Kafka Basic Operations

First let us start implementing “single node-single broker” configuration and we will then migrate our setup to single node-multiple brokers configuration.

Hopefully you would have installed Java, ZooKeeper and Kafka on your machine by now. Before moving to the Kafka Cluster Setup, first you would need to start your ZooKeeper because Kafka Cluster uses ZooKeeper.

> *Note: IF on Ambari no need to start this*

To start Kafka Broker, type the following command:

```
bin/zookeeper-server-start.sh  config/zookeeper.properties
```

After starting Kafka Broker, go to root:

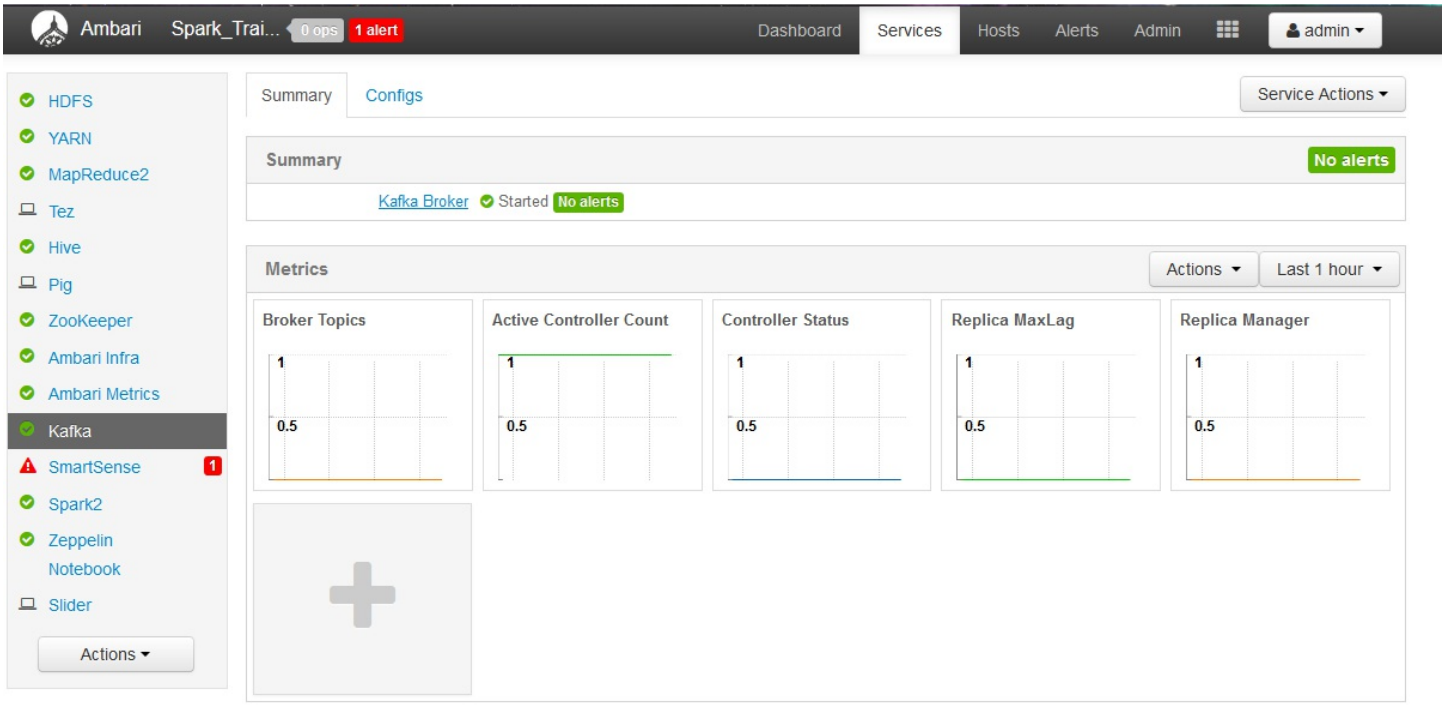
```
[centos@ip-172-30-9-71 ~]$ sudo su -
[root@ip-172-30-9-71 ~]#
```

and type the command `jps` (on the ZooKeeper terminal) and you would see the following response:

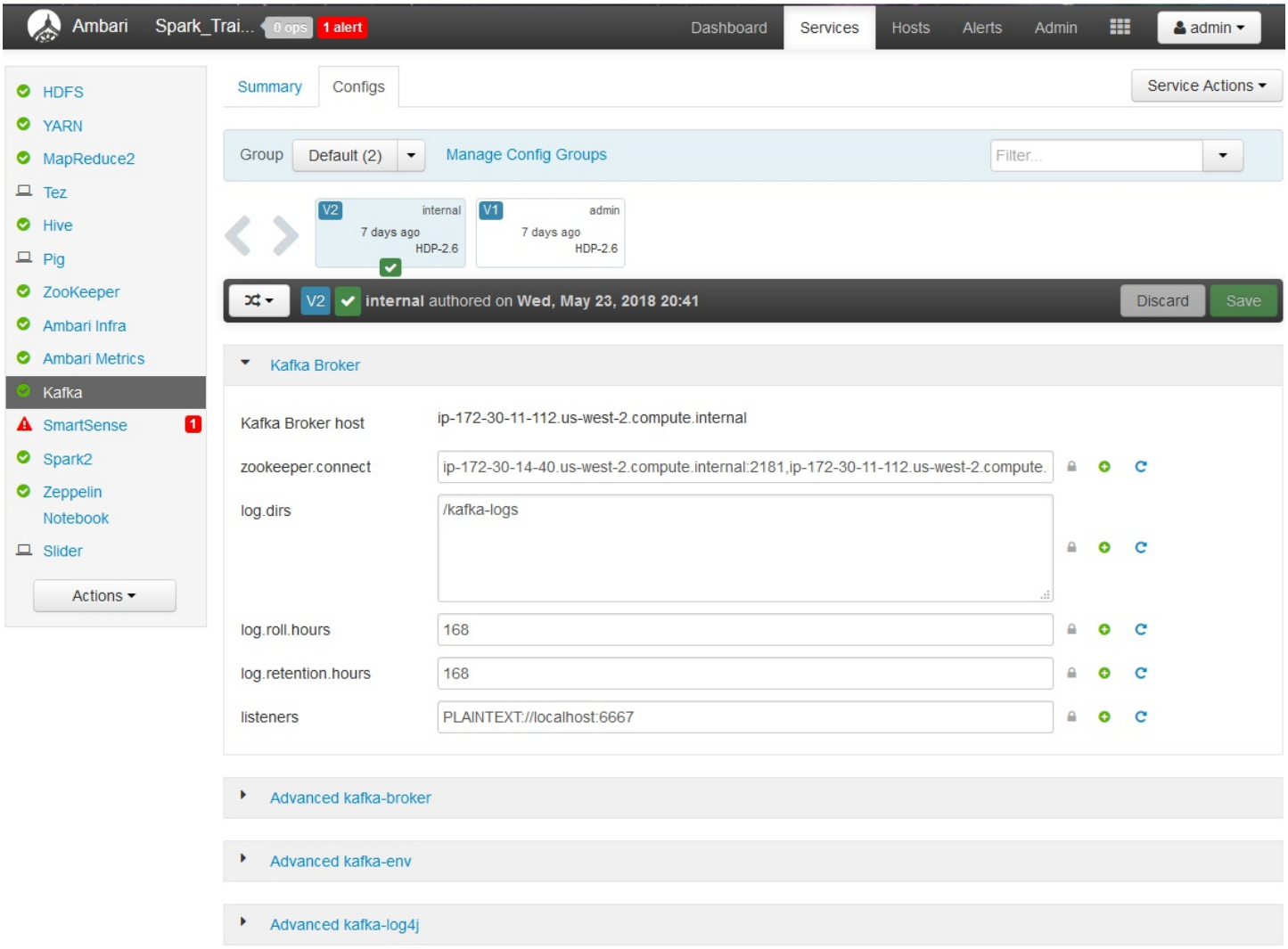
```
6451 Kafka
12099 NodeManager
```

```
6262 DataNode
11526 ZeppelinServer
12792 HistoryServer
11946 LivyServer
6140 QuorumPeerMain
12942 SparkSubmit
3199 Jps
```

If you are on Ambari, Kafka is here:



And the configs are found here:



Now you could see two daemons running on the terminal where `QuorumPeerMain` is ZooKeeper daemon and another one is Kafka daemon.

Single Node-Single Broker Configuration

In this configuration you have a single ZooKeeper and broker id instance. Following are the steps to configure it:

Creating a Kafka Topic

Kafka provides a command line utility named `kafka-topics.sh` to create topics on the server. Open new terminal and type the below example.

Syntax

```
  bin/kafka-topics.sh  --create  --zookeeper  localhost:2181
 --replication-factor  1  --partitions  1  --topic  topic-nam
e
```

Example

```
  bin/kafka-topics.sh  --create  --zookeeper  localhost:2181
 --replication-factor  1  --partitions  1  --topic  Hello-Kaf
ka
```

We just created a topic named “Hello-Kafka” with a single partition and one replica factor. The above created output will be similar to the following output:

Output:

```
  Created topic “Hello-Kafka”
```

Once the topic has been created, you can get the notification in Kafka broker terminal window and the log for the created topic specified in `/tmp/kafka-logs/` in the `config/server.properties` file.

List of Topics

To get a list of topics in Kafka server, you can use the following command:

Syntax

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

Output

```
Hello-Kafka
```

Since we have created a topic, it will list out `Hello-Kafka` only. Suppose, if you create more than one topics, you will get the topic names in the output.

Start Producer to Send Messages

Syntax

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

From the above syntax, two main parameters are required for the producer command line client:

- Broker-list - The list of brokers that we want to send the messages to. In this case we only have one broker.
- The `config/server.properties` file contains broker `port id`, since we know our broker is listening on port 9092, so you can specify it directly.

Topic name

Here is an example for the topic name:

```
bin/kafka-console-producer.sh  --broker-list  localhost:9092
 --topic  Hello-Kafka
```

The producer will wait on input from stdin and publishes to the Kafka cluster. By default, every new line is published as a new message then the default producer properties are specified in `config/producer.properties` file. Now you can type a few lines of messages in the terminal as shown below.

Output

```
$  bin/kafka-console-producer.sh  --broker-list  localhost:90
92  --topic  Hello-Kafka

[2016-01-16  13:50:45,931]  WARN  property  topic  is  not  v
alid  (kafka.utils.Verifia- bleProperties)
Hello
My first message
My second message
```

Start Consumer to Receive Messages

Similar to producer, the default consumer properties are specified in

`config/consumer.properties` file. Open a new terminal and type the below syntax for consuming messages:

Syntax

```
bin/kafka-console-consumer.sh  --zookeeper  localhost:2181  –
topic  topic-name  --from-beginning
```

Example

```
bin/kafka-console-consumer.sh  --zookeeper  localhost:2181  –
topic  Hello-Kafka  --from-beginning
```

Output

```
Hello
My first message
My second message
```

Finally, you are able to enter messages from the producer’s terminal and see them appearing in the consumer’s terminal. As of now, you have a very good understanding on the single node cluster with a single broker. Let us now move on to the multiple brokers configuration.

Single Node with Multiple Brokers Configuration

Before moving on to the multiple brokers cluster setup, first start your ZooKeeper server.

Create Multiple Kafka Brokers

We have one Kafka broker instance already in `config/server.properties`. Now we need multiple broker instances, so copy the existing `server.properties` file into two new config files and rename it as server-one.properties and `server-two.properties`. Then edit both new files and assign the following changes:

```
# The id of the broker. This must be set to a uni
que integer for each broker.
broker.id=1
# The port the socket server listens on
port=9093
# A comma seperated list of directories under which to store
log files
log.dirs=/tmp/kafka-logs-1
```

config/server-one.properties

```
# The id of the broker. This must be set to a uni
que integer for each broker.
broker.id=1
# The port the socket server listens on
port=9093
# A comma seperated list of directories under which to store
```

```
log files
log.dirs=/tmp/kafka-logs-1
```

config/server-two.properties

```
# The id of the broker. This must be set to a uni
que integer for each broker.
broker.id=2
# The port the socket server listens on
port=9094
# A comma seperated list of directories under which to store
log files
log.dirs=/tmp/kafka-logs-2
```

Start Multiple Brokers

After all the changes have been made on three servers then open three new terminals to start each broker one by one.

Now we have three different brokers running on the machine. Try it by yourself to check all the daemons by typing “jps” on the ZooKeeper terminal, then you would see the response:

```
Broker1
bin/kafka-server-start.sh  config/server.properties
Broker2
bin/kafka-server-start.sh  config/server-one.properties
Broker3
```

```
bin/kafka-server-start.sh  config/server-two.properties
```

Creating a Topic

Let us assign the replication factor value as three for this topic because we have three different brokers running. If you have two brokers, then the assigned replica value will be two:

Syntax

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

Example

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

Output

```
created  topic  “Multibrokerapplication”
```

The `describe` command is used to check which broker is listening on the current created topic as shown below:

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

```
 --topic  Multibrokerapplication
```

Output

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

Topic:Multibrokerapplication
PartitionCount:1
ReplicationFactor:3
Configs: Topic:Multibrokerapplication
Partition:0
Leader:0
Replicas:0,2,1
Isr:0,2,1
```

From the above output, we can conclude that first line gives a summary of all the partitions, showing topic name, partition count and the replication factor that we have chosen already. In the second line, each node will be the leader for a randomly selected portion of the partitions.

In our case, we see that our first broker (with `broker.id 0`) is the leader. Then `Replicas:0,2,1` means that all the brokers replicate the topic finally “Isr" is the set of “in-sync” replicas. Well, this is the subset of replicas that are currently alive and caught up by the leader.

Start Producer to Send Messages

This procedure remains the same as in the single broker setup:

Example

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

Output

```
bin/kafka-console-producer.sh  --broker-list  localhost:9092
 --topic  Multibrokerapplication
[2016-01-20  19:27:21,045]  WARN  Property  topic  is  not  v
alid  (kafka.utils.VerifiableProperties)
This  is  single  node-multi  broker  demo This is the second
 message
```

Start Consumer to Receive Messages

This procedure remains the same as shown in the single broker setup.

Example

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

Output

```
bin/kafka-console-consumer.sh  --zookeeper  localhost:2181  –
```

```
-topic  Multibrokerapplication  --from-beginning
This  is  single  node-multi  broker  demo
This is the second message
```

Basic Topic Operations

In this chapter we will discuss the various basic topic operations.

Modifying a Topic

As you have already understood how to create a topic in Kafka Cluster. Now let us modify a created topic using the following command:

Syntax

```
bin/kafka-topics.sh  –zookeeper  localhost:2181  --alter  --t
opic  topic_name  --partitions count
```

Example

We have already created a topic “Hello-Kafka” with single partition count and one replica factor. Now using “alter” command we have changed the partition count:

```
bin/kafka-topics.sh  --zookeeper  localhost:2181  --alter  --
topic  Hello-kafka  --partitions 2
```

Output

```
WARNING: If partitions are increased for a topic that
  has a key, the partition logic or ordering of the
messages will be affected
Adding partitions succeeded!
```

Deleting a Topic

To delete a topic, you can use the following syntax.

Syntax

```
bin/kafka-topics.sh --zookeeper localhost:2181 --delete -
-topic topic_name
```

Example

```
bin/kafka-topics.sh --zookeeper localhost:2181 --delete -
-topic Hello-kafka
```

Output

```
> Topic Hello-kafka marked for deletion
```

> *Note: This will have no impact if `delete.topic.enable` is not set to true*