

# Spark

## Cluster computing

---

J. Fernando Sánchez, Joaquín Salvachúa, Gabriel Huecas

2016

Universidad Politécnica de Madrid



# Background

---

# LISP and functional programming

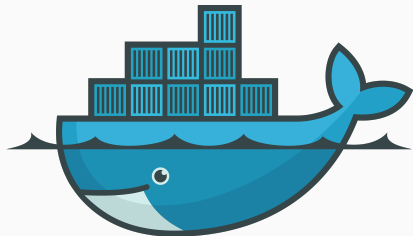
- Higher level programming
- Avoid side effects
- Pattern matching





- *A better Java*
- Functional programming (optional)
- Actors for (coarse) concurrency

- Easy and repeatable deployment
- Lots of pre-built images @ [hub.docker.com](https://hub.docker.com)
- Building block for other tools (swarm, compose, machine...)




docker



**I KNOW SCALA**

memegenerator.net



**I KNOW BIG DATA**

memegenerator.net





Show me.

# Word count in Wikipedia

Problem: find the frequency each word is used in Wikipedia.

We have the text of all wikipedia in a text file<sup>1</sup>. It begins like this:

```
hi world
```

```
hi
```

```
Scala (SKAH-lah) [9] is a general-purpose  
programming language. Scala has full support for  
functional programming and a strong static type  
system. Designed to be concise, [10] many of Scala  
's design decisions were inspired by criticism of  
Java's shortcomings. [8]
```

# Algorithm

- Read every line
- Chunk every line into words
- Count every occurrence
- For every word, sum its occurrences

## Possible results of every step

```
(("hi world"), ("hi") ...)
```

```
List((hi, 1), (hi, 1), (world, 1) ...)
```

```
Map(hi ->(("hi", 1), (hi, 1)),  
     world -> ((world, 1)) ...)
```

```
Map(hi-> 2, world -> 1 ...)
```

## Possible results of every step

```
(("hi world"), ("hi") ...)
```

```
List((hi, 1), (hi, 1), (world, 1) ...)
```

```
Map(hi ->(("hi", 1), (hi, 1)),  
     world -> ((world, 1)) ...)
```

```
Map(hi-> 2, world -> 1 ...)
```

## Possible results of every step

```
(("hi world"), ("hi") ...)
```

```
List((hi, 1), (hi, 1), (world, 1) ...)
```

```
Map(hi ->(("hi", 1), (hi, 1)),  
     world -> ((world, 1)) ...)
```

```
Map(hi-> 2, world -> 1 ...)
```

## Possible results of every step

```
(("hi world"), ("hi") ...)
```

```
List((hi, 1), (hi, 1), (world, 1) ...)
```

```
Map(hi ->(("hi", 1), (hi, 1)),  
     world -> ((world, 1)) ...)
```

```
Map(hi -> 2, world -> 1 ...)
```

## Running the scala shell

We will use docker.

```
docker run -it -v $PWD:Wikipedia.txt:Wiki \  
    --rm williamyeh/scala
```



```
import scala.io.Source
val wiki = Source.fromFile("Wiki").getLines
wiki.flatMap(line=> line.split("_"))
  .map(x=>(x, 1)).toList
  .groupBy(x => x._1)
  .map({case (k, v) =>
      (k, v.foldLeft(0)((a, b) => a+b._2))})
```

```
import scala.io.Source
val wiki = Source.fromFile("Wiki").getLines
wiki.flatMap(line=> line.split("_"))
  .map(x=>(x, 1)).toList
  .groupBy(x => x._1)
  .map({case (k, v) =>
      (k, v.foldLeft(0)((a, b) => a+b._2))})
```

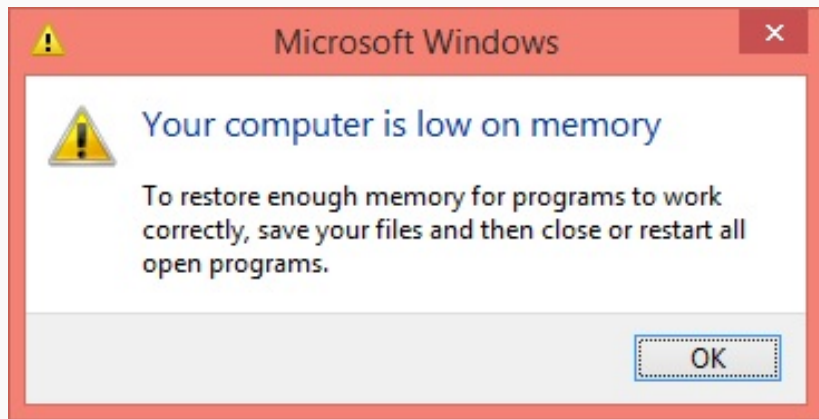
```
import scala.io.Source
val wiki = Source.fromFile("Wiki").getLines
wiki.flatMap(line=> line.split("_"))
  map(x=>(x, 1)).toList
  groupBy(x => x._1)
  map({case (k, v) =>
      (k, v.foldLeft(0)((a, b) => a+b._2))})
```

```
import scala.io.Source
val wiki = Source.fromFile("Wiki").getLines
wiki.flatMap(line=> line.split("_"))
  .map(x=>(x, 1)).toList
  .groupBy(x => x._1)
  .map({case (k, v) =>
    (k, v.foldLeft(0)((a, b) => a+b._2))})
```

```
import scala.io.Source
val wiki = Source.fromFile("Wiki").getLines
wiki.flatMap(line=> line.split("_"))
  .map(x=>(x, 1)).toList
  .groupBy(x => x._1)
  .map({case (k, v) =>
    (k, v.foldLeft(0)((a, b) => a+b._2))})
```

Let's run it in the shell.







What happened?

# Limited resources

- CPU limits our speed
  - Multi-cores help...
  - ...but real parallelism is **hard**
- RAM limits how much data you can process at the same time
  - What if you need more than 128GB?
  - You could use more than one computer...
  - ... but cluster computing is even harder than “local” parallelism
- Functional programming helps a bit

But... this was supposed to be  
fun, wasn't it?

# Introduction to Spark

---

**Apache Spark™** is a fast and general engine for large-scale data processing.

## Speed

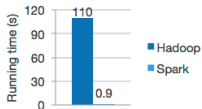
Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Apache Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

## Ease of Use

Write applications quickly in Java, Scala, Python, R.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python and R shells.



Logistic regression in Hadoop and Spark

```
text_file = spark.textFile("hdfs://...")
```

```
text_file.flatMap(lambda line: line.split())  
    .map(lambda word: (word, 1))  
    .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API

### Latest News

Spark wins CloudSort Benchmark as the most efficient engine (Nov 15, 2016)

Spark 2.0.2 released (Nov 14, 2016)

Spark 1.6.3 released (Nov 07, 2016)

Spark 2.0.1 released (Oct 03, 2016)

[Archive](#)

[Download Spark](#)

### Built-in Libraries:

[SQL and DataFrames](#)  
[Spark Streaming](#)  
[MLlib \(machine learning\)](#)  
[GraphX \(graph\)](#)

### Third-Party Packages

Apache Spark™ is a fast and general engine for large-scale data processing.<sup>2</sup>

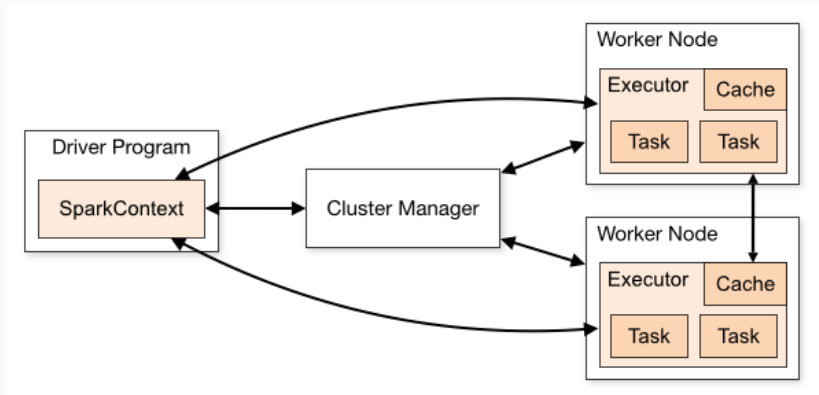
On top of that:

- Open source (Top-level Apache project)
- Plays well with other tools

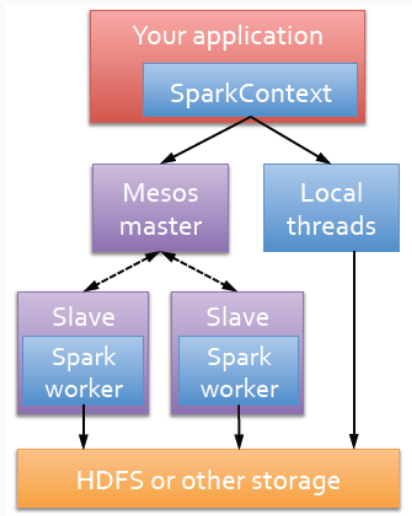
---

<sup>2</sup><http://spark.apache.org>

# Architecture

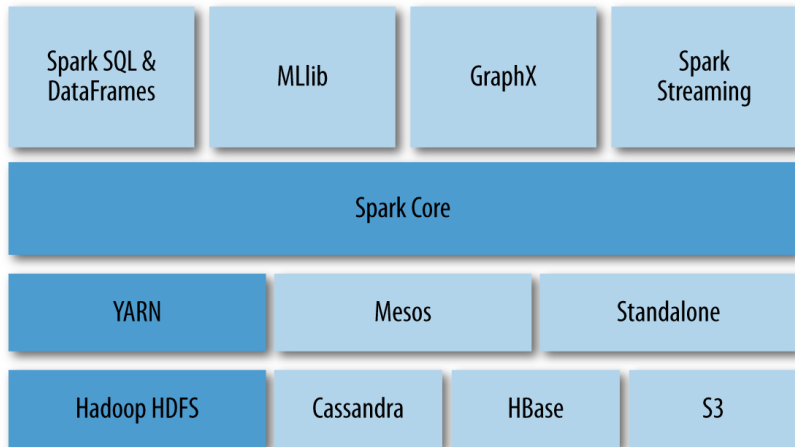


# Programs



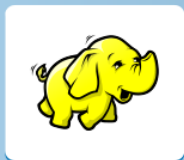


# Ecosystem





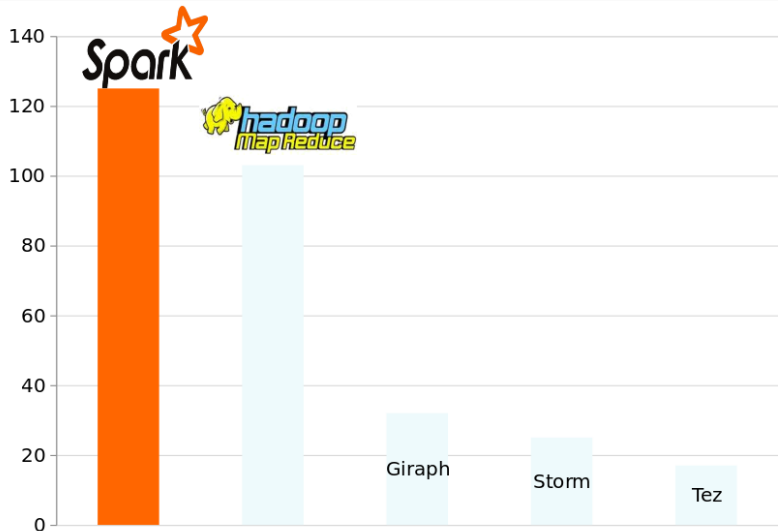
VS.



## Comparison to MapReduce

- In-memory data
  - Less i/o overhead
  - Faster operations
  - Caching
- Better for recursive tasks (e.g. machine learning)
- Some libraries are dropping MapReduce support

# Contributors to Spark/Hadoop 2014



# Project status

apache / spark

mirrored from [git://git.apache.org/spark.git](https://git.apache.org/spark.git)

Watch 1,507

Star 10,735

Fork 10,023

Code

Pull requests 451

Projects 0

Pulse

Graphs

Contributors

Commits

Code frequency

Punch card

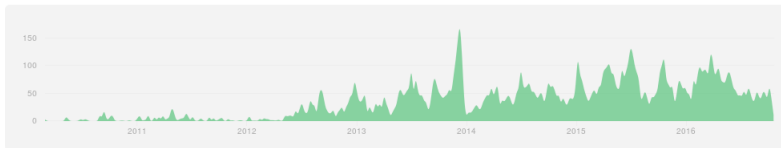
Network

Members

Mar 28, 2010 – Nov 15, 2016

Contributions: Commits

Contributions to master, excluding merge commits



# Overview

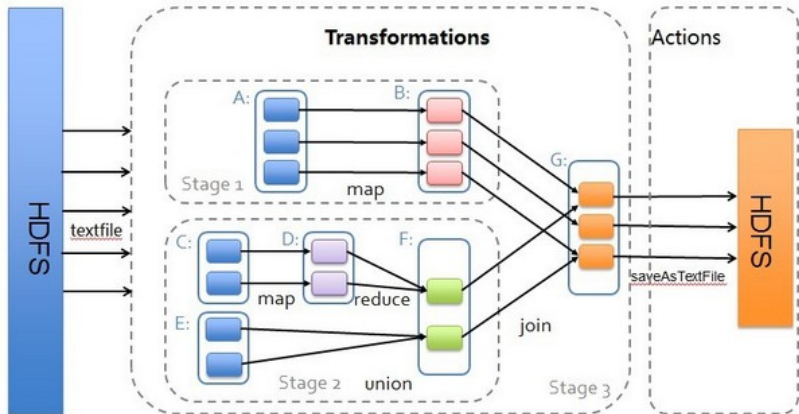
## Data (RDDs/Datasets)

- RDD: Resilient Distributed Dataset
- Collections of objects spread across a cluster, stored in RAM or on Disk
- Built through parallel transformations
- Automatically rebuilt on failure

## Operations

- Transformations (e.g. group, map, groupBy)
- Actions (e.g. count, collect, save)

# Transformations and actions



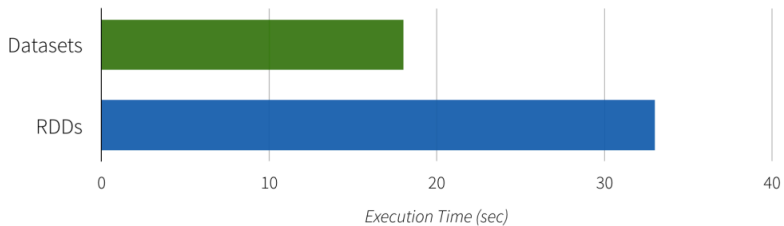
Datasets are the future

- More memory efficient
- Libraries dropping support for RDDs

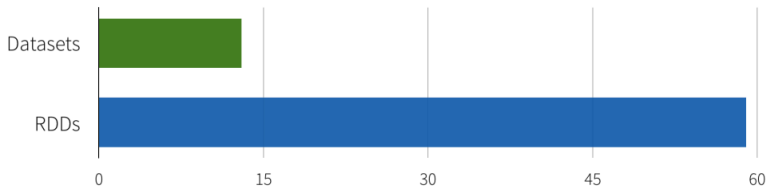


# RDDs vs Datasets

## Distributed Wordcount



## Memory Usage when Caching



### Scala

```
val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

### Python

```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

### Java

Removed, to keep the slides clean :)

sc is the Spark Context (more on this later)

## Different flavors

Language	App	REPL	Performance
Scala	Yes	Yes	😊
Java	Yes	No	😊
Python	Yes	Yes	😄

The Read-eval-print-loop (REPL) is the easiest way to get started and explore datasets. It is just a special Spark application that accepts user input (scala code).

# Working with RDDs

---

From normal data structures

```
val nums = sc.parallelize(List(1, 2, 3))  
val cont = sc.parallelize(List(("a", 1),  
                              ("a", 1),  
                              ("b", 3)))
```

From distributed/local sources

```
sc.textFile('myfile')
```

Note: sc is the spark context in the Spark interpreter

`collect()`

Reminder:

```
nums: List(1, 2, 3)
cont: List(("a", 1), ("a", 1), ("b", 3))
```

## Operations: collect

Runs any pending transformation and returns the real values

```
nums.collect()
```

```
> List(1, 2, 3)
```

```
cont.collect()
```

```
> List((a, 1), (a, 1), (b, 3))
```

Reminder:

```
nums: List(1, 2, 3)
```

```
cont: List(("a", 1), ("a", 1), ("b", 3))
```

# take(N)

Reminder:

```
nums: List(1, 2, 3)  
cont: List(("a", 1), ("a", 1), ("b", 3))
```



## Operations: take

Returns the N first elements

```
nums.take(2)
```

```
> List(1, 2)
```

```
cont.take(1)
```

```
> List((a, 1))
```

Reminder:

```
nums: List(1, 2, 3)
```

```
cont: List(("a", 1), ("a", 1), ("b", 3))
```

count()

Reminder:

```
nums: List(1, 2, 3)
cont: List(("a", 1), ("a", 1), ("b", 3))
```

## Operations: count

Returns the number of elements in a collection

```
nums.count()
```

```
> 3
```

```
cont.count()
```

```
> 3
```

Reminder:

```
nums: List(1, 2, 3)
```

```
cont: List(("a", 1), ("a", 1), ("b", 3))
```

`filter(fn)`

Reminder:

```
nums: List(1, 2, 3)  
cont: List(("a", 1), ("a", 1), ("b", 3))
```

### `filter(fn)`

This time, we need to define a function.

Filter applies that function to every element, and returns those where the function returns true.

For example:

```
val fn = (x:Int) => x > 1
```

Reminder:

```
nums: List(1, 2, 3)  
cont: List(("a", 1), ("a", 1), ("b", 3))
```

## Operations: filter

```
val fn = (x: Int) => x > 1
```

Return a list containing the values where the function returns true

```
nums.filter(fn)
```

```
> List(2, 3)
```

Reminder:

```
nums: List(1, 2, 3)
```

```
cont: List(("a", 1), ("a", 1), ("b", 3))
```

## Quick aside: anonymous functions and underscores

In scala, we can define “anonymous functions”, also known as lambda functions.

## Quick aside: anonymous functions and underscores

In scala, we can define “anonymous functions”, also known as lambda functions.

```
val fn = (x:Int) => x > 1  
cont.filter(fn)
```

is equivalent to:

```
cont.filter((x:Int) => x>1)
```



## Quick aside: anonymous functions and underscores

```
val fn = (x:Int)) => x > 1  
cont.filter(fn)
```

```
cont.filter((x:Int) => x>1)
```

Additionally, the scala compiler is smart enough to infer types in this example. Hence, we could simply write:

```
cont.filter(x => x>1)
```

## Quick aside: anonymous functions and underscores

```
val fn = (x:Int) => x > 1  
cont.filter(fn)
```

```
cont.filter((x:Int) => x>1)
```

```
cont.filter(x => x>1)
```

Furthermore, we could use underscores to replace arguments:

## Quick aside: anonymous functions and underscores

Every new argument in the lambda function represents a parameter

Hence, these two expressions are equivalent

$$\lambda x, y. x + y \Rightarrow x + y$$

Our last example could be written more concisely as:

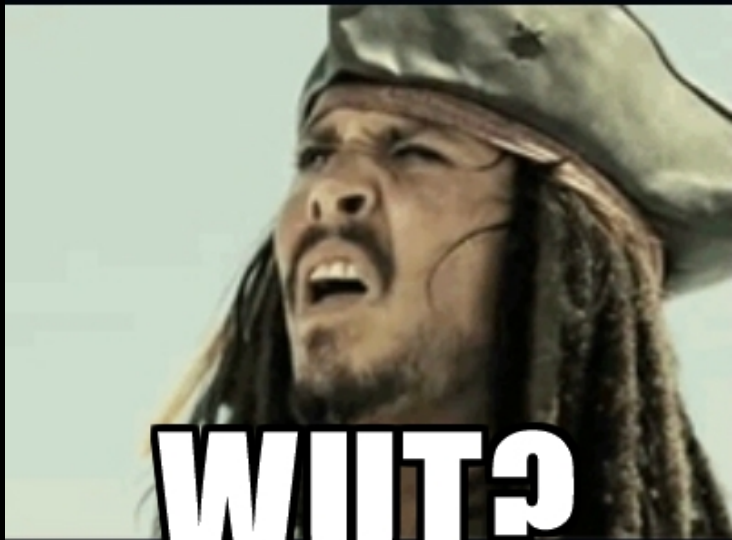
```
nums.filter(>1)  
> List(2, 3)
```

What would this filter do?

```
nums.filter(_. _1 == "a" && _._1 == 1)
```

```
> ???
```

```
<console>:9: error: missing parameter type  
for expanded function ((x$1, x$2) =>  
x$1._1.$eq$a.$amp$a(x$2._1.$eq$1))  
Note: The expected type requires a  
one-argument function accepting a 2-Tuple.
```



**WUT?**

## Operations: filter

Remember, each new underscore represents a new argument.  
So that expression expands to:

```
nums.filter((x, y) => x._1 == "a" &&  
             y._2 == 1)
```



## Operations: filter

The right expression is:

```
nums.filter(x => x._1 == "a" &&  
              x._2 == 1)  
> List((a, 1), (a, 1))
```

map(fn)

Reminder:

```
nums: List(1, 2, 3)  
cont: List(("a", 1), ("a", 1), ("b", 3))
```

## Operations: map

Apply a function to every item in the list

```
cont.map(x._2)
```

```
> List(1, 1, 3)
```

```
nums.map(_*3)
```

```
> List(3, 6, 9)
```

Reminder:

```
nums: List(1, 2, 3)
```

```
cont: List(("a", 1), ("a", 1), ("b", 3))
```

reduce(fn)

Reminder:

```
nums: List(1, 2, 3)  
cont: List(("a", 1), ("a", 1), ("b", 3))
```

## Operations: reduce

Merge elements with an associative function (concisely)

```
nums.reduce(_+_)
```

```
> 6
```

```
cont.reduce((x, y) => (x._1+y._1,  
                      x._2*y._2))
```

```
> (aab, 3)
```

Reminder:

```
nums: List(1, 2, 3)
```

```
cont: List(("a", 1), ("a", 1), ("b", 3))
```

# groupByKey()

Reminder:

```
nums: List(1, 2, 3)  
cont: List(("a", 1), ("a", 1), ("b", 3))
```

# Operations

Group elements of a list by the first item in the tuple

```
cont.groupByKey()
```

```
> [(b, [3]), (a, [1,1])]
```

Reminder:

```
nums: List(1, 2, 3)
```

```
cont: List(("a", 1), ("a", 1), ("b", 3))
```

## reduceByKey(fn)

Reminder:

```
nums: List(1, 2, 3)  
cont: List(("a", 1), ("a", 1), ("b", 3))
```

---

<sup>3</sup>Databricks' post on avoiding groupByKey



# Operations

Group by key and reduce each value

```
cont.reduceByKey((x,y)=>x+y)
```

```
> [(b,3), (a,2)]
```

reduceByKey is more efficient than applying group, map and reduce separately. The reduce function can be given to each worker, which avoids passing unnecessary data. <sup>3</sup>

Reminder:

```
nums: List(1, 2, 3)
```

```
cont: List(("a", 1), ("a", 1), ("b", 3))
```

<sup>3</sup>Databricks' post on avoiding groupByKey

And once you are done, save your results to a file.

```
nums.saveAsTextFile("hdfs://file.txt")
```

## Example: Search in logs

```
val lines = sc.textFile("hdfs://...")
val errors = lines.filter(s =>
    s.startsWith("ERROR"))
val messages = errors.map(s => s.split("\t")._2)
messages.cache()
messages.filter(s => s.contains("mysql")).count()
messages.filter(s => s.contains("php")).count()
```

# Using Spark

---

## Get the repo

```
git clone http://github.com/gettyimages/docker-spark  
cd docker-spark  
docker-compose up  
docker exec -it dockerspark_master_1 bin/spark-shell
```

Move to the repo

```
git clone http://github.com/gettyimages/docker-spark  
cd docker-spark  
docker-compose up  
docker exec -it dockerspark_master_1 bin/spark-shell
```

### Run all the containers

```
git clone http://github.com/gettyimages/docker-spark  
cd docker-spark  
docker-compose up  
docker exec -it dockerspark_master_1 bin/spark-shell
```

Launch spark-shell inside the master container

```
git clone http://github.com/gettyimages/docker-spark  
cd docker-spark  
docker-compose up  
docker exec -it dockerspark_master_1 bin/spark-shell
```



# Demo

```
worker_1 | 16/11/16 11:53:32 INFO worker.Worker: Connecting to master master:7077...
worker_1 | 16/11/16 11:53:32 INFO handler.ContextHandler: Started o.s.j.s.ServletContextH
ndler@77f93121{/metrics/json,null,AVAILABLE}
worker_1 | 16/11/16 11:53:32 INFO client.TransportClientFactory: Successfully created con
nection to master/172.17.0.2:7077 after 25 ms (0 ms spent in bootstraps)
master_1 | 16/11/16 11:53:32 INFO master.Master: Registering worker 172.17.0.3:8881 with
cores, 1024.0 MB RAM
worker_1 | 16/11/16 11:53:32 INFO worker.Worker: Successfully registered with master spar
://master:7077
master_1 | 16/11/16 11:53:47 INFO master.Master: Registering app Spark shell
worker_1 | 16/11/16 11:53:48 INFO spark.SecurityManager: Changing view acls to: root
worker_1 | 16/11/16 11:53:48 INFO spark.SecurityManager: Changing modify acls to: root
worker_1 | 16/11/16 11:53:48 INFO spark.SecurityManager: Changing view acls groups to:
worker_1 | 16/11/16 11:53:48 INFO spark.SecurityManager: Changing modify acls groups to:
worker_1 | 16/11/16 11:53:48 INFO worker.ExecutorRunner: Launch command: "/usr/jdk1.8.0_9
/bin/java" "-cp" "a"/conf:/usr/spark-2.0.1/jars/*:/usr/hadoop-2.7.2/etc/hadoop:/usr/hadoop
```



```
Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_92)
Type in expressions to have them evaluated.
Type :help for more information.
```

```
scala> val a = sc.parallelize(List(1,2,3,4))
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24
```

# Compose.yml Master

master:

```
image: gettyimages/spark
```

```
command: bin/spark-class org.apache.spark.deploy.
```

```
    ↪ master.Master -h master
```

```
hostname: master
```

```
environment:
```

```
    MASTER: spark://master:7077
```

```
    SPARK_CONF_DIR: /conf
```

```
    SPARK_PUBLIC_DNS: localhost
```

```
    ... bunch of ports ...
```

```
volumes:
```

```
    - ./conf/master:/conf
```

# Compose.yml Master

```
master:
  image: gettyimages/spark
  command: bin/spark-class org.apache.spark.deploy.
    ↪ master.Master -h master
  hostname: master
  environment:
    MASTER: spark://master:7077
    SPARK_CONF_DIR: /conf
    SPARK_PUBLIC_DNS: localhost
    ... bunch of ports ...
  volumes:
    - ./conf/master:/conf
```

# Compose.yml Master

```
master:
  image: gettyimages/spark
  command: bin/spark-class org.apache.spark.deploy.
    ↪ master.Master -h master
  hostname: master
  environment:
    MASTER: spark://master:7077
    SPARK_CONF_DIR: /conf
    SPARK_PUBLIC_DNS: localhost
    ... bunch of ports ...
  volumes:
    - ./conf/master:/conf
```

## Compose.yml Master

```
master:
  image: gettyimages/spark
  command: bin/spark-class org.apache.spark.deploy.
    ↪ master.Master -h master
  hostname: master
  environment:
    MASTER: spark://master:7077
    SPARK_CONF_DIR: /conf
    SPARK_PUBLIC_DNS: localhost
    ... bunch of ports ...
  volumes:
    - ./conf/master:/conf
```

---

# Compose.yml Worker

worker:

```
image: gettyimages/spark
```

```
command: bin/spark-class org.apache.spark.deploy.
```

```
  ↪ worker.Worker spark://master:7077
```

```
hostname: worker
```

```
environment:
```

```
  SPARK_CONF_DIR: /conf
```

```
  SPARK_WORKER_CORES: 2
```

```
  SPARK_WORKER_MEMORY: 1g
```

```
links:
```

```
  - master
```

```
volumes:
```

# Compose.yml Worker

```
worker:  
  image: gettyimages/spark  
  command: bin/spark-class org.apache.spark.deploy.  
    ↪ worker.Worker spark://master:7077  
  hostname: worker  
  environment:  
    SPARK_CONF_DIR: /conf  
    SPARK_WORKER_CORES: 2  
    SPARK_WORKER_MEMORY: 1g  
  links:  
    - master  
  volumes:
```

# Compose.yml Worker

```
worker:  
  image: gettyimages/spark  
  command: bin/spark-class org.apache.spark.deploy.  
    ↪ worker.Worker spark://master:7077  
  hostname: worker  
  environment:  
    SPARK_CONF_DIR: /conf  
    SPARK_WORKER_CORES: 2  
    SPARK_WORKER_MEMORY: 1g  
  links:  
    - master  
  volumes:
```

---



# Compose.yml Worker

```
worker:  
  image: gettyimages/spark  
  command: bin/spark-class org.apache.spark.deploy.  
    ↪ worker.Worker spark://master:7077  
  hostname: worker  
  environment:  
    SPARK_CONF_DIR: /conf  
    SPARK_WORKER_CORES: 2  
    SPARK_WORKER_MEMORY: 1g  
  links:  
    - master  
  volumes:
```

- `./data` folder is mounted as `/tmp/data`
  - Copy your datasets there
  - Load them in the shell: `sc.textFile("/tmp/data/...")`
- Master Web UI (localhost:8080)
- Worker Web UI (localhost:8081)
- REPL Web UI (localhost:4040 when launched)

**A word of caution:** as any other app, the shell reserves resources on startup, whether you are using them or not.

## Steps:

- Write the code
- Compile the jar
- Make your data available to every node in the cluster
- Submit it to your cluster

## Example application

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SparkWordCount {
  def main(args: Array[String]) {
    // create Spark context with Spark configuration
    val sc = new SparkContext(new SparkConf().
      ↪ setAppName("Spark_Example"))
    ... Your program ...
  }
```

## Running an application

```
./bin/spark-submit --class <main-class> \  
  --master <master-url> \  
  --deploy-mode <deploy-mode> \  
  --conf <key>=<value> \  
  ... # other options  
<application-jar> \  
[application-arguments]
```

## Full examples

---

# Word frequency in wikipedia, revisited

## Spark

```
val wiki = sc.textFile("Wikipedia.txt")
val counts = wiki.flatMap(line=> line.split("_"))
                  map(word => (word, 1))
                  reduceByKey(_ + _)
```

## Pure scala

```
val wiki = scala.io.Source.fromFile("Wiki").getLines
wiki.flatMap(line=> line.split("_"))
    map(x=>(x, 1)).toList
    groupBy(x => x. 1)
```

Shall we try it in the shell?





## Spark is not magic

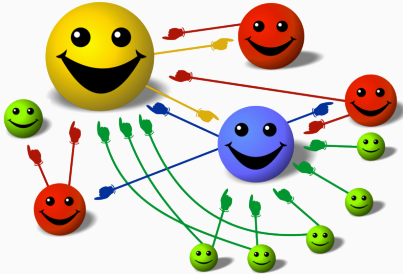
- We still have to add more resources
- Caching may cause the spark version to use **more memory** (this can be configured)

## Spark is not magic

- We still have to add more resources
- Caching may cause the spark version to use **more memory** (this can be configured)

However, it allows us to scale our application

# Page rank



- Created by Google
- Rank given by links and their importance
- Iterative (Perfect for Spark!)

$$\text{PageRank of site} = \sum \frac{\text{Page rank of inbound link}}{\text{Number of links on that page}}$$

OR

$$PR(u) = (1 - d) + d \times \sum \frac{PR(v)}{N(v)}$$

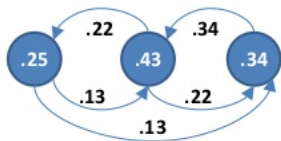
# PageRank toy example



Superstep 0

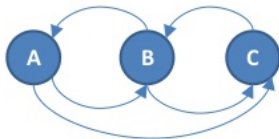


Superstep 1



Superstep 2

Input graph



# Spark Program

```
val links = // RDD of (url, neighbors) pairs
var ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  val contribs = links.join(ranks).flatMap {
    case (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  ranks = contribs.reduceByKey(_ + _)
                    .mapValues(0.15 + 0.85 * _)
}

ranks.saveAsTextFile(...)
```

Next week on SIBD

---

## Next week on SIBD

- Advanced Spark configuration
- Multiple hosts
- Spark ecosystem
- More examples in IBM BlueMix



## Acknowledgements and useful links

---

- Spark programming guide
- Databricks introducing apache spark datasets
- Data Analytics with Hadoop: In-Memory Computing with Spark