



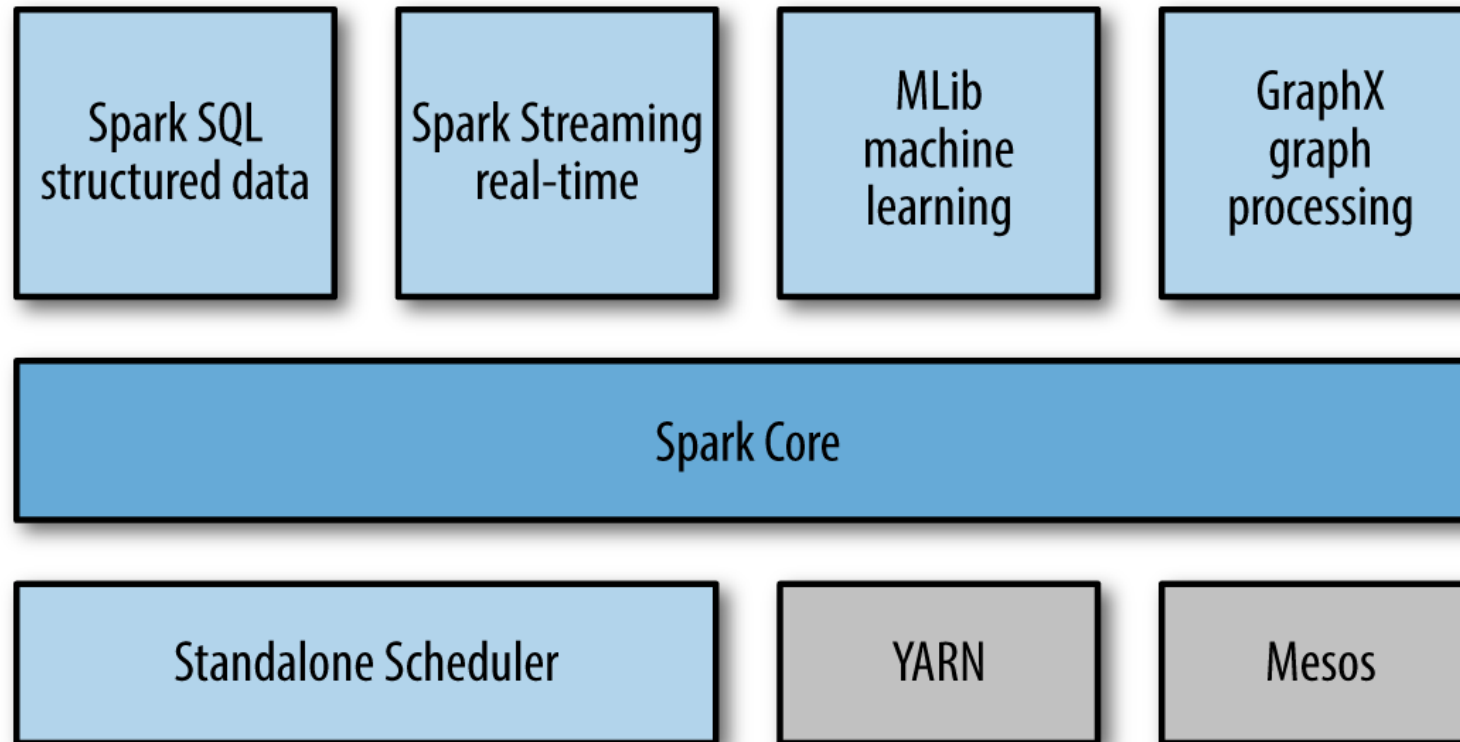
Apache Spark

- ▶ Apache Spark is one of the hottest frameworks in data science
- ▶ It realizes the potential of bringing together both Big Data and machine learning
- ▶ Spark is fast (up to 100x faster than traditional Hadoop MapReduce) due to in-memory operation
- ▶ It offers robust, distributed, fault-tolerant data objects (called RDDs)
- ▶ It integrates beautifully with the world of machine learning and graph analytics through supplementary packages like MLlib and GraphX
- ▶ Spark is implemented on Hadoop/HDFS and written mostly in Scala, a functional programming language which runs on the JVM
- ▶ Spark provides a wonderful Python API called PySpark



The Spark Stack

- ▶ Apache Spark is an open-source cluster-computing framework



PySpark

- ▶ PySpark is the Python API for Spark
- ▶ Public classes:
 - ▶ SparkContext: Main entry point for Spark functionality
 - ▶ RDD: A Resilient Distributed Dataset (RDD), the basic abstraction in Spark
 - ▶ Broadcast: A broadcast variable that gets reused across tasks
 - ▶ Accumulator: An “add-only” shared variable that tasks can only add values to
 - ▶ SparkConf: For configuring Spark
 - ▶ SparkFiles: Access files shipped with jobs
 - ▶ StorageLevel: Finer-grained cache persistence levels
- ▶ <http://spark.apache.org/docs/2.1.0/api/python/pyspark.html>

Download Apache Spark

- ▶ Visit <http://spark.apache.org/downloads.html>

Download Apache Spark™

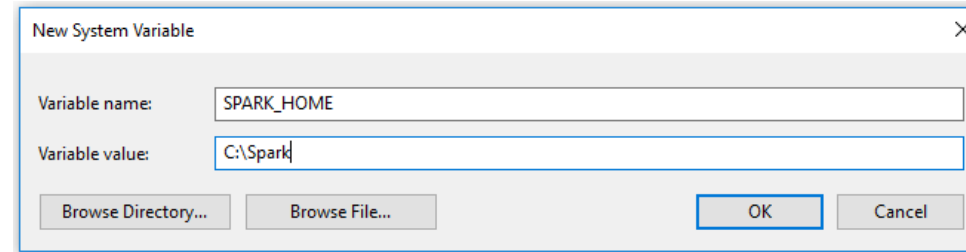
1. Choose a Spark release: ▼
2. Choose a package type: ▼
3. Download Spark: [spark-2.2.2-bin-hadoop2.7.tgz](#)
4. Verify this release using the [2.2.2 signatures and checksums](#) and [project release KEYS](#).

Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build [with Scala 2.10 support](#).

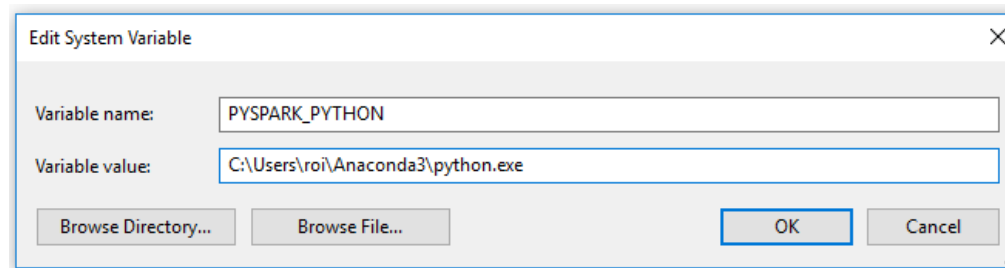
- ▶ Select Spark release 2.2.2 (Jul 02 2018)
 - ▶ Version 2.3.1 has some unresolved dependency conflict issues
- ▶ Select the package type of “Pre-built for Hadoop 2.7 and later,” and click the tar file
- ▶ This will download a compressed TAR file, called spark-2.2.2-bin-hadoop2.7.tgz
- ▶ Unzip this file into a new directory such as C:\Spark
 - ▶ The directory should be with no spaces

Install Apache Spark

- ▶ Set the SPARK_HOME environment variables to C:\Spark

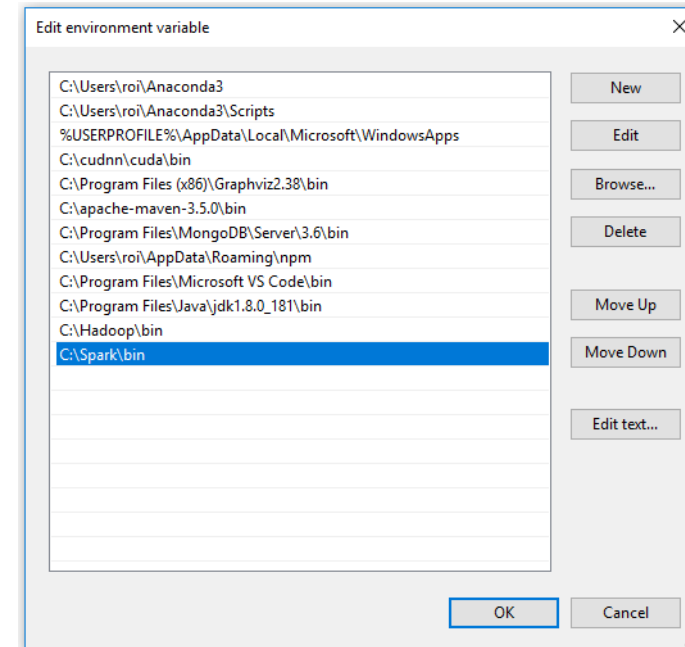


- ▶ Set the PYSPARK_PYTHON environment variable to the location of python.exe:



Install Apache Spark

- ▶ Add C:\Spark\bin to the path variable:



- ▶ In addition, you need to create the folder C:\tmp\hive
- ▶ You also need to set full permission on this folder using the command:

```
winutils chmod 777 C:\tmp\hive
```

- ```
C:\WINDOWS\system32>pyspark
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(
newLevel).
2018-11-14 07:27:55,673 WARN metastore.ObjectStore: Failed to get database global_
temp, returning NoSuchObjectException
Welcome to

 /--\ /--\ /--\ /--\ /--\
 / V _/ V _/ V _/ V _\'
 /___/\./___/\./___/\./___/\./___\ version 2.2.2
 /__\

Using Python version 3.6.4 (default, Jan 16 2018 10:22:32)
SparkSession available as 'spark'.
>>>
```



# Testing

---

- ▶ You can also try out typing a few lines from the official [Quick Start Guide](#):

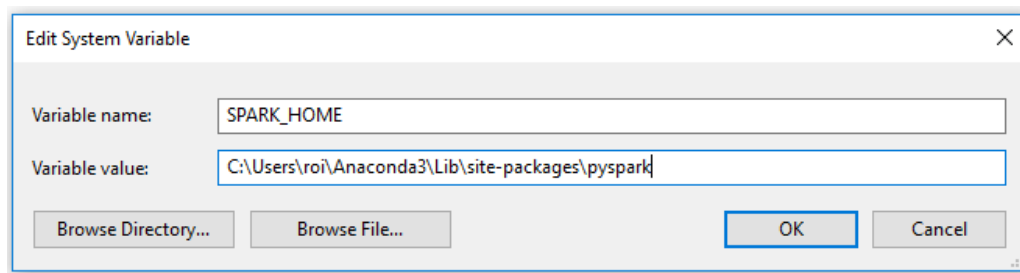
```
>>> textFile = sc.textFile("file:///C:/Spark/README.md")
>>> textFile.count()
103
```

- ▶ To exit Spark type quit()

# Installing PySpark

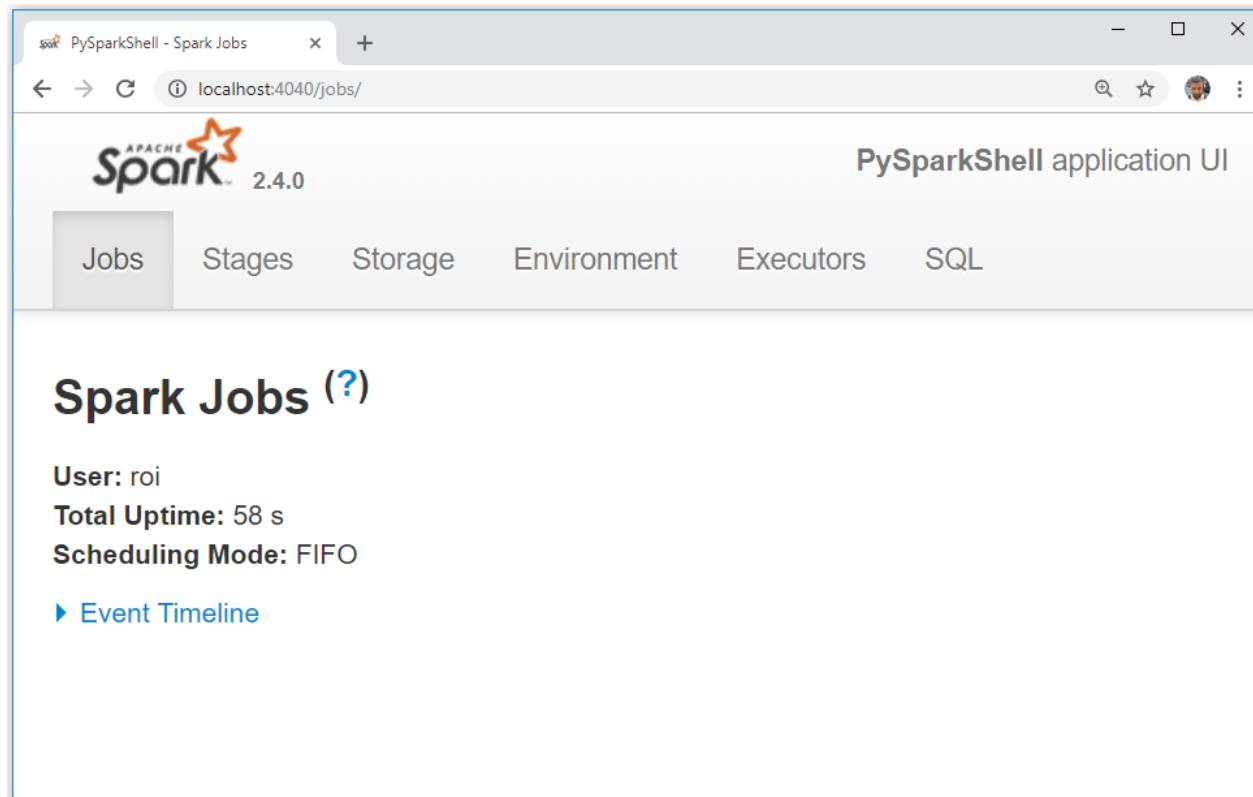
## ► pip install --user pyspark==2.3.2

```
C:\Users\roi>pip install pyspark
Collecting pyspark
 Downloading https://files.pythonhosted.org/packages/88/01/a37e827c2d80c6a754e40e99b9826d978b55254cc6c6672b5b08f2e18a7f/pyspark-2.4.0.tar.gz (213.4MB)
 100% |████████████████████| 213.4MB 97kB/s
Collecting py4j==0.10.7 (from pyspark)
 Downloading https://files.pythonhosted.org/packages/e3/53/c737818eb9a7dc32a7cd4f1396e787bd94200c3997c72c1dbe028587bd76/py4j-0.10.7-py2.py3-none-any.whl (197kB)
 100% |████████████████████| 204kB 1.3MB/s
Building wheels for collected packages: pyspark
 Running setup.py bdist_wheel for pyspark ... done
 Stored in directory: C:\Users\roi\AppData\Local\pip\Cache\wheels\cd\54\c2\abfcc942eddeaa7101228ebd6127a30dbdf903c72db4235b23
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.7 pyspark-2.4.0
```



# Spark UI

- ▶ At this point you should also be able to access the Spark UI from your favorite browser at `http://localhost:4040`



# SparkContext

- ▶ This class is the main entry point for Spark functionality
- ▶ A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster
- ▶ Only one SparkContext may be active per JVM
- ▶ You must stop() the active SparkContext before creating a new one
- ▶ Create a new Jupyter notebook and type

```
import pyspark
```

```
Create a Spark context
sc = pyspark.SparkContext(appName='MyFirstSparkApp')
```

# Resilient Distributed DataSets (RDDs)

---

- ▶ Spark revolves around the concept of a *resilient distributed dataset* (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel
- ▶ There are two ways to create RDDs: *parallelizing* an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat

# Parallelized Collections

- ▶ Parallelized collections are created by calling SparkContext's `parallelize()` method on an existing iterable or collection in your program
- ▶ The elements of the collection are copied to form a distributed dataset that can be operated on in parallel
- ▶ For example, to create a parallelized collection holding the numbers 1 to 5:+

```
data = [1, 2, 3, 4, 5]
dist_data = sc.parallelize(data)
```

- ▶ Once created, the distributed dataset (`dist_data`) can be operated on in parallel
- ▶ For example, we can call `dist_data.reduce(lambda a, b: a + b)` to add up the elements of the list

```
dist_data.reduce(lambda a, b: a + b)
```

15

# Partitions

---

- ▶ Spark manages data using partitions that helps parallelize distributed data processing with minimal network traffic for sending data between executors
- ▶ You can check the number of partitions using the method `getNumPartitions()` of RDD

```
dist_data.getNumPartitions()
```

8

- ▶ Spark can only run 1 concurrent task for every partition of an RDD, up to the number of cores in your cluster
- ▶ So if you have a cluster with 50 cores, you want your RDDs to at least have 50 partitions (and probably x2-3 times that)

# RDD Operations

---

- ▶ RDDs support two types of operations:
  - ▶ *Transformations*, which create a new dataset from an existing one
    - ▶ For example, map is a transformation that passes each dataset element through a function and returns a new RDD representing the results
  - ▶ *Actions*, which return a value to the driver program after running a computation on the dataset
- ▶ For example, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program
- ▶ The transformations are only computed when an action requires a result to be returned to the driver program



# RDD Operations

- ▶ For example, the following script computes  $\pi$  by generating random points in the unit square ((0,0) to (1,1)) and checking how many fall in the unit circle.

```
pi.py
import pyspark
import random

sc = pyspark.SparkContext(appName='pi')
num_samples = 10000000

def inside(p):
 x, y = random.random(), random.random()
 return x ** 2 + y ** 2 < 1

count = sc.parallelize(range(0, num_samples)).filter(inside).count()

pi = 4 * count / num_samples
print(pi)
```

3.1418336

# Launching PySpark on YARN

---

- ▶ There are two deploy modes that can be used to launch Spark applications on YARN:
  - ▶ In cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application.
  - ▶ In client mode, the driver runs in the client process, and the application master is only used for requesting resources from YARN.

- ▶ To launch a Spark script in cluster mode:

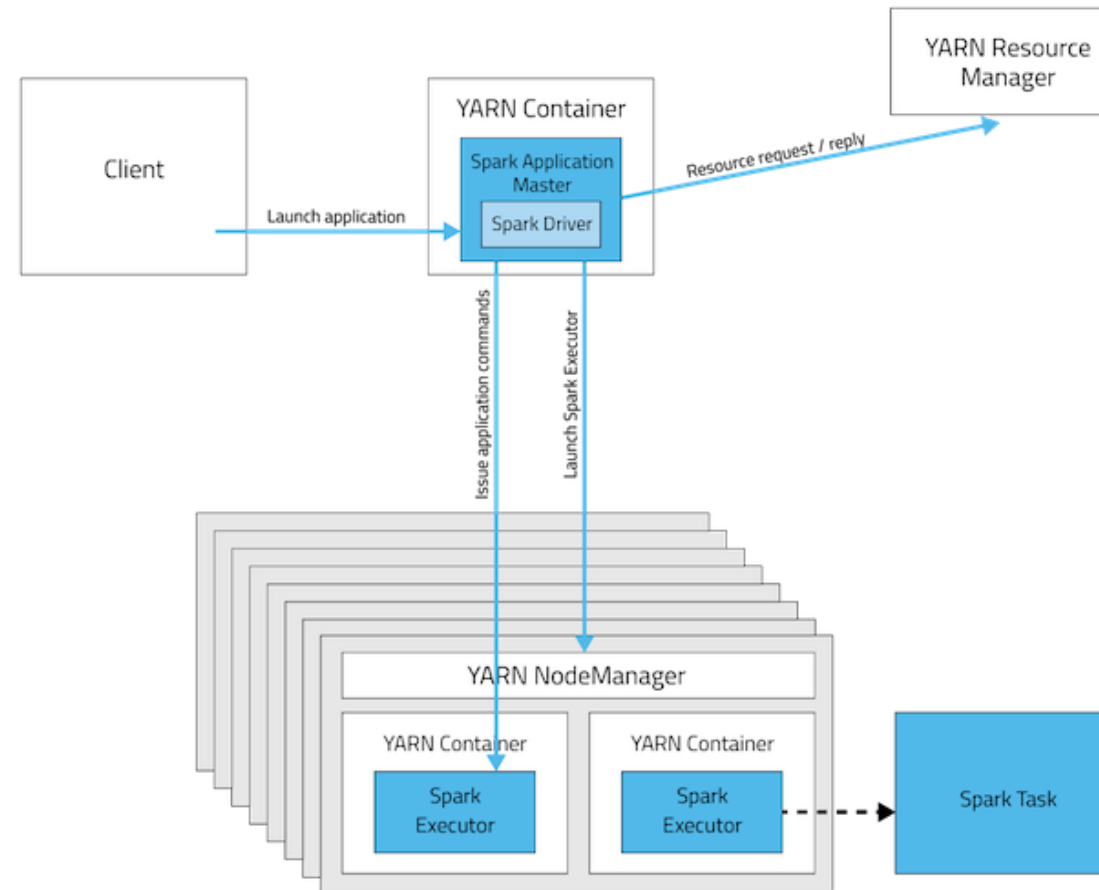
```
spark-submit --master yarn --deploy-mode cluster [path to python script]
```

- ▶ To launch a Spark script in client mode:

```
spark-submit --master yarn --deploy-mode client [path to python script]
```

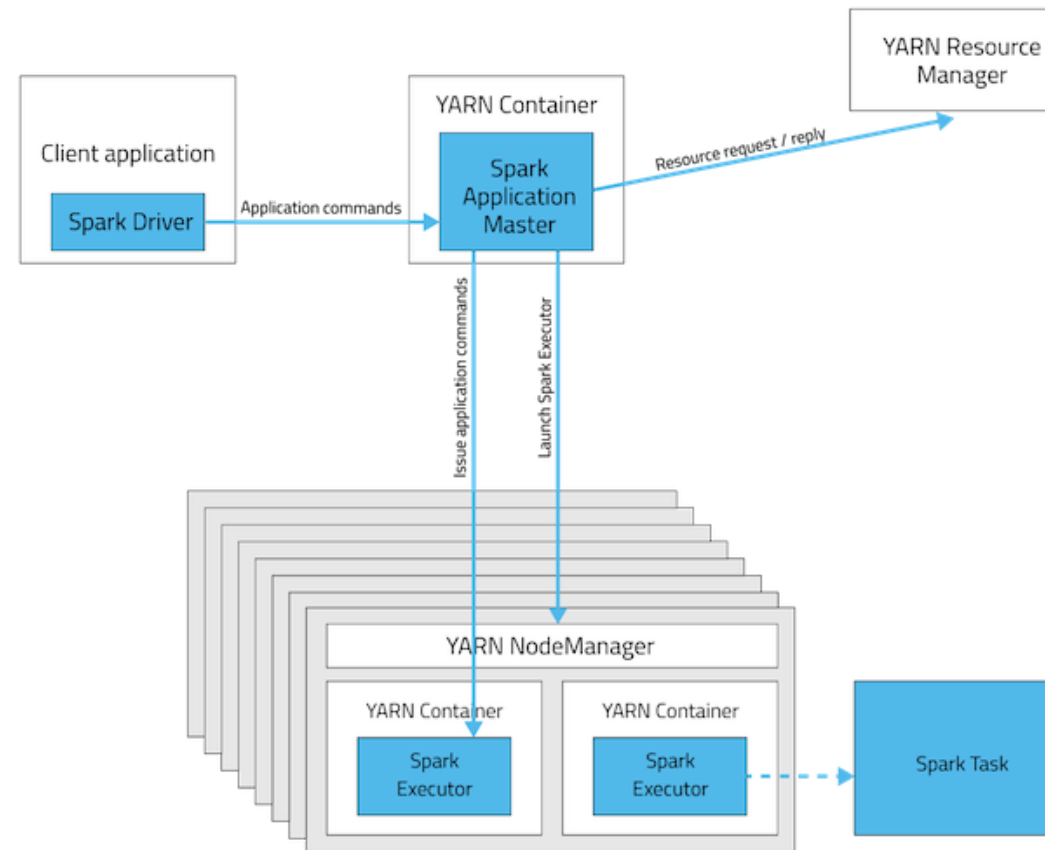
# Cluster Deployment Mode

- ▶ In client mode, the Spark driver runs on the host where the job is submitted



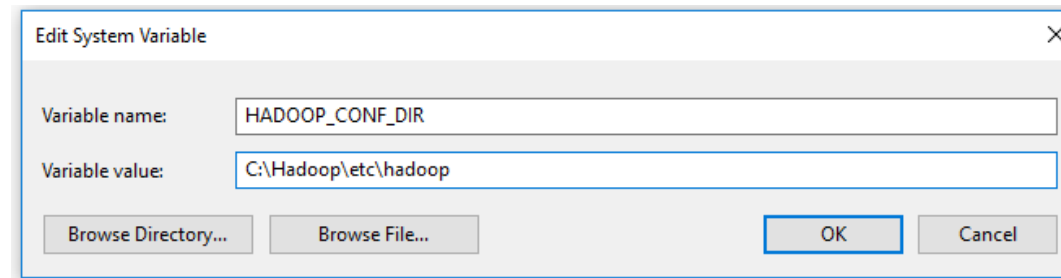
# Client Deployment Mode

- ▶ In client mode, the Spark driver runs on the host where the job is submitted



# Launching Spark on YARN

- ▶ First make sure that the HDFS and YARN daemons are running
- ▶ Set the environment variable HADOOP\_CONF\_DIR to the directory which contains the configuration files for the Hadoop cluster (C:\Hadoop\etc\hadoop)



# Launching Spark on YARN

---

- ▶ To run the pi.py script in yarn:
- ▶ Start command prompt (under administrator)
- ▶ Type the following command

```
spark-submit --master yarn --deploy-mode client C:\PyCharmProjects\PySpark\pi.py
```

# Launching Spark on YARN

- ▶ You should see the result in the console:

```
Administrator: Command Prompt
2018-11-14 07:33:19,325 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, ROI-COMP, executor 2, partition 1, PROCESS_LOCAL, 4822 bytes)
2018-11-14 07:33:19,661 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on ROI-COMP:2578 (size: 3.8 KB, free: 366.3 MB)
2018-11-14 07:33:19,662 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on ROI-COMP:2538 (size: 3.8 KB, free: 366.3 MB)
2018-11-14 07:33:24,200 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 4873 ms on ROI-COMP (executor 2) (1/2)
2018-11-14 07:33:24,215 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 4917 ms on ROI-COMP (executor 1) (2/2)
2018-11-14 07:33:24,217 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool
2018-11-14 07:33:24,218 INFO scheduler.DAGScheduler: ResultStage 0 (count at C:/PyCharmProjects/PySpark/pi.py:12) finished in 4.921 s
2018-11-14 07:33:24,222 INFO scheduler.DAGScheduler: Job 0 finished: count at C:/PyCharmProjects/PySpark/pi.py:12, took 5.102513 s
3.1418668
2018-11-14 07:33:25,282 INFO spark.SparkContext: Invoking stop() from shutdown hook
2018-11-14 07:33:25,288 INFO server.AbstractConnector: Stopped Spark@669ebcb6{HTTP/1.1,[http/1.1]}{0.0.0.0:4041}
```

# Spark UI

- ▶ In the Spark UI you can see the status of the submitted job:

## Details for Job 0

Status: SUCCEEDED

Completed Stages: 1

- ▶ Event Timeline
- ▶ DAG Visualization

### Completed Stages (1)

| Stage Id ▾ | Description                                       | Submitted           | Duration | Tasks:<br>Succeeded/Total | Input  | Output | Shuffle<br>Read | Shuffle<br>Write |
|------------|---------------------------------------------------|---------------------|----------|---------------------------|--------|--------|-----------------|------------------|
| 0          | <a href="#">count at &lt;stdin&gt;:1</a> +details | 2018/11/14 07:29:59 | 1 s      | 2/2                       | 5.6 KB |        |                 |                  |



## Exercise

---

- ▶ Write a script that reads a text file and prints all the unique words in the file in alphabetical order

# SparkML

---

- ▶ MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy.
- ▶ At a high level, it provides tools such as:
  - ▶ ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
  - ▶ Featurization: feature extraction, transformation, dimensionality reduction, and selection
  - ▶ Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
  - ▶ Persistence: saving and load algorithms, models, and Pipelines
  - ▶ Utilities: linear algebra, statistics, data handling, etc.

# SparkML

- ▶ Example for a script in SparkML  
(C:\Spark\examples\src\main\python\ml\kmeans\_example.py)

```
from pyspark.ml.clustering import KMeans
spark = SparkSession.builder.appName("KMeansExample").getOrCreate()

Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

Evaluate clustering by computing Within Set Sum of Squared Errors.
wssse = model.computeCost(dataset)
print("Within Set Sum of Squared Errors = " + str(wssse))

Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
 print(center)

spark.stop()
```

# SparkML

- ▶ When you perform a yarn execution, Spark is looking for the data files on HDFS
- ▶ Thus, first you need to upload the data file to HDFS
- ▶ First create a directory /user/%USERNAME%/data/mllib on HDFS:

```
hdfs dfs -mkdir -p /user/%USERNAME%/data/mllib
```

```
C:\Users\roi>hdfs dfs -mkdir -p /user/%USERNAME%/data/mllib

C:\Users\roi>hdfs dfs -ls /user/roi/data
Found 1 items
drwxr-xr-x - roi supergroup 0 2018-07-30 01:06 /user/roi/data/mllib
```

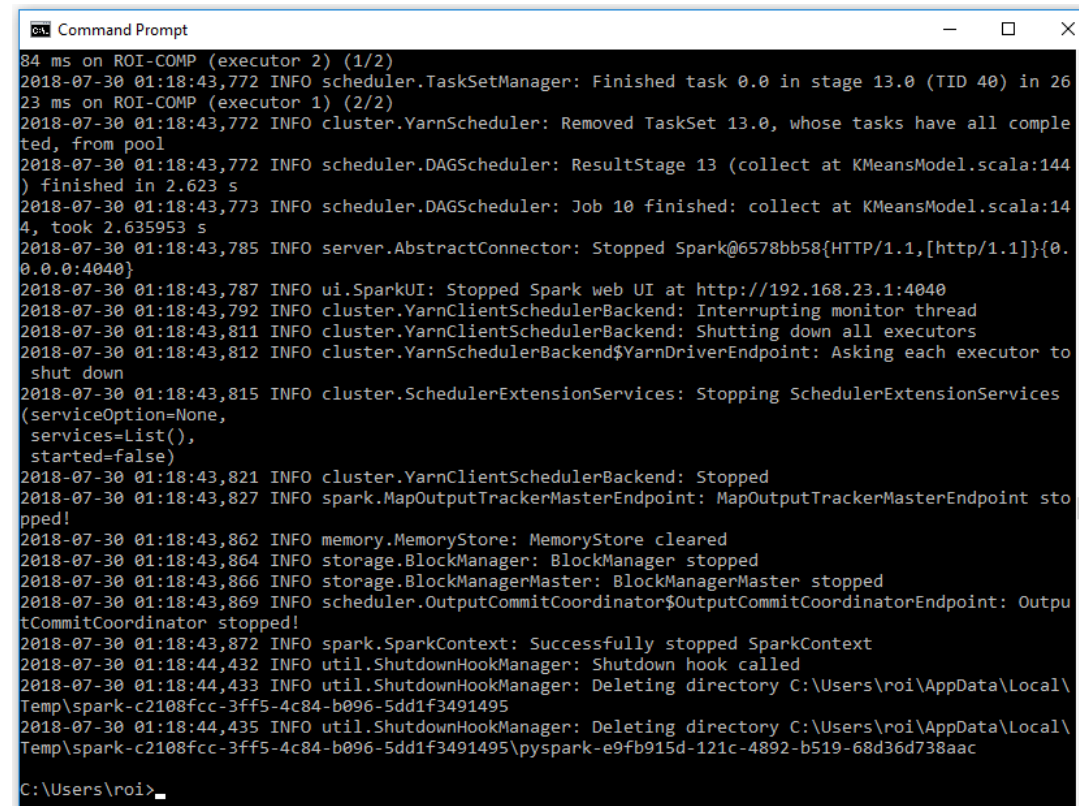
- ▶ Then copy the data file from C:\Spark\data\mllib to /user/%USERNAME%/data/mllib

```
hdfs dfs -put file:///C:/Spark/data/mllib/kmeans_data.txt /user/%USERNAME%/data/mllib
```

# SparkML

- ▶ Type the following command:

```
spark-submit --master yarn --deploy-mode client
C:\Spark\examples\src\main\python\mllib\k_means_example.py
```



```
Command Prompt
84 ms on ROI-COMP (executor 2) (1/2)
2018-07-30 01:18:43,772 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 13.0 (TID 40) in 26
23 ms on ROI-COMP (executor 1) (2/2)
2018-07-30 01:18:43,772 INFO cluster.YarnScheduler: Removed TaskSet 13.0, whose tasks have all comple
ted, from pool
2018-07-30 01:18:43,772 INFO scheduler.DAGScheduler: ResultStage 13 (collect at KMeansModel.scala:144
) finished in 2.623 s
2018-07-30 01:18:43,773 INFO scheduler.DAGScheduler: Job 10 finished: collect at KMeansModel.scala:14
4, took 2.635953 s
2018-07-30 01:18:43,785 INFO server.AbstractConnector: Stopped Spark@6578bb58{HTTP/1.1,[http/1.1]}{0.
0.0:4040}
2018-07-30 01:18:43,787 INFO ui.SparkUI: Stopped Spark web UI at http://192.168.23.1:4040
2018-07-30 01:18:43,792 INFO cluster.YarnClientSchedulerBackend: Interrupting monitor thread
2018-07-30 01:18:43,811 INFO cluster.YarnClientSchedulerBackend: Shutting down all executors
2018-07-30 01:18:43,812 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to
shut down
2018-07-30 01:18:43,815 INFO cluster.SchedulerExtensionServices: Stopping SchedulerExtensionServices
(serviceOption=None,
services=List(),
started=false)
2018-07-30 01:18:43,821 INFO cluster.YarnClientSchedulerBackend: Stopped
2018-07-30 01:18:43,827 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint sto
pped!
2018-07-30 01:18:43,862 INFO memory.MemoryStore: MemoryStore cleared
2018-07-30 01:18:43,864 INFO storage.BlockManager: BlockManager stopped
2018-07-30 01:18:43,866 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2018-07-30 01:18:43,869 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: Outpu
tCommitCoordinator stopped!
2018-07-30 01:18:43,872 INFO spark.SparkContext: Successfully stopped SparkContext
2018-07-30 01:18:44,432 INFO util.ShutdownHookManager: Shutdown hook called
2018-07-30 01:18:44,433 INFO util.ShutdownHookManager: Deleting directory C:\Users\roi\AppData\Local\
Temp\spark-c2108fcc-3ff5-4c84-b096-5dd1f3491495
2018-07-30 01:18:44,435 INFO util.ShutdownHookManager: Deleting directory C:\Users\roi\AppData\Local\
Temp\spark-c2108fcc-3ff5-4c84-b096-5dd1f3491495\pyspark-e9fb915d-121c-4892-b519-68d36d738aac
C:\Users\roi>
```

► Type the following command:

```
2018-07-30 01:18:35,380 INFO scheduler.DAGScheduler: Job 5 finished: reduce at
C:/Spark/examples/src/main/python/mllib/k_means_example.py:46, took 0.555797 s
Within Set Sum of Squared Error = 0.6928203230275529
2018-07-30 01:18:36,529 INFO output.FileOutputCommitter: File Output Committer Algorithm
version is 1
2018-07-30 01:18:36,557 INFO spark.SparkContext: Starting job: saveAsTextFile at
KMeansModel.scala:128
```