skillspeed

for the serious learner

Apache Spark and Scala

Module 7: Shark and SparkSQL

Course Topics

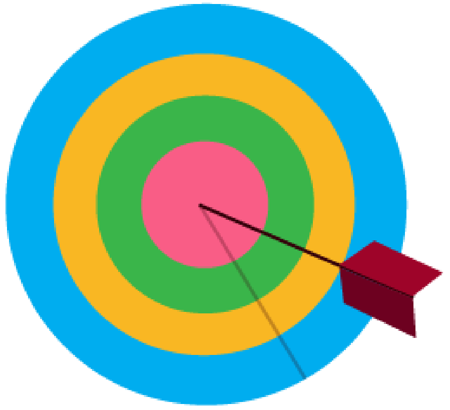
skillspeed
for the serious learner

Module	Topic
Module 1	Getting Started / Introduction to Scala
Module 2	Scala – Essentials and Deep Dive
Module 3	Introducing Traits and OOPS in Scala
Module 4	Functional Programming in Scala
Module 5	Spark and Big Data
Module 6	Understanding RDDs
Module 7	Shark and SparkSQL
Module 8	Advanced Spark Concepts and Project Discussion

Session Objectives

At the end of this module, you will be able to:

- Analyze Hive and Spark SQL Architecture
- Analyze Spark SQL
- Implement a sample example for Spark SQL
- Implement Data Visualization in Spark



Shark

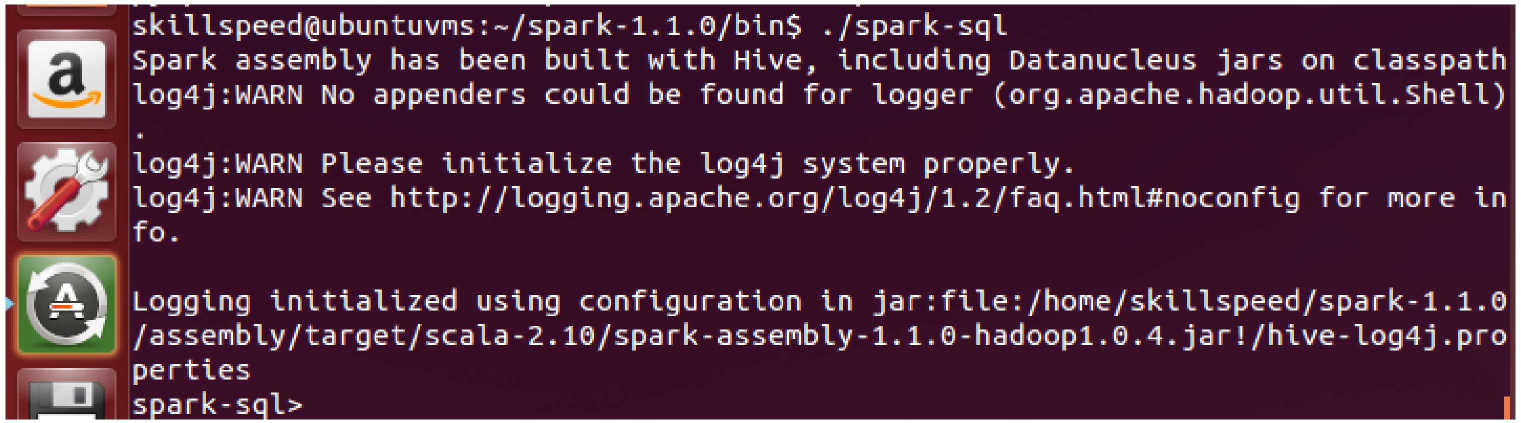
- Shark built on the Hive codebase
- Shark uses the Hive query compiler to parse a HiveQL query and generate an abstract syntax tree, which is then turned into a logical plan with some basic optimizations

WHY Shark to Spark SQL?

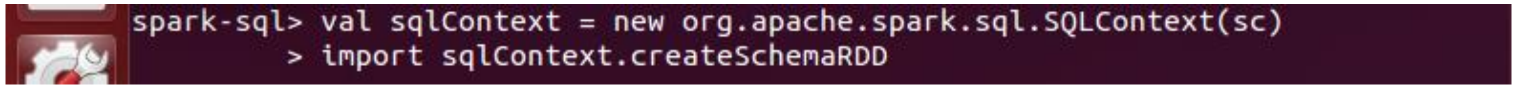
- As we moved to push the boundary of performance optimizations and integrating sophisticated analytics with SQL, we were constrained by the legacy that was designed for MapReduce
- It is for this reason that we are ending development in Shark as a separate project and moving all our development resources to Spark SQL, a new component in Spark
- For Spark users, Spark SQL becomes the narrow-waist for manipulating (semi-) structured data as well as ingesting data from sources that provide schema, such as JSON, Parquet, Hive, or EDWs. It truly unifies SQL and sophisticated analysis, allowing users to mix and match SQL and more imperative programming APIs for advanced analytics

Spark SQL

- Spark SQL allows relational queries through Spark
- The backbone for all these operations is SchemaRDD
- SchemaRDDs are mode of row objects along with the metadata information
- SchemaRDDs are equivalent to RDBMS tables
- They can be constructed from existing RDDs, JSON data sets, Parquet files or Hive QL queries against the data stored in Apache Hive
- Spark SQL needs SQLContext object, which is created from existing SparkContext:



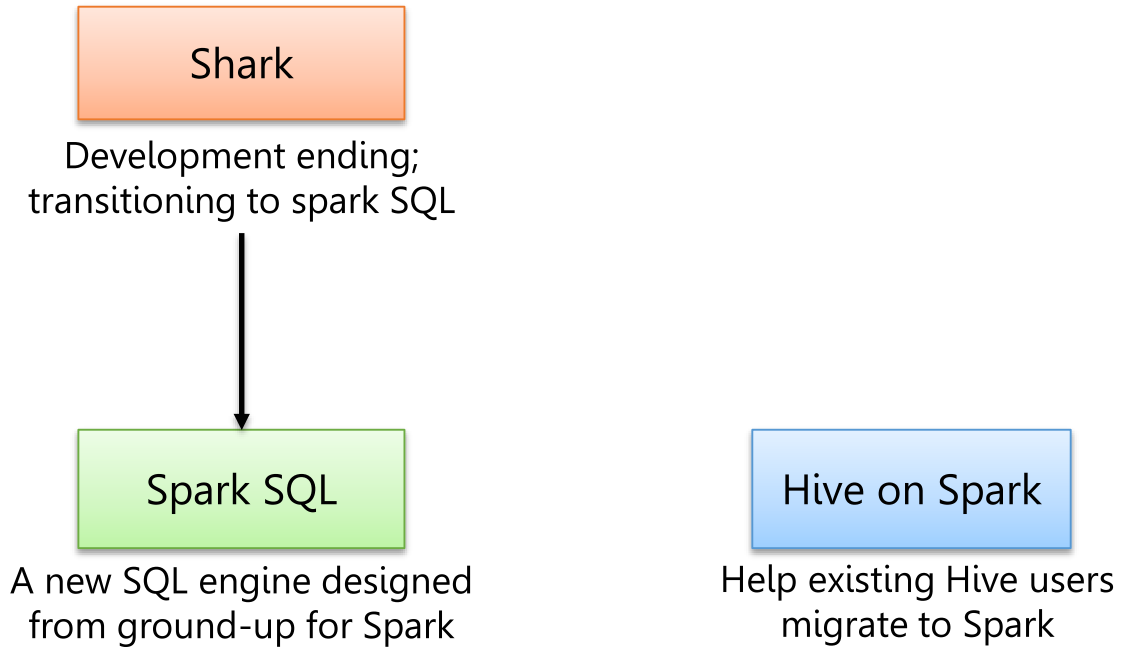
```
skillspeed@ubuntuvms:~/spark-1.1.0/bin$ ./spark-sql
Spark assembly has been built with Hive, including Datanucleus jars on classpath
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell)
.
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.

Logging initialized using configuration in jar:file:/home/skillspeed/spark-1.1.0
/assembly/target/scala-2.10/spark-assembly-1.1.0-hadoop1.0.4.jar!/hive-log4j.pro
perties
spark-sql>
```



```
spark-sql> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
         > import sqlContext.createSchemaRDD
```

Spark SQL (Cont'd)

- Spark SQL lets you query structured data as a distributed dataset (RDD) in Spark, with integrated APIs in Scala and Java
- Shark Project is completely closed now


Shark
Development ending; transitioning to spark SQL
Spark SQL
A new SQL engine designed from ground-up for Spark
Hive on Spark
Help existing Hive users migrate to Spark

Spark SQL (Cont'd)

- Spark SQL lets you query structured data as a distributed dataset (RDD) in Spark, with integrated APIs in Scala and Java
- Spark SQL gives developers the power to integrate SQL commands into applications that also take advantage of MLlib, Spark's machine learning library
- Consider an application that needs to predict which users are likely candidates for a service, based on their profile. Often, such an analysis requires joining data from multiple sources.
- For the purposes of illustration, imagine an application with two tables:

Example:

- Users (userId INT, name String, email STRING, age INT, latitude: DOUBLE, longitude: DOUBLE, subscribed: BOOLEAN)
- Events (userId INT, action INT)

Context in Spark SQL

There are two ways to create context in Spark-SQL:

- SQL Context :

```
scala> import org.apache.spark.sql.-
Scala> var sqlContext = new SQLContext(sc)
```

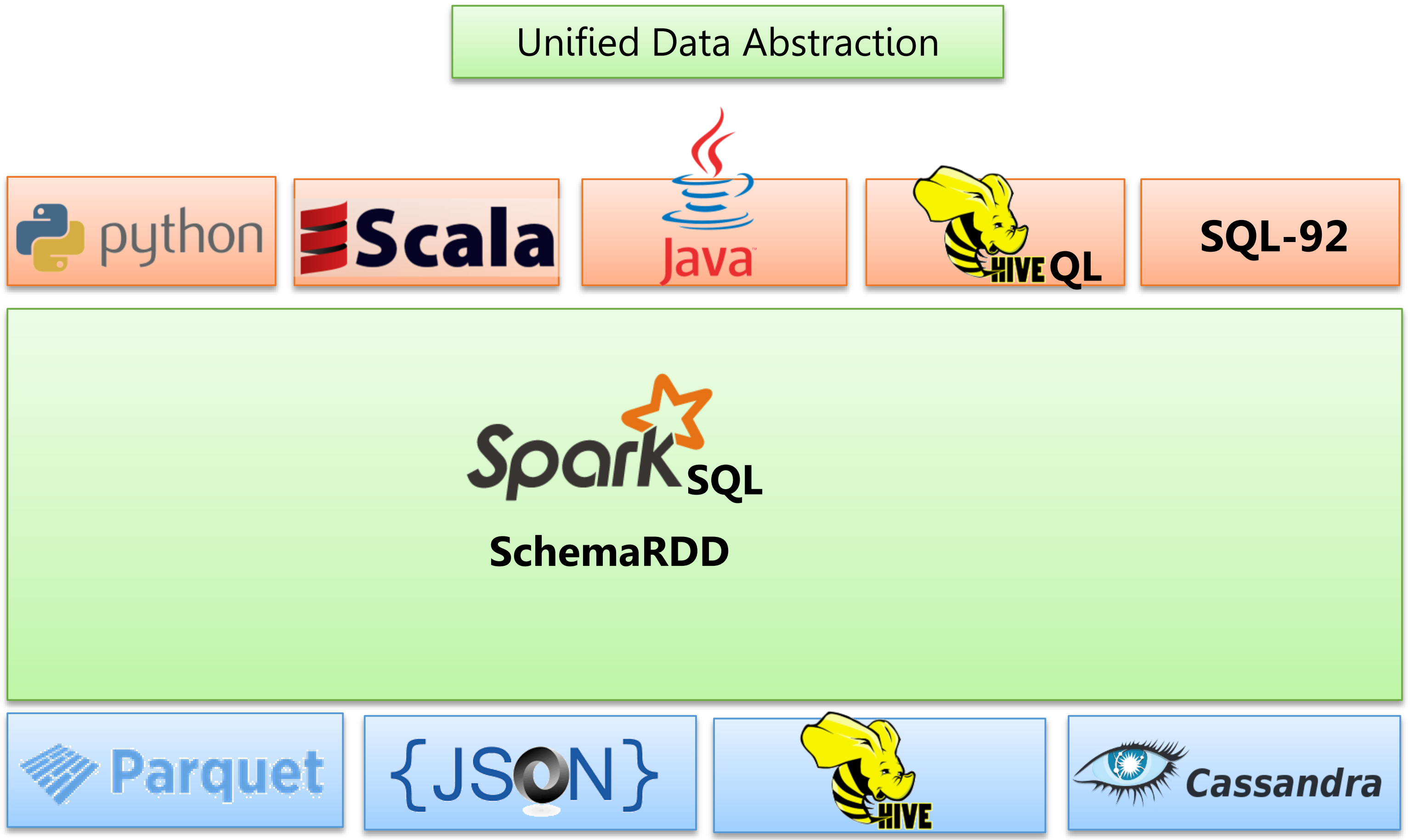
- HiveContext:

```
scala> import org.apache.spark.hive.-
Scala> val hc = new HiveContext(sc)
```

Spark SQL includes a server that exposes its data using JDBC/ODBC

- Query data from HDFS/S3,
- Including formats like Hive/Parquet/JSON
- Support for caching data in-memory in Spark 1.2

Spark SQL


Unified Data Abstraction
python
Scala
Java
HIVE QL
SQL-92
Spark SQL
SchemaRDD
Parquet
{JSON}
HIVE
Cassandra

Caching + API Details

- A full list of SchemaRDD functions can be accessed here:

  http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.SchemaRDD

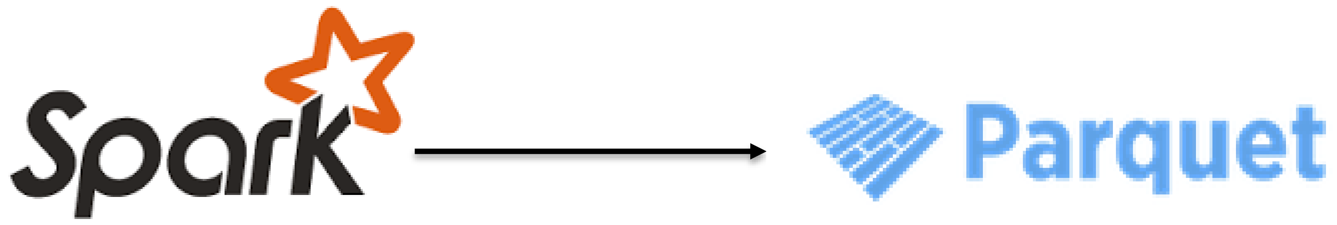
- Spark SQL can cache tables using an in-memory columnar format:
  - Scan only required columns
  - Fewer allocated objects (less GC)
  - Automatically selects best compression

Parquet

- Support for JSON and Parquet file formats Implement Data Visualization in Spark
- Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language

Parquet Compatibility

- Native support for reading data in Parquet:
  - Columnar storage avoids reading unneeded data
  - RDDs can be Written to Parquet files, preserving the schema
  - Converting other slower formats into Parquet for repeated querying

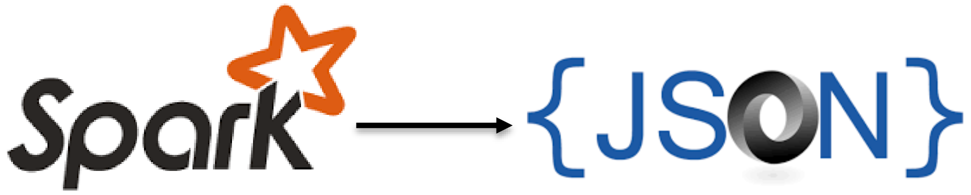


Using Parquet

```
people.write.parquet("people.parquet")

val parquetFile = sqlContext.read.parquet("people.parquet")


parquetFile.registerTempTable("parquetFile")


val teenagers = sqlContext.sql("SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19")


teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
```

JSON Support

- Use jsonFile or jsonRDD to convert a collection of JSON objects into SchemaRDD
- Infers and Union the schema of each record
- Maintains nested structures and Arrays



{JSON} Example

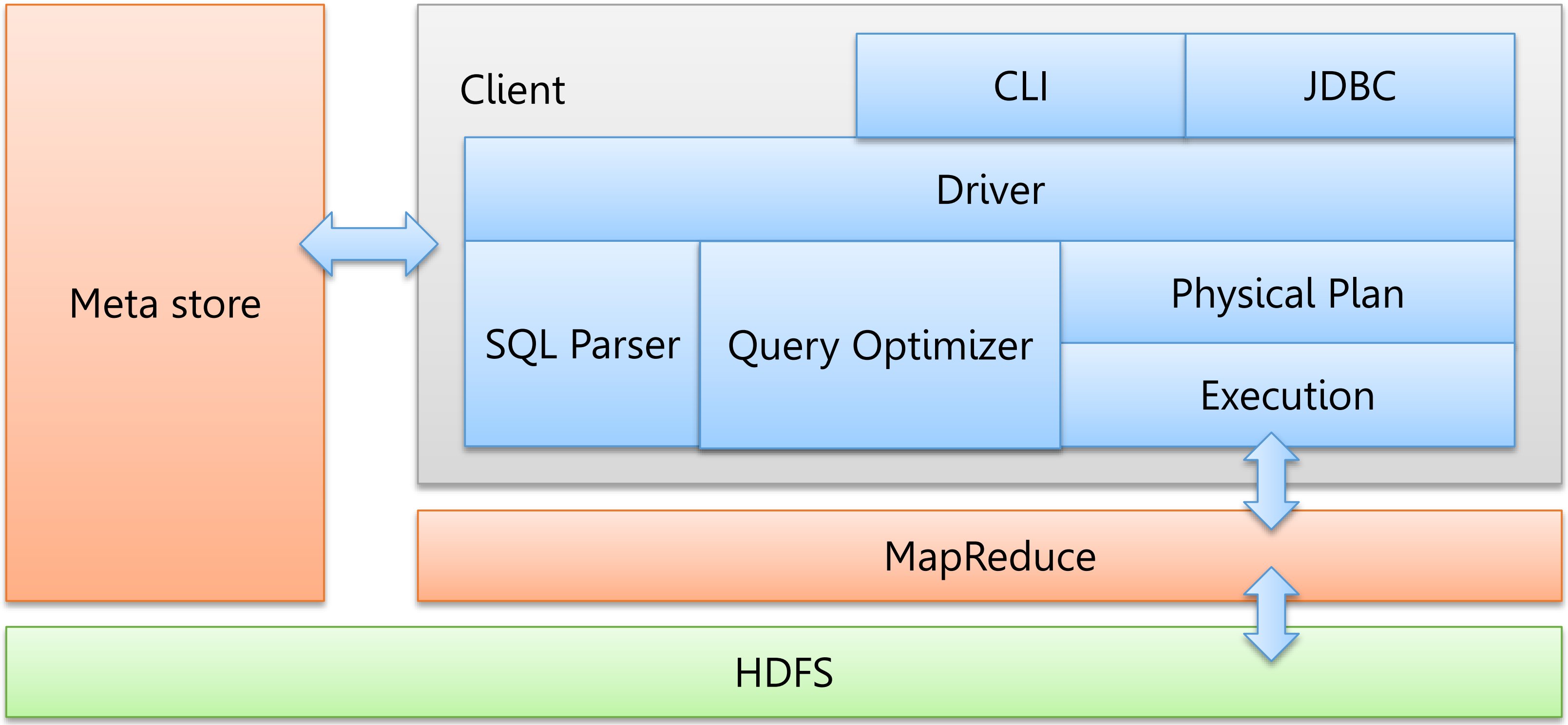
{JSON} Example:

```
#create s SchemaRDD from the file(s) pointed to by path

people = sqlContext.jsonFile(path)

#visualized inferred schema with prontSchema()
People.printSchema()
#root
# / -- age : integer
# / -- name : string

# Register this schemaRDD as a table
People.register.TempTable(“people”)
```

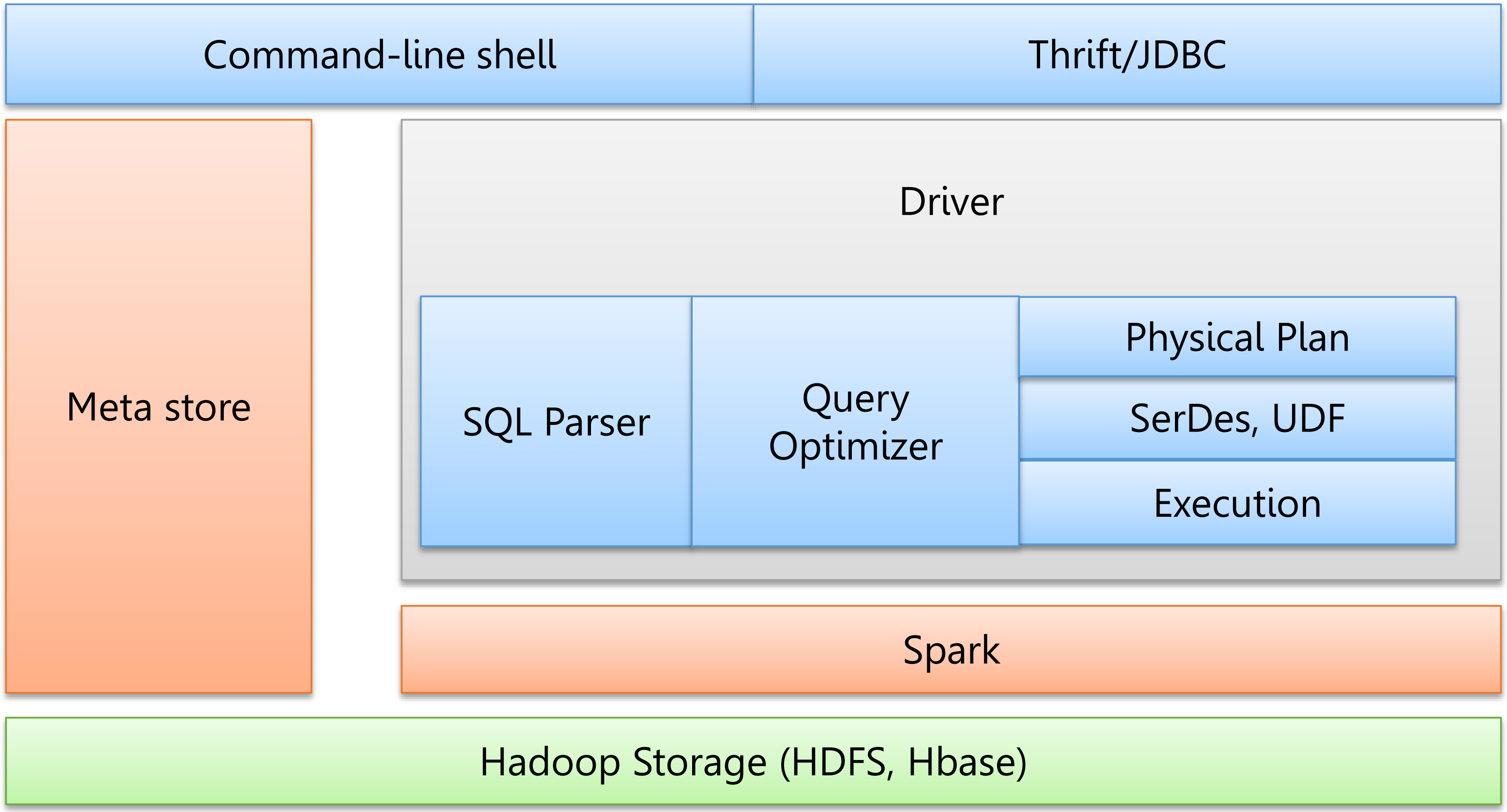
Hive in Hadoop

- Hive is a popular Hadoop Ecosystem component for ad-hoc analysis of large structured datasets
- Hive exposes a query language called HiveQL, which is quite similar to SQL and supports multiple other extensions
- Hive launches Map Reduce jobs internally for executing the ad-hoc queries
- While Map Reduce enables very fast analysis of huge datasets
- It lags behind in the performance when it comes to the analysis of the medium of just large datasets
- Map Reduce involvement means the response time being in the order of several seconds, which might not be favourable for some business cases

Hive Architecture



Spark Architecture



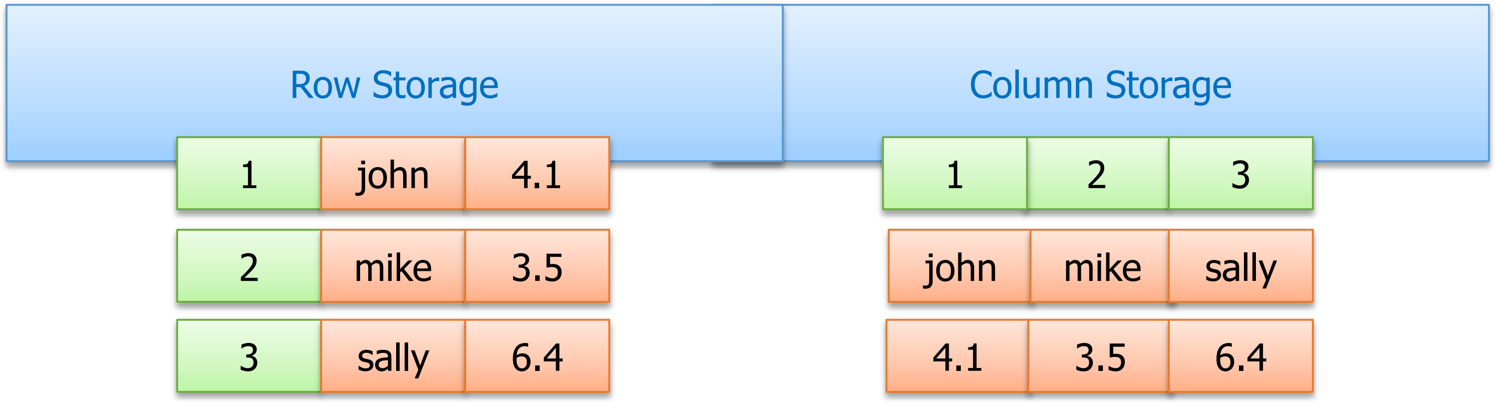
Efficient In-memory Storage

- Extremely fast scheduling
  - Milliseconds Vs. hours in Hadoop MR

- Support for general DAGs
  - Each query is a job rather than stages of jobs

- Many more useful primitives
  - Higher level APIs
  - Broadcast variables

Columnar Memory Store:
Instead, Shark employs column-oriented storage using arrays of primitive types



Efficient In-memory Storage (Cont'd)

- Simply caching Hive records as Java objects is inefficient due to high per-object overhead
- Instead, Shark employs column-oriented storage using arrays of primitive types

3
sally
6.4
4.1
3.5
6.4

Benefit: similarly compact size to serialized data, but >5x faster to access

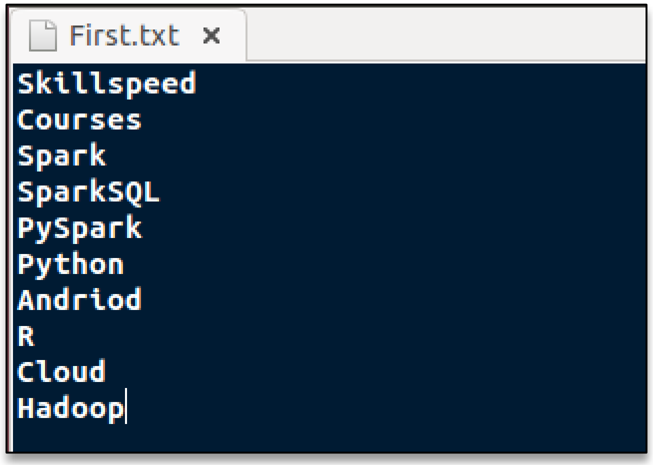
Loading of Data – Hive on Spark Example

We will run an sample Example of Hive on Spark. We will create a table, load data in that table and execute a simple query. When working with Hive, one must construct a HiveContext which inherits from SQLContext

```
skillspeed@ubuntuvms:~/spark-1.1.0/bin$sudo ./spark-shell
                                      (or)
skillspeed@ubuntuvms:~/spark-1.1.0/bin$sudo ./spark-shell
```

Create an input file "First.txt" in your home directory i.e.,

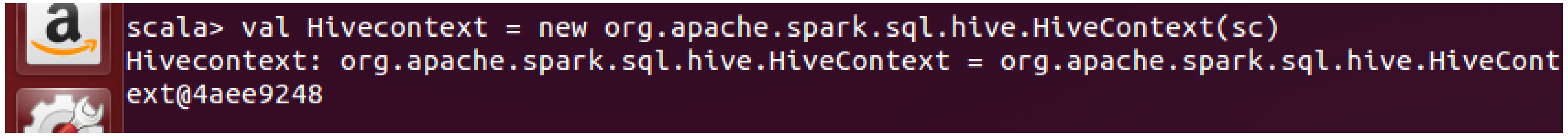
`home/skillspeed/First.txt`



```
Skillspeed
Courses
Spark
SparkSQL
PySpark
Python
Andriod
R
Cloud
Hadoop
```

Loading of Data – Hive on Spark Example (Cont'd)

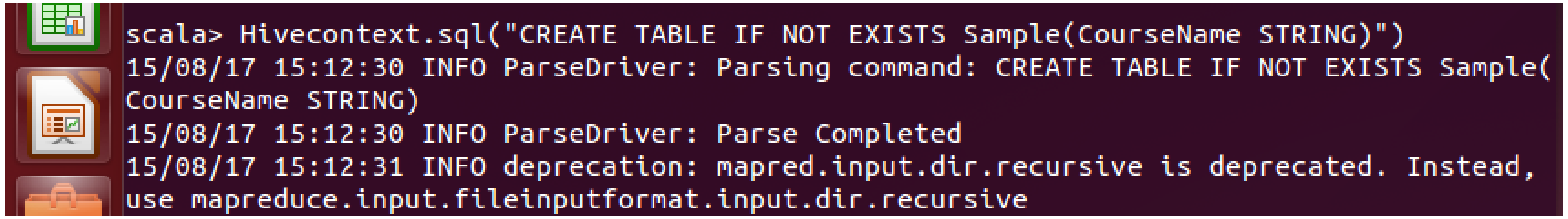
```
Scala > val Hivecontext = new org.apache.spark.sql.hive.HiveContext(sc)
```



```
scala> val Hivecontext = new org.apache.spark.sql.hive.HiveContext(sc)
Hivecontext: org.apache.spark.sql.hive.HiveContext = org.apache.spark.sql.hive.HiveCont
ext@4aee9248
```

Create the table Sample:

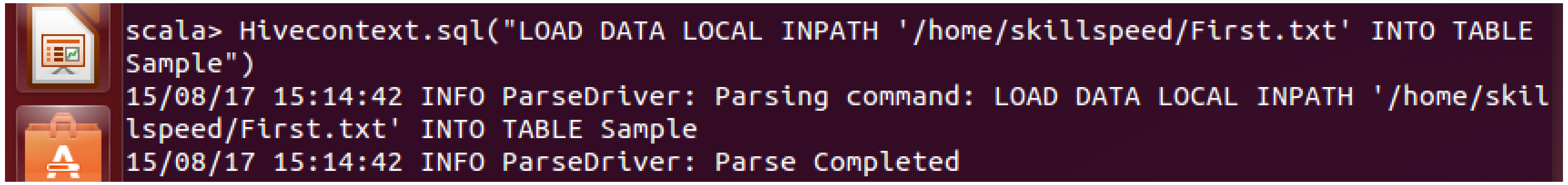
```
Scala > Hivecontext.sql("CREATE TABLE IF NOT EXISTS Sample(CourseName STRING)")
```



```
scala> Hivecontext.sql("CREATE TABLE IF NOT EXISTS Sample(CourseName STRING)")
15/08/17 15:12:30 INFO ParseDriver: Parsing command: CREATE TABLE IF NOT EXISTS Sample(
CourseName STRING)
15/08/17 15:12:30 INFO ParseDriver: Parse Completed
15/08/17 15:12:31 INFO deprecation: mapred.input.dir.recursive is deprecated. Instead,
use mapreduce.input.fileinputformat.input.dir.recursive
```

Loading the data:

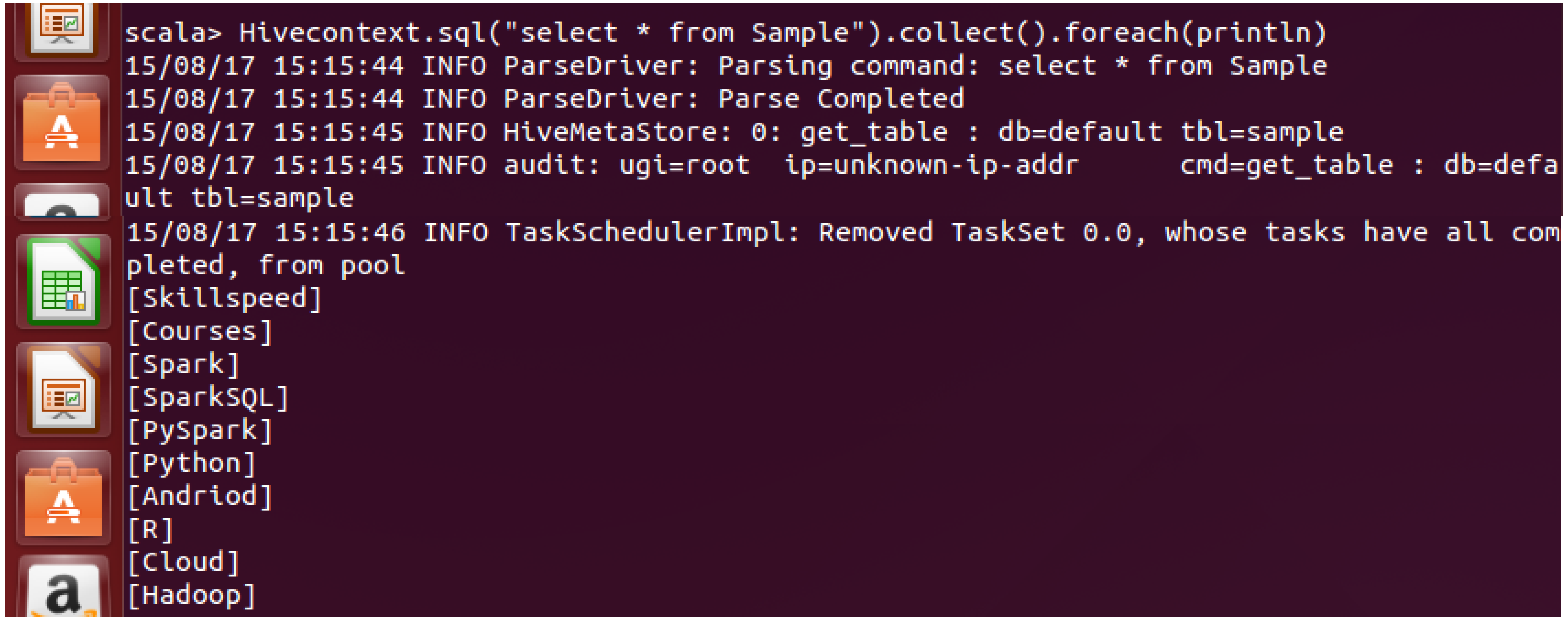
```
Scala > Hivecontext.sql("LOAD DATA LOCAL INPATH 'home/skillspeed/First.txt' INTO TABLE
Sample ")
```



```
scala> Hivecontext.sql("LOAD DATA LOCAL INPATH '/home/skillspeed/First.txt' INTO TABLE
Sample")
15/08/17 15:14:42 INFO ParseDriver: Parsing command: LOAD DATA LOCAL INPATH '/home/skil
lspeed/First.txt' INTO TABLE Sample
15/08/17 15:14:42 INFO ParseDriver: Parse Completed
```

Executing the Query

```
Scala> Hivecontext.sql(“select * from Sample”).collect().foreach(println)
```



```
scala> Hivecontext.sql("select * from Sample").collect().foreach(println)
15/08/17 15:15:44 INFO ParseDriver: Parsing command: select * from Sample
15/08/17 15:15:44 INFO ParseDriver: Parse Completed
15/08/17 15:15:45 INFO HiveMetaStore: 0: get_table : db=default tbl=sample
15/08/17 15:15:45 INFO audit: ugi=root  ip=unknown-ip-addr      cmd=get_table : db=defa
ult tbl=sample
15/08/17 15:15:46 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all com
pleted, from pool
[Skillspeed]
[Courses]
[Spark]
[SparkSQL]
[PySpark]
[Python]
[Andriod]
[R]
[Cloud]
[Hadoop]
```

Spark DataFrame 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
- The DataFrame API is available in Scala, Java, and Python

Starting Point: SQLContext

The entry point into all functionality in Spark SQL is the SQLContext class, or one of its descendants. To create a basic SQLContext, all you need is a SparkContext

```
val sc: SparkContext // An existing SparkContext.
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

// this is used to implicitly convert an RDD to a DataFrame.
import sqlContext.implicits._
```

Creating DataFrames

- With a SQLContext, applications can create DataFrames from an existing RDD, from a Hive table, or from data sources
- As an example, the following creates a DataFrame based on the content of a JSON file:

```
val sc: SparkContext // An existing SparkContext.
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val df = sqlContext.jsonFile("examples/src/main/resources/people.json")

// Displays the content of the DataFrame to stdout
df.show()
```

Questions

skillspeed

for the serious learner

thank you!