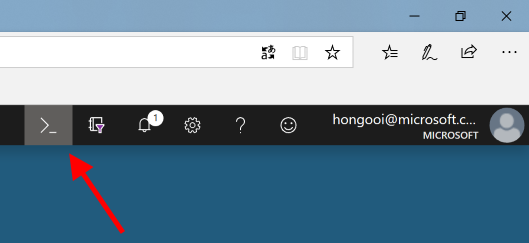
**Before you begin**

The first thing you have to do is create a [service principal](https://docs.microsoft.com/azure/active-directory/develop/app-objects-and-service-principals?WT.mc_id=Revolutions-blog-davidsmi#service-principal-object). This is a security identity used by apps, services and automation tools for programmatic access to Azure resources. You can think of it as a “user identity” (login and password or certificate) with a role and permissions to access your resources. Creating a service principal is a one-time task, after which you can reuse it for all your AzureRMR sessions.

There are various ways to create a service principal. The *easiest* is to have your network administrator create it for you (which is probably also the best option in a corporate environment), but if you’re doing it yourself, you can do it via the Azure cloud shell.

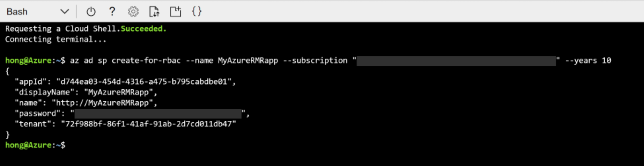
* In the Azure Portal (<https://portal.azure.com/>), click on the Cloud Shell icon:

[](https://revolution-computing.typepad.com/.a/6a010534b1db25970b022ad3a13986200d-pi)

* If you haven’t used the shell before, there will be a dialog box to choose whether to use bash or PowerShell. Choose bash.
* In the shell, type
* az ad sp create-for-rbac --name {app-name} --subscription "{your-subscription-name}" --years {N}

substituting the desired name of your service principal (try to make it memorable to you, and unlikely to clash with other names), your subscription name, and the number of years you want the password to be valid.

* Wait until the app creation is complete. You should see a screen like this.

[](https://revolution-computing.typepad.com/.a/6a010534b1db25970b022ad37b12b7200c-pi)

* Record your tenant ID, app ID, and password.

If you want to allow access at something other than subscription level, you can use the --scopes argument in place of --subscription. For example, to restrict AzureRMR to only the “AnalyticsRG” resource group:

az ad sp create-for-rbac --scopes /subscriptions/{your-subscription-ID}/resourceGroups/AnalyticsRG

This will grant "Contributor" access to the specified subscription/resources, which should be sufficient for anything you’d want to do in R.

**Using AzureRMR**

AzureRMR is a general-purpose interface to Azure Resource Manager (ARM). You can use it to work with nearly any Azure object that ARM can handle: subscriptions, resource groups, resources, templates and so on. The things you can do with AzureRMR include:

* Create a new resource
* Carry out arbitrary operations on a resource
* Delete a resource
* Deploy a template
* Delete a template, and, optionally, any resources that it created
* Create and delete resource groups (if you gave your service principal subscription-level access)

Here is some example code to show AzureRMR in action. Let’s retrieve a resource (in this case, a storage account):

library(AzureRMR)

az <- az\_rm$new(tenant="{tenant\_id}",

app="{app\_id}",

password="{password}")

sub <- az$get\_subscription("{subscription\_id}")

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

stor <- rg$get\_resource(type="Microsoft.Storage/storageAccounts",

name="mystorage")

Step by step:

1. The first thing we do is login to Azure via az\_rm$new(). This authenticates with Azure Resource Manager and stores a token which will be used for further interactions with ARM. AzureRMR will automatically handle such issues as token renewal on expiry.
2. Given the session object, retrieve an Azure subscription with the get\_subscription() method and store its details in a subscription object. Notice that AzureRMR uses R6 classes, so the method is part of the object itself.
3. Given a subscription object, retrieve the details for a resource group with the get\_resource\_group() method.
4. Given a resource group object, retrieve the details for a resource and store them into a resource object. There are many ways of identifying a resource; here, we use the type and the name.

This replicates the hierarchical structure of Azure, whereby subscriptions contain resource groups, which contain individual resources.

Once we have a resource, we can do *things* with it, via the do\_operation() method. In this case, we have a storage account. One of the things we can do with a storage account is retrieve its access keys.

stor$do\_operation("listKeys", http\_verb="POST")

Here’s another example. If we have a virtual machine, we can start it, execute shell commands, and then shut it down again:

vm <- rg$get\_resource(type="Microsoft.Compute/virtualMachines",

name="myVirtualMachine")

vm$do\_operation("start", http\_verb="POST") # may take a while

vm$do\_operation("runCommand",

body=list(

commandId="RunShellScript", # RunPowerShellScript for Windows

script=as.list("ifconfig > /tmp/ifconfig.out")

),

encode="json",

http\_verb="POST")

vm$do\_operation("powerOff", http\_verb="POST")

For the types of operations you can carry out on a resource, consult the [Azure REST API documentation](https://docs.microsoft.com/en-us/rest/api/?view=Azure).

You can also interrogate the fields of a resource object; in particular the properties field can contain arbitrary information about an Azure resource. For example, a storage account’s properties includes the endpoint URIs, and a virtual machine’s properties includes its admin login details.

# file and blob storage endpoint

stor$properties$primaryEndpoints$file

stor$properties$primaryEndpoints$blob

# OS profile for a VM: includes login details

vm$properties$osProfile

To create a new resource, we can use the resource group object’s create\_resource method. The specific arguments to this method depend on the resource being created; again, see the Azure documentation for more details. Let’s create a new storage account:

stor2 <- rg$create\_resource(type="Microsoft.Storage/storageAccounts",

name="mystorage2",

kind="StorageV2",

sku=list(name="Standard\_LRS"))

And to delete a resource, we can either use the resource object’s delete method, or the resource group object’s delete\_resource method. AzureRMR will prompt you for confirmation that you really want to carry out the deletion.

stor2$delete()

In a manner similar to resources, deploying a [template](https://docs.microsoft.com/azure/azure-resource-manager/resource-group-overview?WT.mc_id=Revolutions-blog-davidsmi#template-deployment) is just a matter of calling the resource group object’s deploy\_template method. This takes two arguments, template and parameters. Both arguments should be in JSON format: either the name of a JSON file, a character vector containing the JSON data, or a list containing the parsed JSON (via jsonlite::toJSON).

vm\_tpl <- rg$deploy\_template("myNewVirtualMachine",

template="vm\_template.json",

parameters=list(

os="Windows",

size="Standard\_DS2\_v2",

username="ruser",

password="MyPassword123!" # use a strong password!

))

Normally, deleting a template doesn’t touch the resources it creates: it only deletes the template itself. However, AzureRMR optionally allows you to free any resources created when you delete a template. This is useful when managing complex objects like VMs, which actually consist of multiple individual resources in Azure (storage account, disk, network interface, etc). When you are done with the VM, deleting the template lets you free all these resources with a single command.

vm\_tpl$delete(free\_resources=TRUE)

**Extending**

AzureRMR is meant to be a generic mechanism for working with Resource Manager. You can extend it to provide support for service-specific features; For example, instead of using a generic do\_operation() call to retrieve the access keys for a storage account, AzureStor provides an az\_storage class that has a list\_keys() method. Similarly, the AzureVM package provides an az\_vm\_template class that wraps the VM deployment template, and has methods for interacting with the VM.

Extending AzureRMR also gives you the opportunity to provide a *client* interface to an Azure service. AzureRMR only acts as an interface to Resource Manager; it doesn’t provide facilities to, for example, work with the data that is stored in a storage account. This is provided by AzureStor, which has a client interface to file and blob storage. With this you can upload and download files, list files and containers, create and delete containers, and so on.