Contents

Exercise1: Using Kafka's Command-Line Tools 2
Exercise2: Consuming from Multiple Partitions 10
Exercise3: Writing a Basic Producer 13
Exercise4: Writing a Basic Consumer 20
Exercise5: Accessing Previous Data 21
Exercise6: Using Kafka with Avro 25
Exercise7: Running Kafka Connect in Standalone Mode 35
Exercise8: Using the Kafka Connect REST API 41
Exercise9: Using the JDBC Connector 45
Exercise10: Writing a Kafka Streams Application 49

Exercise1: Using Kafka's Command-Line Tools

In this Exercise, you will start to become familiar with Kafka's command-line tools. You will:

- Verify Kafka is running
- Use a console program to publish a message
- Use a console program to consume a message

Verifying that Kafka is Running Correctly

There are various ways to verify that all of the Kafka system's daemons are running. It is important to verify that Kafka is functioning correctly before you start to use it. You may need to restart a daemon process during the class, if it has terminated for some reason.

1. Open the Terminal Emulator on the desktop.

2. Get the listing of Java processes:

```
[training@confluent-training-vm ~]$ sudo jps
1473 QuorumPeerMain
2210 Jps
1714 KafkaRestMain
1772 SchemaRegistryMain
1693 SupportedKafka
[training@confluent-training-vm ~]$
```

This will display a list of all Java processes running on the VM. The four Kafka-related processes you should see, and their Linux service names, are:

Java Process Name	Service Name
QuorumPeerMain	zookeeper
SupportedKafka	kafka-server
KafkaRestMain	kafka-rest
SchemaRegistryMain	schema-registry

3. If any of the four Java processes are not present, start the relevant service(s) by typing:

```
$ sudo service servicename start
```

Console Publishing and Subscribing

Kafka has built-in command line utilities to publish messages to a topic, and read messages from a topic. These are extremely useful to verify that Kafka is working correctly, and for testing and debugging.

4. Run the command:

```
$ kafka-console-producer
```

This will bring up a list of parameters that the kafka-console-producer program can receive. Take a moment to look through the options.

```
[training@confluent-training-vm ~]$ kafka-console-producer
Read data from standard input and publish it to Kafka.
Option                                   Description
------                                   -----------
--batch-size <Integer: size>             Number of messages to send in a single
                                           batch if they are not being sent
                                           synchronously. (default: 200)
--broker-list <broker-list>              REQUIRED: The broker list string in
                                           the form HOST1:PORT1,HOST2:PORT2.
--compression-codec [compression-codec]  The compression codec: either 'none',
                                           'gzip', 'snappy', or 'lz4'.If
                                           specified without value, then it
                                           defaults to 'gzip'
--key-serializer <encoder_class>         The class name of the message encoder
                                           implementation to use for
                                           serializing keys. (default: kafka.
                                           serializer.DefaultEncoder)
--line-reader <reader_class>             The class name of the class to use for
                                           reading lines from standard in. By
                                           default each line is read as a
                                           separate message. (default: kafka.
                                           tools.
                                           ConsoleProducer$LineMessageReader)
--max-block-ms <Long: max block on       The max time that the producer will
  send>                                    block for during a send request
                                           (default: 60000)
--max-memory-bytes <Long: total memory   The total memory used by the producer
  in bytes>                                to buffer records waiting to be sent
                                           to the server. (default: 33554432)
--max-partition-memory-bytes <Long:      The buffer size allocated for a
  memory in bytes per partition>           partition. When records are received
                                           which are smaller than this size the
                                           producer will attempt to
                                           optimistically group them together
                                           until this size is reached.
                                           (default: 16384)
```

5. Run kafka-console-producer again with the required arguments:

```
$ kafka-console-producer --broker-list
broker1:9092 --topic testing
```

Note: The kafka-console-producer program does not have a prompt character. It will simply wait for input.

```
[training@confluent-training-vm ~]$ kafka-console-producer --broker-list broker1:9092 --topic
testing
```

6. Type:

```
some data
```

And hit 'Enter'. This will publish a message with the value some data to the testing topic. You will see the following output the first time you enter a line of data:

WARN Error while fetching metadata with correlation id 0 :
{testing=LEADER_NOT_AVAILABLE}
(org.apache.kafka.clients.NetworkClient)

This is because the Topic testing does not initially exist. When the first line of text is published to it from kafka-console-producer, it will automatically be created.

```
[training@confluent-training-vm ~]$ kafka-console-producer --broker-list broker1:9092 --topic
testing
Hello
[2017-09-22 20:31:26,374] WARN Error while fetching metadata with correlation id 0 : {testing=
LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
```

7. Type:

```
more data
```

And hit 'Enter'.

```
[training@confluent-training-vm ~]$ kafka-console-producer --broker-list broker1:9092 --topic
testing
Hello
[2017-09-22 20:31:26,374] WARN Error while fetching metadata with correlation id 0 : {testing=
LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
How are you
```

8. Type:

```
final data
```

And hit 'Enter'.

```
[training@confluent-training-vm ~]$ kafka-console-producer --broker-list broker1:9092 --topic
testing
Hello
[2017-09-22 20:31:26,374] WARN Error while fetching metadata with correlation id 0 : {testing=
LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
How are you
What are you doing
```

9. Press Ctrl-c to exit the kafka-console-producer program.

10. Now we will use a Consumer to retrieve the data that was published.

Run the command:

```
$ kafka-console-consumer
```

This will bring up a list of parameters that the kafka-console-consumer can receive. Take a moment to look through the options.

```
[training@confluent-training-vm ~]$ kafka-console-consumer
The console consumer is a tool that reads data from Kafka and outputs it to standard output.
Option                                  Description
------                                  -----------
--blacklist <blacklist>                 Blacklist of topics to exclude from
                                          consumption.
--bootstrap-server <server to connect
  to>
--consumer.config <config file>         Consumer config properties file.
--csv-reporter-enabled                  If set, the CSV metrics reporter will
                                          be enabled
--delete-consumer-offsets               If specified, the consumer path in
                                          zookeeper is deleted when starting up
--enable-systest-events                 Log lifecycle events of the consumer
                                          in addition to logging consumed
                                          messages. (This is specific for
                                          system tests.)
--formatter <class>                     The name of a class to use for
                                          formatting kafka messages for
                                          display. (default: kafka.tools.
                                          DefaultMessageFormatter)
--from-beginning                        If the consumer does not already have
                                          an established offset to consume
                                          from, start with the earliest
                                          message present in the log rather
                                          than the latest message.
--key-deserializer <deserializer for
  key>
--max-messages <Integer: num_messages>  The maximum number of messages to
                                          consume before exiting. If not set,
                                          consumption is continual.
--metrics-dir <metrics directory>       If csv-reporter-enable is set, and
                                          this parameter isset, the csv
                                          metrics will be outputed here
--new-consumer                          Use the new consumer implementation.
--property <prop>                       The properties to initialize the
                                          message formatter.
```

11. Run kafka-console-consumer again with the following arguments:

```
$ kafka-console-consumer \
--bootstrap-server broker1:9092 \
--new-consumer \
--from-beginning \
--topic testing
```

You should see all the messages that you published using kafka-console-producer earlier.

```
[training@confluent-training-vm ~]$ kafka-console-consumer \
> --bootstrap-server broker1:9092 \
> --new-consumer \
> --from-beginning \
> --topic testing
Hello
How are you
What are you doing
```

12. Press Ctrl-c to exit kafka-console-consumer.

Do it Yourself:

13. The kafka-console-producer and kafka-console-consumer programs can be run at the same time. Run kafka-console-producer and kafka-console-consumer in separate terminal windows at the same time to see how kafka-console-consumer receives the events.

14. By default, kafka-console-producer and kafka-console-consumer assume null keys. They can also be run with appropriate arguments to write and read keys as well as values. Re-run the Producer with additional arguments to write (key,value) pairs to the topic:

```
$ kafka-console-producer \
--broker-list broker1:9092 \
--topic testing \
--property parse.key=true \
```

```
--property key.separator=,
```

(When you enter data, separate the key and the value with a comma.)

Then re-run the Consumer with additional arguments to print the key as well as the value:

```
$ kafka-console-consumer \
--bootstrap-server broker1:9092 \
--new-consumer \
--from-beginning \
--topic testing \
--property print.key=true
```

15. Kafka's data in ZooKeeper can be accessed using the zookeeper-shell command. Run it with:

```
$ zookeeper-shell zookeeper1
```

Use the ls command to view the directory structure in ZooKeeper.

```
ls /
```

Exercise2: Consuming from Multiple Partitions

In this Exercise, you will create a topic with multiple Partitions, produce data to those Partitions, and then read it back to observe issues with ordering.

1. Create a topic manually with Kafka's command-line tool, specifying that it should have two Partitions:

```
$ kafka-topics \
--zookeeper zookeeper1:2181 \
--create \
 --topic two-p-topic \
--partitions 2 \
--replication-factor 1
```

Created topic "two-p-topic"

```
[training@confluent-training-vm ~]$ kafka-topics \
> --zookeeper zookeeper1:2181 \
> --create \
> --topic two-p-topic \
> --partitions 2 \
> --replication-factor 1
Created topic "two-p-topic".
[training@confluent-training-vm ~]$
```

2. Use the command-line Producer to write several lines of data to the topic.

```
$ seq 1 100 > numlist
```

```
$ kafka-console-producer \
--broker-list broker1:9092 \
--topic two-p-topic < numlist
```

```
[training@confluent-training-vm ~]$ seq 1 100 > numlist
[training@confluent-training-vm ~]$ kafka-console-producer \
> --broker-list broker1:9092 \
> --topic two-p-topic < numlist
[training@confluent-training-vm ~]$
```

3. Use the command-line Consumer to read the topic

```
$ kafka-console-consumer \
--bootstrap-server broker1:9092 \
--new-consumer \
--from-beginning \
  --topic two-p-topic
```

```
[training@confluent-training-vm ~]$ kafka-console-consumer \
> --bootstrap-server broker1:9092 \
> --new-consumer \
> --from-beginning \
> --topic two-p-topic
```

4. Note the order of the numbers.

```
[training@confluent-training-vm ~]$ kafka-console-consumer \
> --bootstrap-server broker1:9092 \
> --new-consumer \
> --from-beginning \
> --topic two-p-topic
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
```

Rerun the Producer command in step 2, then rerun the Consumer command in step 3 and see if you observe any difference in the output.

```
[training@confluent-training-vm ~]$ kafka-console-producer --broker-list broker1:9092 --topic
two-p-topic < numlist
[training@confluent-training-vm ~]$ kafka-console-consumer --bootstrap-server broker1:9092 --n
ew-consumer --from-beginning --topic two-p-topic
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
```

5. What do you think is happening as the data is being written?

6. Try creating a new topic with three Partitions and running the same steps again.

Exercise3: Writing a Basic Producer

Project directory: helloworld or helloworld_python

In this Exercise you will write a Producer in either Java or Python.

Choosing a Language

In this Exercise, you will write programs using the Kafka API. The class has full solutions for Java, and for Python using the REST interface. You will need to choose which language to write your code in.

Once you have chosen your language, please follow the instructions for that language in the Exercise Manual. A heading with (Java) or (Python) means that section is for a specific language.

Using Eclipse (Java)

1. Go to the project directory (specified at the beginning of the Exercise description).

```
[training@confluent-training-vm helloworld]$ pwd
/home/training/developer/exercise_code/helloworld
[training@confluent-training-vm helloworld]$ ls -ltr
total 8
-rwxr-x--- 1 training training 1740 Sep 25  2016 pom.xml
drwxr-x--- 3 training training 4096 Sep 25  2016 src
[training@confluent-training-vm helloworld]$
```

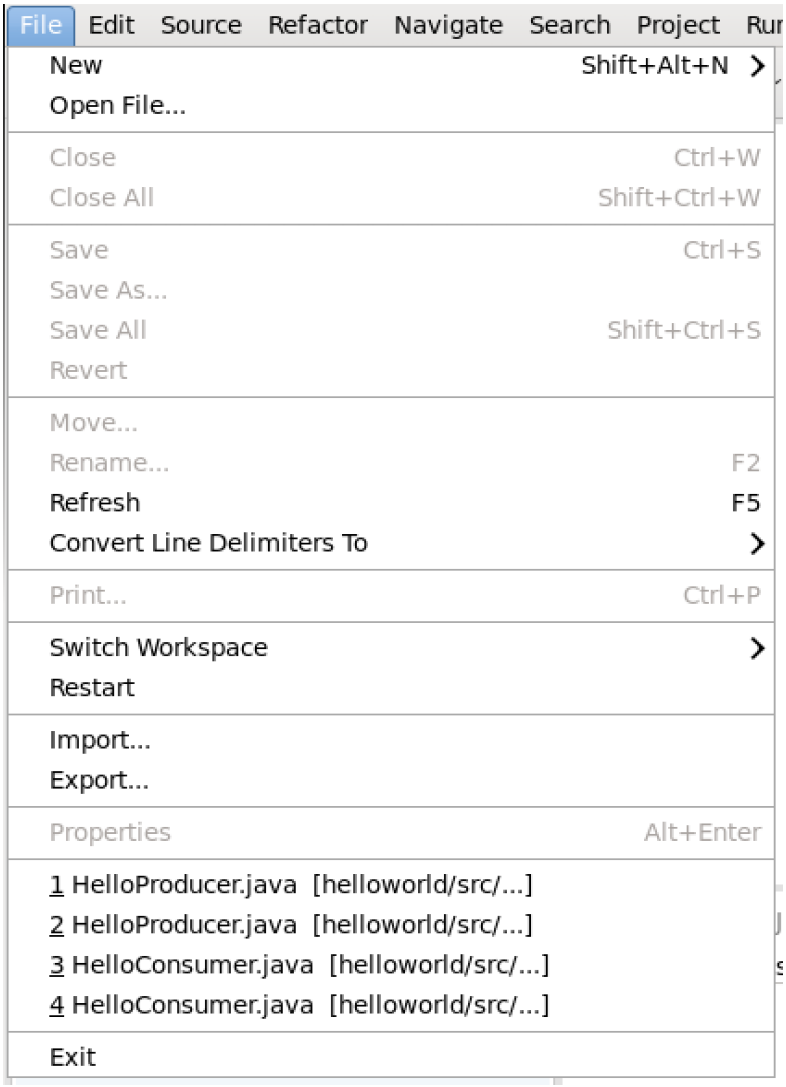
2. Create the Eclipse project with Maven:

```
$ mvn eclipse:eclipse
```

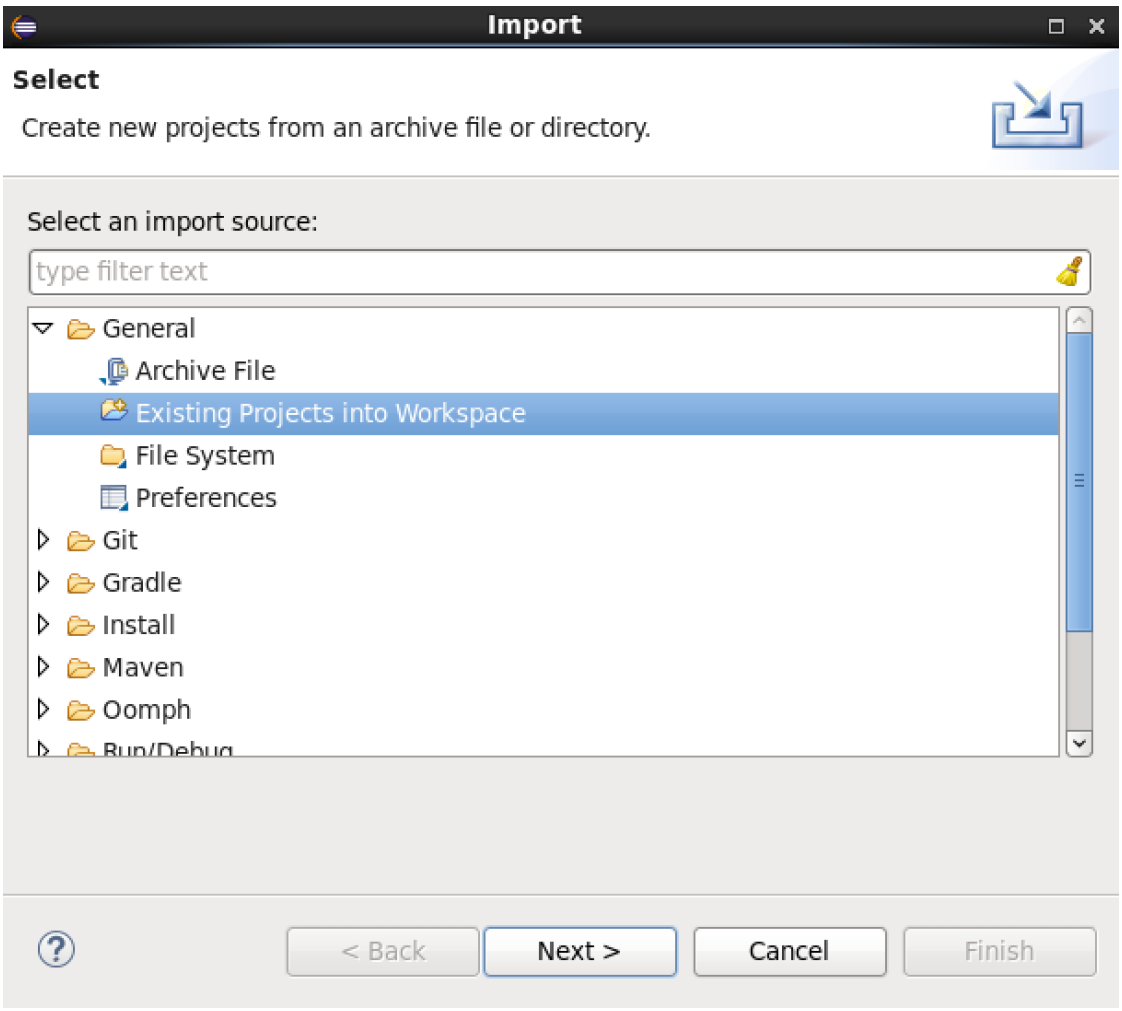
```
[training@confluent-training-vm helloworld]$ mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building helloworld 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-eclipse-plugin:2.9:eclipse (default-cli) > generate-resources @ helloworld >
>
[INFO]
[INFO] <<< maven-eclipse-plugin:2.9:eclipse (default-cli) < generate-resources @ helloworld <
<
[INFO]
[INFO] --- maven-eclipse-plugin:2.9:eclipse (default-cli) @ helloworld ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Wrote Eclipse project for "helloworld" to /home/training/developer/exercise_code/hello
orld.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.577 s
[INFO] Finished at: 2017-09-22T21:10:56-05:00
[INFO] Final Memory: 8M/44M
[INFO] ------------------------------------------------------------------------
[training@confluent-training-vm helloworld]$
```

3. Open Eclipse.

4. Go to File→Import.



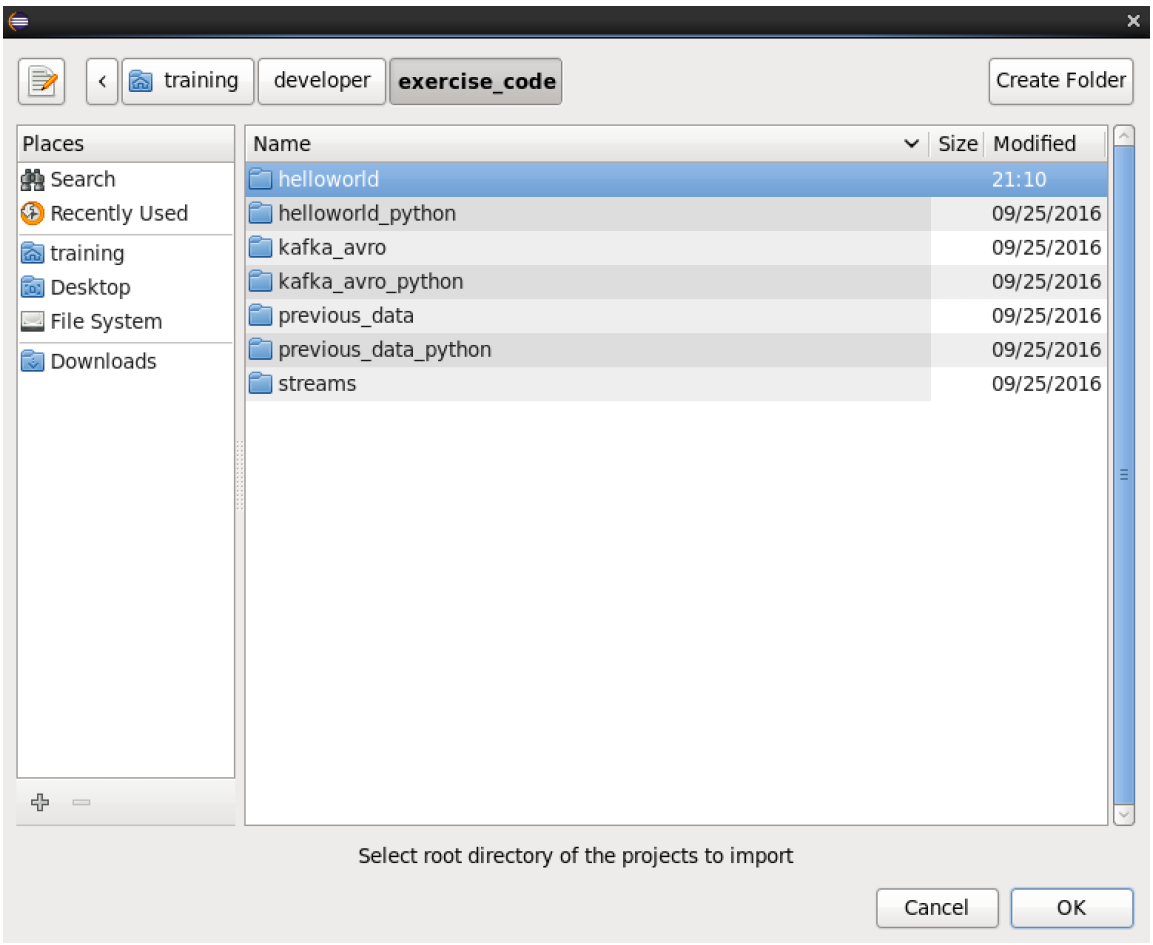
5. Choose General→Existing Projects into Workspace.



6. Click on Next.

7. To select the project's root directory, click on Browse.

8. Navigate to`/home/training/developer/exercise_code/helloworld`



9. Click on OK.

The Projects section of the Eclipse screen will update and add the HelloWorld project.

10. Click on Finish.

Note: You will need to follow a similar procedure for each of the subsequent programming Hands-On Exercises in order to create the project and import it into Eclipse.

Using Geany (Python)

The VM includes a Python editor named Geany. You can use this editor to make it easier to write Python programs. Launch it by going to the Applications menu at the top left of the screen, and choose the Programming sub-menu. Of course, if you prefer you can use a text editor such as vi.

Creating a Producer (Java)

11. Create a KafkaProducer with the following characteristics:

- Connects to broker1:9092
- Sends five messages to a topic named hello_world_topic
- Sends all messages as type String

Creating a Producer (Python)

12. Create a Producer with the following characteristics:

- Sends five messages
- Connects to kafkarest1:8082
- Sends five messages to a topic named hello_world_topic
- Sends all messages as strings
- Checks for HTTP error codes and outputs the error message if one was returned

Testing Your Code

13. To test your code, run the command-line Consumer and have it consume from hello_world_topic.

Do not attempt to write the Consumer code yet – we will do that in the next Hands-On Exercise.

14. Append --property print.key=true to the command-line Consumer to print the key as well as the value.

```
$ kafka-console-consumer \
--bootstrap-server broker1:9092 \
--new-consumer \
--from-beginning \
--topic hello_world_topic \
--property print.key=true
```

```
[training@confluent-training-vm helloworld]$
[training@confluent-training-vm helloworld]$ $ kafka-console-consumer \
> --bootstrap-server broker1:9092 \
> --new-consumer \
> --from-beginning \
> --topic hello_world_topic \
> --property print.key=true
```

```
[training@confluent-training-vm helloworld]$ kafka-console-consumer --bootstrap-server broker
1:9092 --new-consumer --from-beginning --topic hello_world_topic --property print.key=true
[2017-09-22 21:20:09,433] WARN Error while fetching metadata with correlation id 1 : {hello_w
orld_topic=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
firstkey        firstvalue:0
firstkey        firstvalue:1
firstkey        firstvalue:2
firstkey        firstvalue:3
firstkey        firstvalue:4
```

Exercise4: Writing a Basic Consumer

Project directory: helloworld or helloworld_python

In this Exercise, you will write a basic Consumer in either Java or Python. It will consume from the same topic hello_world_topic that the producer in the previous exercise wrote to. By default, the consumer will start reading from the end of the topic, so you may need to execute the producer in the previous exercise again to write new messages to the topic.

```
2017-09-22 21:23:04 INFO  ConsumerCoordinator:219 - Setting newly assigned partitions [hello_world_topic-0]
offset = 5, key = firstkey, value = firstvalue:0
offset = 6, key = firstkey, value = firstvalue:1
offset = 7, key = firstkey, value = firstvalue:2
offset = 8, key = firstkey, value = firstvalue:3
offset = 9, key = firstkey, value = firstvalue:4
```

Creating a Consumer (Java)

1. Write a Java Consumer to read the data you wrote with the Producer you created in the previous Exercise. If you did not have time to finish that Exercise, run the sample solution to write some messages to the topic first.

```
2017-09-22 21:23:04 INFO  AbstractCoordinator:434 - Successfully joined group testgroup with generation 1
2017-09-22 21:23:04 INFO  ConsumerCoordinator:219 - Setting newly assigned partitions [hello_world_topic-0]
offset = 5, key = firstkey, value = firstvalue:0
offset = 6, key = firstkey, value = firstvalue:1
offset = 7, key = firstkey, value = firstvalue:2
offset = 8, key = firstkey, value = firstvalue:3
offset = 9, key = firstkey, value = firstvalue:4
2017-09-22 21:30:58 INFO  ConsumerCoordinator:280 - Revoking previously assigned partitions [hello_world_to
2017-09-22 21:30:58 INFO  AbstractCoordinator:326 - (Re-)joining group testgroup
2017-09-22 21:30:58 INFO  AbstractCoordinator:434 - Successfully joined group testgroup with generation 2
```

Creating a Consumer (Python)

2. Write a Consumer to read the data you wrote with the Producer you created in the previous Exercise. If you did not have time to finish that Exercise, run the sample solution to write some messages to the topic first.

Exercise5: Accessing Previous Data

Project directory: previous_data or previous_data_python

In this Hands-On Exercise, you will create a Consumer which accesses data starting from the beginning of the topic each time it launches.

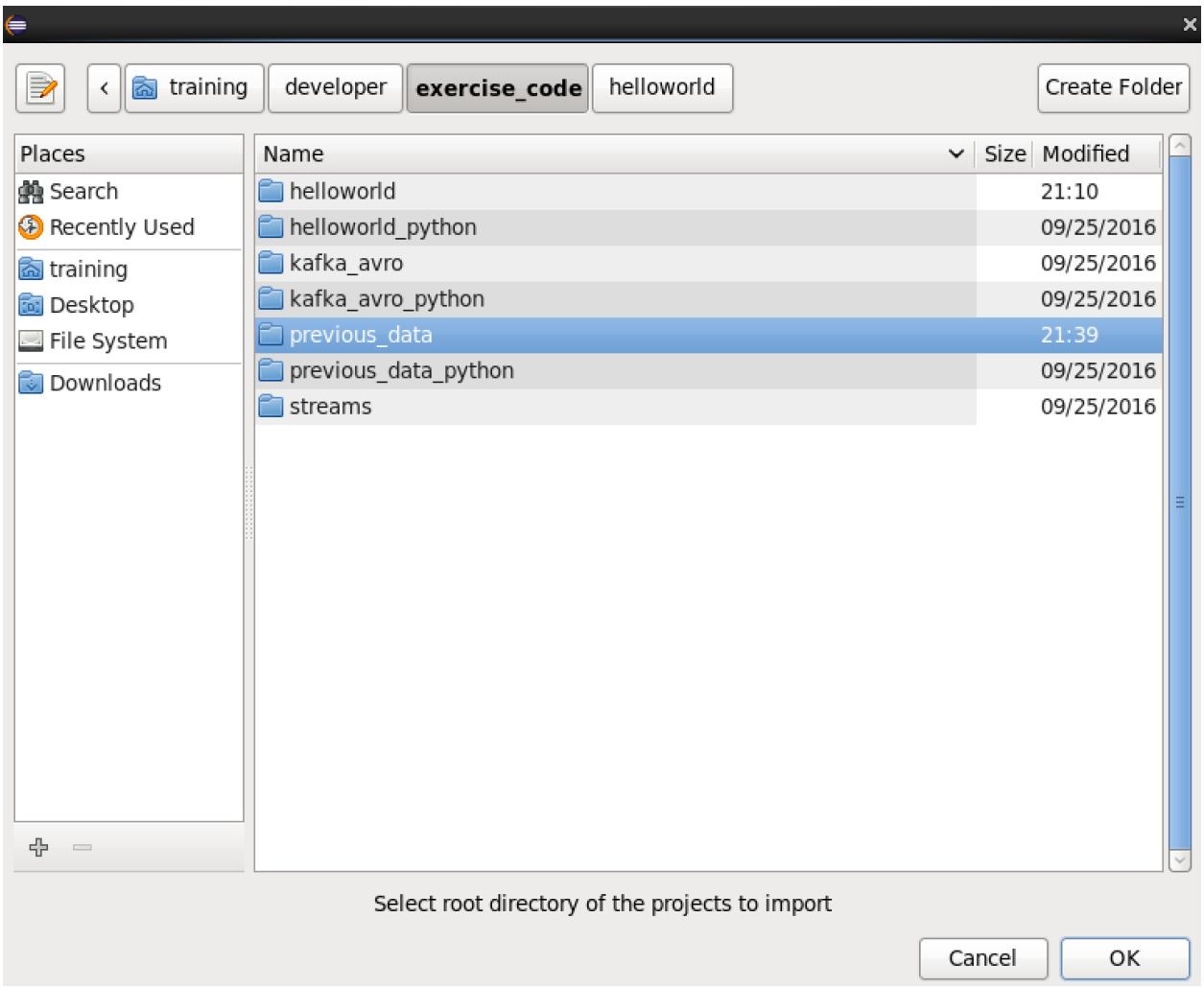
Change directory to previous_data:

```
[training@confluent-training-vm previous_data]$ pwd
/home/training/developer/exercise_code/previous_data
[training@confluent-training-vm previous_data]$ ls -ltr
total 8
-rwxr-x--- 1 training training 1930 Sep 25  2016 pom.xml
drwxr-x--- 3 training training 4096 Sep 25  2016 src
[training@confluent-training-vm previous_data]$
```

Create Eclipse project with Maven:

```
[training@confluent-training-vm previous_data]$ mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building previousdata 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-eclipse-plugin:2.9:eclipse (default-cli) > generate-resources @ previousdata
 >>>
[INFO]
[INFO] <<< maven-eclipse-plugin:2.9:eclipse (default-cli) < generate-resources @ previousdata
 <<<
[INFO]
[INFO] --- maven-eclipse-plugin:2.9:eclipse (default-cli) @ previousdata ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Wrote Eclipse project for "previousdata" to /home/training/developer/exercise_code/pre
vious_data.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.341 s
[INFO] Finished at: 2017-09-22T21:39:14-05:00
[INFO] Final Memory: 8M/44M
[INFO] ------------------------------------------------------------------------
[training@confluent-training-vm previous_data]$
```

Import the existing project in Eclipse workspace



Previous Data Consumer (Java)

1. Write a Consumer which reads all data from a topic each time it starts. Test it using the command-line Producer, or by reading from one of the topics you previously created.

```
2017-09-22 21:43:25 INFO  AbstractCoordinator:434 - Successfully joined group mygroup with generation 1
2017-09-22 21:43:25 INFO  ConsumerCoordinator:219 - Setting newly assigned partitions [hello_world_topic-0]
offset = 0, key = firstkey, value = firstvalue:0
offset = 1, key = firstkey, value = firstvalue:1
offset = 2, key = firstkey, value = firstvalue:2
offset = 3, key = firstkey, value = firstvalue:3
offset = 4, key = firstkey, value = firstvalue:4
offset = 5, key = firstkey, value = firstvalue:0
offset = 6, key = firstkey, value = firstvalue:1
offset = 7, key = firstkey, value = firstvalue:2
offset = 8, key = firstkey, value = firstvalue:3
offset = 9, key = firstkey, value = firstvalue:4
2017-09-22 21:45:40 INFO  ConsumerCoordinator:280 - Revoking previously assigned partitions [hello_world_to
2017-09-22 21:45:40 INFO  AbstractCoordinator:326 - (Re-)joining group mygroup
```

Previous Data Consumer (Python)

2. Write a Consumer which reads all data from a topic each time it starts. Test it using the command-line Producer, or by reading from one of the topics you previously created.

If You Have More Time 1 (Java)

3. Write a multi-threaded Consumer to read data from the two-p-topic topic you created in a previous exercise, and print it to the screen. As you print out the data, report which thread received the message. Investigate what happens if you use more threads than there are partitions in the topic.

If You Have More Time 1 (Python)

4. Write a Consumer which asks you before committing its offsets. If you say no, then the next time the Consumer runs it should re-read the previous messages.

If you have more time 2 (Java and Python)

5. Write a Producer (or modify an already existing one) to write a large number of messages to a topic. Make the messages sequential numbers (1, 2, 3...).

6. Write a Consumer which processes records and manually manages

offsets by writing them to files on disk; it should write the offset after each message. (For simplicity, use a separate file for each Partition of the topic.) To 'process' the message, just display it to the console. When the Consumer starts, it should seek to the correct offset such that it provides exactly-once processing. Modify your Consumer so that it halts randomly during processing, so you can confirm that it performs correctly when restarted.

If you have even more time (Java)

7. Create a topic with two Partitions. Write a Producer which uses a custom Partitioner; it should write random numbers between 1 and 20 to the topic. Send numbers between 1 and 10 to one Partition, and numbers between 11 and 20 to the other. Check that it works correctly by using the multi-threaded Consumer you wrote earlier to read back the data using two threads.

Exercise6: Using Kafka with Avro

Project directory: kafka_avro or kafka_avro_python

In this Hands-On Exercise, you will write and read Avro data to and from a Kafka cluster, and build a processing pipeline to turn text data from one topic into Avro in another. To do this, you will:

- Write a Producer
- Write a program with a Consumer that reads data, converts it to Avro, then uses a Producer to write that Avro data to a new topic

Write a Consumer that reads Avro data from a topic and displays it to the screen

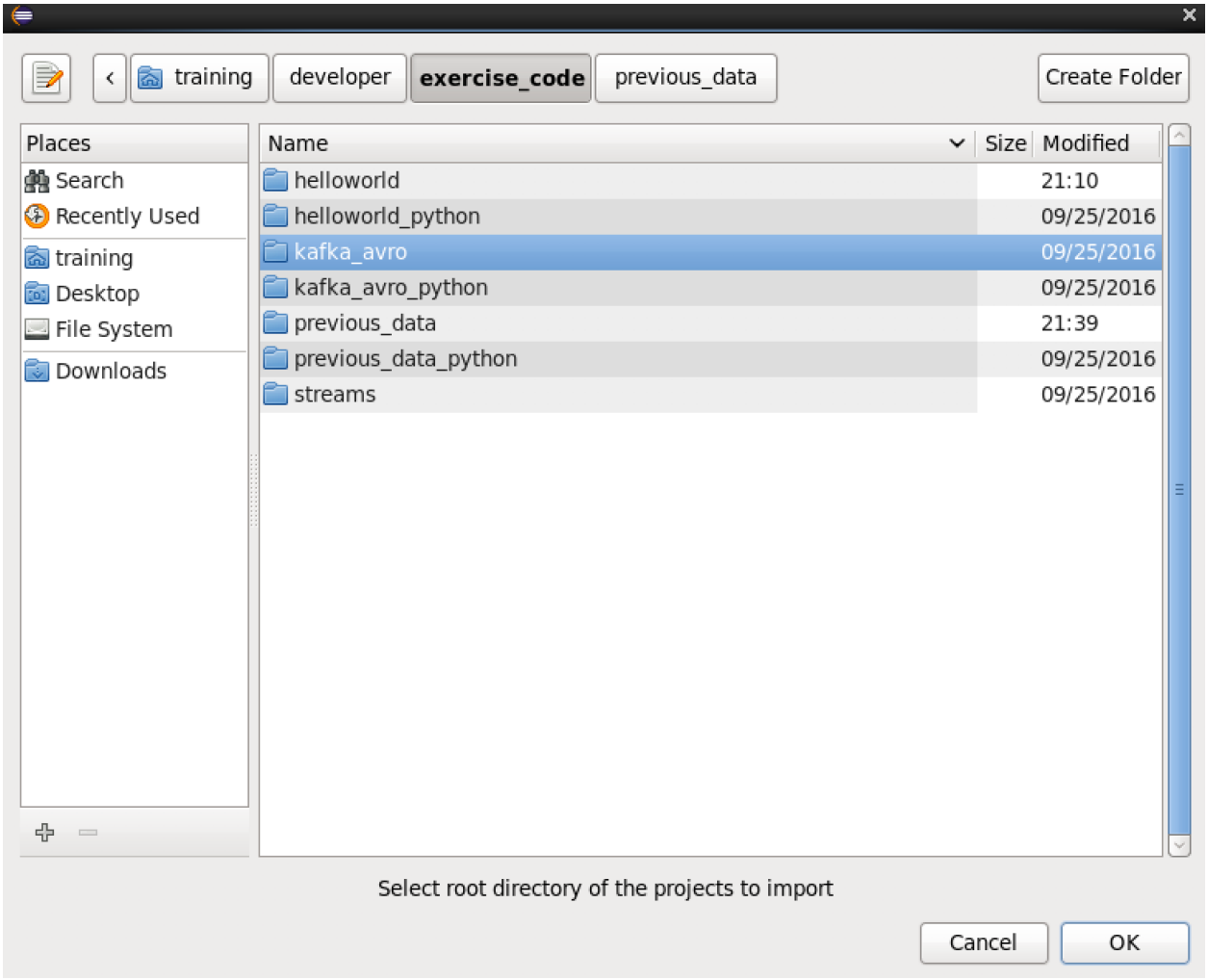
Change directory to kafka_avro

```
[training@confluent-training-vm kafka_avro]$ pwd
/home/training/developer/exercise_code/kafka_avro
[training@confluent-training-vm kafka_avro]$ ls -ltr
total 8
-rwxr-x--- 1 training training 2855 Sep 25  2016 pom.xml
drwxr-x--- 3 training training 4096 Sep 25  2016 src
```

Create Eclipse project using Maven

```
[training@confluent-training-vm kafka_avro]$ mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building kafkaavro 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-eclipse-plugin:2.9:eclipse (default-cli) > generate-resources @ kafkaavro >>>
[INFO]
[INFO] --- avro-maven-plugin:1.7.7:schema (default) @ kafkaavro ---
[INFO]
[INFO] <<< maven-eclipse-plugin:2.9:eclipse (default-cli) < generate-resources @ kafkaavro <<<
[INFO]
[INFO] --- maven-eclipse-plugin:2.9:eclipse (default-cli) @ kafkaavro ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Wrote Eclipse project for "kafkaavro" to /home/training/developer/exercise_code/kafka_av
ro.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.036 s
[INFO] Finished at: 2017-09-22T22:06:25-05:00
[INFO] Final Memory: 9M/44M
[INFO] ------------------------------------------------------------------------
[training@confluent-training-vm kafka_avro]$
```

Import the exiting project to Eclipse workspace



The Dataset

This exercise will be using a subset of Shakespeare's plays. The files are located in /home/training/developer/datasets/shakespeare.

Here is an example line from one of the files:

```
2360  Et tu, Brute?-- Then fall, Caesar!
```

Notice that the line of the play is preceded by the line number.

```
[training@confluent-training-vm shakespeare]$ pwd
/home/training/developer/datasets/shakespeare
[training@confluent-training-vm shakespeare]$ ls -ltr
total 1152
-rwxr-x--- 1 training training 249611 Sep 25  2016 Hamlet.txt
-rwxr-x--- 1 training training 172820 Sep 25  2016 Julius Caesar.txt
-rwxr-x--- 1 training training 149324 Sep 25  2016 Macbeth.txt
-rwxr-x--- 1 training training 167323 Sep 25  2016 Merchant of Venice.txt
-rwxr-x--- 1 training training 212369 Sep 25  2016 Othello.txt
-rwxr-x--- 1 training training 217807 Sep 25  2016 Romeo and Juliet.txt
[training@confluent-training-vm shakespeare]$
```

When Was That Play Written?

In the code you write, you will need to add the year that the play was written to the data. Here are the relevant years:

Hamlet: 1600

Julius Caesar: 1599

Macbeth: 1605

Merchant of Venice: 1596

Othello: 1604

Romeo and Juliet: 1594

Creating the Avro Schemas

You need to create an Avro schema for the key and value of the message.

1. Make sure you are in the right directory. The *.avsc schema files should be in the relative path `kafka_avro/src/main/avro`

```
[training@confluent-training-vm avro]$ pwd
/home/training/developer/exercise_code/kafka_avro/src/main/avro
[training@confluent-training-vm avro]$ ls -ltr
total 8
-rwxr-x--- 1 training training 248 Sep 25  2016 shakespeare_key_solution.avsc
-rwxr-x--- 1 training training 255 Sep 25  2016 shakespeare_value_solution.avsc
```

2. Create a schema for the key with following characteristics:

- The name should be ShakespeareKey
  • A unique namespace, e.g. partial.model
  • The schema should have a field for the name of the work, *e.g.*, Julius Caesar

- The schema should have a field for the year the work was published, *e.g.*, 1599

3. Create a schema for the value with following characteristics:

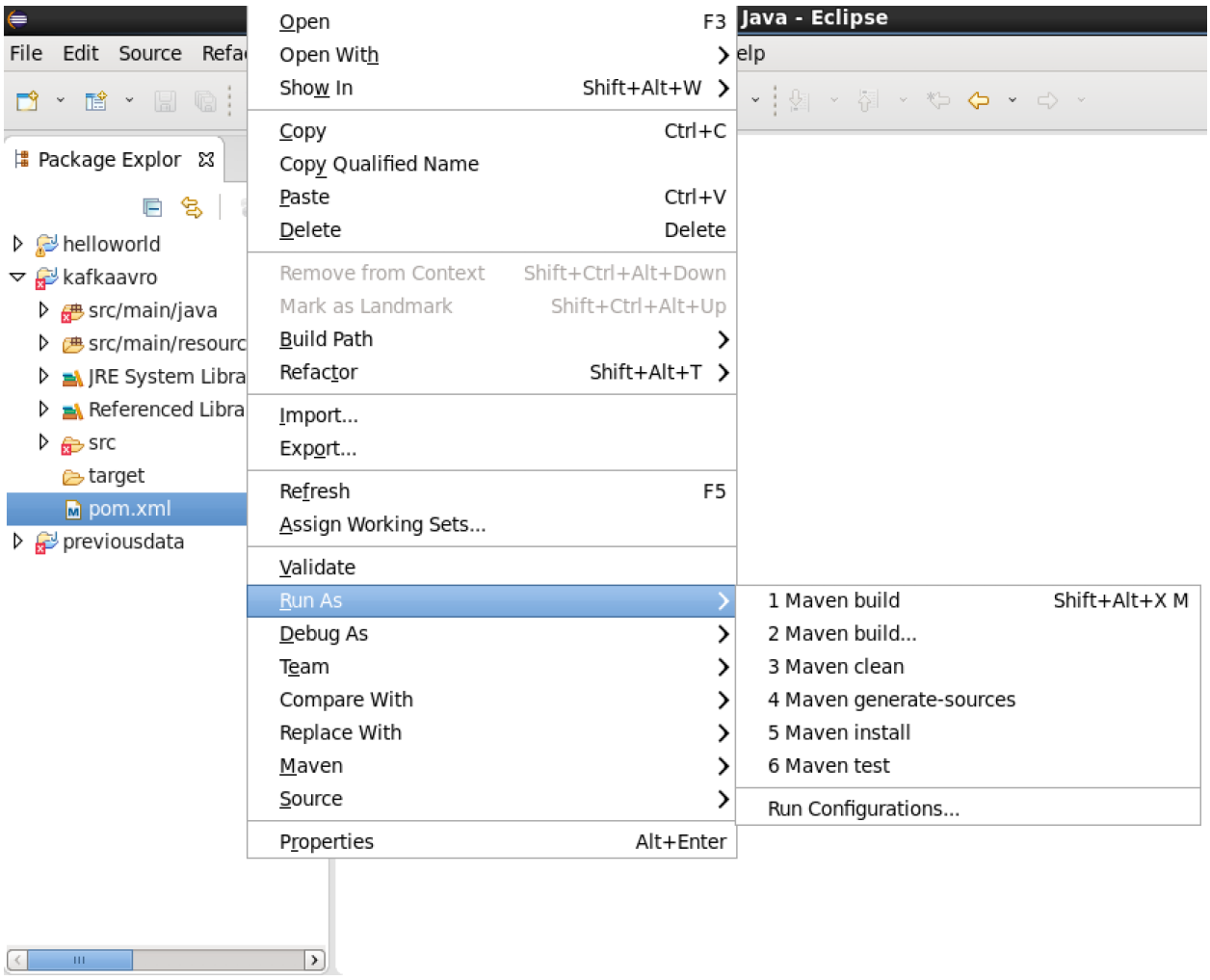
- The name should be ShakespeareValue

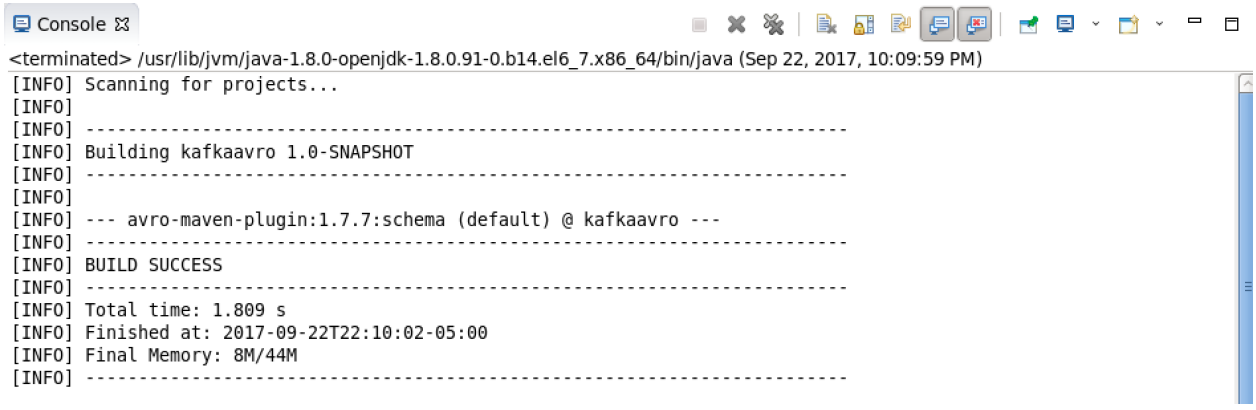
- A unique namespace, e.g. partial.model
  • The schema should have a field for the line number

- The schema should have a field for the line itself

4. Once you have created the schema files, generate the corresponding Java classes. From within Eclipse, right-click on the pom.xml and choose Run As → Maven generate-sources, or from the command line:

```
$ mvn generate-sources
```





5. Verify that the corresponding java files were created for the ShakespeareValue and ShakespeareKey Avro schemas. They are

located in the relative path `kafka_avro/src/main/java/.` From there the subdirectory corresponds to the namespace you provided in the schema file.

6. If you have already imported the kafka_avro project into Eclipse, refresh it by right-clicking on the project in the left-hand pane and selecting 'Refresh'.

Kafka Consumers and Producers (Java)

In this Exercise, you are creating a processing pipeline. A Publisher will read in every line of Shakespeare as a String and write it to a topic. A Consumer will read the data from that first topic, parse it, create Avro objects, and publish the Avro objects to a new topic. The final piece of the pipeline is a Consumer that reads the Avro objects from the topic and writes them to the screen with System.out.println.

7. Create a Producer with the following characteristics:

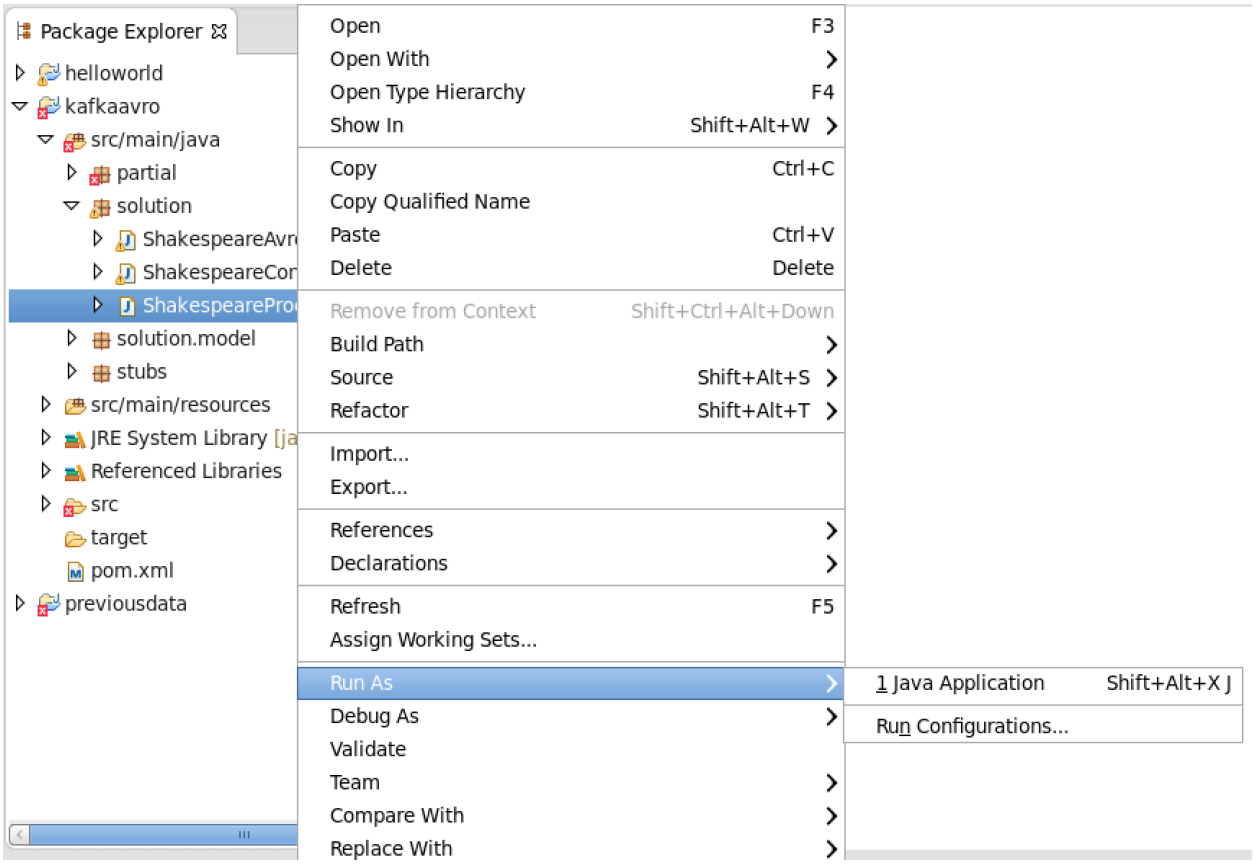
   - Connects to broker1:9092
   - Reads in the files from the shakespeare directory line by line
   - Sends each line as a separate message

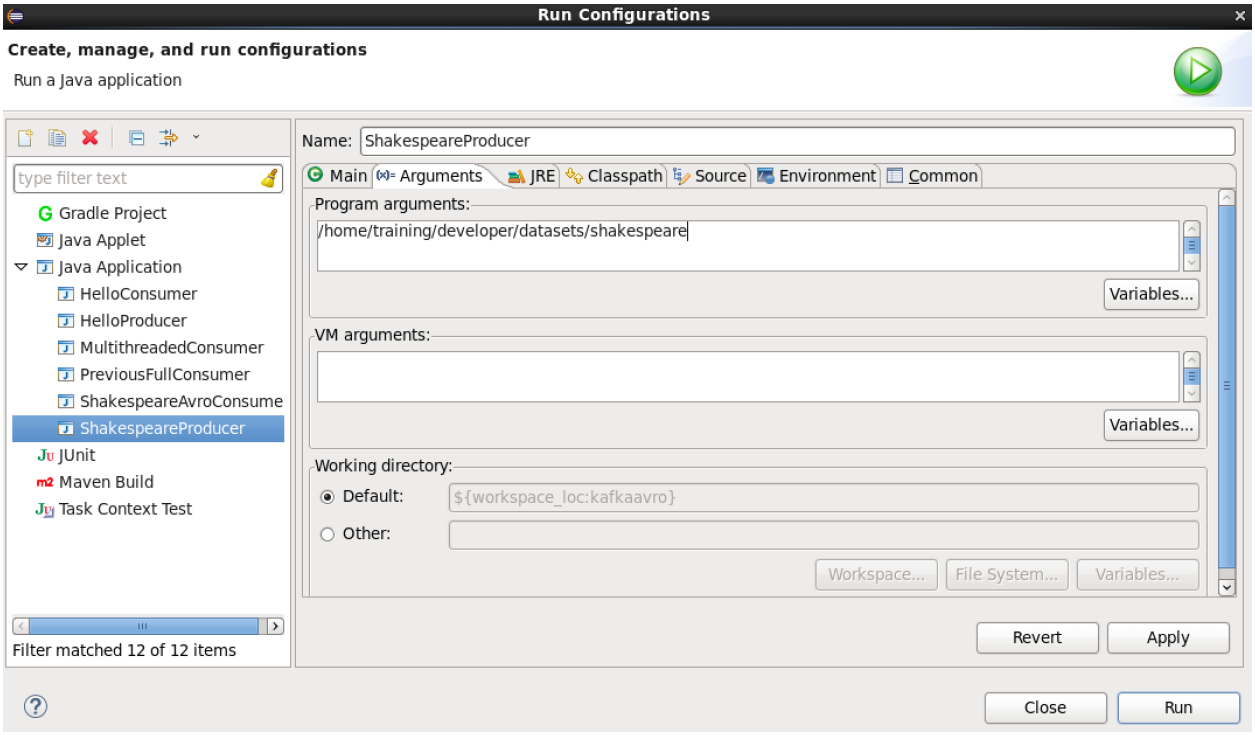
- Sends messages to a topic named shakespeare_topic
  - Sends all messages as type String

- The key should be the name of the play (which is contained in the filename)

- The value should be the line from the play

Note: If you run your Producer from within Eclipse, specify the source directory as an argument. Choose Run As → Run Configurations → Arguments and specify the directory name.


Package Explorer
helloworld
kafkaavro
src/main/java
partial
solution
ShakespeareAvr
ShakespeareCor
ShakespeareProd
solution.model
stubs
src/main/resources
JRE System Library [ja
Referenced Libraries
src
target
pom.xml
previousdata
Open F3
Open With
Open Type Hierarchy F4
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Shift+Ctrl+Alt+Down
Build Path
Source Shift+Alt+S
Refactor Shift+Alt+T
Import...
Export...
References
Declarations
Refresh F5
Assign Working Sets...
Run As
Debug As
Validate
Team
Compare With
Replace With
1 Java Application Shift+Alt+X J
Run Configurations...

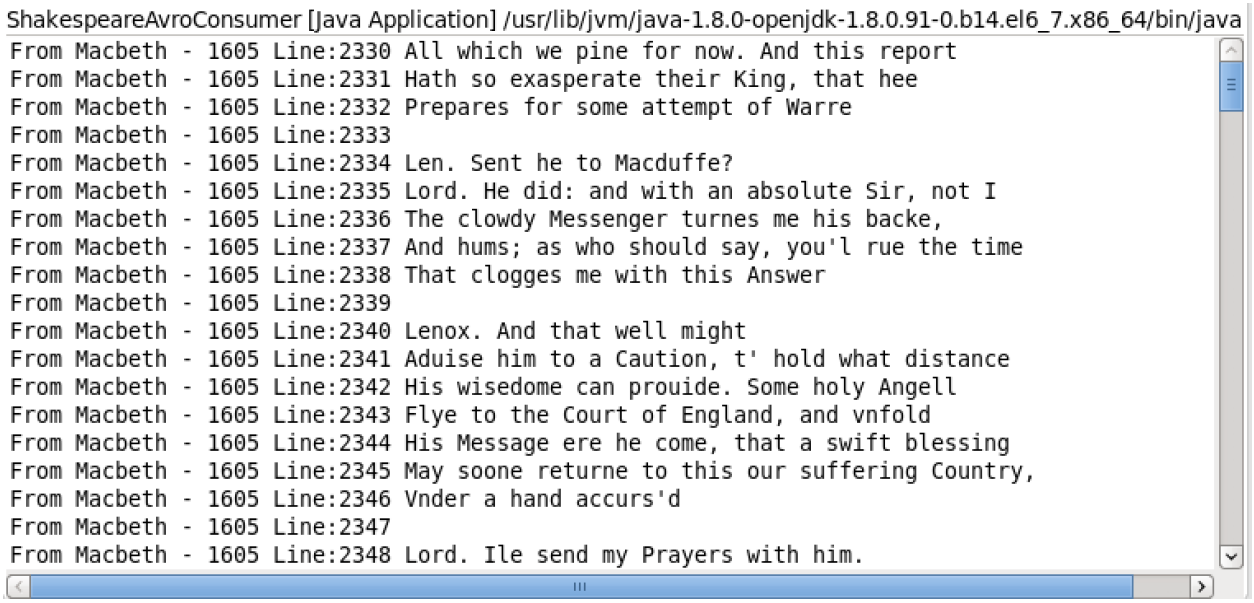

Run Configurations
Create, manage, and run configurations
Run a Java application
type filter text
Gradle Project
Java Applet
Java Application
HelloConsumer
HelloProducer
MultithreadedConsumer
PreviousFullConsumer
ShakespeareAvroConsume
ShakespeareProducer
JUnit
Maven Build
Task Context Test
Filter matched 12 of 12 items
Name: ShakespeareProducer
Main
Arguments
JRE
Classpath
Source
Environment
Common
Program arguments:
/home/training/developer/datasets/shakespeare
Variables...
VM arguments:
Variables...
Working directory:
Default:
${workspace_loc:kafkaavro}
Other:
Workspace...
File System...
Variables...
Revert
Apply
Close
Run

```
2017-09-22 22:25:37 INFO  AppInfoParser:83 - Kafka version : 0.10.0.0-cp1
2017-09-22 22:25:37 INFO  AppInfoParser:84 - Kafka commitId : 7c67d65755e5b1ab
2017-09-22 22:25:37 WARN  NetworkClient:600 - Error while fetching metadata with correlatior
Finished producing file:Merchant of Venice.txt
Finished producing file:Othello.txt
Finished producing file:Hamlet.txt
Finished producing file:Julius Caesar.txt
Finished producing file:Romeo and Juliet.txt
Finished producing file:Macbeth.txt
2017-09-22 22:25:38 INFO  KafkaProducer:658 - Closing the Kafka producer with timeoutMillis
```

8. Create a program containing a Consumer and Producer with the following characteristics:

9. Consumes messages from shakespeare_topic

- Always starts from the beginning of the topic
- Reads all keys and values as type String
- Converts the key and value to Avro objects
- Writes messages to a topic named shakespeare_avro_topic
- The key should be a ShakespeareKey Avro object
- The value should be a ShakespeareValue Avro object



10. Create a Consumer with the following characteristcs:

- Consumes messages from shakespeare_avro_topic
- Always starts from the beginning of the topic
- Consumes all data as Avro objects
- Outputs the key and value of each message to the screen

Kafka Consumers and Producers (Python)

In this Exercise, you are creating a processing pipeline. A Publisher will read in every line of Shakespeare as a string and write it to a topic. A Consumer will read the data from that first topic, parse it, create Avro objects, and publish the Avro objects to a new topic. The final piece of the pipeline is a Consumer that reads the Avro objects from the topic and writes them to the screen with print.

11. Create a Producer with the following characteristics:

- Connects to kafkarest1:8082
- Reads in the files from the shakespeare directory line by line
- Sends each line as a separate message

- Sends messages to a topic named shakespeare_topic
- Sends all messages as strings
- The key should be the name of the play (which is contained in the filename)
- The value should be the line from the play

12. Create a program containing a Consumer and Producer with the following characteristics:

    - Consumes messages from shakespeare_topic
    - Always starts from the beginning of the topic
    - Reads all keys and values as strings
    - Converts the key and value to Avro objects
    - Writes messages to a topic named shakespeare_avro_topic
    - The key should be a ShakespeareKey Avro object
    - The value should be a ShakespeareValue Avro object

You may hit a bug after submitting 1,000 requests to the REST service. This is caused by a bug: https://github.com/confluentinc/schema-registry/issues/172). Work around this issue by stopping at 1,000 requests or combine multiple messages into one request.

13. Create a Consumer with the following characteristics:

    - Consumes messages from shakespeare_avro_topic
    - Always starts from the beginning of the topic
    - Consumes all data as Avro objects
    - Outputs the key and value of each message to the screen

Exercise7: Running Kafka Connect in Standalone Mode

In this Hands-On Exercise, you will run Kafka Connect in standalone mode with two Connectors: a file source Connector, and a file sink Connector. The file source Connector will read lines from a file and produce them to a Kafka topic. The file sink Connector will consume from the same Kafka topic and write messages to a file.

1. Read through the connect-standalone.properties file. This file configures the standalone worker.

```
$ cat /etc/kafka/connect-standalone.properties
```

2. Read through the connect-file-source.properties file. This file configures the file source connector.

```
$ cat /etc/kafka/connect-file-source.properties
```

3. Seed a test file with some test data:

```
$ echo -e "log line 1\nlog line 2" > test.txt
```

```
[training@confluent-training-vm ~]$
[training@confluent-training-vm ~]$ echo -e "log line 1\nlog line 2" > test.txt
[training@confluent-training-vm ~]$
```

4. Run the standalone connector with the file source connector. Note that it may take a few seconds to start.

```
$ connect-standalone \
/etc/kafka/connect-standalone.properties \
/etc/kafka/connect-file-source.properties
```

```
[training@confluent-training-vm ~]$
[training@confluent-training-vm ~]$ connect-standalone \
> /etc/kafka/connect-standalone.properties \
> /etc/kafka/connect-file-source.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/java/confluent-common/slf4j-log4j12-1.7.6.jar!/org
/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/kafka-serde-tools/slf4j-log4j12-1.7.6.jar!/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/kafka-connect-hdfs/slf4j-log4j12-1.7.5.jar!/o
rg/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/share/java/kafka/slf4j-log4j12-1.7.21.jar!/org/slf4j/imp
l/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
[2017-09-22 23:15:20,218] INFO StandaloneConfig values:
        cluster = connect
        rest.advertised.host.name = null
        task.shutdown.graceful.timeout.ms = 5000
        rest.host.name = null
        rest.advertised.port = null
        bootstrap.servers = [localhost:9092]
```

Leave the standalone connector running in the foreground in one terminal.

5. In a separate terminal, run the console consumer to ensure the lines were written to Kafka:

```
$ kafka-console-consumer \
--bootstrap-server broker1:9092 \
--from-beginning \
```

```
--topic connect-test \

--new-consumer
```

```
[training@confluent-training-vm ~]$ echo -e "log line 1\nlog line 2" > test.txt
[training@confluent-training-vm ~]$ kafka-console-consumer \
> --bootstrap-server broker1:9092 \
> --from-beginning \
> --topic connect-test \
> --new-consumer

```

You should see two JSON messages, each of whose payload has one line of the source file.

```
[training@confluent-training-vm ~]$ kafka-console-consumer --bootstrap-server br
oker1:9092 --from-beginning --topic connect-test --new-consumer
{"schema":{"type":"string","optional":false},"payload":"log line 1"}
{"schema":{"type":"string","optional":false},"payload":"log line 2"}
```

You should leave the console consumer running for the duration of the Exercise.

6. Read through the connect-file-sink.properties file. This file configures the file sink connector:

```
$ cat /etc/kafka/connect-file-sink.properties
```

8. In the original terminal with Connect running, press Ctrl-c to stop the Connector. (Two standalone Connectors cannot run at the same time because Connect binds to a port, even in standalone mode. More on this in a later exercise.)

9. Now, restart Connect in standalone mode, passing in both the source and sink Connector configuration files:

```
$ connect-standalone \

/etc/kafka/connect-standalone.properties \
/etc/kafka/connect-file-source.properties \
/etc/kafka/connect-file-sink.properties
```

```
topics=connect-test[training@confluent-training-vm ~]$ connect-standalone \
> /etc/kafka/connect-standalone.properties \
> /etc/kafka/connect-file-source.properties \
> /etc/kafka/connect-file-sink.properties
```

Leave the standalone connector running in the foreground in one terminal.

10. Notice that a new file has been created in the current working directory, called test.sink.txt. View this file to see that the sink connector did what it was expected to do:

```
$ cat test.sink.txt
```

```
[training@confluent-training-vm ~]$ ls -ltr
total 36
drwxr-xr-x 2 training training 4096 Apr 12  2016 Public
drwxr-xr-x 2 training training 4096 Apr 12  2016 Downloads
drwxrwxr-x 2 training training 4096 May 24  2016 assets
drwxrwxr-x 3 training training 4096 May 24  2016 workspace
drwxrwxr-x 9 training training 4096 May 24  2016 eclipse
drwxr-xr-x 2 training training 4096 Jun 18  2016 Desktop
drwxr-x--- 4 training training 4096 Sep 25  2016 developer
-rw-rw-r-- 1 training training   22 Sep 23 01:17 test.txt
-rw-rw-r-- 1 training training   22 Sep 23 01:22 test.sink.txt
[training@confluent-training-vm ~]$
```

```
[training@confluent-training-vm ~]$ cat test.sink.txt
log line 1
log line 2
[training@confluent-training-vm ~]$
```

11. In a separate terminal, append a new line to the source file, test.txt:

```
$ echo "log line 3" >> test.txt
```

12. View the sink file, test.sink.txt:

```
$ cat test.sink.txt
```

```
[training@confluent-training-vm ~]$ echo "log line 3" >> test.txt
[training@confluent-training-vm ~]$ cat test.sink.txt
log line 1
log line 2
log line 3
[training@confluent-training-vm ~]$
```

Notice that the new log line has been written to the sink file as well. Kafka Connect read the new line from the source file and wrote it to the topic; it then read from the topic and appended the new message to the sink file.

13. Switch to the terminal running the console consumer that you started in Step 5. Notice that the new log line has appeared.

```
[training@confluent-training-vm ~]$ kafka-console-consumer --bootstrap-server br
oker1:9092 --from-beginning --topic connect-test --new-consumer
{"schema":{"type":"string","optional":false},"payload":"log line 1"}
{"schema":{"type":"string","optional":false},"payload":"log line 2"}
{"schema":{"type":"string","optional":false},"payload":"log line 3"}
```

14. In each terminal window, press Ctrl-c to terminate the process that is running.

Exercise8: Using the Kafka Connect REST API

In this Exercise, you will interact with the REST API to get a Connector's configuration and update it. The REST API is the only way to create and update Connectors in distributed mode. It can be used in standalone mode, but as shown in the previous exercise, standalone mode is easier to use with the command line. However, to make this exercise simple, the REST calls in this exercise will be made against Connect running in standalone mode.

1. Start the file source and sink connectors in the same way they were started in the previous example.

```
$ connect-standalone \

/etc/kafka/connect-standalone.properties \
/etc/kafka/connect-file-source.properties \
/etc/kafka/connect-file-sink.properties
```

```
[training@confluent-training-vm ~]$ connect-standalone \
> /etc/kafka/connect-standalone.properties \
> /etc/kafka/connect-file-source.properties \
> /etc/kafka/connect-file-sink.properties
```

2. In a separate terminal, run a GET query on the /connectors endpoint to get a list of active connectors:

```
$ curl http://localhost:8083/connectors

  ["local-file-source","local-file-sink"]
```

```
["local-file-source"][training@confluent-training-vm ~]$ curl http://localhost:8
083/connectors ["local-file-source","local-file-sink"]
>
```

3. Get the configuration of the local-file-sink connector

```
$ curl http://localhost:8083/connectors/local-
file-sink/config
```

```
[training@confluent-training-vm ~]$ curl http://localhost:8083/connectors/local-
file-sink/config
{"connector.class":"FileStreamSink","file":"test.sink.txt","tasks.max":"1","topi
cs":"connect-test","name":"local-file-sink"}[training@confluent-training-vm ~]$
```

4. Run the previous command again, but this time redirect the output to a file:

```
$ curl http://localhost:8083/connectors/local-
file-sink/config > \

local-file-sink.config.json
```

```
[training@confluent-training-vm ~]$ curl http://localhost:8083/connectors/local-
file-sink/config > \
> local-file-sink.config.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
124   124  124   124    0     0  14506      0 --:--:-- --:--:-- --:--:--     0
[training@confluent-training-vm ~]$
```

5. Modify local-file-sink.config.json, by changing the file the sink

connector writes to. Open the file in your favorite editor and change test.sink.txt to test-new.sink.txt. Then, perform a REST PUT query to update the configuration:

```
[training@confluent-training-vm ~]$ cat local-file-sink.config.json
{"connector.class":"FileStreamSink","file":"test.sink.txt","tasks.max":"1","topi
cs":"connect-test","name":"local-file-sink"}[training@confluent-training-vm ~]$
[training@confluent-training-vm ~]$
```

```
[training@confluent-training-vm ~]$ cat local-file-sink.config.json
{"connector.class":"FileStreamSink","file":"test-new.sink.txt","tasks.max":"1","
topics":"connect-test","name":"local-file-sink"}
[training@confluent-training-vm ~]$
```

```
$ curl -X PUT
http://localhost:8083/connectors/local-file-
sink/config \

-d @local-file-sink.config.json \ --header
"Content-Type: application/json"
```

```
[training@confluent-training-vm ~]$ curl -X PUT http://localhost:8083/connectors
/local-file-sink/config \
> -d @local-file-sink.config.json \
> --header "Content-Type: application/json"
{"name":"local-file-sink","config":{"connector.class":"FileStreamSink","file":"t
est-new.sink.txt","tasks.max":"1","topics":"connect-test","name":"local-file-sin
k"},"tasks":[{"connector":"local-file-sink","task":0}]}[training@confluent-train
ing-vm ~]$
```

6. Append a new line to the source file:

```
$ echo "foobarbaz" >> test.txt
```

7. Notice that the original sink file test.sink.txt does not contain the new line. However, the new sink file test-new.sink.txt does:

```
$ cat test.sink.txt

$ cat test-new.sink.txt
```

```
[training@confluent-training-vm ~]$ echo "foobarbaz" >> test.txt
[training@confluent-training-vm ~]$ cat test.sink.txt
log line 1
log line 2
log line 3
[training@confluent-training-vm ~]$ cat test-new.sink.txt
foobarbaz
[training@confluent-training-vm ~]$
```

8. In the terminal window where you started the connectors, hit Ctrl-c to terminate the process.

Exercise9: Using the JDBC Connector

In this Exercise, you will create a small test sqlite database and configure a standalone JDBC source Connector to write the contents of the table in the database to a Kafka topic, including new rows when they are created.

1. Open the sqlite3 shell, in preparation for creating a database, test.db:

```
$ sqlite3 test.db
```

```
[training@confluent-training-vm ~]$ sqlite3 test.db
SQLite version 3.6.20
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

2. Create an accounts table, with ID and name columns, and insert two rows:

```
sqlite> CREATE TABLE accounts(id INTEGER
PRIMARY KEY AUTOINCREMENT NOT

NULL, name VARCHAR(255));

sqlite> INSERT INTO accounts(name)
VALUES('alice');

sqlite> INSERT INTO accounts(name)
VALUES('bob');

sqlite> .exit
```

```
sqlite> CREATE TABLE accounts(id INTEGER PRIMARY KEY AUTOINCREMENT NOT
   ...> NULL, name VARCHAR(255));
sqlite> INSERT INTO accounts(name) VALUES('alice');
sqlite> INSERT INTO accounts(name) VALUES('bob');
sqlite> .exit
[training@confluent-training-vm ~]$
```

3. Read through the quickstart-sqlite.properties file. This file configures the JDBC source Connector.

```
$ cat /etc/kafka-connect-jdbc/quickstart-
sqlite.properties
```

4. Stop other standalone Connectors if any are running. Then, in the same directory that you ran the sqlite3 command, run the JDBC Connector in standalone mode:

```
$ connect-standalone \

/etc/kafka/connect-standalone.properties \
/etc/kafka-connect-jdbc/quickstart-
sqlite.properties
```

```
[training@confluent-training-vm ~]$ connect-standalone \
> /etc/kafka/connect-standalone.properties \
> /etc/kafka-connect-jdbc/quickstart-sqlite.properties
```

Leave the standalone connector running in the foreground.

5. In a separate terminal, run the console consumer to see what messages have been produced:

```
$ kafka-console-consumer \
--bootstrap-server broker1:9092 \
--from-beginning \
--topic test-sqlite-jdbc-accounts \
--new-consumer
```

```
[training@confluent-training-vm ~]$ kafka-console-consumer \
> --bootstrap-server broker1:9092 \
> --from-beginning \
> --topic test-sqlite-jdbc-accounts \
> --new-consumer
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"i
d"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"a
ccounts"},"payload":{"id":1,"name":"alice"}}
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"i
d"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"a
ccounts"},"payload":{"id":2,"name":"bob"}}
```

Each message has a lot of JSON metadata because each message is a JSON blob. connect- standalone.properties is configured to use the JsonConverter Converter. Leave the console consumer running in the foreground.

6. In a third terminal, insert a new row into the accounts table:

```
$ sqlite3 test.db
```

```
sqlite> INSERT INTO accounts(name)
VALUES('ted');
```

```
[training@confluent-training-vm ~]$ sqlite3 test.db
SQLite version 3.6.20
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> INSERT INTO accounts(name) VALUES('ted');
sqlite>
```

7. Go back to the terminal running the console consumer. Notice that a third message has been produced for the new row that was just created. Note that in production, AvroConverter paired with the Schema Registry is highly recommended to best support upstream database schema changes.

```
[training@confluent-training-vm ~]$ kafka-console-consumer \
> --bootstrap-server broker1:9092 \
> --from-beginning \
> --topic test-sqlite-jdbc-accounts \
> --new-consumer
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"i
d"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"a
ccounts"},"payload":{"id":1,"name":"alice"}}
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"i
d"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"a
ccounts"},"payload":{"id":2,"name":"bob"}}
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"i
d"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"a
ccounts"},"payload":{"id":3,"name":"ted"}}
```

8. Exit sqlite3 by typing

```
sqlite> .exit
```

9. In the terminal window in which the connector is running, hit Ctrl-c to terminate the process.

Exercise10: Writing a Kafka Streams Application

Project directory: streams

In this Hands-On Exercise, you will write a Kafka Streams application to process data from the topic shakespeare_topic. (If you did not create the topic in a previous Hands-On Exercise, run the ShakespeareProducer application from the kafka_avro Exercise. The key of each message in the topic is the name of the play; the value is a line from the play.

The Kafka Streams API is currently only available as a Java API. If you do not know Java, please spend the Exercise time investigating the sample solution.

Change directory to streams

```
[training@confluent-training-vm exercise_code]$ pwd
/home/training/developer/exercise_code
[training@confluent-training-vm exercise_code]$ cd streams/
[training@confluent-training-vm streams]$ ls -ltr
total 8
-rwxr-x--- 1 training training 3257 Sep 25  2016 pom.xml
drwxr-x--- 3 training training 4096 Sep 25  2016 src
[training@confluent-training-vm streams]$ pwd
/home/training/developer/exercise_code/streams
[training@confluent-training-vm streams]$
```

Create Eclipse project using Maven

```
[training@confluent-training-vm streams]$ mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building streamsexercise 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-eclipse-plugin:2.9:eclipse (default-cli) > generate-resources @ streamsexercise >>>
[INFO]
[INFO] --- avro-maven-plugin:1.7.7:schema (default) @ streamsexercise ---
[INFO]
[INFO] <<< maven-eclipse-plugin:2.9:eclipse (default-cli) < generate-resources @ streamsexercise <<<
[INFO]
[INFO] --- maven-eclipse-plugin:2.9:eclipse (default-cli) @ streamsexercise ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Wrote settings to /home/training/developer/exercise_code/streams/.settings/org.eclipse.jdt.cor
e.prefs
[INFO] Wrote Eclipse project for "streamsexercise" to /home/training/developer/exercise_code/streams.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.465 s
[INFO] Finished at: 2017-09-23T22:16:34-05:00
[INFO] Final Memory: 10M/44M
[INFO] ------------------------------------------------------------------------
[training@confluent-training-vm streams]$
```

1. Create a new Topic based on shakespeare_topic where the value is converted to upper-case.

2. Create a new Topic based on shakespeare_topic which only contains messages from the play Macbeth.

Import the existing project to Eclipse workspace and run the project.

Do It Yourself:

3. Investigate the code in shakespeare_example, which converts shakespeare_topic to a new Avro topic. Write a Kafka Streams application to display the contents of that topic. (Hint: use the print() method.)