



2018
Global Azure
BOOTCAMP

TEKLINKS

TSYS®



Sponsors for Azure Bootcamp

Predictive Analytics with Spark in Azure HDInsight

Ingyu Lee

Troy University

<https://github.com/Azure-Bootcamp-Troy/SparksInHDInsight>

Content

- What is HDInsight?
 - What is Spark?
 - What is RDD?
 - What is DataFrames?
 - What is Spark SQL?
 - What is MLLib?
- References
 - **Microsoft DAT202.3x**: Spark in Azure HDInsight
 - Microsoft DAT202.2x: Real-Time Analytics with Hadoop in Azure HDInsight
 - Microsoft DAT202.1x: Big Data with Hadoop in Azure HDInsight
 - Microsoft Virtual Academy: Spark on HDInsight
 - UCB Sparks: CS100, CS105, CS110, CS120, CS190
 - UCSD Big Data Series (Coursera)
 - Yandex: Big Data Series (Coursera)

HDInsight – What is it?

A standard Apache Hadoop distribution offered as a managed service on Microsoft Azure

- ❖ Based on Hortonworks Data Platform (HDP)
- ❖ Provisioned as clusters on Azure that can run on Windows or Linux servers
- ❖ Offers capacity-on-demand, pay-as-you-go pricing model
- ❖ Integrates with:
 - ❖ Azure Blob Storage and Azure Data Lake Store for Hadoop File System (HDFS)
 - ❖ Azure Portal for management and administration
 - ❖ Visual Studio for application development tooling

In addition to the core, HDInsight supports the Hadoop ecosystem



What is Apache *Spark*?

General, open-source cluster computing engine

Well-suited for machine learning

- Fast iterative procedures
- Efficient communication primitives

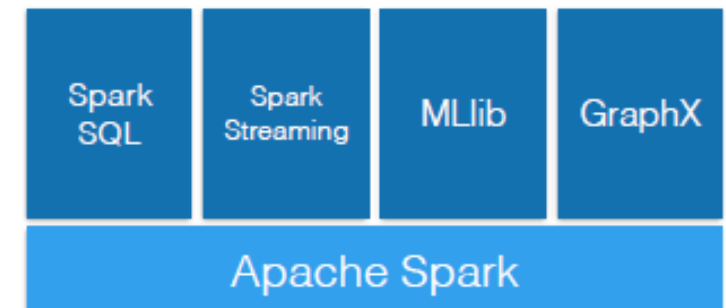
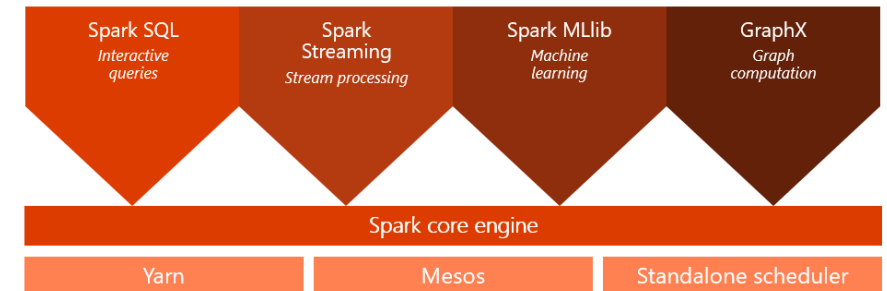
Simple and Expressive

- APIs in Scala, Java, Python, R
- Interactive Shell

Integrated Higher-Level Libraries

Apache Spark: A unified framework

A unified, open source, parallel data processing framework for big data analytics



HDInsight supports Spark

In-memory processing on multiple workloads

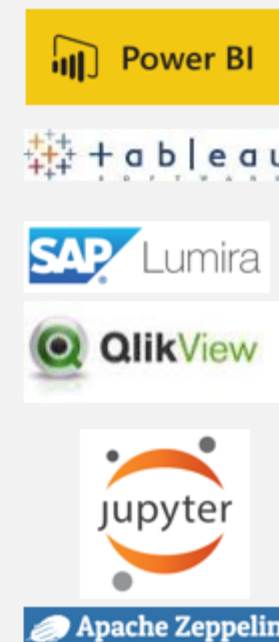
Single execution model for multiple tasks (SQL Query, Spark Streaming, Machine Learning, and Graph)

Processing up to 100x faster performance

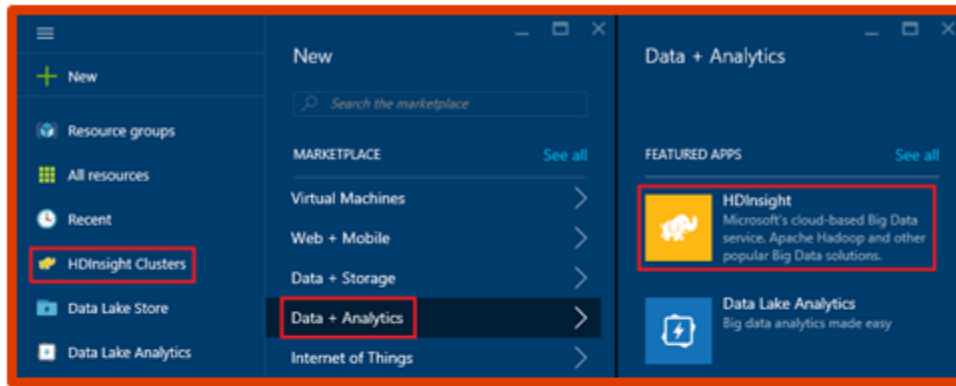
Developer friendly (Java, Python, Scala)

BI tool of choice (Power BI, Tableau, Qlik, SAP)

Notebook experience (Jupyter/iPython, Zeppelin)



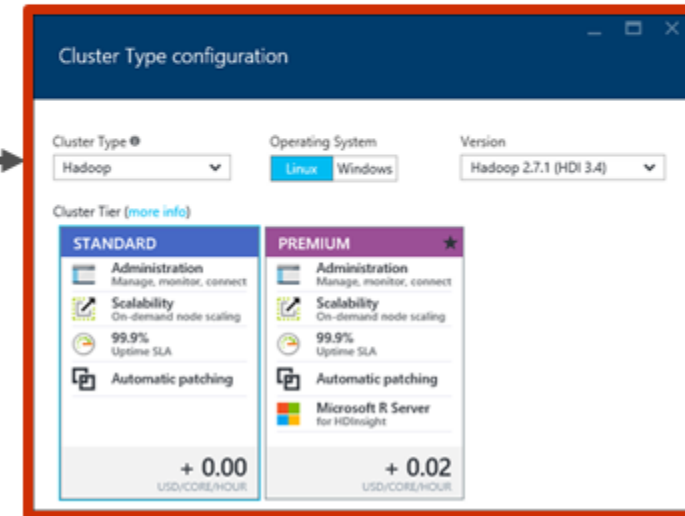
Demo: Creating an HDInsight Spark cluster



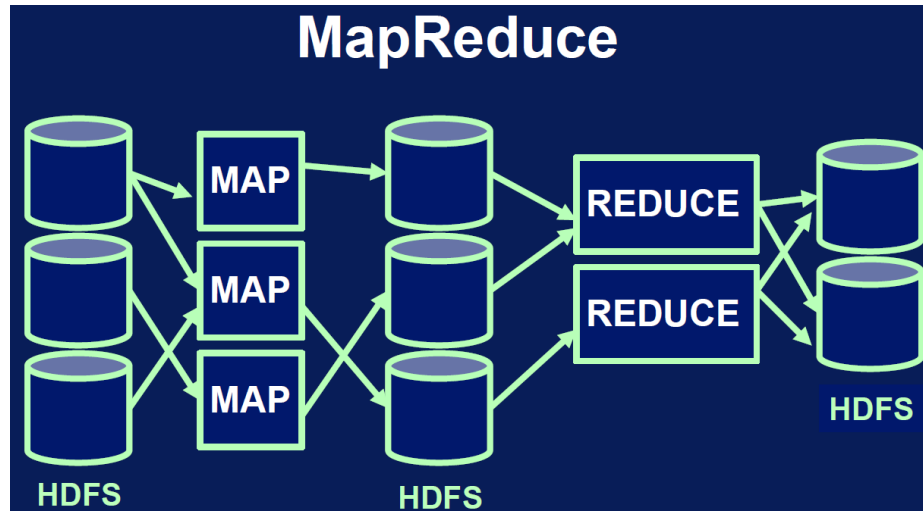
Apache Spark use cases



The Azure console lists all types of HDInsight clusters (HBase, Storm, and Spark) currently provisioned.



Hadoop vs. Spark



Force your pipeline into Map and Reduce steps

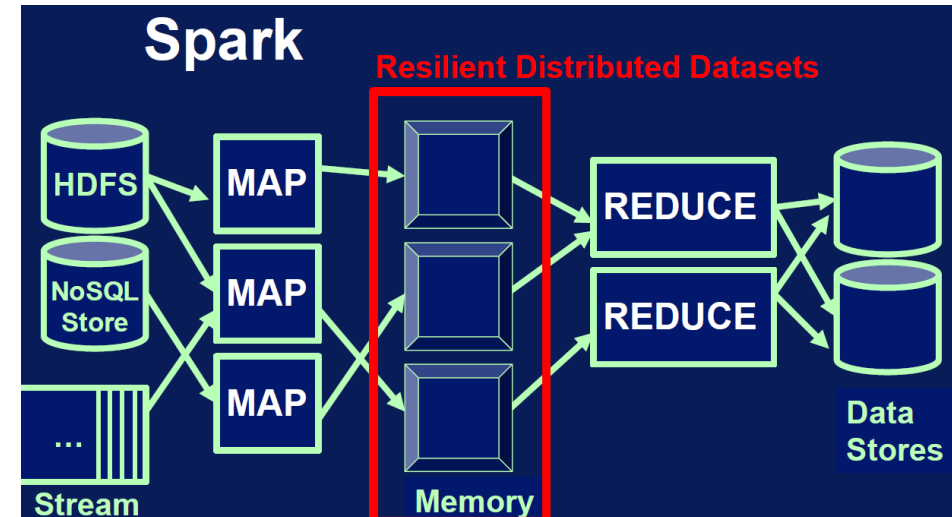
Other workflows? i.e. join, filter, map-reduce-map

Read from disk for each MapReduce job

Iterative algorithms? i.e. machine learning

Only native JAVA programming interface

Other languages?
Interactivity?



- New framework: same features of MapReduce and more
- Capable of reusing Hadoop ecosystem, e.g. HDFS, YARN...
- Born at UC Berkeley

Other workflows? i.e. join, filter, map-reduce-map

~20 highly efficient distributed operations, any combination of them

Interactivity? Other languages?

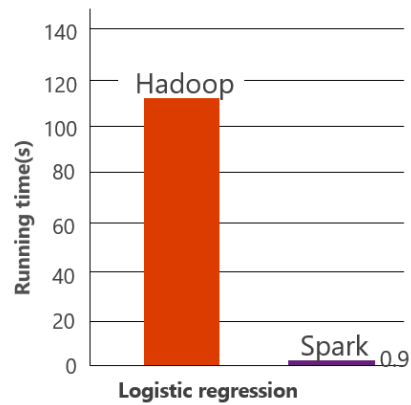
Native Python, Scala (, R) interface. Interactive shells.

Iterative algorithms? i.e. machine learning

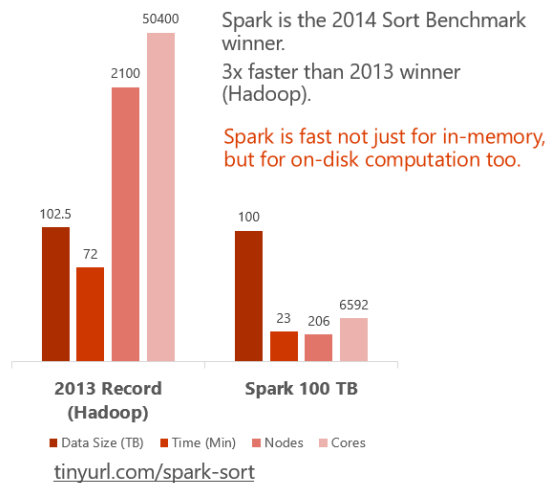
in-memory caching of data, specified by the user

Why spark is faster?

Faster data, faster results

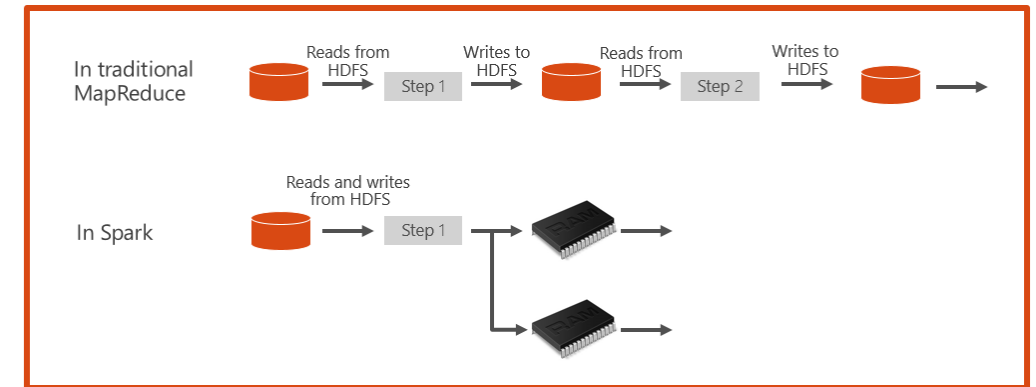


Logistic regression on a 100-node cluster with 100 GB of data.

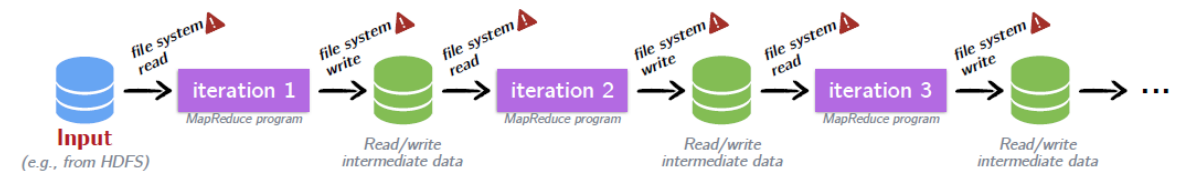


What makes Spark fast?

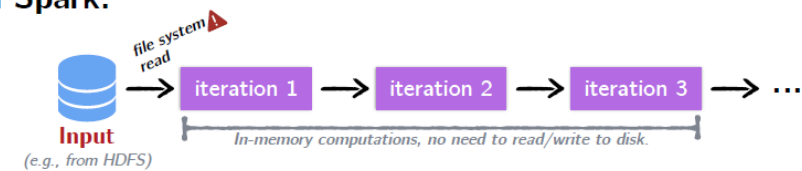
Data sharing between steps of a job



Iteration in Hadoop:

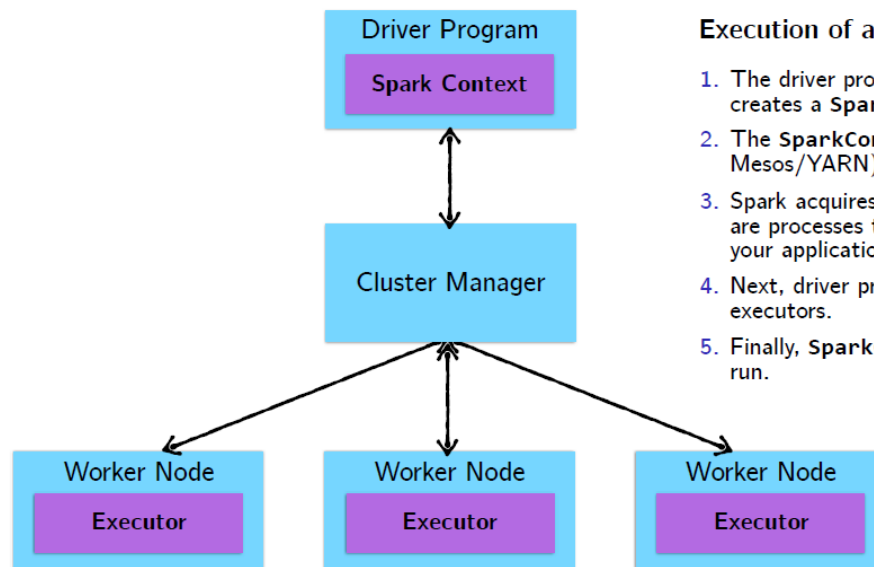


Iteration in Spark:



Spark cluster architecture

- Distributed processing architecture consists of:
 - A *driver program*
 - One or more *worker nodes*
- The driver program uses a spark context to connect to the cluster...
- ...and uses worker nodes to perform operations on RDDs



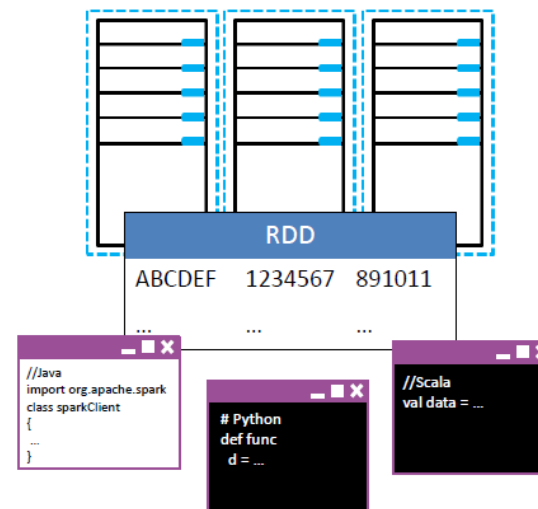
Execution of a Spark program:

1. The driver program runs the Spark application, which creates a **SparkContext** upon start-up.
2. The **SparkContext** connects to a cluster manager (e.g., Mesos/YARN) which allocates resources.
3. Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application.
4. Next, driver program sends your application code to the executors.
5. Finally, **SparkContext** sends tasks for the executors to run.

What is RDD?

- The core abstraction for data in Spark is the *resilient distributed dataset* (RDD)
- An RDD represents a collection of items that can be distributed across compute nodes
- APIs for working with RDDs are provided for Java, Python, and Scala
 - HDInsight distribution includes Python and Scala shells

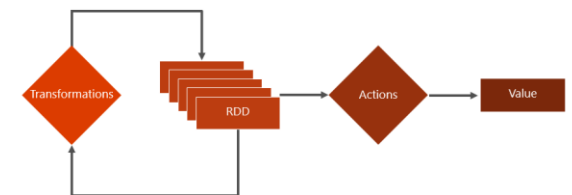
Dataset	Distributed	Resilient
Data storage created from: HDFS, S3, HBase, JSON, text, Local hierarchy of folders	Distributed across the cluster of machines	Recover from errors, e.g. node failure, slow processes
Or created transforming another RDD	Divided in partitions, atomic chunks of data	Track history of each partition, re-run



Programming in Spark

Create RDDs
↓
Apply transformations
↓
Perform actions

RDDs: Transformations and actions



RDD Operations

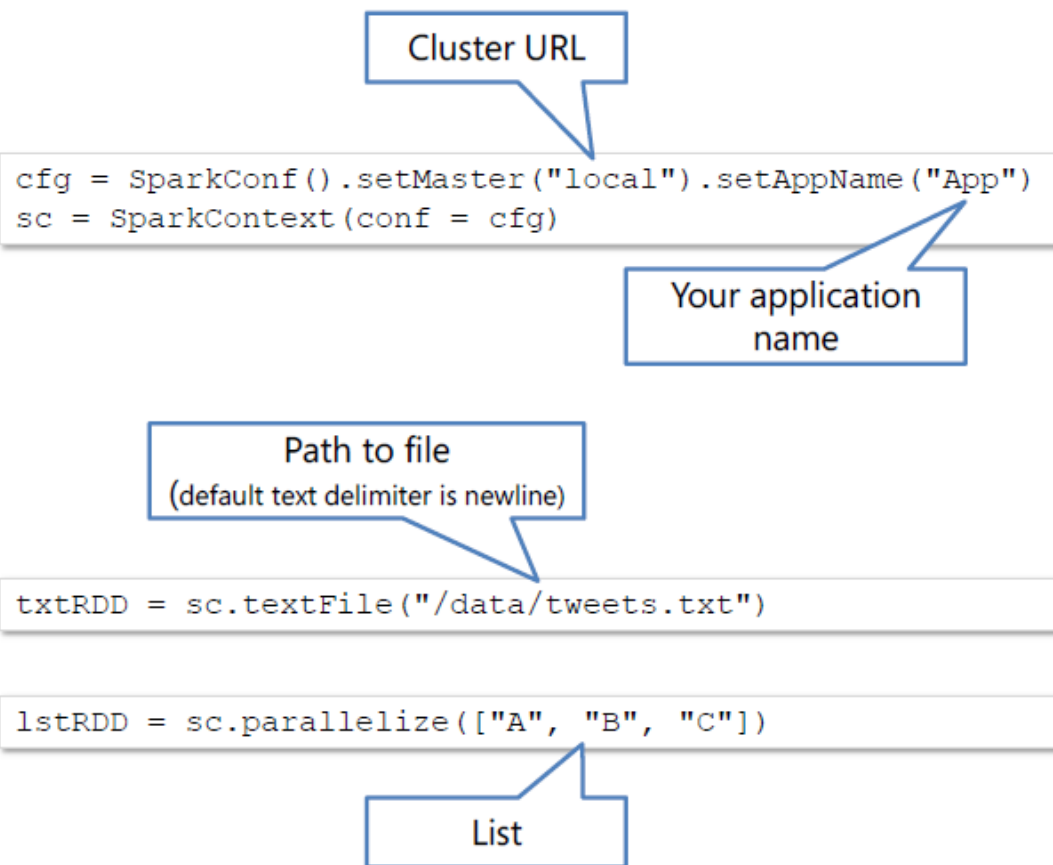
- To create a Spark Context:

1. Create a configuration for your cluster and application
2. Use the configuration to create a context

(Spark shells have one pre-created)

- To create an RDD

- Load from a source
 - Text file, JSON, XML, etc.
- Parallelize a collection



RDD Transformations

Common Transformations:

- **filter**: Creates a filtered RDD
- **flatMap**: Applies a function to each element that returns multiple elements into a new RDD
- **map**: Applies a function to each element that returns an element in a new RDD
- **reduceByKey**: Aggregates values for each key in a key-value pair RDD

```
txt = sc.parallelize(["the owl and the pussycat",  
                    "went to sea"])
```

```
{["the owl and the pussycat"], ["went to sea"]}
```

```
owlTxt = txt.filter(lambda t: "owl" in t)
```

```
{["the owl and the pussycat"]}
```

```
words = owlTxt.flatMap(lambda t: t.split(" "))
```

```
{["the"], ["owl"], ["and"], ["the"], ["pussycat"]}
```

```
kv = words.map(lambda key: (key, 1))
```

```
{["the",1], ["owl",1], ["and",1], ["the",1], ["pussycat",1]}
```

```
counts = kv.reduceByKey(lambda a, b: a + b)
```

```
{["the",2], ["owl",1], ["and",1], ["pussycat",1]}
```

RDD Actions

Common Actions:

- **reduce**: Aggregates the elements of an RDD using a function that takes two arguments
- **count**: Returns the number of elements in the RDD
- **first**: Returns the first element in the RDD
- **collect**: Returns the RDD as an array to the driver program
- **saveAsTextFile**: Saves the RDD as a text file in the specified path

```
nums = sc.parallelize([1, 2, 3, 4])
```

{[1], [2], [3], [4]}

```
nums.reduce(lambda x, y: x + y)
```

10

```
nums.count()
```

4

```
nums.first()
```

1

```
nums.collect()
```

[1, 2, 3, 4]

```
nums.saveAsTextFile("/results")
```

/results/part-00000

Demo: Working with RDDs in Python

- Most operations involve passing a function to a transformation or action
- Functions can be:
 - Explicitly declared
 - Passed inline
 - Python uses `lambda` keyword
 - Scala uses `=>` syntax
 - Java uses function classes or lambdas (Java 8)

```
RDD.filter(function)
```

```
def containsMSTag(txt):  
    return "#ms" in txt  
  
msTwts = txtRDD.filter(containsMSTag)
```

```
#Python  
msTwts = txtRDD.filter(lambda txt: "#ms" in txt)
```

```
//Scala  
val msTwts = txtRDD.filter(txt => txt.contains("#ms"))
```

Ambari Dashboard

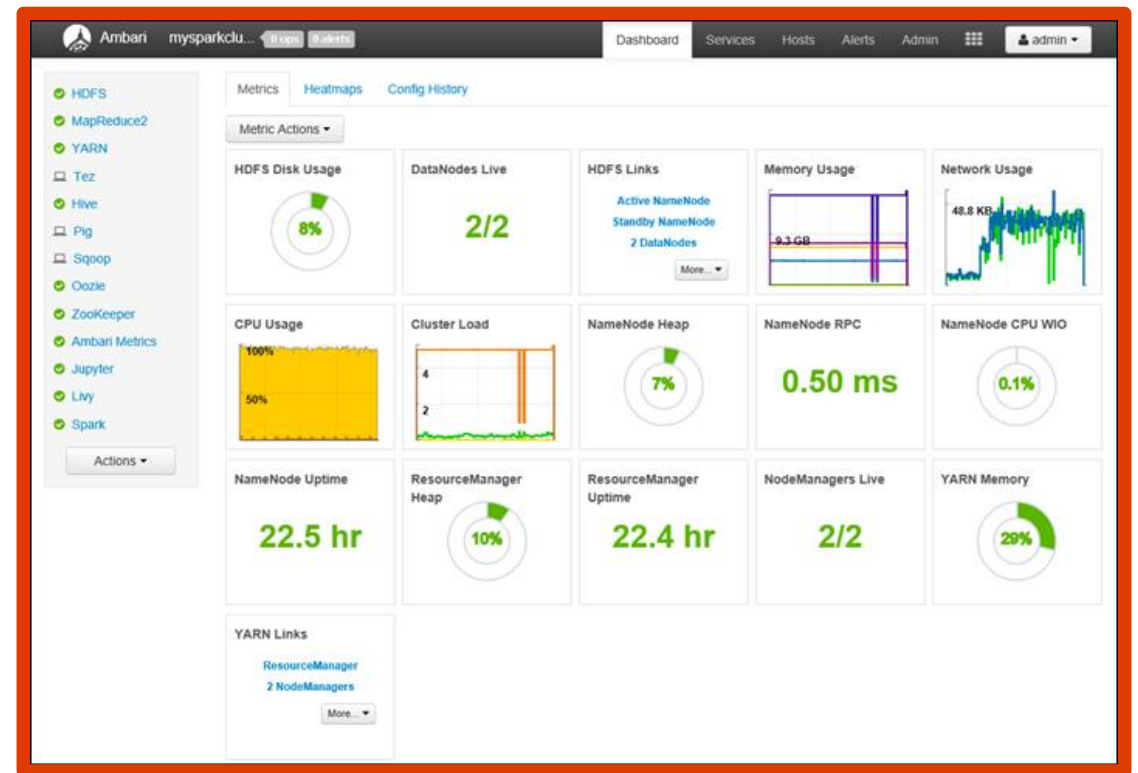


The Ambari Dashboard provides links to access:

Ambari web UI – monitors Spark clusters

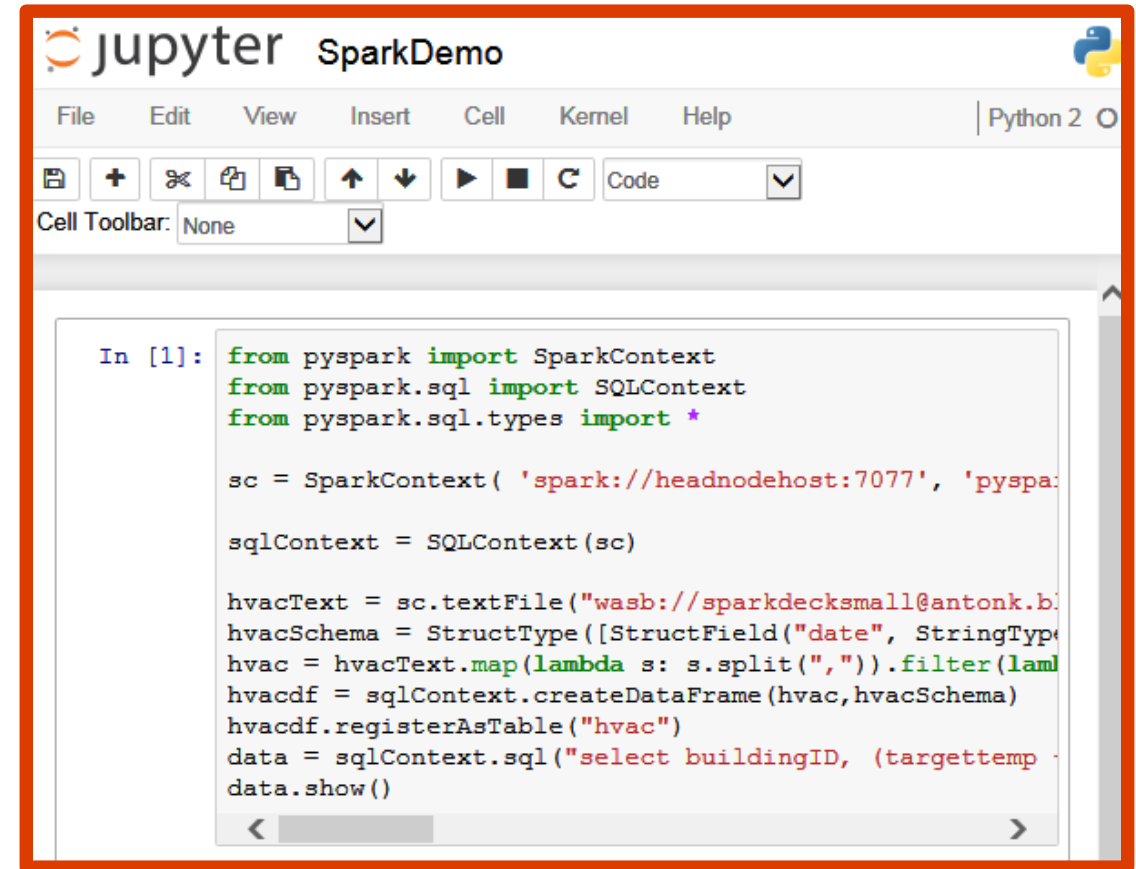
Hosts (Resource Manager) – controls amount of resources allocated to various Spark cluster components

Notebooks – interactive web-based tools to develop and run Spark programs



HDInsight Spark: Jupyter notebooks

- Web-based interactive consoles for
 - Experimentation
 - Collaboration
- Spark HDInsight clusters include Jupyter
 - Interactive Python
 - Interactive Scala



The screenshot shows a Jupyter notebook window titled "jupyter SparkDemo". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". Below the menu is a toolbar with icons for saving, adding, deleting, and running cells. The "Cell Toolbar" is set to "None". The main area displays a code cell with the following Python code:

```
In [1]: from pyspark import SparkContext
        from pyspark.sql import SQLContext
        from pyspark.sql.types import *

        sc = SparkContext( 'spark://headnodehost:7077', 'pyspa:
        sqlContext = SQLContext(sc)

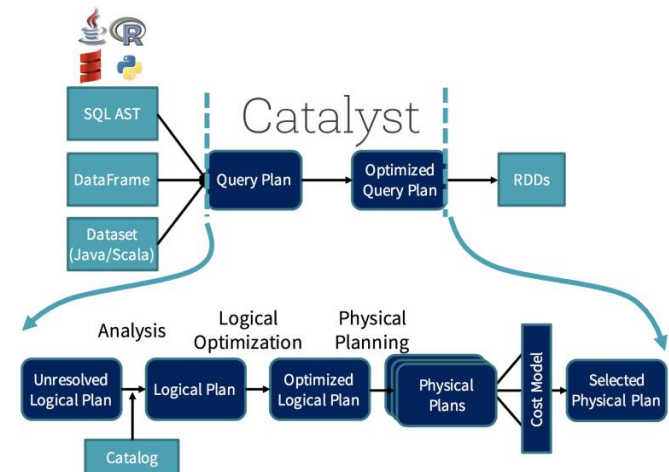
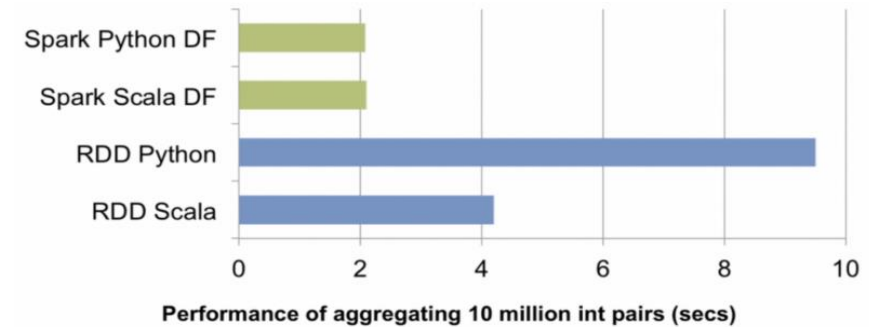
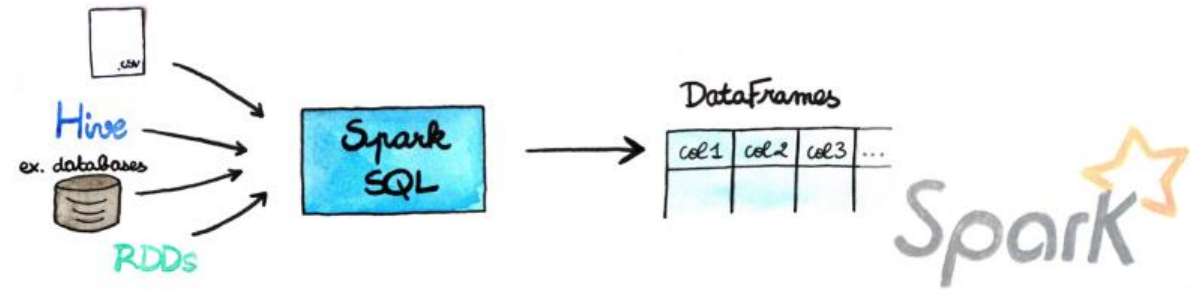
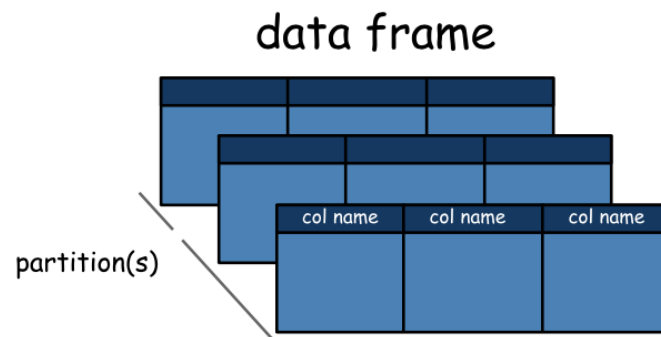
        hvacText = sc.textFile("wasb://sparkdecksmall@antonk.b
        hvacSchema = StructType([StructField("date", StringType
        hvac = hvacText.map(lambda s: s.split(",")).filter(lam
        hvacdf = sqlContext.createDataFrame(hvac,hvacSchema)
        hvacdf.registerAsTable("hvac")
        data = sqlContext.sql("select buildingID, (targettemp
        data.show()
```

What is DataFrames?

DataFrames

Distributed Data organized as named columns

Look just like a table in relational databases



Core abstraction: DataFrames

- A distributed collection of data organized into named columns.
- Similar to RDDs with schema.
- Conceptually equivalent to tables in relational database, or to DataFrames in R/Python.
- With domain-specific functions designed for common tasks:
 - Metadata
 - Sampling
 - Project, filter, aggregation, and join
 - UDFs

RDDs are a collection of opaque objects (such as internal structures unknown to Spark).

User

User

User

User

User

User

DataFrames is a collection of objects with schema that are known to Spark SQL..

Name

Age

Sex

Name

Age

Sex

Name

Age

Sex

Name

Age

Sex

DataFrame Operations

DataFrames provide a domain-specific language for structured data manipulation in Scala, Java, and Python.

```
>>> val df = sqlContext.jsonFile("somejsonfile.json")

>>> df.show()                                // Show the contents of the DataFrame

>>> df.printSchema()                        // Print the schema in a tree format

>>> df.select("name").show()                 // Select and show the name columns

>>> df.select(df("name"),df ("age") +1).show() // Select all but increment the age by 1

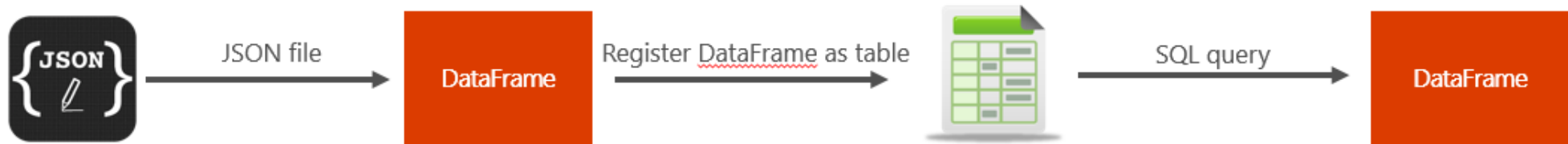
>>> df.filter(df("age") > 21).show()         // Select people older than 21

>>> df.groupBy("age").count().show()         // Count people by age
```

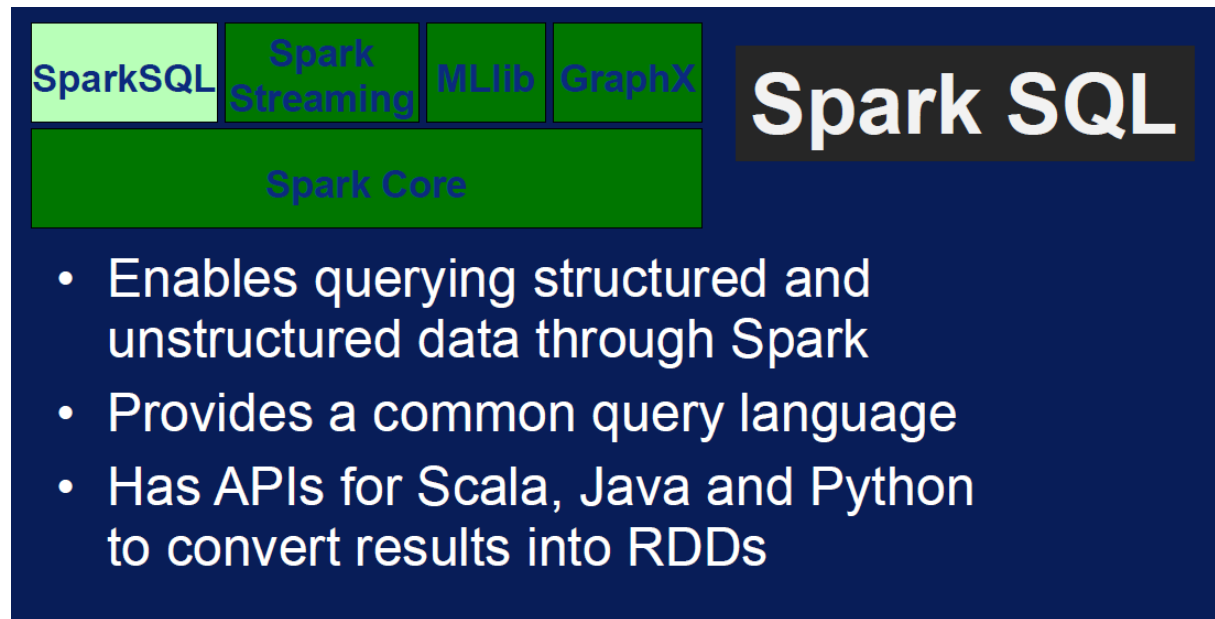
Tables and Queries

A DataFrame can be registered as a table that can then be used in SQL queries.

```
// First create a DataFrame from JSON file.  
>>> val df = sqlContext.jsonFile("Users.json")  
  
// Register the DataFrame as a temporary table. Temp tables exist only during lifetime of this SQLContext  
instance.  
>>> val usertable = sqlContext.registerDataFrameAsTable(df, "UserTable")  
  
// Alternatively, execute a SQL query on the table. The query returns a DataFrame.  
>>> val teenagers = sqlContext.sql("select Age as Years from UserTable where age > 13 and age <= 19")
```



What is Spark SQL?



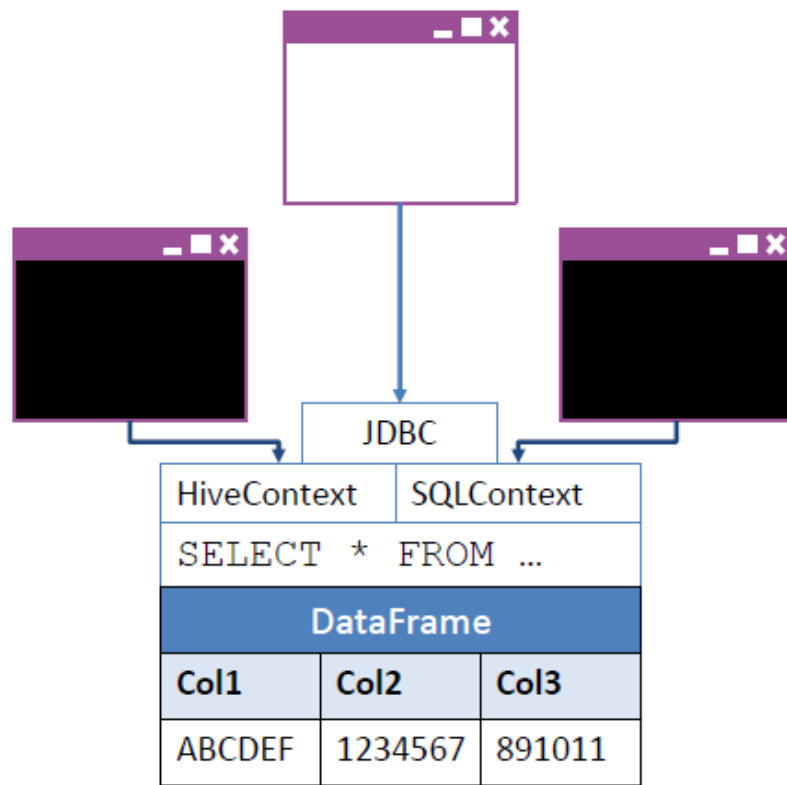
Relational Operations

Perform Relational Processing
such as Declarative Queries

Embed SQL queries
inside Spark
Programs

Spark SQL

- Spark SQL provides a query interface for structured data
- DataFrames are used to abstract RDDs and define a schema
- There are two API entry points:
 - HiveContext
 - SQLContext
- Client applications can connect to Spark SQL using JDBC



Spark SQL

How to go Relational in Spark ?

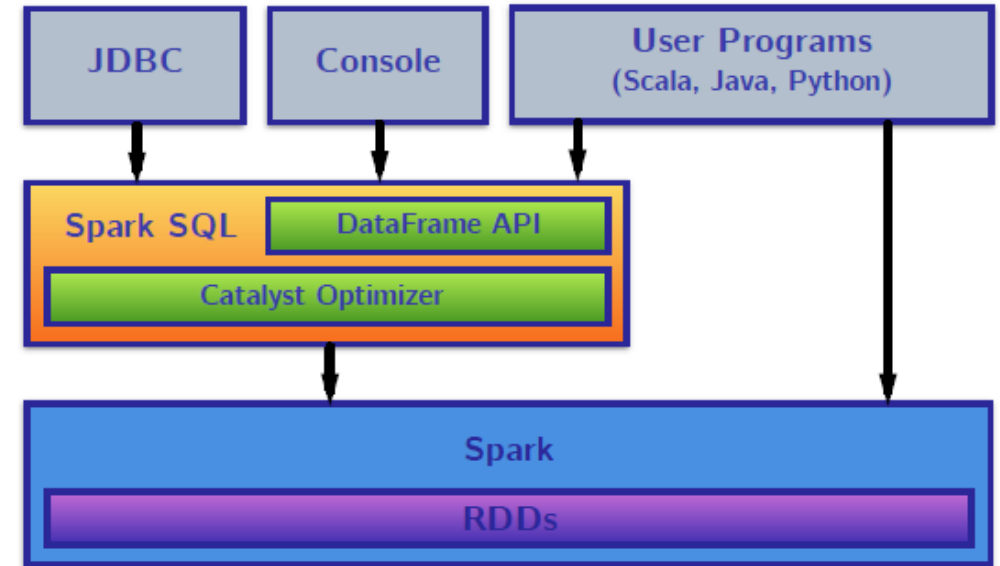
Step 1: Create a SQLContext

```
from pyspark.sql import SQLContext  
sqlContext = SQLContext(sc)
```

How to go Relational in Spark ?

Create a DataFrame from

- an existing RDD
- a Hive table
- data sources



Spark RDDs:



Not much structure.
Difficult to aggressively optimize.

DataFrames/Databases/Hive:

name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean
name: String	balance: Double	risk: Boolean

SELECT
WHERE
ORDER BY
GROUP BY
COUNT

Lots of structure.
Lots of optimization opportunities!

Spark SQL

JSON → DataFrame

```
# Read
df = sqlContext.read.json("/filename.json")

# Display
df.show()
```

DataFrames are just like tables

<http://spark.apache.org/>

```
# Show the content of the DataFrame
df.show()

# Print the schema
df.printSchema()

# Select only the "X" column
df.select("X").show()

# Select everybody, but increment the discount by 5%
df.select(df["name"], df["discount"] + 5).show()

# Select people height greater than 4.0 ft
df.filter(df["height"] > 4.0).show()

# Count people by zip
df.groupBy("zip").count().show()
```

RDD of Row objects → DataFrame

```
# Read
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

# Load a text file and convert each line to a Row.
lines = sc.textFile("filename.txt")
cols = lines.map(lambda l: l.split(","))
data = cols.map(lambda p: Row(name=p[0], zip=int(p[1])))

# Create DataFrame
df = sqlContext.createDataFrame(data)

# Register the DataFrame as a table
df.registerTempTable("table")

# Run SQL
Output = sqlContext.sql("SELECT * FROM table WHERE ...")
```

Spark SQL

Relational on Spark

Connect to variety of databases

Deploy business intelligence tools over Spark

Demo: Using Spark SQL DataFrames

What is Machine Learning?

A Definition

Constructing and studying methods that learn from and make predictions on data

Broad area involving tools and ideas from various domains

- Computer Science
- Probability and Statistics
- Optimization
- Linear Algebra

Some Examples

Face recognition



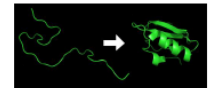
Link prediction



Text or document classification, e.g., spam detection



Protein structure prediction

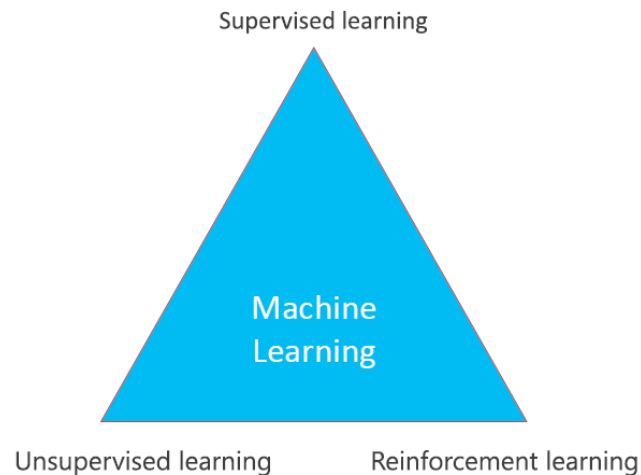


Games, e.g., Backgammon or Jeopardy



How does Machine Learning work?

- Data labels
- Direct feedback
- Predict outcome

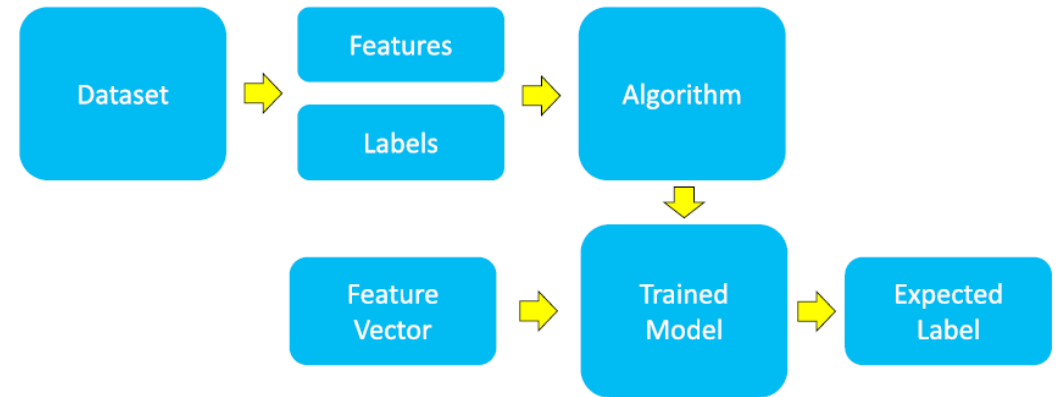


Supervised learning. Learning from labeled observations

- Labels 'teach' algorithm to learn mapping from observations to labels

Unsupervised learning. Learning from unlabeled observations

- Learning algorithm must find latent structure from features alone
- Can be goal in itself (discover hidden patterns, exploratory data analysis)
- Can be means to an end (preprocessing for supervised task)



Observations. Items or entities used for learning or evaluation, e.g., emails

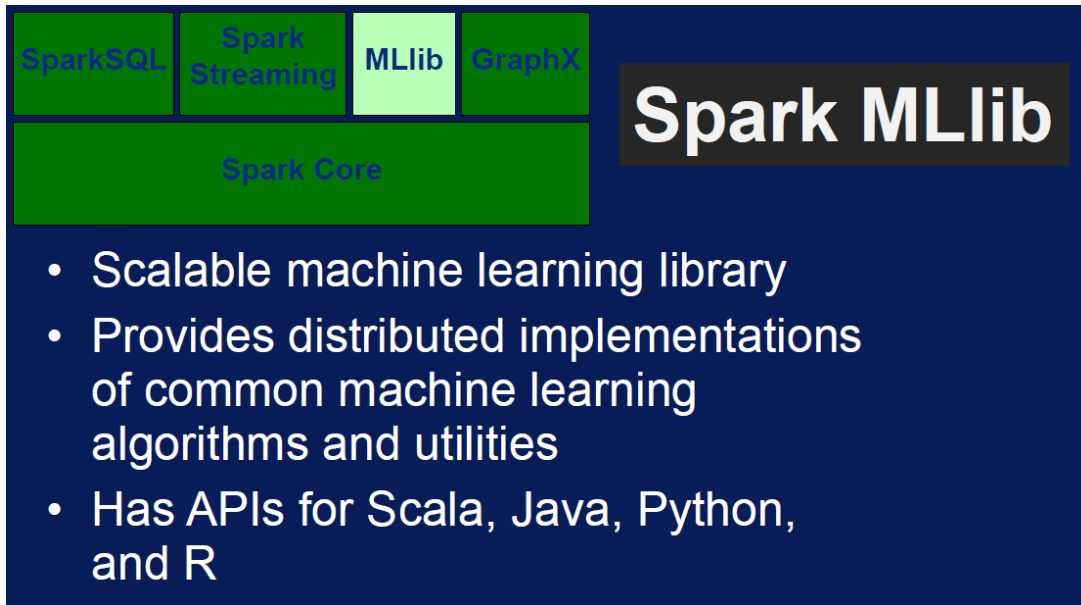
Features. Attributes (typically numeric) used to represent an observation, e.g., length, date, presence of keywords

Labels. Values / categories assigned to observations, e.g., *spam*, *not-spam*

Training and Test Data. Observations used to train and evaluate a learning algorithm, e.g., a set of emails along with their labels

- Training data is given to the algorithm for training
- Test data is withheld at train time

What is MLlib?



MLlib Algorithms & Techniques

- Machine Learning
 - Classification, regression, clustering, etc.
 - Evaluation metrics
- Statistics
 - Summary statistics, sampling, etc.
- Utilities
 - Dimensionality reduction, transformation, etc.

How to.. Machine Learning in Apache Spark?

- All primitives in Spark Machine Learning are *Vectors*
- *Features* are represented by a Vector
- Vectors can contain other Vectors and so be Dense or Sparse
- Spark uses *LabeledPoints* to encapsulate a Vector and a Label
- RDDs are transformed into Vectors through map functions

MLlib Terminology

ML: Transformer

- A *Transformer* is a class which can transform one DataFrame into another DataFrame
- A Transformer implements **transform()**
- Examples
 - HashingTF
 - LogisticRegressionModel
 - Binarizer

```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
val hashingTF = new
HashingTF().setNumFeatures(1000).setInputCol(tokenizer.getOutputCol).s
etOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.01)
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))
```

```
val model = pipeline.fit(training)
```

```
val modelelem = model.transform(test).select("id", "label", "text",
"probability", "prediction")
```

ML: Estimator

- An *Estimator* is a class which can take a DataFrame and produce a Transformer
- An Estimator implements **fit()**
- Examples
 - LogisticRegression
 - StandardScaler
 - Pipeline

```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
val hashingTF = new
HashingTF().setNumFeatures(1000).setInputCol(tokenizer.getOutputCol).s
etOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.01)
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))
```

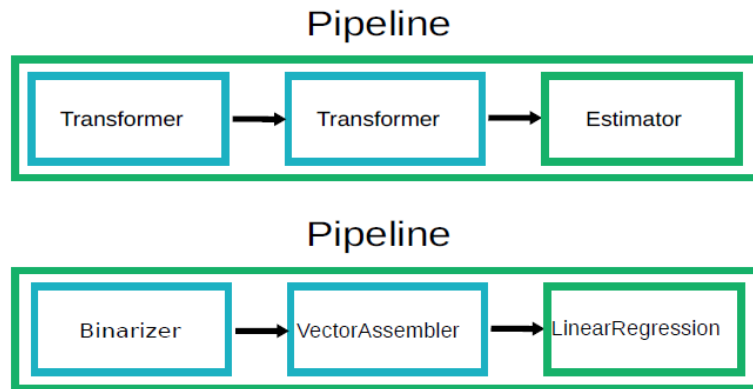
```
val model = pipeline.fit(training)
```

```
val modelelem = model.transform(test).select("id", "label", "text",
"probability", "prediction")
```

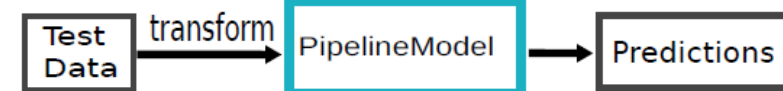
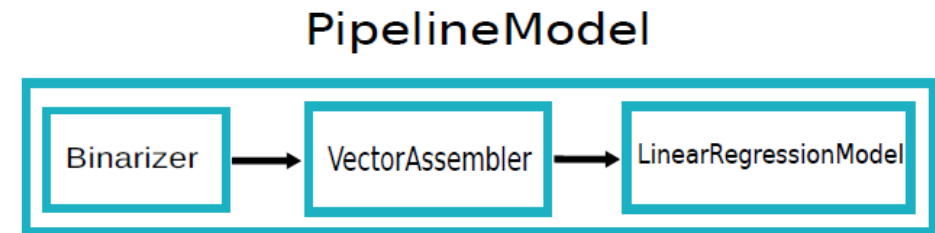
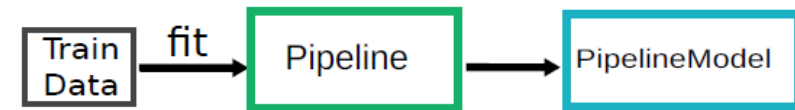
MLlib Terminology

ML: Pipelines

A *Pipeline* is an estimator that contains stages representing a reusable workflow. Pipeline stages can be either estimators or transformers.



ML: PipelineModel



MLlib Examples

MLlib Example – Summary Statistics

- Compute column summary statistics

```
from pyspark.mllib.stat import Statistics
```

1

Data as RDD of Vectors

```
dataMatrix = sc.parallelize([ [1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12] ])
```

2

Compute column summary statistics.

```
summary = Statistics.colStats(dataMatrix)
```

3

```
print(summary.mean())
```

```
print(summary.variance())
```

4

```
print(summary.numNonzeros())
```

MLlib Example – Classification

- Build decision tree model for classification

```
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils
```

1

Read and parse data

```
data = sc.textFile("data.txt")
```

2

3

Decision tree for classification

```
model = DecisionTree.trainClassifier(
    parsedData, numClasses=2)
```

4

```
print(model.toDebugString())
```

5

```
model.save(sc, "decisionTreeModel")
```

6

MLlib Examples

MLlib Example – Clustering

- Build k-means model for clustering

```
from pyspark.mllib.clustering import KMeans, KMeansModel  
from numpy import array
```

Read and parse data

```
data = sc.textFile("data.txt")  
parsedData = data.map(lambda line:  
    array([float(x) for x in line.split(' ')]))
```

k-means model for clustering

```
clusters = Kmeans.train (parsedData, k=3)
```

```
print(clusters.centers)
```

1

2

3

4

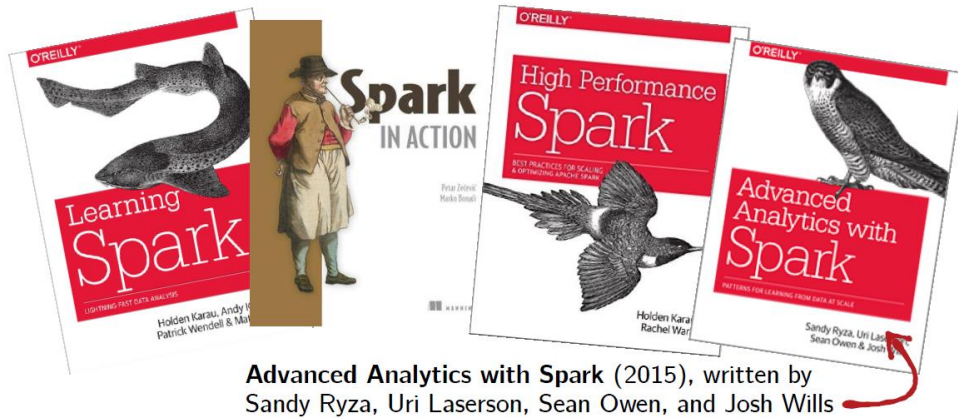
5

Main Take-Aways

- MLlib is Spark's machine learning library.
 - Distributed implementations
- Main categories of algorithms and techniques:
 - Machine learning
 - Statistics
 - Utility for ML pipeline

Now what?

Many excellent books released in the past year or two!



Advanced Analytics with Spark (2015), written by Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills



Big Data Analysis with Apache Spark
Learn how to apply data science techniques using parallel programming in Apache Spark to explore big data.



Distributed Machine Learning with Apache Spark
Learn the underlying principles required to develop scalable machine learning pipelines and gain hands-on experience using Apache Spark.



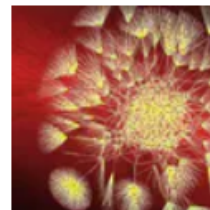
Introduction to Apache Spark
Learn the fundamentals and architecture of Apache Spark, the leading cluster-computing framework among professionals.



Big Data for Data Engineers

Yandex

[Learn More](#)



Big Data

University of California, San Diego

[Learn More](#)