Kafka Producer - Java

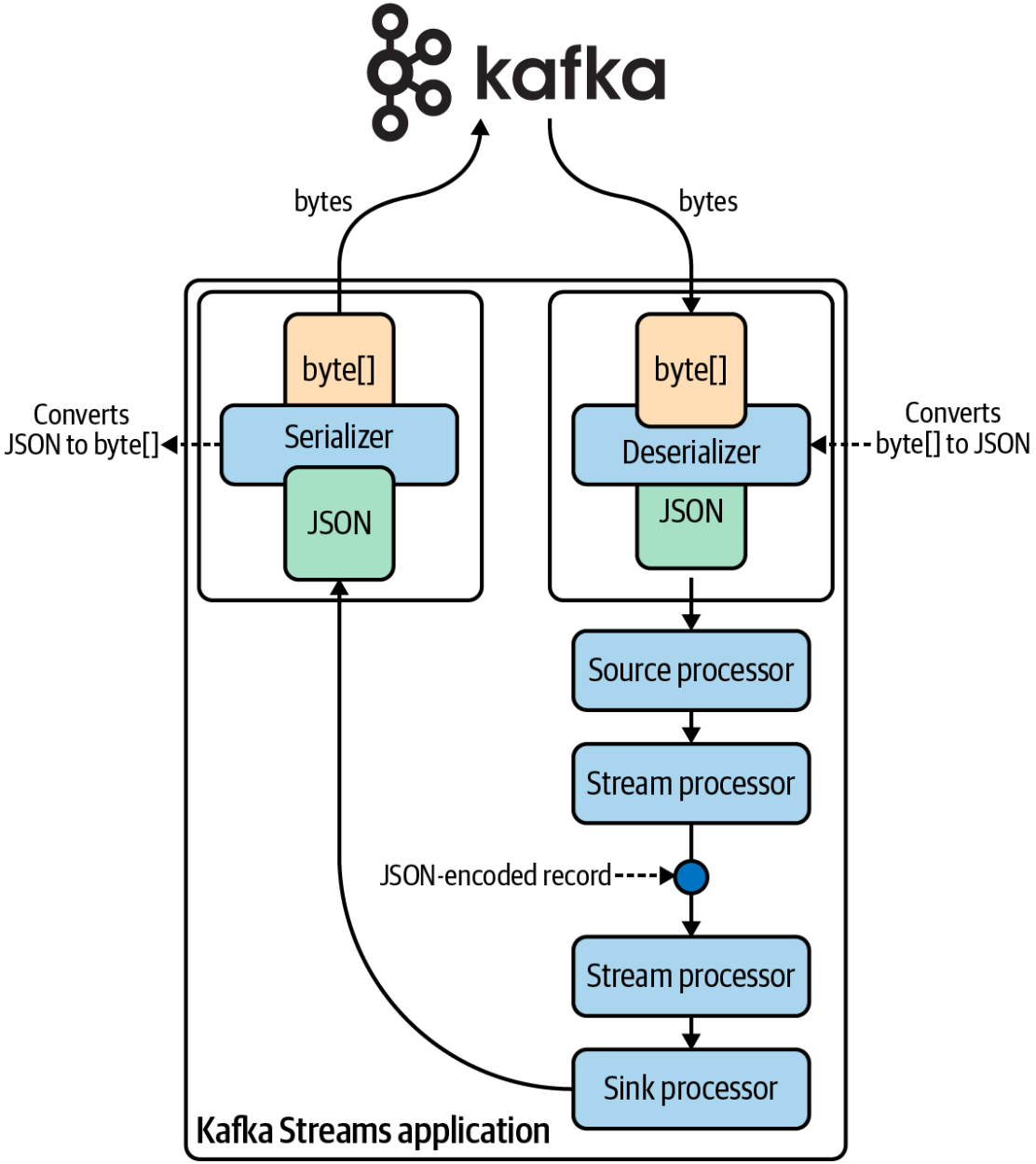
Objectives Create Producer

Tos

- Create simple example that creates a ***Kafka Producer***
- Create a new replicated ***Kafka topic***
- ***Create Producer*** that uses topic to send records
- ***Send records*** with ***Kafka Producer***
- ***Send records asynchronously.***
- ***Send records synchronously***

Ser/Deserialization Architecture

Tos



Create Replicated Kafka Topic

Tos

create-topic.sh

```
#!/usr/bin/env bash
cd ~/kafka-training

## Create topics
kafka/bin/kafka-topics.sh --create \
    --replication-factor 3 \
    --partitions 13 \
    --topic my-example-topic \
    --bootstrap-server localhost:9092

## List created topics
kafka/bin/kafka-topics.sh --list \
--bootstrap-server localhost:9092
```

)92

```
$ ./create-topic.sh
Created topic "my-example-topic".
EXAMPLE_TOPIC
__consumer_offsets
kafkatopic
my-example-topic
my-failsafe-topic
my-topic
```

Gradle/Maven Build script

Tos

kafka-training ×

```
group 'cloudurable-kafka'
version '1.0-SNAPSHOT'

apply plugin: 'java'

sourceCompatibility = 1.8

repositories {
    mavenCentral()
}

dependencies {
    compile 'org.apache.kafka:kafka-clients:0.10.2.0'
    compile 'ch.qos.logback:logback-classic:1.2.2'
}
```

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.6.0</version>
</dependency>
```

Create Kafka Producer to send records

Tos

- Specify bootstrap servers
- Specify client.id
- Specify Record Key serializer
- Specify Record Value serializer

Common Kafka imports and constants

Tos

KafkaProducerExample.java

KafkaProducerExample

```
1  package com.cloudurable.kafka;
2
3  import org.apache.kafka.clients.producer.*;
4  import org.apache.kafka.common.serialization.LongSerializer;
5  import org.apache.kafka.common.serialization.StringSerializer;
6
7  import java.util.Properties;
8
9  public class KafkaProducerExample {
10
11     private final static String TOPIC = "my-example-topic";
12     private final static String BOOTSTRAP_SERVERS =
13             "localhost:9092,localhost:9093,localhost:9094";
```

Create Kafka Producer to send records

Tos

KafkaProducerExample.java

KafkaProducerExample

```
14
15      private static Producer<Long, String> createProducer() {
16          Properties props = new Properties();
17          props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
18                                              BOOTSTRAP_SERVERS);
19          props.put(ProducerConfig.CLIENT_ID_CONFIG, "KafkaExampleProducer");
20          props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
21                                          LongSerializer.class.getName());
22          props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
23                                        StringSerializer.class.getName());
24          return new KafkaProducer<>(props);
25      }
26
```

Send async records with Kafka

Tos

KafkaProducerExample.java ×

KafkaProducerExample › runProducer()

```
27
28      static void runProducer(final int sendMessageCount) throws Exception {
29          final Producer<Long, String> producer = createProducer();
30          long time = System.currentTimeMillis();
31
32          try {
33              for (long index = time; index < time + sendMessageCount; index++) {
34                  final ProducerRecord<Long, String> record =
35                          new ProducerRecord<>(TOPIC, index,
36                                  value: "Hello Mom " + index);
37
38                  RecordMetadata metadata = producer.send(record).get();
39
40                  long elapsedTime = System.currentTimeMillis() - time;
41                  System.out.printf("sent record(key=%s value=%s) " +
42                                  "meta(partition=%d, offset=%d) time=%d\n",
43                          record.key(), record.value(), metadata.partition(),
44                          metadata.offset(), elapsedTime);
45
46              }
47          } finally {
48              producer.flush();
49              producer.close();
50          }
51      }
```

Running the Producer

Tos

```
public static void main(String... args) throws Exception {
    if (args.length == 0) {
        runProducer( sendMessageCount: 5);
    } else {
        runProducer(Integer.parseInt(args[0]));
    }
}
```

KafkaProducer<K,V>

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

Producer<String, String> producer = new KafkaProducer<>(props);
for (int i = 0; i < 100; i++)
    producer.send(new ProducerRecord<String, String>("my-topic", Integer.toString(i), Integer.toString(i)));

producer.close();
```

Async Interface Callback

Tos

org.apache.kafka.clients.producer

Interface Callback

`public interface` **`Callback`**

A callback interface that the user can implement to allow code to execute when the request is complete. This callback will generally execute in the background I/O thread so it should be fast.

Method Summary

Methods

Modifier and Type	Method and Description
`void`	**`onCompletion`**`(`**`RecordMetadata`** `metadata,` **`Exception`** `exception)` A callback method the user can implement to provide asynchronous handling of request completion.

Async Send Method

Tos

send

```
public Future<RecordMetadata> send(ProducerRecord<K,V> record,
                                   Callback callback)
```

Asynchronously send a record to a topic and invoke the provided callback when the send has been acknowledged.

The send is asynchronous and this method will return immediately once the record has been stored in the buffer of records waiting to be sent. This allows sending many records in parallel without blocking to wait for the response after each one.

- Used to send a record to a topic
- provided callback gets called when the send is acknowledged
- Send is asynchronous, and method will return immediately
  - once the record gets stored in the buffer of records waiting to post to the Kafka broker
- Allows sending many records in parallel without blocking

Callback

Tos

```
String val = "Hello Kafka :" + Integer.toString(i);
producer.send(
        new ProducerRecord<String, String>("my-first-topic", Integer.toString(i),val)
        , (recordMetadata, exception) -> {
            if (exception == null) {
                System.out.println("Record written to offset " +
                        recordMetadata.offset() + " timestamp " +
                        recordMetadata.timestamp());
            } else {
                System.err.println("An error occurred");
                exception.printStackTrace(System.err);
            }
        });
```

Progress Console

<terminated> KafkaProducerExample (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_271.jdk/Contents/Home/bin/java (14-Mar-2022, 3:38:16 PM –

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
 Record written to offset 2 timestamp 1647252504793
Sent : 0 Record written to offset 3 timestamp 1647252504878
Sent : 1
```

KafkaProducer send() Method

Tos

- Two forms of send with callback and with no callback both return Future
  - Asynchronously sends a record to a topic
  - Callback gets invoked when send has been acknowledged.
- send is asynchronous and return right away as soon as record has added to send buffer
- Sending many records at once without blocking for response from Kafka broker
- Result of send is a RecordMetadata
  - record partition, record offset, record timestamp
- Callbacks for records sent to same partition are executed in order

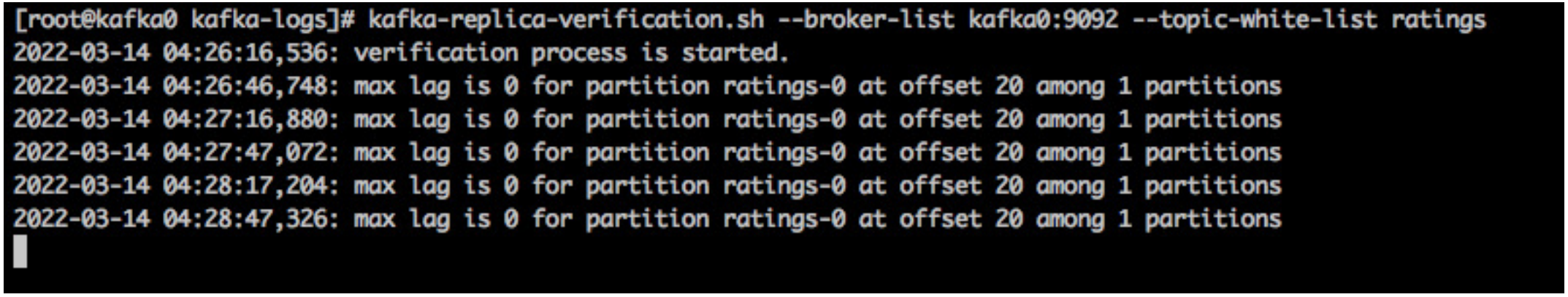
Checking that replication is working

Tos

- Verify that replication is working with kafka-replica-verification
- Utility that ships with Kafka, If lag or outage you will see it as follows:

$ ***kafka/bin/kafka-replica-verification.sh --broker-list localhost:9092 --topic-white-list my-example-topic***
2017-05-17 14:06:46,446: verification process is started.
2017-05-17 14:07:16,416: max lag is 0 for partition [my-example-topic,12] at offset 197 among 13 partitions
2017-05-17 14:07:46,417: max lag is 0 for partition [my-example-topic,12] at offset 201 among 13 partitions

2017-05-17 14:36:47,497: max lag is 11 for partition [my-example-topic,5] at offset 272 among 13 partitions
2017-05-17 14:37:19,408: max lag is 15 for partition [my-example-topic,5] at offset 272 among 13 partitions
…
2017-05-17 14:38:49,607: max lag is 0 for partition [my-example-topic,12] at offset 272 among 13 partitions



Java Kafka Simple Producer Recap

Tos

- Created simple example that creates a ***Kafka Producer***
- Created a new replicated ***Kafka topic***
- ***Created Producer*** that uses topic to send records
- ***Sent records*** with ***Kafka Producer using async and sync send***

Kafka Producer Review

Tos

- What does the Callback lambda do?
- What will happen if the first server is down in the bootstrap list? Can the producer still connect to the other Kafka brokers in the cluster?
- When would you use Kafka async send vs. sync send?
- Why do you need two serializers for a Kafka record?

Lab : Producer in Java – 90 Minutes