02: Apache Kafka multi-broker cluster tutorial

Posted on February 16, 2019

This extends Getting started with Apache Kafka on Mac tutorial. This assumes that the zookeeper & kafka servers are started as per the previous tutorial.

List topics

Create a topic first:

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-topics.sl
```

List topics:

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-topics.sl

test
```

Delete a topic

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-topics.sl
```

This delete command will have no effect if in the Kafka server.properties file, if **delete.topic.enable** is not set to be true.

Get info about a topic

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-topics.sl

Topic:test     PartitionCount:1     ReplicationFacto
    Topic: test     Partition: 0     Leader: 0     Rep

```

Creating topics with partitions & replications with multiple brokers

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-topics.sl

Error while executing topic command : Replication
[2019-02-16 12:36:36,802] ERROR org.apache.kafka.c
 (kafka.admin.TopicCommand$)
```

The above error is thrown because of a single broker. Kafka is a distributed system, hence you need to harness its strength by having more brokers across multiple nodes. Replication gives you fault tolerance.

Stop the Kafka server and make the following changes to have 3 brokers.

Step 1: Copy the**./config/server.properties** file into 2 additional **.properties** files.

```
/usr/local/kafka_2.11-2.1.0]$ cp ./config/server.pı
```

```
/usr/local/kafka_2.11-2.1.0]$ cp ./config/server.pı
```

Step 2: Modify **./config/server1.properties** & **./config/server2.properties** by changing the 3 property values as shown below.

```
/usr/local/kafka_2.11-2.1.0]$ vi ./config/server1.|
```

```

    broker.id=1
    .....
    listeners=PLAINTEXT://:9093
    ....
    log.dirs=/tmp/kafka-logs-1


```

```
/usr/local/kafka_2.11-2.1.0]$ vi ./config/server2.|
```

```

    broker.id=2
    .....
    listeners=PLAINTEXT://:9094
    ....
    log.dirs=/tmp/kafka-logs-2


```

Step 3: Start all 3 brokers (i.e. brokers 0, 1 & 2) on 3 different terminal windows

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-server-s

```

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-server-s

```

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-server-s

```

Step 4: Now create a topic with 5 partitions and a replication factor of 3.

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-topics.sl
```

```
Created topic "my-demo-topic".

```

Step 5: In a cluster, how do you know which broker is doing what? This is where **–describe** comes in handy.

```
/usr/local/kafka_2.11-2.1.0]$ ./bin/kafka-topics.sl
```

It shows the leader & the follower (i.e. replica) broker numbers. We have created 5 partitions for parallelism. "isr" means in sync replicas.

```

Topic:my-demo-topic    PartitionCount:5       Repl
    Topic: my-demo-topic    Partition: 0    Leader
    Topic: my-demo-topic    Partition: 1    Leader
    Topic: my-demo-topic    Partition: 2    Leader
    Topic: my-demo-topic    Partition: 3    Leader
    Topic: my-demo-topic    Partition: 4    Leader

```

Leader is the node responsible for all reads and writes for a given partition. Each broker will be the leader for a randomly selected portion of the partitions. Every partition in a Kafka topic has a write-ahead log where the messages are stored and every message has a unique offset that identifies it's position in the partition's log.

Replicas are the list of nodes that replicate the log for this partition regardless of whether they are the leader or even if they are currently alive. Every topic partition in Kafka is replicated n (e.g. 3) times, where n is the replication factor of the topic. This allows Kafka to automatically failover to these replicas when a server in the cluster fails so that messages remain available in the presence of failures. Replication in Kafka happens at the partition granularity where the partition's write-ahead log is replicated in order to n servers.

"isr" is the subset of the replicas list that is currently alive and caught-up to the leader. When a producer sends a message to the broker, it is written by the leader and replicated to all the partition's replicas. A message is committed only after it has been successfully copied to all the in-sync replicas. The leader for every partition tracks this in-sync replica (aka ISR) list by computing the lag of every replica from itself.

‹ 01: Getting started with Apache Kafka on Mac tutorial

03: Apache Kafka Connector Tutorial ›