# Kafka cluster

## Kafka cluster

Kafka is a data pipeline and data streaming application that provides a fault tolerant, centralized location for log processing and log aggregation.

## Configuration

**Overview**
Configuration files for both Kafka and the zookeeper are location in /opt/kafka/config.
All server instance scripts are located in /opt/kafka/bin.

**Servers**
We will use references to these two broker servers for examples in the documentation.

**broker-00**
Broker.id=0
Zookeeper Instance port - 2181
Kafka Instance port - 9092

**broker-01**
Broker.id=1
Kafka instance port - 9092

### Installing kafka

Download and unpack the kafka binaries on both broker-00 and broker-01.

```
wget http://apache.claz.org/kafka/2.2.0/kafka_2.12-2.2.0.tgz
tar -xzf kafka_2.12-2.2.0.tgz
mv kafka_2.12-2.2.0 /opt/kafka
```

### Zookeeper config

Since we're running Kafka in a clustered fashion, we need to configure and start the zookeeper processes first. Ideally multiple zookeepers should live on their own boxes in a peer fail-over state but
For right now, a single zookeeper instance will live on broker-00

zookeeper's config file is located at /opt/kafka/config/zookeeper.properties
The zookeeper properties file can stay default until multiple zookeepers are configured

```
#config details here
#right now keep the default settings
#introducing multiple zookeepers later will change defaults
```

Now start the zookeeper instance (server start/stop scripts are located in /opt/kafka/bin)
Each script needs the config file we just edited passed along with it

```
#inside the /opt/kafka/bin directory

#start the server in the foreground (useful for testing)
./zookeeper-server-start.sh ../config/zookeeper.properties

#start the server as a daemon
./zookeeper-server-start.sh -daemon ../config/zookeeper.properties
```

## Kafka server config

Now that the zookeeper is running, lets configure and start the two kafka brokers. Each broker needs to have a unique broker ID which we will start from 0.
We also need to add/change a few other parameters listed bellow, anything not listed below can be left as the default values for now.

Kafka's config file is located at /opt/kafka/config/server.properties
This needs to be edited accordingly on both broker nodes

```
#THIS MUST BE UNIQUE FOR EACH BROKER
broker.id=0

#this ensures leadership is transferred when a leader dies
auto.leader.rebalance.enable=true

#change localhost to current broker server
listeners = PLAINTEXT://localhost:9092

#log directories can be located wherever, comma separated for multiple
entries
log.dirs=/var/log/kafka

#comma separated for multiple zookeepers, right now we only have one
located on broker-00
zookeeper.connect=broker-00:2181
```

Now start your Kafka server instances
Remember to pass the config file along with the server start script

```
#inside the /opt/kafka/bin directory

#start the server in the foreground (useful for testing)
./kafka-server-start.sh ../config/server.properties

#start the server as a daemon
./kafka-server-start.sh -daemon ../config/server.properties
```

## Create and manage topics

Now that our zookeeper and kafka brokers are running, we can create a new topic and test our cluster.

Execute the kafka-topics.sh script located in /opt/kafka/bin to create a topic
The following command creates a topic named "topictest" with a replication factor of 1 and 4 kafka partitions.

```
./kafka-topics.sh --create --bootstrap-server broker-00:9092 --replication-
factor 1 --partitions 4 --topic topictest
```

Use the kafka.topics.sh script again to verify the topic was created and to list it's current status.

```
kafka-topics.sh --describe --bootstrap-server broker-00:9092 --topic
topictest

Topic:topictest        PartitionCount:4        ReplicationFactor:1
Configs:segment.bytes=1073741824
        Topic: topictest        Partition: 0        Leader: 0
Replicas: 0,1        Isr: 0,1
        Topic: topictest        Partition: 1        Leader: 0
Replicas: 0,1        Isr: 0,1
        Topic: topictest        Partition: 2        Leader: 0
Replicas: 0,1         Isr: 0,1
        Topic: topictest        Partition: 3        Leader: 0
Replicas: 0,1         Isr: 0,1
```

- Leader: Defines which node in the cluster is responsible for the read/write of that particular partition.
- Replicas: Defines what nodes in the cluster are replicating what partitions.
- Isr: Defines in-sync replica nodes, which are alive and caught up to the leaders.

We can now use kafka-console-producer.sh to publish messages to our new topic.
This publishes two messages "test message 1" and "test message 2" to the topic topictest

```
./kafka-console-producer.sh --broker-list broker-00:9092 --topic topictest
> test message 1
> test message 2
^C
```

Now lets consume these massages with kafka-console-consumer.sh
You should see the the messages you published from the previous step

```
./kafka-console-consumer.sh --bootstrap-server broker-00:9092 --from-
beginning --topic topictest
test message 1
test message 2
^C
```

## Testing fault-tolerance

To test automatic fail-over lets kill the kafka process on the leader node.

```
ps aux | grep -i server.properties
user   30357  1.4  4.2 4731600 338348 pts/1  Sl   10:27   0:10 java -Xmx1G
-Xms1G -server -XX:+UseG1GC....
kill -9 30357
```

Verify that the leader roll has moved to the second broker using the.
Notice that the the leader roll is now held by the second broker and that the first broker is removed as an in-sync replica.

```
./kafka-topics.sh --describe --bootstrap-server broker-01:9092 --topic
topictest


Topic:topictest        PartitionCount:4        ReplicationFactor:1
Configs:segment.bytes=1073741824
        Topic: topictest        Partition: 0        Leader: 1
Replicas: 0,1        Isr: 1
        Topic: topictest        Partition: 1        Leader: 1
Replicas: 0,1        Isr: 1
        Topic: topictest        Partition: 2        Leader: 1
Replicas: 0,1         Isr: 1
        Topic: topictest        Partition: 3        Leader: 1
Replicas: 0,1         Isr: 1
```

After verifying that leadership has changed, verify that you can still consume messages from your topic.

```
./kafka-console-consumer.sh --bootstrap-server broker-01:9092 --from-
beginning --topic topictest
test message 1
test message 2
^C
```