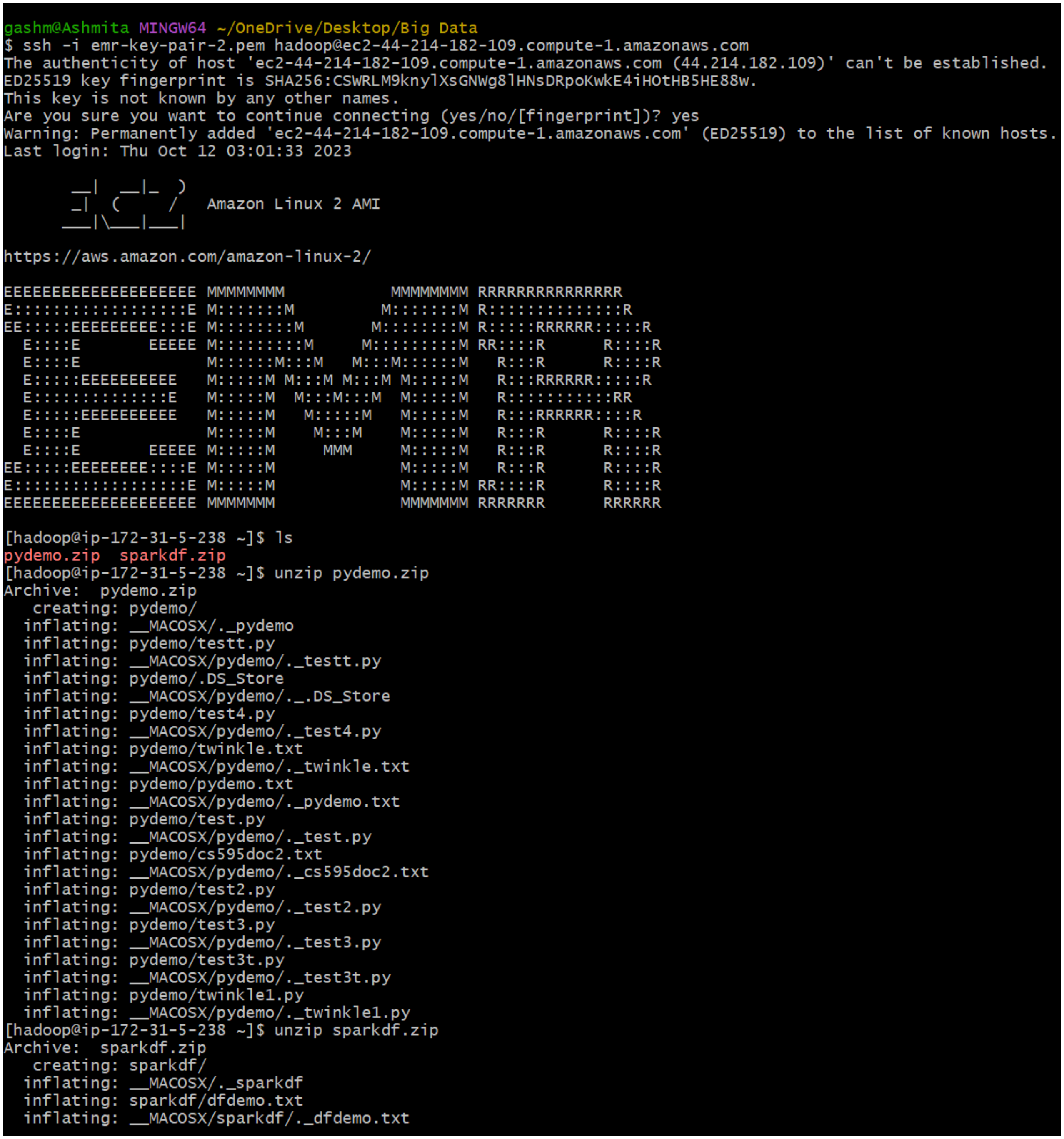
Assignment 7

CSP 554

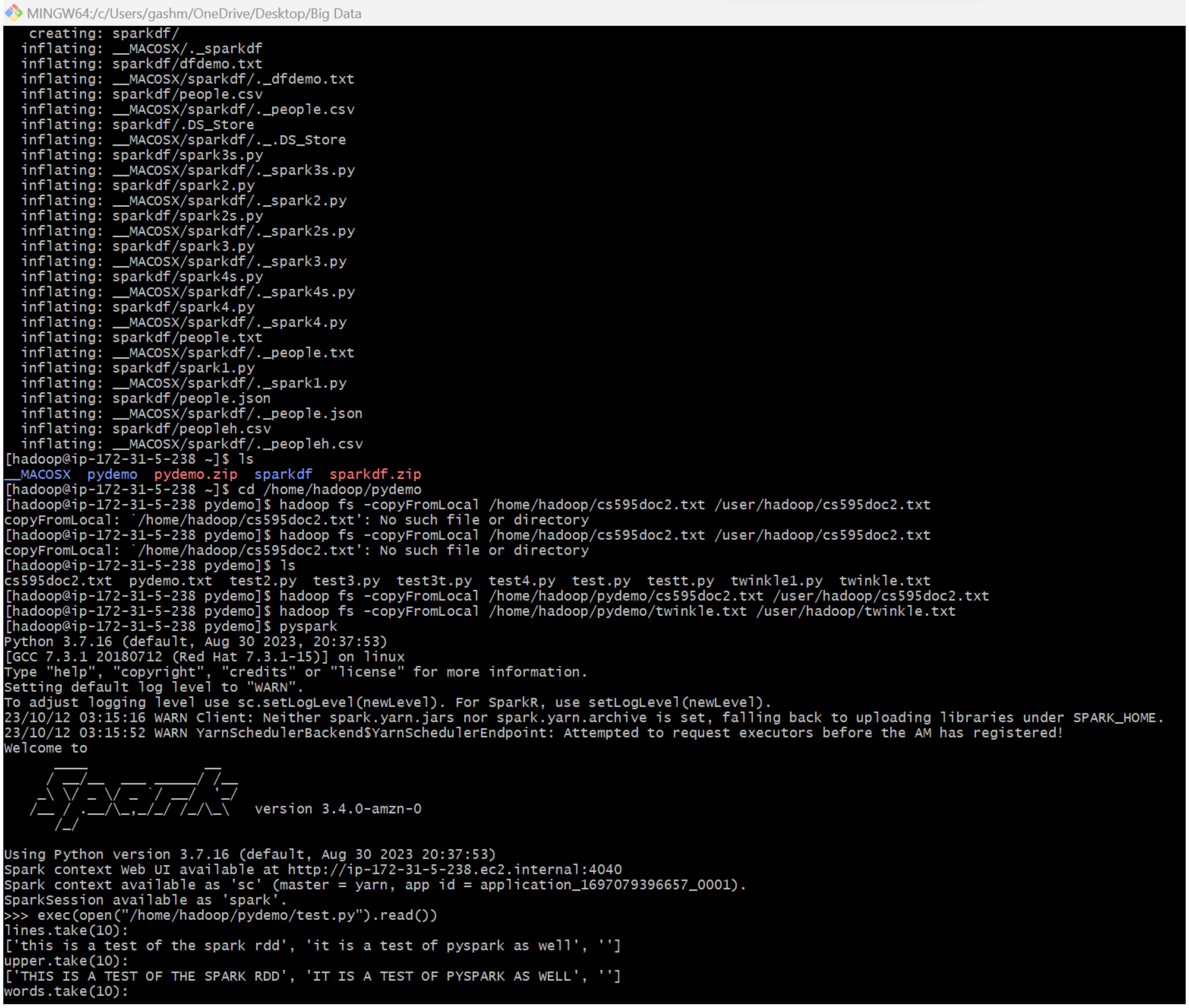
Big Data

Ashmita Gupta (A20512498)

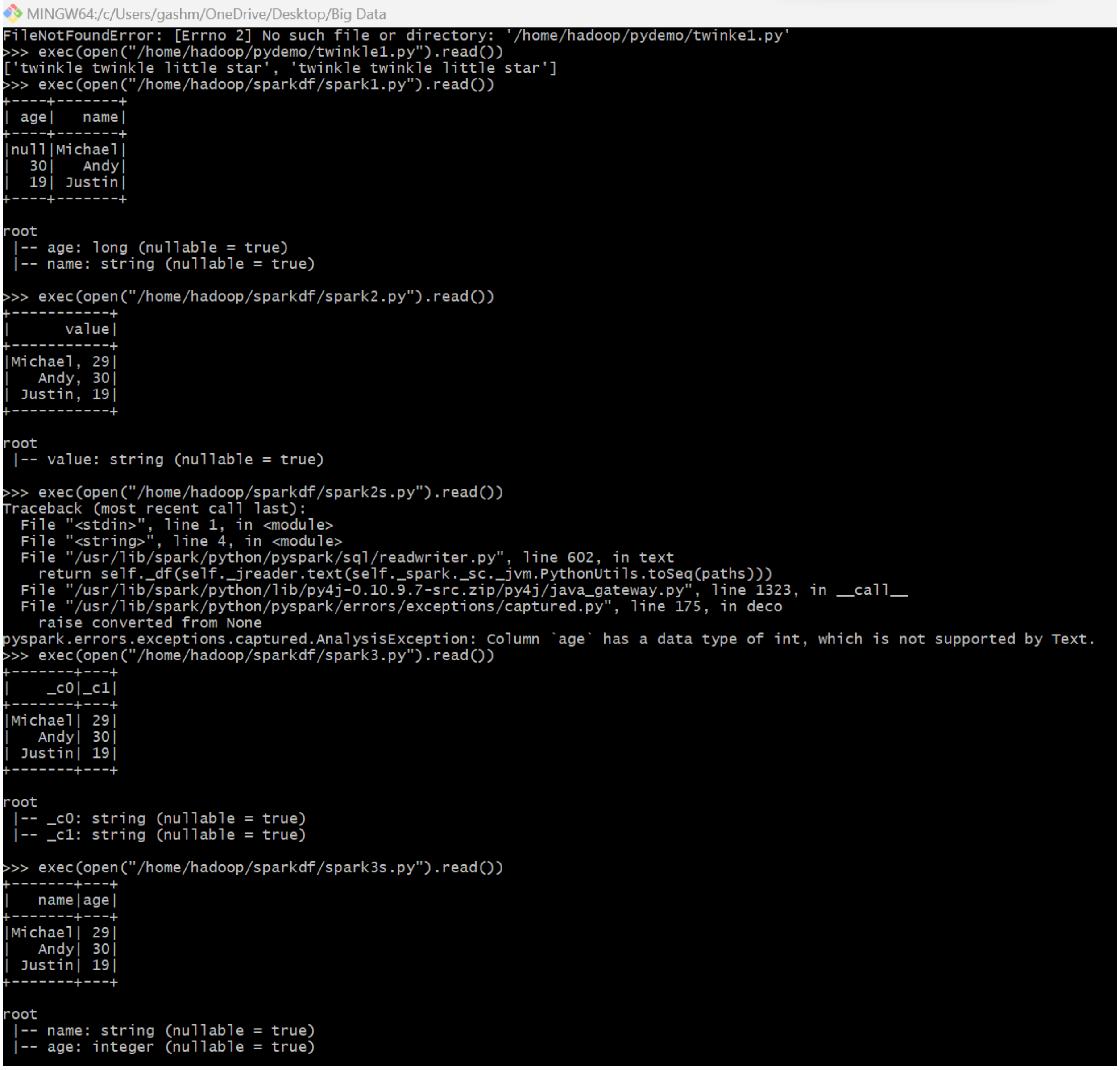
Running the demos:



```
gashm@Ashmita MINGW64 ~/OneDrive/Desktop/Big Data
$ ssh -i emr-key-pair-2.pem hadoop@ec2-44-214-182-109.compute-1.amazonaws.com
The authenticity of host 'ec2-44-214-182-109.compute-1.amazonaws.com (44.214.182.109)' can't be established.
ED25519 key fingerprint is SHA256:CSWRLM9knylXsGNWg8lHNsDRpoKwkE4iHOtHB5HE88w.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-44-214-182-109.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Last login: Thu Oct 12 03:01:33 2023

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-5-238 ~]$ ls
pydemo.zip  sparkdf.zip
[hadoop@ip-172-31-5-238 ~]$ unzip pydemo.zip
Archive:  pydemo.zip
   creating: pydemo/
  inflating: __MACOSX/._pydemo
  inflating: pydemo/testt.py
  inflating: __MACOSX/pydemo/._testt.py
  inflating: pydemo/.DS_Store
  inflating: __MACOSX/pydemo/._.DS_Store
  inflating: pydemo/test4.py
  inflating: __MACOSX/pydemo/._test4.py
  inflating: pydemo/twinkle.txt
  inflating: __MACOSX/pydemo/._twinkle.txt
  inflating: pydemo/pydemo.txt
  inflating: __MACOSX/pydemo/._pydemo.txt
  inflating: pydemo/test.py
  inflating: __MACOSX/pydemo/._test.py
  inflating: pydemo/cs595doc2.txt
  inflating: __MACOSX/pydemo/._cs595doc2.txt
  inflating: pydemo/test2.py
  inflating: __MACOSX/pydemo/._test2.py
  inflating: pydemo/test3.py
  inflating: __MACOSX/pydemo/._test3.py
  inflating: pydemo/test3t.py
  inflating: __MACOSX/pydemo/._test3t.py
  inflating: pydemo/twinkle1.py
  inflating: __MACOSX/pydemo/._twinkle1.py
[hadoop@ip-172-31-5-238 ~]$ unzip sparkdf.zip
Archive:  sparkdf.zip
   creating: sparkdf/
  inflating: __MACOSX/._sparkdf
  inflating: sparkdf/dfdemo.txt
  inflating: __MACOSX/sparkdf/._dfdemo.txt
```



```
   creating: sparkdf/
  inflating: __MACOSX/._sparkdf
  inflating: sparkdf/dfdemo.txt
  inflating: __MACOSX/sparkdf/._dfdemo.txt
  inflating: sparkdf/people.csv
  inflating: __MACOSX/sparkdf/._people.csv
  inflating: sparkdf/.DS_Store
  inflating: __MACOSX/sparkdf/._.DS_Store
  inflating: sparkdf/spark3s.py
  inflating: __MACOSX/sparkdf/._spark3s.py
  inflating: sparkdf/spark2.py
  inflating: __MACOSX/sparkdf/._spark2.py
  inflating: sparkdf/spark2s.py
  inflating: __MACOSX/sparkdf/._spark2s.py
  inflating: sparkdf/spark3.py
  inflating: __MACOSX/sparkdf/._spark3.py
  inflating: sparkdf/spark4s.py
  inflating: __MACOSX/sparkdf/._spark4s.py
  inflating: sparkdf/spark4.py
  inflating: __MACOSX/sparkdf/._spark4.py
  inflating: sparkdf/people.txt
  inflating: __MACOSX/sparkdf/._people.txt
  inflating: sparkdf/spark1.py
  inflating: __MACOSX/sparkdf/._spark1.py
  inflating: sparkdf/people.json
  inflating: __MACOSX/sparkdf/._people.json
  inflating: sparkdf/peopleh.csv
  inflating: __MACOSX/sparkdf/._peopleh.csv
[hadoop@ip-172-31-5-238 ~]$ ls
__MACOSX  pydemo  pydemo.zip  sparkdf  sparkdf.zip
[hadoop@ip-172-31-5-238 ~]$ cd /home/hadoop/pydemo
[hadoop@ip-172-31-5-238 pydemo]$ hadoop fs -copyFromLocal /home/hadoop/cs595doc2.txt /user/hadoop/cs595doc2.txt
copyFromLocal: `/home/hadoop/cs595doc2.txt': No such file or directory
[hadoop@ip-172-31-5-238 pydemo]$ hadoop fs -copyFromLocal /home/hadoop/cs595doc2.txt /user/hadoop/cs595doc2.txt
copyFromLocal: `/home/hadoop/cs595doc2.txt': No such file or directory
[hadoop@ip-172-31-5-238 pydemo]$ ls
cs595doc2.txt  pydemo.txt  test2.py  test3.py  test3t.py  test4.py  test.py  testt.py  twinkle1.py  twinkle.txt
[hadoop@ip-172-31-5-238 pydemo]$ hadoop fs -copyFromLocal /home/hadoop/pydemo/cs595doc2.txt /user/hadoop/cs595doc2.txt
[hadoop@ip-172-31-5-238 pydemo]$ hadoop fs -copyFromLocal /home/hadoop/pydemo/twinkle.txt /user/hadoop/twinkle.txt
[hadoop@ip-172-31-5-238 pydemo]$ pyspark
Python 3.7.16 (default, Aug 30 2023, 20:37:53)
[GCC 7.3.1 20180712 (Red Hat 7.3.1-15)] 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).
23/10/12 03:15:16 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
23/10/12 03:15:52 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.4.0-amzn-0
      /_/

Using Python version 3.7.16 (default, Aug 30 2023 20:37:53)
Spark context Web UI available at http://ip-172-31-5-238.ec2.internal:4040
Spark context available as 'sc' (master = yarn, app id = application_1697079396657_0001).
SparkSession available as 'spark'.
>>> exec(open("/home/hadoop/pydemo/test.py").read())
lines.take(10):
['this is a test of the spark rdd', 'it is a test of pyspark as well', '']
upper.take(10):
['THIS IS A TEST OF THE SPARK RDD', 'IT IS A TEST OF PYSPARK AS WELL', '']
words.take(10):
```

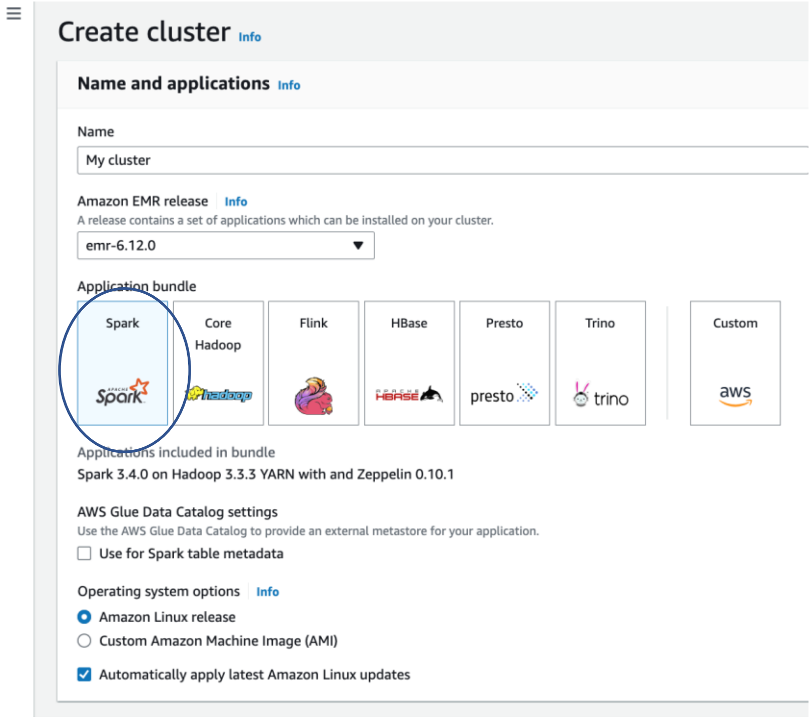


```
FileNotFoundError: [Errno 2] No such file or directory: '/home/hadoop/pydemo/twinke1.py'
>>> exec(open("/home/hadoop/pydemo/twinkle1.py").read())
['twinkle twinkle little star', 'twinkle twinkle little star']
>>> exec(open("/home/hadoop/sparkdf/spark1.py").read())
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+

root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)

>>> exec(open("/home/hadoop/sparkdf/spark2.py").read())
+-----------+
|      value|
+-----------+
|Michael, 29|
|   Andy, 30|
| Justin, 19|
+-----------+

root
 |-- value: string (nullable = true)

>>> exec(open("/home/hadoop/sparkdf/spark2s.py").read())
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 4, in <module>
  File "/usr/lib/spark/python/pyspark/sql/readwriter.py", line 602, in text
    return self._df(self._jreader.text(self._spark._sc._jvm.PythonUtils.toSeq(paths)))
  File "/usr/lib/spark/python/lib/py4j-0.10.9.7-src.zip/py4j/java_gateway.py", line 1323, in __call__
  File "/usr/lib/spark/python/pyspark/errors/exceptions/captured.py", line 175, in deco
    raise converted from None
pyspark.errors.exceptions.captured.AnalysisException: Column `age` has a data type of int, which is not supported by Text.
>>> exec(open("/home/hadoop/sparkdf/spark3.py").read())
+-------+---+
|    _c0|_c1|
+-------+---+
|Michael| 29|
|   Andy| 30|
| Justin| 19|
+-------+---+

root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)

>>> exec(open("/home/hadoop/sparkdf/spark3s.py").read())
+-------+---+
|   name|age|
+-------+---+
|Michael| 29|
|   Andy| 30|
| Justin| 19|
+-------+---+

root
 |-- name: string (nullable = true)
 |-- age: integer (nullable = true)
```

Exercise 1)

Step A

Start up a EMR cluster as previously, but instead of choosing the “Core Hadoop” configuration chose the “Spark” configuration (see below), otherwise proceed as before.



Create cluster Info

Name and applications Info

Name

My cluster

Amazon EMR release Info

A release contains a set of applications which can be installed on your cluster.

emr-6.12.0 ▼

Application bundle

[✓ Spark] Core Hadoop Flink HBase Presto Trino Custom

Applications included in bundle

Spark 3.4.0 on Hadoop 3.3.3 YARN with and Zeppelin 0.10.1

AWS Glue Data Catalog settings

Use the AWS Glue Data Catalog to provide an external metastore for your application.

☐ Use for Spark table metadata

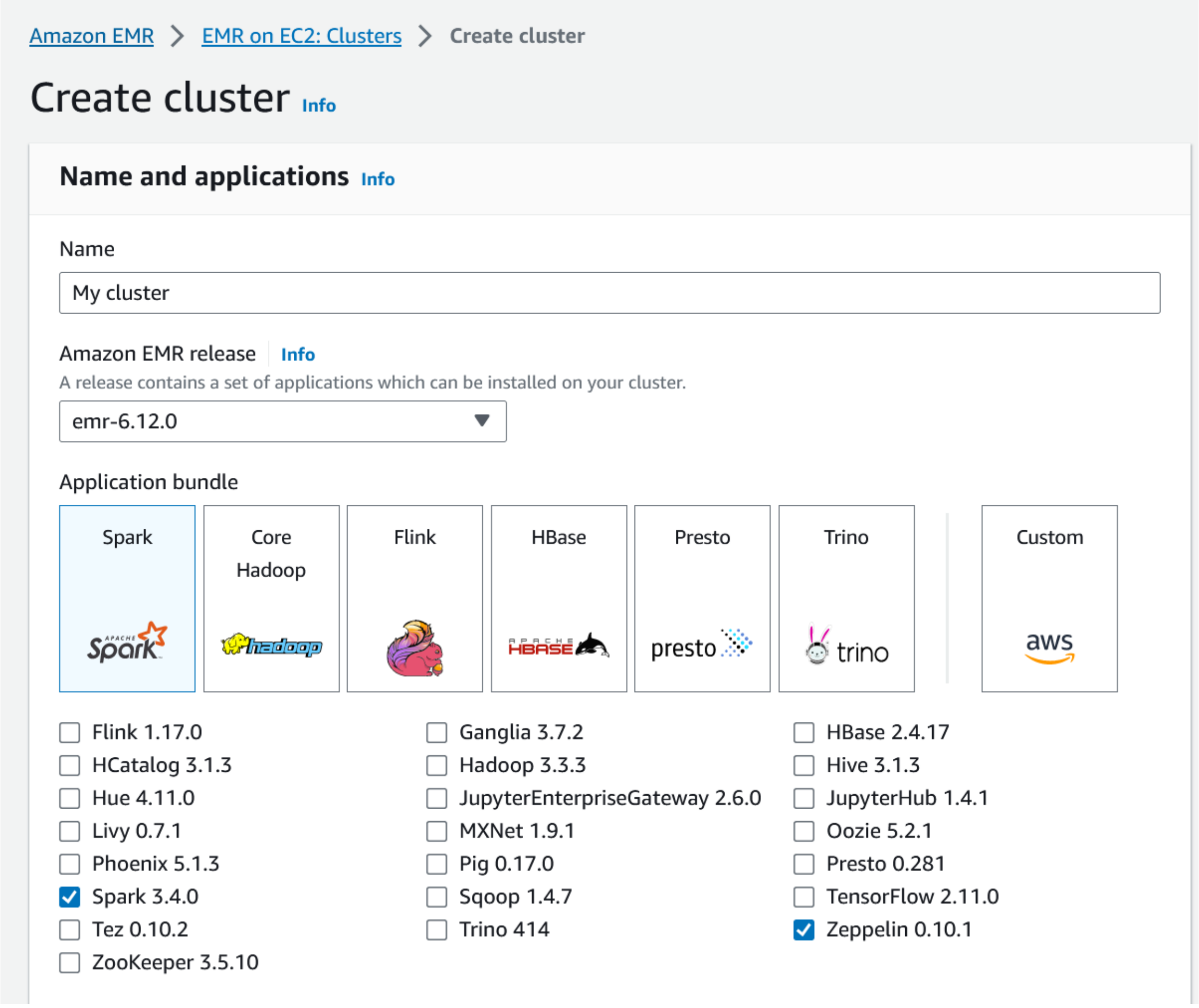
Operating system options Info

● Amazon Linux release

○ Custom Amazon Machine Image (AMI)

☑ Automatically apply latest Amazon Linux updates

Ans: Creating a new cluster with the "Spark" configuration:



Amazon EMR > EMR on EC2: Clusters > Create cluster

Create cluster Info

Name and applications Info

Name

My cluster

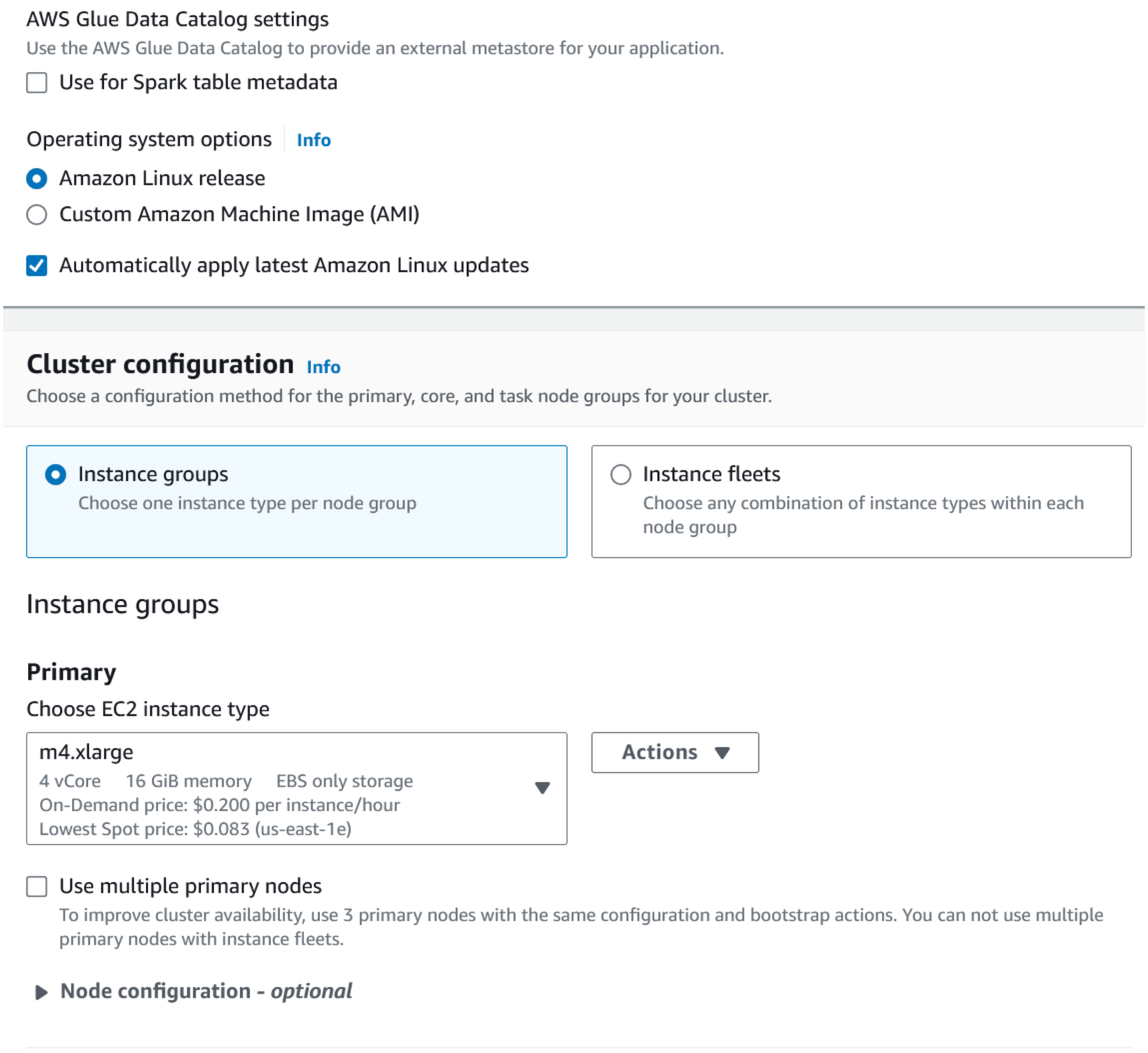
Amazon EMR release Info

A release contains a set of applications which can be installed on your cluster.

emr-6.12.0 ▼

Application bundle

- ☐ Flink 1.17.0
- ☐ HCatalog 3.1.3
- ☐ Hue 4.11.0
- ☐ Livy 0.7.1
- ☐ Phoenix 5.1.3
- ☑ Spark 3.4.0
- ☐ Tez 0.10.2
- ☐ ZooKeeper 3.5.10
- ☐ Ganglia 3.7.2
- ☐ Hadoop 3.3.3
- ☐ JupyterEnterpriseGateway 2.6.0
- ☐ MXNet 1.9.1
- ☐ Pig 0.17.0
- ☐ Sqoop 1.4.7
- ☐ Trino 414
- ☐ HBase 2.4.17
- ☐ Hive 3.1.3
- ☐ JupyterHub 1.4.1
- ☐ Oozie 5.2.1
- ☐ Presto 0.281
- ☐ TensorFlow 2.11.0
- ☑ Zeppelin 0.10.1



AWS Glue Data Catalog settings

Use the AWS Glue Data Catalog to provide an external metastore for your application.

□ Use for Spark table metadata

Operating system options | Info

● Amazon Linux release

○ Custom Amazon Machine Image (AMI)

☑ Automatically apply latest Amazon Linux updates

Cluster configuration Info

Choose a configuration method for the primary, core, and task node groups for your cluster.

● Instance groups
Choose one instance type per node group

○ Instance fleets
Choose any combination of instance types within each node group

Instance groups

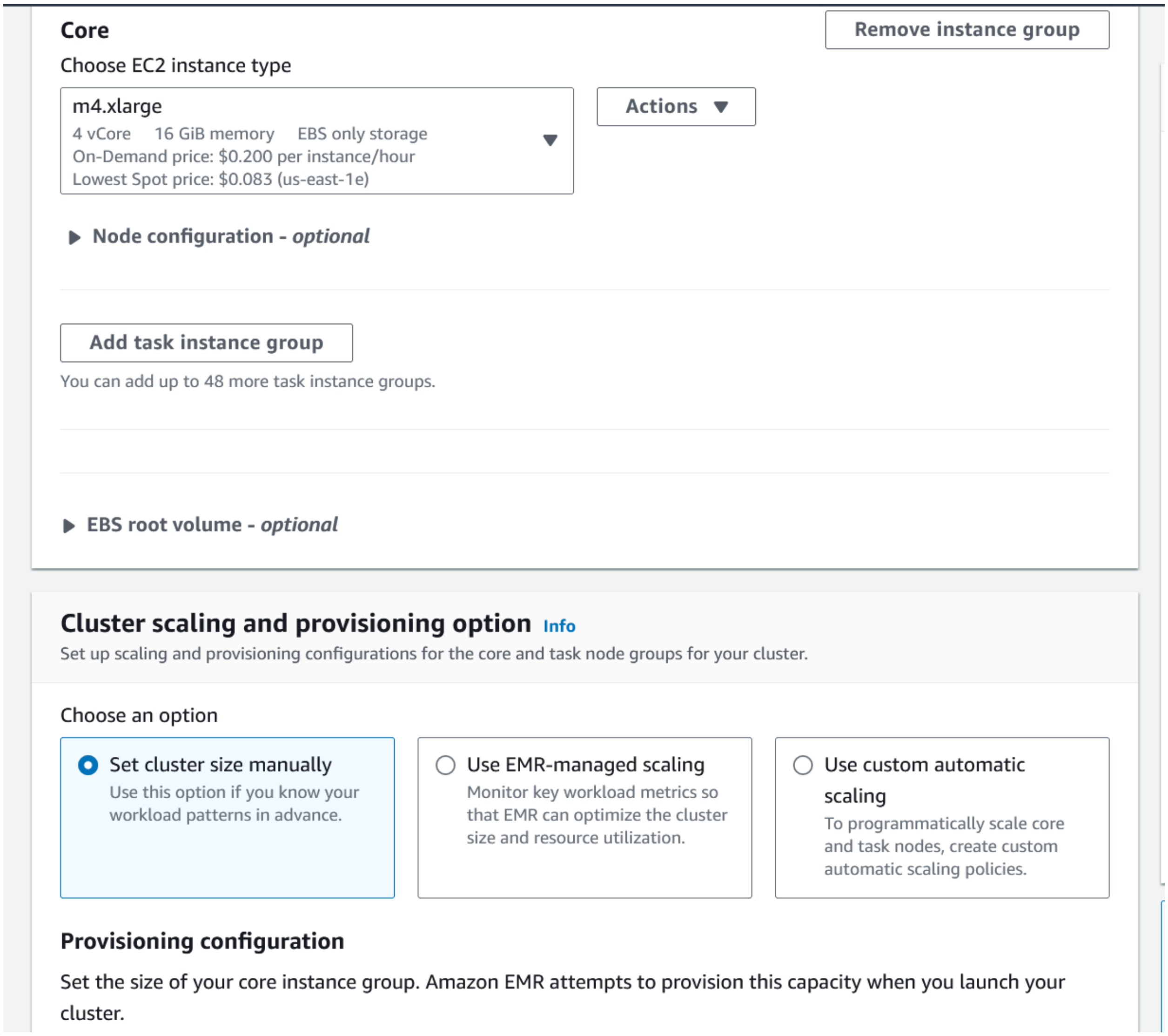
Primary

Choose EC2 instance type

m4.xlarge
4 vCore 16 GiB memory EBS only storage
On-Demand price: $0.200 per instance/hour
Lowest Spot price: $0.083 (us-east-1e)

□ Use multiple primary nodes
To improve cluster availability, use 3 primary nodes with the same configuration and bootstrap actions. You can not use multiple primary nodes with instance fleets.

▶ **Node configuration - *optional***



Core

Choose EC2 instance type

m4.xlarge
4 vCore 16 GiB memory EBS only storage
On-Demand price: $0.200 per instance/hour
Lowest Spot price: $0.083 (us-east-1e)

▶ **Node configuration - *optional***

You can add up to 48 more task instance groups.

▶ **EBS root volume - *optional***

Cluster scaling and provisioning option

Set up scaling and provisioning configurations for the core and task node groups for your cluster.

Choose an option

● Set cluster size manually
Use this option if you know your workload patterns in advance.

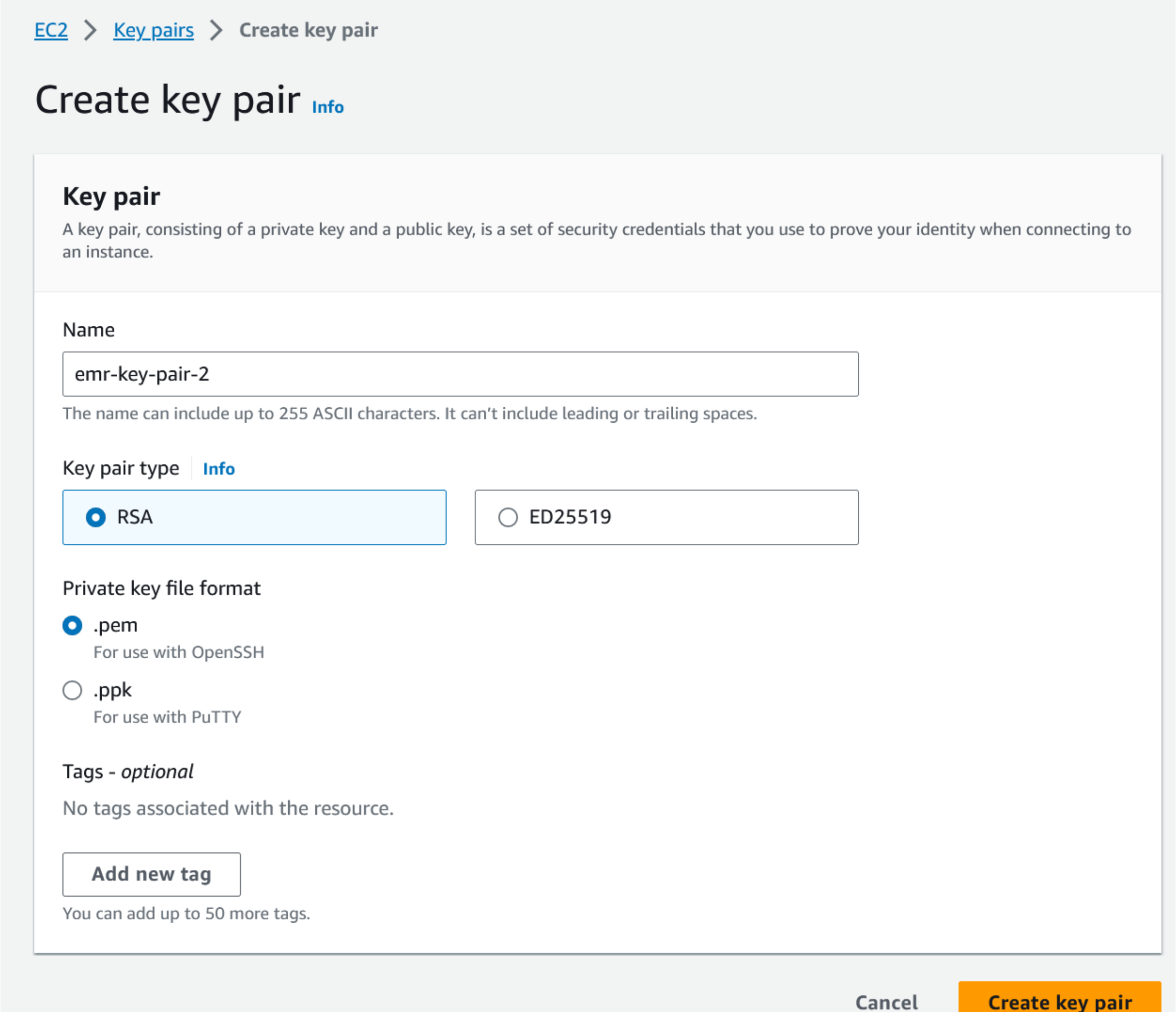
○ Use EMR-managed scaling
Monitor key workload metrics so that EMR can optimize the cluster size and resource utilization.

○ Use custom automatic scaling
To programmatically scale core and task nodes, create custom automatic scaling policies.

Provisioning configuration

Set the size of your core instance group. Amazon EMR attempts to provision this capacity when you launch your cluster.

Creating a new key pair for the cluster:



EC2 > Key pairs > Create key pair

Create key pair Info

Key pair

A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.

Name

emr-key-pair-2

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type Info

● RSA ○ ED25519

Private key file format

● .pem
For use with OpenSSH

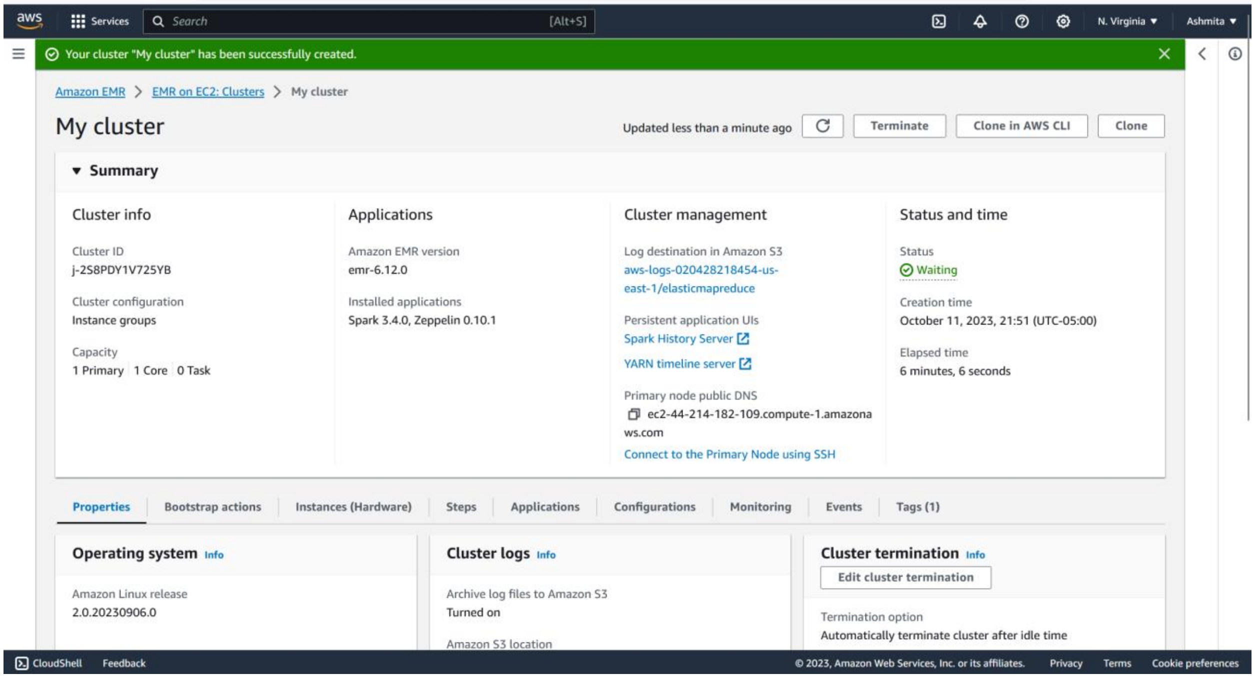
○ .ppk
For use with PuTTY

Tags - *optional*

No tags associated with the resource.

You can add up to 50 more tags.

Cluster created:



Your cluster "My cluster" has been successfully created.

Amazon EMR > EMR on EC2: Clusters > My cluster

My cluster

Updated less than a minute ago

▾ Summary

Cluster info

Cluster ID
j-2S8PDY1V725YB

Cluster configuration
Instance groups

Capacity
1 Primary | 1 Core | 0 Task

Applications

Amazon EMR version
emr-6.12.0

Installed applications
Spark 3.4.0, Zeppelin 0.10.1

Cluster management

Log destination in Amazon S3
aws-logs-020428218454-us-east-1/elasticmapreduce

Persistent application UIs
Spark History Server
YARN timeline server

Primary node public DNS
ec2-44-214-182-109.compute-1.amazonaws.com

Connect to the Primary Node using SSH

Status and time

Status
Waiting

Creation time
October 11, 2023, 21:51 (UTC-05:00)

Elapsed time
6 minutes, 6 seconds

Operating system Info

Amazon Linux release
2.0.20230906.0

Cluster logs Info

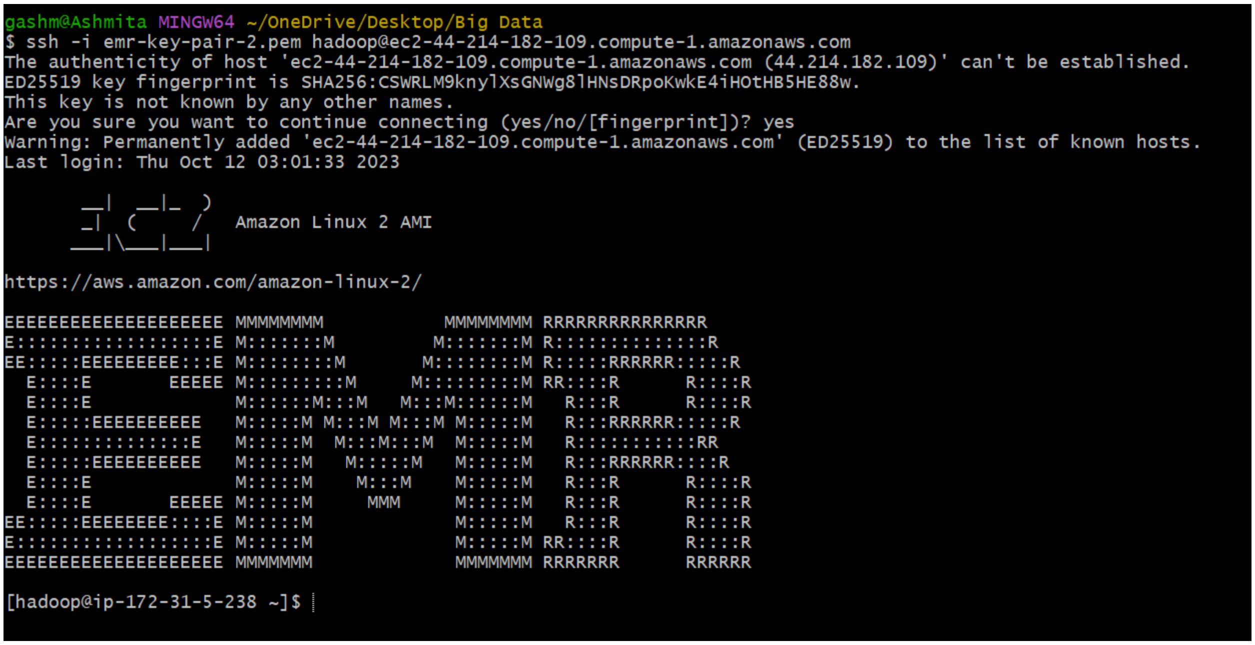
Archive log files to Amazon S3
Turned on

Amazon S3 location

Cluster termination Info

Termination option
Automatically terminate cluster after idle time

ssh to cluster:



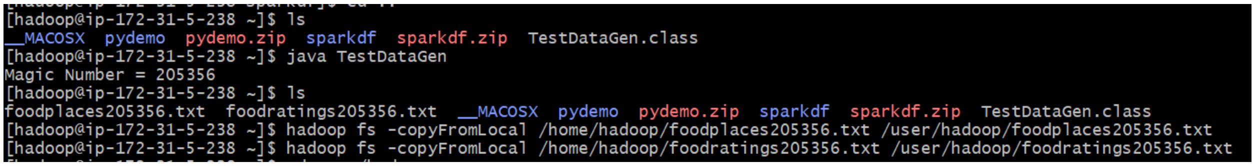
```
gashm@Ashmita MINGW64 ~/OneDrive/Desktop/Big Data
$ ssh -i emr-key-pair-2.pem hadoop@ec2-44-214-182-109.compute-1.amazonaws.com
The authenticity of host 'ec2-44-214-182-109.compute-1.amazonaws.com (44.214.182.109)' can't be established.
ED25519 key fingerprint is SHA256:CSWRLM9kny1XsGNWg8lHNsDRpoKwkE4iHOtHB5HE88w.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-44-214-182-109.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Last login: Thu Oct 12 03:01:33 2023

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-5-238 ~]$
```

Step B

Use the TestDataGen program from previous assignments to generate new data files.

Copy both generated files to the HDFS directory “/user/hadoop”

Ans. Copied both generated files to the HDFS directory “/user/hadoop”



```
[hadoop@ip-172-31-5-238 ~]$ ls
__MACOSX  pydemo  pydemo.zip  sparkdf  sparkdf.zip  TestDataGen.class
[hadoop@ip-172-31-5-238 ~]$ java TestDataGen
Magic Number = 205356
[hadoop@ip-172-31-5-238 ~]$ ls
foodplaces205356.txt  foodratings205356.txt  __MACOSX  pydemo  pydemo.zip  sparkdf  sparkdf.zip  TestDataGen.class
[hadoop@ip-172-31-5-238 ~]$ hadoop fs -copyFromLocal /home/hadoop/foodplaces205356.txt /user/hadoop/foodplaces205356.txt
[hadoop@ip-172-31-5-238 ~]$ hadoop fs -copyFromLocal /home/hadoop/foodratings205356.txt /user/hadoop/foodratings205356.txt
```

Magic number: 205356

Step C

Load the ‘foodratings’ file as a ‘csv’ file into a DataFrame called foodratings. When doing so specify a schema having fields of the following names and types:

Field Name	Field Type
Name	String
food1	Integer
food2	Integer
food3	Integer
food4	Integer
placeid	Integer

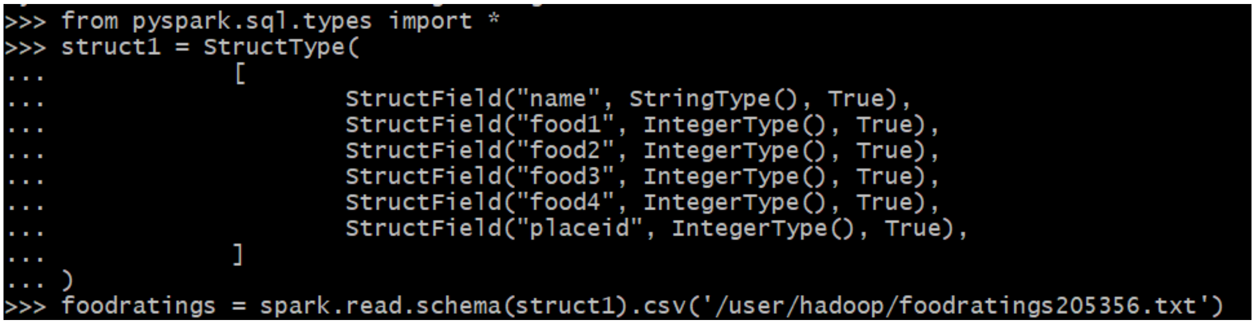
As the results of this exercise provide the magic number, *the code you execute* and screen shots of the following commands:

foodratings.printSchema()

foodratings.show(5)

Ans.

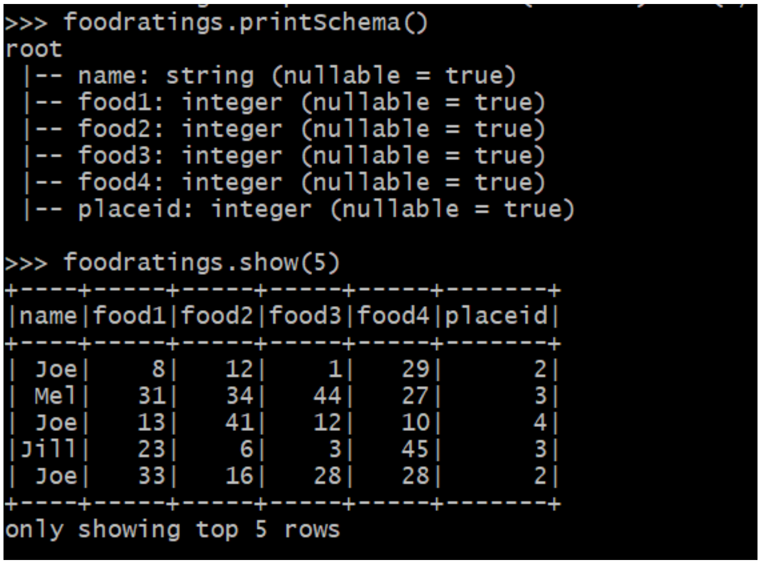
Loading the ‘foodratings’ file as a ‘csv’ file into a DataFrame called foodratings:



```
>>> from pyspark.sql.types import *
>>> struct1 = StructType(
...             [
...                     StructField("name", StringType(), True),
...                     StructField("food1", IntegerType(), True),
...                     StructField("food2", IntegerType(), True),
...                     StructField("food3", IntegerType(), True),
...                     StructField("food4", IntegerType(), True),
...                     StructField("placeid", IntegerType(), True),
...             ]
... )
>>> foodratings = spark.read.schema(struct1).csv('/user/hadoop/foodratings205356.txt')
```

foodratings.printSchema()

foodratings.show(5)



```
>>> foodratings.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Joe|    8|   12|    1|   29|      2|
| Mel|   31|   34|   44|   27|      3|
| Joe|   13|   41|   12|   10|      4|
|Jill|   23|    6|    3|   45|      3|
| Joe|   33|   16|   28|   28|      2|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

<u>Exercise 2)</u>

Load the ‘foodplaces’ file as a ‘csv’ file into a DataFrame called foodplaces. When doing so specify a schema having fields of the following names and types:

Field Name	Field Type
placeid	Integer
placename	String

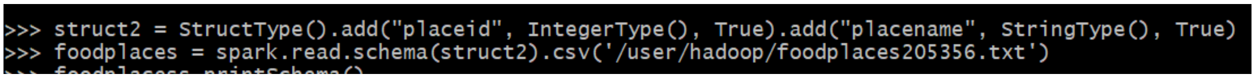
As the results of this exercise provide *the code you execute* and screen shots of the following commands:

foodplaces.printSchema()

foodplaces.show(5)

Ans.

Load the ‘foodplaces’ file as a ‘csv’ file into a DataFrame called foodplaces :

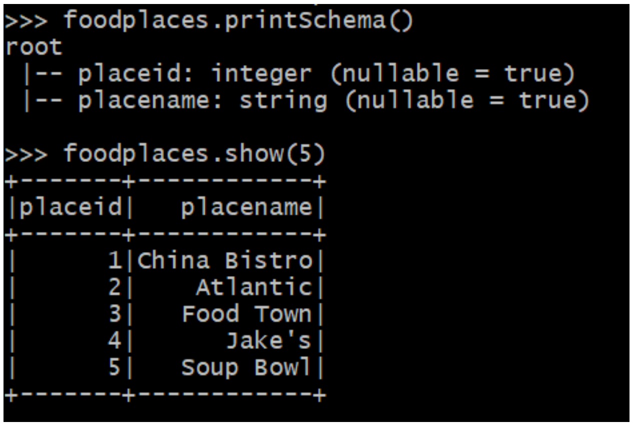


```
>>> struct2 = StructType().add("placeid", IntegerType(), True).add("placename", StringType(), True)
>>> foodplaces = spark.read.schema(struct2).csv('/user/hadoop/foodplaces205356.txt')
```

Magic number: 205356

foodplaces.printSchema()

foodplaces.show(5)



```
>>> foodplaces.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> foodplaces.show(5)
+-------+-----------+
|placeid|  placename|
+-------+-----------+
|      1|China Bistro|
|      2|   Atlantic|
|      3|  Food Town|
|      4|     Jake's|
|      5|  Soup Bowl|
+-------+-----------+
```

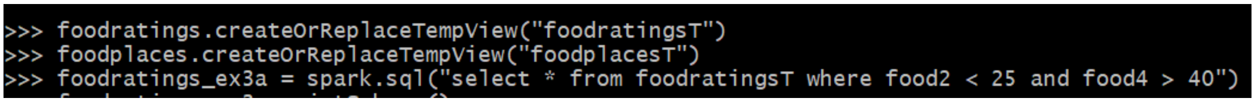
Exercise 3)

Step A

Register the DataFrames created in exercise 1 and 2 as tables called “foodratingsT” and “foodplacesT”

Ans.

Registering the DataFrames as tables called “foodratingsT” and “foodplacesT”:



```
>>> foodratings.createOrReplaceTempView("foodratingsT")
>>> foodplaces.createOrReplaceTempView("foodplacesT")
>>> foodratings_ex3a = spark.sql("select * from foodratingsT where food2 < 25 and food4 > 40")
```

Step B

Use a SQL query on the table “foodratingsT” to create a new DataFrame called foodratings_ex3a holding records which meet the following condition: food2 < 25 and food4 > 40. Remember, when defining conditions in your code use maximum parentheses.

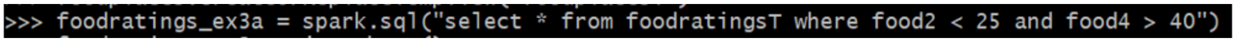
As the results of this step *provide the code you execute* and screen shots of the following commands:

foodratings_ex3a.printSchema()

foodratings_ex3a.show(5)

Ans.

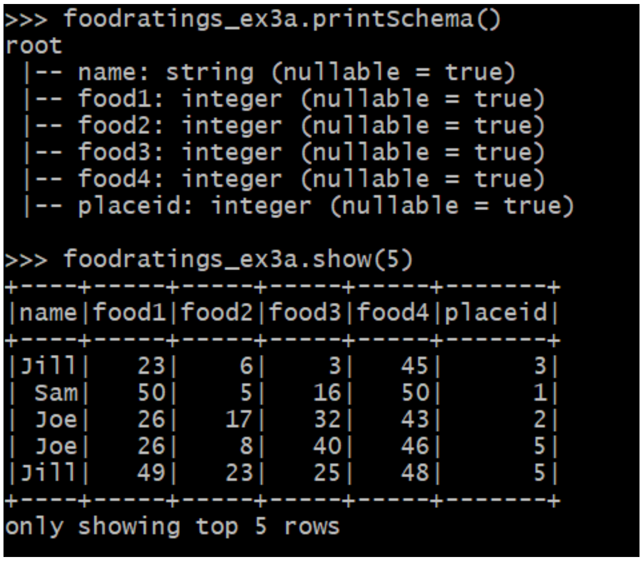
foodratings_ex3a holding records which meet the condition: food2 < 25 and food4 > 40:



```
>>> foodratings_ex3a = spark.sql("select * from foodratingsT where food2 < 25 and food4 > 40")
```

foodratings_ex3a.printSchema()

foodratings_ex3a.show(5)



```
>>> foodratings_ex3a.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex3a.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
|Jill|   23|    6|    3|   45|      3|
| Sam|   50|    5|   16|   50|      1|
| Joe|   26|   17|   32|   43|      2|
| Joe|   26|    8|   40|   46|      5|
|Jill|   49|   23|   25|   48|      5|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

Step C

Use a SQL query on the table “foodplacesT” to create a new DataFrame called foodplaces_ex3b holding records which meet the following condition: placeid > 3

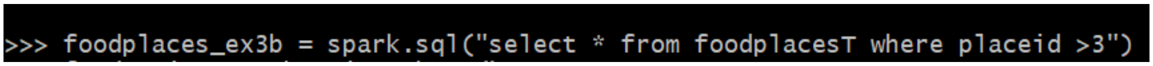
As the results of this step *provide the code you execute* and screen shots of the following commands:

foodplaces_ex3b.printSchema()

foodplaces_ex3b.show(5)

Ans.

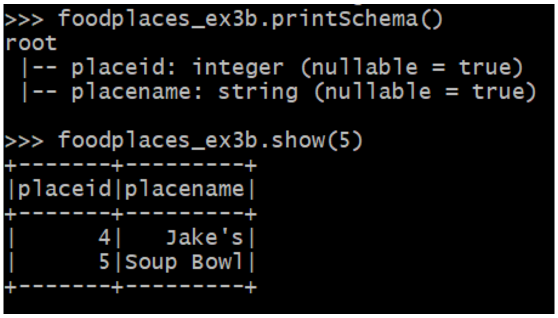
Below is the SQL query on the table “foodplacesT” to create a new DataFrame called foodplaces_ex3b holding records which meet the following condition: placeid > 3



```
>>> foodplaces_ex3b = spark.sql("select * from foodplacesT where placeid >3")
```

foodplaces_ex3b.printSchema()

foodplaces_ex3b.show(5)



```
>>> foodplaces_ex3b.printSchema()
root
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> foodplaces_ex3b.show(5)
+-------+---------+
|placeid|placename|
+-------+---------+
|      4|   Jake's|
|      5|Soup Bowl|
+-------+---------+
```

Exercise 4)

Use a transformation (not a SparkSQL query) on the DataFrame ‘foodratings’ created in exercise 1 to create a new DataFrame called foodratings_ex4 that includes only those records (rows) where the ‘name’ field is “Mel” and food3 < 25.

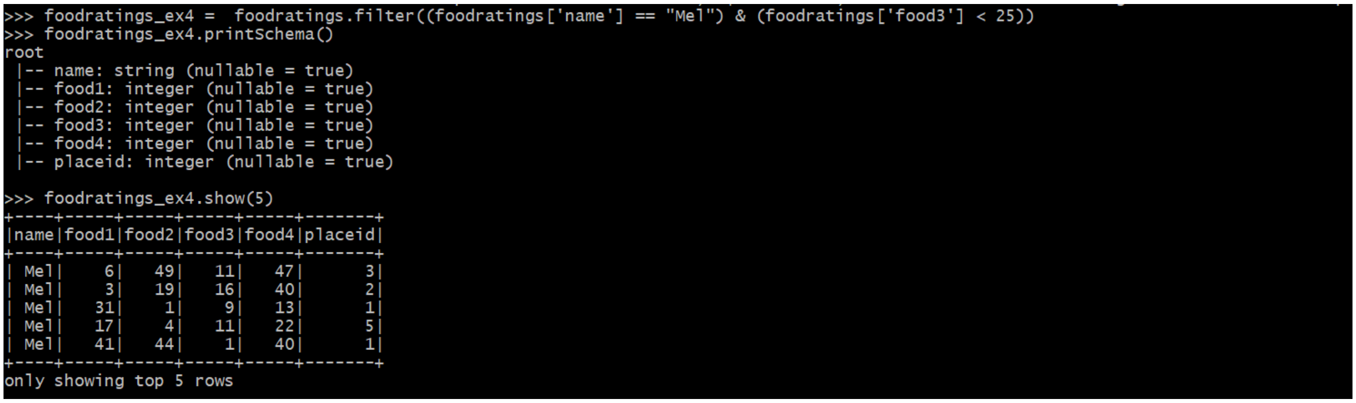
As the results of this step provide the code you execute and screen shots of the following commands:

foodratings_ex4.printSchema()

foodratings_ex4.show(5)

Ans.

New DataFrame called foodratings_ex4 includes only those records where the ‘name’ field is “Mel” and food3 < 25:



```
>>> foodratings_ex4 =  foodratings.filter((foodratings['name'] == "Mel") & (foodratings['food3'] < 25))
>>> foodratings_ex4.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex4.show(5)
+----+-----+-----+-----+-----+-------+
|name|food1|food2|food3|food4|placeid|
+----+-----+-----+-----+-----+-------+
| Mel|    6|   49|   11|   47|      3|
| Mel|    3|   19|   16|   40|      2|
| Mel|   31|    1|    9|   13|      1|
| Mel|   17|    4|   11|   22|      5|
| Mel|   41|   44|    1|   40|      1|
+----+-----+-----+-----+-----+-------+
only showing top 5 rows
```

Exercise 5)

Use a transformation (**not a SparkSQL query**) on the DataFrame ‘foodratings’ created in exercise 1 to create a new DataFrame called foodratings_ex5 that includes only the columns (fields) ‘name’ and ‘placeid’

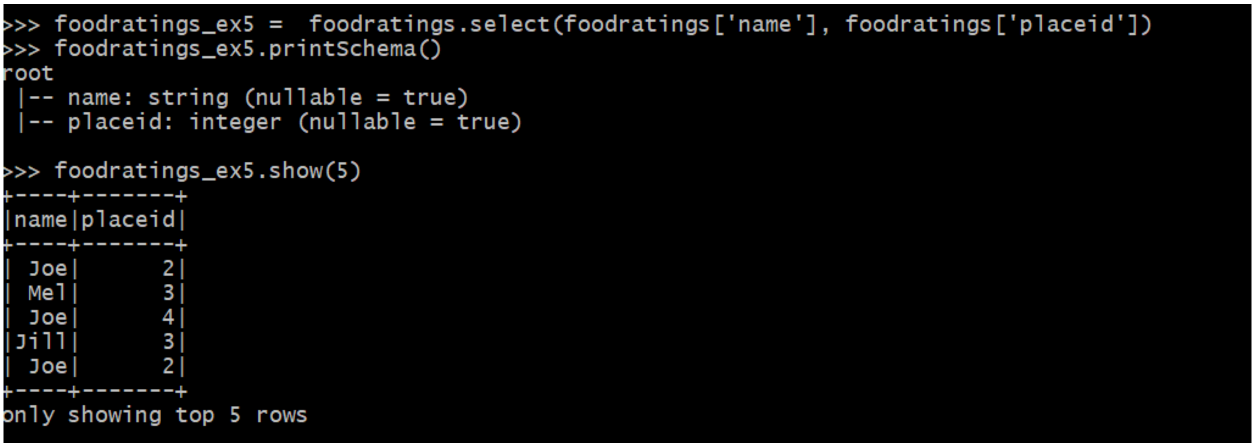
As the results of this step provide the code you execute and screen shots of the following commands:

foodratings_ex5.printSchema()

foodratings_ex5.show(5)

Ans.

New DataFrame called foodratings_ex5 that includes only the columns (fields) ‘name’ and ‘placeid’:



```
>>> foodratings_ex5 =  foodratings.select(foodratings['name'], foodratings['placeid'])
>>> foodratings_ex5.printSchema()
root
 |-- name: string (nullable = true)
 |-- placeid: integer (nullable = true)

>>> foodratings_ex5.show(5)
+----+-------+
|name|placeid|
+----+-------+
| Joe|      2|
| Mel|      3|
| Joe|      4|
|Jill|      3|
| Joe|      2|
+----+-------+
only showing top 5 rows
```

Exercise 6)

Use a transformation (**not a SparkSQL query**) to create a new DataFrame called ex6 which is the inner join, on placeid, of the DataFrames ‘foodratings’ and ‘foodplaces’ created in exercises 1 and 2

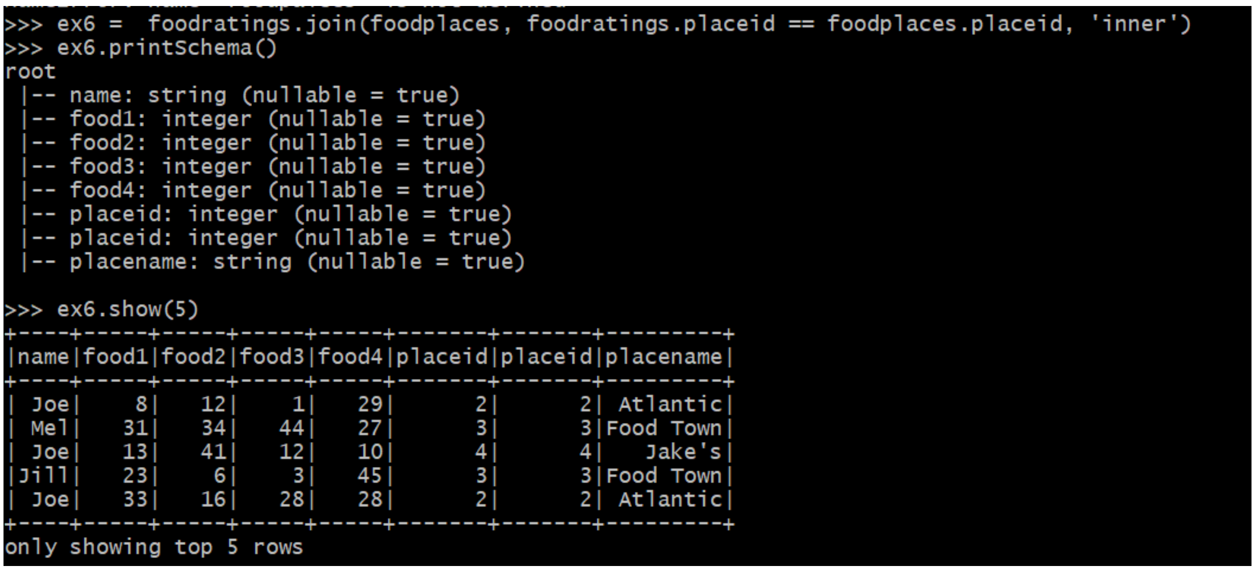
As the results of this step provide the code you execute and screen shots of the following commands:

ex6.printSchema()

ex6.show(5)

Ans.

New DataFrame called ex6 which is the inner join, on placeid, of the DataFrames ‘foodratings’ and ‘foodplaces’ created in exercises 1 and 2:



```
>>> ex6 =  foodratings.join(foodplaces, foodratings.placeid == foodplaces.placeid, 'inner')
>>> ex6.printSchema()
root
 |-- name: string (nullable = true)
 |-- food1: integer (nullable = true)
 |-- food2: integer (nullable = true)
 |-- food3: integer (nullable = true)
 |-- food4: integer (nullable = true)
 |-- placeid: integer (nullable = true)
 |-- placeid: integer (nullable = true)
 |-- placename: string (nullable = true)

>>> ex6.show(5)
+----+-----+-----+-----+-----+-------+-------+---------+
|name|food1|food2|food3|food4|placeid|placeid|placename|
+----+-----+-----+-----+-----+-------+-------+---------+
| Joe|    8|   12|    1|   29|      2|      2| Atlantic|
| Mel|   31|   34|   44|   27|      3|      3|Food Town|
| Joe|   13|   41|   12|   10|      4|      4|   Jake's|
|Jill|   23|    6|    3|   45|      3|      3|Food Town|
| Joe|   33|   16|   28|   28|      2|      2| Atlantic|
+----+-----+-----+-----+-----+-------+-------+---------+
only showing top 5 rows
```