

# USING SPARK WITH HIVE



Here we explain how to use Apache Spark with Hive. That means instead of Hive storing data in Hadoop it stores it in Spark. The reason people use Spark instead of Hadoop is it is an all-memory database. So Hive jobs will run much faster there. Plus it moves programmers toward using a common database if your company runs predominately Spark.

It is also possible to write programs in Spark and use those to connect to Hive data, i.e., go in the opposite direction. But that is not a very likely use case as if you are using Spark you already have bought into the notion of using RDDs (Spark in-memory storage) instead of Hadoop.

Anyway, we discuss the first option here.

*(This tutorial is part of our [Apache Spark Guide](#). Use the right-hand menu to navigate.)*

## Prerequisites and Installation

- You need to [install Hive](#).
- Install Apache Spark from source code (We explain below.) so that you can have a version of Spark without Hive jars already included with it.
- Set HIVE\_HOME and SPARK\_HOME accordingly.
- Install Hadoop. We do not use it except the Yarn resource scheduler is there and jar files. But Hadoop does not need to be running to use Spark with Hive. However, if you are running a Hive or Spark cluster then you can use Hadoop to distribute jar files to the worker nodes by copying them to the HDFS (Hadoop Distributed File System.)

The instructions here are for Spark 2.2.0 and Hive 2.3.0. Just swap the directory and jar file names

below to match the versions you are using. Note that when you go looking for the jar files in Spark there will in several cases be more than one copy. Use the ones in the **dist** folder as shown below.)

First you need to download Spark source code. Then you compile it like this:

```
./dev/make-distribution.sh --name "hadoop2-without-hive" --tgz "-  
Pyarn,hadoop-provided,hadoop-2.7,parquet-provided"
```

Next update `/usr/share/spark/spark-2.2.0/conf/spark-env.sh` and add:

```
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
```

## Link Jar Files

Now we make soft links to certain Spark jar files so that Hive can find them:

```
ln -s /usr/share/spark/spark-2.2.0/dist/jars/spark-network-  
common_2.11-2.2.0.jar /usr/local/hive/apache-hive-2.3.0-bin/lib/spark-  
network-common_2.11-2.2.0.jar  
ln -s /usr/share/spark/spark-2.2.0/dist/jars/spark-core_2.11-2.2.0.jar  
/usr/local/hive/apache-hive-2.3.0-bin/lib/spark-core_2.11-2.2.0.jar  
ln -s /usr/share/spark/spark-2.2.0/dist/jars/scala-library-2.11.8.jar  
/usr/local/hive/apache-hive-2.3.0-bin/lib/scala-library-2.11.8.jar
```

## Start Spark master and worker:

Now start Spark.

```
$SPARK_HOME/sbin/start-all.sh
```

Make a directory to contain log files:

```
mkdir /var/log/spark
```

Edit `$HIVE_HOME/conf/hive-site.xml`:

```
<property>  
<name>hive.execution.engine</name>  
<value>spark</value>  
</property>  
<property>  
<name>spark.master</name>  
<value>spark://(your IP address):7077</value>  
</property>  
<property>  
<name>spark.eventLog.enabled</name>  
<value>>true</value>  
</property>  
<property>  
<name>spark.eventLog.dir</name>  
<value>/var/log/spark</value>
```

```
</property>
<property>
<name>spark.executor.memory</name>
<value>2048m</value>
</property>
<property>
<name>spark.serializer</name>
<value>org.apache.spark.serializer.KryoSerializer</value>
</property>
```

## Run Hive

Now run Hive as shown below. We are running in **local mode** as opposed to using the cluster. Note that we tell Hive to log errors to the console so that we can see if anything goes wrong. Also note that we use `hive` and not `beeline`, the newer Hive CLI. Hive wants its users to use Beeline, but it is not necessary. (We wrote about how to use beeline [here](#).)

```
hive --hiveconf hive.root.logger=INFO,console
```

Edit `/usr/hadoop/hadoop-2.8.1/etc/hadoop/yarn-site.xml`.

```
<configuration>
<property>
<name>yarn.resourcemanager.scheduler.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.
FairScheduler</value>
</property>
</configuration>
```

## Create some Data

Now create a table and insert some data. You have to wait a couple of seconds after you type a command in order for it to run since it is using Spark and Yarn. Remember this is designed to run across a cluster.

```
create table students (student string, age int);
insert into table students values('Walker', 33);
```

The screen will look like this:

```

2017-08-30T11:30:05,083 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: Starting Spark Job = 7d08658c-6cb6-4f96-bc77-a068541cb3d7
2017-08-30T11:30:10,536 INFO [RPC-Handler-3] client.SparkClientImpl: Received spark job ID: 0 for 7d08658c-6cb6-4f96-bc77-a068541cb3d7

Query Hive on Spark job[0] stages: [0]

Status: Running (Hive on Spark job[0])
2017-08-30T11:30:10,892 INFO [main] SessionState:
Query Hive on Spark job[0] stages: [0]
2017-08-30T11:30:10,893 INFO [main] SessionState:
Status: Running (Hive on Spark job[0])
2017-08-30T11:30:10,893 INFO [main] SessionState: Job Progress Format
CurrentTime StageId StageAttemptId: SucceededTasksCount(+RunningTasksCount-FailedTasksCount)/TotalTasksCount
-----
STAGES ATTEMPT STATUS TOTAL COMPLETED RUNNING PENDING FAILED
-----
STAGES ATTEMPT STATUS TOTAL COMPLETED RUNNING PENDING FAILED
-----
Stage-0 ..... 0 FINISHED 1 1 0 0 0
-----
STAGES: 01/01 (=====>>>) 100% ELAPSED TIME: 8.16 s
-----
2017-08-30T11:30:15,943 INFO [main] SessionState: 2017-08-30 11:30:15,941 Stage-0_0: 1/1 Finished
Status: Finished successfully in 8.17 seconds
2017-08-30T11:30:15,945 INFO [main] SessionState: Status: Finished successfully in 8.17 seconds
2017-08-30T11:30:15,956 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: =====Spark Job[7d08658c-6cb6-4f96-bc77-a068541cb3d7] statist
ics=====
2017-08-30T11:30:15,956 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: HIVE
2017-08-30T11:30:15,956 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: CREATED FILES: 1
2017-08-30T11:30:15,957 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: RECORDS OUT: 1 default.students: 1
2017-08-30T11:30:15,957 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: DESERIALIZE ERRORS: 0
2017-08-30T11:30:15,957 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: RECORDS IN: 1
2017-08-30T11:30:15,957 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: Spark Job[7d08658c-6cb6-4f96-bc77-a068541cb3d7] Metrics
2017-08-30T11:30:15,957 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: ExecutorDeserializeTime: 1728
2017-08-30T11:30:15,958 INFO [425753e2-2620-4255-b928-2fade0248d65 main] spark.SparkTask: ExecutorRunTime: 2407

```

Since it takes some time to get the job started, you have time to open the Spark URL on port 8080 to see running programs. Spark removes those when the job completes unless you run the Spark Job History Server.

The screenshot shows the Spark REST API interface in a web browser. The URL is `77.235.46.77:8080/#`. The interface displays the following information:

- REST URL:** `spark://eurovps:6066 (cluster mode)`
- Alive Workers:** 0
- Cores in use:** 0 Total, 0 Used
- Memory in use:** 0.0 B Total, 0.0 B Used
- Applications:** 0 Running, 0 Completed
- Drivers:** 1 Running, 0 Completed
- Status:** ALIVE

**Workers**

Worker Id	Address	State	Cores	Memory
No workers are currently listed.				

**Running Applications**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
No running applications are listed.							

**Running Drivers**

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class
driver-20170830112018-0000	(kill) 2017/08/30 11:20:18	None	SUBMITTED	1	1024.0 MB	org.apache.hive.spark.client.RemoteDriver

**Completed Applications**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
No completed applications are listed.							

**Completed Drivers**

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class
No completed drivers are listed.						

Now you are done.