Confluent KAFKA Administration

Rajesh Pasham

Managing Kafka

- This chapter covers the following topics:
  - Managing consumer groups
  - Dumping log segments
  - Using the GetOffsetShell
  - Using the MirrorMaker tool
  - Replaying log producer
  - Using state change log merger

Managing Kafka

- Managing an Apache Kafka cluster in production can be a difficult task.
- The Kafka authors have developed some command-line tools to make a DevOps team's life easier for debugging, testing, and running a Kafka cluster.
- This chapter covers some of these tools.

Managing consumer groups

- The *ConsumerGroupCommand* tool is valuable when debugging consumer groups.
- This tool allows us to list, describe, and delete consumer groups.
- From the Kafka installation directory, run the following command:
  - $ bin/kafka-consumer-groups --bootstrap-server localhost:9092 ----list
  - The output is something like the following:

    Note: This will only show information about consumers that use the Java consumer API (non-ZooKeeper-based consumers).

    console-consumer-10354
    vipConsumersGroup
    console-consumer-44233

Managing consumer groups

- To see the offsets, use describe on the consumer group as follows:
  - $ bin/kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group vipConsumersGroup

Note: This will only show information about consumers that use the Java consumer API (non-ZooKeeper-based consumers).

```
TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-
OFFSET  LAG       CONSUMER-ID
                HOST            CLIENT-ID
source-topic            0                1                1
          0       consumer-1-be4c31-e197-455b-89fb-
cce53e380a26   /192.168.1.87      consumer-1
```

Managing consumer groups

- As the command says, if old high-level consumers are used and the group metadata is stored in ZooKeeper (with the offsets.storage =zookeeper flag), specify zookeeper instead of bootstrap-server, as follows:
  - $ bin/kafka-consumer-groups --zookeeper localhost:2181 --list
  - The *ConsumerGroupCommand* takes the following arguments:
    - --group <String: consumer group>: This is the consumer group to manipulate

Managing consumer groups

- The *ConsumerGroupCommand* takes the following arguments:
  - --bootstrap-server <String: server to connect>: This is the server to connect to (for consumer groups based on a non-old consumer)
  - --zookeeper <String: urls>: This is the ZooKeeper connection specified as a comma-separated list with elements in the form host:port (for consumer groups based on old consumers)
  - --topic <String: topic>: This the topic that contains the consumer group information to manipulate
  - --list: This lists all the consumer groups of the broker

Managing consumer groups

- The *ConsumerGroupCommand* takes the following arguments:
  - --describe: This describes the consumer group and lists the offset lag (number of messages not yet processed) on a given group
  - --reset-offsets: This resets the offsets of the consumer group
  - --delete: This is passed in a group to delete topic partition offsets and ownership information on the entire consumer group

Dumping log segments

- This tool is for debugging the Kafka log data for various purposes, such as reviewing how the logs have been written and to see the status of the segments.
- Also, it is useful for reviewing the log files generated by Kafka.
- From the Kafka installation directory, run the following command:
  - $ bin/kafka-run-class kafka.tools.DumpLogSegments --deep-iteration --files /tmp/kafka-logs/your-topic-0/00000000000000000000.log

Dumping log segments

- The output is something like the following:

  Dumping /tmp/kafka-logs/source-topic-0/00000000000000000000.log

  Starting offset: 0

  offset: 0 position: 0 CreateTime: 1511661360150 isvalid: true keysize: -1 valuesize: 4 magic: 2 compresscodec: NONE producerId: -1 sequence: -1 isTransactional: false headerKeys: []

Dumping log segments

- The *DumpLogSegments* command parses the log file and dumps its contents to the console; it is useful for debugging a seemingly corrupt log segment.
- The *DumpLogSegments* command takes the following arguments:
  - *--deep-iteration*: If set, it uses deep iteration (complete audit) instead of shallow iteration (superficial audit) to examine the log files.
  - *--files <String: file1, file2, ...>*: This is a mandatory parameter. The comma-separated list of data log files to be dumped.

Dumping log segments

- The *DumpLogSegments* command takes the following arguments:
  - *--max-message-size <Integer: size>*: This is used to offset the size of the largest message. The default value is 5242880.
  - *--print-data-log*: If it is set, the messages' content will be printed when dumping data logs.
  - *--verify-index-only*: If it is set, this process just verifies the index log without printing its content.

Dumping log segments

- The *DumpLogSegments* command takes the following arguments:
  - *--max-message-size <Integer: size>*: This is used to offset the size of the largest message. The default value is 5242880.
  - *--print-data-log*: If it is set, the messages' content will be printed when dumping data logs.
  - *--verify-index-only*: If it is set, this process just verifies the index log without printing its content.

Using the GetOffsetShell

- When debugging an Apache Kafka project, it is sometimes useful to obtain the offset values of the topics.
- For this purpose, this tool comes in handy.
- From the Kafka installation directory, run the following command:
  - $ bin/kafka-run-class kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic source-topic --time -1
- The output is something like the following:
  - source-topic:0:0
  - source-topic:1:0
  - source-topic:2:6
  - source-topic:3:0

Using the GetOffsetShell

- The GetOffsetShell is an interactive shell to get the consumer offsets and takes the following options:
  - **--broker-list <String: hostname:port>:** This specifies the list of server ports to connect to in a comma-separated list in the host:port format.
  - **--max-wait-ms <Integer: ms>:** This specifies the maximum amount of time each fetch request has to wait. The default value is 1000, that is 1 second.
  - **--offsets <Integer: count>:** This specifies the number of offsets returned. By default 1, only one offset.

Using the GetOffsetShell

- The GetOffsetShell is an interactive shell to get the consumer offsets and takes the following options:
  - **--partitions <String: partition ids>:** It is a comma-separated list of partition IDs. If it is not specified, it fetches the offsets for all the partitions.
  - **--time <Long: timestamp>:** It specifies the timestamp of the offsets fetched. -1 for the latest and -2 for the earliest.
  - **--topic <String: topic>:** This is mandatory and it specifies the topic to fetch the offset.

Using the JMX tool

- JMX is Java management extensions.
- For the seasoned Java user, JMX is a technology that provides the tools for managing and monitoring the JVM.
- Kafka has its own JMX tool to get the JMX reports in an easy way.
- From the Kafka installation directory, run the following command:
  - **$ bin/kafka-run-class kafka.tools.JmxTool --jmx-url service:jmx:rmi:///jndi/rmi://:10101/jmxrmi**

Using the JMX tool

- The JMX tool dumps the JMX values to standard output.
- The JMX tool takes the following parameters:
  - **--attributes <String: name>:** This is a comma-separated list of objects with a whitelist of attributes to be queried. All the objects are reported if none are mentioned.
  - **--date-format <String: format>:** This specifies the data format to be used for the time field. The available options are the same as those for java.text.SimpleDateFormat.
  - **--help:** This prints the help message.
  - **--jmx-url <String: service-url>:** This specifies the URL to connect to the poll JMX data. The default value is: service:jmx:rmi:///jndi/rmi://:9999/jmxrmi.

Using the JMX tool

- The JMX tool dumps the JMX values to standard output.
- The JMX tool takes the following parameters:
  - **--object-name <String: name>:** This specifies the JMX object name to be used as a query, it can contain wild cards. If no objects are specified, all the objects will be queried.
  - **--reporting-interval <Integer: ms>:** This specifies the interval in milliseconds with the poll JMX stats. The default value is 2000, that is 2 seconds.
- To view JMX data, there is a popular tool called JConsole.
- To use JConsole, just type the command $ jconsole in a machine with Java installed.

Using the MirrorMaker tool

- The MirrorMaker tool is useful when we need to replicate the same data in a different cluster.
- The MirrorMaker tool continuously copies data between two Kafka clusters.
- From the Kafka installation directory, run this command:
  - $ bin/kafka-run-class kafka.tools.MirrorMaker --consumer.config etc/kafka/consumer.properties --producer.config etc/kafka/producer.properties --whitelist source-topic

Using the MirrorMaker tool

- The MirrorMaker tool takes the following parameters:
  - **--blacklist <String: Java regex(String)>**: This specifies the blacklist of topics to be mirrored. This can be a regular expression as well.
  - **--consumer.config <String: config file>**: This specifies the path to the consumer configuration file to consume from a source cluster. Multiple files may be specified.
  - **--help**: This prints the help message.
  - **--new.consumer**: This is used to create a new consumer in MirrorMaker (it is set by default).
  - **--num.streams<Integer: Number of threads>**: This indicates the number of consumption streams (default: 1).

Using the MirrorMaker tool

- The MirrorMaker tool takes the following parameters:
  - **--producer.config <String: config file>**: This specifies the path to the embedded producer configuration file.
  - **--whitelist <String: Java regex(String)>**: This specifies the whitelist of topics to be mirrored.

Replaying log producer

- The ReplayLogProducer tool is used to move data from one topic to another.
- From the Kafka installation directory, run this command:
  - $ bin/kafka-run-class kafka.tools.ReplayLogProducer --sync --broker-list localhost:9092 --inputtopic source-topic --outputtopic good-topic --zookeeper localhost:2181
- The ReplayLogProducer takes the following parameters:
  - --broker-list <String: hostname:port>: This is a mandatory parameter. It specifies the broker list.

Replaying log producer

- The ReplayLogProducer takes the following parameters:
  - --inputtopic <String: input-topic>: This is a mandatory parameter. It specifies the source topic.
  - --messages <Integer: count>: This specifies the number of messages to send. The default value is -1, meaning infinite.
  - --outputtopic <String: output-topic>: This is a mandatory parameter. It specifies the destination topic.
  - --reporting-interval <Integer: ms>: This specifies the interval in milliseconds to print the progress information. The default value is five seconds.

Replaying log producer

- The ReplayLogProducer takes the following parameters:
  - --threads <Integer: threads>: This specifies the number of working threads. By default, just one thread is used.
  - --sync: If it is specified, the messages are sent synchronously, if not they are sent asynchronously.
  - --zookeeper <String: zookeeper url>: This is a mandatory parameter. It specifies the connection string for the ZooKeeper connection in the host:port format. Specify multiple URLs to allow a fail-over mechanism.

Using state change log merger

- The StateChangeLogMerger tool merges the state change logs from different brokers for easy posterior analysis.
- It is a tool for merging the log files from several brokers to rebuild a unified history of what happened.
- From the Kafka installation directory, run this command:
  - $ bin/kafka-run-class kafka.tools.StateChangeLogMerger --log-regex /tmp/state-change.log* --partitions 0,1,2 --topic source-topic

Using state change log merger

- The StateChangeLogMerger command takes the following parameters:
  - --end-time <String: end>: This specifies the latest timestamp of state change entries to be merged in java.text.SimpleDateFormat
  - --logs <String: file1, file2, ...>: This is used to specify a comma-separated list of state change logs or regex for the log filenames
  - --logs-regex <String: regex>: This is used to specify a regex to match the state change log files to be merged
  - --partitions <String: 0, 1, 2, ...>: This specifies a comma-separated list of partition IDs whose state change logs should be merged

Using state change log merger

- The StateChangeLogMerger command takes the following parameters:
  - --start-time <String: start>: This specifies the earliest timestamp of state change entries to be merged in java.text.SimpleDateFormat
  - --topic <String: topic>: This specifies the topic whose state change logs should be merged