Hands-on Lab: Message Keys and Offset

After completing this lab, you will be able to:

- Use message keys to keep message streams sorted in their original publication state/order

- Use consumer offset to control and track message sequential positions in topic partitions

Lab environment setup and preparation

- Download Kafka, by running the following command:
- Extract kafka from the zip file by running the following command:
- This creates a new directory ‘kafka_2.12-2.8.0’ in the current directory.

```
theia@theiadocker-craigtrupp8:/home/project$ wget
https://archive.apache.org/dist/kafka/2.8.0/kafka_2.12-2.8.0.tgz
--2023-09-24 15:02:39--
https://archive.apache.org/dist/kafka/2.8.0/kafka_2.12-2.8.0.tgz
Resolving archive.apache.org (archive.apache.org)... 65.108.204.189,
2a01:4f9:1a:a084::2
Connecting to archive.apache.org
(archive.apache.org)|65.108.204.189|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 71542357 (68M) [application/x-gzip]
Saving to: 'kafka_2.12-2.8.0.tgz'

kafka_2.12-2.8.0.tgz
100%[=================================================>]  68.23M  19.6MB/s
in 3.5s

2023-09-24 15:02:43 (19.6 MB/s) - 'kafka_2.12-2.8.0.tgz' saved
[71542357/71542357]

theia@theiadocker-craigtrupp8:/home/project$ tar -xzf kafka_2.12-2.8.0.tgz
theia@theiadocker-craigtrupp8:/home/project$ pwd
/home/project
theia@theiadocker-craigtrupp8:/home/project$ ls
kafka_2.12-2.8.0  kafka_2.12-2.8.0.tgz
```

Start Zookeeper

```
theia@theiadocker-craigtrupp8:/home/project$ cd kafka_2.12-2.8.0
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/zookeeper-server-start.sh config/zookeeper.properties
```

Start Apache Kafka Server

```
cd kafka_2.12-2.8.0
bin/kafka-server-start.sh config/server.properties
```

Create a topic and producer for processing bank ATM transactions

Next, we will be creating a bankbranch topic to process the messages that come from the ATM machines of bank branches.

Suppose the messages come from the ATM in the form of a simple JSON object, including an ATM id and a transaction id like the following example:

Suppose the messages come from the ATM in the form of a simple JSON object, including an ATM id and a transaction id like the following example:

```
{"atmid": 1, "transid": 100}
```

To process the ATM messages, let's first create a new topic called `bankbranch`.

- Start a new terminal and go to the extracted `Kafka` folder:

```
cd kafka_2.12-2.8.0
```

- Create a new topic using the `--topic` argument with the name `bankbranch`. In order to simplify the topic configuration and better explain how message key and consumer offset work, here we specify `--partitions 2` argument to create two partitions for this topic.
  You may try other `partitions` settings for this topic if you are interested in comparing the difference.

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic bankbr
```

Now let's list all the topics to see if `bankbranch` has been created successfully.

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
```

We can also use the `--describe` command to check the details of the topic `bankbranch`

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic bank
```

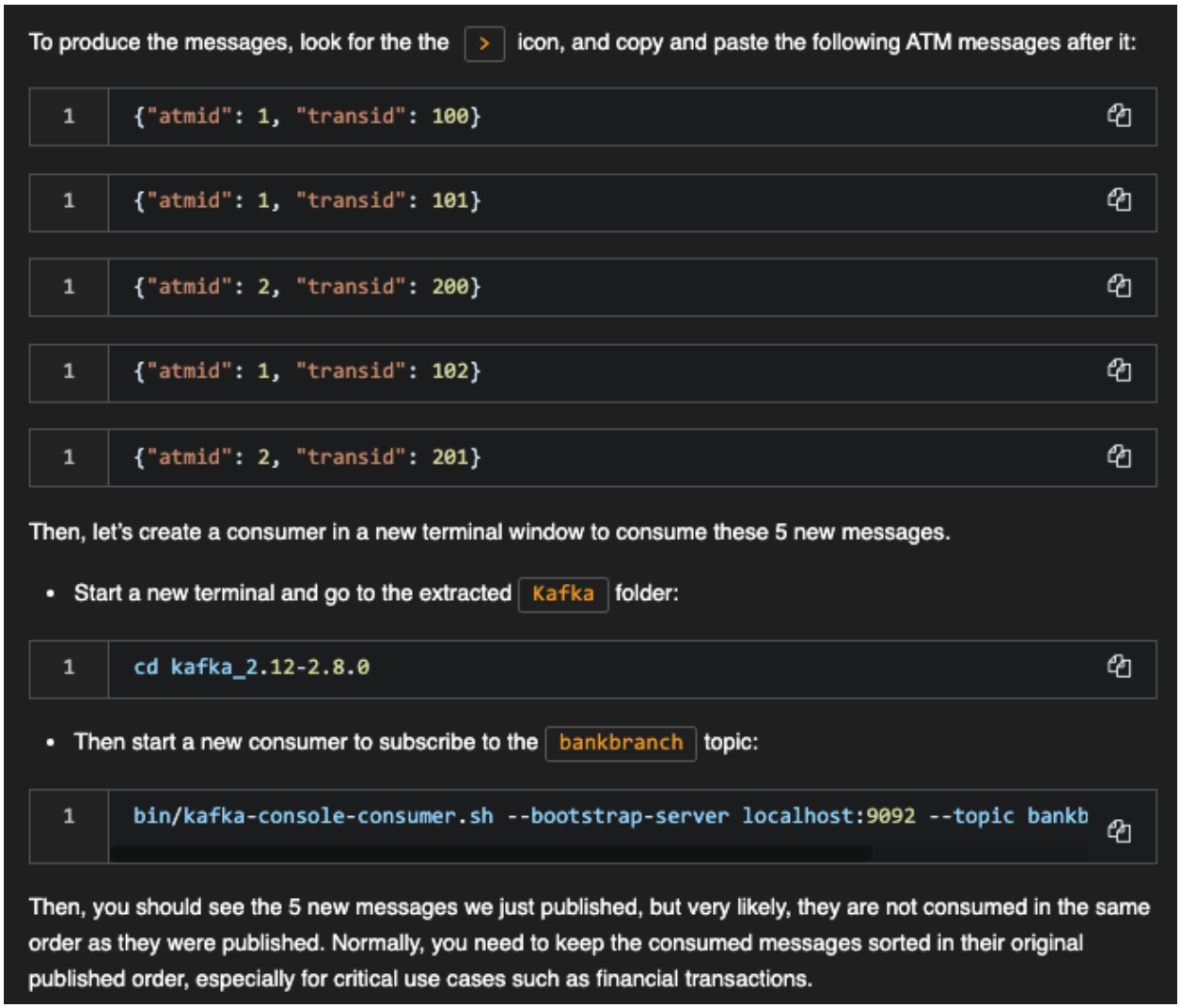
and you can see `bankbranch` has two partitions `Partition 0` and `Partition 1`. If no message keys are specified, messages will be published to these two partitions in an alternating sequence, like this:

`Partition 0` -> `Partition 1` -> `Partition 0` -> `Partition 1` ...

Next, we can create a producer to publish some ATM transaction messages.

- Stay in the same terminal window with the topic details, then create a producer for topic `bankbranch`

```
bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic bankb
```



- Start of Terminal Logging

```
theia@theiadocker-craigtrupp8:/home/project$ cd kafka_2.12-2.8.0
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-topics.sh --bootstrap-server localhost:9092 --list^C
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic
bankbranch
Created topic bankbranch.
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-topics.sh --bootstrap-server localhost:9092
--list__consumer_offsets
bankbranch
```

```
news
weather
```

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic
bankbranch
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-topics.sh --bootstrap-server localhost:9092
--list__consumer_offsets
news
weather
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic
bankbranch  --partitions 2
Created topic bankbranch.
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
__consumer_offsets
bankbranch
news
weather
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic
bankbranch
Topic: bankbranch       TopicId: XUzTOApgRC-Yq7-amAc1NA PartitionCount: 2
ReplicationFactor: 1    Configs: segment.bytes=1073741824
        Topic: bankbranch       Partition: 0    Leader: 0       Replicas: 0
Isr: 0
        Topic: bankbranch       Partition: 1    Leader: 0       Replicas: 0
Isr: 0
```

- Had to delete the first topic (see delet command above as didn't originally create with 2 partitions as detailed

- Create Producer and publish some JSON style messages

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic
bankbranch
>{"atmid": 1, "transid": 100}
>{"atmid": 1, "transid": 101}
```

```
>{"atmid": 2, "transid": 200}
>{"atmid": 1, "transid": 102}
>{"atmid": 2, "transid": 201}
```

- In a new terminal, go to the extracted kafka folder and start a new consumer to subscribe to the bankbranch topic

- Then, you should see the 5 new messages we just published, but very likely, they are not consumed in the same order as they were published. Normally, you need to keep the consumed messages sorted in their original published order, especially for critical use cases such as financial transactions.

```
theia@theiadocker-craigtrupp8:/home/project$ cd kafka_2.12-2.8.0
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
bankbranch --from-beginning
{"atmid": 1, "transid": 100}
{"atmid": 2, "transid": 200}
{"atmid": 2, "transid": 201}
{"atmid": 1, "transid": 101}
{"atmid": 1, "transid": 102}
```

Produce and consume with message keys
In this step, you will be using message keys to ensure that messages with the same key will be consumed in the same order as they were published. In the backend, messages with the same key will be published into the same partition and will always be consumed by the same consumer. As such, the original publication order is kept in the consumer side.

ASt this point, you should have the following four terminals open in Cloud IDE:

Zookeeper terminal
Kafka Server terminal
Producer terminal
Consumer terminal

In the next steps, you will be frequently switching among these terminals.

First, go to the consumer terminal and stop the consumer using Ctrl + C (Windows)
or Command + . (Mac).

Then, switch to the Producer terminal and stop the previous producer.

Ok, we can now start a new producer and consumer, this time using message keys. You can start a new producer with the following message key commands:

--property parse.key=true to make the producer parse message keys

--property key.separator=: define the key separator to be the : character,
so our message with key now looks like the following key-value pair example:

1:{"atmid": 1, "transid": 102}.
Here the message key is 1, which also corresponds to the ATM id, and the value is the transaction JSON object, {"atmid": 1, "transid": 102}.

- Start a new producer with message key enabled:

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-console-producer.sh --bootstrap-server localhos092 --topic
bankbranch --property parse.key=true --property key.separator=:
>
```

```
>1:{"atmid": 1, "transid": 102}
>1:{"atmid": 1, "transid": 103}
>2:{"atmid": 2, "transid": 202}
>2:{"atmid": 2, "transid": 203}
>1:{"atmid": 1, "transid": 104}
```

- Next, switch to the consumer terminal again, and start a new consumer with --property print.key=true --property key.separator=: arguments to print the keys

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
bankbranch --from-beginning --property print.key=true --property
key.separator=:
null:{"atmid": 1, "transid": 100}
null:{"atmid": 2, "transid": 200}
null:{"atmid": 2, "transid": 201}
1:{"atmid": 1, "transid": 102}
```

```
1:{"atmid": 1, "transid": 103}
1:{"atmid": 1, "transid": 104}
null:{"atmid": 1, "transid": 101}
null:{"atmid": 1, "transid": 102}
2:{"atmid": 2, "transid": 202}
2:{"atmid": 2, "transid": 203}
```

Now, you should see that messages that have the same key are being consumed in the same order (e.g., trans102 -> trans103 -> trans104) as they were published.

This is because each topic partition maintains its own message queue, and new messages are enqueued (appended to the end of the queue)
as they get published to the partition. Once consumed, the earliest messages will be dequeued and nno longer be available for consumption.

Recall that with two partitions and no message keys specified, the transaction messages were published to the two partitions
in rotation:

Partition 0: [{"atmid": 1, "transid": 102}, {"atmid": 2, "transid": 202}, {"atmid": 1, "transid": 104}]
Partition 1: [{"atmid": 1, "transid": 103}, {"atmid": 2, "transid": 203}]
As you can see, the transaction messages from atm1 and atm2 got scattered across both partitions. It would be difficult
to unravel this and consume messages from one ATM with the same order as they were published.

However, with message key specified as the atmid value, the messages from the two ATMs will look like the following:

Partition 0: [{"atmid": 1, "transid": 102}, {"atmid": 1, "transid": 103}, {"atmid": 1, "transid": 104}]
Partition 1: [{"atmid": 2, "transid": 202}, {"atmid": 2, "transid": 203}]
Messages with the same key will always be published to the same partition, so that their published order will be preserved within the message queue of each partition.

As such, we can keep the states or orders of the transactions for each ATM.

Consumer Offset
Topic partitions keeps published messages in a sequence, like a list.
Message offset indicates a message's position in the sequence. For example,
the offset of an empty Partition 0 of bankbranch is 0, and if you publish the first message

to the partition, its offset will be 1.

By using offsets in the consumer, you can specify the starting position for message consumption, such as from the beginning to retrieve all messages, or from some later point to retrieve only the latest messages.

Consumer Group
In addition, we normally group related consumers together as a consumer group.
For example, we may want to create a consumer for each ATM in the bank and manage all ATM related consumers
together in a group.

So let's see how to create a consumer group, which is actually very easy with the --group argument.

In the consumer terminal, stop the previous consumer if it is still running.

Run the following command to create a new consumer within a consumer group called atm-app:

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
bankbranch --group atm-app
```

After the consumer within the atm-app consumer group is started, you should not expect any messages to be consumed.
This is because the offsets for both partitions have already reached to the end.
In other words, all messages have already been consumed, and therefore dequeued, by previous consumers.

You can verify that by checking consumer group details.

Stop the consumer.

Show the details of the consumer group atm-app:

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe
--group atm-app
```

```
Consumer group 'atm-app' has no active members.

GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET
LAG             CONSUMER-ID     HOST            CLIENT-ID
atm-app         bankbranch      1          6               6
0               -               -               -
atm-app         bankbranch      0          4               4
0               -               -               -
```

Now you should see the offset information for the topic bankbranch:

Recall that we have published 10 messages in total, and we can see the CURRENT-OFFSET column of partition 1 is 6
and CURRENT-OFFSET of partition 0 is 4, and they add up to 10 messages.

The LOG-END-OFFSETcolumn indicates the last offset or the end of the sequence, which is 6 for partition 1 and 4 for
partition 0. Thus, both partitions have reached the end of their queues and no more messages are available for consumption.

Meanwhile, you can check the LAG column which represents the count of unconsumed messages for each partition.
Currently it is 0 for all partitions, as expected.

Now, let's produce more messages and see how the offsets change.

Switch to the previous producer terminal, and publish two more messages

```
>1:{"atmid": 1, "transid": 105}
>2:{"atmid": 2, "transid": 204}
```

and let's switch back to the consumer terminal and check the consumer group details again:

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe
--group atm-app

Consumer group 'atm-app' has no active members.

GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET
```

```
LAG             CONSUMER-ID     HOST            CLIENT-ID
atm-app         bankbranch      1          6               7
1               -               -               -
atm-app         bankbranch      0          4               5
1               -               -
```

You should see that both offsets have been increased by 1, and the LAG columns for both partitions have become
1. It means we have 1 new message for each partition to be consumed.

Let's start the consumer again and see whether the two new messages will be consumed.

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
bankbranch --group atm-app
{"atmid": 1, "transid": 105}
{"atmid": 2, "transid": 204}
```

OK, now both partitions have reached the end once again. But what if you want to consume the messages again from the beginning?

We can do that via resetting offset in the next step.

Reset offset
We can reset the index with the --reset-offsets argument.

First let's try resetting the offset to the earliest position (beginning) using --reset-offsets --to-earliest.

Stop the previous consumer if it is still running, and run the following command to reset the offset:

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092  --topic
bankbranch --group atm-app --reset-offsets --to-earliest --execute

GROUP                          TOPIC                          PARTITION
NEW-OFFSET
atm-app                        bankbranch                     0          0
atm-app                        bankbranch                     1          0
```

Now the offsets have been set to 0 (the beginning).

- Start the consumer again:

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
bankbranch --group atm-app
{"atmid": 1, "transid": 100}
{"atmid": 2, "transid": 200}
{"atmid": 2, "transid": 201}
{"atmid": 1, "transid": 102}
{"atmid": 1, "transid": 103}
{"atmid": 1, "transid": 104}
{"atmid": 1, "transid": 105}
{"atmid": 1, "transid": 101}
{"atmid": 1, "transid": 102}
{"atmid": 2, "transid": 202}
{"atmid": 2, "transid": 203}
{"atmid": 2, "transid": 204}
```

You should see that all 12 messages are consumed and that all offsets have reached the partition ends again.

In fact, you can reset the offset to any position. For example, let's reset the offset so that we only consume the last two messages.

Stop the previous consumer

Shift the offset to left by 2 using --reset-offsets --shift-by -2:

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092  --topic
bankbranch --group atm-app --reset-offsets --shift-by -2 --execute

GROUP                          TOPIC                          PARTITION
NEW-OFFSET
atm-app                        bankbranch                     0          3
atm-app                        bankbranch                     1          5
```

If you run the consumer again, you should see that we consumed 4 messages, 2 for each partition:

```
theia@theiadocker-craigtrupp8:/home/project/kafka_2.12-2.8.0$
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
bankbranch --group atm-app
{"atmid": 1, "transid": 104}
{"atmid": 1, "transid": 105}
{"atmid": 2, "transid": 203}
{"atmid": 2, "transid": 204}
```