31: Docker Tutorial: Apache Spark streaming in Scala with Apache Kafka on Cloudera quickstart

Posted on July 8, 2019

This extends 27: Docker Tutorial: Apache Kafka with Java API on Cloudera quickstart.

Pre-requisite: Java 8 & Kafka installed with the "**MyTestTopic**" as per the previous tutorials.

Install scala & sbt

300+ Java Interview FAQs

300+ Java FAQs 🔥

16+ Java Key Areas Q&As

150+ Java Architect FAQs

80+ Java Code Quality Q&As

150+ Java Coding Q&As

300+ Big Data Interview FAQs

300+ Big Data FAQs 🔥

Tutorials - Big Data

TUT - ⏯ Starting Big Data

TUT - Starting Spark & Scala

We will install Scala **2.10.7** from "https://www.scala-lang.org/download/2.10.7.html" , which uses Java 8.

Step 1: Install Scala via **curl**.

```
[root@quickstart /]# curl -O -L https://downloads.
```

Copy to "/opt" folder and untar.

```
[root@quickstart /]# cp scala-2.10.7.tgz /opt
[root@quickstart /]# cd /opt
[root@quickstart opt]# tar xfz scala-2.10.7.tgz
[root@quickstart opt]# rm -f scala-2.10.7.tgz
[root@quickstart opt]# ls -ltr
total 16
drwxr-xr-x 5 cloudera     cloudera     4096 Aug  1
drwxr-xr-x 4 cloudera     cloudera     4096 Aug 27
drwxr-xr-x 4 cloudera-scm cloudera-scm 4096 Apr  6
drwxrwxr-x 9         1001         1001 4096 Nov  3
[root@quickstart opt]#
```

Step 2: Install sbt via **curl**.

Go to https://www.scala-lang.org/download/2.10.7.html and click on "**Download SBT**".

```
[root@quickstart opt]# curl -O -L https://piccolo.
```

Untar tha tgz file.

```
[root@quickstart opt]# tar xfz sbt-1.2.8.tgz
[root@quickstart opt]# rm -f sbt-1.2.8.tgz
[root@quickstart opt]# ls -ltr
total 20
drwxr-xr-x 5 cloudera     cloudera     4096 Aug  1
```

TUT - Starting with Python
TUT - Kafka
TUT - Pig
TUT - Apache Storm
TUT - Spark Scala on Zeppelin
TUT - Cloudera
TUT - Cloudera on Docker
TUT - File Formats
TUT - Spark on Docker
TUT - Flume
TUT - Hadoop (HDFS)
TUT - HBase (NoSQL)
TUT - Hive (SQL)
TUT - Hadoop & Spark
TUT - MapReduce
TUT - Spark and Scala
TUT - Spark & Java
TUT - PySpark on Databricks
TUT - Zookeeper

```
drwxr-xr-x 4 cloudera     cloudera     4096 Aug 2
drwxr-xr-x 4 cloudera-scm cloudera-scm 4096 Apr  
drwxrwxr-x 9         1001         1001 4096 Nov  
drwxrwxr-x 5         1000         1000 4096 Dec 3
[root@quickstart opt]#
```

Update ~/.bashrc

Step 3: If you are a root user update the **~/.bashrc** file, and if you are a any other user update the **~/.bash_profile** so that “scala” and “sbt” command can be run from any folder.

```
[root@quickstart opt]# vi ~/.bashrc
```

```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

export JAVA_HOME=/usr/lib/jvm/jre-1.8.0-openjdk.x
export PATH=$JAVA_HOME/bin:$PATH

SCALA_HOME=/opt/scala-2.10.7
SBT_HOME=/opt/sbt

export PATH=$PATH:$SCALA_HOME/bin:$SBT_HOME/bin
```

Activate:

```
[root@quickstart opt]# source ~/.bashrc
```

Check scala command prompt:

```
[root@quickstart opt]# scala
Welcome to Scala version 2.10.7 (OpenJDK 64-Bit Se
Type in expressions to have them evaluated.
Type :help for more information.

scala>

```

Check sbt command prompt:

```
[root@quickstart opt]# sbt
[info] Updated file /opt/project/build.properties:
[info] Loading project definition from /opt/projec
[info] Updating ProjectRef(uri("file:/opt/project/
[info] Done updating.
[info] Set current project to opt (in build file:/
[info] sbt server started at local:///root/.sbt/1.
sbt:opt>

```

Create Scala project structure

Step 4: Unlike maven archetype:generate, sbt does not create the basic project structure. We can create the sbt project structure with a shell script.

```
[root@quickstart ~]# cd ~
[root@quickstart ~]# pwd
/root
[root@quickstart ~]# mkdir projects
[root@quickstart ~]# cd projects/
[root@quickstart projects]# mkdir my-app
[root@quickstart projects]# cd my-app
[root@quickstart my-app]# vi mkdirs4sbt.sh

```

The "**mkdirs4sbt.sh**"

```
#!/bin/sh
```

```
mkdir -p src/{main,test}/{java,resources,scala}
mkdir lib project target

# create an initial build.sbt file
echo 'name := "my-app"
version := "1.0"
scalaVersion := "2.10.7"' > build.sbt

```

```
[root@quickstart my-app]# chmod 755 mkdirs4sbt.sh
[root@quickstart my-app]# ./mkdirs4sbt.sh

```

```
[root@quickstart my-app]# tree
.
├── build.sbt
├── lib
├── mkdirs4sbt.sh
├── project
├── src
│   ├── main
│   │   ├── java
│   │   ├── resources
│   │   └── scala
│   └── test
│       ├── java
│       ├── resources
│       └── scala
└── target

12 directories, 2 files
[root@quickstart my-app]#

```

Add spark dependency in build.sbt

Step 5: To write Spark code spark-core api library is required.

```
[root@quickstart my-app]# vi build.sbt

```

```
name := "my-app"
version := "1.0"
scalaVersion := "2.10.7"

libraryDependencies += "org.apache.spark" %% "sparl
libraryDependencies += "org.apache.spark" %% "sparl
```

```
7 libraryDependencies += "org.apache.spark" %% "sparl
8
```

Create the Spark job in Scala

Step 6: Create the package “com.mycompany.app”.

```
1 [root@quickstart my-app]# mkdir -p src/main/scala/
```

Step 7: Create “SimpleSparkStreaming.scala”.

```
1 [root@quickstart my-app]# vi src/main/scala/com/my
```

```
1  package com.mycompany.app
2
3  import kafka.serializer.StringDecoder
4  import kafka.serializer.DefaultDecoder
5  import org.apache.spark._
6  import org.apache.spark.streaming._
7  import org.apache.spark.streaming.kafka.KafkaUtil
8
9  object SimpleSparkStreaming {
10   def main(args: Array[String]) {
11     val conf = new SparkConf().setAppName("Simple
12     val ssc = new StreamingContext(conf, Seconds(
13
14     // zookeeper 2181, kafka 9092
15     val kafkaParams = Map[String, String]("metada
16     val kafkaTopics = Set("MyTestTopic")
17
18     val directKafkaStream = KafkaUtils.createDire
19       ssc, kafkaParams, kafkaTopics
20     )
21
22
23     directKafkaStream.foreachRDD { rdd =>
24       rdd.foreach { content =>
25           // code to handle the string here
26           println(content)
27       }
28
29     }
30
31     ssc.start()
32     ssc.awaitTermination()
33
34   }
```

```
35 }
36
37
```

Compile & Package with sbt

Step 8: Package it with “sbt”

```
1 [root@quickstart my-app]# sbt package
2 ....
3
```

```
1 [root@quickstart my-app]# ls -ltr target/scala-2.1
2 total 12
3 drwxr-xr-x 5 root root 4096 Jun  8 09:06 resolutio
4 drwxr-xr-x 3 root root 4096 Jun  8 09:06 classes
5 -rw-r--r-- 1 root root 2334 Jun  8 09:06 my-app_2.
6 [root@quickstart my-app]#
7
```

spark-submit to run the spark job

Step 9: Run the Spark job in the jar file via Spark-submit command.

Local client mode

```
1 [root@quickstart my-app]# spark-submit   \
2 --class com.mycompany.app.SimpleSparkStreaming  \
3 --master local \
4 --deploy-mode client \
5 target/scala-2.10/my-app_2.10-1.0.jar
6
```

Publish messages to Kafka topic MyTestTopic

Open a new terminal, and login to the running container:

```
$ docker ps

CONTAINER ID        IMAGE               COMMAND
1c781f94a1c2        gdancik/cloudera    "/usr/bin/
```

```
$ docker exec -it 1c781f94a1c2 /bin/bash

[root@quickstart /]#
```

publish messages to the topic:

```
[root@quickstart /]# /home/kafka/kafka/bin/kafka-c

```

```
>Sending a test message
>to the Kafka topic
>
```

Spark streaming console output

```
.....
(null,Sending a test message)
...................
(null,to the Kafka topic)

```

Where “null” is the key and “Sending a test message” is the value.

If you want to only print the values use “content._2”:

```
.....
   directKafkaStream.foreachRDD { rdd =>
       rdd.foreach { content =>
           // code to handle the string here
           println(content._2)
       }

   }

```

Word count example

Here is the Scala code:

```
1  package com.mycompany.app
2
3  import kafka.serializer.StringDecoder
4  import kafka.serializer.DefaultDecoder
5  import org.apache.spark._
6  import org.apache.spark.streaming._
7  import org.apache.spark.streaming.kafka.KafkaUtils
8
9  object SimpleSparkStreaming {
10   def main(args: Array[String]) {
11     val conf = new SparkConf().setAppName("Simple
12     val ssc = new StreamingContext(conf, Seconds(
13
14     // zookeeper 2181, kafka 9092
15     val kafkaParams = Map[String, String]("metada
16     val kafkaTopics = Set("MyTestTopic")
17
18     val directKafkaStream = KafkaUtils.createDire
19       ssc, kafkaParams, kafkaTopics
20     )
21
22
23     directKafkaStream.map(record => record._2)
24                      .flatMap(line => line.split(
25                      .map(word => (word,1))
26                      .reduceByKey(_+_)
27                      .print()
28
29
30
31     ssc.start()
32     ssc.awaitTermination()
33
34   }
35 }
36
```

Package it with sbt:

```
1 [root@quickstart my-app]# sbt package
2
```

Run the Spark job:

```
[root@quickstart my-app]# spark-submit   \
--class com.mycompany.app.SimpleSparkStreaming  \
--master local \
--deploy-mode client \
target/scala-2.10/my-app_2.10-1.0.jar

```

Publish a message to the topic from a different terminal as illustrated before:

```
>a big brown fox jumped over a brown fence

```

The output on the Spark streaming console will be:

```
-------------------------------------------
Time: 1562508765000 ms
-------------------------------------------
(fox,1)
(a,2)
(big,1)
(fence,1)
(over,1)
(brown,2)
(jumped,1)

```

‹ 30: Docker Tutorial: Apache Spark streaming in Python 3 with Apache Kafka on Cloudera quickstart

01: Getting started with Jenkins on Docker tutorial ›