

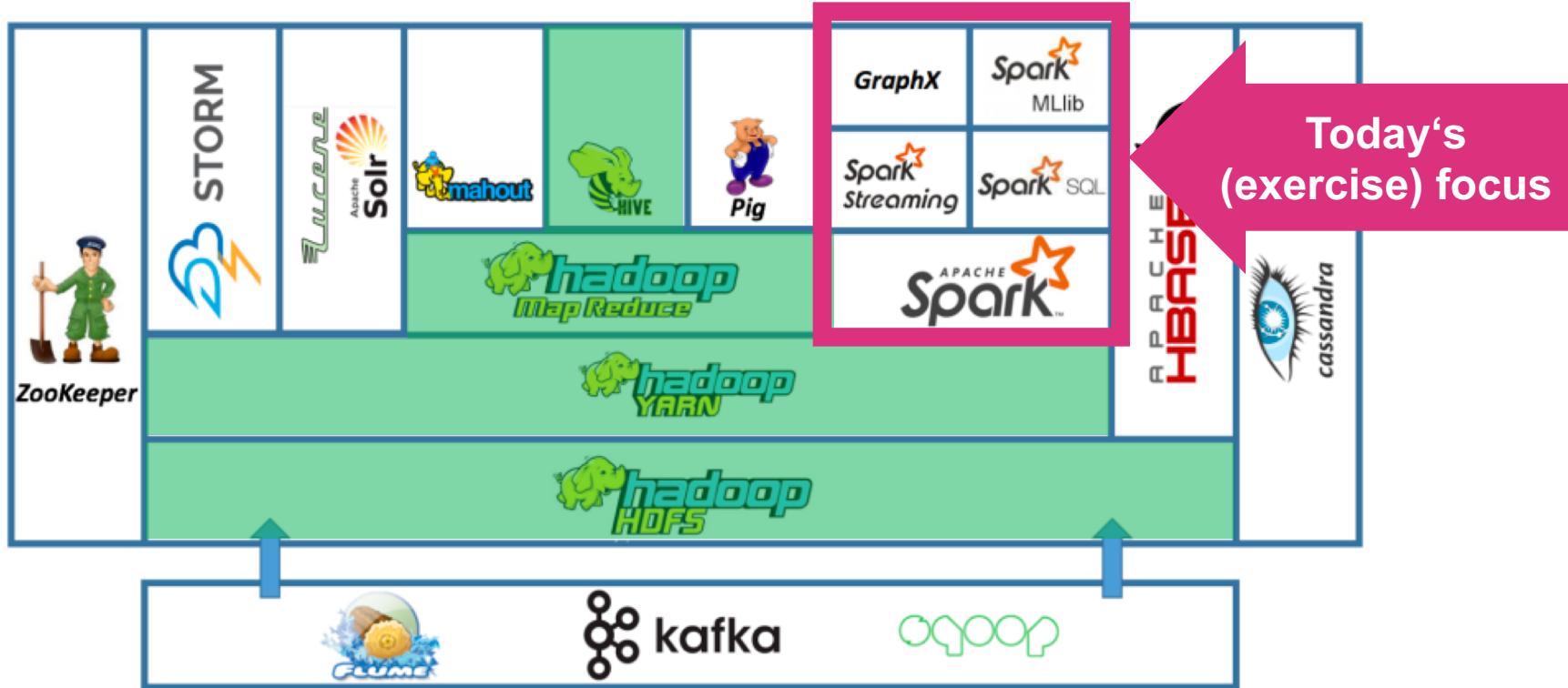
HandsOn – Spark, Scala, PySpark and Jupyter

A quick Introduction to Spark, Scala, PySpark and
Jupyter



www.marcel-mittelstaedt.com

The Hadoop Ecosystem



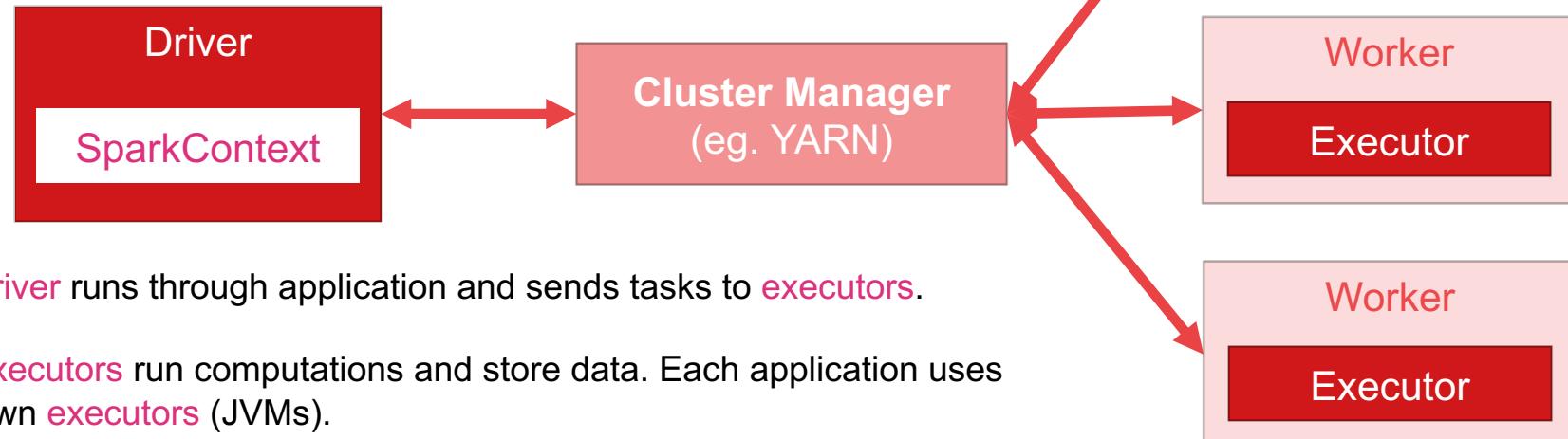
Spark

API Languages:	Java	Scala	Python	R
APIs and Libraries:	Spark SQL	Spark Streaming	Mlib	GraphX
Spark Core				
Ressource/ Cluster Manager:	Standalone	YARN	Mesos	Kubernetes
Data Source APIs:	HDFS, ElasticSearch, Amazon S3, Azure, Redis, Riak, Couchbase, MongoDB, SQL(Hive, Avro, CSV, Parquet, JDBC DB...)			



Spark – Execution Process

1. Spark applications starts and instantiates **SparkContext** (JVM process).
2. Spark acquires **executors** on worker nodes from cluster manager.
3. Cluster manager launches **executors**.



4. **Driver** runs through application and sends tasks to **executors**.
5. **Executors** run computations and store data. Each application uses its own **executors** (JVMs).
6. If any worker crashes, ist tasks will be send to another **executor**.

Spark – RDDs, DataFrame and DataSet

Supported by:

RDD
(2011)

Scala, Java, Python

Idea:

- RDD = immutable **distributed** collection of data
- **partitioned across nodes** of cluster
- can be **operated in parallel** with a low-level API

Typed: typed, no schema

Use for: unstructured data

DataFrame
(2013)

Scala, Java, Python

- based on RDD
- immutable **distributed** collection of data
- but **organized into columns**
- higher level abstraction

untyped, schema

semi-structured and structured data

DataSet
(2015)

Scala, Java

- based on DataFrame API, but type-safe
- immutable **distributed** collection of data
- high-level API
- converted into optimized RDD

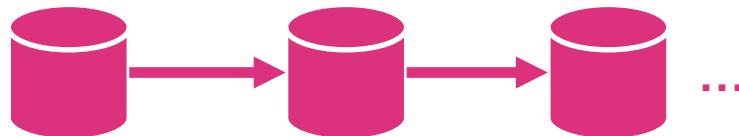
typed, schema

structured data



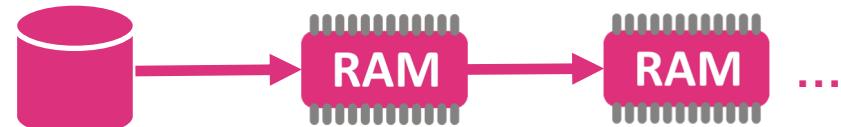
Hadoop MapReduce vs Spark

Hadoop MapReduce



- performs **read** from **HDFS (HDD)** before **every computation**
- performs **write** on **HDFS (HDD)** after **every computation**
- using distributed **disk space**

Spark



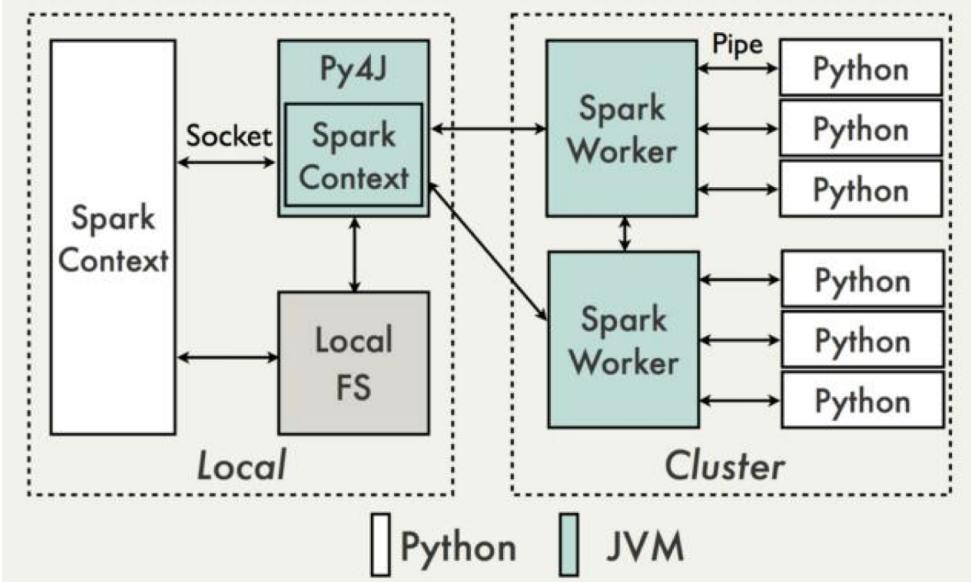
- performs **read** from **RAM** before **every computation (except first)**
- performs **write** on **RAM** after **every computation**
- using distributed **memory**



PySpark

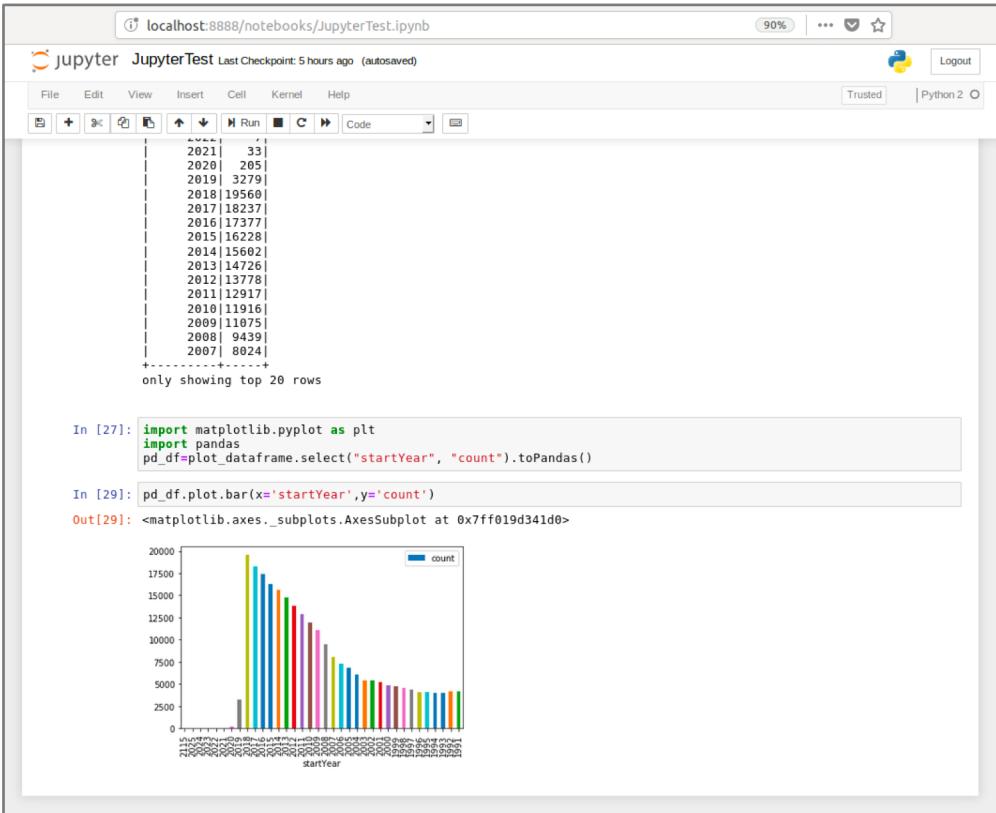


Data Flow



- built on-top of Spark's Java API
- data is processed in Python and cached/shuffled within the JVM
- Spark executors on the cluster start Python interpreter to execute user code
- A Python RDD corresponds to an RDD in the local JVM
- e.g. **sc.textFile()** in Python will call JavaSparkContext **textFile()**

Jupyter (Notebooks)



The screenshot shows a Jupyter Notebook interface running on localhost:8888/notebooks/JupyterTest.ipynb. The notebook has a title "JupyterTest" and a subtitle "Last Checkpoint: 5 hours ago (autosaved)". The top menu includes File, Edit, View, Insert, Cell, Kernel, and Help. The toolbar includes Run, Cell, Kernel, Help, and Code buttons. The status bar shows "Trusted" and "Python 2".

In [27]:

```
import matplotlib.pyplot as plt
import pandas
pd_df=plt_dataframe.select("startYear", "count").toPandas()
```

In [29]:

```
pd_df.plot.bar(x='startYear',y='count')
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff019d341d0>
```

A histogram plot titled "startYear" with "count" on the y-axis (ranging from 0 to 20,000) and "startYear" on the x-axis. The bars are colored in a gradient.

startYear	count
2021	33
2020	205
2019	3279
2018	19560
2017	18237
2016	17377
2015	16228
2014	15602
2013	14726
2012	13778
2011	12917
2010	11916
2009	11075
2008	9439
2007	8024

- Interactive Web IDE
- Kernel-based Notebooks
- Open-source
- Create and share:
 - code,
 - visualizations and
 - narrative text like documentation
- Supports:
 - Python
 - R
 - Scala
 - ...
- Works well with Spark



Exercises Preparation I

Install and Setup Spark



www.marcel-mittelstaedt.com

Download And Install Spark

1. Download Spark 2.3.2 (for Hadoop 2.7++)

```
wget http://mirror.funkfreundelandshut.de/apache/spark/spark-2.3.2/spark-2.3.2-bin-hadoop2.7.tgz
```

2. Extract and Move:

```
tar -xzf http://mirror.funkfreundelandshut.de/apache/spark/spark-2.3.2/spark-2.3.2-bin-hadoop2.7.tgz
```

```
mv spark-2.3.2-bin-hadoop2.7/ spark
```



Configure Spark

1. Setup spark-env.sh

```
cp spark/conf/spark-env.sh.template spark/conf/spark-env.sh
```

```
vi spark/conf/spark-env.sh
```

```
export SPARK_MASTER_IP=localhost
export SPARK_WORKER_CORES=1
export SPARK_WORKER_MEMORY=800m
export SPARK_WORKER_INSTANCES=1
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
SPARK_LOCAL_IP="localhost"
```



Configure Spark

2. Setup slave

```
cp spark/conf/slaves.template spark/conf/slaves
```

```
vi spark/conf/slaves
```

```
# A Spark Worker will be started on each of the machines listed below.  
localhost
```



Configure Spark

3. Setup Environment Variables

```
vi .bashrc
```

```
export SPARK_HOME=/home/hadoop/spark  
export PATH=$SPARK_HOME/bin:$PATH
```

```
source .bashrc
```



Configure Yarn (Yarn Java 8 Bug)

<https://issues.apache.org/jira/browse/YARN-4714>

<https://stackoverflow.com/questions/38988941/running-yarn-with-spark-not-working-with-java-8>

1. Setup yarn-site.xml

```
vi hadoop/etc/hadoop/yarn-site.xml
```

```
<property>
    <name>yarn.nodemanager.pmem-check-enabled</name>
    <value>false</value>
</property>
<property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
</property>
```



Configure Yarn

2. Restart HDFS and Yarn

```
stop-all.sh
```

```
start-dfs.sh
```

```
start-yarn.sh
```



Start Spark (on Yarn)

1. Start Spark Shell:

```
spark-shell --master yarn
```

```
Spark context Web UI available at http://localhost:4040
Spark context available as 'sc' (master = yarn, app id = application_1539278841328_0001).
Spark session available as 'spark'.
Welcome to
```

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_181)
Type in expressions to have them evaluated.
Type :help for more information.

scala>



Start Spark (on Yarn) – Test Install

2. Create Scala Collection `dummy_data` of Numbers 1 – 100:

```
scala> val dummy_data = 1 to 100
dummy_data: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6
, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100)
```

3. Create a Scala RDD `dummy_RDD` of previous collection:

```
scala> val dummy_RDD = sc.parallelize(dummy_data)
dummy_RDD: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallel
ize at <console>:26
```



Start Spark (on Yarn) – Test Install

4. Run simple filter transformation and `collect()` to calculate results:

```
scala> dummy_RDD.filter(_ < 10).collect()
res0: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

5. Now you should see a completed stage in Spark Shell UI (`localhost:4040`):

The screenshot shows the Spark Shell application UI at `marcel-virtualbox:8088/proxy/application_1539278841328_0002/`. The UI has a header with a back arrow, forward arrow, refresh, home, and search icons. The top bar shows battery level (67%), signal strength, and a star icon. Below the header is a navigation bar with tabs: Jobs (selected), Stages, Storage, Environment, and Executors. To the right of the navigation bar is the text "Spark shell application UI". The main content area is titled "Spark Jobs (?)". It displays user information: User: hadoop, Total Uptime: 11 min, Scheduling Mode: FIFO, and Completed Jobs: 1. There is a link to "Event Timeline". Below this is a section titled "Completed Jobs (1)". A table lists the completed job details:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at <console>:26 collect at <console>:26	2018/10/11 20:17:22	0.8 s	1/1	2/2



Start Spark (on Yarn) – Test Install

6. As well as the whole time the Spark Shell Container Running on Yarn:

The screenshot shows the Hadoop YARN web interface at `localhost:8088/cluster/apps/RUNNING`. The title bar says "RUNNING Applications". The left sidebar has sections for Cluster Metrics, Cluster Nodes Metrics, Scheduler Metrics, and Tools. The main content area displays the "Cluster Metrics" table with the following data:

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total
2	0	1	1	3	5 GB	8 GB	0 B	3	8

The "Cluster Nodes Metrics" table shows 1 active node, 0 decommissioning nodes, 0 decommissioned nodes, 0 lost nodes, 0 unhealthy nodes, and 0 rebooted nodes.

The "Scheduler Metrics" table shows the Capacity Scheduler with the following configuration:

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Capacity
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

The "Applications" table lists one running application:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress
application_1539278841328_0002	hadoop	Spark shell	SPARK	default	0	Thu Oct 11 20:08:57 +0200 2018	N/A	RUNNING	UNDEFINED	3	3	5120	0	0	62.5	62.5	0%

At the bottom, it says "Showing 1 to 1 of 1 entries".



Start Spark (Standalone) – Test Install

1. Start Spark Shell:

spark-shell

```
Spark context Web UI available at http://localhost:4040
Spark context available as 'sc' (master = local[*], app id = local-1539282394656)
Spark session available as 'spark'.
Welcome to
```

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_181).
Type in expressions to have them evaluated.
Type :help for more information.

scala>



Start Spark (Standalone) – Test Install

2. Run previous example:

```
scala> val dummy_data = 1 to 100
dummy_data: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100)
```

```
scala> val dummy_RDD = sc.parallelize(dummy_data)
dummy_RDD: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:26
```

```
scala> dummy_RDD.filter(_ < 10).collect()
res0: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
```



Start Spark (Standalone) – Test Install

3. Now you should see a completed stage in Spark Shell UI (*localhost:4040*):

The screenshot shows the Spark shell application UI at `localhost:4040/jobs/`. The UI has a header with a back arrow, forward arrow, refresh button, and a search bar. Below the header, there's a navigation bar with tabs for **Jobs**, **Stages**, **Storage**, **Environment**, and **Executors**. The **Jobs** tab is selected. On the right side of the header, it says "Spark shell application UI". The main content area is titled "Spark Jobs (?)". It displays the following information:

- User: hadoop
- Total Uptime: 6.2 min
- Scheduling Mode: FIFO
- Completed Jobs: 1

Below this, there's a link to "Event Timeline". The "Completed Jobs (1)" section contains a table with one row. The table columns are: Job Id, Description, Submitted, Duration, Stages: Succeeded/Total, and Tasks (for all stages): Succeeded/Total. The data in the table is:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at <console>:26 collect at <console>:26	2018/10/11 20:27:30	0.3 s	1/1	3/3



Start Spark (Standalone) – Test Install

4. And see, nothing Running On Yarn:

The screenshot shows a web browser window for the Hadoop YARN web interface at `localhost:8088/cluster/apps/RUNNING`. The title bar includes the Hadoop logo and the text "RUNNING Applications". The left sidebar has a "Cluster Metrics" section with the following table:

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total	Vcores Reserved
2	0	0	2	0	0 B	8 GB	0 B	0	8	0

Below it is a "Cluster Nodes Metrics" table:

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Under "Scheduler Metrics", there is a table for the Capacity Scheduler:

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

A search bar and a table header for "Show 20 - entries" are present. The main content area displays "No data available in table" and "Showing 0 to 0 of 0 entries".



Exercises Preparation II

Install and Setup PySpark



www.marcel-mittelstaedt.com

Start and Use PySpark (on Yarn) – Test Install

1. As PySpark is already installed, start PySpark Shell and execute previous example as Python code:

```
pyspark --master yarn
```

```
Python 2.7.15rc1 (default, Apr 15 2018, 21:51:34)
Welcome to
```

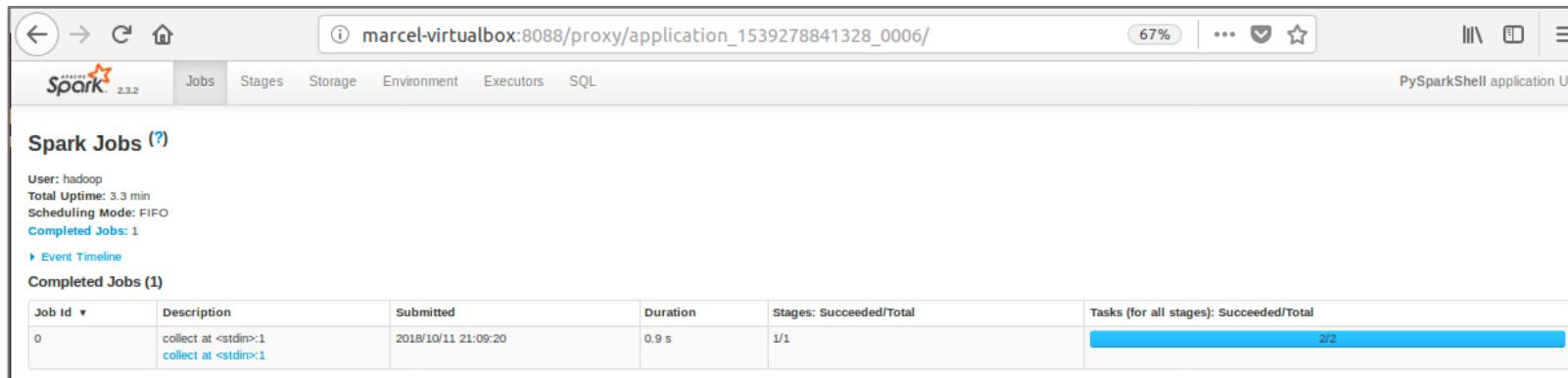
Using Python version 2.7.15rc1 (default, Apr 15 2018 21:51:34)
SparkSession available as 'spark'.

```
>>> dummy_data = range(1, 100)
>>> dummy_rdd = sc.parallelize(dummy_data)
>>> print(dummy_rdd.filter(lambda x: x<10).collect())
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```



Start and Use PySpark (on Yarn) – Test Install

2. Now you should see a completed stage in **PySpark UI** (*localhost:4040*):



The screenshot shows the PySpark UI interface at `marcel-virtualbox:8088/proxy/application_1539278841328_0006/`. The top navigation bar includes links for Spark 2.3.2, Jobs, Stages, Storage, Environment, Executors, and SQL. The main content area is titled "Spark Jobs" with a link to "Event Timeline". Below it, "Completed Jobs (1)" is listed. A table provides details for the single completed job:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at <stdin>:1 collect at <stdin>:1	2018/10/11 21:09:20	0.9 s	1/1	2/2



Start and Use PySpark (on Yarn) – Test Install

3. As well as the related **PySpark** Container Running on Yarn:

The screenshot shows the Hadoop YARN web interface at `localhost:8088/cluster/apps/RUNNING`. The page title is "RUNNING Applications". On the left, there's a sidebar with cluster metrics and a list of applications. One application is highlighted: "application_1539278841328_0006" by "hadoop" with the name "PySparkShell". The application details show it's running on SPARK, in the default queue, with 0 priority, started on Thu Oct 11 21:08:28 +0200 2018, and has 3 containers running, each allocated 5120 CPU Vcores and 0 MB memory.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores	Reserved Memory MB	% of Queue	% of Cluster
application_1539278841328_0006	hadoop	PySparkShell	SPARK	default	0	Thu Oct 11 21:08:28 +0200 2018		RUNNING	UNDEFINED	3	3	5120	0	0	62.5	62.5



Exercises Preparation III

Install and Setup Jupyter (on PySpark)



www.marcel-mittelstaedt.com

Install and Setup Jupyter

1. Install Python

```
sudo apt-get install python
```

2. Install pip for Python

```
sudo apt-get install python-pip
```

3. Install Jupyter

```
pip2 install jupyter
```



Install and Setup Jupyter

4. As Jupyter is installed under `~/.local/bin/jupyter` add path to **\$PATH**:

```
vi .bashrc
```

```
export PATH=$PATH:~/.local/bin
```

```
source .bashrc
```

5. Start Jupyter Notebook

```
jupyter notebook
```



Start and Use Jupyter (on PySpark on Yarn)

6. Open Jupyter Notebook and execute previous example as Python code:

```
jupyter notebook
```

The screenshot shows a Jupyter Notebook interface running on a local host at port 8888. The title bar indicates the notebook is titled "JupyterTest.ipynb". The toolbar includes standard browser navigation icons, a search bar, and a Python logo icon. The main area displays two code cells:

In [1]:

```
import findspark
import os
findspark.init('/home/hadoop/spark')
from pyspark.conf import SparkConf
from pyspark.context import SparkContext
conf = SparkConf().setAppName('Jupyter PySpark Test')
conf.set('spark.yarn.dist.files','file:/home/hadoop/spark/python/lib/pyspark.zip,file:/home/hadoop/spark/python/lib/py4j-0.10.7-src.zip')
conf.setExecutorEnv('PYTHONPATH','pyspark.zip:py4j-0.10.7-src.zip')
conf.setMaster('yarn') # Run on Yarn, not local
sc = SparkContext(conf=conf)
```

In [2]:

```
dummy_data = range(1, 100)
dummy_rdd = sc.parallelize(dummy_data)
print(dummy_rdd.filter(lambda x: x<10).collect())
```

[1, 2, 3, 4, 5, 6, 7, 8, 9]



Use Jupyter (on PySpark on Yarn)

7. Now you should see a completed stage in **PySpark UI** (*localhost:4040*):

The screenshot shows the PySpark UI interface at `marcel-virtualbox:8088/proxy/application_1540030431225_0003/`. The top navigation bar includes links for Jobs, Stages, Storage, Environment, and Executors. The main content area is titled "Spark Jobs (?)". It displays the following information:

- User: hadoop
- Total Uptime: 3.9 min
- Scheduling Mode: FIFO
- Completed Jobs: 1

A link to "Event Timeline" is also present. Below this, a table titled "Completed Jobs (1)" lists one job:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at <ipython-input-2-a93978b47ddd>.3 collect at <ipython-input-2-a93978b47ddd>.3	2018/10/20 13:41:46	0.9 s	1/1	2/2



Use Jupyter (on PySpark on Yarn)

8. As well as the related **PySpark** Container Running on Yarn:

The screenshot shows the Hadoop YARN web interface at `localhost:8088/cluster/apps/RUNNING`. The page title is "RUNNING Applications". On the left, there's a sidebar with a "hadoop" logo and sections for "Cluster Metrics", "Cluster Nodes Metrics", "Scheduler Metrics", and "Tools". The "Cluster Metrics" table shows the following data:

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total
3	1	1	1	3	5 GB	8 GB	0 B	3	8

The "Scheduler Metrics" table shows the following data:

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

The main table lists the "Running Applications":

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores	Reserved Memory MB	% of Queue	% of Cluster	Progress
application_1540030431225_0003	hadoop	Jupyter PySpark Test	SPARK	default	0	Sat Oct 20 13:38:06 +0200 2018	N/A	RUNNING	UNDEFINED	3	3	5120	0	0	62.5	62.5	0

At the bottom, it says "Showing 1 to 1 of 1 entries".



Use Jupyter (on PySpark on Yarn)

9. Basic PySpark operations: Read Files from HDFS into DataFrames:

```
In [3]: from pyspark.sql import SparkSession  
spark = SparkSession(sc)  
  
In [4]: imdb_ratings_dataframe = spark.read.format('com.databricks.spark.csv').options(delimiter = '\t',header ='true',nullValue ='null',inferSchema='true').load('hdfs://user/hadoop/imdb_raw/title_ratings/2018/12/7/title.ratings.tsv')  
  
In [5]: imdb_ratings_dataframe.show(5) # show first 5 lines of tsv file (now a spark dataframe)  
+-----+-----+-----+  
| tconst|averageRating|numVotes|  
+-----+-----+-----+  
|tt0000001|      5.8|     1418|  
|tt0000002|      6.4|     167|  
|tt0000003|      6.6|    1013|  
|tt0000004|      6.4|     100|  
|tt0000005|      6.2|    1712|  
+-----+-----+-----+  
only showing top 5 rows  
  
In [6]: imdb_ratings_dataframe.printSchema() # show schema of tsv file (now dataframe)  
root  
|-- tconst: string (nullable = true)  
|-- averageRating: double (nullable = true)  
|-- numVotes: integer (nullable = true)
```

https://github.com/marcelmittelstaedt/BigData/tree/master/exercises/04_spark_pyspark_jupyter/JupyterTest.html



Use Jupyter (on PySpark on Yarn)

9. Basic PySpark: Operations on DataFrames:

```
In [8]: imdb_ratings_dataframe.count() # show number of rows within dataframe
Out[8]: 872954

In [9]: from pyspark.sql.functions import col, avg
imdb_ratings_dataframe.agg(avg(col("averageRating"))).show() # calculate average movie rating
+-----+
|avg(averageRating)|
+-----+
| 6.929355269579343|
+-----+

In [11]: imdb_ratings_dataframe.filter(col('averageRating') > 9.5).show(5) # filter movie ratings > 9.5 and show first 5
+-----+-----+
| tconst|averageRating|numVotes|
+-----+-----+
|tt0015927|      9.7|      6|
|tt0041069|      9.7|      6|
|tt0050536|      9.7|      7|
|tt0053560|      9.6|      5|
|tt0055416|      9.7|     31|
+-----+-----+
only showing top 5 rows

In [16]: imdb_ratings_dataframe.write.format("csv").save("hdfs://user/hadoop/imdb_ratings")
# saved on HDFS as /user/hadoop/imdb_ratings/part-00000-ce6e8310-64c0-4fb9-b597-d66a92f5309b-c000.csv
```

https://github.com/marcelmittelstaedt/BigData/tree/master/exercises/04_spark_pyspark_jupyter/JupyterTest.html



Use Jupyter (on PySpark on Yarn)

9. Basic PySpark: Join DataFrames:

```
In [16]: imdb_ratings_dataframe.write.format("csv").save("hdfs:///user/hadoop/imdb_ratings")
# saved on HDFS as /user/hadoop/imdb_ratings/part-00000-ce6e8310-64c0-4fb9-b597-d66a92f5309b-c000.csv

In [17]: imdb_movies_dataframe = spark.read.format('com.databricks.spark.csv').options(delimiter = '\t',header ='true',nullValue =
'null',inferSchema='true').load('hdfs:///user/hadoop/imdb_raw/title_basics/2018/12/7/title.basics.tsv')

In [18]: all_imdb_dataframe = imdb_ratings_dataframe.join(imdb_movies_dataframe, imdb_ratings_dataframe.tconst == imdb_movies_data-
frame.tconst)

In [19]: all_imdb_dataframe.select("primaryTitle", "startYear", "averageRating", "numVotes").show(5)
+-----+-----+-----+
| primaryTitle|startYear|averageRating|numVotes|
+-----+-----+-----+
|The Puppet's Nigh...| 1908|      6.5|     126|
|The Lighthouse Ke...| 1911|      7.1|       8|
| The Sands of Dee| 1912|      6.9|      51|
|Zigomar contre Ni...| 1912|      6.8|      10|
|His Favorite Pastime| 1914|      5.1|    814|
+-----+-----+-----+
only showing top 5 rows
```

https://github.com/marcelmittelstaedt/BigData/tree/master/exercises/04_spark_pyspark_jupyter/JupyterTest.html



Use Jupyter (on PySpark on Yarn)

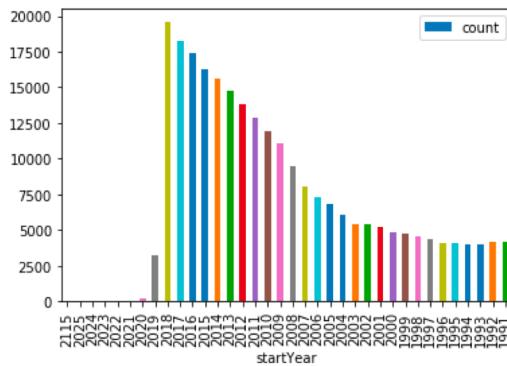
9. Basic Python: Plot results:

```
In [25]: plot_dataframe = imdb_movies_dataframe.filter(col('startYear') != '\N').filter(col('startYear') > 1990).filter(col('titleType') == 'movie').groupBy('startYear').count().sort(col("startYear").desc())
```

```
In [27]: import matplotlib.pyplot as plt
import pandas
pd_df=plot_dataframe.select("startYear", "count").toPandas()
```

```
In [29]: pd_df.plot.bar(x='startYear',y='count')
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff019d341d0>
```





Exercises

Use PySpark Shell or Jupyter Notebooks on
PySpark to solve exercises



PySpark Exercises - IMDB

1. Execute Tasks of previous HandsOn Slides
2. Use PySpark or Jupyter on PySpark to answer following questions:
 - a) How many **movies** are within the IMDB dataset?
 - b) Who is the **oldest** actor/writer/... within the dataset?
 - c) Create a list (`tconst`, `original_title`, `start_year`, `average_rating`, `num_votes`) of movies which are:
 - equal or newer than year 2000
 - have an average rating better than 8
 - have been voted more than 100.000 timessave result (DataFrame) back to HDFS as CSV File.



PySpark Exercises - IMDB

2. Use PySpark or Jupyter on PySpark to answer following questions:

d) How many movies are in list of c)?

e) create following plot with result of c)
(plot visualizes the amount of good
movies per year since 2001)

