Hong Ooi <hongooi@microsoft.com>

Microsoft

AzureR

A family of packages for working with Microsoft Azure: https://github.com/Azure/AzureR

AzureRMR

- Lightweight, extensible R6-based interface to Azure Resource Manager
- Manage subscriptions and resource groups
- Create, update and delete resources and templates
- Work with role-based access control (RBAC)

```
library(AzureRMR)

az <- get_azure_login()
sub <- az$get_subscription("subscription_id")
rg <- sub$get_resource_group("rgname")

# get a resource (storage account)
stor <- rg$get_resource(type="Microsoft.Storage/storageAccounts",
    name="mystorage")

# method chaining works too
stor <- az$
    get_subscription("subscription_id")$
    get_resource_group("rgname")$
    get_resource(type="Microsoft.Storage/storageAccounts", name="mystorage")

# create a new resource group and resource
rg2 <- sub$create_resource_group("newrgname", location="westus")

stor2 <- rg2$create_resource(type="Microsoft.Storage/storageAccounts",
    name="mystorage2",
    kind="Storage",
    sku=list(name="Standard_LRS"))

# tagging
stor2$set_tags(comment="hello world!", createdBy="AzureRMR")

# role-based access control (RBAC)
# this uses the AzureGraph package to retrieve the user info
usr <- AzureGraph::get_graph_login()$
    get_user("username@mytenant.com")
stor2$add_role_assignment(usr, "Storage blob data contributor")
```

AzureAuth

- OAuth authentication for Azure Active Directory
- Supports multiple authentication flows: authorization_code, device_code, resource_owner, client_credentials
- Supports AAD v1.0 and v2.0
- Authenticate with password or private key (certificate)
- Supports managed identities

```
library(AzureAuth)

# if no username/password, use authorization code flow
get_azure_token("https://management.azure.com", tenant="mytenant", app="app_id")

# if password supplied, use client credentials flow
get_azure_token("https://management.azure.com", tenant="mytenant", app="app_id",
                password="mypassword")

# if username and password supplied, use resource owner grant flow
get_azure_token("https://management.azure.com", tenant="mytenant", app="app_id",
                username="user", password="mypassword")

# can supply an explicit method
get_azure_token("https://management.azure.com", tenant="mytenant", app="app_id",
                auth_type="device_code")

# authenticate with a certificate
get_azure_token("https://management.azure.com", tenant="mytenant", app="app_id",
                certificate="mycert.pfx")

# authenticate with a cert in Key Vault (more secure)
mycert <- AzureKeyVault::key_vault("mykeyvault", "mytenant")$
    certificates$
    get("mycert")

get_azure_token("https://management.azure.com", tenant="mytenant", app="app_id",
                certificate=mycert)
```

AzureGraph

- R6-based interface to Microsoft Graph
- Emphasis on registered apps and service principals, to support other packages in family
- Can be extended to work with other services in Graph: SharePoint, OneDrive, Outlook, device management, etc

```
library(AzureGraph)

# authenticate with AAD
gr <- get_graph_login()

# my user information
me <- gr$get_user("me")

# my groups
head(me$list_group_memberships())

# my registered apps
me$list_owned_objects(type="application")

# create an app
# by default, this will have a strong password with duration 1 year
app <- gr$create_app("AzureR_newapp")

# get the associated service principal
app$get_service_principal()

# using it in conjunction with AzureRMR RBAC
AzureRMR::get_azure_login()$
    get_subscription("sub_id")$
    get_resource_group("rgname")$
    add_role_assignment(app, "Contributor")
```

AzureKeyVault

- Resource Manager and client interface to Azure Key Vault
- Secure facility for passwords, cryptographic keys, certificates, storage account logins
  - Encrypt and decrypt with keys
  - Sign and verify certificates
  - Rotate storage account access keys
- On control plane side, manage access policies for vault

```
library(AzureKeyVault)

vault <- key_vault("mykeyvault")

# create a new secret
vault$secrets$create("newsecret", "hidden text")

# create a new RSA key
vault$keys$create("newkey", type="RSA")

# encrypting and decrypting
key <- vault$keys$get("newkey")
plaintext <- "super secret"
ciphertext <- key$encrypt(plaintext)
plaintext == key$decrypt(ciphertext, as_raw=FALSE)
#> [1] TRUE

# create a new self-signed certificate
cert <- vault$certificates$create("newcert",
    subject="CN=mydomain.com",
    x509=cert_x509_properties(dns_names="mydomain.com"))

# import a certificate from a PFX file
vault$certificates$import("importedcert", "mycert.pfx")

# export the certificate as a PEM file
cert$export("mycert.pem")
```

AzureStor

- Resource manager and client interface to Azure blob storage, file storage and Data Lake storage gen2
- List, upload and download files
- Authenticate with access key, SAS or AAD token
- Fast parallel transfers for multiple files
- Download to in-memory connection objects
- Interface to AzCopy v10 commandline tool

```
library(AzureStor)
library(magrittr) # for pipe

blob_endp <- storage_endpoint("https://mystorage.blob.core.windows.net",
    key="access_key")

file_endp <- storage_endpoint("https://mystorage.file.core.windows.net",
    sas="mysas")

token <- AzureAuth::get_azure_token("https://storage.azure.com",
    tenant="mytenant", app="app_id", password="password")
adls_endp <- storage_endpoint("https://mystorage.dfs.core.windows.net",
    token=token)

# create a blob container and upload a file
blob_endp %>%
    create_storage_container("container") %>%
    storage_upload("srcfile", "destfile")

# downloading multiple files in parallel
file_endp %>%
    storage_container("fileshare") %>%
    storage_multidownload("*.csv", "destdir")

# download to a connection object
conn <- adls_endp %>%
    storage_container("adlsfilesystem") %>%
    storage_download("srcfile", NULL)
```

AzureContainers

- Interface to Azure Container Registry, Azure Container Instances and Azure Kubernetes Service
- Push and pull images to and from ACR
- Spin up containers with ACI
- Create AKS clusters and deploy and manage services
- Includes shells to docker, kubectl and helm

```
library(AzureContainers)

rg <- AzureRMR::get_azure_login()$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# create container registry
acr <- rg$create_acr("myacr")

# create Docker image from a predefined Dockerfile
call_docker("build -t newcontainer .")

# get registry endpoint, upload image
reg <- acr$get_docker_registry()
reg$push("newcontainer")

# create Kubernetes cluster with 3 nodes
aks <- rg$create_aks("myakscluster",
    location="australiaeast",
    agent_pools=aks_pools("pool1", 3, "Standard_DS2_v2"))

# give the cluster pull access to the registry
aks_app_id <- aks$properties$servicePrincipalProfile$clientId
acr$add_role_assignment(
    principal=AzureGraph::get_graph_login()$get_app(aks_app_id),
    role="Acrpull")

# get cluster endpoint, deploy from ACR to AKS with yaml definition file
clus <- aks$get_cluster()
clus$create("model1.yaml")
clus$get("service")
```

AzureVM

- Flexible, configurable interface to virtual machines and virtual machine scale sets
- Customise your deployment by virtual network, security rules, IP address, load balancer, and more
- Prebuilt configurations for popular Windows and Linux images

```
library(AzureVM)

rg <- AzureRMR::get_azure_login()$
    get_subscription("sub_id")$
    get_resource_group("rgname")

# default is an Ubuntu 18.04 VM, size Standard_DS3_v2, login via SSH key
vm <- rg$create_vm("myubuntuvm", user_config("myname", "~/.ssh/id_rsa.pub"))

# some things you can do with a VM
vm$run_script("echo hello world! > /tmp/hello.txt")
vm$stop()
vm$resize("Standard_DS2_v2")

# Ubuntu DSVM, GPU-enabled
rg$create_vm("mydsvm", user_config("myname", "~/.ssh/id_rsa.pub"),
    size="Standard_NC6s_v2", config="ubuntu_dsvm")

# Windows Server 2019
sub$create_vm("mywinvm", user_config("myname", password="Use-strong-passwords!"),
    config="windows_2019")

# RHEL scaleset, serving HTTP/HTTPS
sub$create_vm_scaleset("myrhelss", user_config("myname", "~/.ssh/id_rsa.pub"),
    instances=5, config="rhel_8_ss",
    nsg=nsg_config(list(nsg_rule_allow_http, nsg_rule_allow_https)))

# sharing a virtual network between a VM and scaleset
rg$create_vm("headvm", user_config("myname", "~/.ssh/id_rsa.pub"))
vnet <- rg$get_resource(type="Microsoft.Network/virtualNetworks",
    name="headvm-vnet")
rg$create_vm_scaleset("clusterss", user_config("myname", "~/.ssh/id_rsa.pub"),
    instances=5, vnet=vnet, nsg=NULL)
```

AzureKusto

- Resource Manager and client interface to Azure Data Explorer, aka Kusto
- Query data using dplyr and DBI interfaces
  - Built in the manner of dbplyr (delayed execution)
- On control plane side, create and manage database principals

```
library(AzureKusto)
library(dplyr)

Samples <- kusto_database_endpoint(server="https://help.kusto.windows.net",
    database="Samples")

# run a query, return a data frame
run_query(Samples,
    "StormEvents | summarize EventCount = count() by State | order by State asc")

# query parameters supported
run_query(Samples, "MyFunction(1im)", lim=10L)

# command statement (starts with ".")
run_query(Samples, ".show tables | count")

# dplyr interface
StormEvents <- tbl_kusto(Samples, "StormEvents")
qry <- StormEvents %>%
    group_by(State) %>%
    summarize(EventCount=n()) %>%
    arrange(State)

show_query(qry)
## <KQL> database('Samples').['StormEvents']
## | summarize ['EventCount'] = count() by ['State']
## | order by ['State'] asc

collect(qry)
## # A tibble: 67 x 2
##   State       EventCount
##   <chr>            <dbl>
## 1 ALABAMA           1315
## 2 ALASKA             257
## ...
```