Package 'AzureRMR'

September 21, 2023

Title Interface to 'Azure Resource Manager'

Version 2.4.4

Description
A lightweight but powerful R interface to the 'Azure Resource Manager' REST API. The package exposes a comprehensive class framework and related tools for creating, updating and deleting 'Azure' resource groups, resources and templates. While 'AzureRMR' can be used to manage any 'Azure' service, it can also be extended by other packages to provide extra functionality for specific services. Part of the 'AzureR' family of packages.

URL `https://github.com/Azure/AzureRMR` `https://github.com/Azure/AzureR`

BugReports `https://github.com/Azure/AzureRMR/issues`

License MIT + file LICENSE

VignetteBuilder knitr

Depends R (>= 3.3)

Imports AzureGraph (>= 1.2.0), AzureAuth (>= 1.2.1), utils, parallel,
httr (>= 1.3), jsonlite, R6, uuid

Suggests knitr, rmarkdown, testthat, httpuv, AzureStor

RoxygenNote 7.2.1

NeedsCompilation no

Author Hong Ooi [aut, cre],
Microsoft [cph]

Maintainer Hong Ooi `<hongooi73@gmail.com>`

Repository CRAN

Date/Publication 2023-09-21 08:50:02 UTC

R topics documented:

`az_resource` *Azure resource class*

Description

Class representing a generic Azure resource.

Format

An R6 object of class `az_resource`.

Methods

- `new(...)`: Initialize a new resource object. See 'Initialization' for more details.
- `delete(confirm=TRUE, wait=FALSE)`: Delete this resource, after a confirmation check. Optionally wait for the delete to finish.
- `update(...)`: Update this resource on the host.
- `sync_fields()`: Synchronise the R object with the resource it represents in Azure. Returns the `properties$provisioningState` field, so you can query this programmatically to check if a resource has finished provisioning. Not all resource types require explicit provisioning, in which case this method will return NULL.
- `set_api_version(api_version, stable_only=TRUE)`: Set the API version to use when interacting with the host. If `api_version` is not supplied, use the latest version available, either the latest stable version (if `stable_only=TRUE`) or the latest preview version (if `stable_only=FALSE`).
- `get_api_version()`: Get the current API version.
- `get_subresource(type, name)`: Get a sub-resource of this resource. See 'Sub-resources' below.
- `create_subresource(type, name, ...)`: Create a sub-resource of this resource.
- `delete_subresource(type, name, confirm=TRUE)`: Delete a sub-resource of this resource.
- `do_operation(...)`: Carry out an operation. See 'Operations' for more details.

- `set_tags(..., keep_existing=TRUE)`: Set the tags on this resource. The tags can be either names or name-value pairs. To delete a tag, set it to `NULL`.
- `get_tags()`: Get the tags on this resource.
- `create_lock(name, level)`: Create a management lock on this resource.
- `get_lock(name)`: Returns a management lock object.
- `delete_lock(name)`: Deletes a management lock object.
- `list_locks()`: List all locks that apply to this resource. Note this includes locks created at the subscription or resource group level.
- `add_role_assignment(name, ...)`: Adds a new role assignment. See 'Role-based access control' below.
- `get_role_assignment(id)`: Retrieves an existing role assignment.
- `remove_role_assignment(id)`: Removes an existing role assignment.
- `list_role_assignments()`: Lists role assignments.
- `get_role_definition(id)`: Retrieves an existing role definition.
- `list_role_definitions()` Lists role definitions.

Initialization

There are multiple ways to initialize a new resource object. The `new()` method can retrieve an existing resource, deploy/create a new resource, or create an empty/null object (without communicating with the host), based on the arguments you supply.

All of these initialization options have the following arguments in common.

1. `token`: An OAuth 2.0 token, as generated by get_azure_token.
2. `subscription`: The subscription ID.
3. `api_version`: Optionally, the API version to use when interacting with the host. By default, this is NULL in which case the latest API version will be used.
4. A set of *identifying arguments*:
   - `resource_group`: The resource group containing the resource.
   - `id`: The full ID of the resource. This is a string of the form `/subscriptions/{uuid}/resourceGroups/{resourc`
   - `provider`: The provider of the resource, eg `Microsoft.Compute`.
   - `path`: The path to the resource, eg `virtualMachines`.
   - `type`: The combination of provider and path, eg `Microsoft.Compute/virtualMachines`.
   - `name`: The name of the resource instance, eg `myWindowsVM`.

Providing `id` will fill in the values for all the other identifying arguments. Similarly, providing `type` will fill in the values for `provider` and `path`. Unless you provide `id`, you must also provide `name`.

The default behaviour for `new()` is to retrieve an existing resource, which occurs if you supply only the arguments listed above. If you also supply an argument `deployed_properties=NULL`, this will create a null object. If you supply any other (named) arguments, `new()` will create a new object on the host, with the supplied arguments as parameters.

Generally, the easiest way to initialize an object is via the `get_resource`, `create_resource` or `list_resources` methods of the az_resource_group class, which will handle all the gory details automatically.

Operations

The `do_operation()` method allows you to carry out arbitrary operations on the resource. It takes the following arguments:

- `op`: The operation in question, which will be appended to the URL path of the request.
- `options`: A named list giving the URL query parameters.
- `...`: Other named arguments passed to call_azure_rm, and then to the appropriate call in httr. In particular, use body to supply the body of a PUT, POST or PATCH request.
- `http_verb`: The HTTP verb as a string, one of `GET`, `PUT`, `POST`, `DELETE`, `HEAD` or `PATCH`.

Consult the Azure documentation for your resource to find out what operations are supported.

Sub-resources

Some resource types can have sub-resources: objects exposed by Resource Manager that make up a part of their parent's functionality. For example, a storage account (type `Microsoft.Storage/storageAccounts`) provides the blob storage service, which can be accessed via Resource Manager as a sub-resource of type `Microsoft.Storage/storageAccounts/blobServices/default`.

To retrieve an existing sub-resource, use the `get_subresource()` method. You do not need to include the parent resource's type and name. For example, if `res` is a resource for a storage account, and you want to retrieve the sub-resource for the blob container "myblobs", call

```
res$get_subresource(type="blobServices/default/containers", name="myblobs")
```

Notice that the storage account's resource type and name are omitted from the `get_subresource` arguments. Similarly, to create a new subresource, call the `create_subresource()` method with the same naming convention, passing any required fields as named arguments; and to delete it, call `delete_subresource()`.

Role-based access control

AzureRMR implements a subset of the full RBAC functionality within Azure Active Directory. You can retrieve role definitions and add and remove role assignments, at the subscription, resource group and resource levels. See rbac for more information.

See Also

az_resource_group, call_azure_rm, call_azure_url, Resources API reference

For role-based access control methods, see rbac

For management locks, see lock

Examples

```
## Not run:

# recommended way to retrieve a resource: via a resource group object
# storage account:
stor <- resgroup$get_resource(type="Microsoft.Storage/storageAccounts", name="mystorage")
```

```
# virtual machine:
vm <- resgroup$get_resource(type="Microsoft.Compute/virtualMachines", name="myvm")

## carry out operations on a resource

# storage account: get access keys
stor$do_operation("listKeys", http_verb="POST")

# virtual machine: run a script
vm$do_operation("runCommand",
    body=list(
        commandId="RunShellScript", # RunPowerShellScript for Windows
        script=as.list("ifconfig > /tmp/ifconfig.out")
    ),
    encode="json",
    http_verb="POST")

## retrieve properties

# storage account: endpoint URIs
stor$properties$primaryEndpoints$file
stor$properties$primaryEndpoints$blob

# virtual machine: hardware profile
vm$properties$hardwareProfile

## update a resource: resizing a VM
properties <- list(hardwareProfile=list(vmSize="Standard_DS3_v2"))
vm$do_operation(http_verb="PATCH",
    body=list(properties=properties),
    encode="json")

# sync with Azure: useful to track resource creation/update status
vm$sync_fields()

## subresource: create a public blob container
stor$create_subresource(type="blobservices/default/containers", name="mycontainer",
    properties=list(publicAccess="container"))

## delete a subresource and resource
stor$delete_subresource(type="blobservices/default/containers", name="mycontainer")
stor$delete()


## End(Not run)
```

`az_resource_group` *Azure resource group class*

Description

Class representing an Azure resource group.

Format

An R6 object of class `az_resource_group`.

Methods

- `new(token, subscription, id, ...)`: Initialize a resource group object. See 'Initialization' for more details.
- `delete(confirm=TRUE)`: Delete this resource group, after a confirmation check. This is asynchronous: while the method returns immediately, the delete operation continues on the host in the background. For resource groups containing a large number of deployed resources, this may take some time to complete.
- `sync_fields()`: Synchronise the R object with the resource group it represents in Azure.
- `list_templates(filter, top)`: List deployed templates in this resource group. `filter` and `top` are optional arguments to filter the results; see the Azure documentation for more details. If `top` is specified, the returned list will have a maximum of this many items.
- `get_template(name)`: Return an object representing an existing template.
- `deploy_template(...)`: Deploy a new template. See 'Templates' for more details. By default, AzureRMR will set the `createdBy` tag on a newly-deployed template to the value `AzureR/AzureRMR`.
- `delete_template(name, confirm=TRUE, free_resources=FALSE)`: Delete a deployed template, and optionally free any resources that were created.
- `get_resource(...)`: Return an object representing an existing resource. See 'Resources' for more details.
- `create_resource(...)`: Create a new resource. By default, AzureRMR will set the `createdBy` tag on a newly-created resource to the value `AzureR/AzureRMR`.
- `delete_resource(..., confirm=TRUE, wait=FALSE)`: Delete an existing resource. Optionally wait for the delete to finish.
- `resource_exists(...)`: Check if a resource exists.
- `list_resources(filter, expand, top)`: Return a list of resource group objects for this subscription. `filter`, `expand` and `top` are optional arguments to filter the results; see the Azure documentation for more details. If `top` is specified, the returned list will have a maximum of this many items.
- `do_operation(...)`: Carry out an operation. See 'Operations' for more details.
- `set_tags(..., keep_existing=TRUE)`: Set the tags on this resource group. The tags can be either names or name-value pairs. To delete a tag, set it to `NULL`.
- `get_tags()`: Get the tags on this resource group.
- `create_lock(name, level)`: Create a management lock on this resource group (which will propagate to all resources within it).
- `get_lock(name)`: Returns a management lock object.
- `delete_lock(name)`: Deletes a management lock object.
- `list_locks()`: List all locks that apply to this resource group. Note this includes locks created at the subscription level, and for any resources within the resource group.

- `add_role_assignment(name, ...)`: Adds a new role assignment. See 'Role-based access control' below.
- `get_role_assignment(id)`: Retrieves an existing role assignment.
- `remove_role_assignment(id)`: Removes an existing role assignment.
- `list_role_assignments()`: Lists role assignments.
- `get_role_definition(id)`: Retrieves an existing role definition.
- `list_role_definitions()` Lists role definitions.

Initialization

Initializing a new object of this class can either retrieve an existing resource group, or create a new resource group on the host. Generally, the easiest way to create a resource group object is via the `get_resource_group`, `create_resource_group` or `list_resource_groups` methods of the az_subscription class, which handle this automatically.

To create a resource group object in isolation, supply (at least) an Oauth 2.0 token of class AzureToken, the subscription ID, and the resource group name. If this object refers to a *new* resource group, supply the location as well (use the `list_locations` method of the `az_subscription class` for possible locations). You can also pass any optional parameters for the resource group as named arguments to `new()`.

Templates

To deploy a new template, pass the following arguments to `deploy_template()`:

- `name`: The name of the deployment.
- `template`: The template to deploy. This can be provided in a number of ways:
  1. A nested list of name-value pairs representing the parsed JSON
  2. The name of a template file
  3. A vector of strings containing unparsed JSON
  4. A URL from which the template can be downloaded
- `parameters`: The parameters for the template. This can be provided using any of the same methods as the `template` argument.
- `wait`: Optionally, whether or not to wait until the deployment is complete before returning. Defaults to `FALSE`.

Retrieving or deleting a deployed template requires only the name of the deployment.

Resources

There are a number of arguments to `get_resource()`, `create_resource()` and `delete_resource()` that serve to identify the specific resource in question:

- `id`: The full ID of the resource, including subscription ID and resource group.
- `provider`: The provider of the resource, eg `Microsoft.Compute`.
- `path`: The full path to the resource, eg `virtualMachines`.
- `type`: The combination of provider and path, eg `Microsoft.Compute/virtualMachines`.

- `name`: The name of the resource instance, eg `myWindowsVM`.

Providing the `id` argument will fill in the values for all the other arguments. Similarly, providing the `type` argument will fill in the values for `provider` and `path`. Unless you provide `id`, you must also provide `name`.

To create/deploy a new resource, specify any extra parameters that the provider needs as named arguments to `create_resource()`. Like `deploy_template()`, `create_resource()` also takes an optional `wait` argument that specifies whether to wait until resource creation is complete before returning.

Operations

The `do_operation()` method allows you to carry out arbitrary operations on the resource group. It takes the following arguments:

- `op`: The operation in question, which will be appended to the URL path of the request.
- `options`: A named list giving the URL query parameters.
- `...`: Other named arguments passed to call_azure_rm, and then to the appropriate call in httr. In particular, use `body` to supply the body of a PUT, POST or PATCH request, and `api_version` to set the API version.
- `http_verb`: The HTTP verb as a string, one of `GET`, `PUT`, `POST`, `DELETE`, `HEAD` or `PATCH`.

Consult the Azure documentation for what operations are supported.

Role-based access control

AzureRMR implements a subset of the full RBAC functionality within Azure Active Directory. You can retrieve role definitions and add and remove role assignments, at the subscription, resource group and resource levels. See rbac for more information.

See Also

az_subscription, az_template, az_resource, Azure resource group overview, Resources API reference, Template API reference

For role-based access control methods, see rbac

For management locks, see lock

Examples

```
## Not run:

# recommended way to retrieve a resource group object
rg <- get_azure_login("myaadtenant")$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# list resources & templates in this resource group
rg$list_resources()
rg$list_templates()
```

```
# get a resource (virtual machine)
rg$get_resource(type="Microsoft.Compute/virtualMachines", name="myvm")

# create a resource (storage account)
rg$create_resource(type="Microsoft.Storage/storageAccounts", name="mystorage",
    kind="StorageV2",
    sku=list(name="Standard_LRS"))

# delete a resource
rg$delete_resource(type="Microsoft.Storage/storageAccounts", name="mystorage")

# deploy a template
rg$deploy_template("tplname",
    template="template.json",
    parameters="parameters.json")

# deploy a template with parameters inline
rg$deploy_template("mydeployment",
    template="template.json",
    parameters=list(parm1="foo", parm2="bar"))

# delete a template and free resources
rg$delete_template("tplname", free_resources=TRUE)

# delete the resource group itself
rg$delete()


## End(Not run)
```

az_rm *Azure Resource Manager*

Description

Base class for interacting with Azure Resource Manager.

Format

An R6 object of class `az_rm`.

Methods

- `new(tenant, app, ...)`: Initialize a new ARM connection with the given credentials. See 'Authentication' for more details.
- `list_subscriptions()`: Returns a list of objects, one for each subscription associated with this app ID.
- `get_subscription(id)`: Returns an object representing a subscription.

- `get_subscription_by_name(name)`: Returns the subscription with the given name (as opposed to a GUID).
- `do_operation(...)`: Carry out an operation. See 'Operations' for more details.

Authentication

The recommended way to authenticate with ARM is via the get_azure_login function, which creates a new instance of this class.

To authenticate with the `az_rm` class directly, provide the following arguments to the `new` method:

- `tenant`: Your tenant ID. This can be a name ("myaadtenant"), a fully qualified domain name ("myaadtenant.onmicrosoft.com" or "mycompanyname.com"), or a GUID.
- `app`: The client/app ID to use to authenticate with Azure Active Directory. The default is to login interactively using the Azure CLI cross-platform app, but it's recommended to supply your own app credentials if possible.
- `password`: if `auth_type == "client_credentials"`, the app secret; if `auth_type == "resource_owner"`, your account password.
- `username`: if `auth_type == "resource_owner"`, your username.
- `certificate`: If 'auth_type == "client_credentials", a certificate to authenticate with. This is a more secure alternative to using an app secret.
- `auth_type`: The OAuth authentication method to use, one of "client_credentials", "authorization_code", "device_code" or "resource_owner". See get_azure_token for how the default method is chosen, along with some caveats.
- `version`: The Azure Active Directory version to use for authenticating.
- `host`: your ARM host. Defaults to `https://management.azure.com/`. Change this if you are using a government or private cloud.
- `aad_host`: Azure Active Directory host for authentication. Defaults to `https://login.microsoftonline.com/`. Change this if you are using a government or private cloud.
- `...`: Further arguments to pass to `get_azure_token`.
- `scopes`: The Azure Service Management scopes (permissions) to obtain for this login. Only for `version=2`.
- `token`: Optionally, an OAuth 2.0 token, of class AzureToken. This allows you to reuse the authentication details for an existing session. If supplied, all other arguments will be ignored.

Operations

The `do_operation()` method allows you to carry out arbitrary operations on the Resource Manager endpoint. It takes the following arguments:

- `op`: The operation in question, which will be appended to the URL path of the request.
- `options`: A named list giving the URL query parameters.
- `...`: Other named arguments passed to call_azure_rm, and then to the appropriate call in httr. In particular, use `body` to supply the body of a PUT, POST or PATCH request, and `api_version` to set the API version.
- `http_verb`: The HTTP verb as a string, one of `GET`, `PUT`, `POST`, `DELETE`, `HEAD` or `PATCH`.

Consult the Azure documentation for what operations are supported.

See Also

create_azure_login, get_azure_login

Azure Resource Manager overview, REST API reference

Examples

```
## Not run:

# start a new Resource Manager session
az <- az_rm$new(tenant="myaadtenant.onmicrosoft.com", app="app_id", password="password")

# authenticate with credentials in a file
az <- az_rm$new(config_file="creds.json")

# authenticate with device code
az <- az_rm$new(tenant="myaadtenant.onmicrosoft.com", app="app_id", auth_type="device_code")

# retrieve a list of subscription objects
az$list_subscriptions()

# a specific subscription
az$get_subscription("subscription_id")


## End(Not run)
```

`az_role_assignment` *Azure role assignment class*

Description

Azure role assignment class

Format

An R6 object of class `az_role_assignment`.

Fields

- `id`: The full resource ID for this role assignment.
- `type`: The resource type for a role assignment. Always `Microsoft.Authorization/roleAssignments`.
- `name`: A GUID that identifies this role assignment.
- `role_name`: The role definition name (in text), eg "Contributor".
- `properties`: Properties for the role definition.
- `token`: An OAuth token, obtained via get_azure_token.

Methods

- `remove(confirm=TRUE)`: Removes this role assignment.

Initialization

The recommended way to create new instances of this class is via the add_role_assignment and get_role_assignment methods for subscription, resource group and resource objects.

Technically role assignments and role definitions are Azure *resources*, and could be implemented as subclasses of `az_resource`. AzureRMR treats them as distinct, due to limited RBAC functionality currently supported.

See Also

add_role_assignment, get_role_assignment, get_role_definition, az_role_definition

Overview of role-based access control

`az_role_definition` *Azure role definition class*

Description

Azure role definition class

Format

An R6 object of class `az_role_definition`.

Fields

- `id`: The full resource ID for this role definition.
- `type`: The resource type for a role definition. Always `Microsoft.Authorization/roleDefinitions`.
- `name`: A GUID that identifies this role definition.
- `properties`: Properties for the role definition.

Methods

This class has no methods.

Initialization

The recommended way to create new instances of this class is via the get_role_definition method for subscription, resource group and resource objects.

Technically role assignments and role definitions are Azure *resources*, and could be implemented as subclasses of `az_resource`. AzureRMR treats them as distinct, due to limited RBAC functionality currently supported. In particular, role definitions are read-only: you can retrieve a definition, but not modify it, nor create new definitions.

See Also

get_role_definition, get_role_assignment, az_role_assignment

Overview of role-based access control

`az_subscription` *Azure subscription class*

Description

Class representing an Azure subscription.

Format

An R6 object of class `az_subscription`.

Methods

- `new(token, id, ...)`: Initialize a subscription object.
- `list_resource_groups(filter, top)`: Return a list of resource group objects for this subscription. `filter` and `top` are optional arguments to filter the results; see the Azure documentation for more details. If `top` is specified, the returned list will have a maximum of this many items.
- `get_resource_group(name)`: Return an object representing an existing resource group.
- `create_resource_group(name, location)`: Create a new resource group in the specified region/location, and return an object representing it. By default, AzureRMR will set the `createdBy` tag on a newly-created resource group to the value `AzureR/AzureRMR`.
- `delete_resource_group(name, confirm=TRUE)`: Delete a resource group, after asking for confirmation.
- `resource_group_exists(name)`: Check if a resource group exists.
- `list_resources(filter, expand, top)`: List all resources deployed under this subscription. `filter`, `expand` and `top` are optional arguments to filter the results; see the Azure documentation for more details. If `top` is specified, the returned list will have a maximum of this many items.
- `list_locations(info=c("partial", "all"))`: List locations available. The default `info="partial"` returns a subset of the information about each location; set `info="all"` to return everything.
- `get_provider_api_version(provider, type, which=1, stable_only=TRUE)`: Get the current API version for the given resource provider and type. If no resource type is supplied, returns a vector of API versions, one for each resource type for the given provider. If neither provider nor type is supplied, returns the API versions for all resources and providers. Set `stable_only=FALSE` to allow preview APIs to be returned. Set `which` to a number > 1 to return an API other than the most recent.
- `do_operation(...)`: Carry out an operation. See 'Operations' for more details.
- `create_lock(name, level)`: Create a management lock on this subscription (which will propagate to all resources within it).

- `get_lock(name)`: Returns a management lock object.
- `delete_lock(name)`: Deletes a management lock object.
- `list_locks()`: List all locks that exist in this subscription.
- `add_role_assignment(name, ...)`: Adds a new role assignment. See 'Role-based access control' below.
- `get_role_assignment(id)`: Retrieves an existing role assignment.
- `remove_role_assignment(id)`: Removes an existing role assignment.
- `list_role_assignments()`: Lists role assignments.
- `get_role_definition(id)`: Retrieves an existing role definition.
- `list_role_definitions()` Lists role definitions.
- `get_tags()` Get the tags on this subscription.

Details

Generally, the easiest way to create a subscription object is via the `get_subscription` or `list_subscriptions` methods of the az_rm class. To create a subscription object in isolation, call the `new()` method and supply an Oauth 2.0 token of class AzureToken, along with the ID of the subscription.

Operations

The `do_operation()` method allows you to carry out arbitrary operations on the subscription. It takes the following arguments:

- `op`: The operation in question, which will be appended to the URL path of the request.
- `options`: A named list giving the URL query parameters.
- `...`: Other named arguments passed to call_azure_rm, and then to the appropriate call in httr. In particular, use `body` to supply the body of a PUT, POST or PATCH request, and `api_version` to set the API version.
- `http_verb`: The HTTP verb as a string, one of `GET`, `PUT`, `POST`, `DELETE`, `HEAD` or `PATCH`.

Consult the Azure documentation for what operations are supported.

Role-based access control

AzureRMR implements a subset of the full RBAC functionality within Azure Active Directory. You can retrieve role definitions and add and remove role assignments, at the subscription, resource group and resource levels. See rbac for more information.

See Also

Azure Resource Manager overview

For role-based access control methods, see rbac

For management locks, see lock

Examples

```
## Not run:

# recommended way to retrieve a subscription object
sub <- get_azure_login("myaadtenant")$
    get_subscription("subscription_id")

# retrieve list of resource group objects under this subscription
sub$list_resource_groups()

# get a resource group
sub$get_resource_group("rgname")

# check if a resource group exists, and if not, create it
rg_exists <- sub$resource_group_exists("rgname")
if(!rg_exists)
    sub$create_resource_group("rgname", location="australiaeast")

# delete a resource group
sub$delete_resource_group("rgname")

# get provider API versions for some resource types
sub$get_provider_api_version("Microsoft.Compute", "virtualMachines")
sub$get_provider_api_version("Microsoft.Storage", "storageAccounts")


## End(Not run)
```

`az_template` *Azure template class*

Description

Class representing an Azure deployment template.

Format

An R6 object of class `az_template`.

Methods

- `new(token, subscription, resource_group, name, ...)`: Initialize a new template object. See 'Initialization' for more details.
- `check()`: Check the deployment status of the template; throw an error if the template has been deleted.
- `cancel(free_resources=FALSE)`: Cancel an in-progress deployment. Optionally free any resources that have already been created.

- `delete(confirm=TRUE, free_resources=FALSE)`: Delete a deployed template, after a confirmation check. Optionally free any resources that were created. If the template was deployed in Complete mode (its resource group is exclusive to its use), the latter process will delete the entire resource group. Otherwise resources are deleted in the order given by the template's output resources list; in this case, some may be left behind if the ordering is incompatible with dependencies.
- `list_resources()`: Returns a list of Azure resource objects that were created by the template. This returns top-level resources only, not those that represent functionality provided by another resource.
- `get_tags()`: Returns the tags for the deployment template (note: this is not the same as the tags applied to resources that are deployed).

Initialization

Initializing a new object of this class can either retrieve an existing template, or deploy a new template on the host. Generally, the easiest way to create a template object is via the `get_template`, `deploy_template` or `list_templates` methods of the az_resource_group class, which handle the details automatically.

To initialize an object that refers to an existing deployment, supply the following arguments to `new()`:

- `token`: An OAuth 2.0 token, as generated by get_azure_token.
- `subscription`: The subscription ID.
- `resource_group`: The resource group.
- `name`: The deployment name'.

If you also supply the following arguments to `new()`, a new template will be deployed:

- `template`: The template to deploy. This can be provided in a number of ways:
  1. A nested list of R objects, which will be converted to JSON via `jsonlite::toJSON`
  2. A vector of strings containing unparsed JSON
  3. The name of a template file
  4. A URL from which the host can download the template
- `parameters`: The parameters for the template. This can be provided using any of the same methods as the `template` argument.
- `wait`: Optionally, whether to wait until the deployment is complete. Defaults to FALSE, in which case the method will return immediately.

You can use the `build_template_definition` and `build_template_parameters` helper functions to construct the inputs for deploying a template. These can take as inputs R lists, JSON text strings, or file connections, and can also be extended by other packages.

See Also

az_resource_group, az_resource, build_template_definition, build_template_parameters Template overview, Template API reference

Examples

```
## Not run:

# recommended way to deploy a template: via a resource group object

tpl <- resgroup$deploy_template("mydeployment",
    template="template.json",
    parameters="parameters.json")

# retrieve list of created resource objects
tpl$list_resources()

# delete template (will not touch resources)
tpl$delete()

# delete template and free resources
tpl$delete(free_resources=TRUE)


## End(Not run)
```

`build_template_definition`
Build the JSON for a template and its parameters

Description

Build the JSON for a template and its parameters

Usage

```
build_template_definition(...)

## Default S3 method:
build_template_definition(parameters = named_list(),
  variables = named_list(), functions = list(), resources = list(),
  outputs = named_list(), schema = "2019-04-01", version = "1.0.0.0",
  api_profile = NULL, ...)

build_template_parameters(...)

## Default S3 method:
build_template_parameters(...)
```

Arguments

`...`	For `build_template_parameters`, named arguments giving the values of each template parameter. For `build_template_definition`, further arguments passed to class methods.

`parameters`	For `build_template_definition`, the parameter names and types for the template. See 'Details' below.
`variables`	Internal variables used by the template.
`functions`	User-defined functions used by the template.
`resources`	List of resources that the template should deploy.
`outputs`	The template outputs.
`schema, version, api_profile`	Less commonly used arguments that can be used to customise the template. See the guide to template syntax on Microsoft Docs, linked below.

Details

`build_template_definition` is used to generate a template from its components. The main arguments are `parameters`, `variables`, `functions`, `resources` and `outputs`. Each of these can be specified in various ways:

- As character strings containing unparsed JSON text.
- As an R list of (nested) objects, which will be converted to JSON via `jsonlite::toJSON`.
- A connection pointing to a JSON file or object.
- For the `parameters` argument, this can also be a character vector containing the types of each parameter.

`build_template_parameters` is for creating the list of parameters to be passed along with the template. Its arguments should all be named, and contain either the JSON text or an R list giving the parsed JSON.

Both of these are generics and can be extended by other packages to handle specific deployment scenarios, eg virtual machines.

Value

The JSON text for the template definition and its parameters.

See Also

az_template, jsonlite::toJSON

Guide to template syntax

Examples

```
# dummy example
# note that 'resources' arg should be a _list_ of resources
build_template_definition(resources=list(list(name="resource here")))

# specifying parameters as a list
build_template_definition(parameters=list(par1=list(type="string")),
                          resources=list(list(name="resource here")))

# specifying parameters as a vector
```

```
build_template_definition(parameters=c(par1="string"),
                          resources=list(list(name="resource here")))

# a user-defined function
build_template_definition(
    parameters=c(name="string"),
    functions=list(
        list(
            namespace="mynamespace",
            members=list(
                prefixedName=list(
                    parameters=list(
                        list(name="name", type="string")
                    ),
                    output=list(
                        type="string",
                        value="[concat('AzureR', parameters('name'))]"
                    )
                )
            )
        )
    )
)

# realistic example: storage account
build_template_definition(
    parameters=c(
        name="string",
        location="string",
        sku="string"
    ),
    variables=list(
        id="[resourceId('Microsoft.Storage/storageAccounts', parameters('name'))]"
    ),
    resources=list(
        list(
            name="[parameters('name')]",
            location="[parameters('location')]",
            type="Microsoft.Storage/storageAccounts",
            apiVersion="2018-07-01",
            sku=list(
                name="[parameters('sku')]"
            ),
            kind="Storage"
        )
    ),
    outputs=list(
        storageId="[variables('id')]"
    )
)

# providing JSON text as input
build_template_definition(
    parameters=c(name="string", location="string", sku="string"),
    resources='[
        {
            "name": "[parameters(\'name\')]",
            "location": "[parameters(\'location\')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2018-07-01",
            "sku": {
                "name": "[parameters(\'sku\')]"
            },
            "kind": "Storage"
        }
    ]'
)

# parameter values
build_template_parameters(name="mystorageacct", location="westus", sku="Standard_LRS")

build_template_parameters(
    param='{
        "name": "myname",
        "properties": { "prop1": 42, "prop2": "hello" }
    }'
)

param_json <- '{
        "name": "myname",
        "properties": { "prop1": 42, "prop2": "hello" }
    }'
build_template_parameters(param=textConnection(param_json))

## Not run:
# reading JSON definitions from files
build_template_definition(
    parameters=file("parameter_def.json"),
    resources=file("resource_def.json")
)

build_template_parameters(name="myres_name", complex_type=file("myres_params.json"))

## End(Not run)
```

`call_azure_rm` *Call the Azure Resource Manager REST API*

Description

Call the Azure Resource Manager REST API

Usage

```
call_azure_rm(token, subscription, operation, ..., options = list(),
  api_version = getOption("azure_api_version"))

call_azure_url(token, url, ..., body = NULL, encode = "json",
  http_verb = c("GET", "DELETE", "PUT", "POST", "HEAD", "PATCH"),
  http_status_handler = c("stop", "warn", "message", "pass"),
  auto_refresh = TRUE)
```

Arguments

`token`	An Azure OAuth token, of class AzureToken.
`subscription`	For `call_azure_rm`, a subscription ID.
`operation`	The operation to perform, which will form part of the URL path.
`...`	Other arguments passed to lower-level code, ultimately to the appropriate functions in httr.
`options`	A named list giving the URL query parameters.
`api_version`	The API version to use, which will form part of the URL sent to the host.
`url`	A complete URL to send to the host.
`body`	The body of the request, for `PUT`/`POST`/`PATCH`.
`encode`	The encoding (really content-type) for the request body. The default value "json" means to serialize a list body into a JSON object. If you pass an already-serialized JSON object as the body, set encode to "raw".
`http_verb`	The HTTP verb as a string, one of `GET`, `PUT`, `POST`, `DELETE`, `HEAD` or `PATCH`.
`http_status_handler`	How to handle in R the HTTP status code of a response. `"stop"`, `"warn"` or `"message"` will call the appropriate handlers in httr, while `"pass"` ignores the status code.
`auto_refresh`	Whether to refresh/renew the OAuth token if it is no longer valid.

Details

These functions form the low-level interface between R and Azure. `call_azure_rm` builds a URL from its arguments and passes it to `call_azure_url`. Authentication is handled automatically.

Value

If `http_status_handler` is one of `"stop"`, `"warn"` or `"message"`, the status code of the response is checked. If an error is not thrown, the parsed content of the response is returned with the status code attached as the "status" attribute.

If `http_status_handler` is `"pass"`, the entire response is returned without modification.

See Also

httr::GET, httr::PUT, httr::POST, httr::DELETE, httr::stop_for_status, httr::content

create_azure_login *Login to Azure Resource Manager*

Description

Login to Azure Resource Manager

Usage

```
create_azure_login(tenant = "common", app = .az_cli_app_id,
  password = NULL, username = NULL, certificate = NULL,
  auth_type = NULL, version = 2, host = "https://management.azure.com/",
  aad_host = "https://login.microsoftonline.com/", scopes = ".default",
  config_file = NULL, token = NULL,
  graph_host = "https://graph.microsoft.com/", ...)

get_azure_login(tenant = "common", selection = NULL, app = NULL,
  scopes = NULL, auth_type = NULL, refresh = TRUE)

delete_azure_login(tenant = "common", confirm = TRUE)

list_azure_logins()
```

Arguments

`tenant`	The Azure Active Directory tenant for which to obtain a login client. Can be a name ("myaadtenant"), a fully qualified domain name ("myaadtenant.onmicrosoft.com" or "mycompanyname.com"), or a GUID. The default is to login via the "common" tenant, which will infer your actual tenant from your credentials.
`app`	The client/app ID to use to authenticate with Azure Active Directory. The default is to login interactively using the Azure CLI cross-platform app, but you can supply your own app credentials as well.
`password`	If `auth_type == "client_credentials"`, the app secret; if `auth_type == "resource_owner"`, your account password.
`username`	If `auth_type == "resource_owner"`, your username.
`certificate`	If ‘auth_type == "client_credentials", a certificate to authenticate with. This is a more secure alternative to using an app secret.
`auth_type`	The OAuth authentication method to use, one of "client_credentials", "authorization_code", "device_code" or "resource_owner". If `NULL`, this is chosen based on the presence of the `username` and `password` arguments.
`version`	The Azure Active Directory version to use for authenticating.
`host`	Your ARM host. Defaults to `https://management.azure.com/`. Change this if you are using a government or private cloud.
`aad_host`	Azure Active Directory host for authentication. Defaults to `https://login.microsoftonline.com/`. Change this if you are using a government or private cloud.

`scopes`	The Azure Service Management scopes (permissions) to obtain for this login. Only for `version=2`.
`config_file`	Optionally, a JSON file containing any of the arguments listed above. Arguments supplied in this file take priority over those supplied on the command line. You can also use the output from the Azure CLI `az ad sp create-for-rbac` command.
`token`	Optionally, an OAuth 2.0 token, of class AzureToken. This allows you to reuse the authentication details for an existing session. If supplied, the other arguments above to `create_azure_login` will be ignored.
`graph_host`	The Microsoft Graph endpoint. See 'Microsoft Graph integration' below.
`...`	For `create_azure_login`, other arguments passed to `get_azure_token`.
`selection`	For `get_azure_login`, if you have multiple logins for a given tenant, which one to use. This can be a number, or the input MD5 hash of the token used for the login. If not supplied, `get_azure_login` will print a menu and ask you to choose a login.
`refresh`	For `get_azure_login`, whether to refresh the authentication token on loading the client.
`confirm`	For `delete_azure_login`, whether to ask for confirmation before deleting.

Details

`create_azure_login` creates a login client to authenticate with Azure Resource Manager (ARM), using the supplied arguments. The Azure Active Directory (AAD) authentication token is obtained using get_azure_token, which automatically caches and reuses tokens for subsequent sessions. Note that credentials are only cached if you allowed AzureRMR to create a data directory at package startup.

`create_azure_login()` without any arguments is roughly equivalent to the Azure CLI command `az login`.

`get_azure_login` returns a login client by retrieving previously saved credentials. It searches for saved credentials according to the supplied tenant; if multiple logins are found, it will prompt for you to choose one.

One difference between `create_azure_login` and `get_azure_login` is the former will delete any previously saved credentials that match the arguments it was given. You can use this to force AzureRMR to remove obsolete tokens that may be lying around.

Value

For `get_azure_login` and `create_azure_login`, an object of class `az_rm`, representing the ARM login client. For `list_azure_logins`, a (possibly nested) list of such objects.

If the AzureRMR data directory for saving credentials does not exist, `get_azure_login` will throw an error.

Microsoft Graph integration

If the AzureGraph package is installed and the `graph_host` argument is not `NULL`, `create_azure_login` will also create a login client for Microsoft Graph with the same credentials. This is to facilitate working with registered apps and service principals, eg when managing roles and permissions.

Some Azure services also require creating service principals as part of creating a resource (eg Azure Kubernetes Service), and keeping the Graph credentials consistent with ARM helps ensure nothing breaks.

Linux DSVM note

If you are using a Linux Data Science Virtual Machine in Azure, you may have problems running `create_azure_login()` (ie, without any arguments). In this case, try `create_azure_login(auth_type="device_code")`.

See Also

az_rm, AzureAuth::get_azure_token for more details on authentication methods, AzureGraph::create_graph_login for the corresponding function to create a Microsoft Graph login client

Azure Resource Manager overview, REST API reference

Authentication in Azure Active Directory

Azure CLI documentation

Examples

```
## Not run:

# without any arguments, this will create a client using your AAD credentials
az <- create_azure_login()

# retrieve the login in subsequent sessions
az <- get_azure_login()

# this will create a Resource Manager client for the AAD tenant 'myaadtenant.onmicrosoft.com',
# using the client_credentials method
az <- create_azure_login("myaadtenant", app="app_id", password="password")

# you can also login using credentials in a json file
az <- create_azure_login(config_file="~/creds.json")


## End(Not run)
```

`init_pool` *Manage parallel Azure connections*

Description

Manage parallel Azure connections

Usage

```
init_pool(size = 10, restart = FALSE, ...)

delete_pool()

pool_exists()

pool_size()

pool_export(...)

pool_lapply(...)

pool_sapply(...)

pool_map(...)

pool_call(...)

pool_evalq(...)
```

Arguments

`size`	For `init_pool`, the number of background R processes to create. Limit this is you are low on memory.
`restart`	For `init_pool`, whether to terminate an already running pool first.
`...`	Other arguments passed on to functions in the parallel package. See below.

Details

AzureRMR provides the ability to parallelise communicating with Azure by utilizing a pool of R processes in the background. This often leads to major speedups in scenarios like downloading large numbers of small files, or working with a cluster of virtual machines. This functionality is intended for use by packages that extend AzureRMR (and was originally implemented as part of the AzureStor package), but can also be called directly by the end-user.

A small API consisting of the following functions is currently provided for managing the pool. They pass their arguments down to the corresponding functions in the parallel package.

- `init_pool` initialises the pool, creating it if necessary. The pool is created by calling `parallel::makeCluster` with the pool size and any additional arguments. If `init_pool` is called and the current pool is smaller than `size`, it is resized.
- `delete_pool` shuts down the background processes and deletes the pool.
- `pool_exists` checks for the existence of the pool, returning a TRUE/FALSE value.
- `pool_size` returns the size of the pool, or zero if the pool does not exist.
- `pool_export` exports variables to the pool nodes. It calls `parallel::clusterExport` with the given arguments.

- `pool_lapply`, `pool_sapply` and `pool_map` carry out work on the pool. They call `parallel::parLapply`, `parallel::parSapply` and `parallel::clusterMap` with the given arguments.
- `pool_call` and `pool_evalq` execute code on the pool nodes. They call `parallel::clusterCall` and `parallel::clusterEvalQ` with the given arguments.

The pool is persistent for the session or until terminated by `delete_pool`. You should initialise the pool by calling `init_pool` before running any code on it. This restores the original state of the pool nodes by removing any objects that may be in memory, and resetting the working directory to the master working directory.

See Also

parallel::makeCluster, parallel::clusterCall, parallel::parLapply

Examples

```
## Not run:

init_pool()

pool_size()

x <- 42
pool_export("x")
pool_sapply(1:5, function(i) i + x)

init_pool()
# error: x no longer exists on nodes
try(pool_sapply(1:5, function(i) i + x))

delete_pool()


## End(Not run)
```

`is_azure_login` *Informational functions*

Description

These functions return whether the object is of the corresponding AzureRMR class.

Usage

```
is_azure_login(object)

is_subscription(object)

is_resource_group(object)
```

```
is_resource(object)

is_template(object)

is_role_definition(object)

is_role_assignment(object)
```

Arguments

`object`	An R object.

Value

A boolean.

`is_url` *Miscellaneous utility functions*

Description

Miscellaneous utility functions

Usage

```
is_url(x, https_only = FALSE)

get_paged_list(lst, token, next_link_name = "nextLink",
  value_name = "value")
```

Arguments

`x`	For `is_url`, An R object.
`https_only`	For `is_url`, whether to allow only HTTPS URLs.
`lst`	A named list of objects.
`token`	For `get_paged_list`, an Azure OAuth token, of class AzureToken.
`next_link_name, value_name`	For `get_paged_list`, the names of the next link and value components in the `lst` argument. The default values are correct for Resource Manager.

Details

`get_paged_list` reconstructs a complete list of objects from a paged response. Many Resource Manager list operations will return *paged* output, that is, the response contains a subset of all items, along with a URL to query to retrieve the next subset. `get_paged_list` retrieves each subset and returns all items in a single list.

Value

For `get_paged_list`, a list.

For `is_url`, whether the object appears to be a URL (is character of length 1, and starts with the string `"http"`). Optionally, restricts the check to HTTPS URLs only.

`lock` *Management locks*

Description

Create, retrieve and delete locks. These are methods for the `az_subscription`, `az_resource_group` and `az_resource` classes.

Usage

```
create_lock(name, level = c("cannotdelete", "readonly"), notes = "")

get_lock(name)

delete_lock(name)

list_locks()
```

Arguments

- `name`: The name of a lock.
- `level`: The level of protection that the lock provides.
- `notes`: An optional character string to describe the lock.

Details

Management locks in Resource Manager can be assigned at the subscription, resource group, or resource level. They serve to protect a resource against unwanted changes. A lock can either protect against deletion (`level="cannotdelete"`) or against modification of any kind (`level="readonly"`).

Locks assigned at parent scopes also apply to lower ones, recursively. The most restrictive lock in the inheritance takes precedence. To modify/delete a resource, any existing locks for its subscription and resource group must also be removed.

Note if you logged in via a custom service principal, it must have "Owner" or "User Access Administrator" access to manage locks.

Value

The `create_lock` and `get_lock` methods return a lock object, which is itself an Azure resource. The `list_locks` method returns a list of such objects. The `delete_lock` method returns NULL on a successful delete.

The `get_role_definition` method returns an object of class `az_role_definition`. This is a plain-old-data R6 class (no methods), which can be used as input for creating role assignments (see the examples below).

The `list_role_definitions` method returns a list of `az_role_definition` if the `as_data_frame` argument is FALSE. If this is TRUE, it instead returns a data frame containing the most broadly useful fields for each role definition: the definition ID and role name.

See Also

rbac

Overview of management locks

Examples

```
## Not run:

az <- get_azure_login("myaadtenant")
sub <- az$get_subscription("subscription_id")
rg <- sub$get_resource_group("rgname")
res <- rg$get_resource(type="provider_type", name="resname")

sub$create_lock("lock1", "cannotdelete")
rg$create_lock("lock2", "cannotdelete")

# error! resource is locked
res$delete()

# subscription level
rg$delete_lock("lock2")
sub$delete_lock("lock1")

# now it works
res$delete()


## End(Not run)
```

`rbac` *Role-based access control (RBAC)*

Description

Basic methods for RBAC: manage role assignments and retrieve role definitions. These are methods for the `az_subscription`, `az_resource_group` and `az_resource` classes.

Usage

```
add_role_assignment(principal, role, scope = NULL)

get_role_assignment(id)

remove_role_assignment(id, confirm = TRUE)

list_role_assignments(filter = "atScope()", as_data_frame = TRUE)

get_role_definition(id)

list_role_definitions(filter=NULL, as_data_frame = TRUE)
```

Arguments

- `principal`: For `add_role_assignment`, the principal for which to assign a role. This can be a GUID, or an object of class `az_user`, `az_app` or `az_storage_principal` (from the AzureGraph package).
- `role`: For `add_role_assignment`, the role to assign the principal. This can be a GUID, a string giving the role name (eg "Contributor"), or an object of class `[az_role_definition]`.
- `scope`: For `add_role_assignment`, an optional scope for the assignment.
- `id`: A role ID. For `get_role_assignment` and `remove_role_assignment`, this is a role assignment GUID. For `get_role_definition`, this can be a role definition GUID or a role name.
- `confirm`: For `remove_role_assignment`, whether to ask for confirmation before removing the role assignment.
- `filter`: For `list_role_assignments` and `list_role_definitions`, an optional filter condition to limit the returned roles.
- `as_data_frame`: For `list_role_assignments` and `list_role_definitions`, whether to return a data frame or a list of objects. See 'Value' below.

Details

AzureRMR implements a subset of the full RBAC functionality within Azure Active Directory. You can retrieve role definitions and add and remove role assignments, at the subscription, resource group and resource levels.

Value

The `add_role_assignment` and `get_role_assignment` methods return an object of class `az_role_assignment`. This is a simple R6 class, with one method: `remove` to remove the assignment.

The `list_role_assignments` method returns a list of `az_role_assignment` objects if the `as_data_frame` argument is FALSE. If this is TRUE, it instead returns a data frame containing the most broadly useful fields for each assigned role: the role assignment ID, the principal, and the role name.

The `get_role_definition` method returns an object of class `az_role_definition`. This is a plain-old-data R6 class (no methods), which can be used as input for creating role assignments (see the examples below).

The `list_role_definitions` method returns a list of `az_role_definition` if the `as_data_frame` argument is FALSE. If this is TRUE, it instead returns a data frame containing the most broadly useful fields for each role definition: the definition ID and role name.

See Also

az_rm, az_role_definition, az_role_assignment

Overview of role-based access control

Examples

```
## Not run:

az <- get_azure_login("myaadtenant")
sub <- az$get_subscription("subscription_id")
rg <- sub$get_resource_group("rgname")
res <- rg$get_resource(type="provider_type", name="resname")

sub$list_role_definitions()
sub$list_role_assignments()
sub$get_role_definition("Contributor")

# get an app using the AzureGraph package
app <- get_graph_login("myaadtenant")$get_app("app_id")

# subscription level
asn1 <- sub$add_role_assignment(app, "Reader")

# resource group level
asn2 <- rg$add_role_assignment(app, "Contributor")

# resource level
asn3 <- res$add_role_assignment(app, "Owner")

res$remove_role_assignment(asn3$id)
rg$remove_role_assignment(asn2$id)
sub$remove_role_assignment(asn1$id)


## End(Not run)
```

Index