INSA Lyon - Département Télécommunications
By exchange student Jes Rydall Larsen
Supervisor: Frédéric Le Mouël

Kafka and Storm on Raspberry Pi

21st December 2018

Introduction

This step-by-step guide will describe how to set up two Raspberry Pi's to run two similar WordCount examples with Apache Kafka as well as with Apache Storm.

The goal of the guide is to:

1. Set up everything as quickly as possible. No details of how anything is working will be included in the guide.
2. Give the commands to run the basic examples.

What hardware do you need to start?

- (At least) 2 x Raspberry pi model 3, which is already running Raspbian
- A laptop (this guide will assume you run Linux, but Windows should work just as well)
    - Some basic knowledge of how to use the Linux terminal
- An external monitor might be of help

How to read and use the guide

- You can jump over any step in the guide. However the guide will not go through basic Linux commands twice. If you are able to create and remove files & folders, move files, rename files, change directory, unpack folders and use root user, then this should not a problem.
- < > brackets should always be replaced by the information you need to use. This includes removing the brackets.

Table of content

Setup of Raspberry Pi to your WiFI connection

- Take out the MicroSD card, and insert in a computer.
- Ssh is disabled in newer versions of Raspbian for safety reasons. Make a file just called 'ssh' and nothing else in the Boot directory. When the Raspberry Pi reboots next time, it will enable ssh and delete the file.
- Open the file /etc/wpa_supplicant/wpa_supplicant.conf as administrator in the Rootfs directy and insert the following information with wifiname-ssd and password corrected in the bottom of the file.

```
network=
{
ssid="wifiname-ssd"
psk="password"
}
```

- Move the Micro SD card back into the Raspberry Pi. It will now automatically connect to WiFi when turned on.

How to connect with SSH to the Raspberry pi

You need to know the IP adress of the Raspberry Pi. There is two ways to do this.

1) You have an external monitor

- Connect the Raspberry Pi to the monitor. It's IP address should be stated as one of the last lines after booting, if it connected successfully to WiFi.
- Open a terminal and write the following with the given IP address:

```
user@machine:~$ ssh pi@<ip-address here>
```

    - An alternative is to use a tool like PuTTY to connect via SSH. X11 needs to be enabled to do so (SSH→X11→check box).

2) You do not have an external monitor

- Download a tool to search the network for active IP addresses. A recommended tool could be Angry IP Scanner. Use it to search the network, and you should be able to see the Raspberry Pi's as alive hosts if they have successfully connected to the WiFi.
- Open a terminal and write the following with the given IP address from the scan:

```
user@machine:~$ ssh pi@<ip-address here>
```

    - An alternative is to use a tool like PuTTY to connect via SSH. X11 needs to be enabled to do so (SSH→X11→check box enable).

Change of hostname

It can be a good idea to change the hostname since two Raspberry's will be running.

- Go two folders up from the home folder and go to etc folder (cd ../../etc)
- Change the name in the following two files (files can be saved with nano by pressing Ctrl + X, then Y followed by Enter)

```
pi@pi:/etc $ sudo nano hostname
pi@pi:/etc $ sudo nano hosts
```

    - In hosts: find the line starting with 127.0.0.1, and change it to your new hostname.
    - The change will take effect after the next reboot.

Setup of Maven

Create a new folder in the home directory called ‘opt’ with mkdir. Download and unzip Maven to this folder. It needs to be added to the system profile settings where the shell can find Maven. Sudoedit the file etc/profile.d/maven.sh and add the following text (etc folder is two up from home directory).

```
export M2_HOME=/home/pi/opt/maven
export "PATH=$PATH:$M2_HOME/bin"
```

Setup of Zookeeper

Zookeeper will be used for Kafka as well as for Storm.

You can either take out the SD cards and download the latest version of Zookeeper to the card. Or you can use the following command:

```
sudo wget <download link here>
```

Unpack the file with:

```
sudo tar -xzf <foldername>
```

The folder can be renamed with:

```
sudo mv zookeeperFoldername12345-678-9 newZookeeperFoldername
```

Setup of Zookeeper settings

From home directory go to folder ../../var. Create a folder with mkdir called zookeeper. In this folder you need to create a new file called ‘myid’. This folder should contain the id-number of the node. So one Raspberry should have a myid file containing ‘1’ and the other ‘2’. Nothing else!

In the zookeeper folder modify the conf/zoo.cfg file. Add the following settings:

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/var/zookeeper
clientPort=2181
server.1=<ip>:2888:3888
server.2=<ip>:2888:3888
```

The ip should be ’0.0.0.0’ for the node/Raspberry itself. The lines follow this form: server.myid=host:port:port.

Zookeeper commands

To start Zookeeper. Be aware. It will give no warnings if problems occur:

```
sudo ./bin/zkServer.sh start
```

To check if it started successfully (start zookeeper on all nodes before doing this):

```
sudo ./bin/zkServer.sh status
```

To stop zookeeper:

```
sudo ./bin/zkServer.sh stop
```

In case of debugging:

```
sudo /bin/zkServer.sh start-foreground
```

Apache Kafka

Download a binary version of Kafka to the Raspberry's with wget or directly to the SD card.

Set up of Kafka settings

In the Kafka folder, modify the conf/server.properties file, so it include the following:

```
broker.id=<myid> # the id of the node. '1' or '2' in our case.
port=9092
host.name=<ip-address of node>
zookeeper.connect=<ip-address for node with myid 1>:2181,<ip-address for
node with myid 2>:2181

listeners=PLAINTEXT://:9092
log.dirs=/tmp/kafka-logs-2
delete.topic.enable=true
```

The bin/kafka-server-start.sh file also needs to be modified with the following settings:

```
export JMX_PORT=${JMX_PORT:-9999}
export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M" # Otherwise, JVM would complain
not able to allocate the specified memory.
```

bin/kafka-run-class.sh with the following:

```
KAFKA_JVM_PERFORMANCE_OPTS="-client -XX:+UseParNewGC
-XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled
-XX:+CMSScavengeBeforeRemark -XX:+DisableExplicitGC
-Djava.awt.headless=true" # change -server to -client
```

Commands

Run Kafka:

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

Create a topic named ‘streams-plaintext-input’:

```
bin/kafka-topics.sh --create \
    --zookeeper localhost:2181 \
    --replication-factor 1 \
    --partitions 1 \
    --topic streams-plaintext-input
Created topic "streams-plaintext-input"
```

List all topics. Change ‘list’ with ‘describe’ for descriptions of each topic.

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

Create a producer on topic ‘streams-plaintext-input’:

```
sudo bin/kafka-console-producer.sh --broker-list localhost:9092 --topic
streams-plaintext-input
```

Create a basic consumer on topic ‘streams-wordcount-output’, that does *not* work with the WordCount example:

```
sudo bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 \
    --topic streams-wordcount-output
```

Create a consumer that will work with the WordCount example:

```
sudo bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 \
    --topic streams-wordcount-output \
    --from-beginning \
    --formatter kafka.tools.DefaultMessageFormatter \
    --property print.key=true \
    --property print.value=true \
    --property
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
    --property
value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
```

How to set up the WordCount example

Set up a new project structure using Maven:

```
sudo mvn archetype:generate \
    -DarchetypeGroupId=org.apache.kafka \
    -DarchetypeArtifactId=streams-quickstart-java \
    -DarchetypeVersion=1.1.0 \
    -DgroupId=streams.examples \
    -DartifactId=streams.examples \
    -Dversion=0.1 \
    -Dpackage=myapps
```

A new folder ‘streams.examples’ should appear where you ran the command. It includes a lot of examples, but these are not needed for now. Go into the streams.examples folder and remove all examples with:

```
sudo rm src/main/java/myapps/*.java
```

Now add a new file WordCount.java in the folder streams.examples/src/main/java with the following code:

```
package myapps;

import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.Topology;
import org.apache.kafka.streams.kstream.KStream;
```

```
import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Materialized;
import org.apache.kafka.streams.kstream.Produced;

import java.util.Arrays;
import java.util.Locale;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;

// For in-memory key-value store
import org.apache.kafka.streams.state.StoreBuilder;
import org.apache.kafka.streams.state.Stores;

public class WordCount {

      public static void main(String[] args) throws Exception {
      Properties props = new Properties();
      props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-wordcount");
      props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
      props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
      props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.String().getClass());

      final StreamsBuilder builder = new StreamsBuilder();

KStream<String, String> textLines =
builder.stream("streams-plaintext-input");
      KTable<String, Long> wordCounts = textLines
            .flatMapValues(textLine ->
Arrays.asList(textLine.toLowerCase().split("\\W+")))
            .groupBy((key, word) -> word)
            .count(Materialized.<String,
Long>as(Stores.inMemoryKeyValueStore("myStore")));
      wordCounts.toStream().to("streams-wordcount-output",
Produced.with(Serdes.String(), Serdes.Long()));

      final Topology topology = builder.build();
      final KafkaStreams streams = new KafkaStreams(topology, props);
      final CountDownLatch latch = new CountDownLatch(1);

      Runtime.getRuntime().addShutdownHook(new
Thread("streams-shutdown-hook") {
            @Override
```

```
            public void run() {
                  streams.close();
                  latch.countDown();
            }
      });

      try {
            streams.start();
            latch.await();
      } catch (Throwable e) {
            System.exit(1);
      }
      System.exit(0);
      }
}
```

The example WordCount from the official Kafka Streams Quick start guide will not be able to run on the ARM processor of the Raspberry Pi. The example uses a memory store relying on RocksDB to store strings and counts, but RocksDB does not support ARM. The above code instead uses an in-memory key-values store. Some packages has been imported and a ‘)’ has also been added, since the official code cannot compile at the moment without these corrections.

How to compile and run the WordCount example

Open up a consumer on the topic ‘streams-plaintext-input’ and a producer on the topic ‘streams-wordcount-output’.

Run the following command from the streams.examples folder to compile the code:

```
sudo mvn clean package
```

And then run the following to execute the project:

```
sudo mvn exec:java -Dexec.mainClass=myapps.WordCount
```

Extra: How to set up an automatic producer for the WordCount example

Download the code with:

```
git clone https://github.com/sdpatil/KafkaAPIClient.git
```

Update the following dependency in the pom.xml file.

```
      <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>1.1.0</version> <!-- Updated from 0.9.0.0 to 1.1.0 -->
      </dependency>
```

This is due to an internal timestamp bug for producers, which is fixed in the newer version.

Change the Producer.java from folder KafkaAPIClient/src/main/java/com/spnotes/kafka/simple with the following code. That will make the producer produce and send 100.000 strings to topic ‘streams-plaintext-input’ and then shut down.

```
package com.spnotes.kafka.simple;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.util.Properties;
import java.util.Scanner;

// Time stamp
import java.sql.Timestamp;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Created by sunilpatil on 12/28/15.
 * Additions by Jes
 */

public class Producer {
      private static Scanner in;
      public static void main(String[] argv)throws Exception {
      if (argv.length != 1) {
            System.err.println("Please specify 1 parameters ");
            System.exit(-1);
      }
      String topicName = "streams-plaintext-input";
      in = new Scanner(System.in);
      System.out.println("Enter message(type exit to quit)");

      //Configure the Producer
```

```
      Properties configProperties = new Properties();

configProperties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"localhost:909
2");

configProperties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache
.kafka.common.serialization.ByteArraySerializer");

configProperties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apac
he.kafka.common.serialization.StringSerializer");

      org.apache.kafka.clients.producer.Producer producer = new
KafkaProducer(configProperties);
      //String line = in.nextLine();
      String line = "Producer is working and counter is counting";

            ProducerRecord<String, String> rec = new ProducerRecord<String,
String>(topicName,line);
            String timestamp = new SimpleDateFormat("SSSSSS").format(new
Timestamp(System.currentTimeMillis()));
            ProducerRecord<String, String> timestampToSend = new
ProducerRecord<String, String>(topicName,timestamp);
            producer.send(timestampToSend);

      while(!line.equals("exit")) {
            for(int i = 0; i < 100000; i++){
            producer.send(rec);
                  if( !(line.equals("Producer is working and counter is
counting")) ) {
                  break;
                  }
            }
            String timestamp2 = new SimpleDateFormat("SSSSSS").format(new
Timestamp(System.currentTimeMillis()));
            ProducerRecord<String, String> timestampToSend2 = new
ProducerRecord<String, String>(topicName,timestamp2);
            producer.send(timestampToSend2);
            //line = in.nextLine();
            break;
      }

      in.close();
      producer.close();
      }
```

```
}
```

Open a consumer, the WordCount and finally the Producer.

To compile the producer, use:

```
sudo mvn clean compile assembly:single
```

To run the producer, use:

```
sudo java -cp target/KafkaAPIClient-1.0-SNAPSHOT-jar-with-dependencies.jar
com.spnotes.kafka.simple.Producer test
```

Reference for producer:
https://www.javaworld.com/article/3060078/big-data/big-data-messaging-with-kafka-part-1.html

Apache Storm

Download a binary version of Storm to the Raspberry's with wget or directly to the SD card.

Set up of the settings

Update the conf/storm.yaml file with the following settings:

```
storm.zookeeper.servers:
 - "localhost"
storm.local.dir: "</path/to/storm/data(any path)>"
nimbus.host: "localhost"
supervisor.slots.ports:
 - 6700
 - 6701
 - 6702
 - 6703
```

Commands

Start Storm:

```
sudo bin/storm start
```

Start Storm nimbus:

```
sudo bin/storm nimbus
```

Start Storm supervisor:

```
sudo bin/storm supervisor
```

Start Storm UI, which can be accessed at http://<nimbus ip-address>/index.html on any computer connected to the same network:

```
sudo bin/storm ui
```

Kill a topology. Be aware that the topology created with the WordCount example is called 'production-topology'.

```
sudo bin/storm kill <topology-name>
```

How to set up the WordCount example

The example is pre-installed with Storm. You can find it in the folder examples/storm-starter.

How to compile and run the WordCount example

Start a nimbus, supervisor and the UI.

Run the following command from the folder examples/storm-starter.

```
sudo mvn clean install -DskipTests=true
```

Now run the following command to compile and execute the project from the top directory folder of Storm.

```
sudo bin/storm jar examples/storm-starter/target/storm-starter-*.jar
org.apache.storm.starter.WordCountTopology production-topology remote
```

How to close down running terminal tasks and shutdown the Raspberry Pi's

Ctrl + C will close down tasks like producers, consumers and running servers in the terminal.

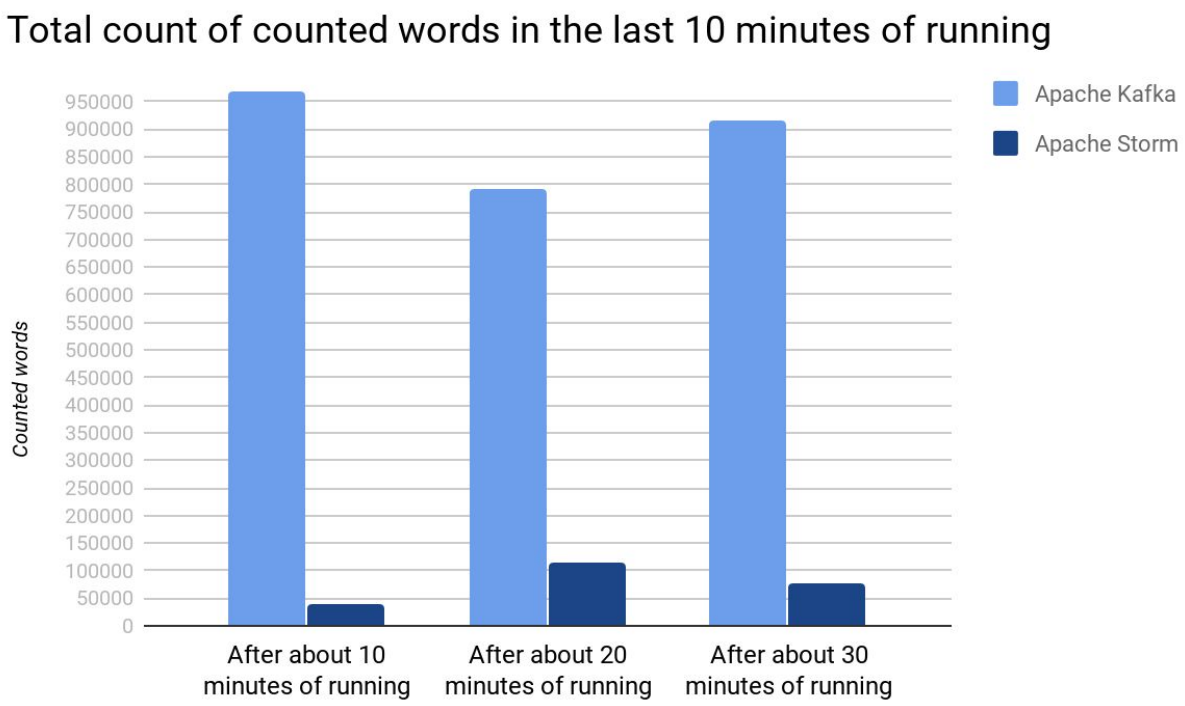
You can kill a Storm topology in the UI.

Use the following command to safely shutdown a Raspberry Pi:

```
sudo shutdown -h now
```

Benchmark

Word counting is compared between Kafka and Storm after 10 minutes of running.



Sources of error

The Storm UI is used to extract the number of processed and counted words in the last 10 minutes of use. The word count is precise. But the 20 and 30 minutes mark was measured using a stopwatch.

The 10 minutes of runtime in Kafka is simply measured using a stop timer. The consumer, WordCount and Producer is stopped *manually* 10, 20 and 30 minutes after being started, and therefore the final word count is more imprecise.

Both Raspberry Pi's used for the test were running for at least an hour before the test. But they have different cooling. The Raspberry Pi for the nimbus in Storm and for the follower in Kafka uses a more powerful fan than the other machine, which was used as supervisor in Storm and leader in Kafka.

References

The following guides and webpages has been used to create this guide.

https://medium.com/@oliver_hu/set-up-a-kafka-cluster-with-raspberry-pi-2859005a9bed

https://kafka.apache.org/11/documentation/streams/

https://www.javaworld.com/article/3060078/big-data/big-data-messaging-with-kafka-part-1.html

https://github.com/apache/storm/blob/master/examples/storm-starter/README.markdown