Data Science @ Scale

Wee Hyong Tok

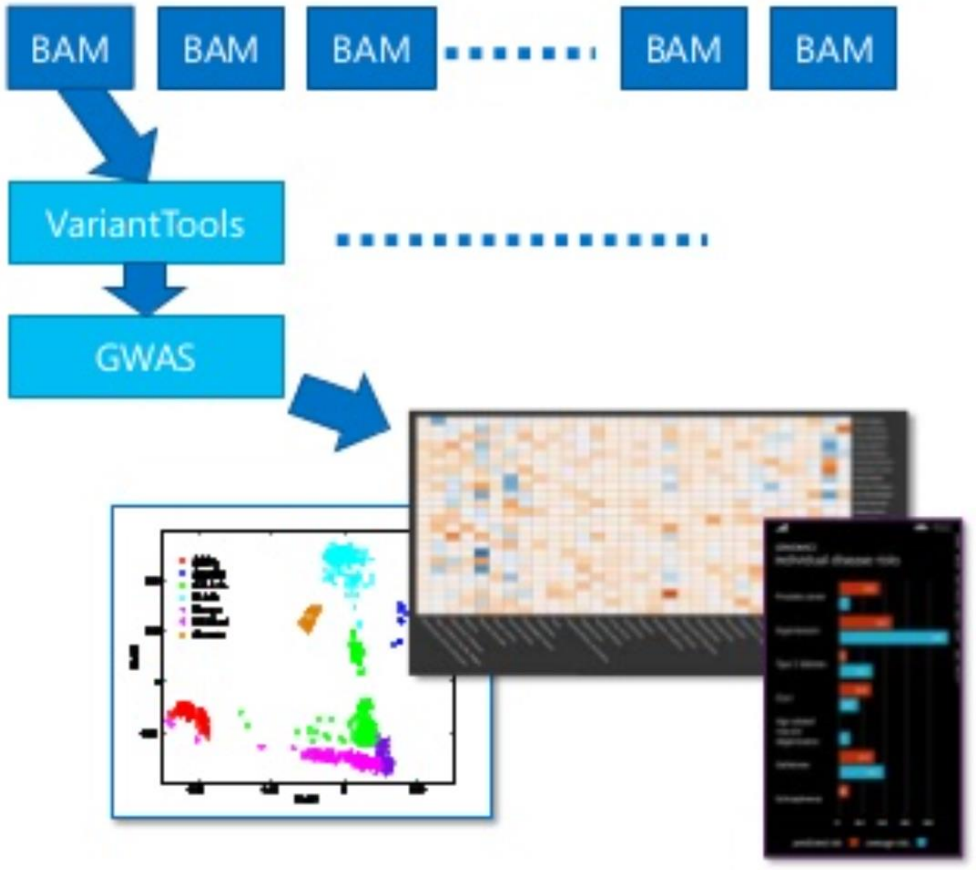
18 Aug 2016

Challenges of Data Science @ Scale

- Limitation of available functions to handle large dataset
- Hardware/infrastructure
- Operationalized pipelines

Example – Genomic Data

Processing Genomic Data



Data
- Public genome data from 1000 Genomes
- About 2TB of raw data

Platform
- HDInsight Hadoop (8 clusters)
  - 1500 cores, 4 data centers
- Microsoft R Server

Processing
- VariantTools R package (Bioconductor)
- Match against NHGRI GWAS catalog

Analytics
- Disease Risk
- Ancestry

Presentation
- Expose as Web Service APIs
- Phone app, Web page, Enterprise applications

Src: Revolution Analytics

Slide 6- 39

Credits: KDD 2016 : John-Mark Agosta, Hang Zhang, Robert Horton, Mario Inchiosa, Srini Kumar, Mengyue (Katherine) Zhao, Vanja Paunic and Debraj GuhaThakurta

Scalable Data Analytics Using R: Single Machines to Spark Clusters

John-Mark Agosta, Hang Zhang, Robert Horton, Mario Inchiosa, Srini Kumar, Mengyue (Katherine) Zhao, Vanja Paunic and Debraj GuhaThakurta

Microsoft

TUTORIAL MATERIAL & SLIDES:

tinyurl.com/KDD2016R

ROOM: Embarcadero
Parc 55 San Francisco,
55 Cyril Magnin St, San Francisco, CA
TIME: 9:00am - 12:00pm
August 17th, 2016, KDD 2016, San Francisco

Tutorial Outline

- Introduction - scaling your R scripts (issues and solutions)
- Hands-on tutorial (end-to-end scalable data analysis in R)
- Practical examples and case studies
- Q&A

Introduction

- What is R?
- What limits the scalability of R scripts?
- What functions and techniques can be used to overcome those limits?
- How do the base and scalable approaches compare?

What is

Language Platform

- The most popular statistical programming language
- A data visualization tool
- Open source

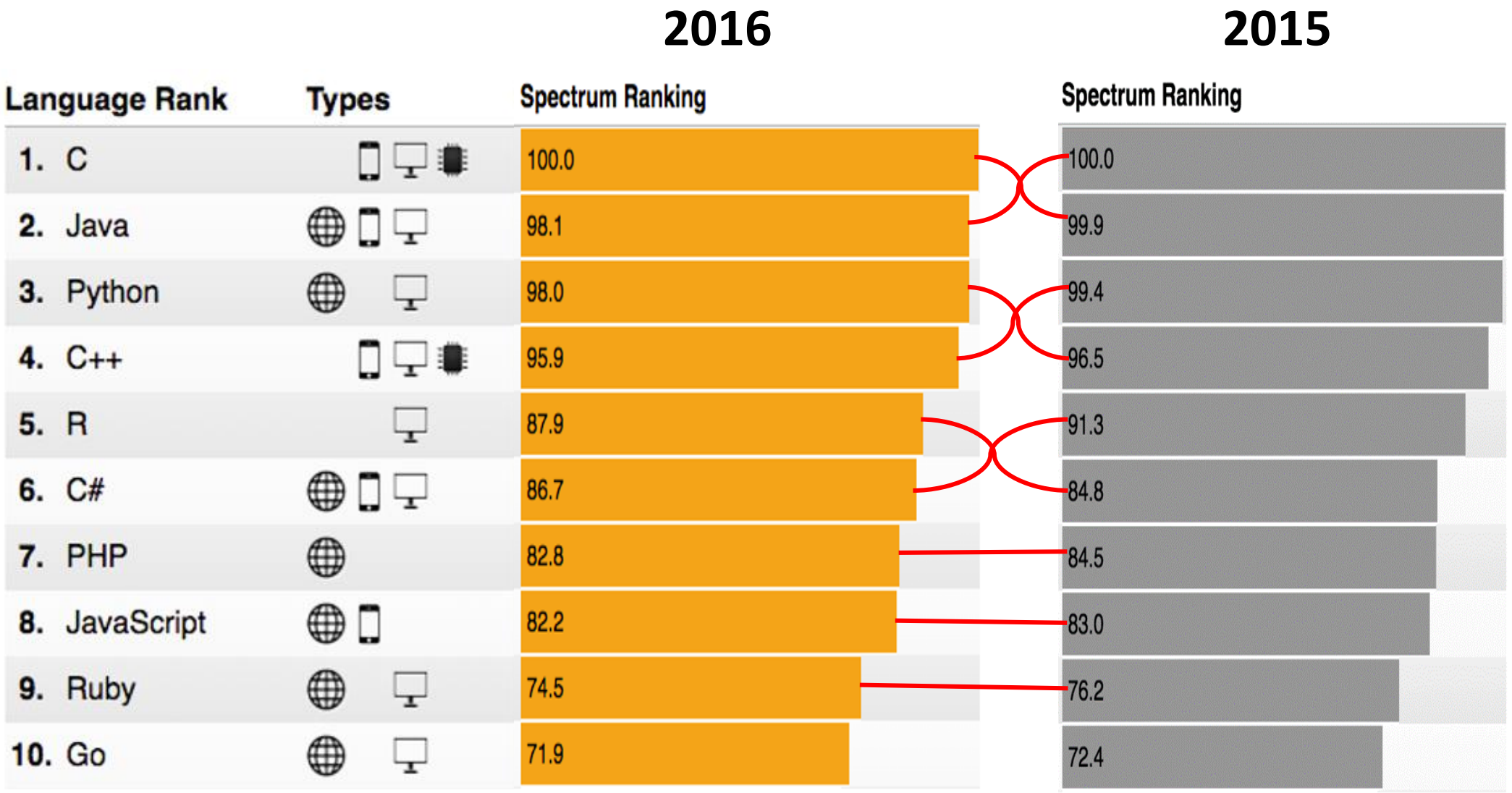
Community

- 2.5+M users
- Taught in most universities
- New and recent grad’s use it
- Thriving user groups worldwide

Ecosystem

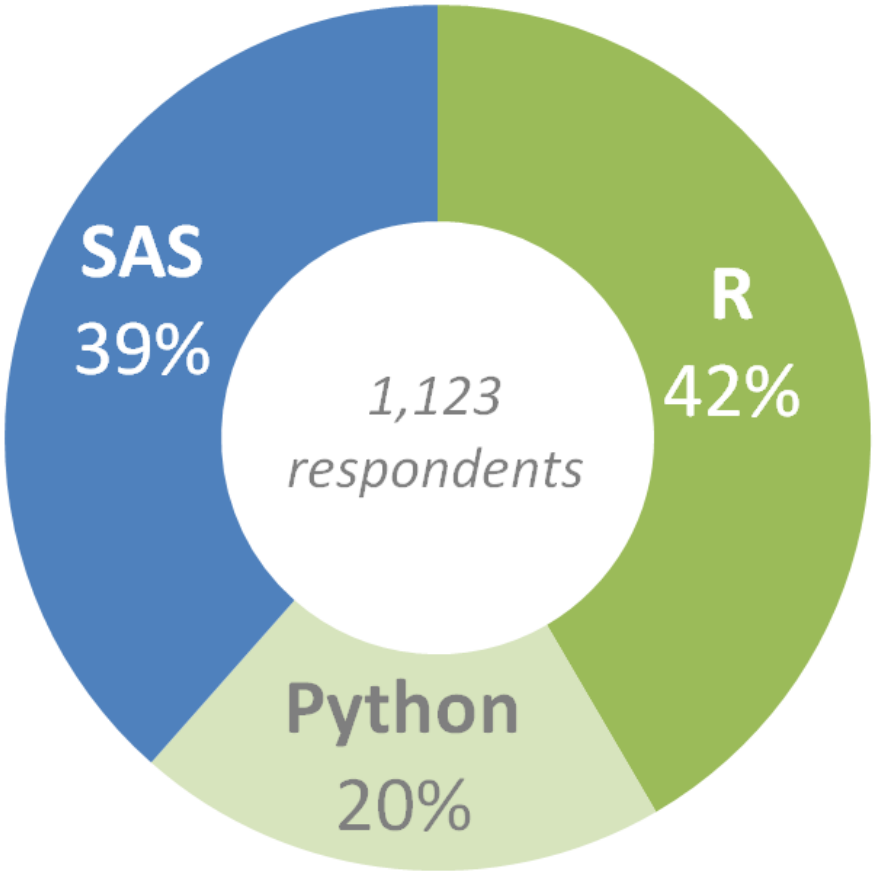
- 8000+ contributed packages
- Rich application & platform integration

R ranks #5 in IEEE Spectrum July 2016



Preferred language by Analytics Pros

Which do you prefer to use: SAS, R, or Python?



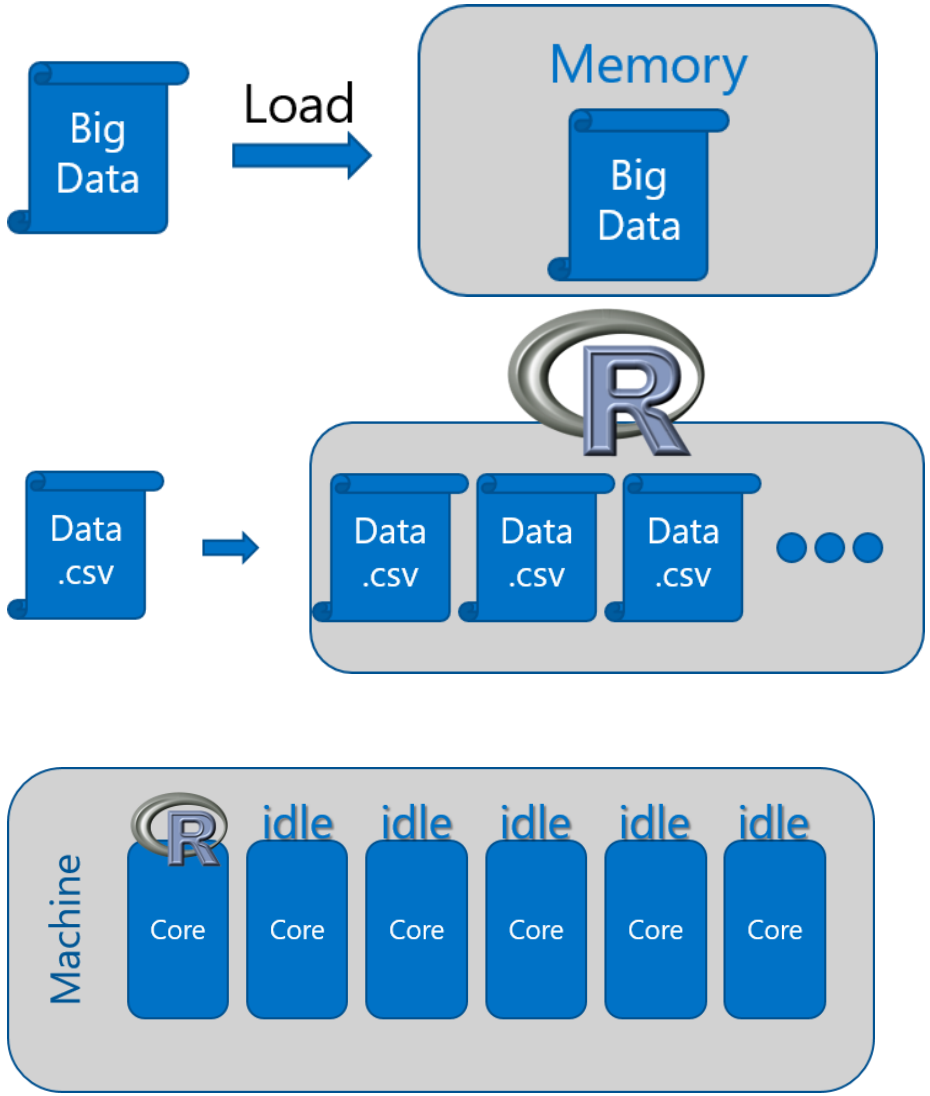
Source: SAS, R or Python Survey 2016, by Burtch Works

Common R use cases

Vertical	Sales & Marketing	Finance & Risk	Customer & Channel	Operations & Workforce
Retail	Demand Forecasting Loyalty Programs Cross-sell & Upsell Customer Acquisition	Fraud Detection Pricing Strategy	Personalization Lifetime Customer Value Product Segmentation	Store Location Demographics Supply Chain Management Inventory Management
Financial Services	Customer Churn Loyalty Programs Cross-sell & Upsell Customer Acquisition	Fraud Detection Risk& Compliance Loan Defaults	Personalization Lifetime Customer Value	Call Center Optimization Pay for Performance
Healthcare	Marketing Mix Optimization Patient Acquisition	Fraud Detection Bill Collection	Population Health Patient Demographics	Operational Efficiency Pay for Performance
Manufacturing	Demand Forecasting Marketing mix Optimization	Pricing Strategy Perf Risk Management	Supply Chain Optimization Personalization	Remote Monitoring Predictive Maintenance Asset Management

R adoption is on a Tear

- In-Memory Operation
- Expensive Data Movement & Duplication
- Lack of Parallelism



Open source R is not enterprise class

Inadequacy of Community Support

Lack of Guaranteed Support Timeliness

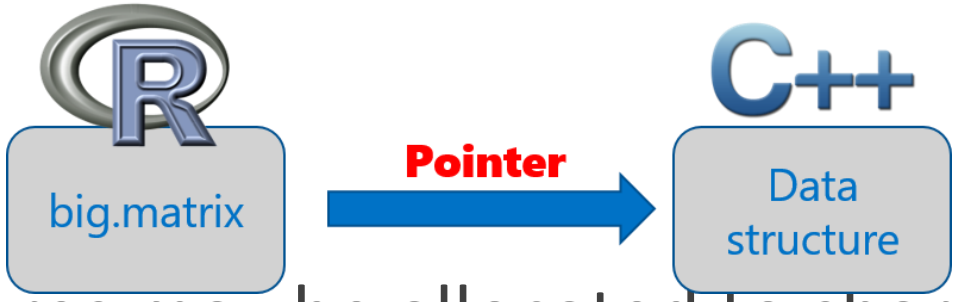
No SLAs or Support Models

Couple of scalable R solutions

- Choose R packages with big data support on single machines
  - The “bigmemory” project
  - “ff” and related packages
- Scale from single machines to distributed computing
  - SparkR
  - sparklyr
  - RevoScaleR (Microsoft R Server)
  - and more!

The bigmemory project

- Coined by Michael Kane and John Emerson at Yale University
- “bigmemory” works with massive matrix-like objects in R
- Combines memory and file-backed data structures: analyze numerical data larger than RAM



- The data structures may be allocated to shared memory

Source: “The Bigmemory Project” by Michael Kane and John Emerson: April 29, 2010.

Brings a small object into memory

- Imports a 20 million rows * 26 columns numerical matrix

```
library("bigmemory")
# Read airline data into a big matrix object (20 million rows * 26 columns)
backing.file    <- "airline_big.bin"
descriptor.file <- "airline_big.desc"
system.time(airline_big <- read.big.matrix("airline_20MM.csv",
                                           header = TRUE,
                                           type = "integer",
                                           sep = ",",
                                           backingfile = backing.file,
                                           descriptorfile = descriptor.file,
                                           shared = TRUE))
```

- big.matrix object is much smaller than its corresponding R matrix

```
> # size of big matrix object = 664 bytes/0.6 KB
> object.size(airline_big)
664 bytes
> # size of R matrix object = 2080002048 bytes/2.08 GB
> airline_matrix <- airline_big[,]
> object.size(airline_matrix)
2080002048 bytes
```

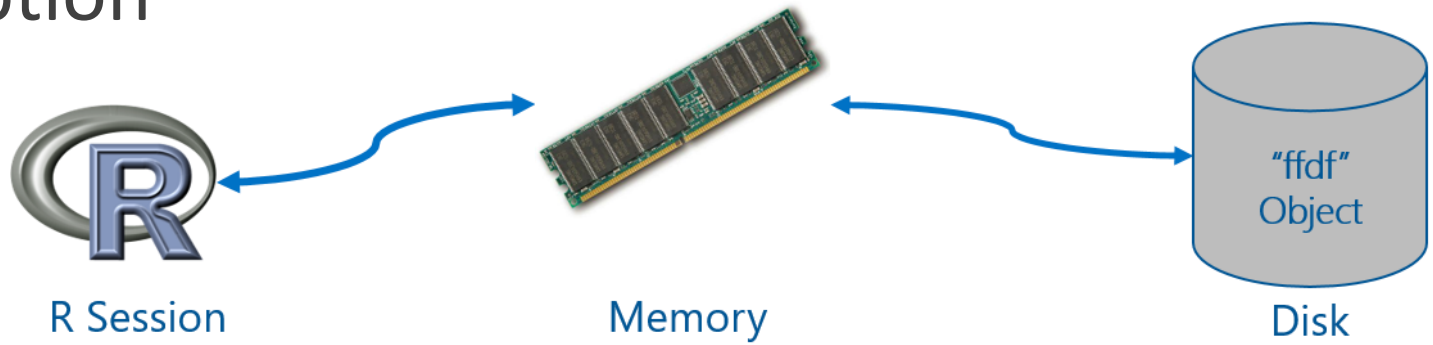
Sister packages and related work

- Set of packages that extends the R programming environment
  - **biganalytics**: provides exploratory data analysis functionality on big.matrix
  - **bigtabulate**: adds table-, tapply-, and split-like behavior for big.matrix
  - **bigalgebra**: performs linear algebra calculations on big.matrix and R matrix
  - **synchronicity**: supports synchronization and may eventually support interprocess communication (ipc) and message passing
- Other related packages
  - **biglm**: provides linear and generalized linear models on big.matrix
  - **Rdsm**: enables shared-memory parallelism with big.matrix

Source: "The Bigmemory Project" by Michael Kane and John Emerson: April 29, 2010.

“ff” package

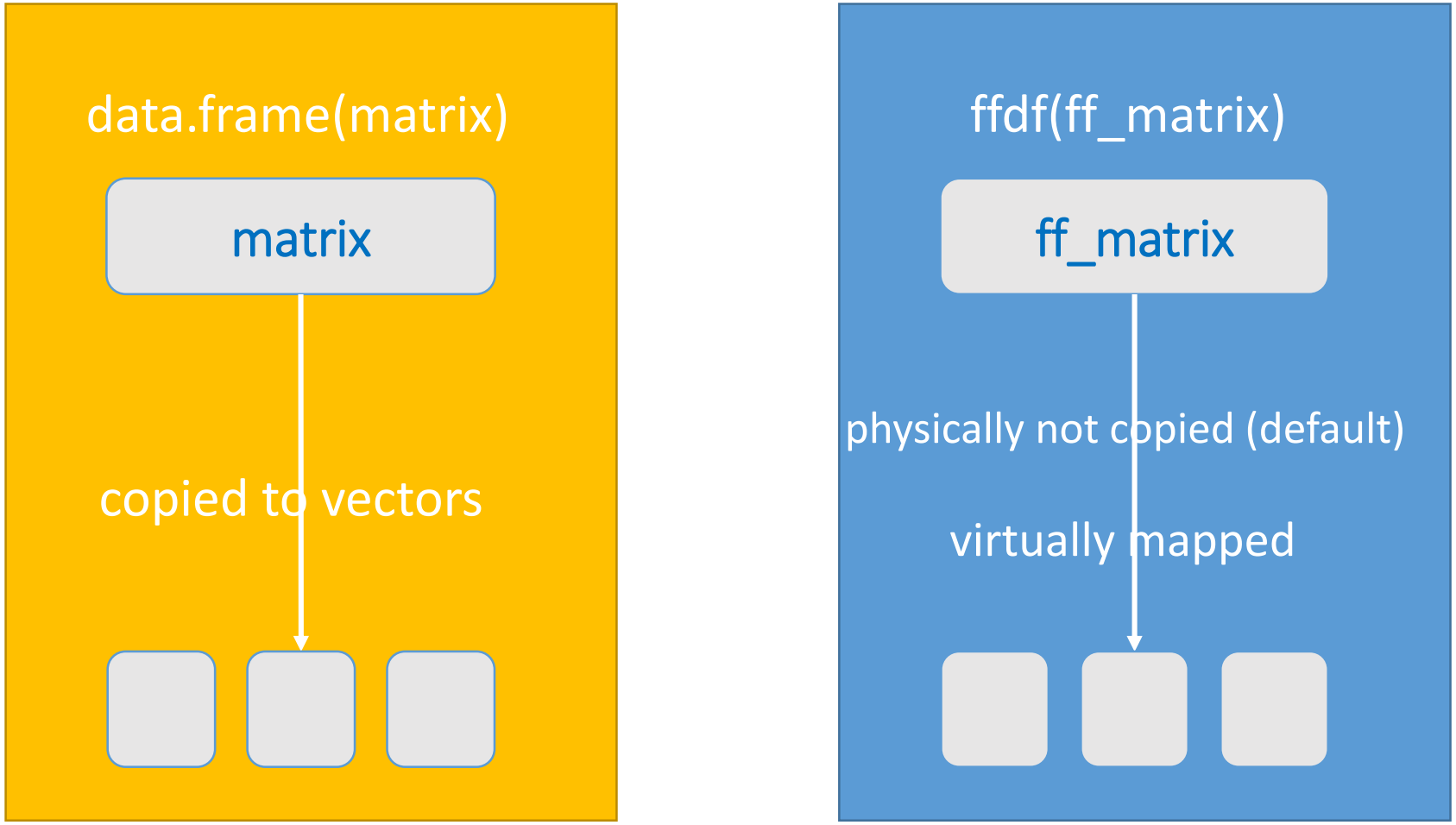
- Provides data structures that are stored on Disk, but behave as if they were in RAM
- Maps only a section in main memory for effective consumption



- Accepts numeric and characters as input data

Source: ff: memory-efficient storage of large data on disk and fast access functions.

ffdf separates virtual from physical storage



Source: Managing data.frames with package 'ff' and fast filtering with package 'bit', by Oehlschlägel, Adler (2009).

“ff” related packages

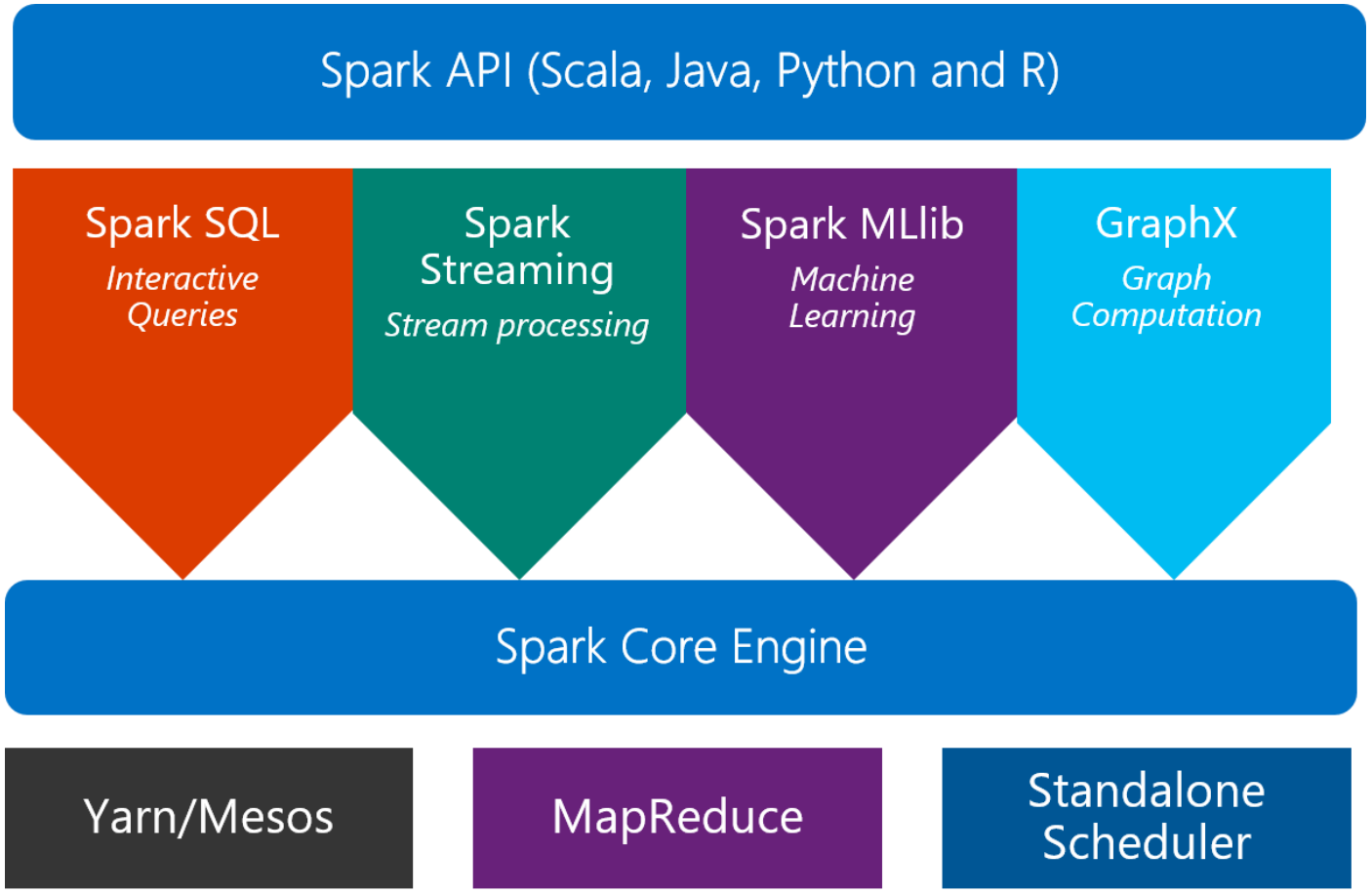
- **ffbase**: adds basic statistical functionality to ff. (Note: *.ff apply on ff vectors, and *.ffdf apply on ffdf.)
  - *Coercions*: as.character.ff(), as.Date_ff_vector(), as.ffdf.ffdf(), as.ram.ffdf()
  - *Selections*: subset.ffdf(), ffwhich(), transform.ffdf(), within.ffdf(), with.ffdf()
  - *Aggregations*: quantile.ff(), hist.ff(), sum.ff(), mean.ff(), range.ff(), tabulate.ff()
  - *Algorithms*: bigglm.ffdf()
- **biglars**: provides least-angle regression, lasso and stepwise regression on ff.

Source: CRAN Task View: High-Performance and Parallel Computing with R, by Dirk Eddelbuettel.

From single machines to distributed computing

- Scale on Spark clusters is one approach

- What is Spark?
  - An unified, open source, parallel, data processing framework for Big Data Analytics



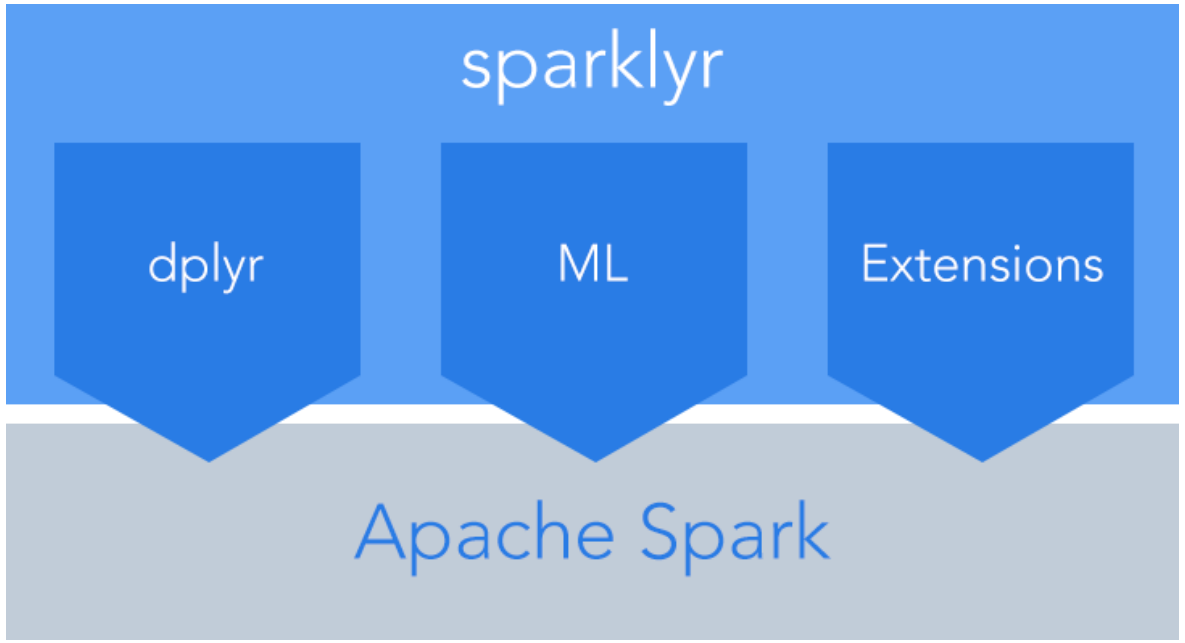
SparkR 1.6: a Spark API

- An R package provides a light-weight frontend to use Apache Spark from R and allows data scientists to analyze large datasets.
- Spark DataFrame is distributed collection of data organized into named columns.
- With **SQLContext**, SparkR can create Spark DataFrames from local R data frames, csv, json and parquet files.
- Using **HiveContext**, it can also access tables from Hive MetaStore.
- Pre-configured on Spark clusters in HDInsight.

Data processing and modeling with SparkR

- Supports functions for structured data processing:
  - *Selections*: select(), filter()
  - *Aggregations*: summarize(), arrange()
  - *Applying UDFs on each partition of a SparkDataFrame*: available in SparkR 2.0!
    - dapply(), dapplyCollect(), gapply(), gapplyCollect()
- Uses ***MLlib*** to fit generalized linear models over *SparkDataFrames*
- More algorithms and model persistence are included in SparkR 2.0
  - Accelerated Failure Time (AFT) Survival Regression Model
  - Naive Bayes Model
  - KMeans Model

sparklyr: R interface for Apache Spark



- Easy installation via devtools

```
# install sparklyr
devtools::install_github("rstudio/sparklyr")
```

- Connect to both local instances of Spark and remote Spark clusters

```
library("sparklyr")
# connect to local instance of Spark
sc <- spark_connect(master = "local")
# connect to remote Spark clusters
sc <- spark_connect(master = "yarn-client")
```

- Loads data into Spark DataFrames from: local R data frames, Hive tables, CSV, JSON, and Parquet files.

Source: http://spark.rstudio.com/

dplyr and ML in sparklyr

- Provides a complete dplyr backend for data manipulation, analysis and visualization

```
# manipulate data with dplyr
library("dplyr")
partitions <- airline_lyr %>%
  mutate(CRSDepTimeHour = floor(CRSDepTime/100)) %>%
  sdf_partition(training = 0.7, test = 0.3, seed = 1099)
```

%>%

- Includes 3 family of functions for machine learning pipeline
  - **ml_***: Machine learning algorithms for analyzing data provided by the *spark.ml* package.
    - K-Means, GLM, LR, Survival Regression, DT, RF, GBT, PCA, Naive-Bayes, Multilayer Perceptron
  - **ft_***: Feature transformers for manipulating individual features.
  - **sdf_***: Functions for manipulating *SparkDataFrames*.

```
# train Random Forest model
rf_model <- partitions$training %>%
  ml_random_forest(response = "IsArrDelayed", features = c("CRSDepTimeHour"),
                   max.bins = 32L, max.depth = 5L, num.trees = 20L)
```

R Server: scale-out R, Enterprise Class!

- 100% compatible with open source R
  - Any code/package that works today with R will work in R Server.
- Ability to parallelize any R function
  - Ideal for parameter sweeps, simulation, scoring.
- Wide range of scalable and distributed *"**rx**"* pre-fixed functions in "RevoScaleR" package.
  - *Transformations*: rxDataStep()
  - *Statistics*: rxSummary(), rxQuantile(), rxChiSquaredTest(), rxCrossTabs()…
  - *Algorithms*: rxLinMod(), rxLogit(), rxKmeans(), rxBTrees(), rxDForest()…
  - *Parallelism*: rxSetComputeContext()

Parallelized & Distributed Analytics

ETL

- Data import – Delimited, Fixed, SAS, SPSS, OBDC
- Variable creation & transformation
- Recode variables
- Factor variables
- Missing value handling
- Sort, Merge, Split
- Aggregate by category (means, sums)

Descriptive Statistics

- Min / Max, Mean, Median (approx.)
- Quantiles (approx.)
- Standard Deviation
- Variance
- Correlation
- Covariance
- Sum of Squares (cross product matrix for set variables)
- Pairwise Cross tabs
- Risk Ratio & Odds Ratio
- Cross-Tabulation of Data (standard tables & long form)
- Marginal Summaries of Cross Tabulations

Statistical Tests

- Chi Square Test
- Kendall Rank Correlation
- Fisher's Exact Test
- Student's t-Test

Predictive Statistics

- Sum of Squares (cross product matrix for set variables)
- Multiple Linear Regression
- Generalized Linear Models (GLM) exponential family distributions: binomial, Gaussian, inverse Gaussian, Poisson, Tweedie. Standard link functions: cauchit, identity, log, logit, probit. User defined distributions & link functions.
- Covariance & Correlation Matrices
- Logistic Regression
- Predictions/scoring for models
- Residuals for all models

Variable Selection

- Stepwise Regression

Machine Learning

- Decision Trees
- Decision Forests
- Gradient Boosted Decision Trees
- Naïve Bayes

Clustering

- K-Means

Sampling

- Subsample (observations & variables)
- Random Sampling

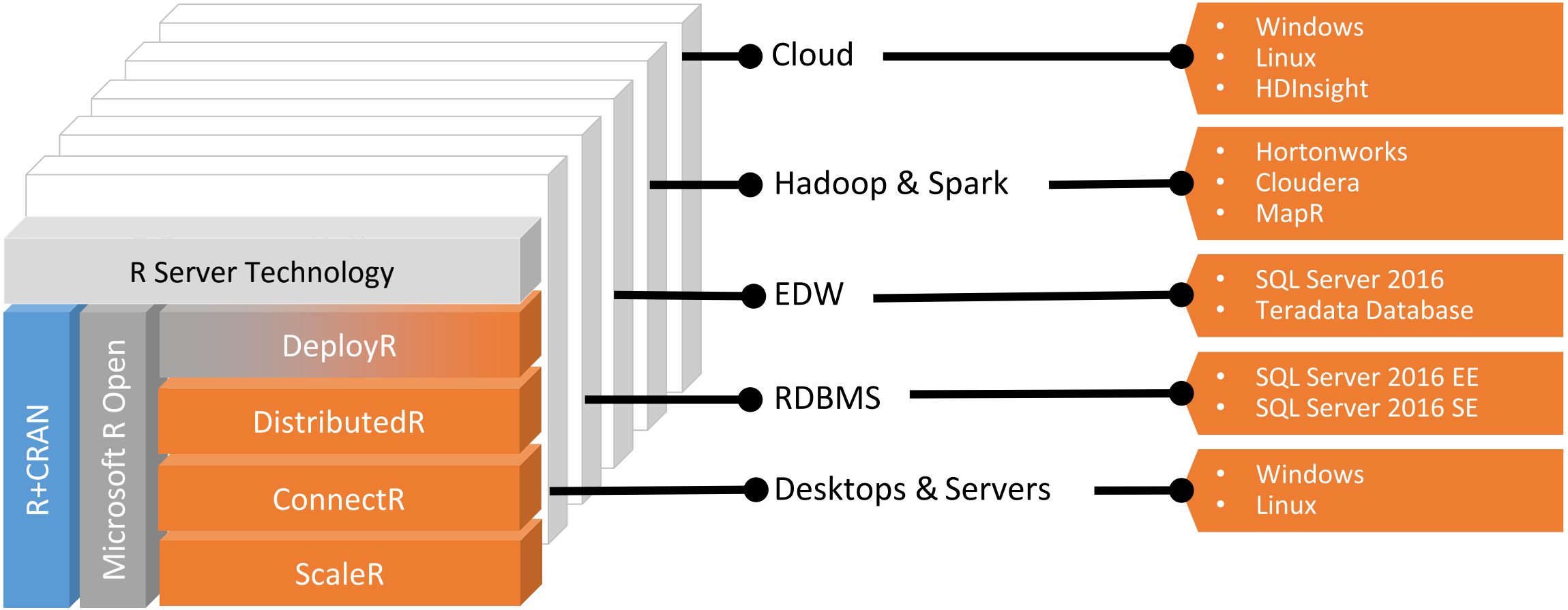
Simulation

- Simulation (e.g. Monte Carlo)
- Parallel Random Number Generation

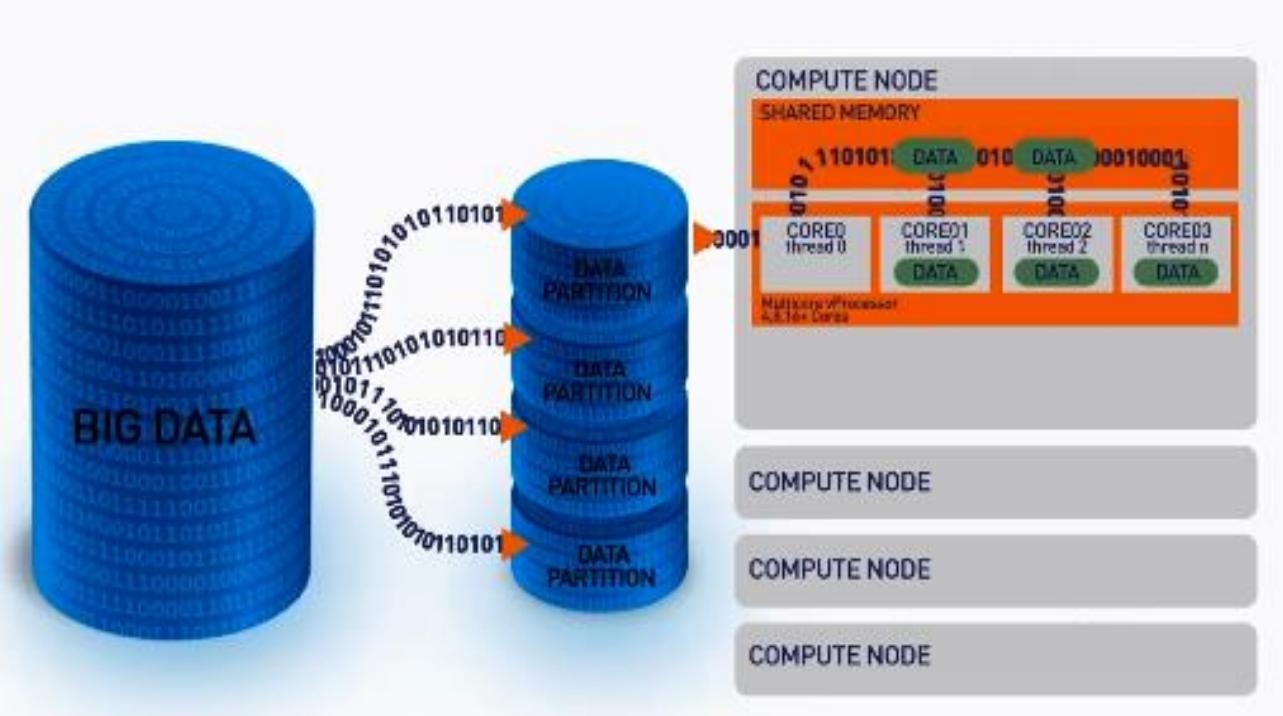
Custom Parallelization

- rxDataStep
- rxExec
- PEMA-R API

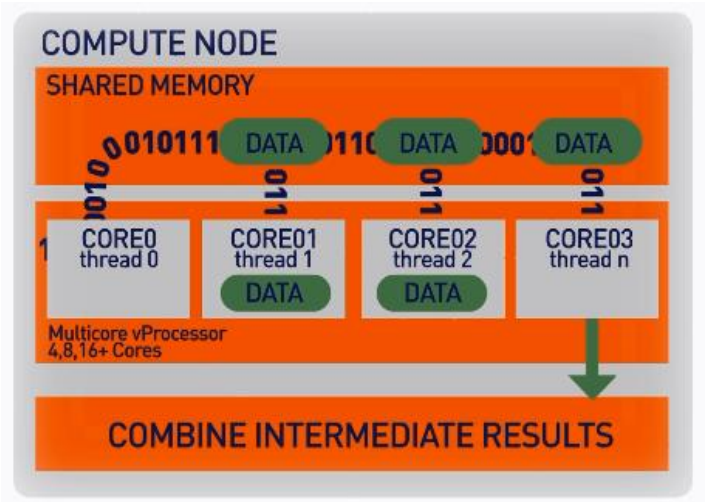
Portable across multiple platforms



ScaleR: parallel + Big Data



Our ScaleR algorithms work inside multiple cores / nodes in parallel at high speed



Stream data into RAM in blocks. "Big Data" can be any data size. We handle Megabytes to Gigabytes to Terabytes…

XDF file format is optimised to work with the ScaleR library and significantly speeds up iterative algorithm processing.

Interim results are collected and combined analytically to produce the output on the entire data set

Write Once - Deploy Anywhere

ScaleR models can be deployed from a server or edge node to run in Spark/Hadoop without any functional R model re-coding.

Compute context R script - sets where the model will run

Local Parallel processing - **Linux or Windows**

```
### SETUP LOCAL ENVIRONMENT VARIABLES ###
myLocalCC <- "localpar"

### LOCAL COMPUTE CONTEXT ###
rxSetComputeContext(myLocalCC)

### CREATE LINUX, DIRECTORY AND FILE OBJECTS ###
linuxFS <- RxNativeFileSystem( )
AirlineDataSet <- RxXdfData("airline_20MM.xdf",
                            fileSystem = linuxFS)
```

In – **Spark/Hadoop**

```
### SETUP SPARK/HADOOP ENVIRONMENT VARIABLES ###
mySparkCC <- RxSpark()          myHadoopCC <- RxHadoopMR()

### HADOOP COMPUTE CONTEXT ###
rxSetComputeContext(mySparkCC) rxSetComputeContext(myHadoopCC)

### CREATE HDFS, DIRECTORY AND FILE OBJECTS ###
hdfsFS <- RxHdfsFileSystem()
AirlineDataSet <- RxXdfData("airline_20MM.xdf",
                            fileSystem = hdfsFS)
```

Functional model R script – does not need to change to run in Spark

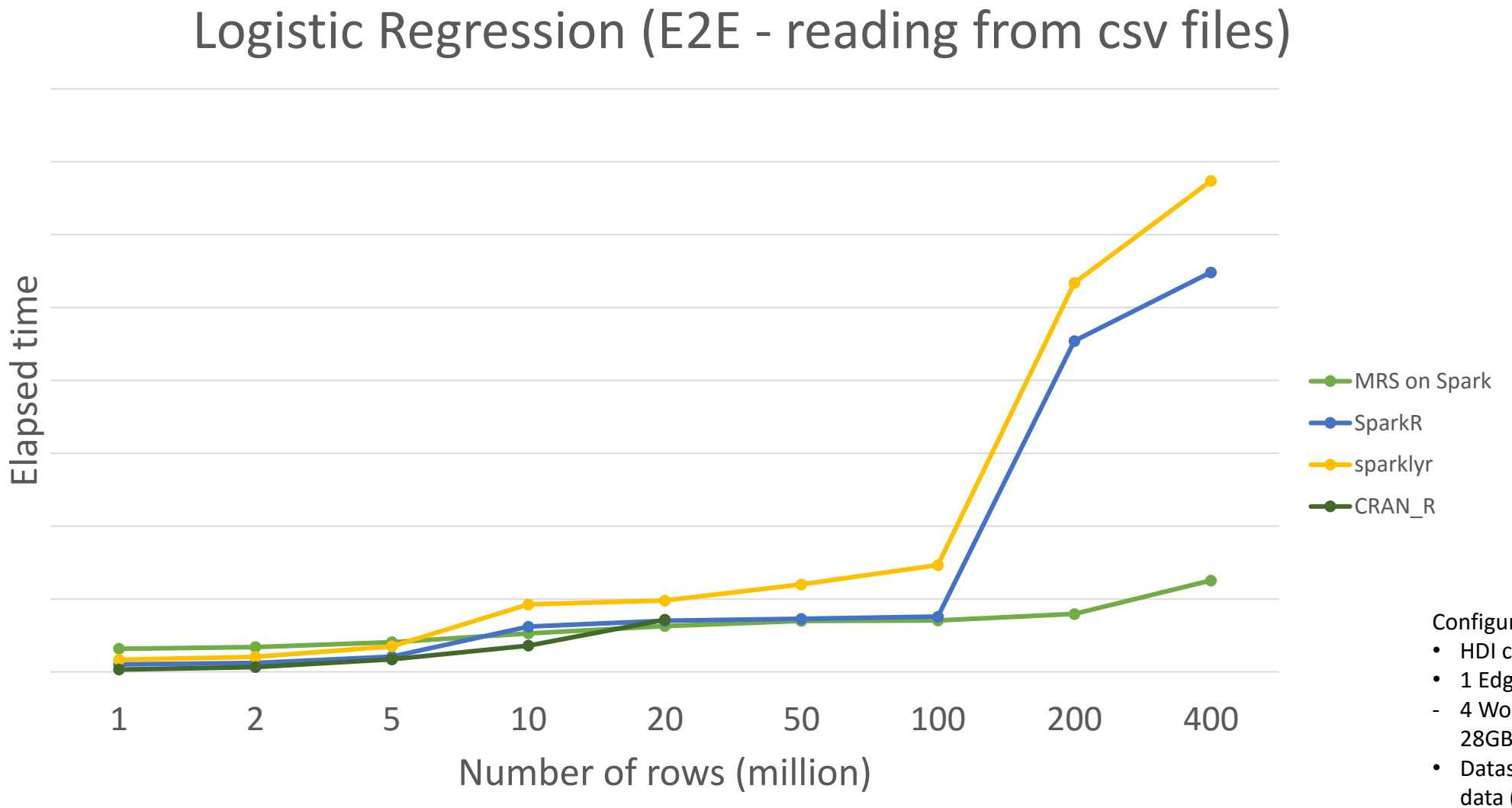
```
### ANALYTICAL PROCESSING ###
### Statistical Summary of the data
      rxSummary( ~ ArrDelay + DayOfWeek, data = AirlineDataSet, reportProgress = 1)

### CrossTab the data
      rxCrossTabs(ArrDelay ~ DayOfWeek, data = AirlineDataSet, means = T)

### Linear model and plot
      hdfsXdfArrLateLinMod <- rxLinMod(ArrDelay ~ DayOfWeek + CRSDepTime, data = AirlineDataSet)
      plot(hdfsXdfArrLateLinMod$coefficients)
```

Base and scalable approaches comparison

Approach	Scalability	Spark	Hadoop	SQL	Teradata	Support
CRAN R[1]	Single machines					Community
bigmemory	Single machines					Community
ff	Single machines					Community
SparkR	Single + Distributed computing	X		X		Community
sparklyr	Single + Distributed computing	X		X		Community
RevoScaleR	Single + Distributed computing	X	X	X	X	Enterprise

1. **CRAN R indicates no additional R packages installed**

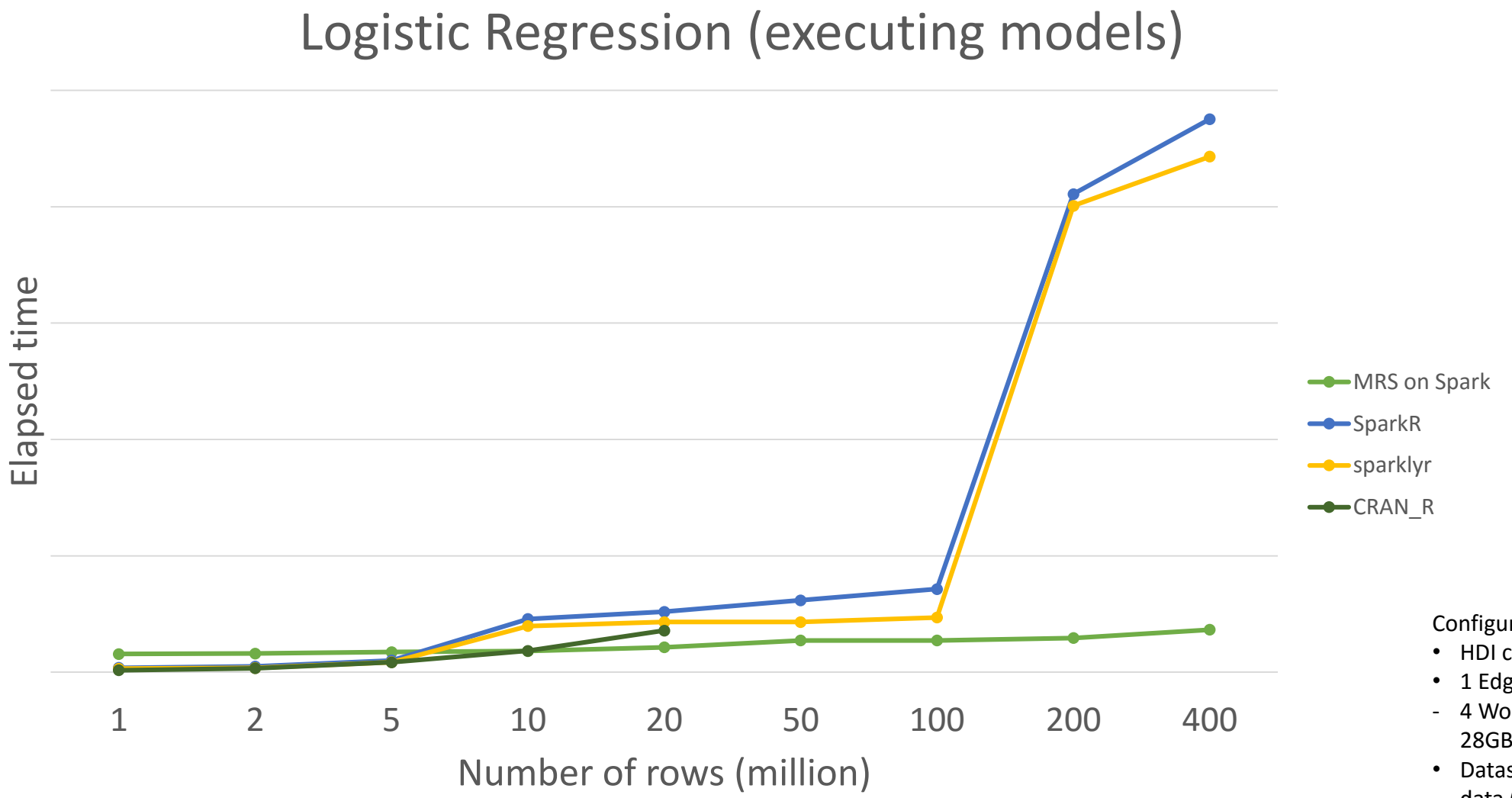
R Server on Spark - faster and more scalable



Configuration:

- HDI cluster size: 7 nodes
- 1 Edge Node: 8 cores, 28GB
- 4 Worker Nodes: 8 cores, 28GB
- Dataset: Duplicated Airlines data (.csv)
- Number of columns: 26

R Server on Spark - substantially faster



Configuration:

- HDI cluster size: 7 nodes
- 1 Edge Node: 8 cores, 28GB
- 4 Worker Nodes: 8 cores, 28GB
- Dataset: Duplicated Airlines data (.csv)
- Number of columns: 26

Overview of open libraries of R templates

- Tutorial material and slides are available:

  tinyurl.com/KDD2016R

- R Server for Machine Learning examples:

  tinyurl.com/2016MRS4ML

An end-to-end solution with R Server and Spark

TUTORIAL

Retail Customer Churn Template using Microsoft R Server/HDInsight/Spark

Microsoft • published on July 15, 2016

View Code

+ Add to Collection

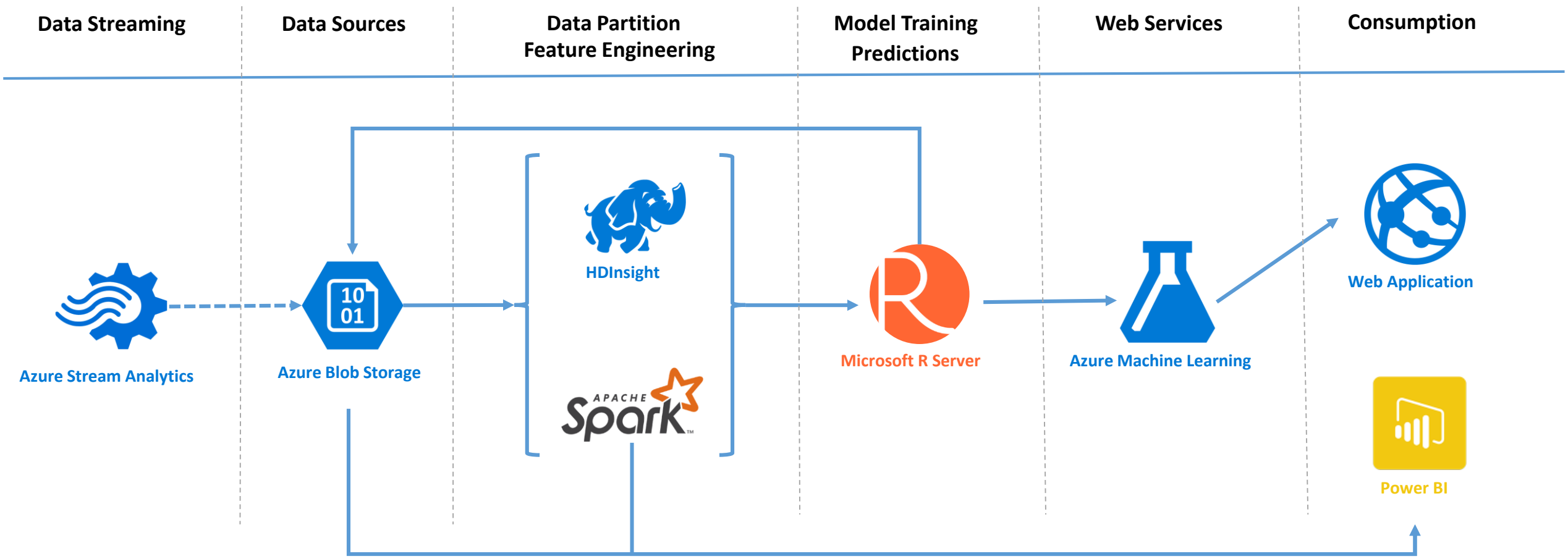
Summary

This template demonstrates how to develop and deploy end-to-end, cloud solutions for Retail Customer Churn using Microsoft R Server, Azure HDInsight with R on Linux, Azure Machine Learning, Spark, Scala, Hive and Power BI.

Description

Customer churn is supremely important for retail, banking, telecommunications and many others customer related industries. Therefore, accurate churn predictions can help businesses proactively conduct better promotion plans, adjust engagement strategies, and make important business decisions. In the retail sector, churn predictions is critical and the loss of customers to competitors must be managed and prevented. The goal of churn predictions is to identify which customers are likely to churn.

Scale on big data - R Server + Azure



Example – New York Taxi Dataset

2013 NYC Taxi and Fare Data – ~40GB uncompressed
170M trips/fares paid

Src: http://bit.ly/2badpSv

2 tasks

- **Regression - Predict the Tip ($) for taxi trips**
  Predict the tip paid for each taxi trip using features of the taxi trip
  Features: Trip distance, fare amount, number of passengers, payment type (credit card, cash etc.), and time of the day

- **Binary classification – Predicting whether a trip will have a tip**
  Predict whether or not a tip was paid for a trip (binary classification).
  Positive: Tip amount > $0
  Negative: Tip amount = $0

Method

Microsoft R server (MRS) + HDInsight

Setting Compute Context

```
## This diverts outputs and messages to a sink file on the R-server node (where R-studio server is installed)
system("rm /home/remoteuser/sinkFile.txt")
sinkFile <- file("/home/remoteuser/sinkFile.txt", open = "wt")
sink(sinkFile)


bigDataDirRoot <- "/share" ;
myNameNode <- "default";
myPort <- 0;
myHadoopCluster <- RxHadoopMR(
  hdfsShareDir = bigDataDirRoot,
  nameNode = myNameNode,
  port= myPort,
  hadoopSwitches = '-Dmapred.task.timeout=86400000 -Dmapreduce.input.fileinputformat.split.minsize=110000000
-libjars /etc/hadoop/conf',
  consoleOutput    = TRUE);


rxSetComputeContext(myHadoopCluster);
hdfsFS <- RxHdfsFileSystem(hostName=myNameNode, port=myPort)
```

Prepare xdf Files

```
# Specify path to input file in HDFS
inputFile <-file.path(bigDataDirRoot,"Data/JoinedTaxiTripFare.100.tsv");
xdfOutFile <- file.path(bigDataDirRoot,"Data/taxiDSXdf");

# Define column classes
taxiColClasses <- c(medallion = "character", hack_license = "character",
                    vendor_id =  "factor", rate_code = "factor",
                    store_and_fwd_flag = "character", pickup_datetime = "character",
                    dropoff_datetime = "character", pickup_hour = "numeric",
                    pickup_week = "numeric", weekday = "numeric",
                    passenger_count = "numeric", trip_time_in_secs = "numeric",
                    trip_distance = "numeric", pickup_longitude = "numeric",
                    pickup_latitude = "numeric", dropoff_longitude = "numeric",
                    dropoff_latitude = "numeric", direct_distance = "numeric",
                    payment_type = "factor", fare_amount = "numeric",
                    surcharge = "numeric", mta_tax = "numeric", tip_amount = "numeric",
                    tolls_amount = "numeric", total_amount = "numeric",
                    tipped = "factor", tip_class = "factor");
```

Prepare xdf Files

```
# Create xdf file
taxiDS <- RxTextData(file = inputFile, colClasses  = taxiColClasses,
                     fileSystem = hdfsFS, delimiter = "\t", firstRowIsColNames = TRUE);
xdfOut <- RxXdfData(file = xdfOutFile, fileSystem = hdfsFS);
capture.output(taxiDSXdf <- rxImport(inData = taxiDS, outFile = xdfOut,
                                     fileSystem = hdfsFS, createCompositeSet = TRUE,
                                     overwrite = TRUE),
                                     file=sinkFile);
```

Redirect output from MapReduce Jobs

Split Data into Train and Test

```
# Assign each observation randomly to training or testing (75% training and 25% testing)

# We also delete some variables not used for modeling, and filter observations which are likely to be invalid
or outliers.

taxiSplitXdfFile <- file.path(bigDataDirRoot,"Data/taxiSplitXdf");

taxiSplitXdf <- RxXdfData(file = taxiSplitXdfFile, fileSystem = hdfsFS);

capture.output(

            rxDataStep(inData = taxiDSXdf, outFile = taxiSplitXdf,

            varsToDrop = c("medallion", "hack_license","store_and_fwd_flag",

                           "pickup_datetime", "rate_code",

                           "dropoff_datetime","pickup_longitude",

                           "pickup_latitude", "dropoff_longitude",

                           "dropoff_latitude ", "direct_distance", "surcharge",

                           "mta_tax", "tolls_amount", "tip_class", "total_amount"),

            rowSelection = (passenger_count > 0 & passenger_count < 8 &

                              tip_amount >= 0 & tip_amount <= 40 &

                              fare_amount > 0 & fare_amount <= 200 &

                              trip_distance > 0 & trip_distance <= 100 &

                              trip_time_in_secs > 10 & trip_time_in_secs <= 7200),
```

Split Data into Train and Test

```
transforms = list( testSplitVar = ( runif( .rxNumRows ) > 0.25 ) ),
                            # 25% test, %75 into training
                            overwrite = TRUE),
             file=sinkFile
             );

# Create training data xdf
taxiTrainXdfFile <- file.path(bigDataDirRoot,"Data/taxiTrainXdf");
trainDS <- RxXdfData(file = taxiTrainXdfFile,  fileSystem = hdfsFS);
capture.output(
             rxDataStep( inData = taxiSplitXdf, outFile = trainDS,
             varsToDrop = c( "testSplitVar"),
             rowSelection = ( testSplitVar == 1),
             overwrite = TRUE), file=sinkFile
             );
```

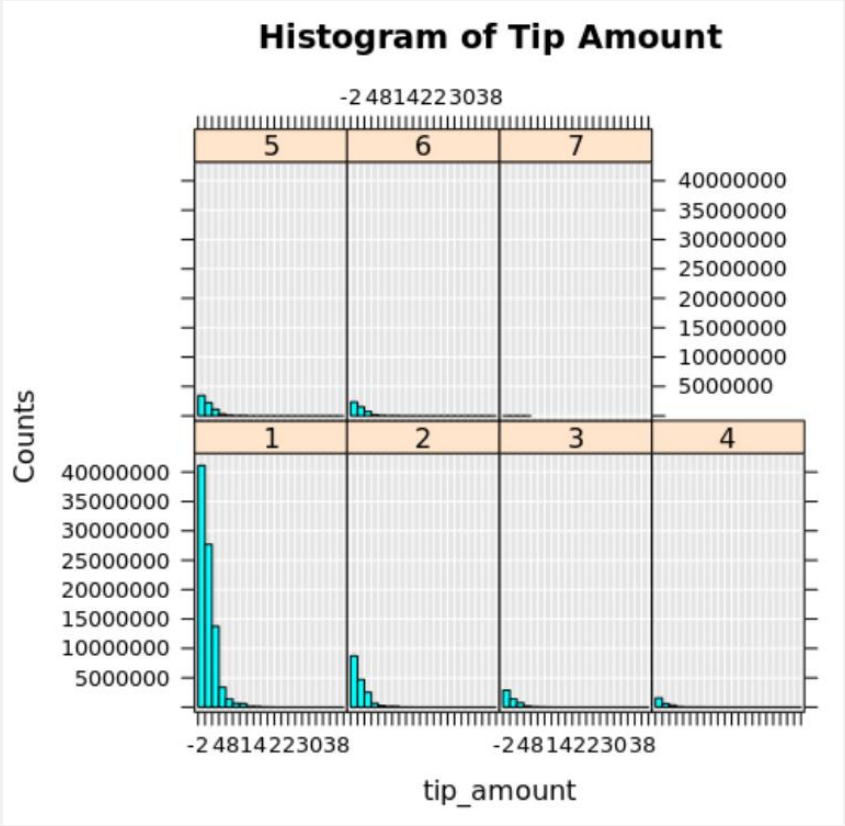
Spli the data into 75% training and 25% testing data

Split Data into Train and Test

```
# Create testing data xdf
taxiTestXdfFile <- file.path(bigDataDirRoot,"Data/taxiTestXdf");
testDS <- RxXdfData(file= taxiTestXdfFile,  fileSystem = hdfsFS);
capture.output(
            rxDataStep( inData = taxiSplitXdf, outFile = testDS,
            varsToDrop = c( "testSplitVar"),
            rowSelection = ( testSplitVar == 0),
            overwrite = TRUE), file=sinkFile
            )
```

Data Exploration

```
capture.output(
    histPlot <- rxHistogram(~tip_amount | passenger_count, numBreaks=20, data = trainDS,
                            title = "Histogram of Tip Amount"),
                file = sinkFile
    )
```



Linear Regression

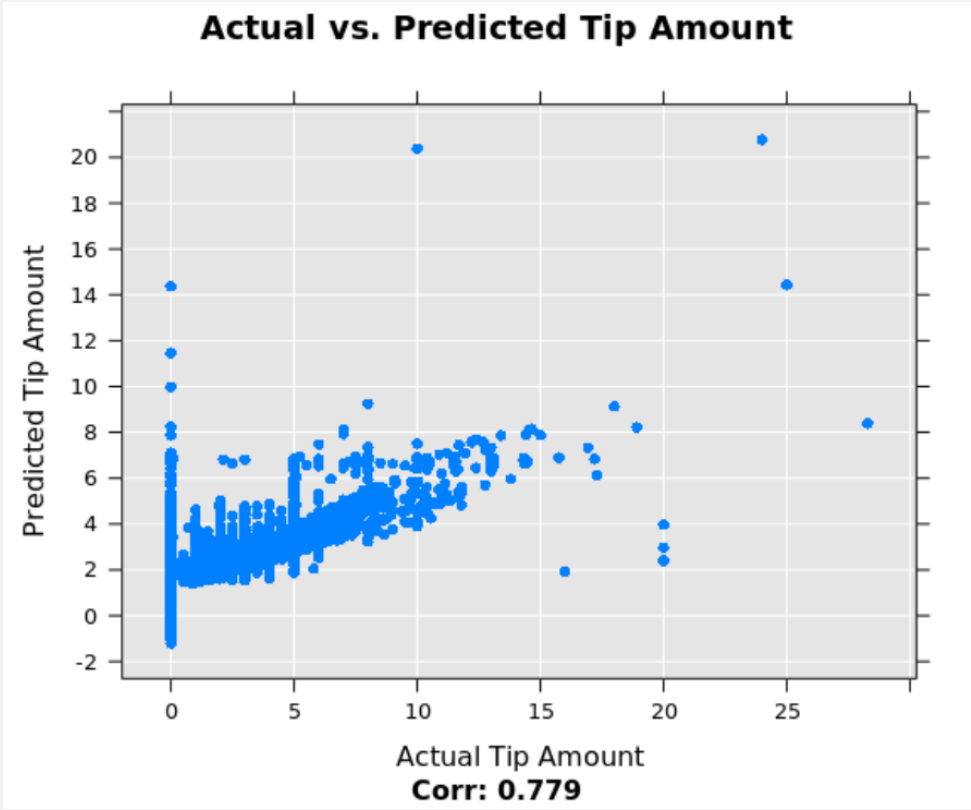
```
## Build the linear regression model
pt1 <- proc.time();
capture.output (
             model.rxLinMod <- rxLinMod(tip_amount ~ fare_amount + vendor_id +
                                        pickup_hour + pickup_week + weekday +
                                        passenger_count  + trip_time_in_secs +
                                        trip_distance + payment_type, data = trainDS),
                                        file=sinkFile
             )

## Get a summary for the model
summary(model.rxLinMod);
```

Evaluate Model

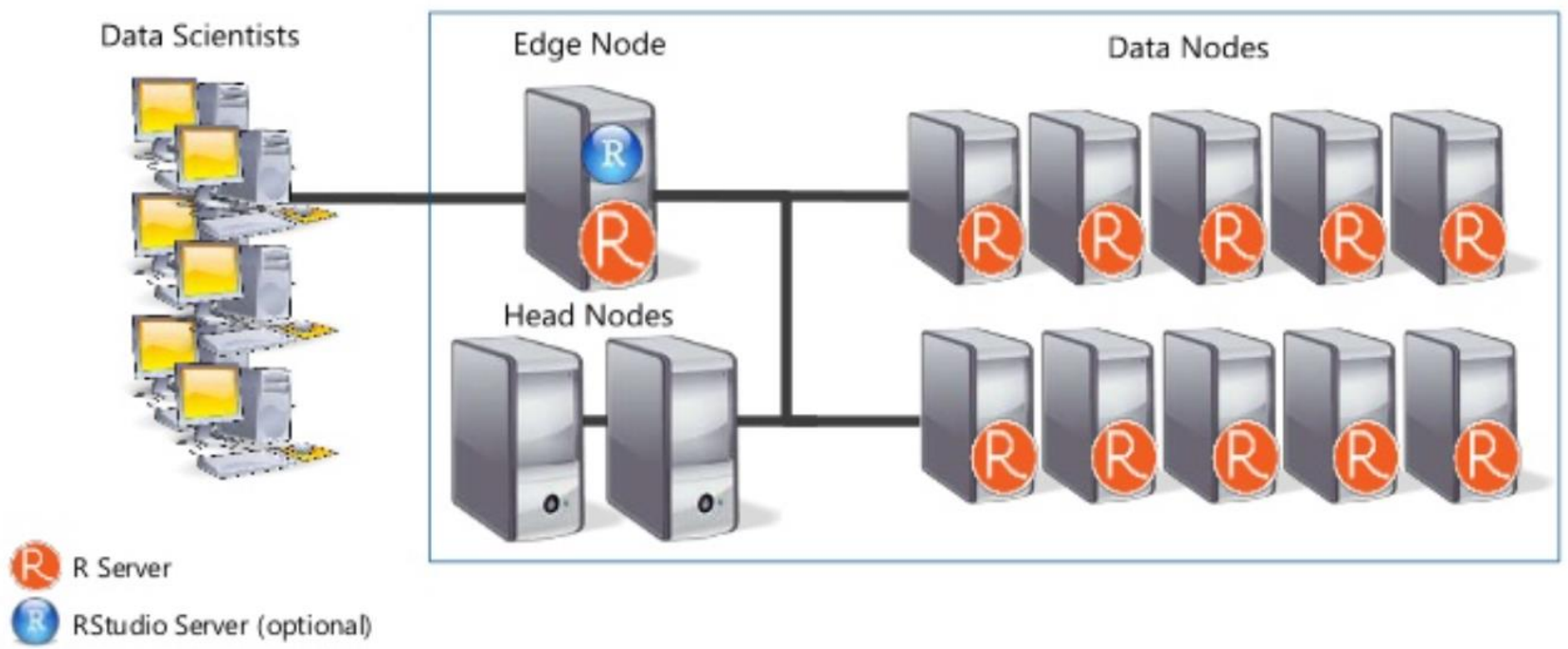
```
# Predict on test data-set, get AUC, accuracy etc.

outputLinMod  <- RxXdfData(file.path(bigDataDirRoot, "Results/PredictedLinMod"),
fileSystem=RxFileSystem(fileSystem = "hdfs"));


capture.output (
                taxiDxPredictLinMod <- rxPredict(modelObject = model.rxLinMod, checkFactorLevels = TRUE,
                                                 data = testDS, outData = outputLinMod,
                                                 type = "response",
                                                 extraVarsToWrite = as.vector(c("tip_amount")),
                                                 predVarNames = "predicted_tipped_amount",
                                                 overwrite = TRUE),
                                          file = sinkFile
                )


capture.output (linModDF <- rxImport(inData = outputLinMod, outFile = NULL), file = sinkFile);
```

Evaluate Model

```
rxSetComputeContext("local");

## Sample a subset of rows for plotting, otherwise plot looks busy
overallCorr <- round(cor.test(linModDF$tip_amount, linModDF$predicted_tipped_amount)$estimate, 3);
linModDFSampled <- linModDF[sample(dim(linModDF)[1], 10000),];
linePlot <- rxLinePlot(predicted_tipped_amount ~ tip_amount,
                       data = linModDFSampled, type = 'p',
                       title = 'Actual vs. Predicted Tip Amount',
                       xTitle = 'Actual Tip Amount',
                       yTitle = 'Predicted Tip Amount',
                       subtitle = paste0('Corr: ', overallCorr)
                       )
```



R Server on HDInsight Architecture



R Server Distributed Processing:

Master R process on Edge node

Apache YARN and Spark

Worker R processes on Data nodes

Src: Revolution Analytics

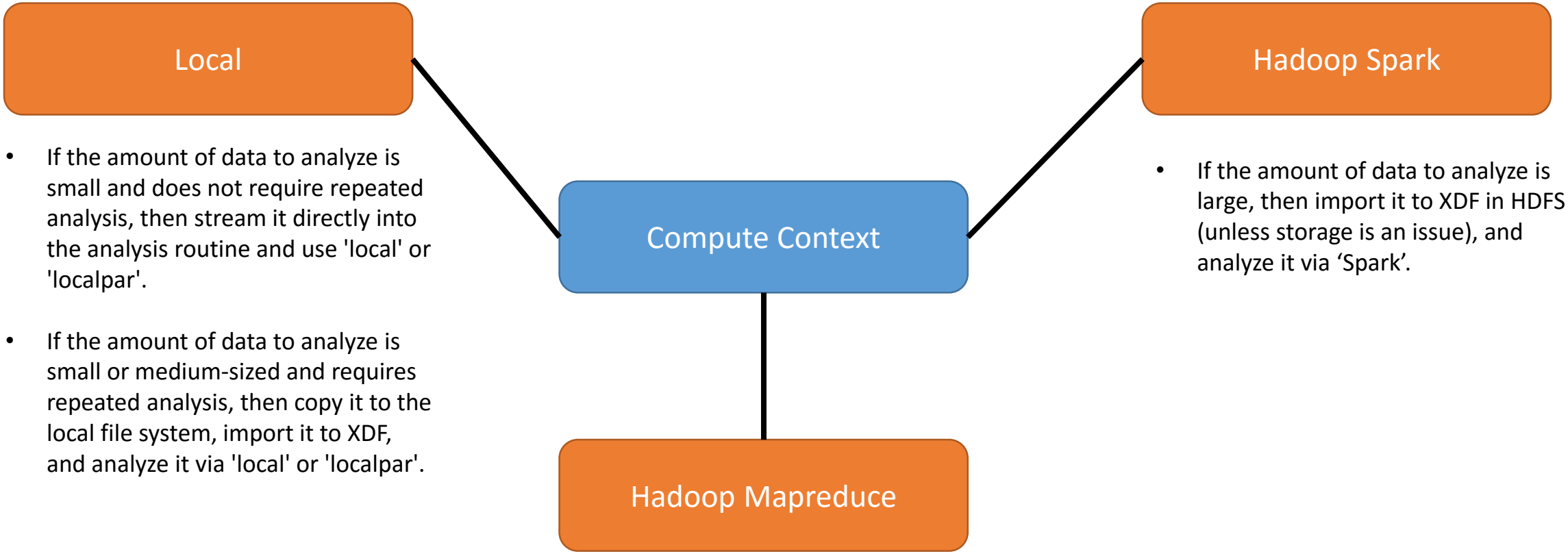
R Server on HDInsight – Different Compute Contexts

Compute context	How to set	Execution context
Local sequential	rxSetComputeContext ('local')	Parallelized execution across the cores of the edge node server, except for rxExec calls which are executed serially
Local parallel	rxSetComputeContext ('localpar')	Parallelized execution across the cores of the edge node server
Spark	RxSpark()	Parallelized distributed execution via Spark across the nodes of the HDI cluster
Map Reduce	RxHadoopMR()	Parallelized distributed execution via Map Reduce across the nodes of the HDI cluster

Principles - Compute Context

- Local file system on Linux is faster than HDFS.
- Repeated analyses are faster if the data is local, and if it's in XDF.
- It's preferable to stream small amounts of data from a text data source; if the amount of data is larger, convert it to XDF prior to analysis.
- Overhead of copying or streaming the data to the edge node for analysis becomes unmanageable for very large amounts of data.

Choosing a Compute Context

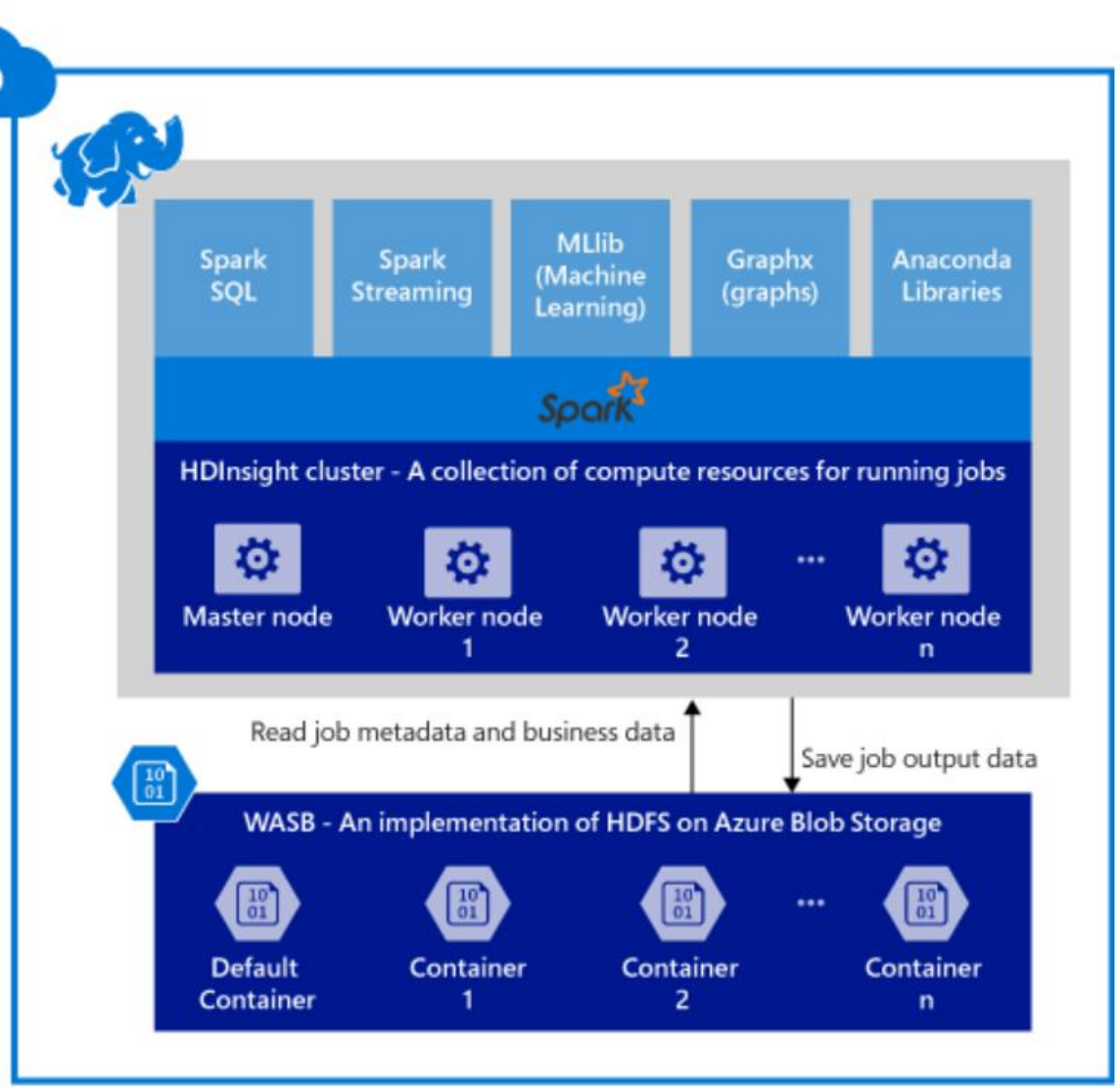


- If the amount of data to analyze is small and does not require repeated analysis, then stream it directly into the analysis routine and use 'local' or 'localpar'.
- If the amount of data to analyze is small or medium-sized and requires repeated analysis, then copy it to the local file system, import it to XDF, and analyze it via 'local' or 'localpar'.

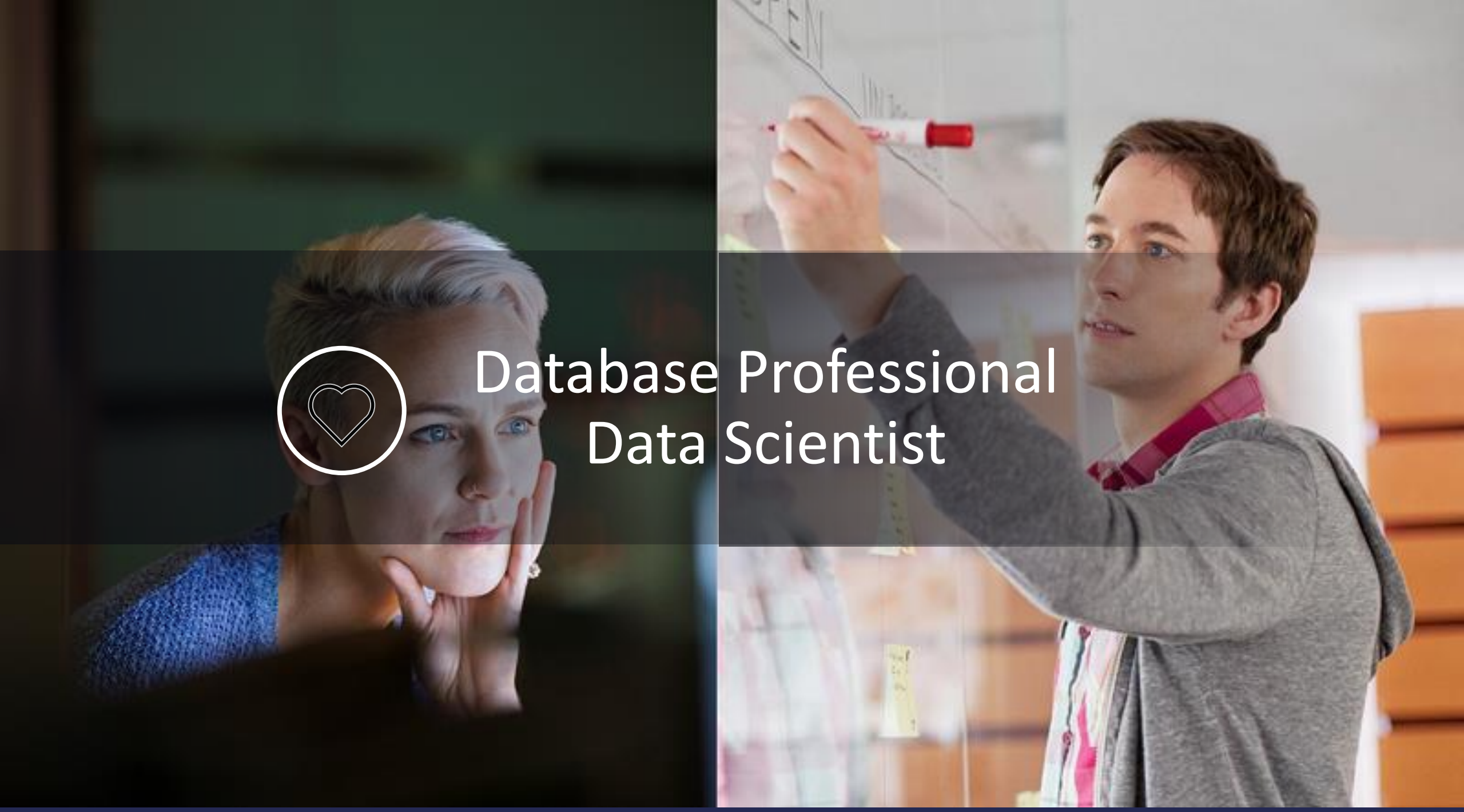
- If the amount of data to analyze is large, then import it to XDF in HDFS (unless storage is an issue), and analyze it via ‘Spark’.

Use only if you encounter an insurmountable problem with use of the Spark compute context since generally it will be slower.

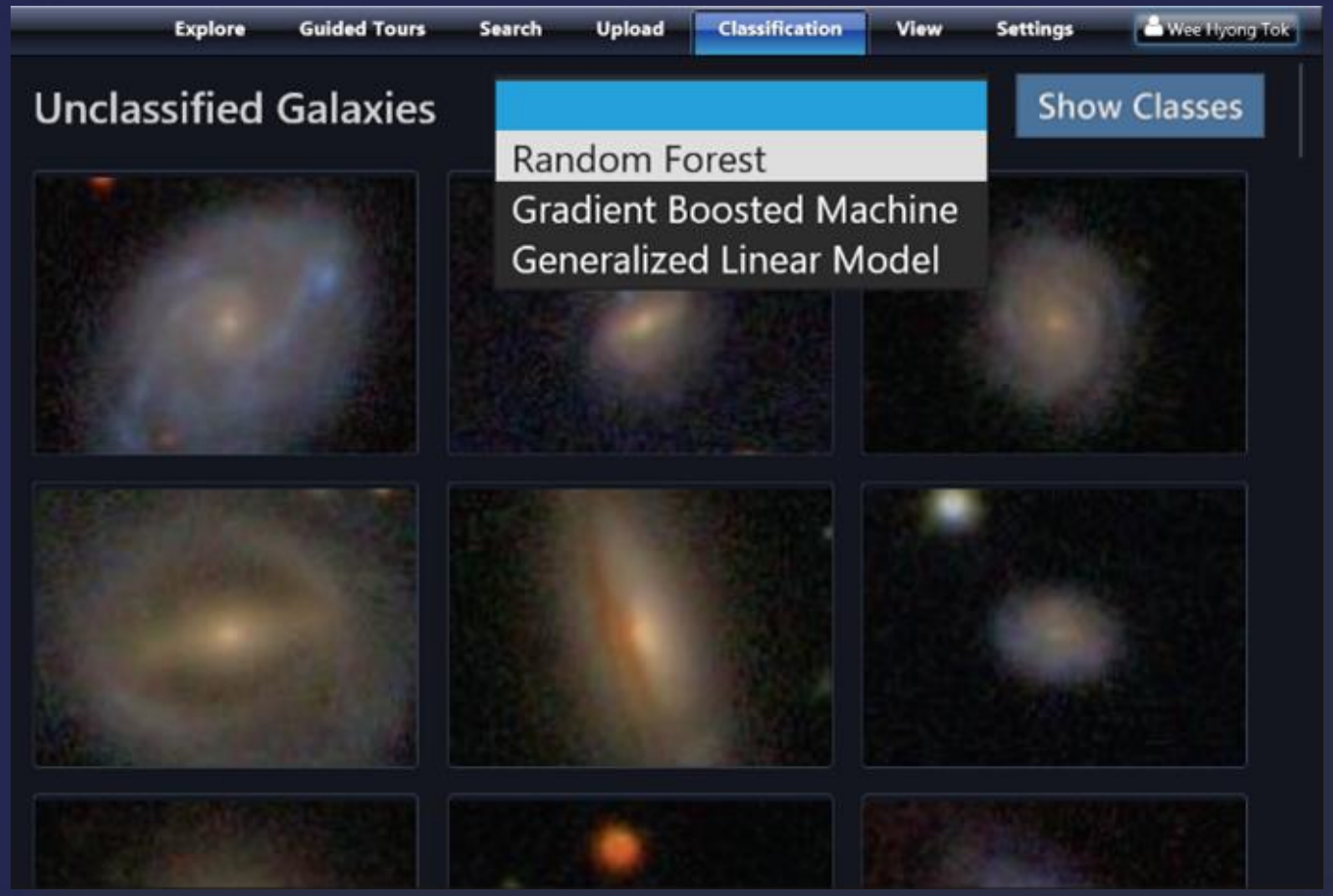
Spark on HDInsight



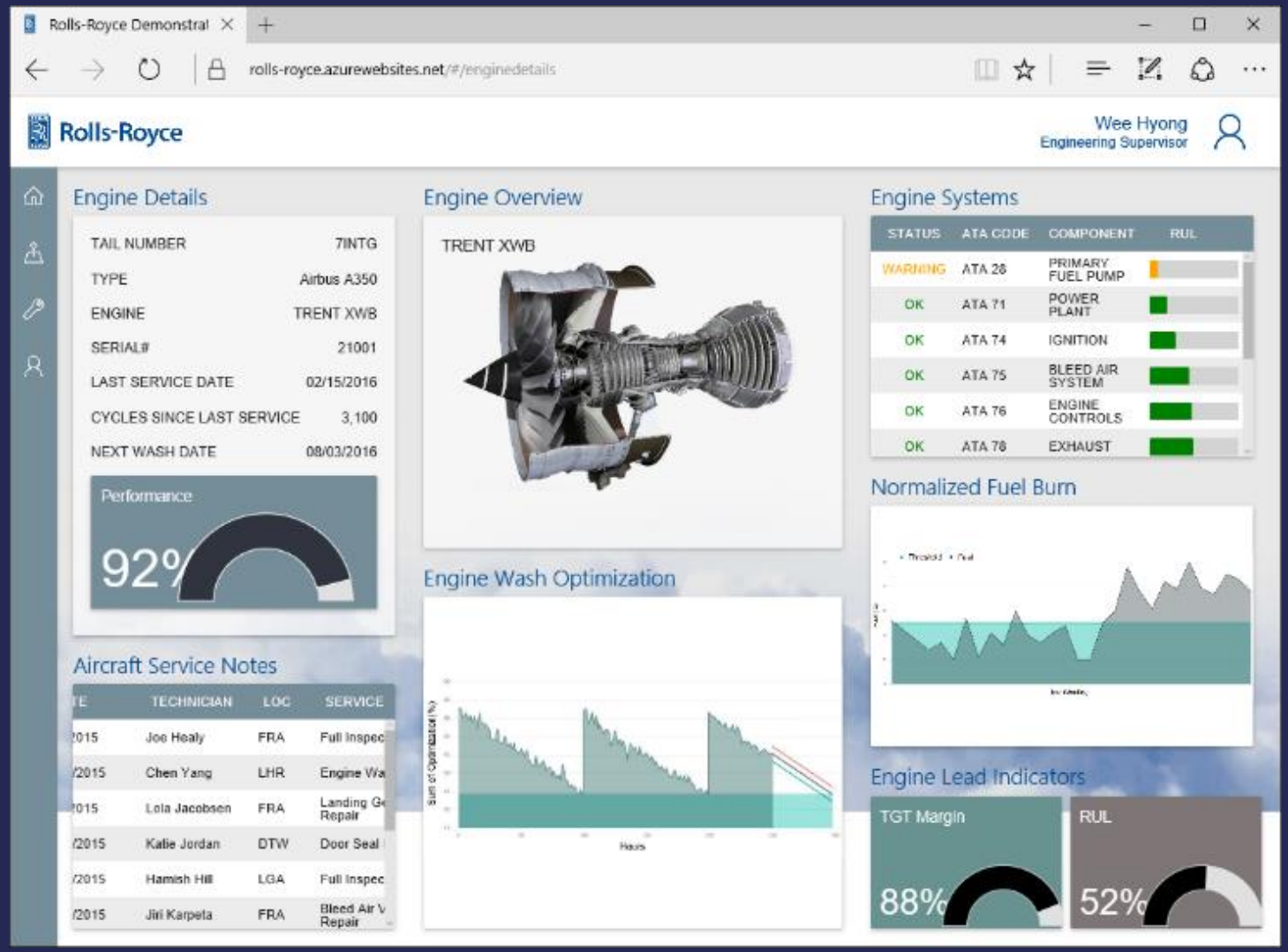
R and Databases



Database Professional
Data Scientist



Helping Astronomers Classify Galaxies



Predicting Remaining Useful Lifetime (RUL) of Engine Systems

Sounds familiar?

- Write SQL queries! (joins, aggregation, filtering…)
- Tune a database
- Import/Export Data
- Do ETL

Recent Years - Innovations in Databases

- Indexes
- In-memory
- Columnstore
- PolyBase
- Temporal
- Lots more! (not an exhaustive list)

Blending the best of both worlds

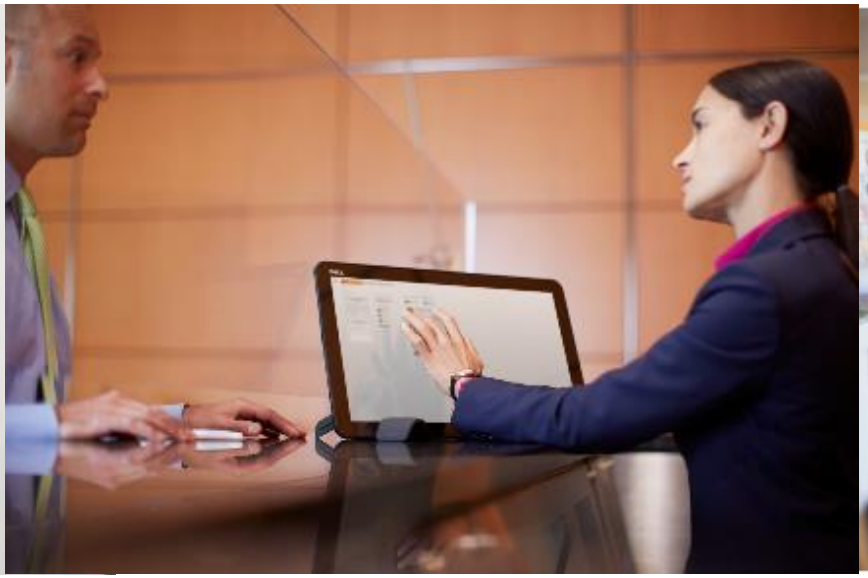
DB + R

What is Customer Churn?

Loss of existing customers or customers switching to a lower value service

(Also referred to as customer attrition)

Managing Customer Churn is Important



Telecommunication

- Customers switching to another mobile service provider,
- Family and Friends switching to another provider
- Corporate customers downgrading least lines

Banking and Finance

- Service Quality
- Attractiveness of banking rates

Retail

- Returning customer
- Designing Loyalty programs
- Long checkout time
- Dis-satisfied with customer service, bad experience in retail store


Customer Profile
(Age, Education, Occupation, Marital status, Gender)
Switching Cost
Number of days till contract termination
Loyalty points, Penalty to switch
Customer Satisfaction
Call Drop Rate, Failure Rate, #Complaints
Social Network
Churn Customer
Usage Trends
Calls last month, Calls Last 6 months,
Data Usage, Unpaid balance
Figuring out
Customer churn

Demo

T-SQL

SQL Server 2016

R Code

App

Database

```
ALTER procedure [dbo].[train_customer_churn_model]
as
begin
execute sp_execute_external_script
  @language = N'R'
, @script = N'
```

R Code

```
'
, @input_data_1 = N'select * from edw_cdr_train_SMOTE'
, @input_data_1_name = N'edw_cdr_train_SMOTE'
, @output_data_1_name = N'rxDForest_model'
with result sets ((model varbinary(max)));
end;
```

R Code

```
ALTER procedure [dbo].[train_customer_churn_model]
as
begin
execute sp_execute_external_script
  @language = N'R'
, @script = N'
         require("RevoScaleR");
        train_vars <- rxGetVarNames(edw_cdr_train_SMOTE)
        train_vars <- train_vars[!train_vars  %in% c("churn")]
        temp<-paste(c("churn",paste(train_vars, collapse="+") ),collapse="~")
        formula<-as.formula(temp)
        forest_model <- rxDForest(formula = formula,
                              data = edw_cdr_train_SMOTE,
                              nTree = 8,
                              maxDepth = 32,
                              mTry = 2,
                              minBucket=1,
                              replace = TRUE,
                              importance = TRUE,
                              seed=8,
                              parms=list(loss=c(0,4,1,0)))
         rxDForest_model <- data.frame(payload = as.raw(serialize(forest_model, connection=NULL)));
'
, @input_data_1 = N'select * from edw_cdr_train_SMOTE'
, @input_data_1_name = N'edw_cdr_train_SMOTE'
, @output_data_1_name = N'rxDForest_model'
with result sets ((model varbinary(max)));
end;
```

Let’s get to where it all started…

SQL Server 2016

Developer Edition

Select an installation type:

Basic

Select Basic installation type to install the SQL Server Database Engine feature with default configuration.

Custom

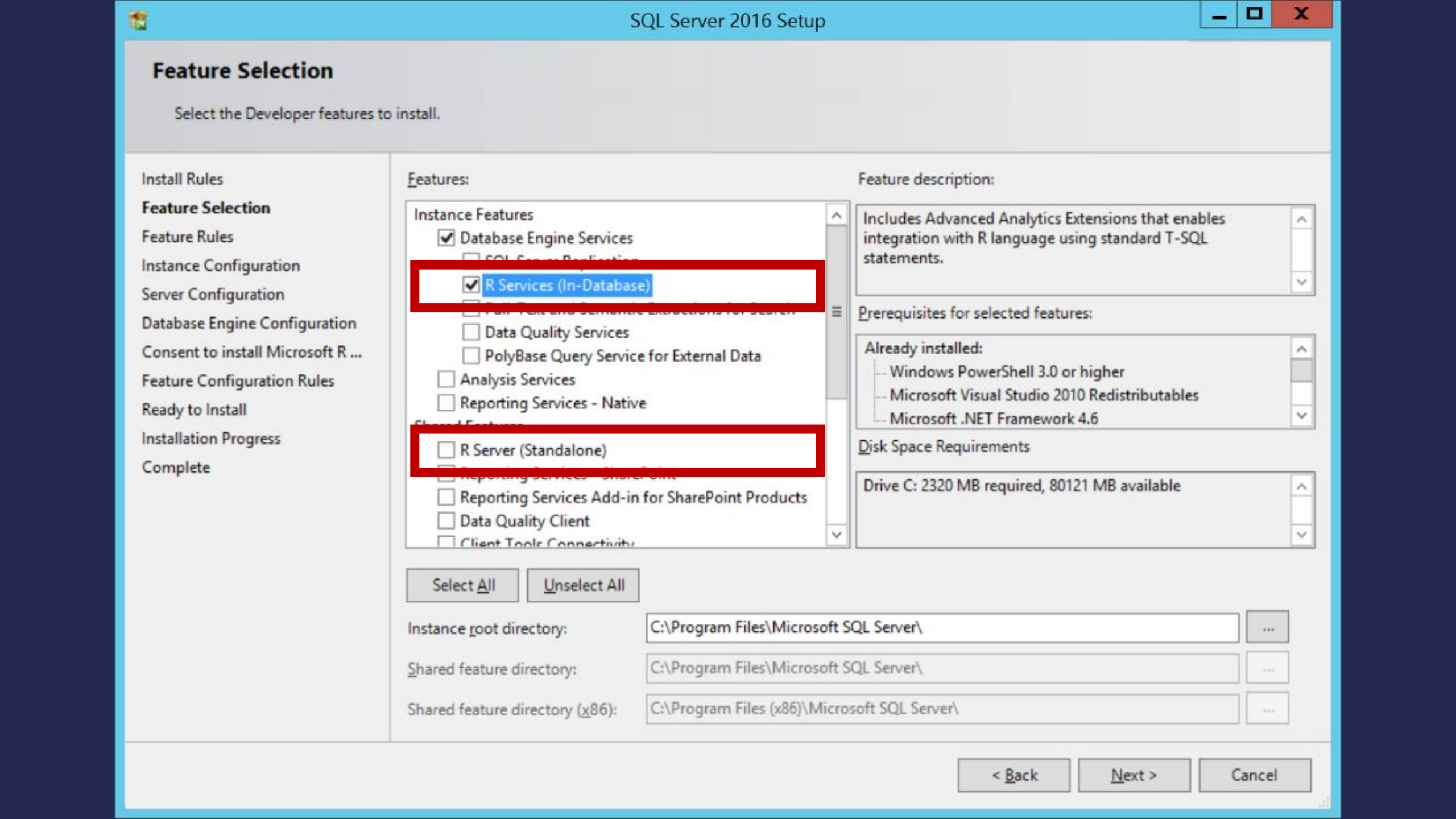
Select Custom installation type to step through the SQL Server installation wizard and choose what you want to install. This installation type is detailed and takes longer than running the Basic install.

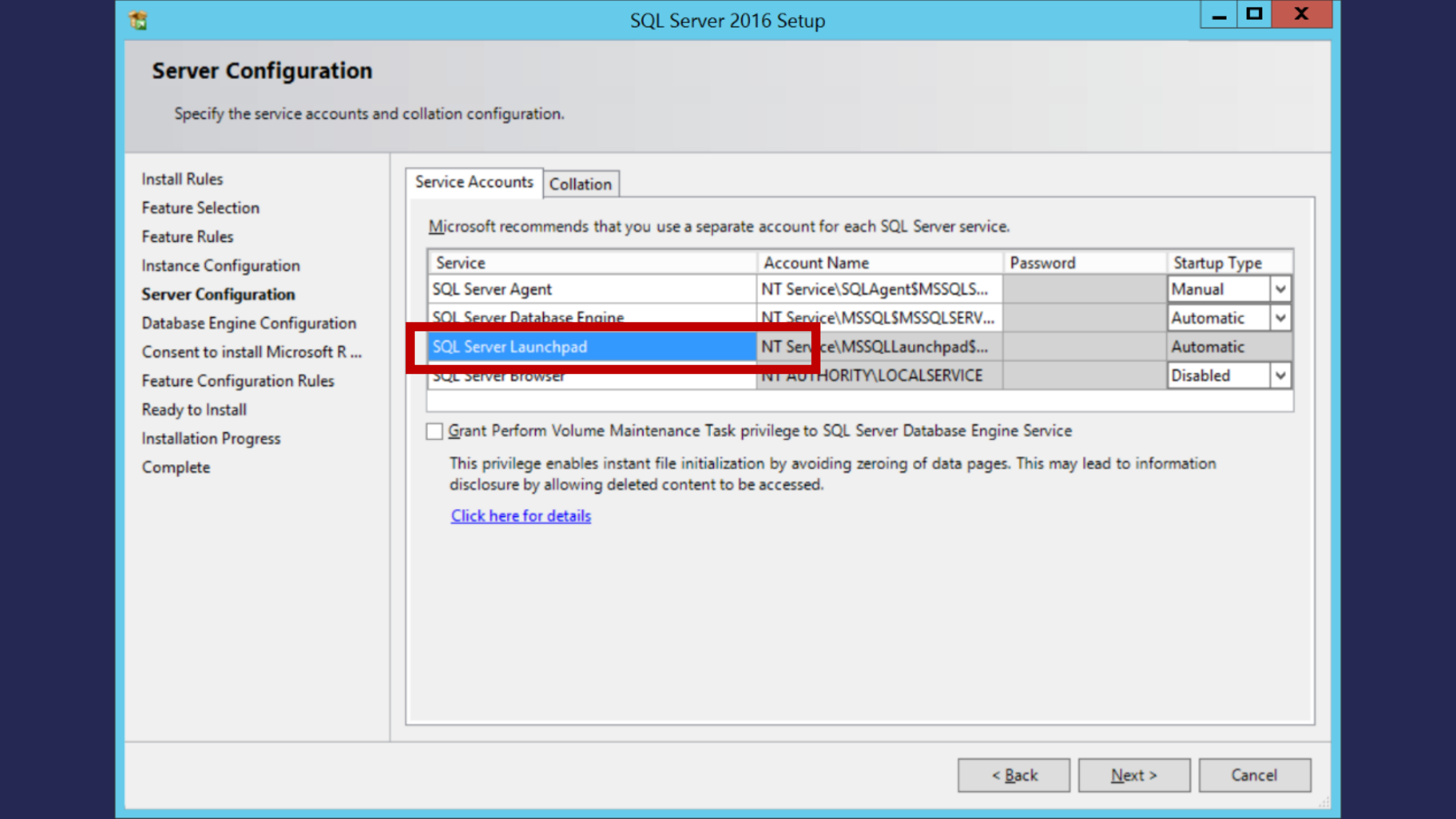
Download Media

Download SQL Server setup files now and install them later on a machine of your choice.

SQL Server 2016 transmits information about your installation experience, as well as other usage and performance data, to Microsoft to help improve the product. To learn more about SQL Server 2016 data processing and privacy controls, please see the Privacy Statement

1.3.3378.1


SQL Server 2016 Setup
Feature Selection
Select the Developer features to install.
Install Rules
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Consent to install Microsoft R ...
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Features:
Instance Features
Database Engine Services
R Services (In-Database)
Data Quality Services
PolyBase Query Service for External Data
Analysis Services
Reporting Services - Native
R Server (Standalone)
Reporting Services Add-in for SharePoint Products
Data Quality Client
Feature description:
Includes Advanced Analytics Extensions that enables integration with R language using standard T-SQL statements.
Prerequisites for selected features:
Already installed:
Windows PowerShell 3.0 or higher
Microsoft Visual Studio 2010 Redistributables
Microsoft .NET Framework 4.6
Disk Space Requirements
Drive C: 2320 MB required, 80121 MB available
Select All
Unselect All
Instance root directory:
C:\Program Files\Microsoft SQL Server\
Shared feature directory:
C:\Program Files\Microsoft SQL Server\
Shared feature directory (x86):
C:\Program Files (x86)\Microsoft SQL Server\
< Back
Next >
Cancel


SQL Server 2016 Setup
Server Configuration
Specify the service accounts and collation configuration.
Install Rules
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Consent to install Microsoft R ...
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Service Accounts
Collation
Microsoft recommends that you use a separate account for each SQL Server service.
Service
Account Name
Password
Startup Type
SQL Server Agent
NT Service\SQLAgent$MSSQLS...
Manual
SQL Server Database Engine
NT Service\MSSQL$MSSQLSERV...
Automatic
SQL Server Launchpad
NT Service\MSSQLLaunchpad$...
Automatic
SQL Server Browser
NT AUTHORITY\LOCALSERVICE
Disabled
Grant Perform Volume Maintenance Task privilege to SQL Server Database Engine Service
This privilege enables instant file initialization by avoiding zeroing of data pages. This may lead to information disclosure by allowing deleted content to be accessed.
Click here for details
< Back
Next >
Cancel

SQL Server 2016 Setup

Consent to install Microsoft R Open

Download and install necessary pre-requisite.

- Install Rules
- Feature Selection
- Feature Rules
- Instance Configuration
- Server Configuration
- Database Engine Configuration
- **Consent to install Microsoft R ...**
- Feature Configuration Rules
- Ready to Install
- Installation Progress
- Complete

Microsoft R Open is an enhanced distribution of R made available by Microsoft under the GNU General Public License v2.

R is © the R Foundation for Statistical Computing. For more information on R-related products and services, visit http://r-project.org.

By clicking "Accept" you are choosing to download Microsoft R Open and install it on your machine, and agreeing to accept patches and updates to this software according to your SQL Server update preferences.

Accept

< Back | Next > | Cancel

Enabling External Scripts

```
Exec sp_configure  'external scripts enabled', 1
Reconfigure  with  override
```

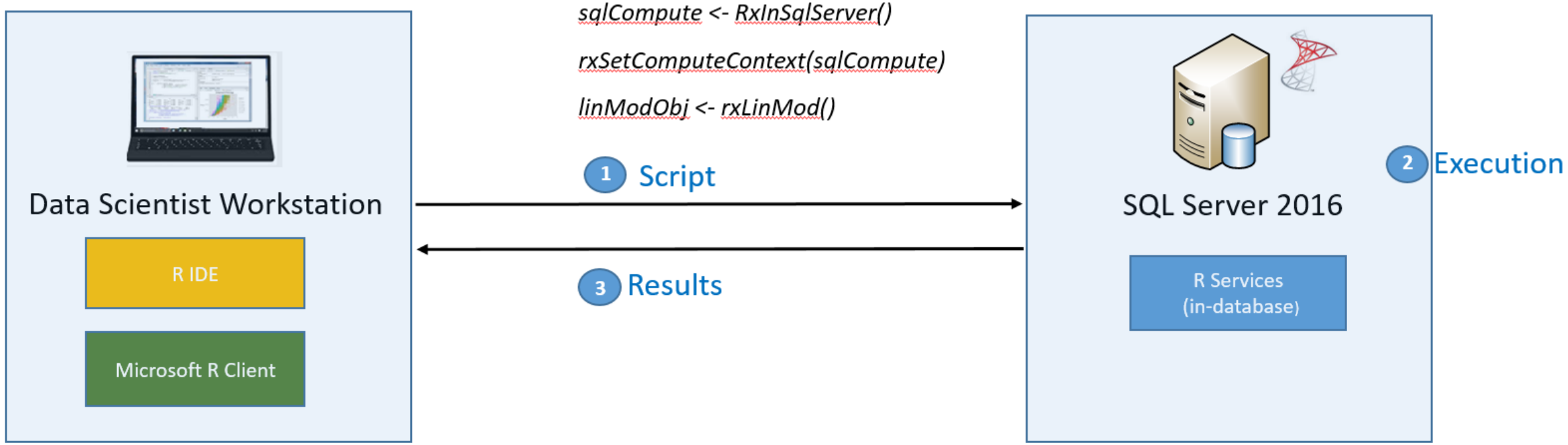
Hello R!

```
exec sp_execute_external_script
     @language =N'R',
     @script=N'OutputDataSet<-InputDataSet',
     @input_data_1 =N'select 1 as hello'
     with result sets (([hello] int not null));
go
```

Will it work for me?

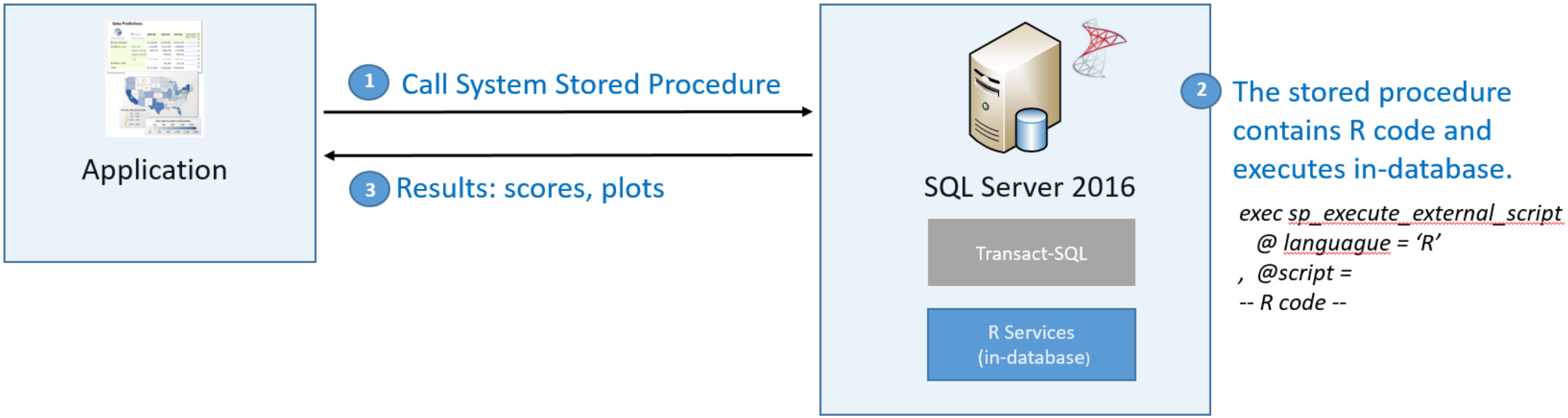
R Services in-database: Data Exploration and Predictive Modeling ('Data Scientist')

Working from my R IDE on my workstation, I can execute an R script that runs in-database, and get the results back.



R Services in-database: Operationalizing R Code via T-SQL ('Developer')

I can call a T-SQL System Stored Procedure from my application and have it trigger R script execution in-database. Results are then returned to my application (predictions, plots, etc).



SQL Server R Services: Intelligence at Your Fingertips

Bring compute to data with In-Database analytics

SQL Server 2016 extensibility mechanism allows secure execution of R scripts on the SQL Server

Operationalize R scripts and models

Use familiar T-SQL stored procedures to invoke R scripts from your application. Embed the returned predictions and plots in your application.

Enterprise Performance and scale

Use SQL Server's in-memory querying and Columnstore Indexes
Leverage RevoScaleR support for large datasets and parallel algorithms with SQL Server 2016 Enterprise Edition.

Get started!

In-database Advanced Analytics with R in SQL Server 2016
http://aka.ms/sql2016r

Sample code and database
http://aka.ms/telcochurnsql2016sample

MSDN Documentation
https://msdn.microsoft.com/en-us/library/mt604845.aspx