I’ll give a short introduction on how to use AzureVM to manage Azure virtual machines, and in particular [Data Science Virtual Machines](https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/) (DSVMs).

**Creating a VM**

Creating a VM is as simple as using the create\_vm method, which is available as part of the az\_subscription and az\_resource\_group classes.

library(AzureRMR)

library(AzureVM)

## using the subscription method

sub <- az\_rm$

new(tenant="{tenant\_id}", app="{app\_id}", password="{password}")$

get\_subscription("{subscription\_id}")

myNewVM <- sub$create\_vm("myNewVM",

location="australiaeast",

username="datascience",

passkey="fAs30q-2a5vF!Z") # be sure to choose a strong password!

## using the resource group method

rg <- sub$create\_resource\_group("myresourcegroup",

location="australiaeast")

myOtherVM <- rg$create\_vm("myOtherVM",

username="datascience",

passkey="l3Kgrf21%?0DFm")

Without any other options, this will create a Windows Server 2016 [Data Science Virtual Machine](https://docs.microsoft.com/azure/machine-learning/data-science-virtual-machine/overview?WT.mc_id=Revolutions-blog-davidsmi#whats-included-in-the-data-science-vm), which is pre-installed with several tools useful for analytics: Python, R (and RStudio), Tensorflow, XGBoost, SQL Server, and so on. The size will be a Standard DS3 V2 VM, which has 4 cores, 14GB of memory, 1TB primary disk, and up to 16x28GB data disks.

A feature of creating the VM under a subscription, as opposed to a resource group, is that it creates a new resource group specifically to hold the VM. This simplifies the task of managing and (eventually) deleting the VM considerably. See “Deleting a VM” below.

You can change the specifications for the VM by providing any of the following arguments:

* os: either “Windows” or “Ubuntu” (for Ubuntu LTS 16.04).
* size: the VM size. Use the az\_subscription$list\_vm\_sizes() method to see what sizes are available in your region. Note that in Azure, a VM’s size is really a broad-ranging label that encapsulates the number of cores, memory, and disk available.
* passkey: if creating an Ubuntu VM, you can supply a public key as the passkey argument.
* userauth\_type: set this to “key” if you supply a public key.

Setting these will determine whether you get a Windows or Ubuntu DSVM, the login details, and how powerful the VM is in terms of cores and memory/disk capacity. For example, this will create a Linux NC-series DSVM using public key authentication:

myLinuxVM <- sub$create\_vm("myLinuxVM",

location="australiaeast",

size="Standard\_NC6s\_v3",

os="Ubuntu",

username="datascience",

passkey=readLines("~/id\_rsa.pub"),

userauth\_type="key")

The list\_vm\_sizes() method will show you what the [available VM sizes](https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes) are for your region.

# examine VM sizes available in australiaeast region

sub$list\_vm\_sizes("australiaeast")

**Retrieving an existing VM**

If you have an existing VM, you can retrieve it with the get\_vm() method. As with create\_vm(), this is available as part of the az\_subscription and az\_resource\_group classes. The only argument you need to supply is the name of the VM.

## using the subscription method

sub <- az\_rm$

new(tenant="{tenant\_id}", app="{app\_id}", password="{password}")$

get\_subscription("{subscription\_id}")

# retrieve the VM we created above

myNewVM <- sub$get\_vm("myNewVM")

## and with the resource group method

rg <- sub$get\_resource\_group("myresourcegroup")

myOtherVM <- rg$get\_vm("myOtherVM")

**Working with a VM**

There are various things you can do with a VM object within R.

To stop (shutdown) a VM, call its stop() method. The deallocate argument sets whether to deallocate its resources as well, with the default being TRUE; you may want to set this to FALSE if you know you will be restarting the VM soon. To restart it, call either the start() or restart() method. The main difference between the two is that restart() will shutdown the VM first if it is currently running.

To sync the object with the resource in Azure, call the sync\_vm\_status() method. The most common situation where you might want to do this is if you create a VM with the argument wait=FALSE. In this case, rather than waiting for provisioning to complete, the create\_vm method will return an incomplete VM object; you then call its sync\_vm\_status() method to update it with how the provisioning is going in Azure.

To dynamically resize a VM, call the resize() method with the new size. This has an optional deallocate argument controlling whether to stop and deallocate the VM first (which is sometimes necessary for successful resizing).

To run a script in the VM (without manually logging in), call the run\_script() method. This will be a PowerShell script if it is a Windows VM, or a bash script if it is Linux. The script is just a character vector.

# simple bash script for executing on a Linux VM

script <-

'#!/bin/bash

var=Hello world!

# redirect output to a file so we know whether it ran successfully

echo "$var" > /tmp/helloworld.txt

'

vm$run\_script(script)

If you login to the VM after running this script, you should find the file helloworld.txt in the /tmp directory.

**Deleting a VM**

Simply deleting a virtual machine object in R, eg with rm(), will not do anything to the VM itself in Azure. To delete the VM and its resources, call the object’s delete() method:

vm$delete()

AzureVM will prompt you for confirmation that you really want to delete the VM. By default, this will also free up all the individual Azure resources used by the VM, such as its storage, network interface, security group, and so on.

If you created the VM using the create\_vm() method of the az\_subscription class, the deletion process is very simple: it simply removes the resource group containing the VM. This is possible because the resource group was created as part of deploying the VM. Be aware that any other resources you may have created in this resource group will also be deleted.

Alternatively, you can use the delete\_vm() method of the az\_subscription and az\_resource\_group classes. These will retrieve the VM of the given name and then call its delete() method.

**GPU enabled VMs**

If you are running deep learning workloads, you’ll want to ensure that your VM is GPU-enabled. In Azure, the various [NC- and ND-series VMs](https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-gpu) are designed for these workloads. You can create a GPU-enabled VM by setting the size argument appropriately, for example

myGpuVM <- sub$create\_vm(size="Standard\_NC12s\_v3", ...)

However, the following caveats apply to GPU-enabled VMs: – Not all regions have GPUs available. To check on availability, use the az\_subscription$list\_vm\_sizes() method and provide your region. – Currently, the supply of GPUs is limited. You must [apply for a quota increase](https://docs.microsoft.com/en-us/azure/azure-supportability/resource-manager-core-quotas-request) if you want to deploy a GPU-enabled VM.

**VM clusters**

You can work with VM clusters (a collection of VMs sharing the same configuration) by using the get\_vm\_cluster, create\_vm\_cluster and delete\_vm\_cluster methods. These are almost identical to get\_vm, create\_vm and delete\_vm with the addition of a clust\_size argument that sets the size of the cluster.

sub <- az\_rm$

new(tenant="{tenant\_id}", app="{app\_id}", password="{password}")$

get\_subscription("{subscription\_id}")

# create a cluster of 5 Ubuntu VMs

vmCluster <- sub$create\_vm\_cluster("vmCluster",

location="australiaeast",

os="Ubuntu",

username="datascience",

passkey=readLines("~/id\_rsa.pub"),

userauth\_type="key",

clust\_size=5)

Most things that you can do with a single VM, you can also do with a VM cluster. For example, running a script with the run\_script() method will run the script on all the VMs in the cluster. Starting, stopping and restarting a cluster similarly carries out the given action on all the VMs.