# Hadoop Smoke Tests

These are basic smoke tests to be used to determine basic functionality of the various parts of a Hadoop cluster. One might use these when setting up a new cluster or after a cluster upgrade.

* [Non-Secured Cluster](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Non-SecuredCluster)
  + [HDFS](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-HDFS)
  + [MapReduce](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-MapReduce)
  + [Hive](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Hive)
  + [HBase](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-HBase)
  + [Impala](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Impala)
  + [Spark](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Spark)
  + [Spark2](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Spark2)
  + [Pig](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Pig)
  + [Solr](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Solr)
  + [Kudu](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Kudu)
    - [Impala](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Impala.1)
  + [Kafka](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Kafka)
  + [Clean It Up](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-CleanItUp)
* [Secured Cluster](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-SecuredCluster)
  + [Preparation](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Preparation)
  + [HDFS](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-HDFS.1)
  + [MapReduce](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-MapReduce.1)
  + [Hive](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Hive.1)
  + [HBase](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-HBase.1)
  + [Impala](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Impala.2)
  + [Spark](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Spark.1)
  + [Spark2](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Spark2.1)
  + [Pig](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Pig.1)
  + [Solr](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Solr.1)
  + [Kafka](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Kafka.1)
  + [Clean It Up](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-CleanItUp.1)
* [Secured Cluster with Sentry](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-SecuredClusterwithSentry)
  + [Preparation](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Preparation.1)
  + [HDFS](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-HDFS.2)
  + [MapReduce](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-MapReduce.2)
  + [Hive](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Hive.2)
  + [HBase](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-HBase.2)
  + [Impala](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Impala.3)
  + [Spark](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Spark.2)
  + [Spark2](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Spark2.2)
  + [Pig](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Pig.2)
  + [Kafka](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-Kafka.2)
  + [Clean It Up](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/90472529/Hadoop+Smoke+Tests#HadoopSmokeTests-CleanItUp.2)

# Non-Secured Cluster

These examples assume a non-secured cluster and use of a non-cluster user (i.e. the user "centos").

## HDFS

Basic HDFS functionality.

hdfs dfs -ls /  
hdfs dfs -put /etc/hosts /tmp/hosts  
hdfs dfs -get /tmp/hosts /tmp/hosts123  
cat /tmp/hosts123

## MapReduce

Pi Estimator

yarn jar /opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 10 1000

## Hive

Create an external table and query it.

For Hive on MapReduce, add "set hive.execution.engine=mr;" to the query.

For Hive on Spark, add "set hive.execution.engine=spark;" to the query.

# Replace $HIVESERVER2 with the correct hostname that is running the HS2  
HIVESERVER2=`hostname`  
  
# Create hive table  
beeline -n `whoami` -u "jdbc:hive2://${HIVESERVER2}:10000/" -e "CREATE TABLE test(id INT, name STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ' STORED AS TEXTFILE;"  
  
# Create HDFS file  
echo "1 justin" >/tmp/hive.$$  
echo "2 michael" >>/tmp/hive.$$  
# You should probably be the hive user here.  
hdfs dfs -put /tmp/hive.$$ /user/hive/warehouse/test/hive  
  
# Query hive table  
beeline -n `whoami` -u "jdbc:hive2://${HIVESERVER2}:10000/" -e "SELECT \* FROM test WHERE id=1;"

## HBase

Create a table and query it.

cat <<EOF >/tmp/hbase.$$  
create 'test', 'cf'  
list 'test'  
put 'test', 'row1', 'cf:a', 'value1'  
scan 'test'  
exit  
EOF  
  
hbase shell -n /tmp/hbase.$$

## Impala

Query the hive table created earlier.

# Replace $IMPALAD with the correct hostname that's running the Impala Daemon  
IMPALAD=`hostname`  
  
impala-shell -i $IMPALAD -q "invalidate metadata;"  
impala-shell -i $IMPALAD -q "SELECT \* FROM test;"

## Spark

Pi Estimator

MASTER=yarn /opt/cloudera/parcels/CDH/lib/spark/bin/run-example SparkPi 100

Wordcount

echo "this is the end. the only end. my friend." > /tmp/sparkin.$$  
hdfs dfs -put /tmp/sparkin.$$ /tmp/  
  
cat <<EOF >/tmp/spark.$$  
val file = sc.textFile("hdfs:///tmp/sparkin.$$")  
val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)  
counts.saveAsTextFile("hdfs:///tmp/sparkout.$$")  
exit  
EOF  
  
cat /tmp/spark.$$ | spark-shell --master yarn-client  
  
hdfs dfs -cat /tmp/sparkout.$$/part-\\*

## Spark2

Pi Estimator

MASTER=yarn /opt/cloudera/parcels/SPARK2/lib/spark2/bin/run-example SparkPi 100

Wordcount

echo "this is the end. the only end. my friend." > /tmp/sparkin2.$$  
hdfs dfs -put /tmp/sparkin2.$$ /tmp/  
  
cat <<EOF >/tmp/spark2.$$  
val file = sc.textFile("hdfs:///tmp/sparkin2.$$")  
val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)  
counts.saveAsTextFile("hdfs:///tmp/sparkout2.$$")  
exit  
EOF  
  
cat /tmp/spark2.$$ | spark2-shell --master yarn-client  
  
hdfs dfs -cat /tmp/sparkout2.$$/part-\\*

## Pig

Query data in a file.

hdfs dfs -copyFromLocal /etc/passwd /tmp/test.pig.passwd.$$  
  
cat <<EOF >/tmp/pig.$$  
A = LOAD '/tmp/test.pig.passwd.$$' USING PigStorage(':');  
B = FOREACH A GENERATE \$0 AS id;  
STORE B INTO '/tmp/test.pig.out.$$';  
EOF  
  
pig /tmp/pig.$$  
  
hdfs dfs -cat /tmp/test.pig.out.$$/part-m-00000

## Solr

Create a test collection. Index it and query it.

SOLRSERVER=solrserverhost  
  
solrctl instancedir --generate /tmp/solr.$$  
solrctl instancedir --create test\_config /tmp/solr.$$  
solrctl collection --create test\_collection -s 1 -c test\_config  
cd /opt/cloudera/parcels/CDH/share/doc/solr-doc\*/example/exampledocs  
java -Durl=http://${SOLRSERVER}:8983/solr/test\_collection/update -jar post.jar \*.xml  
curl "http://${SOLRSERVER}:8983/solr/test\_collection\_shard1\_replica1/select?q=\*%3A\*&wt=json&indent=true"

## Kudu

### **Impala**

Create a Kudu table and query it.

# Replace $IMPALAD with the correct hostname that's running the Impala Daemon  
IMPALAD=`hostname`  
  
impala-shell -i $IMPALAD -q 'CREATE TABLE kudu\_test(id BIGINT, name STRING, PRIMARY KEY(id)) PARTITION BY HASH PARTITIONS 3 STORED AS KUDU;'  
  
impala-shell -i $IMPALAD -q 'INSERT INTO TABLE kudu\_test VALUES (1, "wasim"), (2, "ninad"), (3, "mohsin");'  
  
impala-shell -i $IMPALAD -q 'SELECT \* FROM kudu\_test WHERE id=1;'

## Kafka

Create a test topic. Write to/read from it.

# Replace $ZOOKEEPER and $KAFKA 'localhost' with the correct hostname.  
# Replace the ZOOKEEPER '/kafka' with the correct ZooKeeper root (if you configured one).  
ZOOKEEPER=localhost:2181/kafka  
KAFKA=localhost:9092  
  
kafka-topics --zookeeper ${ZOOKEEPER} --create --topic test --partitions 1 --replication-factor 1  
kafka-topics --zookeeper ${ZOOKEEPER} --list  
  
# Run the consumer and producer in separate windows.  
# Type in text to the producer and watch it appear in the consumer.  
# ^C to quit.  
kafka-console-consumer --zookeeper ${ZOOKEEPER} --new-consumer --topic test  
kafka-console-producer --broker-list ${KAFKA} --topic test

## Clean It Up

Get rid of all the test bits.

hdfs dfs -rm /tmp/hosts  
rm -f /tmp/hosts123  
  
beeline -n `whoami` -u "jdbc:hive2://${HIVESERVER2}:10000/" -e "DROP TABLE test;"  
rm -f /tmp/hive.$$  
  
cat <<EOF >/tmp/hbase-rm.$$  
disable 'test'  
drop 'test'  
exit  
EOF  
hbase shell -n /tmp/hbase-rm.$$  
rm -f /tmp/hbase.$$ /tmp/hbase-rm.$$  
  
hdfs dfs -rm -R /tmp/sparkout.$$ /tmp/sparkin.$$  
rm -f /tmp/spark.$$  
  
hdfs dfs -rm -R /tmp/sparkout2.$$ /tmp/sparkin2.$$  
rm -f /tmp/spark2.$$  
  
hdfs dfs -rm -R /tmp/test.pig.passwd.$$ /tmp/test.pig.out.$$  
rm -f /tmp/pig.$$  
  
solrctl collection --delete test\_collection  
solrctl instancedir --delete test\_config  
sudo su - solr -s /bin/bash -c "hdfs dfs -rm -R -skipTrash /solr/test\_collection"  
rm -rf /tmp/test\_config.$$  
  
impala-shell -i $IMPALAD -q 'DROP TABLE kudu\_test;'  
  
kafka-topics --zookeeper ${ZOOKEEPER}:2181 --delete --topic test

# Secured Cluster

These examples assume a secured (Kerberized) cluster with TLS and use of a non-cluster principal (i.e. the user/principal "centos").

## Preparation

All below commands require Kerberos tickets.

kinit

## HDFS

Basic HDFS functionality.

hdfs dfs -ls /  
hdfs dfs -put /etc/hosts /tmp/hosts  
hdfs dfs -get /tmp/hosts /tmp/hosts123  
cat /tmp/hosts123

## MapReduce

Pi Estimator

yarn jar /opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 10 1000

## Hive

Create an external table and query it.

For Hive on MapReduce, add "set hive.execution.engine=mr;" to the query.

For Hive on Spark, add "set hive.execution.engine=spark;" to the query.

# Replace $HIVESERVER2 with the correct hostname that is running the HS2  
HIVESERVER2=`hostname`  
REALM=`awk '/^default\_realm/{print $3}' /etc/krb5.conf`  
BKOPTS=";principal=hive/\_HOST@${REALM}"  
BTOPTS=";ssl=true;sslTrustStore=/usr/java/default/jre/lib/security/jssecacerts;trustStorePassword=changeit"  
  
# Create hive table  
beeline -u "jdbc:hive2://${HIVESERVER2}:10000/${BKOPTS}${BTOPTS}" -e "CREATE TABLE test(id INT, name STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ' STORED AS TEXTFILE;"  
  
# Create HDFS file  
echo "1 justin" >/tmp/hive.$$  
echo "2 michael" >>/tmp/hive.$$  
hdfs dfs -put /tmp/hive.$$ /user/hive/warehouse/test/hive  
  
# Query hive table  
beeline -u "jdbc:hive2://${HIVESERVER2}:10000/${BKOPTS}${BTOPTS}" -e "SELECT \* FROM test WHERE id=1;"

## HBase

Create a table and query it.

cat <<EOF >/tmp/hbase.$$  
create 'test', 'cf'  
list 'test'  
put 'test', 'row1', 'cf:a', 'value1'  
scan 'test'  
exit  
EOF  
  
hbase shell -n /tmp/hbase.$$

## Impala

Query the hive table created earlier.

# Replace $IMPALAD with the correct hostname that's running the Impala Daemon  
IMPALAD=`hostname`  
IKOPTS="-k"  
ITOPTS="--ssl --ca\_cert=/opt/cloudera/security/x509/ca-chain.cert.pem"  
  
impala-shell -i $IMPALAD $IKOPTS $ITOPTS -q "invalidate metadata;"  
impala-shell -i $IMPALAD $IKOPTS $ITOPTS -q "SELECT \* FROM test;"

## Spark

Pi Estimator

MASTER=yarn /opt/cloudera/parcels/CDH/lib/spark/bin/run-example SparkPi 100

Wordcount

echo "this is the end. the only end. my friend." > /tmp/sparkin.$$  
hdfs dfs -put /tmp/sparkin.$$ /tmp/  
  
cat <<EOF >/tmp/spark.$$  
val file = sc.textFile("hdfs:///tmp/sparkin.$$")  
val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)  
counts.saveAsTextFile("hdfs:///tmp/sparkout.$$")  
exit  
EOF  
  
cat /tmp/spark.$$ | spark-shell --master yarn-client  
  
hdfs dfs -cat /tmp/sparkout.$$/part-\\*

## Spark2

Pi Estimator

MASTER=yarn /opt/cloudera/parcels/SPARK2/lib/spark2/bin/run-example SparkPi 100

Wordcount

echo "this is the end. the only end. my friend." > /tmp/sparkin2.$$  
hdfs dfs -put /tmp/sparkin2.$$ /tmp/  
  
cat <<EOF >/tmp/spark2.$$  
val file = sc.textFile("hdfs:///tmp/sparkin2.$$")  
val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)  
counts.saveAsTextFile("hdfs:///tmp/sparkout2.$$")  
exit  
EOF  
  
cat /tmp/spark2.$$ | spark2-shell --master yarn-client  
  
hdfs dfs -cat /tmp/sparkout2.$$/part-\\*

## Pig

Query data in a file.

hdfs dfs -copyFromLocal /etc/passwd /tmp/test.pig.passwd.$$  
  
cat <<EOF >/tmp/pig.$$  
A = LOAD '/tmp/test.pig.passwd.$$' USING PigStorage(':');  
B = FOREACH A GENERATE \$0 AS id;  
STORE B INTO '/tmp/test.pig.out.$$';  
EOF  
  
pig /tmp/pig.$$  
  
hdfs dfs -cat /tmp/test.pig.out.$$/part-m-00000

## Solr

Create a test collection. Index it and query it.

SOLRSERVER=solrserverhost  
  
solrctl instancedir --generate /tmp/solr.$$  
solrctl instancedir --create test\_config /tmp/solr.$$  
solrctl collection --create test\_collection -s 1 -c test\_config  
cd /opt/cloudera/parcels/CDH/share/doc/solr-doc\*/example/exampledocs  
java -Durl=http://${SOLRSERVER}:8983/solr/test\_collection/update -jar post.jar \*.xml  
curl "http://${SOLRSERVER}:8983/solr/test\_collection\_shard1\_replica1/select?q=\*%3A\*&wt=json&indent=true"

## Kafka

Create a test topic. Write to/read from it.

# Replace $ZOOKEEPER and $KAFKA 'localhost' with the correct hostname.  
# Replace the ZOOKEEPER '/kafka' with the correct ZooKeeper root (if you configured one).  
ZOOKEEPER=localhost:2181/kafka  
KAFKA=localhost:9093  
  
kafka-topics --zookeeper ${ZOOKEEPER} --create --topic test --partitions 1 --replication-factor 1  
kafka-topics --zookeeper ${ZOOKEEPER} --list  
  
# Run the consumer and producer in separate windows.  
# Type in text to the producer and watch it appear in the consumer.  
# ^C to quit.  
kafka-console-consumer --zookeeper ${ZOOKEEPER} --bootstrap-server ${KAFKA} --new-consumer --topic test  
kafka-console-producer --broker-list ${KAFKA} --topic test

## Clean It Up

Get rid of all the test bits.

hdfs dfs -rm /tmp/hosts  
rm -f /tmp/hosts123  
  
beeline -u "jdbc:hive2://${HIVESERVER2}:10000/${BKOPTS}${BTOPTS}" -e "DROP TABLE test;"  
rm -f /tmp/hive.$$  
  
cat <<EOF >/tmp/hbase-rm.$$  
disable 'test'  
drop 'test'  
exit  
EOF  
hbase shell -n /tmp/hbase-rm.$$  
rm -f /tmp/hbase.$$ /tmp/hbase-rm.$$  
  
hdfs dfs -rm -R /tmp/sparkout.$$ /tmp/sparkin.$$  
rm -f /tmp/spark.$$  
  
hdfs dfs -rm -R /tmp/sparkout2.$$ /tmp/sparkin2.$$  
rm -f /tmp/spark2.$$  
  
hdfs dfs -rm -R /tmp/test.pig.passwd.$$ /tmp/test.pig.out.$$  
rm -f /tmp/pig.$$  
  
solrctl collection --delete test\_collection  
solrctl instancedir --delete test\_config  
#kinit solr  
#hdfs dfs -rm -R -skipTrash /solr/test\_collection  
rm -rf /tmp/test\_config.$$  
  
kafka-topics --zookeeper ${ZOOKEEPER} --delete --topic test  
  
  
kdestroy

# Secured Cluster with Sentry

These examples assume a secured (Kerberized) cluster with TLS and Sentry and use of a non-cluster principal (i.e. the user/principal "centos").

DRAFT

## Preparation

Sentry needs to be [configured with roles](https://clairvoyant.atlassian.net/wiki/spaces/HAD/pages/162205639/Enabling+Sentry+via+CM).

HBase needs to be configured with roles.

Hive needs to have admin groups (AD or LDAP) added to the Hive Metastore Access Control and Proxy User Groups Overrride (in addition to hue and hive).

All below commands require Kerberos tickets.

kinit

## HDFS

Basic HDFS functionality.

hdfs dfs -ls /  
hdfs dfs -put /etc/hosts /tmp/hosts  
hdfs dfs -get /tmp/hosts /tmp/hosts123  
cat /tmp/hosts123

## MapReduce

Pi Estimator

yarn jar /opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 10 1000

## Hive

Create an external table and query it.

For Hive on MapReduce, add "set hive.execution.engine=mr;" to the query.

For Hive on Spark, add "set hive.execution.engine=spark;" to the query.

# Replace $HIVESERVER2 with the correct hostname that is running the HS2  
HIVESERVER2=`hostname`  
REALM=`awk '/^default\_realm/{print $3}' /etc/krb5.conf`  
BKOPTS=";principal=hive/\_HOST@${REALM}"  
BTOPTS=";ssl=true;sslTrustStore=/usr/java/default/jre/lib/security/jssecacerts;trustStorePassword=changeit"  
  
# Create hive table  
beeline -u "jdbc:hive2://${HIVESERVER2}:10000/${BKOPTS}${BTOPTS}" -e "CREATE TABLE test(id INT, name STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ' STORED AS TEXTFILE;"  
  
# Insert data  
beeline -u "jdbc:hive2://${HIVESERVER2}:10000/${BKOPTS}${BTOPTS}" -e "INSERT INTO TABLE test VALUES (1, "justin"), (2, "michael");"  
  
# Query hive table  
beeline -u "jdbc:hive2://${HIVESERVER2}:10000/${BKOPTS}${BTOPTS}" -e "SELECT \* FROM test WHERE id=1;"

## HBase

Create a table and query it.

cat <<EOF >/tmp/hbase.$$  
create 'test', 'cf'  
list 'test'  
put 'test', 'row1', 'cf:a', 'value1'  
scan 'test'  
exit  
EOF  
  
hbase shell -n /tmp/hbase.$$

## Impala

Query the hive table created earlier.

# Replace $IMPALAD with the correct hostname that's running the Impala Daemon  
IMPALAD=`hostname`  
IKOPTS="-k"  
ITOPTS="--ssl --ca\_cert=/opt/cloudera/security/x509/ca-chain.cert.pem"  
  
impala-shell -i $IMPALAD $IKOPTS $ITOPTS -q "invalidate metadata;"  
impala-shell -i $IMPALAD $IKOPTS $ITOPTS -q "SELECT \* FROM test;"

## Spark

Pi Estimator

MASTER=yarn /opt/cloudera/parcels/CDH/lib/spark/bin/run-example SparkPi 100

Wordcount

echo "this is the end. the only end. my friend." > /tmp/sparkin.$$  
hdfs dfs -put /tmp/sparkin.$$ /tmp/  
  
cat <<EOF >/tmp/spark.$$  
val file = sc.textFile("hdfs:///tmp/sparkin.$$")  
val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)  
counts.saveAsTextFile("hdfs:///tmp/sparkout.$$")  
exit  
EOF  
  
cat /tmp/spark.$$ | spark-shell --master yarn-client  
  
hdfs dfs -cat /tmp/sparkout.$$/part-\\*

## Spark2

Pi Estimator

MASTER=yarn /opt/cloudera/parcels/SPARK2/lib/spark2/bin/run-example SparkPi 100

Wordcount

echo "this is the end. the only end. my friend." > /tmp/sparkin2.$$  
hdfs dfs -put /tmp/sparkin2.$$ /tmp/  
  
cat <<EOF >/tmp/spark2.$$  
val file = sc.textFile("hdfs:///tmp/sparkin2.$$")  
val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)  
counts.saveAsTextFile("hdfs:///tmp/sparkout2.$$")  
exit  
EOF  
  
cat /tmp/spark2.$$ | spark2-shell --master yarn-client  
  
hdfs dfs -cat /tmp/sparkout2.$$/part-\\*

## Pig

Query data in a file.

hdfs dfs -copyFromLocal /etc/passwd /tmp/test.pig.passwd.$$  
  
cat <<EOF >/tmp/pig.$$  
A = LOAD '/tmp/test.pig.passwd.$$' USING PigStorage(':');  
B = FOREACH A GENERATE \$0 AS id;  
STORE B INTO '/tmp/test.pig.out.$$';  
EOF  
  
pig /tmp/pig.$$  
  
hdfs dfs -cat /tmp/test.pig.out.$$/part-m-00000

## Kafka

Create a test topic.

# Replace $ZOOKEEPER and $KAFKA 'localhost' with the correct hostname.  
# Replace the ZOOKEEPER '/kafka' with the correct ZooKeeper root (if you configured one).  
ZOOKEEPER=localhost:2181/kafka  
KAFKA=localhost:9093  
  
kafka-topics --zookeeper ${ZOOKEEPER} --create --topic test --partitions 1 --replication-factor 1  
kafka-topics --zookeeper ${ZOOKEEPER} --list  
  
# Run the consumer and producer in separate windows.  
# Type in text to the producer and watch it appear in the consumer.  
# ^C to quit.  
kafka-console-consumer --zookeeper ${ZOOKEEPER} --bootstrap-server ${KAFKA} --new-consumer --topic test  
kafka-console-producer --broker-list ${KAFKA} --topic test

## Clean It Up

Get rid of all the test bits.

hdfs dfs -rm /tmp/hosts  
rm -f /tmp/hosts123  
  
beeline -u "jdbc:hive2://${HIVESERVER2}:10000/${BKOPTS}${BTOPTS}" -e "DROP TABLE test;"  
rm -f /tmp/hive.$$  
  
cat <<EOF >/tmp/hbase-rm.$$  
disable 'test'  
drop 'test'  
exit  
EOF  
hbase shell -n /tmp/hbase-rm.$$  
rm -f /tmp/hbase.$$ /tmp/hbase-rm.$$  
  
hdfs dfs -rm -R /tmp/sparkout.$$ /tmp/sparkin.$$  
rm -f /tmp/spark.$$  
  
hdfs dfs -rm -R /tmp/sparkout2.$$ /tmp/sparkin2.$$  
rm -f /tmp/spark2.$$  
  
hdfs dfs -rm -R /tmp/test.pig.passwd.$$ /tmp/test.pig.out.$$  
rm -f /tmp/pig.$$  
  
kafka-topics --zookeeper ${ZOOKEEPER} --delete --topic test  
kdestroy

CDH 5.12 can also use the following for the MR PiEstimator:

yarn jar /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar pi 10 1000

# More Hadoop Smoke Tests

LDAP (not Kerberos) authentication of Impala over TLS via Amazon ELB.

[centos@ip-10-28-0-61 ~]$ cat file  
#!/bin/bash  
echo -n password  
[centos@ip-10-28-0-61 ~]$ ./file  
password[centos@ip-10-28-0-61 ~]$  
[centos@ip-10-28-0-61 ~]$ impala-shell --impalad=bd3.lift361.com:21000 --ssl --ca\_cert=/etc/pki/tls/certs/ca-bundle.crt --ldap --user=marnold --ldap\_password\_cmd=./file --query="SELECT \* FROM test;"  
Starting Impala Shell using LDAP-based authentication  
SSL is enabled  
Connected to bd3.lift361.com:21000  
Server version: impalad version 2.7.0-cdh5.9.1 RELEASE (build 24ad6df788d66e4af9496edb26ac4d1f1d2a1f2c)  
Query: select \* FROM test  
Query submitted at: 2017-01-31 21:43:34 (Coordinator: https://ip-10-28-0-31.ec2.internal:25000)  
Query progress can be monitored at: https://ip-10-28-0-31.ec2.internal:25000/query\_plan?query\_id=f84353ba44948d6e:b0baeaf200000000  
+----+---------+  
| id | name |  
+----+---------+  
| 1 | justin |  
| 2 | michael |  
+----+---------+  
Fetched 2 row(s) in 0.46s