INCEPTEZ TECHNOLOGIES KAFKA WORKOUTS

Start the Zookeeper Coordination service:

cd /usr/local/kafka/bin

./zookeeper-server-start.sh -daemon ../config/zookeeper.properties

Start the Kafka server:

./kafka-server-start.sh -daemon ../config/server.properties

Create a topic (with one replica and one partition)

./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic tk1

We can now see that topic if we run the list topic command:

./kafka-topics.sh --list --zookeeper localhost:2181

If you want to delete the topic (mark for deletion, at the time of flush will be deleted else use delete.topic.enable=true**)**

kafka-topics.sh --delete --zookeeper localhost:2181 --topic tk1

Produce messages using Producer:

Kafka comes with a command line client that will take input from a file or from standard input and send it out as messages to the Kafka cluster. By default each line will be sent as a separate message. Run the producer and then type a few messages into the console to send to the server.

./kafka-console-producer.sh --broker-list localhost:9092 --topic tk1

Start a Consumer

Kafka also has a command line consumer that will dump out messages to standard output.

kafka-console-consumer.sh --zookeeper localhost:2181 --topic tk1 --from-beginning

Only look at the incremental logs

kafka-console-consumer.sh --zookeeper localhost:2181 --topic tk1

If you have each of the above commands running in a different terminal then you should now be able to type messages into the producer terminal and see them appear in the consumer terminal.

Setting up a multi-broker cluster

So far we have been running against a single broker. For Kafka, a single broker is just a cluster of size one, so nothing much changes other than starting a few more broker instances. But just to get feel for it, let's expand our cluster to three nodes (still all on our local machine).

First we make a config file for each of the brokers:

cd /usr/local/kafka

cp config/server.properties config/server-1.properties

cp config/server.properties config/server-2.properties

Now edit these new files and set the following properties:

config/server-1.properties:

```
listeners=PLAINTEXT://:9093
broker.id=1
port=9093
log.dir=/tmp/kafka-logs-1
```

config/server-2.properties:

```
listeners=PLAINTEXT://:9094
broker.id=2
port=9094
log.dir=/tmp/kafka-logs-2
```

The broker.id property is the unique and permanent name of each node in the cluster. We have to override the port and log directory only because we are running these all on the same machine and we want to keep the brokers from all trying to register on the same port or overwrite each others data.

We already have Zookeeper and our single node started, so we just need to start the two new nodes:

```
./bin/kafka-server-start.sh -daemon config/server-1.properties
```

```
./bin/kafka-server-start.sh -daemon config/server-2.properties
```

Now create a new topic with a replication factor of three with a single partition:

./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 1 --topic tr43

./bin/kafka-topics.sh --list --zookeeper localhost:2181

Want to alter partitions use --alter

./bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic tr43 --partitions 4

How can we know which broker is doing what? To see that run the "describe topics" command:

./bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic tr43

Eg:

Topic:tr43 PartitionCount:1 ReplicationFactor:3 Configs:

Topic: tr43 Partition: 0 Leader: 1 Replicas: 1,2,0 Isr: 1,2,0

Here is an explanation of output. The first line gives a summary of all the partitions, each additional line gives information about one partition. Since we have only one partition for this topic there is only one line.

"leader" is the node responsible for all reads and writes for the given partition. Each node will be the leader for a randomly selected portion of the partitions.

"replicas" is the list of nodes that replicate the log for this partition regardless of whether they are the leader.

"isr" is the set of "in-sync" replicas. This is the subset of the replicas list that is currently alive and caught-up to the leader.

We can run the same command on the original topic we created to see where it is:

./bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic tk1

Topic:tk1 PartitionCount:1 ReplicationFactor:1 Configs:

Topic: tk1 Partition: 0 Leader: 0 Replicas: 0 Isr: 0

So there is no surprise there—the original topic has no replicas and is on server 0, the only server in our cluster when we created it.

Let's publish a few messages to our new topic tr43:

./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic tr43

Now let's consume these messages:

./bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic tr43 --from-beginning

Now let's test out fault-tolerance. Broker 1 was acting as the leader so let's kill it:

ps -ef | grep server-1.properties

7564 ttys002

kill -9 7564

Leadership has switched to one of the slaves and node 1 is no longer in the in-sync replica set:

kafka-topics.sh --describe --zookeeper localhost:2181 --topic tr43

Topic:tr43 PartitionCount:1 ReplicationFactor:3 Configs:

Topic: tr43 Partition: 0 Leader: 2 Replicas: 1,2,0 Isr: 2,0

But the messages are still be available for consumption even though the leader that took the writes originally is down:

kafka-console-consumer.sh --zookeeper localhost:2181 --from-beginning --topic tr43