Operating Kafka

Modifying message topics

1. Run the following command from the Kafka installation directory:

```
$ bin/kafka-topics --bootstrap-server localhost:9092 --alter --topic test-topic --partitions 40 --config delete.retention.ms=10000 --delete-config retention.ms
```

This command changes the `delete.retention.ms` to 10 seconds and deletes the configuration `retention.ms`

Kafka does not support reducing the number of partitions for a topic.

There is the `kafka-configs` shell; the syntax to add and remove is as follows:

2. To add a config to a topic, run the following:

```
$ bin/kafka-configs --bootstrap-server host:port --entity-type topics --entity-name topic_name --alter --add-config x=y
```

3. To remove a config from a topic, run the following:

```
$ bin/kafka-configs --bootstrap-server host:port --entity-type topics --entity-name topic_name --alter --delete-config x
```

So, there are two shells to change a topic configuration. The first is `kafka-topics` (explained in a previous recipe), and the second is `kafka-configs`.

Implementing a graceful shutdown

1. First, edit the Kafka configuration file in `etc/kafka/server.properties` and add the following line:

```
controlled.shutdown.enable=true
```

2. Start all the nodes
3. With all the cluster nodes running, shut down one broker with the following command in the Kafka installation directory:

```
$ bin/kafka-server-stop
```

If the setting for a controlled shutdown is enabled, it ensures that a server shutdown happens properly as follows:

- It writes all the logs to disk so that there are no issues with logs when you restart the broker
- If this node is the leader, it makes sure that another node becomes the leader for a partition

This ensures that each partition's downtime is reduced considerably.

It is important to say that a controlled shutdown will only succeed if all the partitions hosted on the broker have replicas (a replication factor greater than one and at least one replica alive).

Balancing leadership

Run the following command from the Kafka installation directory:

```
$ bin/kafka-preferred-replica-election --bootstrap-server
localhost:9092
```

If the list of replicas for a partition is [3, 5, 8], then node 3 is preferred as the leader, rather than nodes 5 or 8. This is because it is earlier in the replica list. By running this command, we tell the Kafka cluster to try to restore leadership to the restored replicas.

To explain how it works, suppose that after the leader stops, new Kafka nodes join the cluster. This command avoids running them as slaves without direct operations assigned and redistributes the load among the available nodes.

Expanding clusters

1. This recipe moves all partitions for existing topics: `topic_1` and `topic_2`. The newly generated brokers are `broker_7` and `broker_8` (suppose that brokers 1 to 6 already exist). After finishing the movement, all partitions for `topic_1` and `topic_2` will exist only in `broker_7` and `broker_8`.
2. The tool only accepts JSON files as input; let's create the JSON file as follows:

```
$ cat to_reassign.json


{"topics": [{"topic": "topic_1"},
            {"topic": "topic_2"}],
"version":1
}
```

3. When the JSON file is ready, use the partition reassignment tool to generate the assignment (note it will not be executed yet) with the following command:

```
$ bin/kafka-reassign-partitions --bootstrap-server
localhost:9092 --topics-to-move-json-file to_reassign.json --
broker-list "7,8" --generate
```

The output is something like this:

```
Current partition replica assignment


{"version":1,
"partitions":[{"topic":"topic_1","partition":0,"replicas":[1
,2]},

{"topic":"topic_1","partition":1,"replicas":[3,4]},

{"topic":"topic_1","partition":2,"replicas":[5,6]},

{"topic":"topic_2","partition":0,"replicas":[1,2]},
```

```
{"topic":"topic_2","partition":1,"replicas":[3,4]},

{"topic":"topic_2","partition":2,"replicas":[5,6]}]
}

Proposed partition reassignment configuration


{"version":1,
"partitions":[{"topic":"topic_1","partition":0,"replicas":[7
,8]},

{"topic":"topic_1","partition":1,"replicas":[7,8]},

{"topic":"topic_1","partition":2,"replicas":[7,8]},

{"topic":"topic_2","partition":0,"replicas":[7,8]},

{"topic":"topic_2","partition":1,"replicas":[7,8]},

{"topic":"topic_2","partition":2,"replicas":[7,8]}]
}
```

Remember that it is just a proposal; no changes have been made to the cluster yet. The final reassignment should be specified in a new JSON file.

4. Once we have generated a new configuration, make some changes from the proposal. Create a new JSON file with the output of the previous step. Modify the destinations of the different partitions.

5. Write a JSON file (`custom-assignment.json`) to move each particular partition to each specific node as needed:

```
{"version":1,
"partitions":[
{"topic":"topic_1","partition":0,"replicas":[7,8]},

{"topic":"topic_1","partition":1,"replicas":[7,8]},
```

```
{"topic":"topic_1","partition":2,"replicas":[7,8]},

{"topic":"topic_2","partition":0,"replicas":[7,8]},

{"topic":"topic_2","partition":1,"replicas":[7,8]}]

{"topic":"topic_2","partition":2,"replicas":[7,8]},
}
```

6. Now, to execute the reassignment, run the following command from the Kafka installation directory:

```
$ bin/kafka-reassign-partitions --bootstrap-server
localhost:9092 --reassignment-json-file custom-
assignment.json --execute
```

The output is something like this:

```
Save this to use as the --reassignment-json-file option during
rollback

Successfully started reassignment of partitions

{"version":1,

"partitions":[{"topic":"topic_1","partition":0,"replicas":[7
,8]},

{"topic":"topic_1","partition":1,"replicas":[7,8]},

{"topic":"topic_1","partition":2,"replicas":[7,8]},

{"topic":"topic_2","partition":0,"replicas":[7,8]},

{"topic":"topic_2","partition":1,"replicas":[7,8]}]

{"topic":"topic_2","partition":2,"replicas":[7,8]},
```

```
}
```

7. Now, run the same command to verify the partition assignment:

```
$ bin/kafka-reassign-partitions --bootstrap-server
localhost:9092 --reassignment-json-file custom-
assignment.json --verify
```

The output is something like this:

```
Status of partition reassignment:
Reassignment of partition [topic_1,0] completed successfully
Reassignment of partition [topic_1,1] completed successfully
Reassignment of partition [topic_1,2] is in progress
Reassignment of partition [topic_2,0] completed successfully
Reassignment of partition [topic_2,1] is in progress
Reassignment of partition [topic_2,2] is in progress
```

The execute step will start moving data from the original replica to the new ones. It will take time, based on how much data is being moved. Finally, to check the status of the movement, run the verify command. It will display the current status of the different partitions.

To perform a rollback, just save the configuration generated in step 2 and apply this recipe, moving the topics to the original configuration.

Increasing the replication factor

This example increases the replication factor of partition 0 of the topic `topic_1` from 2 to 4. Before the increment, the partition's only replica existed on brokers 3 and 4. This example adds more replicas on brokers 5 and 6.

Create a JSON file named `increase-replication.json` with this code:

```
$cat increase-replication.json

{"version":1,

"partitions":[{"topic":"topic_1","partition":0,"replicas":[3
,4,5,6]}]}
```

2. Then, run the following command:

```
$ bin/kafka-reassign-partitions --bootstrap-server
localhost:9092 --reassignment-json-file increase-
replication-factor.json --execute
```

At the beginning, `topic_1` was created, with replication factor 2. The cluster has the brokers 3 and 4. Now, we have added more brokers to the cluster, called 5 and 6.

The JSON file we created indicates the partitions to be modified. In the JSON file, we indicated the topic, partition ID, and the list of replica brokers. Once it executes, the new Kafka brokers will start replicating the topic.

To verify the status of the reassignment, run the following command:

```
$ bin/kafka-reassign-partitions --bootstrap-server
localhost:9092 --reassignment-json-file increase-
replication.json --verify
```

Decommissioning brokers

How to do it...

1. First, gracefully shut down the broker to be removed
2. Once it is shut down, create a JSON file named `change-replication.json` with the following content:

```
{"version":1,
"partitions":[{"topic":"topic1","partition":0,"replicas":[1,
2]}]}
```

3. Reassign the topic to the two living brokers with the `reassign-partitions` command:

```
$ bin/kafka-reassign-partitions --bootstrap-server
localhost:9092 --reassignment-json-file change-
replication.json --execute
```

Checking the consumer position

Here is a tool to check how much the consumers are lagging from the produced messages.

How to do it...

Run the following command from the Kafka directory:

```
$ bin/kafka-consumer-groups --bootstrap-server
localhost:9092 --describe --group vipConsumersGroup
```

The output is something like the following:

```
TOPIC             PARTITION  CURRENT-OFFSET  LOG-END-OFFSET
LAG

CONSUMER-ID                                     HOST
CLIENT-ID

source-topic        0          1                1
0          consumer-1-beff4c31-e197-455b-89fb-cce53e380a26
/192.168.1.87   consumer-1
```