Red Hat
Training and Certification

Streams for Apache Kafka Break-Fix Lab

Red Hat Streams for Apache Kafka 2.7 BF4821L

Student Workbook
Edition 1

Streams for Apache Kafka Break-Fix Lab

Red Hat Streams for Apache Kafka 2.7 BF4821L

Edition 1

Publication date 20240909

Authors:	**Grega Bremec**
Course Architects:	**Grega Bremec**
Editors:	**Grega Bremec**

Table of Contents

Introduction

Streams for Apache Kafka Break-Fix Lab

This lesson is a collection of scenarios whereby antipatterns are demonstrated to cause erroneous behaviour. Students are guided (and encouraged to further research) the behaviour, establish root cause, and sanitise the issues.

Course Objectives

- Identify antipatterns in configuration and application development.
- Correct configuration, code, and behaviour to adhere to best practices.

Audience

- Application Developers
- Infrastructure Engineers

Prerequisites

- AD482 - Developing Event-driven Applications with Apache Kafka and Red Hat AMQ Streams

Chapter 1. Introduction to Break-Fix Labs

Goal

Configure the lab environment to support the break-fix session.

Sections

- Deploying Streams for Apache Kafka (Guided Exercise)

1.1. Guided Exercise Deploying Streams for Apache Kafka

Perform initial configuration of projects and artifacts.

Outcomes

- Ensure the lab environment is configured correctly.
- Test operation of the environment.

Instructions

1. Ensure prerequisites are on your system:
   - Java 17 (or 21) SDK
   - Apache Maven (the lab was tested to work with release 3.9)
   - Streams for Apache Kafka 2.7.0
   - Bourne-compatible shell and basic Unix command-line tools (such as `sort` and `diff`)
   - optionally, an IDE (such as Visual Studio Code), if you want to examine the code

   NOTE You can download Streams for Apache Kafka from https://developers.redhat.com/products/streams-for-apache-kafka/download/.

2. Clone the Git repository with source code and lab materials.
   1. Create a working directory, for example `labs`.

      ```
      $ mkdir labs
      $ cd labs
      ```

   2. Extract the Streams for Kafka ZIP file here.

      ```
      $ unzip -q ~/Downloads/amq-streams-2.7.0-bin.zip
      ```

      NOTE Use your own download directory here, it might be different than `~/Downloads` for you.

   3. Rename the directory to just `kafka` for easier use.

      ```
      $ mv kafka_2.13-3.7.0.redhat-00007 kafka
      ```

   4. Clone the Git repository into the working directory.

```
$ git clone https://github.com/benko/streams-bf-lab-materials/
Cloning into 'streams-bf-lab-materials'...
remote: Enumerating objects: 297, done.
remote: Counting objects: 100% (297/297), done.
remote: Compressing objects: 100% (113/113), done.
remote: Total 297 (delta 92), reused 291 (delta 86), pack-reused 0 (from 0)
Receiving objects: 100% (297/297), 36.70 KiB | 3.06 MiB/s, done.
Resolving deltas: 100% (92/92), done.
```

NOTE If you get a complaint from Git, and the directory looks empty, either remove it and use `git clone -b main`, or enter the directory and use `git checkout main` to switch to the `main` branch.

5. Copy the broker and Zookeeper properties from materials to working directory.

```
$ cp streams-bf-lab-materials/labs/broker* \
    streams-bf-lab-materials/labs/zookeeper.properties .
```

6. The contents of your working directory should now look like the below listing.

```
$ ls -l
total 32
-rw-r--r--@ 1 johndoe  staff  926 10 Sep 14:31 broker0.properties
-rw-r--r--@ 1 johndoe  staff  926 10 Sep 14:31 broker1.properties
-rw-r--r--@ 1 johndoe  staff  926 10 Sep 14:31 broker2.properties
drwxr-xr-x@ 9 johndoe  staff  288 10 Sep 14:36 kafka/
drwxr-xr-x@ 8 johndoe  staff  256 10 Sep 14:29 streams-bf-lab-materials/
-rw-r--r--@ 1 johndoe  staff  101 10 Sep 14:31 zookeeper.properties
```

3. Start the Kafka broker cluster, each service in a separate window/tab.

1. Start Zookeeper first.

```
$ ./kafka/bin/zookeper-server-start.sh zookeeper.properties
[2024-09-11 22:15:50,105] INFO Reading configuration from:
./zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-11 22:15:50,112] INFO clientPortAddress is 0.0.0.0:2181
(org.apache.zookeeper.server.quorum.QuorumPeerConfig)
...
```

2. Start each of the three brokers.

```
$ ./kafka/bin/kafka-server-start.sh broker0.properties
[2024-09-11 22:15:58,087] INFO Registered kafka:type=kafka.Log4jController MBean
(kafka.utils.Log4jControllerRegistration$)
```

```
[2024-09-11 22:15:58,338] INFO Setting -D
jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS
renegotiation (org.apache.zookeeper.common.X509Util)
[2024-09-11 22:15:58,416] INFO Registered signal handlers for TERM, INT, HUP
(org.apache.kafka.common.utils.LoggingSignalHandler)
[2024-09-11 22:15:58,420] INFO starting (kafka.server.KafkaServer)
...
```

```
$ ./kafka/bin/kafka-server-start.sh broker1.properties
[2024-09-11 22:15:58,087] INFO Registered kafka:type=kafka.Log4jController MBean
(kafka.utils.Log4jControllerRegistration$)
[2024-09-11 22:15:58,338] INFO Setting -D
jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS
renegotiation (org.apache.zookeeper.common.X509Util)
[2024-09-11 22:15:58,416] INFO Registered signal handlers for TERM, INT, HUP
(org.apache.kafka.common.utils.LoggingSignalHandler)
[2024-09-11 22:15:58,420] INFO starting (kafka.server.KafkaServer)
...
```

```
$ ./kafka/bin/kafka-server-start.sh broker2.properties
[2024-09-11 22:15:58,087] INFO Registered kafka:type=kafka.Log4jController MBean
(kafka.utils.Log4jControllerRegistration$)
[2024-09-11 22:15:58,338] INFO Setting -D
jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS
renegotiation (org.apache.zookeeper.common.X509Util)
[2024-09-11 22:15:58,416] INFO Registered signal handlers for TERM, INT, HUP
(org.apache.kafka.common.utils.LoggingSignalHandler)
[2024-09-11 22:15:58,420] INFO starting (kafka.server.KafkaServer)
...
```

NOTE When stopping the services, do it in reverse order (brokers first, Zookeeper last). Simply press Ctrl-C in the corresponding window. Alternatively, use the `kafka-server-stop.sh` script with the property file of the broker you want to stop.

3. Test communication. Request a list of topics in the broker cluster.

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

NOTE The command is expected to produce a blank line as its output - we have not created any topics yet.

This concludes the exercise.

Chapter 2. Kafka Producer and Consumer Clients

Goal

Learn how to use the producer and consumer applications. Familiarize yourself with the additional scripts.

Sections

- Getting to know the client apps and tools (and Guided Exercise)

2.1. Getting to know the client apps and tools

Objectives

- Learn about the various command-line options for the client apps.
- Familiarize yourself with the result processing scripts.

About the Producer and Consumer applications

Producer

The producer application is available in the `code/core-api-producer/` directory of the cloned Git repository.

You can run it by changing your working directory to that location, building it, and invoking Maven's `exec` plugin.

```
$ cd streams-bf-lab-materials/code/core-api-producer/

$ mvn clean compile
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-producer >-------------
[INFO] Building core-api-producer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.000 s
[INFO] Finished at: 2024-09-13T11:55:31+02:00
[INFO] ------------------------------------------------------------------------

$ mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-producer >-------------
[INFO] Building core-api-producer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ core-api-producer ---
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
...
```

When started, the producer application will send a specified number of messages to a topic of your choice in the broker cluster.

It will also create a log containing each record that was sent to the broker, which you can then compare with consumer's log for detection of lost or duplicate messages.

The producer supports the following application-specific system properties that affect its behaviour:

- `producer.topic`, the topic to send records to, defaults to `test-topic`
- `producer.num-rolls`, number of send cycles, defaults to `1`
- `producer.num-records-per-roll`, number of records per send cycle, defaults to `100`
- `producer.wait-after-roll`, the amount of time (ms) to wait after each cycle, defaults to `5000`; useful if you need to perform an action after x records, before resuming the sending
- `producer.wait-after-send`, the amount of time (ms) to wait after each send, defaults to `0`; useful to artificially slow down sending
- `producer.local-id`, when writing log files, the sequence to append to the base name, defaults to `-1` (meaning no suffix); useful for multiple producers (i.e. `producer-0.log`, `producer-1.log`, etc.)
- `producer.payload-trunc`, when starting up, whether to truncate the payload log, defaults to `false`

The following producer client options can also be controlled from the command line:

- `producer.acks`, wait for acknowledgments to sent batches, defaults to `all`
- `producer.max-inflight`, maximum unacknowledged batches of records, defaults to `5`
- `producer.idempotent`, idempotency setting, defaults to `true` if the above two are `all` and at most `5`, otherwise `false`
- `producer.batch`, the maximum number of messages in a batch, defaults to `16384`
- `producer.linger`, the amount of time (ms) to wait for more messages before sending the batch anyway, defaults to `0`
- `producer.retries`, the number of send retries when encountering an error, defaults to `2147483647`
- `producer.delivery-timeout`, overall delivery timeout (linger + retry backoff + request timeout), defaults to `120000`
- `producer.request-timeout`, timeout when waiting for a response from the broker, defaults to `30000`
- `producer.retry-backoff`, how long to wait (ms), initially, for a retry to be attempted, defaults to `100`
- `producer.retry-max`, maximum wait before a retry (ms), defaults to `1000`

If you want to set any of the above properties, simply set them on the command line using the `-D` option.

```
$ mvn exec:java -Dproducer.num-records-per-roll=50000 -Dproducer.acks=0
[INFO] Scanning for projects...
```

```
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-producer >-------------
[INFO] Building core-api-producer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ core-api-producer ---
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
...
```

Consumer

The consumer application is available in the `code/core-api-consumer/` directory.

You can run it by changing your working directory to that location, building it, and invoking Maven's `exec` plugin.

```
$ cd streams-bf-lab-materials/code/core-api-consumer/

$ mvn clean compile
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-consumer >-------------
[INFO] Building core-api-consumer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.142 s
[INFO] Finished at: 2024-09-13T15:45:36+02:00
[INFO] ------------------------------------------------------------------------

$ mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-consumer >-------------
[INFO] Building core-api-consumer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ core-api-consumer ---
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
...
```

When started, the consumer application will connect to a topic and start receiving messages from it.

It will also create a log containing each record that was received from the broker. The idea is that comparing producer and consumer payload logs will allow you to detect duplicate and/or lost messages.

The consumer recognises three specific message payloads that cause it to act upon message value:

- `quit`, cleanly shut down
- `crash`, raise a `RuntimeException` and exit abnormally
- `wait`, block by invoking `Thread.sleep()`, the amount depends on `consumer.wait-cmd-period` setting; see below

Information about these three messages is not written into the payload log.

The consumer supports the following application-specific properties that affect its behaviour:

- `consumer.topic`, the topic to receive records from, defaults to `test-topic`
- `consumer.poll-period`, maximum amount of time (ms) to wait for messages to arrive in one poll, defaults to `1000`
- `consumer.wait-after-recv`, the amount of time (ms) to wait after a batch is received, defaults to `0`; useful if you need to perform an action after a batch has been received, but before processing
- `consumer.wait-after-batch`, the amount of time (ms) to wait after a batch is processed, defaults to `0`; useful if you need to perform an action after a batch has been processed
- `consumer.wait-after-record`, the amount of time (ms) to wait after each processed record, defaults to `0`; useful to artificially slow down receiving
- `consumer.ack-every-x-msgs`, manually commit offsets every x records processed, defaults to `null` (i.e. `auto-commit = true`); turns off `auto-commit` if set
- `consumer.ack-after-batch`, manually commit offsets after finishing a batch (and possibly waiting for the requested amount of time, see the various wait options above); turns off `auto-commit` and is exclusive with `ack-every-x-msgs`; if both are set, `ack-after-batch` takes precedence and `ack-every-x-msgs` is turned off
- `consumer.wait-cmd-period`, when receiving a `wait` command, the amount of time (ms) to block for, defaults to `5000`
- `consumer.local-id`, when writing log files, the sequence to append to the base name, defaults to `-1` (meaning no suffix); useful for multiple consumers (i.e. `consumer-0.log`, `consumer-1.log`, etc.)
- `consumer.payload-trunc`, when starting up, whether to truncate the payload log, defaults to `false`

The following consumer client options can also be controlled from the command line:

- `consumer.group-id`, the consumer group to announce, defaults to `test-app`
- `consumer.instance-id`, consumer instance ID (for static consumers), defaults to `null`
- `consumer.auto-commit`, whether to automatically commit offsets, defaults to `true`
- `consumer.ac-interval`, how often (in ms) to commit offsets, defaults to `5000`

- consumer.fetch-min-bytes, minimum amount of data to fetch, defaults to 1
- consumer.max-poll-recs, maximum number of records to fetch each poll, defaults to 500
- consumer.assignment-strategy, partition assignment strategy, one of coop, range, rr, and sticky (default is coop)
- consumer.heartbeat-interval, how often to report liveness to broker (in ms), defaults to 3000
- consumer.session-timeout, how long (in ms) before the consumer is removed from the group for lack of heartbeat, defaults to 45000; brokers may limit the minimum and maximum values for this setting
- consumer.auto-offset-reset, what to do when no consumer offsets are found in the broker, defaults to latest, can also be earliest or none (but do not use none)

If you want to set any of the above properties, simply set them on the command line using the -D option.

```
$ mvn exec:java -Dconsumer.group-id=myapp -Dconsumer.instance-id=consumer0 \
    -Dconsumer.local-id=0 -Dconsumer.poll-period=500 -Dconsumer.max-poll-recs=50
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-consumer >-------------
[INFO] Building core-api-consumer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ core-api-consumer ---
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
```

Other Tools

A couple of scripts exist to aid you in collecting the logs and cleaning up after a session.

- get-logs.sh will sort both producer and consumer logs and store them into producer.log and consumer.log next to the script, removing the original logs in the process
- remove-logdirs.sh will remove all broker, Kafka Connect, and Zookeeper data; it should be executed from the same directory where the broker property files are (e.g. the top-level directory for the labs)

REFERENCES	Break-Fix Labs Client Apps and Tools on GitHub Kafka Producer Configuration Options Kafka Consumer Configuration Options

2.2. Guided Exercise Getting to know the client apps and tools

Test the consumer and producer apps in a reliable setting to verify their correct operation.

Try a simple misconfiguration to verify the scenario works predictably, and demonstrate the tool use.

Outcomes

- Test the consumer and producer apps in a reliable setting to verify their correct operation.
- Try a simple misconfiguration to verify the scenario works predictably.

Prerequisites

Ensure that you have successfully completed the first exercise, setting up the lab environment.

Instructions

This exercise consists of two parts.

The first part is simply testing regular delivery with default client application options. It is a verification of our lab environment of sorts, making sure everything works as expected.

The second part simulates message redelivery after consumer failure, not acknowledging any messages.

1. Ensure your Zookeeper instance is running. Stopp all brokers except `broker0`.
2. Start the consumer application with default settings.
    1. In a terminal window, change your working directory to consumer app and start it.

```
$ cd streams-bf-lab-materials/code/core-api-consumer/

$ git pull
Already up to date.

$ mvn clean compile exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-consumer >-------------
[INFO] Building core-api-consumer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
```

```
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Starting to poll for
batches of up to 500 records / up to 1000 ms...
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.ConsumerRebalanceListenerImpl - NEW
PARTITIONS ASSIGNED: [test-topic-0, test-topic-1, test-topic-2]
...
```

2. Leave the consumer running and open a new terminal window.

3. Start the producer application with default settings.

   1. Change your working directory to the producer app and start it.

```
$ cd streams-bf-lab-materials/code/core-api-producer/

$ git pull
Already up to date.

$ mvn clean compile exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-producer >-------------
[INFO] Building core-api-producer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Starting to produce 100
records per roll, 1 rolls...
...
[kafka-producer-network-thread | producer-1] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Sent: T:test-topic P:2 K:2
V:"I agree with everything you say, but I would attack to the death your right
to say it." -- Tom Stoppard (1937 - )
[kafka-producer-network-thread | producer-1] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Sent: T:test-topic P:2 K:5
V:"There is no nonsense so gross that society will not, at some time, make a
doctrine of it and defend it with every weapon of communal stupidity." --
Robertson Davies
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Total time:  1.297 s
[INFO] Finished at: 2024-09-13T17:23:56+02:00
[INFO] ------------------------------------------------------------------------
```

2. The producer had just sent 100 random quotes to the `test-topic` topic in the broker. Since the topic did not exist beforehand, it was created with default settings.
3. Verify the producer payload log contains all the records.

```
$ wc -l payload.log
     100 payload.log
```

4. Observe the consumer application and note that the received records were printed on the console.
5. Send a `quit` message to consumer to initiate a clean shutdown.
   1. In yet another terminal window, move to the lab directory and use `kafka-console-producer.sh` to send a single message to the `test-topic` topic.

   ```
   $ echo quit | ./kafka/bin/kafka-console-producer.sh \
       --bootstrap-server localhost:9092 --topic test-topic
   ```

   2. Verify that the consumer application had indeed terminated.

   ```
   ...
   [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
   com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:1
   K:91 V:"I am here and you will know that I am the best and will hear me." --
   Leontyne Price, O Magazine, December 2003
   [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
   com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:1
   K:93 V:"I am here and you will know that I am the best and will hear me." --
   Leontyne Price, O Magazine, December 2003
   [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
   com.redhat.training.kafka.coreapi.consumer.Consumer - Received "quit" message.
   Exiting.
   ...
   [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
   org.apache.kafka.common.utils.AppInfoParser - App info kafka.consumer for
   consumer-test-app-1 unregistered
   ```

   3. Verify that the consumer payload log contains all the records.

   ```
   $ wc -l payload.log
        100 payload.log
   ```

6. Collect payload logs and compare them.

   1. In the terminal window where your shell is at the lab top level directory, run the `get-logs.sh` script.

      ```
      $ ./streams-bf-lab-materials/code/get-logs.sh
      ./streams-bf-lab-materials/code/core-api-producer/payload.log
      ./streams-bf-lab-materials/code/core-api-consumer/payload.log
      Producer and/or Consumer logs available in ./streams-bf-lab-materials/code - old
      logs were removed.
      ```

   2. Compare logs and observe they are identical.

      ```
      $ diff streams-bf-lab-materials/code/producer.log \
          streams-bf-lab-materials/code/consumer.log
      ```

      NOTE No output from the `diff` command means the files are identical.

      IMPORTANT It is important that you do NOT delete the topic after the first part of this exercise.

In part two of this exercise, we will simulate some basic misconfiguration to ensure that "incorrect" behaviour also works as expected.

1. Ensure your Zookeeper instance and `broker0` are still running.
2. Start the consumer application with `auto-commit` setting turned off and send some messages to it.

   1. In the *consumer* terminal, start the application with required settings. Leave it running.

      ```
      $ mvn exec:java -Dconsumer.auto-commit=false
      [INFO] Scanning for projects...
      [INFO]
      [INFO] ------------< com.redhat.training.kafka:core-api-consumer >-------------
      [INFO] Building core-api-consumer 1.0.0-SNAPSHOT
      [INFO]   from pom.xml
      [INFO] --------------------------------[ jar ]---------------------------------
      [INFO]
      [INFO] --- exec:3.4.1:java (default-cli) @ core-api-consumer ---
      [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
      com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
      [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
      org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
      ...
      ```

   2. In the *producer* terminal, start the application with the default settings. Wait for it to complete.

```
$ mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-producer >-------------
[INFO] Building core-api-producer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ core-api-producer ---
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
org.apache.kafka.common.utils.AppInfoParser - App info kafka.producer for
producer-1 unregistered
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.680 s
[INFO] Finished at: 2024-09-13T18:51:39+02:00
[INFO] ------------------------------------------------------------------------
```

3. Verify that in the consumer terminal, the messages have actually been received.
4. Terminate the consumer by pressing `Ctrl-C`.

```
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:1
K:91 V:"There is no nonsense so gross that society will not, at some time, make
a doctrine of it and defend it with every weapon of communal stupidity." --
Robertson Davies
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:1
K:93 V:"I agree with everything you say, but I would attack to the death your
right to say it." -- Tom Stoppard (1937 - )
^C
```

3. Restart the consumer and observe its behaviour.
   1. In the consumer terminal, simply restart the application with the same settings.

```
$ mvn exec:java -Dconsumer.auto-commit=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-consumer >-------------
[INFO] Building core-api-consumer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
```

```
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ core-api-consumer ---
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
org.apache.kafka.clients.consumer.internals.ConsumerCoordinator - [Consumer
clientId=consumer-test-app-1, groupId=test-app] (Re-)joining group
```

2. Observe that nothing seems to be going on. Be patient.
3. After a while (it will take almost 45 seconds), consumer receives 100 messages again.

IMPORTANT

Why is the above delay, and then a complete redelivery of all the messages, seen?

The redelivery happens for a simple reason - we prevented the consumer from committing its last-seen message offsets, so the broker resends the last 100 messages again and again, until a consumer acknowledges reception.

Let's examine the reason for the delay in the next steps.

4. Inspect broker logs and establish the cause for delayed processing.
   1. Examine the last 5 log records in the terminal where `broker0` is running.

```
[2024-09-13 19:48:26,283] INFO [GroupCoordinator 0]: Dynamic member with unknown
member id joins group test-app in Stable state. Created a new member id
consumer-test-app-1-96747000-5145-442f-b5d6-10e70b571373 and request the member
to rejoin with this id. (kafka.coordinator.group.GroupCoordinator)
[2024-09-13 19:48:26,285] INFO [GroupCoordinator 0]: Preparing to rebalance
group test-app in state PreparingRebalance with old generation 3
(__consumer_offsets-36) (reason: Adding new member consumer-test-app-1-96747000-
5145-442f-b5d6-10e70b571373 with group instance id None; client reason: need to
re-join with the given member-id: consumer-test-app-1-96747000-5145-442f-b5d6-
10e70b571373) (kafka.coordinator.group.GroupCoordinator)
[2024-09-13 19:48:41,985] INFO [GroupCoordinator 0]: Member consumer-test-app-1-
93e3edf3-1a08-4b26-bf79-6a2d1cc0780b in group test-app has failed, removing it
from the group (kafka.coordinator.group.GroupCoordinator)
[2024-09-13 19:48:41,986] INFO [GroupCoordinator 0]: Stabilized group test-app
generation 4 (__consumer_offsets-36) with 1 members
(kafka.coordinator.group.GroupCoordinator)
[2024-09-13 19:48:41,995] INFO [GroupCoordinator 0]: Assignment received from
leader consumer-test-app-1-96747000-5145-442f-b5d6-10e70b571373 for group test-
app for generation 4. The group has 1 members, 0 of which are static.
(kafka.coordinator.group.GroupCoordinator)
```

2. It looks like the reason for delay is a rebalance of the consumer group `test-app`!

   Because the old consumer was terminated abruptly, using `Ctrl-C`, it did not leave the group cleanly. When the new consumer joins, it is not exactly clear what is going on. Is the new consumer another consumer and we have two of them?

   The broker therefore has to wait until either the old consumer's session times out or it rejoins. Only then can it perform a correct rebalance and assign partitions.

5. Restart the consumer as a static instead of dynamic one.

   1. In the consumer terminal window, press `Ctrl-C` to terminate the consumer.
   2. Restart the consumer, but this time set the `consumer.instance-id` property.

```
$ mvn exec:java -Dconsumer.auto-commit=false -Dconsumer.instance-id=12345
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-consumer >-------------
[INFO] Building core-api-consumer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ core-api-consumer ---
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:1
K:93 V:"Do not pursue what is illusory - property and position: all that is
gained at the expense of your nerves decade after decade and can be confiscated
in one fell night. Live with a steady superiority over life - don't be afraid of
misfortune, and do not yearn after happiness; it is after all, all the same: the
bitter doesn't last forever, and the sweet never fills the cup to overflowing."
-- Alexander Solzhenitsyn (1918 - )
```

   3. Note that there is again a delay as the previous instance did not exit cleanly. Wait for the consumer to receive the messages and then interrupt it using `Ctrl-C`.
   4. Repeat the same command once more. Note that the consumer *immediately* receives the messages - there is no more delay.
   5. Verify that the consumer protocol is now different - observe the last couple of records in the broker log.

```
[2024-09-13 20:49:11,001] INFO [GroupCoordinator 0]: Static member with
groupInstanceId=12345 and unknown member id joins group test-app in Stable
state. Replacing previously mapped member 12345-f2feb934-9a83-47a7-b268-
f6a3b5fe8afa with this groupInstanceId.
```

```
(kafka.coordinator.group.GroupCoordinator)
[2024-09-14 12:28:32,077] INFO [GroupCoordinator 0]: Static member which joins
during Stable stage and doesn't affect selectProtocol will not trigger
rebalance. (kafka.coordinator.group.GroupCoordinator)
```

6. You can see that with an instance ID setting, the client is recognised as having been seen already. There is no more wait for the old client's session to time out. Additionally, there is no rebalancing.

7. Stop the client by pressing `Ctrl-C` in the console.

6. Clean up the lab environment.

   1. Stop the broker by pressing `Ctrl-C` in the terminal window where it is running.

   2. Stop the Zookeeper instance by pressing `Ctrl-C` in the terminal window where it is running.

   3. At the lab top level directory, execute the `remove-logdirs.sh` script that will remove all log directories for the services.

```
$ ./streams-bf-lab-materials/code/remove-logdirs.sh
WARNING: Removing log directories for Zookeeper, all brokers, and Kafka Connect.
         MAKE SURE THE PROCESSES ARE NOT RUNNING!

Continue?
1) Yes
2) No
#? 1
Done.
```

This concludes the guided exercise.

Chapter 3. Producer Delivery Semantics

Goal

Refresh producer delivery options and side effects.

Sections

3.1. Guided Exercise (Not) Requiring Acknowledgments

Investigate how lack of acknowledgment requests affects delivery guarantee.

Outcomes

- See how acknowledgment settings affect delivery reliability.

Prerequisites

Ensure that you have successfully completed the first exercise on setting up the lab environment.

Instructions

1. Ensure Zookeeper and all three brokers are running.
    1. Start Zookeeper first. In the *lab directory* (at the top level), invoke `zookeeper-server-start.sh` script.

       ```
       $ ./kafka/bin/zookeeper-server-start.sh zookeeper.properties
       [2024-09-14 12:18:00,164] INFO Reading configuration from: zookeeper.properties
       (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
       [2024-09-14 12:18:00,165] WARN zookeeper.properties is relative. Prepend ./ to
       indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
       [2024-09-14 12:18:00,169] INFO clientPortAddress is 0.0.0.0:2181
       (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
       ...
       ```

    2. Next, start all three brokers, each in a new terminal.

       ```
       $ ./kafka/bin/kafka-server-start.sh broker0.properties
       [2024-09-14 12:40:54,603] INFO Registered kafka:type=kafka.Log4jController MBean
       (kafka.utils.Log4jControllerRegistration$)
       ...
       [2024-09-14 12:40:55,691] INFO [zk-broker-0-to-controller-alter-partition-
       channel-manager]: Recorded new ZK controller, from now on will use node
       172.16.83.208:9092 (id: 0 rack: null)
       (kafka.server.NodeToControllerRequestThread)
       ```

       ```
       $ ./kafka/bin/kafka-server-start.sh broker1.properties
       [2024-09-14 12:40:54,603] INFO Registered kafka:type=kafka.Log4jController MBean
       (kafka.utils.Log4jControllerRegistration$)
       ...
       [2024-09-14 12:40:55,691] INFO [zk-broker-1-to-controller-alter-partition-
       channel-manager]: Recorded new ZK controller, from now on will use node
       172.16.83.208:9092 (id: 0 rack: null)
       ```

```
(kafka.server.NodeToControllerRequestThread)
```

```
$ ./kafka/bin/kafka-server-start.sh broker2.properties
[2024-09-14 12:40:54,603] INFO Registered kafka:type=kafka.Log4jController MBean
(kafka.utils.Log4jControllerRegistration$)
...
[2024-09-14 12:40:55,691] INFO [zk-broker-2-to-controller-alter-partition-
channel-manager]: Recorded new ZK controller, from now on will use node
172.16.83.208:9092 (id: 0 rack: null)
(kafka.server.NodeToControllerRequestThread)
```

2. Remove any payload logs that might have been left over from before.

   1. From the top-level lab directory, run the `remove-payload-logs.sh` script.

```
$ ./streams-bf-lab-materials/code/remove-payload-logs.sh
Removing payload logs in producer/consumer working directories...
./streams-bf-lab-materials/code/core-api-producer/payload.log
./streams-bf-lab-materials/code/core-api-consumer/payload.log
Done.
```

3. Create a non-replicated topic with multiple partitions.

   1. In the same terminal, invoke the `kafka-topics.sh` script with `--create` command.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --partitions 6 --replication-factor 1 --create
Created topic test-topic.
```

   2. Inspect the topic configuration, ensuring that each of the brokers is the leader for two partitions.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --describe
Topic: test-topic  TopicId: YWDp... PartitionCount: 6  ReplicationFactor: 1
Configs:
  Topic: test-topic   Partition: 0   Leader: 2   Replicas: 2   Isr: 2
  Topic: test-topic   Partition: 1   Leader: 1   Replicas: 1   Isr: 1
  Topic: test-topic   Partition: 2   Leader: 0   Replicas: 0   Isr: 0
  Topic: test-topic   Partition: 3   Leader: 2   Replicas: 2   Isr: 2
  Topic: test-topic   Partition: 4   Leader: 1   Replicas: 1   Isr: 1
  Topic: test-topic   Partition: 5   Leader: 0   Replicas: 0   Isr: 0
```

4. Send some records without acknowledgment receipts, stopping one of the brokers along the way.

   1. In the producer window, start the application to send 500000 messages without

acknowledgments. *Be prepared to act quickly in the next step as the send only takes around 15 seconds!*

```
$ mvn exec:java -Dproducer.acks=0 -Dproducer.num-records-per-roll=500000
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Setting idempotence to
false as acks != all.
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:
    acks = 0
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Starting to produce 500000
records per roll, 1 rolls...
[kafka-producer-network-thread | producer-1] INFO
org.apache.kafka.clients.Metadata - [Producer clientId=producer-1] Cluster ID:
cXuKjWSMSb2qRiXU0kUTdw
...
```

2. While the producer is sending, stop one of the brokers by pressing `Ctrl-C` in the terminal where it is running. Restart it, and proceed to stop a different broker. Repeat this, alternating between brokers, a couple of times.
3. With one of the brokers stopped, observe that the producer has halted - it is not sending messages.
4. Inspect the topic again, using the `kafka-topics.sh` command at the top of the lab directory.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --describe
Topic: test-topic  TopicId: YWDp... PartitionCount: 6  ReplicationFactor: 1
Configs:
  Topic: test-topic   Partition: 0   Leader: 2    Replicas: 2   Isr: 2
  Topic: test-topic   Partition: 1   Leader: none Replicas: 1   Isr: 1
  Topic: test-topic   Partition: 2   Leader: 0    Replicas: 0   Isr: 0
  Topic: test-topic   Partition: 3   Leader: 2    Replicas: 2   Isr: 2
  Topic: test-topic   Partition: 4   Leader: none Replicas: 1   Isr: 1
  Topic: test-topic   Partition: 5   Leader: 0    Replicas: 0   Isr: 0
```

Note that two of the partitions have lost their leader - the broker that was just terminated.

NOTE Depending on which of the brokers you stopped, you might see different output.

5. Restart the stopped broker and wait for the sending to complete.

6. Run the consumer to see how many records were received by the brokers.

    1. In the consumer terminal, start the application, telling it to receive all messages from the topic, from the beginning.

    ```
    $ mvn exec:java -Dconsumer.auto-offset-reset=earliest
    [INFO] Scanning for projects...
    ...
    [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
    com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
    [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
    org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
    ...
    [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
    com.redhat.training.kafka.coreapi.consumer.Consumer - Starting to poll for
    batches of up to 500 records / up to 1000 ms...
    ...
    [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
    com.redhat.training.kafka.coreapi.consumer.Consumer - Received 500 records.
    Processing.
    [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
    com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:5
    K:2 V:"It's going to come true like you knew it, but it's not going to feel like
    you think." -- Rosie O'Donnell, Today Show interview, 04-08-08
    ...
    ```

    2. Cleanly shut down the consumer when it stops printing records - send it a `quit` message.

    ```
    $ echo quit | ./kafka/bin/kafka-console-producer.sh \
        --bootstrap-server localhost:9092 --topic test-topic
    ```

    3. Collect the logs with `get-logs.sh`. You can issue the command in the consumer, or the producer terminal.

    ```
    $ ../get-logs.sh
    WARNING: Existing logs will be overwritten. Do you want to continue?
    1) Yes
    2) No
    #? 1
    ../core-api-producer/payload.log
    ../core-api-consumer/payload.log
    Producer and/or Consumer logs available in .. - old logs were removed.
    ```

    4. Compare the logs with `diff` - a number of messages should be missing.

    ```
    $ diff ../producer.log ../consumer.log  | wc -l
         297
    ```

```
$ diff ../producer.log ../consumer.log  | tail -n5
< test-topic,2,394140,"We are advertis'd by our loving friends." -- William
Shakespeare (1564 - 1616)
< test-topic,2,394142,"Anyone who goes to a psychiatrist ought to have his head
examined." -- Samuel Goldwyn (1882 - 1974)
< test-topic,2,394145,"I think the world is run by 'C' students." -- Al McGuire
< test-topic,2,394153,"Any fool can tell the truth, but it requires a man of
some sense to know how to lie well." -- Samuel Butler (1835 - 1902)
< test-topic,2,394155,"I wish you sunshine on your path and storms to season
your journey. I wish you peace in the world in which you live... More I cannot
wish you except perhaps love to make all the rest worthwhile." -- Robert A. Ward
```

NOTE The first character on each line pointing to the left means that the record was found in `producer.log`, but not in `consumer.log`, meaning it never got delivered to the consumer.

IMPORTANT You might need to restart the sending and repeat the interruptions if you get no lost messages. Try the `-Dproducer.linger=200 -Dproducer.max-inflight=100` options if nothing else helps. The loss of messages depends on many factors. Generally speaking, the more responsive the system is, the less likely it is for us to observe the above behaviour.

7. Delete the topic.
   1. From the top-level lab directory, issue the `kafka-topics.sh` command to delete `test-topic`.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --delete
```

Key Takeaways

If producer does not wait for acknowledgments from partition leaders, messages can be lost in the case of broker outage or network communication issues.

This concludes the guided exercise.

3.2. Guided Exercise Replicating Topics

Configure topic replication in combination with acks to improve reliability.

Outcomes

- Test delivery with and without acknowledgments with replicated topics.

Prerequisites

Ensure that you have successfully completed the first exercise on setting up the lab environment. Your Zookeeper and all three brokers should be running.

Instructions

1. Remove any payload logs that might have been left over from before. From the top-level lab directory, run the `remove-payload-logs.sh` script.

   ```
   $ ./streams-bf-lab-materials/code/remove-payload-logs.sh
   Removing payload logs in producer/consumer working directories...
   ./streams-bf-lab-materials/code/core-api-producer/payload.log
   ./streams-bf-lab-materials/code/core-api-consumer/payload.log
   Done.
   ```

2. Create a replicated topic with multiple partitions.

   1. In the same terminal, invoke the `kafka-topics.sh` script with `--create` command.

      ```
      $ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
          --topic test-topic --partitions 6 --replication-factor 2 --create
      Created topic test-topic.
      ```

   2. Inspect the topic configuration, ensuring that each of the brokers is the leader for two partitions, and each partition shows two replicas that are in-sync.

      ```
      $ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
          --topic test-topic --describe
      Topic: test-topic  TopicId: _wtM...  PartitionCount: 6  ReplicationFactor: 2
      Configs:
        Topic: test-topic  Partition: 0  Leader: 0  Replicas: 0,1  Isr: 0,1
        Topic: test-topic  Partition: 1  Leader: 2  Replicas: 2,0  Isr: 2,0
        Topic: test-topic  Partition: 2  Leader: 1  Replicas: 1,2  Isr: 1,2
        Topic: test-topic  Partition: 3  Leader: 0  Replicas: 0,2  Isr: 0,2
        Topic: test-topic  Partition: 4  Leader: 2  Replicas: 2,1  Isr: 2,1
        Topic: test-topic  Partition: 5  Leader: 1  Replicas: 1,0  Isr: 1,0
      ```

3. Proceed as in the previous exercise. Start the producer, telling it to send 500000 records, not

requesting any acknowledgments.

```
$ mvn exec:java -Dproducer.acks=0 -Dproducer.num-records-per-roll=500000
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Setting idempotence to false
as acks != all.
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:
    acks = 0
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Starting to produce 500000
records per roll, 1 rolls...
[kafka-producer-network-thread | producer-1] INFO org.apache.kafka.clients.Metadata
- [Producer clientId=producer-1] Cluster ID: cXuKjWSMSb2qRiXU0kUTdw
...
```

4. While sending, stop one of the brokers by pressing `Ctrl-C` in the terminal where it is running.
5. Observe that the producer keeps sending records without a noticeable pause or slow-down.
6. Inspect the topic again. Notice that its leaders have changed, but not the replica assignment. The lost broker just does not show in the `Isr` list any more.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --describe
Topic: test-topic  TopicId: _wtM...  PartitionCount: 6  ReplicationFactor: 2
Configs:
  Topic: test-topic  Partition: 0  Leader: 0  Replicas: 0,1  Isr: 0
  Topic: test-topic  Partition: 1  Leader: 2  Replicas: 2,0  Isr: 2,0
  Topic: test-topic  Partition: 2  Leader: 2  Replicas: 1,2  Isr: 2
  Topic: test-topic  Partition: 3  Leader: 0  Replicas: 0,2  Isr: 0,2
  Topic: test-topic  Partition: 4  Leader: 2  Replicas: 2,1  Isr: 2
  Topic: test-topic  Partition: 5  Leader: 0  Replicas: 1,0  Isr: 0
```

7. After sending completes, restart the stopped broker.
8. Inspect the topic to see that `Isr` reflects the restored broker, but the leaders have not changed.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --describe
Topic: test-topic  TopicId: _wtM...  PartitionCount: 6  ReplicationFactor: 2
Configs:
  Topic: test-topic  Partition: 0  Leader: 0  Replicas: 0,1  Isr: 0,1
  Topic: test-topic  Partition: 1  Leader: 2  Replicas: 2,0  Isr: 2,0
```

```
    Topic: test-topic  Partition: 2  Leader: 2  Replicas: 1,2  Isr: 2,1
    Topic: test-topic  Partition: 3  Leader: 0  Replicas: 0,2  Isr: 0,2
    Topic: test-topic  Partition: 4  Leader: 2  Replicas: 2,1  Isr: 2,1
    Topic: test-topic  Partition: 5  Leader: 0  Replicas: 1,0  Isr: 0,1
```

9. Run the consumer to see how many records were received by the brokers.

   1. In the consumer terminal, start the application, telling it to receive all messages from the topic, from the beginning.

      ```
      $ mvn exec:java -Dconsumer.auto-offset-reset=earliest
      [INFO] Scanning for projects...
      ...
      [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
      com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
      [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
      org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
      ...
      [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
      com.redhat.training.kafka.coreapi.consumer.Consumer - Starting to poll for
      batches of up to 500 records / up to 1000 ms...
      ...
      [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
      com.redhat.training.kafka.coreapi.consumer.Consumer - Received 500 records.
      Processing.
      [com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
      com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:5
      K:2 V:"It's going to come true like you knew it, but it's not going to feel like
      you think." -- Rosie O'Donnell, Today Show interview, 04-08-08
      ...
      ```

   2. Cleanly shut down the consumer when it stops printing records - send it a `quit` message.

      ```
      $ echo quit | ./kafka/bin/kafka-console-producer.sh \
          --bootstrap-server localhost:9092 --topic test-topic
      ```

   3. Collect the logs with `get-logs.sh`. You can issue the command in the consumer, or the producer terminal.

      ```
      $ ../get-logs.sh
      WARNING: Existing logs will be overwritten. Do you want to continue?
      1) Yes
      2) No
      #? 1
      ../core-api-producer/payload.log
      ../core-api-consumer/payload.log
      Producer and/or Consumer logs available in .. - old logs were removed.
      ```

4. Compare producer and consumer logs.

```
$ diff ../producer.log ../consumer.log  | wc -l
     367

$ diff ../producer.log ../consumer.log  | tail -n5
< test-topic,4,47608,"If you think you can do a thing or think you can't do a
thing, you're right." -- Henry Ford (1863 - 1947), (attributed)
402710d402524
< test-topic,4,47611,"Any fool can tell the truth, but it requires a man of some
sense to know how to lie well." -- Samuel Butler (1835 - 1902)
402715d402528
< test-topic,4,47614,"There is no nonsense so gross that society will not, at
some time, make a doctrine of it and defend it with every weapon of communal
stupidity." -- Robertson Davies
```

IMPORTANT

You might need to restart the sending and repeat the interruptions if you get no lost messages. Try the `-Dproducer.linger=200 -Dproducer.max-inflight=100` options if nothing else helps. The loss of messages depends on many factors. Generally speaking, the more responsive the system is, the less likely it is for us to observe the above behaviour.

The above proves that without acknowledgment requests, not even replication can prevent record loss.

Let us repeat the above with `-Dproducer.acks=1`.

1. Recreate the topic with the same settings.

   1. In the top-level lab directory, use `kafka-topics.sh` to delete and recreate the topic.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --delete

$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --partitions 6 --replication-factor 2 --create
Created topic test-topic.
```

2. Start the producer, telling it to send 500000 records, requesting acknowledgments from the leader only.

```
$ mvn exec:java -Dproducer.acks=1 -Dproducer.num-records-per-roll=500000
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
```

```
com.redhat.training.kafka.coreapi.producer.Producer - Setting idempotence to false
as acks != all.
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:
    acks = 0
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Starting to produce 500000
records per roll, 1 rolls...
[kafka-producer-network-thread | producer-1] INFO org.apache.kafka.clients.Metadata
- [Producer clientId=producer-1] Cluster ID: cXuKjWSMSb2qRiXU0kUTdw
...
```

3. While sending, stop two out of the three brokers. If your system has the `pkill` command, terminate them using `pkill -f -KILL brokerX` (where X is the number of the broker you want to stop). Otherwise terminate them by pressing `Ctrl-C` in the terminal where they are running.
4. Observe the producer again stopped sending. Restart the last one of the two stopped brokers you stopped.

IMPORTANT The last broker you stopped will have been the leader for half of the partitions in the topic at the time you stopped it. If you restart the other broker, it will be unable to resume the leader role.

5. When the producer completes its sending, restart the brokers.
6. Run the consumer and wait for it to stop receiving any new messages.

```
$ mvn exec:java -Dconsumer.auto-offset-reset=earliest
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Starting to poll for batches
of up to 500 records / up to 1000 ms...
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Received 500 records.
Processing.
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:5
K:2 V:"It's going to come true like you knew it, but it's not going to feel like
you think." -- Rosie O'Donnell, Today Show interview, 04-08-08
...
```

7. Cleanly shut down the consumer when it stops printing records - send it a `quit` message.

```
$ echo quit | ./kafka/bin/kafka-console-producer.sh \
    --bootstrap-server localhost:9092 --topic test-topic
```

8. Collect the logs with `get-logs.sh`. You can issue the command in the consumer, or the producer terminal.

```
$ ../get-logs.sh
WARNING: Existing logs will be overwritten. Do you want to continue?
1) Yes
2) No
#? 1
../core-api-producer/payload.log
../core-api-consumer/payload.log
Producer and/or Consumer logs available in .. - old logs were removed.
```

9. Compare producer and consumer logs.

```
$ diff ../producer.log ../consumer.log  | wc -l
    1101
```

Observe that there are still lost records.

IMPORTANT

It is going to become increasingly difficult to get consistant data loss as you activate more and more protective mechanisms. Repeat the tests as many times as necessary, adding disruptions to your system if necessary. Disruption can be anything from a process reading the disks heavily, to a high CPU load, or both.

10. Delete the topic. From the top-level lab directory, issue the `kafka-topics.sh` command to delete `test-topic`.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --delete
```

Key Takeaways

If producer does not wait for acknowledgments from partition leaders, not even replication helps prevent data loss. Messages can even be lost in the case of broker outage when acknowledgments are only received from leaders for individual partitions.

This concludes the guided exercise.

3.3. Guided Exercise In-Sync Replicas

Configure topics with a minimum number of in-sync replicas.

Outcomes

- Configure a topic and the producer for reliable at-least-once delivery.

Prerequisites

Ensure that you have successfully completed the first exercise on setting up the lab environment. Your Zookeeper and all three brokers should be running.

Instructions

1. Remove any payload logs that might have been left over from before. From the top-level lab directory, run the `remove-payload-logs.sh` script.

```
$ ./streams-bf-lab-materials/code/remove-payload-logs.sh
Removing payload logs in producer/consumer working directories...
./streams-bf-lab-materials/code/core-api-producer/payload.log
./streams-bf-lab-materials/code/core-api-consumer/payload.log
Done.
```

2. Create a replicated topic with multiple partitions and `min.insync.replicas` setting.

   1. In the same terminal, invoke the `kafka-topics.sh` script with `--create` command.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --partitions 6 --replication-factor 2 \
    --config min.insync.replicas=2 --create
Created topic test-topic.
```

   2. Inspect the topic configuration, ensuring that each of the brokers is the leader for two partitions, and each partition shows two replicas that are in-sync.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --describe
Topic: test-topic  TopicId: _wtM...  PartitionCount: 6  ReplicationFactor: 2
Configs: min.insync.replicas=2
  Topic: test-topic  Partition: 0  Leader: 0  Replicas: 0,1  Isr: 0,1
  Topic: test-topic  Partition: 1  Leader: 2  Replicas: 2,0  Isr: 2,0
  Topic: test-topic  Partition: 2  Leader: 1  Replicas: 1,2  Isr: 1,2
  Topic: test-topic  Partition: 3  Leader: 0  Replicas: 0,2  Isr: 0,2
  Topic: test-topic  Partition: 4  Leader: 2  Replicas: 2,1  Isr: 2,1
  Topic: test-topic  Partition: 5  Leader: 1  Replicas: 1,0  Isr: 1,0
```

3. Test the producer and simulate a broker failure. Ensure the producer requests acknowledgments from all replicas up to `min.insync` number.

```
$ mvn exec:java -Dproducer.acks=all -Dproducer.num-records-per-roll=500000
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Starting to produce 500000
records per roll, 1 rolls...
[kafka-producer-network-thread | producer-1] INFO org.apache.kafka.clients.Metadata
- [Producer clientId=producer-1] Cluster ID: cXuKjWSMSb2qRiXU0kUTdw
...
```

4. While sending, stop one of the brokers by pressing `Ctrl-C` in the terminal where it is running.
5. In the producer terminal, notice that we start seeing errors about insufficient number of replicas.

```
...
[kafka-producer-network-thread | producer-1] WARN
org.apache.kafka.clients.producer.internals.Sender - [Producer clientId=producer-1]
Got error produce response with correlation id 1503 on topic-partition test-topic-
0, retrying (2147483609 attempts left). Error: NOT_ENOUGH_REPLICAS
[kafka-producer-network-thread | producer-1] WARN
org.apache.kafka.clients.producer.internals.Sender - [Producer clientId=producer-1]
Got error produce response with correlation id 1505 on topic-partition test-topic-
3, retrying (2147483609 attempts left). Error: NOT_ENOUGH_REPLICAS
...
```

IMPORTANT Clearly with `min.insync.replicas` set to the same number as replication level, you can not afford to lose a single broker.

6. Restart the broker and wait for the producer to finish.
7. Recreate the topic with three replicas and `min.insync.replicas` set to two.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --delete

$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --partitions 6 --replication-factor 3 \
    --config min.insync.replicas=2 --create
Created topic test-topic.
```

NOTE It may take a while for the delete operation to complete. If you receive an error about `test-topic` being marked for deletion, either wait a couple of seconds and retry, or stop all the brokers and Zookeeper, and use the `remove-logdirs.sh` script to completely remove the data stores.

8. Restart the producer with the same settings, then stop two brokers when it starts sending messages.
9. Observe that the producer stops again, with the same error as before.

```
...
[kafka-producer-network-thread | producer-1] WARN
org.apache.kafka.clients.producer.internals.Sender - [Producer clientId=producer-1]
Got error produce response with correlation id 584 on topic-partition test-topic-2,
retrying (2147483642 attempts left). Error: NOT_ENOUGH_REPLICAS
[kafka-producer-network-thread | producer-1] WARN
org.apache.kafka.clients.producer.internals.Sender - [Producer clientId=producer-1]
Got error produce response with correlation id 585 on topic-partition test-topic-4,
retrying (2147483641 attempts left). Error: NOT_ENOUGH_REPLICAS
...
```

   The difference this time is that you had to stop two out of three brokers for this to happen.

10. Restart one of the stopped brokers. Observe that the producer resumes sending.
11. There is no need to restart the third broker when the producer completes its sending. Collect the messages in the consumer and compare the payload logs.
    1. Start the consumer instructing it to reset its offsets to earliest message possible.

```
$ mvn exec:java -Dconsumer.auto-offset-reset=earliest
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Starting to poll for
batches of up to 500 records / up to 1000 ms...
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Received 500 records.
Processing.
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Received: T:test-topic P:5
K:2 V:"It's going to come true like you knew it, but it's not going to feel like
you think." -- Rosie O'Donnell, Today Show interview, 04-08-08
```

```
...
```

2. Cleanly shut down the consumer when it stops printing records - send it a `quit` message.

   ```
   $ echo quit | ./kafka/bin/kafka-console-producer.sh \
       --bootstrap-server localhost:9092 --topic test-topic
   ```

3. Collect the logs with `get-logs.sh`. You can issue the command in the consumer, or the producer terminal.

   ```
   $ ../get-logs.sh
   WARNING: Existing logs will be overwritten. Do you want to continue?
   1) Yes
   2) No
   #? 1
   ../core-api-producer/payload.log
   ../core-api-consumer/payload.log
   Producer and/or Consumer logs available in .. - old logs were removed.
   ```

4. Compare producer and consumer logs.

   ```
   $ diff ../producer.log ../consumer.log  | wc -l
          0
   ```

   This time around, there is no data loss.

   Feel free to retry the test from step 7 onwards with varying degrees of severity in stopping the brokers (such with `kill` signals).

12. Delete the topic when done. From the top-level lab directory, issue the `kafka-topics.sh` command to delete `test-topic`.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --delete
```

Key Takeaways

The only way to prevent data loss is to replicate topics and set their `min.insync.replicas` to two or more and ensure producers request acknowledgments from all replicas up to that number.

If you want the cluster to be fault-tolerant at the same time, make sure the topic's minimum ISR setting is at least one less than the total number of available replicas.

This concludes the guided exercise.

3.4. Guided Exercise Idempotent Broker

Turn on producer idempotency protocol to prevent message duplication.

Outcomes

- Turn off producer idempotence protocol to see duplicate messages.

Prerequisites

Ensure that you have successfully completed the first exercise on setting up the lab environment. Your Zookeeper and all three brokers should be running.

Instructions

1. Remove any payload logs that might have been left over from before. From the top-level lab directory, run the `remove-payload-logs.sh` script.

```
$ ./streams-bf-lab-materials/code/remove-payload-logs.sh
Removing payload logs in producer/consumer working directories...
./streams-bf-lab-materials/code/core-api-producer/payload.log
./streams-bf-lab-materials/code/core-api-consumer/payload.log
Done.
```

2. Create a replicated topic with multiple partitions and `min.insync.replicas` setting. In the same terminal, invoke the `kafka-topics.sh` script with `--create` command.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --partitions 6 --replication-factor 3 \
    --config min.insync.replicas=2 --create
Created topic test-topic.
```

3. Start the producer with idempotence protocol turned off. Set the maximum number of unacknowledged batches very high. Allow incremental batching of records by setting linger to one second. The acknowledgment setting is not strictly necessary, but it slightly improves the chances of seeing duplicates. Execute the below command in the producer source code directory.

```
$ mvn exec:java -Dproducer.num-records-per-roll=500000 \
    -Dproducer.idempotent=false -Dproducer.max-inflight=1000 \
    -Dproducer.linger=1000 -Dproducer.acks=1
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
...
```

4. While the producer is sending, stop and restart various brokers several times in rapid succession. The idea is to briefly interrupt sending, not to cause any longer outage.
5. When the producer finishes its sending, start the consumer and receive all the records.

```
$ mvn exec:java -Dconsumer.auto-offset-reset=earliest
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
...
```

6. When the consumer no longer shows any new messages, stop it cleanly.

```
$ echo quit | ./kafka/bin/kafka-console-producer.sh \
    --bootstrap-server localhost:9092 --topic test-topic
```

7. Collect the logs and compare them.

```
$ ../get-logs.sh
WARNING: Existing logs will be overwritten. Do you want to continue?
1) Yes
2) No
#? 1
../core-api-producer/payload.log
../core-api-consumer/payload.log
Producer and/or Consumer logs available in .. - old logs were removed.

$ diff ../producer.log ../consumer.log  | wc -l
    3317

$ diff ../producer.log ../consumer.log  | tail -n5
> test-topic,5,105680,"What makes the engine go? Desire, desire, desire." --
Stanley Kunitz, O Magazine, September 2003
417365a419034
> test-topic,5,105684,"We shall find peace. We shall hear the angels, we shall see
the sky sparkling with diamonds." -- Anton Chekhov (1860 - 1904), 1897
417366a419036
> test-topic,5,105688,"Fall seven times, stand up eight." -- Japanese Proverb
```

NOTE The first character on each line pointing to the right means that the record was found in `consumer.log`, but not in `producer.log`, meaning the same record was delivered to the consumer multiple times.

8. Turn on the producer idempotence protocol. Restart the application without any additional options beyond the number of records. Producer settings default to using the idempotence protocol.

```
$ mvn exec:java -Dproducer.num-records-per-roll=500000
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.producer.Producer.main()] INFO
com.redhat.training.kafka.coreapi.producer.Producer - Opening payload log...
...
```

9. While the producer is sending, stop and restart various brokers several times in rapid succession. The idea is to briefly interrupt sending, not to cause any longer outage.
10. When the producer finishes its sending, start the consumer and receive all the records.

```
$ mvn exec:java
[INFO] Scanning for projects...
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
...
```

11. When the consumer no longer shows any new messages, stop it cleanly.

```
$ echo quit | ./kafka/bin/kafka-console-producer.sh \
    --bootstrap-server localhost:9092 --topic test-topic
```

12. Collect the logs and compare them.

```
$ ../get-logs.sh
WARNING: Existing logs will be overwritten. Do you want to continue?
1) Yes
2) No
#? 1
../core-api-producer/payload.log
../core-api-consumer/payload.log
Producer and/or Consumer logs available in .. - old logs were removed.

$ diff ../producer.log ../consumer.log  | wc -l
    0
```

Note that there are no duplicate records this time.

13. Delete the topic. From the top-level lab directory, issue the kafka-topics.sh command to delete test-topic.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --delete
```

Key Takeaways

Producer idempotence protocol ensures that brokers can recognise and throw away any records that the producer has already sent, but still allows them to acknowledge these records. This effectively prevents duplicate message delivery and ensures at-most-once delivery guarantee.

Together with topic replication, correctly configured `min.insync.replicas`, and producer setting `acks=all`, this allows for an exactly-once delivery.

This concludes this guided exercise.

Chapter 4. Consumer Delivery Semantics

Goal

Refresh consumer delivery options and side effects.

Sections

4.1. Guided Exercise Committing Offsets Too Early

Observe how too frequent offset commit affects delivery guarantee.

Outcomes

- Experience message loss with large batches and frequent offset commit.

Prerequisites

Ensure that you have successfully completed the first exercise on setting up the lab environment. Your Zookeeper and all three brokers should be running.

Instructions

1. Remove any payload logs that might have been left over from before. From the top-level lab directory, run the `remove-payload-logs.sh` script.

```
$ ./streams-bf-lab-materials/code/remove-payload-logs.sh
Removing payload logs in producer/consumer working directories...
./streams-bf-lab-materials/code/core-api-producer/payload.log
./streams-bf-lab-materials/code/core-api-consumer/payload.log
Done.
```

2. Create a topic that will be used in this exercise. In the same terminal, invoke the `kafka-topics.sh` script with `--create` command.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --partitions 6 --create
Created topic test-topic.
```

3. Run the producer, sending 1500 records to the topic.

```
$ mvn exec:java -Dproducer.num-records-per-roll=1500
...
```

4. Run the consumer. Include the `instance-id` property - this just ensures we do not have to wait for rebalance upon restart. Include the `local-id` property - this makes the consumer use a separate payload log every time we change this property. Instruct the consumer to manually commit offsets every 50 processed records. Make it wait for 200 ms after each processed record, to slow it down a bit and allow you to act while it is receiving.

   After you see a `Seen 50 records, committing offsets` message in the log, press `Ctrl-C` in the console to interrupt it.

```
$ mvn exec:java -Dconsumer.auto-offset-reset=earliest \
    -Dconsumer.ack-every-x-msgs=50 -Dconsumer.wait-after-record=200 \
    -Dconsumer.instance-id=12345 -Dconsumer.local-id=1
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.redhat.training.kafka:core-api-consumer >-------------
[INFO] Building core-api-consumer 1.0.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ core-api-consumer ---
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Opening payload log...
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Received 500 records.
Processing.
...
[com.redhat.training.kafka.coreapi.consumer.Consumer.main()] INFO
com.redhat.training.kafka.coreapi.consumer.Consumer - Seen 50 records, committing
offsets as ackEveryNum == 50
...
^C
```

5. Ensure the payload log recorded the processed records.

```
$ ls -l payload-1.log
-rw-r--r--@ 1 johndoe  staff 14989 16 Sep 12:12 payload-1.log

$ wc -l payload-1.log
      53 payload-1.log
```

6. Restart the consumer without any special options (except `auto-offset-reset`, `instance-id` and `local-id`). The `auto-offset-reset` option ensures we restart receiving from the last committed offset. Ensure `local-id` is different from the previous run. Let it receive all the records.

```
$ mvn exec:java -Dconsumer.auto-offset-reset=earliest \
    -Dconsumer.instance-id=12345 -Dconsumer.local-id=2
...
```

7. Cleanly shut down the consumer when it stops printing records - send it a `quit` message.

```
$ echo quit | ./kafka/bin/kafka-console-producer.sh \
    --bootstrap-server localhost:9092 --topic test-topic
```

8. Collect the logs and compare them.

```
$ ../get-logs.sh
WARNING: Existing logs will be overwritten. Do you want to continue?
1) Yes
2) No
#? 1
../core-api-producer/payload.log
../core-api-consumer/payload-1.log
../core-api-consumer/payload-2.log
Producer and/or Consumer logs available in .. - old logs were removed.

$ diff ../producer.log ../consumer.log | wc -l
     497

$ diff ../producer.log ../consumer.log | tail -n5
< test-topic,5,9980,"Any fool can tell the truth, but it requires a man of some
sense to know how to lie well." -- Samuel Butler (1835 - 1902)
< test-topic,5,9986,"Do not pursue what is illusory - property and position: all
that is gained at the expense of your nerves decade after decade and can be
confiscated in one fell night. Live with a steady superiority over life - don't be
afraid of misfortune, and do not yearn after happiness; it is after all, all the
same: the bitter doesn't last forever, and the sweet never fills the cup to
overflowing." -- Alexander Solzhenitsyn (1918 - )
< test-topic,5,9988,"I am here and you will know that I am the best and will hear
me." -- Leontyne Price, O Magazine, December 2003
< test-topic,5,9990,"It's going to come true like you knew it, but it's not going
to feel like you think." -- Rosie O'Donnell, Today Show interview, 04-08-08
< test-topic,5,9992,"The nation behaves well if it treats the natural resources as
assets which it must turn over to the next generation increased, and not impaired,
in value." -- Theodore Roosevelt (1858 - 1919), Speech before the Colorado Live
Stock Association, Denver, Colorado, August 19, 1910
```

NOTE Your number of lost records will be different, depending on the first received batch size and the amount of time you waited before interrupting the consumer.

9. Delete the topic. From the top-level lab directory, issue the `kafka-topics.sh` command to delete `test-topic`.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --delete
```

Key Takeaways

Receiving large batches of records and acknowledging reception too early (because auto-commit interval is set too frequently) can lead to data loss.

This concludes the guided exercise.

4.2. Guided Exercise Committing Offsets Too Late

Observe how lazy offset commit affects delivery guarantee.

Outcomes

- Experience duplicate messages with lazy offset commits after a rebalance.

Prerequisites

Ensure that you have successfully completed the first exercise on setting up the lab environment. Your Zookeeper and all three brokers should be running.

Instructions

1. Stop all the brokers. You can leave Zookeeper running. At the top-level lab directory, use `kafka-server-stop.sh` script to shut them down.

```
$ ./kafka/bin/kafka-server-stop.sh broker0.properties
$ ./kafka/bin/kafka-server-stop.sh broker1.properties
$ ./kafka/bin/kafka-server-stop.sh broker2.properties
```

2. Add lower allowances for minimum consumer heartbeat and session timeout. In all three broker property files, add the following lines at the end.

```
group.consumer.min.heartbeat.interval.ms=500
group.consumer.min.session.timeout.ms=500
group.min.session.timeout.ms=500
```

3. Restart all three brokers.
4. After all the brokers are running, verify they are using correct configuration. In the lab top-level directory, use the `kafka-configs.sh` command to interrogate about currently active settings.

```
$ ./kafka/bin/kafka-configs.sh --bootstrap-server localhost:9092 \
    --entity-type brokers --all --describe | \
    grep -E "(^All|group.*\\.min\\.(heartbeat|session))"
All configs for broker 0 are:
  group.consumer.min.heartbeat.interval.ms=500 sensitive=false
synonyms={STATIC_BROKER_CONFIG:group.consumer.min.heartbeat.interval.ms=500,
DEFAULT_CONFIG:group.consumer.min.heartbeat.interval.ms=5000}
  group.consumer.min.session.timeout.ms=500 sensitive=false
synonyms={STATIC_BROKER_CONFIG:group.consumer.min.session.timeout.ms=500,
DEFAULT_CONFIG:group.consumer.min.session.timeout.ms=45000}
  group.min.session.timeout.ms=500 sensitive=false
synonyms={STATIC_BROKER_CONFIG:group.min.session.timeout.ms=500,
DEFAULT_CONFIG:group.min.session.timeout.ms=6000}
```

```
All configs for broker 1 are:
...
All configs for broker 2 are:
...
```

Ensure all three brokers report the same settings for the above properties.

5. Create a topic with multiple partitions. In the same directory, use the `kafka-topics.sh` script to create the `test-topic`.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --partitions 6 --create
Created topic test-topic.
```

6. Start the producer to send 50000 records to the topic.

```
$ mvn exec:java -Dproducer.num-records-per-roll=50000
...
```

7. Start a consumer. Set its auto-commit interval to something very lazy. Also ensure that a rebalance will happen fairly quickly when we add another consumer in the next step. Slow the consumer down a little bit to give yourself more time to repeat the command in another window in the next step.

```
$ mvn exec:java -Dconsumer.auto-offset-reset=earliest \
    -Dconsumer.ac-interval=30000 -Dconsumer.assignment-strategy=rr \
    -Dconsumer.session-timeout=1000 -Dconsumer.heartbeat-interval=500 \
    -Dconsumer.wait-after-batch=2500 -Dconsumer.fetch-min-bytes=16374 \
    -Dconsumer.max-poll-recs=5000 -Dconsumer.payload-trunc=false \
    -Dconsumer.local-id=1
```

8. In another terminal window, repeat the same command. Only change the consumer's `local-id` property.

```
$ mvn exec:java -Dconsumer.auto-offset-reset=earliest \
    -Dconsumer.ac-interval=30000 -Dconsumer.assignment-strategy=rr \
    -Dconsumer.session-timeout=1000 -Dconsumer.heartbeat-interval=500 \
    -Dconsumer.wait-after-batch=2500 -Dconsumer.fetch-min-bytes=16374 \
    -Dconsumer.max-poll-recs=5000 -Dconsumer.payload-trunc=false \
    -Dconsumer.local-id=2
```

IMPORTANT You must issue the second command while the first consumer is still receiving messages. Increase the `wait-after-batch` value if the first consumer finishes too quickly.

9. While both consumers are receiving messages, stop one of them for at least 3-4 seconds and then restart it, to force two more repartitioning events.
10. After both consumers stop printing new records on the console, send two quit messages to terminate them cleanly.

```
$ echo quit | ./kafka/bin/kafka-console-producer.sh \
    --bootstrap-server localhost:9092 --topic test-topic

$ echo quit | ./kafka/bin/kafka-console-producer.sh \
    --bootstrap-server localhost:9092 --topic test-topic
```

11. Collect the payload logs and compare them.

```
$ ../get-logs.sh
WARNING: Existing logs will be overwritten. Do you want to continue?
1) Yes
2) No
#? 1
../core-api-producer/payload.log
../core-api-consumer/payload-1.log
../core-api-consumer/payload-2.log
Producer and/or Consumer logs available in .. - old logs were removed.

$ diff ../producer.log ../consumer.log | wc -l
    9977

$ diff ../producer.log ../consumer.log | tail -n5
> test-topic,3,9984,"We shall find peace. We shall hear the angels, we shall see
the sky sparkling with diamonds." -- Anton Chekhov (1860 - 1904), 1897
33373a38372
> test-topic,3,999,"Your primary goal should be to have a great life. You can still
have a good day, enjoy your child, and ultimately find happiness, whether your ex
is acting like a jerk or a responsible person. Your happiness is not dependent upon
someone else." -- Julie A., M.A. Ross and Judy Corcoran, Joint Custody with a Jerk:
Raising a Child with an Uncooperative Ex, 2011
33375a38375
> test-topic,3,9995,"I wish you sunshine on your path and storms to season your
journey. I wish you peace in the world in which you live... More I cannot wish you
except perhaps love to make all the rest worthwhile." -- Robert A. Ward
```

12. Delete the topic. From the top-level lab directory, issue the `kafka-topics.sh` command to delete `test-topic`.

```
$ ./kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 \
    --topic test-topic --delete
```

Key Takeaways

Acknowledging multiple records after processing them might expose the consumer to a rebalance event, in which case it will potentially be unable to commit its offsets to the partitions it was assigned to *prior to rebalance*. The brokers will treat those records as undelivered, sending them to consumers again, in effect causing duplicate messages to be seen on the consumer side.

This concludes this guided exercise.