Package ‘AzureStor’

March 4, 2020

Title  Storage Management in 'Azure'
Version  3.1.1
Description  Manage storage in Microsoft's 'Azure' cloud: <https://azure.microsoft.com/services/storage>. On the admin side, 'AzureStor' includes features to create, modify and delete storage accounts. On the client side, it includes an interface to blob storage, file storage, and 'Azure Data Lake Storage Gen2': upload and download files and blobs; list containers and files/blobs; create containers; and so on. Authenticated access to storage is supported, via either a shared access key or a shared access signature (SAS). Part of the 'AzureR' family of packages.

License  MIT + file LICENSE


BugReports  https://github.com/Azure/AzureStor/issues

VignetteBuilder  knitr

Depends  R (>= 3.3),
Imports  utils, R6, httr, mime, openssl, xml2, AzureRMR (>= 2.3.0)
Suggests  knitr, jsonlite, testthat, processx

RoxygenNote  7.0.2.9000

NeedsCompilation  no

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

Maintainer  Hong Ooi <hongooi@microsoft.com>

Repository  CRAN

Date/Publication  2020-03-04 16:50:02 UTC

R topics documented:

acquire_lease ................................................................. 2
adls_filesystem .................................................................. 3
az_storage ......................................................................... 5
acquire_lease

Operations on blob leases

Description
Manage leases for blobs and blob containers.

Usage
acquire_lease(container, blob = "", duration = 60, lease = NULL)
break_lease(container, blob = "", period = NULL)
release_lease(container, blob = "", lease)
renew_lease(container, blob = "", lease)
change_lease(container, blob = "", lease, new_lease)

Arguments
container A blob container object.
blob The name of an individual blob. If not supplied, the lease applies to the entire container.
duration For acquire_lease, The duration of the requested lease. For an indefinite duration, set this to -1.
lease For acquire_lease an optional proposed name of the lease; for release_lease, renew_lease and change_lease, the name of the existing lease.
period For breaklease, the period for which to break the lease.
new_lease For changelease, the proposed name of the lease.

Details
Leasing is a way to prevent a blob or container from being accidentally deleted. The duration of a lease can range from 15 to 60 seconds, or be indefinite.

Value
For acquirelease and changelease, a string containing the lease ID.

See Also
blob_container, Leasing a blob, Leasing a container

---

**adls_filesystem**

*Operations on an Azure Data Lake Storage Gen2 endpoint*

Description
Get, list, create, or delete ADLSgen2 filesystems.

Usage

```r
adls_filesystem(endpoint, ...)  
## S3 method for class 'character'
adls_filesystem(endpoint, key = NULL, token = NULL,  
    sas = NULL, api_version = getOption("azure_storage_api_version"), ...)  
## S3 method for class 'adls_endpoint'
adls_filesystem(endpoint, name, ...)  
## S3 method for class 'adls_filesystem'
print(x, ...)  
list_adls_filesystems(endpoint, ...)  
## S3 method for class 'character'
list_adls_filesystems(endpoint, key = NULL,  
    token = NULL, sas = NULL,  
    api_version = getOption("azure_adls_api_version"), ...)  
## S3 method for class 'adls_endpoint'
list_adls_filesystems(endpoint, ...)  
```
create_adls_filesystem(endpoint, ...)

## S3 method for class 'character'
create_adls_filesystem(endpoint, key = NULL,
                        token = NULL, sas = NULL,
                        api_version = getOption("azure_adls_api_version"), ...)

## S3 method for class 'adls_filesystem'
create_adls_filesystem(endpoint, ...)

## S3 method for class 'adls_endpoint'
create_adls_filesystem(endpoint, name, ...)

delete_adls_filesystem(endpoint, ...)

## S3 method for class 'character'
delete_adls_filesystem(endpoint, key = NULL,
                         token = NULL, sas = NULL,
                         api_version = getOption("azure_adls_api_version"), ...)

## S3 method for class 'adls_filesystem'
delete_adls_filesystem(endpoint, ...)

## S3 method for class 'adls_endpoint'
delete_adls_filesystem(endpoint, name, confirm = TRUE, ...)

Arguments

endpoint Either an ADLSgen2 endpoint object as created by \texttt{storage_endpoint} or \texttt{adls_endpoint},
or a character string giving the URL of the endpoint.

... Further arguments passed to lower-level functions.

key, token, sas If an endpoint object is not supplied, authentication credentials: either an access
                 key, an Azure Active Directory (AAD) token, or a SAS, in that order of priority. Currently the sas argument is unused.

api_version If an endpoint object is not supplied, the storage API version to use when interacting with the host. Currently defaults to "2018-11-09".

name The name of the filesystem to get, create, or delete.

x For the print method, a filesystem object.

close For deleting a filesystem, whether to ask for confirmation.

Details

You can call these functions in a couple of ways: by passing the full URL of the filesystem, or by passing the endpoint object and the name of the filesystem as a string.

If authenticating via AAD, you can supply the token either as a string, or as an object of class AzureToken, created via \texttt{AzureRMR::get_azure_token}. The latter is the recommended way of doing it, as it allows for automatic refreshing of expired tokens.
Value

For `adls_filesystem` and `create_adls_filesystem`, an S3 object representing an existing or created filesystem respectively.

For `list_adls_filesystems`, a list of such objects.

See Also

`storage_endpoint`, `az_storage`, `storage_container`  

Examples

```r
## Not run:

endp <- adls_endpoint("https://mystorage.dfs.core.windows.net/", key="access_key")

# list ADLSgen2 filesystems
list_adls_filesystems(endp)

# get, create, and delete a filesystem
adls_filesystem(endp, "myfs")
create_adls_filesystem(endp, "newfs")
delete_adls_filesystem(endp, "newfs")

# alternative way to do the same
adls_filesystem("https://mystorage.dfs.core.windows.net/myfs", key="access_key")
create_adls_filesystem("https://mystorage.dfs.core.windows.net/newfs", key="access_key")
delete_adls_filesystem("https://mystorage.dfs.core.windows.net/newfs", key="access_key")

## End(Not run)
```

---

az_storage | Storage account resource class

Description

Class representing a storage account, exposing methods for working with it.

Methods

The following methods are available, in addition to those provided by the `AzureRMR::az_resource` class:

- `new(...)`: Initialize a new storage object. See 'Initialization'.
- `list_keys()`: Return the access keys for this account.
- `get_account_sas(...)`: Return an account shared access signature (SAS). See 'Shared access signatures' for more details.
• `get_blob_endpoint(key, sas)`: Return the account’s blob storage endpoint, along with an access key and/or a SAS. See ‘Endpoints’ for more details.
• `get_file_endpoint(key, sas)`: Return the account’s file storage endpoint.
• `regen_key(key)`: Regenerates (creates a new value for) an access key. The argument key can be 1 or 2.

Initialization

Initializing a new object of this class can either retrieve an existing storage account, or create a account on the host. Generally, the best way to initialize an object is via the `get_storage_account`, `create_storage_account` or `list_storage_accounts` methods of the `az_resource_group` class, which handle the details automatically.

Shared access signatures

The simplest way for a user to access files and data in a storage account is to give them the account’s access key. This gives them full control of the account, and so may be a security risk. An alternative is to provide the user with a shared access signature (SAS), which limits access to specific resources and only for a set length of time.

To create an account SAS, call the `get_account_sas()` method with the following arguments:

• `start`: The starting access date/time, as a Date or POSIXct value. Defaults to the current time.
• `expiry`: The ending access date/time, as a Date or POSIXct value. Defaults to 8 hours after the start time.
• `services`: Which services to allow access to. A string containing a combination of the letters `b`, `f`, `q`, `t` for blob, file, queue and table access. Defaults to `bfqt`.
• `permissions`: Which permissions to grant. A string containing a combination of the letters `r` (read), `w` (write), `d` (delete), `l` (list), `a` (add), `c` (create), `u` (update) , `p` (process). Defaults to `r`.
• `resource_types`: Which levels of the resource type hierarchy to allow access to. A string containing a combination of the letters `s` (service), `c` (container), `o` (object). Defaults to `sco`.
• `ip`: An IP address or range to grant access to.
• `protocol`: Which protocol to allow, either "http", "http,https" or "https". Defaults to NULL, which is the same as "http,https".
• `key`: the access key used to sign (authorise) the SAS.

Endpoints

The client-side interaction with a storage account is via an endpoint. A storage account can have several endpoints, one for each type of storage supported: blob, file, queue and table.

The client-side interface in AzureStor is implemented using S3 classes. This is for consistency with other data access packages in R, which mostly use S3. It also emphasises the distinction between Resource Manager (which is for interacting with the storage account itself) and the client (which is for accessing files and data stored in the account).

To create a storage endpoint independently of Resource Manager (for example if you are a user without admin or owner access to the account), use the `blob_endpoint` or `file_endpoint` functions.
If a storage endpoint is created without an access key and SAS, only public (anonymous) access is possible.

See Also

blob_endpoint, file_endpoint, create_storage_account, get_storage_account, delete_storage_account, Date, POSIXt, Azure Storage Provider API reference, Azure Storage Services API reference

Examples

## Not run:

```
# recommended way of retrieving a resource: via a resource group object
stor <- resgroup$get_storage_account("mystorage")

# list account access keys
stor$list_keys()

# regenerate a key
stor$regen_key(1)

# generate a shared access signature for blob storage, expiring in 7 days time
today <- Sys.time()
stor$get_account_sas(expiry=today + 7*24*60*60, services="b", permissions="rw")

# storage endpoints
stor$get_blob_endpoint()
stor$get_file_endpoint()
```

## End(Not run)

---

**blob_container**  
*Operations on a blob endpoint*

**Description**

Get, list, create, or delete blob containers.

**Usage**

`blob_container(endpoint, ...)`

```
## S3 method for class 'character'
blob_container(endpoint, key = NULL, token = NULL, 
               sas = NULL, api_version = getOption("azure_storage_api_version"), ...)

## S3 method for class 'blob_endpoint'
blob_container(endpoint, name, ...)
```
## S3 method for class 'blob_container'
print(x, ...)

list_blob_containers(endpoint, ...)

## S3 method for class 'character'
list_blob_containers(endpoint, key = NULL, token = NULL,
sas = NULL, api_version = getOption("azure_storage_api_version"), ...)

download_blob_containers(endpoint, ..., key = NULL, token = NULL,
sas = NULL, api_version = getOption("azure_storage_api_version"), ...)

## S3 method for class 'blob_endpoint'
list_blob_containers(endpoint, ...)

create_blob_container(endpoint, ...)

## S3 method for class 'character'
create_blob_container(endpoint, key = NULL,
token = NULL, sas = NULL,
api_version = getOption("azure_storage_api_version"), ...)

delete_blob_container(endpoint, ...)

## S3 method for class 'blob_container'
delete_blob_container(endpoint, ...)  

create_blob_container(endpoint, name,
  public_access = c("none", "blob", "container"), ...)

delete_blob_container(endpoint, ...)  

## S3 method for class 'blob_endpoint'
delete_blob_container(endpoint, name, confirm = TRUE, lease = NULL, ...)

Arguments

- **endpoint**: Either a blob endpoint object as created by `storage_endpoint`, or a character string giving the URL of the endpoint.
- **...**: Further arguments passed to lower-level functions.
- **key, token, sas**: If an endpoint object is not supplied, authentication credentials: either an access key, an Azure Active Directory (AAD) token, or a SAS, in that order of priority. If no authentication credentials are provided, only public (anonymous) access to
the share is possible.

**api_version**
If an endpoint object is not supplied, the storage API version to use when interacting with the host. Currently defaults to "2018-11-09".

**name**
The name of the blob container to get, create, or delete.

**x**
For the print method, a blob container object.

**public_access**
For creating a container, the level of public access to allow.

**confirm**
For deleting a container, whether to ask for confirmation.

**lease**
For deleting a leased container, the lease ID.

**Details**
You can call these functions in a couple of ways: by passing the full URL of the share, or by passing the endpoint object and the name of the container as a string.

If authenticating via AAD, you can supply the token either as a string, or as an object of class AzureToken, created via AzureRMR::get_azure_token. The latter is the recommended way of doing it, as it allows for automatic refreshing of expired tokens.

**Value**
For blob_container and create_blob_container, an S3 object representing an existing or created container respectively.

For list_blob_containers, a list of such objects.

**See Also**

`storage_endpoint`, `az_storage`, `storage_container`

**Examples**

```r
## Not run:

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

# list containers
list_blob_containers(endp)

# get, create, and delete a container
blob_container(endp, "mycontainer")
create_blob_container(endp, "newcontainer")
delete_blob_container(endp, "newcontainer")

# alternative way to do the same
blob_container("https://mystorage.blob.core.windows.net/mycontainer", key="access_key")
create_blob_container("https://mystorage.blob.core.windows.net/newcontainer", key="access_key")
delete_blob_container("https://mystorage.blob.core.windows.net/newcontainer", key="access_key")

# authenticating via AAD
token <- AzureRMR::get_azure_token(resource="https://storage.azure.com/",
```
call_azcopy

tenant="myaadtenant",
app="myappid",
password="mypassword")
blob_container("https://mystorage.blob.core.windows.net/mycontainer", token=token)

## End(Not run)

call_azcopy Call the azcopy file transfer utility

Description
Call the azcopy file transfer utility

Usage
call_azcopy(..., env = NULL, silent = FALSE)

Arguments
... Arguments to pass to AzCopy on the commandline. If no arguments are sup-
plied, a help screen is printed.
env A named character vector of environment variables to set for AzCopy.
silent Whether to print the output from AzCopy to the screen; also sets whether an
error return code from AzCopy will be propagated to an R error.

Details
AzureStor has the ability to use the Microsoft AzCopy commandline utility to transfer files. To
enable this, ensure the processx package is installed and set the argument use_azcopy=TRUE in any
call to an upload or download function; AzureStor will then call AzCopy to perform the file transfer
rather than relying on its own code. You can also call AzCopy directly with the call_azcopy
function.

AzureStor requires version 10 or later of AzCopy. The first time you try to run it, AzureStor will
check that the version of AzCopy is correct, and throw an error if it is version 8 or earlier.

The AzCopy utility must be in your path for AzureStor to find it. Note that unlike earlier versions,
Azcopy 10 is a single, self-contained binary file that can be placed in any directory.

Value
A list, invisibly, with the following components:

- status: The exit status of the AzCopy command. If this is NA, then the process was killed
  and had no exit status.
- stdout: The standard output of the command.
- stderr: The standard error of the command.
- timeout: Whether AzCopy was killed because of a timeout.
copy_url_to_storage

See Also

processx::run, download_blob, download_azure_file, download_adls_file

AzCopy page on Microsoft Docs
AzCopy GitHub repo

Examples

## Not run:

```r
endp <- storage_endpoint("https://mystorage.blob.core.windows.net", sas="mysas")
cont <- storage_container(endp, "mycontainer")

# print various help screens
call_azcopy("help")
call_azcopy("help", "copy")

# calling azcopy to download a blob
storage_download(cont, "myblob.csv", use_azcopy=TRUE)

# calling azcopy directly (must specify the SAS explicitly in the source URL)
call_azcopy("copy",
  "https://mystorage.blob.core.windows.net/mycontainer/myblob.csv?mysas",
  "myblob.csv")

## End(Not run)
```

---

### copy_url_to_storage

Upload and download generics

#### Description

Upload and download generics

#### Usage

```r
copy_url_to_storage(container, src, dest, ...)

multicopy_url_to_storage(container, src, dest, ...)
```

## S3 method for class 'blob_container'

```r
copy_url_to_storage(container, src, dest, ...)
```

## S3 method for class 'blob_container'

```r
multicopy_url_to_storage(container, src, dest, ...)
```

```r
storage_upload(container, ...)
```
## S3 method for class 'blob_container'
storage_upload(container, ...)

## S3 method for class 'file_share'
storage_upload(container, ...)

## S3 method for class 'adls_filesystem'
storage_upload(container, ...)

storage_multiupload(container, ...)

## S3 method for class 'blob_container'
storage_multiupload(container, ...)

## S3 method for class 'file_share'
storage_multiupload(container, ...)

## S3 method for class 'adls_filesystem'
storage_multiupload(container, ...)

storage_download(container, ...)

## S3 method for class 'blob_container'
storage_download(container, ...)

## S3 method for class 'file_share'
storage_download(container, ...)

## S3 method for class 'adls_filesystem'
storage_download(container, ...)

storage_multidownload(container, ...)

## S3 method for class 'blob_container'
storage_multidownload(container, ...)

## S3 method for class 'file_share'
storage_multidownload(container, ...)

## S3 method for class 'adls_filesystem'
storage_multidownload(container, ...)

download_from_url(src, dest, key = NULL, token = NULL, sas = NULL, ..., overwrite = FALSE)

upload_to_url(src, dest, key = NULL, token = NULL, sas = NULL, ...)
Arguments

- **container**: A storage container object.
- **src, dest**: For `upload_to_url` and `download_from_url`, the source and destination files to transfer.
- **...**: Further arguments to pass to lower-level functions.
- **key, token, sas**: Authentication arguments: an access key, Azure Active Directory (AAD) token or a shared access signature (SAS). If multiple arguments are supplied, a key takes priority over a token, which takes priority over a SAS. For `upload_to_url` and `download_to_url`, you can also provide a SAS as part of the URL itself.
- **overwrite**: For downloading, whether to overwrite any destination files that exist.

Details

copy_url_to_storage transfers the contents of the file at the specified HTTP[S] URL directly to storage, without requiring a temporary local copy to be made. multicopy_url_to_storage does the same, for multiple URLs at once. Currently methods for these are only implemented for blob storage.

These functions allow you to transfer files to and from a storage account.

storage_upload, storage_download, storage_multiupload and storage_multidownload take as first argument a storage container, either for blob storage, file storage, or ADLSgen2. They dispatch to the corresponding file transfer functions for the given storage type.

upload_to_url and download_to_url allow you to transfer a file to or from Azure storage, given the URL of the source or destination. The storage details (endpoint, container name, and so on) are obtained from the URL.

By default, the upload and download functions will display a progress bar while they are downloading. To turn this off, use `options(azure_storage_progress_bar=FALSE)`. To turn the progress bar back on, use `options(azure_storage_progress_bar=TRUE)`.

See Also

- `storage_container`, `blob_container`, `file_share`, `adls_filesystem`
- `download_blob`, `download_azure_file`, `download_adls_file`, `call_azcopy`

Examples

```r
## Not run:

# download from blob storage
bl <- storage_endpoint("https://mystorage.blob.core.windows.net/", key="access_key")
cont <- storage_container(bl, "mycontainer")
storage_download(cont, "bigfile.zip", "/bigfile.zip")

# same download but directly from the URL
download_from_url("https://mystorage.blob.core.windows.net/mycontainer/bigfile.zip",
    "/bigfile.zip",
    key="access_key")
```
# upload to ADLSgen2
ad <- storage_endpoint("https://myadls.dfs.core.windows.net/", token=mytoken)
cont <- storage_container(ad, "myfilesystem")
create_storage_dir(cont, "newdir")
storage_upload(cont, "files.zip", "newdir/files.zip")

# same upload but directly to the URL
upload_to_url("files.zip",
    "https://myadls.dfs.core.windows.net/myfilesystem/newdir/files.zip",
    token=mytoken)

## End(Not run)

---

**create_storage_account**

Create Azure storage account

---

**Description**

Method for the `AzureRMR::az_resource_group` class.

**Usage**

```r
create_storage_account(name, location, kind = "StorageV2", replication = "Standard_LRS",
                       access_tier = "hot", https_only = TRUE, hierarchical_namespace_enabled = FALSE,
                       properties = list(), ...)
```

**Arguments**

- `name`: The name of the storage account.
- `location`: The location/region in which to create the account. Defaults to the resource group location.
- `kind`: The type of account, either "StorageV2" (the default), "FileStorage" or "BlobStorage".
- `replication`: The replication strategy for the account. The default is locally-redundant storage (LRS).
- `access_tier`: The access tier, either "hot" or "cool", for blobs.
- `https_only`: Whether a HTTPS connection is required to access the storage.
- `hierarchical_namespace_enabled`: Whether to enable hierarchical namespaces, which are a feature of Azure Data Lake Storage Gen 2 and provide more efficient way to manage storage. See 'Details' below.
- `properties`: A list of other properties for the storage account.
- `...`: Other named arguments to pass to the `az_storage` initialization function.
create_storage_account

Details

This method deploys a new storage account resource, with parameters given by the arguments. A storage account can host multiple types of storage:

- blob storage
- file storage
- table storage
- queue storage
- Azure Data Lake Storage Gen2

Accounts created with kind = "BlobStorage" can only host blob storage, while those with kind = "FileStorage" can only host file storage. Accounts with kind = "StorageV2" can host all types of storage. Currently, AzureStor provides an R interface to ADLSgen2, blob and file storage.

Currently (as of October 2019), if hierarchical namespaces are enabled, the blob API for the account is disabled. The blob endpoint is still accessible, but blob operations on the endpoint will fail. Full interoperability between blobs and ADLSgen2 is planned for later in 2019.

Value

An object of class az_storage representing the created storage account.

See Also

get_storage_account, delete_storage_account, az_storage

Azure Storage documentation, Azure Storage Provider API reference, Azure Data Lake Storage hierarchical namespaces

Examples

```r
## Not run:

rg <- AzureRM::az_rm:
  new(tenant="myaadtenant.onmicrosoft.com", app="app_id", password="password")$
  get_subscription("subscription_id")$
  get_resource_group("rgname")

# create a new storage account
rg$create_storage_account("mystorage", kind="StorageV2")

# create a blob storage account in a different region
rg$create_storage_account("myblobstorage",
  location="australiasoutheast",
  kind="BlobStorage")
```

## End(Not run)
delete_storage_account

*Delete an Azure storage account*

**Description**

Method for the `AzureRMR::az_resource_group` class.

**Usage**

```r
delete_storage_account(name, confirm=TRUE, wait=FALSE)
```

**Arguments**

- `name`: The name of the storage account.
- `confirm`: Whether to ask for confirmation before deleting.
- `wait`: Whether to wait until the deletion is complete.

**Value**

NULL on successful deletion.

**See Also**

`create_storage_account`, `get_storage_account`, `az_storage`, [Azure Storage Provider API reference](#)

**Examples**

```r
## Not run:
rg <- AzureRMR::az_rm$
   new(tenant="myaadtenant.onmicrosoft.com", app="app_id", password="password")$
   get_subscription("subscription_id")$
   get_resource_group("rgname")

# delete a storage account
rg$delete_storage_account("mystorage")
```

## End(Not run)
do_container_op

Carry out operations on a storage account container or endpoint

Description

Carry out operations on a storage account container or endpoint

Usage

do_container_op(container, operation = "", options = list(),
headers = list(), http_verb = "GET", ...)

call_storage_endpoint(endpoint, path, options = list(), headers = list(),
body = NULL, ..., http_verb = c("GET", "DELETE", "PUT", "POST", "HEAD",
"PATCH"), http_status_handler = c("stop", "warn", "message", "pass"),
timeout = getOption("azure_storage_timeout"), progress = NULL,
return_headers = (http_verb == "HEAD"))

Arguments

container, endpoint
   For do_container_op, a storage container object (inheriting from storage_container).
   For call_storage_endpoint, a storage endpoint object (inheriting from storage_endpoint).
operation
   The container operation to perform, which will form part of the URL path.
options
   A named list giving the query parameters for the operation.
headers
   A named list giving any additional HTTP headers to send to the host. Note that
   AzureStor will handle authentication details, so you don’t have to specify these
   here.
http_verb
   The HTTP verb as a string, one of GET, DELETE, PUT, POST, HEAD or PATCH.
...                          Any additional arguments to pass to httr::VERB.
path
   The path component of the endpoint call.
body
   The request body for a PUT/POST/PATCH call.
http_status_handler
   The R handler for the HTTP status code of the response. "stop", "warn" or
   "message" will call the corresponding handlers in httr, while "pass" ignores
   the status code. The latter is primarily useful for debugging purposes.
timeout
   Optionally, the number of seconds to wait for a result. If the timeout interval
   elapses before the storage service has finished processing the operation, it re-
   turns an error. The default timeout is taken from the system option azure_storage_timeout;
   if this is NULL it means to use the service default.
progress
   Used by the file transfer functions, to display a progress bar.
return_headers
   Whether to return the (parsed) response headers, rather than the body. Ignored
   if http_status_handler="pass".
Details

These functions form the low-level interface between R and the storage API. `do_container_op` constructs a path from the operation and the container name, and passes it and the other arguments to `call_storage_endpoint`.

Value

Based on the `http_status_handler` and `return_headers` arguments. If `http_status_handler` is "pass", the entire response is returned without modification.

If `http_status_handler` is one of "stop", "warn" or "message", the status code of the response is checked, and if an error is not thrown, the parsed headers or body of the response is returned. An exception is if the response was written to disk, as part of a file download; in this case, the return value is NULL.

See Also

`blob_endpoint`, `file_endpoint`, `adls_endpoint`  
`blob_container`, `file_share`, `adls_filesystem`  
`httr::GET`, `httr::PUT`, `httr::POST`, `httr::PATCH`, `httr::HEAD`, `httr::DELETE`

Examples

```r
## Not run:
# get the metadata for a blob
bl_endp <- blob_endpoint("storage_acct_url", key="key")
cont <- storage_container(bl_endp, "containername")
do_container_op(cont, "filename.txt", options=list(comp="metadata"), http_verb="HEAD")

## End(Not run)
```

file_share

Operations on a file endpoint

Description

Get, list, create, or delete file shares.

Usage

```r
file_share(endpoint, ...)
```

## S3 method for class 'character'
```r
file_share(endpoint, key = NULL, token = NULL,
          sas = NULL, api_version = getOption("azure_storage_api_version"), ...)
```
## S3 method for class 'file_endpoint'

file_share(endpoint, name, ...)  

## S3 method for class 'file_share'

print(x, ...)

list_file_shares(endpoint, ...)

## S3 method for class 'character'

list_file_shares(endpoint, key = NULL, token = NULL,  
sas = NULL, api_version = getOption("azure_storage_api_version"), ...)

## S3 method for class 'file_endpoint'

list_file_shares(endpoint, ...)

create_file_share(endpoint, ...)

## S3 method for class 'character'

create_file_share(endpoint, key = NULL, token = NULL,  
sas = NULL, api_version = getOption("azure_storage_api_version"), ...)

## S3 method for class 'file_share'

create_file_share(endpoint, ...)

## S3 method for class 'file_endpoint'

create_file_share(endpoint, name, ...)

delete_file_share(endpoint, ...)

## S3 method for class 'character'

delete_file_share(endpoint, key = NULL, token = NULL,  
sas = NULL, api_version = getOption("azure_storage_api_version"), ...)

## S3 method for class 'file_share'

delete_file_share(endpoint, ...)

## S3 method for class 'file_endpoint'

delete_file_share(endpoint, name, confirm = TRUE, ...)

### Arguments

- **endpoint**: Either a file endpoint object as created by `storage_endpoint`, or a character string giving the URL of the endpoint.
- **...**: Further arguments passed to lower-level functions.
- **key, token, sas**: If an endpoint object is not supplied, authentication credentials: either an access key, an Azure Active Directory (AAD) token, or a SAS, in that order of priority.
- **api_version**: If an endpoint object is not supplied, the storage API version to use when interacting with the host. Currently defaults to "2018-11-09".
**get_storage_account**

- **name**: The name of the file share to get, create, or delete.
- **x**: For the print method, a file share object.
- **confirm**: For deleting a share, whether to ask for confirmation.

**Details**

You can call these functions in a couple of ways: by passing the full URL of the share, or by passing the endpoint object and the name of the share as a string.

**Value**

For `file_share` and `create_file_share`, an S3 object representing an existing or created share respectively.

For `list_file_shares`, a list of such objects.

**See Also**

`storage_endpoint`, `az_storage`, `storage_container`

**Examples**

```r
## Not run:

dp <- file_endpoint("https://mystorage.file.core.windows.net/", key="access_key")

# list file shares
list_file_shares(dp)

# get, create, and delete a file share
file_share(dp, "myshare")
create_file_share(dp, "newshare")
delete_file_share(dp, "newshare")

# alternative way to do the same
file_share("https://mystorage.file.file.windows.net/myshare", key="access_key")
create_file_share("https://mystorage.file.core.windows.net/newshare", key="access_key")
delete_file_share("https://mystorage.file.core.windows.net/newshare", key="access_key")

## End(Not run)
```

---

**get_storage_account** *Get existing Azure storage account(s)*

**Description**

Methods for the `AzureRMR::az_resource_group` and `AzureRMR::az_subscription` classes.
Usage

get_storage_account(name)
list_storage_accounts()

Arguments

- name: For get_storage_account(), the name of the storage account.

Details

The AzureRMR::az_resource_group class has both get_storage_account() and list_storage_accounts() methods, while the AzureRMR::az_subscription class only has the latter.

Value

For get_storage_account(), an object of class az_storage representing the storage account.
For list_storage_accounts(), a list of such objects.

See Also

create_storage_account, delete_storage_account, az_storage, Azure Storage Provider API reference

Examples

## Not run:

```r
rg <- AzureRMR::az_rm$
  new(tenant="myaadtenant.onmicrosoft.com", app="app_id", password="password")$
  get_subscription("subscription_id")$
  get_resource_group("rgname")$

# get a storage account
rg$get_storage_account("mystorage")
```

## End(Not run)

---

**get_storage_metadata**  
Get/set user-defined metadata for a storage object

Description

Get/set user-defined metadata for a storage object
get_storage_metadata

Usage

get_storage_metadata(object, ...)

## S3 method for class 'blob_container'
get_storage_metadata(object, blob, ...)

## S3 method for class 'file_share'
get_storage_metadata(object, file, isdir, ...)

## S3 method for class 'adls_filesystem'
get_storage_metadata(object, file, ...)

set_storage_metadata(object, ...)

## S3 method for class 'blob_container'
set_storage_metadata(object, blob, ..., keep_existing = TRUE)

## S3 method for class 'file_share'
set_storage_metadata(object, file, isdir, ..., keep_existing = TRUE)

## S3 method for class 'adls_filesystem'
set_storage_metadata(object, file, ..., keep_existing = TRUE)

Arguments

object A blob container, file share or ADLS filesystem object.
...
For the metadata setters, name-value pairs to set as metadata for a blob or file.
blob, file Optionally the name of an individual blob, file or directory within a container.
isdir For the file share method, whether the file argument is a file or directory. If omitted, get_storage_metadata will auto-detect the type; however this can be slow, so supply this argument if possible.
keep_existing For the metadata setters, whether to retain existing metadata information.

Details

These methods let you get and set user-defined properties (metadata) for storage objects.

Value

get_storage_metadata returns a named list of metadata properties. If the blob or file argument is present, the properties will be for the blob/file specified. If this argument is omitted, the properties will be for the container itself.

set_storage_metadata returns the same list after setting the object’s metadata, invisibly.

See Also

blob_container, file_share, adls_filesystem

get_storage_properties for standard properties
get_storage_properties

Examples

## Not run:

```r
fs <- storage_container("https://mystorage.dfs.core.windows.net/myshare", key="access_key")
create_storage_dir("newdir")
storage_upload(shared, "iris.csv", "newdir/iris.csv")

set_storage_metadata(fs, "newdir/iris.csv", name1="value1")
# will be list(name1="value1")
get_storage_metadata(fs, "newdir/iris.csv")

set_storage_metadata(fs, "newdir/iris.csv", name2="value2")
# will be list(name1="value1", name2="value2")
get_storage_metadata(fs, "newdir/iris.csv")

set_storage_metadata(fs, "newdir/iris.csv", name3="value3", keep_existing=FALSE)
# will be list(name3="value3")
get_storage_metadata(fs, "newdir/iris.csv")

# deleting all metadata
set_storage_metadata(fs, "newdir/iris.csv", keep_existing=FALSE)
```

## End(Not run)

get_storage_properties

Get storage properties for an object

Description

Get storage properties for an object

Usage

```r
get_storage_properties(object, ...)
```

## S3 method for class 'blob_container'
```r
get_storage_properties(object, blob, ...)
```

## S3 method for class 'file_share'
```r
get_storage_properties(object, file, isdir, ...)
```

## S3 method for class 'adls_filesystem'
```r
get_storage_properties(object, file, ...)
```

get_adls_file_acl(filesystem, file)

get_adls_file_status(filesystem, file)
get_storage_properties

Arguments

object  A blob container, file share, or ADLS filesystem object.

...  For compatibility with the generic.

blob, file  Optionally the name of an individual blob, file or directory within a container.

isdir  For the file share method, whether the file argument is a file or directory. If omitted, get_storage_properties will auto-detect the type; however this can be slow, so supply this argument if possible.

filesystem  An ADLS filesystem.

Value

get_storage_properties returns a list describing the object properties. If the blob or file argument is present for the container methods, the properties will be for the blob/file specified. If this argument is omitted, the properties will be for the container itself.

get_adls_file_acl returns a string giving the ADLSgen2 ACL for the file.

get_adls_file_status returns a list of ADLSgen2 system properties for the file.

See Also

blob_container, file_share, adls_filesystem

get_storage_metadata for getting and setting user-defined properties (metadata)

Examples

```r
## Not run:

fs <- storage_container("https://mystorage.dfs.core.windows.net/myshare", key="access_key")
create_storage_dir("newdir")
storage_upload(share, "iris.csv", "newdir/iris.csv")

get_storage_properties(fs)
get_storage_properties(fs, "newdir")
get_storage_properties(fs, "newdir/iris.csv")

# these are ADLS only
get_adls_file_acl(fs, "newdir/iris.csv")
get_adls_file_status(fs, "newdir/iris.csv")

## End(Not run)
```
list_adls_files

Operations on an Azure Data Lake Storage Gen2 filesystem

Description
Upload, download, or delete a file; list files in a directory; create or delete directories; check file existence.

Usage
list_adls_files(filesystem, dir = "/", info = c("all", "name"), recursive = FALSE)
multiupload_adls_file(filesystem, src, dest, recursive = FALSE, blocksize = 2^22, lease = NULL, use_azcopy = FALSE, max_concurrent_transfers = 10)
upload_adls_file(filesystem, src, dest = basename(src), blocksize = 2^24, lease = NULL, use_azcopy = FALSE)
multidownload_adls_file(filesystem, src, dest, recursive = FALSE, blocksize = 2^24, overwrite = FALSE, use_azcopy = FALSE, max_concurrent_transfers = 10)
download_adls_file(filesystem, src, dest = basename(src), blocksize = 2^24, overwrite = FALSE, use_azcopy = FALSE)
delete_adls_file(filesystem, file, confirm = TRUE)
create_adls_dir(filesystem, dir)
delete_adls_dir(filesystem, dir, recursive = FALSE, confirm = TRUE)
adls_file_exists(filesystem, file)

Arguments
filesystem An ADLSgen2 filesystem object.
dir, file A string naming a directory or file respectively.
info Whether to return names only, or all information in a directory listing.
recursive For the multiupload/download functions, whether to recursively transfer files in subdirectories. For list_adls_files, and delete_adls_dir, whether the operation should recurse through subdirectories. For delete_adls_dir, this must be TRUE to delete a non-empty directory.
src, dest The source and destination paths/files for uploading and downloading. See 'Details' below.
blocksize  The number of bytes to upload/download per HTTP(S) request.

lease    The lease for a file, if present.

use_azcopy Whether to use the AzCopy utility from Microsoft to do the transfer, rather than doing it in R.

max_concurrent_transfers  For multiupload_adls_file and multidownload_adls_file, the maximum number of concurrent file transfers. Each concurrent file transfer requires a separate R process, so limit this if you are low on memory.

overwrite When downloading, whether to overwrite an existing destination file.

confirm Whether to ask for confirmation on deleting a file or directory.

Details

upload_adls_file and download_adls_file are the workhorse file transfer functions for ADLS-gen2 storage. They each take as inputs a single filename as the source for uploading/downloading, and a single filename as the destination. Alternatively, for uploading, src can be a textConnection or rawConnection object; and for downloading, dest can be NULL or a rawConnection object. If dest is NULL, the downloaded data is returned as a raw vector, and if a raw connection, it will be placed into the connection. See the examples below.

multiupload_adls_file and multidownload_adls_file are functions for uploading and downloading multiple files at once. They parallelise file transfers by using the background process pool provided by AzureRMR, which can lead to significant efficiency gains when transferring many small files. There are two ways to specify the source and destination for these functions:

- Both src and dest can be vectors naming the individual source and destination pathnames.
- The src argument can be a wildcard pattern expanding to one or more files, with dest naming a destination directory. In this case, if recursive is true, the file transfer will replicate the source directory structure at the destination.

upload_adls_file and download_adls_file can display a progress bar to track the file transfer. You can control whether to display this with options(azure_storage_progress_bar=TRUE|FALSE); the default is TRUE.

Value

For list_adls_files, if info="name", a vector of file/directory names. If info="all", a data frame giving the file size and whether each object is a file or directory.

For download_adls_file, if dest=NULL, the contents of the downloaded file as a raw vector.

For adls_file_exists, either TRUE or FALSE.

AzCopy

upload_azure_file and download_azure_file have the ability to use the AzCopy commandline utility to transfer files, instead of native R code. This can be useful if you want to take advantage of AzCopy's logging and recovery features; it may also be faster in the case of transferring a very large number of small files. To enable this, set the use_azcopy argument to TRUE.
Note that AzCopy only supports SAS and AAD (OAuth) token as authentication methods. AzCopy also expects a single filename or wildcard spec as its source/destination argument, not a vector of filenames or a connection.

See Also

`adls_filesystem`, `az_storage`, `storage_download`, `call_azcopy`

Examples

```r
## Not run:

fs <- adls_filesystem("https://mystorage.dfs.core.windows.net/myfilesystem", key="access_key")

list_adls_files(fs, "/")
list_adls_files(fs, "/", recursive=TRUE)

create_adls_dir(fs, "/newdir")

upload_adls_file(fs, "/~bigfile.zip", dest="/newdir/bigfile.zip")
download_adls_file(fs, "/newdir/bigfile.zip", dest="~/bigfile_downloaded.zip")

delete_adls_file(fs, "/newdir/bigfile.zip")
delete_adls_dir(fs, "/newdir")

# uploading/downloading multiple files at once
multiupload_adls_file(fs, "/data/logfiles/*.zip")
multidownload_adls_file(fs, "/monthly/jan*.", "/data/january")

# you can also pass a vector of file/pathnames as the source and destination
src <- c("file1.csv", "file2.csv", "file3.csv")
dest <- paste0("uploaded_", src)
multiupload_adls_file(share, src, dest)

# uploading serialized R objects via connections
json <- jsonlite::toJSON(iris, pretty=TRUE, auto_unbox=TRUE)
con <- textConnection(json)
upload_adls_file(fs, con, "iris.json")

rds <- serialize(iris, NULL)
con <- rawConnection(rds)
upload_adls_file(fs, con, "iris.rds")

# downloading files into memory: as a raw vector, and via a connection
rawvec <- download_adls_file(fs, "iris.json", NULL)
rawToChar(rawvec)

con <- rawConnection(raw(0), "r+")
download_adls_file(fs, "iris.rds", con)
unserialize(con)

## End(Not run)
```
list_azure_files  Operations on a file share

Description

Upload, download, or delete a file; list files in a directory; create or delete directories; check file existence.

Usage

list_azure_files(share, dir = "/", info = c("all", "name"), prefix = NULL, recursive = FALSE)

upload_azure_file(share, src, dest = basename(src), create_dir = FALSE, blocksize = 2^22, use_azcopy = FALSE)

multiupload_azure_file(share, src, dest, recursive = FALSE, create_dir = recursive, blocksize = 2^22, use_azcopy = FALSE, max_concurrent_transfers = 10)

download_azure_file(share, src, dest = basename(src), blocksize = 2^22, overwrite = FALSE, use_azcopy = FALSE)

multidownload_azure_file(share, src, dest, recursive = FALSE, blocksize = 2^22, overwrite = FALSE, use_azcopy = FALSE, max_concurrent_transfers = 10)

delete_azure_file(share, file, confirm = TRUE)

create_azure_dir(share, dir, recursive = FALSE)

delete_azure_dir(share, dir, recursive = FALSE, confirm = TRUE)

azure_file_exists(share, file)

Arguments

- **share**: A file share object.
- **dir, file**: A string naming a directory or file respectively.
- **info**: Whether to return names only, or all information in a directory listing.
- **prefix**: For list_azure_files, filters the result to return only files and directories whose name begins with this prefix.
- **recursive**: For the multiupload/download functions, whether to recursively transfer files in subdirectories. For list_azure_dir, whether to include the contents of any subdirectories in the listing. For create_azure_dir and delete_azure_dir, whether to recursively create/delete each component of a nested directory path.
Note that in all cases this can be slow, so try to use a non-recursive solution if possible.

**src, dest**  The source and destination files for uploading and downloading. See `Details` below.

**create_dir**  For the uploading functions, whether to create the destination directory if it doesn’t exist. Again for the file storage API this can be slow, hence is optional.

**blocksize**  The number of bytes to upload/download per HTTP(S) request.

**use_azcopy**  Whether to use the AzCopy utility from Microsoft to do the transfer, rather than doing it in R.

**max_concurrent_transfers**  For `multiupload_azure_file` and `multidownload_azure_file`, the maximum number of concurrent file transfers. Each concurrent file transfer requires a separate R process, so limit this if you are low on memory.

**overwrite**  When downloading, whether to overwrite an existing destination file.

**confirm**  Whether to ask for confirmation on deleting a file or directory.

### Details

`upload_azure_file` and `download_azure_file` are the workhorse file transfer functions for file storage. They each take as inputs a single filename as the source for uploading/downloading, and a single filename as the destination. Alternatively, for uploading, `src` can be a `textConnection` or `rawConnection` object; and for downloading, `dest` can be `NULL` or a `rawConnection` object. If `dest` is `NULL`, the downloaded data is returned as a raw vector, and if a raw connection, it will be placed into the connection. See the examples below.

`multiupload_azure_file` and `multidownload_azure_file` are functions for uploading and downloading multiple files at once. They parallelise file transfers by using the background process pool provided by `AzureRMR`, which can lead to significant efficiency gains when transferring many small files. There are two ways to specify the source and destination for these functions:

- Both `src` and `dest` can be vectors naming the individual source and destination pathnames.
- The `src` argument can be a wildcard pattern expanding to one or more files, with `dest` naming a destination directory. In this case, if `recursive` is true, the file transfer will replicate the source directory structure at the destination.

`upload_azure_file` and `download_azure_file` can display a progress bar to track the file transfer. You can control whether to display this with `options(azure_storage_progress_bar=TRUE|FALSE)`; the default is `TRUE`.

### Value

For `list_azure_files`, if `info="name"`, a vector of file/directory names. If `info="all"`, a data frame giving the file size and whether each object is a file or directory.

For `download_azure_file`, if `dest=NULL`, the contents of the downloaded file as a raw vector.

For `azure_file_exists`, either TRUE or FALSE.
AzCopy

upload_azure_file and download_azure_file have the ability to use the AzCopy commandline utility to transfer files, instead of native R code. This can be useful if you want to take advantage of AzCopy’s logging and recovery features; it may also be faster in the case of transferring a very large number of small files. To enable this, set the use_azcopy argument to TRUE.

Note that AzCopy only supports SAS and AAD (OAuth) token as authentication methods. AzCopy also expects a single filename or wildcard spec as its source/destination argument, not a vector of filenames or a connection.

See Also

file_share, az_storage, storage_download, call_azcopy

AzCopy version 10 on GitHub

Examples

## Not run:

share <- file_share("https://mystorage/file.core.windows.net/myshare", key="access_key")

list_azure_files(share, "/")
list_azure_files(share, "/", recursive=TRUE)

create_azure_dir(share, "/newdir")

upload_azure_file(share, "/~bigfile.zip", dest="/newdir/bigfile.zip")
download_azure_file(share, "/newdir/bigfile.zip", dest="/bigfile_downloaded.zip")

delete_azure_file(share, "/newdir/bigfile.zip")
delete_azure_dir(share, "/newdir")

# uploading/downloading multiple files at once
multupload_azure_file(share, "/data/logfiles/*.zip")
multidownload_azure_file(share, "/monthly/jan*.", "/data/january")

# you can also pass a vector of file/pathnames as the source and destination
src <- c("file1.csv", "file2.csv", "file3.csv")
dest <- paste0("uploaded_", src)
multiupload_azure_file(share, src, dest)

# uploading serialized R objects via connections
json <- jsonlite::toJSON(iris, pretty=TRUE, auto_unbox=TRUE)
con <- textConnection(json)
upload_azure_file(share, con, "iris.json")

rds <- serialize(iris, NULL)
con <- rawConnection(rds)
upload_azure_file(share, con, "iris.rds")

# downloading files into memory: as a raw vector, and via a connection
rawvec <- download_azure_file(share, "iris.json", NULL)
rawToChar(rawvec)

con <- rawConnection(raw(0), "r+")
download_azure_file(share, "iris.rds", con)
unserialize(con)

## End(Not run)

---

**list_blobs**  
*Operations on a blob container or blob*

**Description**  
Upload, download, or delete a blob; list blobs in a container; check blob availability.

**Usage**

```r
list_blobs(container, dir = "/", info = c("partial", "name", "all"),
prefix = NULL, recursive = TRUE)
```

```r
upload_blob(container, src, dest = basename(src), type = "BlockBlob",
blocksize = 2^24, lease = NULL, use_azcopy = FALSE)
```

```r
multiupload_blob(container, src, dest, recursive = FALSE,
type = "BlockBlob", blocksize = 2^24, lease = NULL,
use_azcopy = FALSE, max_concurrent_transfers = 10)
```

```r
download_blob(container, src, dest = basename(src), blocksize = 2^24,
overwrite = FALSE, lease = NULL, use_azcopy = FALSE)
```

```r
multidownload_blob(container, src, dest, recursive = FALSE,
blocksize = 2^24, overwrite = FALSE, lease = NULL, use_azcopy = FALSE,
max_concurrent_transfers = 10)
```

```r
delete_blob(container, blob, confirm = TRUE)
```

```r
blob_exists(container, blob)
```

```r
copy_url_to_blob(container, src, dest, lease = NULL, async = FALSE)
```

```r
multicopy_url_to_blob(container, src, dest, lease = NULL, async = FALSE,
max_concurrent_transfers = 10)
```

**Arguments**

- `container` A blob container object.
dir For list_blobs, a string naming the directory. Note that blob storage does not support real directories; this argument simply filters the result to return only blobs whose names start with the given value.

info For list_blobs, level of detail about each blob to return: a vector of names only; the name, size, and whether this blob represents a directory; or all information.

prefix For list_blobs, an alternative way to specify the directory.

recursive This argument is for consistency with the methods for the other storage types. It is not used for blob storage.

src, dest The source and destination files for uploading and downloading. See 'Details' below.

type When uploading, the type of blob to create. Currently only block blobs are supported.

blocksize The number of bytes to upload/download per HTTP(S) request.

lease The lease for a blob, if present.

use_azcopy Whether to use the AzCopy utility from Microsoft to do the transfer, rather than doing it in R.

max_concurrent_transfers For multiupload_blob and multidownload_blob, the maximum number of concurrent file transfers. Each concurrent file transfer requires a separate R process, so limit this if you are low on memory.

overwrite When downloading, whether to overwrite an existing destination file.

blob A string naming a blob.

confirm Whether to ask for confirmation on deleting a blob.

async For copy_url_to_blob and multicopy_url_to_blob, whether the copy operation should be asynchronous (proceed in the background).

Details

upload_blob and download_blob are the workhorse file transfer functions for blobs. They each take as inputs a single filename as the source for uploading/downloading, and a single filename as the destination. Alternatively, for uploading, src can be a textConnection or rawConnection object; and for downloading, dest can be NULL or a rawConnection object. If dest is NULL, the downloaded data is returned as a raw vector, and if a raw connection, it will be placed into the connection. See the examples below.

multiupload_blob and multidownload_blob are functions for uploading and downloading multiple files at once. They parallelise file transfers by using the background process pool provided by AzureRMR, which can lead to significant efficiency gains when transferring many small files. There are two ways to specify the source and destination for these functions:

• Both src and dest can be vectors naming the individual source and destination pathnames.

• The src argument can be a wildcard pattern expanding to one or more files, with dest naming a destination directory. In this case, if recursive is true, the file transfer will replicate the source directory structure at the destination.
upload_blob and download_blob can display a progress bar to track the file transfer. You can control whether to display this with options(azure_storage_progress_bar=TRUE|FALSE); the default is TRUE.

copy_url_to_blob transfers the contents of the file at the specified HTTP[S] URL directly to blob storage, without requiring a temporary local copy to be made. multicopy_url_to_blob does the same, for multiple URLs at once. These functions have a current file size limit of 256MB.

Value

For list_blobs, details on the blobs in the container. For download_blob, if dest=NULL, the contents of the downloaded blob as a raw vector. For blob_exists a flag whether the blob exists.

AzCopy

upload_blob and download_blob have the ability to use the AzCopy commandline utility to transfer files, instead of native R code. This can be useful if you want to take advantage of AzCopy’s logging and recovery features; it may also be faster in the case of transferring a very large number of small files. To enable this, set the use_azcopy argument to TRUE.

Note that AzCopy only supports SAS and AAD (OAuth) token as authentication methods. AzCopy also expects a single filename or wildcard spec as its source/destination argument, not a vector of filenames or a connection.

See Also

blob_container, az_storage, storage_download, call_azcopy

AzCopy version 10 on GitHub

Examples

## Not run:

cont <- blob_container("https://mystorage.blob.core.windows.net/mycontainer", key="access_key")
list_blobs(cont)

upload_blob(cont, "/bigfile.zip", dest="bigfile.zip")
download_blob(cont, "bigfile.zip", dest="/bigfile_downloaded.zip")
delete_blob(cont, "bigfile.zip")

# uploading/downloading multiple files at once
multiupload_blob(cont, "/data/logfiles/*.zip", "/uploaded_data")
multiupload_blob(cont, "myproj/*") # no dest directory uploads to root
multidownload_blob(cont, "jan*.*", "/data/january")

# you can also pass a vector of file/pathnames as the source and destination
src <- c("file1.csv", "file2.csv", "file3.csv")
dest <- paste0("uploaded_", src)
multiupload_blob(cont, src, dest)
# uploading serialized R objects via connections
json <- jsonlite::toJSON(iris, pretty=TRUE, auto_unbox=TRUE)
con <- textConnection(json)
upload_blob(cont, con, "iris.json")

rds <- serialize(iris, NULL)
con <- rawConnection(rds)
upload_blob(cont, con, "iris.rds")

# downloading files into memory: as a raw vector, and via a connection
rawvec <- download_blob(cont, "iris.json", NULL)
rawToChar(rawvec)

con <- rawConnection(raw(0), "r+")
download_blob(cont, "iris.rds", con)
unserialize(con)

# copy from a public URL: Iris data from UCI machine learning repository

## End(Not run)

---

**sign_request**

*Signs a request to the storage REST endpoint with a shared key*

**Description**

Signs a request to the storage REST endpoint with a shared key

**Usage**

```
sign_request(endpoint, ...)
```

**Arguments**

- `endpoint`  
  An endpoint object.

- `...`  
  Further arguments to pass to individual methods.

**Details**

This is a generic method to allow for variations in how the different storage services handle key authorisation. The default method works with blob, file and ADLSgen2 storage.

**Value**

A named list of request headers. One of these should be the Authorization header containing the request signature.
storage_container

Description

Storage client generics

Usage

storage_container(endpoint, ...)

## S3 method for class 'blob_endpoint'
storage_container(endpoint, name, ...)

## S3 method for class 'file_endpoint'
storage_container(endpoint, name, ...)

## S3 method for class 'adls_endpoint'
storage_container(endpoint, name, ...)

## S3 method for class 'character'
storage_container(endpoint, key = NULL, token = NULL, sas = NULL, ...)

create_storage_container(endpoint, ...)

## S3 method for class 'blob_endpoint'
create_storage_container(endpoint, name, ...)

## S3 method for class 'file_endpoint'
create_storage_container(endpoint, name, ...)

## S3 method for class 'adls_endpoint'
create_storage_container(endpoint, name, ...)

## S3 method for class 'storage_container'
create_storage_container(endpoint, ...)

## S3 method for class 'character'
create_storage_container(endpoint, key = NULL, token = NULL, sas = NULL, ...)

delete_storage_container(endpoint, ...)

## S3 method for class 'blob_endpoint'
delete_storage_container(endpoint, name, ...)

## S3 method for class 'file_endpoint'
delete_storage_container(endpoint, name, ...)

## S3 method for class 'storage_container'
delete_storage_container(endpoint, ...)

## S3 method for class 'character'
delete_storage_container(endpoint, key = NULL, token = NULL, sas = NULL, ...)
## S3 method for class 'adls_endpoint'
delete_storage_container(endpoint, name, ...)

## S3 method for class 'storage_container'
delete_storage_container(endpoint, ...)

## S3 method for class 'character'
delete_storage_container(endpoint, key = NULL,
                         token = NULL, sas = NULL, confirm = TRUE, ...)

list_storage_containers(endpoint, ...)

## S3 method for class 'blob_endpoint'
list_storage_containers(endpoint, ...)

## S3 method for class 'file_endpoint'
list_storage_containers(endpoint, ...)

## S3 method for class 'adls_endpoint'
list_storage_containers(endpoint, ...)

## S3 method for class 'character'
list_storage_containers(endpoint, key = NULL, token = NULL, sas = NULL, ...)

list_storage_files(container, ...)

## S3 method for class 'blob_container'
list_storage_files(container, ...)

## S3 method for class 'file_share'
list_storage_files(container, ...)

## S3 method for class 'adls_filesystem'
list_storage_files(container, ...)

create_storage_dir(container, ...)

## S3 method for class 'blob_container'
create_storage_dir(container, ...)

## S3 method for class 'file_share'
create_storage_dir(container, dir, ...)

## S3 method for class 'adls_filesystem'
create_storage_dir(container, dir, ...)

delete_storage_dir(container, ...)
### S3 method for class 'blob_container'

delete_storage_dir(container, ...)

### S3 method for class 'file_share'

delete_storage_dir(container, dir, ...)

### S3 method for class 'adls_filesystem'

delete_storage_dir(container, dir, confirm = TRUE, ...)

delete_storage_file(container, ...)

### S3 method for class 'blob_container'

delete_storage_file(container, file, ...)

### S3 method for class 'file_share'

delete_storage_file(container, file, ...)

### S3 method for class 'adls_filesystem'

delete_storage_file(container, file, confirm = TRUE, ...)

storage_file_exists(container, file, ...)

### S3 method for class 'blob_container'

storage_file_exists(container, file, ...)

### S3 method for class 'file_share'

storage_file_exists(container, file, ...)

### S3 method for class 'adls_filesystem'

storage_file_exists(container, file, ...)

---

**Arguments**

- **endpoint**
  A storage endpoint object, or for the character methods, a string giving the full URL to the container.

- **...**
  Further arguments to pass to lower-level functions.

- **name**
  For the storage container management methods, a container name.

- **key, token, sas**
  For the character methods, authentication credentials for the container: either an access key, an Azure Active Directory (AAD) token, or a SAS. If multiple arguments are supplied, a key takes priority over a token, which takes priority over a SAS.

- **confirm**
  For the deletion methods, whether to ask for confirmation first.

- **container**
  A storage container object.

- **file, dir**
  For the storage object management methods, a file or directory name.
Details

These methods provide a framework for all storage management tasks supported by AzureStor. They dispatch to the appropriate functions for each type of storage.

Storage container management methods:
- `storage_container` dispatches to `blob_container`, `file_share` or `adls_filesystem`
- `create_storage_container` dispatches to `create_blob_container`, `create_file_share` or `create_adls_filesystem`
- `delete_storage_container` dispatches to `delete_blob_container`, `delete_file_share` or `delete_adls_filesystem`
- `list_storage_containers` dispatches to `list_blob_containers`, `list_file_shares` or `list_adls_filesystems`

Storage object management methods:
- `list_storage_files` dispatches to `list_blobs`, `list_azure_files` or `list_adls_files`
- `create_storage_dir` dispatches to `create_azure_dir` or `create_adls_dir`; throws an error if passed a blob container
- `delete_storage_dir` dispatches to `delete_azure_dir` or `delete_adls_dir`; throws an error if passed a blob container
- `delete_storage_file` dispatches to `delete_blob`, `delete_azure_file` or `delete_adls_file`

See Also

`storage_endpoint`, `blob_container`, `file_share`, `adls_filesystem`
`list_blobs`, `list_azure_files`, `list_adls_files`

Similar generics exist for file transfer methods; see the page for `storage_download`.

Examples

```r
## Not run:

# storage endpoints for the one account
bl <- storage_endpoint("https://mystorage.blob.core.windows.net/", key="access_key")
fl <- storage_endpoint("https://mystorage.file.core.windows.net/", key="access_key")

list_storage_containers(bl)
list_storage_containers(fl)

# creating containers
cont <- create_storage_container(bl, "newblobcontainer")
fs <- create_storage_container(fl, "newfileshare")

# creating directories (if possible)
create_storage_dir(cont, "newdir") # will error out
create_storage_dir(fs, "newdir")

# transfer a file
```
storage_endpoint

storage_upload(bl, "~/file.txt", "storage_file.txt")
storage_upload(cont, "~/file.txt", "newdir/storage_file.txt")

## End(Not run)

storage_endpoint Create a storage endpoint object

Description

Create a storage endpoint object, for interacting with blob, file, table, queue or ADLSgen2 storage.

Usage

storage_endpoint(endpoint, key = NULL, token = NULL, sas = NULL, api_version)

blob_endpoint(endpoint, key = NULL, token = NULL, sas = NULL,
             api_version = getOption("azure_storage_api_version"))

file_endpoint(endpoint, key = NULL, token = NULL, sas = NULL,
               api_version = getOption("azure_storage_api_version"))

adls_endpoint(endpoint, key = NULL, token = NULL, sas = NULL,
               api_version = getOption("azure_adls_api_version"))

## S3 method for class 'storage_endpoint'
print(x, ...)

## S3 method for class 'adls_endpoint'
print(x, ...)

Arguments

endpoint The URL (hostname) for the endpoint. This must be of the form http[s]://[account-name].{type}.{core-host-name}, where type is one of "dfs" (corresponding to ADLSgen2), "blob", "file", "queue" or "table". On the public Azure cloud, endpoints will be of the form https://[account-name].{type}.core.windows.net.

key The access key for the storage account.

token An Azure Active Directory (AAD) authentication token. This can be either a string, or an object of class AzureToken created by AzureRMR::get_azure_token. The latter is the recommended way of doing it, as it allows for automatic refreshing of expired tokens.

sas A shared access signature (SAS) for the account.

api_version The storage API version to use when interacting with the host. Defaults to "2018-11-09".

x For the print method, a storage endpoint object.

... For the print method, further arguments passed to lower-level functions.
storage_endpoint

Details

This is the starting point for the client-side storage interface in AzureRMR. storage_endpoint is a generic function to create an endpoint for any type of Azure storage while adls_endpoint, blob_endpoint and file_endpoint create endpoints for those types.

If multiple authentication objects are supplied, they are used in this order of priority: first an access key, then an AAD token, then a SAS. If no authentication objects are supplied, only public (anonymous) access to the endpoint is possible.

Value

storage_endpoint returns an object of S3 class "adls_endpoint", "blob_endpoint", "file_endpoint", "queue_endpoint" or "table_endpoint" depending on the type of endpoint. All of these also inherit from class "storage_endpoint": adls_endpoint, blob_endpoint and file_endpoint return an object of the respective class.

Note that while endpoint classes exist for all storage types, currently AzureStor only includes methods for interacting with ADLSgen2, blob and file storage.

See Also

create_storage_account, adls_filesystem, create_adls_filesystem, file_share, create_file_share, blob_container, create_blob_container

Examples

## Not run:

# obtaining an endpoint from the storage account resource object
stor <- AzureRMR::get_azure_login()$get_subscription("sub_id")$get_resource_group("rgname")$get_storage_account("mystorage")$stor$get_blob_endpoint()

# creating an endpoint standalone
blob_endpoint("https://mystorage.blob.core.windows.net/", key="access_key")

# using an OAuth token for authentication -- note resource is `storage.azure.com`
token <- AzureAuth::get_azure_token("https://storage.azure.com",
    "myaadtenant", "app_id", "password")
adls_endpoint("https://myadlsstorage.dfs.core.windows.net/", token=token)

## End(Not run)
Index

acquire_lease, 2
adls_endpoint, 4, 18
adls_endpoint (storage_endpoint), 39
adls_file_exists (list_adls_files), 25
adls_filesystem, 3, 13, 18, 22, 24, 27, 38, 40
az_resource_group, 6
az_storage, 5, 5, 9, 14–16, 20, 21, 27, 30, 33
azcopy (call_azcopy), 10
azure_file_exists (list_azure_files), 28
AzureRMR::az_resource, 5
AzureRMR::az_resource_group, 14, 16, 20
AzureRMR::az_subscription, 20
AzureRMR::get_azure_token, 4, 9, 39
blob_container, 3, 7, 13, 18, 22, 24, 33, 38, 40
blob_endpoint, 6, 7, 18
blob_endpoint (storage_endpoint), 39
blob_exists (list_blobs), 31
break_lease (acquire_lease), 2
call_azcopy, 10, 13, 27, 30, 33
call_storage_endpoint
  (do_container_op), 17
change_lease (acquire_lease), 2
copy_url_to_blob (list_blobs), 31
copy_url_to_storage, 11
create_adls_dir (list_adls_files), 25
create_adls_filesystem, 40
create_adls_filesystem
  (adls_filesystem), 3
create_azure_dir (list_azure_files), 28
create_blob_container, 40
call_blob_container (blob_container), 7
call_file_share, 40
call_file_share (file_share), 18
create_storage_account, 7, 14, 16, 21, 40
call_create_storage_container
  (storage_container), 35
create_storage_dir (storage_container), 35
Date, 7
delete_adls_dir (list_adls_files), 25
delete_adls_file (list_adls_files), 25
delete_adls_filesystem
  (adls_filesystem), 3
delete_azure_dir (list_azure_files), 28
delete_azure_file (list_azure_files), 28
delete_blob (list_blobs), 31
delete_blob_container (blob_container), 7
delete_file_share (file_share), 18
delete_storage_account, 7, 15, 16, 21
delete_storage_container
  (storage_container), 35
delete_storage_dir (storage_container), 35
delete_storage_file
  (storage_container), 35
do_container_op, 17
download_adls_file, 11, 13
download_adls_file (list_adls_files), 25
download_azure_file, 11, 13
download_azure_file (list_azure_files), 28
download_blob, 11, 13
download_blob (list_blobs), 31
download_from_url
  (copy_url_to_storage), 11
endpoint (storage_endpoint), 39
file_endpoint, 6, 7, 18
file_endpoint (storage_endpoint), 39
file_share, 13, 18, 18, 22, 24, 30, 38, 40
get_adls_file_acl
  (get_storage_properties), 23
get_adls_file_status
        (get_storage_properties), 23
get_storage_account, 7, 15, 16, 20
get_storage_metadata, 21, 24
get_storage_properties, 22, 23
httr::DELETE, 18
httr::GET, 18
httr::HEAD, 18
httr::PATCH, 18
httr::POST, 18
httr::PUT, 18
list_adls_files, 25, 38
list_adls_filesystems
        (adls_filesystem), 3
list_azure_files, 28, 38
list_blob_containers (blob_container), 7
list_blobs, 31, 38
list_file_shares (file_share), 18
list_storage_accounts
        (get_storage_account), 20
list_storage_containers
        (storage_container), 35
list_storage_files (storage_container), 35
multicopy_url_to_blob (list_blobs), 31
multicopy_url_to_storage (copy_url_to_storage), 11
multidownload_adls_file
        (list_adls_files), 25
multidownload_azure_file
        (list_azure_files), 28
multidownload_blob (list_blobs), 31
multiupload_adls_file
        (list_adls_files), 25
multiupload_azure_file
        (list_azure_files), 28
multiupload_blob (list_blobs), 31
POSIXt, 7
print.adls_endpoint (storage_endpoint), 39
print.adls_filesystem
        (adls_filesystem), 3
print.blob_container (blob_container), 7
print.file_share (file_share), 18
print.storage_endpoint
        (storage_endpoint), 39
processx::run, 11
queue_endpoint (storage_endpoint), 39
rawConnection, 26, 29, 32
release_lease (acquire_lease), 2
renew_lease (acquire_lease), 2
set_storage_metadata
        (get_storage_metadata), 21
sign_request, 34
storage_container, 5, 9, 13, 20, 35
storage_download, 27, 30, 33, 38
storage_download (copy_url_to_storage), 11
storage_endpoint, 4, 5, 8, 9, 19, 20, 38, 39
storage_file_exists
        (storage_container), 35
storage_generics (storage_container), 35
storage_multidownload
        (copy_url_to_storage), 11
storage_multiupload
        (copy_url_to_storage), 11
storage_upload (copy_url_to_storage), 11
table_endpoint (storage_endpoint), 39
textConnection, 26, 29, 32
upload_adls_file (list_adls_files), 25
upload_azure_file (list_azure_files), 28
upload_blob (list_blobs), 31
upload_to_url (copy_url_to_storage), 11