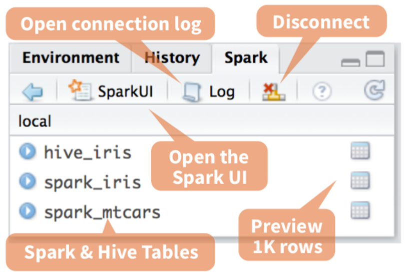
Data Science in Spark with Sparklyr : : CHEAT SHEET

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 MLlib** or **H2O** Sparkling Water.

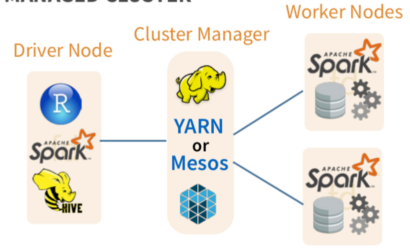
Starting with **version 1.044, RStudio Desktop, Server and Pro include integrated support for the sparklyr package**. You can create and manage connections to Spark clusters and local Spark instances from inside the IDE.

RStudio Integrates with sparklyr

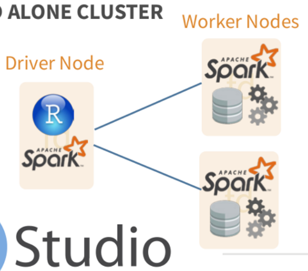


Cluster Deployment

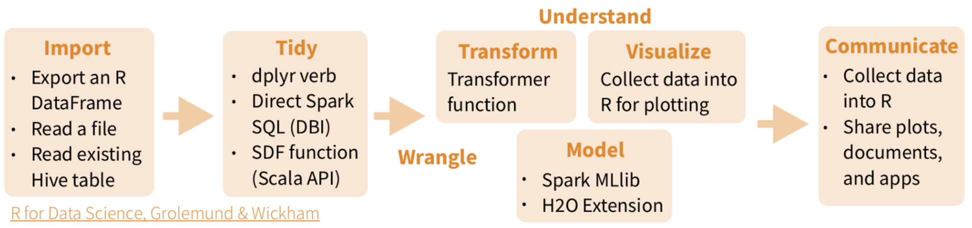
MANAGED CLUSTER



STAND ALONE CLUSTER



Data Science Toolchain with Spark + sparklyr



R for Data Science, Grolemund & Wickham

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 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")

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 SPARK STANDALONE CLUSTER

1. Install RStudio Server or RStudio Pro on one of the existing nodes or a server in the same LAN
2. Install a local version of Spark:
   spark_install (version = "2.0.1")
3. Open a connection
   spark_connect(master="spark://host:port", version = "2.0.1", spark_home = spark_home_dir())

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(tidyr);
set.seed(100)
```

Install Spark locally

```
spark_install("2.0.1")
```

Connect to local version

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

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

Copy data to Spark memory

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

Partition data

```
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"))
```

Spark ML Decision Tree Model

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

Create reference to Spark table

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

Bring data back into R memory for plotting

```
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)
```

Disconnect