

# Package ‘AzureCosmosR’

January 19, 2021

**Title** Interface to the 'Azure Cosmos DB' 'NoSQL' Database Service

**Version** 1.0.0

**Description** An interface to 'Azure CosmosDB': <<https://azure.microsoft.com/en-us/services/cosmos-db/>>. On the admin side, 'AzureCosmosR' provides functionality to create and manage 'Cosmos DB' instances in Microsoft's 'Azure' cloud. On the client side, it provides an interface to the 'Cosmos DB' SQL API, letting the user store and query documents and attachments in 'Cosmos DB'. Part of the 'AzureR' family of packages.

**URL** <https://github.com/Azure/AzureCosmosR>  
<https://github.com/Azure/AzureR>

**BugReports** <https://github.com/Azure/AzureCosmosR/issues>

**License** MIT + file LICENSE

**VignetteBuilder** knitr

**Depends** R (>= 3.3)

**Imports** utils, AzureRMR (>= 2.3.3), curl, openssl, jsonlite, httr,  
uuid, vctrs (>= 0.3.0)

**Suggests** AzureTableStor, mongolite, DBI, odbc, dplyr, testthat, knitr,  
rmarkdown

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Hong Ooi [aut, cre],  
Andrew Liu [ctb] (Assistance with Cosmos DB),  
Microsoft [cph]

**Maintainer** Hong Ooi <[hongooi73@gmail.com](mailto:hongooi73@gmail.com)>

**Repository** CRAN

**Date/Publication** 2021-01-18 23:50:05 UTC

## R topics documented:

az_cosmosdb	2
bulk_delete	3

bulk_import . . . . .	5
cosmos_endpoint . . . . .	6
cosmos_mongo_endpoint . . . . .	9
create_cosmosdb_account . . . . .	11
delete_cosmosdb_account . . . . .	12
do_cosmos_op . . . . .	13
get_cosmosdb_account . . . . .	14
get_cosmos_container . . . . .	15
get_cosmos_database . . . . .	17
get_document . . . . .	19
get_partition_key . . . . .	21
get_stored_procedure . . . . .	22
get_udf . . . . .	24
query_documents . . . . .	26

## Index 29

---

az_cosmosdb	<i>Azure Cosmos DB account class</i>
-------------	--------------------------------------

---

### Description

Class representing an Azure Cosmos DB account. For working with the data inside the account, see [cosmos\\_endpoint](#) and [cosmos\\_database](#).

### Methods

The following methods are available, in addition to those provided by the [AzureRMR::az\\_resource](#) class:

- `list_keys(read_only=FALSE)`: Return the access keys for this account.
- `regen_key(kind)`: Regenerate (change) an access key. `kind` should be one of "primary", "secondary", "primaryReadonly" or "secondaryReadonly".
- `get_endpoint(interface, ...)`: Return a default endpoint object for interacting with the data. See 'Endpoints' below.
- `get_sql_endpoint(key, key_type)`: Return an object representing the core (SQL) endpoint of the account.
- `get_table_endpoint(key)`: Return an object representing the table storage endpoint of the account.
- `get_mongo_endpoint(collection, key, mongo_options)`: Return an object representing the MongoDB endpoint of the account.

### Details

Initializing a new object of this class can either retrieve an existing Cosmos DB resource, or create a new resource on the host. Generally, the best way to initialize an object is via the `get_cosmosdb_account` or `create_cosmosdb_account` methods of the [AzureRMR::az\\_resource\\_group](#) class, which handle the details automatically.

## Endpoints

Azure Cosmos DB provides multiple APIs for accessing the data stored within the account. You choose at account creation the API that you want to use: core (SQL), table storage, Mongo DB, Apache Cassandra, or Gremlin. The following methods allow you to create an endpoint object corresponding to these APIs.

- `get_endpoint(interface=NULL,...)`: Return an endpoint object for interacting with the data. The default `interface=NULL` will choose the interface that you selected at account creation. Otherwise, set `interface` to one of "sql", "table", "mongo", "cassandra" or "gremlin" to create an endpoint object for that API. It's an error to select an interface that the Cosmos DB account doesn't actually provide.
- `get_sql_endpoint(key,key_type=c("master","resource"))`: Return an endpoint object for the core (SQL) API, of class `cosmos_endpoint`. A master key provides full access to all the data in the account; a resource key provides access only to a chosen subset of the data.
- `get_table_endpoint(key)`: Return an endpoint object for the table storage API, of class `AzureTableStor::table_endpoint`.
- `get_mongo_endpoint(key,mongo_options)`: Return an endpoint object for the MongoDB API, of class `cosmos_mongo_endpoint`. `mongo_options` should be an optional named list of parameters to set in the connection string.

Note that `AzureCosmosR` provides a client framework only for the SQL API. To use the table storage API, you will also need the `AzureTableStor` package, and to use the MongoDB API, you will need the `mongolite` package. Currently, the Cassandra and Gremlin APIs are not supported.

As an alternative to `AzureCosmosR`, you can also use the ODBC protocol to interface with the SQL API. By installing a suitable ODBC driver, you can then talk to Cosmos DB in a manner similar to other SQL databases. An advantage of the ODBC interface is that it fully supports cross-partition queries, unlike the REST API. A disadvantage is that it does not support nested document fields; functions like `array_contains()` cannot be used, and attempts to reference arrays and objects may return incorrect results.

## See Also

[get\\_cosmosdb\\_account](#), [create\\_cosmosdb\\_account](#), [delete\\_cosmosdb\\_account](#)

[cosmos\\_endpoint](#), [cosmos\\_database](#), [cosmos\\_container](#), [query\\_documents](#), [cosmos\\_mongo\\_endpoint](#), [AzureTableStor::table\\_endpoint](#), [mongolite::mongo](#)

---

bulk\_delete

*Delete a set of documents from an Azure Cosmos DB container*

---

## Description

Delete a set of documents from an Azure Cosmos DB container

**Usage**

```
bulk_delete(container, ...)

## S3 method for class 'cosmos_container'
bulk_delete(
  container,
  query,
  partition_key,
  procname = "_AzureCosmosR_bulkDelete",
  headers = list(),
  ...
)
```

**Arguments**

container	A Cosmos DB container object, as obtained by <code>get_cosmos_container</code> or <code>create_cosmos_container</code> .
query	A query specifying which documents to delete.
partition_key	Optionally, limit the deletion only to documents with this key value.
procname	The stored procedure name to use for the server-side import code. Change this if, for some reason, the default name is taken.
headers, ...	Optional arguments passed to lower-level functions.

**Details**

This is a convenience function to delete multiple documents from a container. It works by creating a stored procedure and then calling it with the supplied query as a parameter. This function is not meant for production use.

**Value**

The number of rows deleted.

**See Also**

[bulk\\_import](#), [cosmos\\_container](#)

**Examples**

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")
db <- get_cosmos_database(endp, "mydatabase")
cont <- create_cosmos_container(db, "mycontainer", partition_key="sex")

# importing the Star Wars data from dplyr
bulk_import(cont, dplyr::starwars)

# deleting a subset of documents
```

```

bulk_delete(cont, "select * from mycontainer c where c.gender = 'masculine'")

# deleting documents for a specific partition key value
bulk_delete(cont, "select * from mycontainer", partition_key="male")

# deleting all documents
bulk_delete(cont, "select * from mycontainer")

## End(Not run)

```

---

bulk\_import

---

*Import a set of documents to an Azure Cosmos DB container*


---

## Description

Import a set of documents to an Azure Cosmos DB container

## Usage

```

bulk_import(container, ...)

## S3 method for class 'cosmos_container'
bulk_import(
  container,
  data,
  init_chunksize = 1000,
  verbose = TRUE,
  procname = "_AzureCosmosR_bulkImport",
  ...
)

```

## Arguments

container	A Cosmos DB container object, as obtained by <code>get_cosmos_container</code> or <code>create_cosmos_container</code> .
...	Optional arguments passed to lower-level functions.
data	The data to import. Can be a data frame, or a string containing JSON text.
init_chunksize	The number of rows to import per chunk. <code>bulk_import</code> can adjust this number dynamically based on observed performance.
verbose	Whether to print updates to the console as the import progresses.
procname	The stored procedure name to use for the server-side import code. Change this if, for some reason, the default name is taken.

**Details**

This is a convenience function to import a dataset into a container. It works by creating a stored procedure and then calling it in a loop, passing the to-be-imported data in chunks. The dataset must include a column for the container's partition key or an error will result.

Note that this function is not meant for production use. In particular, if the import fails midway through, it will not clean up after itself: you should call `bulk_delete` to remove the remnants of a failed import.

**Value**

A list containing the number of rows imported, for each value of the partition key.

**See Also**

[bulk\\_delete](#), [cosmos\\_container](#)

**Examples**

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")
db <- get_cosmos_database(endp, "mydatabase")
cont <- create_cosmos_container(db, "mycontainer", partition_key="sex")

# importing the Star Wars data from dplyr
# notice that rows with sex=NA are not imported
bulk_import(cont, dplyr::starwars)

# importing from a JSON file
writeLines(jsonlite::toJSON(dplyr::starwars), "starwars.json")
bulk_import(cont, "starwars.json")

## End(Not run)
```

---

cosmos\_endpoint

*Client endpoint for Azure Cosmos DB core API*

---

**Description**

Client endpoint for Azure Cosmos DB core API

**Usage**

```
cosmos_endpoint(
  host,
  key,
  key_type = c("master", "resource"),
```

```

    api_version = getOption("azure_cosmosdb_api_version")
)

call_cosmos_endpoint(
  endpoint,
  path,
  resource_type,
  resource_link,
  options = list(),
  headers = list(),
  body = NULL,
  encode = "json",
  do_continuations = TRUE,
  http_verb = c("GET", "DELETE", "PUT", "POST", "PATCH", "HEAD"),
  num_retries = 10,
  ...
)

process_cosmos_response(response, ...)

## S3 method for class 'response'
process_cosmos_response(
  response,
  http_status_handler = c("stop", "warn", "message", "pass"),
  return_headers = NULL,
  simplify = FALSE,
  ...
)

## S3 method for class 'list'
process_cosmos_response(
  response,
  http_status_handler = c("stop", "warn", "message", "pass"),
  return_headers = NULL,
  simplify = FALSE,
  ...
)

```

### Arguments

host	For <code>cosmos_endpoint</code> , the host URL for the endpoint. Typically of the form <code>https://{account-name}.documents.azure.com:443/</code> (note the port number).
key	For <code>cosmos_endpoint</code> , a string containing the password for the endpoint. This can be either a master key or a resource token.
key_type	For <code>cosmos_endpoint</code> , the type of the key, either "master" or "resource".
api_version	For <code>cosmos_endpoint</code> , the API version to use.
endpoint	For <code>call_cosmos_endpoint</code> , a Cosmos DB endpoint object, as returned by <code>cosmos_endpoint</code> .

path	For <code>call_cosmos_endpoint</code> , the path in the URL for the endpoint call.
resource_type	For <code>call_cosmos_endpoint</code> , the type of resource: for example, "dbs" for a database, "colls" for a collection (container), "docs" for a document, etc.
resource_link	For <code>call_cosmos_endpoint</code> , a string to pass to the API for authorization purposes. See the Cosmos DB API documentation for more information.
options	For <code>call_cosmos_endpoint</code> , query options to include in the request URL.
headers	For <code>call_cosmos_endpoint</code> , any HTTP headers to include in the request. You don't need to include authorization headers as <code>call_cosmos_endpoint</code> will take care of the details.
body	For <code>call_cosmos_endpoint</code> , the body of the request if any.
encode	For <code>call_cosmos_endpoint</code> , the encoding (really content-type) of the request body. The Cosmos DB REST API uses JSON, so there should rarely be a need to change this argument.
do_continuations	For <code>call_cosmos_endpoint</code> , whether to automatically handle paged responses. If FALSE, only the initial response is returned.
http_verb	For <code>call_cosmos_endpoint</code> , the HTTP verb for the request. One of "GET", "POST", "PUT", "PATCH", "HEAD" or "DELETE".
num_retries	For <code>call_cosmos_endpoint</code> , how many times to retry a failed request. Useful for dealing with rate limiting issues.
...	Arguments passed to lower-level functions.
response	For <code>process_cosmos_response</code> , the returned object from a <code>call_cosmos_endpoint</code> call. This will be either a single <code>httr</code> request object, or a list of such objects.
http_status_handler	For <code>process_cosmos_response</code> , the R handler for the HTTP status code of the response. "stop", "warn" or "message" will call the corresponding handlers in <code>httr</code> , while "pass" ignores the status code. The latter is primarily useful for debugging purposes.
return_headers	For <code>process_cosmos_response</code> , whether to return the headers from the response object(s), as opposed to the body. Defaults to TRUE if the original endpoint call was a HEAD request, and FALSE otherwise.
simplify	For <code>process_cosmos_response</code> , whether to convert arrays of objects into data frames via the <code>simplifyDataFrame</code> argument to <code>jsonlite::fromJSON</code> .

## Details

These functions are the basis of the SQL API client framework provided by `AzureCosmosR`. The `cosmos_endpoint` function returns a client object, which can then be passed to other functions for querying databases and containers. The `call_cosmos_endpoint` function sends calls to the REST endpoint, the results of which are then processed by `process_cosmos_response`.

In most cases, you should not have to use `call_cosmos_endpoint` directly. Instead, use `do_cosmos_op` which provides a slightly higher-level interface to the API, by providing sensible defaults for the `resource_type` and `resource_link` arguments and partially filling in the request path.

As an alternative to `AzureCosmosR`, you can also use the ODBC protocol to interface with the SQL API. By installing a suitable ODBC driver, you can then talk to Cosmos DB in a manner similar to

other SQL databases. An advantage of the ODBC interface is that it fully supports cross-partition queries, unlike the REST API. A disadvantage is that it does not support nested document fields; functions like `array_contains()` cannot be used, and attempts to reference arrays and objects may return incorrect results.

Note that AzureCosmosR is a framework for communicating directly with the *core* Cosmos DB client API, also known as the "SQL" API. Cosmos DB provides other APIs as options when creating an account, such as Cassandra, MongoDB, table storage and Gremlin. These APIs are not supported by AzureCosmosR, but you can use other R packages for working with them. For example, you can use AzureTableStor to work with the table storage API, or mongolite to work with the MongoDB API.

### Value

For `cosmos_endpoint`, an object of S3 class `cosmos_endpoint`.

For `call_cosmos_endpoint`, either a `httr` response object, or a list of such responses (if a paged query, and `do_continuations` is `TRUE`).

For `process_cosmos_response` and a single response object, the content of the response. This can be either the parsed response body (if `return_headers` is `FALSE`) or the headers (if `return_headers` is `TRUE`).

For `process_cosmos_response` and a list of response objects, a list containing the individual contents of each response.

### See Also

[do\\_cosmos\\_op](#), [cosmos\\_database](#), [cosmos\\_container](#), [az\\_cosmosdb](#)

`httr::VERB`, which is what carries out the low-level work of sending the HTTP request.

### Examples

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")

# properties for the Cosmos DB account
call_cosmos_endpoint(endp, "", "", "") %>%
  process_cosmos_response()

## End(Not run)
```

---

`cosmos_mongo_endpoint` *MongoDB endpoint for Azure Cosmos DB*

---

### Description

MongoDB endpoint for Azure Cosmos DB

**Usage**

```

cosmos_mongo_endpoint(
  host,
  key,
  mongo_options = list(),
  connection_string = NULL
)

cosmos_mongo_connection(endpoint, ...)

## S3 method for class 'cosmos_mongo_endpoint'
cosmos_mongo_connection(endpoint, collection, database, ...)

```

**Arguments**

host	For <code>cosmos_mongo_endpoint</code> , the URL of the Cosmos DB MongoDB endpoint. Usually of the form "https://account-name.mongo.cosmos.azure.com:443/".
key	For <code>cosmos_mongo_endpoint</code> , a string containing the access key (password) for the endpoint. Can be either a read-write or read-only key.
mongo_options	For <code>cosmos_mongo_endpoint</code> , a named list containing any additional parameters for the MongoDB connection string.
connection_string	Alternatively, the full connection string for the MongoDB endpoint. If this is supplied, all other arguments to <code>cosmos_mongo_endpoint</code> are ignored. Note that if you already have the full connection string, you most likely do not need <code>AzureCosmosR</code> and can call <code>mongolite::mongo</code> directly.
endpoint	For <code>cosmos_mongo_connection</code> , a MongoDB endpoint object as obtained from <code>cosmos_mongo_endpoint</code> .
...	Optional arguments passed to lower-level functions.
collection, database	For <code>cosmos_mongo_connection</code> , the collection and database to connect to.

**Details**

These functions act as a bridge between the Azure resource and the functionality provided by the `mongolite` package.

**Value**

For `cosmos_mongo_endpoint`, an object of S3 class `cosmos_mongo_endpoint`.

For `cosmos_mongo_connection`, an object of class `mongolite::mongo` which can then be used to interact with the given collection.

**See Also**

[az\\_cosmosdb](#), [mongolite::mongo](#)

For the SQL API client framework: [cosmos\\_endpoint](#), [cosmos\\_database](#), [cosmos\\_container](#), [query\\_documents](#)

## Examples

```
## Not run:

mendp <- cosmos_mongo_endpoint("https://mymongoacct.mongo.cosmos.azure.com:443",
  key="mykey")

cosmos_mongo_connection(mendp, "mycollection", "mydatabase")

## End(Not run)
```

---

create\_cosmosdb\_account

*Create Azure Cosmos DB account*

---

## Description

Method for the [AzureRMR::az\\_resource\\_group](#) class.

## Usage

```
create_cosmosdb_account(
  name,
  location = self$location,
  interface = c("sql", "cassandra", "mongo", "table", "graph"),
  serverless = FALSE,
  free_tier = FALSE,
  properties = list(),
  ...
)
```

## Arguments

- `name`: The name of the Cosmos DB account.
- `location`: The location/region in which to create the account. Defaults to the resource group's location.
- `interface`: The default API by which to access data in the account.
- `serverless`: Whether this account should use provisioned throughput or a serverless mode. In the latter, you are charged solely on the basis of the traffic generated by your database operations. Serverless mode is best suited for small-to-medium workloads with light and intermittent traffic that is hard to forecast; it is currently (January 2021) in preview.
- `free_tier`: Whether this account should be in the free tier, in which a certain amount of database operations are provided free of charge. You can have one free tier account per subscription.
- `properties`: Additional properties to set for the account.
- `wait`: Whether to wait until the Cosmos DB account provisioning is complete.
- `...`: Optional arguments to pass to `az_cosmosdb$new()`.

## Details

This method creates a new Azure Cosmos DB account in the given resource group. Azure Cosmos DB is a globally distributed multi-model database that supports the document, graph, and key-value data models.

The ARM resource object provides methods for working in the management plane. For working in the data plane, AzureCosmosR provides a client framework that interfaces with the core (SQL) API. Other packages provide functionality for other APIs, such as AzureTableStor for table storage and mongolite for MongoDB.

## Value

An object of class `az_cosmosdb` representing the Cosmos DB account.

## See Also

[get\\_cosmosdb\\_account](#), [delete\\_cosmosdb\\_account](#)

For the SQL API client framework: [cosmos\\_endpoint](#), [cosmos\\_database](#), [cosmos\\_container](#), [query\\_documents](#)

For the table storage API: [AzureTableStor::table\\_endpoint](#)

For the MongoDB API: [cosmos\\_mongo\\_endpoint](#), [mongolite::mongo](#)

---

delete\_cosmosdb\_account

*Delete Azure Cosmos DB account*

---

## Description

Method for the [AzureRMR::az\\_resource\\_group](#) class.

## Usage

```
delete_cosmosdb_account(name, confirm = TRUE, wait = FALSE)
```

## Arguments

- `name`: The name of the Cosmos DB account.
- `confirm`: Whether to ask for confirmation before deleting.
- `wait`: Whether to wait until the deletion has completed before returning.

## Details

This method deletes an existing Azure Cosmos DB account.

**See Also**

[create\\_cosmosdb\\_account](#), [get\\_cosmosdb\\_account](#)

For the SQL API client framework: [cosmos\\_endpoint](#), [cosmos\\_database](#), [cosmos\\_container](#), [query\\_documents](#)

For the table storage API: [AzureTableStor::table\\_endpoint](#)

For the MongoDB API: [cosmos\\_mongo\\_endpoint](#), [mongolite::mongo](#)

---

do\_cosmos\_op

*Carry out a Cosmos DB operation*


---

**Description**

Carry out a Cosmos DB operation

**Usage**

```
do_cosmos_op(object, ...)

## S3 method for class 'cosmos_endpoint'
do_cosmos_op(object, ...)

## S3 method for class 'cosmos_database'
do_cosmos_op(object, path = "", resource_type = "dbs", resource_link = "", ...)

## S3 method for class 'cosmos_container'
do_cosmos_op(
  object,
  path = "",
  resource_type = "colls",
  resource_link = "",
  ...
)

## S3 method for class 'cosmos_document'
do_cosmos_op(
  object,
  path = "",
  resource_type = "docs",
  resource_link = "",
  headers = list(),
  ...
)
```

**Arguments**

object	A Cosmos DB endpoint, database, container or document object.
...	Arguments passed to lower-level functions.
path	The (partial) URL path for the operation.
resource_type	The type of resource. For most purposes, the default value should suffice.
resource_link	The resource link for authorization. For most purposes, the default value should suffice.
headers	Any optional HTTP headers to include in the API call.

**Details**

do\_cosmos\_op provides a higher-level interface to the Cosmos DB REST API than call\_cosmos\_endpoint. In particular, it sets the resource\_type and resource\_link arguments to sensible defaults, and fills in the beginning of the URL path for the REST call.

**Value**

The result of call\_cosmos\_endpoint: either a httr response object, or a list of such objects. Call process\_cosmos\_response to extract the result of the call.

**Examples**

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")

db <- get_cosmos_database(endp, "mydatabase")
do_cosmos_op(db) %>%
  process_cosmos_response()

cont <- get_cosmos_container(db, "mycontainer")
do_cosmos_op(cont) %>%
  process_cosmos_response()

## End(Not run)
```

---

get\_cosmosdb\_account *Get Azure Cosmos DB account*

---

**Description**

Method for the [AzureRMR::az\\_resource\\_group](#) class.

**Usage**

```
get_cosmosdb_account(name)
list_cosmosdb_accounts()
```

**Arguments**

- name: The name of the Cosmos DB account.

**Details**

get\_cosmosdb\_account retrieves the details for an existing Azure Cosmos DB account. list\_cosmosdb\_accounts retrieves all the Cosmos DB accounts within the resource group.

**Value**

For get\_cosmosdb\_account, an object of class az\_cosmosdb representing the Cosmos DB account. For list\_cosmosdb\_accounts, a list of such objects.

**See Also**

[create\\_cosmosdb\\_account](#), [delete\\_cosmosdb\\_account](#)

For the SQL API client framework: [cosmos\\_endpoint](#), [cosmos\\_database](#), [cosmos\\_container](#), [query\\_documents](#)

For the table storage API: [AzureTableStor::table\\_endpoint](#)

For the MongoDB API: [cosmos\\_mongo\\_endpoint](#), [mongolite::mongo](#)

---

get\_cosmos\_container    *Methods for working with Azure Cosmos DB containers*

---

**Description**

Methods for working with Azure Cosmos DB containers

**Usage**

```
get_cosmos_container(object, ...)

## S3 method for class 'cosmos_database'
get_cosmos_container(object, container, ...)

## S3 method for class 'cosmos_endpoint'
get_cosmos_container(object, database, container, ...)

create_cosmos_container(object, ...)

## S3 method for class 'cosmos_database'
create_cosmos_container(
  object,
  container,
  partition_key,
  partition_version = 2,
  autoscale_maxRUs = NULL,
```

```

    manual_RUs = NULL,
    headers = list(),
    ...
)

delete_cosmos_container(object, ...)

## S3 method for class 'cosmos_database'
delete_cosmos_container(object, container, confirm = TRUE, ...)

## S3 method for class 'cosmos_container'
delete_cosmos_container(object, ...)

list_cosmos_containers(object, ...)

## S3 method for class 'cosmos_database'
list_cosmos_containers(object, ...)

```

### Arguments

object	A Cosmos DB database object, as obtained from <code>get_cosmos_database</code> or <code>create_cosmos_database</code> , or for <code>delete_cosmos_container.cosmos_container</code> , the container object.
container	The name of the container.
database	For <code>get_cosmos_container.cosmos_endpoint</code> , the name of the database that includes the container.
partition_key	For <code>create_cosmos_container</code> , the name of the partition key.
partition_version	For <code>create_cosmos_container</code> , the partition version. Can be either 1 or 2. Version 2 supports large partition key values (longer than 100 bytes) but requires API version 2018-12-31 or later. Use version 1 if the container needs to be accessible to older Cosmos DB SDKs.
autoscale_maxRUs, manual_RUs	For <code>create_cosmos_container</code> , optional parameters for the maximum request units (RUs) allowed. See the Cosmos DB documentation for more details.
headers, ...	Optional arguments passed to lower-level functions.
confirm	For <code>delete_cosmos_container</code> , whether to ask for confirmation before deleting.

### Details

These are methods for working with Cosmos DB containers using the core (SQL) API. A container is analogous to a table in SQL, or a collection in MongoDB.

`get_cosmos_container`, `create_cosmos_container`, `delete_cosmos_container` and `list_cosmos_containers` provide basic container management functionality.

`get_partition_key` returns the name of the partition key column in the container, and `list_partition_key_values` returns all the distinct values for this column. These are useful when working with queries that have to be mapped across partitions.

**Value**

For `get_cosmos_container` and `create_cosmos_container`, an object of class `cosmos_container`. For `list_cosmos_containers`, a list of such objects.

**See Also**

[cosmos\\_container](#), [query\\_documents](#), [bulk\\_import](#), [bulk\\_delete](#)

**Examples**

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")
db <- get_cosmos_database(endp, "mydatabase")

create_cosmos_container(db, "mycontainer", partition_key="sex")

list_cosmos_containers(db)

cont <- get_cosmos_container(db, "mycontainer")

delete_cosmos_container(cont)

## End(Not run)
```

---

get\_cosmos\_database    *Methods for working with Azure Cosmos DB databases*

---

**Description**

Methods for working with Azure Cosmos DB databases

**Usage**

```
get_cosmos_database(object, ...)

## S3 method for class 'cosmos_endpoint'
get_cosmos_database(object, database, ...)

create_cosmos_database(object, ...)

## S3 method for class 'cosmos_endpoint'
create_cosmos_database(
  object,
  database,
  autoscale_maxRUs = NULL,
  manual_RUs = NULL,
```

```

    headers = list(),
    ...
)

delete_cosmos_database(object, ...)

## S3 method for class 'cosmos_endpoint'
delete_cosmos_database(object, database, confirm = TRUE, ...)

## S3 method for class 'cosmos_database'
delete_cosmos_database(object, ...)

list_cosmos_databases(object, ...)

## S3 method for class 'cosmos_endpoint'
list_cosmos_databases(object, ...)

```

### Arguments

<code>object</code>	A Cosmos DB endpoint object as obtained from <code>cosmos_endpoint</code> , or for <code>delete_cosmos_database</code> . the database object.
<code>database</code>	The name of the Cosmos DB database.
<code>autoscale_maxRUs, manual_RUs</code>	For <code>create_cosmos_database</code> , optional parameters for the maximum request units (RUs) allowed. See the Cosmos DB documentation for more details.
<code>headers, ...</code>	Optional arguments passed to lower-level functions.
<code>confirm</code>	For <code>delete_cosmos_database</code> , whether to ask for confirmation before deleting.

### Details

These are methods for managing Cosmos DB databases using the core (SQL) API.

### Value

`get_cosmos_database` and `create_cosmos_database` return an object of class `cosmos_database`.  
`list_cosmos_databases` returns a list of such objects.

### Examples

```

## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")

create_cosmos_database(endp, "mydatabase")

list_cosmos_databases(endp)

db <- get_cosmos_database(endp, "mydatabase")

```

```
delete_cosmos_database(db)
```

```
## End(Not run)
```

---

```
get_document
```

*Methods for working with Azure Cosmos DB documents*

---

## Description

Methods for working with Azure Cosmos DB documents

## Usage

```
get_document(object, ...)
```

```
create_document(object, ...)
```

```
## S3 method for class 'cosmos_container'  
create_document(object, data, headers = list(), ...)
```

```
list_documents(object, ...)
```

```
## S3 method for class 'cosmos_container'  
list_documents(  
  object,  
  partition_key = NULL,  
  as_data_frame = FALSE,  
  metadata = TRUE,  
  headers = list(),  
  ...  
)
```

```
delete_document(object, ...)
```

```
## S3 method for class 'cosmos_container'  
delete_document(  
  object,  
  id,  
  partition_key,  
  headers = list(),  
  confirm = TRUE,  
  ...  
)
```

```
## S3 method for class 'cosmos_document'  
delete_document(object, ...)
```

**Arguments**

object	A Cosmos DB container object, as obtained by <code>get_cosmos_container</code> or <code>create_cosmos_container</code> .
data	For <code>create_document</code> , the document data. This can be either a string containing JSON text, or a (possibly nested) list containing the parsed JSON.
headers, ...	Optional arguments passed to lower-level functions.
partition_key	For <code>get_document</code> and <code>delete_document</code> , the value of the partition key for the desired document. For <code>list_documents</code> , restrict the returned list only to documents with this key value.
as_data_frame	For <code>list_documents</code> , whether to return a data frame or a list of Cosmos DB document objects. Note that the default value is <code>FALSE</code> , unlike <a href="#">query_documents</a> .
metadata	For <code>get_document</code> and <code>list_documents</code> , whether to include Cosmos DB document metadata in the result.
id	The document ID.
confirm	For <code>delete_cosmos_container</code> , whether to ask for confirmation before deleting.

**Details**

These are low-level functions for working with individual documents in a Cosmos DB container. In most cases you will want to use [query\\_documents](#) to issue queries against the container, or [bulk\\_import](#) and [bulk\\_delete](#) to create and delete documents.

**Value**

`get_document` and `create_document` return an object of S3 class `cosmos_document`. The actual document contents can be found in the `data` component of this object.

`list_documents` returns a list of `cosmos_document` objects if `as_data_frame` is `FALSE`, and a data frame otherwise.

**See Also**

[query\\_documents](#), [bulk\\_import](#), [bulk\\_delete](#), [cosmos\\_container](#)

**Examples**

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")
db <- get_cosmos_database(endp, "mydatabase")

cont <- get_cosmos_container(db, "mycontainer")

# a list of document objects
list_documents(cont)

# a data frame
```

```
list_documents(cont, as_data_frame=TRUE)

# a single document
doc <- get_document(cont, "mydocumentid")
doc$data

delete_document(doc)

## End(Not run)
```

---

get\_partition\_key      *Container partition key information*

---

### **Description**

Container partition key information

### **Usage**

```
get_partition_key(container)

list_partition_key_values(container)

list_partition_key_ranges(container)
```

### **Arguments**

container      An object of class `cosmos_container`.

### **Details**

These are functions to facilitate working with a Cosmos DB container, which often requires knowledge of its partition key.

### **Value**

For `get_partition_key`, the name of the partition key column as a string.

For `list_partition_key_values`, a character vector of all the values of the partition key.

For `list_partition_key_ranges`, a character vector of the IDs of the partition key ranges.

---

get\_stored\_procedure *Methods for working with Azure Cosmos DB stored procedures*

---

## Description

Methods for working with Azure Cosmos DB stored procedures

## Usage

```
get_stored_procedure(object, ...)  
  
## S3 method for class 'cosmos_container'  
get_stored_procedure(object, procname, ...)  
  
list_stored_procedures(object, ...)  
  
create_stored_procedure(object, ...)  
  
## S3 method for class 'cosmos_container'  
create_stored_procedure(object, procname, body, ...)  
  
exec_stored_procedure(object, ...)  
  
## S3 method for class 'cosmos_container'  
exec_stored_procedure(object, procname, parameters = list(), ...)  
  
## S3 method for class 'cosmos_stored_procedure'  
exec_stored_procedure(object, ...)  
  
replace_stored_procedure(object, ...)  
  
## S3 method for class 'cosmos_container'  
replace_stored_procedure(object, procname, body, ...)  
  
## S3 method for class 'cosmos_stored_procedure'  
replace_stored_procedure(object, body, ...)  
  
delete_stored_procedure(object, ...)  
  
## S3 method for class 'cosmos_container'  
delete_stored_procedure(object, procname, confirm = TRUE, ...)  
  
## S3 method for class 'cosmos_stored_procedure'  
delete_stored_procedure(object, ...)
```

**Arguments**

object	A Cosmos DB container object, as obtained by <code>get_cosmos_container</code> or <code>create_cosmos_container</code> , or for <code>delete_stored_procedure.cosmos_stored_procedure</code> , the stored procedure object.
...	Optional arguments passed to lower-level functions.
procname	The name of the stored procedure.
body	For <code>create_stored_procedure</code> and <code>replace_stored_procedure</code> , the body of the stored procedure. This can be either a character string containing the source code, or the name of a source file.
parameters	For <code>exec_stored_procedure</code> , a list of parameters to pass to the procedure.
confirm	For <code>delete_stored_procedure</code> , whether to ask for confirmation before deleting.

**Details**

These are methods for working with stored procedures in Azure Cosmos DB using the core (SQL) API. In the Cosmos DB model, stored procedures are written in JavaScript and associated with a container.

**Value**

For `get_stored_procedure` and `create_stored_procedure`, an object of class `cosmos_stored_procedure`. For `list_stored_procedures`, a list of such objects.

**See Also**

[cosmos\\_container](#), [get\\_udf](#)

**Examples**

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")
db <- get_cosmos_database(endp, "mydatabase")
cont <- create_cosmos_container(db, "mycontainer", partition_key="sex")

# a simple stored procedure
src <- 'function helloworld() {
  var context = getContext();
  var response = context.getResponse();
  response.setBody("Hello, World");
}'
create_stored_procedure(cont, "helloworld", src)
sproc <- get_stored_procedure(cont, "helloworld")
exec_stored_procedure(sproc)

# more complex example: uploading data
sproc2 <- create_stored_procedure(cont, "myBulkUpload",
  body=system.file("srcjs/bulkUpload.js", package="AzureCosmosR"))
```

```
list_stored_procedures(cont)

sw_male <- dplyr::filter(dplyr::starwars, sex == "male")
exec_stored_procedure(sproc2, parameters=list(sw_male))

delete_stored_procedure(sproc)
delete_stored_procedure(sproc2)

## End(Not run)
```

---

get\_udf

*Methods for working with Azure Cosmos DB user-defined functions*

---

## Description

Methods for working with Azure Cosmos DB user-defined functions

## Usage

```
get_udf(object, ...)

## S3 method for class 'cosmos_container'
get_udf(object, funcname, ...)

list_udfs(object, ...)

create_udf(object, ...)

## S3 method for class 'cosmos_container'
create_udf(object, funcname, body, ...)

replace_udf(object, ...)

## S3 method for class 'cosmos_container'
replace_udf(object, funcname, body, ...)

## S3 method for class 'cosmos_udf'
replace_udf(object, body, ...)

delete_udf(object, ...)

## S3 method for class 'cosmos_container'
delete_udf(object, funcname, confirm = TRUE, ...)

## S3 method for class 'cosmos_udf'
delete_udf(object, ...)
```

## Arguments

object	A Cosmos DB container object, as obtained by <code>get_cosmos_container</code> or <code>create_cosmos_container</code> , or for <code>delete_udf.cosmos_udf</code> , the function object.
...	Optional arguments passed to lower-level functions.
funcname	The name of the user-defined function.
body	For <code>create_udf</code> and <code>replace_udf</code> , the body of the function. This can be either a character string containing the source code, or the name of a source file.
confirm	For <code>delete_udf</code> , whether to ask for confirmation before deleting.

## Details

These are methods for working with user-defined functions (UDFs) in Azure Cosmos DB using the core (SQL) API. In the Cosmos DB model, UDFs are written in JavaScript and associated with a container.

## Value

For `get_udf` and `create_udf`, an object of class `cosmos_udf`. For `list_udfs`, a list of such objects.

## See Also

[cosmos\\_container](#), [get\\_stored\\_procedure](#)

## Examples

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")
db <- get_cosmos_database(endp, "mydatabase")

# importing the Star Wars data from dplyr
cont <- endp %>%
  get_cosmos_database(endp, "mydatabase") %>%
  create_cosmos_container(db, "mycontainer", partition_key="sex")

create_udf(cont, "times2", "function(x) { return 2*x; }")

list_udfs(cont)

# UDFs in queries are prefixed with the 'udf.' identifier
query_documents(cont, "select udf.times2(c.height) t2 from cont c")

delete_udf(cont, "times2")

## End(Not run)
```

---

query_documents	<i>Query an Azure Cosmos DB container</i>
-----------------	---

---

### Description

Query an Azure Cosmos DB container

### Usage

```
query_documents(container, ...)

## S3 method for class 'cosmos_container'
query_documents(
  container,
  query,
  parameters = list(),
  cross_partition = TRUE,
  partition_key = NULL,
  by_pkrange = FALSE,
  as_data_frame = TRUE,
  metadata = TRUE,
  headers = list(),
  ...
)
```

### Arguments

container	A Cosmos DB container object, as obtained by <code>get_cosmos_container</code> or <code>create_cosmos_container</code> .
query	A string containing the query text.
parameters	A named list of parameters to pass to a parameterised query, if required.
cross_partition, partition_key, by_pkrange	Arguments that control how to handle cross-partition queries. See 'Details' below.
as_data_frame	Whether to return the query result as a data frame, or a list of Cosmos DB document objects.
metadata	Whether to include Cosmos DB document metadata in the query result.
headers, ...	Optional arguments passed to lower-level functions.

### Details

This is the primary function for querying the contents of a Cosmos DB container (table). The query argument should contain the text of a SQL query, optionally parameterised. If the query contains parameters, pass them in the `parameters` argument as a named list.

Cosmos DB is a partitioned key-value store under the hood, with documents stored in separate physical databases according to their value of the partition key. The Cosmos DB REST API has limited support for cross-partition queries: basic SELECTs should work, but aggregates and more complex queries may require some hand-hacking.

The default `cross_partition=TRUE` runs the query for all partition key values and then attempts to stitch the results together. To run the query for only one key value, set `cross_partition=FALSE` and `partition_key` to the desired value. You can obtain all the values of the key with the [list\\_partition\\_key\\_values](#) function.

The `by_pkrange` argument allows running the query separately across all *partition key ranges*. Each partition key range corresponds to a separate physical partition, and contains the documents for one or more key values. You can set this to `TRUE` to run a query that fails when run across partitions; the returned object will be a list containing the individual query results from each pkrange.

As an alternative to `AzureCosmosR`, you can also use the ODBC protocol to interface with the SQL API. By installing a suitable ODBC driver, you can then talk to Cosmos DB in a manner similar to other SQL databases. An advantage of the ODBC interface is that it fully supports cross-partition queries, unlike the REST API. A disadvantage is that it does not support nested document fields; functions like `array_contains()` cannot be used, and attempts to reference arrays and objects may return incorrect results.

## Value

`query_documents` returns the results of the query. Most of the time this will be a data frame, or list of data frames if `by_pkrange=TRUE`.

## See Also

[cosmos\\_container](#), [cosmos\\_document](#), [list\\_partition\\_key\\_values](#), [list\\_partition\\_key\\_ranges](#)

## Examples

```
## Not run:

endp <- cosmos_endpoint("https://myaccount.documents.azure.com:443/", key="mykey")

# importing the Star Wars data from dplyr
cont <- endp %>%
  get_cosmos_database(endp, "mydatabase") %>%
  create_cosmos_container(db, "mycontainer", partition_key="sex")

bulk_import(cont, dplyr::starwars)

query_documents(cont, "select * from mycontainer")

# removing the Cosmos DB metadata cruft
query_documents(cont, "select * from mycontainer", metadata=FALSE)

# a simple filter
query_documents(cont, "select * from mycontainer c where c.gender = 'masculine'")

# run query for one partition key -- zero rows returned
```

```
query_documents(cont, "select * from mycontainer c where c.gender = 'masculine'",
                partition_key="female")

# aggregates will fail -- API does not fully support cross-partition queries
try(query_documents(cont, "select avg(c.height) avgheight from mycontainer c"))
# Error in process_cosmos_response.response(response, simplify = as_data_frame) :
# Bad Request (HTTP 400). Failed to complete Cosmos DB operation. Message:
# ...

# run query separately by pkrange and combine the results manually
query_documents(
  cont,
  "select avg(c.height) avgheight, count(1) n from mycontainer c",
  by_pkrange=TRUE
)

## End(Not run)
```

# Index

- az\_cosmosdb, [2](#), [9](#), [10](#)
- AzureRMR::az\_resource, [2](#)
- AzureRMR::az\_resource\_group, [2](#), [11](#), [12](#), [14](#)
- AzureTableStor::table\_endpoint, [3](#), [12](#), [13](#), [15](#)
  
- bulk\_delete, [3](#), [6](#), [17](#), [20](#)
- bulk\_import, [4](#), [5](#), [17](#), [20](#)
  
- call\_cosmos\_endpoint (cosmos\_endpoint), [6](#)
- cosmos\_container, [3](#), [4](#), [6](#), [9](#), [10](#), [12](#), [13](#), [15](#), [17](#), [20](#), [23](#), [25](#), [27](#)
- cosmos\_container (get\_cosmos\_container), [15](#)
- cosmos\_database, [2](#), [3](#), [9](#), [10](#), [12](#), [13](#), [15](#)
- cosmos\_database (get\_cosmos\_database), [17](#)
- cosmos\_document, [27](#)
- cosmos\_document (get\_document), [19](#)
- cosmos\_endpoint, [2](#), [3](#), [6](#), [10](#), [12](#), [13](#), [15](#)
- cosmos\_mongo\_connection (cosmos\_mongo\_endpoint), [9](#)
- cosmos\_mongo\_endpoint, [3](#), [9](#), [12](#), [13](#), [15](#)
- cosmos\_stored\_procedure (get\_stored\_procedure), [22](#)
- create\_cosmos\_container (get\_cosmos\_container), [15](#)
- create\_cosmos\_database (get\_cosmos\_database), [17](#)
- create\_cosmosdb\_account, [3](#), [11](#), [13](#), [15](#)
- create\_document (get\_document), [19](#)
- create\_stored\_procedure (get\_stored\_procedure), [22](#)
- create\_udf (get\_udf), [24](#)
  
- delete\_cosmos\_container (get\_cosmos\_container), [15](#)
- delete\_cosmos\_database (get\_cosmos\_database), [17](#)
- delete\_cosmosdb\_account, [3](#), [12](#), [12](#), [15](#)
- delete\_document (get\_document), [19](#)
- delete\_stored\_procedure (get\_stored\_procedure), [22](#)
- delete\_udf (get\_udf), [24](#)
- do\_cosmos\_op, [9](#), [13](#)
  
- exec\_stored\_procedure (get\_stored\_procedure), [22](#)
  
- get\_cosmos\_container, [15](#)
- get\_cosmos\_database, [17](#)
- get\_cosmosdb\_account, [3](#), [12](#), [13](#), [14](#)
- get\_document, [19](#)
- get\_partition\_key, [21](#)
- get\_stored\_procedure, [22](#), [25](#)
- get\_udf, [23](#), [24](#)
  
- htr::VERB, [9](#)
  
- jsonlite::fromJSON, [8](#)
  
- list\_cosmos\_containers (get\_cosmos\_container), [15](#)
- list\_cosmos\_databases (get\_cosmos\_database), [17](#)
- list\_cosmosdb\_accounts (get\_cosmosdb\_account), [14](#)
- list\_documents (get\_document), [19](#)
- list\_partition\_key\_ranges, [27](#)
- list\_partition\_key\_ranges (get\_partition\_key), [21](#)
- list\_partition\_key\_values, [27](#)
- list\_partition\_key\_values (get\_partition\_key), [21](#)
- list\_stored\_procedures (get\_stored\_procedure), [22](#)
- list\_udfs (get\_udf), [24](#)

mongolite::mongo, [3](#), [10](#), [12](#), [13](#), [15](#)

process\_cosmos\_response  
    (cosmos\_endpoint), [6](#)

query\_documents, [3](#), [10](#), [12](#), [13](#), [15](#), [17](#), [20](#), [26](#)

replace\_stored\_procedure  
    (get\_stored\_procedure), [22](#)

replace\_udf (get\_udf), [24](#)