Spark
PySpark and Spark SQL

Dr. Villanes

While we wait…

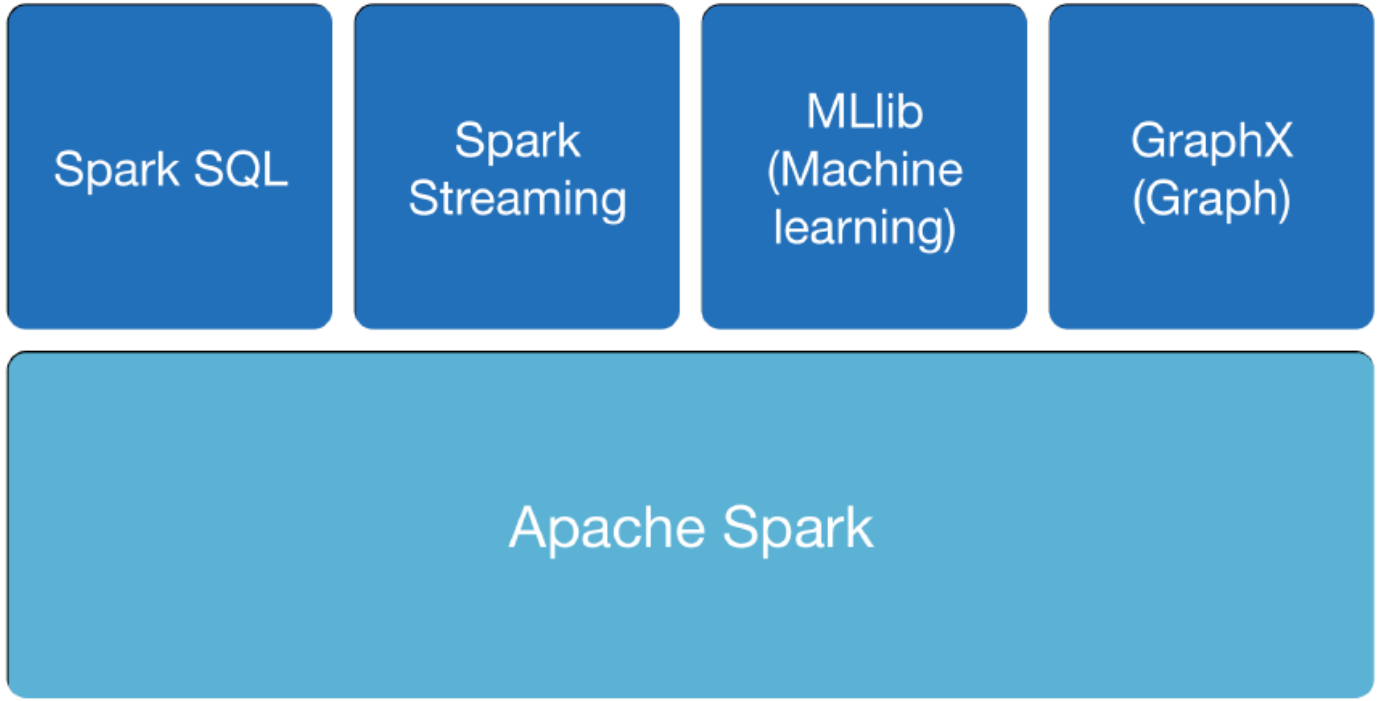
- Login to the AWS Student portal
- Download Data from Moodle

What are we going to do today?

1. Create an EMR Spark Cluster
2. Create a S3 bucket (make it PUBLIC)
3. RDDs vs Dataframes
4. PySpark
5. Spark SQL
6. Cheat sheet
7. PySpark Homework (due today at midnight)

Apache Spark Components

- Combine SQL, streaming, and complex analytics.



A Tale of Two Apache Spark Data Structures…

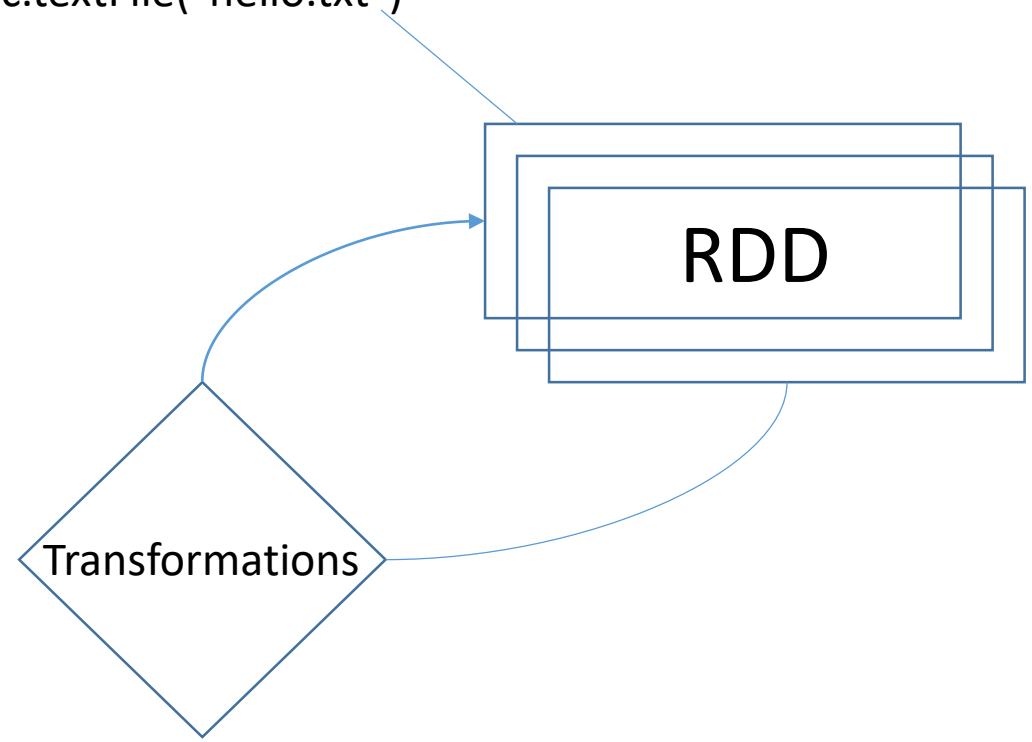
RDDs

DataFrames

- Spark revolved around the concept of a *resilient distributed dataset* (RDD)
- Representation of the data that is coming into your system, and allows to do computations on top of it.
- **Fault-tolerant collection of elements that can be operated on in parallel**.
- They are resilient because they allow lineage: whenever there is a failure in the system, they can go back, and re-compute themselves using all the prior information
- Two types of operations: transformations and actions

Resilient Distributed Datasets (RDDs)

1. txt=sc.textFile("hello.txt")



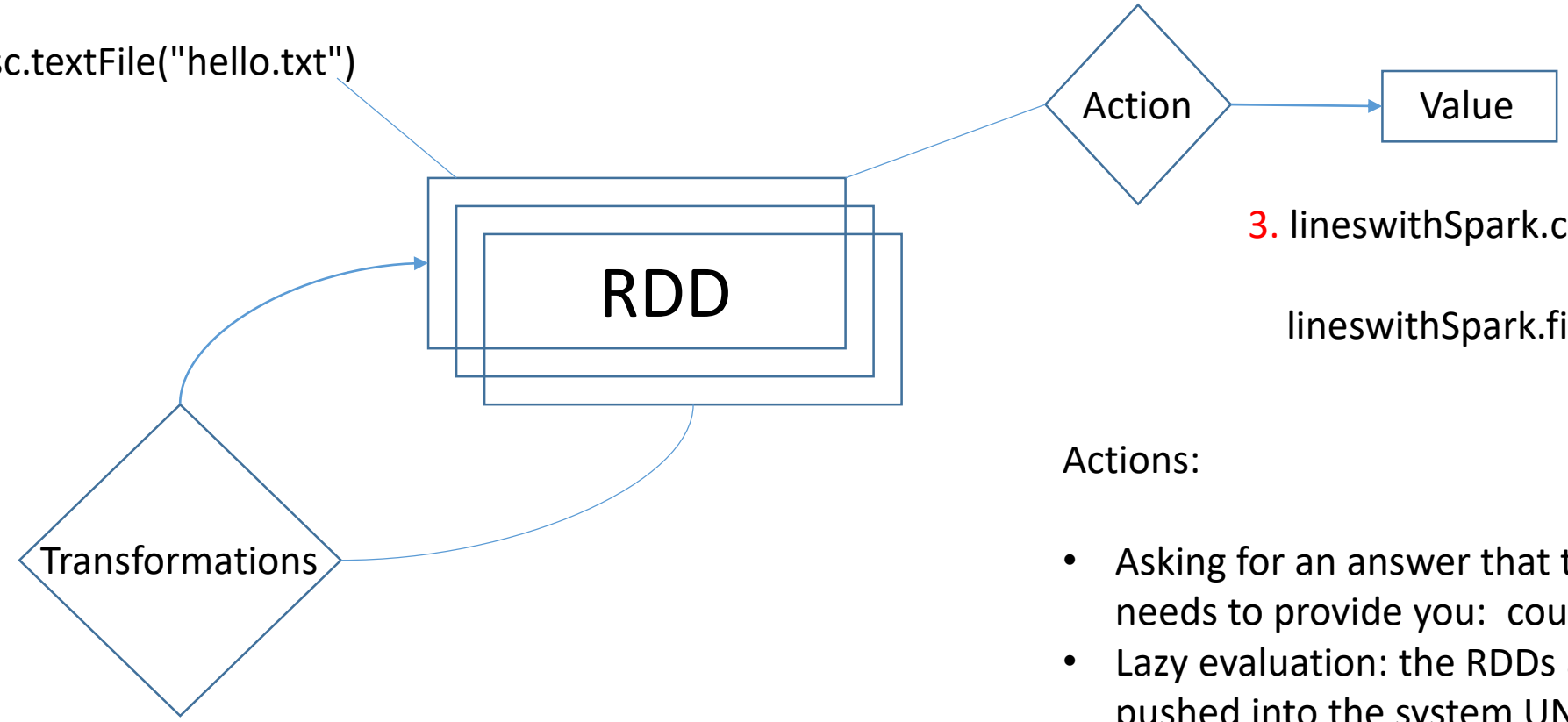
2. lineswithSpark = txt.filter(lambaline: "Spark")

Transformations:

- What you do to RDDs to get other resulting RDDs.
- Examples of transformations:
    1. opening a file;
    2. doing functions like filter that create new RDDs

Resilient Distributed Datasets (RDDs)

1. txt=sc.textFile("hello.txt")



3. lineswithSpark.count()

lineswithSpark.first()

2. lineswithSpark = textFile.filter(lambaline: "Spark")

Actions:

- Asking for an answer that the system needs to provide you: count or first()
- Lazy evaluation: the RDDs are not pushed into the system UNTIL an action is required.

A Tale of Two Apache Spark Data Structures…

RDDs

- Spark revolved around the concept of a *resilient distributed dataset* (RDD)
- Representation of the data that is coming into your system, and allows to do computations on top of it.
- **Fault-tolerant collection of elements that can be operated on in parallel**.
- They are resilient because they allow lineage: whenever there is a failure in the system, they can go back, and re-compute themselves using all the prior information
- Two types of operations: transformations and actions

DataFrames

- A DataFrame is a distributed collection of data
- Unlike an RDD, data is organized into named columns, like a table in a relational database.
- Designed to make large data sets processing even easier, DataFrame allows developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction.
- Makes Spark accessible to a wider audience, beyond specialized data engineers.
- Dataframes were released with Spark 2.0 (2016)

PySpark

Before we go into PySpark….

Apache Spark is an open-source cluster-computing framework, built around speed, ease of use, and streaming analytics

Ease of Use

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

Python + Spark = PySpark

- PySpark is the collaboration of Apache Spark and Python.
- **Apache Spark** is an open-source cluster-computing framework, built around speed, ease of use, and streaming analytics.
- **Python** is a general-purpose, high-level programming language.


```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Spark's Python DataFrame API

PySpark Core Concepts

Two important core concepts:

- SparkSession: Main entry point for DataFrame and SQL functionality.
- Dataframe: A distributed collection of data grouped into named columns.

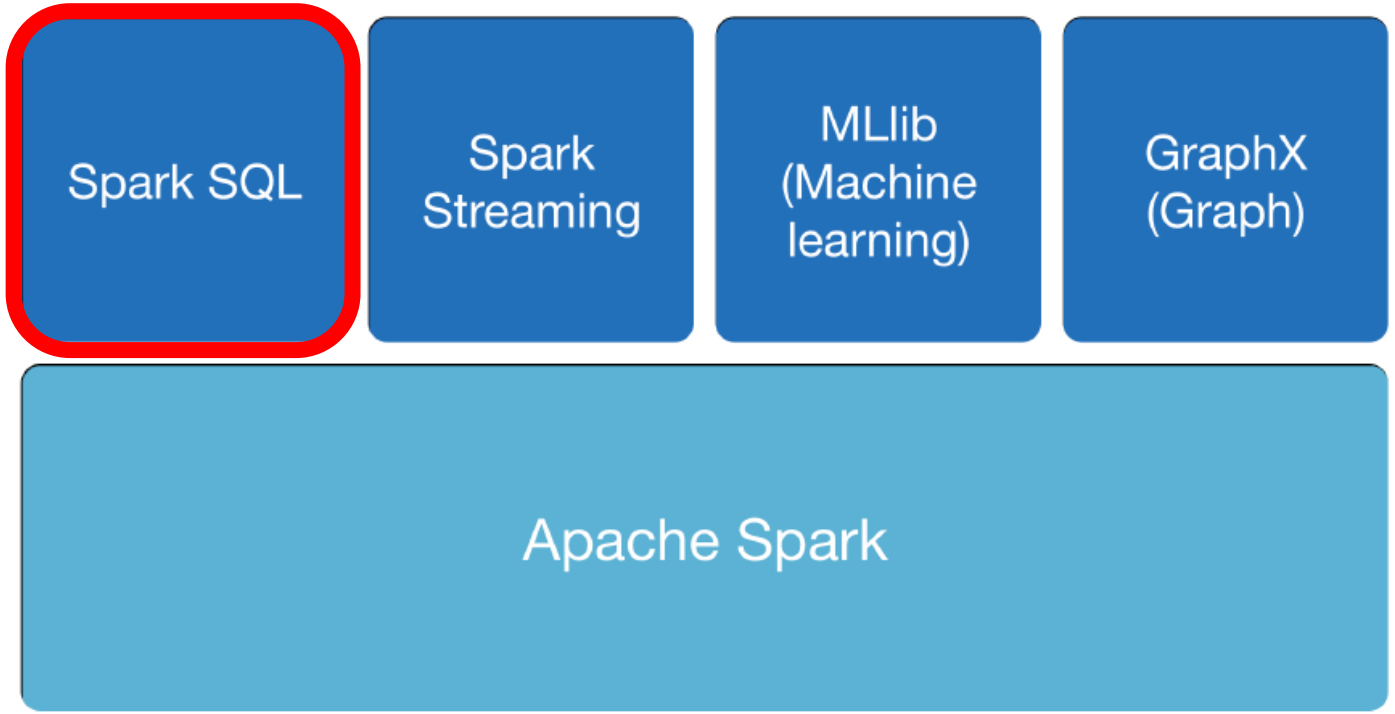
In EMR, SparkSession is automatically created for you. The SparkSession is accessible through a variable called *spark*.

Documentation: https://spark.apache.org/docs/latest/api/python/index.html

Spark SQL

Apache Spark Components

- Combine SQL, streaming, and complex analytics.



Apache Spark SQL

- Spark SQL is a Spark's module for working with structured data.
- There are several ways to interact with Spark SQL including SQL, the DataFrames API and the Datasets API. However, PySpark only implements the DataFrames API.
- A DataFrame is a distributed collection of data organized into named columns. It is **conceptually equivalent to a table in a relational database or a data frame in R/Python**, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs.
- Executes SQL queries written using either a basic SQL syntax or HiveQL.

PySpark & Spark SQL Cheat Sheet

Spark SQL is Apache Spark's module for working with structured data.

Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as views, execute SQL over views, and read csv, json, txt and parquet files.

In EMR, SparkSession is automatically created for you. The SparkSession is accessible through a variable called *spark*.
If you want to verify your Spark version:
>>> spark.version

Creating DataFrames

From Spark Data Sources

CSV
```
>>> df = spark.read.csv("s3a://bucket_name/airlines.csv", inferSchema = True, header=True)
```
JSON
```
>>> df2 = spark.read.json("s3a://bucket_name/customer.json")
>>> df2.show( )
>>> df3 = spark.read.load("s3a://bucket_name/people.json", format="json")
```
Parquet files
```
>>> df4 = spark.read.load("s3a://bucket_name/users.parquet")
```
TXT files
```
>>> df5 = spark.read.text("s3a://bucket_name/people.txt")
```

View the DataFrame

Show() - Displays the top 20 rows of DataFrame in a tabular form.
```
>>> df.show()
```
Show(n) - Displays the top *n* rows of DataFrame in a tabular form.
```
>>> df.show(n)
```

Inspect Data

>>> df.describe().show()	Compute summary statistics
>>> df.columns	Return the columns of df
>>> df.count()	Count the number of rows in df
>>> df.distinct().count()	Count the number of distinct rows in df
>>> df.printSchema()	Print the schema of df

Queries

```
>>> from pyspark.sql import functions as F
```
Select
```
>>>  df.select ("firstName") .show( )
>>> df.select("firstName","lastName")  \
       .show( )
>>> df.select (df ["firstName"] ,df ["age"]+ 1)
>>> df.select(df['age'] > 24).show( )
```
Show all entries in *firstName* column
Show all entries in *firstName*, and *lastName*
Show all entries in *firstName* and *age*, add *1* to the entries of *age*
Show all entries where *age >24*

When
```
>>> df.select ("firstName",
                F.when(df.age > 30, 1) \
               .otherwise(0)) \
       .show( )
>>> df[df.firstName.isin ("Jane","Boris")]
                .show( )
```
Show *firstName* and 0 or 1 depending on age > 30

Show *firstName* if in the given options

Like
```
 >>> df.select("firstName",
               df.lastName.like ("Smith")) \
        .show( )
```
Show *firstName*, and *lastName* if *lastName* is like Smith

Startswith – Endswith
```
>>> df.select("firstName",
              df.lastName \
               .startswith("Sm"))  \
       .show( )
>>> df.select(df.lastName.endswith("th"))  \
       .show( )
```
Show *firstName*, and *lastName* if *lastName* starts with *Sm*

Show last names ending in *th*

Substring
```
>>> df.select(df.firstName.substr(1,  3)  \
                          .alias("name"))  \
       .collect( )
```
Return substrings of *firstName*

Between
```
>>> df.select (df.age.between(22, 24)) \
       .show()
```
Show *age* if values between 22 and 24

Adding Columns

```
>>> from pyspark.sql.functions import log

>>> df2 = df.withColumn("new_column", log("rating"))
>>> df2.show()
```

Duplicate Values

```
>>> df = df.dropDuplicates( )
```

GroupBy

```
>>> df.groupBy("age")  \
       .count( )   \
       .show( )
```
Group by age, count the members in the groups

Filter

```
>>> df.filter(df["age"]>24).show( )
```
Filter entries of age, only keep those records of which the values are > 24

Missing & Replacing Values

```
>>> df.na.fill(50).show( )
>>> df.na.drop().show()
>>> df.na \
       .replace(10, 20) \
       .show()
```
Replace null values
Return new df omitting rows with null values
Return new df replacing one value with another

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> sqlDF = spark.sql("SELECT * FROM customer").show()
```

Output

Data Structure

```
>>> rdd1 = df.rdd
>>> df.toPandas()
```
Convert *df* into an RDD
Return the contents of *df* as Pandas *DataFrame*

Write & Save to Files

```
>>> df.select("firstName", "city") .write.save("nameAndCity.parquet")
>>> df.select("firstName", "age") .write  .save("namesAndAges.json",format="json")
```

One piece of advice…

- Use the Spark documentation directly as much as possible: https://spark.apache.org/docs/latest/index.html
- If you consult blogs or StackOverFlow, check the date: 2016 might be too old for example…

Update on Datacamp…

- New space is coming. You will get an invitation once the new space is approved.