**How to set up VM and deploy model as a service on Azure Linux VM**

**Linux VM Konfiguration on Azure**

1. **Linux VM in Azure Portal erstellen:** Microsoft R Server (Version 9.1.0) for Linux (Ubuntu version 16.04), 2 cores, 16GB
2. **Ports im Azure Portal freigeben**: bei Dashboard und der entsprechenden VM auf Netzwerk-Tab klicken -> Regel für eingehenden Port hinzufügen, Ports 80, 12800, 8000, 8080, 8004, 443 freigeben
3. **DNS Namen für VM erstellen (weil IP-Adresse beim Herunterfahren jeweils wechselt**): In Übersichts-tab auf DNS-Name erstellen klicken
4. **Mit VM via Git Bash Shell verbinden:** ssh benutzernamen@IP-Adresse oder ssh benutzernamen@dnsname (bei uns: ssh [soga@lin-op-vm.westeurope.cloudapp.azure.com](mailto:soga@lin-op-vm.westeurope.cloudapp.azure.com))

(beforehand: switch Zühlke VPN off!)

1. **R Server konfigurieren:**

Führe folgende Befehle in mit VM verbundener Shell aus:

* cd /usr/lib64/microsoft-r/rserver/o16n/9.1.0/Microsoft.RServer.Utils.AdminUtil
* sudo dotnet Microsoft.RServer.Utils.AdminUtil.dll
* Folge Anweisungen in Link <https://docs.microsoft.com/en-us/machine-learning-server/install/operationalize-r-server-one-box-config> -> How to perform a one-box configuration: (für R Server 9.1.0), Punkt 2

1. **Firewall settings der VM anpassen**

* Siehe File HowTo\_Change\_Firewall\_Settings\_of\_Linux\_VM

1. **Get your files from your local machine to your vm:** Clone from github or bitbucket repository to copy your whole project to the vm
2. **Shut down VM after usage** (!!!): Go to overview (in azure dashboard) and klick on Beenden

<https://buildazure.com/2017/03/16/properly-shutdown-azure-vm-to-save-money/>

Be aware of the following: Since this causes Azure to release the server resources associated with the Virtual Machine, it not only releases the CPU and Memory resources but also the Dynamic IP Address allocation. Due to this, when you Start the VM back up again, the IP Address will likely change. If you require the IP Address to never change for your VM, then you’ll need to configure a Static IP Address for the VM.

**If you want to deploy model as a service using plumber API:**

Start R on VM in shell using sudo R (otherwhise plumber API creation not working)

**Docker installation auf ubuntu:**

<https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/#os-requirements>

**If you want to deploy model as a service using MS R Server Framework:**

Client side (on your local machine)

1. Install R client on your laptop, available at:

<https://docs.microsoft.com/en-us/machine-learning-server/r-client/install-on-windows>

1. Use the MS R Open distribution instead of the open source version of R in RStudio:

Tools -> Global Options -> General -> R Version

Old, not needed: Server side (on VM)

1. Install the necessary packages in the correct library (the one which the REST Apis use):

cd /usr/lib64/microsoft-r/rserver/o16n/9.1.0/rserve

sudo R

install.packages(“randomForest”)

library(“randomForest”)

Client side (on your local machine) TO DO: UPDATE, see r file ms\_rclient.R

1. Old: Hack to install necessary packages in R Server (everything installed is temporary, per login-connection)
2. Deploy model as a service (using data.frames as input) as described in

<https://docs.microsoft.com/de-de/machine-learning-server/operationalize/how-to-deploy-web-service-publish-manage-in-r#workflow>

* Section5 (you don’t need to use the rx-functions of the revoscaleR package)

Small example:

# Create api which installs packages:

remoteLogin("http://lin-op-vm.westeurope.cloudapp.azure.com:12800",

session = FALSE)

## hack to install the necessary packages ####

# works when this api belongs to same remoteLogin-connection as the prediction apis

install.pkg <- function(num){

install.packages("randomForest") # NEEDED ALWAYS!!

library(randomForest)

0

}

api\_installpkg <- publishService(

"installpkg",

code = install.pkg,

inputs = list(num = "numeric"),

outputs = list(answer = "numeric"),

v = "v1.0.0"

)

# api-call

installres <- api\_installpkg$install.pkg(9)

modelsmall <- readRDS(file = "models/model\_rf\_1000.rds")

predictsmall <- function(dataframe){

predict(modelsmall, newdata = dataframe, type = "response")

}

apismalldf <- publishService(

"modelsmalldf",

code = predictsmall,

model = modelsmall,

inputs = list(dataframe = "data.frame"),

outputs = list(answer = "numeric"),

v = "v1.0.0"

)

# post call to REST Apis

dtest <- readRDS("mnist\_dataframes/mnist\_test\_dataframe.rds")

result <- api\_small$predictsmall(dtest[1,-785])

str(result)

**Infos to R-Server Directory on VM**

/usr/lib64/microsoft-r/rserver/o16n/9.1.0/rserve

/Rserv.conf: Configuration of R Server, executed when starting R Server (set working directory, encoding, source RScripts/source.R)

/RScripts/source.R: Configuration of R before starting server (vi source.R)

/workdir/Rserv9.1.0 contains the temporary libraries (one per connection)

This is not the same as “normal” R library:

usr/lib64/microsoft-r/3.3/lib64/R/library

**Infos about remote execution (remote r session)**

remoteLogin("http://lin-op-vm.westeurope.cloudapp.azure.com:12800",

session = TRUE)

<https://docs.microsoft.com/en-us/machine-learning-server/r/how-to-execute-code-remotely>

<https://tomaztsql.wordpress.com/2017/06/14/installing-r-packages-with-rxinstallpackages-in-microsoft-r-server/>

<https://docs.microsoft.com/en-us/machine-learning-server/r-reference/revoscaler/rxinstallpackages>

**Infos about R package management in R Server (not very helpful up to now)**

https://docs.microsoft.com/en-us/machine-learning-server/operationalize/configure-manage-r-packages#mrsdeploy

**How to do a POST request from Postman:**

Setup Postman as described in

<https://blogs.msdn.microsoft.com/mlserver/2017/02/22/rest-calls-using-postman-for-r-server-o16n-2/>

Data frame representation in Json is given at the bottom of:

<https://blogs.msdn.microsoft.com/mlserver/2017/02/22/rest-calls-using-postman-for-r-server-o16n-2/>

**consume APIs from other languages (Authentification etc)**

<https://docs.microsoft.com/de-de/machine-learning-server/operationalize/how-to-build-api-clients-from-swagger-for-app-integration>