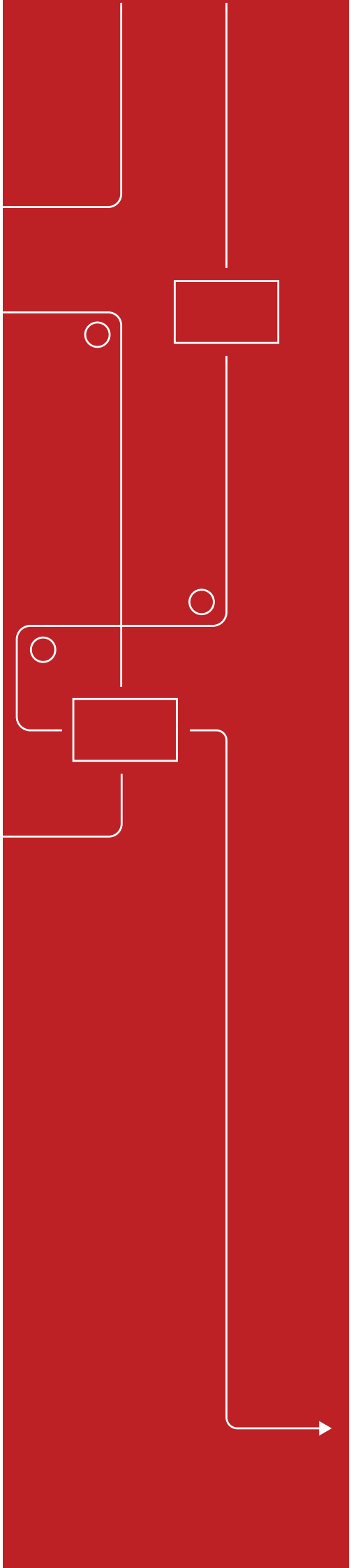

Red Hat
Training and Certification

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**

Joyent Node.js open source or commercial project.

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 for the labs.
- Test operation of the environment.

Instructions

1. Ensure prerequisites are on your system:
   - Java 17 (or 21) SDK
   - Streams for Apache Kafka 2.7.0
   - some sort of IDE (such as Visual Studio 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. Rename the directory to just `kafka` for easier use.

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

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

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

   3. 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 an error saying the files can not be cloned, use the `-b main` option to force Git to switch to the `main` branch immediately.

4. 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 .

$ 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/
-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, always do it in reverse order (broker2 first, etc., then Zookeeper last).

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 this 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.

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.

It supports the following application-specific 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`
- `producer.wait-after-send`, the amount of time (ms) to wait after each send, defaults to `500`

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

- `producer.acks`, acknowledgments to request, defaults to `all`
- `producer.max-inflight`, maximum unacknowledged messages, 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`

Consumer

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

It 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 each batch is received, defaults to `0`
- `consumer.wait-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 name, defaults to `-1` (meaning no suffix)
- `consumer.payload-trunc`, when starting up, whether to truncate the payload log, defaults to `false`

The following consumer 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, 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.assignment-strategy`, partition assignment strategy, one of `cooperative`, `range`, `rr`, and `sticky` (default is `cooperative`)
- `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 disconnected for lack of heartbeat, defaults to `45000`
- `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`)

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
- `remove-logdirs.sh` will remove all broker and Zookeeper data, it should be executed from the same directory where the broker property files are

REFERENCES	Break-Fix Labs Client Apps and Tools on GitHub