Package 'AzureQstor'

June 2, 2020

Title Interface to 'Azure Queue Storage'

Version 1.0.0

Description An interface to 'Azure Queue Storage'. This is a cloud service for storing large numbers of messages, for example from automated sensors, that can be accessed remotely via authenticated calls using HTTP or HTTPS. Queue storage is often used to create a backlog of work to process asynchronously. Part of the 'AzureR' family of packages.

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

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

License MIT + file LICENSE

Depends R (>= 3.3)

Imports utils, AzureRMR (>= 2.0.0), AzureStor (>= 3.0.0), openssl, httr

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.0

NeedsCompilation no

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

Maintainer Hong Ooi <`hongooi@microsoft.com`>

Repository CRAN

Date/Publication 2020-06-02 12:30:02 UTC

R topics documented:

`QueueMessage` *R6 class representing a message from an Azure storage queue*

Description

This class stores the data, metadata and behaviour associated with a message.

To generate a message object, call one of the methods exposed by the `StorageQueue` class.

Public fields

`queue` The queue this message is from, an object of class `StorageQueue`

`id` The message ID.

`insertion_time` The message insertion (creation) time.

`expiry_time` The message expiration time.

`text` The message text.

`receipt` A pop receipt. This is present if the message was obtained by means other than peeking, and is required for updating or deleting the message.

`next_visible_time` The time when this message will be next visible.

`dequeue_count` The number of times this message has been read.

Methods

Public methods:

- `QueueMessage$new()`
- `QueueMessage$delete()`
- `QueueMessage$update()`
- `QueueMessage$print()`
- `QueueMessage$clone()`

Method `new()`**:** Creates a new message object. Rather than calling the `new` method manually, objects of this class should be created via the methods exposed by the `StorageQueue` object.

Usage:

```
QueueMessage$new(message, queue)
```

Arguments:

`message` Details about the message.

`queue` Object of class `StorageQueue`.

Method `delete()`**:** Deletes this message from the queue.

Usage:

```
QueueMessage$delete()
```

Returns: NULL, invisibly.

Method `update()`**:** Updates this message in the queue.

This operation can be used to continually extend the invisibility of a queue message. This functionality can be useful if you want a worker role to "lease" a message. For example, if a worker role calls `get_messages` and recognizes that it needs more time to process a message, it can continually extend the message's invisibility until it is processed. If the worker role were to fail during processing, eventually the message would become visible again and another worker role could process it.

Usage:

```
QueueMessage$update(visibility_timeout, text = self$text)
```

Arguments:

`visibility_timeout` The new visibility timeout (time to when the message will again be visible).

`text` Optionally, new message text, either a raw or character vector. If a raw vector, it is base64-encoded, and if a character vector, it is collapsed into a single string before being sent to the queue.

Returns: The message object, invisibly.

Method `print()`**:** Print method for this class.

Usage:

```
QueueMessage$print(...)
```

Arguments:

`...` Not currently used.

Returns: The message object, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
QueueMessage$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:

endp <- storage_endpoint("https://mystorage.queue.core.windows.net", key="key")
queue <- storage_queue(endp, "queue1")

msg <- queue$get_message()
msg$update(visibility_timeout=60, text="updated message")
msg$delete()


## End(Not run)
```

queue_endpoint *Create a queue endpoint object*

Description

Create a queue endpoint object

Usage

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

Arguments

`endpoint`	The URL (hostname) for the endpoint, of the form http[s]://{account-name}.queue.{core-host-name}. On the public Azure cloud, endpoints will be of the form https://{account-name}.queue.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"`.

Details

This is the queue storage counterpart to the endpoint functions defined in the AzureStor package.

Value

An object of class `queue_endpoint`, inheriting from `storage_endpoint`.

See Also

`AzureStor::storage_endpoint`, `AzureStor::blob_endpoint`, `storage_queue`

Examples

```
## Not run:

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

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


## End(Not run)
```

`StorageQueue` *R6 class representing an Azure storage queue*

Description

A storage queue holds messages. A queue can contain an unlimited number of messages, each of which can be up to 64KB in size. Messages are generally added to the end of the queue and retrieved from the front of the queue, although first in, first out (FIFO) behavior is not guaranteed.

To generate a queue object, use one of the `storage_queue`, `list_storage_queues` or `create_storage_queue` functions rather than calling the `new()` method directly.

Public fields

`endpoint` A queue endpoint object. This contains the account and authentication information for the queue.

`name` The name of the queue.

Methods

Public methods:

- `StorageQueue$new()`
- `StorageQueue$create()`
- `StorageQueue$delete()`
- `StorageQueue$clear()`
- `StorageQueue$get_metadata()`
- `StorageQueue$set_metadata()`
- `StorageQueue$get_message()`
- `StorageQueue$get_messages()`
- `StorageQueue$peek_message()`

- `StorageQueue$peek_messages()`
- `StorageQueue$pop_message()`
- `StorageQueue$pop_messages()`
- `StorageQueue$put_message()`
- `StorageQueue$update_message()`
- `StorageQueue$delete_message()`
- `StorageQueue$print()`
- `StorageQueue$clone()`

Method `new()`**:** Initialize the queue object. Rather than calling this directly, you should use one of the `storage_queue`, `list_storage_queues` or `create_storage_queue` functions.

Note that initializing this object is a local operation only. If a queue of the given name does not already exist in the storage account, it has to be created remotely by calling the `create` method.

Usage:

```
StorageQueue$new(endpoint, name)
```

Arguments:

`endpoint` An endpoint object.

`name` The name of the queue.

Method `create()`**:** Creates a storage queue in Azure, using the storage endpoint and name from this R6 object.

Usage:

```
StorageQueue$create()
```

Returns: The queue object, invisibly.

Method `delete()`**:** Deletes this storage queue in Azure.

Usage:

```
StorageQueue$delete(confirm = TRUE)
```

Arguments:

`confirm` Whether to ask for confirmation before deleting.

Returns: The queue object, invisibly.

Method `clear()`**:** Clears (deletes) all messages in this storage queue.

Usage:

```
StorageQueue$clear()
```

Method `get_metadata()`**:** Retrieves user-defined metadata for the queue.

Usage:

```
StorageQueue$get_metadata()
```

Returns: A named list of metadata properties.

Method `set_metadata()`**:** Sets user-defined metadata for the queue.

Usage:

```
StorageQueue$set_metadata(..., keep_existing = TRUE)
```

Arguments:

`...` Name-value pairs to set as metadata.

`keep_existing` Whether to retain existing metadata information.

Returns: A named list of metadata properties, invisibly.

Method `get_message()`: Reads a message from the front of the storage queue.

When a message is read, the consumer is expected to process the message and then delete it. After the message is read, it is made invisible to other consumers for a specified interval. If the message has not yet been deleted at the time the interval expires, its visibility is restored, so that another consumer may process it.

Usage:

```
StorageQueue$get_message()
```

Returns: A new object of class `QueueMessage`.

Method `get_messages()`: Reads several messages at once from the front of the storage queue.

When a message is read, the consumer is expected to process the message and then delete it. After the message is read, it is made invisible to other consumers for a specified interval. If the message has not yet been deleted at the time the interval expires, its visibility is restored, so that another consumer may process it.

Usage:

```
StorageQueue$get_messages(n = 1)
```

Arguments:

`n` How many messages to read. The maximum is 32.

Returns: A list of objects of class `QueueMessage`.

Method `peek_message()`: Reads a message from the storage queue, but does not alter its visibility.

Note that a message obtained via the `peek_message` or `peek_messages` method will not include a pop receipt, which is required to delete or update it.

Usage:

```
StorageQueue$peek_message()
```

Returns: A new object of class `QueueMessage`.

Method `peek_messages()`: Reads several messages at once from the storage queue, without altering their visibility.

Note that a message obtained via the `peek_message` or `peek_messages` method will not include a pop receipt, which is required to delete or update it.

Usage:

```
StorageQueue$peek_messages(n = 1)
```

Arguments:

`n` How many messages to read. The maximum is 32.

Returns: A list of objects of class `QueueMessage`.

Method `pop_message()`: Reads a message from the storage queue, removing it at the same time. This is equivalent to calling `get_message` and `delete_message` successively.

Usage:

```
StorageQueue$pop_message()
```

Returns: A new object of class `QueueMessage`.

Method `pop_messages()`: Reads several messages at once from the storage queue, and then removes them.

Usage:

```
StorageQueue$pop_messages(n = 1)
```

Arguments:

`n` How many messages to read. The maximum is 32.

Returns: A list of objects of class `QueueMessage`.

Method `put_message()`: Writes a message to the back of the message queue.

Usage:

```
StorageQueue$put_message(text, visibility_timeout = NULL, time_to_live = NULL)
```

Arguments:

`text` The message text, either a raw or character vector. If a raw vector, it is base64-encoded, and if a character vector, it is collapsed into a single string before being sent to the queue.

`visibility_timeout` Optional visibility timeout after being read, in seconds. The default is 30 seconds.

`time_to_live` Optional message time-to-live, in seconds. The default is 7 days.

Returns: The message text, invisibly.

Method `update_message()`: Updates a message in the queue. This requires that the message object must include a pop receipt, which is present if it was obtained by means other than peeking.
This operation can be used to continually extend the invisibility of a queue message. This functionality can be useful if you want a worker role to "lease" a message. For example, if a worker role calls `get_messages` and recognizes that it needs more time to process a message, it can continually extend the message's invisibility until it is processed. If the worker role were to fail during processing, eventually the message would become visible again and another worker role could process it.

Usage:

```
StorageQueue$update_message(msg, visibility_timeout, text = msg$text)
```

Arguments:

`msg` A message object, of class `QueueMessage`.

`visibility_timeout` The new visibility timeout (time to when the message will again be visible).

`text` Optionally, new message text, either a raw or character vector. If a raw vector, it is base64-encoded, and if a character vector, it is collapsed into a single string before being sent to the queue.

Returns: The message object, invisibly.

Method `delete_message()`**:** Deletes a message from the queue. This requires that the message object must include a pop receipt, which is present if it was obtained by means other than peeking.

Usage:

```
StorageQueue$delete_message(msg)
```

Arguments:

`msg` A message object, of class `QueueMessage`.

Method `print()`**:** Print method for this class.

Usage:

```
StorageQueue$print(...)
```

Arguments:

`...` Not currently used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
StorageQueue$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`QueueMessage`

Examples

```
## Not run:

endp <- storage_endpoint("https://mystorage.queue.core.windows.net", key="key")

# to talk to an existing queue
queue <- storage_queue(endp, "queue1")

# to create a new queue
queue2 <- create_storage_queue(endp, "queue2")

# various ways to delete a queue (will ask for confirmation first)
queue2$delete()
delete_storage_queue(queue2)
delete_storage_queue(endp, "queue2")

# to get all queues in this storage account
queue_lst <- list_storage_queues(endp)

# working with a queue: put, get, update and delete messages
queue$put_message("new message")
msg <- queue$get_message()
msg$update(visibility_timeout=60, text="updated message")
queue$delete_message(msg)
```

```
# delete_message simply calls the message's delete() method, so this is equivalent
msg$delete()

# retrieving multiple messages at a time (up to 32)
msgs <- queue$get_messages(30)

# deleting is still per-message
lapply(msgs, function(m) m$delete())

# you can use the process pool from AzureRMR to do this in parallel
AzureRMR::init_pool()
AzureRMR::pool_lapply(msgs, function(m) m$delete())
AzureRMR::delete_pool()


## End(Not run)
```

`storage_queue` *Message queues*

Description

Get, list, create, or delete queues.

Usage

```
storage_queue(endpoint, ...)

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

## S3 method for class 'queue_endpoint'
storage_queue(endpoint, name, ...)

list_storage_queues(endpoint, ...)

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

## S3 method for class 'queue_endpoint'
list_storage_queues(endpoint, ...)

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

create_storage_queue(endpoint, ...)

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

## S3 method for class 'queue_endpoint'
create_storage_queue(endpoint, name, ...)

## S3 method for class 'StorageQueue'
create_storage_queue(endpoint, ...)

delete_storage_queue(endpoint, ...)

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

## S3 method for class 'queue_endpoint'
delete_storage_queue(endpoint, name, ...)

## S3 method for class 'StorageQueue'
delete_storage_queue(endpoint, confirm = TRUE, ...)
```

Arguments

`endpoint`	Either a queue 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 queue to get, create, or delete.
`confirm`	For deleting a queue, whether to ask for confirmation.

Details

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

Value

For `storage_queue` and `create_storage_queue`, an object of class `StorageQueue`. For `list_storage_queues`, a list of such objects.

See Also

`StorageQueue`, `queue_endpoint`

Examples

```
## Not run:

endp <- storage_endpoint("https://mystorage.queue.core.windows.net", key="key")

# to talk to an existing queue
queue <- storage_queue(endp, "queue1")

# to create a new queue
queue2 <- create_storage_queue(endp, "queue2")

# various ways to delete a queue (will ask for confirmation first)
queue2$delete()
delete_storage_queue(queue2)
delete_storage_queue(endp, "queue2")

## End(Not run)
```

Index