Kafka Message key and offset

Estimated time needed: **40** minutes

Objectives

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

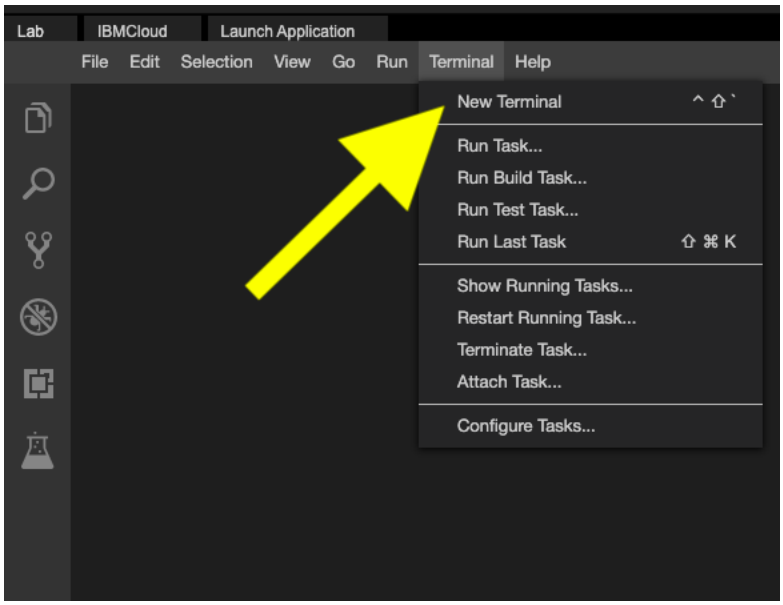
Important notice about this lab environment

Please be aware that sessions for this lab environment are not persistent.
A new environment is created for you every time you connect to this lab.
Any data you may have saved in an earlier session will get lost.
To avoid losing your data, please plan to complete these labs in a single session.

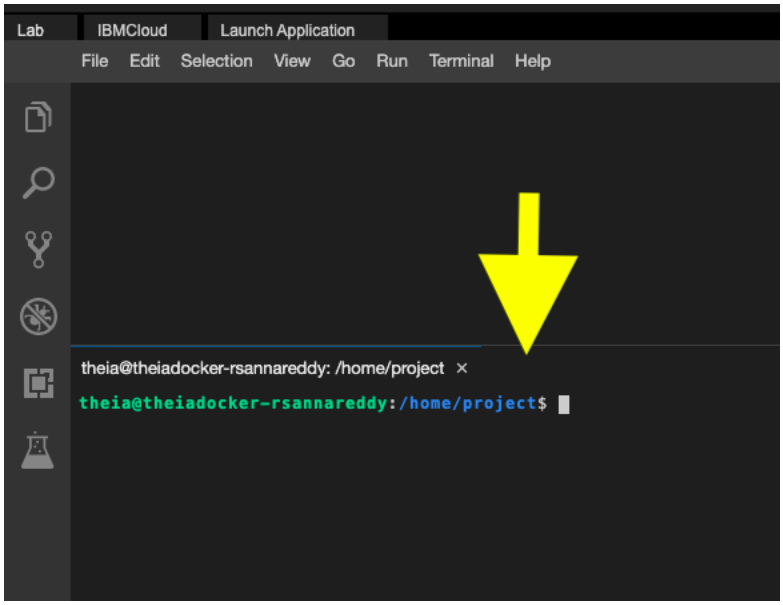
Lab environment setup and preparation

Download and Extract Apache Kafka

Open a new terminal, by clicking on the menu bar and selecting **Terminal**->**New Terminal**, as shown in the following image.



This will open a new terminal at the bottom of the screen.



Run the following commands on the newly opened terminal. (You can copy the code by clicking on the little copy button on the lower-right of the following codeblock and then paste it, wherever you wish.)

Download Kafka, by running the following command:

```
1. 1
```

```
1. wget https://archive.apache.org/dist/kafka/2.8.0/kafka_2.12-2.8.0.tgz
```

Copied!

Extract kafka from the zip file by running the following command:

```
1. 1
```

```
1. tar -xzf kafka_2.12-2.8.0.tgz
```

Copied!

This creates a new directory 'kafka_2.12-2.8.0' in the current directory.

Start ZooKeeper

ZooKeeper is required for Kafka to work. Start the ZooKeeper server.

```
1. 1
2. 2
```

```
1. cd kafka_2.12-2.8.0
2. bin/zookeeper-server-start.sh config/zookeeper.properties
```

Copied!

When ZooKeeper starts you should see an output like this:

```
theia@theiadocker-rsannareddy:/home/project/kafka_2.12-2.8.0$ bin/zookeeper-server-start.sh config/zookeeper.properties
JVMJ9VM007W Command-line option unrecognised: -Xlog:gc*:file=/home/project/kafka_2.12-2.8.0/bin/../logs/zookeeper-gc.log:time
,tags:filecount=10,filesize=100M
[2021-08-24 13:19:41,160] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.Qu
orumPeerConfig)
[2021-08-24 13:19:41,163] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.
zookeeper.server.quorum.QuorumPeerConfig)
[2021-08-24 13:19:41,178] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-08-24 13:19:41,178] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-08-24 13:19:41,183] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-08-24 13:19:41,183] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-08-24 13:19:41,183] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-08-24 13:19:41,183] WARN Either no config or no quorum defined in config, running  in standalone mode (org.apache.zooke
eper.server.quorum.QuorumPeerMain)
[2021-08-24 13:19:41,190] INFO Log4j 1.2 jmx support found and enabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2021-08-24 13:19:41,217] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.Qu
orumPeerConfig)
[2021-08-24 13:19:41,217] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.
zookeeper.server.quorum.QuorumPeerConfig)
```

You can be sure it has started when you see an output like this:

```
[2021-08-24 13:19:41,253] INFO Server environment:os.memory.max=512MB (org.apache.zookeeper.server.ZooKeeperServer)
[2021-08-24 13:19:41,253] INFO Server environment:os.memory.total=512MB (org.apache.zookeeper.server.ZooKeeperServer)
[2021-08-24 13:19:41,256] INFO minSessionTimeout set to 6000 (org.apache.zookeeper.server.ZooKeeperServer)
[2021-08-24 13:19:41,256] INFO maxSessionTimeout set to 60000 (org.apache.zookeeper.server.ZooKeeperServer)
[2021-08-24 13:19:41,257] INFO Created server with tickTime 3000 minSessionTimeout 6000 maxSessionTimeout 60000 datadir /tmp/
zookeeper/version-2 snapdir /tmp/zookeeper/version-2 (org.apache.zookeeper.server.ZooKeeperServer)
[2021-08-24 13:19:41,275] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory (org.apach
e.zookeeper.server.ServerCnxnFactory)
[2021-08-24 13:19:41,281] INFO Configuring NIO connection handler with 10s sessionless connection timeout, 1 selector thread(
s), 4 worker threads, and 64 kB direct buffers. (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2021-08-24 13:19:41,290] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2021-08-24 13:19:41,314] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2021-08-24 13:19:41,321] INFO Snapshotting: 0x0 to /tmp/zookeeper/version-2/snapshot.0 (org.apache.zookeeper.server.persiste
nce.FileTxnSnapLog)
[2021-08-24 13:19:41,324] INFO Snapshotting: 0x0 to /tmp/zookeeper/version-2/snapshot.0 (org.apache.zookeeper.server.persiste
nce.FileTxnSnapLog)
[2021-08-24 13:19:41,345] INFO PrepRequestProcessor (sid:0) started, reconfigEnabled=false (org.apache.zookeeper.server.PrepR
equestProcessor)
[2021-08-24 13:19:41,355] INFO Using checkIntervalMs=60000 maxPerMinute=10000 (org.apache.zookeeper.server.ContainerManager)
```

ZooKeeper, as of this version, is required for Kafka to work.
ZooKeeper is responsible for the overall management of a Kafka cluster.
It monitors the Kafka brokers and notifies Kafka if any broker or partition goes down,
or if a new broker or partition comes up.

Start Apache Kafka Server

Start a new terminal.

Run the following command to start Kafka server

```
1. 1
2. 2
```

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

Copied!

When Kafka starts, you should see an output like this:

```
theia@theiadocker-rsannareddy:/home/project$ cd kafka_2.12-2.8.0
theia@theiadocker-rsannareddy:/home/project/kafka_2.12-2.8.0$ bin/kafka-server-start.sh config/server.properties
JVMJ9VM007W Command-line option unrecognised: -Xlog:gc*:file=/home/project/kafka_2.12-2.8.0/bin/../logs/kafkaServer-gc.log:t
ime,tags:filecount=10,filesize=100M
[2021-08-24 13:23:34,367] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2021-08-24 13:23:34,856] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS re
negotiation (org.apache.zookeeper.common.X509Util)
[2021-08-24 13:23:34,975] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHan
dler)
[2021-08-24 13:23:34,984] INFO starting (kafka.server.KafkaServer)
[2021-08-24 13:23:34,985] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2021-08-24 13:23:35,026] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper
.ZooKeeperClient)
[2021-08-24 13:23:35,037] INFO Client environment:zookeeper.version=3.5.9-83df9301aa5c2a5d284a9940177808c01bc35cef, built on
 01/06/2021 20:03 GMT (org.apache.zookeeper.ZooKeeper)
[2021-08-24 13:23:35,037] INFO Client environment:host.name=theiadocker-rsannareddy (org.apache.zookeeper.ZooKeeper)
[2021-08-24 13:23:35,037] INFO Client environment:java.version=11.0.11 (org.apache.zookeeper.ZooKeeper)
[2021-08-24 13:23:35,037] INFO Client environment:java.vendor=AdoptOpenJDK (org.apache.zookeeper.ZooKeeper)
[2021-08-24 13:23:35,037] INFO Client environment:java.home=/usr/lib/jvm/java-jdk-11.0.11+9 (org.apache.zookeeper.ZooKeeper)
[2021-08-24 13:23:35,037] INFO Client environment:java.class.path=/usr/lib/jvm/java-jdk-11.0.11+9/lib:/home/project:/home/pr
oject/kafka_2.12-2.8.0/bin/../libs/activation-1.1.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/aopalliance-repackaged-2.
6.1.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/argparse4j-0.7.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/audience-
annotations-0.5.0.jar:/home/project/kafka_2.12-2.8.0/bin/../libs/commons-cli-1.4.jar:/home/project/kafka_2.12-2.8.0/bin/../l
```

You can be sure it has started when you see an output like this:

```
[2021-08-24 13:23:37,628] INFO [Transaction Marker Channel Manager 0]: Starting (kafka.coordinator.transaction.TransactionMa
rkerChannelManager)
[2021-08-24 13:23:37,628] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.TransactionCoo
rdinator)
[2021-08-24 13:23:37,645] INFO Updated cache from existing <empty> to latest FinalizedFeaturesAndEpoch(features=Features{},
epoch=0). (kafka.server.FinalizedFeatureCache)
[2021-08-24 13:23:37,701] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOpera
tionReaper)
[2021-08-24 13:23:37,764] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListen
er$ChangeEventProcessThread)
[2021-08-24 13:23:37,825] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Starting socket server acceptors and processo
rs (kafka.network.SocketServer)
[2021-08-24 13:23:37,845] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Started data-plane acceptor and processor(s)
for endpoint : ListenerName(PLAINTEXT) (kafka.network.SocketServer)
[2021-08-24 13:23:37,846] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Started socket server acceptors and processor
s (kafka.network.SocketServer)
[2021-08-24 13:23:39,165] INFO Kafka version: 2.8.0 (org.apache.kafka.common.utils.AppInfoParser)
[2021-08-24 13:23:39,166] INFO Kafka commitId: ebb1d6e21cc92130 (org.apache.kafka.common.utils.AppInfoParser)
[2021-08-24 13:23:39,166] INFO Kafka startTimeMs: 1629811417846 (org.apache.kafka.common.utils.AppInfoParser)
[2021-08-24 13:23:39,192] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
[2021-08-24 13:23:39,384] INFO [broker-0-to-controller-send-thread]: Recorded new controller, from now on will use broker th
eiadocker-rsannareddy:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
```

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:

```
1. 1
```

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

Copied!

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:

```
1. 1
1. cd kafka_2.12-2.8.0
```

Copied!

- 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.

```
1. 1
1. bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic bankbranch  --partitions 2
```

Copied!

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

```
1. 1
1. bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
```

Copied!

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

```
1. 1
1. bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic bankbranch
```

Copied!

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`

```
1. 1
1. bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic bankbranch
```

Copied!

To produce the messages, look for the the `>` icon, and copy and paste the following ATM messages after it:

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

Copied!

```
1. 1
1. {"atmid": 1, "transid": 101}
```

Copied!

```
1. 1
1. {"atmid": 2, "transid": 200}
```

Copied!

```
1. 1
1. {"atmid": 1, "transid": 102}
```

Copied!

```
1. 1
1. {"atmid": 2, "transid": 201}
```

Copied!

Then, let's create a consumer in a new terminal window to consume these 5 new messages.

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

```
1. 1
1. cd kafka_2.12-2.8.0
```

Copied!

- Then start a new consumer to subscribe to the `bankbranch` topic:

```
1. 1
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --from-beginning
```

Copied!

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.

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:

```
1
1. bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic bankbranch --property parse.key=true --property key.separator=:
```

Copied!

- Once you see `>` symbol, you can start to produce the following messages, where you define each key to match the ATM id for each message:

```
1
1. 1:{"atmid": 1, "transid": 102}
```

Copied!

```
1
1. 1:{"atmid": 1, "transid": 103}
```

Copied!

```
1
1. 2:{"atmid": 2, "transid": 202}
```

Copied!

```
1
1. 2:{"atmid": 2, "transid": 203}
```

Copied!

```
1
1. 1:{"atmid": 1, "transid": 104}
```

Copied!

- 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

```
1
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --from-beginning --property print.key=true --property key.separator=:
```

Copied!

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`:

```
1
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --group atm-app
```

Copied!

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`:

```
1. 1
```

```
1. bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group atm-app
```

Copied!

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

```
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               -               -               -
```

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-OFFSET`column 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. 1
```

```
1. 1:{"atmid": 1, "transid": 105}
```

Copied!

```
1. 1
```

```
1. 2:{"atmid": 2, "transid": 204}
```

Copied!

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

```
1. 1
```

```
1. bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group atm-app
```

Copied!

```
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.

```
1. 1
```

```
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --group atm-app
```

Copied!

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:

```
1. 1
```

```
1. bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092  --topic bankbranch --group atm-app --reset-offsets --to-earliest --execute
```

Copied!

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

- Start the consumer again:

```
1. 1
```

```
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --group atm-app
```

Copied!

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`:

```
1. 1
```

```
1. bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092  --topic bankbranch --group atm-app --reset-offsets --shift-by -2 --execute
```

Copied!

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

```
1. 1
```

```
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --group atm-app
```

Copied!

Summary

In this lab, you have learned how to include message keys in publication to keep their message states/order.
You have also learned how to reset the offset to control the message consumption starting point.

Authors

Yan Luo

Other Contributors

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-10-27	1.0	Yan Luo	Created initial version of the lab