Package 'AzureStor'

May 24, 2019

Title Storage Management in 'Azure'

Version 2.0.2

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

License MIT + file LICENSE

URL https://github.com/Azure/AzureStor

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

VignetteBuilder knitr

Depends R (>= 3.3),

Imports utils, parallel, R6, httr, mime, openssl, xml2, AzureRMR

Suggests knitr, jsonlite, testthat

RoxygenNote 6.1.1

NeedsCompilation no

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

Maintainer Hong Ooi <hongooi@microsoft.com>

Repository CRAN

Date/Publication 2019-05-24 18:00:03 UTC

R topics documented:

`acquire_lease` *Operations on blob leases*

Description

Manage leases for blobs and blob containers.

Usage

```
acquire_lease(container, blob = "", duration = 60, lease = NULL)

break_lease(container, blob = "", period = NULL)

release_lease(container, blob = "", lease)

renew_lease(container, blob = "", lease)

change_lease(container, blob = "", lease, new_lease)
```

Arguments

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

```
adls_filesystem(endpoint, ...)

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

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

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

list_adls_filesystems(endpoint, ...)

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

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

create_adls_filesystem(endpoint, ...)

## S3 method for class 'character'
```

```
create_adls_filesystem(endpoint, key = NULL,
  token = NULL, sas = NULL,
  api_version = getOption("azure_adls_api_version"), ...)

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

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

delete_adls_filesystem(endpoint, ...)

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

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

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

Arguments

endpoint	Either an ADLSgen2 endpoint object as created by 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 interacting with the host. Currently defaults to "2018-06-17".
name	The name of the filesystem to get, create, or delete.
x	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.

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

Value

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

```
## 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.

Usage

```
az_storage
```

Format

An object of class `R6ClassGenerator` of length 24.

Methods

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

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

Initialization

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

Shared access signatures

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

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

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

Endpoints

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

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

To create a storage endpoint independently of Resource Manager (for example if you are a user without admin or owner access to the account), use the blob_endpoint or file_endpoint functions.

If a storage endpoint is created without an access key and SAS, only public (anonymous) access is possible.

See Also

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

Examples

```
## Not run:

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

# list account access keys
stor$list_keys()

# regenerate a key
stor$regen_key(1)

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

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


## End(Not run)
```

---

blob_container *Operations on a blob endpoint*

Description

Get, list, create, or delete blob containers.

Usage

```
blob_container(endpoint, ...)

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

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

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

list_blob_containers(endpoint, ...)

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

## 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 "2018-03-28".
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.

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

Value

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

```
## 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)
```

---

call_azcopy *Call the azcopy file transfer utility*

Description

Call the azcopy file transfer utility

Usage

```
call_azcopy(...)

azcopy_login(force = FALSE)
```

Arguments

`...`	Arguments to pass to AzCopy on the commandline. If no arguments are supplied, a help screen is printed.
`force`	For `azcopy_login`, whether to force AzCopy to relogin. If `FALSE` (the default), and AzureStor has detected that AzCopy has already logged in, this has no effect.

Details

AzureStor has the ability to use the Microsoft AzCopy commandline utility to transfer files. To enable this, 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, passing it any arguments as required.

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.

AzCopy uses its own mechanisms for authenticating with Azure Active Directory, which is independent of the OAuth tokens used by AzureStor. AzureStor will try to ensure that AzCopy has previously authenticated before trying to transfer a file with a token, but this may not always succeed. You can run `azcopy_login(force=TRUE)` to force it to authenticate.

See Also

AzCopy page on Microsoft Docs

AzCopy GitHub repo

`create_storage_account` *Create Azure storage account*

Description

Method for the AzureRMR::az_resource_group class.

Usage

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

Arguments

- `name`: The name of the storage account.
- `location`: The location/region in which to create the account. Defaults to the resource group location.
- `kind`: The type of account, either `"StorageV2"` (the default), `"FileStorage"` or `"BlobStorage"`.
- `replication`: The replication strategy for the account. The default is locally-redundant storage (LRS).
- `access_tier`: The access tier, either `"hot"` or `"cool"`, for blobs.
- `https_only`: Whether a HTTPS connection is required to access the storage.

- `hierarchical_namespace_enabled`: Whether to enable hierarchical namespaces, which are a feature of Azure Data Lake Storage Gen 2 and provide more 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. Currently, AzureStor provides an R interface to ADLSgen2, blob and file storage.

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

Value

An object of class `az_storage` representing the created storage account.

See Also

get_storage_account, delete_storage_account, az_storage

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

Examples

```
## 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

```
## 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)
```

`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, confirm = TRUE,
  ...)
```

Arguments

endpoint	Either a file endpoint object as created by storage_endpoint, or a character string giving the URL of the endpoint.
...	Further arguments passed to lower-level functions.
key, token, sas	If an endpoint object is not supplied, authentication credentials: either an access key, an Azure Active Directory (AAD) token, or a SAS, in that order of priority.
api_version	If an endpoint object is not supplied, the storage API version to use when interacting with the host. Currently defaults to "2018-03-28".
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

```
## 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")

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


## End(Not run)
```

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

Description

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

Usage

```
get_storage_account(name)
list_storage_accounts()
```

Arguments

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

Details

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

Value

For `get_storage_account()`, an object of class `az_storage` representing the storage account.

For `list_storage_accounts()`, a list of such objects.

See Also

create_storage_account, delete_storage_account, az_storage, Azure Storage Provider API reference

Examples

```
## Not run:

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_properties` *Get storage properties for an endpoint or container*

Description

Get storage properties for an endpoint or container

Usage

```
get_storage_properties(object)

## S3 method for class 'storage_endpoint'
get_storage_properties(object)

## S3 method for class 'blob_container'
get_storage_properties(object)

## S3 method for class 'file_share'
get_storage_properties(object)

get_blob_properties(container, blob)

get_azure_file_properties(share, file)

get_azure_dir_properties(share, dir)
```

Arguments

`object`	An storage object.
`container, share`	A blob container or file share.
`blob, file, dir`	The name of an individual blob, file or directory.

Details

The `get_storage_properties` generic returns a list of properties for the given storage object. There are methods defined for objects of class `storage_endpoint`, `blob_container` and `file_share`. Similar functions are defined for individual blobs, files and directories.

Value

A list describing the object properties.

See Also

storage_endpoint, blob_container, file_share, [Blob service properties reference[(https://docs.microsoft.com/en-us/rest/api/storageservices/get-blob-service-properties). File service properties reference, Blob container properties reference, File share properties reference

---

`init_pool` *Parallelise multiple file transfers in the background*

Description

Parallelise multiple file transfers in the background

Usage

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

delete_pool()
```

Arguments

`max_concurrent_transfers`	The maximum number of concurrent file transfers to support, which translates into the number of background R processes to create. Each concurrent transfer requires a separate R process, so limit this is you are low on memory.
`restart`	For `init_pool`, whether to terminate an already running pool first.
`...`	Other arguments passed on to `parallel::makeCluster`.

Details

AzureStor can parallelise file transfers by utilizing a pool of R processes in the background. This often leads to significant speedups when transferring multiple small files. The pool is created by calling `init_pool`, or automatically the first time that a multiple file transfer is begun. It remains persistent for the session or until terminated by `delete_pool`.

If `init_pool` is called and the current pool is smaller than `max_concurrent_transfers`, it is resized.

See Also

multiupload_blob, multidownload_blob, parallel::makeCluster

`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.

Usage

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

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

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

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

download_adls_file(filesystem, src, dest, blocksize = 2^24,
  overwrite = FALSE, use_azcopy = FALSE)

delete_adls_file(filesystem, file, confirm = TRUE)

create_adls_dir(filesystem, dir)

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

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 `list_adls_files`, and `delete_adls_dir`, whether the operation should recurse through subdirectories. For `delete_adls_dir`, this must be TRUE to delete a non-empty directory.
`src, dest`	The source and destination files for uploading and downloading. Paths are allowed. For uploading, `src` can also be a textConnection or rawConnection object to allow transferring in-memory R objects without creating a temporary file.

blocksize	The number of bytes to upload/download per HTTP(S) request.
lease	The lease for a file, if present.
use_azcopy	Whether to use the AzCopy utility from Microsoft to do the transfer, rather than doing it in R.
max_concurrent_transfers	For multiupload_adls_file and multidownload_adls_file, the maximum number of concurrent file transfers. Each concurrent file transfer requires a separate R process, so limit this if you are low on memory.
overwrite	When downloading, whether to overwrite an existing destination file.
confirm	Whether to ask for confirmation on deleting a file or directory.

Details

upload_adls_file and download_adls_file are the workhorse file transfer functions for ADLS-gen2 storage. They each take as inputs a *single* filename or connection as the source for uploading/downloading, and a single filename as the destination.

multiupload_adls_file and multidownload_adls_file are functions for uploading and downloading *multiple* files at once. They parallelise file transfers by deploying a pool of R processes in the background, which can lead to significantly greater efficiency when transferring many small files. They take as input a *wildcard* pattern as the source, which expands to one or more files. The dest argument should be a directory.

The file transfer functions also support working with connections to allow transferring R objects without creating temporary files. For uploading, src can be a textConnection or rawConnection object. For downloading, dest can be NULL or a rawConnection object. In the former case, the downloaded data is returned as a raw vector, and for the latter, it will be placed into the connection. See the examples below.

By default, the upload and download functions will display a progress bar to track the file transfer. 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).

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.

See Also

adls_filesystem, az_storage, storage_download, call_azcopy

Examples

```
## Not run:

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

list_adls_files(fs, "/")
```

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

create_adls_dir(fs, "/newdir")

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

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

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

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

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

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

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


## End(Not run)
```

`list_azure_files` *Operations on a file share*

Description

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

Usage

```
list_azure_files(share, dir, info = c("all", "name"), prefix = NULL)

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

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

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

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

delete_azure_file(share, file, confirm = TRUE)

create_azure_dir(share, dir)

delete_azure_dir(share, dir, confirm = TRUE)
```

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.
src, dest	The source and destination files for uploading and downloading. For uploading, src can also be a textConnection or rawConnection object to allow transferring in-memory R objects without creating a temporary file.
blocksize	The number of bytes to upload/download per HTTP(S) request.
use_azcopy	Whether to use the AzCopy utility from Microsoft to do the transfer, rather than doing it in R.
max_concurrent_transfers	For multiupload_azure_file and multidownload_azure_file, the maximum number of concurrent file transfers. Each concurrent file transfer requires a separate R process, so limit this if you are low on memory.
overwrite	When downloading, whether to overwrite an existing destination file.
confirm	Whether to ask for confirmation on deleting a file or directory.

Details

upload_azure_file and download_azure_file are the workhorse file transfer functions for file storage. They each take as inputs a *single* filename or connection as the source for uploading/downloading, and a single filename as the destination.

multiupload_azure_file and multidownload_azure_file are functions for uploading and downloading *multiple* files at once. They parallelise file transfers by deploying a pool of R processes in the background, which can lead to significantly greater efficiency when transferring many small files. They take as input a *wildcard* pattern as the source, which expands to one or more files. The dest argument should be a directory.

The file transfer functions also support working with connections to allow transferring R objects without creating temporary files. For uploading, src can be a textConnection or rawConnection

object. For downloading, dest can be NULL or a rawConnection object. In the former case, the downloaded data is returned as a raw vector, and for the latter, it will be placed into the connection. See the examples below.

By default, the upload and download functions will display a progress bar to track the file transfer. 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).

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.

See Also

file_share, az_storage, storage_download, call_azcopy

AzCopy version 10 on GitHub

Examples

```
## Not run:

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

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

create_azure_dir(share, "/newdir")

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

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

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

# 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.

Usage

```
list_blobs(container, info = c("partial", "name", "all"),
  prefix = NULL)

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

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

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

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

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

Arguments

`container`	A blob container object.
`info`	For `list_blobs`, level of detail about each blob to return: a vector of names only; the name, size and last-modified date (default); or all information.
`prefix`	For `list_blobs`, filters the result to return only blobs whose name begins with this prefix.
`src, dest`	The source and destination files for uploading and downloading. See 'Details' below.For uploading, `src` can also be a textConnection or rawConnection object to allow transferring in-memory R objects without creating a temporary file. For downloading,

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

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

`lease` The lease for a blob, if present.

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

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

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

`blob` A string naming a blob.

`confirm` Whether to ask for confirmation on deleting a blob.

Details

`upload_blob` and `download_blob` are the workhorse file transfer functions for blobs. They each take as inputs a *single* filename or connection as the source for uploading/downloading, and a single filename as the destination.

`multiupload_blob` and `multidownload_blob` are functions for uploading and downloading *multiple* blobs at once. They parallelise file transfers by deploying a pool of R processes in the background, which can lead to significantly greater efficiency when transferring many small files. They take as input a wildcard pattern as the source, which expands to one or more files. The `dest` argument should be a directory.

The file transfer functions also support working with connections to allow transferring R objects without creating temporary files. For uploading, `src` can be a textConnection or rawConnection object. For downloading, `dest` can be NULL or a `rawConnection` object. In the former case, the downloaded data is returned as a raw vector, and for the latter, it will be placed into the connection. See the examples below.

By default, the upload and download functions will display a progress bar to track the file transfer. 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)`.

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.

See Also

blob_container, az_storage, storage_download, call_azcopy

AzCopy version 10 on GitHub

Examples

```
## Not run:

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

list_blobs(cont)

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

delete_blob(cont, "bigfile.zip")

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

# 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)


## End(Not run)
```

---

storage_container *Storage client generics*

Description

Storage client generics

Usage

```
storage_container(endpoint, ...)

## S3 method for class 'blob_endpoint'
```

```
storage_container(endpoint, name, ...)

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

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

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

create_storage_container(endpoint, ...)

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

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

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

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

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

delete_storage_container(endpoint, ...)

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

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

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

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

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

```
list_storage_containers(endpoint, ...)

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

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

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

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

list_storage_files(container, ...)

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

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

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

create_storage_dir(container, ...)

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

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

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

delete_storage_dir(container, ...)

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

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

## S3 method for class 'adls_filesystem'
delete_storage_dir(container, dir,
  confirm = TRUE, ...)
```

```
delete_storage_file(container, ...)

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

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

## S3 method for class 'adls_filesystem'
delete_storage_file(container, file,
  confirm = TRUE, ...)
```

Arguments

endpoint	A storage endpoint object, or for the character methods, a string giving the full URL to the container.
...	Further arguments to pass to lower-level functions.
name	For the storage container management methods, a container name.
key, token, sas	For the character methods, authentication credentials for the container: either an access key, an Azure Active Directory (AAD) token, or a SAS. If multiple arguments are supplied, a key takes priority over a token, which takes priority over a SAS.
confirm	For the deletion methods, whether to ask for confirmation first.
container	A storage container object.
file, dir	For the storage object management methods, a file or directory name.

Details

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

Storage container management methods:

- storage_container dispatches to blob_container, file_share or adls_filesystem
- create_storage_container dispatches to create_blob_container, create_file_share or create_adls_filesystem
- delete_storage_container dispatches to delete_blob_container, delete_file_share or delete_adls_filesystem
- list_storage_containers dispatches to list_blob_containers, list_file_shares or list_adls_filesystems

Storage object management methods:

- list_storage_files dispatches to list_blobs, list_azure_files or list_adls_files
- create_storage_dir dispatches to create_azure_dir or create_adls_dir; throws an error if passed a blob container

- `delete_storage_dir` dispatches to `delete_azure_dir` or `delete_adls_dir`; throws an error if passed a blob container
- `delete_storage_file` dispatches to `delete_blob`, `delete_azure_file` or `delete_adls_file`

See Also

storage_endpoint, blob_container, file_share, adls_filesystem

list_blobs, list_azure_files, list_adls_files

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

Examples

```
## 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 *Create a storage endpoint object*

Description

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

Usage

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

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

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

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

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

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

Arguments

endpoint	The URL (hostname) for the endpoint. This must be of the form http[s]://{account-name}.{type}.{ where type is one of "dfs" (corresponding to ADLSgen2), "blob", "file", "queue" or "table". On the public Azure cloud, endpoints will be of the form https://{account-name}.{type}.core.windows.net.
key	The access key for the storage account.
token	An Azure Active Directory (AAD) authentication token. This can be either a string, or an object of class AzureToken created by AzureRMR::get_azure_token. The latter is the recommended way of doing it, as it allows for automatic refreshing of expired tokens.
sas	A shared access signature (SAS) for the account.
api_version	The storage API version to use when interacting with the host. Defaults to "2018-06-17" for the ADLSgen2 endpoint, and "2018-03-28" for the others.
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. Note that authentication with a SAS is not currently supported by ADLSgen2.

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.

Currently AzureStor only includes methods for interacting with ADLSgen2, blob and file storage.

See Also

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

Examples

```
## Not run:

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

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

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


## End(Not run)
```

---

storage_upload *Upload and download generics*

Description

Upload and download generics

Usage

```
storage_upload(container, ...)

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

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

## S3 method for class 'adls_filesystem'
```

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

storage_multiupload(container, ...)

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

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

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

storage_download(container, ...)

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

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

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

storage_multidownload(container, ...)

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

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

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

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.
...	Further arguments to pass to lower-level functions.
src, dest	The source and destination files to transfer.
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

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, `storage_download` and `download_from_url` 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

```
## 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)
```

Index