**Setup kafka + Tutorials**

* Make sure, Java 1.8 or greater is installed.
* Go to the root of the machine by command “cd /”.
* Use below command to download kafka.

>> wget <http://apachemirror.wuchna.com/kafka/2.6.0/kafka_2.12-2.6.0.tgz>

* Now, to unzip it, use the below command, use your zip name instead of this one.

>> tar -xvf *kafka\_2.12-2.6.0.tgz*

* Now, set PATH by below command.

>> cd

>> export PATH=/kafka\_2.12-2.6.0/bin:$PATH

* Check if the kafka command is working or not by below command.

>> kafka-topics.sh

* Now, we’ll create a data directory for kafka and zookeeper. Use the below commands.

>> cd /

>> cd *{kafka\_folder}*

>> mkdir data/zookeeper

>> mkdir data/kafka

* Now, we’ve to add this data path to zookeeper.properties and server.properties under the config folder of kafka.
* So, keep your terminal in the root of the kafka folder and use this command to edit zookeeper.properties.

>> nano config/zookeeper.properties

* It’ll open the nano editor with zookeeper.properties. In there, find ‘dataDir’, and modify it like below.

dataDir=/kafka\_2.12-2.6.0/data/zookeeper

Save it and exit from nano.

* Now, we’ll add kafka dir server.properties to save the logs in the data folder we just created for kafka, so similar to above, follow below commands (in the root of kafka home dir).

>> nano config/server.properties.

* It’ll open the nano editor with server.properties. In there, find ‘log.dirs’, and modify it like below.

log.dirs=/kafka\_2.12-2.6.0/data/kafka

Save it and exit from nano.

* Now, we’re good to start kafka and zookeeper, but first, we’ll start zookeeper and then kafka.
* So, for that, open two terminals to run two servers, bring the current directory to root folder of kafka (‘cd /{*kafka folder})*, and run below commands

Terminal 1:

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

* Make sure it is started by checking for port 2181 in logs.

Terminal 2:

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

* Make sure it is started by checking [Kafka Server if=0] in logs.
* That’s it, we just setted up the environment for kafka, now get started with Kafka CLI tutorials to use it.

**Tutorial :**

* Kafka is a distributed message streaming platform that uses publish and subscribe mechanism to stream records.
* In here, the meaning of records, data and message will be same.
* If you’ve started KafkaServer as above instruction, we started them once only, so that means we’ve only one broker running.
* To create topic in kafka, we’ve to pass several mandatory arguments with it, which are --zookeeper and it’s host:port, --topic*topic\_name,* --create, --partitions *partition\_size(integer),* --replication-factor *replication\_factor\_size(integer).*
* Now, in kafka, you can’t pass the size of replication factor greater then number of brokers running on instance. So here, we’ve to pass 1 as a replication factor because we’ve started only one kafka server or broker.
* To create a topic named fiest-topic with partition size of 3 and repl-fact 1, command is given below.

>> kafka-topics.sh --zookeeper localhost:2181 --topic first\_topic --create --partitions 3 --replication-factor 1

O/p : Created topic first\_topic.

* To know if our topic is created or not, run this command.

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

* It’ll show you all the topics list.
* Now, to describe a particular topic, run below command..

>> kafka-topics.sh -zookeeper localhost:2181 --topic first\_topic --describe

O/p :

Topic: first\_topic PartitionCount: 3 ReplicationFactor: 1 Configs:

Topic: first\_topic Partition: 0 Leader: 0 Replicas: 0 Isr: 0

Topic: first\_topic Partition: 1 Leader: 0 Replicas: 0 Isr: 0

Topic: first\_topic Partition: 2 Leader: 0 Replicas: 0 Isr: 0

* In above output, topic is name of topic, second column shows the partitions 0, 1 and 2. Third column shows leader which is 0 that means broker Id is 0. And if you check KafkaServer log, you’ll get this many times like broker 0. So, that means the leading broker’s id is zero. Replicas and ISR also shows the broker Id to zero.
* To delete the topic,

>> kafka-topics.sh --zookeeper localhost:2181 --topic first\_topic --delete

Now, if we try to list all topics, first\_topic should be disappeared from there.

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

* Now, well launch the kafka console producer, so create a topic from the instructions above as first\_topic which will be used for kafka-console-producer.
* In this console producer, we’ll produce the messages as hello topic first, i’m vrushank and so on…..

>> kafka-console-producer.sh --broker-list localhost:9092 --topic first\_topic

* If above command run successfully, then it’ll ask for message by > sign.
* So, i’ll pass messages as

>hello topic first

>i'm vrushank

>learning kafka

>how're you..

* And then, to exit producing messages, press Ctrl + C.
* Now, these messages are published.
* In above case, producer was using default properties, let’s use acks properties.

>> kafka-console-producer.sh --broker-list localhost:9092 --topic first\_topic --producer-property acks=all

* And we’ll pass messages as

>just for fun

>fun learning

* And these messages will be published with acks properties.
* Now, what if we use producer with a topic which does not exist. Well, in that case, kafka will create that topic for you, but when you publish first message, it’ll give you a warning.
* Let’s use new\_topic which doesn’t exists.

>> kafka-console-producer.sh --broker-list localhost:9092 --topic new\_topic

>hey, this topic doesn't exist

[2020-10-14 02:59:11,469] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 3 : {new\_topic=LEADER\_NOT\_AVAILABLE} (org.apache.kafka.clients.NetworkClient)

>this is new message

>new one message

>^C

* In above case, it’ll create new topic, we can check the log of kafka server, we can see it is creating new topic for us, so we can list all the topics and describe the new\_topic to see what parameterized of topic is created by default.

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

first\_topic

new\_topic

>> kafka-topics.sh -zookeeper localhost:2181 --topic new\_topic --describe

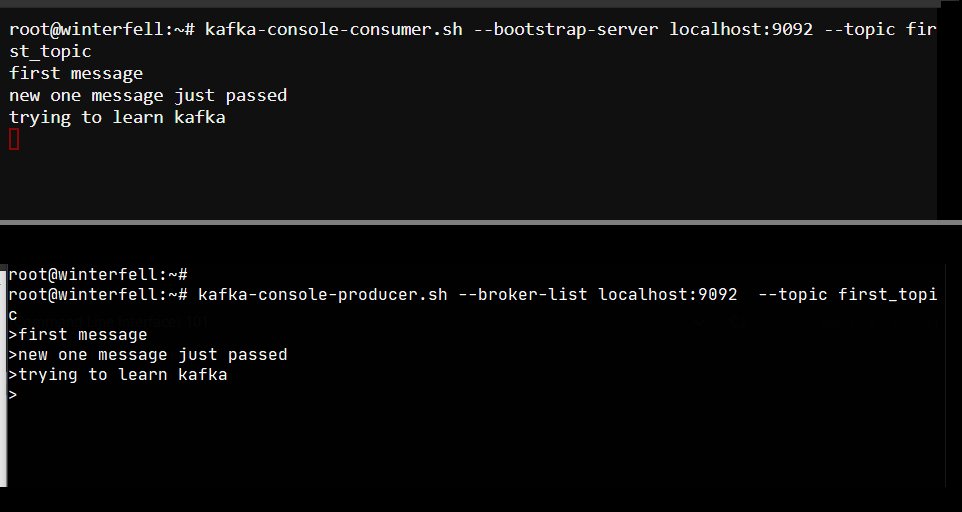
Topic: new\_topic PartitionCount: 1 ReplicationFactor: 1 Configs:

Topic: new\_topic Partition: 0 Leader: 0 Replicas: 0 Isr: 0

* So, as we can see, the default created topic is with parameters which are not expected, may be we need more petitions and replicas.
* So, it is strongly recommended to create topic first and then use it.
* Still, if we want to configure the default configuration for auto creating new partitions, then we can apply changes in server.properties under config folder. Open server.properties in config folder by nano editor and change line where num.partitions=1 to num.partition=3. Now restart the kafka server and try using producer for topic which does not exists. It’ll create that topic by default with 3 partitions.
* Now, let’s try to consume these messages.
* For that, we’ll use kafka-console-consumer.sh, the required arguments are --bootstrap-server *kafka serverhost:port,* --topic *topicname.*
* Let’s try to consume messages for first\_topic.

>> kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic first\_topic

* Now, it’ll not show us all the messages we produced before in first\_topic, but it’ll wait for new messages in first\_topic, so now, open another instance of terminal, and by kafka-console-producer, produce the new messages in it, and the consumer will show those new messages immediately.



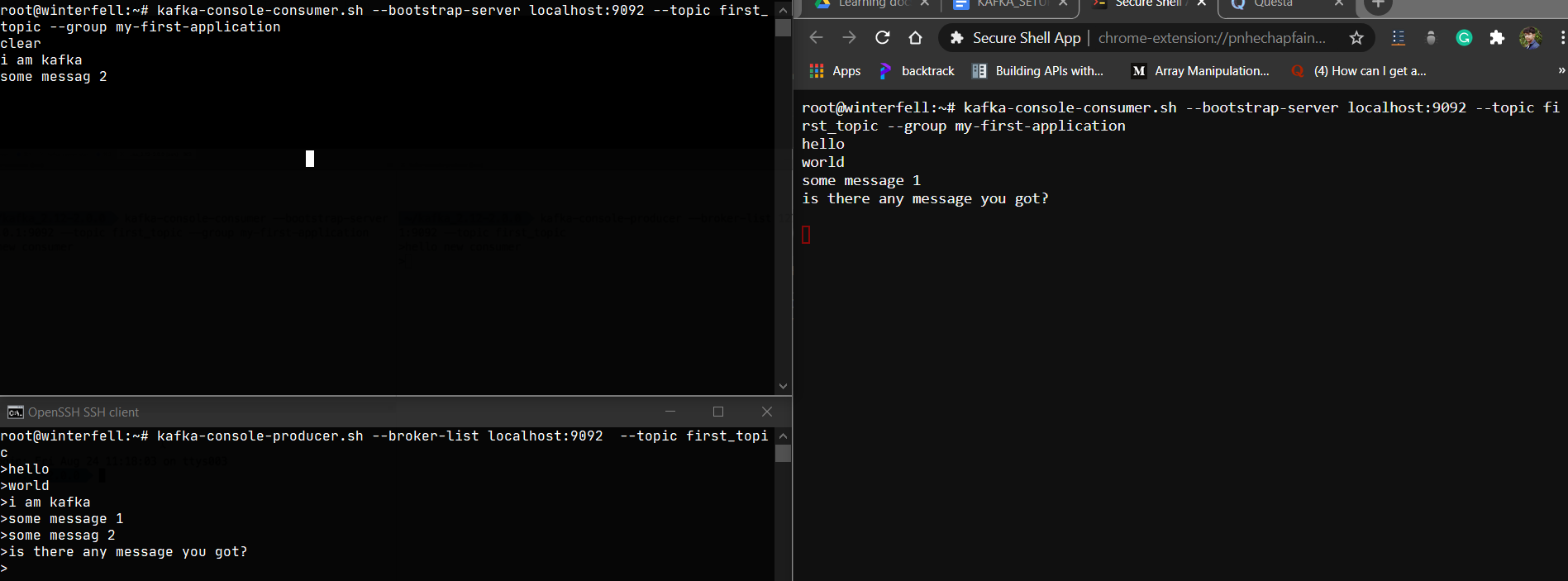
* Now, what if we want to retrieve all the message from the beginning of the topic. For that, in consumer command, we’ve to pass the additional argument as --from-beginning.
* So, the command is...

>> kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic first\_topic --from-beginning

* It’ll show us all the messages from the beginning and will wait for new messages from first\_topic.

**Consumer grouping**

* We can group the consumers, so kafka will load balance for the messages to all of the topics from same group.
* Ex. for there is a group named first\_group, we’re running multiple instances of terminal and executing the consumers as a group, and then producing the message from producer, in this case, the messages will e splitted to the instances og the group. Take a look at below screenshot.



* In above SS, top left and right side windows are of consumers, they are running with same group named my-first-application.
* And in left bottom window, we’re passing the messages and as we can see, the messages are splitted with proper load balancing between multiple instance of same grouped topics. Now, if we run another one instance of the same group, it’ll start splitting messages for 3 instances, and if i remove or stop one instance, then it will rebalance the sharing load of the messages accordingly.
* In the grouping part, the if we consume some messages once, and stop that consumer, and run it again, it’ll not show us the messages that it consumed earlier, because kafka offset keep tracking of the messages for grouped topics, (i.e. message no. 2 is consumed for this group and so on…).
* The command for consumer group is..

>> kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic first\_topic --group my-first-application

* In above command, we’re creating group as my-first-application. Try that command in multiple terminals and produce messages from producer.
* Experiment :
* Open a terminal, run a producer and publish some message for first\_topic.
* Open another terminal, run consumer for first\_topic with new group as my-second-application and --from-beginning.
* Now, you’ll get all the messages that you published earlier, and now, if you produce messages from producer terminal, you’ll retrieve them in consumer window.
* Now, stop the consumer.and run it again without --from-beginning, and publish some messages, you’ll get them in the consumer window.
* Now, stop the consumer again, and publish some more messages in producer window, and then run the consumer again without --from-beginning.
* Here, we can notice that, if we use consumer groups, then in the last step, it retrieved only the new messages, the messages it consumed earlier are not retrieved. The kafka offsets keep tract of what messages are consumed and what are not.