A few weeks ago, I introduced the [AzureR](https://blog.revolutionanalytics.com/2018/11/azurer-intro.html) family of packages for working with Azure in R. Since then, I’ve also written articles on how to use [AzureRMR](https://blog.revolutionanalytics.com/2018/11/azurermr-azure-resource-manager.html) to interact with Azure Resource Manager, how to use [AzureVM](https://blog.revolutionanalytics.com/2018/12/azurevm.html) to manage virtual machines, and how to use [AzureContainers](https://blog.revolutionanalytics.com/2018/12/azurecontainers.html) to deploy R functions with Azure Kubernetes Service. This article is the next in the series, and covers AzureStor: an interface to [Azure storage](https://docs.microsoft.com/azure/storage/?WT.mc_id=Revolutions-blog-davidsmi).

**The Resource Manager interface: creating and deleting storage accounts**

[AzureStor](https://github.com/cloudyr/AzureStor) implements an interface to Azure Resource Manager, which you can use manage storage accounts: creating them, retrieving them, deleting them, and so forth. This is done via the appropriate methods of the az\_resource\_group class. For example, the following code shows how you might create a new storage account from scratch.

library(AzureStor)

# get the resource group for the storage account

rg <- AzureRMR::az\_rm$

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

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

get\_resource\_group("myresourcegroup")

# create the storage account

# by default, this will be in the resource group's region

rg$create\_storage\_account("mynewstorage")

Without any options, this will create a storage account with the following parameters:

* General purpose account (all storage types supported)
* Locally redundant storage (LRS) replication
* Hot access tier (for blob storage)
* HTTPS connection required for access

You can change these by setting the arguments to create\_storage\_account(). For example, to create an account with geo-redundant storage replication and the default blob access tier set to “cool”:

rg$create\_storage\_account("myotherstorage",

replication="Standard\_GRS",

access\_tier="cool")

To retrieve an existing storage account, use the get\_storage\_account() method. Only the storage account name is required.

# retrieve one of the accounts created above

stor2 <- rg$get\_storage\_account("myotherstorage")

Finally, to delete a storage account, you simply call its delete() method. Alternatively, you can call the delete\_storage\_account() method of the az\_resource\_group class, which will do the same thing. In both cases, AzureStor will prompt you for confirmation that you really want to delete the storage account.

rg$delete\_storage\_account("mynewstorage")

stor2$delete() # if you have the storage account object

**The client interface: working with storage**

**Storage endpoints**

Perhaps the more relevant part of AzureStor for most users is its client interface to storage. With this, you can upload and download files and blobs, create containers and shares, list files, and so on. Unlike the ARM interface, the client interface uses S3 classes. This is for a couple of reasons: it is more familiar to most R users, and it is consistent with most other data manipulation packages in R, in particular the [tidyverse](https://tidyverse.org/).

The starting point for client access is the storage\_endpoint object, which stores information about the endpoint of a storage account: the URL that you use to access storage, along with any authentication information needed. The easiest way to obtain an endpoint object is via the storage account resource object’s get\_blob\_endpoint() and get\_file\_endpoint() methods:

# get the storage account object

stor <- AzureRMR::az\_rm$

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

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

get\_resource\_group("myresourcegroup")$

get\_storage\_account("mynewstorage")

stor$get\_blob\_endpoint()

# Azure blob storage endpoint

# URL: <https://mynewstorage.blob.core.windows.net/>

# Access key:

# Account shared access signature:

# Storage API version: 2018-03-28

stor$get\_file\_endpoint()

# Azure file storage endpoint

# URL: <https://mynewstorage.file.core.windows.net/>

# Access key:

# Account shared access signature:

# Storage API version: 2018-03-28

This shows that the base URL to access blob storage is <https://mynewstorage.blob.core.windows.net/>, while that for file storage is <https://mynewstorage.file.core.windows.net/>. While it’s not displayed, the endpoint objects also include the access key necessary for authenticated access to storage; this is obtained directly from the storage account resource.

More practically, you will usually want to work with a storage endpoint without having to go through the process of authenticating with Azure Resource Manager (ARM). Often, you may not have any ARM credentials to start with. In this case, you can create the endpoint object directly with blob\_endpoint() and file\_endpoint():

# same as above

blob\_endp <- blob\_endpoint(

"<https://mynewstorage.blob.core.windows.net/>",

key="mystorageaccesskey")

file\_endp <- file\_endpoint(

"<https://mynewstorage.file.core.windows.net/>",

key="mystorageaccesskey")

Notice that when creating the endpoint this way, you have to provide the access key explicitly.

Instead of an access key, you can provide a [shared access signature (SAS)](https://docs.microsoft.com/en-us/azure/storage/common/storage-dotnet-shared-access-signature-part-1) to gain authenticated access. The main difference between using a key and a SAS is that the former unlocks access to the *entire* storage account. A user who has a key can access all containers and files, and can read, modify and delete data without restriction. On the other hand, a user with a SAS can be limited to have access only to specific files, or be limited to read access, or only for a given span of time, and so on. This is usually much better in terms of security.

Usually, the SAS will be given to you by your system administrator. However, if you have the storage acccount resource object, you can generate and use a SAS as follows. Note that generating a SAS requires the storage account’s access key.

# shared access signature: read/write access, container+object access, valid for 12 hours

now <- Sys.time()

sas <- stor$get\_account\_sas(permissions="rw",

resource\_types="co",

start=now,

end=now + 12 \* 60 \* 60,

key=stor$list\_keys()[1])

# create an endpoint object with a SAS, but without an access key

blob\_endp <- stor$get\_blob\_endpoint(sas=sas)

If you don’t have a key or a SAS, you will only have access to unauthenticated (public) containers and file shares.

**Container and object access: blob containers, file shares, blobs, files**

Given an endpoint object, AzureStor provides the following methods for working with containers:

* blob\_container: get an existing blob container
* create\_blob\_container: create a new blob container
* delete\_blob\_container: delete a blob container
* list\_blob\_containers: return a list of blob container objects
* file\_share: get an existing file share
* create\_file\_share: create a new file share
* delete\_file\_share: delete a file share
* list\_file\_shares: return a list of file share objects

Here is some example blob container code showing their use. The file share code is similar.

# an existing container

cont <- blob\_container(blob\_endp, "mycontainer")

# create a new container and allow

# unauthenticated (public) access to blobs

newcont <- create\_blob\_container(blob\_endp, "mynewcontainer",

public\_access="blob")

# delete the container

delete\_blob\_container(newcont)

# piping also works

library(magrittr)

blob\_endp %>%

blob\_container("mycontainer")

As a convenience, instead of providing an endpoint object and a container name, you can also provide the full URL to the container. If you do this, you’ll also have to supply any necessary authentication details such as the access key or SAS.

cont <- blob\_container(

"<https://mynewstorage.blob.core.windows.net/mycontainer>",

key="mystorageaccountkey")

share <- file\_share(

"<https://mynewstorage.file.core.windows.net/myshare>",

key="mystorageaccountkey")

Given a blob container or file share object, use the list\_blobs() and list\_azure\_files() functions to list the storage objects they contain. The “azure” in list\_azure\_files is to avoid any confusion with R’s regular list.files function.

# list blobs inside a blob container

list\_blobs(cont)

# Name Last-Modified Content-Length

# 1 fs.txt 2018-10-13 11:34:30 132

# 2 fs2.txt 2018-10-13 11:04:36 731930

# if you want only the filenames

list\_blobs(cont, info="name")

# [1] "fs.txt" "fs2.txt"

# and for files inside a file share

list\_azure\_files(share, "/")

# name type size

# 1 100k.txt File 100000

# 2 fs.txt File 132

To transfer files and blobs, use the following functions:

* upload\_blob/download\_blob: transfer a file to or from a blob container.
* upload\_azure\_file/download\_azure\_file: transfer a file to or from a file share.
* upload\_to\_url: upload a file to a destination given by a URL. This dispatches to either upload\_blob or upload\_azure\_file as appropriate.
* download\_from\_url: download a file from a source given by a URL, the opposite of upload\_from\_url. This is analogous to base R’s download.file but with authentication built in.

# upload a file to a blob container

blob\_endp <- blob\_endpoint(

"<https://mynewstorage.blob.core.windows.net/>",

key="mystorageaccesskey")

cont <- blob\_container(blob\_endp, "mycontainer")

upload\_blob(cont, src="myfile", dest="myblob")

# again, piping works

blob\_endpoint(

"<https://mynewstorage.blob.core.windows.net/>",

key="mystorageaccesskey") %>%

blob\_container("mycontainer") %>%

upload\_blob("myfile", "myblob")

# download a blob, overwriting any existing destination file

download\_blob(cont, "myblob", "myfile", overwrite=TRUE)

# as a convenience, you can download directly from an Azure URL

download\_from\_url(

"<https://mynewstorage.blob.core.windows.net/mycontainer/myblob>",

"myfile",

key="mystorageaccesskey",

overwrite=TRUE)

File shares have the additional feature of supporting directories. To create and delete directories, use create\_azure\_dir() and delete\_azure\_dir():

list\_azure\_files(share, "/")

# name type size

# 1 100k.txt File 100000

# 2 fs.txt File 132

# create a directory under the root of the file share

create\_azure\_dir(share, "newdir")

# confirm that the directory has been created

list\_azure\_files(share, "/")

# name type size

# 1 100k.txt File 100000

# 2 fs.txt File 132

# 3 newdir Directory NA

# delete the directory

delete\_azure\_dir(share, "newdir")

The [AzureStor package](https://github.com/cloudyr/AzureStor) is available now on Github.