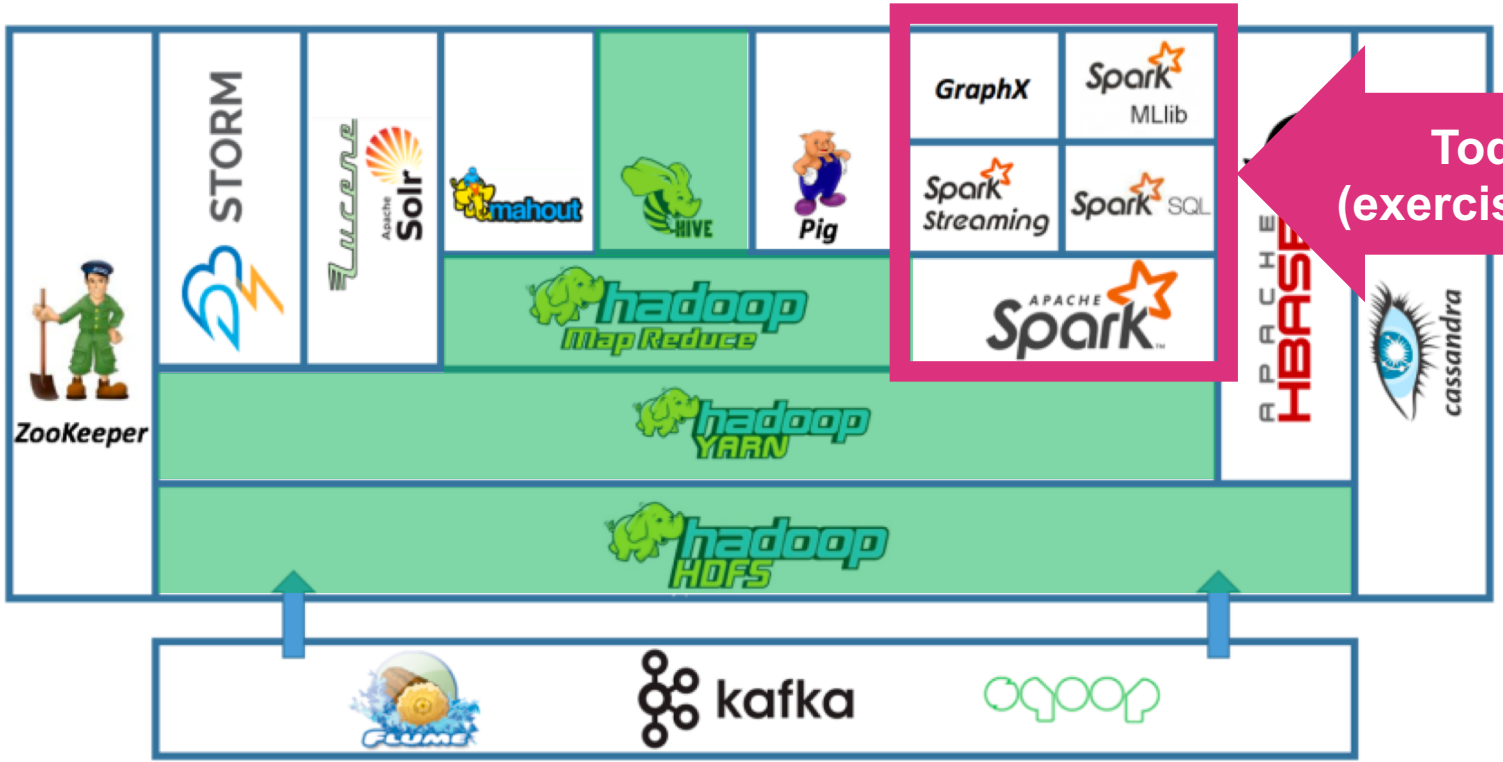
HandsOn – **Spark, Scala, PySpark and Jupyter**

A quick Introduction to Spark, Scala, PySpark and Jupyter

The Hadoop Ecosystem



Spark

API Languages: Java Scala Python R

APIs and Libraries: Spark SQL | Spark Streaming | Mlib | GraphX

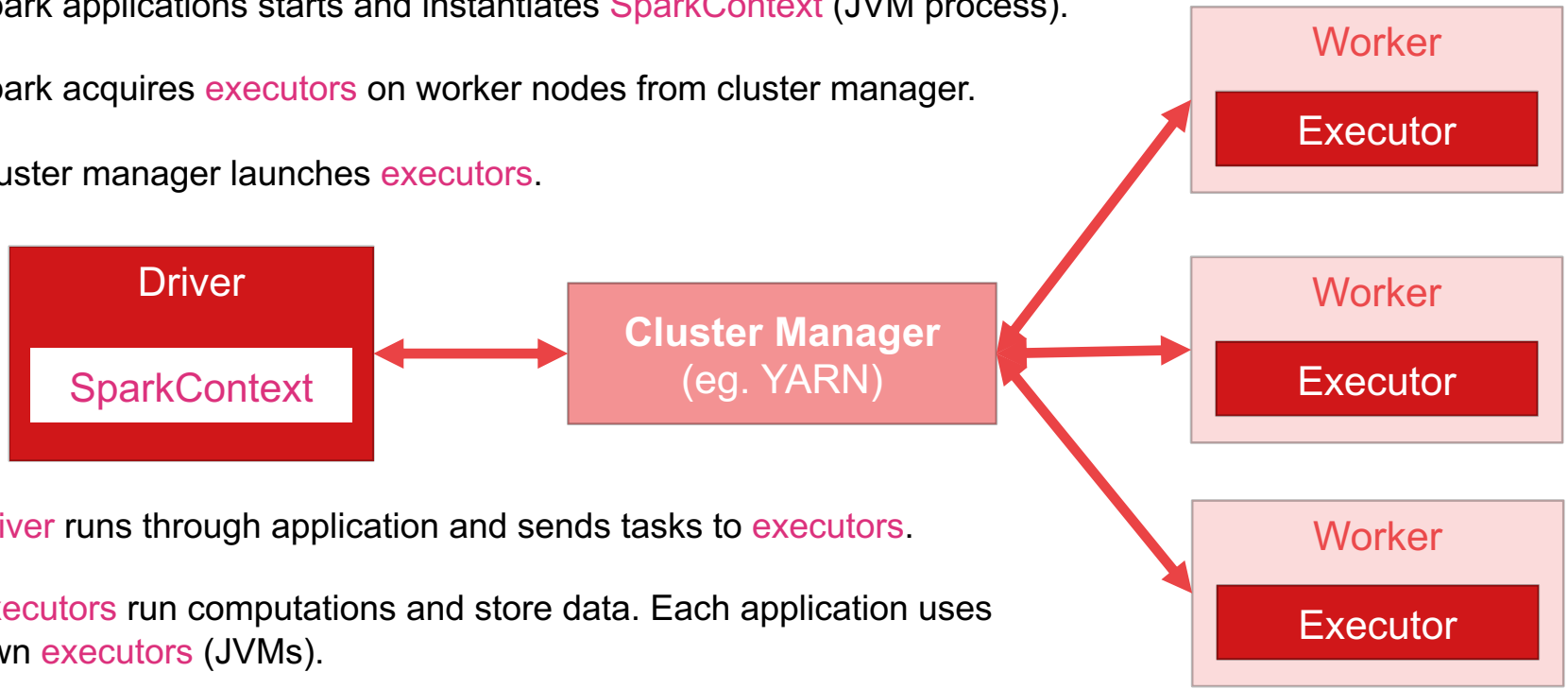
Spark Core

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.



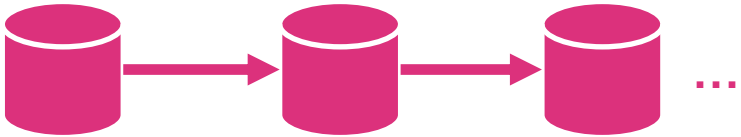
4. Driver runs through application and sends tasks to executors.
5. Executors run computations and store data. Each application uses its own executors (JVMs).
6. If any worker crashes, ist taks will be send to another executor.

Spark – RDDs, DataFrame and DataSet

	RDD (2011)	DataFrame (2013)	DataSet (2015)
Supported by:	Scala, Java, Python	Scala, Java, Python	Scala, Java
Idea:	- RDD = immutable distributed collection of data - partitioned across nodes of cluster - can be operated in parallel with a low-level API	- based on RDD - immutable distributed collection of data - but organized into columns - higher level abstraction	- based on DataFrame API, but type-safe - immutable distributed collection of data - high-level API - converted into optimized RDD
Typed:	typed, no schema	untyped, schema	typed, schema
Use for:	unstructured data	semi-structured and structured data	structured data

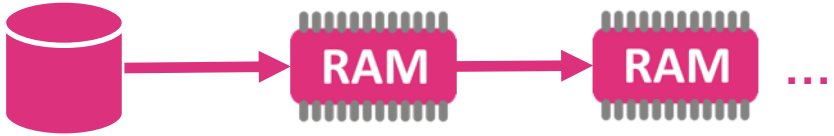
Hadoop MapReduce vs Spark

Hadoop MapReduce



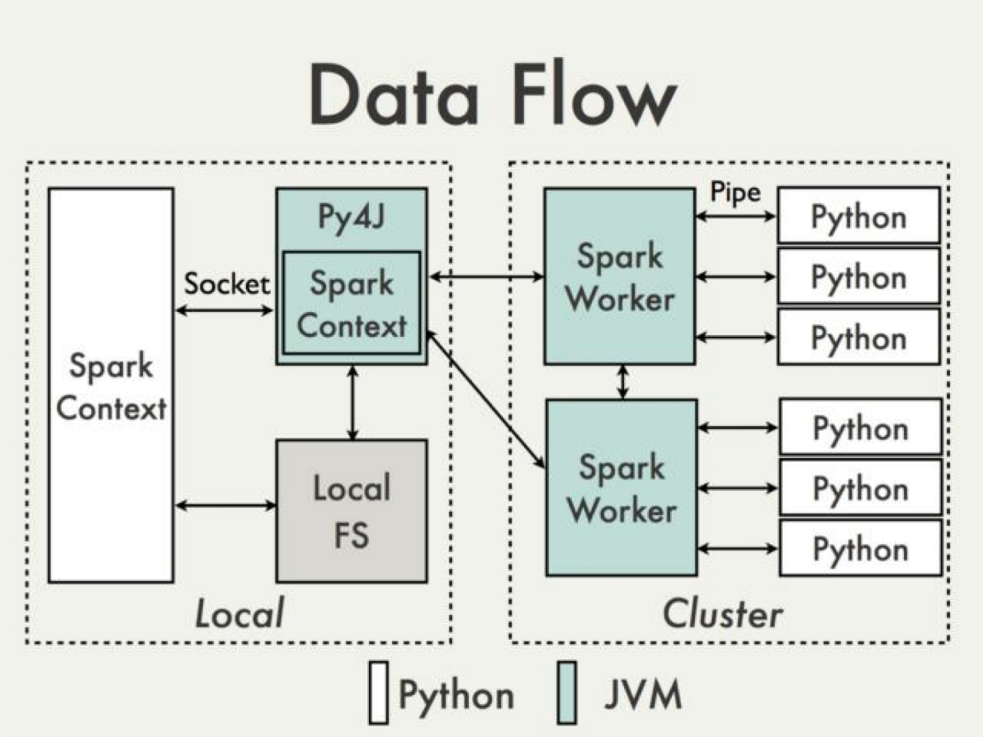
- performs read from HDFS (HDD) before every computation
- performs write on HDFS (HDD) after every computation
- using distributed disk space

Spark



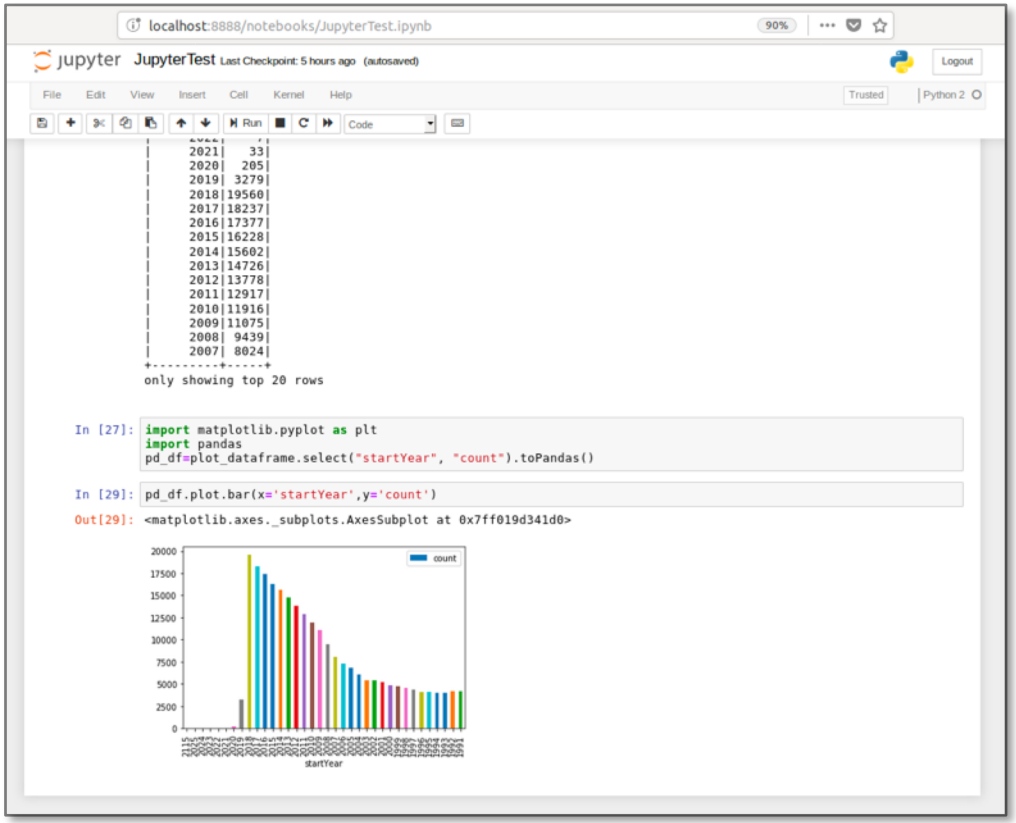
- performs read from RAM before e very computation (except first)
- performs write on RAM after every computation
- using distributed **memory**

PySpark



- built on-top of Sparks 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 corrresponds to an RDD in
- the local JVM
- e.g. **sc.textFile()** in Python will call JavaSparkContext **textFile()**

Jupyter (Notebooks)



- 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

Start Hadoop Cluster and Test Spark Shell
(Word Count Example)

Start Gcloud VM and Connect

1. Start Gcloud Instance:

```
gcloud compute instances start big-data
```

2. Connect to Gcloud instance via SSH (on Windows using Putty):

```
ssh hans.wurst@XXX.XXX.XXX.XXX
```


Pull and Start Docker Container

1. Pull Docker Image:

```
docker pull marcelmittelstaedt/spark_base:latest
```

2. Start Docker Image:

```
docker network create --driver bridge bigdatanet
docker run -dit --name hadoop \
          -p 8088:8088 -p 9870:9870 -p 9864:9864 -p 10000:10000 \
          -p 8032:8032 -p 8030:8030 -p 8031:8031 -p 9000:9000 \
          -p 8888:8888 --net bigdatanet \
          marcelmittelstaedt/spark_base:latest
```

3. Wait till first Container Initialization finished:

```
docker logs hadoop

[…]
Stopping nodemanagers
Stopping resourcemanager
Container Startup finished.
```


Start Hadoop Cluster

1. Get into Docker container:

```
docker exec -it hadoop bash
```

2. Switch to hadoop user:

```
sudo su hadoop
```

```
cd
```

3. Start Hadoop Cluster:

```
start-all.sh
```


Add Test text file to HDFS (Faust 1)

1. Download test Text File (Faust_1.txt):

```
wget https://raw.githubusercontent.com/marcelmittelstaedt/BigData/master/exercises/winter_semester_2019-2020/01_hadoop/sample_data/Faust_1.txt
```

2. Upload file to HDFS:

```
hadoop fs -put Faust_1.txt /user/hadoop/Faust_1.txt
```


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_1572177196643_0001).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.4
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_222)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```


Start Spark – WordCount Example (Scala)

1. Execute Word Count Example in Scala:

```
scala> val text_file = sc.textFile("/user/hadoop/Faust_1.txt")
scala> val words = text_file.flatMap(line => line.split(" "))
scala> val counts = words.map(word => (word, 1))
scala> val reduced_counts = counts.reduceByKey((count1, count2) => count1 + count2)
scala> val sorted_counts = reduced_counts.sortBy(- _._2)
```

```
scala> sorted_counts.take(10)

res0: Array[(String, Int)] = Array(("",1603), (und,509), (die,463), (der,440), (ich,435), (Und,
400), (nicht,346), (zu,319), (ist,291), (ein,284))
```

2. Save results to HDFS:

```
scala> sorted_counts.saveAsTextFile("/user/hadoop/Faust_1_WordCounts_Scala.txt")
```


Start Spark – WordCount Example (Scala)

3. Get results from HDFS to local filesystem:

```
hadoop fs -get /user/hadoop/Faust_1_WordCounts_Scala.txt/part-00000 Faust_1_WordCounts_Scala.txt
```

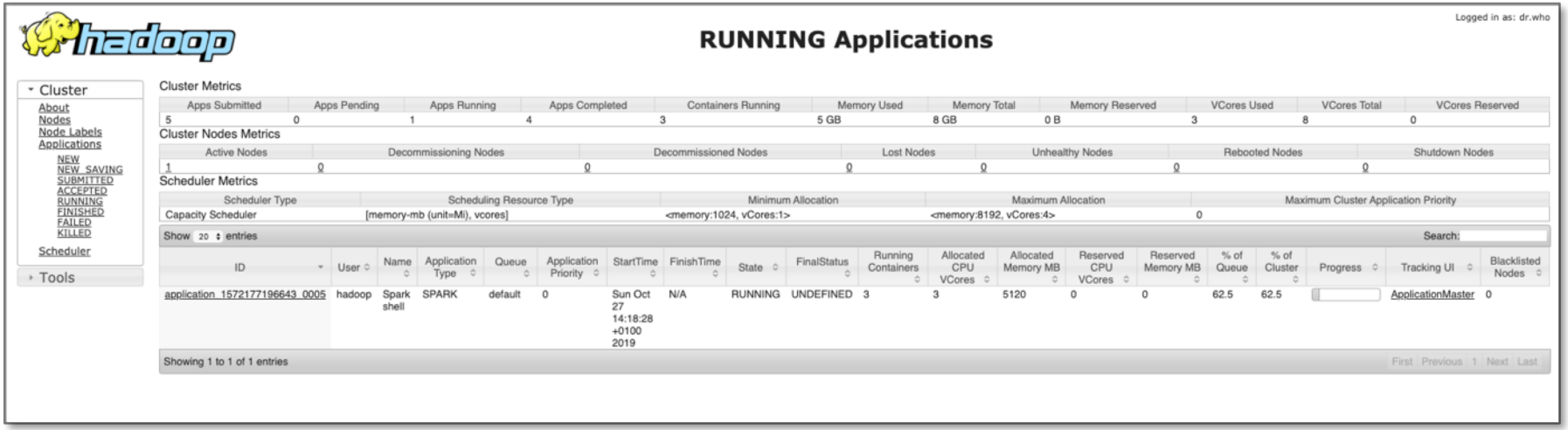
4. Check Result:

```
head -10 Faust_1_WordCounts_Scala.txt

(,1603)
(und,509)
(die,463)
(der,440)
(ich,435)
(Und,400)
(nicht,346)
(zu,319)
(ist,291)
(ein,284)
```

Start Spark (on Yarn) – WordCount Example

5. See Spark Shell Container Running on Yarn http://xxx.xxx.xxx.xxx:8088/cluster :



Exercises Preparation II

Test PySpark Shell (Word Count Example)

Start 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 3.6.8 (default, Oct  7 2019, 12:59:55)
[GCC 8.3.0] on linux
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).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.4
      /_/

Using Python version 3.6.8 (default, Oct  7 2019 12:59:55)
SparkSession available as 'spark'.
>>>
```

PySpark – WordCount Example (Python)

1. Execute Word Count Example in Python:

```
>>> text_file = spark.read.text("/user/hadoop/Faust_1.txt").rdd.map(lambda r: r[0])
>>> words = text_file.flatMap(lambda line: line.split(" "))
>>> counts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)
>>> output = counts.collect()
>>> sorted_output = sorted(output,key=lambda x:(-x[1],x[0]))
```

```
>>> print(sorted_output[:10])

[('', 1603), ('und', 509), ('die', 463), ('der', 440), ('ich', 435), ('Und', 400), ('nicht', 34
6), ('zu', 319), ('ist', 291), ('ein', 284)]
```

2. Save results to HDFS:

```
>>> counts.saveAsTextFile("/user/hadoop/Faust_1_WordCounts_Python.txt")
```


PySpark – WordCount Example (Python)

3. Get results from HDFS to local filesystem:

```
hadoop fs -get /user/hadoop/Faust_1_WordCounts_Python.txt/part-00000 Faust_1_WordCounts_Python.txt
```

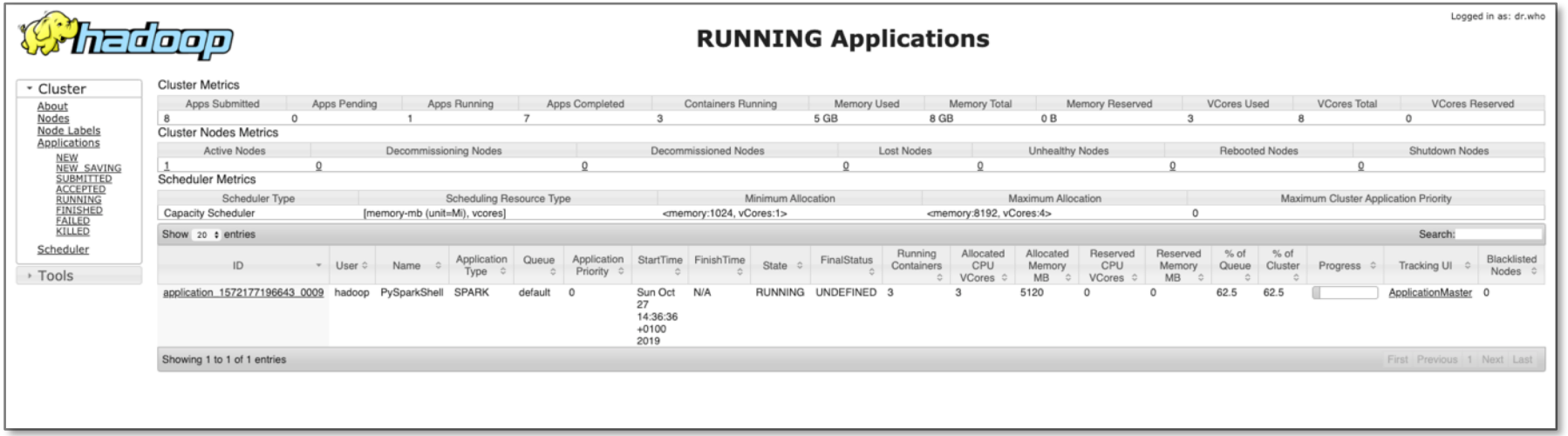
4. Check Result:

```
head -10 Faust_1_WordCounts_Python.txt

('Johann', 1)
('Wolfgang', 1)
('von', 133)
('Goethe:', 1)
('Faust,', 8)
('Der', 130)
('Tragödie', 1)
('erster', 2)
('Teil', 6)
('', 1603)
```

PySpark (on Yarn) – WordCount Example

5. See PySpark Shell Container Running on Yarn http://xxx.xxx.xxx.xxx:8088/cluster :



Exercises Preparation III

Start and work with Jupyter Notebooks
(on PySpark)

Start Jupyter

1. Start Jupyter Notebook

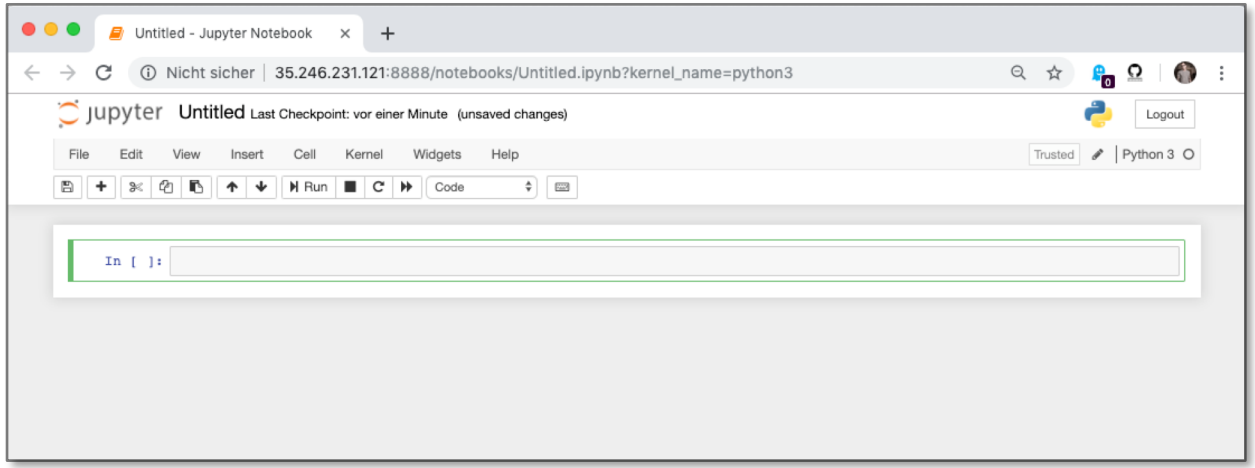
```
jupyter notebook

[I 14:02:39.790 NotebookApp] Writing notebook server cookie secret to /home/hadoop/.local/share/jupyter/runtime/notebook_cookie_secret
[I 14:02:40.957 NotebookApp] Serving notebooks from local directory: /home/hadoop
[I 14:02:40.957 NotebookApp] The Jupyter Notebook is running at:
[I 14:02:40.957 NotebookApp] http://e0f4472dcb12:8888/?token=76b68a3c700b415790b019e07a3dd46a0d068c153a732d27
[I 14:02:40.957 NotebookApp]  or http://127.0.0.1:8888/?token=76b68a3c700b415790b019e07a3dd46a0d068c153a732d27
[I 14:02:40.957 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 14:02:40.979 NotebookApp] No web browser found: could not locate runnable browser.
[C 14:02:40.980 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/hadoop/.local/share/jupyter/runtime/nbserver-8624-open.html
    Or copy and paste one of these URLs:
        http://e0f4472dcb12:8888/?token=76b68a3c700b415790b019e07a3dd46a0d068c153a732d27
     or http://127.0.0.1:8888/?token=76b68a3c700b415790b019e07a3dd46a0d068c153a732d27
```

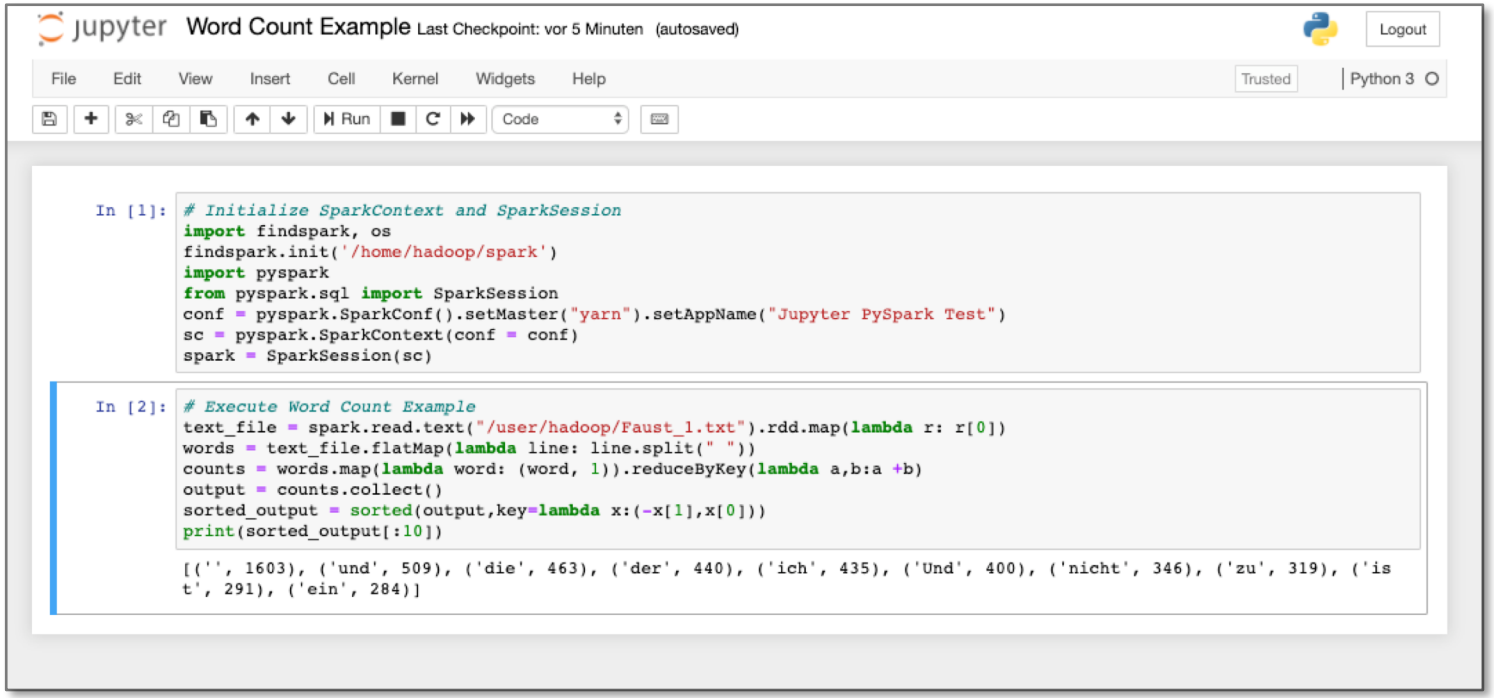

Start Jupyter

2. Open Notebook in Browser: http://XXX.XXX.XXX.XXX:8888/?token=[XYZXYZXYZ]



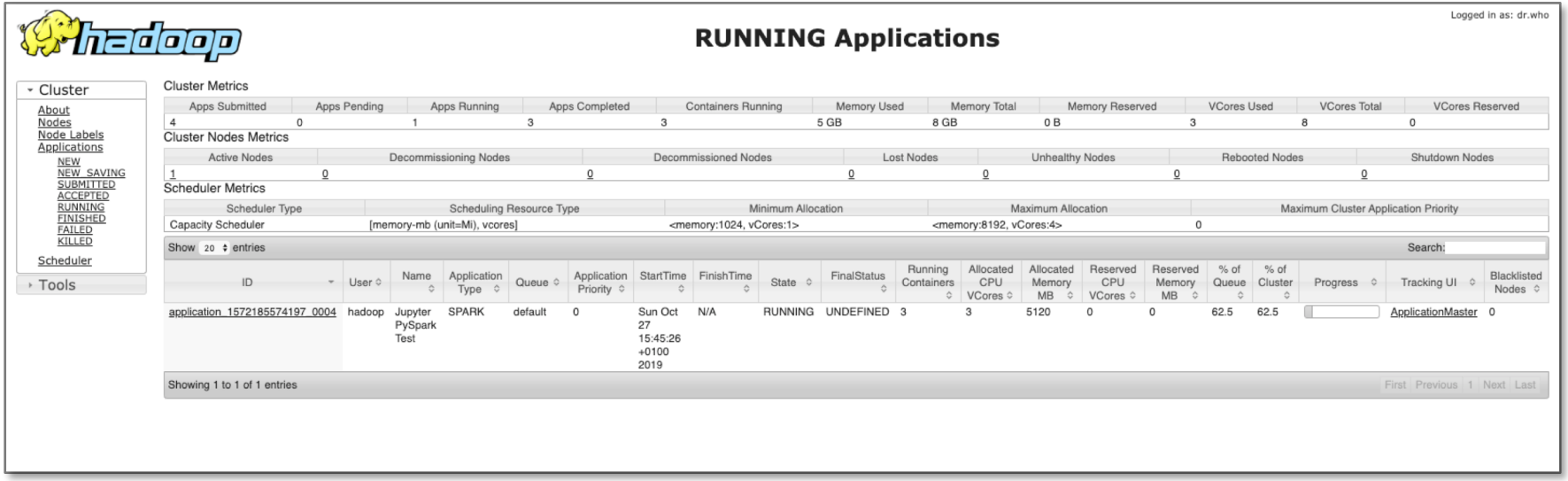
Use Jupyter (Word Count Example)

1. Execute previous Word Count example



Use Jupyter (Word Count Example)

2. See **Jupyter** PySpark Container Running on Yarn http://xxx.xxx.xxx.xxx:8088/cluster :



Get some data…

1. Get some IMDb data:

```
wget https://datasets.imdbws.com/title.basics.tsv.gz && gunzip title.basics.tsv.gz
wget https://datasets.imdbws.com/title.ratings.tsv.gz && gunzip title.ratings.tsv.gz
```

2. Put them into HDFS:

```
hadoop fs -mkdir /user/hadoop/imdb
```

```
hadoop fs -mkdir /user/hadoop/imdb/title_basics
hadoop fs -mkdir /user/hadoop/imdb/title_ratings
```

```
hadoop fs -put title.basics.tsv /user/hadoop/imdb/title_basics/title.basics.tsv
hadoop fs -put title.ratings.tsv /user/hadoop/imdb/title_ratings/title.ratings.tsv
```

PySpark Operations

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

```
In [5]: # Read title.basics.tsv into Spark dataframe
imdb_title_basics_dataframe = spark.read.format('csv').options(\
    header='true', delimiter='\t', nullValue='null', inferschema='true')\
    .load('/user/hadoop/imdb/title_basics/title.basics.tsv')

In [6]: imdb_title_basics_dataframe.printSchema() # Print Schema of title_basics dataframe

root
 |-- tconst: string (nullable = true)
 |-- titleType: string (nullable = true)
 |-- primaryTitle: string (nullable = true)
 |-- originalTitle: string (nullable = true)
 |-- isAdult: integer (nullable = true)
 |-- startYear: string (nullable = true)
 |-- endYear: string (nullable = true)
 |-- runtimeMinutes: string (nullable = true)
 |-- genres: string (nullable = true)

In [7]: imdb_title_basics_dataframe.show(5) # Show first 5 rows of title_basics dataframe

+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
|   tconst|titleType|        primaryTitle|       originalTitle|isAdult|startYear|endYear|runtimeMinutes|              genres|
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
|tt0000001|    short|          Carmencita|          Carmencita|      0|     1894|     \N|             1|   Documentary,Short|
|tt0000002|    short|Le clown et ses c...|Le clown et ses c...|      0|     1892|     \N|             5|     Animation,Short|
|tt0000003|    short|      Pauvre Pierrot|      Pauvre Pierrot|      0|     1892|     \N|             4|Animation,Comedy,...|
|tt0000004|    short|         Un bon bock|         Un bon bock|      0|     1892|     \N|            \N|     Animation,Short|
|tt0000005|    short|    Blacksmith Scene|    Blacksmith Scene|      0|     1893|     \N|             1|        Comedy,Short|
+---------+---------+--------------------+--------------------+-------+---------+-------+--------------+--------------------+
only showing top 5 rows
```

https://github.com/marcelmittelstaedt/BigData/tree/master/exercises/ winter_semester_2019-2020/04_spark_pyspark_jupyter/PySparkExamples.html

PySpark Operations

2. Basic PySpark: Operations on DataFrames (filter, aggregate, sort…):

```
In [5]: imdb_title_basics_dataframe.count() # show number of rows within title_basics dataframe
Out[5]: 6262637
```

```
In [11]: # Get column titleTypes values with counts and ordered descending
from pyspark.sql.functions import desc
imdb_title_basics_dataframe.groupBy("titleType").count().orderBy(desc("count")).show()
```

```
+------------+-------+
|   titleType|  count|
+------------+-------+
|   tvEpisode|4392190|
|       short| 707983|
|       movie| 533869|
|       video| 245253|
|    tvSeries| 173337|
|     tvMovie| 120824|
|tvMiniSeries|  28114|
|   tvSpecial|  25479|
|   videoGame|  24322|
|     tvShort|  11266|
+------------+-------+
```

```
In [28]: # Calculate average Movie length in minutes
from pyspark.sql.functions import avg
imdb_title_basics_dataframe.filter(imdb_title_basics_dataframe['titleType']=='movie')\
    .agg(avg('runtimeMinutes')).show()
```

```
+------------------+
|avg(runtimeMinutes)|
+------------------+
|  88.34293031750659|
+------------------+
```

https://github.com/marcelmittelstaedt/BigData/tree/master/exercises/ winter_semester_2019-2020/04_spark_pyspark_jupyter/PySparkExamples.html

www.marcel-mittelstaedt.com

PySpark Operations

3. PySpark SQL: Join DataFrames:

In [12]:
```
# Read title.ratings.tsv into Spark dataframe
imdb_title_ratings_dataframe = spark.read.format('csv').options(\
    header='true', delimiter='\t', nullValue='null', inferschema='true')\
    .load('/user/hadoop/imdb/title_ratings/title.ratings.tsv')
```

In [13]:
```
imdb_title_ratings_dataframe.printSchema() # Print Schema of title_ratings dataframe
```

```
root
 |-- tconst: string (nullable = true)
 |-- averageRating: double (nullable = true)
 |-- numVotes: integer (nullable = true)
```

In [14]:
```
imdb_title_ratings_dataframe.show(5) # Show first 5 rows of title_ratings dataframe
```

```
+---------+-------------+--------+
|   tconst|averageRating|numVotes|
+---------+-------------+--------+
|tt0000001|          5.6|    1543|
|tt0000002|          6.1|     186|
|tt0000003|          6.5|    1201|
|tt0000004|          6.2|     114|
|tt0000005|          6.1|    1921|
+---------+-------------+--------+
only showing top 5 rows
```

In [15]:
```
# JOIN Data Frames
title_basics_and_ratings_df = imdb_title_basics_dataframe.join(imdb_title_ratings_dataframe, \
                              imdb_title_basics_dataframe.tconst == imdb_title_ratings_dataframe.tconst)
```

In [25]:
```
top_tvseries=title_basics_and_ratings_df.filter(title_basics_and_ratings_df['titleType']=='tvSeries')\
                              .filter(title_basics_and_ratings_df['numVotes'] > 200000)\
                              .orderBy(desc('averageRating'))\
                              .select('originalTitle', 'startYear', 'endYear', 'averageRating', 'numVotes')
top_tvseries.show(5)
```

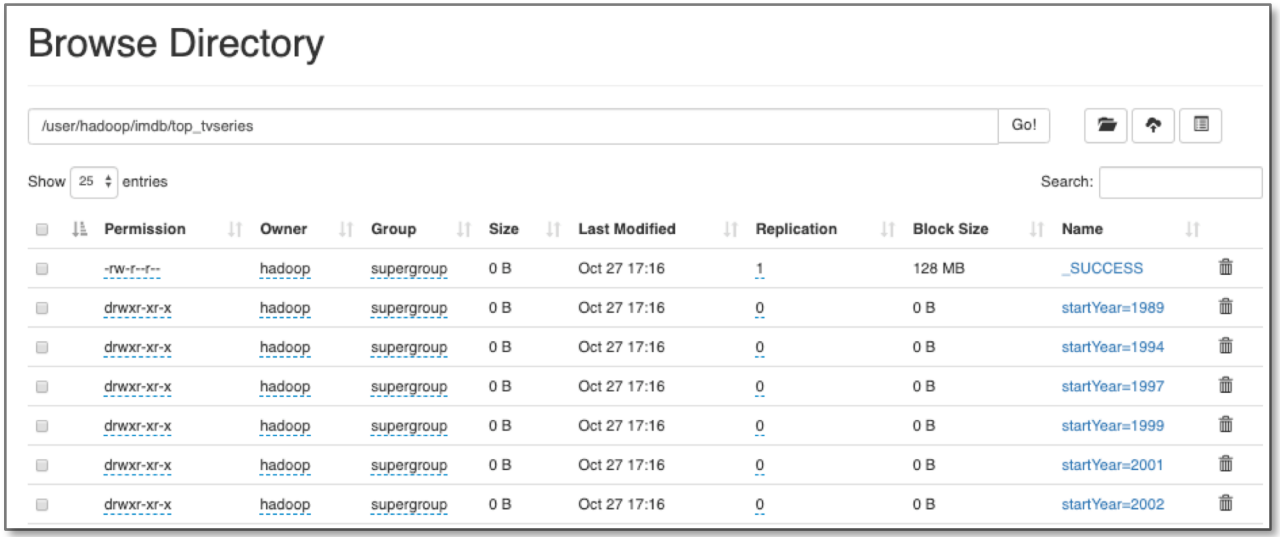
```
+--------------------+---------+-------+-------------+--------+
|       originalTitle|startYear|endYear|averageRating|numVotes|
+--------------------+---------+-------+-------------+--------+
|        Breaking Bad|     2008|   2013|          9.5| 1271805|
|     Game of Thrones|     2011|   2019|          9.4| 1598911|
|      Rick and Morty|     2013|     \N|          9.3|  296206|
|            The Wire|     2002|   2008|          9.3|  254371|
|Avatar: The Last ...|     2005|   2008|          9.2|  200077|
+--------------------+---------+-------+-------------+--------+
only showing top 5 rows
```

https://github.com/marcelmittelstaedt/BigData/tree/master/exercises/ winter_semester_2019-2020/04_spark_pyspark_jupyter/PySparkExamples.html

PySpark Operations

4. Basic PySpark: Save DataFrames as File to HDFS:

```
In [27]: top_tvseries.write.format('parquet')\ # Could also be CSV, but parquet requires less space
                           .partitionBy('startYear')\ # partition data by startYear (not mandatory but useful)
                           .mode('overwrite')\ # overwrite existing data
                           .save('/user/hadoop/imdb/top_tvseries') # where to save
```



Browse Directory

/user/hadoop/imdb/top_tvseries Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	Oct 27 17:16	1	128 MB	_SUCCESS
drwxr-xr-x	hadoop	supergroup	0 B	Oct 27 17:16	0	0 B	startYear=1989
drwxr-xr-x	hadoop	supergroup	0 B	Oct 27 17:16	0	0 B	startYear=1994
drwxr-xr-x	hadoop	supergroup	0 B	Oct 27 17:16	0	0 B	startYear=1997
drwxr-xr-x	hadoop	supergroup	0 B	Oct 27 17:16	0	0 B	startYear=1999
drwxr-xr-x	hadoop	supergroup	0 B	Oct 27 17:16	0	0 B	startYear=2001
drwxr-xr-x	hadoop	supergroup	0 B	Oct 27 17:16	0	0 B	startYear=2002

PySpark Operations

5. PySpark SQL: Save DataFrames as (temporary Hive) Table:

```
In [17]: title_basics_and_ratings_df.select('originalTitle', 'titleType', 'startYear', \
                                            'endYear', 'numVotes', 'averageRating')\
                                    .write.saveAsTable('movies_and_ratings')
```

6. PySpark SQL : Read from (temporary Hive) Table:

```
In [18]: result_df = spark.sql("""SELECT originalTitle, averageRating FROM movies_and_ratings WHERE
                                  numVotes > 200000 AND titleType= 'movie' AND averageRating > 8.5 AND startYear > 2010
                                  ORDER BY averageRating DESC LIMIT 10""")
                                ).show(10)
+-------------+-------------+
|originalTitle|averageRating|
+-------------+-------------+
|        Joker|          8.9|
| Interstellar|          8.6|
+-------------+-------------+
```

PySpark Operations

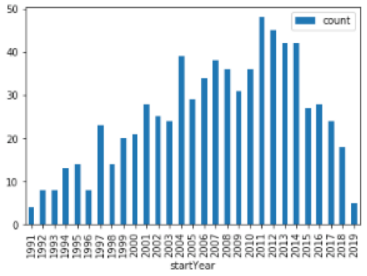
7. Basic Python: Plot results:

```
In [6]: good_movies_df = title_basics_and_ratings_df.filter(title_basics_and_ratings_df['titleType']=='movie')\
                          .filter(title_basics_and_ratings_df['numVotes'] > 200000)\
                          .filter(title_basics_and_ratings_df['startYear'] > 1990)\
                          .groupBy('startYear')\
                          .count()\
                          .orderBy('startYear')
        good_movies_df.show(5)
```

```
+---------+-----+
|startYear|count|
+---------+-----+
|     1991|    4|
|     1992|    8|
|     1993|    8|
|     1994|   13|
|     1995|   14|
+---------+-----+
only showing top 5 rows
```

```
In [11]: import matplotlib.pyplot as plt
         import pandas
         pandas_dataframe = good_movies_df.select('startYear', 'count').toPandas()
         pandas_dataframe.plot.bar(x='startYear', y='count')
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd4e21a2240>
```



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 **tv series** are within the IMDB dataset?

    b) Who is the **youngest** 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 2010
    - have an average rating better than 8
    - have been voted more than 100.000 times

    save 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)