

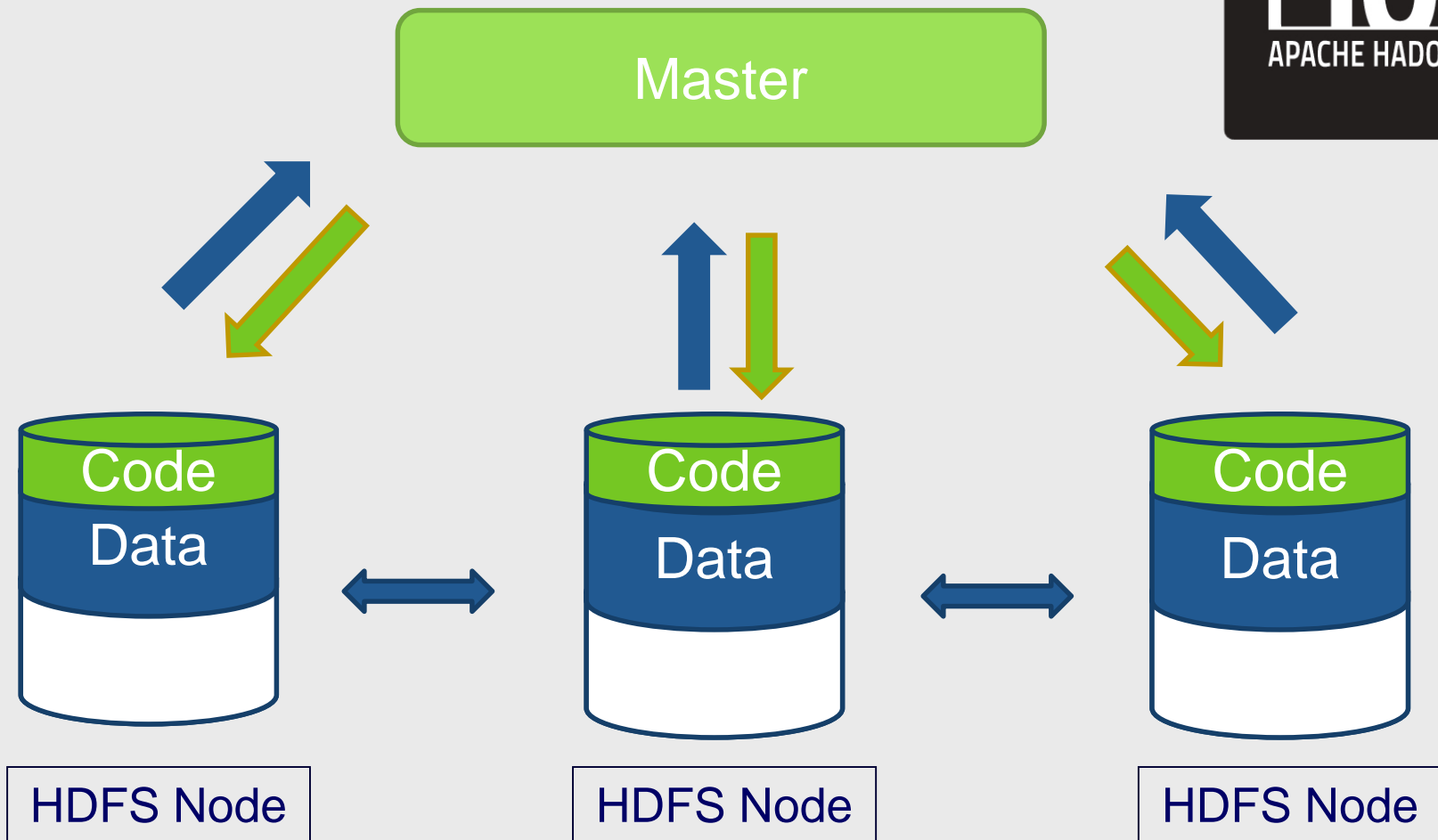


The Big Connection - R and Big Data

Bence Arató

arato@biconsulting.hu

rstats.budapestbi.hu



- SQL on Hadoop – why?
 - Familiar interface for most users
 - BI tools (like Tableau, Power BI etc) also uses SQL to connect
- Many different engines
 - Hive, Impala, Drill, Presto, ...
 - Most offers ODBC/JDBC driver, usable from R

Hadoop Apache Project Commercial Support Tracker December 2017					
	Amazon EMR 5.11	Cloudera CDH 5.13	Google	Hortonworks HDP 2.6.2	MapR MEP 3.0.1
Total supported projects					
Apache HDFS	2.7.3	2.6.0	2.8.1	2.7.3	API
Apache Mapreduce	2.7.3	2.6.0	2.8.1	2.7.3	2.7.0+
Apache YARN	2.7.3	2.6.0	2.8.1	2.7.3	2.7.0+
Apache Hive	2.3.2	1.2	2.1.1	2.1.0	2.1.1
Apache Pig	0.17	0.12.0	0.16.0	0.16.0	0.16
Apache Spark	2.2.1	2.2	2.2.0	2.1.1	2.1.0
Apache Avro	X	1.7.6	1.8.2	1.7.5	1.7.4
Apache Flume	X	1.7.0	1.7.0	1.5.2	1.7
Apache HBase	1.3.1 +S3	1.2	1.3.1	1.1.2	1.1.8
Apache Kafka	X	0.11	0.11.0.1	0.10.1.2	0.9 (Streams)
Apache Oozie	4.3.0	4.1.0	4.3.0	4.2.0	4.3.0
Apache Parquet	X	1.51	1.9.0	1.8.1	1.8.1
Apache Sqoop	1.4.6	1.4.8	1.99.4	1.4.6	1.4.6
Apache Zookeeper	3.4.10	3.4.5	3.4.6	3.4.6	3.4.5
Hue	4.0.1	4.00	3.11.0	2.6.1	3.12

Hadoop distributions may have a preferred engine (for example Hortonworks -> Hive, Cloudera -> Impala)

- Steps for trying Impala-based access
 - Download the Cloudera QuickStart VM
 - Download the Impala ODBC driver

cloudera PRODUCTS SOLUTIONS DOWNLOADS MORE

QuickStarts for CDH 5.12

Virtualized clusters for easy installation on your desktop!

Cloudera QuickStart VMs (single-node cluster) make it easy to quickly get hands-on with CDH for testing, demo, and self-learning purposes, and include Cloudera Manager for managing your cluster. Cloudera QuickStart VM also includes a tutorial, sample data, and scripts for getting started.

Get Started Now

Platform: Virtual Box

[GET IT NOW →](#)

** Cloudera QuickStarts, deployed via Docker containers or VMs, are not intended or supported for use in production. **

** Cloudera provides the latest release of its software as a free download as Quickstart VMs. Older versions will not be available for download. The current release is CDH 5.12. **

System Requirements | Installed Products | Getting Started

Cloudera QuickStart virtual machines (VMs) include everything you need to try CDH, Cloudera Manager, Cloudera Impala, and Cloudera Search. The VM uses a package-based install. This allows you to work with or without Cloudera Manager. Parcels do not work with the VM unless you first migrate your CDH installation to use parcels. On your production systems, Cloudera recommends that you use parcels.

cloudera Products Solutions Downloads More

Impala ODBC Connector 2.5.41 for Cloudera Enterprise

Get Started

Version: Impala ODBC Connector 2.5.41

Operating System: Windows OS Version: 64 bit

[GET IT NOW →](#)

The Cloudera ODBC Driver for Impala enables your enterprise users to access Hadoop data through Business Intelligence (BI) applications with ODBC support.

The driver achieves this by translating Open Database Connectivity (ODBC) calls from the application into SQL and passing the SQL queries to the underlying Impala engine.

Thank you for downloading the Impala ODBC Connector for Cloudera Enterprise

Your download should begin shortly. Please [Click Here](#) if it does not start automatically.

[Connector Documentation](#)
Guides describing how to install and use Cloudera connectors.

- The ODBC R Package
 - DBI interface for ODBC
 - Maintained by Rstudio



The screenshot shows a web browser window displaying the RStudio website page for the ODBC R package. The browser address bar shows the URL `https://db.rstudio.com/odbc/`. The page title is "Databases using R from R Studio". The left sidebar contains a navigation menu with categories: Packages, RStudio, and Best Practices. Under "Packages", the "odbc" package is selected. The main content area displays the package name "odbc" and a description: "The goal of the 'odbc' package is to provide a DBI-compliant interface to [Open Database Connectivity](#) (ODBC) drivers. This allows for an efficient, easy to setup connection to any database with ODBC drivers available, including [SQL Server](#), [Oracle](#), [MySQL](#), [PostgreSQL](#), [SQLite](#) and others. The implementation builds on the [nanodbc](#) C++ library." Below the description is a section titled "Usage" with a paragraph: "All of the following examples assume you have already created a connection `con`. See [Connecting to a database](#) for more information on establishing a connection." A section titled "TABLE AND FIELD INFORMATION" follows, with a paragraph: "`dbListTables()` is used for listing all existing tables in a database." Below this is a code block containing R code examples:

```
dbListTables(con)

# List tables beginning with f
dbListTables(con, table_name = "f%")

# List all fields in the 'flights' database
dbListFields(con, "flights")
```

```
library(odbc)
library(DBI)

drv <- odbc::odbc()

con <- dbConnect(drv,
                 driver = "Cloudera ODBC Driver for Impala",
                 host = "localhost",
                 port = 21050,
                 database = "default",
                 uid = "",
                 pwd = ""
)

#list available tables
dbListTables(con)
dbListTables(con, table_name = "%port%")
```

```
# list fields
```

```
dbListFields(con, "airports")
```

```
# Load all data from SQL into a local data.frame
```

```
df_airports <- dbReadTable(con, "airports")
```

```
str(df_airports)
```

```
# Read data using an SQL query
```

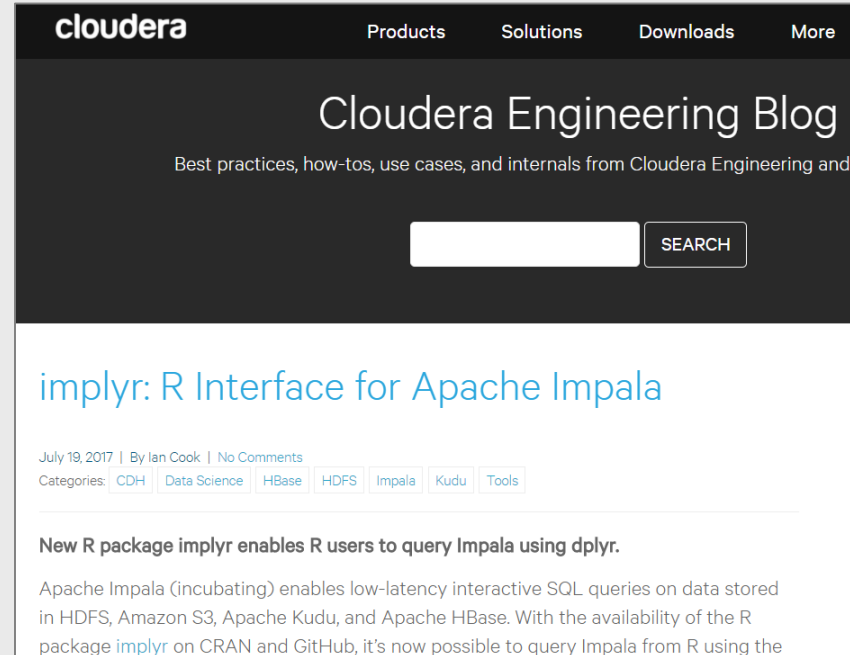
```
query_results= dbGetQuery(con, "select * from airports where faa='SFO'")
```

```
head(query_results)
```

```
query_results= dbGetQuery(con, "select * from airports where name like '%London%'")
```

```
head(query_results)
```


- The implyr package
 - dplyr SQL backend for Impala
 - Developed by Cloudera, Ian Cook
 - Uses the ODBC connector for data access



```
# packages
install.packages("implyr")
library(odbc)
library(implyr)
library(dplyr)

drv <- odbc::odbc()

impala <- src_impala(
  drv = drv,
  driver = "Cloudera ODBC Driver for Impala",
  host = "host",
  port = 21050,
  database = "default",
  uid = "username",
  pwd = "password"
)

# list available tables
src_tbls(impala)
```

```
# create airports reference
airports_tbl <- tbl(impala, "airports")
airports_tbl

# Running SQL - refresh Impala metadata
dbExecute(impala, "refresh airports")

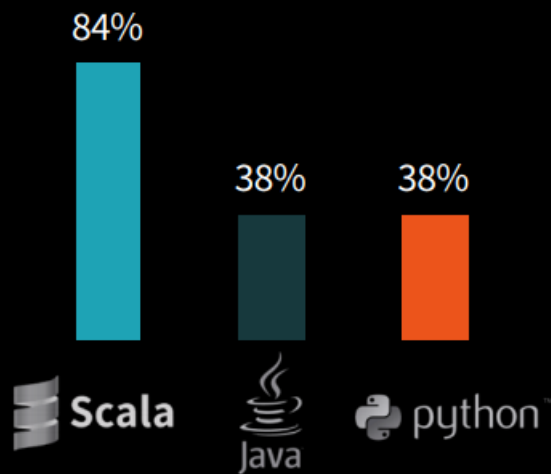
# Query data using SQL
airport_cnt <- dbGetQuery( impala, "select count(*) from airports")
airport_cnt

airport_sfo <- dbGetQuery( impala, "select * from airports where faa='SFO'")
head(airport_sfo)

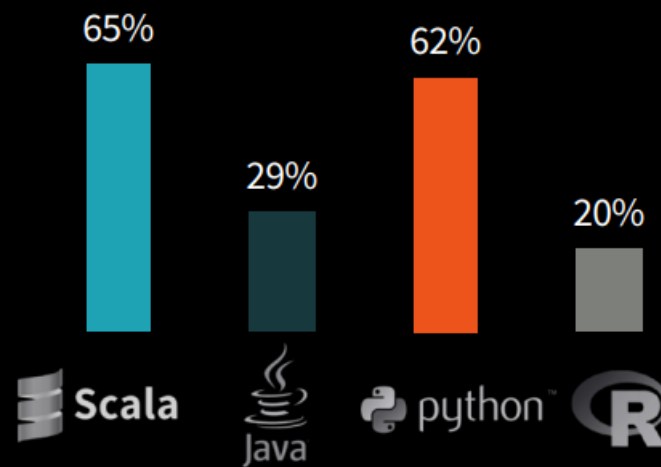
# same using dplyr
airports_tbl %>% filter( FAA == "SFO")
```

Languages Used for Spark

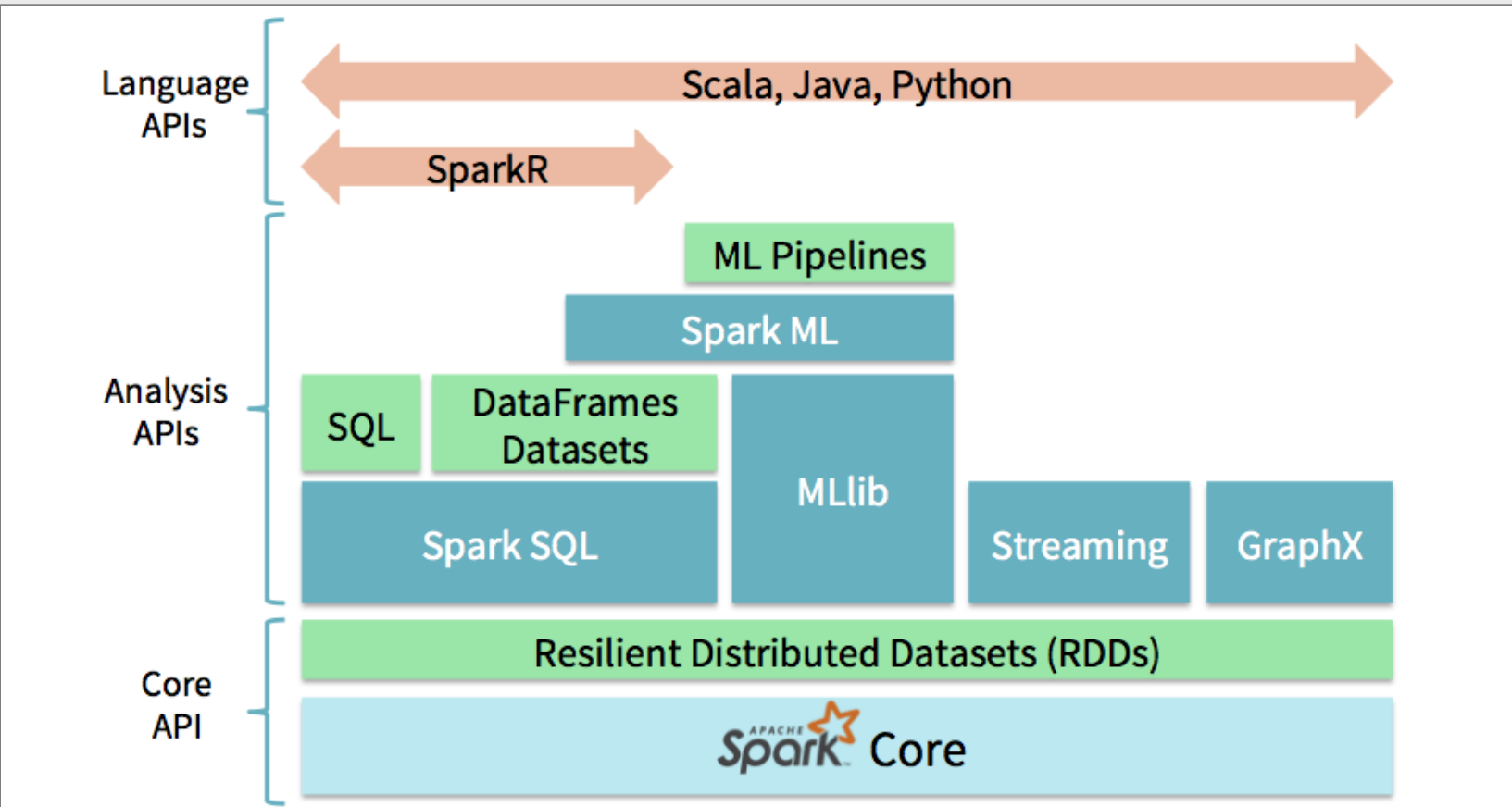
2014 Languages Used



2016 Languages Used



+ Everyone uses SQL (95%)



- R access for Spark (examples)
 - SparkR (Databricks)
 - RevoScaler (Microsoft)
 - sparklyr (Rstudio)

The screenshot shows the Databricks documentation page for SparkR. The page title is "SparkR Overview". It describes SparkR as a package that provides a light-weight frontend to use Apache Spark from R. The page lists several topics to be covered in the overview, including creating SparkR DataFrames, using Spark SQL, and machine learning. A note at the bottom states: "Databricks also supports sparklyr, please see the documentation for more information."

The screenshot shows the sparklyr website, which is the R interface for Apache Spark. The page title is "sparklyr: R interface for Apache Spark". It features a navigation menu on the left with categories like "Using sparklyr", "Guides", "Deployment Examples", and "Reference". The main content area includes a diagram showing sparklyr connecting to dplyr, ML, and Extensions, all of which interface with Apache Spark. Below the diagram, there are installation instructions for CRAN and a local version of Spark, along with a command to upgrade to the latest version of sparklyr from GitHub.

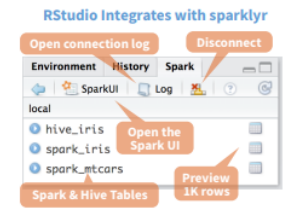
Data Science in Spark with Sparklyr : : CHEAT SHEET



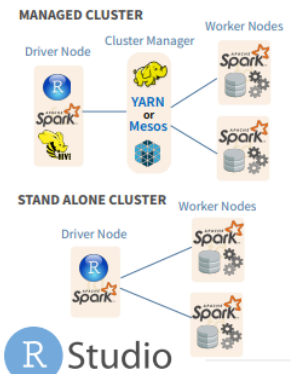
Intro

sparklyr is an R interface for Apache Spark™, it provides a complete **dplyr** backend and the option to query directly using **Spark SQL** statement. With sparklyr, you can orchestrate distributed machine learning using either **Spark's MLib** or **H2O Sparkling Water**.

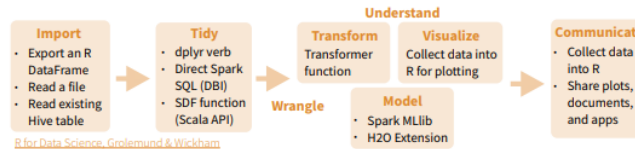
Starting with **version 1.044**, **RStudio Desktop, Server and Pro** include integrated support for the **sparklyr** package. You can create and manage connections to Spark clusters and local Spark instances from inside the IDE.



Cluster Deployment



Data Science Toolchain with Spark + sparklyr



Getting Started

LOCAL MODE (No cluster required)

1. Install a local version of Spark:
`spark_install("2.0.1")`
2. Open a connection
`sc <- spark_connect(master = "local")`

ON A YARN MANAGED CLUSTER

1. Install RStudio Server or RStudio Pro on one of the existing nodes, preferably an edge node
2. Locate path to the cluster's Spark Home Directory, it normally is `/usr/lib/spark`
3. Open a connection
`spark_connect(master="yarn-client", version = "1.6.2", spark_home = [Cluster's Spark path])`

ON A MESOS MANAGED CLUSTER

1. Install RStudio Server or Pro on one of the existing nodes
2. Locate path to the cluster's Spark directory
3. Open a connection
`spark_connect(master="[mesos URL]", version = "1.6.2", spark_home = [Cluster's Spark path])`

USING LIVY (Experimental)

1. The Livy REST application should be running on the cluster
2. Connect to the cluster
`sc <- spark_connect(method = "livy", master = "http://host:port")`

Tuning Spark

EXAMPLE CONFIGURATION

```
config <- spark_config()
config$spark.executor.cores <- 2
config$spark.executor.memory <- "4G"
sc <- spark_connect(master="yarn-client",
config = config, version = "2.0.1")
```

IMPORTANT TUNING PARAMETERS with defaults

- spark.yarn.am.cores
- spark.yarn.am.memory 512m
- spark.network.timeout 120s
- spark.executor.memory 1g
- spark.executor.cores 1
- spark.executor.instances
- spark.executor.extraJavaOptions
- spark.executor.heartbeatInterval 10s
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

Using sparklyr

A brief example of a data analysis using Apache Spark, R and sparklyr in local mode

```
library(sparklyr); library(dplyr); library(ggplot2);
library(tidyrr);
set.seed(100)
```

```
spark_install("2.0.1")
sc <- spark_connect(master = "local")
```

```
import_iris <- copy_to(sc, iris, "spark_iris",
overwrite = TRUE)
```

```
partition_iris <- sdf_partition(
import_iris, training=0.5, testing=0.5)
```

```
sdf_register(partition_iris,
c("spark_iris_training", "spark_iris_test"))
```

Create a hive metadata for each partition

```
tidy_iris <- tbl(sc, "spark_iris_training") %>%
select(Species, Petal_Length, Petal_Width)
```

```
model_iris <- tidy_iris %>%
ml_decision_tree(response="Species",
features=c("Petal_Length", "Petal_Width"))
```

```
test_iris <- tbl(sc, "spark_iris_test")
```

```
pred_iris <- sdf_predict(
model_iris, test_iris) %>%
collect
```

```
pred_iris %>%
inner_join(data.frame(prediction=0.2,
lab=model_iris$model.parameters$labels)) %>%
ggplot(aes(Petal_Length, Petal_Width, col=lab)) +
geom_point()
```

```
spark_disconnect(sc)
```