Package ‘AzureStor’

May 25, 2022

Title Storage Management in 'Azure'

Version 3.7.0

Description Manage storage in Microsoft's 'Azure' cloud: <https://azure.microsoft.com/en-us/product-categories/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 (>= 1.4.0), mime, openssl, xml2, vctrs, AzureRMR (>= 2.3.0)

Suggests AzureAuth, readr, knitr, rmarkdown, jsonlite, testthat, processx, uuid

RoxygenNote 7.1.1

NeedsCompilation no

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

Maintainer Hong Ooi <hongooi73@gmail.com>

Repository CRAN

Date/Publication 2022-05-25 07:00:02 UTC
R topics documented:

- acquire_lease ........................................... 2
- adls_filesystem ........................................ 3
- az_storage .............................................. 6
- blob_container ......................................... 8
- call_azcopy ............................................ 11
- copy_url_to_storage ................................. 12
- create_blob_snapshot ............................... 15
- create_storage_account ......................... 16
- delete_storage_account ...................... 18
- do_container_op ...................................... 19
- file_share ........................................... 21
- get_account_sas ..................................... 23
- get_storage_account ................................ 27
- get_storage_metadata ............................. 28
- get_storage_properties ......................... 30
- list_adls_files ..................................... 32
- list_azure_files ................................... 35
- list_blobs ........................................... 38
- list_blob_versions ................................. 43
- sign_request ........................................... 44
- storage_container ................................ 44
- storage_endpoint .................................... 49
- storage_save_rds .................................. 52
- storage_write_delim ......................... 53

Index .......................... 55

---

acquire_lease  

*Operations on blob leases*

**Description**

Manage leases for blobs and blob containers.

**Usage**

```r
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 `break_lease`, the period for which to break the lease.
- **new_lease**: For `change_lease`, 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 `acquire_lease` and `change_lease`, 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_storage_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_storage_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_storage_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 `storage_endpoint` or `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 inter-
  acting with the host. Currently defaults to "2019-07-07".

name    The name of the filesystem to get, create, or delete.
For the print method, a filesystem object.

confirm 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 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 'Creating a shared access signature' below.
- get_user_delegation_key(...): Returns a key that can be used to construct a user delegation SAS.
- get_user_delegation_sas(...): Return a user delegation SAS.
- revoke_user_delegation_keys(): Revokes all user delegation keys for the account. This also renders all SAS’s obtained via such keys invalid.
- 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 an 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.

Creating a shared access signature

Note that you don’t need to worry about this section if you have been given a SAS, and only want to use it to access storage.

AzureStor supports generating three kinds of SAS: account, service and user delegation. An account SAS can be used with any type of storage. A service SAS can be used with blob and file storage, while a user delegation SAS can be used with blob and ADLS2 storage.

To create an account SAS, call the get_account_sas() method. This has the following signature:

get_account_sas(key=self$list_keys()[1], start=NULL, expiry=NULL, services="bqtf", permissions="rl", resource_types="sco", ip=NULL, protocol=NULL)
To create a service SAS, call the `get_service_sas()` method, which has the following signature:

```r
get_service_sas(key=self$list_keys()[1], resource, service, start=NULL, expiry=NULL, permissions="r", resource_type=NULL, ip=NULL, protocol=NULL, policy=NULL, snapshot_time=NULL)
```

To create a user delegation SAS, you must first create a user delegation key. This takes the place of the account’s access key in generating the SAS. The `get_user_delegation_key()` method has the following signature:

```r
get_user_delegation_key(token=self$token, key_start=NULL, key_expiry=NULL)
```

Once you have a user delegation key, you can use it to obtain a user delegation sas. The `get_user_delegation_sas()` method has the following signature:

```r
get_user_delegation_sas(key, resource, start=NULL, expiry=NULL, permissions="rl", resource_type="c", ip=NULL, protocol=NULL, snapshot_time=NULL)
```

(Note that the key argument for this method is the user delegation key, not the account key.)

To invalidate all user delegation keys, as well as the SAS’s generated with them, call the `revoke_user_delegation_keys()` method. This has the following signature:

```r
revoke_user_delegation_keys()
```

See the Shared access signatures page for more information about this topic.

### 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
- Create an account SAS, Create a user delegation SAS, Create a service SAS
Examples

```r
## 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)

# 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**

```r
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"), ...)

# 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"), ...)

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

## S3 method for class 'blob_endpoint'
create_blob_container(endpoint, name,
                       public_access = c("none", "blob", "container"), ...)
delete_blob_container(endpoint, ...)

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

## S3 method for class '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 "2019-07-07".
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/",
  tenant="myaadtenant",
  app="myappid",
  password="mypassword")
blob_container("https://mystorage.blob.core.windows.net/mycontainer", token=token)

## End(Not run)
```
Description

Call the azcopy file transfer utility

Usage

call_azcopy(..., env = NULL,
    silent = getOption("azure_storage_azcopy_silent", FALSE))

Arguments

... Arguments to pass to AzCopy on the commandline. If no arguments are supplied, 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. Defaults to the value of the azure_storage_azcopy_silent option, or FALSE if this is unset.

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.
See Also

processx::run, download_blob, download_azure_file, download_adls_file

AzCopy page on Microsoft Docs
AzCopy GitHub repo

Examples

```r
## Not run:

dp <- storage_endpoint("https://mystorage.blob.core.windows.net", sas="mysas")
cnt <- storage_container(dp, "mycontainer")

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

# calling azcopy to download a blob
storage_download(cnt, "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'
copy_url_to_storage(container, src, dest, ...)

## S3 method for class 'blob_container'
multicopy_url_to_storage(container, src, dest, ...)

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_blob_snapshot**  
*Create, list and delete blob snapshots*

Description

Create, list and delete blob snapshots

Usage

create_blob_snapshot(container, blob, ...)

list_blob_snapshots(container, blob)

delete_blob_snapshot(container, blob, snapshot, confirm = TRUE)

Arguments

- **container**: A blob container.
- **blob**: The path/name of a blob.
- **...**: For create_blob_snapshot, an optional list of name-value pairs that will be treated as the metadata for the snapshot. If no metadata is supplied, the metadata for the base blob is copied to the snapshot.
- **snapshot**: For delete_blob_snapshot, the specific snapshot to delete. This should be a datetime string, in the format yyyy-mm-ddTHH:MM:SS.SSSSSSSZ. To delete all snapshots for the blob, set this to "all".
- **confirm**: Whether to ask for confirmation on deleting a blob’s snapshots.

Details

Blobs can have snapshots associated with them, which are the contents and optional metadata for the blob at a given point in time. A snapshot is identified by the date and time on which it was created.
create_blob_snapshot creates a new snapshot, list_blob_snapshots lists all the snapshots, and delete_blob_snapshot deletes a given snapshot or all snapshots for a blob.

Note that snapshots are only supported if the storage account does NOT have hierarchical namespaces enabled.

Value

For create_blob_snapshot, the datetime string that identifies the snapshot.

For list_blob_snapshots a vector of such strings, or NULL if the blob has no snapshots.

See Also

Other AzureStor functions that support blob snapshots by passing a snapshot argument: download_blob, get_storage_properties, get_storage_metadata

Examples

```r
## Not run:

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

snap_id <- create_blob_snapshot(cnt, "myfile", tag1="value1", tag2="value2")

list_blob_snapshots(cnt, "myfile")

get_storage_properties(cnt, "myfile", snapshot=snap_id)

# returns list(tag1="value1", tag2="value2")

get_storage_metadata(cnt, "myfile", snapshot=snap_id)

download_blob(cnt, "myfile", snapshot=snap_id)

# delete all snapshots

delete_blob_snapshots(cnt, "myfile", snapshot="all")

## End(Not run)
```

---

**create_storage_account**

*Create Azure storage account*

**Description**

Method for the AzureRMR::az_resource_group class.
create_storage_account

Usage

create_storage_account(name = "StorageV2", location = "Standard_LRS", replication = "hot", https_only = TRUE, hierarchical_namespace_enabled = TRUE, 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 a 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.

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. AzureStor provides an R interface to ADLSgen2, blob and file storage, while the Azure-Qstor and AzureTableStor packages provide interfaces to queue and table storage respectively.

Value

An object of class az_storage representing the created storage account.

See Also

gt_storage_account, delete_storage_account, az_storage

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

## Not run:

rg <- AzureRMR::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

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**

```r
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 returns 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
## 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

file_share(endpoint, ...)

## S3 method for class 'character'
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, ...)
## 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 "2019-07-07".
- **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:
endp <- file_endpoint("https://mystorage.file.core.windows.net/", key="access_key")

# list file shares
list_file_shares(endp)

# get, create, and delete a file share
file_share(endp, "myshare")
create_file_share(endp, "newshare")
delete_file_share(endp, "newshare")
```
## Generate 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. There are three kinds of SAS: account, service and user delegation.

### Usage

```r
get_account_sas(account, ...)  ## S3 method for class 'az_storage'
get_account_sas(account, key = account$list_keys()[1], ...)  

## S3 method for class 'storage_endpoint'
get_account_sas(account, key = account$key, ...)

## Default S3 method:
get_account_sas(account, key, start = NULL, expiry = NULL, services = "bqtf", permissions = "rl", resource_types = "sco", ip = NULL, protocol = NULL, auth_api_version = getOption("azure_storage_api_version"), ...)

get_user_delegation_key(account, ...)  ## S3 method for class 'az_resource'
get_user_delegation_key(account, token = account$token, ...)

## S3 method for class 'blob_endpoint'
get_user_delegation_key(account, token = account$token, key_start = NULL, key_expiry = NULL, ...)

revoke_user_delegation_keys(account)  ## S3 method for class 'az_storage'
revoke_user_delegation_keys(account)
```
get_user_delegation_sas(account, ...)  

## S3 method for class 'az_storage'  
get_user_delegation_sas(account, key, ...)  

## S3 method for class 'blob_endpoint'  
get_user_delegation_sas(account, key, ...)  

## Default S3 method:  
get_user_delegation_sas(account, key, resource,  
start = NULL, expiry = NULL, permissions = "rl", resource_type = "c",  
ip = NULL, protocol = NULL, snapshot_time = NULL,  
directory_depth = NULL,  
auth_api_version = getOption("azure_storage_api_version"), ...)  

get_service_sas(account, ...)  

## S3 method for class 'az_storage'  
get_service_sas(account, resource, service = c("blob",  
"file"), key = account$list_keys()[1], ...)  

## S3 method for class 'storage_endpoint'  
get_service_sas(account, resource, key = account$key, ...)  

## Default S3 method:  
get_service_sas(account, resource, key, service,  
start = NULL, expiry = NULL, permissions = "rl",  
resource_type = NULL, ip = NULL, protocol = NULL, policy = NULL,  
n snapshot_time = NULL, directory_depth = NULL,  
auth_api_version = getOption("azure_storage_api_version"), ...)  

Arguments  

account An object representing a storage account. Depending on the generic, this can  
be one of the following: an Azure resource object (of class az_storage); a  
client storage endpoint (of class storage_endpoint); a blob storage endpoint  
(of class blob_endpoint); or a string with the name of the account.  

... Arguments passed to lower-level functions.  

key For get_account_sas, the account key, which controls full access to the storage  
account. For get_user_delegation_sas, a user delegation key, as obtained  
from get_user_delegation_key.  

start, expiry The start and end dates for the account or user delegation SAS. These should be  
Date or POSIXct values, or strings coercible to such. If not supplied, the default  
is to generate start and expiry values for a period of 8 hours, starting from 15  
minutes before the current time.  

services For get_account_sas, the storage service(s) for which the SAS is valid. De-  
defaults to qtbf, meaning blob (including ADLS2), queue, table and file storage.
The permissions that the SAS grants. The default value of rl (read and list) essentially means read-only access.

For an account SAS, the resource types for which the SAS is valid. For get_account_sas the default is sco meaning service, container and object. For get_user_delegation_sas the default is c meaning container-level access (including blobs within the container). Other possible values include "b" (a single blob) or "d" (a directory).

The IP address(es) or IP address range(s) for which the SAS is valid. The default is not to restrict access by IP.

The protocol required to use the SAS. Possible values are https meaning HTTPS-only, or https, http meaning HTTP is also allowed. Note that the storage account itself may require HTTPS, regardless of what the SAS allows.

The storage API version to use for authenticating.

For get_user_delegation_key, an AAD token from which to obtain user details. The token must have https://storage.azure.com as its audience.

For get_user_delegation_key, the start and end dates for the user delegation key.

For get_user_delegation_sas and get_service_sas, the resource for which the SAS is valid. Both types of SAS allow this to be either a blob container, a directory or an individual blob; the resource should be specified in the form containername[/dirname[/blobname]]. A service SAS can also be used with file shares and files, in which case the resource should be of the form sharename[/path-to-filename].

For a service or user delegation SAS, the type of resource for which the SAS is valid. For blob storage, the default value is "b" meaning a single blob. For file storage, the default value is "f" meaning a single file. Other possible values include "bs" (a blob snapshot), "c" (a blob container), "d" (a directory in a blob container), or "s" (a file share). Note however that a user delegation SAS only supports blob storage.

For a user delegation or service SAS, the blob snapshot for which the SAS is valid. Only required if resource_type[s]="bs".

For a service SAS, the depth of the directory, starting at 0 for the root. This is required if resource_type="d" and the account has a hierarchical namespace enabled.

For a service SAS, the storage service for which the SAS is valid: either "blob" or "file". Currently AzureStor does not support creating a service SAS for queue or table storage.

For a service SAS, optionally the name of a stored access policy to correlate the SAS with. Revoking the policy will also invalidate the SAS.

Listed here are S3 generics and methods to obtain a SAS for accessing storage; in addition, the az_storage resource class has R6 methods for get_account_sas, get_service_sas, get_user_delegation_key and revoke_user_delegation_keys which simply call the corresponding S3 method.
Note that you don’t need to worry about these methods if you have been given a SAS, and only want to use it to access a storage account.

An account SAS is secured with the storage account key. An account SAS delegates access to resources in one or more of the storage services. All of the operations available via a user delegation SAS are also available via an account SAS. You can also delegate access to read, write, and delete operations on blob containers, tables, queues, and file shares. To obtain an account SAS, call `get_account_sas`.

A service SAS is like an account SAS, but allows finer-grained control of access. You can create a service SAS that allows access only to specific blobs in a container, or files in a file share. To obtain a service SAS, call `get_service_sas`.

A user delegation SAS is a SAS secured with Azure AD credentials. It’s recommended that you use Azure AD credentials when possible as a security best practice, rather than using the account key, which can be more easily compromised. When your application design requires shared access signatures, use Azure AD credentials to create a user delegation SAS for superior security.

Every SAS is signed with a key. To create a user delegation SAS, you must first request a user delegation key, which is then used to sign the SAS. The user delegation key is analogous to the account key used to sign a service SAS or an account SAS, except that it relies on your Azure AD credentials. To request the user delegation key, call `get_user_delegation_key`. With the user delegation key, you can then create the SAS with `get_user_delegation_sas`.

To invalidate all user delegation keys, as well as the SAS’s generated with them, call `revoke_user_delegation_keys`.

See the examples and Microsoft Docs pages below for how to specify arguments like the services, permissions, and resource types. Also, while not explicitly mentioned in the documentation, ADLS-gen2 storage can use any SAS that is valid for blob storage.

See Also

- `blob_endpoint`, `file_endpoint`, `Date`, `POSIXt`
- Azure Storage Provider API reference, Azure Storage Services API reference
- Create an account SAS, Create a user delegation SAS, Create a service SAS

Examples

# account SAS valid for 7 days
get_account_sas("mystorage", "access_key", start=Sys.Date(), expiry=Sys.Date() + 7)

# SAS with read/write/create/delete permissions
get_account_sas("mystorage", "access_key", permissions="rwcd")

# SAS limited to blob (+ADLS2) and file storage
get_account_sas("mystorage", "access_key", services="bf")

# SAS for file storage, allows access to files only (not shares)
get_account_sas("mystorage", "access_key", services="f", resource_types="o")

# getting the key from an endpoint object
endp <- storage_endpoint("https://mystorage.blob.core.windows.net", key="access_key")
get_account_sas(endp, permissions="rwcd")
get_storage_accounts

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.
get_storage_metadata

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

Usage

get_storage_metadata(object, ...)

## S3 method for class 'blob_container'
get_storage_metadata(object, blob, snapshot = NULL, version = NULL, ...)

## 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)
get_storage_metadata

## 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.
- **snapshot, version**: For the blob method of `get_storage_metadata`, optional snapshot and version identifiers. These should be datetime strings, in the format "yyyy-mm-ddTHH:MM:SS.SSSSSS". Ignored if `blob` is omitted.
- **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

### 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")

set_storage_metadata(fs, "newdir/iris.csv", name1="value1")
# will be list(name1="value1")
get_storage_metadata(fs, "newdir/iris.csv")
```
```r
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'
get_storage_properties(object, blob, snapshot = NULL, version = NULL, ...)

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

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

```
get_adls_file_acl(filesystem, file)
get_adls_file_status(filesystem, file)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>A blob container, file share, or ADLS filesystem object.</td>
</tr>
<tr>
<td>...</td>
<td>For compatibility with the generic.</td>
</tr>
<tr>
<td>blob, file</td>
<td>Optionally the name of an individual blob, file or directory within a container.</td>
</tr>
</tbody>
</table>
**get_storage_properties**

**snapshot, version**
For the blob method of `get_storage_properties`, optional snapshot and version identifiers. These should be datetime strings, in the format "yyyy-mm-ddTHH:MM:SS.SSSSSSZ". Ignored if blob is omitted.

**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)
- `list_blob_snapshots` to obtain the snapshots for a blob

**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**

```r
list_adls_files(filesystem, dir = "/", info = c("all", "name"), recursive = FALSE)
```

```r
multiupload_adls_file(filesystem, src, dest, recursive = FALSE, blocksize = 2^22, lease = NULL, put_md5 = FALSE, use_azcopy = FALSE, max_concurrent_transfers = 10)
```

```r
upload_adls_file(filesystem, src, dest = basename(src), blocksize = 2^24, lease = NULL, put_md5 = FALSE, use_azcopy = FALSE)
```

```r
multidownload_adls_file(filesystem, src, dest, recursive = FALSE, blocksize = 2^24, overwrite = FALSE, check_md5 = FALSE, use_azcopy = FALSE, max_concurrent_transfers = 10)
```

```r
download_adls_file(filesystem, src, dest = basename(src), blocksize = 2^24, overwrite = FALSE, check_md5 = FALSE, use_azcopy = FALSE)
```

```r
delete_adls_file(filesystem, file, confirm = TRUE)
```

```r
create_adls_dir(filesystem, dir)
```

```r
delete_adls_dir(filesystem, dir, recursive = FALSE, confirm = TRUE)
```

```r
adls_file_exists(filesystem, file)
```

```r
adls_dir_exists(filesystem, dir)
```

**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.
The source and destination paths/files for uploading and downloading. See 'Details' below.

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

The lease for a file, if present.

For uploading, whether to compute the MD5 hash of the file(s). This will be stored as part of the file’s properties.

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

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.

When downloading, whether to overwrite an existing destination file.

For downloading, whether to verify the MD5 hash of the downloaded file(s). This requires that the file’s Content-MD5 property is set. If this is TRUE and the Content-MD5 property is missing, a warning is generated.

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.

adls_file_exists and adls_dir_exists test for the existence of a file and directory, respectively.

**AzCopy:**

upload_azure_file and download_azure_file have the ability to use the AzCopy command-line 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.

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.

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
multupload_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)
multupload_adls_file(fs, src, dest, 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
list_azure_files

```r
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**

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

```r
upload_azure_file(share, src, dest = basename(src), create_dir = FALSE,
blocksize = 2^22, put_md5 = FALSE, use_azcopy = FALSE)
```

```r
multiupload_azure_file(share, src, dest, recursive = FALSE,
create_dir = recursive, blocksize = 2^22, put_md5 = FALSE,
use_azcopy = FALSE, max_concurrent_transfers = 10)
```

```r
download_azure_file(share, src, dest = basename(src), blocksize = 2^22,
overwrite = FALSE, check_md5 = FALSE, use_azcopy = FALSE)
```

```r
multidownload_azure_file(share, src, dest, recursive = FALSE,
blocksize = 2^22, overwrite = FALSE, check_md5 = FALSE,
use_azcopy = FALSE, max_concurrent_transfers = 10)
```

```r
delete_azure_file(share, file, confirm = TRUE)
```

```r
create_azure_dir(share, dir, recursive = FALSE)
```

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

```r
azure_file_exists(share, file)
```

```r
azure_dir_exists(share, dir)
```
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`, whether to recursively create each component of a nested directory path. For `delete_azure_dir`, whether to delete a subdirectory’s contents first. 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.
put_md5  For uploading, whether to compute the MD5 hash of the file(s). This will be stored as part of the file’s properties.
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.
check_md5  For downloading, whether to verify the MD5 hash of the downloaded file(s). This requires that the file’s Content-MD5 property is set. If this is TRUE and the Content-MD5 property is missing, a warning is generated.
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.
list_azure_files

- 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.

azure_file_exists and azure_dir_exists test for the existence of a file and directory, respectively.

AzCopy:
upload_azure_file and download_azure_file have the ability to use the AzCopy command-line 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.

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.

See Also
file_share, az_storage, storage_download, call_azcopy
AzCopy version 10 on GitHub

Examples

```r
## 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
multiupload_azure_file(share, "/data/logfiles/*.zip")
```
# 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; create or delete directories; 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 = c("BlockBlob",
            "AppendBlob"), blocksize = if (type == "BlockBlob") 2^24 else 2^22,
            lease = NULL, put_md5 = FALSE, append = FALSE, use_azcopy = FALSE)
```

```r
multiupload_blob(container, src, dest, recursive = FALSE,
               type = c("BlockBlob", "AppendBlob"), blocksize = if (type == "BlockBlob")
               2^24 else 2^22, lease = NULL, put_md5 = FALSE, append = FALSE,
               use_azcopy = FALSE, max_concurrent_transfers = 10)
```
download_blob(container, src, dest = basename(src), blocksize = 2^24, overwrite = FALSE, lease = NULL, check_md5 = FALSE, use_azcopy = FALSE, snapshot = NULL, version = NULL)

multidownload_blob(container, src, dest, recursive = FALSE, blocksize = 2^24, overwrite = FALSE, lease = NULL, check_md5 = FALSE, use_azcopy = FALSE, max_concurrent_transfers = 10)

delete_blob(container, blob, confirm = TRUE)

create_blob_dir(container, dir)

delete_blob_dir(container, dir, recursive = FALSE, confirm = TRUE)

blob_exists(container, blob)

blob_dir_exists(container, dir)

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

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

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, blob type, and whether this blob represents a directory; or all information.
prefix For list_blobs, an alternative way to specify the directory.
recursive For the multiupload/download functions, whether to recursively transfer files in subdirectories. For list_blobs, whether to include the contents of any subdirectories in the listing. For delete_blob_dir, whether to recursively delete subdirectory contents as well.
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 and append blobs are supported.
blocksize The number of bytes to upload/download per HTTP(S) request.
lease The lease for a blob, if present.
put_md5 For uploading, whether to compute the MD5 hash of the blob(s). This will be stored as part of the blob’s properties. Only used for block blobs.
append

When uploading, whether to append the uploaded data to the destination blob. Only has an effect if \texttt{type="AppendBlob"}. If this is \texttt{FALSE} (the default) and the destination append blob exists, it is overwritten. If this is \texttt{TRUE} and the destination does not exist or is not an append blob, an error is thrown.

use_azcopy

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

max_concurrent_transfers

For \texttt{multiupload_blob} and \texttt{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.

check_md5

For downloading, whether to verify the MD5 hash of the downloaded blob(s). This requires that the blob’s \texttt{Content-MD5} property is set. If this is \texttt{TRUE} and the \texttt{Content-MD5} property is missing, a warning is generated.

snapshot, version

For \texttt{download_blob}, optional snapshot and version identifiers. These should be datetime strings, in the format "yyyy-mm-ddTHH:MM:SS.SSSSSSSSZ". If omitted, download the base blob.

blob

A string naming a blob.

confirm

Whether to ask for confirmation on deleting a blob.

async

For \texttt{copy_url_to_blob} and \texttt{multicopy_url_to_blob}, whether the copy operation should be asynchronous (proceed in the background).

auth_header

For \texttt{copy_url_to_blob} and \texttt{multicopy_url_to_blob}, an optional Authorization HTTP header to send to the source. This allows copying files that are not publicly available or otherwise have access restrictions.

Details

\texttt{upload_blob} and \texttt{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, \texttt{src} can be a \texttt{textConnection} or \texttt{rawConnection} object; and for downloading, \texttt{dest} can be \texttt{NULL} or a \texttt{rawConnection} object. If \texttt{dest} is \texttt{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.

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

\texttt{upload_blob} and \texttt{download_blob} can display a progress bar to track the file transfer. You can control whether to display this with \texttt{options(azure_storage_progress_bar=TRUE|FALSE)}; the default is \texttt{TRUE}. 

multiupload_blob can upload files either as all block blobs or all append blobs, but not a mix of both.

blob_exists and blob_dir_exists test for the existence of a blob and directory, respectively. delete_blob deletes a blob, and delete_blob_dir deletes all blobs in a directory (possibly recursively). This will also delete any snapshots for the blob(s) involved.

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.

The following points should be noted about AzCopy:

- It 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.
- Currently, it does not support appending data to existing blobs.

Directories:
Blob storage does not have true directories, instead using filenames containing a separator character (typically '/') to mimic a directory structure. This has some consequences:

- The isdir column in the data frame output of list_blobs is a best guess as to whether an object represents a file or directory, and may not always be correct. Currently, list_blobs assumes that any object with a file size of zero is a directory.
- Zero-length files can cause problems for the blob storage service as a whole (not just AzureStor). Try to avoid uploading such files.
- create_blob_dir and delete_blob_dir are guaranteed to function as expected only for accounts with hierarchical namespaces enabled. When this feature is disabled, directories do not exist as objects in their own right: to create a directory, simply upload a blob to that directory. To delete a directory, delete all the blobs within it; as far as the blob storage service is concerned, the directory then no longer exists.
- Similarly, the output of list_blobs(recursive=TRUE) can vary based on whether the storage account has hierarchical namespaces enabled.
- blob_exists will return FALSE for a directory when the storage account does not have hierarchical namespaces enabled.

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.

See Also
blob_container, az_storage, storage_download, call_azcopy, list_blob_snapshots, list_blob_versions
AzCopy version 10 on GitHub Guide to the different blob types
Examples

## Not run:

```r
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")

# append blob: concatenating multiple files into one
upload_blob(cont, "logfile1", "logfile", type="AppendBlob", append=FALSE)
upload_blob(cont, "logfile2", "logfile", type="AppendBlob", append=TRUE)
upload_blob(cont, "logfile3", "logfile", type="AppendBlob", append=TRUE)

# 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
copy_url_to_blob(cont,
  "iris.csv")

## End(Not run)
list_blob_versions  List and delete blob versions

Description
List and delete blob versions

Usage
list_blob_versions(container, blob)
delete_blob_version(container, blob, version, confirm = TRUE)

Arguments
- container: A blob container.
- blob: The path/name of a blob.
- version: For delete_blob_version, the specific version to delete. This should be a datetime string, in the format yyyy-mm-ddTHH:MM:SS.SSSSSSZ.
- confirm: Whether to ask for confirmation on deleting a blob version.

Details
A version captures the state of a blob at a given point in time. Each version is identified with a version ID. When blob versioning is enabled for a storage account, Azure Storage automatically creates a new version with a unique ID when a blob is first created and each time that the blob is subsequently modified.

A version ID can identify the current version or a previous version. A blob can have only one current version at a time.

When you create a new blob, a single version exists, and that version is the current version. When you modify an existing blob, the current version becomes a previous version. A new version is created to capture the updated state, and that new version is the current version. When you delete a blob, the current version of the blob becomes a previous version, and there is no longer a current version. Any previous versions of the blob persist.

Versions are different to snapshots:
- A new snapshot has to be explicitly created via create_blob_snapshot. A new blob version is automatically created whenever the base blob is modified (and hence there is no create_blob_version function).
- Deleting the base blob will also delete all snapshots for that blob, while blob versions will be retained (but will typically be inaccessible).
- Snapshots are only available for storage accounts with hierarchical namespaces disabled, while versioning can be used with any storage account.

Value
For list_blob_versions, a vector of datetime strings which are the IDs of each version.
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**

```r
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  
*Storage client generics*

**Description**

Storage client generics

**Usage**

```r
storage_container(endpoint, ...)
```

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

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

```r
## 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 '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, dir, ...)

## 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, dir, ...)

## 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, ...)

storage_dir_exists(container, dir, ...)

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

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

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

create_storage_snapshot(container, file, ...)

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

list_storage_snapshots(container, ...)

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

delete_storage_snapshot(container, file, ...)

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

list_storage_versions(container, ...)

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

delete_storage_version(container, file, ...)

## S3 method for class 'blob_container'
delete_storage_version(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_blob_dir`, `create_azure_dir` or `create_adls_dir`
- `delete_storage_dir` dispatches to `delete_blob_dir`, `delete_azure_dir` or `delete_adls_dir`
- `delete_storage_file` dispatches to `delete_blob`, `delete_azure_file` or `delete_adls_file`
- `storage_file_exists` dispatches to `blob_exists`, `azure_file_exists` or `adls_file_exists`
- `storage_dir_exists` dispatches to `blob_dir_exists`, `azure_dir_exists` or `adls_dir_exists`
- `create_storage_snapshot` dispatches to `create_blob_snapshot`
- `list_storage_snapshots` dispatches to `list_blob_snapshots`
- `delete_storage_snapshot` dispatches to `delete_blob_snapshot`
- `list_storage_versions` dispatches to `list_blob_versions`
- `delete_storage_version` dispatches to `delete_blob_version`
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, service)
```

```
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"))
```

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

## End(Not run)
```
storage_endpoint

```r
storage_endpoint(endpoint, key = NULL, token = NULL, sas = NULL,
                  api_version = getOption("azure_storage_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 "2019-07-07".

- **service**
  - For `storage_endpoint`, the service endpoint type: either "blob", "file", "adls", "queue" or "table". If this is missing, it is inferred from the endpoint hostname.

- **x**
  - For the print method, a storage endpoint object.

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

### 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.
Storage emulators

AzureStor supports connecting to the Azure SDK and Azurite emulators for blob and queue storage. To connect, pass the full URL of the endpoint, including the account name, to the `blob_endpoint` and `queue_endpoint` methods (the latter from the AzureQstor package). The warning about an unrecognised endpoint can be ignored. See the linked pages, and the examples below, for details on how to authenticate with the emulator.

Note that the Azure SDK emulator is no longer being actively developed; it’s recommended to use Azurite for development work.

See Also

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

Examples

```r
## 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)

## Azurite storage emulator:

# connecting to Azurite with the default account and key (these also work for the Azure SDK)
azurite_account <- "devstoreaccount1"
azurite_key <-
  "Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTQr/KBHBeksoGMDw=="
blob_endpoint(paste0("http://127.0.0.1:10000/", azurite_account), key=azurite_key)

# to use a custom account name and key, set the AZURITE_ACCOUNTS env var before starting Azurite
Sys.setenv(AZURITE_ACCOUNTS="account1:key1")
blob_endpoint("http://127.0.0.1:10000/account1", key="key1")

## End(Not run)"
Storage Save RDS

Save and load R objects to/from a storage account

Description

Save and load R objects to/from a storage account

Usage

storage_save_rds(object, container, file, ...)

storage_load_rds(container, file, ...)

storage_save_rdata(..., container, file, envir = parent.frame())

storage_load_rdata(container, file, envir = parent.frame(), ...)

Arguments

object An R object to save to storage.
container An Azure storage container object.
file The name of a file in storage.
... Further arguments passed to saveRDS, memDecompress, save and load as appropriate.
envir For storage_save_rdata and storage_load_rdata, the environment from which to get objects to save, or in which to restore objects, respectively.

Details

These are equivalents to saveRDS, readRDS, save and load for saving and loading R objects to a storage account. They allow datasets and objects to be easily transferred to and from an R session, without having to manually create and delete temporary files.

See Also

storage_download, download_blob, download_azure_file, download_adls_file, save, load, saveRDS

Examples

## Not run:

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

storage_save_rds(iris, cont, "iris.rds")
irisnew <- storage_load_rds(iris, "iris.rds")
identical(iris, irisnew) # TRUE
storage_write_delim

storage_save_rdata(iris, mtcars, container=cont, file="dataframes.rdata")
storage_load_rdata(cont, "dataframes.rdata")

## End(Not run)

---

**storage_write_delim**  
*Read and write a data frame to/from a storage account*

---

**Description**

Read and write a data frame to/from a storage account

**Usage**

`storage_write_delim(object, container, file, delim = "\t", ...)`

`storage_write_csv(object, container, file, ...)`

`storage_write_csv2(object, container, file, ...)`

`storage_read_delim(container, file, delim = "\t", ...)`

`storage_read_csv(container, file, ...)`

`storage_read_csv2(container, file, ...)`

**Arguments**

- **object**: A data frame to write to storage.
- **container**: An Azure storage container object.
- **file**: The name of a file in storage.
- **delim**: For `storage_write_delim` and `storage_read_delim`, the field delimiter. Defaults to `\t` (tab).
- **...**: Optional arguments passed to the file reading/writing functions. See 'Details'.

**Details**

These functions let you read and write data frames to storage. `storage_read_delim` and `write_delim` are for reading and writing arbitrary delimited files. `storage_read_csv` and `write_csv` are for comma-delimited (CSV) files. `storage_read_csv2` and `write_csv2` are for files with the semicolon `;` as delimiter and comma `,` as the decimal point, as used in some European countries.

If the readr package is installed, they call down to `read_delim`, `write_delim`, `read_csv2` and `write_csv2`. Otherwise, they use `read_delim` and `write.table`. 
See Also

`storage_download, download_blob, download_azure_file, download_adls_file, write.table, read.csv, readr::write_delim, readr::read_delim`

Examples

```r
## Not run:

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

storage_write_csv(iris, cont, "iris.csv")
# if readr is not installed
irisnew <- storage_read_csv(cont, "iris.csv", stringsAsFactors=TRUE)
# if readr is installed
irisnew <- storage_read_csv(cont, "iris.csv", col_types="nnnnf")

all(mapply(identical, iris, irisnew)) # TRUE

## End(Not run)
```
acquire_lease, 2
adls_dir_exists (list_adls_files), 32
adls_endpoint, 4, 20
adls_endpoint (storage_endpoint), 49
adls_file_exists (list_adls_files), 32
adls_filesystem, 3, 14, 20, 29, 31, 34, 49, 51
az_resource_group, 6
az_storage, 5, 6, 10, 17, 18, 22, 25, 28, 34, 37, 41
azcopy (call_azcopy), 11
azure_dir_exists (list_azure_files), 35
azure_file_exists (list_azure_files), 35
AzureRMR::az_resource, 6
AzureRMR::az_resource_group, 16, 18, 27
AzureRMR::get_azure_token, 5, 10, 50
blob_container, 3, 8, 14, 20, 29, 31, 41, 49, 51
blob_dir_exists (list_blobs), 38
blob_endpoint, 7, 20, 26
blob_endpoint (storage_endpoint), 49
blob_exists (list_blobs), 38
break_lease (acquire_lease), 2
call_azcopy, 11, 14, 34, 37, 41
call_storage_endpoint
 (do_container_op), 19
change_lease (acquire_lease), 2
copy_url_to_blob (list_blobs), 38
copy_url_to_storage, 12
create_adls_dir (list_adls_files), 32
create_adls_filesystem, 51
create_adls_filesystem (adls_filesystem), 3
create_azure_dir (list_azure_files), 35
create_blob_container, 51
create_blob_container (blob_container), 8
create_blob_dir (list_blobs), 38
create_blob_snapshot, 15
create_file_share, 51
create_file_share (file_share), 21
create_storage_account, 7, 16, 18, 28, 51
create_storage_container (storage_container), 44
create_storage_dir (storage_container), 44
create_storage_snapshot (storage_container), 44
Date, 7, 26
delete_adls_dir (list_adls_files), 32
delete_adls_file (list_adls_files), 32
delete_adls_filesystem (adls_filesystem), 3
delete_azure_dir (list_azure_files), 35
delete_azure_file (list_azure_files), 35
delete_blob (list_blobs), 38
delete_blob_container (blob_container), 8
delete_blob_dir (list_blobs), 38
delete_blob_snapshot (create_blob_snapshot), 15
delete_blob_version (list_blob_versions), 43
delete_file_share (file_share), 21
delete_storage_account, 7, 17, 18, 28
delete_storage_container (storage_container), 44
delete_storage_dir (storage_container), 44
delete_storage_file (storage_container), 44
delete_storage_snapshot (storage_container), 44
delete_storage_version (storage_container), 44
do_container_op, 19
download_adls_file, 12, 14, 52, 54
download_adls_file (list_adls_files), 32
download_azure_file, 12, 14, 52, 54
download_azure_file (list_azure_files), 35
download_blob, 12, 14, 52, 54
download_blob (list_blobs), 38
download_from_url
  (copy_url_to_storage), 12
endpoint (storage_endpoint), 49
file_endpoint, 7, 20, 26
file_endpoint (storage_endpoint), 49
file_share, 14, 20, 21, 29, 31, 37, 49, 51
get_account_sas, 23
get_adls_file_acl
  (get_storage_properties), 30
get_adls_file_status
  (get_storage_properties), 30
get_service_sas (get_account_sas), 23
get_storage_account, 7, 17, 18, 27
get_storage_metadata, 16, 28, 31
get_storage_properties, 16, 29, 30
get_user_delegation_key
  (get_account_sas), 23
get_user_delegation_sas
  (get_account_sas), 23
http::DELETE, 20
http::GET, 20
http::HEAD, 20
http::PATCH, 20
http::POST, 20
http::PUT, 20
list_adls_files, 32, 49
list_adls_filesystems
  (adls_filesystem), 3
list_azure_files, 35, 49
list_blob_containers (blob_container), 8
list_blob_snapshots, 31, 41
list_blob_snapshots
  (create_blob_snapshot), 15
list_blob_versions, 41, 43
list_blobs, 38, 49
list_file_shares (file_share), 21
list_storage_accounts
  (get_storage_account), 27
list_storage_containers
  (storage_container), 44
list_storage_files (storage_container), 44
list_storage_snapshots
  (storage_container), 44
list_storage_versions
  (storage_container), 44
load, 52
multicopy_url_to_blob (list_blobs), 38
multicopy_url_to_storage
  (copy_url_to_storage), 12
multidownload_adls_file
  (list_adls_files), 32
multidownload_azure_file
  (list_azure_files), 35
multidownload_blob (list_blobs), 38
multiupload_adls_file
  (list_adls_files), 32
multiupload_azure_file
  (list_azure_files), 35
multiupload_blob (list_blobs), 38
POSIXt, 7, 26
print.adls_endpoint (storage_endpoint), 49
print.adls_filesystem
  (adls_filesystem), 3
print.blob_container (blob_container), 8
print.file_share (file_share), 21
print.storage_endpoint
  (storage_endpoint), 49
processx::run, 12
queue_endpoint (storage_endpoint), 49
rawConnection, 33, 36, 40
read.csv, 54
readr::read_delim, 54
readr::write_delim, 54
release_lease (acquire_lease), 2
renew_lease (acquire_lease), 2
revoke_user_delegation_keys
  (get_account_sas), 23
sas (get_account_sas), 23
save, 52
saveRDS, 52
set_storage_metadata (get_storage_metadata), 28
Shared access signatures, 7
shared-access-signature (get_account_sas), 23
shared_access_signature (get_account_sas), 23
sign_request, 44
snapshots, 43
storage_container, 5, 10, 14, 22, 44
storage_dir_exists (storage_container), 44
storage_download, 34, 37, 41, 49, 52, 54
storage_download (copy_url_to_storage), 12
storage_endpoint, 4, 5, 9, 10, 22, 49, 49
storage_file_exists (storage_container), 44
storage GENERICS (storage_container), 44
storage_load_rdata (storage_save_rds), 52
storage_load_rds (storage_save_rds), 52
storage_multidownload (copy_url_to_storage), 12
storage_multiupload (copy_url_to_storage), 12
storage_read_csv (storage_write_delim), 53
storage_read_csv2 (storage_write_delim), 53
storage_read_delim (storage_write_delim), 53
storage_save_rdata (storage_save_rds), 52
storage_save_rds, 52
storage_upload (copy_url_to_storage), 12
storage_write_csv (storage_write_delim), 53
storage_write_csv2 (storage_write_delim), 53
storage_write_delim, 53

table_endpoint (storage_endpoint), 49
textConnection, 33, 36, 40

upload_adls_file (list_adls_files), 32
upload_azure_file (list_azure_files), 35
upload_blob (list_blobs), 38
upload_to_url (copy_url_to_storage), 12
write.table, 54