Spark on HDInsight

Ingyu Lee

Troy University

https://github.com/Azure-Bootcamp-Troy/SparkInHDInsight

Content

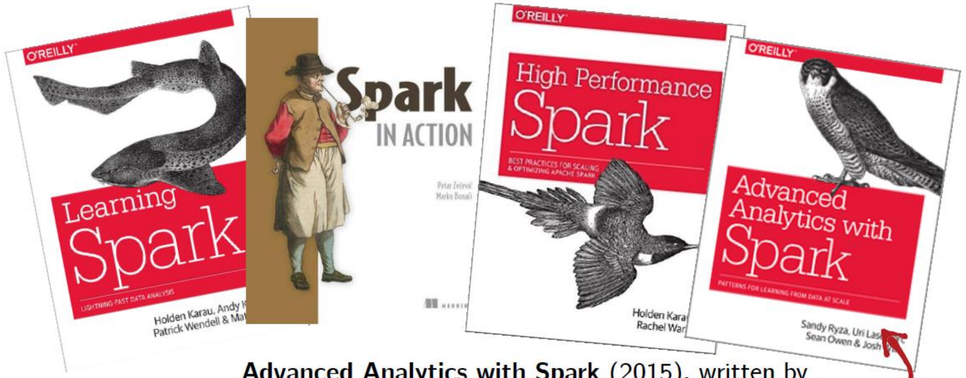
- What is **Spark** on **HDInsight**?
- Demo: Create Spark Cluster on Azure
- What is an **RDD, DataFrames, Spark SQL**?
- Demo: Operations of RDD, DataFrames and SQL
- What is **MLlib**?
- Demo: Machine Learning using Spark

- For Whom this talk
  - Business users to analyze Big Data using Spark on Azure Cluster.
- Prerequisite
  - None

References

This talk is based on this course

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

Microsoft
Implementing Predictive Analytics with Spark in

Microsoft
Processing Real-Time Data Streams in Azure

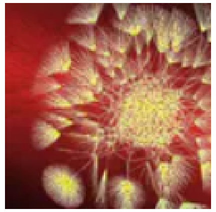
Microsoft
Processing Big Data with Hadoop in Azure

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

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

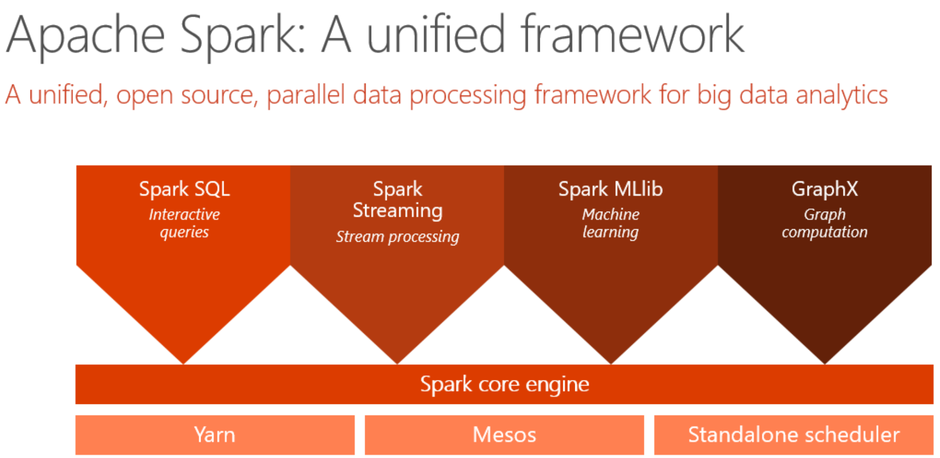
Well-suited for machine learning

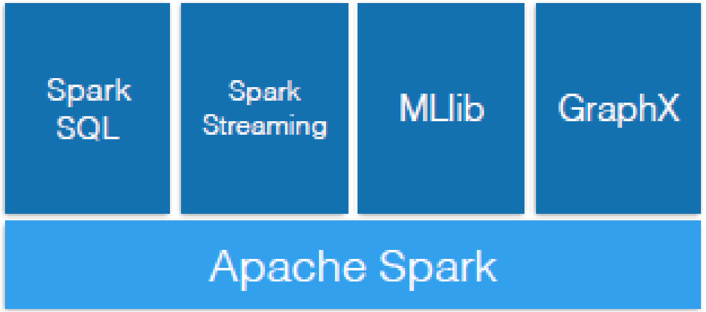
- Fast iterative procedures
- Efficient communication primitives

Simple and Expressive

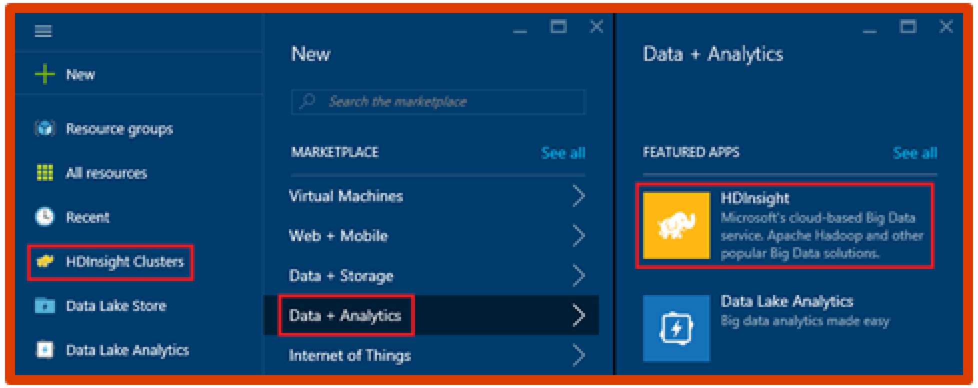
- APIs in Scala, Java, Python, R
- Interactive Shell

Integrated Higher-Level Libraries



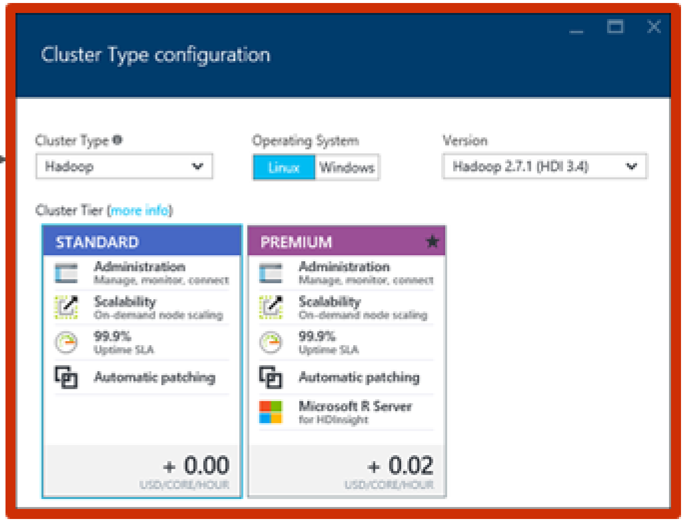


Demo: Creating an HDInsight Spark cluster

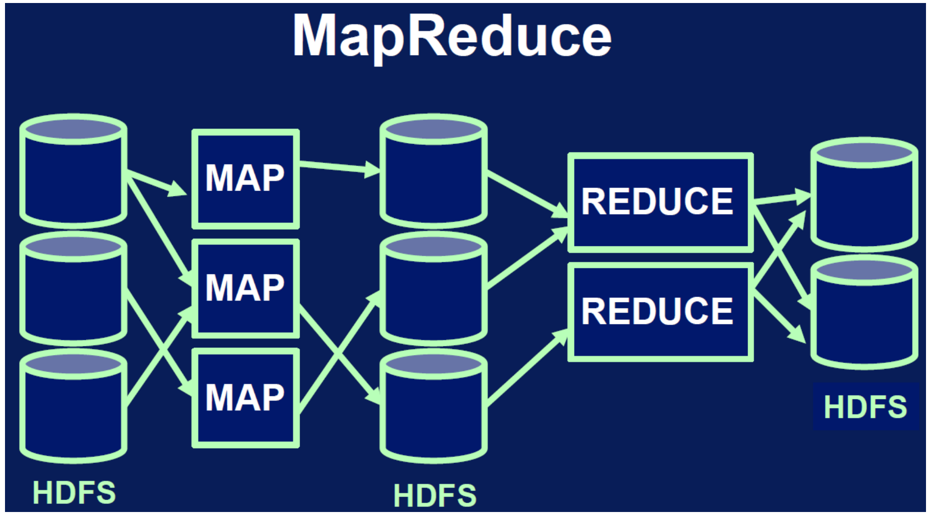


Apache Spark use cases

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

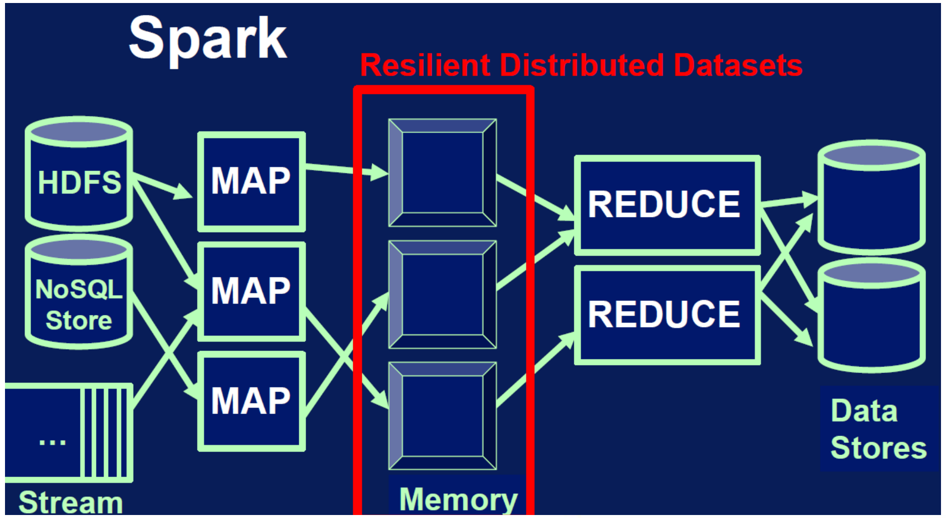
Other workflows? i.e. join, filter, map-reduce-map

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?

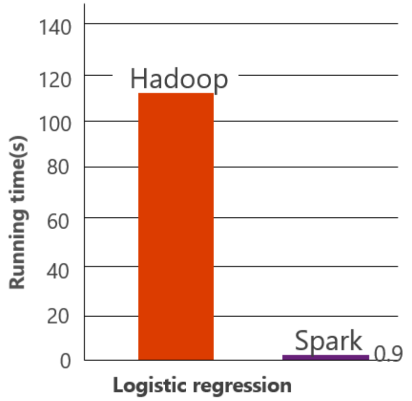
Native Python, Scala (, R) interface. Interactive shells.

Iterative algorithms? i.e. machine learning

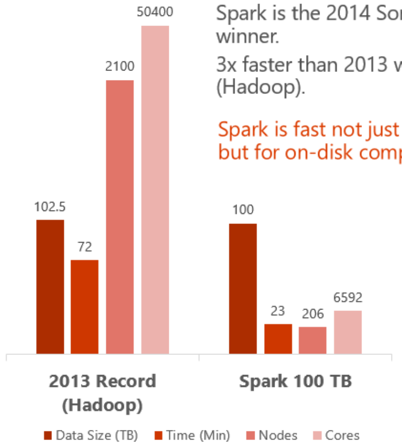
in-memory caching of data, specified by the user

Why sparks is faster?

Faster data, faster results



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



Spark is the 2014 Sort Benchmark winner.

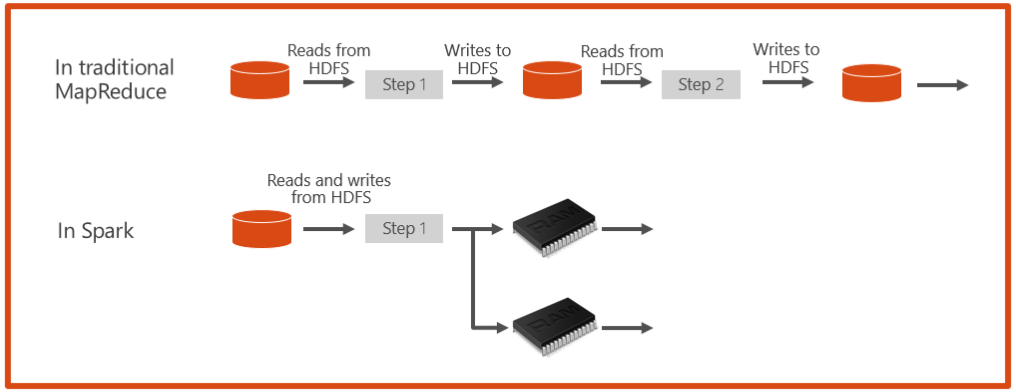
3x faster than 2013 winner (Hadoop).

Spark is fast not just for in-memory, but for on-disk computation too.

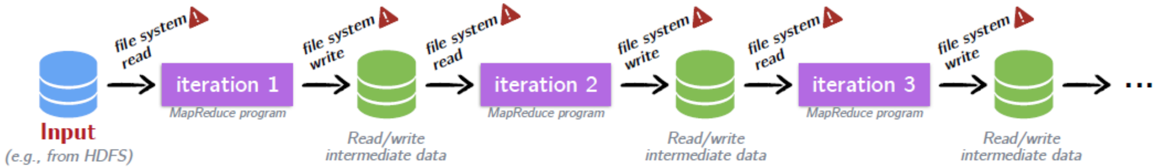
tinyurl.com/spark-sort

What makes Spark fast?

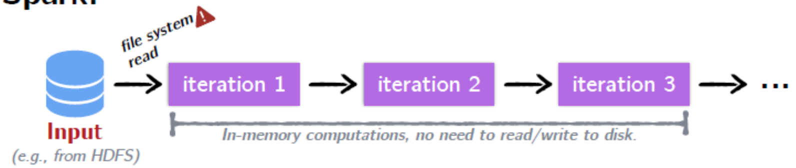
Data sharing between steps of a job



Iteration in Hadoop:

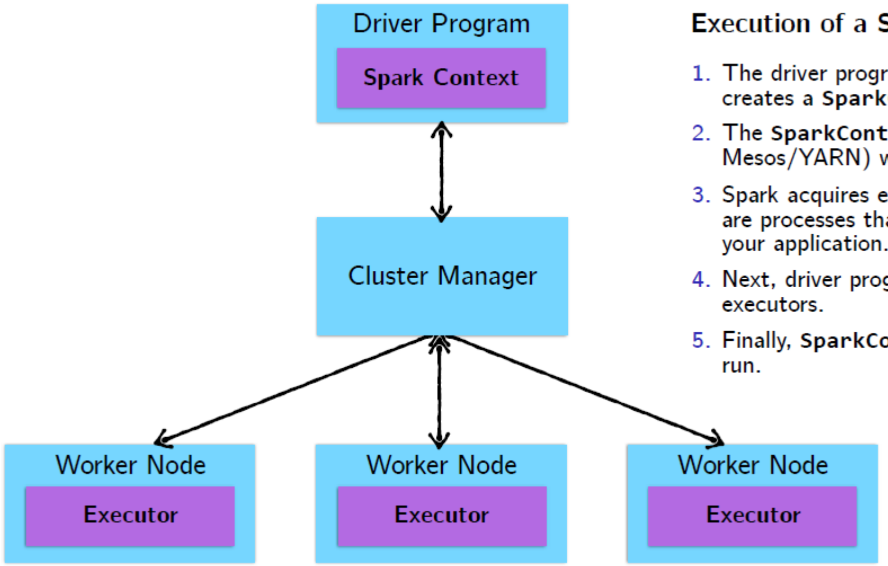


Iteration in Spark:



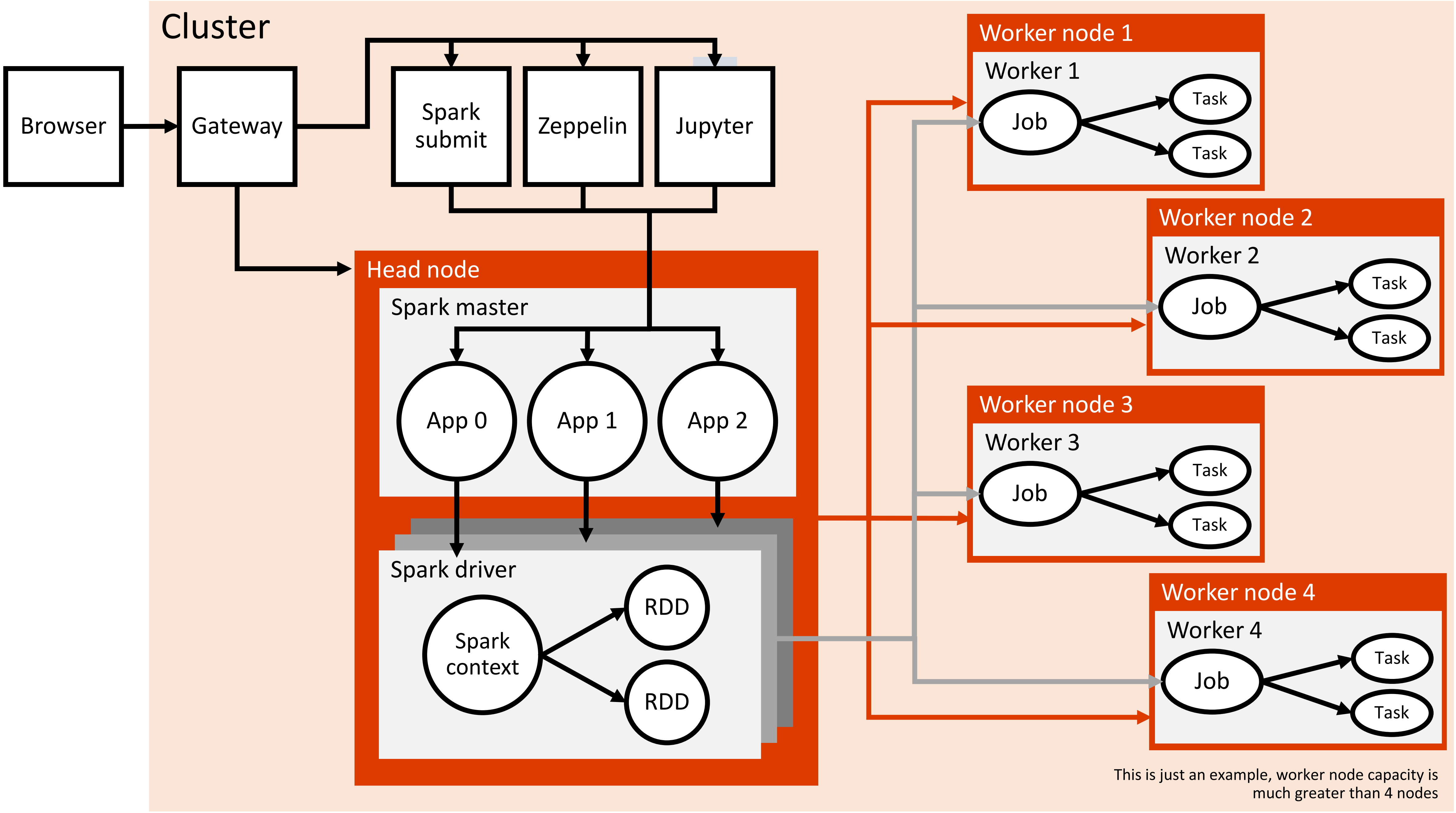
Spark cluster architecture

- 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



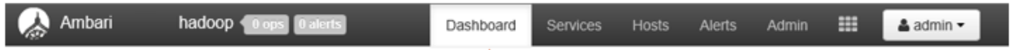
Execution of a Spark program:

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.



Cluster
Browser
Gateway
Spark submit
Zeppelin
Jupyter
Head node
Spark master
App 0
App 1
App 2
Spark driver
Spark context
RDD
RDD
Worker node 1
Worker 1
Job
Task
Task
Worker node 2
Worker 2
Job
Task
Task
Worker node 3
Worker 3
Job
Task
Task
Worker node 4
Worker 4
Job
Task
Task
This is just an example, worker node capacity is much greater than 4 nodes

Demo: Ambari Dashboard

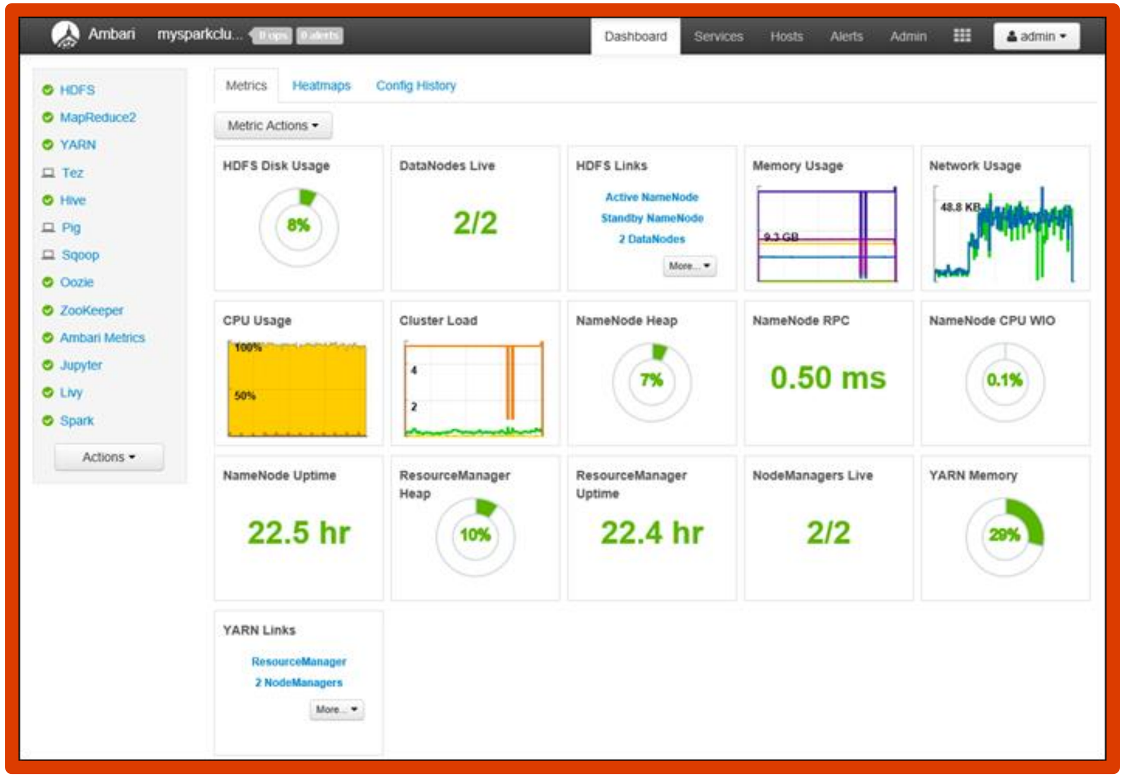


The Ambari Dashboard provides links to access:

Ambari web UI – monitors Spark clusters

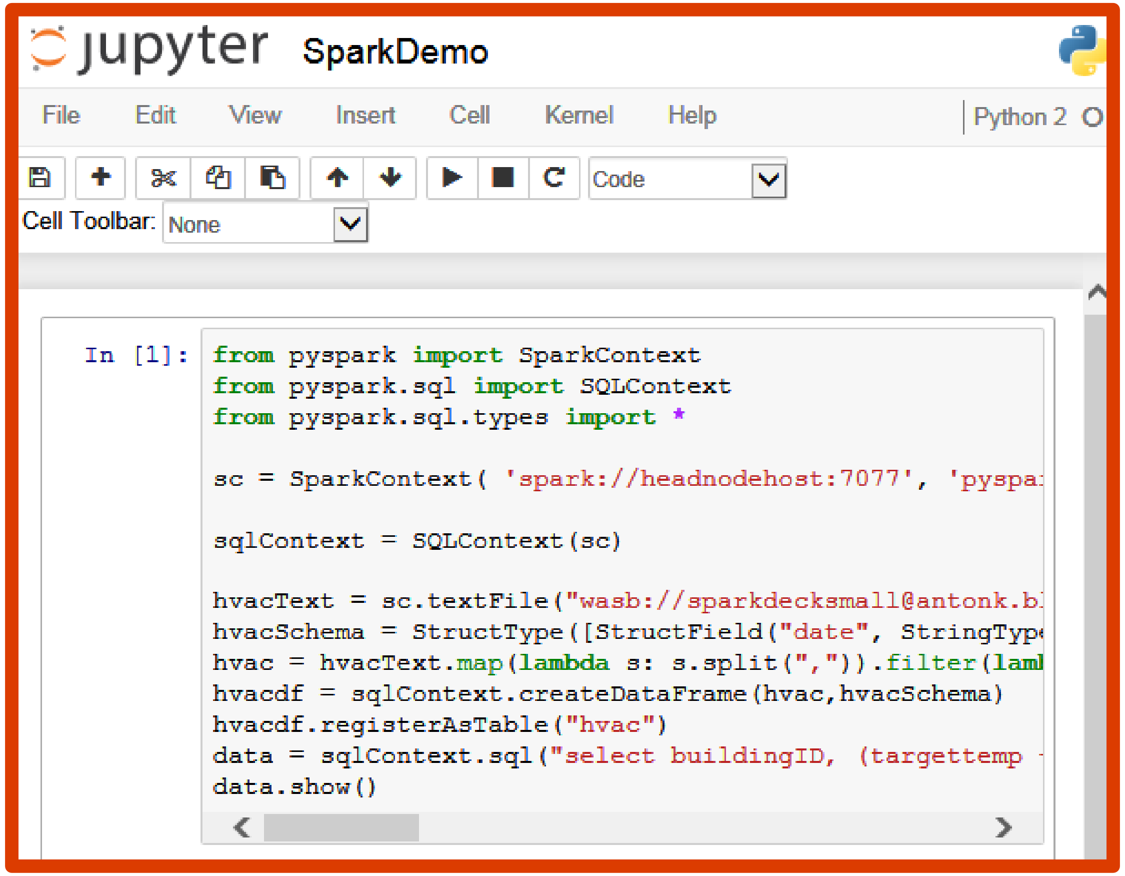
Hosts (Resource Manager) – controls amount of resources allocated to various Spark cluster components

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



Demo: Jupyter notebooks

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



What is RDD?

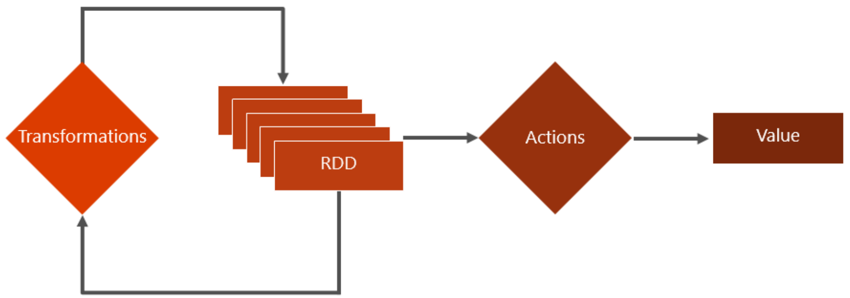
- The core abstraction for data in Spark is the *resilient distributed dataset* (RDD)
- An RDD represents a collection of items that can be distributed across compute nodes
- APIs for working with RDDs are provided for Java, Python, and Scala
  – HDInsight distribution includes Python and Scala shells

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

Resilient Distributed Datasets or RDDs address this by enabling fault-tolerant, distributed, in-memory computations.

RDD Operations

RDDs: Transformations and actions



Some Transformations

Transformation	Description
`map(func)`	return a new distributed dataset formed by passing each element of the source through a function *func*
`filter(func)`	return a new dataset formed by selecting those elements of the source on which *func* returns true
`distinct([numTasks]))`	return a new dataset that contains the distinct elements of the source dataset
`flatMap(func)`	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)

Key-Value Transformation	Description
`reduceByKey(func)`	return a new distributed dataset of (K,V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) ➔ V
`sortByKey()`	return a new dataset (K,V) pairs sorted by keys in ascending order
`groupByKey()`	return a new dataset of (K, Iterable<V>) pairs

Some Actions

Action	Description
`reduce(func)`	aggregate dataset's elements using function *func*. *func* takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel
`take(n)`	return an array with the first *n* elements
`collect()`	return all the elements as an array WARNING: make sure will fit in driver program
`takeOrdered(n, key=func)`	return n elements ordered in ascending order or as specified by the optional key function

Transformations and actions: Sample code

Sample code to search through error messages in a log file (stored in HDFS)

```
val file = spark.textFile("hdfs://...")

val errors = file.filter(line => line.contains("ERROR"))

// Cache errors
errors.cache()

// Count all the errors
errors.count()

// Count errors mentioning MySQL
errors.filter(line => line.contains("Web")).count()

// Fetch the MySQL errors as an array of strings
errors.filter(line => line.contains("Error")).collect()
```

Transformation: `hdfs(), filter()`
Action: `count(), collect()`
Cache() `is a method`

RDD transformations

Sample code of all RDD transformations

```
// Returns a new distributed data set formed by passing each
element of the source through a function func.
map(func)

// Returns a new data set formed by selecting those elements
of the source on which func returns true.
filter(func)

// Returns a new data set that contains the union of the
elements in the source data set and in the argument.
union(otherDataset)


// Returns a new data set that contains the distinct elements of
the source data set.
distinct(([numTasks]))
```

RDD actions

Sample code of all RDD actions

```
// Writes the elements of the data set as a text file (or a set
of text files) in a given directory in either the local
filesystem, HDFS, or other Hadoop-supported file systems.
Spark will call ToString on each element to convert it to a
line of text in the file.
saveAsTextFile(path)

// Returns a “Map” of (K, Int) pairs with the count of each key.
Only available on RDDs of type (K, V).
countByKey()

// Returns all the elements of the data set as an array at the
driver program. Usually useful after a filter or other operation
returns a sufficiently small subset of the data.
collect()
```

Demo: RDD

RDDs can be created from stable storage or by transforming other RDDs.

```
In [ ]: fruits = spark.sparkContext.textFile('wasb:///example/data/fruits.txt')
        yellowThings = spark.sparkContext.textFile('wasb:///example/data/yellowthings.txt')
```

Transformations create a new dataset from an existing one. Transformations are lazy, meaning that no transformation is executed until you execute an action.

```
# map
fruitsReversed = fruits.map(lambda fruit: fruit[::-1])
```

```
# filter
shortFruits = fruits.filter(lambda fruit: len(fruit) <= 5)
```

```
# flatMap
characters = fruits.flatMap(lambda fruit: list(fruit))
```

```
# union
fruitsAndYellowThings = fruits.union(yellowThings)
```

```
# intersection
yellowFruits = fruits.intersection(yellowThings)
```

```
# distinct
distinctFruitsAndYellowThings = fruitsAndYellowThings.distinct()
distinctFruitsAndYellowThings
```

```
# groupByKey
yellowThingsByFirstLetter = yellowThings.map(lambda thing: (thing[0], thing)).groupByKey()
```

```
# reduceByKey
numFruitsByLength = fruits.map(lambda fruit: (len(fruit), 1)).reduceByKey(lambda x, y: x + y)
```

Actions return a value to the driver program after running a computation on the dataset.

```
# collect
fruitsArray = fruits.collect()
yellowThingsArray = yellowThings.collect()
fruitsArray
```

```
# count
numFruits = fruits.count()
numFruits
```

```
# take
first3Fruits = fruits.take(3)
first3Fruits
```

```
# reduce
letterSet = fruits.map(lambda fruit: set(fruit)).reduce(lambda x, y: x.union(y))
letterSet
```

What is 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

Creating DataFrames from data sources

A Spark data source can read in-data to create DataFrames, which has a schema that Spark understands. Examples include: JSON files, JDBC source, Parquet, and Hive tables.

```
>>> val df = sqlContext.jsonFile(”somejasonfile.json“)                    //from JSON file*

>>> val df = hiveContext.table(”somehivetable“)                                //from a
Hive Table

>>> val df = sqlContext.parquetFile(”someparquetsource“)                 //from a parquet
file

>>> val df = sqlContext.load(source="jdbc", url=“UrlToConnect", dbtable=“tablename")//from JDBC source
```

* Note that the file that is offered as *jsonFile* is not a typical JSON file. Each line must contain a separate, self-contained valid JSON object. A regular multi-line JSON file will most often fail.

Creating DataFrames from RDDs

Create DataFrames from existing RDDs in two ways:

1. **Use reflection:** Infer the schema of an RDD that contains specific types of objects. This approach leads to more concise code. It works well when you already know the schema while writing your Spark application.

2. **Specify the schema programmatically:** Enables you to construct a schema then apply it to an existing RDD. This method is more verbose, but it enables you to construct DataFrames when the columns and their types are not known until runtime.

Create DataFrames from existing a JSON RDD, using the *jsonRDD* function.

```
>>> val df = sqlContext.jsonRDD(anUserRDD)
```

DataFrame Operations

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

```
>>> val df = sqlContext.jsonFile(”somejasonfile.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
```

What is Spark SQL?

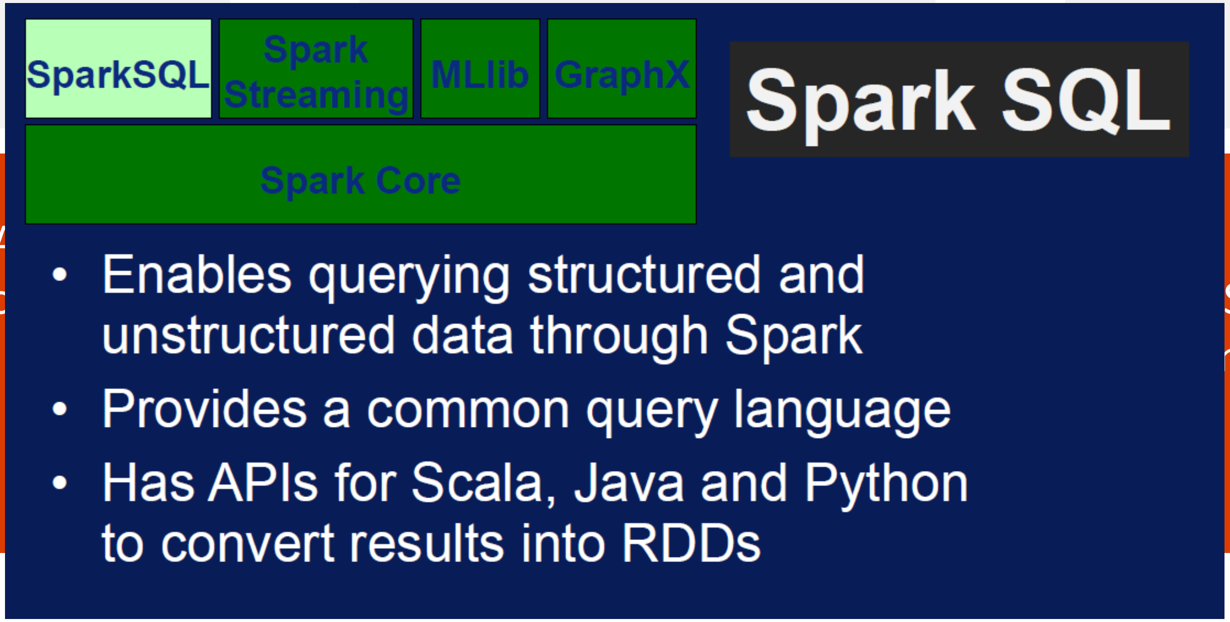
1
Create an Azure storage account

2
HDInsight makes Apache Spark available as a service in cloud.

3
Run Spark SQL statements using notebooks

HDInsight uses A
Blob storage acc
storing data.

n interactive
SQL statements
notebooks.



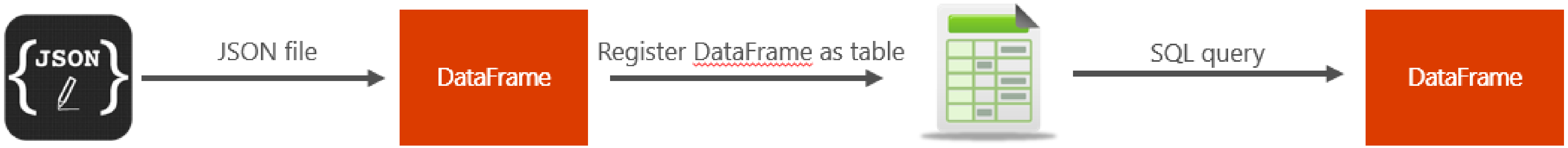
Tables and Queries

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”)
```



Hive tables

Spark SQL also supports reading and writing data stored in **Apache Hive**.

```
// Create a HiveContext, which is derived from SQLContext.
>>> val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
>>> sqlContext.sql("CREATE TABLE IF NOT EXISTS UserTable (key INT, value
STRING)")
>>> sqlContext.sql("LOAD DATA LOCAL INPATH 'user.txt' INTO TABLE UserTable")
// Queries are expressed in HiveQL.
>>> val df = sqlContext.sql("FROM UserTable SELECT Name, Age")
```

Spark SQL also supports reading and writing data stored in **Apache Hive**.

```
// Create a HiveContext, which is derived from SQLContext.
>>> val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
>>> sqlContext.sql("CREATE TABLE IF NOT EXISTS UserTable (key INT, value
STRING)")
>>> sqlContext.sql("LOAD DATA LOCAL INPATH 'user.txt' INTO TABLE UserTable")
// Queries are expressed in HiveQL.
>>> val df = sqlContext.sql("FROM UserTable SELECT Name, Age")
```

Spark SQL Operations

Spark RDD API

```
rdd
rdd1 = rdd.map(lambda x: x.split("\t"))
rdd2 = rdd1.map(lambda x: (x[0], x[2]))
```

VS

SQL on Spark

```
select user_id, url from access_log
```

- No data parsing
- Code optimization
- Syntax is easier
- No overhead

Demo: DataFrame and SQL

create a dataframe from a CSV file as shown below.

```
df = spark.read.csv('wasb:///HdiSamples/HdiSamples/SensorSampleData/building/building.csv', header=True, inferSchema=True)
```

```
# show the content of the dataframe
df.show()
```

```
# Print the dataframe schema in a tree format
df.printSchema()
```

```
# Create an RDD from the dataframe
dfrdd = df.rdd
dfrdd.take(3)
```

```
dfrdd = df.rdd
dfrdd.take(3)
```

```
# Retrieve a given number of rows from the dataframe
df.limit(3).show()
```

```
df.limit(3).show()
```

```
# Retrieve specific columns from the dataframe
df.select('BuildingID', 'Country').limit(3).show()
```

```
# Use GroupBy clause with dataframe
df.groupBy('HVACProduct').count().select('HVACProduct', 'count').show()
```

run SQL queries over dataframes once you register them as temporary tables within the SparkSession

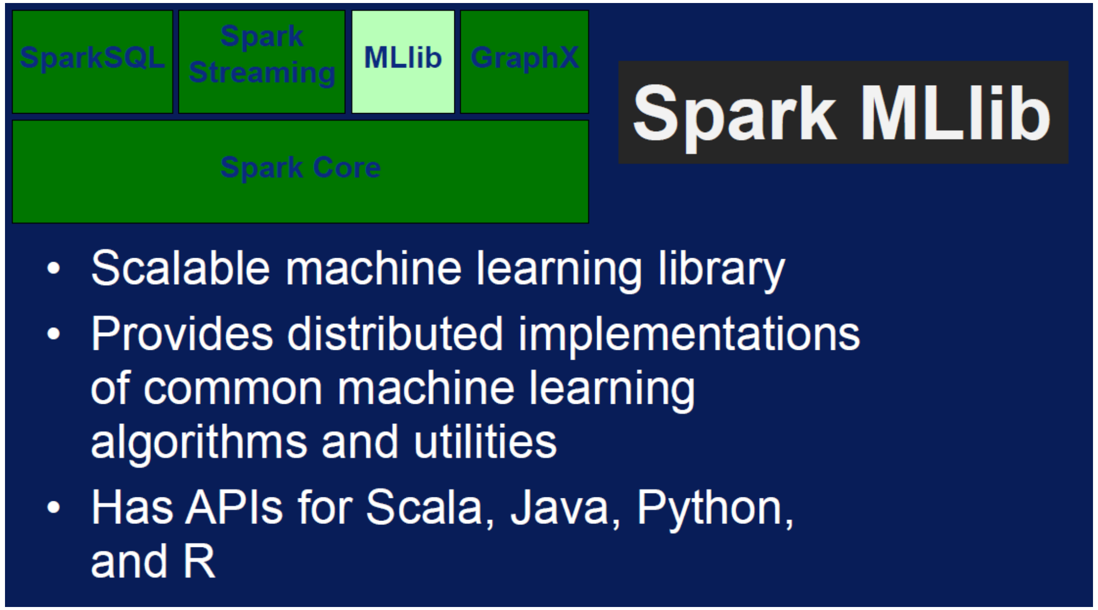
```
# Register the dataframe as a temporary table called HVAC
df.registerTempTable('HVAC')
```

```
%%sql
SELECT * FROM HVAC WHERE BuildingAge >= 10
```

```
%%sql
SELECT BuildingID, Country FROM HVAC LIMIT 3
```

What is MLlib?

- MLlib is a collection of **machine learning algorithms optimized to run in a parallel, distributed manner on Spark clusters**.
- MLlib helps lead to better performance on large data sets.
- MLlib integrates seamlessly with other Spark components.
- MLlib applications are developed in Java, Scala, and Python.



Type	Algorithms
Supervised	Classification and regression: • Linear models (SVMs) logistic regression and linear regression • Naïve Bayes • Decision trees • Ensembles of trees (random forest, gradient-boosted trees) • Isotonic regression
Unsupervised	Clustering: • k-means and streaming k-means • Gaussian mixture • Power iteration clustering (PIC) • Latent Dirichlet Allocation (LDA)
Recommendation	Collaborative filtering: • Alternating least squares (ALS)

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

Example: Classification I

Import Spark SQL and Spark ML Libraries

First, import the libraries you will need:

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
```

Step 2: Load Source Data

You will load this data into a DataFrame and display it.

```
csv = spark.read.csv('wasb:///data/flights.csv', inferSchema=True, header=True)
csv.show()
```

```
+----------+---------+-------+---------------+-------------+--------+--------+
|DayofMonth|DayOfWeek|Carrier|OriginAirportID|DestAirportID|DepDelay|ArrDelay|
+----------+---------+-------+---------------+-------------+--------+--------+
|        19|        5|     DL|          11433|        13303|      -3|       1|
|        19|        5|     DL|          14869|
```

Step 3: Prepare the Data

```
data = csv.select("DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay",
                  ((col("ArrDelay") > 15).cast("Int").alias("Late")))
data.show()
```

```
+----------+---------+---------------+-------------+--------+----+
|DayofMonth|DayOfWeek|OriginAirportID|DestAirportID|DepDelay|Late|
+----------+---------+---------------+-------------+--------+----+
|        19|        5|          11433|        13303|      -3|   0|
|        19|        5|          14869|        12478|       0|   0|
|        19|        5|          14057|        14869|      -4|   0|
```

Example: Classification II

Step 4: Split the Data

```
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

Step 5: Prepare the Training Data

```
assembler = VectorAssembler(inputCols = ["DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay"],
                            outputCol="features")
training = assembler.transform(train).select(col("features"), col("Late").alias("label"))
training.show()
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[1.0,1.0,10140.0,...|    0|
```

Step 6: Train a Classification Model

```
lr = LogisticRegression(labelCol="label",featuresCol="features",maxIter=10,regParam=0.3)
model = lr.fit(training)
print "Model trained!"
```

```
Model trained!
```

Example: Classification III

Step 7: Prepare the Testing Data

```
testing = assembler.transform(test).select(col("features"), col("Late").alias("trueLabel"))
testing.show()
```

```
+--------------------+---------+
|            features|trueLabel|
+--------------------+---------+
|[1.0,1.0,10140.0,...|        0|
|[1.0,1.0,10140.0,...|        0|
```

Step 8: Test the Model

```
prediction = model.transform(testing)
predicted = prediction.select("features", "prediction", "probability", "trueLabel")
predicted.show(100)
```

```
+--------------------+----------+--------------------+---------+
|            features|prediction|         probability|trueLabel|
+--------------------+----------+--------------------+---------+
|[1.0,1.0,10140.0,...|       0.0|[0.83121028425635...|        0|
|[1.0,1.0,10140.0,...|       0.0|[0.82357817040192...|        0|
```

Example: Regression I

Import Spark SQL and Spark ML Libraries

First, import the libraries you will need:

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
```

```
Starting Spark application
```

Step 2: Load Source Data

```
csv = spark.read.csv('wasb:///data/flights.csv', inferSchema=True, header=True)
csv.show()
```

```
+----------+---------+-------+---------------+-------------+--------+--------+
|DayofMonth|DayOfWeek|Carrier|OriginAirportID|DestAirportID|DepDelay|ArrDelay|
+----------+---------+-------+---------------+-------------+--------+--------+
|        19|        5|     DL|          11433|        13303|      -3|       1|
|        19|        5|     DL|          14869|        12478|       0|      -8|
|        19|        5|     DL|          14057|        14869|      -4|     -15|
```

Step 3: Prepare the Data

```
data = csv.select("DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay", "ArrDelay")
data.show()
```

```
+----------+---------+---------------+-------------+--------+--------+
|DayofMonth|DayOfWeek|OriginAirportID|DestAirportID|DepDelay|ArrDelay|
+----------+---------+---------------+-------------+--------+--------+
|        19|        5|          11433|        13303|      -3|       1|
```

Example: Regression II

Step 4: Split the Data

```
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

```
Training Rows: 1891270  Testing Rows: 810948
```

Step 5: Prepare the Training Data

```
assembler = VectorAssembler(inputCols = ["DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay"],
                            outputCol="features")
training = assembler.transform(train).select(col("features"), (col("ArrDelay").cast("Int").alias("label")))
training.show()
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[1.0,1.0,10140.0,...|  -11|
```

Step 6: Train a Regression Model

```
lr = LinearRegression(labelCol="label",featuresCol="features", maxIter=10, regParam=0.3)
model = lr.fit(training)
print "Model trained!"
```

```
Model trained!
```

Example: Regression III

Step 7: Prepare the Testing Data

```
testing = assembler.transform(test).select(col("features"), (col("ArrDelay")).cast("Int").alias("trueLabel"))
testing.show()
```

```
+--------------------+---------+
|            features|trueLabel|
+--------------------+---------+
|[1.0,1.0,10140.0,...|       -5|
|[1.0,1.0,10140.0,...|      -25|
```

Step 8: Test the Model

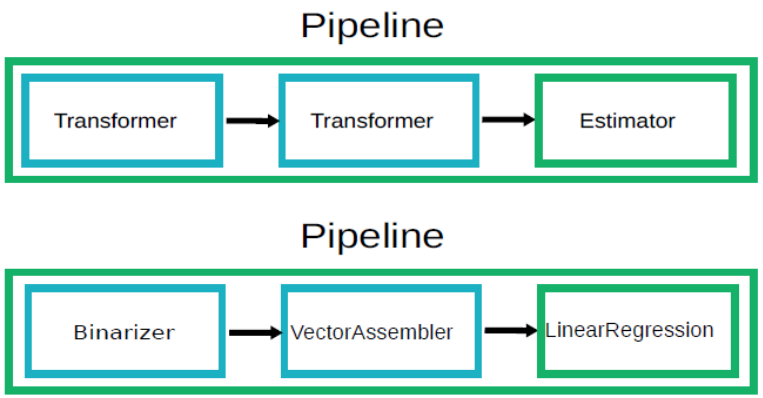
```
prediction = model.transform(testing)
predicted = prediction.select("features", "prediction", "trueLabel")
predicted.show()
```

```
+--------------------+-------------------+---------+
|            features|         prediction|trueLabel|
+--------------------+-------------------+---------+
|[1.0,1.0,10140.0,...|-3.7579519642834542|       -5|
|[1.0,1.0,10140.0,...| -9.743936006015753|      -25|
```

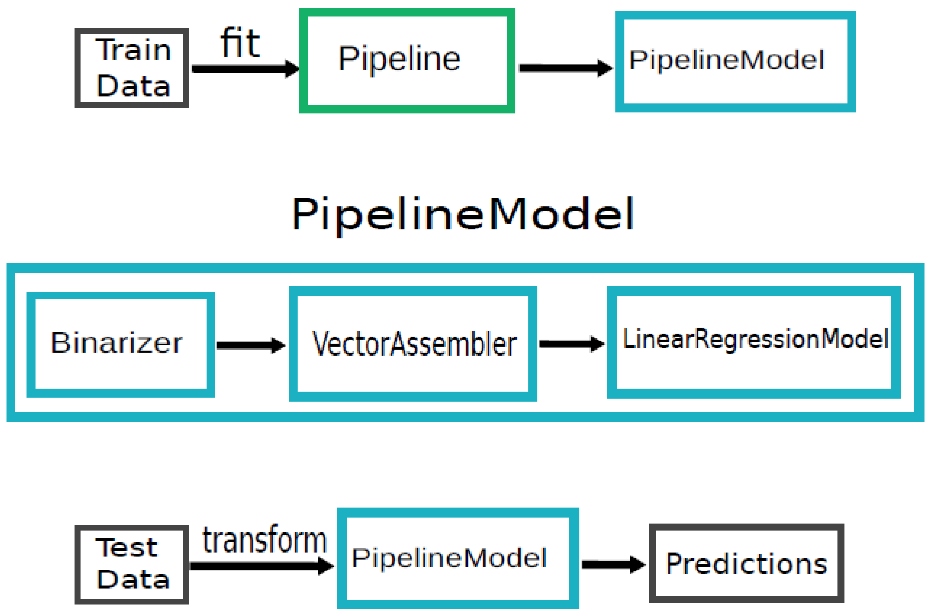
What is Pipeline?

ML: Pipelines

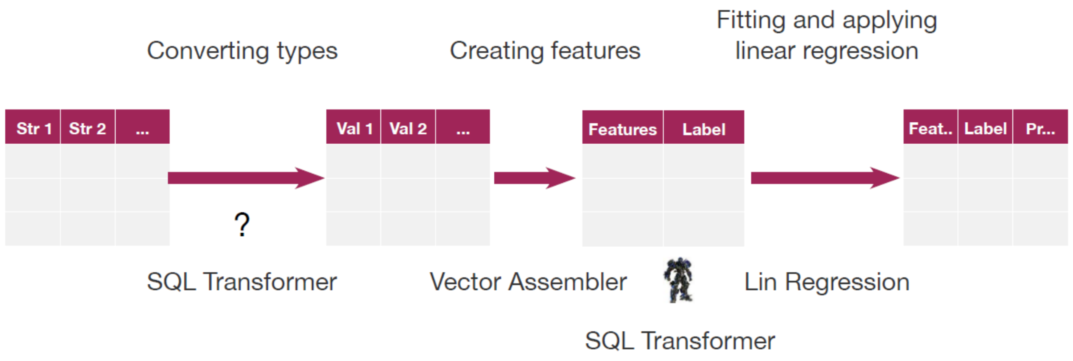
A *Pipeline* is an estimator that contains stages representing a resusable workflow. Pipeline stages can be either estimators or transformers.



ML: PipelineModel



Pipeline



Example: Pipeline

Step 1: Define the Pipeline

```
strIdx = StringIndexer(inputCol = "Carrier", outputCol = "CarrierIdx")
catVect = VectorAssembler(inputCols = ["CarrierIdx", "DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID"],
                          outputCol="catFeatures")
catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol = "idxCatFeatures")
numVect = VectorAssembler(inputCols = ["DepDelay"], outputCol="numFeatures")
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(), outputCol="normFeatures")
featVect = VectorAssembler(inputCols=["idxCatFeatures", "normFeatures"], outputCol="features")
dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
pipeline = Pipeline(stages=[strIdx, catVect, catIdx, numVect, minMax, featVect, dt])
```

```
piplineModel = pipeline.fit(train)
print "Pipeline complete!"
```

Step 2: Run the Pipeline as an Estimator

```
prediction = piplineModel.transform(test)
predicted = prediction.select("features", "prediction", "trueLabel")
predicted.show(100, truncate=False)
```

Step 3: Test the Pipeline Model

Example: Text Analysis

Define the Pipeline

The pipeline for the model consist of the following stages:

- A Tokenizer to split the tweets into individual words.
- A StopWordsRemover to remove common words such as "a" or "the" that have little predictive value.
- A HashingTF class to generate numeric vectors from the text values.
- A LogisticRegression algorithm to train a binary classification model.

```
tokenizer = Tokenizer(inputCol="SentimentText", outputCol="SentimentWords")
swr = StopWordsRemover(inputCol=tokenizer.getOutputCol(), outputCol="MeaningfulWords")
hashTF = HashingTF(inputCol=swr.getOutputCol(), outputCol="features")
lr = LogisticRegression(labelCol="label", featuresCol="features", maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, swr, hashTF, lr])
```

:

```
piplineModel = pipeline.fit(train)
print "Pipeline complete!"
```

Step 2: Run the Pipeline as an Estimator

Step 3: Test the Pipeline Model

```
prediction = piplineModel.transform(test)
predicted = prediction.select("SentimentText", "prediction", "trueLabel")
predicted.show(100, truncate = False)
```

Example: Clustering

Step 1: Create the K-Means Model

```
assembler = VectorAssembler(inputCols = ["Age", "MaritalStatus", "IncomeRange", "Gender", "TotalChildren",
                                         "ChildrenAtHome", "Education", "Occupation", "HomeOwner", "Cars"], outputCol="features")
train = assembler.transform(customers)

kmeans = KMeans(featuresCol=assembler.getOutputCol(), predictionCol="cluster", k=5, seed=0)
model = kmeans.fit(train)
print "Model Created!"
```

```
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

Step 2: Get the Cluster Centers

```
prediction = model.transform(train)
prediction.groupBy("cluster").count().orderBy("cluster").show()
```

```
prediction.select("CustomerName", "cluster").show(50)
```

Step 3: Predict Clusters

Example: Recommendation I

Import the ALS class

In this exercise, you will use the Alternating Least Squa

```
from pyspark.ml.recommendation import ALS
```

Load Source Data

The source data for the recommender is in two files - one containing numeric IDs for movies and users, a
of the movies.

```
ratings = spark.read.csv('wasb:///data/ratings.csv', inferSchema=True, header=True)
movies = spark.read.csv('wasb:///data/movies.csv', inferSchema=True, header=True)
ratings.join(movies, "movieId").show()
```

Prepare the Data

To prepare the data, split it into a training set and a test set.

```
data = ratings.select("userId", "movieId", "rating")
splits = data.randomSplit([0.7, 0.3])
train = splits[0].withColumnRenamed("rating", "label")
test = splits[1].withColumnRenamed("rating", "trueLabel")
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

Example: Recommendation II

Build the Recommender

The ALS class is an estimator, so you can use its **fit** method to traing a model, or you can include it in a pipeline. Ratl
label, the ALS algorithm requries a numeric user ID, item ID, and rating.

```
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="label")
model = als.fit(train)
```

Test the Recommender

Now that you've trained the recommender, you can see how accurately it predicts known ratings in the test set.

```
prediction = model.transform(test)
prediction.join(movies, "movieId").select("userId", "title", "prediction", "trueLabel").show(100, truncate=False)
```

Questions