

# Kafka Consumer - Java

- ❖ Create simple example that creates a ***Kafka Consumer***
  - ❖ that consumes messages from the ***Kafka Producer*** we wrote
- ❖ ***Create Consumer*** that uses topic from first example to receive messages
- ❖ ***Process messages*** from Kafka with ***Consumer***
- ❖ Demonstrate how Consumer Groups work

- ❖ **Specify bootstrap servers**
- ❖ **Specify Consumer Group**
- ❖ **Specify Record Key deserializer**
- ❖ **Specify Record Value deserializer**
- ❖ **Subscribe to Topic from last session**

# Common Kafka imports and constants

KafkaConsumerExample.java x

KafkaConsumerExample

```
1 package com.cloudurable.kafka;
2 import org.apache.kafka.clients.consumer.*;
3 import org.apache.kafka.clients.consumer.Consumer;
4 import org.apache.kafka.common.serialization.LongDeserializer;
5 import org.apache.kafka.common.serialization.StringDeserializer;
6
7 import java.util.Collections;
8 import java.util.Properties;
9
10 public class KafkaConsumerExample {
11
12     private final static String TOPIC = "my-example-topic";
13     private final static String BOOTSTRAP_SERVERS =
14         "localhost:9092,localhost:9093,localhost:9094";
15 }
```

# Create Consumer using Topic to Receive Records

```
KafkaConsumerExample.java x
KafkaConsumerExample createConsumer()

16 private static Consumer<Long, String> createConsumer() {
17     final Properties props = new Properties();
18     props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
19             BOOTSTRAP_SERVERS);
20     props.put(ConsumerConfig.GROUP_ID_CONFIG,
21             "KafkaExampleConsumer");
22     props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
23             LongDeserializer.class.getName());
24     props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
25             StringDeserializer.class.getName());
26
27     // Create the consumer using props.
28     final Consumer<Long, String> consumer =
29         new KafkaConsumer<>(props);
30
31     // Subscribe to the topic.
32     consumer.subscribe(Collections.singletonList(TOPIC));
33     return consumer;
34 }
```

# Process messages from Kafka with Consumer

Tos

```
KafkaConsumerExample.java x
KafkaConsumerExample

40 static void runConsumer() throws InterruptedException {
41     final Consumer<Long, String> consumer = createConsumer();
42
43     final int giveUp = 100;    int noRecordsCount = 0;
44
45     while (true) {
46         final ConsumerRecords<Long, String> consumerRecords =
47             consumer.poll( timeout: 1000);
48
49         if (consumerRecords.count()==0) {
50             noRecordsCount++;
51             if (noRecordsCount > giveUp) break;
52             else continue;
53         }
54
55         consumerRecords.forEach(record -> {
56             System.out.printf("Consumer Record:(%d, %s, %d, %d)\n",
57                 record.key(), record.value(),
58                 record.partition(), record.offset());
59         });
60
61         consumer.commitAsync();
62     }
63     consumer.close();
64     System.out.println("DONE");
```

- ❖ `poll()` method returns fetched records based on current partition offset
- ❖ Blocking method waiting for specified time if no records available
- ❖ When/If records available, method returns straight away
- ❖ Control the maximum records returned by the `poll()` with `props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 100);`
- ❖ `poll()` is not meant to be called from multiple threads



# Running both Consumer then Producer

Tos

```
68 public static void main(String... args) throws Exception {  
69     runConsumer();  
70 }  
71
```

```
Run KafkaConsumerExample  
ssl.protocol = TLS  
ssl.provider = null  
ssl.secure.random.implementation = null  
ssl.trustmanager.algorithm = PKIX  
ssl.truststore.location = null  
ssl.truststore.password = null  
ssl.truststore.type = JKS  
value.deserializer = class org.apache.kafka.common.serialization.StringDeserializer  
  
15:17:35.267 [main] INFO o.a.kafka.common.utils.AppInfoParser - Kafka version : 0.10.2.0  
15:17:35.267 [main] INFO o.a.kafka.common.utils.AppInfoParser - Kafka commitId : 576d93a8dc0cf421  
15:17:35.384 [main] INFO o.a.k.c.c.i.AbstractCoordinator - Discovered coordinator 10.0.0.115:9093  
15:17:35.391 [main] INFO o.a.k.c.c.i.ConsumerCoordinator - Revoking previously assigned partitions  
15:17:35.391 [main] INFO o.a.k.c.c.i.AbstractCoordinator - (Re-)joining group KafkaExampleConsumer  
15:17:42.257 [main] INFO o.a.k.c.c.i.AbstractCoordinator - Successfully joined group KafkaExampleC  
15:17:42.259 [main] INFO o.a.k.c.c.i.ConsumerCoordinator - Setting newly assigned partitions [my-e  
Consumer Record:(1494973064716, Hello Mom 1494973064716, 6, 4)  
Consumer Record:(1494973064719, Hello Mom 1494973064719, 10, 6)  
Consumer Record:(1494973064718, Hello Mom 1494973064718, 9, 9)  
Consumer Record:(1494973064717, Hello Mom 1494973064717, 12, 9)  
Consumer Record:(1494973064720, Hello Mom 1494973064720, 4, 8)
```



```
logback.xml x
1 <configuration>
2   <appender name="STDOUT"
3     class="ch.qos.logback.core.ConsoleAppender">
4     <encoder>
5       <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level
6         %logger{36} - %msg%n</pattern>
7     </encoder>
8   </appender>
9
10  <logger name="org.apache.kafka" level="INFO"/>
11  <logger name="org.apache.kafka.common.metrics" level="INFO"/>
12
13  <root level="debug">
14    <appender-ref ref="STDOUT" />
15  </root>
16 </configuration>
```

❖ Kafka uses sl4j

❖ Set level to DEBUG to see what is going on

# Try this: Consumers in Same Group

Tos

- ❖ Three consumers and one producer sending 25 records
- ❖ Run three consumers processes
- ❖ Change Producer to send 25 records instead of 5
- ❖ Run one producer
- ❖ What happens?

# Outcome 3 Consumers Load Share

## Consumer 0 (key, value, partition, offset)

```
Consumer Record: (1495042369488, Hello Mom 1495042369488, 0, 9)
Consumer Record: (1495042369490, Hello Mom 1495042369490, 3, 9)
Consumer Record: (1495042369498, Hello Mom 1495042369498, 3, 10)
Consumer Record: (1495042369504, Hello Mom 1495042369504, 3, 11)
Consumer Record: (1495042369508, Hello Mom 1495042369508, 3, 12)
Consumer Record: (1495042369491, Hello Mom 1495042369491, 4, 9)
Consumer Record: (1495042369503, Hello Mom 1495042369503, 4, 10)
Consumer Record: (1495042369505, Hello Mom 1495042369505, 4, 11)
Consumer Record: (1495042369494, Hello Mom 1495042369494, 2, 9)
Consumer Record: (1495042369499, Hello Mom 1495042369499, 2, 10)
```

## Consumer 1 (key, value, partition, offset)

```
Consumer Record: (1495042369486, Hello Mom 1495042369486, 12, 10)
Consumer Record: (1495042369493, Hello Mom 1495042369493, 12, 11)
Consumer Record: (1495042369507, Hello Mom 1495042369507, 12, 12)
Consumer Record: (1495042369487, Hello Mom 1495042369487, 9, 12)
Consumer Record: (1495042369492, Hello Mom 1495042369492, 10, 7)
Consumer Record: (1495042369495, Hello Mom 1495042369495, 10, 8)
Consumer Record: (1495042369501, Hello Mom 1495042369501, 11, 8)
Consumer Record: (1495042369506, Hello Mom 1495042369506, 11, 9)
```

## Consumer 2 (key, value, partition, offset)

```
Consumer Record: (1495042369509, Hello Mom 1495042369509, 6, 6)
Consumer Record: (1495042369497, Hello Mom 1495042369497, 7, 11)
Consumer Record: (1495042369500, Hello Mom 1495042369500, 7, 12)
Consumer Record: (1495042369496, Hello Mom 1495042369496, 5, 7)
Consumer Record: (1495042369510, Hello Mom 1495042369510, 5, 8)
Consumer Record: (1495042369489, Hello Mom 1495042369489, 8, 9)
Consumer Record: (1495042369502, Hello Mom 1495042369502, 8, 10)
```

## Producer

```
sent record(key=1495042369509 value=Hello Mom 1495042369509) meta(partition=6, offset=6) t
sent record(key=1495042369487 value=Hello Mom 1495042369487) meta(partition=9, offset=12) t
sent record(key=1495042369486 value=Hello Mom 1495042369486) meta(partition=12, offset=10) t
sent record(key=1495042369493 value=Hello Mom 1495042369493) meta(partition=12, offset=11) t
sent record(key=1495042369507 value=Hello Mom 1495042369507) meta(partition=12, offset=12) t
sent record(key=1495042369488 value=Hello Mom 1495042369488) meta(partition=0, offset=9) t
sent record(key=1495042369490 value=Hello Mom 1495042369490) meta(partition=3, offset=9) t
sent record(key=1495042369498 value=Hello Mom 1495042369498) meta(partition=3, offset=10) t
sent record(key=1495042369504 value=Hello Mom 1495042369504) meta(partition=3, offset=11) t
sent record(key=1495042369508 value=Hello Mom 1495042369508) meta(partition=3, offset=12) t
sent record(key=1495042369497 value=Hello Mom 1495042369497) meta(partition=7, offset=11) t
sent record(key=1495042369500 value=Hello Mom 1495042369500) meta(partition=7, offset=12) t
sent record(key=1495042369492 value=Hello Mom 1495042369492) meta(partition=10, offset=7) t
sent record(key=1495042369495 value=Hello Mom 1495042369495) meta(partition=10, offset=8) t
sent record(key=1495042369491 value=Hello Mom 1495042369491) meta(partition=4, offset=9) t
sent record(key=1495042369503 value=Hello Mom 1495042369503) meta(partition=4, offset=10) t
sent record(key=1495042369505 value=Hello Mom 1495042369505) meta(partition=4, offset=11) t
sent record(key=1495042369496 value=Hello Mom 1495042369496) meta(partition=5, offset=7) t
sent record(key=1495042369510 value=Hello Mom 1495042369510) meta(partition=5, offset=8) t
sent record(key=1495042369489 value=Hello Mom 1495042369489) meta(partition=8, offset=9) t
sent record(key=1495042369502 value=Hello Mom 1495042369502) meta(partition=8, offset=10) t
sent record(key=1495042369501 value=Hello Mom 1495042369501) meta(partition=11, offset=8) t
sent record(key=1495042369506 value=Hello Mom 1495042369506) meta(partition=11, offset=9) t
sent record(key=1495042369494 value=Hello Mom 1495042369494) meta(partition=2, offset=9) t
sent record(key=1495042369499 value=Hello Mom 1495042369499) meta(partition=2, offset=10) t
```

**Which consumer owns partition 10? How many Consumer Records objects did Consumer 0 get?**

**What is the next offset from Partition 5 that Consumer 2 should get?**

**Why does each consumer get unique**

# Try this: Consumers in Different Groups

- ❖ Three consumers with unique group and one producer sending 5 records
- ❖ Modify Consumer to have unique group id
- ❖ Run three consumers processes
- ❖ Run one producer
- ❖ What happens?

# Pass Unique Group Id

```
KafkaConsumerExample.java x
KafkaConsumerExample createConsumer()

16
17 private static Consumer<Long, String> createConsumer() {
18     final Properties props = new Properties();
19
20     props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
21               BOOTSTRAP_SERVERS);
22
23     props.put(ConsumerConfig.GROUP_ID_CONFIG,
24               "KafkaExampleConsumer" +
25               System.currentTimeMillis());
26
27     props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
28               LongDeserializer.class.getName());
29     props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
30               StringDeserializer.class.getName());
31
32
33     //Take up to 100 records at a time
34     props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 100);
```



## Consumer 0(key, value, partition, offset)

```
Consumer Record:(1495043607696, Hello Mom 1495043607696, 0, 10)
Consumer Record:(1495043607699, Hello Mom 1495043607699, 7, 13)
Consumer Record:(1495043607700, Hello Mom 1495043607700, 2, 11)
Consumer Record:(1495043607697, Hello Mom 1495043607697, 10, 9)
Consumer Record:(1495043607698, Hello Mom 1495043607698, 10, 10)
```

## Consumer 1(key, value, partition, offset)

```
Consumer Record:(1495043607696, Hello Mom 1495043607696, 0, 10)
Consumer Record:(1495043607699, Hello Mom 1495043607699, 7, 13)
Consumer Record:(1495043607700, Hello Mom 1495043607700, 2, 11)
Consumer Record:(1495043607697, Hello Mom 1495043607697, 10, 9)
Consumer Record:(1495043607698, Hello Mom 1495043607698, 10, 10)
```

## Consumer 2(key, value, partition, offset)

```
Consumer Record:(1495043607696, Hello Mom 1495043607696, 0, 10)
Consumer Record:(1495043607699, Hello Mom 1495043607699, 7, 13)
Consumer Record:(1495043607700, Hello Mom 1495043607700, 2, 11)
Consumer Record:(1495043607697, Hello Mom 1495043607697, 10, 9)
Consumer Record:(1495043607698, Hello Mom 1495043607698, 10, 10)
```

## Producer

```
sent (key=1495043832401 ) meta(partition=7, offset=14)
sent (key=1495043832400 ) meta(partition=1, offset=11)
sent (key=1495043832404 ) meta(partition=6, offset=7)
sent (key=1495043832402 ) meta(partition=0, offset=11)
sent (key=1495043832403 ) meta(partition=3, offset=13)
```

**Which consumer(s) owns partition 10?**

**How many ConsumerRecords objects did Consumer 0 get?**

**What is the next offset from Partition 2 that Consumer 2 get?**

**Why does each Consumer get the same messages?**

# Try this: Consumers in Groups

- ❖ Modify consumer: change group id back to non-unique value
- ❖ Make the batch size 5
- ❖ Add a 100 ms delay in the consumer after each message poll and print out record count and partition count
- ❖ Modify the Producer to run 10 times with a 30 second delay after each run and to send 50 messages each run
- ❖ Run producer



# Modify Consumer

```
KafkaConsumerExample.java x
KafkaConsumerExample
8 import java.util.Properties;
9
10 public class KafkaConsumerExample {
11
12     private final static String TOPIC = "my-example-topic";
13     private final static String BOOTSTRAP_SERVERS =
14         "localhost:9092,localhost:9093,localhost:9094";
15
16
17     private static Consumer<Long, String> createConsumer() {
18         final Properties props = new Properties();
19
20         props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
21             BOOTSTRAP_SERVERS);
22
23         props.put(ConsumerConfig.GROUP_ID_CONFIG,
24             "KafkaExampleConsumer");
25
26         props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
27             LongDeserializer.class.getName());
28         props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
29             StringDeserializer.class.getName());
30
31         props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 5);
```

❖ Change group name to common name

❖ Change batch size to 5

# Add a 100 ms delay to Consumer after poll

```
47 try {
48     final int giveUp = 1000; int noRecordsCount = 0;
49
50     while (true) {
51         final ConsumerRecords<Long, String> consumerRecords =
52             consumer.poll( timeout: 1000);
53
54         if (consumerRecords.count() == 0) {
55             noRecordsCount++;
56             if (noRecordsCount > giveUp) break;
57             else continue;
58         }
59
60         System.out.printf("New ConsumerRecords par count %d count %d\n",
61             consumerRecords.partitions().size(),
62             consumerRecords.count());
63
64         consumerRecords.forEach(record -> {
65             System.out.printf("Consumer Record:(%d, %s, %d, %d)\n",
66                 record.key(), record.value(),
67                 record.partition(), record.offset());
68         });
69         Thread.sleep( millis: 100);
70         consumer.commitAsync();
71     }
72 }
73 finally {
74     consumer.close();
75 }
```

# Modify Producer: Run 10 times, add 30 second delay

Tos

```
KafkaProducerExample.java x
KafkaProducerExample main()
104
105 ► public static void main(String... args)
106     throws Exception {
107     for (int index = 0; index < 10; index++) {
108         runProducer( sendMessageCount: 50);
109         Thread.sleep( millis: 30_000);
110     }
111 }
112 }
113
```

- ❖ Run 10 times
- ❖ Add 30 second delay
- ❖ Send 50 records

# Notice one or more partitions per ConsumerRecords

KafkaConsumerExample	KafkaConsumerExample	KafkaConsumerExample
↑	New ConsumerRecords par count 1 count 4	
↓	Consumer Record:(1495055263352, Hello Mom 1495055263352, 2, 189)	
↺↻	Consumer Record:(1495055263355, Hello Mom 1495055263355, 2, 190)	
↻↺	Consumer Record:(1495055263365, Hello Mom 1495055263365, 2, 191)	
⬇	Consumer Record:(1495055263368, Hello Mom 1495055263368, 2, 192)	
⬇	New ConsumerRecords par count 2 count 4	
⬇	Consumer Record:(1495055263340, Hello Mom 1495055263340, 0, 175)	
⬇	Consumer Record:(1495055263362, Hello Mom 1495055263362, 0, 176)	
⬇	Consumer Record:(1495055263335, Hello Mom 1495055263335, 4, 176)	
⬇	Consumer Record:(1495055263358, Hello Mom 1495055263358, 4, 177)	
⬇	New ConsumerRecords par count 2 count 5	
⬇	Consumer Record:(1495055263338, Hello Mom 1495055263338, 1, 219)	
⬇	Consumer Record:(1495055263341, Hello Mom 1495055263341, 1, 220)	
⬇	Consumer Record:(1495055263339, Hello Mom 1495055263339, 3, 185)	
⬇	Consumer Record:(1495055263354, Hello Mom 1495055263354, 3, 186)	
⬇	Consumer Record:(1495055263371, Hello Mom 1495055263371, 3, 187)	
⬇	New ConsumerRecords par count 1 count 5	
⬇	Consumer Record:(1495055263351, Hello Mom 1495055263351, 1, 221)	
⬇	Consumer Record:(1495055263353, Hello Mom 1495055263353, 1, 222)	
⬇	Consumer Record:(1495055263356, Hello Mom 1495055263356, 1, 223)	
⬇	Consumer Record:(1495055263366, Hello Mom 1495055263366, 1, 224)	
⬇	Consumer Record:(1495055263367, Hello Mom 1495055263367, 1, 225)	

# Now run it again but..

- ❖ Run the consumers and producer again
- ❖ Wait 30 seconds
- ❖ While the producer is running kill one of the consumers and see the records go to the other consumers
- ❖ Now leave just one consumer running, all of the messages should go to the remaining consumer
  - ❖ Now change consumer batch size to 500 `props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 500)`
  - ❖ and run it again



# Output form batch size 500

Tos

```
New ConsumerRecords par count 7 count 28
Consumer Record:(1495056566578, Hello Mom 1495056566578, 5, 266)
Consumer Record:(1495056566591, Hello Mom 1495056566591, 5, 267)
Consumer Record:(1495056566603, Hello Mom 1495056566603, 5, 268)
Consumer Record:(1495056566605, Hello Mom 1495056566605, 5, 269)
Consumer Record:(1495056566581, Hello Mom 1495056566581, 8, 238)
Consumer Record:(1495056566592, Hello Mom 1495056566592, 8, 239)
Consumer Record:(1495056566597, Hello Mom 1495056566597, 8, 240)
Consumer Record:(1495056566598, Hello Mom 1495056566598, 8, 241)
Consumer Record:(1495056566607, Hello Mom 1495056566607, 8, 242)
Consumer Record:(1495056566609, Hello Mom 1495056566609, 8, 243)
Consumer Record:(1495056566625, Hello Mom 1495056566625, 8, 244)
Consumer Record:(1495056566626, Hello Mom 1495056566626, 8, 245)
Consumer Record:(1495056566584, Hello Mom 1495056566584, 10, 253)
Consumer Record:(1495056566585, Hello Mom 1495056566585, 10, 254)
Consumer Record:(1495056566594, Hello Mom 1495056566594, 10, 255)
Consumer Record:(1495056566601, Hello Mom 1495056566601, 10, 256)
Consumer Record:(1495056566618, Hello Mom 1495056566618, 10, 257)
Consumer Record:(1495056566619, Hello Mom 1495056566619, 10, 258)
Consumer Record:(1495056566593, Hello Mom 1495056566593, 11, 230)
Consumer Record:(1495056566600, Hello Mom 1495056566600, 11, 231)
Consumer Record:(1495056566586, Hello Mom 1495056566586, 2, 265)
Consumer Record:(1495056566596, Hello Mom 1495056566596, 1, 296)
Consumer Record:(1495056566624, Hello Mom 1495056566624, 1, 297)
Consumer Record:(1495056566595, Hello Mom 1495056566595, 4, 242)
Consumer Record:(1495056566604, Hello Mom 1495056566604, 4, 243)
Consumer Record:(1495056566610, Hello Mom 1495056566610, 4, 244)
Consumer Record:(1495056566622, Hello Mom 1495056566622, 4, 245)
Consumer Record:(1495056566623, Hello Mom 1495056566623, 4, 246)
New ConsumerRecords par count 5 count 22
Consumer Record:(1495056566599, Hello Mom 1495056566599, 6, 236)
Consumer Record:(1495056566613, Hello Mom 1495056566613, 6, 237)
Consumer Record:(1495056566620, Hello Mom 1495056566620, 6, 238)
Consumer Record:(1495056566579, Hello Mom 1495056566579, 9, 267)
Consumer Record:(1495056566583, Hello Mom 1495056566583, 9, 268)
```

- ❖ Created simple example that creates a ***Kafka Consumer*** to consume messages from our ***Kafka Producer***
- ❖ Used the replicated ***Kafka topic*** from first example
- ❖ ***Created Consumer*** that uses topic to receive messages
- ❖ ***Processed records*** from Kafka with ***Consumer***
- ❖ ***Consumers*** in same group divide up and share partitions
- ❖ ***Each Consumer groups gets a copy of the same data (really has a unique set of offset partition pairs per Consumer Group)***



- ❖ How did we demonstrate Consumers in a Consumer Group dividing up topic partitions and sharing them?
- ❖ How did we demonstrate Consumers in different Consumer Groups each getting their own offsets?
- ❖ How many records does poll get?
- ❖ Does a call to poll ever get records from two different partitions?

## **Lab : Consumer – Java API**