

Apache Spark and Scala

Module 4: Shark and SparkSQL

Module 1

Getting Started /
Introduction to Scala

Module 2

RDD and Spark
Streaming

Module 3

Scala Basics

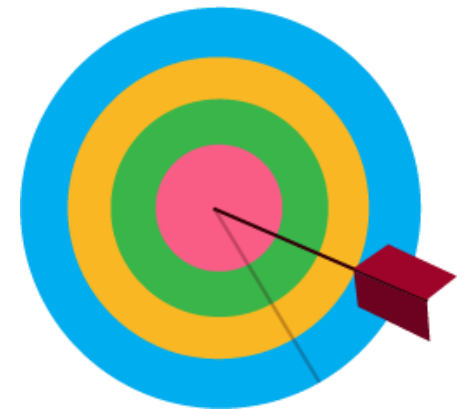
Module 4

SparkSQL – Real-time
Analysis

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 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

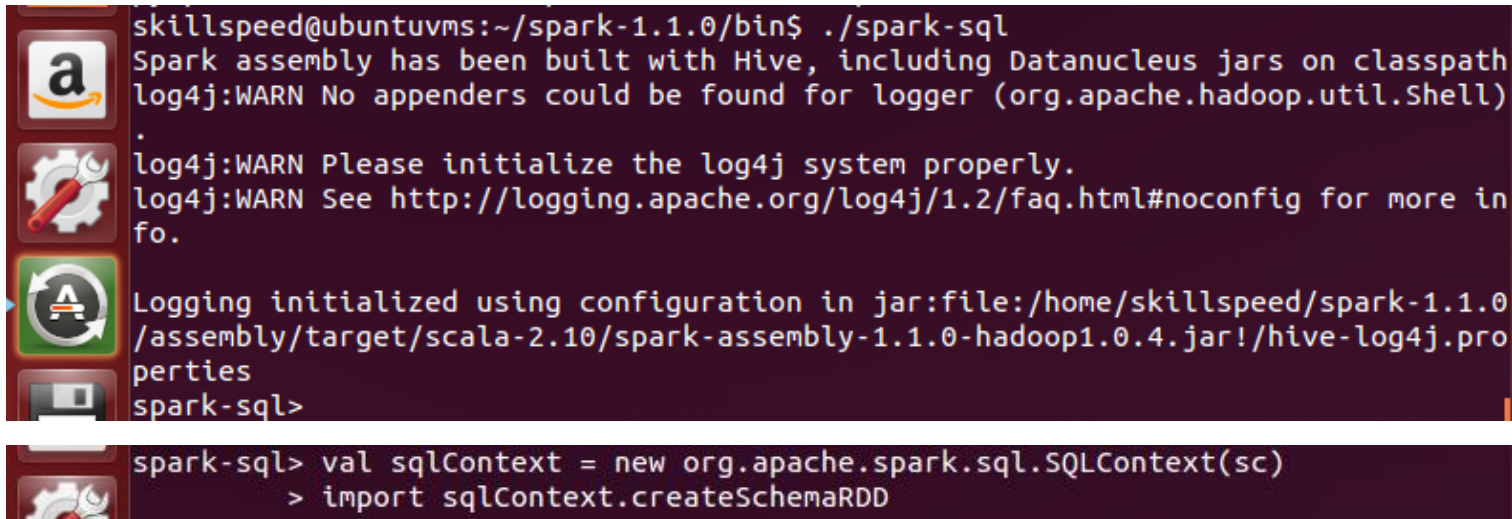
WHY Shark to Spark SQL?

- ▶ 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 made 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:

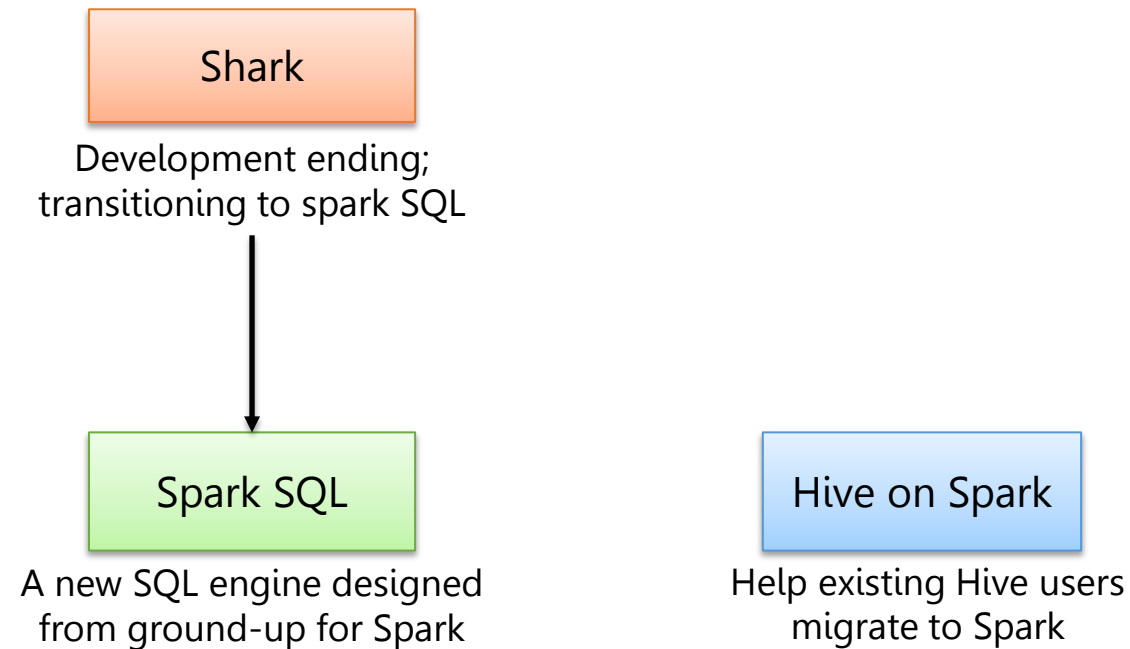


```
skillspeed@ubuntuvm:~/spark-1.1.0/bin$ ./spark-sql
Spark assembly has been built with Hive, including Datanucleus jars on classpath
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell)
.
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig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



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)
```

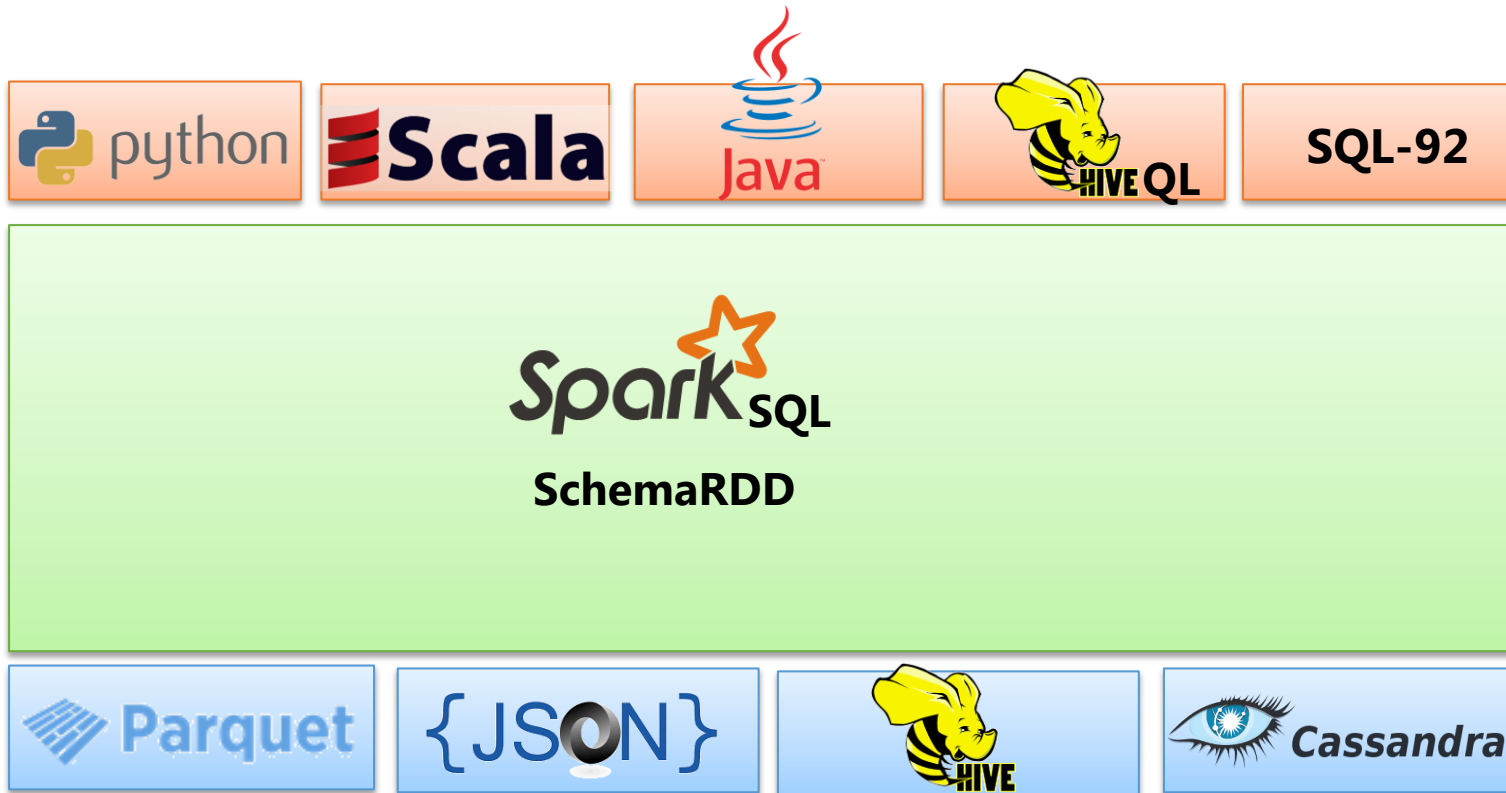
➤ HiveContext:

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

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

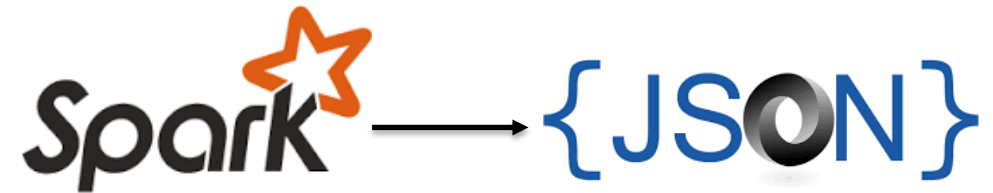
- Query data from HDFS/S3,
- Including formats like Hive/Parquet/JSON
- Support for caching data in-memory in Spark 1.2

Unified Data Abstraction



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 printSchema()
People.printSchema()
#root
# / -- age : integer
# / -- name : string

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

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()
```



thank
you!