Hands-on Lab: Kafka Message Keys and Offset

Estimated time needed: **40** minutes

Introduction

In this lab, you will keep the message streams added to the partitions in topics in a Kafka broker sorted in their original publication order and state using offset and groups.

Objectives

After completing this lab, you will be able to:

- Use message keys to keep message streams sorted in their original publication state and order
- Use consumer offset to control and track message sequential positions in topic partitions

About Skills Network Cloud IDE

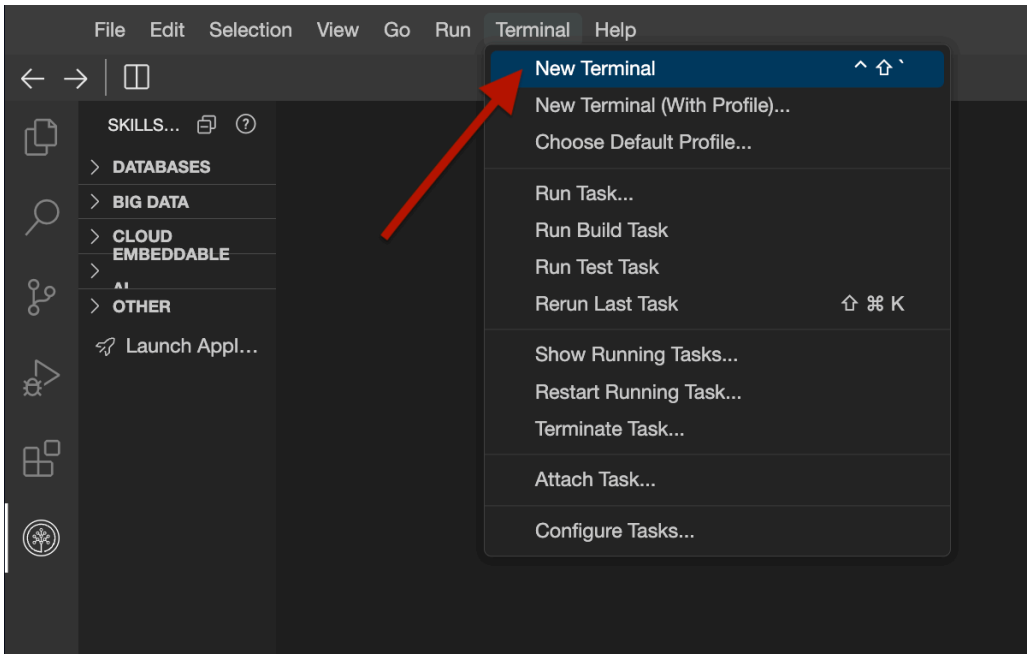
Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands-on labs related to courses and projects. Theia is an open-source IDE (Integrated Development Environment) that can be run on desktops or in the cloud. To complete this lab, we will use the Cloud IDE based on Theia running in a Docker container.

Important notice about this lab environment

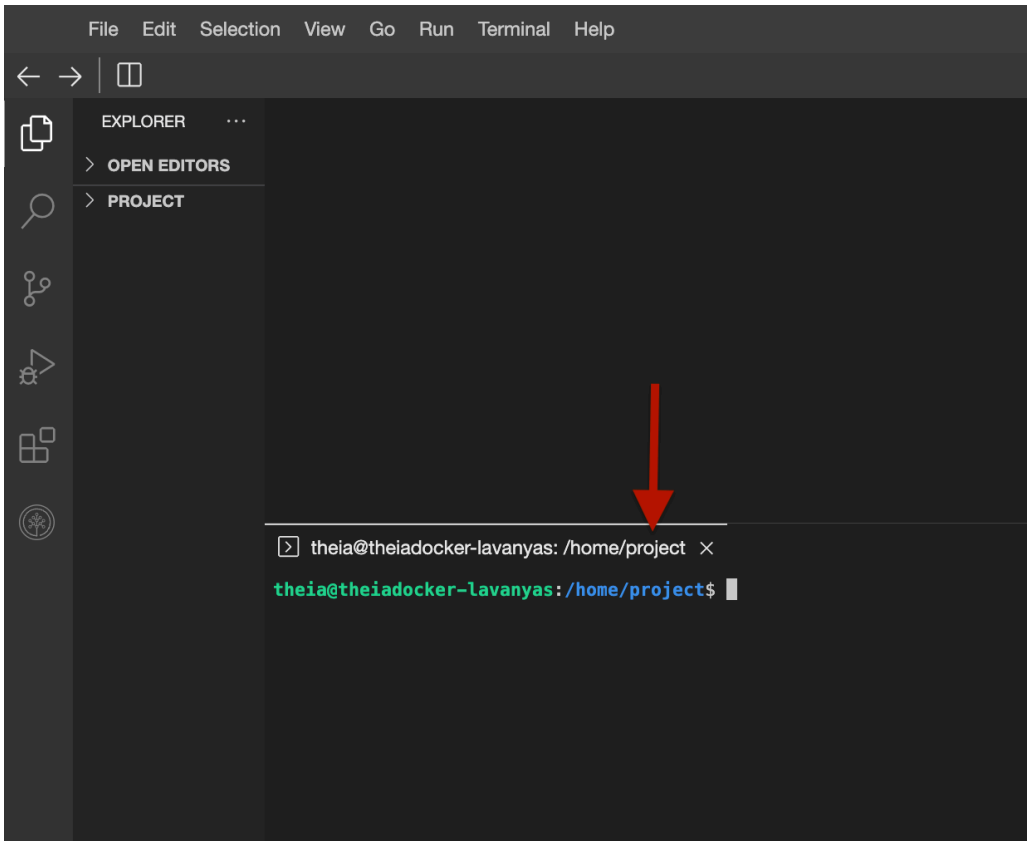
Please be aware that sessions for this lab environment are not persistent. A new environment is created for you every time you connect to this lab, and any data you may have saved in an earlier session will be lost. To avoid losing your data, please plan to complete these labs in a single session.

Exercise 1: Download and extract Kafka

1. Open a new terminal, by clicking the menu bar and selecting **Terminal**->**New Terminal**.



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



Run the commands below on the newly opened terminal.

Note: You can copy the code by clicking the little copy button on the bottom right of the code below and then paste it, wherever you wish.

2. Download Kafka by running the command below:

```
1
```

```
wget https://downloads.apache.org/kafka/3.7.0/kafka_2.12-3.7.0.tgz
```

Copied! Executed!

3. Extract Kafka from the zip file by running the command below.

```
1
```

```
tar -xzf kafka_2.12-3.7.0.tgz
```

Copied! Executed!

Note: This command creates a directory named `kafka_2.12-3.7.0` in the current directory.

Exercise 2: Configure KRaft and start server

1. Change to the `kafka_2.12-3.7.0` directory.

```
1
```

```
cd kafka_2.12-3.7.0
```

Copied! Executed!

2. Generate a Cluster UUID that will uniquely identify the Kafka cluster.

```
1
```

```
KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
```

Copied! Executed!

This cluster ID will be used by the KRaft controller.

3. KRaft requires the log directories to be configured. Run the following command to configure the log directories passing the cluster ID.

```
1
```

```
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c config/kraft/server.properties
```

Copied! Executed!

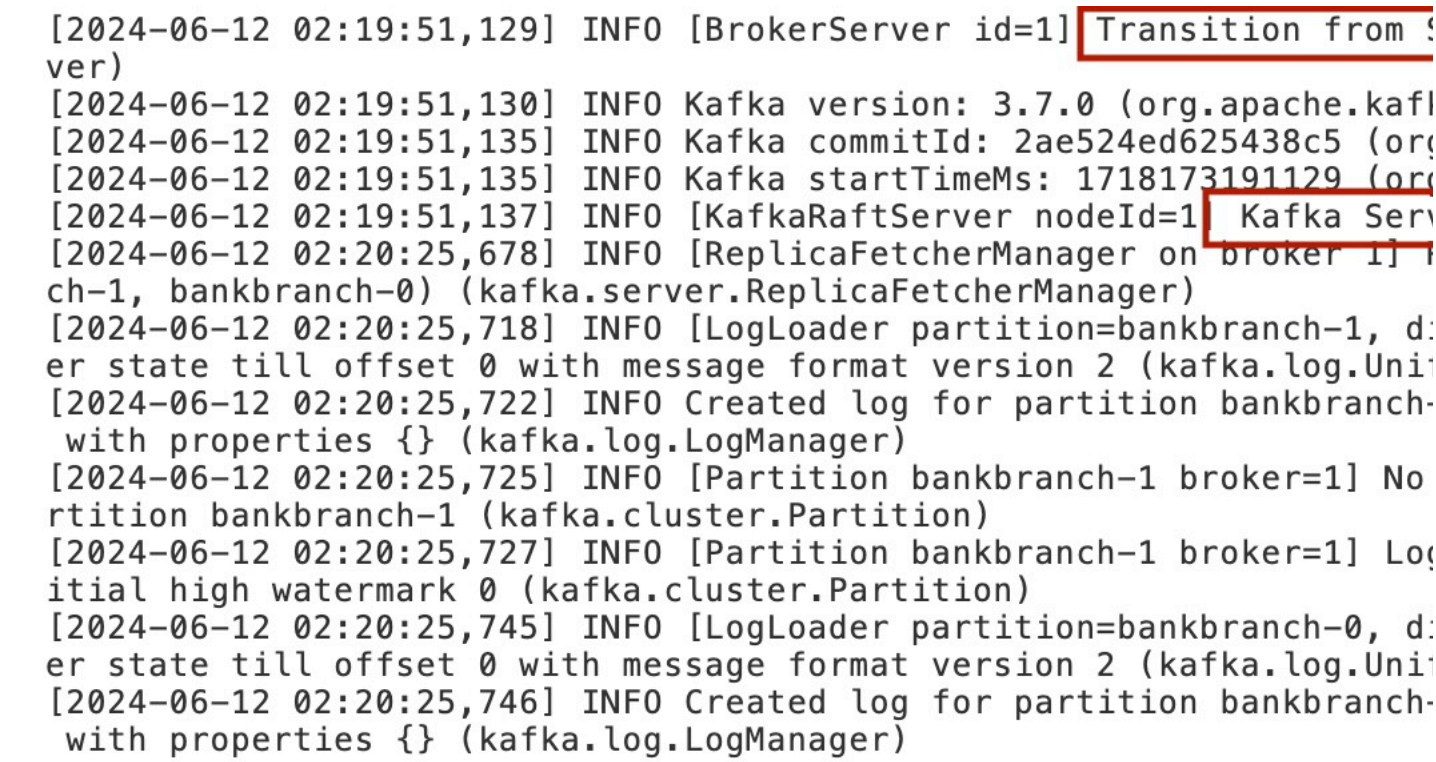
4. Now that KRaft is configured, you can start the Kafka server by running the following command.

```
1
```

```
bin/kafka-server-start.sh config/kraft/server.properties
```

Copied! Executed!

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



Exercise 3: Create a topic and producer for processing bank ATM transactions

Next, you will create a `bankbranch` topic to process messages from bank branch ATM machines.

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, you will first need to create a new topic called `bankbranch`.

1. Open a new terminal and change to the `kafka_2.12-3.7.0` directory.

```
1
cd kafka_2.12-3.7.0
```

Copied! Executed!

2. Create a new topic using the `--topic` argument with the name `bankbranch`. To simplify the topic configuration and better explain how message key and consumer offset work, you specify the `--partitions 2` argument to create two partitions for this topic. To compare the differences, you may try other `partitions` settings for this topic.

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

Copied! Executed!

3. List all topics to check if `bankbranch` has been created successfully.

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

Copied! Executed!

4. You can also use the `--describe` command to check the details of the topic `bankbranch`.

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

Copied! Executed!

You can view the `bankbranch` as 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, you can create a producer to publish ATM transaction messages.

5. Run the following command in the same terminal window with the topic details to create a producer for the topic `bankbranch`.

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

Copied! Executed!

6. To produce the messages, look for the `>` icon and add the following ATM messages after the icon:

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

Copied! Executed!

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

Copied! Executed!

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

Copied! Executed!

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

Copied! Executed!

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

Copied! Executed!

Then, create a consumer in a new terminal window to consume these five new messages.

7. Open a new terminal and change to the `kafka_2.12-3.7.0` directory.

```
1
1. cd kafka_2.12-3.7.0
```

Copied! Executed!

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

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

Copied! Executed!

You should be able to view the five new messages that you published. However, the messages may not be consumed in the same order as they were published. Typically, you will need to keep the consumed messages sorted in their original published order, especially for critical use cases, such as financial transactions.

Exercise 4: Produce and consume with message keys

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

At this point, you should have the following three terminals open in Cloud IDE:

- Kafka Server terminal
- Producer terminal
- Consumer terminal

In the next steps, you will frequently switch between 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.

You will now start a new producer and consumer using message keys. You can start a new producer with the following message key options:

- `--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}`.

1. Start a new producer with the message key enabled.

```
1
```

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

Copied! Executed!

2. Once you see the > 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. 1:{"atmid": 1, "transid": 103}
```

Copied! Executed!

```
1. 1
```

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

Copied! Executed!

```
1. 1
```

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

Copied! Executed!

```
1. 1
```

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

Copied! Executed!

```
1. 1
```

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

Copied! Executed!

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

```
1. 1
```

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

Copied! Executed!

Now, you should see that messages with the same key are being consumed in the same order (for example: `trans102 -> trans103 -> trans104`) as they were published.

Each topic partition maintains its message queue, and new messages are enqueued (appended to the end of the queue) as they are published to the partition. Once consumed, the earliest messages are dequeued and no longer 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": 100}, {"atmid": 2, "transid": 200}, {"atmid": 2, "transid": 201}]`
- Partition 1: `[{"atmid": 1, "transid": 101}, {"atmid": 1, "transid": 102}]`

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 in the same order as they were published.

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

- Partition 0: `[{"atmid": 1, "transid": 103}, {"atmid": 1, "transid": 104}, {"atmid": 1, "transid": 105}]`
- 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, you can keep the states or orders of the transactions for each ATM.

Exercise 5: Consumer offset

Topic partitions keep published messages in a sequence, such as 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`.

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, you normally group related consumers together as a consumer group.

For example, you 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.

1. In the consumer terminal, stop the previous consumer if it is still running.
2. Run the following command to create a new consumer within a consumer group called `atm-app`:

```
1. 1
```

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

Copied! Executed!

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 the end. In other words, previous consumers have already consumed all messages and therefore queued them.

You can verify that by checking consumer group details.

3. Stop the consumer.
4. Show the details of the consumer group `atm-app`:

```
1
```

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

Copied! Executed!

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

Recall that you have published `10` messages in total, and you can see the `CURRENT-OFFSET` column of partition 1 and partition 0 add up to 10 messages.

The `LOG-END-OFFSET`column indicates the last offset or the end of the sequence. 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 number of unconsumed messages for each partition.

Currently, it is `0` for all partitions, as expected.

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

5. Switch to the previous producer terminal and publish two more messages:

```
1
```

```
1:{"atmid": 1, "transid": 106}
```

Copied! Executed!

```
1
```

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

Copied! Executed!

6. Switch back to the consumer terminal and check the consumer group details again.

```
1
```

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

Copied! Executed!

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

7. Start the consumer again and see whether the two new messages will be consumed.

```
1
```

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

Copied! Executed!

Both partitions have reached the end once again.

Reset offset

Next, let's look at how you can set the partitions to consume the messages again from the beginning through resetting offset.

You 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`.

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

```
1
```

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

Copied! Executed!

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

2. Start the consumer again:

```
1
```

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

Copied! Executed!

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

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

3. Stop the previous consumer.
4. Shift the offset to the left by two using `--reset-offsets --shift-by -2`:

```
1
```

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

Copied! Executed!

5. If you run the consumer again, you should see that you consumed four messages, 2 for each partition:

```
1
```

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

Copied! Executed!

6. Stop your producer, consumer and the Kafka server.

Authors

Lavanya T S

Other Contributors

Yan Luo