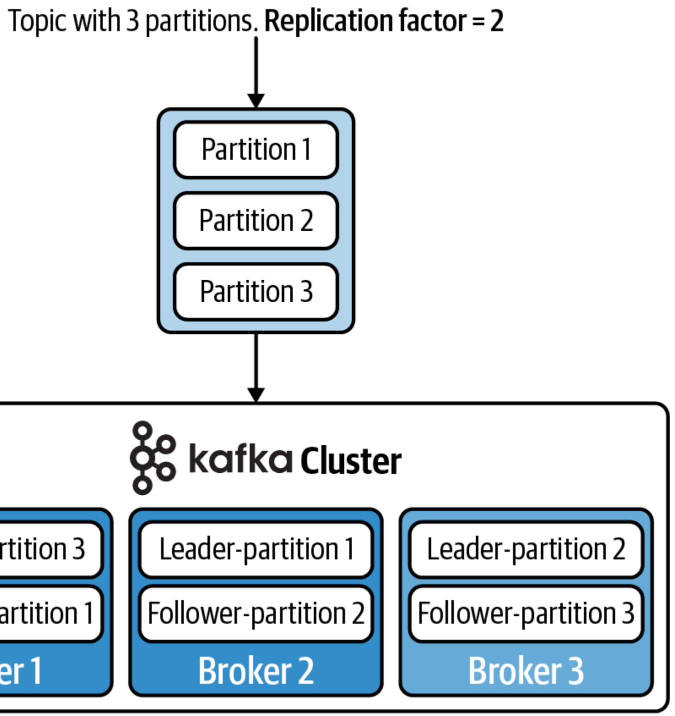
Kafka Cluster and Broker

- Kafka operates as a cluster, and multiple machines, called *brokers*, are involved in the storage and retrieval of data.
  - Kafka clusters can be quite large, and can even span multiple data centers and geo- graphic regions.



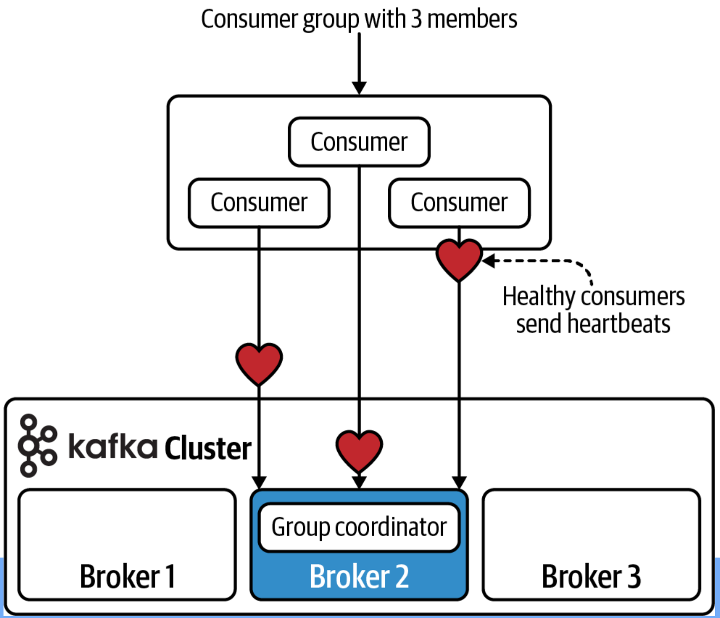
Kafka Cluster and Broker

- To achieve fault tolerance and high availability, you can set a replication factor when configuring the topic.



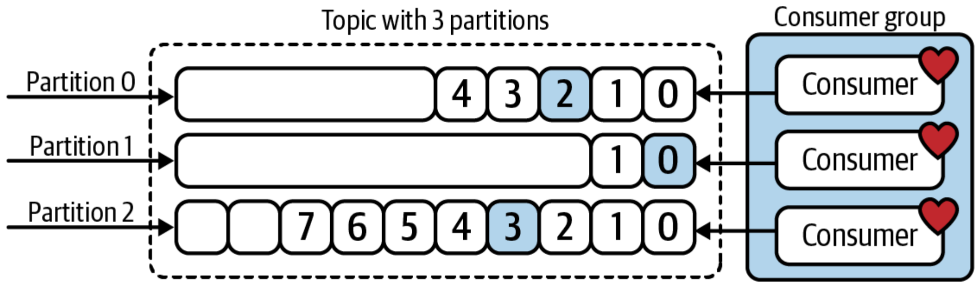
Customer Groups

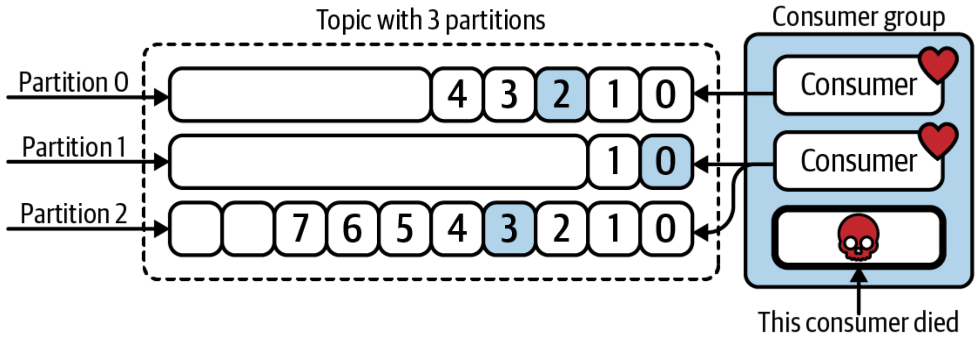
- Consumer groups are made up of multiple cooperating consumers, and the member- ship of these groups can change over time.
  - every consumer group is assigned to a special broker called the *group coordinator*, which is responsible for receiving heartbeats from the consumers, and triggering a *rebalance* of work whenever a consumer is marked as dead.



Customer Groups

- Every active member of the consumer group is eligible to receive a partition assignment.



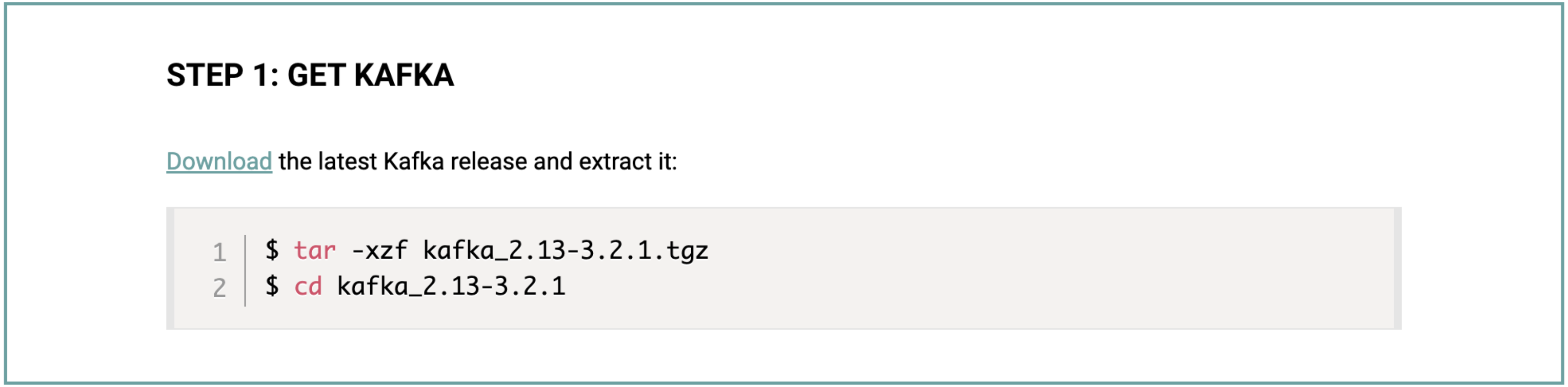


Apache Kafka Core APIs

- Kafka has four core APIs:
  - The Producer API allows an application to publish a stream of records to one or more Kafka topics.
  - The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
  - The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
  - The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

Kafka - Quickstart

- Quickstart
  - https://kafka.apache.org/quickstart

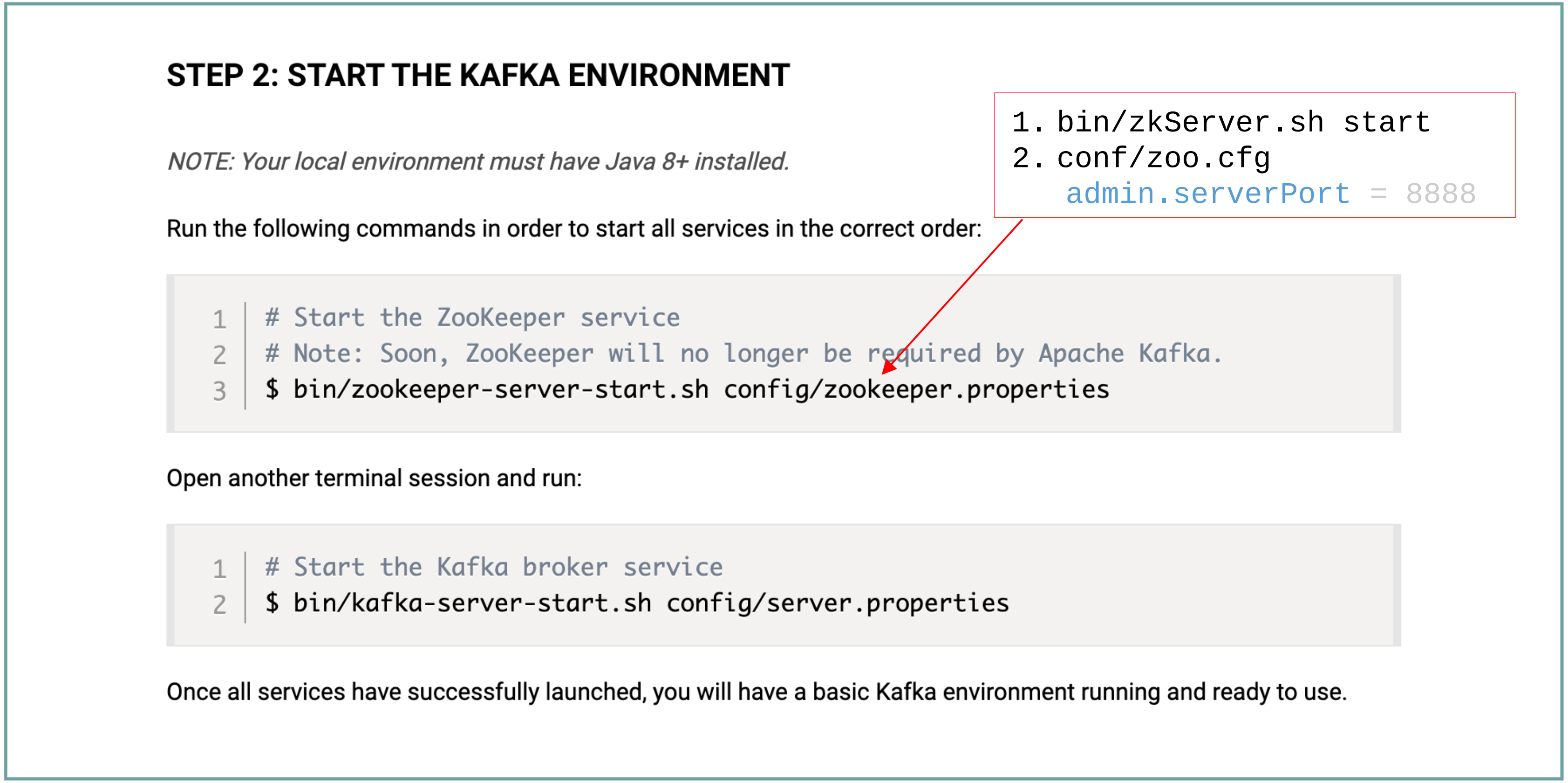


STEP 1: GET KAFKA

Download the latest Kafka release and extract it:

```
$ tar -xzf kafka_2.13-3.2.1.tgz
$ cd kafka_2.13-3.2.1
```

Kafka - Quickstart



STEP 2: START THE KAFKA ENVIRONMENT

NOTE: Your local environment must have Java 8+ installed.

Run the following commands in order to start all services in the correct order:

```
# Start the ZooKeeper service
# Note: Soon, ZooKeeper will no longer be required by Apache Kafka.
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

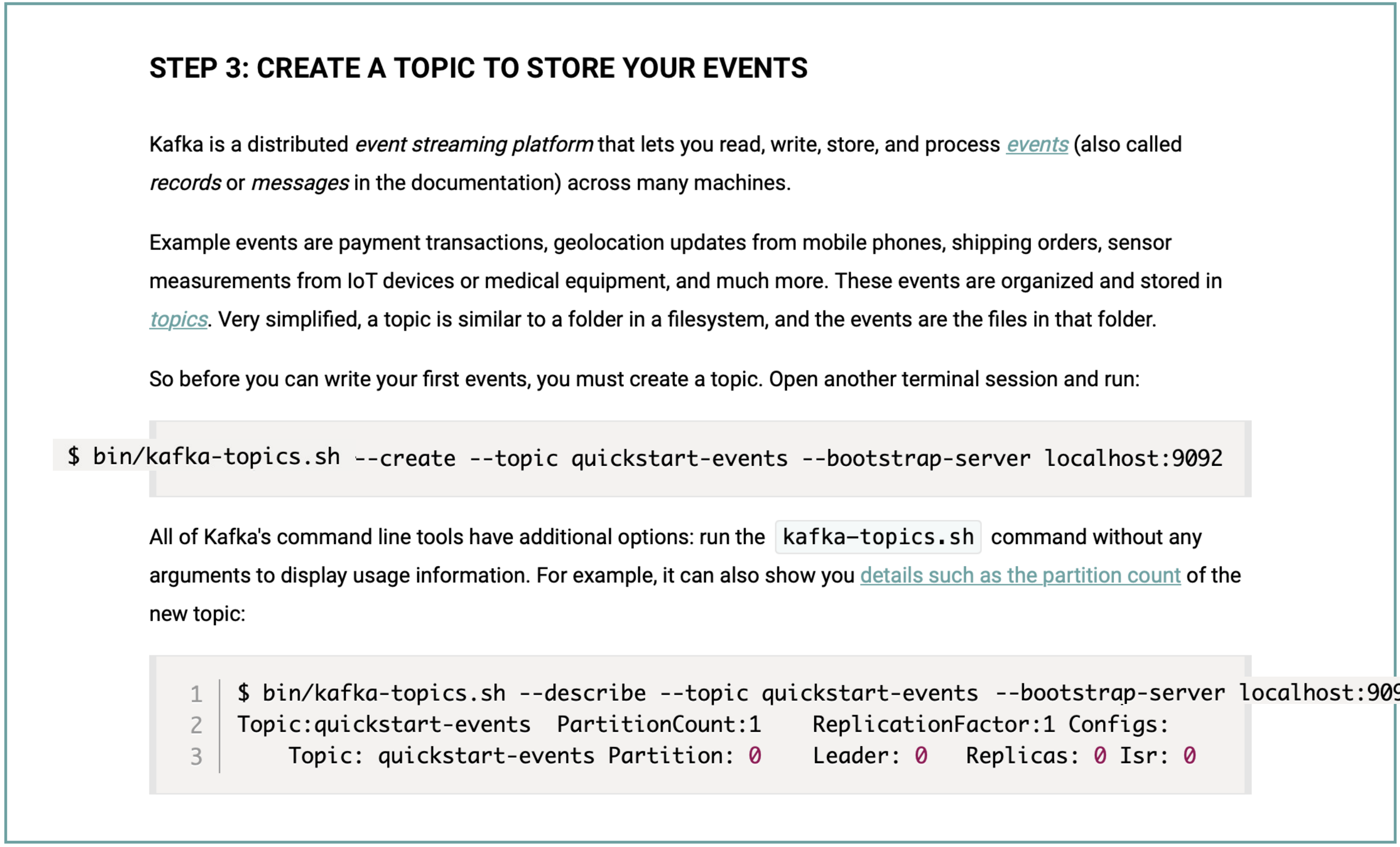
```
1. bin/zkServer.sh start
2. conf/zoo.cfg
   admin.serverPort = 8888
```

Open another terminal session and run:

```
# Start the Kafka broker service
$ bin/kafka-server-start.sh config/server.properties
```

Once all services have successfully launched, you will have a basic Kafka environment running and ready to use.

Kafka - Quickstart



STEP 3: CREATE A TOPIC TO STORE YOUR EVENTS

Kafka is a distributed *event streaming platform* that lets you read, write, store, and process *events* (also called *records* or *messages* in the documentation) across many machines.

Example events are payment transactions, geolocation updates from mobile phones, shipping orders, sensor measurements from IoT devices or medical equipment, and much more. These events are organized and stored in *topics*. Very simplified, a topic is similar to a folder in a filesystem, and the events are the files in that folder.

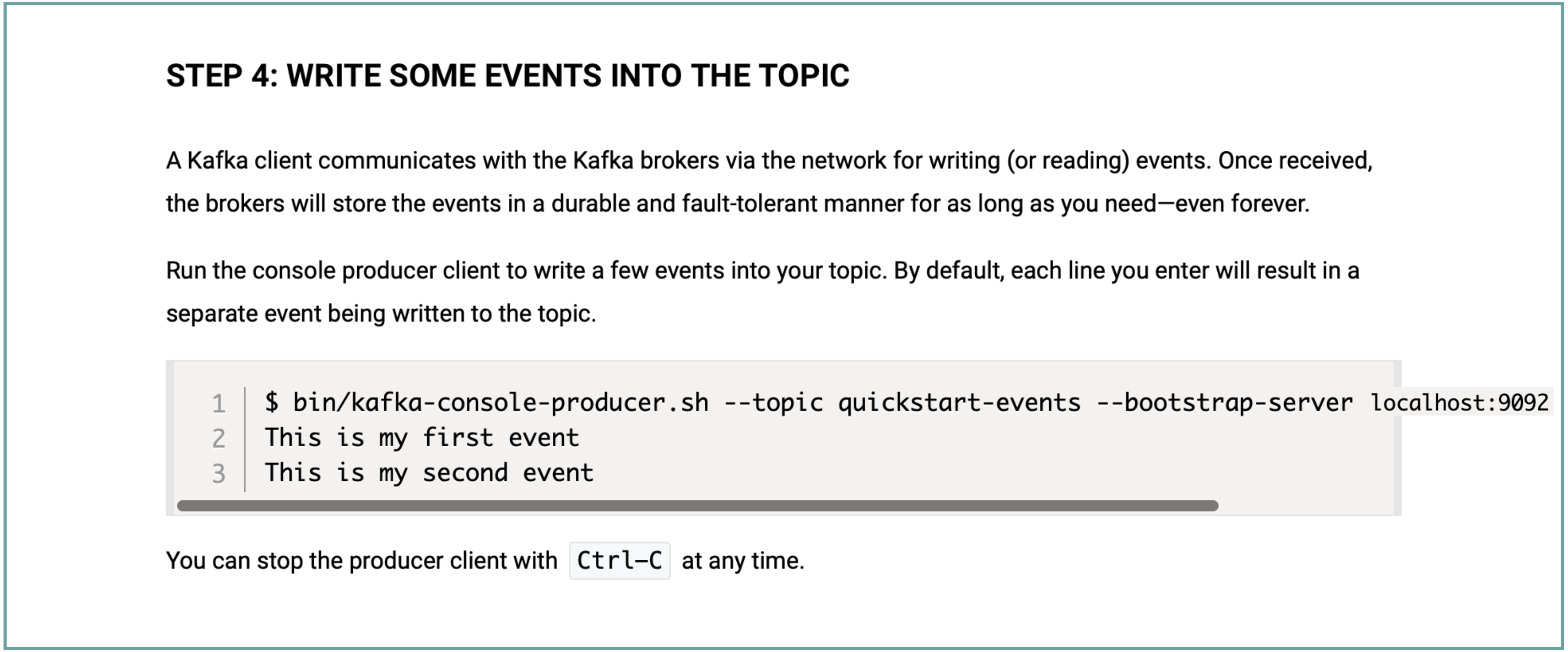
So before you can write your first events, you must create a topic. Open another terminal session and run:

```
$ bin/kafka-topics.sh --create --topic quickstart-events --bootstrap-server localhost:9092
```

All of Kafka's command line tools have additional options: run the `kafka-topics.sh` command without any arguments to display usage information. For example, it can also show you details such as the partition count of the new topic:

```
$ bin/kafka-topics.sh --describe --topic quickstart-events --bootstrap-server localhost:9092
Topic:quickstart-events  PartitionCount:1    ReplicationFactor:1 Configs:
    Topic: quickstart-events Partition: 0    Leader: 0    Replicas: 0 Isr: 0
```

Kafka - Quickstart



STEP 4: WRITE SOME EVENTS INTO THE TOPIC

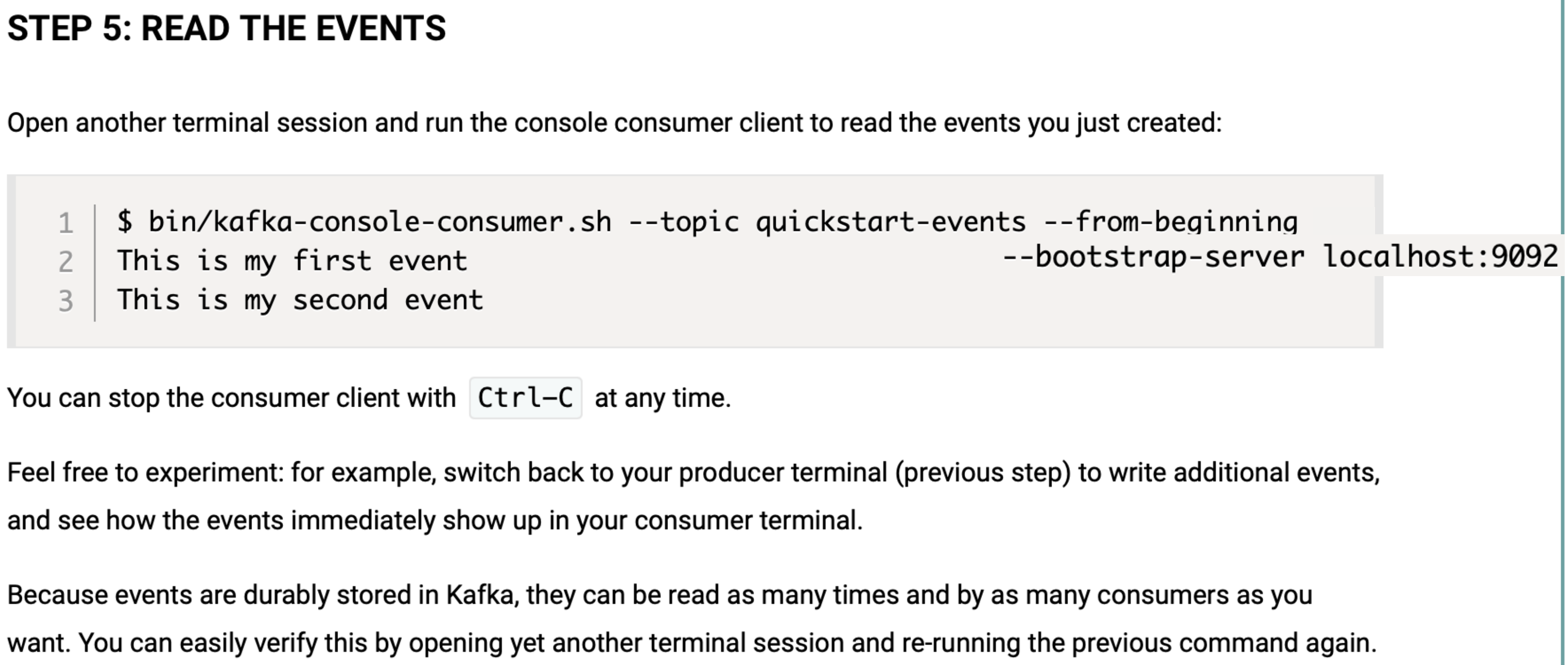
A Kafka client communicates with the Kafka brokers via the network for writing (or reading) events. Once received, the brokers will store the events in a durable and fault-tolerant manner for as long as you need—even forever.

Run the console producer client to write a few events into your topic. By default, each line you enter will result in a separate event being written to the topic.

```
$ bin/kafka-console-producer.sh --topic quickstart-events --bootstrap-server localhost:9092
This is my first event
This is my second event
```

You can stop the producer client with `Ctrl-C` at any time.

Kafka - Quickstart



STEP 5: READ THE EVENTS

Open another terminal session and run the console consumer client to read the events you just created:

```
$ bin/kafka-console-consumer.sh --topic quickstart-events --from-beginning
                                                          --bootstrap-server localhost:9092
This is my first event
This is my second event
```

You can stop the consumer client with `Ctrl-C` at any time.

Feel free to experiment: for example, switch back to your producer terminal (previous step) to write additional events, and see how the events immediately show up in your consumer terminal.

Because events are durably stored in Kafka, they can be read as many times and by as many consumers as you want. You can easily verify this by opening yet another terminal session and re-running the previous command again.

Apache Kafka

- Access Kafka with Java
  - To use Producer API and Consumer API, you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.5.1</version>
</dependency>
```

Apache Kafka

KafkaPro.java

```
public class KafkaPro {
    public static void main(String[] args) throws Exception {
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.setProperty("transactional.id", "my-transactional-id");

        Producer<String, String> producer = null;

        try {
            producer = new KafkaProducer<String, String>(props, new StringSerializer(), new
StringSerializer());
            producer.initTransactions();

            producer.beginTransaction();
            for (int i = 0; i < 100; i++)
                producer.send(new ProducerRecord<>("test", Integer.toString(i), "Message " +
Integer.toString(i)));
            producer.commitTransaction();
        } catch (ProducerFencedException e) {
            producer.close();
        } catch (OutOfOrderSequenceException e) {
            producer.close();
        } catch (AuthorizationException e) {
            producer.close();
        } catch (KafkaException e) {
            producer.abortTransaction();
        }
        producer.close();
    }
```

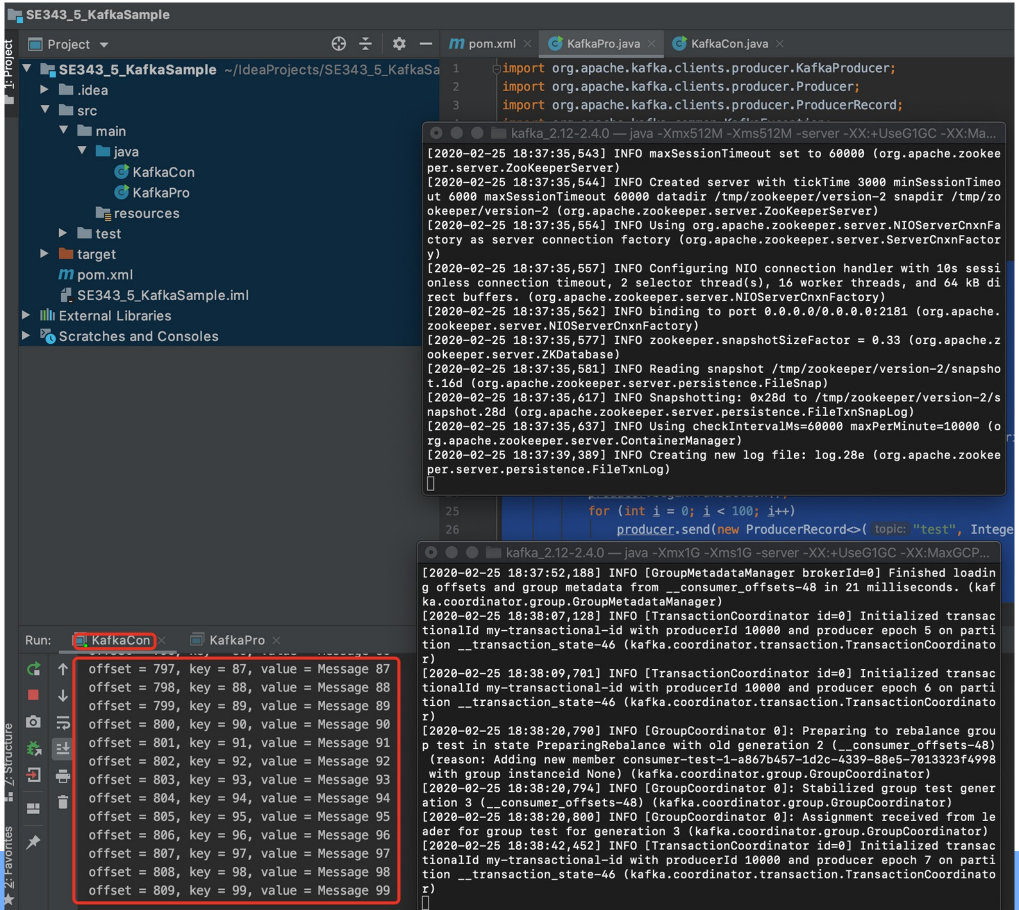
Apache Kafka

KafkaCon.java

```
public class KafkaCon {
    public static void main(String[] args) throws Exception {
        Properties props = new Properties();
        props.setProperty("bootstrap.servers", "localhost:9092");
        props.setProperty("group.id", "test");
        props.setProperty("enable.auto.commit", "true");
        props.setProperty("auto.commit.interval.ms", "1000");
        props.setProperty("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.setProperty("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
        consumer.subscribe(Arrays.asList("test"));

        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            for (ConsumerRecord<String, String> record : records)
                System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(),
record.value());
        }
    }
}
```

Apache Kafka

RElilable, INtelligent & Scalable Systems



```java
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
```

```
kafka_2.12-2.4.0 — java -Xmx512M -Xms512M -server -XX:+UseG1GC -XX:Ma...
[2020-02-25 18:37:35,543] INFO maxSessionTimeout set to 60000 (org.apache.zookee
per.server.ZooKeeperServer)
[2020-02-25 18:37:35,544] INFO Created server with tickTime 3000 minSessionTimeo
ut 6000 maxSessionTimeout 60000 datadir /tmp/zookeeper/version-2 snapdir /tmp/zo
okeeper/version-2 (org.apache.zookeeper.server.ZooKeeperServer)
[2020-02-25 18:37:35,554] INFO Using org.apache.zookeeper.server.NIOServerCnxnFa
ctory as server connection factory (org.apache.zookeeper.server.ServerCnxnFactor
y)
[2020-02-25 18:37:35,557] INFO Configuring NIO connection handler with 10s sessi
onless connection timeout, 2 selector thread(s), 16 worker threads, and 64 kB di
rect buffers. (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2020-02-25 18:37:35,562] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.
zookeeper.server.NIOServerCnxnFactory)
[2020-02-25 18:37:35,577] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.z
ookeeper.server.ZKDatabase)
[2020-02-25 18:37:35,581] INFO Reading snapshot /tmp/zookeeper/version-2/snapsho
t.16d (org.apache.zookeeper.server.persistence.FileSnap)
[2020-02-25 18:37:35,617] INFO Snapshotting: 0x28d to /tmp/zookeeper/version-2/s
napshot.28d (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2020-02-25 18:37:35,637] INFO Using checkIntervalMs=60000 maxPerMinute=10000 (o
rg.apache.zookeeper.server.ContainerManager)
[2020-02-25 18:37:39,389] INFO Creating new log file: log.28e (org.apache.zookee
per.server.persistence.FileTxnLog)
```

```java
        for (int i = 0; i < 100; i++)
            producer.send(new ProducerRecord<>( topic: "test", Intege
```

```
offset = 797, key = 87, value = Message 87
offset = 798, key = 88, value = Message 88
offset = 799, key = 89, value = Message 89
offset = 800, key = 90, value = Message 90
offset = 801, key = 91, value = Message 91
offset = 802, key = 92, value = Message 92
offset = 803, key = 93, value = Message 93
offset = 804, key = 94, value = Message 94
offset = 805, key = 95, value = Message 95
offset = 806, key = 96, value = Message 96
offset = 807, key = 97, value = Message 97
offset = 808, key = 98, value = Message 98
offset = 809, key = 99, value = Message 99
```

```
kafka_2.12-2.4.0 — java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCP...
[2020-02-25 18:37:52,188] INFO [GroupMetadataManager brokerId=0] Finished loadin
g offsets and group metadata from __consumer_offsets-48 in 21 milliseconds. (kaf
ka.coordinator.group.GroupMetadataManager)
[2020-02-25 18:38:07,128] INFO [TransactionCoordinator id=0] Initialized transac
tionalId my-transactional-id with producerId 10000 and producer epoch 5 on parti
tion __transaction_state-46 (kafka.coordinator.transaction.TransactionCoordinato
r)
[2020-02-25 18:38:09,701] INFO [TransactionCoordinator id=0] Initialized transac
tionalId my-transactional-id with producerId 10000 and producer epoch 6 on parti
tion __transaction_state-46 (kafka.coordinator.transaction.TransactionCoordinato
r)
[2020-02-25 18:38:20,790] INFO [GroupCoordinator 0]: Preparing to rebalance grou
p test in state PreparingRebalance with old generation 2 (__consumer_offsets-48)
 (reason: Adding new member consumer-test-1-a867b457-1d2c-4339-88e5-7013323f4998
 with group instanceid None) (kafka.coordinator.group.GroupCoordinator)
[2020-02-25 18:38:20,794] INFO [GroupCoordinator 0]: Stabilized group test gener
ation 3 (__consumer_offsets-48) (kafka.coordinator.group.GroupCoordinator)
[2020-02-25 18:38:20,800] INFO [GroupCoordinator 0]: Assignment received from le
ader for group test for generation 3 (kafka.coordinator.group.GroupCoordinator)
[2020-02-25 18:38:42,452] INFO [TransactionCoordinator id=0] Initialized transac
tionalId my-transactional-id with producerId 10000 and producer epoch 7 on parti
tion __transaction_state-46 (kafka.coordinator.transaction.TransactionCoordinato
r)
```

Spring for Kafka

REliable, INtelligent & Scalable Systems

- Consumer

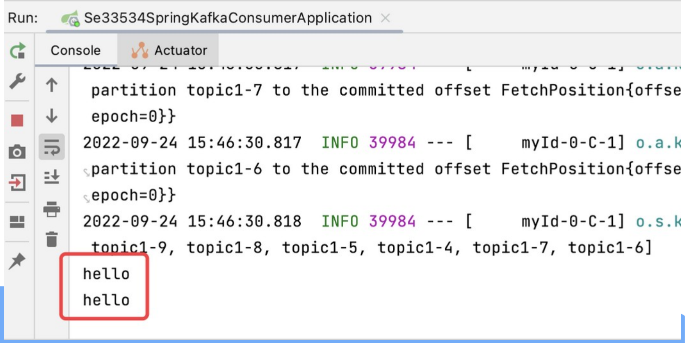
```
@SpringBootApplication
public class ConsumerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConsumerApplication.class, args);
    }

    @Bean
    public NewTopic topic() {
        return TopicBuilder.name("topic1")
                .partitions(10)
                .replicas(1)
                .build();
    }

    @KafkaListener(id = "myId", topics = "topic1")
    public void listen(String in) {
        System.out.println(in);
    }
}
```

Spring for Kafka

REliable, INtelligent & Scalable Systems

- Producer

```
@SpringBootApplication
public class KafkaProducerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ProducerApplication.class, args);
    }

    @Bean
    public NewTopic topic() {
        return TopicBuilder.name("topic1")
                .partitions(10)
                .replicas(1)
                .build();
    }

    @Bean
    public ApplicationRunner runner(KafkaTemplate<String, String> template) {
        return args -> {
            template.send("topic1", "hello");
        };
    }
}
```



```
 partition topic1-7 to the committed offset FetchPosition{offse
 epoch=0}}
2022-09-24 15:46:30.817  INFO 39984 --- [     myId-0-C-1] o.a.k
 partition topic1-6 to the committed offset FetchPosition{offse
 epoch=0}}
2022-09-24 15:46:30.818  INFO 39984 --- [     myId-0-C-1] o.s.k
 topic1-9, topic1-8, topic1-5, topic1-4, topic1-7, topic1-6]
hello
hello
```

Place Order with Kafka

- BankController

```
@RestController
public class BankController {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @RequestMapping("/send")
    public void send() {
        String data = "Tom,Jerry,80";
        kafkaTemplate.send("topic1",  "key", data);
        System.out.println(data);
    }
}
```

Place Order with Kafka

REliable, INtelligent & Scalable Systems

- BankListener

```
@Component
public class BankListener {
    @Autowired
    private BankService bankService;

    @KafkaListener(topics = "topic1", groupId = "group_topic_test")
    public void topicListener(ConsumerRecord<String, String> record) {
        String[] value = record.value().split(",");
        bankService.transfer(value[0], value[1], Integer.valueOf(value[2]));
    }
}
```

Place Order with Kafka

- SQL Script

```sql
DROP TABLE IF EXISTS `bank`;
CREATE TABLE `bank` (
`account` varchar(45) NOT NULL,
`balance` int DEFAULT NULL,
PRIMARY KEY (`account`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

References

- Apache Kafka Quick Start
  - https://kafka.apache.org/quickstart
- Spring for Apache Kafka
  - https://docs.spring.io/spring-kafka/docs/current/reference/html/#spring-boot-consumer-app
- 如何轻松在 SpringBoot 中正确配置并运行 Kafka
  - https://blog.csdn.net/Eternal_Blue/article/details/125293622
- 详细解决 zookeeper 启动占用 8080 端口方法
  - https://blog.csdn.net/tianynnb/article/details/128617174

Thank You!