**Kafka + Spark Streaming**

This is an example of building a Proof-of-concept for Kafka + Spark streaming from scratch. This is meant to be a resource for video tutorial I made, so it won't go into extreme detail on certain steps. It can still be used as a follow-along tutorial if you like.

Also, this isn't meant to explain the design of Kafka/Hadoop, instead it's an actual hands-on example. I'd recommend learning the basics of these technologies before jumping in.

When considering this POC, I thought Twitter would be a great source of streamed data, plus it would be easy to peform simple transformations on the data. So for this example, we will

* create a stream of tweets that will be sent to a Kafka queue
* pull the tweets from the Kafka cluster
* calculate the character count and word count for each tweet
* save this data to a Hive table

To do this, we are going to set up an environment that includes

* a single-node Kafka cluster
* a single-node Hadoop cluster
* Hive and Spark

**1. VM Setup**

Although you might be able to follow along on your host computer (depending on your OS), I'd recommend setting up a virtual machine.

I used a [Lubuntu 19.04](https://lubuntu.net/downloads/) VM on [VirtualBox](https://www.virtualbox.org/wiki/Downloads) and gave it 4 GiB memory and 15 GiB storage, but feel free to use your favorite Linux distro.

You will need internet access from this VM so make sure that is configured correctly. If this is your first time using VBox, you might also want to look into installing the guest additions that allow full screen use.

**2. Install Kafka**

"Installing" Kafka is done by downloading the code from one of the several mirrors. After finding the latest binaries from [the downloads page](https://kafka.apache.org/downloads), choose one of the mirror sites and wget it into your home directory.

~$ wget http://apache.claz.org/kafka/2.2.0/kafka\_2.12-2.2.0.tgz

After that you will need to unpack it. At this point, I also like to rename the Kafka to something a little more concise.

~$ tar -xvf kafka\_2.12-2.2.0.tgz

~$ mv kafka\_2.12-2.2.0.tgz kafka

Before continuing with Kafka, we'll need to install Java.

~$ sudo apt install openjdk-8-jdk -y

Test the Java installation by checking the version.

~$ java -version

Now you can cd into the kafka/ directory and start a Zookeeper instance, create a Kafka broker, and publish/subscribe to topics. You can get a feel for this by walking thru the [Kafka Quickstart](https://kafka.apache.org/quickstart), but it will also be covered later in this example.

While we are here, we should install a Python package that will allow us to connect to our Kafka cluster. But first, you'll need to make sure that you have Python 3 and pip installed on your system. For Lubuntu, Python 3 was already installed and accessible with python3, but I had to install pip3 with

~$ pip3 install kakfa-python

Confirm this installation with

~$ pip3 list | grep kafka

**3. Install Hadoop**

Similar to Kafka, we first need to download the binaries from a mirror. All releases can be found [here](https://hadoop.apache.org/releases.html). I used Hadoop 2.8.5. Select a mirror, download it, then unpack it to your home directory.

* NOTE: I wrote out this step as using wget, but feel free to download the tar thru a browser (sometimes it is faster i think)
* NOTE #2: Again, I renamed my directory for convenience.

~$ wget https://archive.apache.org/dist/hadoop/common/hadoop-2.8.5/hadoop-2.8.5.tar.gz

~$ tar -xvf hadoop-2.8.5.tar.gz

~$ mv hadoop-2.8.5.tar.gz hadoop

~$ cd hadoop

~/hadoop$ pwd

/home/<USER>/hadoop

Edit .bashrc found in your home directory, and add the following.

export HADOOP\_HOME=/home/<USER>/hadoop

export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export HADOOP\_INSTALL=$HADOOP\_HOME

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export YARN\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export PATH=$PATH:$HADOOP\_HOME/sbin:$HADOOP\_HOME/bin

export HADOOP\_OPTS="-Djava.library.path=$HADOOP\_HOME/lib/native"

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

Be sure to insert your username on the first line, for example, for me it is export HADOOP\_HOME=/home/davis/hadoop. Also, double check that your $JAVA\_HOME is correct. This should be the default install location, but it may go somewhere else.

Also also, remember to execute your .bashrc after editing it so that the changes take place (this will be important later on)

~$ source .bashrc

Next, we will need to edit/add some configuration files. From the Hadoop home folder (the one named hadoop that is in your home directory), cd into etc/hadoop.

Add the following to hadoop-env.sh

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

export HADOOP\_CONF\_DIR=${HADOOP\_CONF\_DIR:-"/home/<USER>/hadoop/etc/hadoop"}

Replace the file core-site.xml with the following:

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:9000</value>

</property>

</configuration>

Replcae the file hdfs-site.xml with the following:

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.permission</name>

<value>false</value>

</property>

</configuration>

As mentioned earlier, we are just setting up a single-node Hadoop cluster. This isn'y very realistic, but it works for this example. For this to work, we need to allow our machine to SSH into itself.

First, install SSH with

~$ sudo apt install openssh-server openssh-client -y

Then, set up password-less authentication

~$ ssh-keygen -t rsa

~$ cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

Then, try SSH-ing into the machine (type exit to quit the SSH session and return)

~$ ssh localhost

With Hadoop configured and SSH setup, we can start the Hadoop cluster and test the installation.

~$ hdfs namenode -format

~$ start-dfs.sh

NOTE: These commands should be available anywhere since we added them to the PATH during configuration. If you're having troubles, hdfs and start-dfs are located in hadoop/bin and hadoop/sbin, respectively.

Finally, test that Hadoop was correctly installed by checking the Hadoop Distributed File System (HDFS):

~$ hadoop fs -ls /

If this command doesn't return an error, then we can continue!

**4. Install Hive**

Different releases of Hive can be found [here](https://hive.apache.org/downloads.html), or from the the [Apache Archive](http://archive.apache.org/dist/hive/). I downloaded used Hive 2.3.5 for this example. Be sure to download the binaries, rather than the source.

~$ wget http://archive.apache.org/dist/hive/hive-2.3.5/apache-hive-2.3.5-bin.tar.gz

~$ tar -xvf apache-hive-2.3.5-bin.tar.gz

~$ mv apache-hive-2.3.5-bin.tar.gz hive

Add the following to your .bashrc and run it with source

export HIVE\_HOME=/home/<USER>/hive

export PATH=$PATH:$HIVE\_HOME/bin

Give it a quick test with

~$ hive --version

Add the following directories and permissions to HDFS

~$ hadoop fs -mkdir -p /user/hive/warehouse

~$ hadoop fs -mkdir -p /tmp

~$ hadoop fs -chmod g+w /user/hive/warehouse

~$ hadoop fs -chmod g+w /tmp

Inside ~/hive/conf/, create/edit hive-env.sh and add the following

export HADOOP\_HOME=/home/<USER>/hadoop

export HADOOP\_HEAPSIZE=512

export HIVE\_CONF\_DIR=/home/<USER>/hive/conf

While still in ~/hive/conf, create/edit hive-site.xml and add the following

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:derby:;databaseName=/home/davis/hive/metastore\_db;create=true</value>

<description>JDBC connect string for a JDBC metastore.</description>

</property>

<property>

<name>hive.metastore.warehouse.dir</name>

<value>/user/hive/warehouse</value>

<description>location of default database for the warehouse</description>

</property>

<property>

<name>hive.metastore.uris</name>

<value>thrift://localhost:9083</value>

<description>Thrift URI for the remote metastore.</description>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>org.apache.derby.jdbc.EmbeddedDriver</value>

<description>Driver class name for a JDBC metastore</description>

</property>

<property>

<name>javax.jdo.PersistenceManagerFactoryClass</name>

<value>org.datanucleus.api.jdo.JDOPersistenceManagerFactory</value>

<description>class implementing the jdo persistence</description>

</property>

<property>

<name>hive.server2.enable.doAs</name>

<value>false</value>

</property>

</configuration>

(optional) Since Hive and Kafka are running on the same system, you'll get a warning message about some SLF4J logging file. From your Hive home you can just rename the file

~/hive$ mv lib/log4j-slf4j-impl-2.6.2.jar lib/log4j-slf4j-impl-2.6.2.jar.bak

Now we need to create a database schema for Hive to work with using schematool

~$ schematool -initSchema -dbType derby

We are now ready to enter the Hive shell and create the database for holding tweets. First, we need to start the Hive Metastore server with the following command.

~$ hive --services metastore

This should give some output that indicates that the metastore server is running. You'll need to keep this running, so open up a new terminal tab to continue with the next steps.

Now, enter the Hive shell with the hive command

~$ hive

...

hive>

Make the database for storing our Twitter data:

hive> CREATE TABLE tweets (text STRING, words INT, length INT)

> ROW FORMAT DELIMITED FIELDS TERMINATED BY '\\|'

> STORED AS TEXTFILE;

You can use SHOW TABLES; to double check that the table was created.

**5. Install Spark**

Download from <https://spark.apache.org/downloads.html>, make sure you choose the option for Hadoop 2.7 or later (unless you used and earlier version).

Unpack it, rename it

~$ tar -xvf Downloads/spark-2.4.3-bin-hadoop2.7.tgz

~$ mv spark-2.4.3-bin-hadoop2.7.tgz spark

Although I have been able to run Spark before without installing Scala, we can avoid some issues by ensuring Scala is installed on our system.

~$ sudo apt install scala -y

Test with

~$ scala -version

Instead of writing Scala code, we will write our Spark transformer in Python, so we will need pyspark.

~$ pip3 install pyspark

Check with

~$ pip3 list | grep spark

Now we need to add the Spark /bin files to the path, so open up .bashrc and add the following

export PATH=$PATH:/home/<USER>/spark/bin

export PYSPARK\_PYTHON=python3

By setting the PYSPARK\_PYTHON variable, we can use PySpark with Python3, the version of Python we have been using so far.

After running source .bashrc, try entering the PySpark shell

~$ pyspark

...

Using Python version ....

SparkSession available as 'spark'.

>>>

One last thing! We need a JAR file that wasn't included in PySpark that will allow us to connect to Kafka. Download the JAR file with the artifact ID spark-streaming-kafka-0-8-assembly\_2.11 from [search.maven.org](https://search.maven.org/search?q=a:spark-streaming-kafka-0-8-assembly_2.11).

**6. Write the Code**

Alright, we are ready to "write" the code for the stream producer and the stream consumer! I already have the code written, so all you need to do is copy and paste. I do a brief walkthough of the code in the video.

**Stream Producer**

As mentioned previously, we are going to create a stream of Tweets. This requires a set of API keys available from a Twitter developer account. This is free, but often takes time to get approved. The process of optaining API keys is pretty simply, plus I'm sure there are resources out there if you get stuck.

* **OPTION 1**

If you have Twitter API keys, open a new Python file on your VM with whatever name you like, and paste in the contents of tweet\_stream.py found in the files/ folder of this repo.

You'll need to add your API keys on lines 35-39, as well as install the Twitter Python package with pip3 install tweepy

* **OPTION 2**

If you don't have Twitter API keys, you can simulate "fake tweets". Open a new Python file on your VM and give it a name, then paste in the contents of fake\_tweet\_stream.py from the files/ folder of this repo.

Depending on your Linux distro, you should have a "wordlist" located somewhere in the /usr/share directory. My distro had one located in /usr/share/dict/words. Edit line 11 of this file to match your distro.

**Stream Consumer + Spark Transformer**

Open a new Python file on your VM and give it a name. Paste in the contents of transformer.py found in the files/ folder of this repo.

**7. Run the Example**

Now we are ready to run the example. Since we will be running multiple processes at once, we will have several terminals open at once. If your terminal emulator supports tabs, I recommend using that, but seperate terminal windows also works fine.

* **Terminal 1 -- Zookeeper**

In your first terminal window/tab, we need to start a Zookeeper instance. This is required for running a Kafka cluster

~$ cd kafka/

~/kafka$ bin/zookeeper-server-start.sh config/zookeeper.properties

This will write a lot of output to the console. As long as you don't hit an error and are returned to the prompt, it should be working. Keep this process running, and open a new terminal window/tab.

* **Terminal 2 -- Kafka Broker**

Now we need to start the Kafka broker.

~$ cd kafka/

~/kafka$ bin/kafka-server-start.sh config/server.properties

This will also write a lot of output to the console. If you didn't hit an error, keep this server running and open a new terminal.

* **Terminal 3 -- Kafka Config**

Now that we have our Kafka cluster running, we need to create the topic that will hold our tweets.

First check if any topics exist on the broker (this should return nothing if you just started the server)

~/kafka$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092

Now make a new topic named tweets

~/kafka$ bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic tweets

Run the --list command from above to make sure the tweets topic was successfully created.

You can either close this terminal, or just keep it open and use it for the next step.

* **Terminal 4 -- Hive Metastore**

~$ hive --services metastore

There shouldn't be much output besides a message that says the metastore server has started.

Keep this terminal running and open up a new one.

* **Terminal 5 -- Hive**

In order to make sure the transformed data is being stored in Hive, we need to enter the Hive shell so that we can query the tables. Write a query to count the records in the tweets table. This should return nothing, but as we start the stream producer / consumer, we can use the up-arrow to run this query again to check if the data is coming through.

~$ hive

...

hive> use default;

hive> select count(\*) from tweets;

Keep this one running and open up a new terminal.

* **Terminal 6 -- Stream Producer**

In this new terminal, we are going to run the stream producer. Mine is named tweet\_stream.py, but just use whatever you named yours.

If want to change the file permissions with chmod 755 tweet\_stream.py, you can run the stream producer with a simple ./

~$ ./tweet\_stream.py

If not, just run it with python3

~$ python3 tweet\_stream.py

This script should produce output to the console everytime a tweet is sent to the Kafka cluster, so you should be able to know whether or not the stream producer is working. Keep this running and open up a new terminal.

* **Terminal 7 -- Stream Consumer + Spark Transformer**

Now we are ready to run the consumer. Since we want to run it as a Spark job, we'll use spark-submit. NOTE: I moved my extra JAR to the home directory, but if yours is in a different location, you'll need to provide the correct path.

~$ spark-submit --jars spark-streaming-kafka-0-8-assembly\_2.11-2.4.3.jar transformer.py

This should produce a lot of logging output. Keep this running as long as you want the example to be running.

* **Did It Work?**

If you were able to run the producer script and the spark transformer, things should be working correctly! You should be able to see small dataframes being printed to the console in the terminal where the spark transformer is running.

Next, go back to "terminal #5 (Hive shell), and run the select count(\*) to see if the data is being written to Hive. If you get something greater than zero, it's working! You can investigate further with different queries.