Microsoft

MRS WebServices Operationalisation Features

Contents

Overview

Overview

In this lab we will explore the functionality of the mrsdeploy R package. You will learn how to use this package to connect remotely to the R server 9.0.1 instance and configure, use and publish webservices to operationalized an R model.

The mrsdeploy package is installed as part of R Server 9.0 or R Client on all supported platforms but you must have R Server to leverage its functionality.

This lab focuses on describing the main features of mrsdeploy, namely remote execution and webservices deployment as well as describing its configuration and architecture.

Prerequisites

Basic knowledge of R and understanding of Web Deployment concepts such as web services and APS.NET.

Either a VM or a local installation of Microsoft R Server 9.0.1 and optionally Microsoft Client 3.3.2.

Introductory video Operationalization with R Server.

System requirements

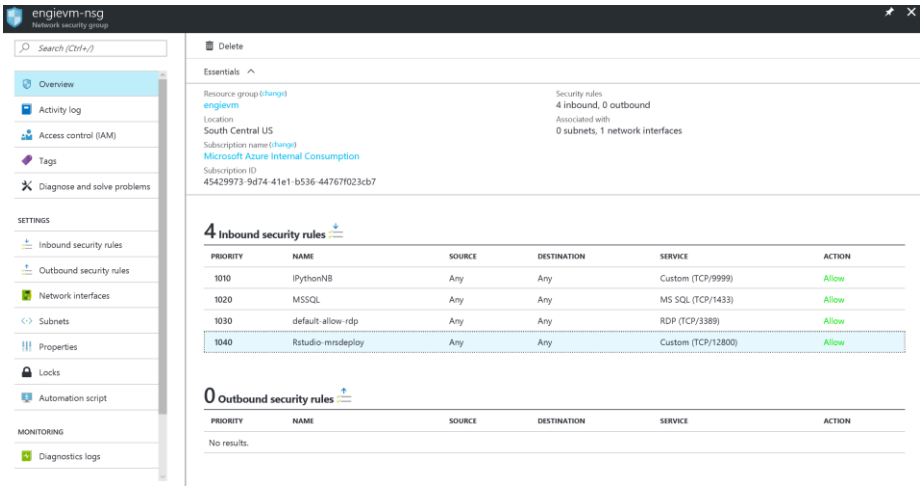
Provisioning the Data Science VM from Azure.

If using the MRC to connect to the VM then the VM needs to have its webservices deployment port opened as follows:

On the VM main page go to Network Interface, and click on the left hand side on Network Security Groups, add a rule to the Inbound Security Rules to allow access from the client to the webservices port using TCP protocol:

+Add>Inbound security rule for port 12800 TCP protocol (12800 is default after configuration but can be changed).

.



MRS and MRC: Basic Architecture

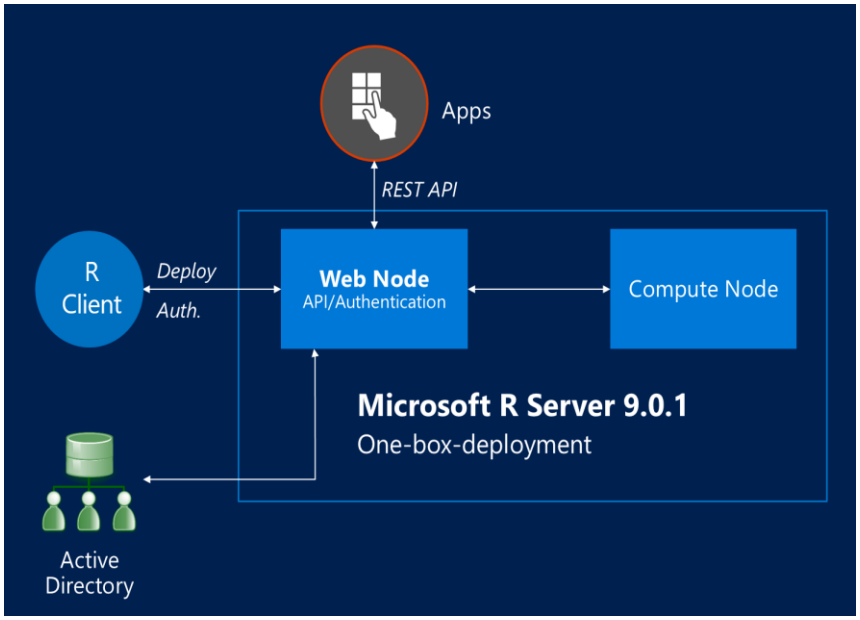
The basic installation of Microsoft R Server installs the R executable and the MRS packages as well as the necessary software to run and deploy Webservices as listed below:

Component	Version
Microsoft AS OLE DB Provider for SQL Server 2016	13.0.1601.5
Microsoft .NET Core	1.0.1
Microsoft MPI	7.1.12437.25
Microsoft Visual C++ 2013 Redistributable	12.0.30501.0
Microsoft Visual C++ 2015 Redistributable	14.0.23026.0

However, we can configure MR as a dedicated node providing either webservices or compute services. In this web, many nodes providing similar services can be grouped in clusters and front-ended with a Load Balancer.

All configurations have at least a single web node and single compute node:

- A web node acts as an HTTP REST endpoint with which users can interact directly to make API calls. The web node accesses data in the database, and send jobs to the compute node.
- A compute node is used to execute R code as a session or service. Each compute node has its own pool of R shells.



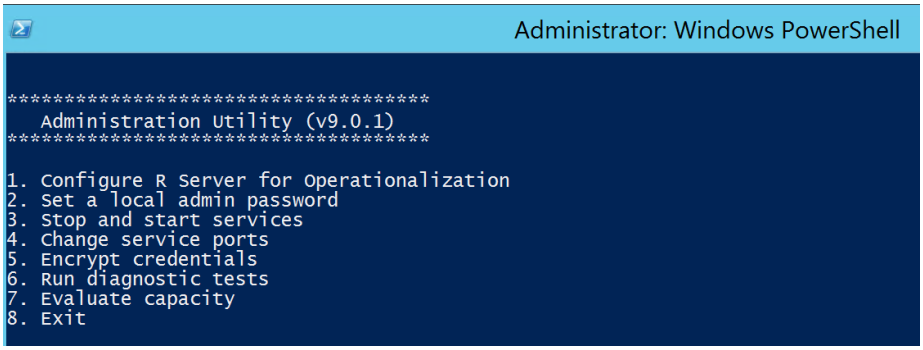
In order to provide the services required for web deployment, we'll need to run the Admin configuration tool (on powershell). The Admin Tool enables and configures IIS and ASP.NET (these are normally available by default on Windows Server) for remote management of IIS websites and exposes the Web Deploy Handler endpoint to clients. The details of the configuration within the webserver/application server are beyond the scope of this guide.

MRS Deployment configuration

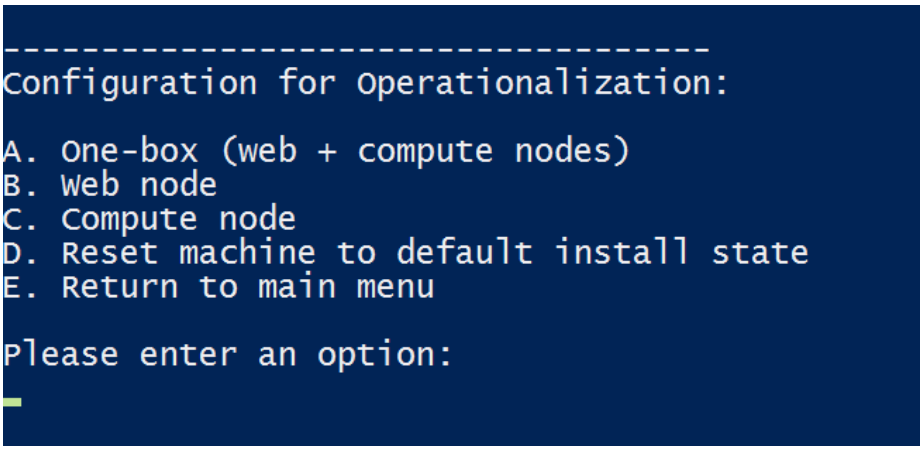
The configuration of MRS nodes is done via the powershell Administration Tool. The Tool is located under:

C:\Program Files\Microsoft\R Server\R_SERVER\DeployR

Right click on the file runAdminUtils and run it in Powershell. You should get a blue command prompt:



→Select option 1. Configure R Server for Operationalization



→Select A.One-box (web + compute nodes).

Running this option will configure the Endpoint and the password for authentication of the remote execution and webservices.

The default url for endpoint is (but the port can be changed): http://localhost:12800/

Remote Execution I
Login

Remote login

The remote command line allows you to directly interact with an R Server 9.0.1 instance on another machine. You can enter 'R' code just as you would in a local R console. R code entered at the remote command line executes on the remote server.

Requirements for using this feature include:

- All nodes must be version 9.0.1.
- R Server must be operationalized before it can host remote sessions invoked from another machine.

```
#At the R command prompt

endpoint <- "http://hostname:12800" #Your VM hostname or Public IP ad
dress

remoteLogin(endpoint, session = TRUE, diff = TRUE,commandline=TRUE)
```

*Please note that the parameter session=TRUE allows for the connection and remote execution.

**The parameter diff=TRUE creates a report of the differences between the local environment and the remote MRS environment.

***A `diff` report is available so you can see and manage differences between the local and remote R environments. The diff report contains information regarding R versions, R packages installed locally, but not on the remote session, and differences between R package versions. This report is shown by default when you log in, but can be run anytime by executing the function: `diffLocalRemote()`.

For this exercise we will load some data from the local session to the VM where MRS is running. Then we will transform the data by executing R on the server.

Remote Execution II
Transfers

Transfers: Files, Objects and Workspaces

MRS remote execution allows for the transfer of either files, R objects or complete R Workspaces from the local environment to the MRS server and vice versa. All within the comfort of the MRS remote execution!

File Transfer

```
#Use R file MRSLAB09_RE_FileTransfer.R included with this LAB
#create iris.csv in remote environment
write.csv(iris,file = "iris.csv")

#Other commands
listRemoteFiles()
deleteRemoteFile("iris.csv")
listRemoteFiles()

#create iris.csv locally
write.csv(iris,file = "iris.csv")
#Check file is there
dir()

#Transfer files form local environment to server
putLocalFile("iris.csv")

#Other commands
listRemoteFiles()
deleteRemoteFile("iris.csv")
listRemoteFiles()
```

Object Transfer

```
##Use R file MRSLAB09_RE_FileTransfer.R
```

```
#Tranfer any local R object to server

IRIS<-iris

#Check IRIS is a data.frame
str(IRIS)

#Send local object to remote VM
putLocalObject("IRIS",name="remoteIRIS")

#Delete local object
rm(IRIS)

#get remote object to local environment

getRemoteObject("remoteIRIS")
```

Workspace Transfer

```
#Use R file MRSLAB_RE_FileTransfer.R
#Take all objects form the local R session and load them into the rem
ote R session

putLocalWorkspace()
```

```
#Take all objects from the remote R session and load them into the lo
cal R session.

getRemoteWorkspace()
```

Remote Execution III Snapshots

Remote environment Snapshots

A snapshot is an image of a remote R session saved to Microsoft R Server, which includes:

- The session's workspace along with the installed R packages
- Any files and artifacts in the working directory

A snapshot can be loaded into any subsequent remote R session for the user who created it. For example, suppose you want to execute a script that needs three R packages, a reference data file, and a model object. Instead of loading these items each time you want to execute the script, create a snapshot of an R session containing them. Then, you can save time later by retrieving this snapshot using its ID to get the session contents exactly as they were at the time the snapshot was created.

Snapshots are only accessible to the user who creates them and cannot be shared across users.

The following functions are available for working with snapshots: `listSnapshots()`, `createSnapshot()`, `loadSnapshot()`, `downloadSnapshot()` and `deleteSnapshot()`.

```
#Use R file MRSLAB_SN_Snapshots.R

# show that the server R environment is empty

ls()
pause()

#Create a snapshot – give it a name description
#please note snapshots are manipulated by ID

Snap1<-createSnapshot("SNAP1")

# list out the snapshots
#Snapshots can be loaded by ID not by Name
snaps <- do.call(rbind.data.frame, listSnapshots())

#Check your created snap is there
snaps

# load in a snapshot, resume the remote session and look at the R env
ironment
loadSnapshot(snaps$id[1])
resume()
ls()
pause()

# download the snapshot from the server
downloadSnapshot(snaps$id[1], file = "mySnap.zip")
unzip("mySnap.zip")
```

WebServices Data Prep & modeling

Putting it all together: Data prep and modeling

```
#Use R file MRSLAB09_webservicesDeploy.R
#If not already installed on the server install**

#Load packages remotely
#If not already installed on the server install uncomment the code be
low and execute
#X <- c("stringr","plyr","lubridate","randomForest","reshape2","ggplo
t2")
#lapply(X, install.packages, character.only = T)
#lapply(X,library,character.only=T)


library(stringr)
library(plyr)
library(lubridate)
library(randomForest)
library(reshape2)
```

```
library(ggplot2)

#pause the R session
pause()

#getting the local directory
getwd()

#upload the local file into the MRS on the VM
putLocalFile("C:/Downloads/LoanStats3a.csv")

#Resume session on the server
resume()
```

**Please note that the REMOTE session has only read access to the library directory hence the installation of packages will have to be performed on the server.

Now start the transformations on the server by executing commands via the >REMOTE prompt

```
#Check file is there
dir()

#Get exact directory path where we uploaded the csv file
getwd()


# Read from csv file and create your model remotely
#*****************************************************************
#The below is a directory example, replace it
#with the output of getwd()

loans <- read.csv("C:/Program Files/Microsoft/R Server/R_SERVER/Deplo
yR/rserve/workdir/conn2232/LoanStats3a.csv", h=T, stringsAsFactors=F,
skip=1)

#take a peak...
head(loans,5)

#annoying column; just get rid of it
loans[,'desc'] <- NULL

summary(loans)
#almost all NA, so just get rid of it
loans[,'mths_since_last_record'] <- NULL

#get rid of fields that are mainly NA
poor_coverage <- sapply(loans, function(x) {
  coverage <- 1 - sum(is.na(x)) / length(x)
  coverage < 0.8
})
loans <- loans[,poor_coverage==FALSE]

bad_indicators <- c("Late (16-30 days)", "Late (31-120 days)", "Defau
lt", "Charged Off")

loans$is_bad <- ifelse(loans$loan_status %in% bad_indicators, 1,
                       ifelse(loans$loan_status=="", NA,
                              0))
table(loans$loan_status)
table(loans$is_bad)

head(loans,5)

loans$issue_d <- as.Date(loans$issue_d,"%m/%d/%y")
```

```
loans$year_issued <- year(loans$issue_d)
loans$month_issued <- month(loans$issue_d)
loans$earliest_cr_line <- as.Date(loans$earliest_cr_line,"%m/%d/%y")
loans$revol_util <- str_replace_all(loans$revol_util, "[%]", "")
loans$revol_util <- as.numeric(loans$revol_util)
loans$int_rate<- as.numeric(loans$int_rate)


# Select Train data
idx <- runif(nrow(loans)) > 0.75
train <- loans[idx == FALSE,]

#Train the model
loanDataModel <- rxDForest(is_bad ~ revol_util + int_rate + annual_in
c + total_rec_prncp , train)
```

Web Services Publishing & consuming

Putting it all together: publishing and consuming webservices

Now that the model has been trained we want to take a snapshot of the MRS environment we are working on with its functions and packages. See snapshots section for more information.

```
#stop the remote session
pause()

#the snapshot function will take a snapshot from the local session of
#the remote MRS session running on the VM
snapshot<-createSnapshot("loanPredictSnapshot")

#have a look at the content . It should be a string of characters
Snapshot

# Then create locally the Scoring Function

loanPredictService <- function(revol_util, int_rate, annual_inc, tota
l_rec_prncp) {
  inputData <- data.frame(revol_util = revol_util, int_rate = int_rat
e, annual_inc = annual_inc,  total_rec_prncp = total_rec_prncp)
  prediction <- rxPredict(loanDataModel, inputData)
  loanScore <- prediction$is_bad_Pred
}
#resume the session
resume()

#Exit will log out the session completely
exit
```

Now we will log back into the Web Server node (in this case the MRS VM has been configured as one-box with both webservices and compute node on the same VM.

```
# Remote Login, a prompt will show up to input user and pwd informati
on

endpoint <- "http://hostname:12800"
#Your VM hostname or Public IP address

remoteLogin(endpoint, session = FALSE, diff = FALSE)
#please note that to deploy webservices the session parameter needs #
to be set to FALSE

# pseudo `unique` service name so no collision in example
service_name <- paste0("loanPredictService", round(as.numeric(Sys.tim
e()), 0))

# Publish service
```

```
api <- publishService(
  service_name,
  code = loanPredictService,
  snapshot=snapshot,
  inputs = list(revol_util = 'numeric', int_rate = 'numeric', annual_
inc = 'numeric', total_rec_prncp = 'numeric'),
  outputs = list(loanScore = 'numeric'),
  v = 'v2.0.0'
)


# Show API capabilities
api$capabilities()

#Consume the service
system.time(loanScore <- api$loanPredictService(1, 1, 1, 1))

loanScore$output("loanScore")

#List all services
services <- listServices()
services

#Generate swagger json file
cat(api$swagger(), file = "C:/DIRECTORY/loanPredict.json")

#Logout
remoteLogout()
```

Swagger

The **OpenAPI Specification** (originally known as the Swagger Specification) is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful Web services.

A variety of tools can generate code, documentation and test cases given an interface file.

Development of the OpenAPI Specification (OAS) is overseen by the Open API Initiative, an open source collaborative project of the Linux Foundation

Swagger Editor
Swagger UI

Terms of Use

By using this Hands-on Lab, you agree to the following terms:

The technology/functionality described in this Hands-on Lab is provided by Microsoft Corporation in a "sandbox" testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the Hands-on Lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. You may not modify copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this Hands-on Lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCITONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

FEEDBACK. If you give feedback about the technology features, functionality and/or concepts described in this Hands-on Lab to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.

DISCLAIMER

This lab contains only a portion of the features and enhancements in Microsoft Azure. Some of the features might change in future releases of the product.