

Getting started with Kafka cluster lab

Understanding Kafka Failover

This Kafka lab picks up right where the first Kafka lab from the command line left off. The first lab has instructions on how to run ZooKeeper and use Kafka utils.

In this lab, we are going to run many Kafka Nodes on our development laptop so that you will need at least 16 GB of RAM for local dev machine. You can run just two servers if you have less memory than 16 GB. We are going to create a replicated topic. We then demonstrate consumer failover and broker failover. We also demonstrate load balancing Kafka consumers. We show how, with many groups, Kafka acts like a Publish/Subscribe. But, when we put all of our consumers in the same group, Kafka will load share the messages to the consumers in the same group (more like a queue than a topic in a traditional MOM sense).

If not already running, then start up ZooKeeper (`./run-zookeeper.sh` from the first lab). Also, shut down Kafka from the first lab.

Next, you need to copy server properties for three brokers (detailed instructions to follow). Then we will modify these Kafka server properties to add unique Kafka ports, Kafka log locations, and unique Broker ids. Then we will create three scripts to start these servers up using these properties, and then start the servers. Lastly, we create replicated topic and use it to demonstrate Kafka consumer failover, and Kafka broker failover.

Create three new Kafka server-n.properties files

In this section, we will copy the existing Kafka `server.properties` to `server-0.properties`, `server-1.properties`, and `server-2.properties`. Then we change `server-0.properties` to set `log.dirs` to `"/logs/kafka-0"`. Then we modify `server-1.properties` to set `port` to `9093`, `broker.id` to `1`, and `log.dirs` to `"/logs/kafka-1"`. Lastly modify `server-2.properties` to use `port` `9094`, `broker.id` `2`, and `log.dirs` `"/logs/kafka-2"`.

Copy server properties file

```
$ ~/kafka-training
$ mkdir -p lab2/config
$ cp kafka/config/server.properties kafka/lab2/config/server-0.properties
$ cp kafka/config/server.properties kafka/lab2/config/server-1.properties
$ cp kafka/config/server.properties kafka/lab2/config/server-2.properties
```

With your favorite text editor change server-0.properties so that `log.dirs` is set to `./logs/kafka-0`. Leave the rest of the file the same. Make sure `log.dirs` is only defined once.

~/kafka-training/lab2/config/server-0.properties

```
broker.id=0
port=9092
log.dirs=./logs/kafka-0
...
```

With your favorite text editor change `log.dirs`, `broker.id` and `log.dirs` of `server-1.properties` as follows.

~/kafka-training/lab2/config/server-1.properties

```
broker.id=1
port=9093
log.dirs=./logs/kafka-1
...
```

With your favorite text editor change `log.dirs`, `broker.id` and `log.dirs` of `server-2.properties` as follows.

~/kafka-training/lab2/config/server-2.properties

```
broker.id=2
port=9094
log.dirs=./logs/kafka-2
...
```

Create Startup scripts for three Kafka servers

The startup scripts will just run `kafka-server-start.sh` with the corresponding properties file.

~/kafka-training/lab2/start-1st-server.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config

cd ~/kafka-training

## Run Kafka
kafka/bin/kafka-server-start.sh \
"$CONFIG/server-0.properties"
```

~/kafka-training/lab2/start-2nd-server.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config

cd ~/kafka-training

## Run Kafka
kafka/bin/kafka-server-start.sh \
"$CONFIG/server-1.properties"
```

~/kafka-training/lab2/start-3rd-server.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config

cd ~/kafka-training
```

```
## Run Kafka
kafka/bin/kafka-server-start.sh \
"$CONFIG/server-2.properties"
```

Notice we are passing the Kafka server properties files that we created in the last step.

Now run all three in separate terminals/shells.

Run Kafka servers each in own terminal from ~/kafka-training/lab2

```
~/kafka-training/lab2
$ ./start-1st-server.sh
```

```
...
$ ./start-2nd-server.sh
```

```
...
$ ./start-3rd-server.sh
```

Give the servers a minute to startup and connect to ZooKeeper.

Create Kafka replicated topic my-failsafe-topic

Now we will create a replicated topic that the console producers and console consumers can use.

~/kafka-training/lab2/create-replicated-topic.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

kafka/bin/kafka-topics.sh --create \
--zookeeper localhost:2181 \
```

```
--replication-factor 3 \  
--partitions 13 \  
--topic my-failsafe-topic
```

Notice that the replication factor gets set to 3, and the topic name is `my-failsafe-topic`, and like before it has 13 partitions.

Then we just have to run the script to create the topic.

Run create-replicated-topic.sh

```
~/kafka-training/lab2  
$ ./create-replicated-topic.sh
```

Start Kafka Consumer that uses Replicated Topic

Next, create a script that starts the consumer and then start the consumer with the script.

~/kafka-training/lab2/start-consumer-console-replicated.sh

```
#!/usr/bin/env bash  
cd ~/kafka-training  
  
kafka/bin/kafka-console-consumer.sh \  
  --bootstrap-server localhost:9094,localhost:9092 \  
  --topic my-failsafe-topic \  
  --from-beginning
```

Notice that a list of Kafka servers is passed to `--bootstrap-server` parameter. Only, two of the three servers get passed that we ran earlier. Even though only one broker is needed, the consumer client will learn about the other broker from just one server. Usually, you list multiple brokers in case there is an outage so that the client can connect.

Now we just run this script to start the consumer.

Run start-consumer-console-replicated.sh

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh
```

Start Kafka Producer that uses Replicated Topic

Next, we create a script that starts the producer. Then launch the producer with the script you create.

~/kafka-training/lab2/start-consumer-producer-replicated.sh

```
#!/usr/bin/env bash  
cd ~/kafka-training  
  
kafka/bin/kafka-console-producer.sh \  
--broker-list localhost:9092,localhost:9093 \  
--topic my-failsafe-topic
```

Notice we start Kafka producer and pass it a list of Kafka Brokers to use via the parameter `--broker-list`.

Now use the start producer script to launch the producer as follows.

Run start-producer-console-replicated.sh

```
~/kafka-training/lab2  
$ ./start-consumer-producer-replicated.sh
```

Now send messages

Now send some message from the producer to Kafka and see those messages consumed by the consumer.

Producer Console

```
~/kafka-training/lab2
$ ./start-consumer-producer-replicated.sh
Hi Mom
How are you?
How are things going?
Good!
```

Consumer Console

```
~/kafka-training/lab2
$ ./start-consumer-console-replicated.sh
Hi Mom
How are you?
How are things going?
Good!
```

Now Start two more consumers and send more messages

Now Start two more consumers in their own terminal window and send more messages from the producer.

Producer Console

```
~/kafka-training/lab2
$ ./start-consumer-producer-replicated.sh
Hi Mom
How are you?
How are things going?
Good!
```

```
message 1  
message 2  
message 3
```

Consumer Console 1st

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh  
Hi Mom  
How are you?  
How are things going?  
Good!  
message 1  
message 2  
message 3
```

Consumer Console 2nd in new Terminal

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh  
Hi Mom  
How are you?  
How are things going?  
Good!  
message 1  
message 2  
message 3
```

Consumer Console 2nd in new Terminal

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh  
Hi Mom  
How are you?  
How are things going?
```



```
Good!  
message 1  
message 2  
message 3
```

Notice that the messages are sent to all of the consumers because each consumer is in a different consumer group.

Change consumer to be in their own consumer group

Stop the producers and the consumers from before, but leave Kafka and ZooKeeper running.

Now let's modify the `start-consumer-console-replicated.sh` script to add a Kafka *consumer group*. We want to put all of the consumers in same *consumer group*. This way the consumers will share the messages as each consumer in the *consumer group* will get its share of partitions.

~/kafka-training/lab2/start-consumer-console-replicated.sh

```
#!/usr/bin/env bash  
cd ~/kafka-training  
  
kafka/bin/kafka-console-consumer.sh \  
  --bootstrap-server localhost:9094,localhost:9092 \  
  --topic my-failsafe-topic \  
  --consumer-property group.id=mygroup
```

Notice that the script is the same as before except we added `--consumer-property group.id=mygroup` which will put every consumer that runs with this script into the `mygroup` consumer group.

Now we just run the producer and three consumers.

Run this three times - start-consumer-console-replicated.sh

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh
```

Run Producer Console

```
~/kafka-training/lab2  
$ ./start-consumer-producer-replicated.sh
```

Now send seven messages from the Kafka producer console.

Producer Console

```
~/kafka-training/lab2  
$ ./start-consumer-producer-replicated.sh  
m1  
m2  
m3  
m4  
m5  
m6  
m7
```

Notice that the messages are spread evenly among the consumers.

1st Kafka Consumer gets m3, m5

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh  
m3  
m5
```

Notice the first consumer gets messages m3 and m5.

2nd Kafka Consumer gets m2, m6

```
~/kafka-training/lab2
$ ./start-consumer-console-replicated.sh
m2
m6
```

Notice the second consumer gets messages m2 and m6.

3rd Kafka Consumer gets m1, m4, m7

```
~/kafka-training/lab2
$ ./start-consumer-console-replicated.sh
m1
m4
m7
```

Notice the third consumer gets messages m1, m4 and m7.

Notice that each consumer in the group got a share of the messages.

Kafka Consumer Failover

Next, let's demonstrate consumer failover by killing one of the consumers and sending seven more messages. Kafka should divide up the work to the consumers that are running.

First, kill the third consumer (CTRL-C in the consumer terminal does the trick).

Now send seven more messages with the Kafka console-producer.

Producer Console - send seven more messages m8 through m14

```
~/kafka-training/lab2
$ ./start-consumer-producer-replicated.sh
m1
...
m8
m9
```

```
m10  
m11  
m12  
m13  
m14
```

Notice that the messages are spread evenly among the remaining consumers.

1st Kafka Consumer gets m8, m9, m11, m14

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh  
m3  
m5  
m8  
m9  
m11  
m14
```

The first consumer got m8, m9, m11 and m14.

2nd Kafka Consumer gets m10, m12, m13

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh  
m2  
m6  
m10  
m12  
m13
```

The second consumer got m10, m12, and m13.

We killed one consumer, sent seven more messages, and saw Kafka spread the load to remaining consumers. ***Kafka consumer failover works!***

Create Kafka Describe Topic Script

You can use `kafka-topics.sh` to see how the Kafka topic is laid out among the Kafka brokers. The `--describe` will show partitions, ISRs, and broker partition leadership.

~/kafka-training/lab2/describe-topics.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

# List existing topics
kafka/bin/kafka-topics.sh --describe \
    --topic my-failsafe-topic \
    --zookeeper localhost:2181
```

Let's run `kafka-topics.sh --describe` and see the topology of our `my-failsafe-topic`.

Run describe-topics

We are going to lists which broker owns (leader of) which partition, and list replicas and ISRs of each partition. ISRs are replicas that are up to date. Remember there are 13 topics.

Topology of Kafka Topic Partition Ownership

```
~/kafka-training/lab2
$ ./describe-topics.sh
Topic: my-failsafe-topic    PartitionCount: 13    ReplicationFactor: 3
Configs:
```

Topic: my-failsafe-topic	Partition: 0	Leader: 2	Replicas: 2,0,1
Isr: 2,0,1			
Topic: my-failsafe-topic	Partition: 1	Leader: 0	Replicas: 0,1,2
Isr: 0,1,2			
Topic: my-failsafe-topic	Partition: 2	Leader: 1	Replicas: 1,2,0
Isr: 1,2,0			
Topic: my-failsafe-topic	Partition: 3	Leader: 2	Replicas: 2,1,0
Isr: 2,1,0			
Topic: my-failsafe-topic	Partition: 4	Leader: 0	Replicas: 0,2,1
Isr: 0,2,1			
Topic: my-failsafe-topic	Partition: 5	Leader: 1	Replicas: 1,0,2
Isr: 1,0,2			
Topic: my-failsafe-topic	Partition: 6	Leader: 2	Replicas: 2,0,1
Isr: 2,0,1			
Topic: my-failsafe-topic	Partition: 7	Leader: 0	Replicas: 0,1,2
Isr: 0,1,2			
Topic: my-failsafe-topic	Partition: 8	Leader: 1	Replicas: 1,2,0
Isr: 1,2,0			
Topic: my-failsafe-topic	Partition: 9	Leader: 2	Replicas: 2,1,0
Isr: 2,1,0			
Topic: my-failsafe-topic	Partition: 10	Leader: 0	Replicas: 0,2,1
Isr: 0,2,1			
Topic: my-failsafe-topic	Partition: 11	Leader: 1	Replicas: 1,0,2
Isr: 1,0,2			
Topic: my-failsafe-topic	Partition: 12	Leader: 2	Replicas: 2,0,1
Isr: 2,0,1			

Notice how each broker gets a share of the partitions as leaders and followers. Also, see how Kafka replicates the partitions on each broker.

Test Broker Failover by killing 1st server

Let's kill the first broker, and then test the failover.

Kill the first broker

```
$ kill `ps aux | grep java | grep server-0.properties | tr -s " " | cut -d "
" -f2`
```

You can stop the first broker by hitting CTRL-C in the broker terminal or by running the above command.

Now that the first Kafka broker has stopped, let's use Kafka `topics describe` to see that new leaders were elected!

Run describe-topics again to see leadership change

```
~/kafka-training/lab2/solution
```

```
$ ./describe-topics.sh
```

```
Topic:my-failsafe-topic    PartitionCount:13    ReplicationFactor:3
```

```
Configs:
```

```
    Topic: my-failsafe-topic    Partition: 0    Leader: 2    Replicas: 2,0,1
Isr: 2,1
```

```
    Topic: my-failsafe-topic    Partition: 1    Leader: 1    Replicas: 0,1,2
Isr: 1,2
```

```
    Topic: my-failsafe-topic    Partition: 2    Leader: 1    Replicas: 1,2,0
Isr: 1,2
```

```
    Topic: my-failsafe-topic    Partition: 3    Leader: 2    Replicas: 2,1,0
Isr: 2,1
```

```
    Topic: my-failsafe-topic    Partition: 4    Leader: 2    Replicas: 0,2,1
Isr: 2,1
```

```
    Topic: my-failsafe-topic    Partition: 5    Leader: 1    Replicas: 1,0,2
Isr: 1,2
```

```
    Topic: my-failsafe-topic    Partition: 6    Leader: 2    Replicas: 2,0,1
Isr: 2,1
```

```
    Topic: my-failsafe-topic    Partition: 7    Leader: 1    Replicas: 0,1,2
Isr: 1,2
```

```
    Topic: my-failsafe-topic    Partition: 8    Leader: 1    Replicas: 1,2,0
Isr: 1,2
```

```
    Topic: my-failsafe-topic    Partition: 9    Leader: 2    Replicas: 2,1,0
Isr: 2,1
```

```
    Topic: my-failsafe-topic    Partition: 10    Leader: 2    Replicas: 0,2,1
Isr: 2,1
```

```
    Topic: my-failsafe-topic    Partition: 11    Leader: 1    Replicas: 1,0,2
```

```
Isr: 1,2
    Topic: my-failsafe-topic    Partition: 12    Leader: 2    Replicas: 2,0,1
Isr: 2,1
```

Notice how Kafka spreads the leadership over the 2nd and 3rd Kafka brokers.

Show Broker Failover Worked

Let's prove that failover worked by sending two more messages from the producer console.

Then notice if the consumers still get the messages.

Send the message m15 and m16.

Producer Console - send m15 and m16

```
~/kafka-training/lab2
$ ./start-consumer-producer-replicated.sh
m1
...
m15
m16
```

Notice that the messages are spread evenly among the remaining live consumers.

1st Kafka Consumer gets m16

```
~/kafka-training/lab2
$ ./start-consumer-console-replicated.sh
m3
m5
m8
m9
m11
m14
```



```
...  
m16
```

The first Kafka broker gets m16.

2nd Kafka Consumer gets m15

```
~/kafka-training/lab2  
$ ./start-consumer-console-replicated.sh  
m2  
m6  
m10  
m12  
m13  
...  
m15
```

The second Kafka broker gets m15.

Kafka broker Failover WORKS!

Kafka Cluster Failover Review

Why did the three consumers not load share the messages at first?

They did not load share at first because they were each in a different consumer group. Consumer groups each subscribe to a topic and maintain their own offsets per partition in that topic.

How did we demonstrate failover for consumers?

We shut a consumer down. Then we sent more messages. We observed Kafka spreading messages to the remaining cluster.

How did we show failover for producers?

We didn't. We showed failover for Kafka brokers by shutting one down, then using the producer console to send two more messages. Then we saw that the producer used the remaining Kafka brokers. Those Kafka brokers then delivered the messages to the live consumers.

What tool and option did we use to show ownership of partitions and the ISR's?

We used `kafka-topics.sh` using the `--describe` option.