Apache Spark with Hortonworks Data Platform

Saptak Sen [@saptak]

Technical Product Manager

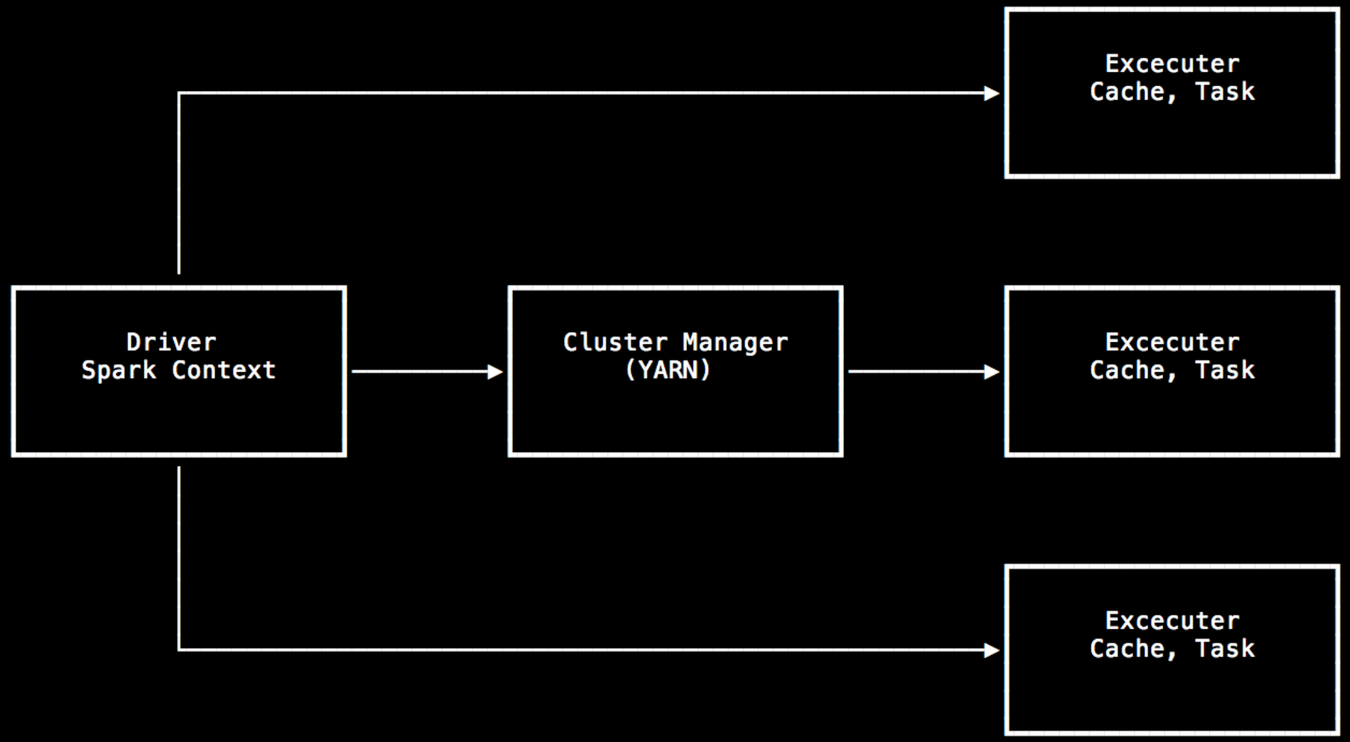
What we will do?

- Setup
  - YARN Queues for Spark
  - Zeppelin Notebook
- Spark Concepts
- Hands-On

Get the slides from: http://l.saptak.in/sparkseattle

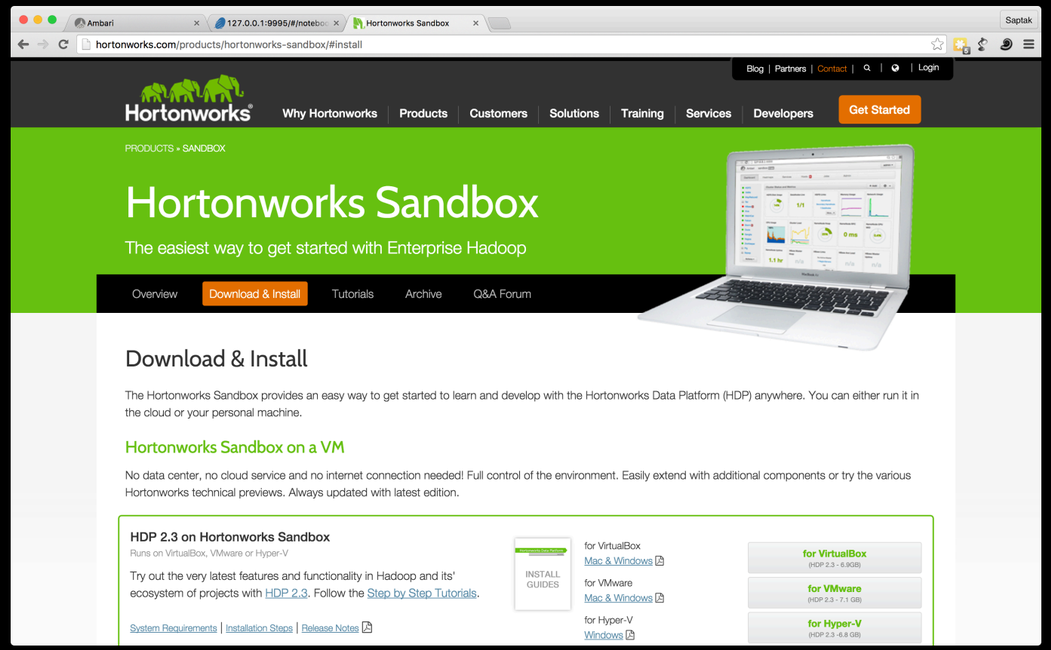
What is Apache Spark?

A fast and general engine for large-scale data processing.



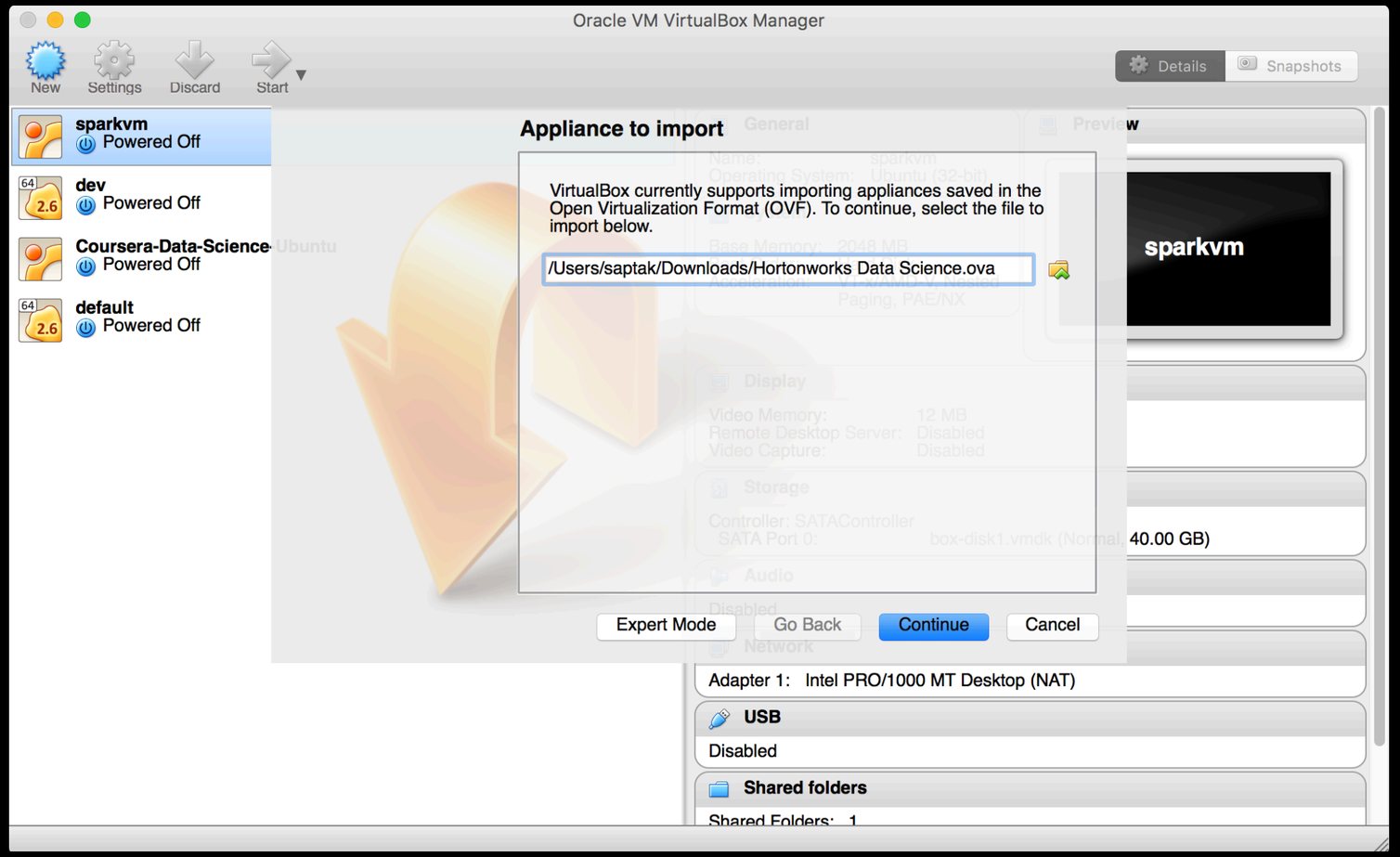
Setup

Download Hortonworks Sandbox

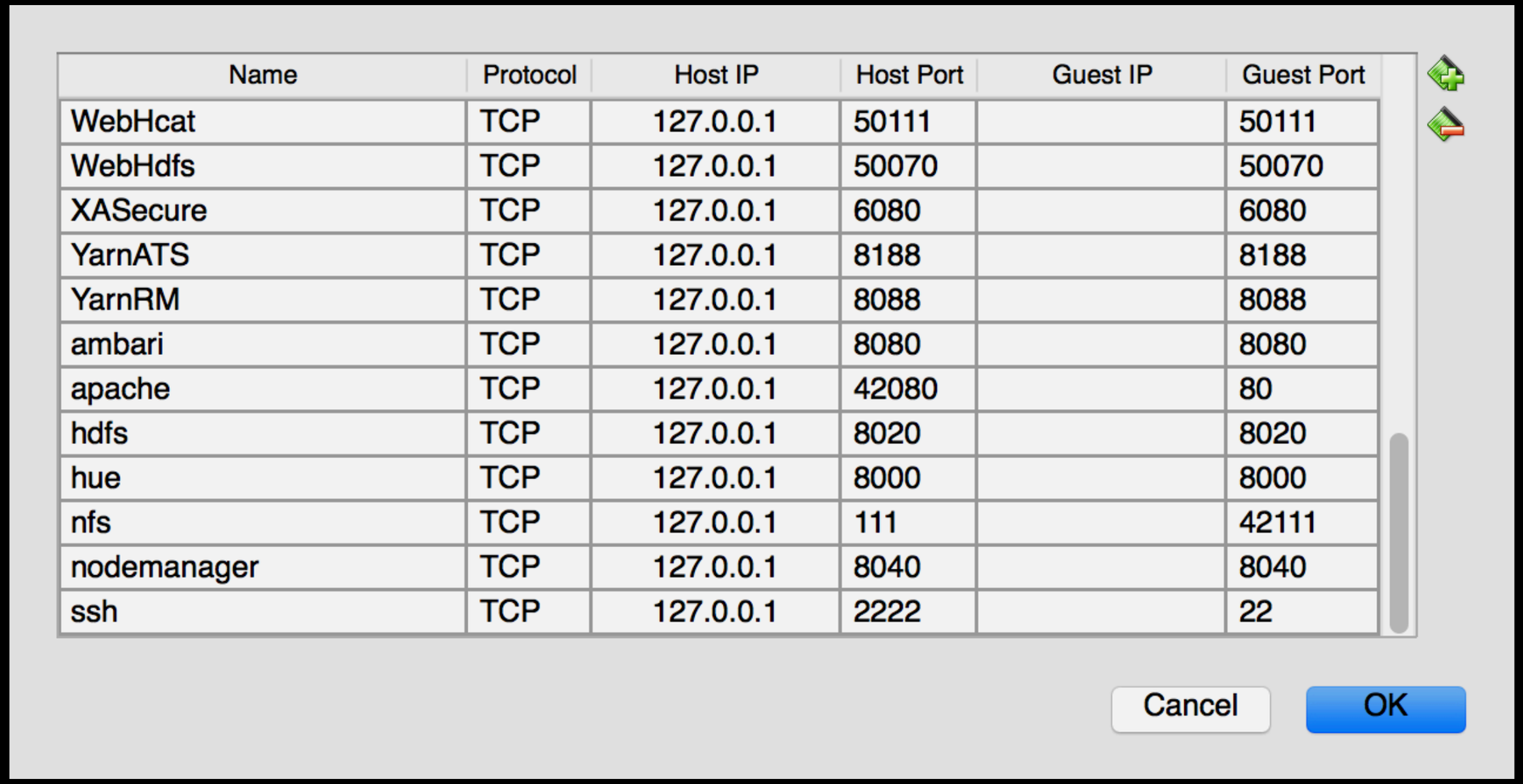


http://hortonworks.com/sandbox

Configure Hortonworks Sandbox

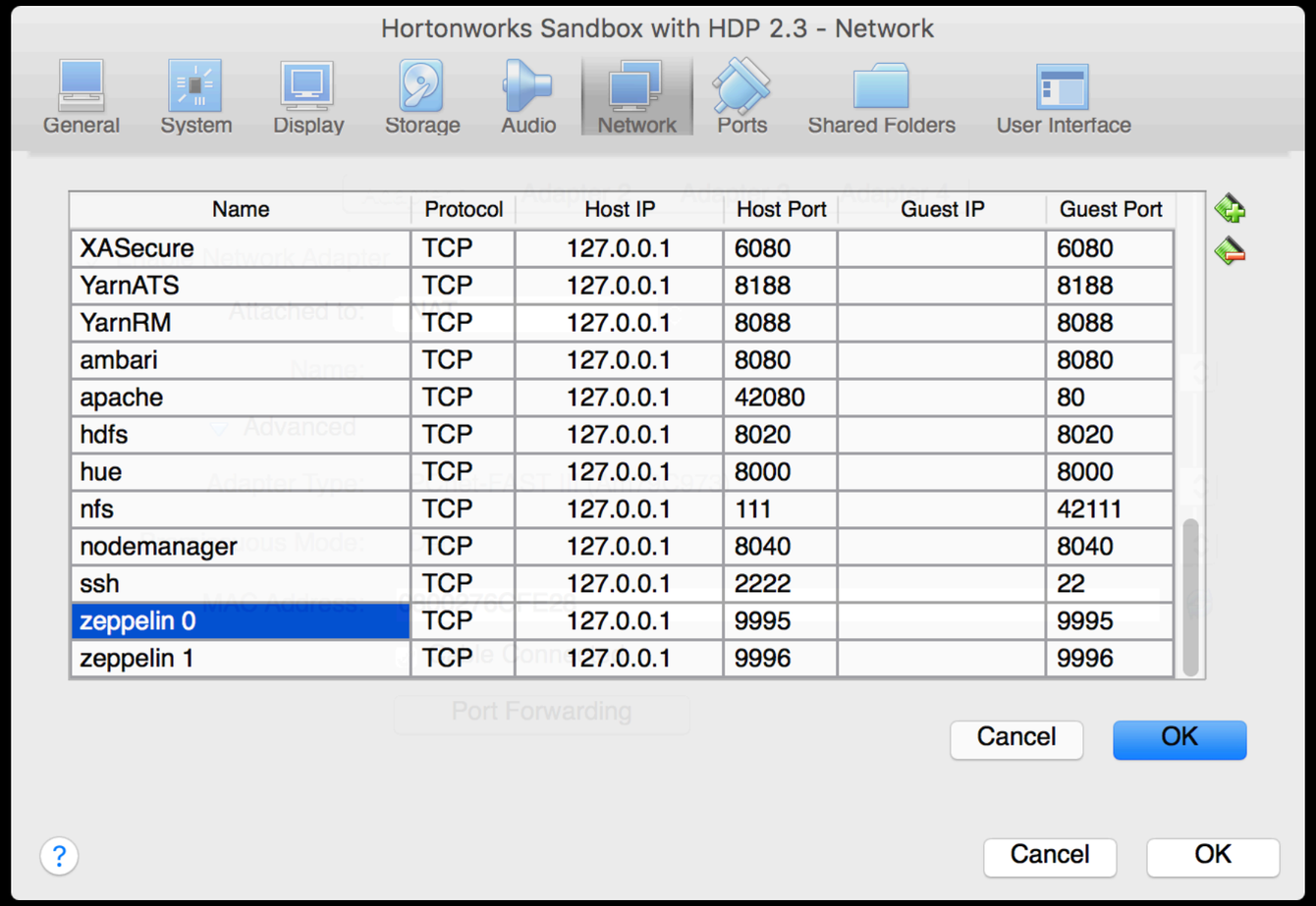


Open network settings for VM

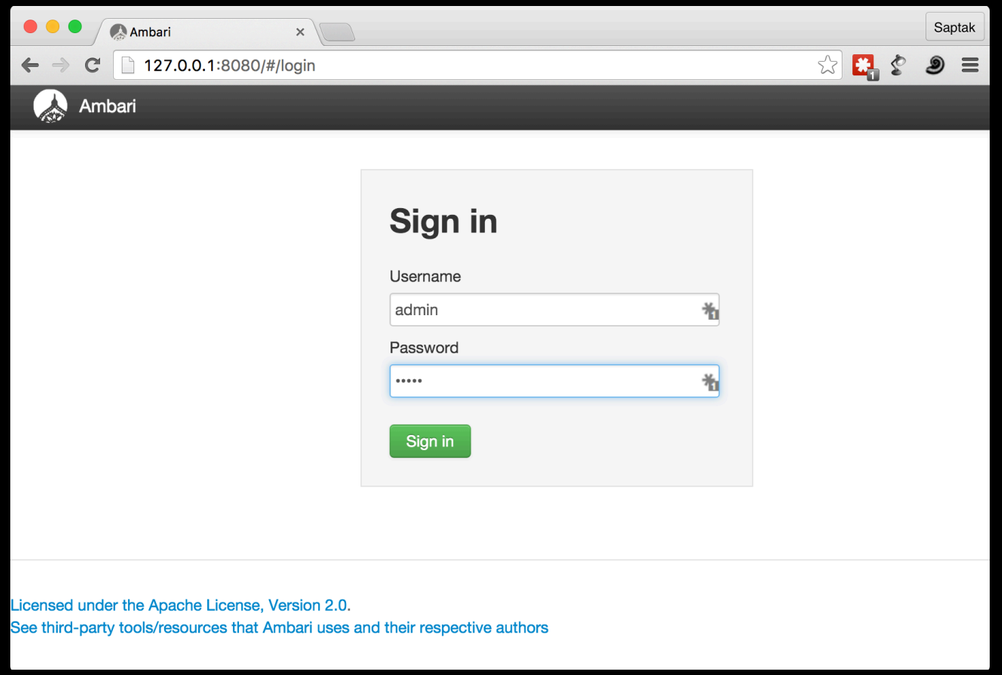


Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
WebHcat	TCP	127.0.0.1	50111		50111
WebHdfs	TCP	127.0.0.1	50070		50070
XASecure	TCP	127.0.0.1	6080		6080
YarnATS	TCP	127.0.0.1	8188		8188
YarnRM	TCP	127.0.0.1	8088		8088
ambari	TCP	127.0.0.1	8080		8080
apache	TCP	127.0.0.1	42080		80
hdfs	TCP	127.0.0.1	8020		8020
hue	TCP	127.0.0.1	8000		8000
nfs	TCP	127.0.0.1	111		42111
nodemanager	TCP	127.0.0.1	8040		8040
ssh	TCP	127.0.0.1	2222		22

Open port for Zeppelin

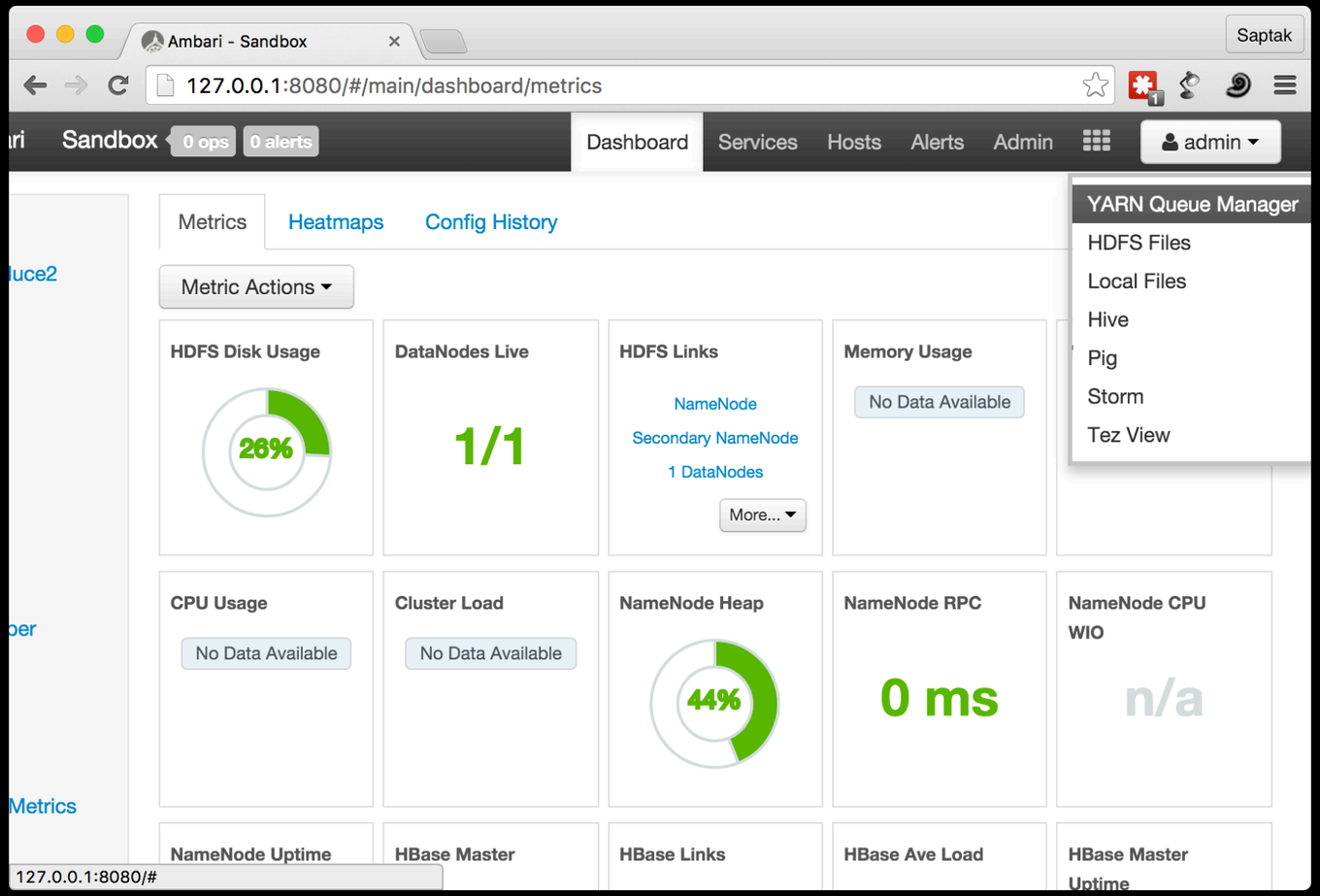


Login to Ambari

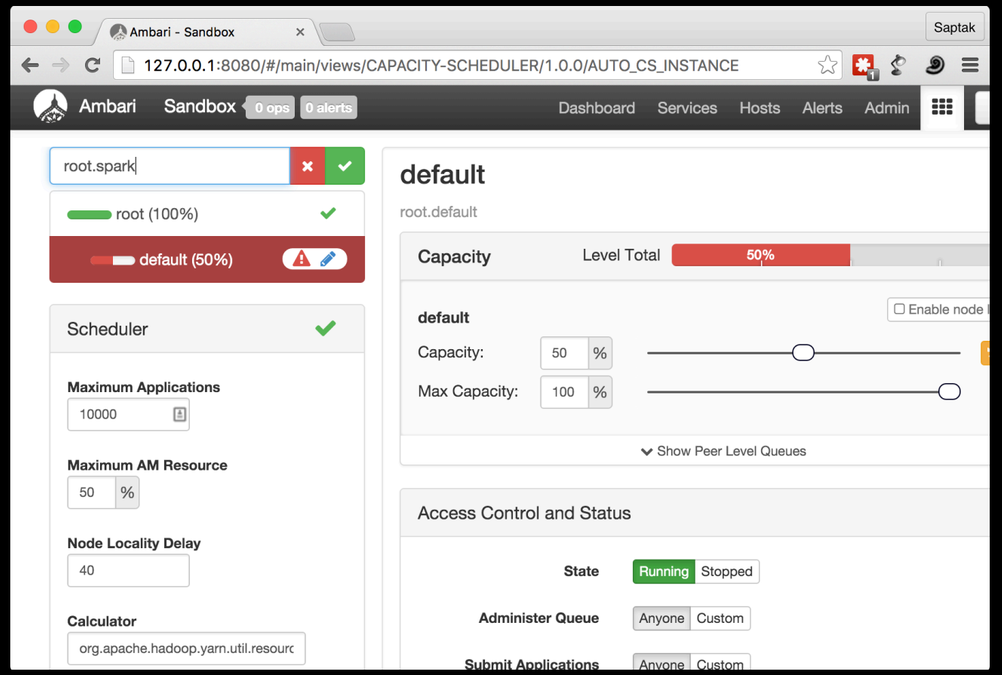


Ambari is at port 8080. Username and password is admin/admin

Open YARN Queue Manager

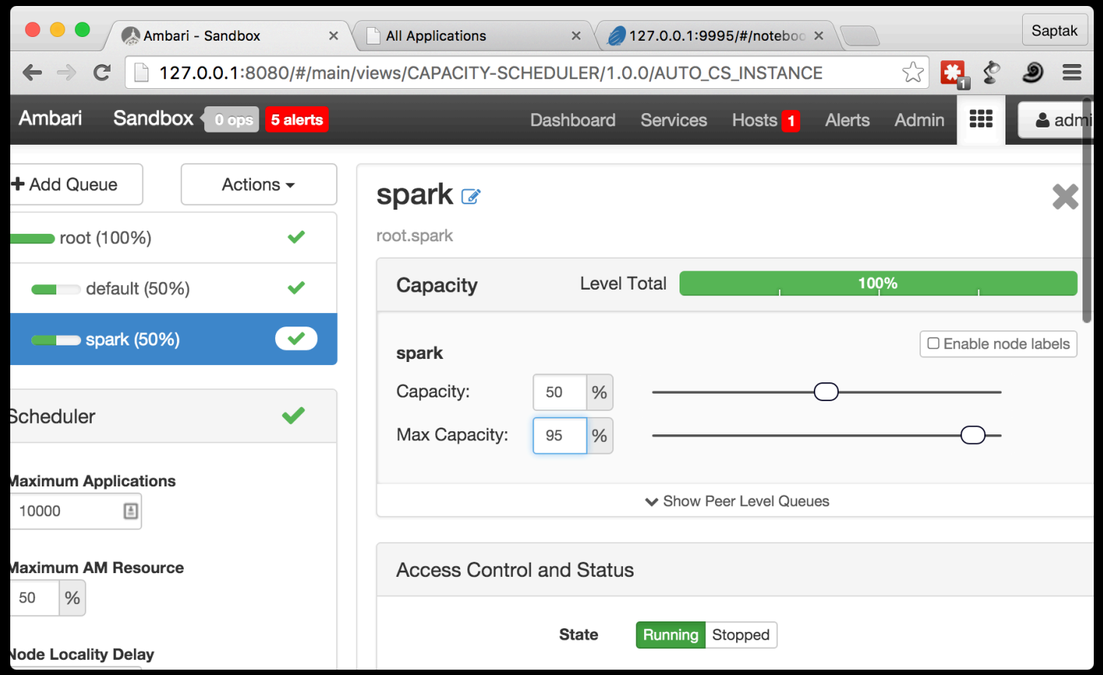


Create a dedicated queue for Spark



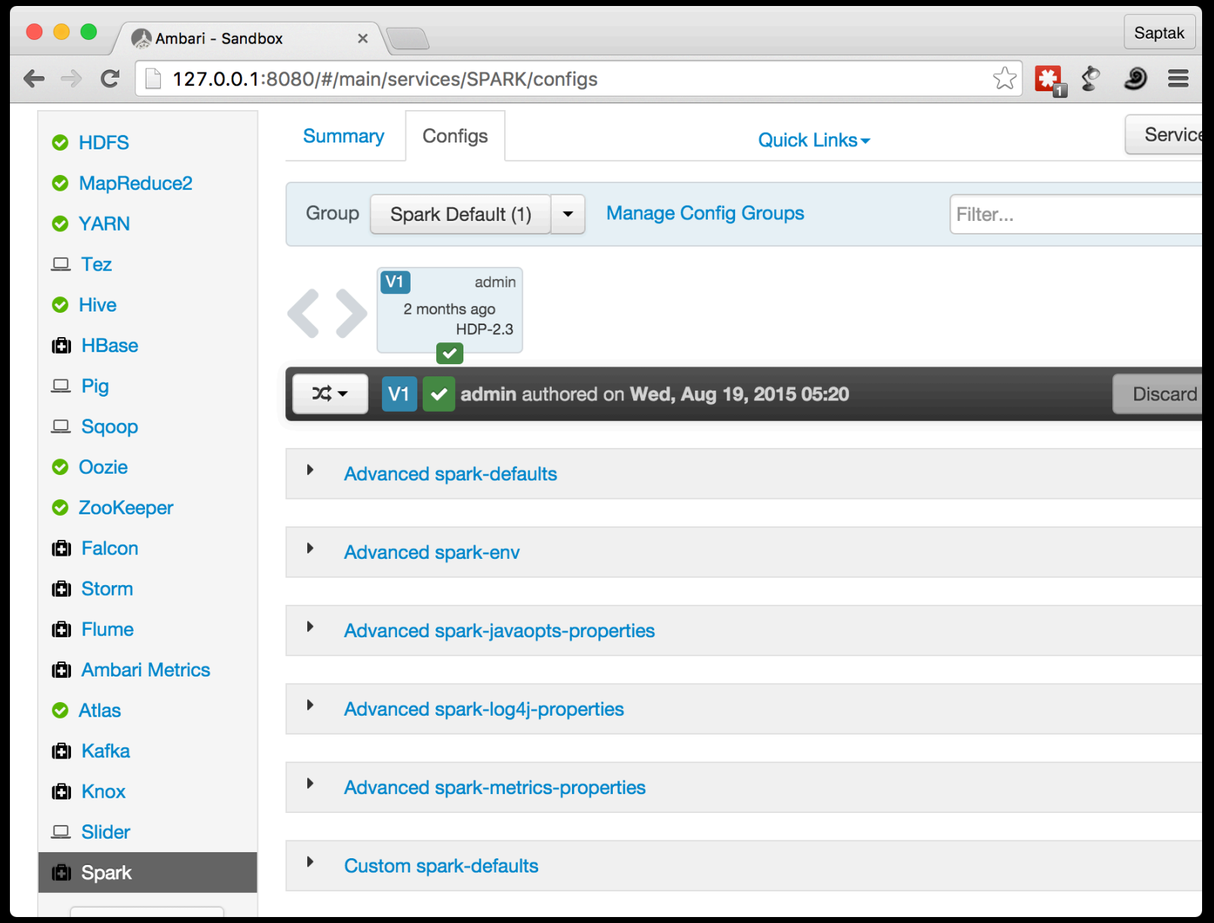
Set name to `root.spark`, Capacity to `50%` and Max Capacity to `95%`

Save and Refresh Queues

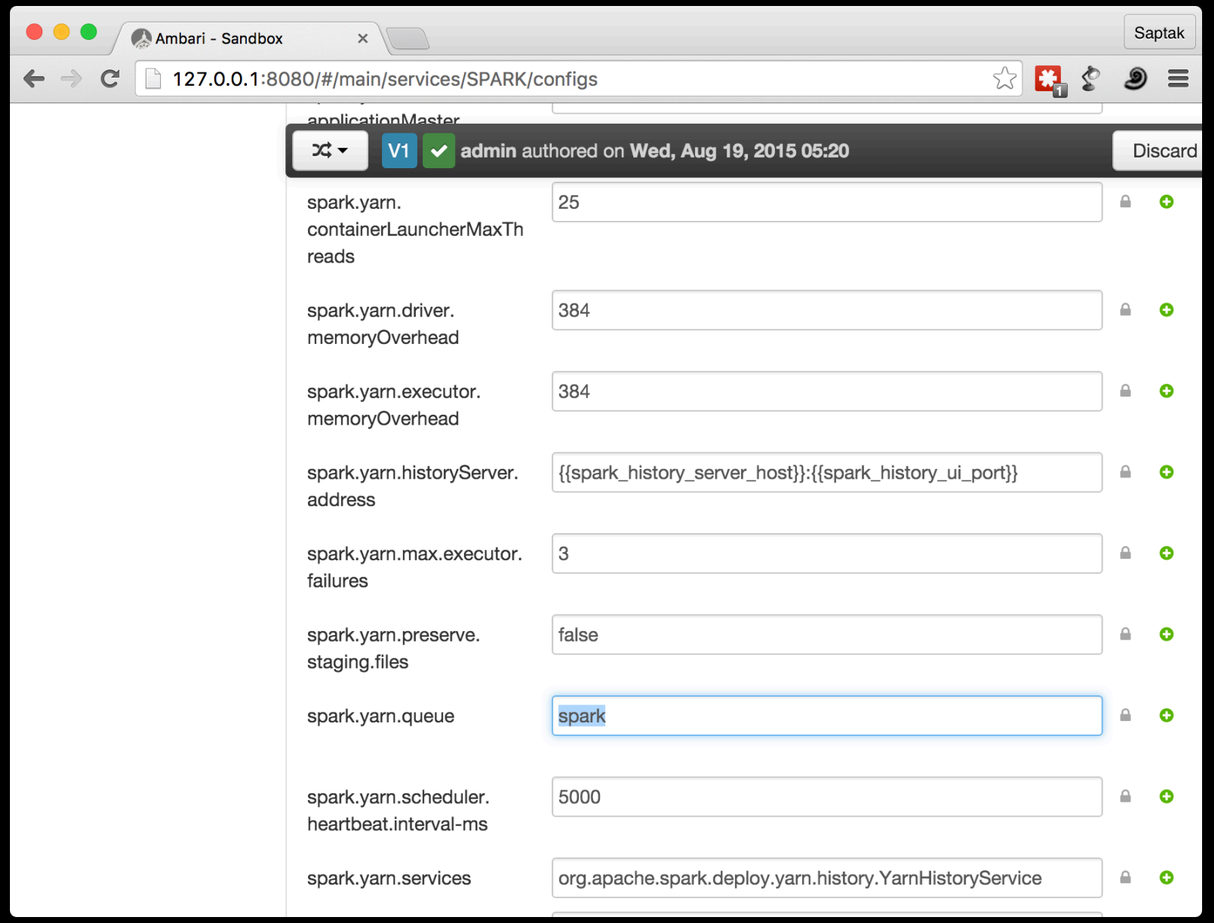


Adjust capacity of the default queue to 50% if required before saving

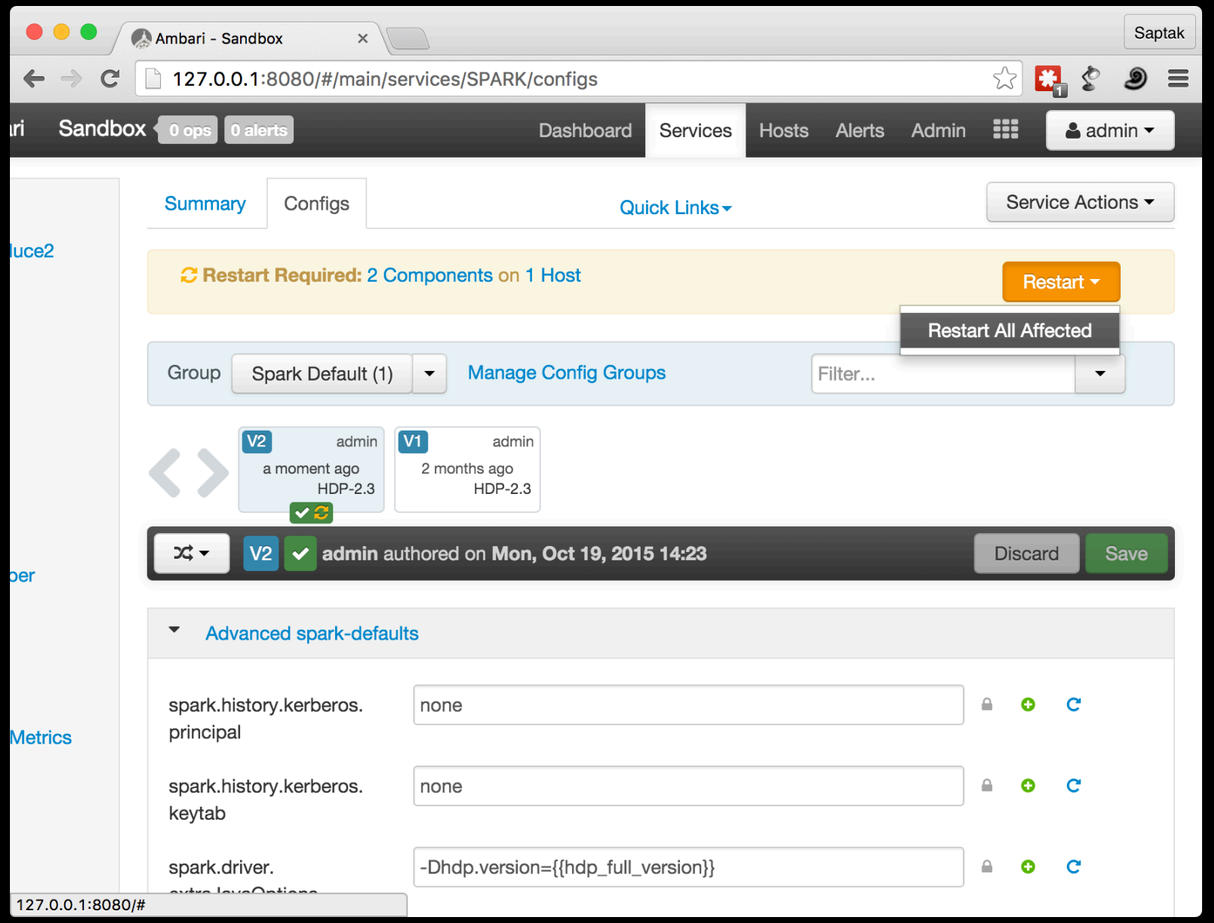
Open Apache Spark configuration



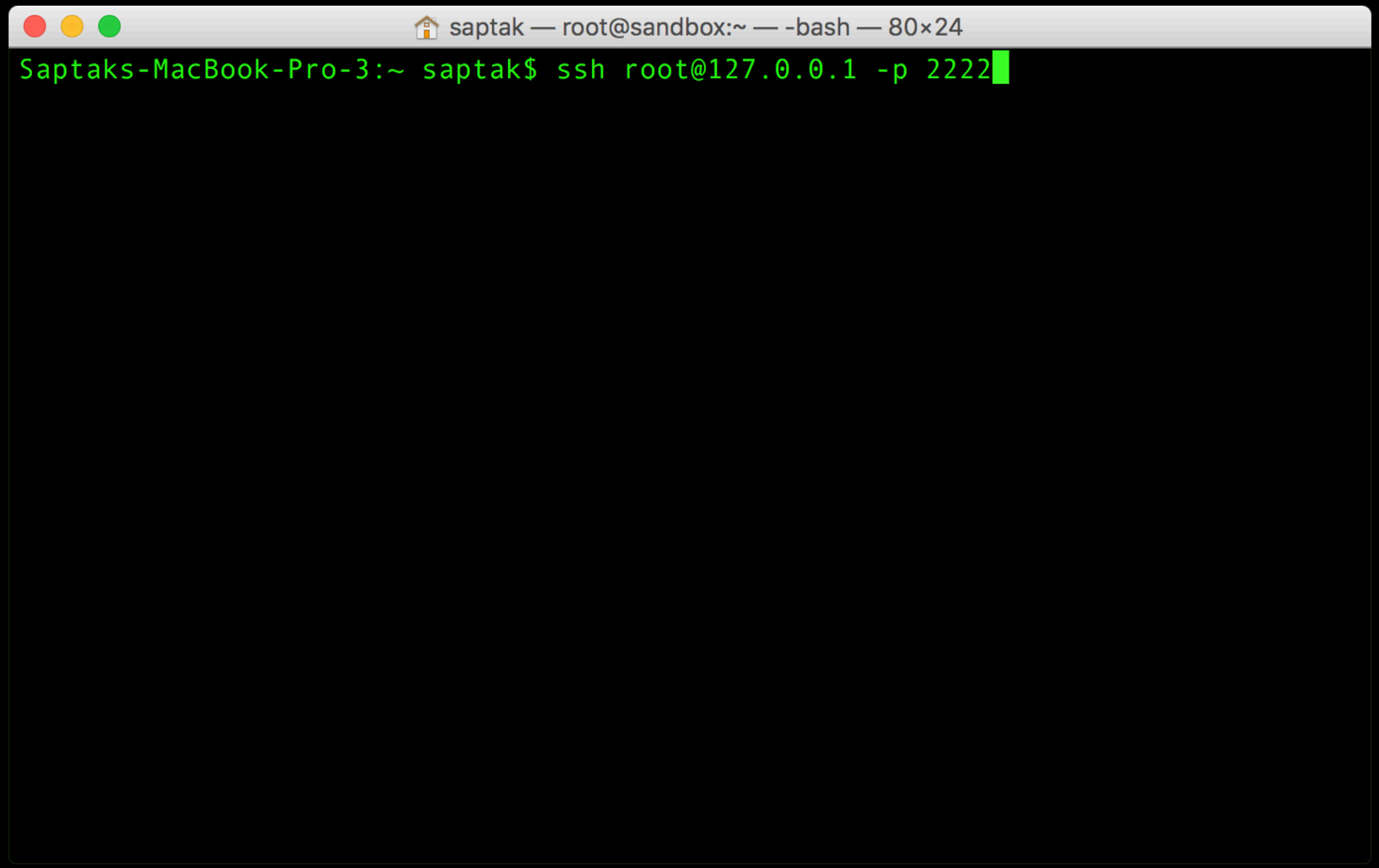
Set the 'spark' queue as the default queue for Spark



Restart All Affected Services

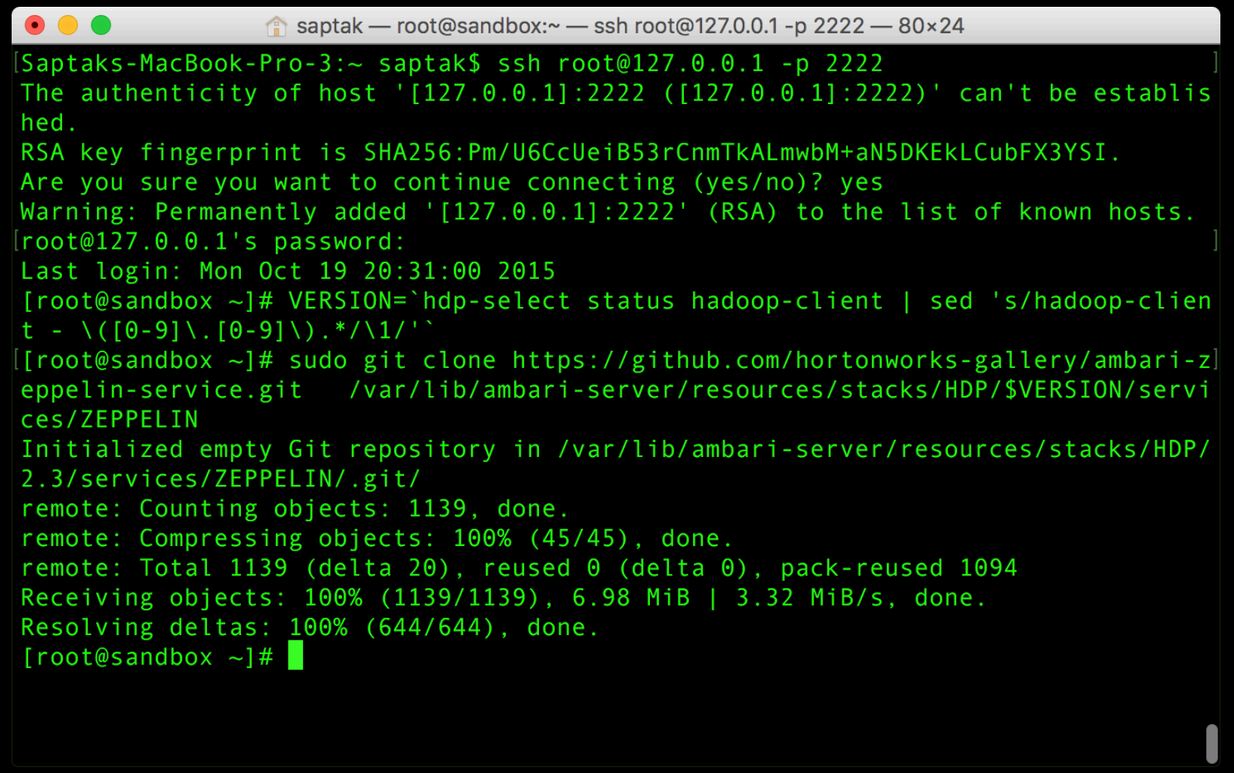


SSH into your Sandbox



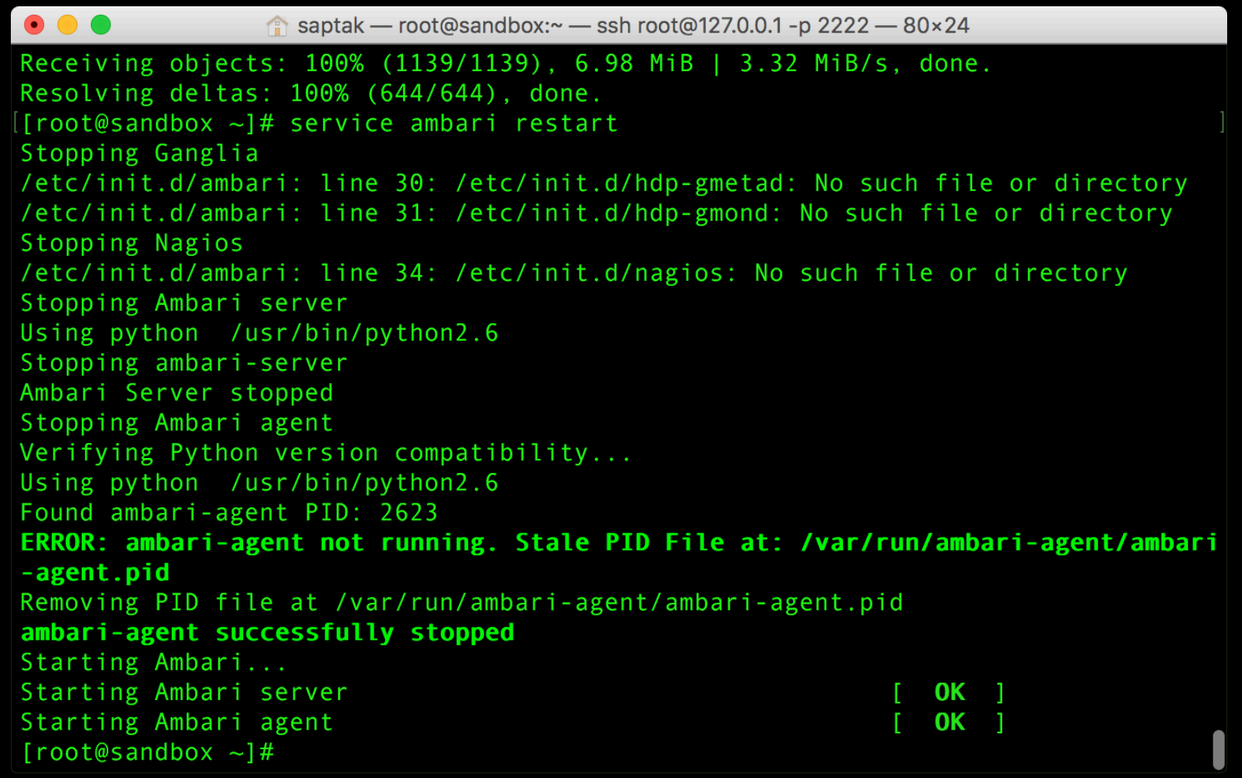
Install the Zeppelin Service for Ambari

```
VERSION=`hdp-select status hadoop-client | sed 's/hadoop-client - \([0-9]\.[0-9]\).*/\1/'`
sudo git clone https://github.com/hortonworks-gallery/ambari-zeppelin-service.git   /var/lib/ambari-server/resources/stacks/HDP/$VERSION/services/ZEPPELIN
```

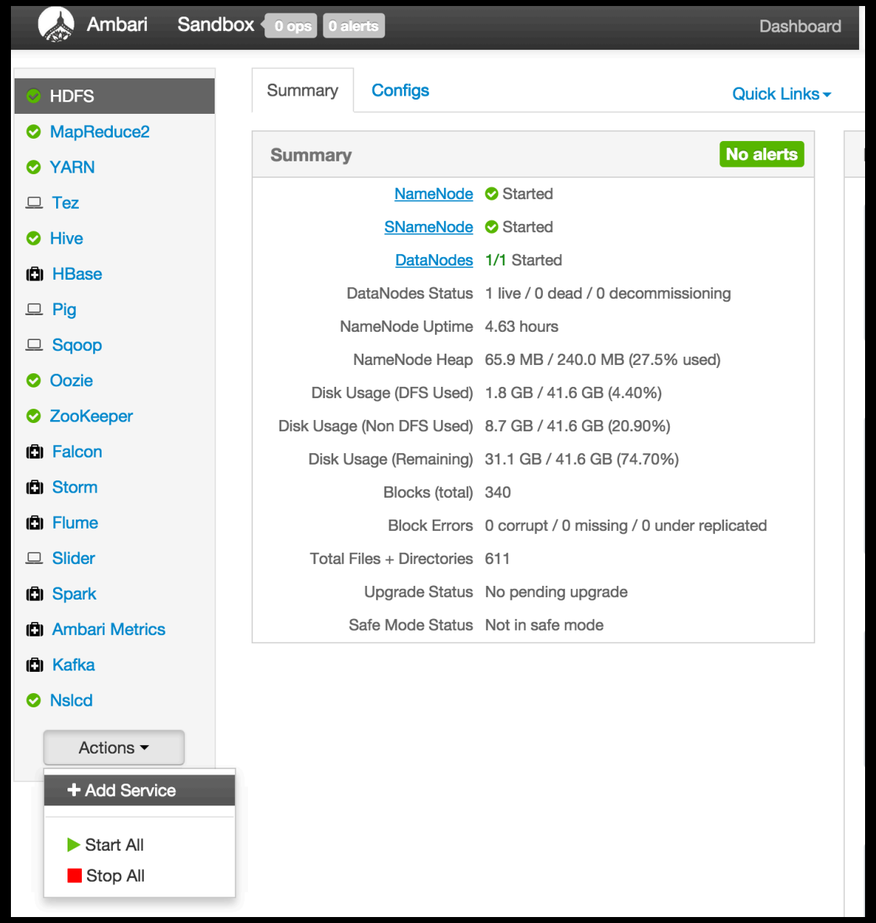


Restart Ambari to enumerate the new Services

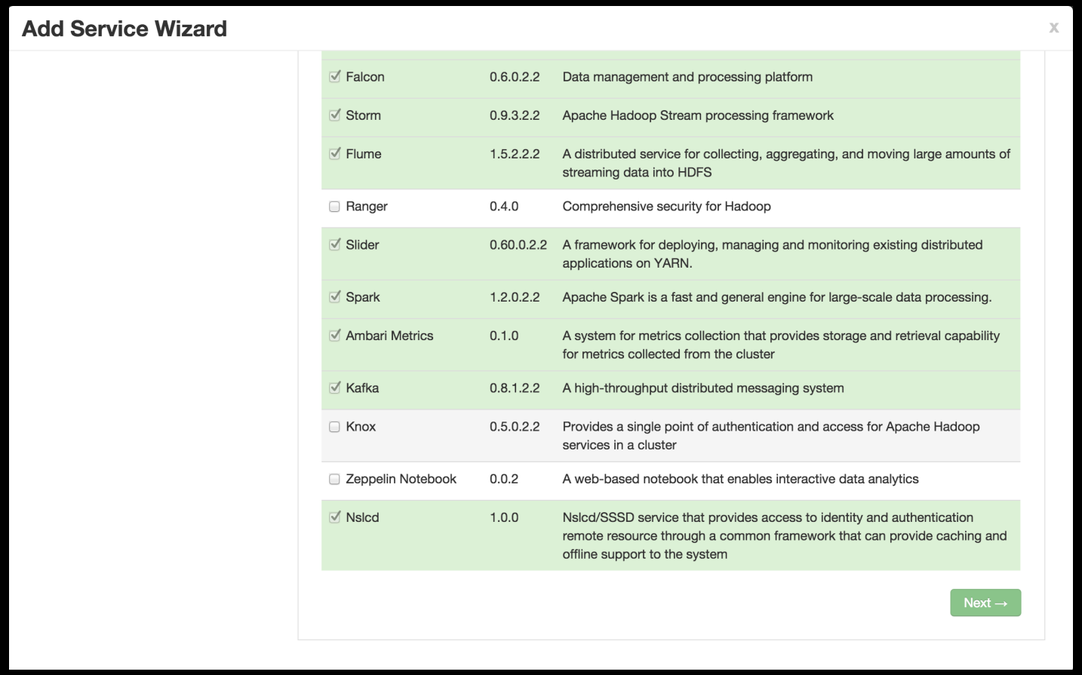
```
service ambari restart
```



Go to Ambari to Add the Zeppelin Service

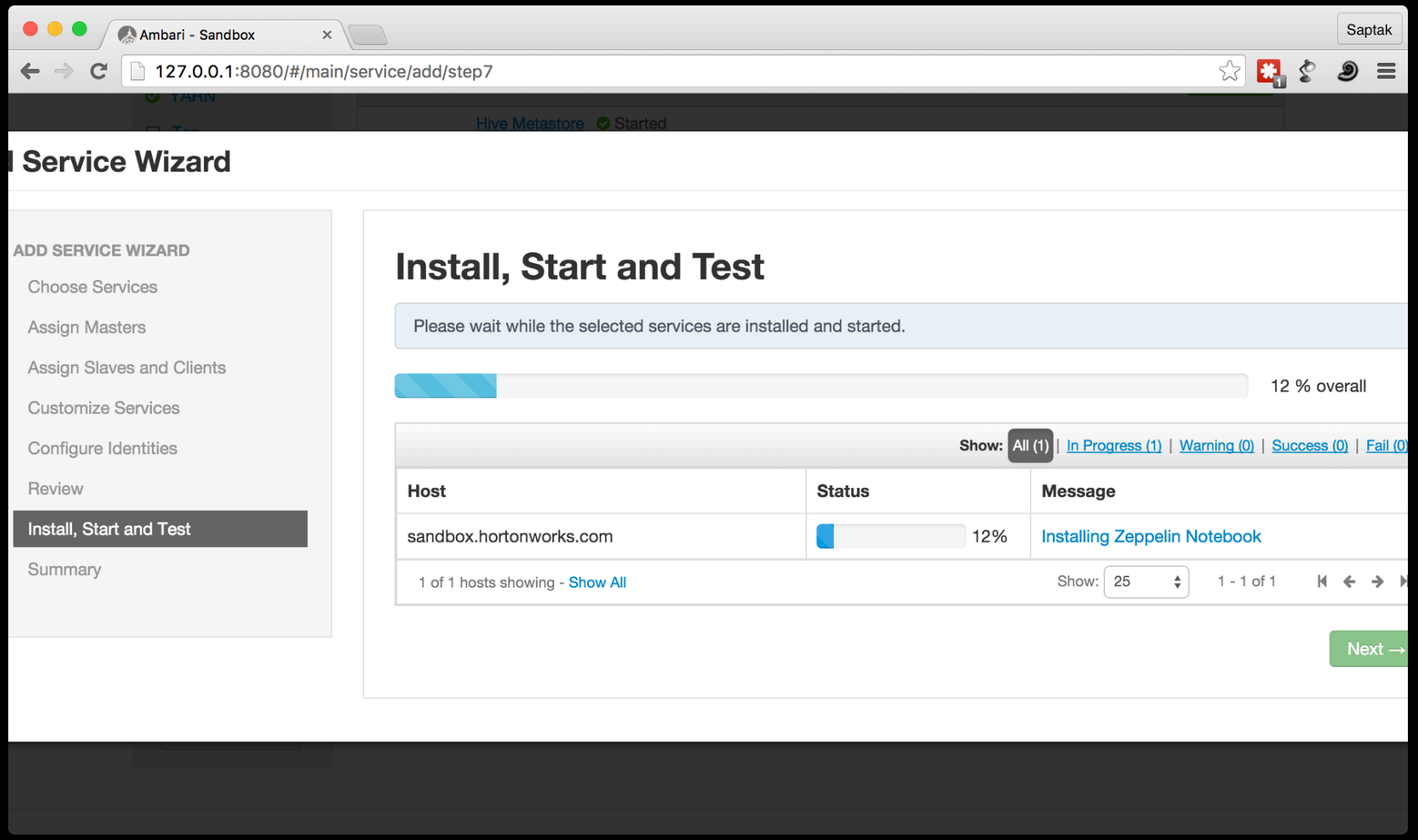


Select Zeppelin Notebook

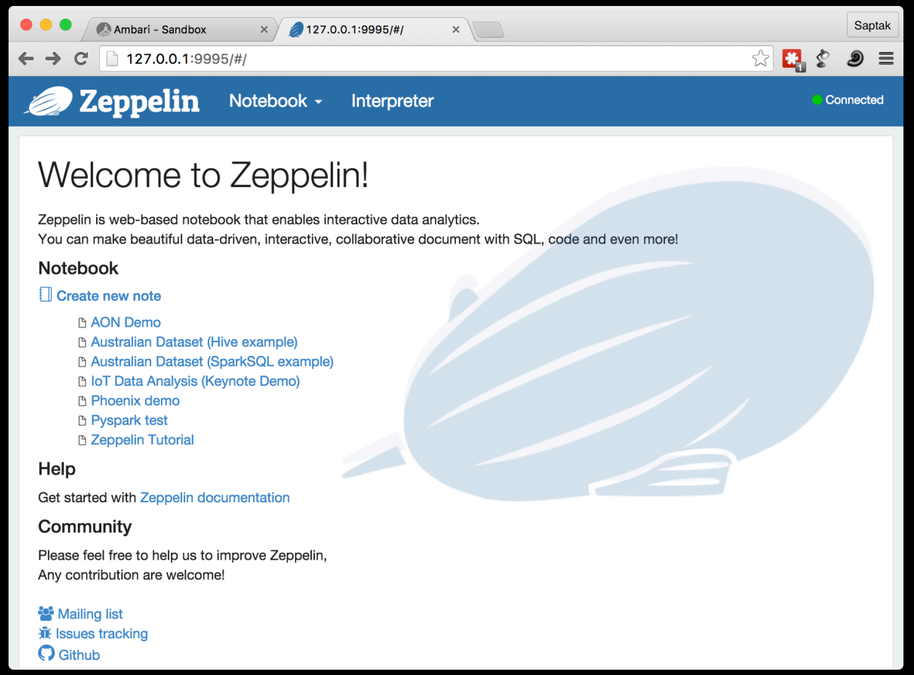


click Next and select default configurations unless you know what you are doing

Wait for Zeppelin deployment to complete



Launch Zeppelin from your browser



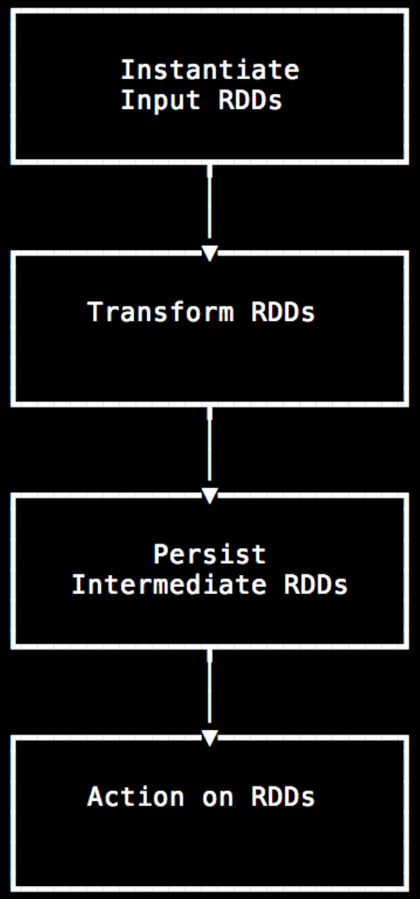
Zeppelin is at port 9995

Concepts

RDD

- Resilient
- Distributed
- Dataset

Program Flow



SparkContext

- Created by your driver Program
- Responsible for making your RDD's resilient and Distributed
- Instantiates RDDs
- The Spark shell creates a "sc" object for you

Creating RDDs

```
myRDD = parallelize([1,2,3,4])
myRDD = sc.textFile("hdfs:///tmp/shakepeare.txt")
#                       file://, s3n://


hiveCtx = HiveContext(sc)
rows = hiveCtx.sql("SELECT name, age from users")
```

Creating RDDs

- Can also create from:
  - JDBC
  - Cassandra
  - HBase
  - Elasticsearch
  - JSON, CSV, sequence files, object files, ORC, Parquet, Avro

Passing Functions to Spark

Many RDD methods accept a function as a parameter

```
myRdd.map(lambda x: x*x)
```

is the same things as

```
def sqrN(x):
  return x*x

myRdd.map(sqrN)
```

RDD Transformations

- map
- flatmap
- filter
- distinct
- sample
- union, intersection, subtract, cartesian

Map example

```
myRdd = sc. parallelize([1,2,3,4])
myRdd.map(lambda x: x+x)
```

- Results: 1, 2, 6, 8

Transformations are lazily evaluated

RDD Actions

- collect
- count
- countByValue
- take
- top
- reduce

Reduce

```
sum = rdd.reduce(lambda x, y: x + y)
```

Aggregate

```
sumCount = nums.aggregate(
                (0, 0),
                (lambda acc, value: (acc[0] + value, acc[1] + 1)),
                (lambda acc1 , acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])))

return sumCount[0] / float(sumCount[1])
```

Imports

```
from pyspark import SparkConf, SparkContext
import collections
```

Setup the Context

```
conf = SparkConf().setMaster("local").setAppName("foo")
sc = SparkContext(conf = conf)
```

First RDD

```
lines = sc.textFile("hdfs:///tmp/shakepeare.txt")
```

Extract the Data

```
ratings = lines.map(lambda x: x.split()[2])
```

Aggregate

```
result = ratings.countByValue()
```

Sort and Print the results

```
sortedResults = collections.OrderedDict(sorted(result.items()))

for key, value in sortedResults.iteritems():
  print "%s %i" % (key, value)
```

Run the Program

```
spark-submit ratings-counter.py
```

Key, Value RDD

```
#list of key, value RDDs where the value is 1.
totalsByAge = rdd.map(lambda x: (x, 1))
```

With Key, Value RDDs, we can:
- reduceByKey()
- groupByKey()
- sortByKey()
- keys(), values()

```
rdd.reduceByKey(lambda x, y: x + y)
```

Set operations on Key, Value RDDs

With two lists of key, value RDDs, we can:

- join
- rightOuterJoin
- leftOuterJoin
- cogroup
- subtractByKey

Mapping just values

When mapping just values in a Key, Value RDD, it is more efficient to use:

- mapValues()
- flatMapValues()

as it maintains the original partitioning.

Persistence

```
result = input.map(lambda x: x * x)
result.persist().is_cached
print(result.count())
print(result.collect().mkString(","))
```

Hands-On

Download the Data

```
#remove existing copies of dataset from HDFS
hadoop fs -rm  /tmp/kddcup.data_10_percent.gz

wget http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz \
-O /tmp/kddcup.data_10_percent.gz
```

Load data in HDFS

```
hadoop fs -put /tmp/kddcup.data_10_percent.gz /tmp
hadoop fs -ls -h /tmp/kddcup.data_10_percent.gz

rm  /tmp/kddcup.data_10_percent.gz
```

First RDD

```
input_file = "hdfs:///tmp/kddcup.data_10_percent.gz"

raw_rdd = sc.textFile(input_file)
```

Retrieving version and configuration information

```
sc.version
sc.getConf.get("spark.home")
System.getenv().get("PYTHONPATH")
System.getenv().get("SPARK_HOME")
```

Count the number of rows in the dataset

```
print raw_rdd.count()
```

Inspect what the data looks like

```
print raw_rdd.take(5)
```

Spreading data across the cluster

```
a = range(100)

data = sc.parallelize(a)
```

Filtering lines in the data

```
normal_raw_rdd = raw_rdd.filter(lambda x: 'normal.' in x)
```

Count the filtered RDD

```
normal_count = normal_raw_rdd.count()

print normal_count
```

Importing local libraries

```
from pprint import pprint

csv_rdd = raw_rdd.map(lambda x: x.split(","))

head_rows = csv_rdd.take(5)

pprint(head_rows[0])
```

Using non-lambda functions in transformation

```
def parse_interaction(line):
    elems = line.split(",")
    tag = elems[41]
    return (tag, elems)

key_csv_rdd = raw_rdd.map(parse_interaction)
head_rows = key_csv_rdd.take(5)
pprint(head_rows[0]
```

Using collect() on the Spark driver

```
all_raw_rdd = raw_rdd.collect()
```

Extracting a smaller sample of the RDD

```
raw_rdd_sample = raw_rdd.sample(False, 0.1, 1234)
print raw_rdd_sample.count()
print raw_rdd.count()
```

Local sample for local processing

```
raw_data_local_sample = raw_rdd.takeSample(False, 1000, 1234)

normal_data_sample = [x.split(",") for x in raw_data_local_sample if "normal." in x]

normal_data_sample_size = len(normal_data_sample)

normal_ratio = normal_data_sample_size/1000.0

print normal_ratio
```

Subtracting on RDD from another

```
attack_raw_rdd = raw_rdd.subtract(normal_raw_rdd)

print attack_raw_rdd.count()
```

Listing a distinct list of protocols

```
protocols = csv_rdd.map(lambda x: x[1]).distinct()

print protocols.collect()
```

Listing a distinct list of services

```
services = csv_rdd.map(lambda x: x[2]).distinct()
print services.collect()
```

Cartesian product of two RDDs

```
product = protocols.cartesian(services).collect()

print len(product)
```

Filtering out the normal and attack events

```
normal_csv_data = csv_rdd.filter(lambda x: x[41]=="normal.")
attack_csv_data = csv_rdd.filter(lambda x: x[41]!="normal.")
```

Calculating total normal event duration and attack event duration

```
normal_duration_data = normal_csv_data.map(lambda x: int(x[0]))
total_normal_duration = normal_duration_data.reduce(lambda x, y: x + y)
print total_normal_duration

attack_duration_data = attack_csv_data.map(lambda x: int(x[0]))
total_attack_duration = attack_duration_data.reduce(lambda x, y: x + y)
print total_attack_duration
```

Calculating average duration of normal and attack events

```
normal_count = normal_duration_data.count()
attack_count = attack_duration_data.count()

print round(total_normal_duration/float(normal_count),3)
print round(total_attack_duration/float(attack_count),3)
```

Use aggegrate() function to calculate the average in a single scan

```
normal_sum_count = normal_duration_data.aggregate(
    (0,0), # the initial value
    (lambda acc, value: (acc[0] + value, acc[1] + 1)), # combine val/acc
    (lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1]))
)

print round(normal_sum_count[0]/float(normal_sum_count[1]),3)

attack_sum_count = attack_duration_data.aggregate(
    (0,0), # the initial value
    (lambda acc, value: (acc[0] + value, acc[1] + 1)), # combine value with acc
    (lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])) # combine accumulators
)

print round(attack_sum_count[0]/float(attack_sum_count[1]),3)
```

Creating a key, value RDD

```
key_value_rdd = csv_rdd.map(lambda x: (x[41], x))

print key_value_rdd.take(1)
```

Reduce by key and aggregate

```
key_value_duration = csv_rdd.map(lambda x: (x[41], float(x[0])))
durations_by_key = key_value_duration.reduceByKey(lambda x, y: x + y)

print durations_by_key.collect()
```

Count by key

```
counts_by_key = key_value_rdd.countByKey()
print counts_by_key
```

Combine by Key to get duration and attempts by type

```
sum_counts = key_value_duration.combineByKey(
    (lambda x: (x, 1)),
    # the initial value, with value x and count 1
    (lambda acc, value: (acc[0]+value, acc[1]+1)),
    # how to combine a pair value with the accumulator: sum value, and increment count
    (lambda acc1, acc2: (acc1[0]+acc2[0], acc1[1]+acc2[1]))
    # combine accumulators
)

print sum_counts.collectAsMap()
```

Sorted average duration by type

```
duration_means_by_type = sum_counts\
                         .map(lambda (key,value): (key,\
                         round(value[0]/value[1],3))).collectAsMap()

# Print them sorted
for tag in sorted(duration_means_by_type, key=duration_means_by_type.get, reverse=True):
    print tag, duration_means_by_type[tag]
```

Creating a Schema and creating a DataFrame

```
row_data = csv_rdd.map(lambda p: Row(
    duration=int(p[0]),
    protocol_type=p[1],
    service=p[2],
    flag=p[3],
    src_bytes=int(p[4]),
    dst_bytes=int(p[5])
    )
)

interactions_df = sqlContext.createDataFrame(row_data)
interactions_df.registerTempTable("interactions")
```

Quering with SQL

```
SELECT duration, dst_bytes FROM interactions
WHERE protocol_type = 'tcp'
AND duration > 1000
AND dst_bytes = 0
```

Printing the Schema

```
interactions_df.printSchema()
```

Counting events by protocol

```
interactions_df.select("protocol_type", "duration", "dst_bytes")\
               .groupBy("protocol_type").count().show()

interactions_df.select("protocol_type", "duration",\
                       "dst_bytes").filter(interactions_df.duration>1000)\
                       .filter(interactions_df.dst_bytes==0)\
                       .groupBy("protocol_type").count().show()
```

Modifying the labels

```
def get_label_type(label):
    if label!="normal.":
        return "attack"
    else:
        return "normal"

row_labeled_data = csv_rdd.map(lambda p: Row(
    duration=int(p[0]),
    protocol_type=p[1],
    service=p[2],
    flag=p[3],
    src_bytes=int(p[4]),
    dst_bytes=int(p[5]),
    label=get_label_type(p[41])
    )
)
interactions_labeled_df = sqlContext.createDataFrame(row_labeled_data)
```

Counting by labels

```
interactions_labeled_df.select("label", "protocol_type")\
                       .groupBy("label", "protocol_type").count().show()
```

Group by label and protocol

```
interactions_labeled_df.select("label", "protocol_type")\
                       .groupBy("label", "protocol_type").count().show()

interactions_labeled_df.select("label", "protocol_type", "dst_bytes")\
                       .groupBy("label", "protocol_type",\
                       interactions_labeled_df.dst_bytes==0).count().show()
```

Questions

More resources

- More tutorials: http://hortonworks.com/tutorials
- Hortonworks gallery: http://hortonworks-gallery.github.io

My co-ordinates

- Twitter: @saptak
- Website: http://saptak.in