

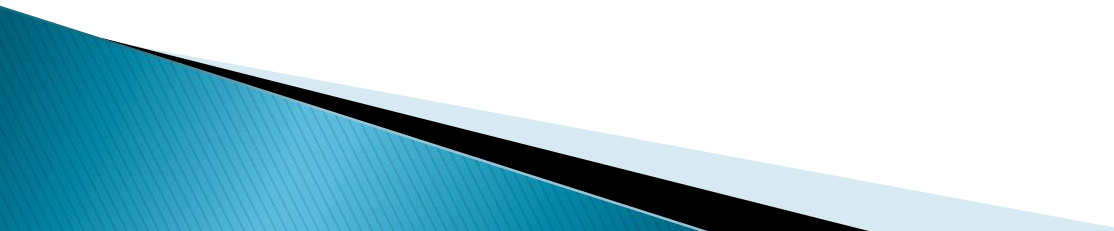
Confluent KAFKA Administration

Rajesh Pasham

Operating Kafka

- ▶ This chapter covers the following topics:
 - Adding or removing topics
 - Modifying message topics
 - Implementing a graceful shutdown
 - Balancing leadership
 - Expanding clusters
 - Increasing the replication factor
 - Decommissioning brokers
 - Checking the consumer position

Adding or removing topics

- ▶ The first chapters explained how to create a topic.
 - ▶ The power behind the tool is that it can add topics programmatically or manually, and can enable the Kafka option to automatically add topics.
 - ▶ In production, it is recommended that you disable automatic topic creation to eliminate programming errors where data is accidentally pushed to a topic that it didn't mean to create at the beginning.
- 

Adding or removing topics

- ▶ Go to the Kafka installation directory and create a topic called test-topic:
 - `$ bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --topic test-topic --partitions 5 --replication 2`
- ▶ The output should be as follows:
 - Created topic "test-topic".
- ▶ Describe the test-topic topic with the following command:
 - `$ bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic test-topic`

Adding or removing topics

- ▶ The output should be as follows:
 - topic:test-topic PartitionCount:5 ReplicationFactor:2
Configs:
 - Topic: test-topic Partition: 0 Leader: 0 Replicas: 0 Isr: 0
 - Topic: test-topic Partition: 1 Leader: 0 Replicas: 0 Isr: 0
 - Topic: test-topic Partition: 2 Leader: 0 Replicas: 0 Isr: 0
 - Topic: test-topic Partition: 3 Leader: 0 Replicas: 0 Isr: 0
 - Topic: test-topic Partition: 4 Leader: 0 Replicas: 0 Isr: 0


Adding or removing topics

- ▶ Delete the test-topic with the following command:
 - `$ bin/kafka-topics.sh --delete --bootstrap-server localhost:9092 --topic test-topic`
- ▶ The output should be as follows:

Topic test-topic is marked for deletion.

Note: This command will not have impact if `delete.topic.enable` in configuration file is not set to true.
- ▶ The replication factor indicates how many servers replicate each message that is written.
- ▶ For example, if the replication factor is four, it indicates that three servers can fail before the data is lost.
- ▶ It is recommended to use a replication factor greater than one to reboot the machines without interrupting the service.

Adding or removing topics

- ▶ The partition number indicates how many logs the topic will be divided into.
 - ▶ Remember that each partition must fit entirely on a single server.
 - ▶ It is clear that if four partitions are specified, the topic will be handled by no more than four servers.
 - ▶ The partition number also impacts the parallelism of the consumers.
 - ▶ Each partition has its own directory under the Kafka log directory.
- 

Adding or removing topics

- ▶ This directory name (log.dir and log.dirs are specified in the config/server.properties) consists of the topic name followed by a dash and the partition ID.
- ▶ The directory name cannot be over 255 characters long, limiting the topic name length.
- ▶ The kafka-topics shell takes parameters; some are explained as follows:
 - --create: This is specified to create a topic.
 - --delete: This is specified to delete a topic. The server configuration must have delete.topic.enable=true. By default, this is set as true. When it is false, the topic cannot be deleted.

Adding or removing topics

- ▶ The kafka-topics shell takes parameters; some are explained as follows:
 - `--describe`: This lists the details for the given topics.
 - `--if-exists`: This parameter is used when altering or deleting topics; the action will only execute if the topic exists.
 - `--if-not-exists`: This parameter is used when creating topics; the action will only execute if the topic does not already exist.
 - `--list`: This is specified to list all topics.
 - `--topic <String: name>`: This specifies the topic name.
 - `--partitions <Integer: num>`: This is used to specify the number of partitions to be created for the topic.

Adding or removing topics

- ▶ The kafka-topics shell takes parameters; some are explained as follows:
 - `--replication-factor <Integer: num>`: This specifies the number of replicas to be created for the topic. As explained, this number must be less than the number of nodes in the cluster.
 - `--bootstrap-server <String: urls>`: This specifies the bootstrap server string; it is a comma-separated list in the format, host:port.

Adding or removing topics

- ▶ Other configurations needed for the topic can be specified by using the following conventions:
 - `--config <String: name=value>`: This is used to override the default properties set on the server
 - `--delete-config <String: name>`: This specifies that a topic configuration override be removed for an existing topic

Modifying message topics

- ▶ Once created, topics can be modified.
- ▶ For example, when a new node is added to the cluster or a different parallelism is needed.
- ▶ Sometimes, deleting the topic and starting over is not the correct solution.
- ▶ Run the following command from the Kafka installation directory:
 - `$ bin/kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic test-topic --partitions 40 --config delete.retention.ms=10000 --delete-config retention.ms`

Modifying message topics

- ▶ 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:
 - To add a config to a topic, run the following:
 - `$ bin/kafka-configs.sh --bootstrap-server host:portt --entity-type topics --entity-name topic_name --alter --add-config x=y`

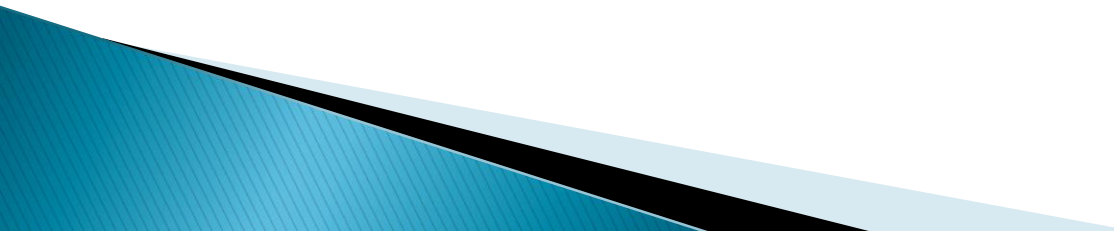
Modifying message topics

- ▶ To remove a config from a topic, run the following:
 - `$ bin/kafka-configs.sh --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`.
- ▶ The `kafka-configs` shell takes parameters; some are explained here:
 - `--add-config<String>`: This is the configuration to add, in a comma-separated list in the format `k1=v1, k2=[v1,v2,v2], k3=v3`.

Modifying message topics

- ▶ The kafka-configs shell takes parameters; some are explained here:
 - `--alter`: This is used to modify a configuration for an entity.
 - `--delete-config <String>`: This is the configuration to be removed (comma-separated list).
 - `--describe`: This parameter lists the current configurations for the given entity.
 - `--entity-name <String>`: This is the name of the entity.
 - `--entity-type <String>`: This is the type of the entity; it could be topics, clients, users, or brokers.
 - `--bootstrap-server <String: urls>`: This is a mandatory parameter and specifies the ZooKeeper connect string. It is a comma-separated list in the format host:port.

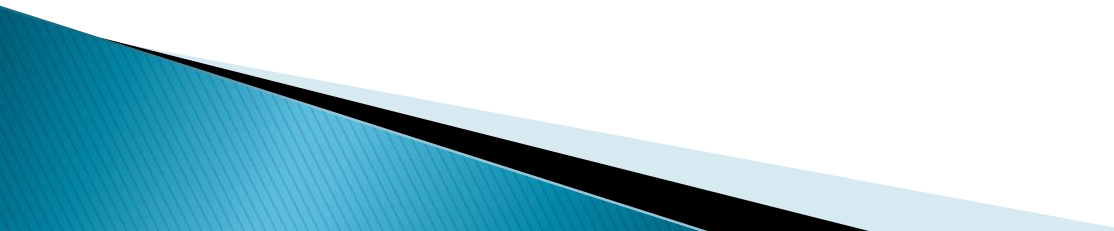
Implementing a graceful shutdown

- ▶ In production, you may experience an abrupt shutdown caused by inevitable circumstances; for example, a power outage or a sudden machine reboot.
 - ▶ But more often, there are planned shutdowns for machine maintenance or configuration changes.
 - ▶ In these situations, the smooth shutdown of a node in the cluster is desirable, maintaining the cluster up and running without data loss.
- 

Implementing a graceful shutdown

- ▶ First, edit the Kafka configuration file in `config/server.properties` and add the following line:
 - `controlled.shutdown.enable=true`
- ▶ Start all the nodes
- ▶ With all the cluster nodes running, shut down one broker with the following command in the Kafka installation directory:
 - `$ bin/kafka-server-stop.sh`

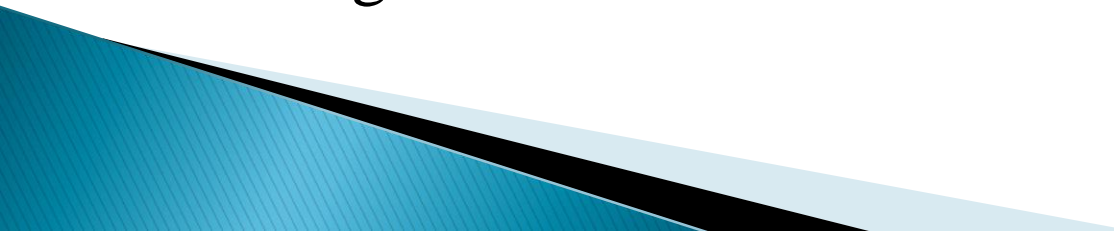
Implementing a graceful shutdown

- ▶ 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

- ▶ A leader broker of a topic partition can be crashed or stopped, and then the leadership is transferred to another replica.
- ▶ This might produce an imbalance in the lead Kafka brokers (an imbalance is when the leader is dead or unreachable).
- ▶ To recover from this imbalance, we need **balancing leadership**.
- ▶ Run the following command from the Kafka installation directory:
 - `$ bin/kafka-preferred-replica-election.sh --zookeeper localhost:2181/chroot`

Balancing leadership

- ▶ 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.
- 

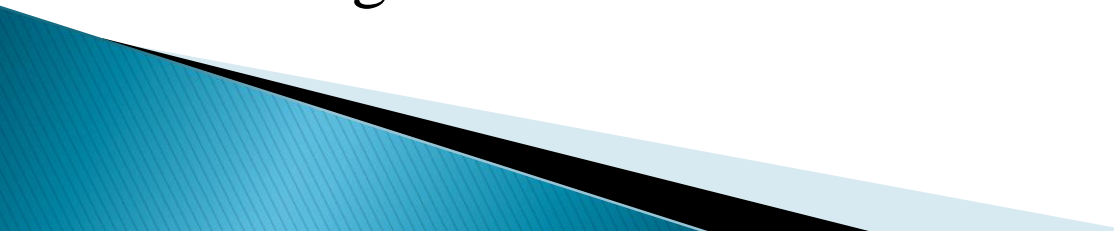
Balancing leadership

- ▶ The command takes the following parameter:
 - `--zookeeper <String: urls>`: This is the mandatory parameter. It specifies the ZooKeeper connect string and is a comma-separated list in the format `host:port`. This parameter is useful if you have more than one Kafka cluster using the same ZooKeeper cluster.

Expanding clusters

- ▶ A leader broker of a topic partition can be crashed or stopped, and then the leadership is transferred to another replica.
- ▶ This might produce an imbalance in the lead Kafka brokers (an imbalance is when the leader is dead or unreachable).
- ▶ To recover from this imbalance, we need **balancing leadership**.
- ▶ Run the following command from the Kafka installation directory:
 - `$ bin/kafka-preferred-replica-election.sh --zookeeperlocalhost:2181/chroot`

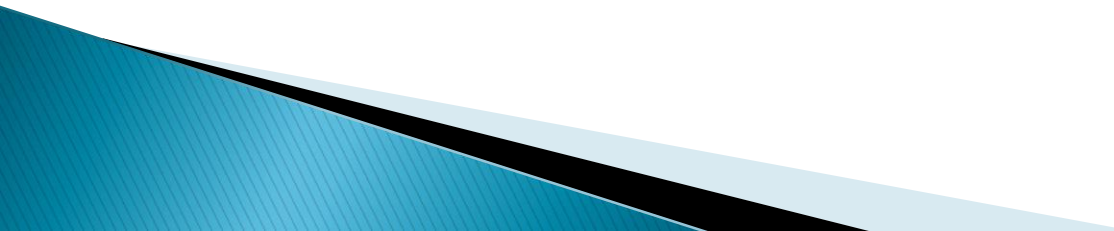
Expanding clusters

- ▶ 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.
- 

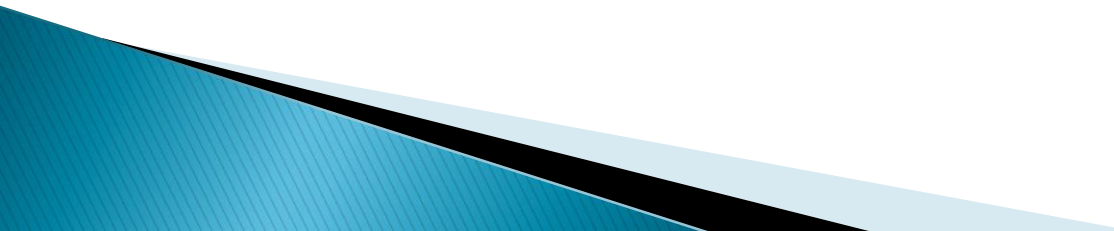
Expanding clusters

- ▶ The command takes the following parameter:
 - `--zookeeper <String: urls>`: This is the mandatory parameter. It specifies the ZooKeeper connect string and is a comma-separated list in the format `host:port`. This parameter is useful if you have more than one Kafka cluster using the same ZooKeeper cluster.

Expanding clusters

- ▶ Adding nodes to an existing cluster is not the same as building a new Kafka cluster.
 - ▶ Adding nodes to an existing cluster is easy.
 - ▶ We do this by assigning them a unique broker ID, but they are not going to receive data automatically.
 - ▶ A cluster reconfiguration is needed to indicate which partition replicas go where.
 - ▶ Then, the partitions will move to the newly added nodes.
- 

Expanding clusters

- ▶ 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`.
- 

Expanding clusters

- ▶ 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  
}
```

Expanding clusters

- ▶ 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.sh --zookeeper localhost:2181 --topics-to-move-json-file to_reassign.json --broker-list "7,8" --generate`
- ▶ The output is something like this:
Current partition replica assignment

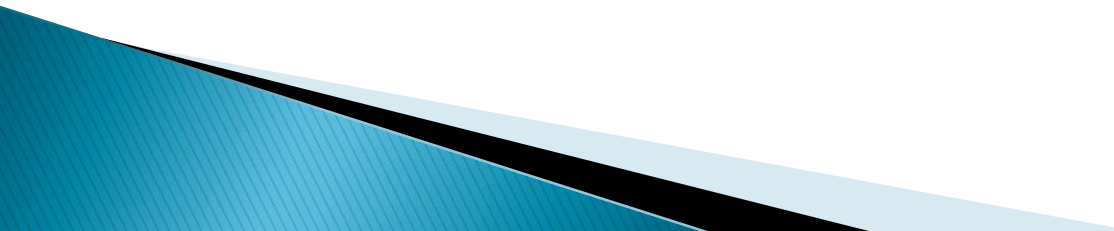
```
{ "version":1,  
  "partitions":[ { "topic":"topic_1","partition":0,"replicas":[1,2]},  
                  { "topic":"topic_1","partition":1,"replicas":[3,4]},  
                  { "topic":"topic_1","partition":2,"replicas":[5,6]},  
                  { "topic":"topic_2","partition":0,"replicas":[1,2]},  
                  { "topic":"topic_2","partition":1,"replicas":[3,4]},  
                  { "topic":"topic_2","partition":2,"replicas":[5,6]}}  
}
```

Expanding clusters

- ▶ The output is something like this:
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] } ]  
}
```

Expanding clusters

- ▶ 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.
 - ▶ 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.
- 

Expanding clusters

- ▶ 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]},  
}
```

Expanding clusters

- ▶ Now, to execute the reassignment, run the following command from the Kafka installation directory:
 - `$ bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --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]}]  
}
```


Expanding clusters

- ▶ Now, run the same command to verify the partition assignment:
 - **\$ bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --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

Expanding clusters

► How it works...

- The first step creates a JSON file with the topics to reassign.
- The second step generates a candidate configuration for the specified Kafka topics using the reassign partitions tool. This tool takes the following parameters:
 - `--broker-list <String: brokerlist>`: These are the brokers to which the partitions need to be reassigned, in the form 0, 1, 2. Required if `--topics-to-move-json-file` is used to generate reassignment configuration.
 - `--execute`: This is used to start the reassignment, as specified in `--reassignment-json-file`.
 - `--generate`: This is used to generate a candidate partition reassignment configuration. As seen, it does not execute it.

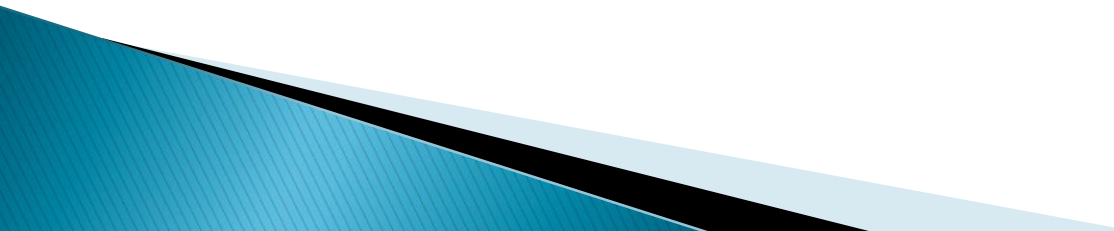
Expanding clusters

► How it works...

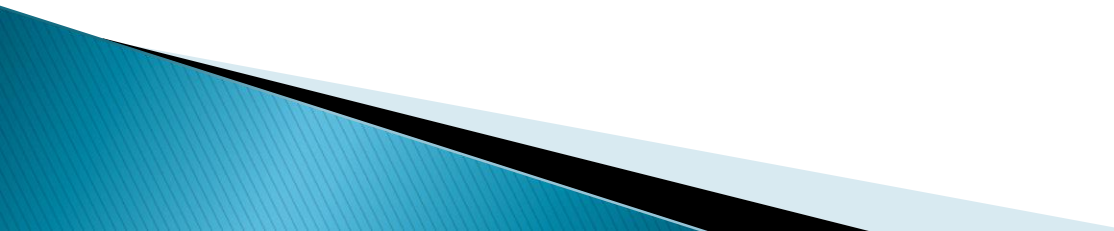
- `--reassignment-json-file <String: file>`: This is the JSON filename of the partition reassignment configuration.
- `--topics-to-move-json-file <String: file>`: This is used to generate a new assignment configuration, moving the partitions of the specified topics to the list of brokers indicated by the `--broker-list` option.
- `--verify`: This is used to verify whether the new assignment has completed as specified in the `--reassignment-json-file`.
- `--zookeeper <String: urls>`: This is a mandatory parameter: the connection string for the ZooKeeper connection, in the form `host:port`. Multiple URLs mean allowing fail-over.

Expanding clusters

► **How it works...**

- 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

- ▶ In cases where more machines are added to the Kafka cluster, increasing the replication factor means moving replicas for a topic to these new machines.
 - ▶ 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.
- 

Increasing the replication factor

- ▶ 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] } ] }
```

- ▶ Then, run the following command:
 - `$ bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --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.

Increasing the replication factor

- ▶ 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.
- ▶ The parameters this command takes are indicated in the previous recipe.
- ▶ To verify the status of the reassignment, run the following command:
 - `$ bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file increase-replication.json --verify`

Decommissioning brokers

- ▶ Removing some Kafka nodes from a cluster is called **decommissioning**.
- ▶ Decommissioning is not automatic; some reassignment must be applied to allow replicas to move to the live brokers.
- ▶ **Getting ready**
 - For this recipe, Kafka must be installed, ZooKeeper running, and a Kafka cluster running with at least three nodes.
 - A topic called topic1 with replication factor 3 should be running on the cluster.

Decommissioning brokers

► How to do it...

- First, gracefully shut down the broker to be removed
- 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]}]}
```
- Reassign the topic to the two living brokers with the `reassign-partitions` command:
  - ```
$ bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --  
  reassignment-json-file change-replication.json --execute
```

Decommissioning brokers

► How it works...

- After shutting down the node, proceed with the decommission of the partitions of that broker.
- Internally, the shutdown steps are as follows:
 - The logs for all the lead partitions on that node are flushed to disk
 - After the lead is transferred, the node is finally shut down
- In the JSON file, we specify which partition must be part of which replica. Obviously, we are removing all references to the decommissioned node.
- Running the command will update the partition replication information in the Kafka cluster with the instructions in the JSON file.

Checking the consumer position

- ▶ Here is a tool to check how much the consumers are lagging from the produced messages.
- ▶ **Getting ready**
 - For this recipe, Kafka must be installed, ZooKeeper running, and the broker running with some topics created on it.
 - Also, a consumer must be running to read from this topic.
- ▶ **How to do it...**
 - Run the following command from the Kafka directory:
 - `$ bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group vipConsumersGroup`

Checking the consumer position

- ▶ The output is something like the following:

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
source-topic	0		1				
0					consumer-1-beff4c31-e197-455b-89fb-cce53e380a26		
/192.168.1.87					consumer-1		

- ▶ **How it works...**

- The Kafka-Consumer-Groups command takes the following arguments:
 - `--group <String: consumer group>`: This is the consumer group to manipulate

Checking the consumer position

- `--bootstrap-server <String: server to connect>`: This is the server to connect to (for consumer groups based on non-old consumers)
 - `--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 is the topic whose consumer group information we manipulate
 - `--list` : This lists all the consumer groups of the broker
 - `--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 into a group to delete topic partition offsets and ownership information on the entire consumer group
- 