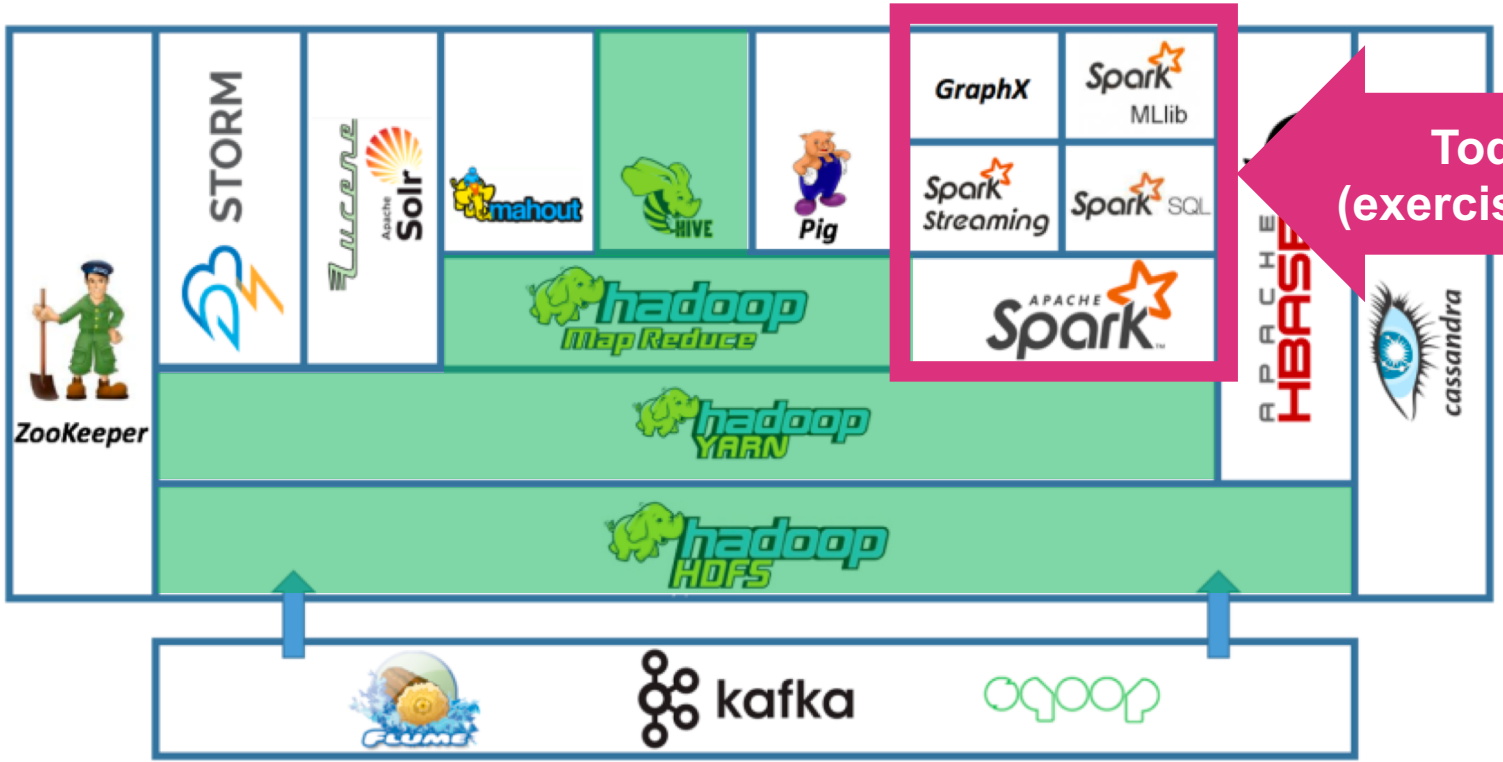
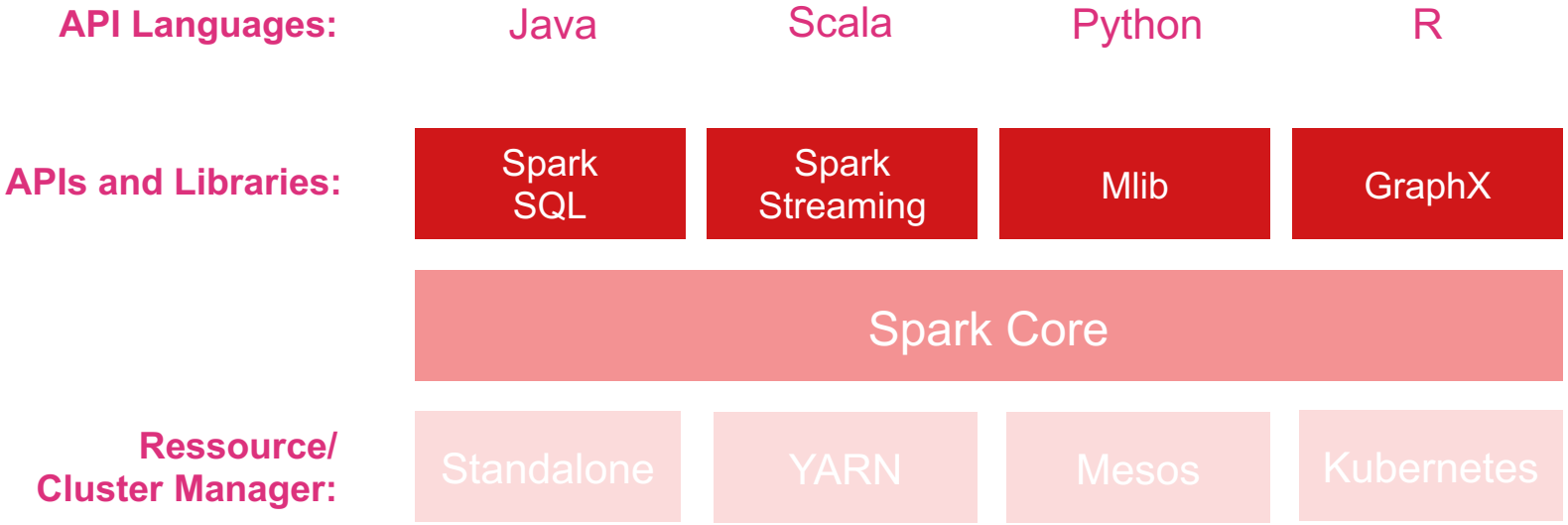
HandsOn – **Spark, Scala, PySpark and Jupyter**

A quick Introduction to Spark, Scala, PySpark and Jupyter

The Hadoop Ecosystem

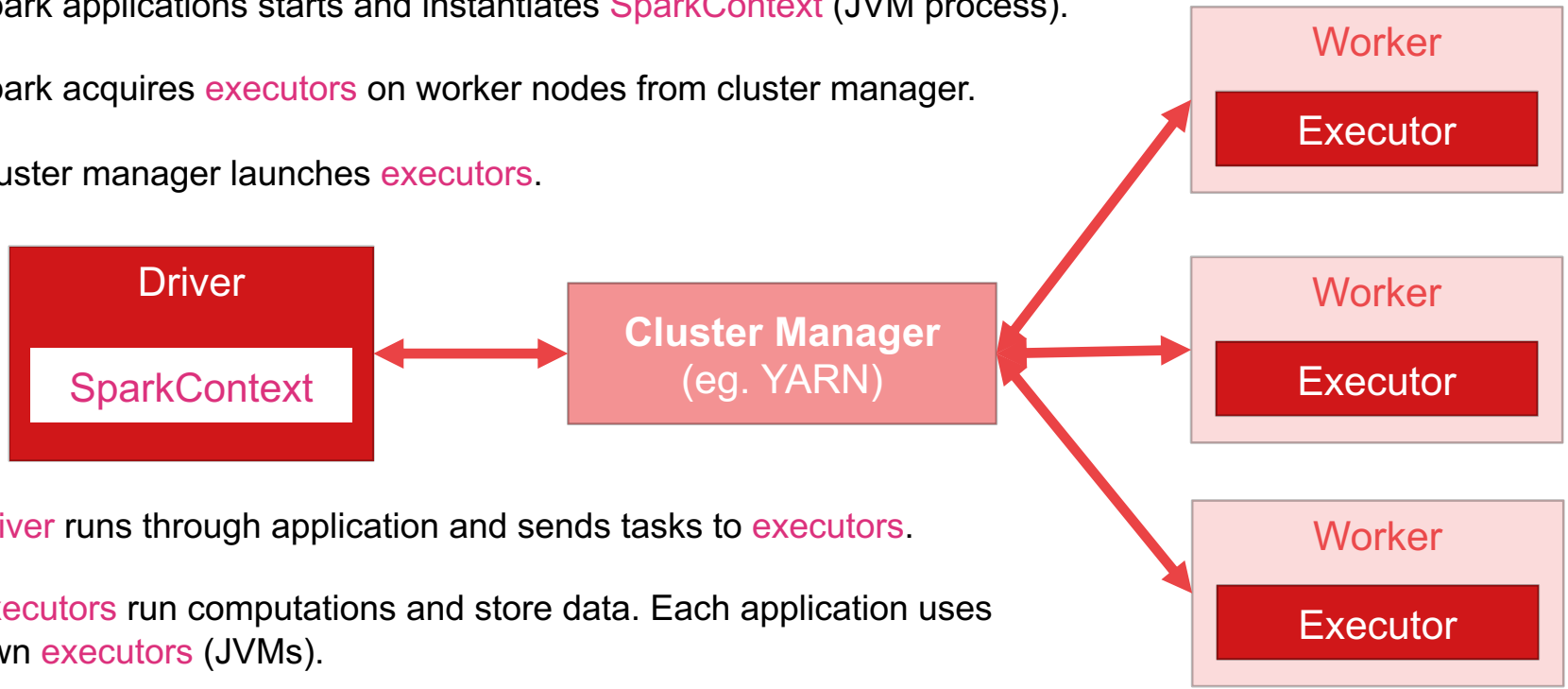


Spark


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, Couch-base, 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.



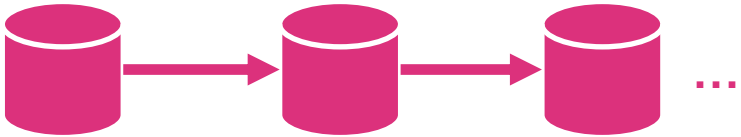
4. Driver runs through application and sends tasks to executors.
5. Executors run computations and store data. Each application uses its own executors (JVMs).
6. If any worker crashes, ist taks will be send to another executor.

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

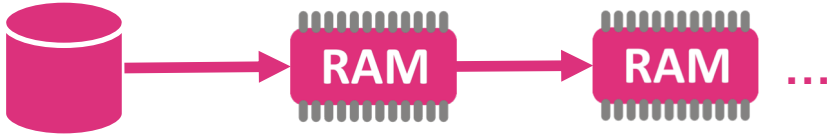
Hadoop MapReduce vs Spark

Hadoop MapReduce



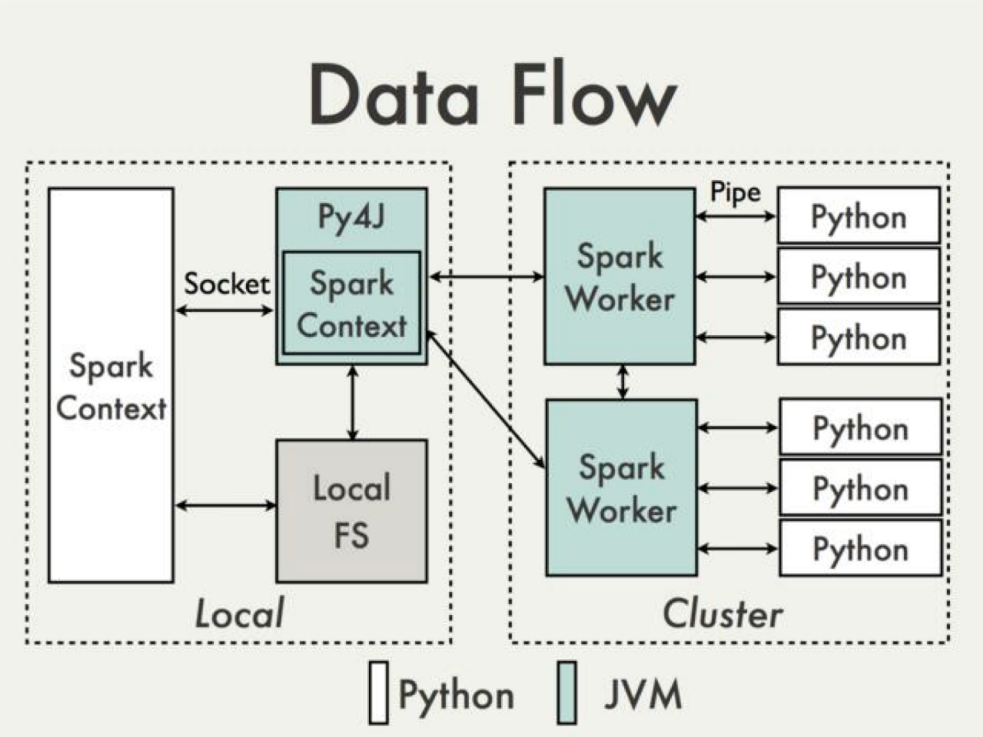
- performs read from HDFS (HDD) before every computation
- performs write on HDFS (HDD) after every computation
- using distributed disk space

Spark



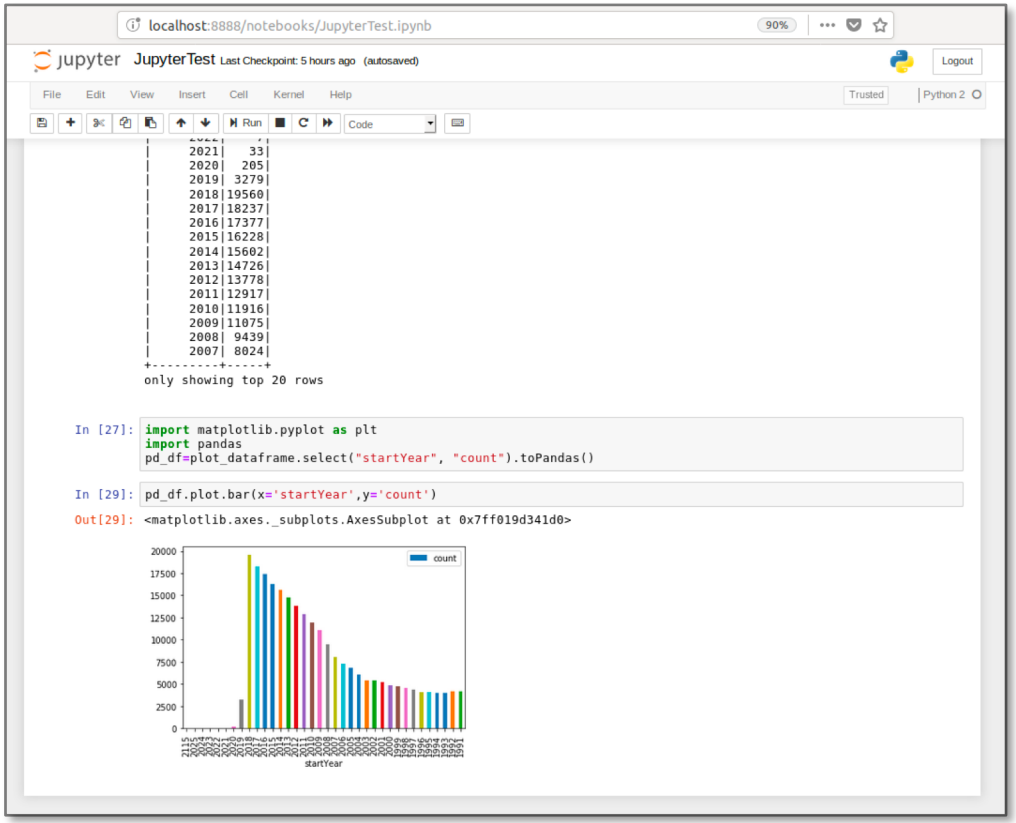
- performs read from RAM before e very computation (except first)
- performs write on RAM after every computation
- using distributed **memory**

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 t he local JVM
- e.g. **sc.textFile()** in Python will call JavaSp arkContext **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

Install and Setup Spark

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
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.2
      /_/

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:

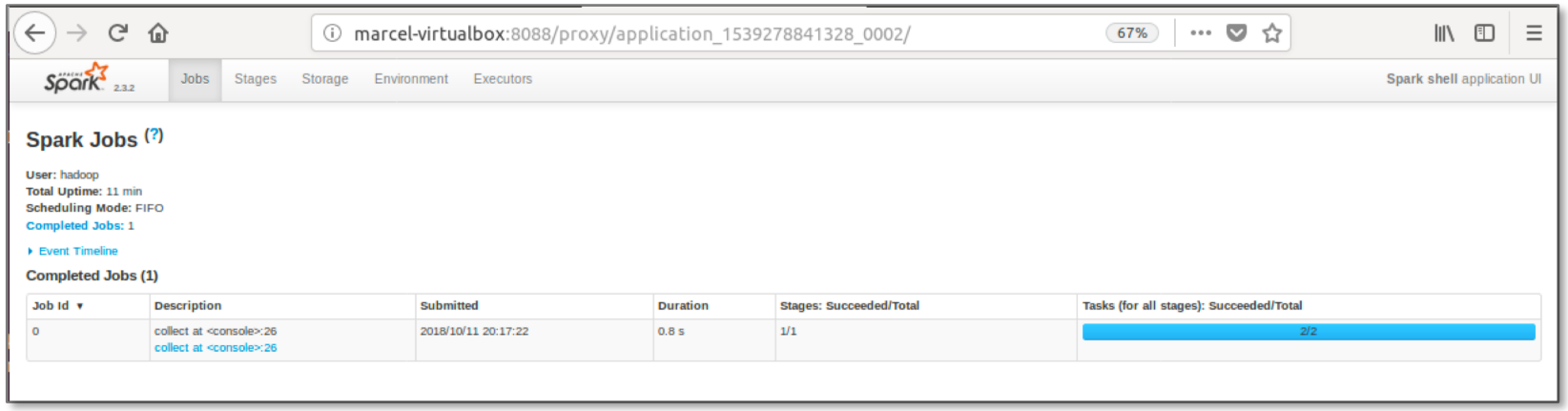
```
scala> val dummy_RDD = sc.parallelize(dummy_data)
dummy_RDD: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallel
ize at <console>:26
```


Start Spark (on Yarn) – Test Install

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

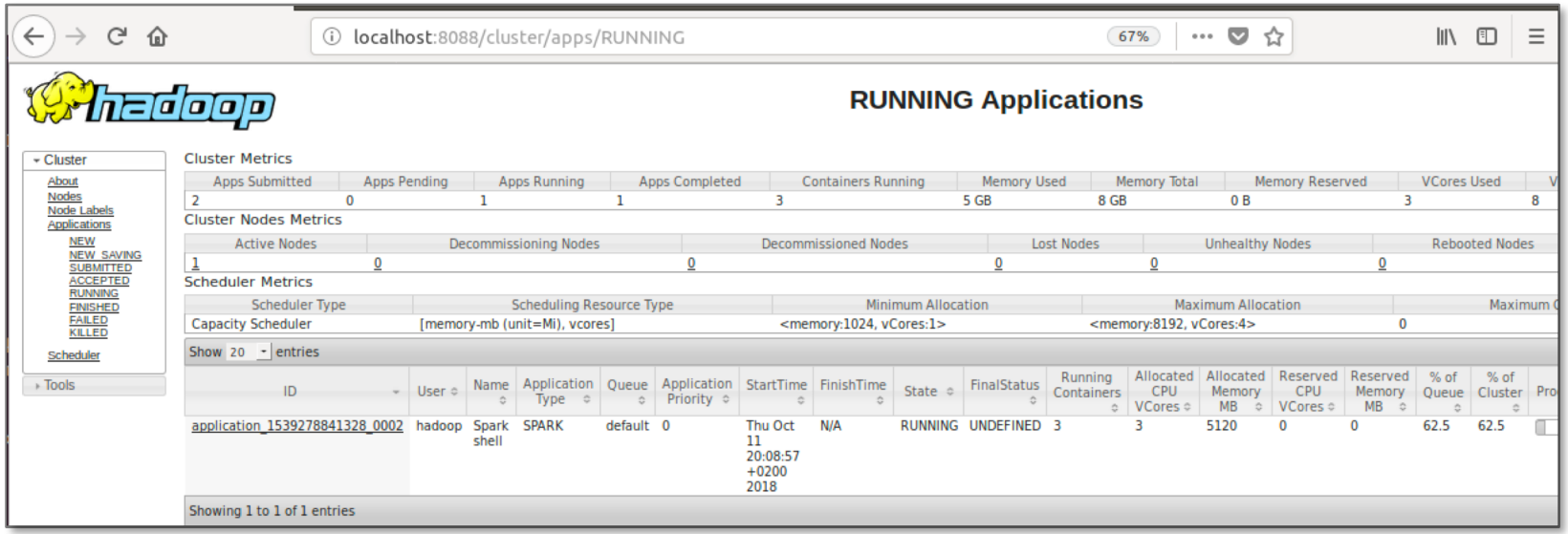
```
scala> dummy_RDD.filter(_ < 10).collect()
res0: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

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



Start Spark (on Yarn) – Test Install

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



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
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.2
      /_/

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:

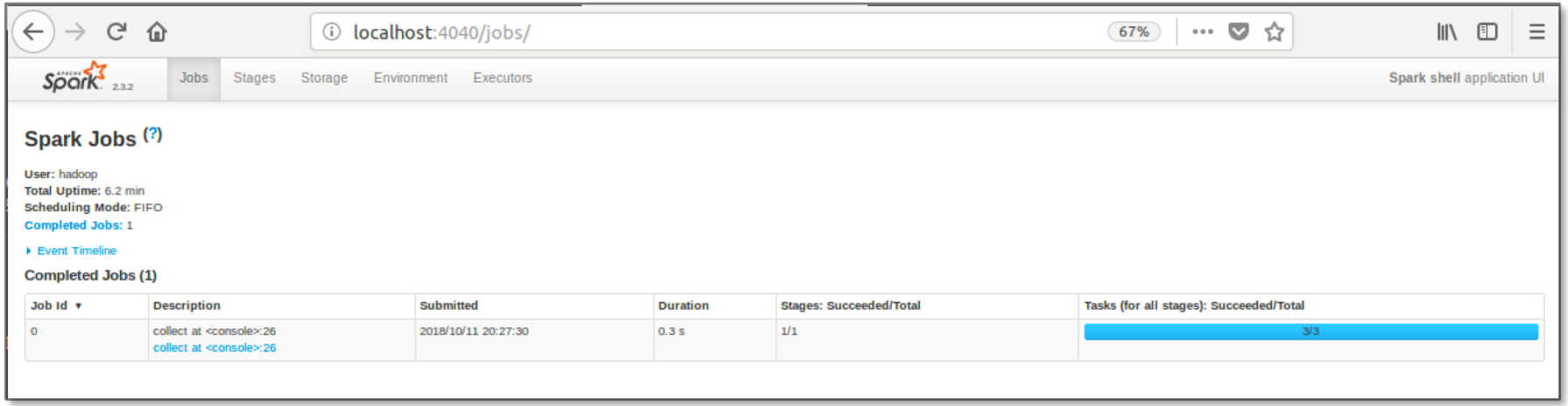
```
scala> val dummy_data = 1 to 100
dummy_data: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1
1, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 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)
```

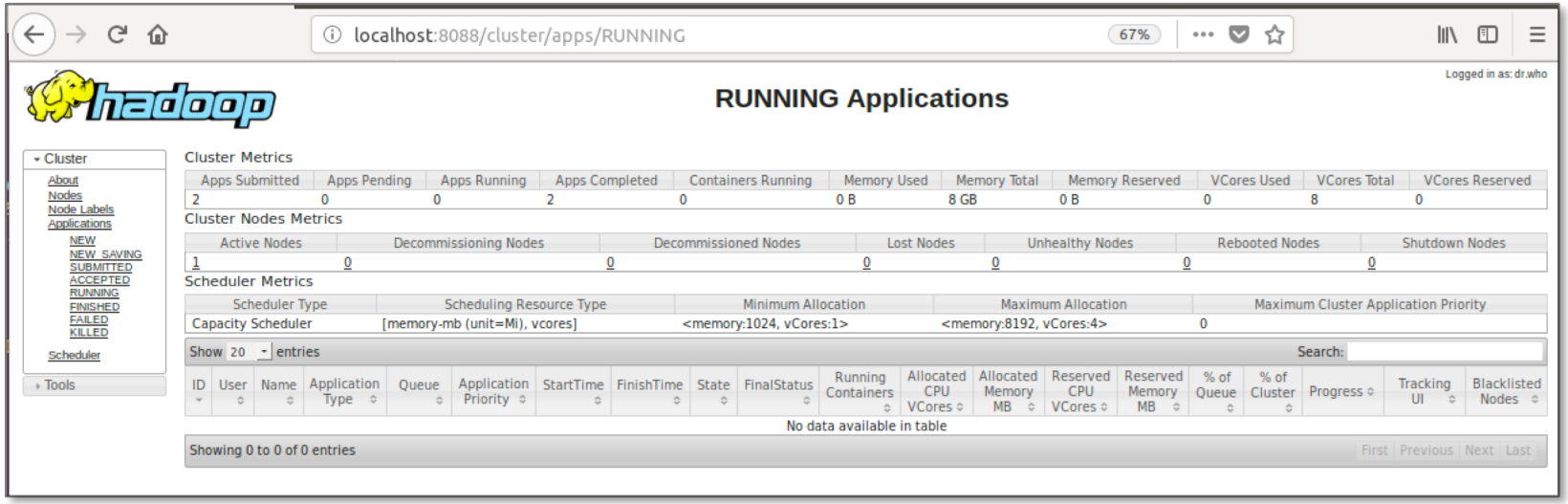

Start Spark (Standalone) – Test Install

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



Start Spark (Standalone) – Test Install

4. And see, nothing Running On Yarn:



Exercises Preparation II

Install and Setup PySpark

Start and Use PySpark (on Yarn) – Test Install

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

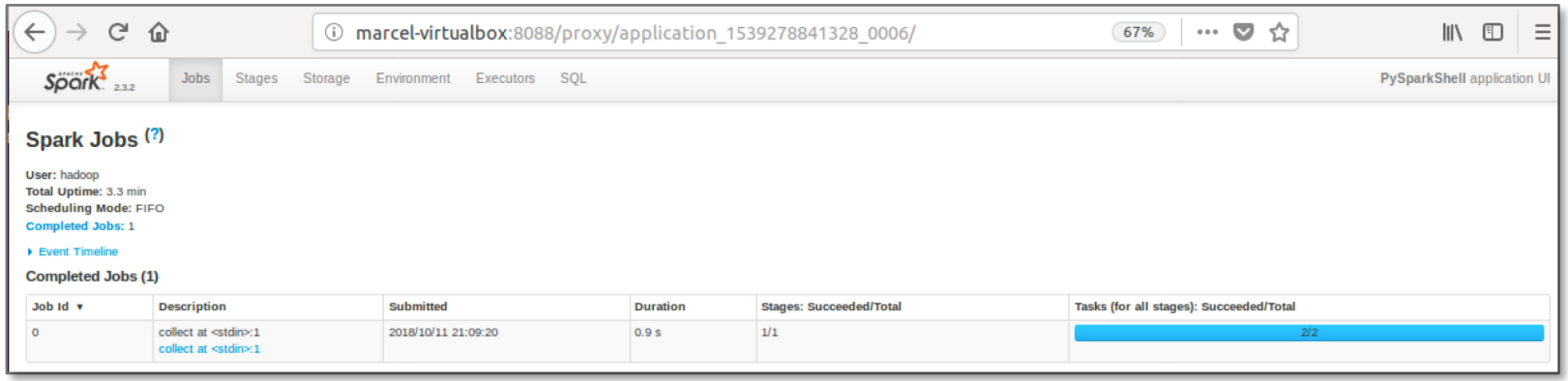
```
pyspark --master yarn
```

```
Python 2.7.15rc1 (default, Apr 15 2018, 21:51:34)
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.2
      /_/

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]
>>>
```

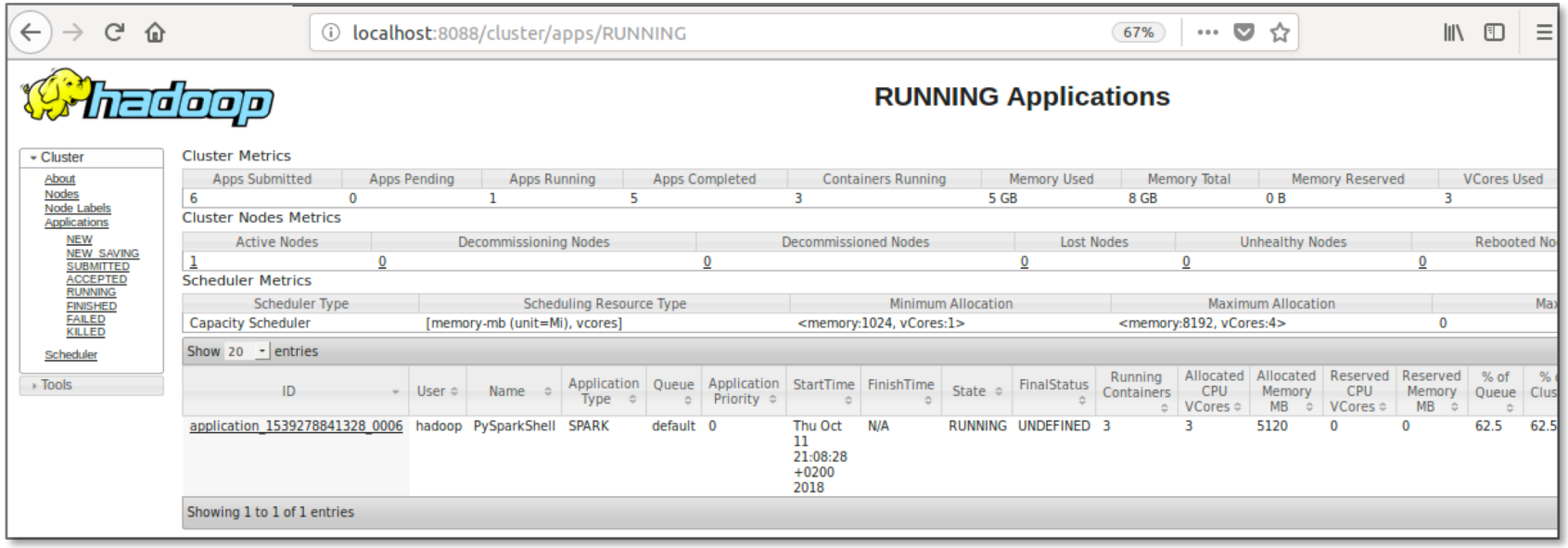

Start and Use PySpark (on Yarn) – Test Install

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



Start and Use PySpark (on Yarn) – Test Install

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



Exercises Preparation III

Install and Setup Jupyter (on PySpark)

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
```

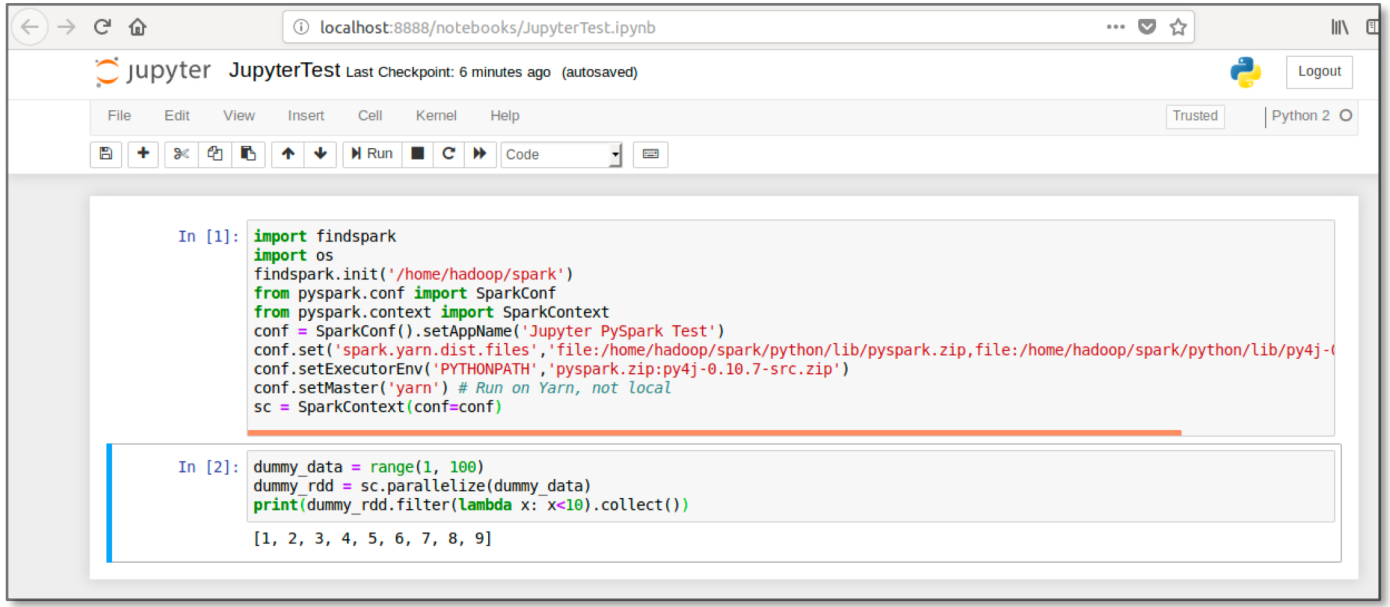
5. Start Jupyter Notebook

```
jupyter notebook
```


Start and Use Jupyter (on PySpark on Yarn)

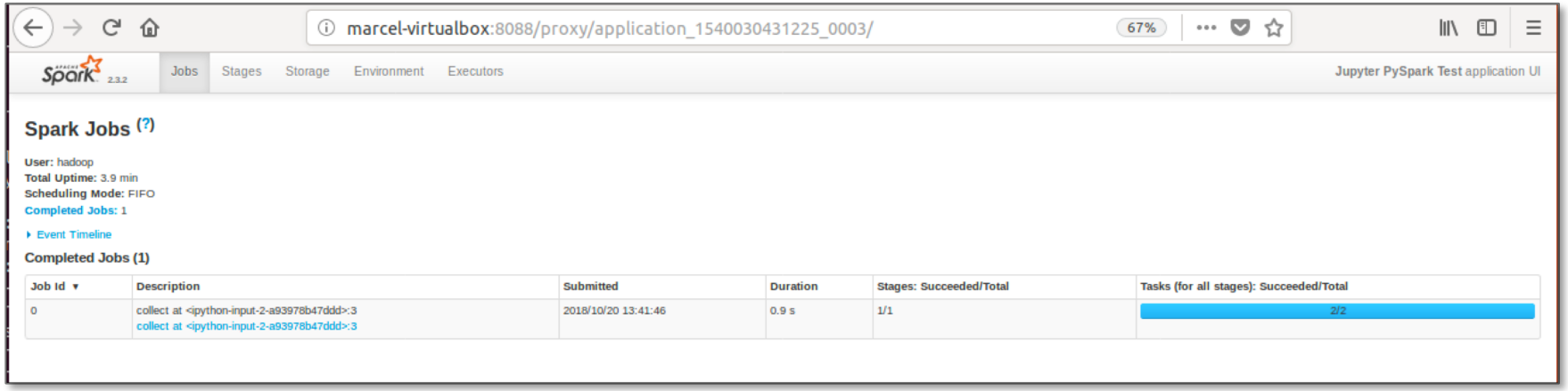
6. Open Jupyter Notebook and execute previous example as Python code:

```
jupyter notebook
```



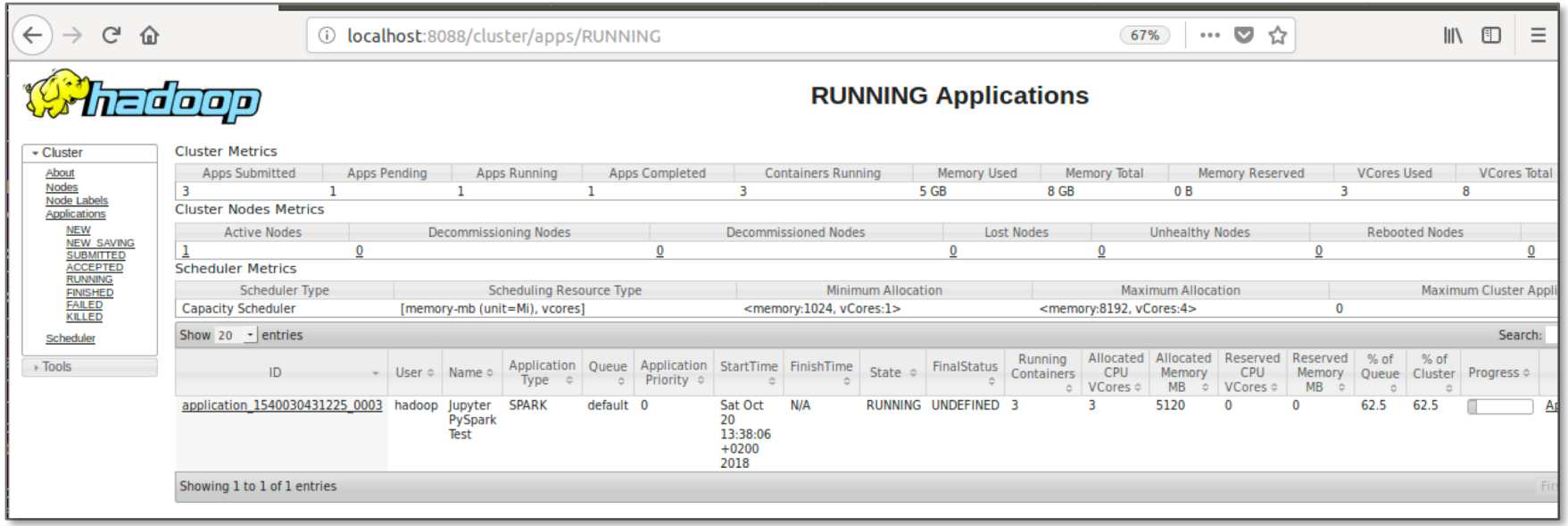
Use Jupyter (on PySpark on Yarn)

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



Use Jupyter (on PySpark on Yarn)

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



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)

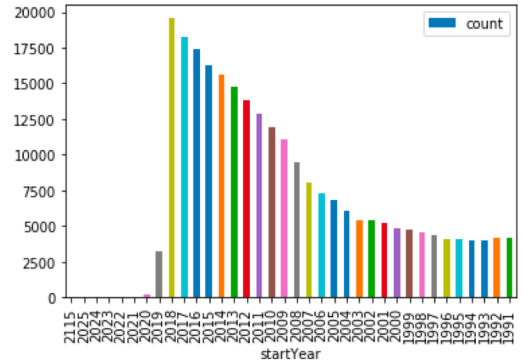
9. Basic Python: Plot results:

```
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 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)