Package: azkit (via r-universe)

May 20, 2026

Title Azure storage authentication toolkit

Version 0.3.1

Description Handles authentication and basic tasks with Azure blob storage.

License MIT + file LICENSE

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.5.0)

Imports arrow, AzureAuth, AzureRMR, AzureStor, cli, dplyr, glue, httr2, purrr, rlang, tibble, withr, yyjsonr

Suggests httpuv, readr, testthat (>= 3.0.0)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake libxml2-dev libssl-dev

Repository https://the-strategy-unit.r-universe.dev

Date/Publication 2026-05-20 14:49:18 UTC

RemoteUrl https://github.com/The-Strategy-Unit/azkit

RemoteRef HEAD

RemoteSha 98e6a2a2d551b8365b6acac5d587f7f5c04362c4

Contents

`check_container_class` *Check that a container looks like a real container*

Description

Check that a container looks like a real container

Usage

```
check_container_class(container)
```

Arguments

`container`	An Azure container object, as returned by get_container

`check_nzchar` *Check if a supplied non-NULL value is a string with >0 characters*

Description

Will error if x is equal to `""`, or if it is otherwise missing or invalid. With the exception that if x is NULL, then NULL will be passed through.

Usage

```
check_nzchar(x, message, pf = parent.frame())
```

Arguments

x	The object to be checked
message	A custom error message, as a string. Will be shown to the user if the check does not pass. Can include glue variables and {cli} semantic markup. Variable values will be searched for in the environment of the caller function (not in the environment of check_nzchar()). This makes it easier to include informative values in the message.
pf	Set as parent.frame() so variables in the caller environment can be used in the custom error message.

---

check_scalar_type *An alternative to stopifnot/assert_that etc*

Description

This function makes it easy to use the is_scalar_* functions from {rlang} to check the type of x, *and* that length(x) == 1, and supports the seamless use of glue strings in the custom error message. Possible values for the type parameter are: "character", "logical", "list", "integer", "double", "string", "bool", "bytes", "raw", "vector", "complex".

Usage

```
check_scalar_type(x, type, message, pf = parent.frame())
```

Arguments

x	The object to be checked
type	A string defining the R object type that x is checked to be
message	A custom error message, as a string. Will be shown to the user if the predicate check does not succeed. Can include glued variables and {cli} semantic markup.
pf	Set as parent.frame() so variables in the caller environment can be used in the custom error message.

See Also

check_that

check_that *An alternative to stopifnot/assert_that etc*

Description

If the predicate function is true of `x` then `x` is returned. Otherwise, an error is thrown with a custom `message`.

Usage

```
check_that(x, predicate, message, pf = parent.frame())
```

Arguments

`x`	The object to be checked
`predicate`	The predicate function used to check `x`
`message`	A custom error message, as a string. Will be shown to the user if the predicate check does not succeed. Can include `glued` variables and `{cli}` semantic markup.
`pf`	Set as `parent.frame()` so variables in the caller environment can be used in the custom error message.

See Also

check_vec

check_vec *An alternative to stopifnot/assert_that etc*

Description

This function makes it easy to use the `{purrr}` functions `every()`, `some()` and `none()` to handle vector inputs of length >= 1, and supports the seamless use of `glue` strings in the custom error message. Not suitable for checking if `length(x) == 1` as it will check vectors element-wise, so will potentially return `TRUE` even if `length(x) > 1`

Usage

```
check_vec(
  x,
  predicate,
  message,
  which = c("every", "some", "none"),
  pf = parent.frame()
)
```

Arguments

x	The object to be checked
predicate	The predicate function used to check elements of x
message	A custom error message, as a string. Will be shown to the user if the predicate check does not succeed. Can include glued variables and {cli} semantic markup. Variable values will be searched for in the environment of the caller function (not in the environment of check_vec()). This makes it easier to include informative values in the message.
which	One of "every", "some", "none", as a string. Defines which {purrr} function to use when applying the predicate. "every", the default, means that check_vec() will succeed if every value of x satisfies the predicate. "none" can be used to generate an inverse predicate, or the situation where success means that none of the elements of x satisfies the predicate. "some" is unlikely to be useful often, but it is available.
pf	Set as parent.frame() so variables in the caller environment can be used in the custom error message.

See Also

check_scalar_type()

---

cst_error_msg *A custom error message generator for the* check_scalar_type *function*

Description

A custom error message generator for the check_scalar_type function

Usage

```
cst_error_msg(text)
```

Arguments

text	The main text of the error message

ct_error_msg *A custom error message generator for the* check_that *function*

Description

A custom error message generator for the check_that function

Usage

```
ct_error_msg(text)
```

Arguments

text	The main text of the error message

cv_error_msg *A custom error message generator for the* check_vec *function*

Description

A custom error message generator for the check_vec function

Usage

```
cv_error_msg(text)
```

Arguments

text	The main text of the error message

generate_resource *Generate appropriate values for the* resource *parameter in* get_auth_token

Description

A helper function to generate appropriate values. Ensure that the version argument matches the aad_version argument to get_auth_token. It's unlikely that you will ever want to set authorise to FALSE but it's here as an option since AzureAuth::get_azure_token supports it. Similarly, you are likely to want to keep refresh turned on (this argument has no effect on v1 tokens, it only applies to v2).

Usage

```
generate_resource(
  version = 1,
  url = "https://storage.azure.com",
  path = "/.default",
  authorise = TRUE,
  refresh = TRUE
)
```

Arguments

version	numeric. The AAD version, either 1 or 2 (1 by default)
url	The URL of the Azure resource host
path	For v2, the path designating the access scope
authorise	Boolean, whether to return a token with authorisation scope, (TRUE, the default) or one that just provides authentication. You are unlikely to want to turn this off
refresh	Boolean, applies to v2 tokens only, whether to return a token that has a refresh token also supplied.

Value

A scalar character, or (in most v2 situations) a character vector

---

get_auth_token *Get Azure authentication token*

Description

This function retrieves an Azure token for a specified resource.

This method avoids the need to refresh by re-authenticating online. It seems that this only works with v1 tokens. (v2 tokens always seem to refresh via online re-authentication, but they *ought* to refresh automatically.) To instead generate a completely fresh token, set `force_refresh = TRUE` in get_auth_token

Usage

```
get_auth_token(
  resource = generate_resource(),
  tenant = "common",
  client_id = NULL,
  auth_method = "authorization_code",
  aad_version = 1,
  force_refresh = FALSE,
  ...
)

refresh_token(token)
```

Arguments

resource	For v1, a simple URL such as "https://storage.azure.com/" should be supplied. For v2, a vector specifying the URL of the Azure resource for which the token is requested as well as any desired scopes. See AzureAuth::get_azure_token for details. Use generate_resource to help provide an appropriate string or vector. If setting version to 2, ensure that the aad_version argument is also set to 2. Both are set to use AAD version 1 by default.
tenant	A string specifying the Azure tenant. Defaults to "common". See AzureAuth::get_azure_token for other values.
client_id	A string specifying the application ID (aka client ID). If NULL, (the default) the function attempts to obtain the client ID from the Azure Resource Manager token, or prompts the user to log in to obtain it.
auth_method	A string specifying the authentication method. Defaults to "authorization_code". To use a secret, pass "client_credentials" instead and provide the secret using the password argument in .... See AzureAuth::get_azure_token for more information.
aad_version	Numeric. The AAD version, either 1 or 2 (1 by default)
force_refresh	logical. Whether to use a stored token if available (FALSE, the default), or try to obtain a new one from Azure (TRUE). This may be useful if you wish to generate a new token with the same resource value as an existing token, but a different tenant or auth_method. Note that you can also try refresh_token, which should cause an existing token to refresh itself, without obtaining a new token from Azure via online reauthentication
...	Optional arguments (eg token_args or use_cache) to be passed on to AzureAuth::get_managed_token or AzureAuth::get_azure_token, for example to overwrite any opf their default values or to supply a password
token	An Azure authentication token

Details

It will try to get a managed token when used within a managed resource such as Azure VM or Azure App Service.

If this method does not return a token, it will try to retrieve a user token using the provided parameters, requiring the user to have authenticated using their device. If force_refresh is set to TRUE, a fresh web authentication process should be launched. Otherwise it will attempt to use a cached token matching the given resource, tenant and aad_version.

Value

An Azure token object

An Azure authentication token

Examples

```
## Not run:
# Get a token for the default resource
```

```
token <- get_auth_token()

# Force generation of a new token via online reauthentication
token <- get_auth_token(force_refresh = TRUE)

# Get a token for a specific resource and tenant
token <- get_auth_token(
  resource = "https://graph.microsoft.com",
  tenant = "my-tenant-id"
)

# Get a token using a specific app ID
token <- get_auth_token(client_id = "my-app-id")

# Use a secret
token <- get_auth_token(
 tenant = "my-tenant-id",
 client_id = "my-app-id",
 auth_method = "client_credentials",
 password = "123459878&%^"
)

## End(Not run)
```

`get_container` *Get Azure storage container*

Description

It may be helpful to set the environment variable "AZ_STORAGE_EP". This can contain your usual Azure storage endpoint URL should you not wish to pass it in explicitly to the function. You may find it helpful to use list_container_names to get a list of available container names.

Usage

```
get_container(
  container_name,
  endpoint_url = Sys.getenv("AZ_STORAGE_EP"),
  token = get_auth_token()
)
```

Arguments

`container_name`	Name of the container as a string.
`endpoint_url`	An Azure endpoint URL.
`token`	An Azure authentication token, or a function that returns one. Uses get_auth_token by default.

Value

An Azure blob container (list object of class "blob_container")

`imds_available` *Check if Azure Instance Metadata Service (IMDS) is Available*

Description

This function checks if the Azure Instance Metadata Service (IMDS) is available by attempting to make a request to the IMDS endpoint. The result is cached in an environment variable for future use, saving the need for repeated checks.

Usage

```
imds_available()
```

Details

You can also set the `IMDS_AVAILABLE` environment variable manually to "TRUE" or "FALSE" to override the automatic check, which can be useful for testing or in environments where the check may not work correctly.

`list_container_names` *Return a list of container names that are found at the endpoint*

Description

Return a list of container names that are found at the endpoint

Usage

```
list_container_names(
  endpoint_url = Sys.getenv("AZ_STORAGE_EP"),
  token = get_auth_token()
)
```

Arguments

`endpoint_url`	An Azure endpoint URL.
`token`	An Azure authentication token, or a function that returns one. Uses get_auth_token by default.

Value

A character vector of all container names found

`list_files` *List files in a container*

Description

Lists all files (recursively, if desired) found in a container within a given directory (`dir`). The search can be restricted to files with a specific extension.

Usage

```
list_files(container, dir = "", ext = "", recursive = FALSE)
```

Arguments

`container`	An Azure container object, as returned by get_container
`dir`	(optional) The directory of the container to list files within. `""` (the root directory of the container) by default
`ext`	(optional) A string giving the extension of a particular file type to restrict the list to. No need to include the initial ".". The default, `""`, means no filtering by file extension will be applied.
`recursive`	logical: whether to list files recursively. Default `FALSE`

Details

The function does not support filtering by file name, only by file extension.

The returned file list (character vector) contains the full paths to the files, ready to be passed perhaps to a `read_azure_*` function, or filtered further. If you just want the names of the files without the folder path, use basename to extract these.

Value

A vector of file names, or an empty character vector if none found

Examples

```
## Not run:
  list_files(get_container("example"), ext = "csv")

## End(Not run)
```

read_azure_csv *Read a csv file from Azure storage*

Description

Read a csv file from Azure storage

Usage

```
read_azure_csv(container, file, ...)
```

Arguments

container	An Azure container object, as returned by get_container
file	string The path to the file to be read.
...	optional arguments to be passed through to readr::read_delim

Value

A tibble

read_azure_file *Read any file from Azure storage*

Description

Read any file from Azure storage

Usage

```
read_azure_file(container, file, ...)
```

Arguments

container	An Azure container object, as returned by get_container
file	string The path to the file to be read.
...	optional arguments to be passed through to AzureStor::download_blob

Value

A raw data stream

read_azure_json *Read a json file from Azure storage*

Description

Read a json file from Azure storage

Usage

```
read_azure_json(container, file, ...)
```

Arguments

container	An Azure container object, as returned by get_container
file	string The path to the file to be read.
...	optional arguments to be passed through to yyjsonr::read_json_raw

Value

A list

read_azure_jsongz *Read a json.gz file from Azure storage*

Description

Read a json.gz file from Azure storage

Usage

```
read_azure_jsongz(container, file, ...)
```

Arguments

container	An Azure container object, as returned by get_container
file	string The path to the file to be read.
...	optional arguments to be passed through to yyjsonr::read_json_file

Value

A list

read_azure_parquet *Read a parquet file from Azure storage*

Description

Read a parquet file from Azure storage

Usage

```
read_azure_parquet(container, file, ...)
```

Arguments

container	An Azure container object, as returned by get_container
file	string The path to the file to be read.
...	optional arguments to be passed through to arrow::read_parquet

Value

A tibble

Examples

```
## Not run:
  read_azure_parquet(cont, "data/folder/path/1.parquet")

## End(Not run)
```

read_azure_rds *Read an rds file from Azure storage*

Description

Read an rds file from Azure storage

Usage

```
read_azure_rds(container, file, ...)
```

Arguments

container	An Azure container object, as returned by get_container
file	string The path to the file to be read.
...	optional arguments to be passed through to AzureStor::storage_load_rds. For example, a compression type (one of c("unknown", "gzip", "bzip2", "xz", "zstd", "none")) can be provided using the argument type, which will be passed on to memDecompress via AzureStor::storage_load_rds.

Value

The data object that was stored in the rds file

`read_azure_table` *Read in data from an Azure table*

Description

Read in data from an Azure table

Usage

```
read_azure_table(
  table_name,
  table_endpoint = Sys.getenv("AZ_TABLE_EP"),
  token = get_auth_token(),
  filter = NULL,
  select = NULL,
  top = NULL
)
```

Arguments

`table_name`	Name of the table to be read.
`table_endpoint`	An Azure table endpoint URL.
`token`	An Azure authentication token, or a function that returns one. Uses get_auth_token by default.
`filter`	An OData filter string to filter the results.
`select`	An OData select string to specify which properties to return.
`top`	An integer specifying the maximum number of records to return.

Value

A tibble

`read_azure_table_single_entity` *Read in data from an Azure table*

Description

Read in data from an Azure table

Usage

```
read_azure_table_single_entity(
  table_name,
  partition_key,
  row_key,
  table_endpoint = Sys.getenv("AZ_TABLE_EP"),
  token = get_auth_token()
)
```

Arguments

`table_name`	Name of the table to be read.
`partition_key`	The partition key of the entity to be read.
`row_key`	The row key of the entity to be read.
`table_endpoint`	An Azure table endpoint URL.
`token`	An Azure authentication token, or a function that returns one. Uses get_auth_token by default.

Value

A tibble

Index