**Kafka Practice**

CONTENTS

Notes Regarding the Practice 2

Connect to the Kafka Environment 2

Creating Kafka Topics (10) 2

Writing and Reading Kafka Topics (15) 2

Using Kafka Connect to Read From a Source File (15) 3

Using Kafka Connect to Write to a Destination File (20) 5

Kafka Administration (20) 6

Using Kafka Connect to Read From a Database (MySQL) (20) 7

Using Kafkadrop to Monitor Kafka Topics (Optional but Recommended–No Points) 10

NOTES REGARDING THE PRACTICE

When you see the dollar sign in a command you need to execute, note this is just a command prompt indication. You do not need to actually write the "$", as it is not part of the command. For all the tasks below, write all the commands you used in the Practice answers document. When you complete it, you can submit this document for review.

CONNECT TO THE THE KAFKA ENVIRONMENT

* Make sure the Kafka environment is up and running:

**docker ps -a**

* Open a BASH session in the practice environment. Be sure to include the "-l" flag to load scripts.

**docker exec -it kafka /bin/bash**

CREATING KAFKA TOPICS (10)

* View the Kafka related scripts in the Kafka "Bin" directory

Hint: View the PATH variable contents to see the location of the directory. You should see ~25 scripts

* Check the following:
* The value of KAFKA\_HOME in your environment
* If there are any configured topics currently in Kafka
* Create a new Kafka topic named **kafka-tst-01**, which has one partition and runs on one node.
* Check again and make sure your topic was created.

WRITING AND READING KAFKA TOPICS (15)

* Write a command which reads data from a Kafka topic and writes it to the standard output. Think, which script should you use for this task?

Note: Your screen should "hang" waiting for incoming messages. You can always break using Ctrl+c, but do not do this yet.

* Open a second terminal window and connect to the Docker there too.
* Write a command which reads data from the standard input and writes it to your Kafka topic. Think, which script should you use for this task?

Note: You should get the ">" prompt, waiting for you to enter your text.

* Write the text "Hello world! This is Kafka!" in the producer side (standard input, Session 2)
* Ensure the consumer receives the messages (Standard output, Session 1).
* Produce a few more messages: "Event 1", "Event 2".. and see how they are consumed. Think if you can have another producer writing to the same topic.
* Check if your assumption is correct: Open the third session and create the second producer writing to the same topic.
* Produce some events and make sure they are consumed by the consumer session.
* Think if you can have another consumer reading from the same topic.
* Check if your assumption is correct: Open the 4th session and create the second consumer reading from the same topic.
* Answer the questions: Which consumer will now receive new events? Just one of them? Both of them? Round robin?
* Produce an event and check if your assumption is correct.
* Think if you can create a new consumer that will read all events generated since the beginning of the topic.
* Create such a consumer and check if your assumption is correct.
* Delete the Kafka topic you created. You should get the message: "Topic kafka-tst-01 is marked for deletion..."
* Make sure the topic was deleted.

Note: This can take a few minutes.

USING KAFKA CONNECT TO READ FROM A SOURCE FILE (15)

* Create the following directories in the system for using them in the Kafka practice:

**mkdir -p /kafka/{confFiles,srcFiles,sinkFiles}**

* Verify the directories were created:

**ls -l /kafka/**

* Use the "VI" editor to create a file name **connect-file-source.properties** in the **confFiles** directory, with the following contents:
* Fill up the missing values

**#-------------------------------------------------- #Content of connect-file-source.properties**

**#--------------------------------------------------**

**#A logical name for your task name=kafka-file-source-task**

**#Class of connector to use connector.class=*<Value>***

**#Number of parallel tasks to launch – Provides scalability tasks.max=1**

**#Local file to monitor for new events file=/kafka/srcFiles/sourceFile.log**

**#Topic to publish data to topic=*<Value>***

* Exit and save the file (ESC + :wq)
* Insert a few records to the source file using the following script:

**$ echo 'Event 1 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/sourceFile.log $ sleep 1**

**$ echo 'Event 2 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/sourceFile.log $ sleep 1**

**$ echo 'Event 3 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/sourceFile.log**

* Verify the contents of the source file you created

**cat /kafka/srcFiles/sourceFile.log**

* Issue the Kafka Connect Standalone script with the required parameters to read the created source file. See the guidelines document for a few pointers if you require assistance.

Note: In this practice task, you only read from a file (source) and do not write to a file (sink). So, address only the relevant parameters.

* You can ignore any messages you receive when running the script, as it throws a lot of WARNING and INFO messages. As long as the script does not crash, everything is fine.
* Create a consumer that reads data from the topic you are using and write them to the screen.

Note: Specify a flag to ensure you get all messages from the beginning of the queue (all rows in the file source, including rows that were sent earlier to the topic).

* Write a few more messages to the file source:

**$ echo 'Event 1 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/sourceFile.log $ sleep 1**

**$ echo 'Event 2 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/sourceFile.log**

**$ sleep 1**

**$ echo 'Event 3 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/sourceFile.log**

* Make sure you received them on the consumer's side.
* List current Kafka topics and make sure you see the topic you specified.
* Stop both processes (producer and consumer) but do not drop the topic.

USING KAFKA CONNECT TO WRITE TO A DESTINATION FILE (20)

Use the "VI" editor to overwrite the file you created in the previous practice named **connect-file-source.properties**, with the following contents.

* Fill up the missing values

**#-------------------------------------------------- #Content of connect-file-source.properties**

**#--------------------------------------------------**

**#A logical name for your task name=kafka-file-source-task-2**

**#Class of connector to use connector.class=*<Value>***

**#Number of parallel tasks to launch – Provides scalability tasks.max=1**

**#Local file to monitor for new events file=/kafka/srcFiles/newSourceFile.log**

**#Topic to publish data to (use a new topic name) topic=*<Value-sink>***

* Exit and save the file (ESC + :wq)
* Use "VI" editor to create a file named **connect-file-sink.properties**, with the following contents. Fill up the missing values

**#-------------------------------------------------- #Content of connect-file-sink.properties**

**#--------------------------------------------------**

**#A logical name for your task name=*<new-meaningful-task-name>***

**#Class of connector to use connector.class=*<Value>***

**#Number of parallel tasks to launch - Provides scalability tasks.max=1**

**#Target File to write all events to file=/kafka/sinkFiles/targetFile.log**

**#Topics to subscribe**

**topics=*<Same-topic-name-as-source>***

* Write a few messages to the new source file:

**$ echo 'Event 1 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/newSourceFile.log $ sleep 1**

**$ echo 'Event 2 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/newSourceFile.log $ sleep 1**

**$ echo 'Event 3 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/newSourceFile.log**

* Ensure the contents of the source file you created:

**$ cat /kafka/srcFiles/newSourceFile.log**

* Create the target file manually:

**$ touch /kafka/sinkFiles/targetFile.log**

* Issue the Kafka Connect Standalone script with the required parameters to read the source file you created and write to the target (sink) file. See the guidelines document for a few pointers if you require assistance.

Note: In this practice task, you read from a file (source) and write to a file (sink). So, address both relevant parameters.

* You can ignore any messages you receive when running the script, as it throws a lot of WARNING and INFO messages. As long as the script does not crash, everything is fine.
* Ensure the messages arrived at the file target destination. (Use "***cat***")
* Create a consumer that reads data from the topic you are using and writes them to the screen.

Note: Specify a flag to ensure you get all messages from the beginning of the queue (all rows in the file source, including rows that were sent earlier to the topic).

* List current Kafka topics and make sure you see the topic you specified.
* Insert a few more rows into the source file:

**$ echo 'Event 1 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/newSourceFile.log $ sleep 1**

**$ echo 'Event 2 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/newSourceFile.log $ sleep 1**

**$ echo 'Event 3 | ' $(hostname) ' | ' $(date) >> /kafka/srcFiles/newSourceFile.log**

* Run the following script that creates new messages in the file for 30 seconds:

**$ for v in {1..30}; do echo Event-$v '-' `date` >> /kafka/srcFiles/newSourceFile.log && sleep 1; done**

* On another terminal window, ensure the messages arrive at the file target destination. (Use "***cat***")
* Leave all processes active (producer and both consumers).

KAFKA ADMINISTRATION (20)

This task builds upon the previous one: "Using Kafka Connect to read from a source file," and assumes the consumers and producer processes are still active.

* List the Kafka consumer groups.
* Check out which groups and consumers you see.
* Check the Kafka consumers' offsets.
* Try to run with different consumer groups (change the group name in the group parameter).
* Add "curl" to the environment:

**apk add curl**

* Use the Kafka REST API to get the following information from Kafka Connect (port 8083). (Use the **json.tool** to get more readable results. See the guidelines for more info.)
* Current running connectors
* Active tasks for one of the running connectors
* Status for one of the running connectors
* Get the connector's configuration
* View the documentation using the Guidelines document link, and perform a few more queries.
* Use the information you found above to reset the file-sink consumer (screen output) to the earliest point possible for the topic it is consuming.
* See if you can change the offset while it is active.
* Stop it if required and change the offset.
* Activate it again and ensure it consumed all records from the top of the file.
* Stop and restart it without changing the offset and see that no new records have been consumed, as the offset did not change.
* Use the following command to check the rowcount:

**cat /kafka/sinkFiles/targetFile.log | wc -l**

USING KAFKA CONNECT TO READ FROM A DATABASE (MYSQL) (20)

* Add and run the MySQL container as described in the installation guide.
* Ensure it is up and running using the **docker stats** command.
* Open a new terminal window and connect to the MySQL container:

**docker exec -it mysql /bin/bash**

### Inside the MySQL Container:

### Connect to MySQL, and run the following script:

### **mysql -uroot -proot**

* Run the following script:

**drop database if exists srcdb; create database srcdb;**

**use srcdb; create table src\_events( event\_id int primary key, event\_timestamp timestamp not null**

**);**

**insert into src\_events values(1, sysdate()); select sleep(1);**

**insert into src\_events values(2, sysdate()); select sleep(1);**

**insert into src\_events values(3, sysdate());**

**create table web\_logins( login\_time timestamp, login\_count int**

**);**

**insert into web\_logins values(sysdate(), 0);**

**exit;**

* Verify the tables and rows were created successfully:

**mysql -uroot -proot srcdb -e 'select \* from src\_events'**

* Minimize the terminal window connected to the MySQL container, and continue with the Kafka container connection.

### Inside the Kafka Container

### Ensure the directory names in the Kafka practice directory:

### **ls -l /kafka/**

### Use the "VI" editor to create a file name **connect-jdbc-source.properties** in the **confFiles** directory with the following contents.

* Fill up the missing values
* Notes:
  + - Get the information from the table named "src\_events"
    - We want to identify new rows according to values in the "event\_id" column.
    - MySQL database name is "srcdb"
    - MySQL username and password are root:root
* You can refer to the Kafka Practice Guidelines documents for assistance and hints.

**#-------------------------------------------------- #Content of connect-jdbc-source.properties**

**#--------------------------------------------------**

**#Name of the connector name=Kafka-jdbc-source-task-1**

**#Connector class to be used.**

**connector.class=*<Value>***

**#JDBC connector URL for mysql. make sure the mysql driver is in classpath. connection.url=jdbc:mysql://*<Gateway-IP>*:3306/*<db-***

***name>*?user=*<username>*&password=*<password>*&allowPublicKeyRetrieval=true**

**#List of tables to publish. you can also use blacklists table.whitelist=*<Value>***

**#No. of parallel tasks - Ideally one per table tasks.max=1**

**#How frequently to poll from the database for new records poll.interval.ms=2000**

**#mode - incrementing or timestamp+incrementing mode=*<Value>***

**incrementing.column.name=*<Value>***

**#Topic name to be created - This will create a topic with the prefix you specify, with the table name appended topic.prefix=*<Value>***

* Exit and save the file (ESC + :wq)
* Issue the Kafka Connect Standalone script with the required parameters to read the source table from MySQL using the created properties file.
* See the guidelines document for a few pointers if you require assistance.

Note: In this practice, you only read from a table (source) and do not write to any table or file (sink). So, address only the relevant parameters.

* As long as the script does not crash, everything is fine.
* In case of an error, **Address is already in use** refer to the troubleshooting guide.
* Create a consumer that reads data from the topic you are using and write them to the screen.

Note: Specify a flag to ensure you get all messages from the beginning of the queue (all rows in the file source, including rows that were sent earlier to the topic).

* On the MySQL container, insert a few more records and verify they are ingested by Kafka and consumed by the consumer process:

**$ mysql -uroot -proot srcdb -e 'insert into src\_events select max(event\_id)+1, sysdate() from src\_events'**

**$ mysql -uroot -proot srcdb -e 'select \* from src\_events'**

* Verify you received them on the consumer side.
* List current Kafka topics and verify you see the topic you have specified.
* Stop both processes (producer and consumer) and drop the topic.

USING KAFDROP TO MONITOR KAFKA TOPICS (OPTIONAL BUT RECOMMENDED–NO POINTS)

Note that using Kafdrop for monitoring topics is recommended but optional and does not affect your grade for this task.

* You can use one of the tools for Kafka monitoring, for example, Kafdrop.
* Follow the guideline in the <https://github.com/obsidiandynamics/kafdrop> to install the tool and visualize Kafka topics.
* Explore this Web UI and see how you can view and manage your Kafka topics using a graphical interface.