Module 10

Introduction to Spark

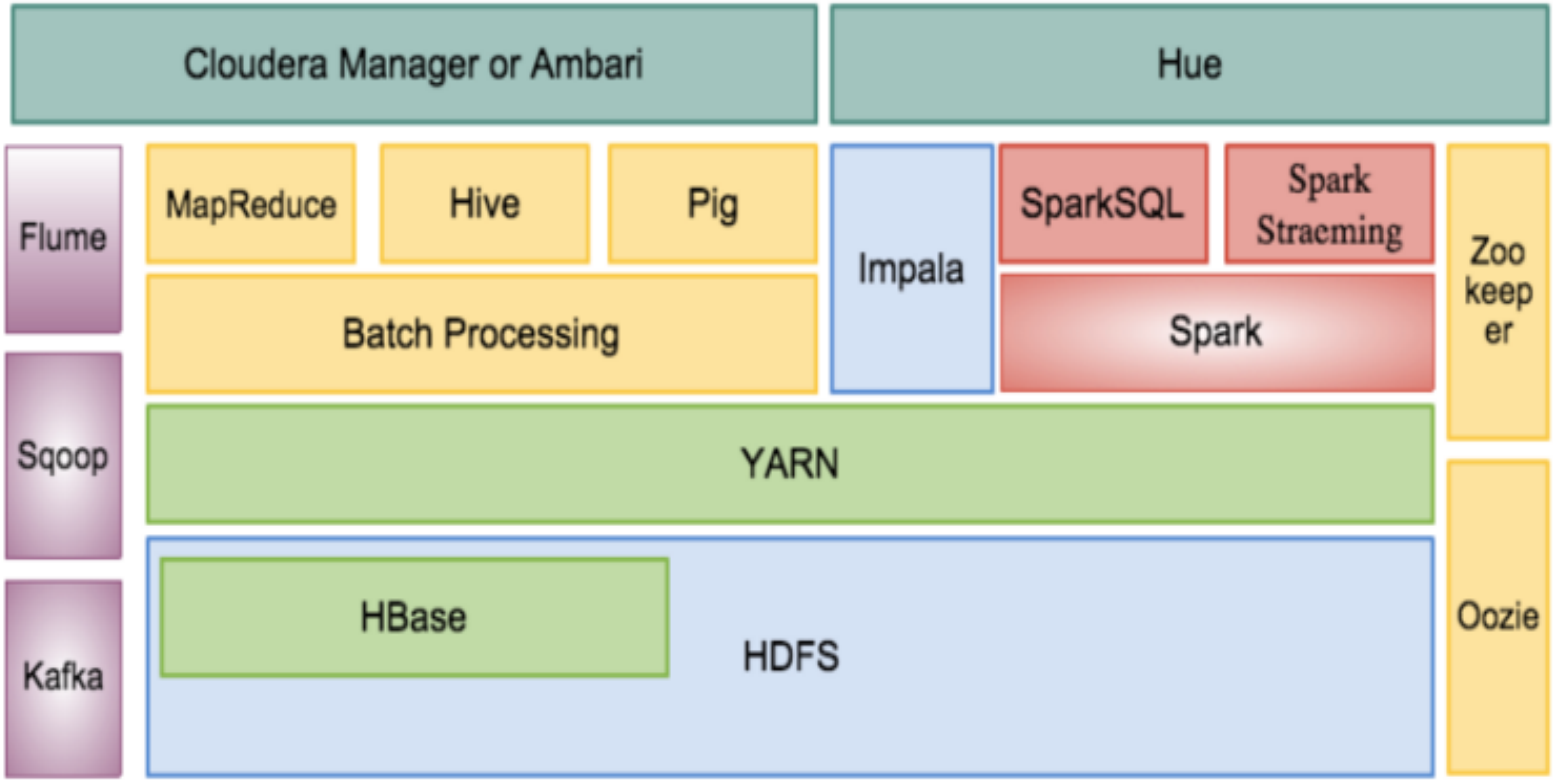
Thanachart Numnonda, Executive Director, IMC Institute

Mr.Aekanun Thongtae, Big Data Consultant, IMC Institute

Thanisa Numnonda, Faculty of Information Technology,

King Mongkut's Institute of Technology Ladkrabang

Hadoop Ecosystem



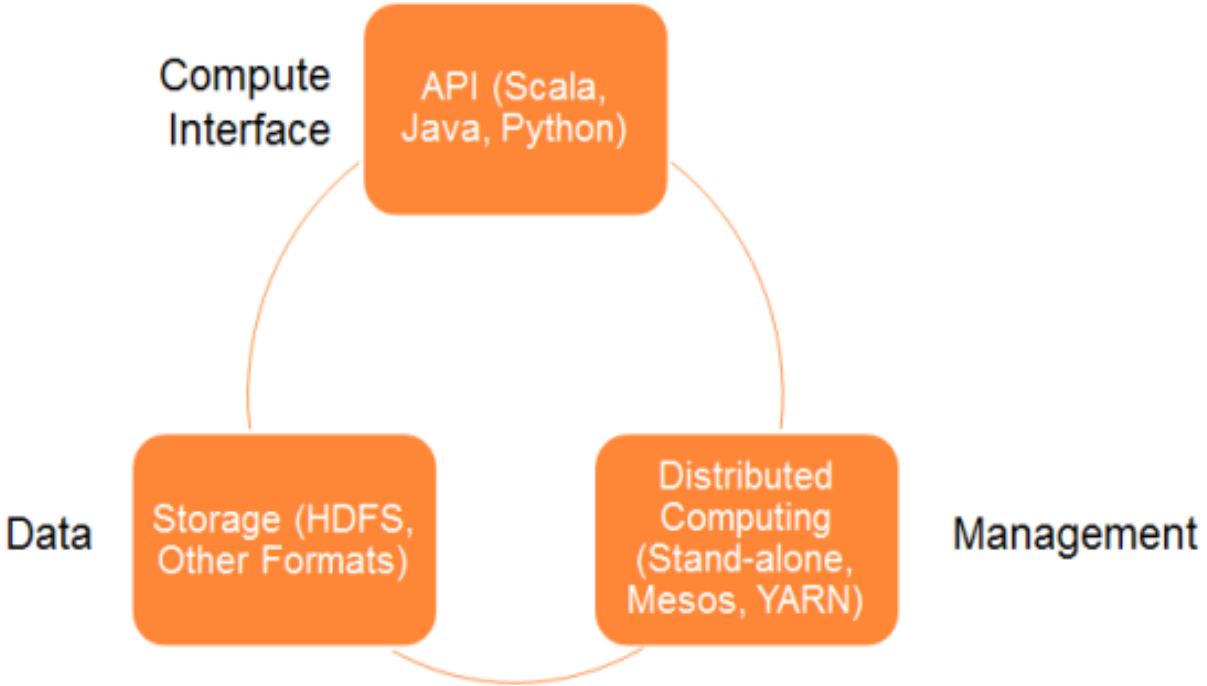
Introduction

A fast and general engine for large scale data processing

An open source big data processing framework built around speed, ease of use, and sophisticated analytics. Spark enables applications in Hadoop clusters to run up to 100 times faster in memory and 10 times faster even when running on disk.

What is Spark?

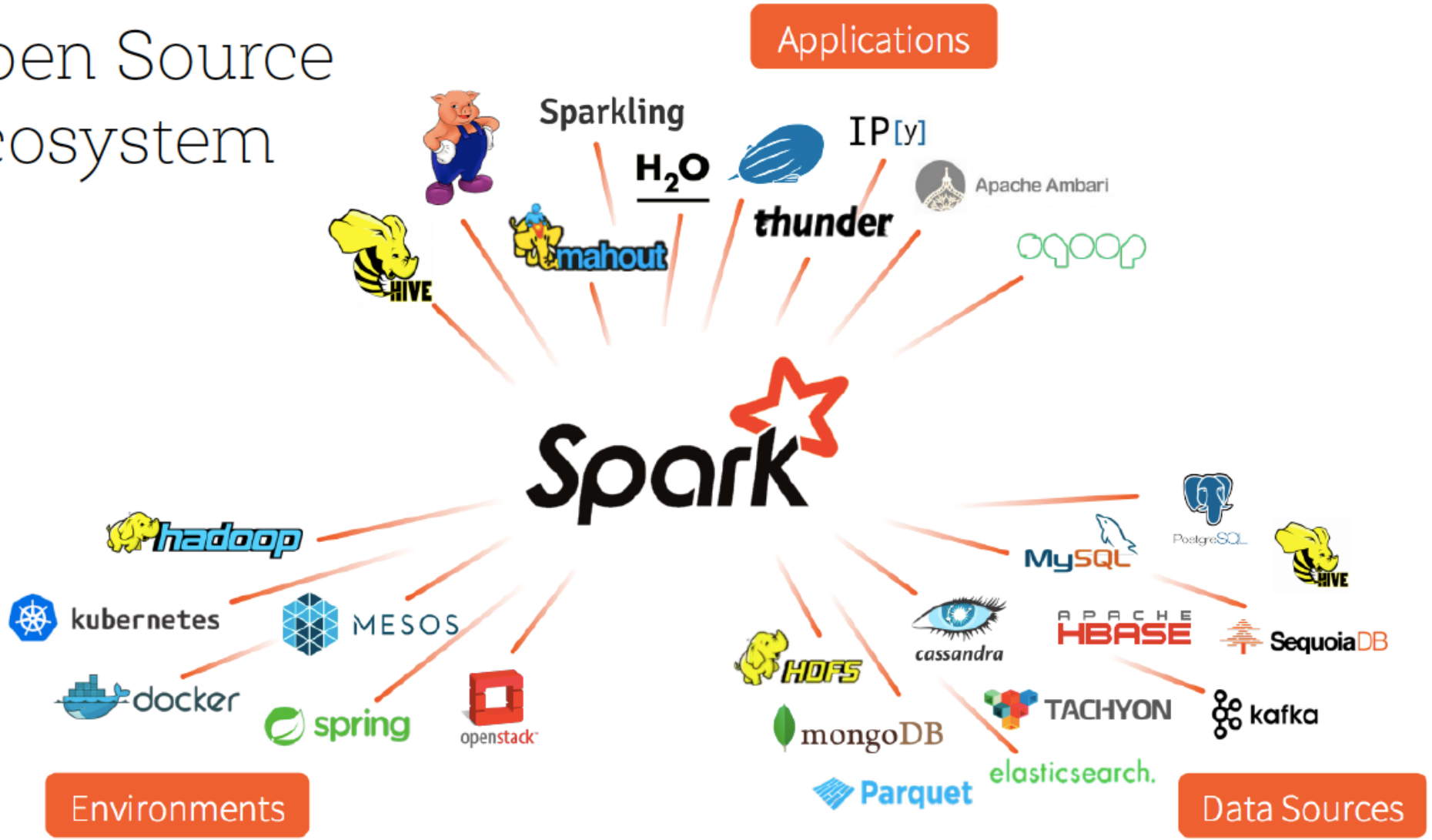
- Framework for distributed processing.
- In-memory, fault tolerant data structures
- Flexible APIs in Scala, Java, Python, SQL, R
- Open source



Why Spark?

- Handle Petabytes of data
- Significant faster than MapReduce
- Simple and intutive APIs
- General framework
  - Runs anywhere
  - Handles (most) any I/O
  - Interoperable libraries for specific use-cases

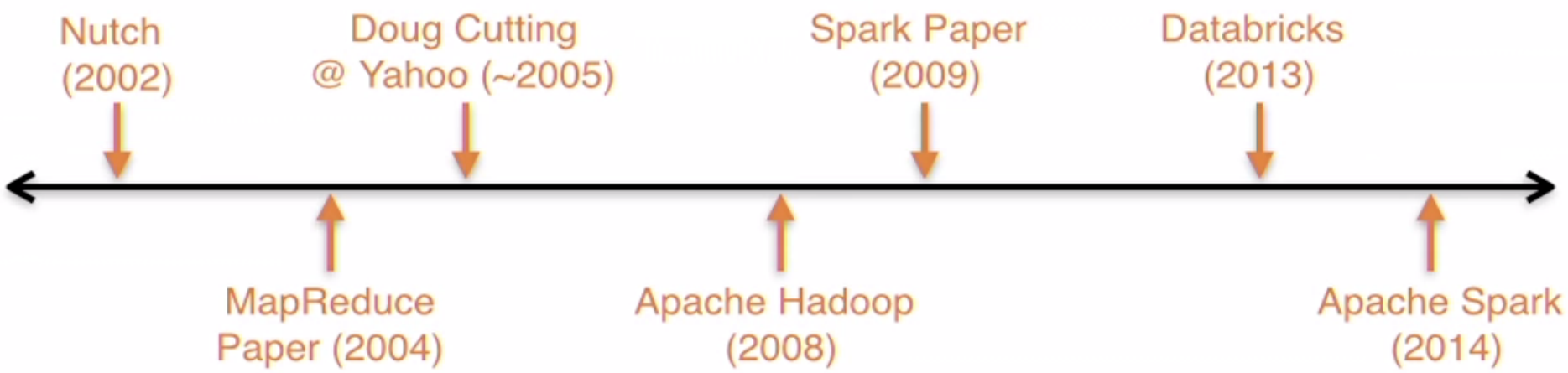
Open Source Ecosystem

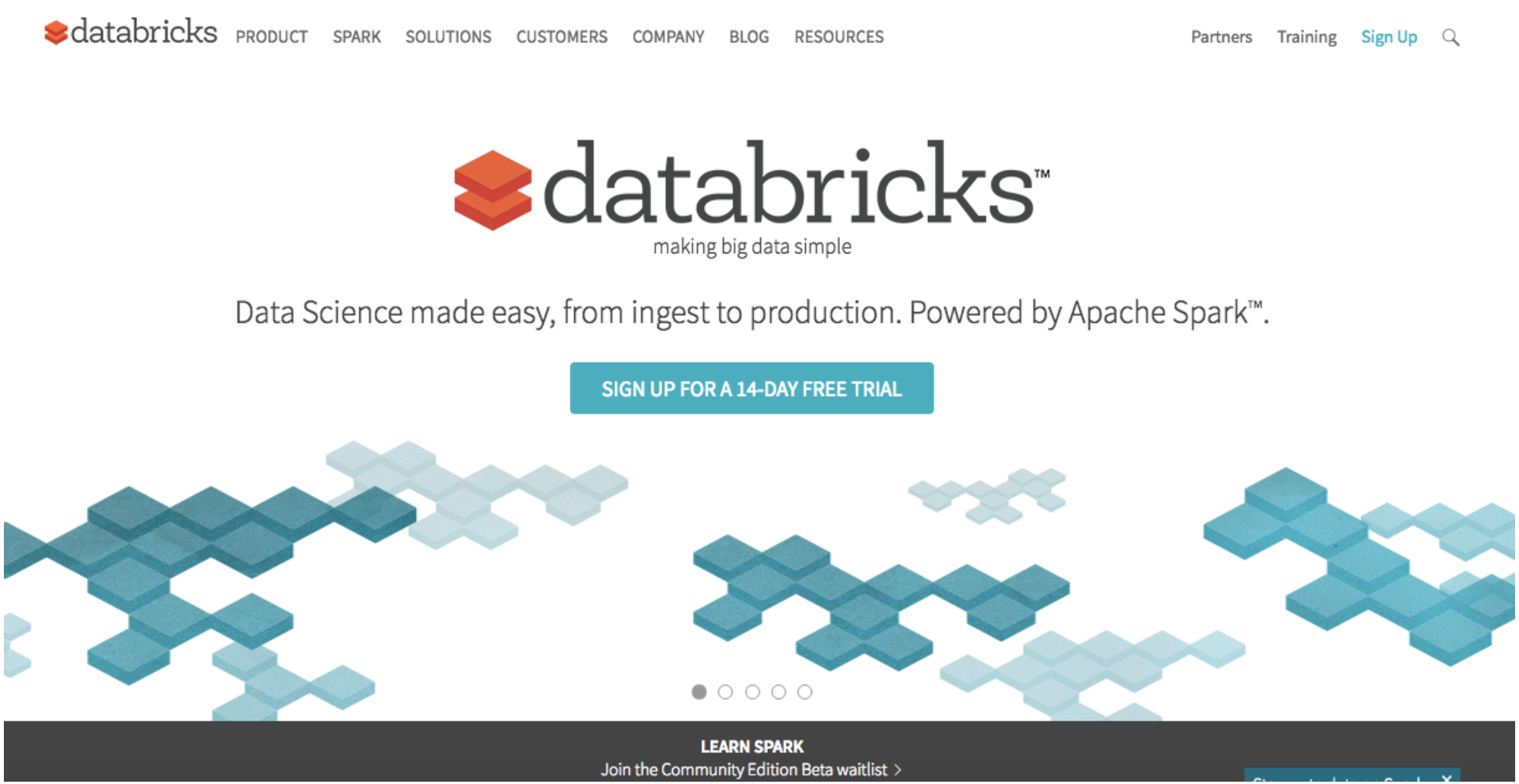


Source: Jump start into Apache Spark and Databricks

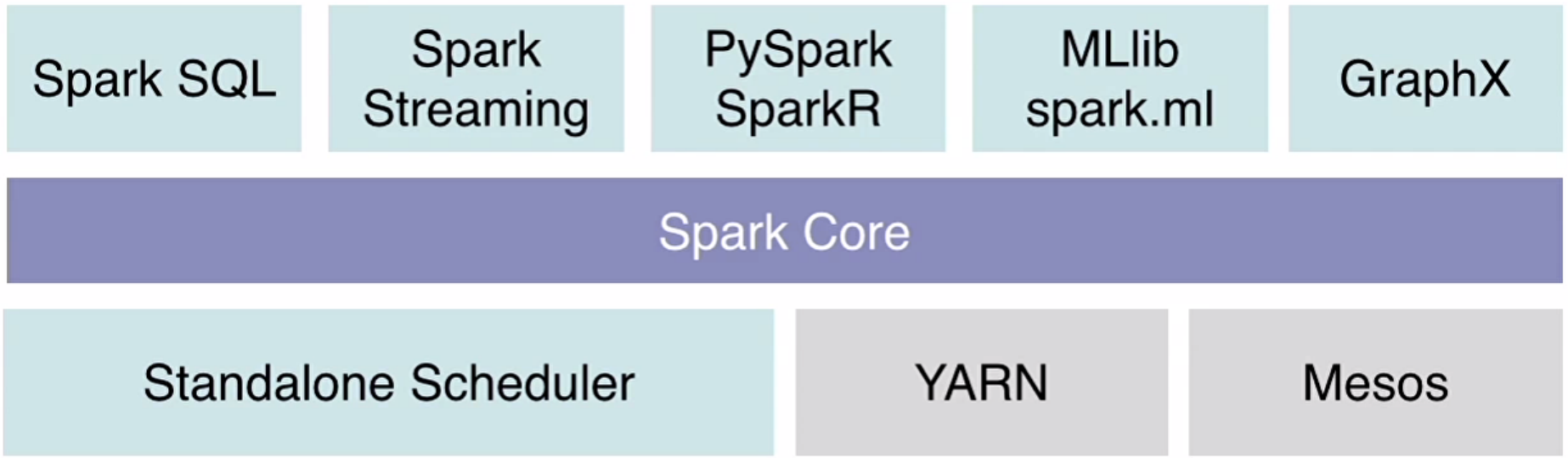
Spark: History

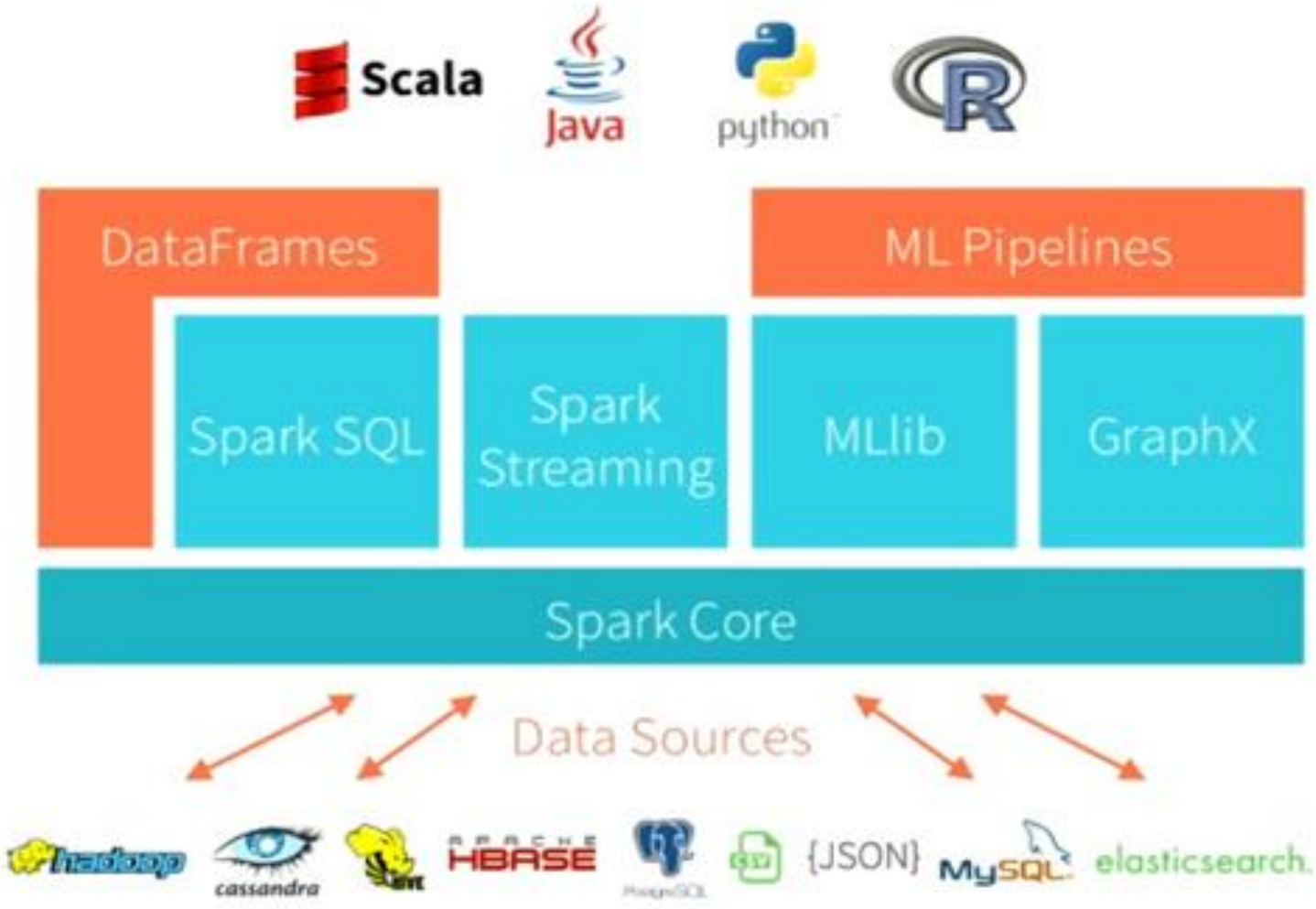
- Founded by AMPlab, UC Berkeley
- Created by Matei Zaharia (PhD Thesis)
- Maintained by Apache Software Foundation
- Commercial support by Databricks



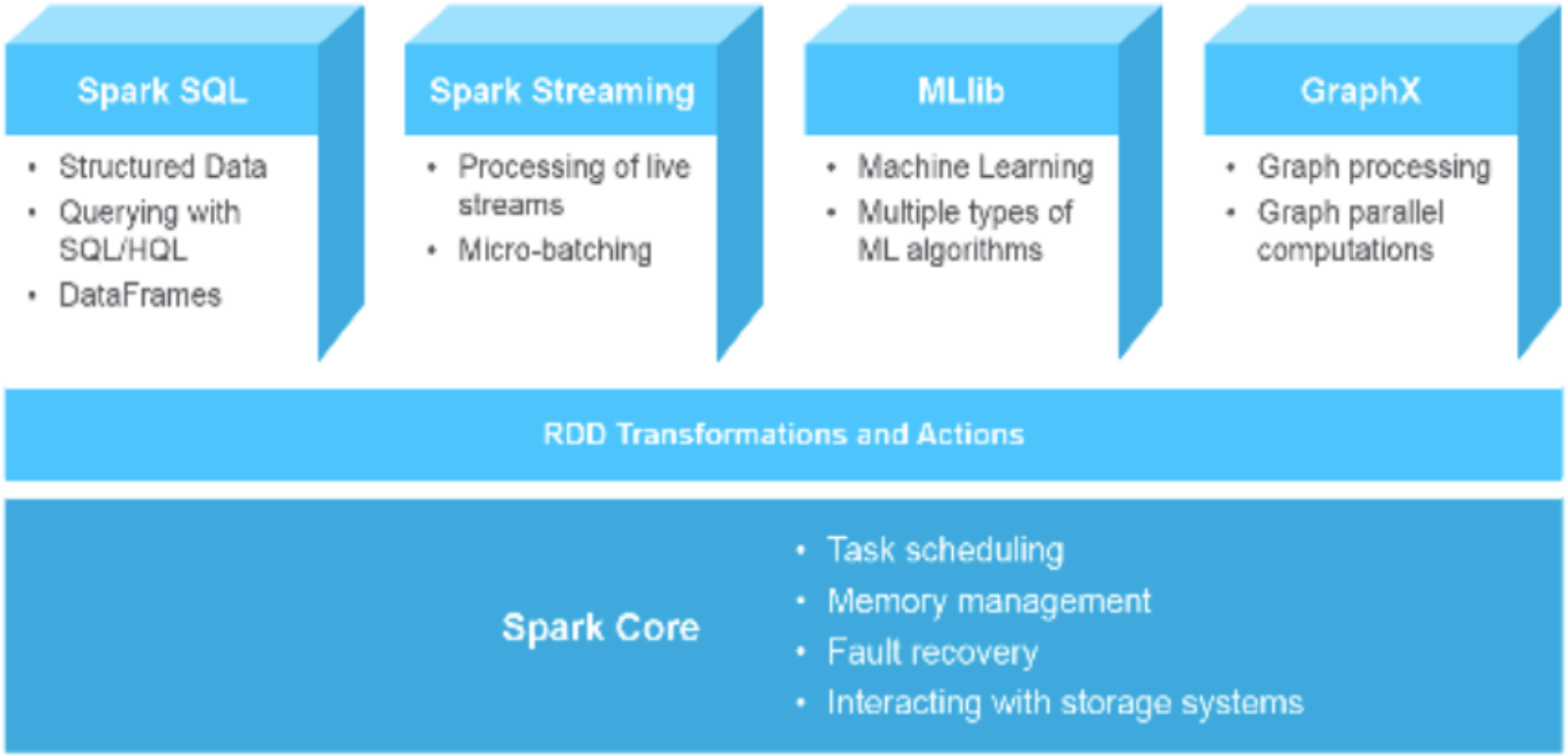

databricks
PRODUCT
SPARK
SOLUTIONS
CUSTOMERS
COMPANY
BLOG
RESOURCES
Partners
Training
Sign Up
databricks™
making big data simple
Data Science made easy, from ingest to production. Powered by Apache Spark™.
SIGN UP FOR A 14-DAY FREE TRIAL
LEARN SPARK
Join the Community Edition Beta waitlist >

Spark Platform

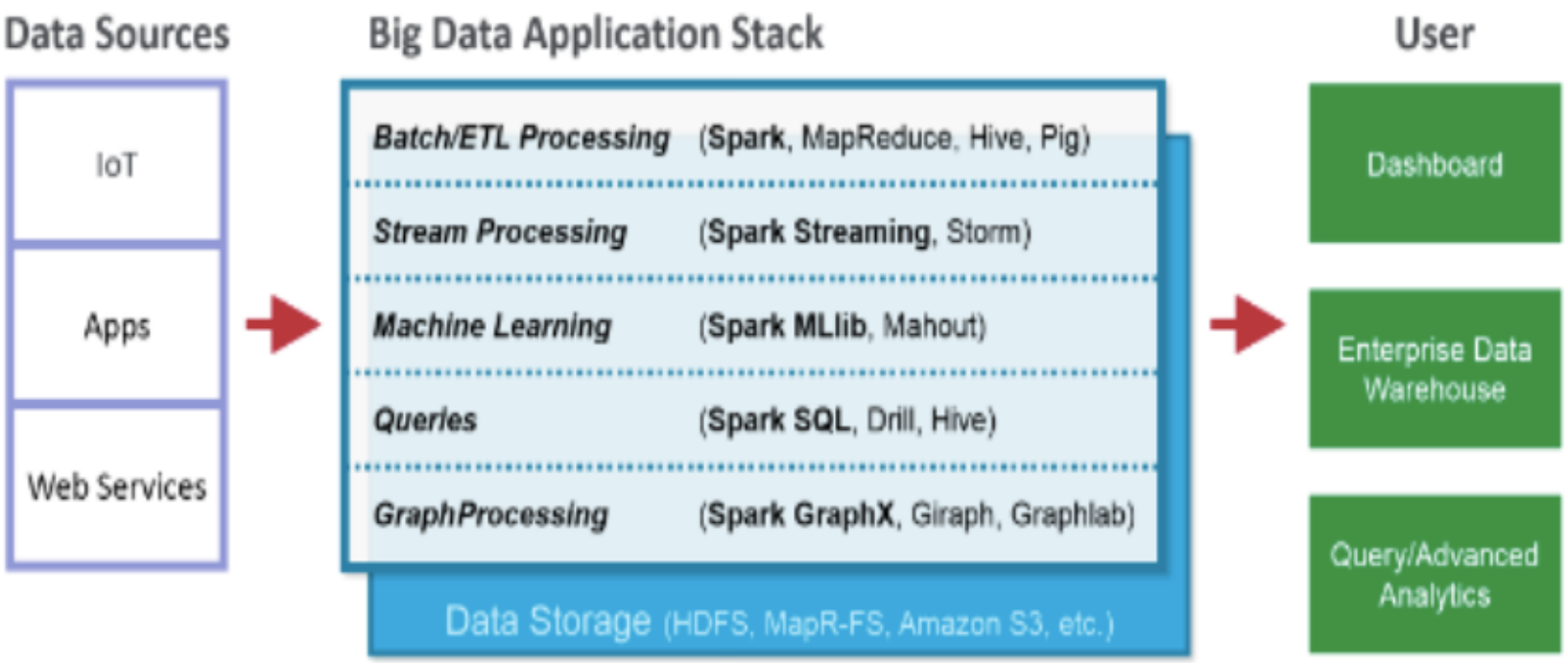



Scala
Java
python
DataFrames
ML Pipelines
Spark SQL
Spark Streaming
MLlib
GraphX
Spark Core
Data Sources
hadoop
cassandra
HBASE
{JSON}
MySQL
elasticsearch.

Spark Platform



Source: MapR Academy



Source: MapR Academy

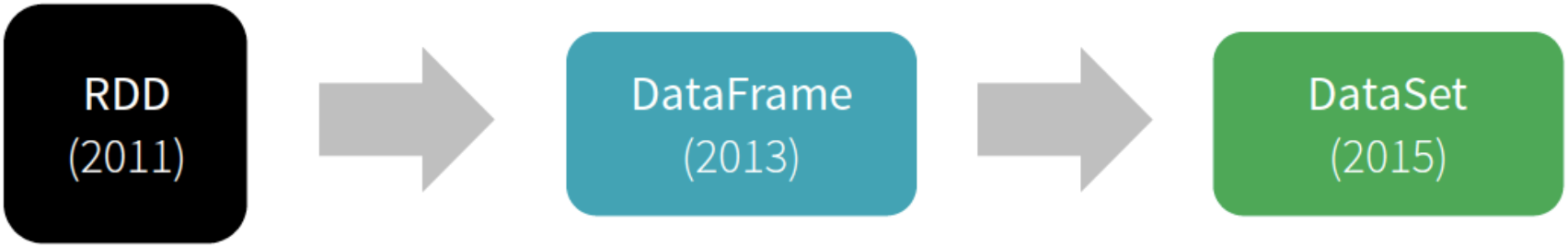
Notable Users

Source: Jump start into Apache Spark and Databricks

Do we still need Hadoop?

- Yes, why Hadoop?
  - HDFS
  - YARN
  - MapReduce is mature and still be appropriate for certain workloads
  - Other services: Sqoop, Flume, etc.
- But you can still use other resource management, storages
  - Spark Standalone
  - Amazon S3
  - Mesos

History of Spark APIs



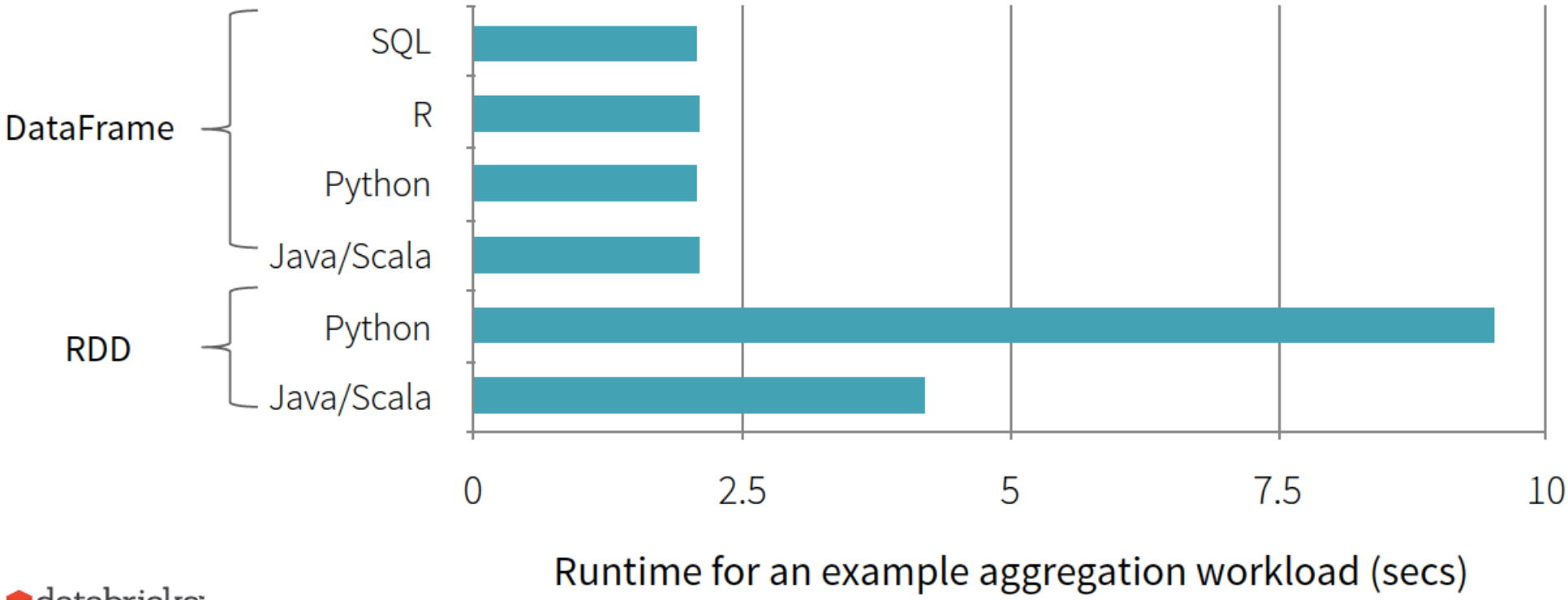
- Distribute collection of JVM objects
- Functional Operators (map, filter, etc.)

- Distribute collection of Row objects
- Expression-based operations and UDFs
- Logical plans and optimizer
- Fast/efficient internal representations

- Internally rows, externally JVM objects
- "Best of both worlds" **type safe + fast**

Source: Jump start into Apache Spark and Databricks

Benefit of Logical Plan: Performance Parity Across Languages



databricks

Source: Jump start into Apache Spark and Databricks

What is a RDD?

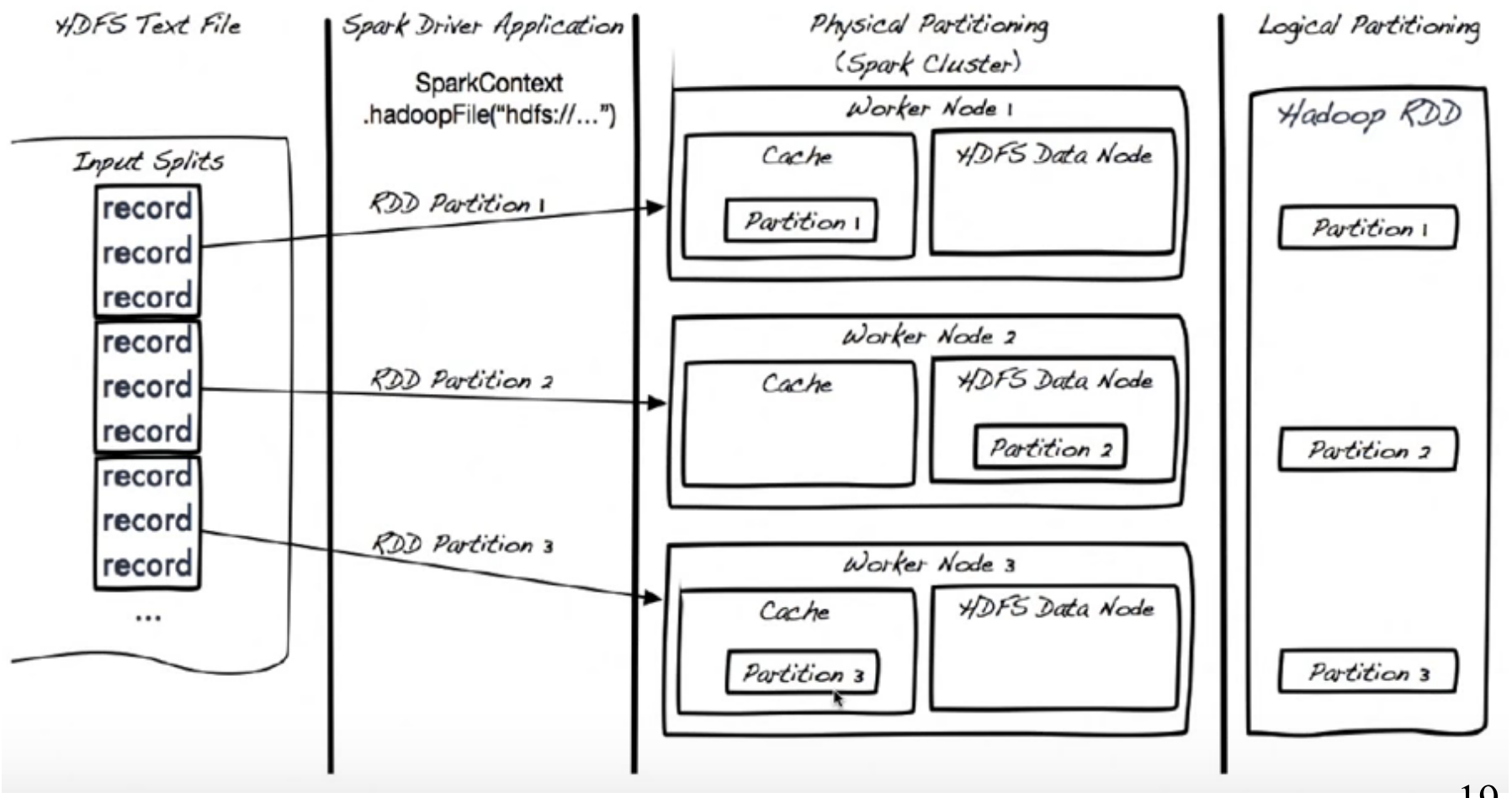
- Resilient: if the data in memory (or on a node) is lost, it can be recreated.
- Distributed: data is chucked into partitions and stored in memory acress the custer.
- Dataset: initial data can come from a table or be created programmatically

RDD:

- Fault tollerant
- Immutable
- Three methods for creating RDD:
  - Parallelizing an existing correction
  - Referencing a dataset
  - Transformation from an existing RDD
- Types of files supported:
  - Text files
  - SequenceFiles
  - Hadoop InputFormat

RDD Creation

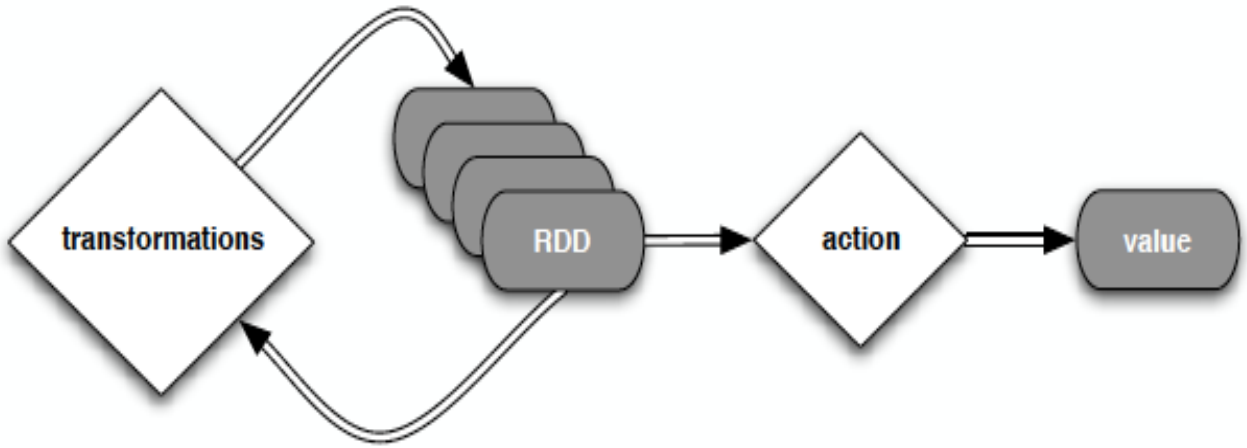
```
hdfsData = sc.textFile("hdfs://data.txt")
```



Source: Pspark: A brain-friendly introduction

RDD: Operations

- Transformations: transformations are lazy (not computed immediately)
- Actions: the transformed RDD gets recomputed when an action is run on it (default)



Direct Acyclic Graph (DAG)

- View the DAG
  linesLength.toDebugString
- Sample DAG

```
res5: String =
MappedRDD[4] at map at <console>:16 (3 partitions)
  MappedRDD[3] at map at <console>:16 (3 partitions)
    FilteredRDD[2] at filter at <console>:14 (3 partitions)
      MappedRDD[1] at textFile at <console>:12 (3 partitions)
        HadoopRDD[0] at textFile at <console>:12 (3 partitions)
```

Functions Deconstructed

```
import random
flips = 1000000

# lazy eval
coins = xrange(flips)

# lazy eval, nothing executed
heads = sc.parallelize(coins) \
          .map(lambda i: random.random()) \
          .filter(lambda r: r < 0.51) \
          .count()
```

Python Generator

Create RDD

Transformations

Action (materialize result)

Spark:Transformation

transformation	*description*
map(*func*)	return a new distributed dataset formed by passing each element of the source through a function *func*
filter(*func*)	return a new dataset formed by selecting those elements of the source on which *func* returns true
flatMap(*func*)	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)
sample(*withReplacement, fraction, seed*)	sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*
union(*otherDataset*)	return a new dataset that contains the union of the elements in the source dataset and the argument
distinct([*numTasks*]))	return a new dataset that contains the distinct elements of the source dataset

Spark:Transformation

transformation	description
groupByKey([*numTasks*])	when called on a dataset of (K, V) pairs, returns a dataset of (K, Seq[V]) pairs
reduceByKey(*func*, [*numTasks*])	when called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function
sortByKey([*ascending*], [*numTasks*])	when called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument
join(*otherDataset*, [*numTasks*])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
cogroup(*otherDataset*, [*numTasks*])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, Seq[V], Seq[W]) tuples – also called groupWith
cartesian(*otherDataset*)	when called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)

Single RDD Transformation

filter females to analyze female buying patterns

```
male1, male2, female1 -> female1
```

map squared values

```
2, 5, 6 -> 4, 25, 36
```

flatMap to break up a sentence into words

```
my name is ray -> my, name, is, ray
```

find the distinct values in a dataset

```
apple, apple, banana -> apple, banana
```

sample two values at random

```
apple, banana, guava -> banana, apple
```

Source: Jspark: A brain-friendly introduction

Multiple RDD Transformation

union

```
apple, orange, banana, guava,
banana, pear
```

intersection

```
banana
```

subtract anything shown in Dataset B from Dataset A

```
apple, orange
```

cartesian (every possible pair combo)

```
(apple, guava), (apple, banana), ...
```

Source: Jspark: A brain-friendly introduction

Pair RDD Transformation

- reduceByKey
- groupByKey
- combineByKey
- mapValues
- flatMapValues
- keys
- values
- subtractByKey
- join
- rightOuterJoin
- leftOuterJoin
- cogroup
- sortByKey

Source: Jspark: A brain-friendly introduction

Spark:Actions

action	description
`reduce(func)`	aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel
`collect()`	return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data
`count()`	return the number of elements in the dataset
`first()`	return the first element of the dataset – similar to *take(1)*
`take(n)`	return an array with the first *n* elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
`takeSample(withReplacement, fraction, seed)`	return an array with a random sample of *num* elements of the dataset, with or without replacement, using the given random number generator seed

Spark:Actions

action	description
saveAsTextFile(*path*)	write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call `toString` on each element to convert it to a line of text in the file
saveAsSequenceFile(*path*)	write the elements of the dataset as a Hadoop `SequenceFile` in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's `Writable` interface or are implicitly convertible to `Writable` (Spark includes conversions for basic types like `Int`, `Double`, `String`, etc).
countByKey()	only available on RDDs of type `(K, V)`. Returns a `Map` of `(K, Int)` pairs with the count of each key
foreach(*func*)	run a function *func* on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems

Spark: Persistence

transformation	*description*
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as serialized Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc	Same as the levels above, but replicate each partition on two cluster nodes.

Accumulators

- Similar to a MapReduce “Counter”
- A global variable to track metrics about your Spark program
  for debugging.
- Reasoning: Excutors do not communicate with each other.
- Sent back to driver

Broadcast Variables

- Similar to a MapReduce “Distributed Cache”
- Sends read-only values to worker nodes.
- Great for lookup tables, dictionaries, etc.

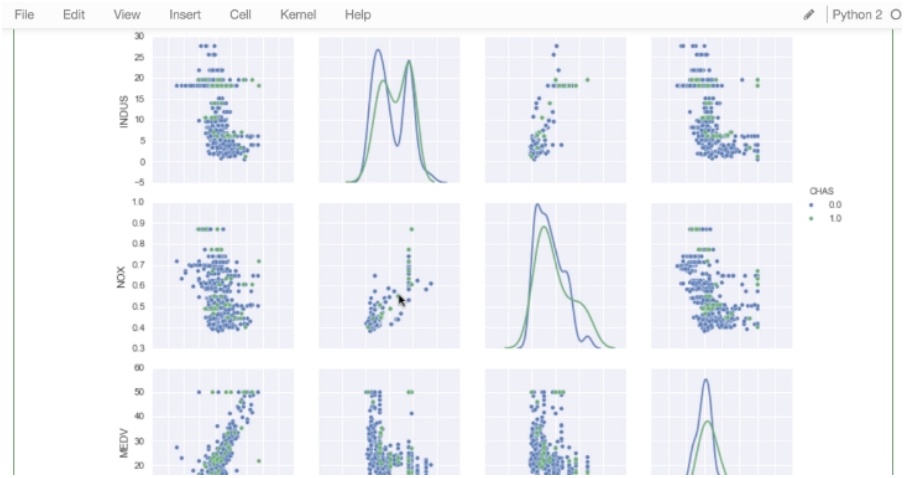
cloudera

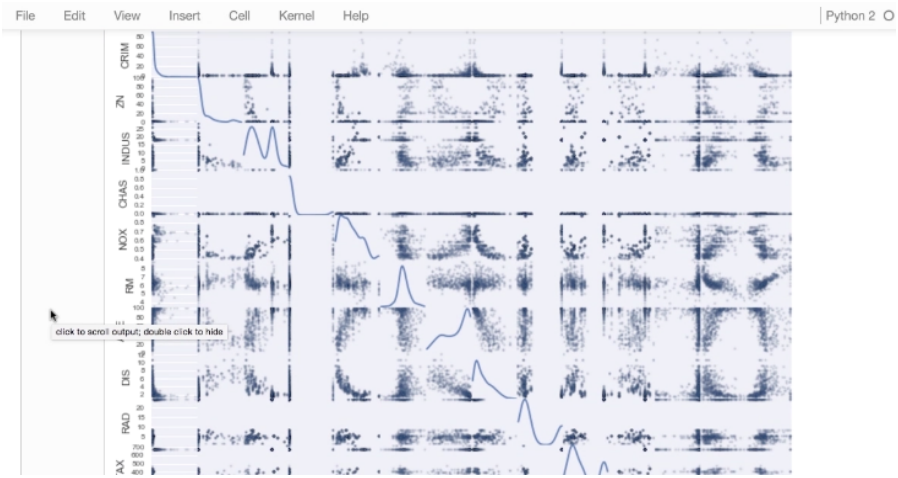
- Server-Client application: allows you to edit and run your notebooks via a web browser.
- Two main components:
  - A kernel is a program that runs and introspects the user's code.
  - The dashboard of the application: can also be used to manage the kernels
- IPython is original, Fernando Pérez starts developing in late 2001.
  - Lastly, in 2014, Project Jupyter started as a spin-off project from IPython.
  - IPython is now the name of the Python backend, which is also known as the kernel.

ANACONDA
Powered by Continuum Analytics

Why is Jupyter ?

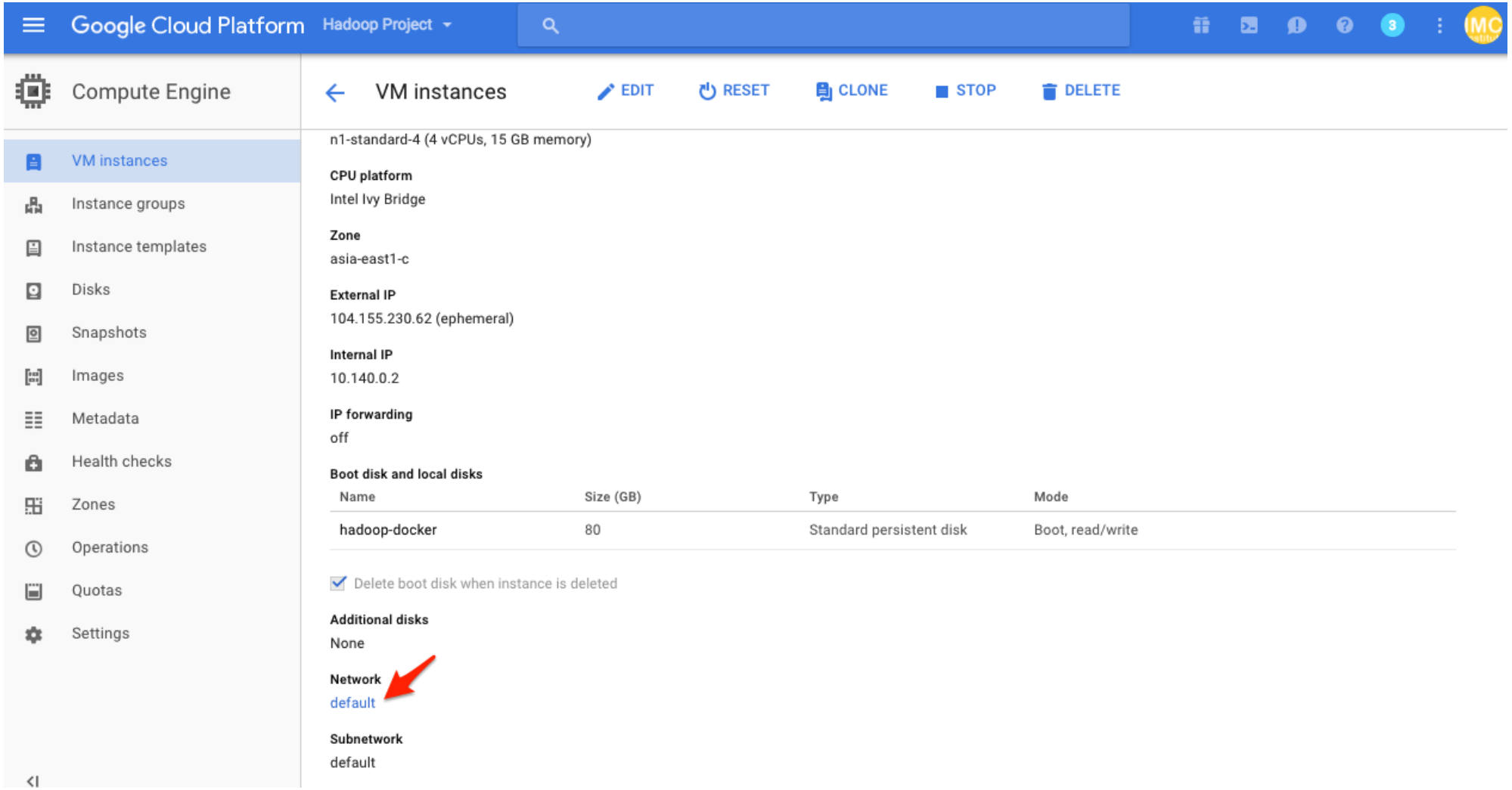
- IDEs
- Can be saved and easily shared in .ipynb JSON format.
- Statistical data visualization, such as Seaborn



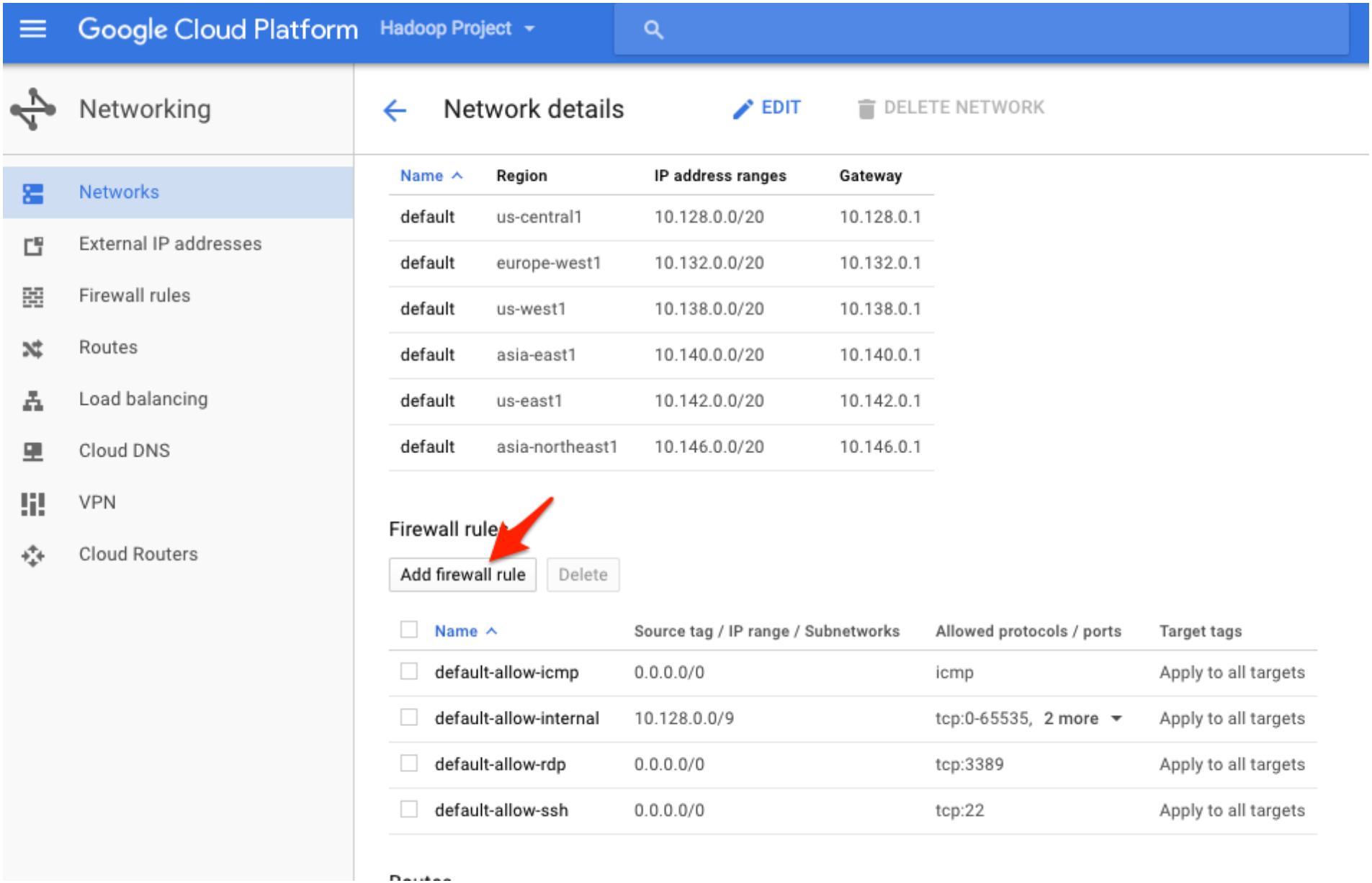


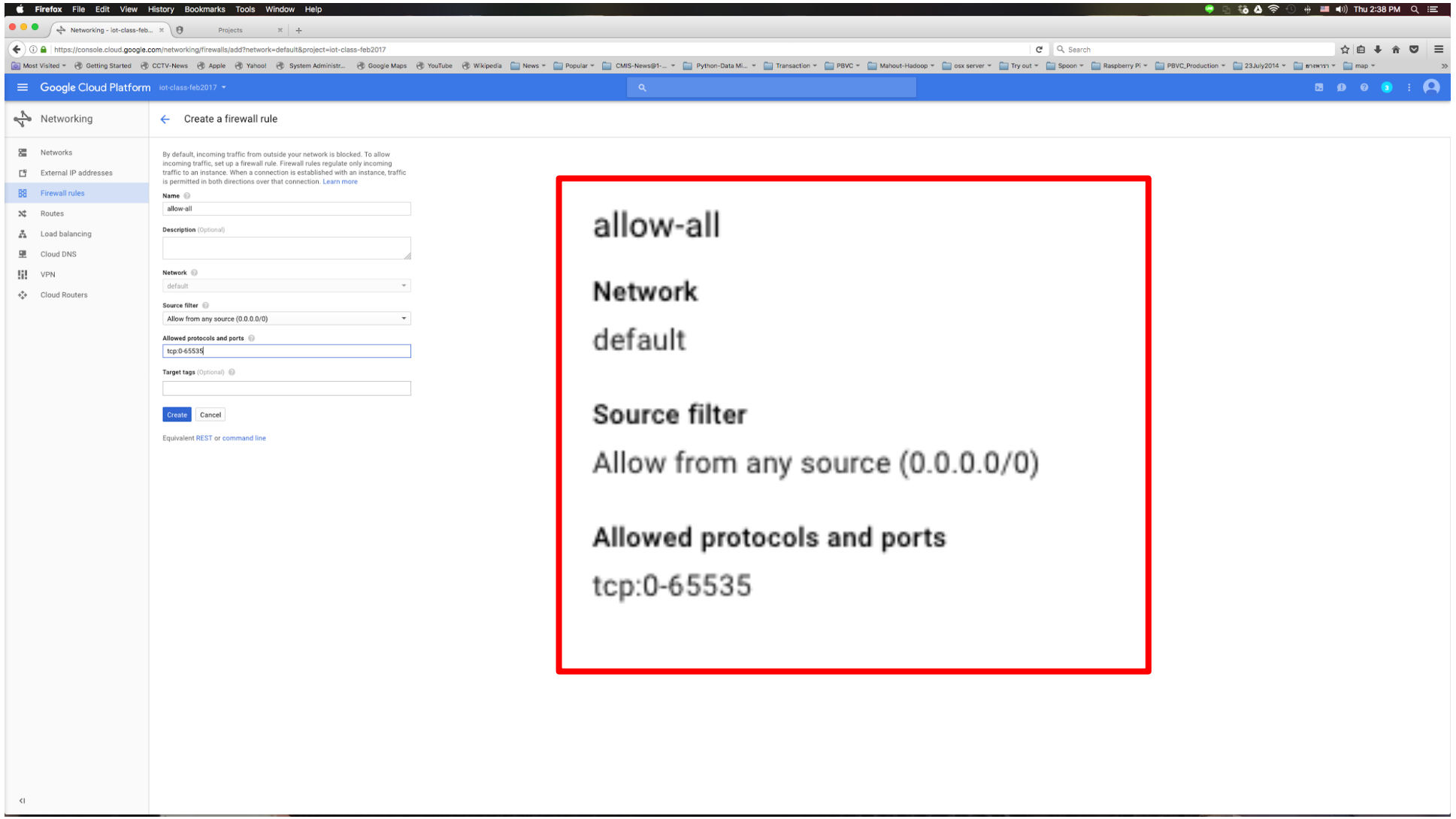
Source: Jonathan W., “Jupyter Notebook for Data Science Teams,” Infinite Skills, 2016

Config Firewall: Select the Network



Config Firewall:




Google Cloud Platform
Networking
Create a firewall rule
Networks
External IP addresses
Firewall rules
Routes
Load balancing
Cloud DNS
VPN
Cloud Routers
Name
allow-all
Description (Optional)
Network
default
Source filter
Allow from any source (0.0.0.0/0)
Allowed protocols and ports
tcp:0-65535
Target tags (Optional)
Create
Cancel
Equivalent REST or command line
allow-all
Network
default
Source filter
Allow from any source (0.0.0.0/0)
Allowed protocols and ports
tcp:0-65535

Firewall rules

Add firewall rule | Delete

Name	Source tag / IP range / Subnetworks	Allowed protocols / ports	Target tags
allow-all	0.0.0.0/0	tcp:0-65535	Apply to all targets
default-allow-http	0.0.0.0/0	tcp:80	http-server
default-allow-https	0.0.0.0/0	tcp:443	https-server
default-allow-icmp	0.0.0.0/0	icmp	Apply to all targets
default-allow-internal	10.128.0.0/9	tcp:0-65535, udp:0-65535, 1 more	Apply to all targets
default-allow-rdp	0.0.0.0/0	tcp:3389	Apply to all targets
default-allow-ssh	0.0.0.0/0	tcp:22	Apply to all targets

Getting Started

- **Delete the exising containers**

```
$ sudo docker rm $(sudo docker ps -a -q)

$ sudo docker ps -a
```

```
ubuntu@ip-172-31-8-114:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
```

- **Run a Cloudera container**

```
$ sudo docker run -v /home:/mnt --hostname=quickstart.cloudera --
privileged=true -t -i -p 8888:8888 -p 8880:8880 cloudera/quickstart
/usr/bin/docker-quickstart
```

```
Using CATALINA_BASE:   /var/lib/oozie/tomcat-deployment
Using CATALINA_HOME:   /usr/lib/bigtop-tomcat
Using CATALINA_TMPDIR: /var/lib/oozie
Using JRE_HOME:        /usr/java/jdk1.7.0_67-cloudera
Using CLASSPATH:       /usr/lib/bigtop-tomcat/bin/bootstrap.jar
Using CATALINA_PID:    /var/run/oozie/oozie.pid
Starting Solr server daemon:                               [  OK  ]
Using CATALINA_BASE:   /var/lib/solr/tomcat-deployment
Using CATALINA_HOME:   /usr/lib/solr/../bigtop-tomcat
Using CATALINA_TMPDIR: /var/lib/solr/
Using JRE_HOME:        /usr/java/jdk1.7.0_67-cloudera
Using CLASSPATH:       /usr/lib/solr/../bigtop-tomcat/bin/bootstrap.jar
Using CATALINA_PID:    /var/run/solr/solr.pid
Started Impala Catalog Server (catalogd) :                 [  OK  ]
Started Impala Server (impalad):                           [  OK  ]
[root@quickstart /]#
```

1. Install neccessary applications

```
[root@quickstart /]# yum install wget
[root@quickstart /]# wget https://bootstrap.pypa.io/get-pip.py
[root@quickstart /]# python get-pip.py
[root@quickstart /]# yum install nano
```

2. Install the Anaconda, Jupyter and some Modules

```
[root@quickstart /]# wget
https://repo.continuum.io/archive/Anaconda2-4.2.0-Linux-x86_64.sh

[root@quickstart /]# bash Anaconda2-4.2.0-Linux-x86_64.sh
```

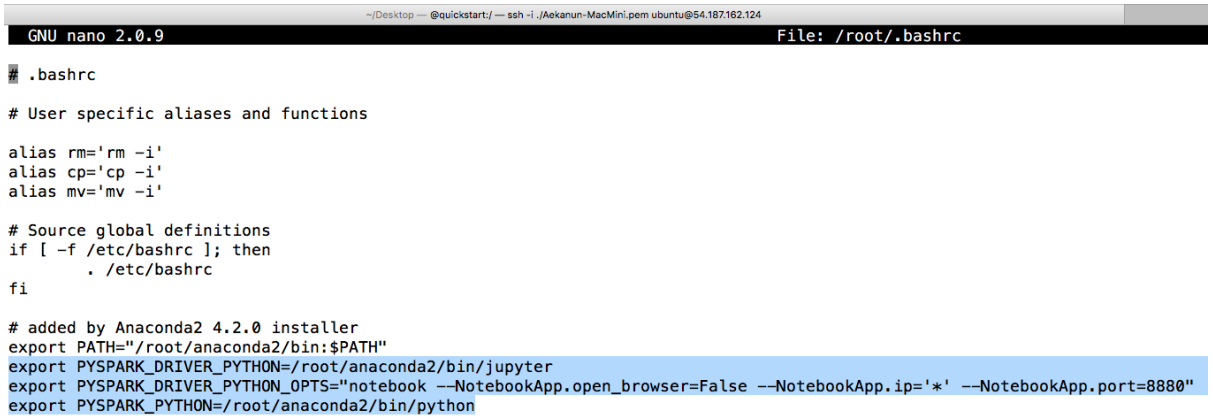
```
installation finished.
Do you wish the installer to prepend the Anaconda2 install location
to PATH in your /root/.bashrc ? [yes|no]
[no] >>> yes
```

```
[root@quickstart /]# export
PYSPARK_DRIVER_PYTHON=/root/anaconda2/bin/jupyter

[root@quickstart /]# export
PYSPARK_DRIVER_PYTHON_OPTS="notebook --
NotebookApp.open_browser=False --NotebookApp.ip='*' --
NotebookApp.port=8880"

[root@quickstart /]# export
PYSPARK_PYTHON=/root/anaconda2/bin/python

[root@quickstart /]# nano /root/.bashrc
```



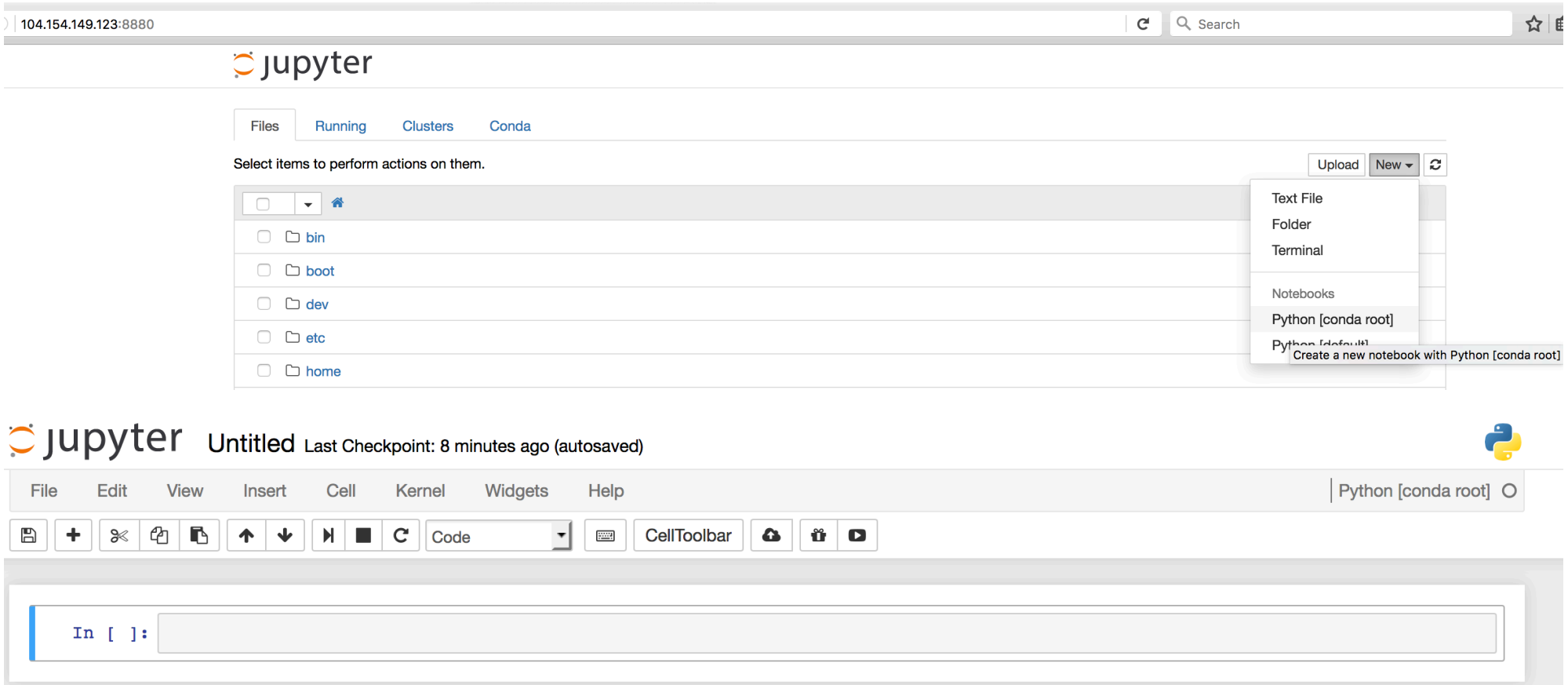
3. Run Jupyter notebook with Pyspark

```
[root@quickstart /]# pyspark
```

```
[root@quickstart /]# pyspark
[I 05:35:56.029 NotebookApp] [nb_conda_kernels] enabled, 2 kernels found
[W 05:35:56.507 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recommended.
[W 05:35:56.507 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using authentication. This is highly insecure and not recommended.
[I 05:35:56.552 NotebookApp] ✓ nbpresent HTML export ENABLED
[W 05:35:56.552 NotebookApp] ✗ nbpresent PDF export DISABLED: No module named nbbrowserpdf.exporters.pdf
[I 05:35:56.556 NotebookApp] [nb_conda] enabled
[I 05:35:56.595 NotebookApp] [nb_anacondacloud] enabled
[I 05:35:56.599 NotebookApp] Serving notebooks from local directory: /
[I 05:35:56.600 NotebookApp] 0 active kernels
[I 05:35:56.600 NotebookApp] The Jupyter Notebook is running at: http://[all ip addresses on your system]:8880/
[I 05:35:56.600 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

4. Open Web Browser

- Type URL **<External IP>:8880**
- Choose **New Python(conda root)**



Functional tools in Python

- map
- filter
- reduce
- lambda

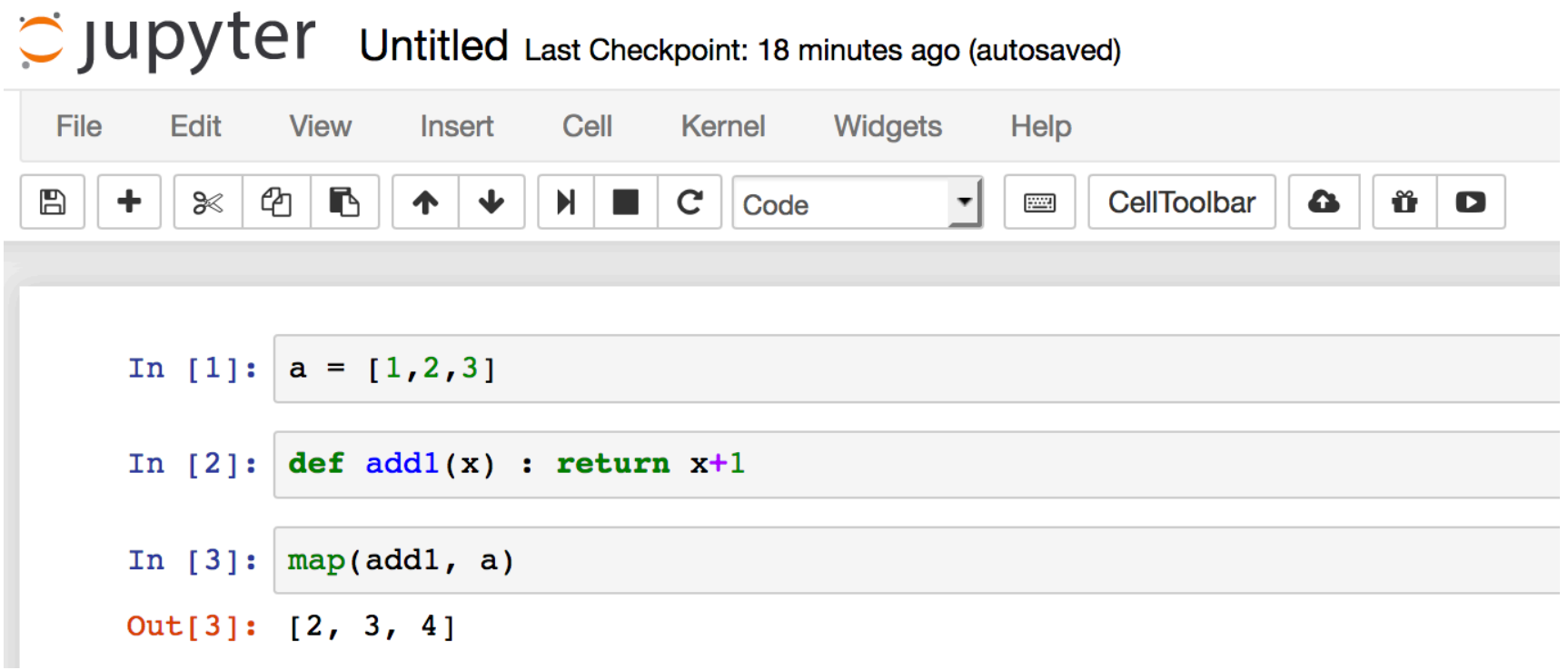
Map

```
>>> a = [1,2,3]

>>> def add1(x) : return x+1

>>> map(add1, a)

     Result: [2,3,4]
```



Filter

```
>>> a = [1,2,3,4]

>>> def isOdd(x) : return x%2==1

>>> filter(isOdd, a)

     Result: [1,3]
```

Reduce

```
>>> a= [1,2,3,4]

>>> def add(x,y) : return x+y

>>> reduce(add, a)

     Result: 10
```

lambda

```
>>> (lambda x: x + 1)(3)
      Result: 4

>>> map((lambda x: x + 1), [1,2,3])

     Result: [2,3,4]
```

Exercises

- (lambda x: 2*x)(3) => ?
- map(lambda x: 2*x, [1,2,3]) =>
- map(lambda t: t[0], [(1,2), (3,4), (5,6)]) =>
- reduce(lambda x,y: x+y, [1,2,3]) =>
- reduce(lambda x,y: x+y, map(lambda t: t[0], [(1,2), (3,4), (5,6)]))=>

Hands-On: Word Count
(LAB 1)

Download an example text file

```
[root@quickstart /]# wget
https://s3.amazonaws.com/imcbucket/input/pg2600.txt
```

```
[root@quickstart /]# cd
[root@quickstart ~]# mkdir guest1
[root@quickstart ~]# cd guest1/
[root@quickstart guest1]# wget https://s3.amazonaws.com/imcbucket/input/pg2600.txt
--2016-11-07 14:08:20--  https://s3.amazonaws.com/imcbucket/input/pg2600.txt
Resolving s3.amazonaws.com... 54.231.114.108
Connecting to s3.amazonaws.com|54.231.114.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3291648 (3.1M) [text/plain]
Saving to: “pg2600.txt”
```

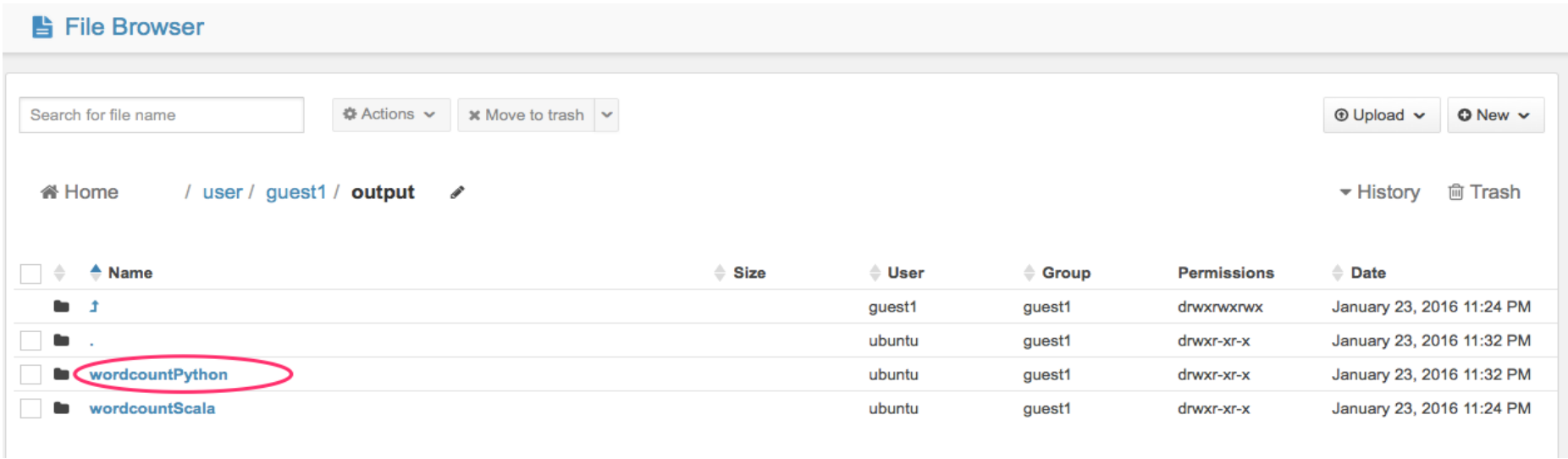
Upload data to Hadoop

```
[root@quickstart /]# hadoop fs -mkdir -p /user/cloudera/input
[root@quickstart /]# hadoop fs -put pg2600.txt /user/cloudera/input/
[root@quickstart /]# hadoop fs -ls /user/cloudera/input
```

```
[root@quickstart guest1]# hadoop fs -put pg2600.txt /user/cloudera/input/
[root@quickstart guest1]# hadoop fs -ls /user/cloudera/input
Found 1 items
-rw-r--r--   1 root cloudera    3291648 2016-11-07 14:11 /user/cloudera/input/pg2600.txt
[root@quickstart guest1]#
```

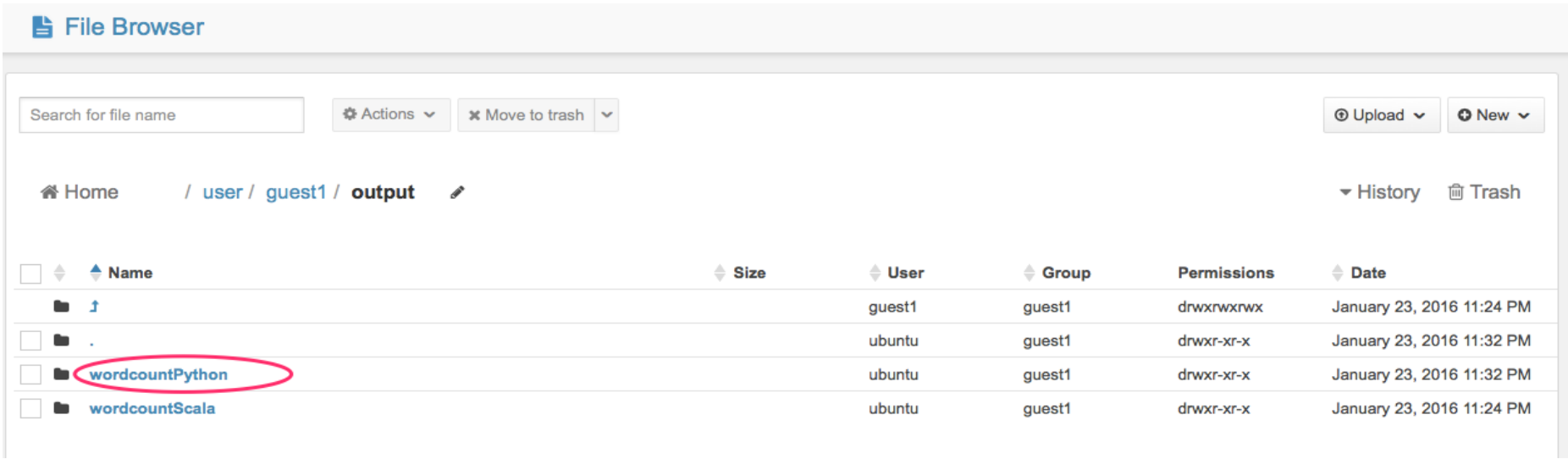
Spark Program in Python: WordCount

```
>>> from operator import add
>>> file = sc.textFile("hdfs:///user/cloudera/input/pg2600.txt")
>>> wc = file.flatMap(lambda x: x.split(' ')).map(lambda x: (x,
1)).reduceByKey(add)
>>>
wc.saveAsTextFile("hdfs:///user/cloudera/output/wordcountPython")
```

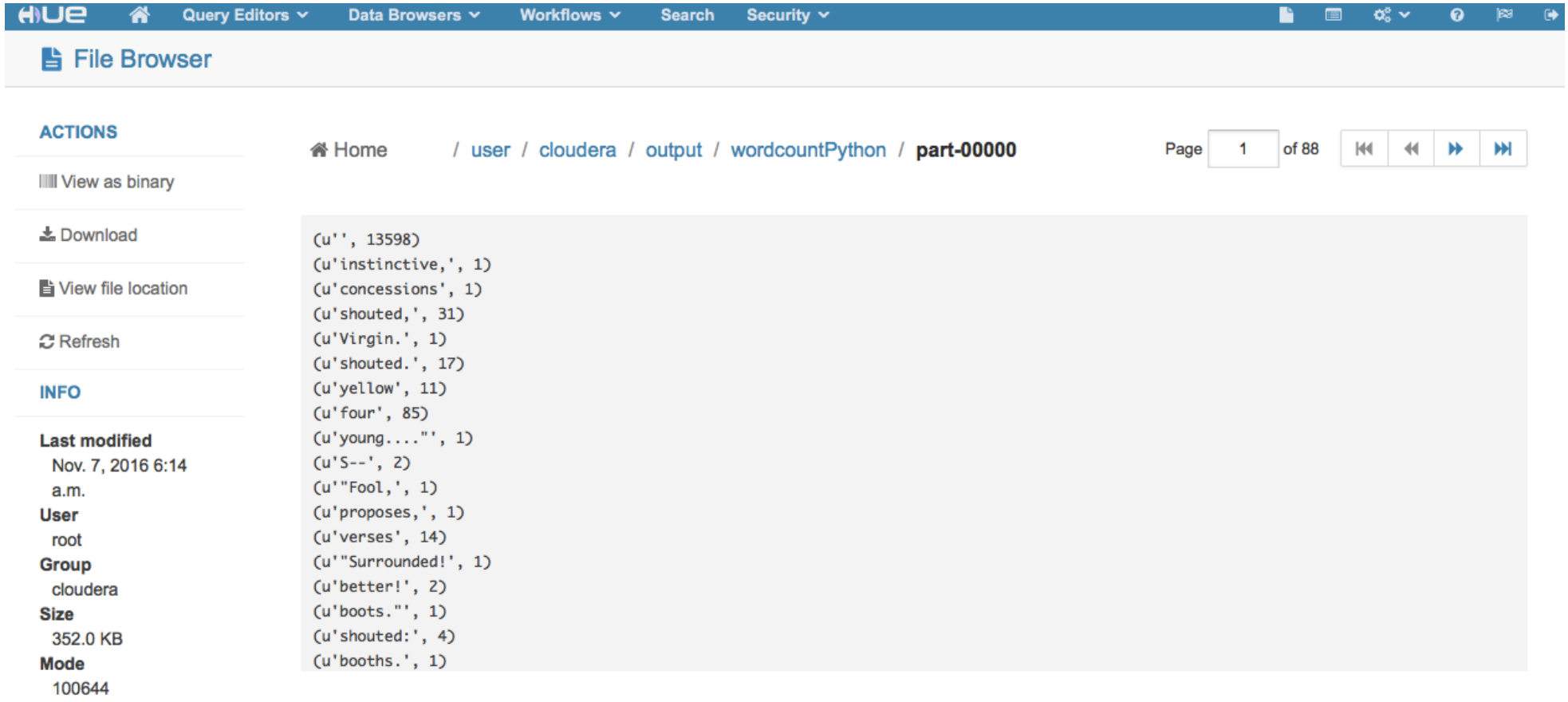


Spark Program in Python: WordCount

```
>>> from operator import add
>>> file = sc.textFile("hdfs:///user/cloudera/input/pg2600.txt")
>>> wc = file.flatMap(lambda x: x.split(' '))
>>> wc.take(1)
>>> wc1 = wc.map(lambda x: (x, 1))
>>> wc1.take(1)
>>> wc2 = wc1.reduceByKey(add)
>>> wc2.take(1)
```



WordCount output



Hands-On: Find Big Spenders

(LAB 2)

Spark Program : Toy_data.txt

Upload a dataset to HDFS

```
[root@quickstart /]# wget
https://s3.amazonaws.com/imcbucket/data/toy_data.txt
[root@quickstart /]# hadoop fs -put toy_data.txt /user/cloudera/input
```

Home / user / cloudera / **input** History Trash

Name	Size	User	Group	Permissions	Date
↑		cloudera	cloudera	drwxr-xr-x	June 14, 2016 01:19 AM
.		cloudera	cloudera	drwxr-xr-x	June 14, 2016 07:32 AM
pg2600.txt	3.1 MB	root	cloudera	-rw-r--r--	June 13, 2016 09:29 PM
toy_data.txt	136 bytes	root	cloudera	-rw-r--r--	June 14, 2016 07:32 AM

Start pyspark

```
[root@quickstart /]# pyspark
```

Spark Program : Find Big Spenders

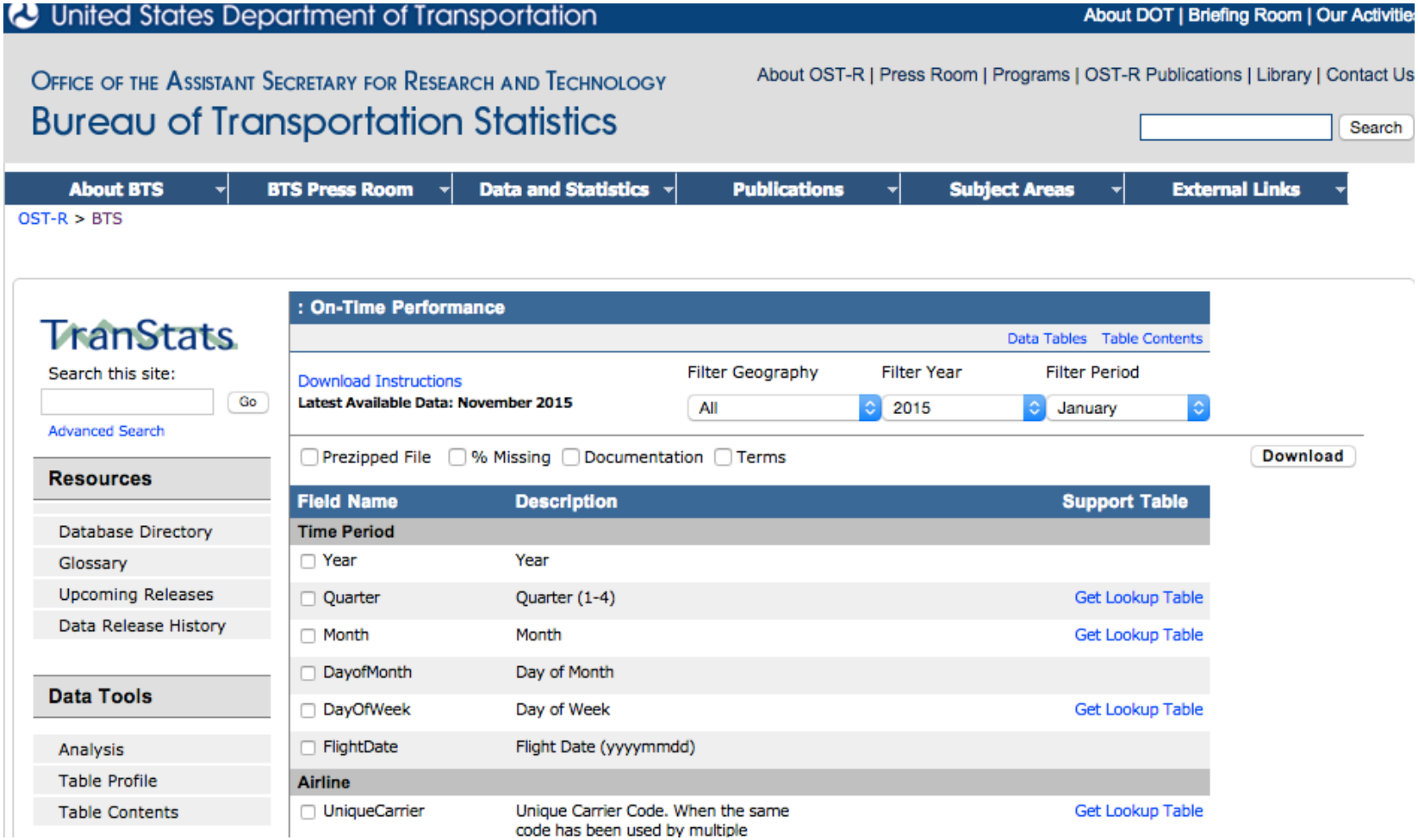
```
>>> file_rdd =
sc.textFile("hdfs:///user/cloudera/input/toy_data.txt")
>>> import json
>>> json_rdd = file_rdd.map(lambda x: json.loads(x))
>>> big_spenders = json_rdd.map(lambda x:
tuple((x.keys()[0],int(x.values()[0]))))
>>> big_spenders.reduceByKey(lambda x,y: x + y).filter(lambda
x: x[1] > 5).collect()
```

Hands-On: Finding Best/Worst Airlines

(LAB 3)

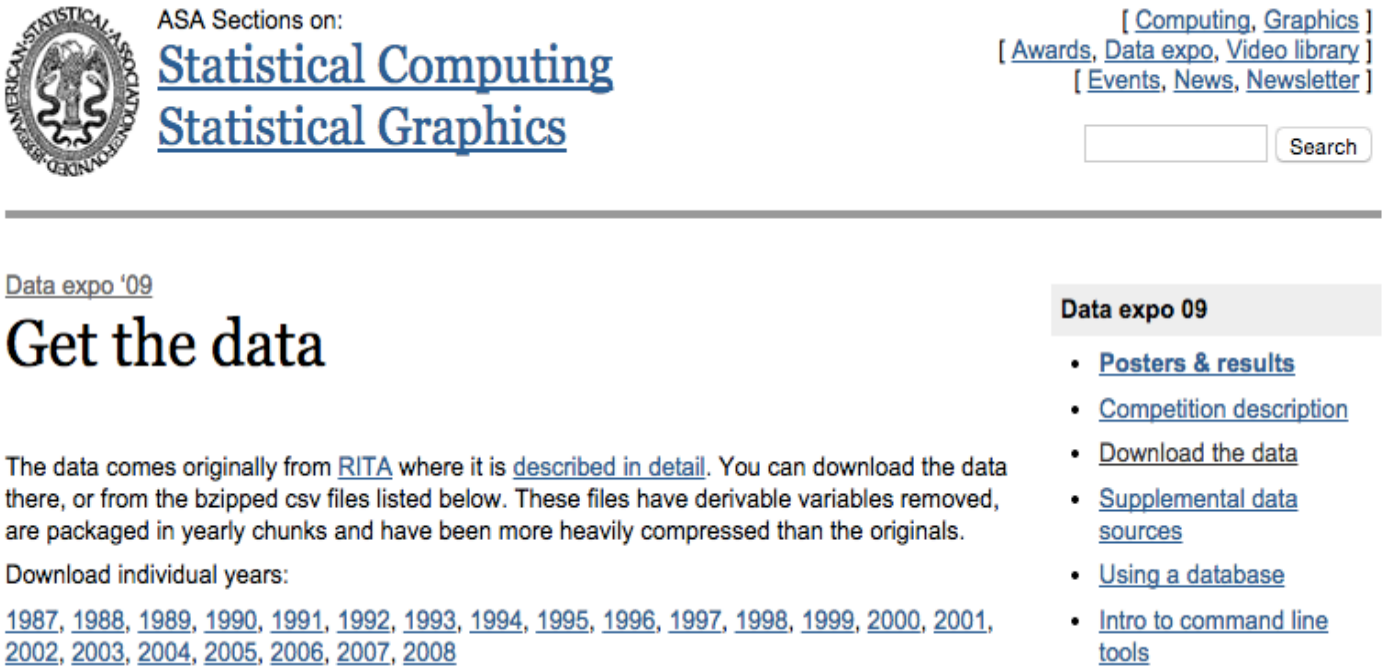
Flight Details Data

http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236



Flight Details Data

http://stat-computing.org/dataexpo/2009/the-data.html



ASA Sections on:
Statistical Computing
Statistical Graphics

[Computing, Graphics]
[Awards, Data expo, Video library]
[Events, News, Newsletter]

Search

Data expo '09

Get the data

The data comes originally from RITA where it is described in detail. You can download the data there, or from the bzipped csv files listed below. These files have derivable variables removed, are packaged in yearly chunks and have been more heavily compressed than the originals.

Download individual years:

1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008

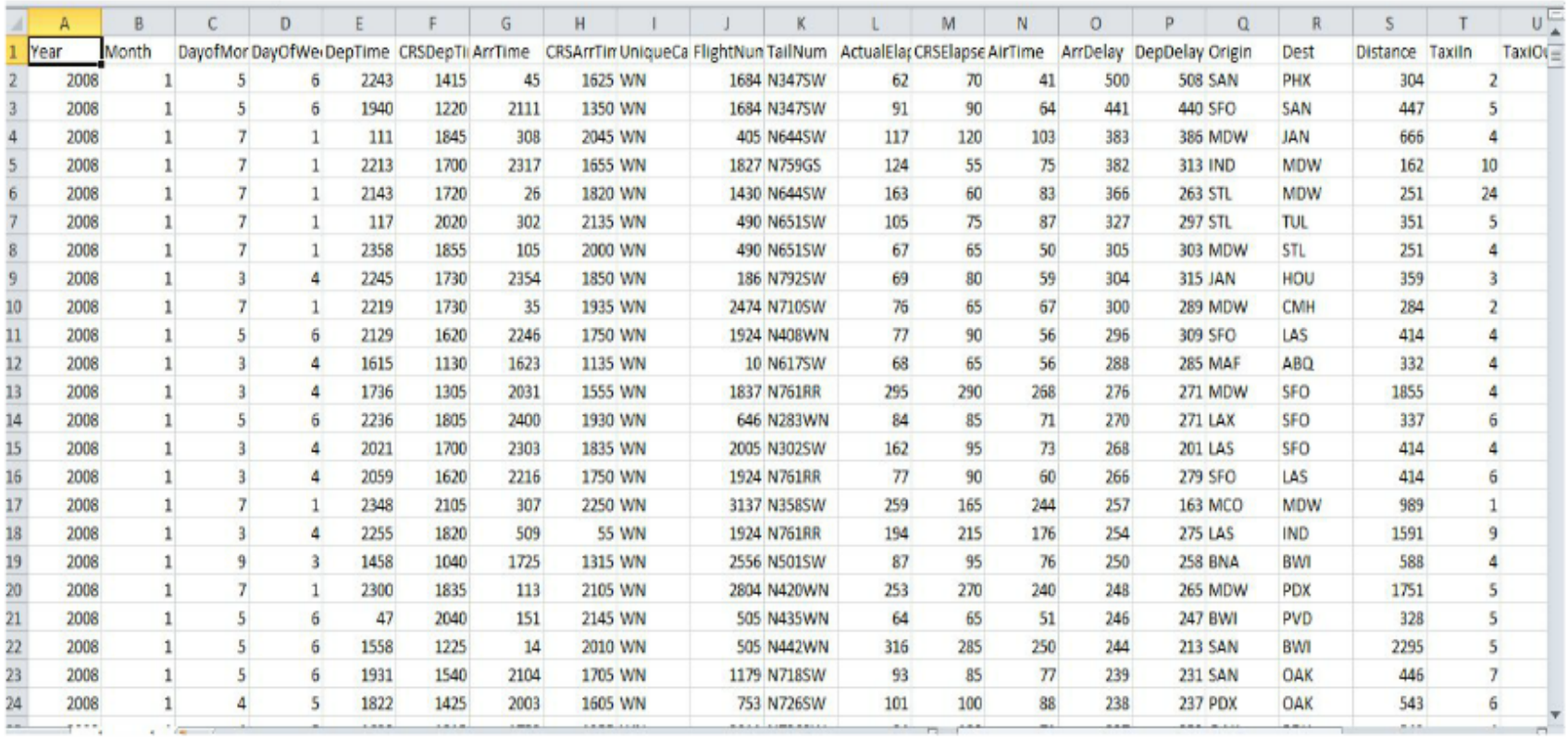
Data expo 09

- **Posters & results**
- Competition description
- Download the data
- Supplemental data sources
- Using a database
- Intro to command line tools

https://s3.amazonaws.com/imcbucket/data/flights/2016-jan.csv

	Name	Description
1	Year	1987-2008
2	Month	1-12
3	DayofMonth	1-31
4	DayOfWeek	1 (Monday) - 7 (Sunday)
5	DepTime	actual departure time (local, hhmm)
6	CRSDepTime	scheduled departure time (local, hhmm)
7	ArrTime	actual arrival time (local, hhmm)
8	CRSArrTime	scheduled arrival time (local, hhmm)
9	UniqueCarrier	unique carrier code
10	FlightNum	flight number
11	TailNum	plane tail number
12	ActualElapsedTime	in minutes
13	CRSElapsedTime	in minutes
14	AirTime	in minutes
15	ArrDelay	arrival delay, in minutes
16	DepDelay	departure delay, in minutes
17	Origin	origin IATA airport code
18	Dest	destination IATA airport code
19	Distance	in miles
20	TaxiIn	taxi in time, in minutes
21	TaxiOut	taxi out time in minutes
22	Cancelled	was the flight cancelled?
23	CancellationCode	reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
24	Diverted	1 = yes, 0 = no
25	CarrierDelay	in minutes
26	WeatherDelay	in minutes
27	NASDelay	in minutes
28	SecurityDelay	in minutes
29	LateAircraftDelay	in minutes

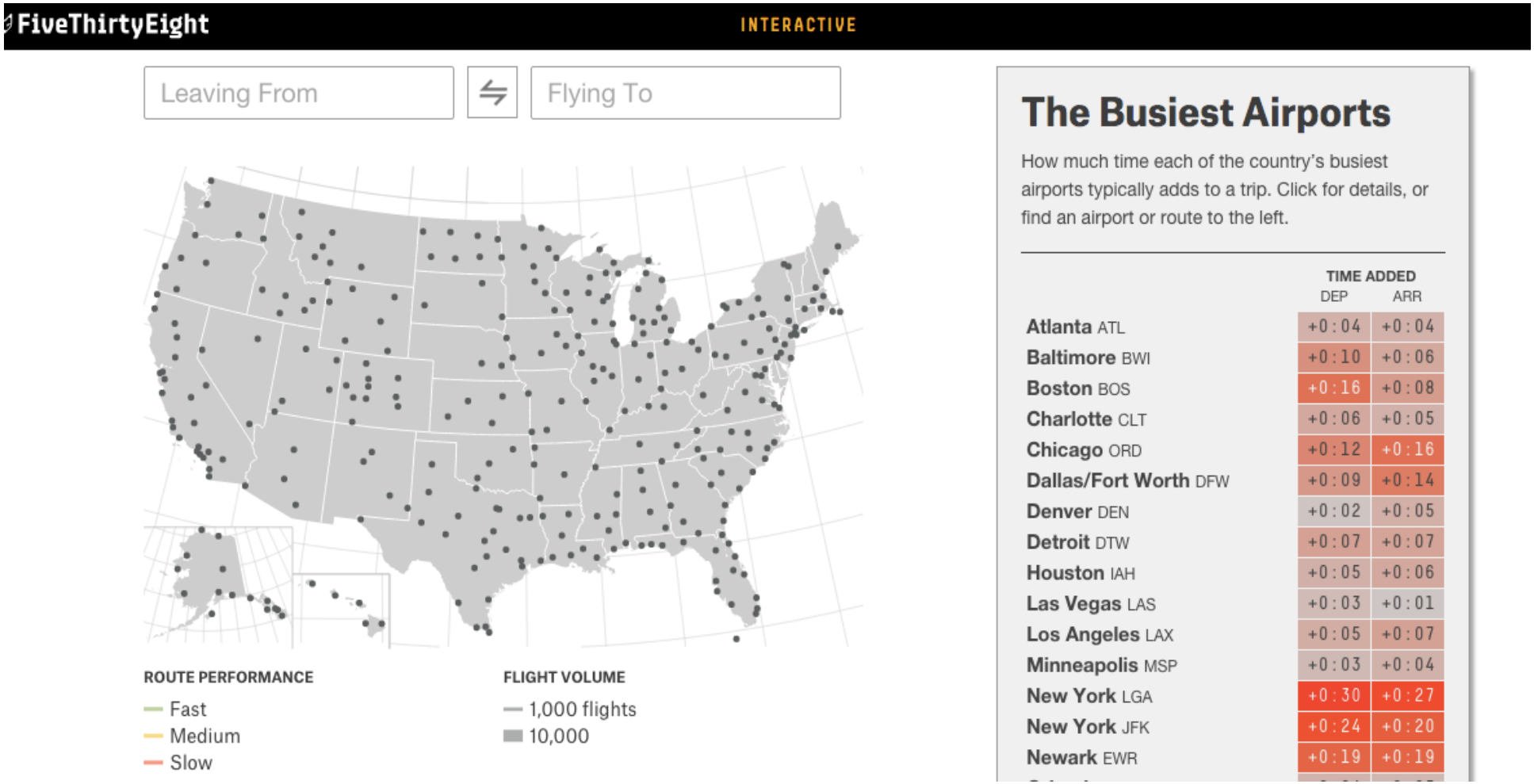
Snapshot of Dataset



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Year	Month	DayofMor	DayOfWe	DepTime	CRSDepTi	ArrTime	CRSArrTin	UniqueCa	FlightNun	TailNum	ActualEla	CRSElapse	AirTime	ArrDelay	DepDelay	Origin	Dest	Distance	TaxiIn	TaxiO
2	2008	1	5	6	2243	1415	45	1625	WN	1684	N347SW	62	70	41	500	508	SAN	PHX	304	2	
3	2008	1	5	6	1940	1220	2111	1350	WN	1684	N347SW	91	90	64	441	440	SFO	SAN	447	5	
4	2008	1	7	1	111	1845	308	2045	WN	405	N644SW	117	120	103	383	386	MDW	JAN	666	4	
5	2008	1	7	1	2213	1700	2317	1655	WN	1827	N759GS	124	55	75	382	313	IND	MDW	162	10	
6	2008	1	7	1	2143	1720	26	1820	WN	1430	N644SW	163	60	83	366	263	STL	MDW	251	24	
7	2008	1	7	1	117	2020	302	2135	WN	490	N651SW	105	75	87	327	297	STL	TUL	351	5	
8	2008	1	7	1	2358	1855	105	2000	WN	490	N651SW	67	65	50	305	303	MDW	STL	251	4	
9	2008	1	3	4	2245	1730	2354	1850	WN	186	N792SW	69	80	59	304	315	JAN	HOU	359	3	
10	2008	1	7	1	2219	1730	35	1935	WN	2474	N710SW	76	65	67	300	289	MDW	CMH	284	2	
11	2008	1	5	6	2129	1620	2246	1750	WN	1924	N408WN	77	90	56	296	309	SFO	LAS	414	4	
12	2008	1	3	4	1615	1130	1623	1135	WN	10	N617SW	68	65	56	288	285	MAF	ABQ	332	4	
13	2008	1	3	4	1736	1305	2031	1555	WN	1837	N761RR	295	290	268	276	271	MDW	SFO	1855	4	
14	2008	1	5	6	2236	1805	2400	1930	WN	646	N283WN	84	85	71	270	271	LAX	SFO	337	6	
15	2008	1	3	4	2021	1700	2303	1835	WN	2005	N302SW	162	95	73	268	201	LAS	SFO	414	4	
16	2008	1	3	4	2059	1620	2216	1750	WN	1924	N761RR	77	90	60	266	279	SFO	LAS	414	6	
17	2008	1	7	1	2348	2105	307	2250	WN	3137	N358SW	259	165	244	257	163	MCO	MDW	989	1	
18	2008	1	3	4	2255	1820	509	55	WN	1924	N761RR	194	215	176	254	275	LAS	IND	1591	9	
19	2008	1	9	3	1458	1040	1725	1315	WN	2556	N501SW	87	95	76	250	258	BNA	BWI	588	4	
20	2008	1	7	1	2300	1835	113	2105	WN	2804	N420WN	253	270	240	248	265	MDW	PDX	1751	5	
21	2008	1	5	6	47	2040	151	2145	WN	505	N435WN	64	65	51	246	247	BWI	PVD	328	5	
22	2008	1	5	6	1558	1225	14	2010	WN	505	N442WN	316	285	250	244	213	SAN	BWI	2295	5	
23	2008	1	5	6	1931	1540	2104	1705	WN	1179	N718SW	93	85	77	239	231	SAN	OAK	446	7	
24	2008	1	4	5	1822	1425	2003	1605	WN	753	N726SW	101	100	88	238	237	PDX	OAK	543	6	

FiveThirtyEight

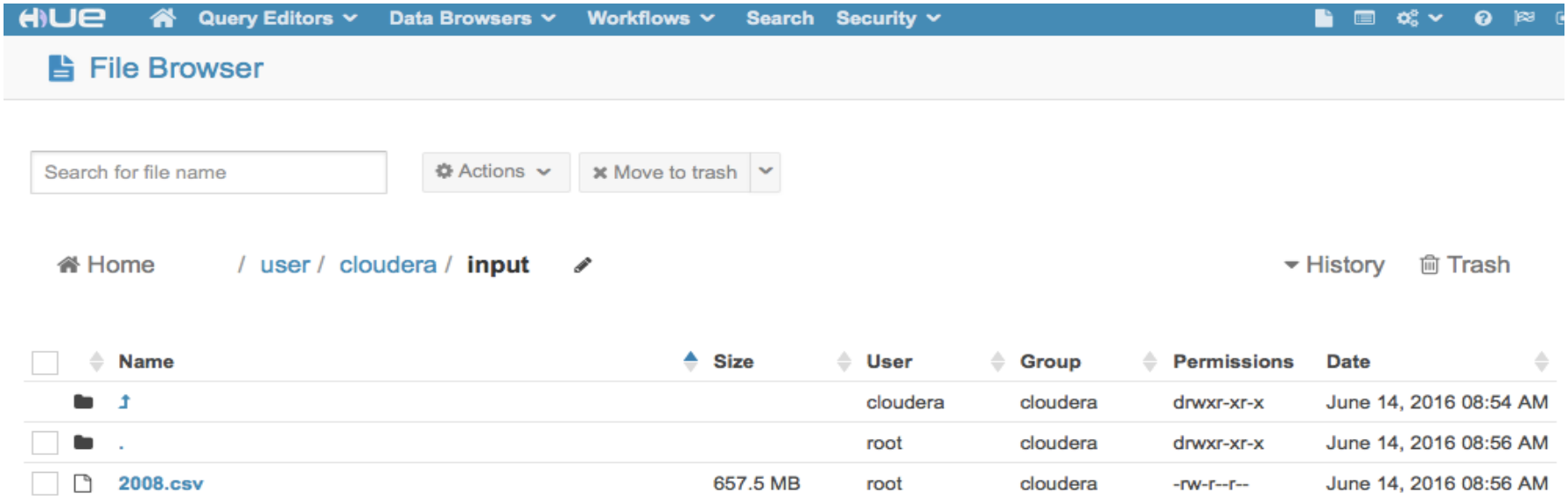
http://projects.fivethirtyeight.com/flights/



Spark Program : Upload Flight Data

Load a dataset from the Amazon and Upload a dataset to HDFS

```
[root@quickstart /]# wget
https://s3.amazonaws.com/imcbucket/data/flights/2008.csv
[root@quickstart /]# hadoop fs -put 2008.csv
/user/cloudera/input
[root@quickstart /]# pyspark
```



Spark Program : Navigating Flight Data

```
>>> airline =
sc.textFile("hdfs:///user/cloudera/input/2008.csv")
>>> airline.take(2)
```

```
[u'Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,UniqueC
arrier,FlightNum,TailNum,ActualElapsedTime,CRSElapsedTime,AirTime,ArrDelay,DepDe
lay,Origin,Dest,Distance,TaxiIn,TaxiOut,Cancelled,CancellationCode,Diverted,Carr
ierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay', u'2008,1,3,4,20
03,1955,2211,2225,WN,335,N712SW,128,150,116,-14,8,IAD,TPA,810,4,8,0,,0,NA,NA,NA,
NA,NA']
```

Spark Program : Preparing Data

```
>>> header_line = airline.first()
>>> header_list = header_line.split(',')
>>> airline_no_header = airline.filter(lambda row: row !=
header_line)
>>> airline_no_header.first()
>>> def make_row(row):
...     row_list = row.split(',')
...     d = dict(zip(header_list,row_list))
...     return d
...
>>> airline_rows = airline_no_header.map(make_row)
>>> airline_rows.take(5)
```

Spark Program : Define convert_float function

```
>>> def convert_float(value):
...     try:
...             x = float(value)
...             return x
...     except ValueError:
...             return 0
...
>>>
```

Spark Program : Finding best/worst airline

```
>>> carrier_rdd = airline_rows.map(lambda
row:(row['UniqueCarrier'],convert_float(row['ArrDelay'])))
>>> carrier_rdd.take(2)
```

```
16/06/17 17:48:06 INFO scheduler.DAGScheduler: Job 5 finished: runJob at PythonRDD.s
cala:393, took 0.062345 s
[(u'WN', -14.0), (u'WN', 2.0)]
```

Spark Program : Finding best/worst airlines

```
>>> mean_delays_dest =
carrier_rdd.groupByKey().mapValues(lambda delays:
sum(delays)/len(delays))
>>> mean_delays_dest.sortBy(lambda t:t[1],
ascending=True).take(10)
>>> mean_delays_dest.sortBy(lambda t:t[1],
ascending=False).take(10)
```

```
In [11]: mean_delays_dest = carrier_rdd.groupByKey().mapValues(lambda delays: sum(delays.data)/len(delays.data))

In [12]: mean_delays_dest.sortBy(lambda t:t[1], ascending=True).take(10)

Out[12]: [(u'AQ', -2.8708974358974357),
          (u'HA', 1.2518519716624075),
          (u'US', 2.800998260539828),
          (u'9E', 3.987490846961191),
          (u'AS', 4.721360405553864),
          (u'WN', 5.115703380225903),
          (u'F9', 6.084135669681085),
          (u'OO', 6.43893863978179),
          (u'NW', 7.293465879672776),
          (u'DL', 7.716164635751918)]

In [13]: mean_delays_dest.sortBy(lambda t:t[1], ascending=False).take(10)

Out[13]: [(u'AA', 12.202853434950445),
          (u'OH', 11.404110178283158),
          (u'YV', 11.322566979170753),
          (u'UA', 11.001550560048052),
          (u'B6', 10.859381613638567),
          (u'CO', 10.809820575966226),
          (u'XE', 10.320298523403915),
          (u'EV', 10.00033146217589),
          (u'MQ', 9.496970610952266),
          (u'FL', 8.988157472371256)]
```